Special Feature 1 情報処理の基礎確認
Special Feature 2 メールシステムのしくみ

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

2015 September 9

2015年9月18日発行
毎月1回18日発行
通巻365号
（発刊299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

エンジニアの夏期講習

特講

special feature 1

正規表現 SQL オブジェクト指向

苦手克服のベストプラクティス

しくみをご存じですか?

special feature 2

メールシステムの教科書

日本語もバイナリもちゃんと届くのはなぜか

extra feature

なぜ俺の提案は通らないのか?

冷静と情熱の間には、お金の川が流れている

好評! 短期連載 Jamesのセキュリティレッスン「Wiresharkの新ノウハウ」

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。

2015 September
9
Software Design

第1特集

特講 エンジニアの夏期講習

正規表現 SQL オブジェクト指向

苦手克服のベストプラクティス

contents

2015
September
9
Software Design

contents

第2特集

特別企画

短期連載

Solution Flash

Catch up trend

アラカルト

Column

Development

OS/Network

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

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。

OSとネットワーク、
IT環境を支えるエンジニアの総合誌

Software Design

毎月18日発売

PDF電子版
Gihyo Digital
Publishingにて
販売開始

年間定期購読と電子版販売のご案内

1年購読(12回)

14,880円(税込み、送料無料)1冊あたり1,240円(6%割引)

PDF電子版の購入については

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

をご覧ください。
PDF電子版の年間定期購読も受け付けております。

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

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

Fujisan.co.jpからのお申し込み方法

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

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

大人の自由時間

（やりたいときが、始めどき！）

mini　新たな興味が見つかる好奇心BOOK、創刊!!

定年のない世代が、充実したOFFタイムを過ごすための
「今どき」だけど「渋い」趣味&生活実用をテーマとした大人時間の入門書。

◉各書ともB6 判　定価（本体1280円+税）

ISBN978-4-7741-7398-6

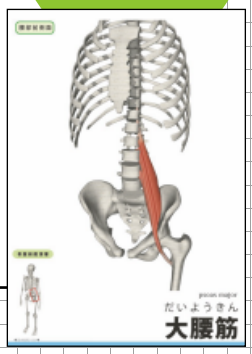
効く筋肉が見える筋トレ図鑑

比嘉一雄◉著

自重トレーニングで30才の体を取り戻そう

健康で長生きする人が持っているのは「筋肉の量」だった！
CG解説で鍛える筋肉がイメージできるので効果がさらにアップ。
自重トレーニングなら思い立ったらすぐ自宅で始められます。

CG解説で筋肉が見える！



眠っていたアナログレコードが甦る！

思い出のアナログレコードを再び楽しむ

かんたんきれいにデジタル化

藤本健・大坪知樹◉著

大好きなアーティストのCD音源化されなかったアルバムや特別盤などを、
現在のオーディオ環境で再生するための方法、教えます！
ハイレゾ化の記事もあります。

ISBN978-4-7741-7396-2

ISBN978-4-7741-7397-9

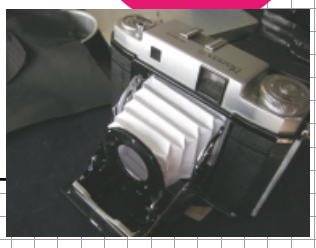
I Love クラシックカメラ

セイリー育緒◉著

はじめてのフィルムカメラ修理

シンプルな機構のフィルムカメラは機械いじりの楽しみを教えてくれます。
自分の手で修理、調整したカメラへの愛着感はハンパなしです。
是非チャレンジを！

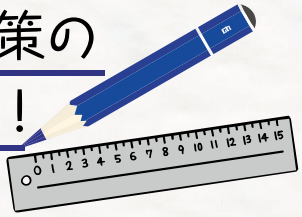
フィルムカメラよ永遠に！



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

あなたを
合格へと導く
一冊があります！

効率よく学習できる
試験対策の
大定番！



岡嶋裕史　著
A5判／624ページ
定価（本体2980円+税）
ISBN978-4-7741-7207-1

エディフィストラーニング株式会社　著
B5判／384ページ
定価（本体2980円+税）
ISBN978-4-7741-7208-8

岡嶋裕史　著
A5判／656ページ
定価（本体2880円+税）
ISBN978-4-7741-6937-8

エディフィストラーニング株式会社　著
B5判／392ページ
定価（本体2980円+税）
ISBN978-4-7741-7456-3

岡嶋裕史　著
B6判／320ページ
定価（本体1680円+税）
ISBN978-4-7741-6942-2

左門至峰・平田賀一　著
A5判／352ページ
定価（本体2380円+税）
ISBN978-4-7741-7294-1

金子則彦　著
A5判／680ページ
定価（本体3300円+税）
ISBN978-4-7741-6941-5

大滝みや子・岡嶋裕史　著
A5判／736ページ
定価（本体2980円+税）
ISBN978-4-7741-6924-8

大滝みや子　著
B6判／384ページ
定価（本体1480円+税）
ISBN978-4-7741-6710-7

加藤昭・高見澤秀幸・矢野龍王　著
B5判／456ページ
定価（本体1880円+税）
ISBN978-4-7741-7440-2

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

第81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。

Apache Spark

並列分散処理基盤「Apache Spark」

Apache Sparkは、大規模データを極めて高速に処理することができるオープンソースの並列分散処理フレームワークです。大規模データに対応した分散処理基盤としてはApache Hadoopが有名ですが、SparkはHadoopとは異なるしくみでデータ処理を実現しており、Hadoopの苦手とする分野を補完できるとして注目を集めています。

Sparkのおもな特徴としては次のようなものが挙げられています。

- **スピード**……Hadoopに比べてインメモリで100倍、ディスクでも10倍高速
- **使いやすさ**……Scala、Java、Python、R向けのAPIが用意されており、シンプルなコードで簡単に並列処理を記述できる
- **普遍性**……SQLやストリーム処理、機械学習、解析などのさまざまなライブラリが用意されている
- **どこでも動く**……スタンドアロンだけでなく、クラウドやHadoop、Mesosなどさまざまな環境上で動作する。また、HDFSやCassandra、HBase、S3といったデータソースにもアクセスできる

現在、大規模データ処理のスタンダードとなっているHadoopは、MapReduceと呼ばれるしくみを利用して大量データの並列処理を実現しています。処理するサーバを増やすことで簡単にスループットを高めることができる点が大きな強みですが、その一方で、1つのデータセットに対して繰り返し計算を行うような処理では、ディスクI/Oやデータ転送のコストが無視できずに処理効率が大きく低下するという弱みがあります。

それに対してSparkが得意としているのが、まさしくこの大量データを繰り返し変換していくような処理です。したがってSparkは、このような処理が頻繁に登場する機械学習やグラフアルゴリズム、インタラクティブなデータマイニングなどの分野においてとくに大きな力を発揮することができます。一方で、クラスタの総メモリに収まりきらない大きさのデータ処理や、大きなデータセットを少しずつ更新するような処理を苦手としているという側面もあります。

Sparkのしくみ

Sparkが高速な並列分散処理を実現するうえで鍵となっているのが、DAG（Directed Acyclic Graph：有向非循環グラフ）ベースの実行エンジンと、RDD（Resilient Distributed Dataset）と呼ばれる抽象化データセットです。

Sparkの実行エンジンでは、繰り返し処理をDAGと呼ばれるグラフを用いて表現します。MapReduceの場合、基本的にはMapとReduceという2つのステップを逐次実行することが求められます。それに対してDAGでは、ツリー構造を形成可能な複数のステップを持てるため、MapReduceよりも汎用性が高く、より複雑な計算処理を効率的な形で表現できるとのことです。さらに、MapReduceのように中間結果をディスクに書き込まずインメモリで動作するため、ディスクI/Oのオーバーヘッドを排除して高速に動作するというメリットもあります。

もう1つの鍵であるRDDはコレクションのようなデータ構造で、次のような性質を備えることにより、並列分散によるレイテンシの低い計算処理を実現しています。

- イミュータブル（不変）である
- パーティション単位で分割され、複数のサーバ上に分散配置される
- 再利用のためにメモリ上にキャッシュされる（可能な限りインメモリで処理される）
- 遅延計算される

このRDDにデータを保持したうえで、変換処理を繰り返し実施することによって目的の結果を得るというのが、Sparkを使った基本的なデータ処理の流れになります。

上記に加えて、耐障害性を確保するためにRDDはデータが変換された経緯の情報を保持しています。これによって、もし必要なデータに欠損があった場合でも、失われる前のデータから目的のデータを再構築することができるわけです。

前述のように、SparkとHadoopにはそれぞれ得意・不得意な分野があり、決して競合する性質のものではありません。したがって、今後は両者を組み合わせて利用するモデルが一般的になるものと考えられます。SD

Apache Spark
http://spark.apache.org/

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。

DIGITAL GADGET vol.201

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

広告におけるデジタルの役目

デジタル技術と広告技術

ネットの世界を楽しんだり、デジタルコンテンツを楽しんでいると、知らず知らずの間に多くの広告に触れています。目が勝手にフィルタリングして見ないようにしているかもしれませんし、有益な広告であれば注意深く観ているかもしれません。

Webページの中に細長く表示されるバナー広告から、記事やコンテンツの中に溶け込んだ形で掲載するネイティブ広告、検索したキーワードと連動した広告など、広告の形態もいろいろ進化してきました。さらにWebページだけでなく、アプリ内の広告も浸透してきました。

広告制作も少しずつ変化しつつあります。さまざまな広告媒体を活用し、多角的に展開することによって、視聴者がより広く商品やサービスにかかわるようになってきています。加えて、さまざまな媒体に対して同じような広告を展開するのではなく、SNSにはSNS用の広告、WebにはWeb用の広告、動画広告はまた別な形でと媒体の特性にあわせて少しだけ形の違う広告を打ち出すことにより、商品やサービスをより親密に感じてもらえるように工夫されてきています。

さらに広告にどとまらず、サービスに必要となるハードウェアを作ってしまったり、スタートアップ支援のような形で、投資しつつプロジェクトを後押しするような新しいタイプのかかわり方をしていることが伝わってくる広告制作も出現してきました。

ネット広告の量と予算額が多くなるにともなって、「少ない予算で工夫して挑戦的なことを試そう!」という風潮とはまた別に、多くの人に届く確実な表現で、伝わる広告を作ろうという風潮もあり、大きく表現が変化していく時代になってきているようです。

Cannes Lions 2015より

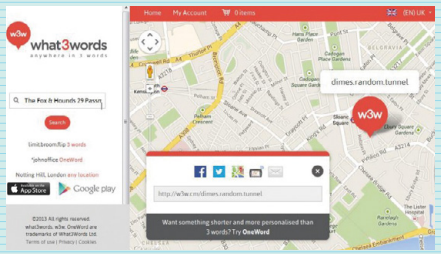
2015年6月に、広告を中心としたクリエイティブフェスティバル、Cannes Lions 2015が開催されました。単なる広告からさまざまなクリエイティブに活躍の場を拡げているCannes Lions 2015フェスティバルから、とくにモバイル端末や最新テクノロジを活用した、広告戦略のいくつかを紹介しましょう。

Cannes Lions
http://www.canneslions.com/

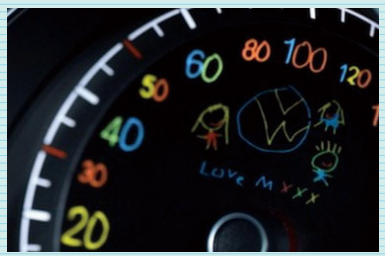
サイバー部門
(Webサイトや、
デジタル分野の広告)より

Look At Me
- Samsung Electronics -
Cheil Worldwide
http://pages.samsung.com/ca/lookatme/English/

単に恥ずかしがりだったり、さまざまな要因で、人の目を見て話せない子供たち向けに、アプリの力を借りてコミュニケーションを取りやすく仕向けるための、サムソンの社会貢献プロジェクト。ゲーミフィケーションの要素も取り入れられています。



« 世界中のどこでも英単語3語で表現して位置を示す「What3Words」



» 子供が手書きで描いたスピードメーターに図柄を差し替えるVolkswagenのプロジェクト「Reduce Speed Dial」

※本記事で紹介しているものは国内未発表・未発売のものを含んでおります。

広告におけるデジタルの役目

The Other Side
- Honda Motor Europe -
Wieden+kennedy
http://www.hondatheotherside.com/

良いお父さんの白い車と、ちょっと悪者の赤い車。キーボードの"R"キーを押すだけで、映像が切り替わるという表現のWebサイト。単なるファミリーカーではなく、走る楽しみも素晴らしいということがとても伝わってきます。

Clever Buoy
- Optus -
M&c Saatchi
https://cleverbuoy.com.au/

海水浴場のサメの出現通知を最新のセンサーを搭載したブイと、衛星通信とスマートフォンデバイスで解決するという案。

Safety Truck
- Samsung -
Leo Burnett Argentina
http://global.samsungtomorrow.com/
the-safety-truck-could-revolutionize-road-safety/

背面にディスプレイを搭載し、トラックの前方の様子が後続車にも見えるトラック。

モバイル部門
(スマートフォン、タブレット端末向けの広告)より

Ea Sports Madden Giferator: An Art, Copy & Code Project With Google
- Ea Sports -
Heat, Grow, Google
http://giferator.easports.com/create/team

実際のフットボールの試合シーンと同じ場面を、ゲーム映像として作ってしまうサイト。素材は一気にソーシャルに広がり、広告としても活用される。

Makeup Genius
- L'oréal Paris -
Mccann Paris
http://www.lorealparisjapan.jp/
makeup_genius/

スマホアプリでメイクアップの指南。スマホのインカメラと顔認識技術の進化により、かなり自然なバーチャルメイクシミュレーションが実現。

Base Phoneaddress
- Base -
Ddb Brussels
http://www.ddb.be/work/BASE/
phone-address/

スマートフォンの現在地を新しい物理アドレスとして利用するためのしくみ。たとえば公園にいるときにスマートフォンでピザを注文し、到着までに公園内を移動していても、ちゃんと届けてくれる。

Backmeapp
- Procter & Gamble -
Leo Burnett Italy

女性が深夜、徒歩で帰宅する途中、スマートフォンアプリを活用して知人に遠隔で見守ってもらうシステム。イスラエルのアプリ。

Sos Sms
- Mexican Red Cross -
Grey México

医療情報を赤十字に登録しておき、緊急時にSMSで問い合わせるだけで、救急隊員が簡易カルテを見られるシステム。

The Fun Queue
- Liseberg -
Shout
http://liseberg.com/en/home/Amusement-Park/
Rides--Attractions/Helix--The-next-level/

遊園地専用アプリ。一緒に競争して楽しむアプリで、結果が良いと遊園地のアトラクションの列をVIP待遇してもらえる。

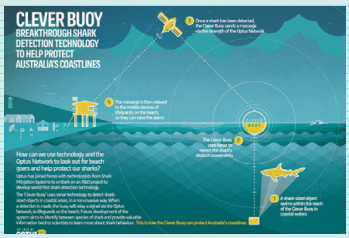
Found
- Mars -
Colenso Bbdo

アプリを活用して、迷子犬を近所の人たちで見つけるためのサービス。

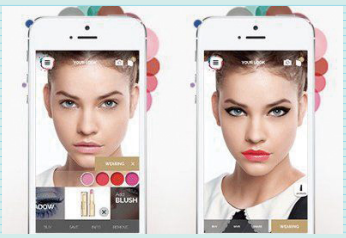
プロダクトデザイン部門

The Lucky Iron Fish Project
- Lucky Iron Fish -
Geometry Global
http://www.luckyironfish.com/

貧血防止のため、調理のときに鉄の板を入れるよう啓蒙していたのだが一向に活用されない。その鉄を小さな魚の形にすることによって祈りや想いを込めることができ、多くの人に使われるようになったという、デザインの力を

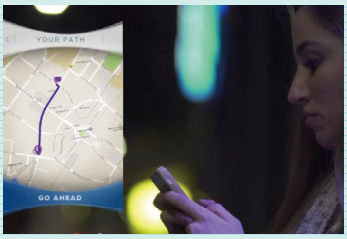


Clever Buoy


Makeup Genius

Found

Safety Truck



Backmeapp

感じるプロジェクト。

この先の広告の形態

SF映画『マイノリティ・リポート』や、最近公開された『ゼロの世界』では、歩く人についてきて話しかけるというプライベート広告が登場します。遠い将来も、形や表現方法は変われど「広告」というものはなくならず発展しているように思えます。

今年のCannes Lionsの作品群から感じられる要素は次のようなものです。

- ソーシャルメディアの活用。口コミで話題を拡げるのは当然のことになっている
- 手法や技法にはこだわらず、何を伝えるのか、何が伝えられたのかが重要視される
- 単に広告主と広告制作との関係だけではなく、運命共同体としての広告の価値が上がっている
- 新しい技術を取り入れつつ、その新しい技術に振り回されない表現が求められている
- 単なる商品広告というだけでなく、社会や社会のしくみにどれだけ影響を与えるかが評価される

デジタル技術やスマートフォン関連技術は進化し、人々のコミュニケーションや、購買欲、知識欲といったさまざまな欲求はますます強くなり、人々の移動量も、得られる情報量も増えてきます。あふれる情報の中で、必要なものを取捨選択し、より必要なものを必要なだけ手にいれるのです。

保険会社ガイコのYouTube広告は、最初の5秒で伝えるべきことをすべて言い切ってしまい、あとは出演者が静止したままという動画広告です。最初だけ観て、すぐクリックして飛ばされてしまうYouTube広告を逆手に取った手法です。広告の形、見せ方も、デジタルの世界に合わせて進化しているのです。SD

Gadget 1

Hammerhead

http://hammerhead.io/

自転車専用の方向指示器

Hammerheadは自転車専用の方向指示器で、一般的に使われているスマートフォン単体で地図画面を見る方法より使いやすく、安全です。開発元のR/GAはNIKE fuelbandの初期の企画／開発にも携わっていました。何か解くべき課題を見つけ、それを解く方法として、スマホアプリやWebサービスに留まることなく専用のハードウェアを作ってしまうのは、1つの大きな流れのようです。名前のとおりシュモクザメの頭の形をしており、LEDの点滅によって方向を示し、視線を移さず判断することができます。

Gadget 2

Write More

http://issueplusdesign.jp/writemore/

筆記音の出るお絵描きボード

書くことが楽しくなるお絵描きボード。カリカリ、サラサラといった「筆記音」が出ます。書くときの音を大きくすることで、より筆記用具を使うモチベーションになったり、正しく描く、素早く描く、注意深く描くことの手助けにもなるそうです。音を楽しみながら、ひらがなを学べる練習帳が用意されています。この切り替え可能な筆記音の強調フィードバックは、クロスモーダル（感覚間相互作用）と呼ばれる研究を活かしたもので、iPhoneアプリとWrite Moreボードとを一緒に使用して実現しています。

Gadget 3

Owlet Baby Care

https://www.owletcare.com/

乳幼児の様子を知るデバイス

乳幼児の足首につけるデジタルデバイス。赤ちゃんの様子を少し離れた部屋でもリモートで知ることのできる専用アプリが用意されています。「スマートソックス」と呼ばれるこのデバイスは、心拍や酸素レベルを知ることができ、常に赤ちゃんのことを気にしていなければいけない親に、少しの間、代わりをしてくれるデバイスです。バッテリーは約2日間持ち、絶縁ケースにくるまれているため感電などの心配もないそうです。赤ちゃんが寝返りしたときにはアラートで知らせてくれます。250ドルで販売の予定です。

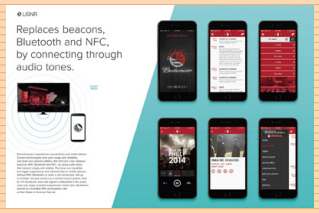
Gadget 4

Lisnr

http://lisnr.com/

音トリガーデバイス

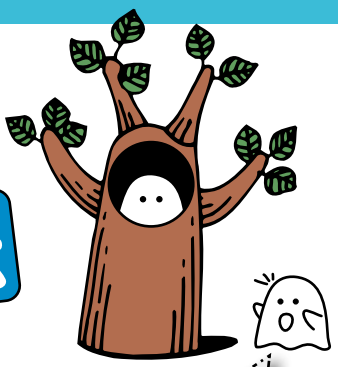
人間の耳に聞こえない高周波をトリガーとして活用できるデバイス。その手軽さにより、NFCタグの用途を置き換えるとも言われています。R/GA Acceleratorという広告企業によるスタートアップ支援プロジェクトから出てきたプロダクトです。Lisnrの標語は「音でデータの力を解き放つ」です。ショップのBGMや映画館のアナウンス、テレビ番組から流れる音などさまざまな音に信号を載せることができます。さらにAPI、SDK、ポータルサイトの提供で、多くの開発者が利用できるようになる模様です。



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

28

結城浩の再発見の発想法



Token

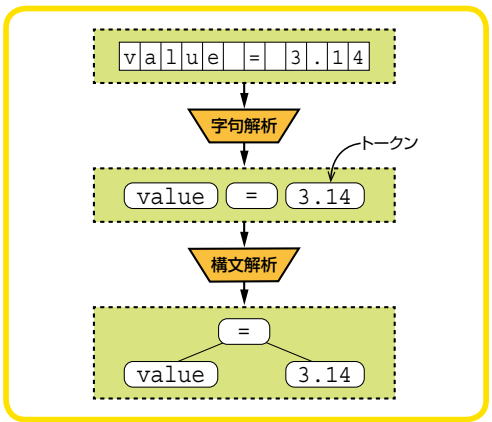
Token——トークン

トークンとは

トークン(Token)とは、**何かを表す1つのまとまったもの**のことです。もともとは古い英語で「しるし」という意味の言葉から生まれた単語らしいです。

「何かを表す1つのまとまったもの」では抽象的すぎるので具体例を挙げましょう。コンパイラがソースをコンパイルするときには最初に「字句解析」と「構文解析」という2つの処理を行います。字句解析はソースを1文字ずつ読んでトークンの列に変換する処理で、構文解析はトークンを1個ずつ読んで構文木を作る処理です。この様子を**図1**に示します。ソース中に現れる文字をいくつか合わせた**文法的に意味のあるまとまり**がここでのトークンなのです。

▼図1　字句解析と構文解析



2段階の処理にしないで、まとめて処理すればいいのに——と思いたくなりますが、それを行うと、コンパイラが複雑になってしまうのです。value = 3.14という代入文から構文木を作るとき、v a l u e = 3 . 1 4という10文字からいきなり作るのではなく、value という変数名、=という代入演算子、そして3.14という数値リテラルという3つのトークンから作ります。人間が"value"という文字の列をまとめて変数名として認識するように、コンパイラも、いったんトークンというまとまりを作ったほうが処理しやすいのです。

トークンリング

トークンのまったく別の例として、今はもう使われなくなりましたが、**トークンリング**というネットワークについてお話しします。

トークンリングでは、通信を行うコンピュータたちが円環(リング)状に接続されています。そしてそのリングの上を1つのトークンがぐるぐる回っています(**図2**)。ここでのトークンは「送信権限のしるし」として使われます。つまり、トークンを持っているコンピュータだけがデータを送信できるのです。

データを送信したいコンピュータは、自分にトークンがやってくるのを待ちます。トークンが流れてきたら、そのトークンの上に送信したいデータと宛先を乗せ、またトークンリングに

流します。データが乗ったトークンを受け取ったコンピュータは、宛先が自分だったらそのデータを受信し、自分以外だったら次のコンピュータにトークンを回します。トークンリングでは、このようにトークンをぐるぐる回して通信を行います。

「ネットワークを流れているトークンは唯一である」という条件で、送信の衝突が起きないようにしているのですね。

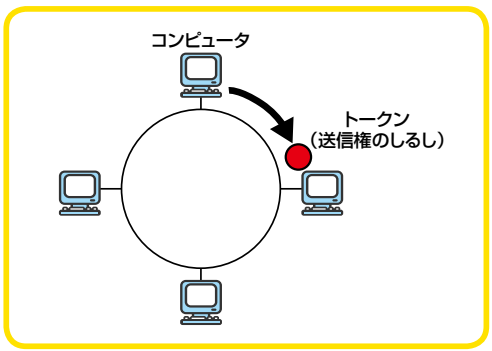
ゲームセンターのコイン

一般の生活でもトークンは使われています。たとえば、ゲームセンターで使われている**ゲーム用のコイン**はトークンの一種です。ゲームをしたい利用者は、お金を払ってコインを購入し、ゲームを行うときにはコインだけを使うことになります。

コインというトークンはどんな役割を果たしているでしょうか。

まず、ゲームセンターに設置されているゲームマシンをシンプルにする効果があります。ゲームマシンに実際のお金を処理する機能(両替やお釣りの機能)を付ける必要がなくなり、1種類のコインだけを処理すればいいからです。お金をコインに変換する両替機は、文字の列をトークンに変換する字句解析と少し似ていますね。また、コインというトークンがあれば、ゲームセンターの運営者はお金の管理が楽になるでしょう。お金を扱うところをコイン両替の場所だけに集中できるからです。さらに、コインはゲームセンターの外では価値がありませんから、盗難防止の効果もあるでしょう。

▼図2　トークンリング



トークンと定型化

字句解析で作られるトークンも、ゲームセンターのコインも、**定型化**の役目を果たしています。種類が多すぎては扱いにくいので、トークンという形に定型化して管理を楽にしようという発想です。

この場合、目的によって「何種類にそろえるか」は変化するかもしれません。ゲームセンターのコインは1種類ですが、カジノで使うチップは金額によって何種類もありますね。

トークンと唯一性

トークンリングで使われているトークンは、「同時に2ヵ所に存在できない」という制約を持っています。これは現実世界の物体が持つ制約を仮想的に作り、それを使って排他制御しているのです。複数のものが並行に動作しているときにはどうしても衝突が発生しますが、トークンが持つ**唯一性**を利用するのは一法ですね。

たとえば、カンファレンスでは「発言者用のマイク」がトークンの役割を果たすことがよくあります。複数人が同時に話し出したら収集がつかなくなるので、「発言する権利」を持っているしるしとしてマイクを使うのです。

トークンの個数は、同時に活動できる主体の最大数を定めます。たとえば、カンファレンスで使うマイクを2本にしておけば、同時に発言できる人を最大2人に設定したことになります。こうしておけば、質問者と解答者の2人だけが発言できる状況を自然に作れますね。

◆　◆　◆

あなたの周りを見回して、多種類が混乱を生んでいるものをトークンに変換できないでしょうか。あるいは、複数のものが衝突する状況をトークンで整理できないでしょうか。

ぜひ、考えてみてください。SD

耽溺せよ電子工作

おとなラズパイリレー

第11回「Raspberry Pi 2を大人買いしてLinuxクラスタを作ろう（前編）」

竹迫 良範

おとなラズパイリレーは、Raspberry Piを文字どおり「リレー」し、好奇心旺盛なITエンジニアが電子工作をするという企画です。前編で構想を練り、後編で実装します。1年を通してどんなデバイスができあがるのか？……今回は、日本最大のハッキングコンテストを運営するSECCON実行委員長の竹迫良範さんによるRaspberry Piのミルフィーユ？――です。

Writer 竹迫 良範（たけさこ よしのり） サイボウズ・ラボ株式会社

Raspberry Pi 2を10台大人買いしてみる

1GBのメモリとARM Cortex-A7 900MHzクアッドコアCPUを搭載したRaspberry Pi 2がたったの$35で購入できるということで、思い切って10台ほど大人買いしてみました。

用途はあとで考えるとして、10台のRaspberry Pi 2上でLinuxを同時に起動できればPCクラスタシステムを安価に構築することができます。持ち運びが可能なちょっとしたスパコンを自作する気分になれるので、まずは物理設計から始めましょう。

Raspberry Pi 2を安価にラッキングする方法

Raspberry Pi 2のケースは1個あたり1,000円前後するのですが、これだと10個買うと1万円程度になってしまいます。また、密閉型のケースだと十分に排気されずCPUの熱がこもりやすくなるので、長時間動かすと高温になってしまい動作が不安定になってしまうリスクも考えられます。Raspberry Pi 2の基板には直径3mmの穴が4個あいているので、ここに2cm以上の棒を挿しこんで立てていけば積み上げることができそうです。ちょうど良い形状の六角オネジ・メネジ「MB3-20」が秋月電子通商に売っているのでこれを買いました（**写真1**）。

▼**写真1** 六角オネジ・メネジ「MB3-20」の外観

「MB3-20」は1個だと30円なのですが、100個以上まとめて購入すると単価が19円になるので、これもごっそり100個（1,900円）で買ってしまいましょう。「MB3-20」はL=20mm、L1=6mmの大きさで、中の穴は直径3mmなので、ラズパイをスタックするにはちょうど良い大きさなのです（**図1**）。

ただし、Raspberry Pi 2の基板にある3mmの穴にはネジの切込みが入っていないため人間の指の力で挿入することはできません。ペンチで六角ネジの側をつかんでねじ込めないことはないのですが、不安定で作業効率が悪いので、六角ラチェットドライバーを調達してきます。

実は5.5mmの六角ビットを挿入できるラチェットドライバーの種類は少なく（※JIS規格、ISO規格では5mmの次の大きさが6mmと決められているため5.5mmの対応数が少ないのです）、たまたま電器店で見つけたのがE-Value T型ラチェットドライバーセット「ERD-3」です（**写真2**）。

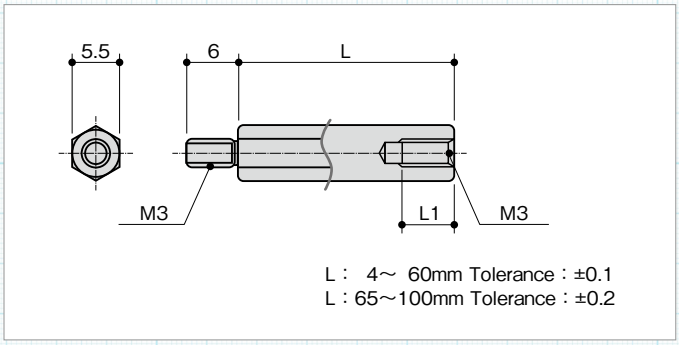
型番「ERD-3」を検索すればAmazonでも購入できます。このドライバーを使うと作業効率が大幅にアップするので、Raspberry Pi 2を積み重ねる作業では必需品となります。ぜひ一家に1セット常備しておきましょう。このセットには5.5mmの六角ビットが付属しており、ラチェットドライバーの差込角ソケットに挿入して使用します。

Raspberry Pi 2の4個の穴に六角ネジを配置して、ネジが斜めにならないよう慎重に垂直にねじ込んでいきます（**写真3**）。

ここで基板から少しネジがはみ出るのがミソで、このはみ出たマージンを利用して次のラズパイを重ねていくことができるのです。10台のRaspberry Pi 2全部を連結するのはちょっと不安なので、5台セットで2個組み上げてみました（**写真4**）。

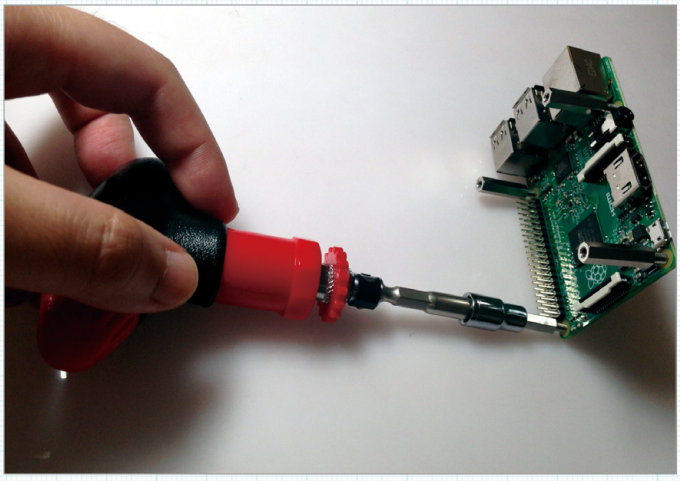
Raspberry Pi 2の個体を認識できるようにするためラベルプリンタであらかじめ番号を振っておきます。12mmのラベルテープを使うとLANコネク

▼図1 六角オネジ・メネジ「MB3-20」はL=20mm、L1=6mmなのでラズパイにぴったりフィット



▼写真2 E-Value T型ラチェットドライバーセット「ERD-3」

▼写真3 斜めにならないよう垂直にねじ込んでいく



タの横にちょうど貼ることができて視認性も良好です。

まさかの電源供給問題が発生

10台のRaspberry Pi 2にまとめて電源を供給するために手元にあったUSBハブを試したところ、1ポートあたり最大0.4Aの電源供給力しかなく、Linuxの起動途中でカーネルパニックになるなどRaspberry Pi 2が正常に動作しませんでした。

Raspberry Pi 2に同封されていた取扱説明書を改めてよく見てみたところ、5VのDC電源で最大1,500～2,000mAの供給が必要と書いてありました。これは明らかに電力供給不足です。

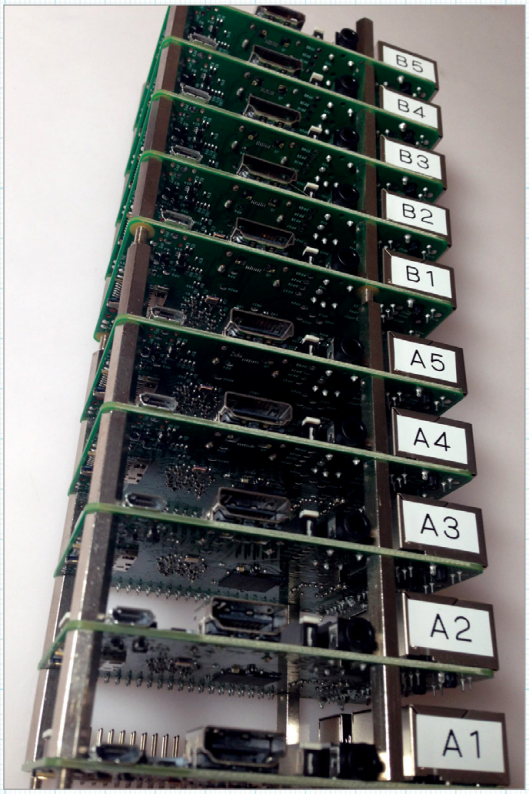
昔の機種のRaspberry Pi Model Bでは最大700～1,500mAだったようで(**写真5**)、Raspberry Pi 2でCPUの性能が上がった分、電力消費も大きくなっているようでした。

組み込み機器だと意外とこういった基礎的な部分で問題が起きてしまうものです。

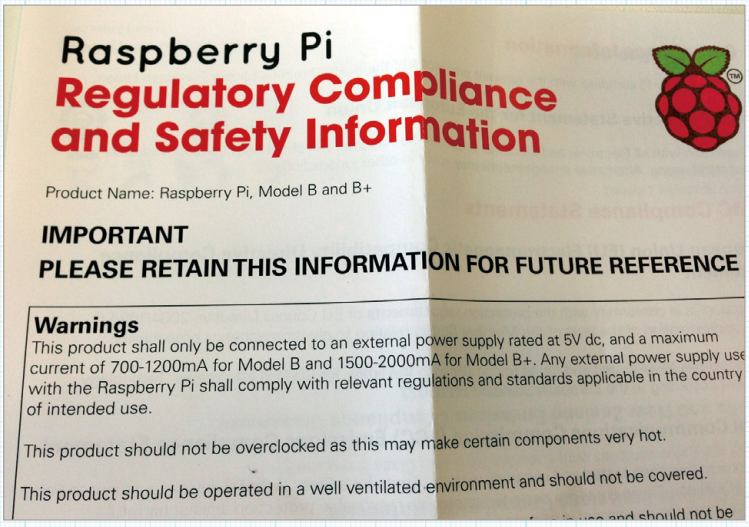
しかし、よくよく考えると普通のUSB充電器でも最大1A程度しか対応しないものが多いので、2Aまで対応しているものを選ぶとなるとちょっとたいへんです。USB充電器やバッテリーで困ったときはAnkerということで、調べてみると6ポートで最大12Aのフルスピード充電(各ポートごとで最大2.4A)が可能なUSB 60W急速充電器「A2123511」がAmazonで売っていましたので即買いです(**写真6**)。

最近はスマートフォンやタブレットなどUSBで充電する機器も増えてきているので、何台か買いそろえ

▼**写真4**　Raspberry Pi 2タワーの完成



▼**写真5**　Raspberry Piの取扱説明書には最大消費電力が記載されている


Raspberry Pi
Regulatory Compliance and Safety Information

Product Name: Raspberry Pi, Model B and B+

IMPORTANT
PLEASE RETAIN THIS INFORMATION FOR FUTURE REFERENCE

Warnings
This product shall only be connected to an external power supply rated at 5V dc, and a maximum current of 700-1200mA for Model B and 1500-2000mA for Model B+. Any external power supply used with the Raspberry Pi shall comply with relevant regulations and standards applicable in the country of intended use.

This product should not be overclocked as this may make certain components very hot.

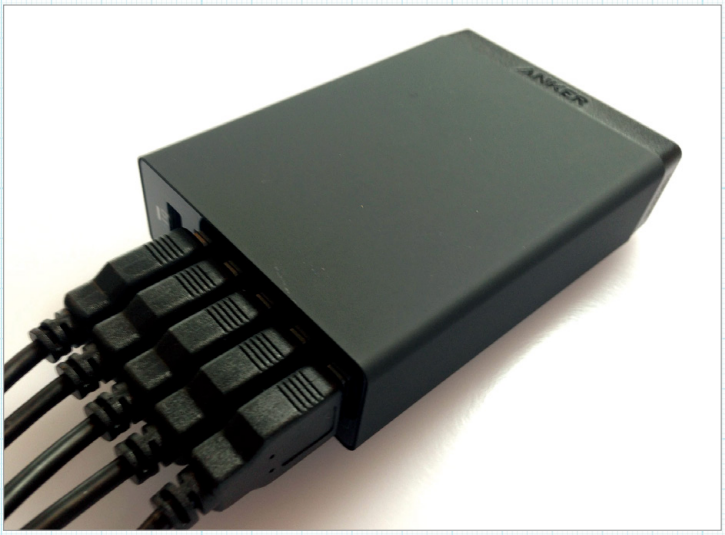
This product should be operated in a well ventilated environment and should not be covered.

ておいて損はないでしょう。ここでも大人買いです。

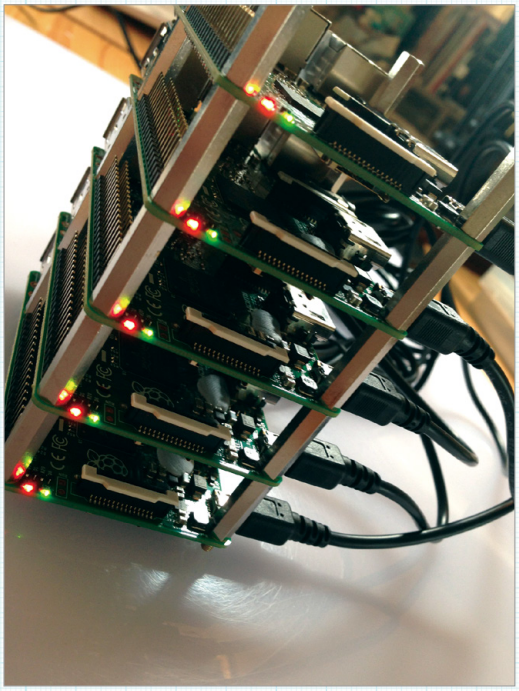
microUSBケーブルを必要本数買いそろえて、Raspberry Pi 2タワーに電源供給です(**写真7**)。USB簡易電圧・電流チェッカーもAmazonや上海問屋で探せば売っているので、実際の電源の供給がどの程度なのかUSBケーブルの間に差し込むとわかります(**写真8**)。

USB電流チェッカーがあると、LANがリンクアップすると消費電流が0.20A→0.27Aに増えたり、USBキーボードを接続すると0.32Aになったり、microSDカードの読み書きが大量に発生すると一時的に0.42Aになったりと、リアルタイムで数字が見えてわかるので、これを眺めているだけでも相当面白いです。USB電流チェッカーも一家に1個買いそろえておくと良いと思います！

▼**写真6**　Anker 60W 6ポートUSB急速充電器「A2123511」



▼**写真7**　microUSBケーブルで接続して電源供給



タワーRaspberry Piに命を吹き込む

次回はRaspberry Pi 2のLinuxクラスタを安定稼働させるための実装テクニック編です。SD

▼**写真8**　USB簡易電圧・電流チェッカー

● **執筆協力**

RSコンポーネンツ㈱Raspberry Piに興味のある方は次のサイトをチェック
http://jp.rs-online.com/web/generalDisplay.html?id=raspberrypi

かまぷの部屋

第14献 ゲスト：篠田 佳奈さん

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

篠田 佳奈（しのだ かな）さん

㈱BLUE代表。日本発のセキュリティ国際会議「CODE BLUE」を運営。米国留学を機にITを学び、セキュリティの調査研究、翻訳やコンサルティングなどを中心にキャリアを積む。情報セキュリティ国際会議「Black Hat Japan」の企画運営のほか、APWGやSECCONなど、国内外問わずさまざまなイベントに携わる。2014年には「TEDxKids@Chiyoda」で講演を行う。
CODE BLUE：http://codeblue.jp/2015/
Facebook：https://www.facebook.com/kana39

（鎌田）篠田さんは情報セキュリティ分野で活躍されていますが、まずはご出身や学生時代、キャリアのことを簡単に教えてください。

（篠田）出身は奈良県です。ITとの出会いは遅く、初めてコンピュータに触ったのは高校の授業ででした。そのころは、特別にコンピュータに興味を持ってはいませんでした。高校卒業後はジュエリーショップや不動産屋など、いろいろなことをしていました。そのころのキャリアプランは「手に職を」ではなく「結婚して母親になる」だったんです。

現在のようなキャリアを目指されていたわけではないんですね。

小学生の頃にTVで見た通訳さんが素敵に見え、育児しながら通訳ガイドもいいかなと考え、留学を決めました。今思えば、通訳さんが自分の知らない言語の暗号化と復号をしていて魅力的に見えたのかもしれません（笑）。米国留学後、進路を決めるときにカウンセラーから勧められたのが『コンピュータ学科』でした。

それがこの業界に入るきっかけだったんでしょうか。どうしてコンピュータ学科だったのでしょうか。

スカラシップ（奨学金）をとりながらアルバイトを4つ掛け持ちしていたので、勉強する時間が限られていました。ですので、より実用的で、大量の書物と英文レポート提出を必要としないコースを希望したのです。それでネットワーク技術を学ぶことになったのですが、自分にピッタリはまりました。プログラミングの学習でコードを書いたり、BSD環境でネットワークのノード解析をしたり、トラブルシュートしたり、かなり実用的な授業でした。おかげで、時間の余裕と、将来役に立つ実践的な技術が身につきました。

そこでの経験から、コンピュータ関連の職業に就くことに決めたのですね。

卒業して帰国することになり、国内で就職活動をしました。じつは米国では、新卒のエンジニアでも高給なのですが、日本はちょうどそのころ、就職氷河期で、条件は悪かったのです。それでも帰国子女特有の押しの強さ（笑）で何社かから内定をもらいました。自分のできることをアピールしたのがよかったのだと思います。その中で人事の人が優しそうで、研修がしっかりしていそうなプロシードを選びました。

その後、何社か転職されているようですが。

セキュリティのサポートからセキュリティリサーチャー、コンサルタント。そしてカンファレンスのコー




ディネータ、という流れです。リサーチャーをしていたころ、ちょうどDES（Data Encryption Standard）からAES（Advanced Encryption Standard）へ暗号方式が変わるときでした。そこで数学を一から勉強しなおし、暗号技術をより理解できるようになりました。途中でOpen PGP（Open Pretty Good Privacy；暗号化ソフトウェア）のRFC文書の翻訳もしました。大きな転身のきっかけは『Black Hat』というセキュリティイベントとの出会いです。「Black Hat Japan の運営をやってみないか？」というお話しが来たんです。

Black Hatで苦労されたことはありましたか？

たとえば、脆弱性を見つけて初めて攻撃対策がとれる、パッチを当てることができるので、セキュリティ研究者は米国などでは尊敬されるのですが、まだまだ当時の日本ではハッカー＝クラッカーとして『けしからん的なもの』と見られ、スポンサー集めに苦労しました。私は海外の動きばかりみていたので、この価値観のズレがそこまで大きいことは知らずに苦労しました。真面目な国際会議なのに……。

スポンサーを集めるのに苦労されたのですね。どうされたのですか？

あらゆる努力をしました。会場ホテルを含む大幅な値切り交渉や、細かいコストカット、価値がわからない方々へめげずに説明する、営業・広報・受付・通訳翻訳・国内海外との調整・手作り資料、なんでもやりました。ここでの経験があって、今の『CODE BLUE』があります。世界で活躍するハッカーたち、世界で十分通用する非英語圏のハッカーたちに、スポットライトがあたるステージを用意したかったのです。

CODE BLUEを創設した理由は何ですか？

周辺国には当たり前にある、その国発の会議が日本にはないことを不思議に思い、Black Hatのようなベンダ中立な日本発の国際会議を作りたかったのです。Black Hatは、そこで講演すると一目置かれる世界最高峰のカンファレンスです。CODE BLUE初回の基調講演をしたJeff Moss氏が発起人です。もともとは、彼が主催するメーリングリストのオフ会をベガスで開催したものがDEFCONとなり、さまざまな種類のハッカーが集まる場となっていったのですが、その数年後「真剣に討議できる場が別にほしい」との声が高まったことでBlack Hatが生まれました。CODE BLUEの「BLUE」は海を表現して、日本は海に囲まれて鎖国になっている、それを「CODE（技術）」の海でつなげていくという意味があるんです。

今後のCODE BLUEの目標は何ですか？　人材育成でしょうか？　安全なIT社会の形成でしょうか？

その両方かな。海外のカンファレンスに出るたび、ムーヴメントを肌で感じています。日本のためだけではなく、アジア全体に埋もれている有能なハッカーを世界に発信したい。トップの層を動かして全体の人材の底上げをしていきたい。このイベントでいろんな層の人に関わってもらいたい。一石二鳥だけではなく三鳥も四鳥も期待されています。CODE BLUEはみんなのいろんな夢を抱えている船なんですよ。大変ですが、やれるだけやってみようと思っています。

今年のCODE BLUEは10月に開催されるそうですが。

はい。10月28～29日に東京のベルサール新宿グランドで行います。興味をお持ちの方はぜひ、ご参加ください。

今日は真剣なお話、どうもありがとうございました。 SD



ツボイのなんでもネットにつなげちまえ道場

第三回

LED点滅の極意（中編）

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

はじめに

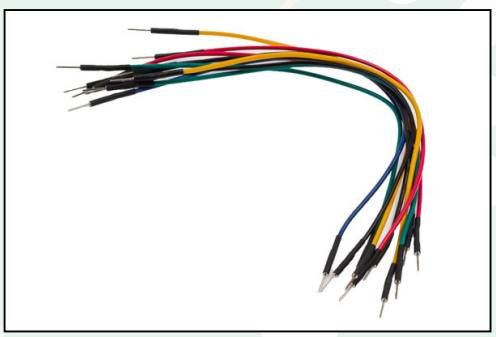
前回はmbed LPC1768に搭載されているLEDを点滅させてみました。今回は、ブレッドボードを使って、電流制限抵抗とLEDをマイコンに接続して点滅させてみましょう。

ジャンパワイヤ

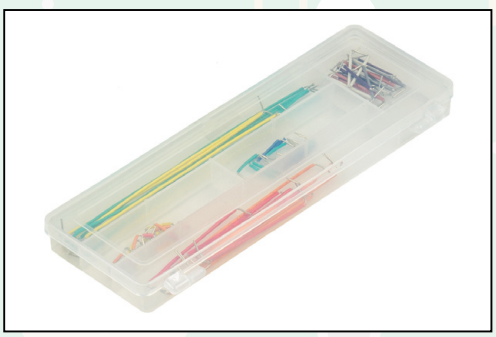
ブレッドボードを使って配線をするには、ジャンパワイヤという電線を使います。ジャンパワイヤには大きく分けて、柔らかくて長いもの（**写真1**）と、短くて固いもの（**写真2**）の2種類があります。どちらも機能は同じで、ブレッドボードの穴にジャンパワイヤの端を差し込み、ブレッドボードの中でつながっていない場所どうしを接続するためのものです。

使い分けにはとくに決まりはなく、好みでよいでしょう。短くて固いもののほうがいろいろな長さのものが入っていて安価で、ブレッドボードの表面から浮き上がらずに配線できるので、不意に抜けてしまうことが少ないです。一方で、長くて柔らかいもののほうが、配線の自由度が高く、ジャンパワイヤの長さを気にせずに簡単に配線ができるというメリットがあります。筆者は、両方を併用して配線しています。

▼写真1　柔らかくて長いジャンパワイヤ



▼写真2　短くて固いジャンパワイヤ



つなげてみる

今回必要になる部品は**表1**のとおりです。

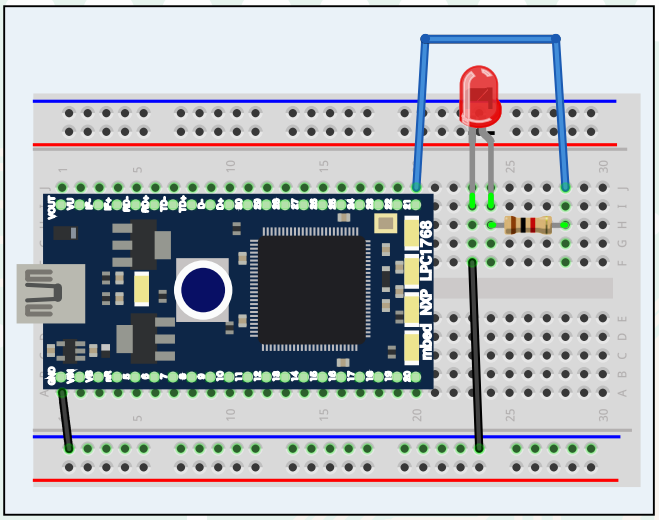
これらの部品を使って、つなげてみましょう。**図1**に接続例を記します。

電流制限抵抗には、1kΩの抵抗を使いました。1kΩの抵抗は、本体に茶色、黒色、赤色、金色の順で帯（カラーコード）が刷られています。スルーホール実装用の抵抗の多くは、最初の2桁と、3番目の指数を使って抵抗値を表しています（**図2**）。この場合、茶色は1、黒は0で10を、

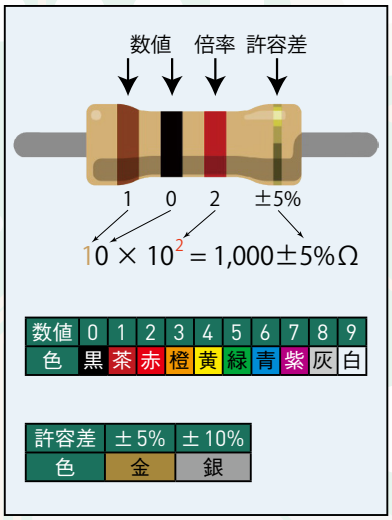
▼表1　部品表

部品名	入手先	参考価格
mbed LPC1768	ssci.to/250	¥5,940
ブレッドボード	ssci.to/313	¥270
固いジャンパワイヤ	ssci.to/314	¥270
抵抗コンデンサLED詰め合わせパック	ssci.to/1218	¥680

▼図1　接続例



▼図2　抵抗のカラーコード

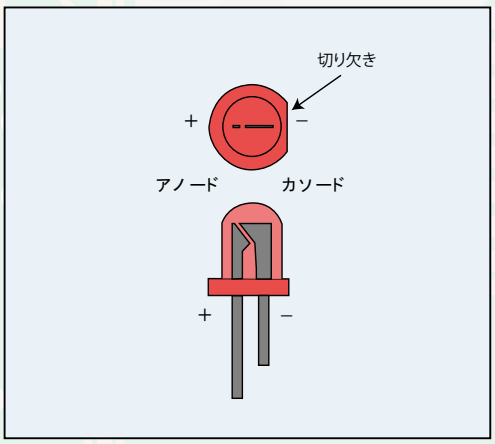


赤の2は10^2の指数部分を表しています。この場合、10×10^2=1,000で1kΩです。最後の金色は、抵抗値の許容差が±5%であるということを表しています。こういった抵抗の中には薄い炭素の膜が入っていて、電気を流れにくくする役割を果たしています。抵抗は取り付ける方向が決まっていませんので、どちら向きに取り付けてもかまいません。抵抗をブレッドボードに差し込むため、適当に抵抗の両端にある針金部分を曲げてください。

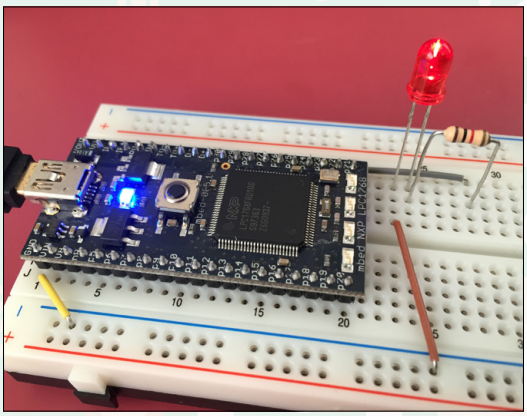
LEDは、第1回で述べたように、取り付ける方向が決まっています。今回は赤色のLEDを使ってみましたが、この赤色の樹脂の中を見てみると、2つの金属板が入っているのが見えると思います。たいていの場合、この金属の板のうち、大きいほうがカソード（マイナス極）です。ただ、LEDによっては大きさの関係が逆だったりします。樹脂部分の切り欠きがある側がカソードだったりもしますので、参考にしてください（**図3**）。

接続し終えると、**写真3**のようになります。抵抗やLEDについている線（リード線）を曲げただけで挿し込むと、このようにブレッドボードから部品が高く浮き上がってしまいます。このくらいの部品数であれば、ブレッドボードの上

▼図3　LEDの切り欠きの向き



▼写真3　実際の接続例



にある部品も少なく、ショートさせてしまう危険は少ないのですが、やはり切ってしまったほうがよいでしょう。ジャンパワイヤのブレッドボードの穴に差し込む部分程度の長さに部品のリード線を切ってみましょう。部品のリード線を切るには、マイクロニッパーという工具を使います。ニッパーには、プラモデル用のもありますが、ここでは金属を切るためのものを使いましょう。百円均一ショップなどでも売っていますが、筆者は㈱エンジニアのNP-05[注1]を愛用しています。

ソフトウェア

今回は、LEDをmbed LPC1768のp21に接続しました。ですので、前回基板の上にあるLEDを光らせてみたコードを少し書き換えて、p21の出力をコントロールするように変更します(**リスト1**)。

mbedの開発環境は、マイコンの入出力操作を抽象化して手軽に使えるように作られています。mbedのオンラインコンパイラを立ち上げ、前回作った「mbed_blinky」のDigitalOutで引数として渡しているLED1をp21に変更してください。あとはビルドし、バイナリファイルをダウンロードしておきます。そして、mbed LPC1768をパソコンに接続して認識されたドライブにその枚なりファイルをドラッグ&ドロップしてコピーします。

注1) http://www.engineer.jp/products/nipper/np01/item_03/np-05

▼リスト1　LEDを点滅させるプログラム

```
#include "mbed.h"

DigitalOut myled(p21); // DigitalOutクラスのコンストラクタ

int main() {
  while(1) {
    myled = 1; // 指定されたGPIOポートをHIGHに
    wait(0.2); // 0.2秒待つ
    myled = 0; // 指定されたGPIOポートをLOWに
    wait(0.2);
  }
}
```

HIGHとLOW

リスト1のとおり、LEDを点滅させるために、myledをTrueにしたりFalseにしたりしています。myledに1を書き込むと、p21はHIGH、つまりマイコンの電源(3.3V)に接続された状態になり、LEDに電流が流れてLEDが点灯します(**図4**)。

myledに0を書き込むと、p21はLOW、つまりGNDに接続された状態になります。LEDの両端がGNDに接続されるため電流は流れず、消灯します。

図4では、わかりやすくするためにマイコンの中にスイッチがあるように描きました。しかし、実際にマイコンの中に機械的なスイッチが入っているわけではありません。実際には中にトランジスタが入っています。I/Oレジスタと呼ばれる記憶領域に書き込まれた値に応じて、このトランジスタをONしたりOFFしたりして、ピンの状態が変わります。**図5**(マイコンの内部構造)のように、ピンに対応するI/Oレジスタにソフトウェアで値を書き込むと、トランジスタがマイコン内部の電源とピンをつないだり、GNDとピンをつないだりします。

このトランジスタも半導体ですので、流すことのできる電流には制限があります。今回、電流制限抵抗は1kΩを使いました。また、赤色LEDの順方向電圧はたいてい2V程度ですので、第1回で紹介した計算式、

$$(V-Vf)=If\times R \quad (電圧=電流\times抵抗)$$

を使うと、(3.3-2)=If×1,000から、順電流 If=0.0013A=1.3mAということがわかります。もっと抵抗値の小さい330Ωを電流制限抵抗に使っても、順電流は4mA程度です。mbed LPC1768に搭載されているマイコンのデータシートに掲載されている図(**図6**)を見ても、6mA程度流しても3Vが得られることが

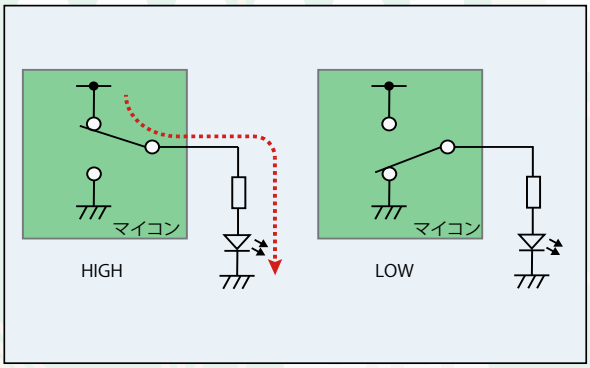
わかります。この程度の電流ならば流せることが確認できます。

こうして流す電流に応じて電圧が下がってしまうのは、トランジスタ（正確にはFET、電界効果トランジスタ）には「オン抵抗」と呼ばれる抵抗が内部にあるからです。トランジスタがONになっているとき、スイッチはつながっているものの、そこには抵抗があります。結果、マイコンがHIGHのときには、マイコンの中の電源からトランジスタのオン抵抗と電流制限抵抗を通じてLEDが接続されていることになります。

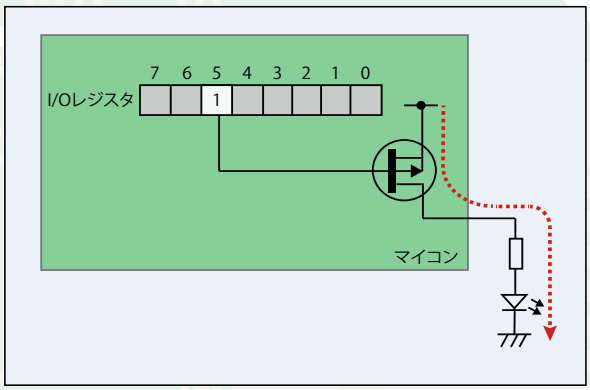
このオン抵抗の値はFETのデータシートに書いてあります。なぜかマイコンのデータシートではたいてい、**図6**のように出力電圧として表記されています。LEDに流す電流を正確に求めるには、このオン抵抗を計算に入れることになりますが、そもそも使っている抵抗の許容差が±5%もありますから、筆者は普段、気にしていません。

マイコンのピンで電流を流したり止めたりできるのであれば、モーターを回すことができるかもしれないと考えるかもしれません。しかし、マイコンにモーターを直接つないで回すことはできません。モーターが消費する電流は、オモチャなどで使われるものでも500mA以上あるからです。より大きな電流をマイコンでコントロールするには、トランジスタやモータードライバと呼ばれる半導体を使う必要があります。SD

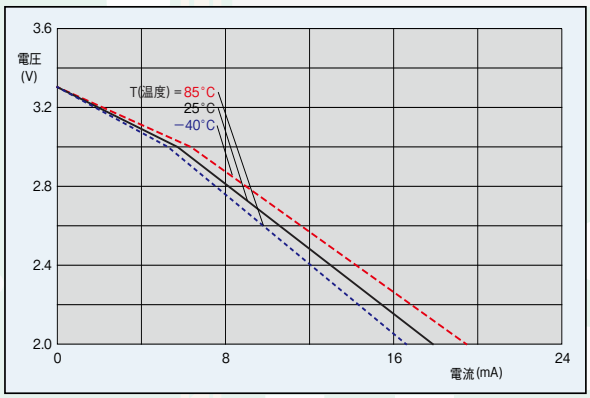
▼図4　HIGHとLOW



▼図5　マイコンの内部構造



▼図6　LPC1768のピンが出力できる電流と電圧



コラム　High-drive ?

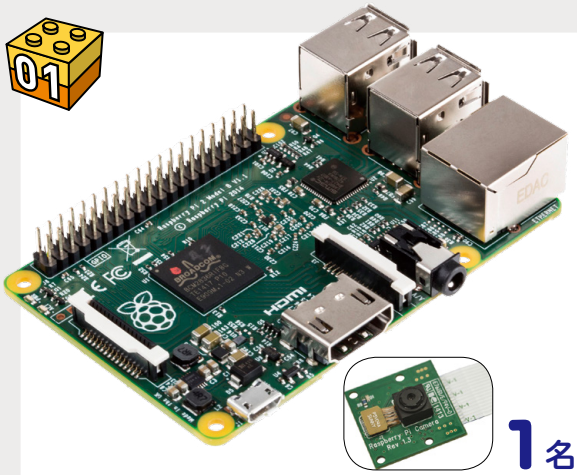
マイコンによっては、High-drive outputなどという名前で、もっと大きな電流を流しても電圧が下がらないピンを搭載しているものもあります。こういうマイコンのピンには、オン抵抗の値が低いトランジスタが中に入っています。

読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」からアクセスし、アンケートにご協力ください（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入力いただいた方の中から抽選でプレゼントを差し上げます。締め切りは**2015年9月17日**です。プレゼントの発送まで日数がかかる場合がありますが、ご了承ください。

ご記入いただいた個人情報は、プレゼントの抽選および発送以外の目的で使用することはありません。アンケートの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。入力いただいた個人情報は、作業終了後に責任を持って破棄いたします。

01 「Raspberry Pi 2 Model B」&「Camera module」セット



1名

旧型「Raspberry Pi Model B+」から、速さ最大6倍（クワッドコアARM Cortex-A7）・メモリ容量2倍（1GB RAM）と格段にパワーアップしました。さまざまなデバイスと接続して自分だけのガジェットが作れます。今回はRaspberry Pi本体と接続できる、5Mピクセルセンサー搭載の「カメラモジュール」とセットでのご提供です（ロゴ入りのペンケースもお付けします）。

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

02 USBチャージ機能付き雷ガードタップ P3U3-JP

1名

3つのACコンセント（計14.5A）、3つのUSBポート（計5V/2.1A）を持つ雷ガードタップ。両方に、サージ保護機能が付いています。また、タブレット端末を立てかけられるスタンドが付いており、充電しながらの操作ができて便利です。

提供元 シュナイダーエレクトリック　http://www.apc.co.jp/

03 はてなTシャツ2015 Mackerel Tシャツ

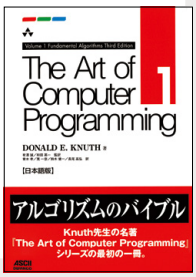
ブログサービスなどを展開する、はてなのノベルティTシャツ。"Mackerel"は本誌の連載でも取り上げている、同社が展開中のサーバ監視ツール。さまざまな外部ツールと連携して、簡単にサーバを管理できるのが特徴です。Tシャツのサイズはしサイズのみとなります。

提供元 はてな
http://www.hatena.ne.jp

mackerel 3名

04 The Art of Computer Programming Volume 1

Donald E. Knuth 著



世界的に有名な計算機科学者クヌース博士によるアルゴリズムの名著、その第3版の翻訳本。アスキーが2004年に発行したものを、今年新しく発足した新レーベル「アスキードワンゴ」が再刊行しました。

提供元 ドワンゴ
http://info.dwango.co.jp

2名

05 できるPRO Red Hat Enterprise Linux 7

平 初、できるシリーズ編集部 著

新人のITシステム管理者向け、RHEL 7の入門本です。Web、DNS、メール、DBといった各種サーバの構築・運用管理を学べます。Dockerの実行についても章を設けて解説しています。

提供元 インプレス
http://www.impress.co.jp

2名

06 クラウドを支えるこれからの暗号技術

光成 滋生 著

クラウドサービスなどで利用される新しい暗号技術を中心に、暗号の基礎および暗号技術を支える数学を解説した1冊です。「Developers Summit 2015」で行われた人気の講演を書籍化したものです。

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

2名

07 Dockerエキスパート養成読本

杉山 貴章 ほか 著

コンテナ型仮想化技術「Docker」を特集した、8人の著者によるムック本。Dockerの概要と最新動向、ソフトウェア開発・運用の現場で活用するためのノウハウをゼロから学べます。

提供元 技術評論社
http://gihyo.jp

2名

エンジニアの夏期講習

第1特集

特講

正規表現・SQL・オブジェクト指向
苦手克服のベストプラクティス

IT 業界に入った多くの若い技術者が驚くのは、文字コードの違いや検索をいかに効率化するかといったデータの扱い方ではないでしょうか。またデータベースから望みのデータを選び出す方法「SQL」も習得しておかねばならない重要な技術です。さらにソフトウェア開発の現場に出たときはオブジェクト指向も理解していなければなりません。しかし苦手なまま日々業務を過ごしてしまうことも少なくありません。

これら 3 つの技術に対して、数多くの開発現場を経験してきた先輩が先生となり学習のコツを演習形式で特別講義します。題して「エンジニアの夏期講習」。今夏、本誌で学習経験をするか否かで大きくその後の成長で差がつきます。本特集でエンジニアの底力をつけましょう!

CONTENTS

第01時限

エンジニアの共通言語

正規表現をマスターする

アンチパターンから正解を導く

Author とみたまさひろ Twitter @tmtms

基本・実践に分けて「正規表現」を学びます。前半では正規表現の概要、種類、構成要素といった基本を解説。後半では、演習問題を解きながら、メールアドレスをすべて正規表現で表すことに挑戦します。アンチパターンを示しながら解説しているので、どこが間違っているかを考えながら、正解の正規表現を導き出しましょう。

正規表現とは

正規表現は、ある文字列に適合するような文字列を表記するための規則のことです。英語ではRegular expressionと言い、プログラム中の関数名や変数名ではregexやregexpと書かれることがあります。

UNIX系OSのシェルで作業をしていると、ファイル名を指定するのに`*`、`?`、`[...]`などの特殊な記号を使うことが多いでしょう。たとえば、`*.txt`は拡張子が`.txt`であるファイル名に適合しますし、`[abc]*`はa、b、cのいずれかの文字から始まるファイル名に適合します。

正規表現はこのシェルの持つファイル名の適合機能を、より高度にしたものと言えます。シェルの特殊記号と同じような働きを正規表現で記述するとどのようになるかを**表1**に示します。

また、正規表現は通常、文字列の一部に一致します。文字列全体に一致させたい場合は先頭(`^`)と末尾(`$`)を明示する必要があります。たとえば、先ほどのシェルでの`*.txt`と同等の正規表現は`^.*\.txt$`となります。

UNIX系OSでは、grep、sed、awkなどのコマンドで、文字列のマッチングを行うために正規表現が使われます。grepコマンド名中の"re"はregular expressionのことです[注1]。有名なテキストエディタのVimでも、検索や置換には正規表現の知識が必須です。また、UNIX系OS以外でも、プログラム中で文字列処理を行うのに正規表現はよく使用されます。PerlやRuby、JavaScriptなどのスクリプト言語では、数値や文字列と同じように、正規表現を記述するための専用の構文が用意されています[注2]。このことからも正規表現の重要性がわかるでしょう。

基本的な正規表現

実際にgrepを使っていくつかの正規表現を試してみましょう。あるディレクトリの下に**図1**のようなファイル／ディレクトリがあるとします。

注1) Wikipediaには、grepという名前はエディタ「ed」の、正規表現に一致する行を表示するコマンド`g/re/p`(globally search a regular expression and print)が由来と書かれています。

注2) CやJavaでは構文で正規表現をサポートしていないため、正規表現用の関数を呼び出す必要があります。

▼表1　シェルのファイル名適合機能と正規表現の対応

シェル	正規表現	意味
?	.	任意の1文字
*	.*	0文字以上の任意の文字列
[abc]	[abc]	abcのいずれかの1文字

「ls -l」の結果からhoge.txtだけを抽出するためにgrepコマンドを使用します(図2)。しかし、hoge.txtだけでなくhogeetxtも抽出されてしまいました。これは、正規表現で.は任意の1文字にマッチするためです。.をそのままの文字として扱うためには\.と指定します。今度はちゃんとhoge.txtの行だけが抽出されました(図3)。

次に、hogeディレクトリの行を抽出したいとします。hogeと指定するとhoge.txtやhoge.datも出力されてしまいます(図4)。

通常、正規表現のマッチングは対象の文字列中でパターンに一致する文字列が含まれているかどうかを判定します。図4では、hoge.txt中にもhogeが含まれているので、正規表現に適合したとみなされてしまっています。

▼図1　ファイル／ディレクトリの例

```
% ls -l
total 8
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:14 HOGE.txt
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:14 Hoge.txt
drwxrwxr-x 2 tommy tommy 4096 Jun 21 22:14 fuga
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:14 fuga.txt
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:14 hige.txt
drwxrwxr-x 2 tommy tommy 4096 Jun 21 22:14 hoge
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:14 hoge.dat
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:14 hoge.txt
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:17 hogeetxt
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:14 huge.txt
```

▼図2　hoge.txtを抽出したつもりが……

```
% ls -l | grep 'hoge.txt'
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:17 hoge.txt
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:17 hogeetxt
```

▼図3　hoge.txtを抽出

```
% ls -l | grep 'hoge\.txt'
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:17 hoge.txt
```

▼図4　hogeディレクトリの行を抽出したつもりが……

```
% ls -l | grep 'hoge'
drwxrwxr-x 2 tommy tommy 4096 Jun 21 22:17 hoge
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:17 hoge.dat
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:17 hoge.txt
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:17 hogeetxt
```

今回の場合はhogeで終わる行を指定できれば良さそうです。正規表現は文字だけではなく文字の位置を指定することもできます。行末は$で表します。ですので、hoge$と指定すれば「hogeで終わる行」の意味になります(図5)。

ディレクトリの行だけを抽出したい場合は、「dで始まる行」を指定できれば良さそうです。行頭は^で表せるので、^dを指定すれば良いです(図6)。

次に、hoge.txtとhuge.txtを抽出してみます。2文字め以外は共通ですので、2文字めに任意の文字を表す.を指定してみます(図7)。hige.txtも引っかかってしまいました。ほしいのは2文字めがoかuだけです。このような場合は、角括弧でくくられた中のいずれかの文字という指定の[ ]を使用します。今回はh[ou]ge\.txtと指定すればhoge.txtまたはhuge.txtに一致するようになります(図8)。

hoge.txtとfuga.txtの行を抽出してみます。この2つは3文字めがgであること以外共通点はなさそうです。図7のように.を使って..g.\.txtと指定するとhige.txtやhuge.txt

▼図5　hogeディレクトリの行を抽出

```
% ls -l | grep 'hoge$'
drwxrwxr-x 2 tommy tommy 4096 Jun 21 22:17 hoge
```

▼図6　ディレクトリの行を抽出

```
% ls -l | grep '^d'
drwxrwxr-x 2 tommy tommy 4096 Jun 21 22:17 fuga
drwxrwxr-x 2 tommy tommy 4096 Jun 21 22:17 hoge
```

▼図7　hoge.txtとhuge.txtの行を抽出したつもりが……

```
% ls -l | grep 'h.ge\.txt'
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:17 hige.txt
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:17 hoge.txt
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:17 huge.txt
```

▼図8　hoge.txtとhuge.txtの行を抽出

```
% ls -l | grep 'h[ou]ge\.txt'
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:17 hoge.txt
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:17 huge.txt
```

も一致してしまいます。|で、正規表現を2つ以上並べていずれかに一致するという指定を行うことができます(**図9**)。この機能は標準の正規表現(基本正規表現)ではなく、拡張正規表現(コラム「正規表現の種類」参照)の機能です。拡張正規表現であることを示すためにgrepに-Eオプションを与えています。また、括弧でくくって一部だけを指定することもできます(**図10**)。

▼図9　hoge.txtとfuga.txtの行を抽出

```
% ls -l | grep -E 'hoge\.txt|fuga\.txt'
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:17 fuga.txt
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:17 hoge.txt
```

▼図10　hoge.txtとfuga.txtの行を抽出(括弧でくくって部分指定)

```
% ls -l | grep -E '(hoge|fuga)\.txt'
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:17 fuga.txt
-rw-rw-r-- 1 tommy tommy    0 Jun 21 22:17 hoge.txt
```

エスケープシーケンス

PCRE(Perl Compatible Regular Expression)では\に続けて英字を記述すると特別な意味になります(**表2**)。実装によってはこれら以外の表記も有効な場合があります。

文字クラス

[ ]はくくられた文字列のどれか1文字を表します。たとえば、**[0123456789ABCDEFabcdef]**は16進数の1文字を表します。文字コード順で連続している範囲の文字は、範囲の先頭と末尾の文字を-で連結することで表せます。先ほどの16進数の1文字は**[0-9A-Fa-f]**と書けます。

-そのものを**[ ]**中に含めたい場合は先頭か末尾に記述します。**[a-z-]**は英小文字と-を表します。PCREでは\-と書くことで-そのものを表せます。**[a\-z]**と記述するとa、-、

Column

正規表現の種類

■基本正規表現

正規表現はPOSIXで規格化されていますが、実際には処理系ごとにさまざまな拡張がされています。基本正規表現はgrep、sedなどのコマンドが標準で使用できる正規表現です。

■拡張正規表現

基本正規表現を拡張したものです。grepでは「grep -E」として実行すると拡張正規表現の記法が使用できます。sedコマンドでは「sed -r」として使用できます。

実は基本正規表現でも拡張正規表現でもそこまで機能に違いはありませんが、表記上\の有無が異なります。基本正規表現で( )や{ }の機能を使うには\(\)、\{ \}と記述します。

拡張正規表現にあり、基本正規表現にない機能は+、?、|です。ですが、Linuxの正規表現ライブラリはこれらの前に\を置いて、\+、\?、\|と書くことで、基本正規表現でも使用できます。

■Perl互換正規表現(PCRE)

Perlの正規表現はPOSIXの拡張正規表現よりもさらに高度な機能を持っています。Perlの正規表現と互換の正規表現をライブラリ化したものがPCREと呼ばれています。PCREを使用することでPerl以外のプログラムでも、強力な正規表現を使用できるようになります。多くのプログラムがPCREまたはPCREを基にした正規表現ライブラリを使用しています。なお、grepコマンドはgrep -Pとして実行するとPCRE正規表現の記法が使用できます。

基本正規表現や拡張正規表現では、記号を指定したときにそのままでメタ文字になるのか、\を前に置いたときにメタ文字になるのかが、記号ごとに異なっていてややこしいです。しかし、PCREでは\を前に置いたときには記号そのままの文字として扱うという規則があるので明確です(すべての記号がメタ文字というわけではないので、\を前に置かなくてもそのままの文字として扱われる記号もあります)。

zのいずれかの文字という意味になります。

角括弧の中の最初の文字が^で始まる場合は否定の意味になります。[^a-z]は英小文字以外の文字を表します。

また、[ ]の中にはエスケープシーケンスを記述することもできます。たとえば[\d]は[0-9]と同じ意味になります。

[ ]中に含める文字の種類をシンボルでも指定できます。たとえば[[:alnum:]]は[A-Za-z0-9]と同じです(表3)。[[:alnum:]]は[[:alpha:][:digit:]]と書いても同じ意味です。

これらの文字の適合条件はC言語のisXXXXX()関数と同じです。たとえば[:print:]はisprint()関数が真になる文字に適合します。

繰り返し

hoge、hooge、hoooogeにマッチする正規表現を考えてみます。h(o|oo|ooo)geでもいいのですが、同じ文字の繰り返しを指定する表記があります。この場合はメタ文字を使ってho{1,3}geと記述できます。メタ文字{n,m}は直前の文字をn回からm回までの範囲で繰り返すことを表します。{n,}と指定するとn回以上で上限なし、{,m}は0回～m回、{n}はちょうどn回の指定になります。基本正規表現では{、}の代わりに\{、\}と記述します。

繰り返す対象はメタ文字でもかまいません。.{2,5}は、2～5文字の任意の文字列に適合します。[a-z]{1,3}は1～3文字の英小文字に適合します。

よく使われるものには、より簡単な記述が用意されています。*は0回以上、?は0～1回、+は1回以上を表します。それぞれ、{0,}、{0,1}、{1,}と同じです。繰り返し指定がないものは{1}が省略されているという考え方もできます。

PCREではさらに、これらの表記の後ろに?を追加することで、最短一致として働くようになります。たとえば:1:2:3:という文字列に対し、正規表現:.*:は:1:2:3:全体に適合しますが、:.*?:では:1:だけに適合するようになります。

位置指定

文字ではなく位置を指定するための表記もあります。すでに説明しましたが、^、$はそれぞれ行頭、行末という位置を指定しており、文字に一致しているわけではありません。ほかにも\A、\zで文字列の先頭と末尾、\bで単語の境界を表します。

PCREでは、より複雑な位置指定もできます。(?=)は括弧の中のパターンに一致する位置を意味します。たとえば(?=abc)aは文字列abcの先頭のa1文字に適合します。(?!)を使うと否定になり、括弧内のパターンに一致しない位置を意味します。(?!abc)aはabc以外の

▼表2　PCREのエスケープシーケンス

エスケープシーケンス	値
\f	ASCII FF(0x0C)
\n	ASCII LF(0x0A)
\r	ASCII CR(0x0D)
\t	ASCII HT(0x09)
\v	ASCII VT(0x0B)
\s	空白文字。ASCII SPACE(0x20)と\f \n \r \t \v
\S	\s以外の文字
\d	数字
\D	数字以外
\w	英数字と_
\W	\w以外の文字

▼表3　おもなシンボル

文字クラス	意味
[:alnum:]	英数字
[:alpha:]	英字
[:blank:]	空白とタブ(\tと0x20)
[:cntrl:]	制御文字(0x00～0x1F、0x7F)
[:digit:]	数字
[:graph:]	印字可能文字
[:lower:]	英小文字
[:print:]	印字可能文字(空白文字0x20を含む)
[:punct:]	英数字以外の印字可能文字
[:space:]	空白文字(\f \n \r \t \vと0x20)
[:upper:]	英大文字
[:xdigit:]	16進数字

文字列のaに適合します。

グループ化

正規表現を括弧でくくってグループ化できます。単に(hoge)だけですとhogeと同じ意味ですが、後ろに繰り返しを指定すると異なってきます。hoge+はhoge、hogee、hogeeeなどに適合しますが、(hoge)+はhoge、hogehoge、hogehogehogeなどに適合します。これは括弧でくくった表現を1つの固まりとして扱っているためです。ここでの括弧は入れ子にもできます。(hoge(fuga)?)*はhoge、hogefuga、hogefugahogehogeなどに適合します。

また、括弧でくくった正規表現に適合した文字列を、正規表現中で再利用できます。たとえば、123-456-789や123_456_789のように3組の数字が-または_で区切られている文字列を考えてみます。これに適合する正規表現は[0-9]+[-_][0-9]+[-_][0-9]+のようになるでしょう。しかし、123-456_789のように-と_が混在していてはいけないとしたらどうでしょうか。[0-9]+[-_][0-9]+[-_][0-9]+は123-456_789にも適合してしまいます。最初の区切り文字が-だとしたら次の区切り文字も-でないといけません。この場合は[0-9]+([-_])[0-9]+\1[0-9]+のようにするとうまくいきます。\1は正規表現中に最初に表れる括弧に適合した文字を表します。同様に2番めの括弧は\2、3番めの括弧は\3です。このような、適合した文字列を後ろで使用するような括弧を「キャプチャ」といいます。

グループ化とキャプチャは同じ表記ですが、括弧がたくさん表れるような複雑な正規表現では、キャプチャせずにグループ化だけしたいこともあります。その場合は(?:)のように括弧の中を?:で始めます。これにより、この括弧はキャプチャとしては働きません[注3]。

注3) これはPCREの機能です。

```
% echo abcabc | grep -P '(...)\1'
abcabc
% echo abcabc | grep -P '(?:...)\1'
grep: reference to non-existent subpattern
```

大文字小文字

正規表現は通常、大文字と小文字を区別しますが、それらを区別しないようにする機能もあります。ただし、正規表現中では通常指定できません。外部からフラグを与える必要があります。grepコマンドでは「grep -i」と指定します。sedコマンドでは「s/re/str/i」のように「i」を付けます。PCREでは正規表現中に(?i)と書くことにより、それ以降のパターンで大文字小文字を区別しなくなります。たとえば、abc(?i)abcはabcABCに適合しますが、ABCABCには適合しません。

文字コード

同じ文字でもエンコーディングが異なればバイト表現が異なります。

エンコーディング	「あ」のバイト表現
UTF-8	E3 81 82
EUC-JP	A4 A2
CP932	82 A0

逆に同じバイト列でもエンコーディングによって別の文字になります。

エンコーディング	C2 A9が表す文字
UTF-8	©
EUC-JP	息
CP932	ｼｩ

処理しようとする文字列のエンコーディングが正しくなければ文字を正しく判別できず、正規表現で正しく処理できません。grepやsedコマンドはコマンド実行時のロケールに依存して、処理対象のデータのエンコーディングを決定します。ロケールは、LC_ALL環境変

Column

文字コード

正規表現の扱うデータの最小単位は「文字」です。1文字が複数バイトで構成されているマルチバイト文字でも文字単位で処理されます。正規表現 . は任意の1文字を表します。半角英字「A」でもひらがなの「あ」でも同様です。コンピュータ上で文字を扱えるようにするため、文字には1対1に対応した番号が割り当てられています。番号を付けた文字の集まりを文字集合と言います。文字集合にはASCII、JIS X 0208、Unicodeなどがあります。文字集合ごとに番号の体系は異なりますし、同じ文字に付けられている番号も異なります。たとえば「あ」はJISコードで4区2点、Unicodeでは3042といった具合です。

さらにコンピュータで処理しやすいように、これらの番号をある方式でバイト列に変換します。この方式をエンコーディング方式（文字符号化方式）と呼びます。「文字コード」はあいまいな用語で、文脈によってエンコーディング方式のことだったり、特定の文字をあるエンコーディングで表現したバイト列のことだったりします。「このテキストファイルの文字コードはUTF-8です」「『あ』の文字コードはE3 81 82です」などのように。日本語は歴史的な経緯により、複数のエンコーディング方式が使用されています。

ISO-2022-JP

すべての文字が7bit（0x0～00x7Fの範囲内）に収まるようなエンコーディング方式です。初期のインターネットメールは7bitしか使用できなかったため、これが使用されていました。現在でも日本語のメールにはISO-2022-JPが多く使用されています[注A]。ISO-2022-JPは複数の文字集合が含まれ（ASCII、JIS X 0201（英数記号）、JIS X 0208）、エスケープシーケンスによって文字集合を切り替える方式になっています。そのため文字単位としては扱いにくく、文字単位の処理をする場合はほかのエンコーディング方式の文字列に変換してから処理するのが一般的です。ISO-2022-JPは「JISコード」と言われることもあります。

EUC-JP

UNIX系OSでよく使用されていました。文字集合はASCII、JIS X 0201（半角カナ）、JIS X 0208、JIS X 0212です。ISO-2022-JPとは異なり、1文字を1～3バイトで表現できるマルチバイト文字です。最近はUnicodeが使用されるようになっているため、あまり使われていないと思います。

CP932

Windowsでよく使用されています。文字集合は「マイクロソフト標準キャラクタセット」で、JIS X 0201、JIS X 0208と、それに加え「①、㈱、髙、﨑」などのJIS X 0208に含まれない文字が含まれています。CP932とは別にSHIFT_JISという似た規格がありますが、SHIFT_JISの文字集合はJIS X 0201、JIS X 0208だけですので、厳密には異なります。一般にシフトJISと言われているエンコーディングはCP932のことです。

UTF-8

文字集合Unicodeのエンコーディングで、1文字1～4バイトです。ISO-2022-JP、EUC-JP、CP932のエンコーディングはJISの文字集合を基本としていましたが、UTF-8の文字集合はそれらとは異なりUnicodeです。Unicodeは世界で使われているすべての文字を含めることを目的として作られ、日本語の文字はISO-2022-JP、EUC-JP、CP932の文字すべてが含まれています。UTF-8の1バイト文字はASCIIと互換があり、使い勝手が良いので広く使われています。「♥」「㊗」「ｶ」など、CP932にも格納されていない記号や絵文字が多く含まれているため、最近はメールでもUTF-8が使用されることがけっこうあります。

注A）ISO-2022-JPで表現できる文字集合には半角カナが含まれていませんでした。そのため、以前は「インターネットで半角カナは使用してはいけない」と言われたものでした。また「①」「㈱」などの字も含んでいませんでしたが、Windowsは半角カナやこれらの文字も無理やり含めてISO-2022-JPとしていたため、Windowsから送られたメールをWindows以外の環境で見ると文字化けしたものでした。

数の値、LC_CTYPE環境変数の値、LANG環境変数の値の順で、最初に見つかったものが使用されます。

図11では、文字を*に置換しています。UTF-8ロケールでは「E3 81 82」のバイト列が「あ」1文字とみなされ、*1文字に置換されますが、Cロケールでは各バイトが1文字とみなされ*3文字に置換されています。

また、**[:alnum:]**などの文字クラスもロケールの影響を受けます。**図12**では、UTF-8ロケールでは全角のＡＢＣが**[:alnum:]**にマッチしますが、Cロケールではマッチしないことを示しています。

Rubyの正規表現

Rubyの正規表現エンジンは「鬼雲(Onigmo)」[注4]です。PCREと同等の機能が使用できます。Rubyプログラム中で正規表現を使用する場合、通常は**/abc/**のように**/**でくくってリテラルで表記します。

```
if str =~ /abc/
  # 文字列strが正規表現abcに適合した場合の処理
end
```

これでは正規表現中に**/**を含められないので、**/**は**\/**と表記します。ただしこれはRubyのリテラル表記の制限ですので、正規表現として**/**に特別な意味があるわけではありません。Rubyの正規表現リテラルにはほかにも**%r{abc}**という表記があります。**/**を多用するような正規表現の場合は、この表記を使用するのが良いでしょう。**%r{ }**でなくても**%r[ ]**、**%r( )**、**%r| |**などを使うことができます。

リテラル表記の直後にオプションを付けることもできます(**表4**)。

また、リテラル表記以外にも、**Regexp.new("...")**としてプログラム実行中に動的に正規表現を作成することもできます。次のように、作成済みの正規表現オブジェクトを埋め込んで、新たな正規表現を作成することもできます(**(?-mix:bar)**はm、i、xオプションが指定されていないことを意味します)。

```
re1 = /bar/
re2 = /foo#{re1}baz/
  #=> /foo(?-mix:bar)baz/
```

Rubyは文字列オブジェクトや正規表現オブジェクトごとにエンコーディングを持ちます。非ASCII文字が含まれていて、異なるエンコーディングの文字列と正規表現をマッチングさ

▼表4　リテラル表記のオプション

オプション	意味
i	大文字小文字を区別しない
m	.が改行に適合する
x	正規表現中の空白とコメントを無視する
u	UTF-8エンコーディング
e	EUC-JPエンコーディング
s	CP932エンコーディング

注4) URL https://github.com/k-takata/Onigmo

▼図11　エンコーディングを指定して文字置換

```
% printf "\xe3\x81\x82\n" | LC_ALL=ja_JP.UTF-8 sed -e 's/./*/g'
*
% printf "\xe3\x81\x82\n" | LC_ALL=C sed -e 's/./*/g'
***
```

▼図12　ロケールによる文字クラスの差異

```
% echo ABCＡＢＣ | LC_ALL=ja_JP.UTF-8 sed -e 's/[[:alnum:]]/*/g'
******
% echo ABCＡＢＣ | LC_ALL=C sed -e 's/[[:alnum:]]/*/g'
***ＡＢＣ
```

せようとすると次のようにエラーになるので注意しましょう。

```
re = /ほげ/                    # UTF-8
str = "ほげ".encode("cp932")  # CP932

str =~ re
# Encoding::CompatibilityError:
incompatible encoding regexp match ➢
(UTF-8 regexp with Windows-31J string)
```

演習：メールアドレスに適合する正規表現を作る

メールアドレスのパターンは複雑で、正規表現では表せないと言われることがときどきありますが、そんなことはありません。パターンが入れ子にでもなっていない限り、たいていの文字列パターンは正規表現で表すことができます[注5]。

メールアドレスの規則をRFC 5321から抜き出してみます(**リスト1**、一部RFC 5322、RFC 5234からも抜粋)[注6]。この表記はABNF(Augmented Backus-Naur Form)と言うもので、RFC 5234で定義されています。今回はABNFについての説明はしませんが、知らなくてもなんとなく読めるのではないかと思います。日本語で簡単に説明すると**図13**のようになります。

基本編

これら規則を満たす、メールアドレス全体の正規表現はどのようになるでしょうか。複雑ですのでローカルパートとドメインに分けて考えましょう。ヒントを元に、解き進めてみてください。

▼リスト1　メールアドレスの規則

```
Mailbox        = Local-part "@" ( Domain / ➢
address-literal )
Local-part     = Dot-string / Quoted-string
Dot-string     = Atom *("."  Atom)
Atom           = 1*atext
atext          = ALPHA / DIGIT /
                 "!" / "#" /
                 "$" / "%" /
                 "&" / "'" /
                 "*" / "+" /
                 "-" / "/" /
                 "=" / "?" /
                 "^" / "_" /
                 "`" / "{" /
                 "|" / "}" /
                 "~"
Quoted-string  = DQUOTE *QcontentSMTP DQUOTE
QcontentSMTP   = qtextSMTP / quoted-pairSMTP
qtextSMTP      = %d32-33 / %d35-91 / %d93-126
quoted-pairSMTP = %d92 %d32-126
Domain         = sub-domain *("." sub-domain)
sub-domain     = Let-dig [Ldh-str]
Let-dig        = ALPHA / DIGIT
Ldh-str        = *( ALPHA / DIGIT / "-" ) Let-dig
ALPHA          =  %x41-5A / %x61-7A
DIGIT          =  %x30-39
DQUOTE         =  %x22
```

▼図13　メールアドレスの規則(日本語)

❶メールアドレスは@でローカルパートとドメインに分割される
❷ローカルパートは次のいずれか
　❷－(1)英数字と記号(**! # $ % & ' * + - / = ? ^ _ ` { | } ~**)で構成される1文字以上の文字列を**.**で連結したもの
　❷－(2)**"**と\を除く0x20-0x7Eの範囲の文字、または「\と0x20-0x7Eを組み合わせたもの」の0回以上の連続を**"**でくくったもの
❸ドメインはサブドメインを**.**で連結したもの
❹サブドメインは英数字と**-**(ただし**-**は先頭と末尾には置けない)
❺ABNFには含まれていないが、次の長さ制限もある
　❺－(1)ローカルパートの最大長は64文字(RFC 5321 4.5.3.1.1)
　❺－(2)ドメインの最大長は255文字(RFC 5321 4.5.3.1.2)
　❺－(3)メールアドレス全体の最大長は256文字(RFC 5321 4.5.3.1.3)
　❺－(4)サブドメインの最大長は63文字(RFC 1035 2.3.4)

注5) メールヘッダ上のFromやToの値はメールアドレス以外にも表示名やコメントが含まれる可能性があり、さらに複雑です。コメントは入れ子にできるので正規表現で表すのは難しいですが、PCREでは**?R**で再帰できるので、実現は可能かもしれません。

注6) **Mailbox**の**address-literal**はIPアドレス表記です。通常のメールアドレスではないので今回は無視します。

ローカルパート

問題1 ローカルパートの規則❷ー(1)

英数字と記号(`! # $ % &' * + - / = ? ^ _ ` { | } ~`)で構成される1文字以上の文字列を`.`で連結したものを正規表現で表してみましょう。

ヒント 単純に、この規則に現れる文字種を並べると次のようになります。

```
[0-9a-zA-Z!#$%&'*+\-/=?^_`{|}~.]+
```

これは間違いです。この正規表現は`abc..xyz`や`.abc`にも適合しますが、`.`は連結文字ですので連続していたり先頭や末尾に現れたりしてはいけません。

あるパターン`X`が文字`d`で結合されているということは、`X`、`XdX`、`XdXdX`、`XdXdXdX`……という並びになるということです。つまり、`X`の後ろに`dX`が0回以上現れるということです。`dX`をひとまとまりとして扱うにはグループ化して`(dX)`とします。`(dX)`が0回以上現れるということは後ろに`*`を付けて`(dX)*`とすれば良いです。よって、`X`が`d`で結合されるというパターンは`X(dX)*`と表せます。

→解答は**リスト2**

問題2 ローカルパートの規則❷ー(2)

`"`と`\`を除く0x20〜0x7Eの範囲の文字、または「`\`と0x20〜0x7Eを組み合わせたもの」の0回以上の連続を`"`でくくったものを正規表現で表してみましょう。

ヒント `"`と`\`を除く0x20〜0x7Eは`[\x20\x21\x23-\x5b\x5d-\x7e]`として表せます。`\`と0x20〜0x7Eの組み合わせは`\\[\x20-\x7e]`です。AまたはBというパターンは`(A|B)`で表せ、これの0回以上の繰り返しは`(A|B)*`となります。

→解答は**リスト3**

問題3 ローカルパート全体を**問題1**と**問題2**の答えを組み合わせて作ってみましょう。

→解答は**リスト4**

ドメイン

ドメイン(❸)はサブドメインを`.`で連結したものですので、まずサブドメインを解決します。

問題4 サブドメイン❹

サブドメインは英数字と`-`(ただし`-`は先頭と末尾には置けない)に適合する正規表現を表してみましょう。

ヒント サブドメインは英数字と`-`ですが、先頭と末尾には`-`を置けません。これは単に、先頭と末尾のパターンと途中のパターンが異なるというだけです。先頭と末尾は英数字ですので`[0-9a-zA-Z]`、途中は英数字と`-`ですので`[0-9a-zA-Z-]`と表記できます。これを単純に組み合わせると次のようになります。

```
[0-9a-zA-Z][0-9a-zA-Z-]*[0-9a-zA-Z]
```

しかしこれは間違いです。この正規表現は1文字の文字列にはマッチしません。途中のパターンは`*`で0文字以上となっていますが、先頭と末尾は1個ずつないといけないことになっているため、最低2文字ないとマッチしないことになります。そこで次のようにしてみます。

```
[0-9a-zA-Z][0-9a-zA-Z-]*[0-9a-zA-Z]?
```

末尾のパターンに`?`を付けることにより、0個か1個と指定しています。これで1文字の文字列にもマッチするようになりました。

しかし、これも間違いです。これだと先頭文字＋途中文字にマッチするため、`abc-`のように`-`で終わる場合にもマッチしてしまいます。

先頭文字を`S`、途中文字を`M`、末尾文字を`E`とすると、`S`、`SE`、`SME`にはマッチし、`SM`にはマッチしないようなパターンを作れば良いです。`M`があれば必ず`E`もあるようにするには、2つをグループ化して`S(M*E)?`のようにすれば良いでしょう。

→解答は**リスト5**

さらにサブドメインには最大63文字という長さの制限があります。先頭文字と末尾文字を含めて63文字ということは途中文字を最大61文字に制限すれば良いということですので、**リスト5**の中で、`*`の代わりに`{,61}`を指定します(**リスト6**)。

問題5 ドメイン

ドメインを正規表現で表してみましょう。

ヒント ドメインはサブドメインを.で連結したものですので、ローカルパート❷ー(2)と同じように、X(dX)*形式で表せます。

→解答は**リスト7**

問題6 メールアドレス全体を正規表現で表してみましょう。
→解答は**リスト8**

ローカルパートとドメインの正規表現ができたので@で連結すればメールアドレス全体の正規表現が完成です。文字列の先頭と末尾にも一致するように\Aと\zも追加しましょう。

発展編

読みやすくする

ちょっと長くて読みにくいのでRubyプログラム中でのリテラル表記として書き直してみます(**リスト9**)。xオプションを指定すると、正規表現の意味を変更せずに空白や改行、コメントを入れることができます。/は正規表現リテラルの終わりの記号とみなされないように\/としてエスケープしています。

また、PCREでは、何度も現れるパターンに(?<name> ...)で名前を付けて、\g<name>で再利用できます(**リスト10**)。かなり見やすくなったと思うのですが、いかがででしょうか。

長さ制限を追加する

今回作成した正規表現にはメールアドレス全体、ローカルパート、ドメインの長さ制限(❺ー(1)～(3))がありません。これらの長さはプログラムで簡単にチェックできるので、通常はあまり正規表現には記述しないと思いますが、今回はこの長さ制限も正規表現に行わせてみましょう。

今回の場合は(?=)を使えば、長さの制限も実現できます。たとえば(?=.{5,10}\z)は任意の5～10文字のあとに文字列末尾があるよ

▼リスト2　問題1の答え　❷ー(1)を正規表現で表す

```
[0-9a-zA-Z!#$%&'*+\-/=?^_`{|}~]+(\.[0-9a-zA-Z!#$%&'*+\-/=?^_`{|}~]+)*
```

▼リスト3　問題2の答え　❷ー(2)を正規表現で表す

```
"([\x20\x21\x23-\x5b\x5d-\x7e]|\\[\x20-\x7e])*"
```

▼リスト4　問題3の答え　❷ー(1)+❷ー(2)を正規表現で表す

```
([0-9a-zA-Z!#$%&'*+\-/=?^_`{|}~]+(\.[0-9a-zA-Z!#$%&'*+\-/=?^_`{|}~]+)*|"([\x20\x21\x23-
\x5b\x5d-\x7e]|\\[\x20-\x7e])*")
```

▼リスト5　問題4の答え　❹を正規表現で表す

```
[0-9a-zA-Z]([0-9a-zA-Z-]*[0-9a-zA-Z])?
```

▼リスト6　リスト5に文字数制限を追加

```
[0-9a-zA-Z]([0-9a-zA-Z-]{,61}[0-9a-zA-Z])?
```

▼リスト7　問題5の答え　❸+❹ドメインを正規表現で表す

```
[0-9a-zA-Z]([0-9a-zA-Z-]{,61}[0-9a-zA-Z])?(\.[0-9a-zA-Z]([0-9a-zA-Z-]{,61}[0-9a-zA-Z])?)*
```

▼リスト8　❷ー(1)+❷ー(2)+❸+❹を正規表現で表す　問題6の答え

```
\A([0-9a-zA-Z!#$%&'*+\-/=?^_`{|}~]+(\.[0-9a-zA-Z!#$%&'*+\-/=?^_`{|}~]+)*|"([\x20\x21\x23-
\x5b\x5d-\x7e]|\\[\x20-\x7e])*")@[0-9a-zA-Z]([0-9a-zA-Z-]{,61}[0-9a-zA-Z])?(\.[0-9a-zA-Z]
([0-9a-zA-Z-]{,61}[0-9a-zA-Z])?)*\z
```

位置に適合します。つまりこの位置以降の文字数が50文字であることを指定できるのです。先頭でこれを使用することで文字列全体の長さの範囲を制限できます(**リスト11**)。

また、同じように先頭で`(?=.{,64}@[^@]{,255}\z)`と指定することで、ローカルパートが64文字以内、ドメインが255文字以内という制限になります。

▼リスト9　メールアドレスの正規表現をRubyのリテラル表記として書き直す

```
/\A
  # local-part
  (  # dot-string
     [0-9a-zA-Z!#$%&'*+\-\/=?^_`{|}~]+(\.[0-9a-zA-Z!#$%&'*+\-\/=?^_`{|}~]+)*
  |  # quoted-string
     "([\x20\x21\x23-\x5b\x5d-\x7e]|\\[\x20-\x7e])*"
  )@
  # domain
  [0-9a-zA-Z]([0-9a-zA-Z-]{,61}[0-9a-zA-Z])?(\.[0-9a-zA-Z]([0-9a-zA-Z-]{,61}[0-9a-zA-Z])?)*
\z/x
```

▼リスト10　リスト9をさらに簡略化

```
/\A
  # local-part
  (  # dot-string
     (?<atom>[0-9a-zA-Z!#$%&'*+\-\/=?^_`{|}~]+)(\.\g<atom>)*
  |  # quoted-string
     "([\x20\x21\x23-\x5b\x5d-\x7e]|\\[\x20-\x7e])*"
  )@
  # domain
  (?<sub_domain>[0-9a-zA-Z]([0-9a-zA-Z-]{,61}[0-9a-zA-Z])?)(\.\g<sub_domain>)*
\z/x
```

▼リスト11　リスト10に文字数制限を追加

```
/\A
  # 全体で256文字以下
  (?=.{,256}\z)
  # local-partは64文字以下でdomainは255文字以下
  (?=.{,64}@[^@]{,255}\z)
  # local-part
  (  # dot-string
     (?<atom>[0-9a-zA-Z!#$%&'*+\-\/=?^_`{|}~]+)(\.\g<atom>)*
  |  # quoted-string
     "([\x20\x21\x23-\x5b\x5d-\x7e]|\\[\x20-\x7e])*"
  )@
  # domain
  (?<sub_domain>[0-9a-zA-Z]([0-9a-zA-Z-]{,61}[0-9a-zA-Z])?)(\.\g<sub_domain>)*
\z/x
```

正規表現を部品化して組み立てる

Rubyでは、作成済みの正規表現を埋め込んで正規表現を作ることができます。これを利用して、ABNF規則から機械的にメールアドレスの正規表現を作成することもできます(**リスト12**)。最終的にできあがる正規表現は人間の目には読みにくいものになりますが、意味のある要素ごとに小さく作って組み合わせることができるので、実際のプログラムではこの方法も有用だと思います。

おわりに

正規表現の基本的な説明から、複雑なものまで紹介してきましたが、これがすべてではありません。正規表現を詳しく説明するとそれだけで書籍1冊になるくらいの高度な機能を持っています。使いこなせると、とても強力な武器になります。正規表現を知らなかった人は、簡単なことからで良いので、ぜひ使ってみてください。SD

▼リスト12　Rubyの機能を使って、正規表現を部品の集まりとして作成

```
atext = /[0-9a-zA-Z!#$%&'*+\-\/=?^_`{|}~]/
atom = /#{atext}+/
dot_string = /#{atom}(\.#{atom})*/
qtextsmtp = /[\x20\x21\x23-\x5b\x5d-\x7e]/
quoted_pairsmtp = /\\[\x20-\x7e]/
qcontentsmtp = /#{qtextsmtp}|#{quoted_pairsmtp}/
quoted_string = /"#{qcontentsmtp}*"/
local_part = /#{dot_string}|#{quoted_string}/
let_dig = /[0-9a-zA-Z]/
ldh_str = /[0-9a-zA-Z-]*#{let_dig}/
sub_domain = /#{let_dig}#{ldh_str}?/
domain = /#{sub_domain}(\.#{sub_domain})*/
mailbox = /\A#{local_part}@#{domain}\z/
```

第02時限

思いどおりにSQLを組めるようになりたい!
スマートにSQLを書くコツ
リレーショナルモデルと正規化の重要性

Author 奥野 幹也(おくの みきや) 日本オラクル㈱ MySQL Global Business Unit

本稿では、みなさんにスマートにSQLを書くコツを伝授したいと思います。コツと言っても、何も秘伝のように、ミステリアスで魔法のような方法ではなく、至極当たり前のことばかりですので、みなさん肩の力を抜いて読んでみてください。

はじめに

「思ったとおりの結果を得られるSQLを書くことができない」あるいは「思ったとおりの結果は得られるけど、SQLがスパゲティになってしまい、あとで思ったとおりの結果が返らなくなった」という経験はないでしょうか。

よく**プログラムは思ったとおりではなく、書かれたとおりに動く**と言いますが、SQLもこの例外から外れることはできません。むしろSQLはたいへん癖のある言語であり、ほかのプログラム言語を書くよりもずっと難易度は高くなります。少ない記述で大量の仕事をさせることもできるため、SQLが書かれたとおりに動いた結果、意味もなくサーバが高負荷になってしまうというようなケースも、少なくありません。

なぜ思いどおりのSQLが書けないのか

思いどおりのSQLを書けない原因はいろいろありますが、その中でもとくに重大なものとして、筆者は次の2つを挙げたいと思います。

・データベース設計とアルゴリズムの間に乖離(かいり)がある
・リレーショナルモデルを知らない

この2点について少し詳しく見て行きましょう。

データ構造とアルゴリズム

プログラマのみなさんであれば、あるアルゴリズムを実行するのには、そのためのデータ構造を準備しなければならないということを、よくご存じではないかと思います。言い換えると、**データ構造とアルゴリズムはセット**だということです。両者を切り離すことはできません。

このことを思い出していただくために、少し例を挙げてみましょう。

整数の演算に適したデータ型は何でしょう。もちろん使用するプログラミング言語しだいではありますが、たとえばC言語であればint型でしょう。桁数が増えればまずはlong long intなど、多くの桁数をカバーできるデータ型の利用を検討するべきでしょう。それでも足りなければ、本当は任意の桁数を扱えるライブラリを使うべきです。もしここで判断を誤って可変長文字列で数値を表現してしまった場合どうなるでしょう。可変長文字列ならば、たしかに任意の桁数の数値を表現できます。しかし肝心の演算はまったくできません。文字列に対して定義された演算は、連結や部分

※本稿において示されている見解は、筆者自身の見解ですので、所属する団体の見解を反映したものではありません。ご了承ください。

文字列の抽出、パターンマッチなどであって、四則演算ではないからです。例え数字しか格納されていなくても、自動的にその数字の意味を理解して演算するということはありません。このように、**演算や操作にはそれに合ったデータ型がある**ということはプログラミングにとって極めて本質的で重要なことです。データに対する演算や操作は、適切な型あるいは構造を持つデータにしか適用できませんし、データ構造に合った演算しか容易されていないのです。

「何を当たり前のことを言ってるんだ!?」と呆れた方もいらっしゃるかもしれません。ここであらためてこのようなことを言う理由は、汎用言語を用いたプログラミングではわかる上記のことが、なぜかデータベースでは見落とされてしまうという風潮がしばしば見られるからです。そのことをよく表すフレーズとして、**「データベースはただの入れ物」**というものがあります。これはデータ構造なんか関係ないと言ってるのも同然です。そのような認識では、データベースへのアクセスが非効率になって当然なのです。

リレーショナルモデルは道具

データモデルを用いてデータを表現する場合、当然ながら、そのデータモデルがどのようなものであるかを知っている必要があります。道具の使い方がわからないのに、それをどうやって使えばよいというのでしょう。筆者は最近、趣味で自転車にハマってるのですが、自転車を整備する道具の1つに振れ取り台というものがあります。振れ取りという作業を知らない人に、この道具は何の用途で使うのかを尋ねても、きっと答えることはできないでしょう。**道具は使い方を理解してこそ使える**のです。データモデルは道具です。その使い方を理解しなければ、正しい使い方はできません。

たとえばC言語でint型の変数を、整数値を表すことに利用できるのは、int型が整数値を表すものであり、それに対してどのような演算が可能なのかということを、プログラマが知っているからです。つまり、データモデルを理解するということは、**そのデータモデルで表されたデータがどのような意味や性質を持っているか**ということや、**どのような演算が定義されているか**ということを知ることにほかなりません。そのうえで、そのデータモデルにおいて**どのような応用ができるか**といったことや、**どのような用途では使えないか**ということを知れば、データモデルを自由自在に使いこなすことができるでしょう。

リレーショナルデータベース上でデータを表現するには、リレーショナルモデルという道具の使い方を理解する必要があります。それを知らずにリレーショナルモデルを批判したり、NoSQLへ移行すれば人件費が下がると言ったりしないでほしいものです。

リレーショナルモデルを知る

リレーショナルモデルとは何かということをきっちりと本稿で説明できればよいのですが、残念ながら筆者に与えられたページ数ではとても足りません。リレーショナルモデルがどのようなものか、基本的な使い方とデータベース設計の基礎、そして応用方法といったことについて知りたい方は、ぜひ拙著[注1]を読んでください。筆者が言いたいことは、あらかたこちらの本の中で述べています。もしくはリレーショナルモデルを主題にした、もっと分厚くてきちっとした内容の本を読むのもよいでしょう。たとえばC.J.Date[注2]やJ.Celko[注3]です。

リレーショナルモデルを理解するには述語論理や集合の知識が必須ですので、それらを

注1）『理論から学ぶデータベース実践入門 ―リレーショナルモデルによる効率的なSQL(Web+DB PRESS plus)』、奥野幹也著、技術評論社、ISBN978-4-7741-7197-5

注2）URL http://www.amazon.co.jp/C.-J.-Date/e/B000AQ6OJA

注3）『プログラマのためのSQL すべてを知り尽くしたいあなたに』、ジョー・セルコ著、翔泳社、ISBN978-4-798128023

よく知らないという人は、そういった分野についても一度勉強しておいたほうがよいでしょう。拙著でもリレーショナルモデルを理解するのに必要最低限の内容をカバーしていますので、そちらを読んでいただいてもかまいません。ただ、若干解説が駆け足過ぎた感があり、難しいという声をよく耳にしますので、読んでみてもよくわからないという人は教科書をめくってみましょう。述語論理や集合は、整数と同じく数学上の概念です。みなさんはきっと整数の扱いには精通されていらっしゃると思います。なぜ同じ数学上の概念であるにもかかわらず、述語論理や集合はあまり理解されていないかというと、単にカリキュラム上の問題ではないかと思います。つまり、整数については小学校のころからずっと勉強しているのでみなさんはとても詳しくなりやすく、述語論理や集合などは高校や大学になってから学ぶため、慣れ親しんでいないだけではないかと思うわけです。これらはリレーショナルモデル以外にも応用が利く概念ですので、数学の基礎知識としてぜひマスタしておきましょう。費やした時間はけっして無駄になりません。

ただ単に、「ほかの本で勉強してください」というだけでは本稿の話が進みませんので、リレーショナルモデルの概要を大まかにかいつまんで紹介したいと思います。

リレーショナルモデルは集合の世界

リレーショナルモデルについて、非常によくある間違いの中に、**リレーションとはテーブル同士の関係性のことである**とか**リレーショナルモデルとはER図を使ってテーブルを設計することである**というようなものがあります。これは完全な誤りです。

リレーショナルモデルで**リレーション**と言えば、データを操作する単位のことで、SQLでいうところの**テーブル**に相当します。このことを聞いて、「えっ?!」となってしまった方はご用心。もしかすると、今までちゃんとしたリレーショナルモデルの解説を読んだことがなかったのかもしれません。これを機に、あらためてリレーショナルモデルを学習してみてはいかがでしょうか。

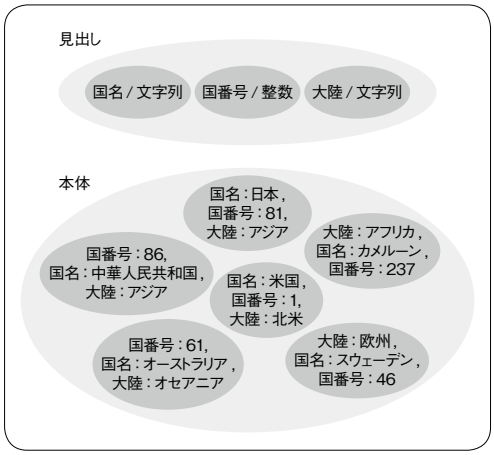
リレーショナルモデルはリレーション[注4]を単位として、さまざまな演算をするデータモデルです。リレーションは集合ですが、任意の集合ではなく、特徴的な構造を持った集合です。リレーションの要素はタプル(組、*tuple*)と呼ばれ、1つのリレーション内のタプルはすべて同じ構造になっています。タプルも集合であり、その要素はアトリビュート値(属性値、*attribute value*)と呼ばれます。アトリビュート値はそれ以上分解することができない値となっています。アトリビュート値にはでたらめな値が表れるわけではなく、あらかじめどのような値になるべきかということがわかっており、そのような値全体は集合として定義されているものと考えます。その集合をドメイン(定義域、*domain*)と呼びます。つまりアトリビュート値は、ドメインの1つの要素であり、ドメインは集合であるためにその要素を分解することはできないということになります。また、タプルの集合はリレーションの本体(*body*)と言われ、リレーションに対する演算はおもに本体が対象になります。そのリレーションのタプルにどのようなアトリビュートが含まれているかということを示すため、見出し(*headding*)が設けられています。見出しも集合です。見出しの要素はアトリビュート(属性、*attribute*)と呼ばれ、名前と型のペアになっています。アトリビュートの具体的な値がアトリビュート値というわけです。

リレーションの例とドメインの例を、それぞれ**図1**、**図2**に示します。

図1のリレーションには、いくつかの国が格納されています。本体のすべての要素、つまりタプルはすべて国名、国番号、大陸という

注4) リレーションは関係とも呼ばれますが、日本語と混同しやすいため、本稿ではカタカナを用いています。

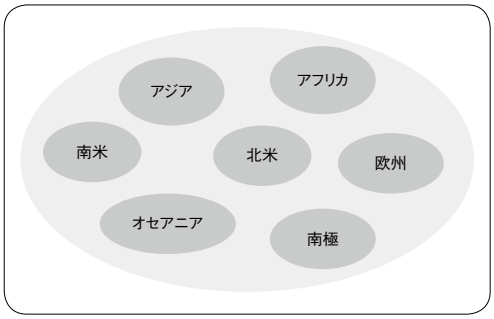
▼図1　リレーションの例



名前のアトリビュートを持っていることが、見出しからわかります。また、このリレーションのアトリビュートの1つである大陸のドメインを表したのが**図2**です。リレーション（本体、見出し）もタプルもドメインも集合ですので、要素間に順序はありません。

リレーショナルモデルがとことんまで集合をベースとしたデータモデルになっているということがおわかりでしょうか。集合という概念を理解しないと、リレーショナルモデルは使いこなせないのです。

▼図2　ドメインの例



リレーションの演算

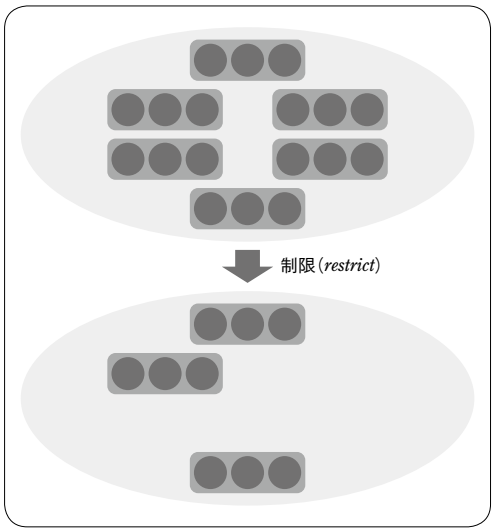
リレーションの構造について見たところで、次は演算について紹介しましょう。リレーションに対する演算は、集合の操作に基づいたものとなります。そのほかに、リレーショナルモデル特有の操作、つまりリレーションがすべて同じ構造を持ったタプルを要素としているという性質を利用した操作が、いくつか定義されています。**図3**～**6**は、代表的なリレーションの演算の一例です[注5]。

それぞれの演算の意味は次のとおりです。

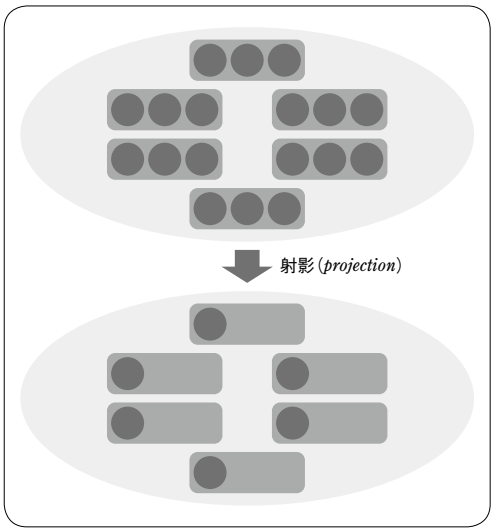
・制限（*restrict*）

注5） ほかにも演算はたくさんあります。

▼図3　制限



▼図4　射影



1つのリレーションを特定のタプルだけに絞り込む操作

・射影（*projection*）
1つのリレーションを特定のアトリビュートだけに絞り込む操作

・和（*union*）
2つのリレーションの両方に含まれるタプルを求める操作。ただし重複は排除される

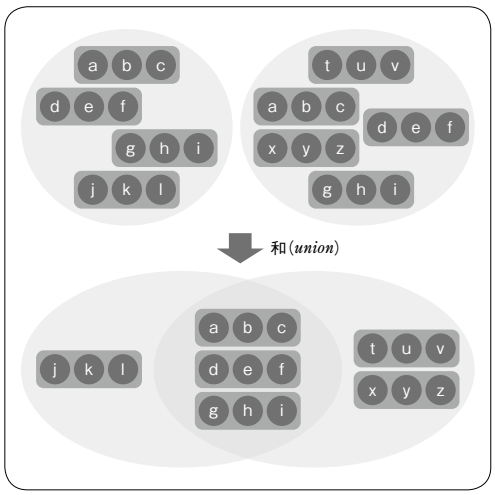
・結合（*join*）
同じアトリビュートを含んだ2つのリレーションのタプルを総当たりで組み合わせて得られるタプルのうち、同じアトリビュートに対して共通の値を持つタプルを求める操作

リレーションの演算の性質の大きな特徴は、**演算の入力も結果もリレーションになる**ということです。つまり、ある演算をした結果得られるリレーションを、別の演算の入力にできるのです[注6]。リレーションの演算をいくつも組み合わせて、複雑な演算を表現できるところが、リレーショナルモデルの真骨頂なのです。

こういった演算について知らない、よくわからないということでしたら、まずはリレーショナルモデルをしっかりと学んでください。それがSQLを上手に使いこなすための近道です。筆者は、**リレーショナルモデルなんてわからなくてもSQLを使いこなせる、などという無責任なことは絶対に言えません**から。

注6）このような性質を閉包性と言います。

▼図5　和

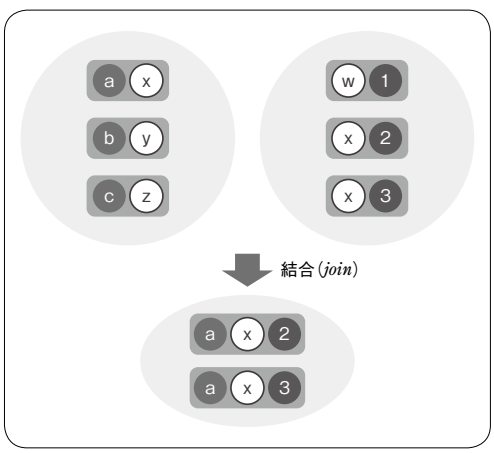


SQLとリレーショナルモデル

賢明な読者の方であればすでにお気づきかもしれません。リレーショナルモデルを知れば知るほど、**SQLとリレーショナルモデルは別物ではないか??**という疑念が沸き起こってくることでしょう。それもそのはず、両者の間には大きな乖離（かいり）があるのです。

SQLはたしかにリレーショナルモデルをもとにして考案された言語です。しかし、それ以上のこともできてしまいます。なぜならば、リレーショナルモデルは集合をベースとして構築されているため、集合で表現できないデータはリレーショナルモデルでは扱うことができないからです。そのため、SQL上のオブジェクトと、リレーショナルモデル上のオブジェクトの間にも違いがあります。SQLではテーブル、行、列というオブジェクトが、リレーショナルモデルではリレーション、タプル、アトリビュートというオブジェクトに相当しますが、名前が違うことからも察するように、それぞれ性質は異なります。テーブルや行は集合で

▼図6　結合



はないのです。そのことをきちんと理解してSQLを記述しないと、リレーショナルモデルに沿った使い方はできないのです。

このSQLとリレーショナルモデルの違いに気づかずにSQLを書いてしまうと、リレーショナルモデルをまったく無視した使い方になってしまいます。もっと言うと、リレーショナルモデルをまったく知らなくても、SQLを使えてしまうのです。もちろん、リレーショナルモデルを知らなければ、適切なデータベース設計ができるわけもなく、その結果クエリもスパゲティになり、正確な結果が返らず、性能も出ず、いつになっても製品やサービスをリリースできないないというような状況になりかねません。これはSQLに潜む大きな罠です。恐ろしいですね。

集合と述語論理

リレーショナルモデルを理解するうえで、もうひとつ忘れてはいけないのが述語論理です。本稿では、リレーショナルモデルは集合に基づいたものであると説明してきましたが、実はリレーショナルモデルは述語論理(*predicate logic*)に根ざしたデータモデルでもあります。これはなぜかというと、**集合と述語論理が1対1で対応する**からです。

本稿では、リレーションは特殊な構造を持った集合であるということをすでに述べましたが、集合と述語論理が一対一で対応するのであれば、リレーションは述語論理でどのように表現できるのでしょうか。詳細な説明を省いて単刀直入に言うと、**リレーションとは真となる命題の集合**なのです。リレーションの要素、つまり**タプルはすべて命題であり、それを評価した結果は真になる**ということです。リレーションの演算は、1つあるいは複数の真の集合から、別の真の集合を導く論理演算であると言えます。話が長くなるので詳細な解説は省きますが[注7]、リレーションの演算が論理演算であるという点は極めて重要です。なぜならば、リレーションの演算、すなわち**クエリ(問い合わせ)が、論理演算にほかならない**ということを意味するからです。

さて、真の命題とはいったいいかなるものでしょうか。リレーショナルモデルでは、**真の命題とは事実である**と考えます。つまり、ある1つあるいは複数の事実から、論理的な演算、つまり推論によって、別の事実を導き出すというのが、リレーションの演算の実体なのです。このため、リレーショナルモデルが適合できる範囲のクエリは、**漏れや不確実性とは無縁であり、すでに判明している事実から論理演算によって、論理的に正しい結果が得られる**ことになります。リレーショナルモデルがいかに強力なデータモデルであるかがおわかりでしょうか?

論理の天敵:矛盾

論理的に正しい答えが得られるということは、極めて強力な特徴です。しかし強力であるがゆえに、反対に脆(もろ)い面も持ち合わせています。それは、**論理演算であるがゆえに、矛盾に弱い**というものです。

矛盾とは、正しいとされる命題に食い違いが生じている状態です[注8]。2つの命題が真であると仮定すると、どちらが正しいのか判断がつかないようなもののことを指します。たとえば、「太郎は15歳です」という命題と「太郎は18歳です」という命題がともに真だと仮定しましょう。もし両方が真なのであれば、いったい太郎さんの年齢はいくつなのでしょうか?この問いに対しては、いくら考えても答えはでません。なぜならば、2つの命題が矛盾しているからです。

論理演算の結果、正しい答えが得られるのは、その前提となる命題が本当に正しい場合、つ

注7) 興味のある方は、拙著(注1)をご参照ください。

注8) リレーションに生じた矛盾を異常(*anomaly*)と呼びます。

まり矛盾を含んでいない場合のみです。矛盾が含まれていると、途端にデータモデルの有効性は崩壊してしまいます。このような矛盾を生じないようにするにはどうすればよいのでしょうか。この問に対する一般的な答えが、**データの重複をなくす**ということです。つまり、同じ命題(＝タプル)が繰り返し出現するのを避けるということです。データに矛盾が生じるのは、重複したデータが含まれているからであり、間違って片方だけを更新してしまうと矛盾になってしまうというわけです。

リレーションは集合ですので、その要素そのものに重複が起きることはありません。ではどこに重複が起きるかというと、タプル全体ではなく、タプルの一部ということになります。

正規化の重要性

命題＝タプルですが、命題と言えば先ほどの「太郎は10歳です」という例のように文章になっており、これをそのままデータとして用いるのは都合がよくありません。データをリレーションで表現する場合には、つまり命題をタプルとして表現するには、変動するパラメータの部分だけを抜き出すことになります。また、先ほどの命題のように、単に人物と年齢が格納されただけのリレーションでは、通常矛盾は生じません。なぜならば、リレーショナルモデルにはキーという概念があるからです。**キーはタプルの値を一意に特定するためのもの**であり、1つのリレーション内において、同じ値のキーが2度出現することはありません。たとえば名前、年齢という2つのアトリビュートが設定されたリレーションでは、名前がキーに設定されていると、同じ名前について異なる複数のタプルが格納されることはないからです。

リレーションの内部に、矛盾の温床となる重複が生じるのは、リレーションがもう少し複雑な場合です。**表1**は矛盾が生じているリレーションの例です。このリレーションは、ある学校の生徒の名前と部活、学年がアトリビュートとして含まれています。

同じ生徒が複数回出現しているのは、おそらく部活を掛け持ちしているためでしょう。山田太郎さんは野球部とパソコン部に所属していることがわかりますが、学年が一方では2、もう一方では3と、同じではありません。いったいどちらが正解なのかということは、先ほども述べたとおり、このリレーションをいくら眺めたところで判断できません。なぜなら、リレーションはどちらのタプル＝命題も、真であるということを示すからです。**このような矛盾が生じる余地がないように、リレーションを分解する作業が正規化です。**正規化にはいくつかの段階があり、段階が上がるほど重複のないよい状態になるとされています。

実は、正規化をしてもしなくても、リレーションにはリレーショナルモデルの演算を適用できるのですが、正しい結果が得られるかどうか、そして演算を適用しやすいかどうかということが違ってくるのです。**すっきりとしたSQLを書くには、その前段階として、操作する対象のテーブルが、しっかりと設計されている必要があります。**とくに矛盾は大敵ですから、矛盾が生じないようにするためには、重複を解消する正規化という作業は欠かせないのです。

高速で正確なクエリを書くためのテクニック

本稿ではここまで、アルゴリズムとデータ構造はセットであること、そのためプログラ

▼表1　矛盾が生じている例

名前	部活	学年
山田太郎	野球	2
山田太郎	パソコン	3
鈴木次郎	水泳	2
佐藤三郎	水泳	1
佐藤三郎	吹奏楽	1
田中士郎	野球	2

マは適切なデータモデルを選択しなければならないこと、リレーショナルデータベース上で扱うデータモデルであるリレーショナルモデルについてを、かなり大雑把に説明してきました。これらをふまえたうえで、実践的にSQLを上手に書くためにはどうすればよいかということを、解説したいと思います。

SQLを宣言的に記述する

リレーショナルモデルは深いテーマですので、雑誌の1コーナーではとても解説することはできません。ですので、リレーショナルモデルをよく知らないという方には、雰囲気だけでもつかんでいただければ幸いなのですが、本稿の内容程度ではやはり不十分です。一度きっちりとリレーショナルモデルについて学ぶべきでしょう。繰り返しになりますが、**リレーショナルモデルがわからなくてもSQLぐらい使いこなせるなどという無責任なことは絶対に言えません。**

リレーショナルモデルに沿ってクエリを記述した場合には、**クエリが宣言的になる**という特徴があります。すでに判明している事実(リレーション)に対して、論理演算を適用し、漏れなくかつ正確な答えが得られるからです。**ループや条件分岐を駆使する必要はありません。**1回のSELECTで、必要な答えはすべて得られるのです。

リレーショナルモデルの限界

リレーショナルモデルをきちんと理解したならば、リレーショナルモデルができることとできないこと、向き不向きなどがわかるようになります。リレーションは集合の一種ですので、集合で表現できないデータ、あるいは集合演算ではうまく解決できないアルゴリズムには、適用するべきではないのです。**SQLを宣言的に記述できるのは、リレーショナルモデルが適用できる場合だけ**であることは、忘れてはなりません。

リレーショナルモデルの範疇(はんちゅう)でないデータは世の中に溢れています。たとえば、履歴、グラフ、ツリー、キュー、スタック、行列、正規言語、3次元構造などです。こういったデータをリレーショナルデータベース上で格納するだけなら可能ですが、肝心のクエリはリレーショナルモデルでうまく表現できません。クエリがうまく表現できなくても、別々のデータベースを使い分けるのは不便ですから、そのようなデータがリレーショナルデータベース上に格納されることは多々あります。

SQLはリレーショナルモデルがベースにはなっているものの、完全にデータモデルを再現しているわけではなく、それ以外のこともたくさんできるのはこのためです。そのような場合には、**リレーショナルモデルについての定石はすべて捨て去って、ストアドプロシージャでループや条件分岐を駆使したり、アプリケーション側のコード内でデータを加工する**といった対応が必要になるでしょう。

大切なのは、今扱おうとしているデータやアルゴリズムが、**リレーショナルモデルの範疇で扱うべきなのかどうかということを、明確に把握すること**です。

SQLについて知る

SQLは、リレーショナルモデルに基づいた言語ではありますが、リレーショナルモデルとの間には大きな乖離があります。そのため、リレーショナルモデルだけを理解しても、SQLを理解することはできません。**リレーショナルモデルがどのようにSQLにマッピングするか**ということを知る必要があります。

SQLはとても柔軟な言語であり、きちんと理解するのは一苦労です。SQLでテーブル内のデータを参照するためのコマンドといえばSELECTですが、それ以外にテーブル内のデータを参照する手段はありません。SELECTという1つのコマンドに、参照のための膨大な機能が詰め込まれているのです。本稿で紹介し

たリレーションの演算である制限、射影、和、結合はもちろんSELECTに詰め込まれていますし、それ以外にも外部結合やサブクエリ、ソート、要約など、SELECTだけでも覚えるべきことは満載です[注9]。

それだけではありません。ストアドプロシージャやトリガー、ビュー、トランザクション、ユーザ権限の管理などなど、データを操作する以外にも覚えることはたくさんあります。リレーショナルモデルでは表現できないデータを扱うには、ストアドプロシージャなどは大いに活用する必要があるでしょう。

自分が欲しいデータは何か

とても根本的なことですが、クエリをきちんと書くためには、**いったい自分が何のデータがほしいのか**ということが明確になっていなければなりません。何を当たり前のことを言ってるのかと思われるかもしれませんが、これが意外にも見落とされがちなのです。

何のデータがほしいのかというのは、言い換えればアプリケーションが何をするべきものなのかということです。つまり今自分が作っているアプリケーションが何をするものかということを明確に理解しなければ、クエリをきちんと記述することもできないのです。しかしこれは一朝一夕には行きません。小さなプロジェクトや、自分が最初から書いているプログラムであれば理解することは難しいことではないかもしれません。しかし、プロジェクトが巨大であったり、前任者から仕様書などを引き継いでプロジェクトへ参加することになった場合などには、アプリケーションが何をするものなのか、本来の正しい振る舞いはどのようなものかということを理解することは、とても困難な作業となるでしょう。

アプリケーションの正しい振る舞いを知る、そしてアプリケーションが何のためにそのクエリを実行するのかを理解することが、上手にSQLクエリを書くための前提条件なのです。

データベース設計の見直し

アプリケーションの動作を知るというのは、とても難しいテーマです。開発を継続する中で新たな要件が追加されたり、コードを書いていくに連れてアプリケーションの役割が明確になったりすることで、アプリケーションにどんどん改良が加えられ、元のプログラムの姿から大きく違うものになってしまうということは、珍しくありません。

アプリケーションの振る舞いが変われば、当然それにしたがってデータベースのほうも変更されるべきです。しかしながら、既存のデータベースをそのまま使い続けるケースが非常に多く見られます。バグのないプログラムを書くことができないのと同じように、いっさい問題のないデータベースを最初から設計できるはずがありません。必要に応じてデータベース設計の妥当性を検証し、変更ができることが重要です。**データベース設計は変更があって当然**だと考えておきましょう。変更をするには、どのようにしてデータベースをリファクタリングするべきかということを知っておく必要があります。リファクタリングの手法については、拙著[注1]でも触れていますので、ご一読いただければと思います。

実装について知る

SQLクエリは、高速に実行されなければ価値がありません。いくら正確な答えが得られようとも、いくらデータベース設計がきっちりとできていようとも、実用的な応答速度ならびにスループットが得られなければ、使い物にはならないからです。

性能のよいクエリを記述する上で最も大切なことは、実装を知るということです。どのようにクエリが処理されるのかを知らなければ、コンピュータの仕事量を見積もることはでき

注9）リレーショナルモデルの結合は、SQLでは内部結合に相当します。

ません。コンピュータのリソースは有限ですから、限られたリソースを有効活用するために、同じ内容の仕事でも、できるだけ効率的に実行するに越したことはありません。どのようにすれば最も効率的な実行ができるのかということを考えるには、実装について詳しくなる必要があるのです。リレーショナルデータベースで言えば、インデックスの構造やオプティマイザの挙動、実行計画の詳細、トランザクションの振る舞い、I/O周りの挙動などが相当します。**実装を理解したうえで、クエリが効率的に実行できるかどうかを判断しながら記述する**必要があるでしょう。

高級な言語を使用すれば、その言語の使い方だけを理解すれば十分であり、中身まで知らなくてもよいというような意見を耳にすることがあります。しかし、筆者はその考え方には反対です。やはりコンピュータ上で実行するものである以上、内部で何が行われているかを理解することは、実用的なプログラムを作る上で必要不可欠です。データベースソフトウェアの実装だけでなく、OSやネットワーク、ストレージと言った、コンピュータの動作原理に関係する知識も必要です。そういった知識をふまえ、実際にプログラムがどのように実行されるかを想像することが大切なのです。まれにO/Rマッパーを使えばSQLを知らなくてもよいという意見も耳にしますが、当然それはよくない考えだと筆者は思います。

また、いくら実装を知ったところで、人間が頭の中でプログラムの動作を完全にシミュレートできるわけはありません。したがって、本当に満足な性能が得られるかどうかは、実際にクエリを実行してたしかめるしかありません。つまり、クエリの性能は、ベンチマークテストによってたしかめる必要があるのです。クエリを書くときには実装を理解して効率的に実行できるような記述を心がけつつ、最終的にベンチマークテストでたしかめることをお勧めします。

トランザクション

リレーショナルデータベース上で実行されるのは参照系のクエリだけではありません。更新系の処理も大量に、しかも同時に実行されるのが一般的です。同時に、しかも大量に更新が行われると、同時に同じデータへのアクセスも発生してしまうことになります。当然ながら、同じデータへアクセスすると、データに不整合が生じてしまう可能性があります。このような問題についての解説で、最もよく用いられるのが銀行の口座で、(A)引き出しと(B)預け入れの操作を同時に行うという例です。それぞれの操作は大雑把に言うと次のようなことを処理します[注10]。

- A-1:口座の残高を確認する
- A-2:口座の残高と引き出す金額x円の差額を計算する
- A-3:口座の残高がマイナスでなければ、x円を客へ支払い、口座の残高を書き込む

- B-1:口座の残高を確認する
- B-2:客がy円を支払う
- B-3:口座の残高と預け入れる金額y円の合計を計算する
- B-4:口座の残高を書き込む

(A)の処理も(B)の処理も、それぞれ順番に実行されれば何の問題もありません。しかし、同時に実行してしまうと結果に不整合が生じてしまいます。たとえば現在の口座の残高が100万円だったとしましょう。そこへ、20万円引き出す処理と、30万円預け入れる処理を同時に行ったとします。もし、(A-2)と(A-3)の間に(B)の処理がすべて割り込んでしまうと、残高は本来110万円のはずが、(B)によって更新された残高は完全に上書きされてしまい、結果は80万円になってしまうでしょう。これ

注10) 手数料などについては考慮していません。

は大問題です。

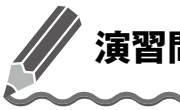
このように、**同時アクセスによって生じるデータの不整合を防ぐ手段がトランザクション**です。そのほかにも、トランザクションはデータベースサーバがクラッシュしたときに、データを正常な状態へリカバリするといった機能も提供します。データが正しく保たれてこそ、データベースは役に立ちますから、トランザクションはデータを保護するために欠かせない技術なのです。本稿ではトランザクションについての詳細には踏み込みませんが、とても重要な技術ですので、ぜひしっかりと勉強してみてください。

テスト

プログラムにはテストが必須です。これは汎用プログラミング言語を用いた場合だけでなく、SQLにも当てはまります。先ほど述べたようなベンチマークテストだけでなく、そもそもSQLクエリによって想定したデータが得られるかどうか、あるいは想定どおりに更新が行われるかといった機能面についても、しっかりとテストする必要があります。そのようなテストがなければ、本当にそのSQLが正しいのかどうかということは保証できません。

テストを記述するのは、SQLそのものに対してではなく、アプリケーションがデータベースへアクセスする機能に対して行います。最終的に、アプリケーションが正常にデータを得られるかどうか、正常に更新ができるかどうか、つまりアプリケーションの振る舞いが正しいかどうかが重要なのです。その点についてしっかりとテストしておけば、データベース側で何らかの変更を行った場合でも、アプリケーションの機能が損なわれないかどうかをたしかめることができます。

しっかりとテストをすることで、アプリケーションの品質が高まると同時に、データベースのリファクタリングがやりやすくなるのです。



演習問題

それでは本稿の締めくくりとして、少し演習問題を出しておきたいと思います。次に挙げるような内容のデータが、リレーショナルモデルに当てはまるかどうかどうかについて考えてみてください。

問題1 レシピサービスのデータ構造

あなたは今、レシピの共有をテーマとしたWebサービスを作ろうとしていると仮定してください。まず、次のようなデータを格納するテーブルを設計するでしょう。

- ユーザ情報：ユーザID、名前（ハンドル名）、メールアドレス、ログイン情報
- レシピ：レシピID、料理の名前、材料、手順、食べる人数
- お気に入りのレシピ：ユーザID、レシピID

まずはこの3つの情報をそのまま3つのテーブルで表現してみてください。すると第一正規形にならないことに気づくと思いますので、まずは正規化に取り組んでみましょう。

Q これらのテーブルはリレーショナルモデルの範疇にあるでしょうか？　ないでしょうか？

問題2 レシピ検索機能

レシピの検索について考えてみましょう。おそらくユーザは料理のカテゴリから作りたい料理のレシピを検索するでしょうから、料理のカテゴリの情報をつけたほうがよさそうですが、どのように表現するかは悩ましいところです。ツリー構造にして大カテゴリ（たとえば肉のおかず）、中カテゴリ（牛肉）、小カテゴリ（煮込み）のように区別するか、それとも単なるタグにして、1つの料理に日本料理、鍋料理、魚料理と言ったタグをつけるべきか、それとも両方にすべきかという選択があるで

しょう。いずれにしても、そういった情報は別のテーブルで表現するのがよいでしょう。

Q こういった検索機能はリレーショナルモデルの範疇にあるでしょうか？　ないでしょうか？

問題3 ソーシャル機能

次に、あなたはソーシャル機能を拡充することを思いつきます。よいレシピを作成するユーザとお友達になる、あるいはフォローする機能。レシピの更新や日々の出来事などをタイムラインとして綴る機能。さらに友人のタイムラインをまとめて時系列で表示する機能などが考えられます。

Q 次の機能がリレーショナルモデルの範疇かどうかについて考えてみてください。

・友達リスト
・個人のタイムラインを表示する
・友人やフォローしている人のタイムラインをまとめて表示する
・友達の候補をユーザに提示する

解答例

先ほどの3つの問いに対する解答例を解説します。先に読み進める前に、ぜひ先ほどの問題についてご一考してみてください。

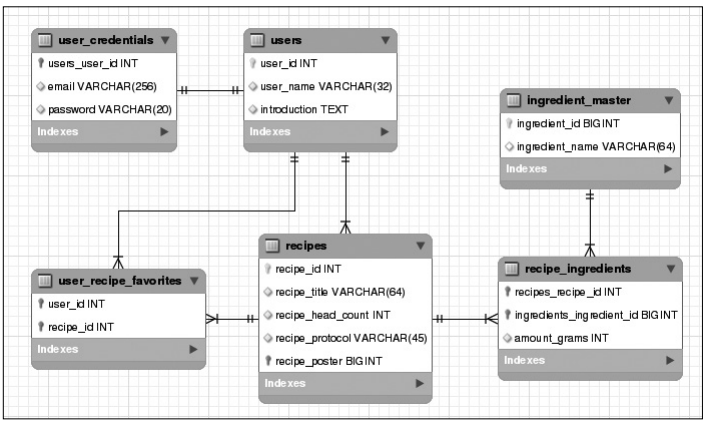
解答1 レシピサービスのデータ構造

いずれもリレーショナルモデルの範疇で設計することができます。ユーザの集合、レシピの集合、どのユーザがどのレシピを気に入っているかという事実の集合です。材料については、レシピテーブルの1行として表現すると、第一正規形を満たしませんので、どの材料がどのレシピで使われているかということを示すテーブルを作成するべきでしょう。ただし同じ材料が複数のレシピに出現することを考えると、材料の種類一覧を表すテーブルが合ったほうがよいでしょう。

手順をどう表現するかは、やや課題があります。手順全体を1つの文書のように扱うのであれば、1つのカラムとして表現してもよいでしょう。1つのステップをデータの単位とするなら、別のテーブルで表現するべきですが、その場合はリレーショナルモデルの範疇にはなりません。なぜならば、手順の各ステップには前後関係があるからです。集合は要素の間に順序があってはいけませんので、手順はリレーショナルモデルでは扱えません。

図7にER図を示します。この解答例では、手順は1つの文書であるという前提に

▼図7　A1のER図



基づいています。また、ユーザのパスワード認証の部分だけ別テーブル分けています。

解答2 レシピ検索機能

まず、カテゴリを設定する方法についてですが、大カテゴリ、中カテゴリといった構造は、階層構造≒ツリー構造になってしまいますので、リレーショナルモデルは適用できません。ツリーはSQL上でも表現する方法はありますので、リレーショナルデータベース上で管理しても問題はありませんが、アプリケーションが別の方法で管理してもかまいません。

タグは少々やっかいです。というのも、タグをつけるという行為自身は、リレーショナルモデルの範疇で行うことができます。たとえばタグテーブルは**図8**のように設計できるでしょう。

タグから目的のレシピを探すためのクエリも、リレーショナルモデルに沿ったものになります。問題は、そのクエリが実用的な性能にはならないという点です。タグを使って検索をする場合、1つのタグを指定しただけでは、膨大な数のレシピが表示されるでしょう。たとえば「肉料理」というタグがある場合、レシピの1/3ぐらいは該当するかもしれません。そこで、検索結果を絞り込むために、ユーザは複数のタグを指定したくなるでしょう。そうすると、そのクエリは2つのタグを検索した結果を、結合(JOIN)するという構造を持ったものになります。いわば自己結合です。次に、そのようなクエリの例を示します[注11]。

```
SELECT a.recipe_id
FROM recipe_tags t1
INNER JOIN recipe_tags t2
USING (recipe_id)
WHERE t1.tag = '肉料理'
AND t2.tag = '煮込み料理'
```

t1、t2双方から、膨大な数の検索結果がヒットしますので、リレーショナルデータベースは、膨大な行数を操作する必要があり、満足な性能が出るはずがありません。**リレーショナルモデルに適合する場合でも、実装上効率的にクエリを解決できず、満足な性能が得られない場合がある**ということは覚えておきましょう。この問題に対しては、拙著[注1]で詳しく解説していますので、そちらを参照してください。

解答3 ソーシャル機能

それぞれの機能について考えてみましょう。

友達リスト

これはリレーショナルモデルの範疇で設計することができます。**図9**は友達リストのテーブルの例です。このように、2つのカラムを用いて友だち関係であることを示しています。

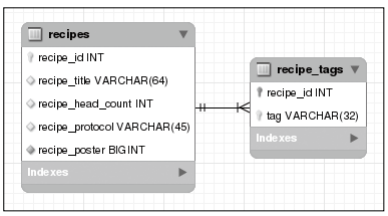
このテーブルにデータを格納するには2つの方式が考えられるでしょう。

- user_id_1とuser_id_2で、どちらが大きいIDになるかを事前に決めておく
- 友達であることをフォローで表現する。互いに友達であると認識している場合には、相互にフォローする

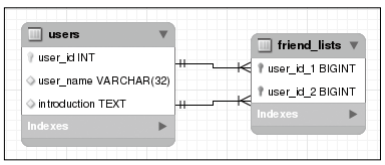
前者の方法では列に順序が生じてしまうた

注11) このテーブルでは、1つのrecipe_idに対して、複数のタグを指定することができます。

▼図8 タグの例



▼図9 友達リストの例



め、リレーショナルモデルにはなりません。後者はデータ量は増えますが、リレーショナルモデルに合致しますので、リレーショナルデータベース上では好ましい設計であると言えます。

個人のタイムラインを表示する

集合は要素間に順序がないため、時系列で並んだデータは集合にはなり得ません。したがってタイムラインはリレーショナルモデルには適合しないデータであると言えます。ただし、個人のタイムラインは1つのテーブルとして表現するのは難しくありませんし、クエリもORDER BY句を用いればすんなりと書けてしまうでしょう。次の例は、**user_timelines**というテーブルに対して**ID=1000**のユーザのタイムラインを10件取得するものです(テーブル定義は省略しますが、データは**tl_id**の順に格納されているものとします)。

```
SELECT tl_id, tl_content, tl_timestamp
FROM user_timelines
WHERE tl_user_id = 1000
ORDER BY tl_id DESC LIMIT 10
```

ページ送りをしたい場合、たとえば最後に取得したデータの**tl_id**が**10000**だった場合には、WHERE句に**tl_id < 10000**という条件を追加すればよいでしょう。

友人やフォローしている人のタイムラインをまとめて表示する

この場合も先ほどと同様、リレーショナルモデルでは表現できないデータです。しかも、データの出どころがたくさんあるため、ORDER BY句がインデックスで解決できない公算が高く、さらに悩ましいデータ構造となります。検索を高速化するよい方法は、フォローしているすべてのユーザに対して、事前に更新があったことを**配信**してしまうことです。配信済みのデータは、上記の場合と同様のクエリでタイムラインを取得することができます。問題は、配信をどのように実装するかということでしょう。とくにフォロワーが多い人が更新した場合には、配信先も多くなってしまいますので、それに応じてデータ量も増え、場合によっては膨大な量のデータを操作しなくてはならないことになるでしょう。そのため、コンテンツ本体は別テーブルに分け、IDだけを配信するのが現実的でしょう。

友達の候補をユーザに提示する

リレーショナルモデルでは表現できない課題です。ユーザの候補かどうかということは、論理的に導くことができないからです。友達の友達を表示するということなら難しくはありませんし、それだけならリレーショナルモデルの範疇です。しかし、友達かどうかの判定基準はそれだけではないはずですので、候補を選び出すには、レコメンドエンジンのような、もっと複雑なアルゴリズムが必要になるでしょう。

こういったよくありがちなテーマであっても、リレーショナルモデルの範疇とそうでないデータに分かれてしまうということを、ご理解いただけたでしょうか。設計とは無限にある可能性の中から1つを選択することですので、この解答以外にも、もっとよいものがあるかもしれないということは、付け加えておきたいと思います。

まとめ

本稿では、アルゴリズムとデータ構造が切り離せない存在であることを説明しました。大切なことは、今必要なアルゴリズムは何かということを見極め、そのうえで適切なデータモデルを選択するということです。リレーショナルデータベースはリレーショナルモデルが基本的なデータ構造ですから、リレーショナルモデルが適している場合に真価を発揮します。リレーショナルモデルはたいへん応用が利く

データモデルであるため、適しているケースはとても多く、これからもリレーショナルデータベースは広く使われることでしょう。

万物に完璧なものはなく、リレーショナルモデルも例外ではありません。リレーショナルモデルをよく理解すれば、自ずとその限界もわかってきます。リレーショナルモデルでは表現できないデータは、リレーショナルモデルを適用すべきではありません。そのデータに合ったデータ構造を用いましょう。幸か不幸か、SQLはリレーショナルモデル以外のデータも扱えるようになっています。それは、SQLがリレーショナルモデルを基礎としつつも、完全には一致していないということに起因します。リレーショナルモデルに適合しないデータであっても、SQLの範疇でカバーできるケースがたくさんあるということを覚えておきましょう。

さらに実践でSQLを利用するには、満足な性能が得られなければなりません。そのためにはリレーショナルデータベースや、その配下にあるOSなどの実装を理解する必要があるでしょう。また、トランザクションをきっちりと実践したり、テストをしっかり書くといったことも重要です。

プログラマの生産性は、できる人とそうでない人では100倍も違うと言われています。たいへんインパクトのある数字ですが、SQLも例外ではありません。リレーショナルモデルをきちんと理解し、リレーショナルモデルを適用できるデータかどうかを見極められるかどうかで、プログラマの生産性は大きく異なって来るでしょう。しかもデータベースの場合は、同じ設計をたくさんの人で使いまわすため、データベース設計がよくなかった場合の影響は甚大です。ぜひリレーショナルモデルをきちんと理解したうえで実践し、快適なデータベースライフを送ってください。SD

第03時限

Javaを使いこなしていますか?

オブジェクト指向の実践的な考え方とやり方

変更に強いプログラムの書き方

Author 増田 亨(ますだ とおる)　ギルドワークス㈱(http://guildworks.jp/)

なぜオブジェクト指向で開発するのか

ソフトウェア開発は変更コストとの戦いです。コードが1行もないところから、コードの追加と変更を繰り返す作業がソフトウェアの開発です。初期の開発が終わり、ソフトウェアが使われ始めると、想定外の使われ方に対応したり、機能の追加や変更の要求に応えるために、ソフトウェアの変更作業が続きます。

オブジェクト指向は、このソフトウェアの変更を楽で安全にするための実践的な考え方であり工夫です。

同じ機能は、いろいろな書き方で実現できます。そして、書き方の違いによって、変更がたいへんになることもあれば、変更が簡単になることもあります。

この記事では、Javaの例を使いながら、オブジェクト指向を使って、変更がやりやすいプログラムの書き方、その考え方とやり方を紹介します。

変更コストの敵を知る

変更がたいへんになるプログラムの特徴は次のとおりです。

- どこに何が書いてあるかわからない
- 同じ修正があちこちで必要
- 変更の副作用がこわくて変更ができない／変更したら思わぬところに副作用が起きた

なぜ、このようなことが起きるのでしょうか。どのようなことに気をつけてソフトウェアを設計すれば、どこに何が書いてあるかわかりやすく、変更が必要な個所が少なく、変更の副作用の心配をしなくてよくなるのでしょうか。

オブジェクト指向では、次の点に注目して、この問題の改善に取り組みます。

- データの扱い方の工夫
- 場合分けの書き方の工夫
- プログラムの分割のしかたの工夫
- 名前の付け方の工夫

具体的に見ていきましょう。

関連するデータとロジックは同じクラスに書く

オブジェクト指向では、データとロジック(データの操作)を、1つのクラスにまとめることを重視します。

逆に言えば、次のようなプログラムは、オブジェクト指向「らしくない」書き方です。

- `getter/setter`だけの「データのいれものクラス」と、ロジックを集めた「機能クラス」とに分ける
- `String getXxxx()`の形式で、オブジェクト内部のインスタンス変数を、加工も処理もせず、そのまま取得する

・オブジェクトを引数なしのコンストラクタで生成する
・void setXxxx(引数)形式のメソッドを使って、オブジェクトのインスタンス変数を書き換える
・StringUtil、DateUtilなど、データを操作するロジックを集めたライブラリクラス(共通サブルーチン集)を使う

オブジェクト指向は、これとは逆の発想です。

・データを操作するロジックは、そのデータをインスタンス変数に持つクラスに一緒にする(データクラスと機能クラスに分けない)
・オブジェクトを生成するときは、必ず操作対象のデータをコンストラクタの引数として渡す
・オブジェクトに、getXxxx()の形式で内部のインスタンス変数を要求しない
・オブジェクト内部のインスタンス変数を、外から変更しない(setterメソッドを使わない)

このようにする理由は3つです。

・データとロジックを別のクラスに分けると、データを操作するロジックが、あちこちに重複しやすくなる
・オブジェクトのインスタンス変数を書き換える操作は(状態が変わるので)、プログラムの動作が不安定になりやすい
・データを引数で渡すことを繰り返すと、そのデータを使った判断/加工/計算のロジックが、どこ書いてあるか、見つけ出すのに苦労する

データと、そのデータを使った判断/加工/計算のロジックを同じクラスに書いておけば、そのデータを扱うロジックを変更するときに、そのデータを持つクラスに注目すればよくなります。1つの要求変更でプログラムのあちこちを調べ回る必要はありません。ロジックは、そのクラスに一元化されていますから、あちこちに同じ変更を繰り返す必要はありません。

オブジェクト内部のデータ(インスタンス変数)を上書きはしないようにすれば、そのオブジェクトの挙動が安定します。メソッドを呼び出すタイミングや順番が変わっても、同じ結果を返すようになります。使う側にとって、安心して使えます。

このように、データとロジックを一緒のクラスにまとめる、という発想が、オブジェクト指向の基本アイデアの1つです。

基本のデータ型を使うロジックを用途ごとのクラスにまとめる

データとロジックを同じクラスにまとめる、具体的なやり方を説明します。

基本は、プログラミング言語に用意されている汎用的なデータ型を使って、用途を限定した「独自のクラス」を作ることです。

Java言語に用意されている基本的なデータ型は**表1**のものがあります。

これらの基本的なデータ型とそのメソッドは、いろいろな用途に使えるように汎用的に設計されています。

扱える数字の範囲、文字列の長さ、文字の種類、日付の範囲、コレクションが持てる要素数。どれも実質的には無制限です。

また、それぞれのクラスに用意されているメソッドも、add(引数)、plus(引数)、get(引数)など、任意の引数を渡して、どのような範囲の操作もできるようになっています。

このような汎用的なデータ型と汎用的な操作をそのまま使うと、次の問題が起きがちです。

▼表1　Javaでよく使う基本的なデータ型

数値	Integer, BigDecimal
文字列	String, StringBuilder
日付時刻	LocalDate, LocalDateTime
コレクション	List<型>,Set<型>,Map<型>

・業務的に正しくないデータが混入しても動いてしまう
・プログラムのどこを見ても同じような記述が並ぶ(基本データ型だらけのコード)
・plus(1)など汎用的な操作が、どのような意図なのか、判断しにくい

「誕生日」を扱うロジックを考えてみましょう。

LocalDateクラスは、10億年前から10億年後まで20億年分の日付を扱えます。通常の「誕生日」の概念から考えると、ほとんどが異常なデータです。

LocalDateクラスは、年月日の任意のフィールドで、任意の加算/減算ができます。「誕生日」について、知りたいこと、やりたいことは、それほど、汎用的なデータ操作ではありません。

オブジェクト指向らしい設計は、汎用的なデータ型であるLocalDateを内部に持つ、独自の「誕生日」クラスを作ることです。

たとえば**リスト1**のようなクラスです。

汎用的なLocalDateを内部に持ち、用途を「誕生日」に限定した独自のクラスを作るこのやり方のメリットは次のとおりです。

・「誕生日」に関するロジックを1個所に集約できる
・間違った日付が混入しないので、安心して使える
・オブジェクトの持つ日付を変更できないので、どのタイミングでメソッドを呼び出しても、同じ結果であることを保証できる

DateOfBirthのような独自のクラスを作らない場合と比較すると、違いがよくわかります。

誕生日のデータを持つLocalDateオブジェクトを使えば、誕生日の妥当性のチェックや、月日だけを取り出す操作は、プログラムのどこでも簡単に記述できます。しかし、これが問題です。誕生日に関するロジックを変更したいときに、プログラム中のすべてのLocalDate型を調べることになりかねません。LocalDate型は「誕生日」以外の用途にも使いますから、プログラムの中で、「誕生日に関連する」個所を特定することは、たいへんな作業です。

一方、DateOfBirthクラスを使えば、ロジックは、このクラスだけに集約されています。また、変更の影響範囲は、DateOfBirthクラスを使っている個所だけに限定できます。

この「誕生日」クラスが、オブジェクト指向で変更コストを下げる典型的な書き方です。

同じ考え方で、次のような目的を限定したクラスをたくさん作るのがオブジェクト指向らしい設計スタイルなのです。

・数量、単価、消費税、人数、……
・申込日、予定日、期間、期日、……
・氏名、電話番号、メールアドレス、備考、メモ、……

このように、アプリケーションで使うデー

▼リスト1　用途を限定した独自の誕生日クラス

```
class DateOfBirth {

  @NotEmpty(message = "必須")
  @Past(message = "過去日付")
  LocalDate date;

  DateOfBirth( LocalDate date ) {
    this.date = date;
  }

  MonthDay withoutYear() {
    return MonthDay.from(date);
  }

  Year onlyYear() {
    return date.getYear();
  }

  int daysTo() {
    LocalDate today = LocalDate.now();
    Period period = date.until(today);
    return period.getDays();
  }

  String asText() {
    DateTimeFormatter formatter = ↵
DateTimeFormatter.ofPattern("y年M月d日生");
    return date.format(formatter);
  }
}
```

タを扱うための用途限定のクラスを設計するパターンを「値オブジェクト(VO:Value Object)」と呼びます。

数値、文字列、日付を扱う基本的なデータ型を内部に持って、用途を限定した「値オブジェクト」を作ることが、ソフトウェアの変更を楽に安全にするオブジェクト指向らしいプログラミングのやり方です。

データの意図と操作の意図のレベルを合わせる

オブジェクト指向で「データとロジック」を一体で考えるのは、単純に、1つのクラスにデータとロジックをまとめて書く、というだけではありません。

もっと大切なのは、そのクラスで扱うデータの意図と、そのデータを操作するメソッドの意図の「レベル」を合わせることです。わかりにくいですね。DateOfBirthを例にして説明しましょう。**図1**を参考にしてください。

LocalDateは、内部にint year、short month、short dateのプリミティブなデータ型を持っています。intとshortの使い分けは、メモリの使用効率を意識した選択です。

LocalDateのメソッドの中身は、このintやshortのインスタンス変数を判断/加工/計算するロジックです。

LocalDateのメソッドの中のコードは、メモリ効率などコンピュータのしくみを強く意識した内容です。「コンピュータのしくみを意識する」という点で、レベルが一致しています。

LocalDateを外側から見てみましょう。クラス名からわかるとおり、このクラスは「日付」を(現地のタイムゾーンで)扱うことを意図しています。

それと対応して、LocalDateのメソッド名はplusDays()/atStartOfDay()/isBefore()など、日付を意識した名前になっています。「日付」を扱うという点で、レベルが一致しています。

そして、LocalDateを使ったDateOfBirthクラスは、いろいろな日付の中でも「誕生日」だけに用途を限定したクラスです。

メソッドも「誕生日」を使った判断/加工/計算に使うメソッド名だけです。LocalDateのように汎用的な日付計算はできません。「誕生日」という関心事のレベルで一致しています。

このように関心事のレベルをそろえながら、int/shortを使ってLocalDateクラスを定義し、LocalDateクラスを使って、DateOfBirthクラスを階層的に作っていくのがオブジェクト指向らしいプログラミングのやり方です。

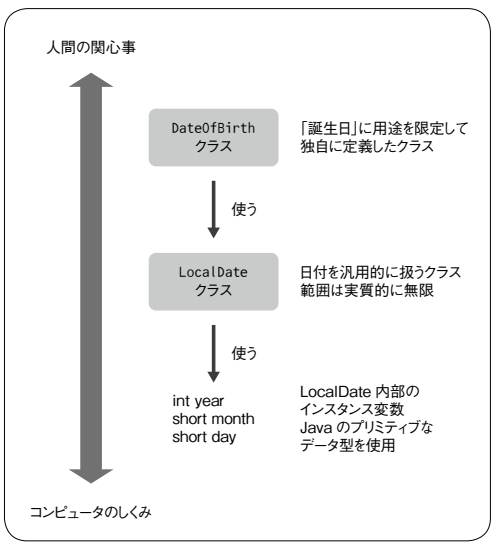
コンピュータのしくみを意識したレベルのデータ/ロジックや、それを使った汎用的なデータ型は、Javaが言語として用意しています。

オブジェクト指向を使って、アプリケーションを開発する技術者がやるべきことは、DateOfBirhtクラスのように、ソフトウェアを利用する人たちの関心事を直接表現できるクラスを発見し、実装することなのです。

ドメインオブジェクト

「誕生日」クラスのように、ソフトウェアを利用する人たちの関心事に直接対応するオブジェクトを「ドメインオブジェクト」と呼びます。

▼図1　データの扱い方の階層



ソフトウェアの対象領域(ドメイン)の関心事に由来するオブジェクト、という意味です。

オブジェクト指向のプログラミングでは、このドメインオブジェクトの発見と実装が主要な課題です。ドメインオブジェクトは、利用者の関心事に直接対応します。ですので、利用者の言葉で語られる、機能の修正や拡張の対象となる個所は、クラス名から簡単にかつ正確に特定できます。どこに何が書いてあるか調べるのがたいへん、という問題を解決できるわけです。また、利用者の関心事の単位にデータとロジックを整理したドメインオブジェクトは、変更の副作用を減らします。

「誕生日」という関心事についての変更は、「締切日」という別の関心事に影響はしません。

`DateUtil`クラスに日付の計算処理を集約した、オブジェクト指向でないスタイルだと、「誕生日」と「締切日」の両方で、同じ日付計算メソッドを使っている可能性があります。「誕生日」に関する仕様を変更したつもりが、「締切日」の判定に影響がでた、というようなことが起きる原因です。

クラス以外にも、変更の対象個所を特定したり、影響範囲を狭くコントロールするしくみがあります。複数クラスのグループを宣言するパッケージ(package)宣言です。

パッケージ名も、利用者の関心事、利用者の要求の説明に使われる言葉を使うようして、どこに何が書いてあるかわかりやすくします。

そして、パッケージ内のクラスとメソッドは、可能な限り、Javaのデフォルトのスコープ(パッケージプライベート)にします。パッケージプライベートのクラスやメソッドは、パッケージの外部から参照できません。変更が外部に波及することを制限する良いプログラミングスタイルです。

コレクション型の扱い

`List<Order>`など、コレクション型のデータは、ソフトウェア変更を難しくする原因になりがちです。

コレクションの操作は、ループ処理が多く、バグが混入しやすく、変更がやりにくい個所です。また、コレクションの要素の追加や削除、要素の内容の変更を行うと、状態が変わるため、プログラムの挙動が不安定になります。

コレクション型のロジックの変更はやっかいで危険な作業です。

コレクション型も、数値／文字列／日付を扱う「値オブジェクト」と同じように扱います。コレクション型のインスタンス変数と、そのコレクション型を操作するロジックを1個所に集めた独自のクラスを作るのがオブジェクト指向らしいやり方です(**リスト2**)。

この設計のやり方を、コレクション型の変数1つを特別扱いするクラス、という意味で「ファーストクラスコレクション」と呼びます。こうすることで、変更の対象となるロジックが書かれた場所を特定しやすく、変更の影響範囲を狭く閉じ込めやすくなります。

また「注文履歴」クラスも、利用者の関心事に直接対応しています。つまり「ファーストクラスコレクション」は、ドメインオブジェクトを発見して実装する方法でもあるのです。

ソフトウェアを利用する人たちの関心事に合わせた「値オブジェクト」や「ファーストクラスコレクション」を積極的に作ることが、オブジェクト指向らしい、ソフトウェアの変更が楽で安全になる工夫です。

次に変更の大敵である「場合分け」との戦い方を説明します。

場合分けの書き方の工夫

if-elseは、変更がたいへんになる

「場合分け」の書き方の違いは、ソフトウェアの変更コストに大きく影響します。

具体例で見てみましょう(**リスト3**)。

給付金(payAmount)の金額を、「死亡時」「退職時」「通常時」で、場合分けする処理を、if-else構文で記述した例です。

この書き方は、もっと変更が楽で安全になるように書き換えることができます。

早期リターン

前の例は、一時変数resultを使い、if文を終わったあとで、resultを返しました。

しかし、この例では、条件に一致すれば、その時点で給付金額が確定します。ですので一時変数は使わず、条件文の中でただちにreturnできます(**リスト4**)。

▼リスト2　注文履歴クラス

```
class OrderHistory {

  List<Order> orders;

  OrderHistory( List<Order> orders ) {
    this.orders = orders;
  }

  Quantity totalOfThisYear() {
    // 今年注文された総数
  }

  Orders lastOf() {
    // 直近の注文
  }

  PendingOrders pending() {
    // 未受注の注文一覧
  }
}
```

▼リスト3　if-elseを使った書き方の例

```
double getPayAmount(){
  double result;
  if(isDead()){
    result = deadAmount();
  } else if(isRetired()){
    result = retiredAmount();
  } else {
    result = normalAmount();
  }
  return result;
}
```

先ほどの例よりも、一時変数がなくなっているので、その分、シンプルになりました。これが「早期リターン」という書き方です。

elseを使わない書き方

「早期リターン」で書くと、実はelse句を書く必要はありません(**リスト5**)。

- 「死亡した時」は、ただちに「死亡時の金額」を返す
- 「退職した時」は、ただちに「退職時の金額」を返す
- それ以外の一般的な場合は、「通常の金額」を返す

ずいぶんとすっきりしたコードになりました。最初の一時変数を使ったif-else形式の書き方と比較してみてください。

特殊な場合をif文で判定し、else句を使わずに早期リターンするこの書き方を「ガード節」と呼びます。

「リファクタリング」本の「条件記述の単純化」で紹介されているテクニックです。if-else構文を見つけたら、「早期リターン」や「ガード節」に書き換えることを検討してみましょう。ちょっとした書き方の変更ですが、「場合分け」のコー

▼リスト4　金額が確定したらすぐリターンする

```
double getPayAmount(){
  if(isDead()){
    return deadAmount();
  } else if(isRetired()){
    return retiredAmount();
  } else {
    return normalAmount();
  }
}
```

▼リスト5　早期リターンの例

```
double getPayAmount(){
  if(isDead())  return deadAmount();
  if(isRetired()) return retiredAmount();
  return normalAmount();
}
```

ドがすっきりし、変更時にバグが紛れ込む可能性を減らせます。

複文より単文

「ガード節」と「早期リターン」を適用した最後の例は、独立した3つの「文」が並ぶシンプルな分岐構造です。

if-else構文を使った最初の2つの例は、if文のelse句の中に、さらにif文を書いている構造です。つまり文の中に文を書く「複文」構造です。「複文」構造がわかりにくいのは、自然言語でも、プログラミング言語でも同じです。最後の例がすっきりしてわかりやすいのは、文の中に文を書いた「複文」ではなく、「単文」を並べたシンプルな構造だからです。else-ifのような「複文」構造を見つけたら、「早期リターン」+「ガード節」で「単文」構造にできないか検討してみましょう。「単文」構造にできれば、コードは読みやすく変更しやすくなります。

処理の独立性

「単文」構造にした最後の例は、処理(文)の独立性も高くなっています。「複文」は、「文」と「文」の関係があきらかに「密結合」ですね。「単文」を並べる方式は、それぞれの「文」の結合度(影響度合い)が下がります。

たとえば「離婚したときの給付金額」を追加してみましょう。単文を並べた方式は、別の単文を追加するだけです。簡単で間違いが減ります(**リスト6**)。

この3つのif文は、順番を入れ替えても何も問題がおきません。処理どうしの関係が「疎結合」になっている良い点です。

このように、同じ場合分けでも、密結合の「複文」構造ではなく、できるだけ、独立性の高い「単文」に分解することが、オブジェクト指向らしい場合分けの書き方の基本アイデアです。

オブジェクト指向らしい場合分け

今までの例は、if文を使って、処理の場合分けを書きました。Javaなどオブジェクト指向のしくみを取り入れた言語では、「場合分け」は、もっと別の書き方ができます。

発想は単純です。

＜場合ごとのロジック、それぞれ独立した別のクラスにしてしまう＞

場合分けを、「単文」を並べる方式にすると、それぞれのif文とif文の関係が疎結合になりました。これを発展させて、場合ごとのロジックを別々のクラスに分けて書く、という発想です(**リスト7**)。

こうすれば、それぞれの「場合」のロジックをどこに書くかが明確です。

「死亡時」の金額の計算方法を変更するために、DeadAmountクラスを修正しても、他のクラスに影響しません。場合ごとの変更を分離し独立させて扱うことが、この書き方の狙いであり、メリットです。

場合ごとにクラスを分けると、複数のクラスを使い分けるのはたいへんになります。そこで3つの場合ごとのクラスの違いを意識せずに同じように扱うしくみを導入します。それが「インターフェース宣言」です(**リスト8**)。

「給付金額」を知りたいクライアント(使う側)

▼リスト6　離婚の場合を追加する変更

```
double getPayAmount(){
  if(isDead())  return deadAmount();
  if(isRetired()) return retiredAmount();
  if(isSeparated()) return separatedAmount();
  return normalAmount();
}
```

▼リスト7　場合ごとのクラス

```
class DeadAmount {  double getAmount(){...}; }
class RetiredAmount { double getAmount(){...}; }
class  NormalAmount { double getAmount(){...};}
```

のクラスは、こんな書き方になります(**リスト9**)。

クラス図で描くと、こんな感じです(**図2**)。

クライアントクラスは、PayAmountという「型」を意識しているだけで、「死亡時」「退職時」「通常時」の場合分けは、意識していません。これが「多態」とか「ポリモフィズム」と呼ぶ、オブジェクト指向らしい場合分けの書き方の例です。

場合ごとのオブジェクトをif文なしに生成する

先ほどの例では、Clientクラスには場合分けは登場しません。しかし、インスタンス変数で宣言したPayAmount型で参照する実装クラスのオブジェクトを生成するときに、どこかにif文かswitch文が必要になりそうです。

Javaの場合は、列挙型(enum)という便利なしくみがあって、実装クラスの生成も、if文なしに記述できます。

たとえばこんなenumを宣言します(**リスト10**)。

こうすると、PayAmount型で参照するオブジェクトは、タイプ名の文字列から、生成できます(**リスト11**)。

Enum#valueOf()メソッドは、if文やswitch文を使わずに、場合ごとに異なるオブジェクトを生成できる便利でわかりやすい方法です。

▼リスト8　場合ごとのクラスを束ねるしくみ

```
interface PayAmount {
   dboule getAmount();
}

//それぞれのクラスでインターフェースを実装する
class DeadAmount implements PayAmount {...}
class RetiredAmount implements PayAmount {...}
class  NormalAmount implements PayAmount {...}
```

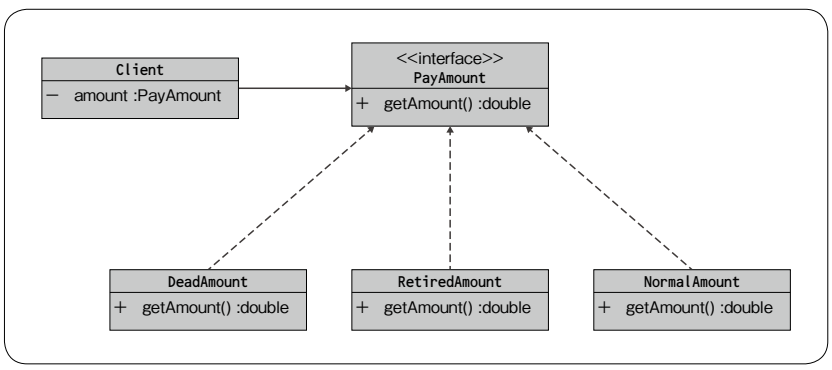
▼リスト9　給付金額を知りたいクラス

```
class Client {
   private PayAmount amount ;

  double getAmount() {
    return amount.getAmount();
  }
}
```

▼リスト10　enumで場合ごとのクラスを宣言する

```
enum PaymentType{
   dead( new DeadAmount() ),
   retired( new RetiredAmount() ),
   normal( new NormalAMount() );

   private  PayAmount amount;

   private PaymentType( Payamount amount ) {
     this.amount = amount;
   }

  double getAmount() {
    return amount.getAmount();
  }
}
```

▼図2　クラス図



オブジェクト指向で場合分けの変更コストを下げる

業務アプリケーションでは「区分」や「種別」ごとのビジネスルールが複雑になりがちです。また、「区分」や「種別」は変更が必要になることも多く、変更ミスやバグの温床になります。

「区分」や「種別」を使ったビジネスルールを記述するプログラミングでは、一時変数とif-elseを使った最初の書き方と、最後に紹介した「多態」「列挙型(enum)」を使った書き方では、ソフトウェアの変更コストがそうとう違ってきます。

「プログラムを動かす」だけなら、どの書き方でも同じです。しかし「変更コスト」を考えると、if-elseを使った「複文」構造は、できるだけ避けるべきです。

オブジェクト指向らしい設計の考え方とやり方

部品から組み立てる

「値オブジェクト」「ファーストクラスコレクション」「列挙型」を使った場合ごとのクラス分け。いずれも、プログラムの全体構造ではなく、アプリケーション機能を実現する部品に注目したアプローチです。

オブジェクト指向は、部品を発見し、部品を実装しながら、全体を組み立ていく、ボトムアップを重視したアプローチです。

オブジェクト指向の、この部品からアプローチするスタイルは、ジグソーパズルに似ています。

最初はどのピースがどこにはまるかがわかっていません。

▼リスト11　タイプ名の文字列から生成する

```
PaymentType  type = Payamount.
valueOf("dead");
  ……(中略)……
double amount = type.getAmount();
```

わかりやすいピースをなんとか見つけて配置していきます。そのピースのまわりにだんだん別のピースがはまり始めます。

そういうピースの塊が、あちこちで成長を始めますが、まだ大きな空白も残ったままです。

しかし、あるところまでピースの塊の成長が進むと、急激に空白がうまります。部品から出発して全体を組み立てていくオブジェクト指向のボトムアップのアプローチはこれとよく似ています。

実際には、オブジェクト指向によるボトムアップのアプローチは、もう少し複雑です。

登場する部品は、ジグソーパズルのピースのように、最初から決まっているわけではありません。部品そのものも、手探りで作っていく感じです。また、ジグソーパズルでは、最終形の写真があることが普通です。しかし、ソフトウェアの開発では、最終形が具体的に見えてくるのは開発の終盤です。

もちろん、最初の段階で、ある程度の最終形を想定してから開発を進めます。しかし、実際には、開発の途中で発見されるさまざまな事象により、最終形が当初の想定とは異なる姿になるほうが普通でしょう。

全体像と部分をいったりきたりしながら積極的に変更する

開発の早い段階で全体構造を決めてから段階的に詳細化する機能分割のようなアプローチはオブジェクト指向らしくないアプローチです。

オブジェクト指向でも、全体のイメージを想定はしますが、単純で明確な部品を発見し、それを早い段階から実装することを重視します。

この部品重視のやり方は、オブジェクト指向が「変更コスト」を下げることを重視した技術であることと深く関係しています。

開発は終盤になるほど変更コストが高くなる、という従来のアプローチだと、変更を避けるために、可能な限り設計を前倒しして精緻に決めようとします。

ソフトウェアのコード量が増えても「変更を楽に安全」にできる、というオブジェクト指向の発想だと、開発の前半で設計に時間をかけることは費用対効果が悪いと考えます。

あまり細かいことを把握できていない初期の段階では、ざっくりとした全体像を掴む程度にし、部品を見つけながら、ジグソーパズルのように完成に向かって、部分部分を成長させていく、という発想です。

部分を作っていくと、全体に対する知見も増えます。その知見を活かして全体像も見直しながら開発を進めます。

このようなインクリメンタルな開発という、アジャイル手法の考え方の根底にあるのは、オブジェクト指向技術で「変更コスト」を下げることができる、ソフトウェアの規模が大きくなっても、変更を楽に安全にできるように設計できる、という考え方なのです。

また「リファクタリング」も、オブジェクト指向の「変更を楽に安全にする」という発想から生まれています。

実際に動いてから発見したことを反映してソフトウェアを変更するほうが、良い設計に早くたどり着ける、という発想です。

また「リファクタリング」を実践することで、変更コストの高い設計を、徐々に改善して、トータルの変更コストを下げる、という発想です。

全体は「ツリー」ではなく「ネットワーク」

オブジェクト指向ではソフトウェアの全体構造を「ツリー」構造とはとらえません。

さまざまなドメインオブジェクトがネットワーク状に連携することで、必要な機能を提供する、ネットワーク構造としてソフトウェアの構造を設計します。

機能分解からトップダウン式で機能を詳細化するアプローチは、要求の整理の手法としては有効です。

しかし、その機能分解の構造を、そのままプログラムの構造に反映すると、変更コストが高くなります。詳細機能レベルでは同じデータを使うことが多いので、あちこちの詳細機能のプログラムに、同じデータを操作する似たようなコードが重複します。その結果、変更のためにあちこちを調べ、広い範囲のコードを書き換え、副作用がないことを検証するために大量のテストが必要になります。

この問題の解決策の1つがオブジェクト指向であることはこの記事で説明してきたとおりです。 機能の段階的詳細化ではなく、ソフトウェアを利用する人たちの関心事、ソフトウェアに対する要求を説明する「言葉」に注目して、それを、そのまま「ドメインオブジェクト」としてプログラム単位とするのがオブジェクト指向のアプローチです。

そして、言葉の組み合わせ方でさまざまな文章を生みだせるように、「ドメインオブジェクト」の組み合わせ方、オブジェクトのネットワーク構造で、さまざま機能を実現し、組み合わせ方によって、機能の変更や拡張に柔軟に対応するのがオブジェクト指向らしい設計の考え方です。

クラス図もネットワークを意識して描く

部品を発見し、部品の組み合わせ方を考える手段の1つがクラス図です。

UML(統一モデリングン言語)は、クラス図を始め、さまざまなモデリングや設計のため図法が標準化されています。

オブジェクト指向の設計に、すべてのUML仕様を理解し、使いこなす必要はありません(というか、最近のUMLは複雑すぎて、そもそも全体を習得するのは無理です)。

しかし、標準化された記法は、共通の言語として、意図を正しく伝達しあうためにはたいへん便利です。UMLの基本的な記法、とくにクラス図の基本事項はチーム内の共通理解にしておくことは大切です。

クラス図の書き方の基本は、ジグソーパズルと一緒です。部品となりそうなドメインオ

ブジェクトの候補クラスを見つけながら配置していきます。

クラスの候補は、要件定義書／ユースケース／画面仕様書／データ仕様書などに登場する言葉と言い回しが良い手掛かりになります。

部品が見つかるたびに、配置を変更したり、部品と部品の関係を見直しながら、クラス図上で、ざっくりと全体を組み立ててみます。

そして役に立ちそうな部品、使うことがほぼ確実な部品を見つけたら、実際にプログラミングして設計の妥当性を検証します。プログラミングの過程で、新たな部品や関係を発見できることもよくあります。

そうやって、クラス図(やパッケージ図)で全体像をとらえながら、実際のコードで部分を作っていきます。クラス図の全体像とコードの実装をいったりきたりしながら、全体も部分も段階的に完成させていくのがオブジェクト指向らしい開発のやり方です。

なお、クラス図に詳細なクラス定義をすべて描くのは費用対効果の悪いやり方です。

全体像がわかることは大切ですが、詳細はコードで確認すれば十分だし、そのほうが正確です。

クラス図とER図

オブジェクト指向の設計に使うクラス図とテーブルの設計に使うER(Entity Relation)図は、どちらも箱と矢印を使った似た記法です。

しかし、その意味は大きく異なります。とくに矢印の意味が異なります。ER図では、矢印は「キー」に注目した、テーブルとテーブルの(データとデータの)関係を示します。

クラス図の矢印には、そういう意味はありません。

クラス図の矢印は、サービスを依頼するクラス(クライアント)と、サービスを提供するクラス(サーバ)とのクラス間の「協力関係」を示しています。

ですので、それぞれのクラスが保持するデータの間には、何も関係がなくても、あるクラスと別のクラスがクライアント／サーバの関係になることができます。

クラス図の矢印は、矢印の先にあるクラスに対して、「そのクラスに仕事を依頼する」とか「そのクラスの持つサービスを使う」という意味です。

全体の組み立て方

オブジェクト指向は、このクラスのクライアント／サーバの関係で全体を組み立てます。このネットワーク上に組み立てるやり方が、オブジェクト指向でプログラミングした場合に、変更が楽で安全になる理由の1つです。

機能に追加が必要になったときは、サーバ役のクラスを追加する、あるいはサーバ役のクラスだけを取り替えればが良いわけです。

またクライアントとサーバの関係は、実行時に動的に切り替えることができるのがオブジェクト指向の特徴です。実行時に、その状況に最適な仕事の依頼先を自動的に見つけて、仕事を依頼するスタイルで、全体を組み立てておくわけです。

演習問題

1つ演習問題をやってみましょう。

問題

ある美術館の入場料金は**表2**のとおりです。

この料金表をコンソール(Sytem.out)に表示するプログラムを書いてください。

▼表2　入場料金表

高齢者	-(無料)
大人	2,000円(通常料金)
子供	1,000円(半額)

ヒント

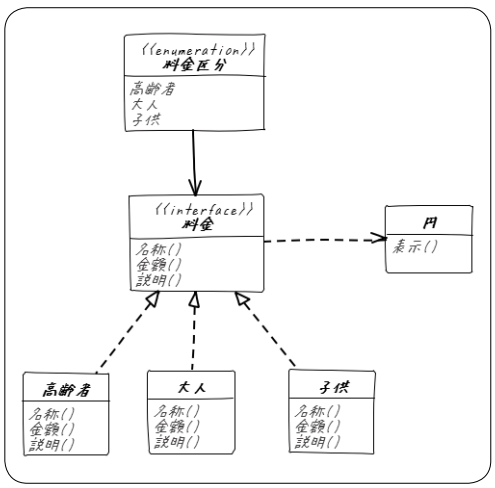
(1)「列挙型(enum)」と「多態」を活用してください。
(2)「列挙型」のすべての要素は、**values()**メソッドで取得できます。
(3)高齢者の金額表示**("-")**はガード節で出し分けてください。

解答例

図3にクラス図を示します。サンプルプログラムは**リスト12**～**リスト15**です。

解説

▼図3　クラスのラフスケッチ



▼リスト12　表示プログラム

```
public class Client {
    public static void main(String[] args) {
        for(ChargeType type : ChargeType.values() ) {
            System.out.println( type );
        }
    }
}
```

▼リスト13　料金タイプ(列挙型)

```
public enum ChargeType implements Charge {
    silver(new Silver()),
    adult(new Adult()),
    child(new Child());

    Charge charge ;
```

▼リスト13　料金タイプ(列挙型)　(続き)

```
    private ChargeType(Charge charge) {
        this.charge = charge;
    }

    @Override
    public String label() {
         return charge.label();
   }

    @Override
    public Yen amount() {
        return charge.amount();
    }

    @Override
    public String description() {
        return charge.description();
    }
    @Override
    public String toString() {
        return String.format("%s : %s (%s)", label(), amount().asText(), description());
    }
}
```

▼リスト14　料金のインターフェース宣言と実装クラス

```
interface Charge {
    String label();
    Yen amount();
    String description();
}

public class Silver implements Charge {
    @Override
    public String label() {
        return "高齢者";
    }

    @Override
    public Yen amount() {
        return new Yen(0);
    }

    @Override
    public String description() {
        return "無料";
    }
}

public class Adult implements Charge {
    @Override
    public String label() {
        return "大人";
    }

    @Override
    public Yen amount() {
        return new Yen(2000);
```

次ページに続く

▼リスト14　料金のインターフェース宣言と実装クラス（続き）

```
    }

    @Override
    public String description() {
        return "通常料金";
    }
}

public class Child implements Charge {

    @Override
    public String label() {
        return "子供";
    }

    @Override
    public Yen amount() {
        return new Yen(1000);
    }

    @Override
    public String description() {
        return "半額";
    }
}
```

▼リスト15　金額

```
public class Yen {
    int value;

    Yen(int value) {
        this.value = value;
    }

    String asText() {
        if(value == 0) return "-";
        return String.format("%d円 ↵
",value);
    }
}
```

このように設計すると、たとえば、「女性料金」や「幼児料金」を追加するときに、プログラムの変更が楽で安全になります。

・メインの表示プログラムの変更は不要
・既存の料金やその説明を記述したクラスは変更の対象外（独立性が高いのでほかの料金の追加をしても影響しない）

まとめ

オブジェクト指向の基本は次のとおりです。

・データとロジックをまとめる（合わせて考える）
・人間の関心事に沿って部品化する（クラス名やパッケージ名を業務の言葉と対応させる）
・部分を発見し、固めながら、ボトムアップでネットワーク状に全体を組み立てていく

こうすることで、以下が実現できます。

・どこに何が書いてあるか特定しやすくする
・1つの関心事にかかわるコードを1個所に集める（プログラムのあちこちに分散させない）
・機能の変更があったときに、ごく一部の部品の変更や、既存部品の組み合わせ方で変更できる
・部品の用途を限定し、独立性を高め、変更の影響範囲を狭く閉じ込めやすくする

こうすることでソフトウェアの変更が楽で安全になります。

オブジェクト指向らしくない書き方は次のとおりです。

・データクラスと機能クラスを分ける
・同じ処理をあちこちに重複して書く
・（独自のクラスは積極的に作らないで）プログラミング言語の基本データ型とその操作だけでプログラミングする
・場合分けの構造を、if/else文やswitch文の複文構造で作り込む

このようなオブジェクト指向らしくない書き方と比べると、オブジェクト指向が、変更が楽で安全になる理由が実感できるはずです。SD

Column

オブジェクト指向は総合格闘技

ソフトウェア開発は複雑さとの戦いです。その複雑さと戦うために、さまざまなアイデアや技法が生まれました。

オブジェクト指向もソフトウェアの複雑さと戦うための工夫の1つです。

しかし、オブジェクト指向を一言で説明し理解するのが難しいことは、みなさんがご存じのとおりです。本を読んでも、ネット上の情報を調べても、すっきりと理解できません。

それは、オブジェクト指向が、1つの単純な原理をもとに発展させた考え方ではなく、次のような、さまざまなアイデアや技法を組み合わせたものだからです。

・構造化プログラミング
・抽象データ型
・モジュラープログラミング
・動的束縛
・メッセージによる協調動作
……

オブジェクト指向は、これらの考え方の、よく言えば「集大成」です。悪く言えば「ごった煮」です。ソフトウェアの複雑さと戦うためのさまざまな技法から良いところ取り込み組み合わせた総合格闘技がオブジェクト指向なのです。そのため、何をもってオブジェクト指向とするかが、あいまいで、定義がしにくいのです。

オブジェクト指向をわかりにくくしている、もう1つの理由が、プログラミング言語です。

たとえば、Javaはオブジェクト指向の言語と言われます。しかし、Javaは「オブジェクト指向」以外にも、「手続き型」や「関数型」のスタイルでも記述できる、なんでもありの言語です。Javaを使ってプログラミングすれば、オブジェクト指向のプログラミングになるわけではありません。

Javaは、C言語からの移行しやすさを重視して開発された経緯もあり、C言語そのままの「手続き型のプログラミング」ができる言語です。int/float/charなどのプリミティブなデータ型、固定サイズの配列宣言、++や+=の演算子、staticメソッド、……。これらは、オブジェクト指向プログラミングの道具ではなく、手続き型プログラミングの道具です。

また、Java 8で関数型プログラミングの考え方を取り入れたStream APIが追加され、関数型に近いプログラミングもできるようになりました。

つまり、Javaを使ってプログラミングをするといっても、オブジェクト指向／手続き型／関数型のスタイルが混在できるわけです。

筆者は、オブジェクト指向のスタイルを重視していますので、次のような方針で、Javaを使っています。

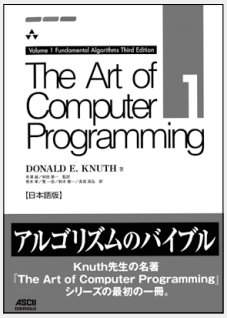
・int/float/charなどプリミティブなデータ型は使わない
・++や+=など同じ変数を書き換える演算子を使わない
・staticメソッドはほんとうに必要な場所以外では使わない
・Stream APIを使う場所は「ファーストクラスコレクション」の内部に限定する
・オブジェクトはできるだけ不変にする

この記事で説明した、ソフトウェアの変更を楽に安全にする方法をJavaで実践するには、このような方針が良いのではないかと、考えています。

最後に参考としてオブジェクト指向設計を学ぶための推薦図書を挙げておきます。

●『新装版リファクタリング』(オーム社、2014)
この本の設計改善テクニックを覚えるとオブジェクト指向の設計力が自然と身につきます。第1章「最初の例」と第3章「コードの不吉な臭い」から読み始めてみるのが良いでしょう。

●『エリック・エヴァンスのドメイン駆動設計』(翔泳社、2011)
オブジェクト指向設計の考え方を深く学ぶための名著です。難解ですが、何度も読み直す価値があります。「まえがき」をじっくり読んで理解するだけでも、学びがたくさんあります。



The Art of Computer Programming Volume 1

Donald E. Knuth 著／有澤 誠、和田 英一 監訳／青木 孝、筧 一彦、鈴木 健一、長尾 高弘 訳
B5判／688ページ
4,800円＋税
ドワンゴ
ISBN＝978-4-04-869402-5

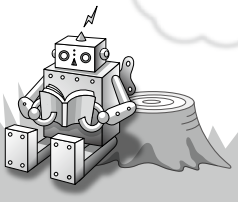
計算機科学者ドナルド・クヌースによる、「アルゴリズムの解析」についての一連のシリーズ第1巻である。各アルゴリズムについて、アセンブリ言語、数式、UMLなどを使ってその原理・考え方を徹底的に解説している。本書は600ページにも及ぶ長編だが、章立ては「基礎概念」「情報構造」の2章のみ。前者では帰納法といった数学の基礎知識からサブルーティンといった基本的なプログラミング技法について、後者では線形リスト・木構造について、それぞれ大量の演習問題を載せながら解説している。演習問題にはそれぞれ00～50のレートが振られているので、挑戦する前に難易度を把握できる。ちなみに、「00」は頭の中でただちに解ける問題、「50」は執筆時点でまだ満足のいく解が読者から得られていない研究問題とのことだ。

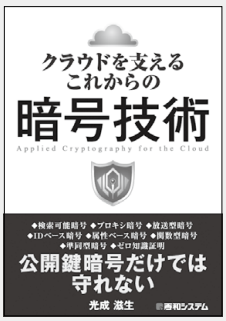
できるPRO Red Hat Enterprise Linux 7

平 初、できるシリーズ編集部 著
B5判／344ページ
3,000円＋税
インプレス
ISBN＝978-4-8443-3839-0

新人のサーバ管理者向けに書かれた、Red Hat Enterprise Linux（RHEL）7によるサーバ構築の解説書。Web、FTP、ファイル、DHCP、プロキシ、DNS、メール、データベース、CMSと、企業における基幹業務のためのおもなサーバシステムの構築方法を解説している。各々のサーバの説明は、主要ソフトのインストールと設定方法、運用においてのキーポイントに留められてるので、実運用やトラブルシューティングについてはそれぞれ別の専門書をあたるといいだろう。ファイアウォールについては、旧式のiptablesおよび新方式のfirewalldの設定方法を両方掲載している。firewalldには、ネットワークを抽象化して「ゾーン」に分けて管理するという新しいしくみが導入されているが、その点が重点的に解説されている。

SD BOOK REVIEW





クラウドを支えるこれからの暗号技術

光成 滋生 著
A5判／240ページ
2,800円＋税
秀和システム
ISBN＝978-4-7980-4413-2

電子商取引やクラウドサービスで必須の暗号技術について解説する本。共通鍵暗号、公開鍵暗号などの現在よく使われている暗号技術や、2000年以降に登場した新しい暗号技術を取り上げる。新しい暗号とは、暗号化したまま検索したり計算したりできる暗号、多数の人に一度に効率よく暗号化されたコンテンツを配信するための暗号などを指す。本書は数式を使った説明も多いので、内容を理解するには高校～大学の数学知識がないと厳しいかもしれない。ただ、文章と図からだけでも、その暗号は「どんな条件のもとで安全性が保たれるのか」「どれほど破られにくいのか」「弱点は何か」ということが把握できる。昨今、暗号ソフトの脆弱性が発見されることがたまにあるが、そのような事象を読み解く際にも、本書の知識は役に立つだろう。

Dockerエキスパート養成読本

杉山 貴章、大瀧 隆太、Yugui（Yuki Sonoda）、中津川 篤司、前佛 雅人、松原 豊、米林 正明、松本 勇気 著
B5判／112ページ
1,980円＋税
技術評論社
ISBN＝978-4-7741-7441-9

2013年の登場以来、ITインフラ技術として注目が高まっている「Docker」。本書の前半では、Dockerの概要やそれを取り巻く最新動向、Docker Engine、Doclerfile、管理ツール「Kubernetes」の入門など、これから実際に試してみようという人に向けた解説がまとめられている。後半では「実践編」として、Dockerによる環境構築の自動化、CIツール活用、そしてニュースアプリを展開する「Gunosy」でDockerを実戦投入・検証した様子の記録など、Dockerに取り組みたい人が今知っておきたい解説が盛りだくさん。国内ではまだ開発環境での利用例が多く、本格的な導入はこれから広まっていくと思われるコンテナ型仮想化技術について、まさに今押さえておきたい知識が収められた1冊であるだろう。

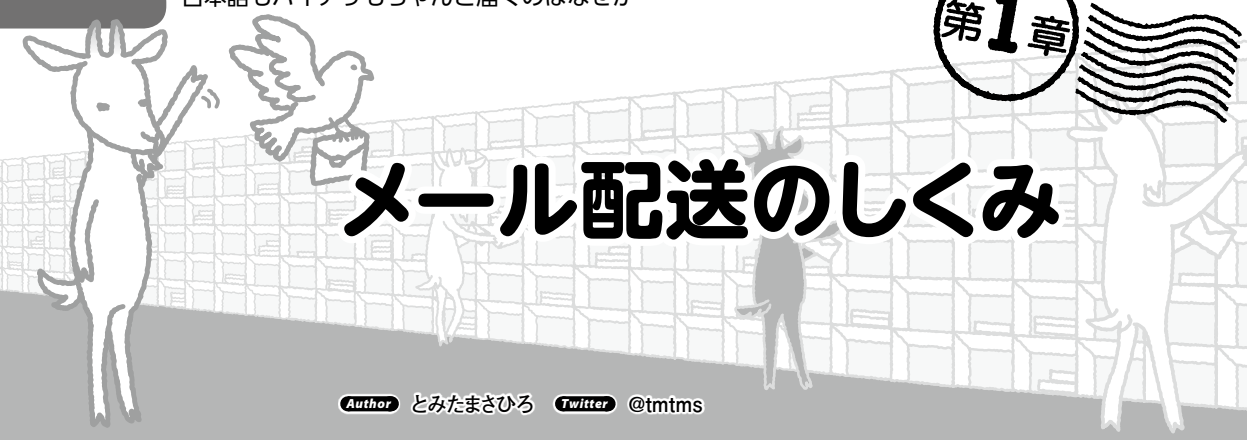
文字化けやスパムの
原因がわかった！

第2特集

メールシステムの教科書

日本語もバイナリもちゃんと届くのはなぜか

日々当たり前に使っているメールも、その裏側はもちろんサーバ、プロトコルが関与するネットワークの世界です。メールクライアントソフトの送受信ボタンを押したとき、その裏側でサーバはどんなプロトコルでやりとりをして、自分のメールを回収し、目的のサーバに送り届けているのか。一方で届いたデータに目を移せば、本来ASCII文字しか扱えないメールシステムに、どうやって日本語やバイナリを含ませているのか。そして、情報漏えいや外部攻撃の火種となるメールに対するセキュリティはどうなっているのか。本特集でその基本をしっかりと整理しておきましょう。



第1章

メール配送のしくみ

Author とみたまさひろ Twitter @tmtms

当たり前に使っているメールですが、それがどのようにして相手まで届くのか、正しく説明できますか？ SMTP、POP、IMAPなどのプロトコルの役割をきちんと理解していますか？ まずはそれらの基本を整理しましょう。

メールが届くまで

英語の「mail」は日本語で言うと郵便や郵便物という意味で、電子メールのことは「email」と言います。しかし、日本語で「メール」と言うと電子メールを意味することが多いでしょう。本記事でも以降では「メール」と表記します。

紙の郵便とメールは当然異なるものですが、配送のしくみはかなり似ています。メールを書いてからメールが届くまで、どのようなしくみでメールが送られているか説明します。

メールアドレス

メールアドレスはhoge@example.comの形式です。「@」はアットマークという記号ですが、まさにatの意味です。hoge@example.comはhoge at example.comで、example.comの場所のhogeさんということを表しています。郵便で言うと、example.comが住所でhogeが名前に相当します。

メールを出す

メールの送信は、郵便の場合で言うと手紙を書いて封筒に入れ、封筒に宛先と差出人を書いてポストに投函するまでに相当します。

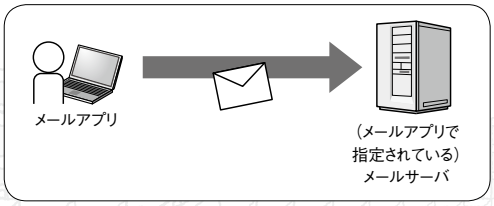
メールアプリで宛先、件名、本文を入力し、もし必要ならば添付ファイルを追加して送信ボタンを押します。ここから先の処理は利用者が意識することはありません。

メールアプリで送信ボタンを押すと、宛先、件名、本文、添付ファイルなどから1つのメッセージ形式に組み立てられます。このメッセージが封筒の中の手紙に相当します。メッセージの形式については第2章で説明します。

メールはメールアプリに設定されたメールサーバに送られます(**図1**)。一般家庭では、契約しているインターネット接続プロバイダのメールサーバを使用することが多いでしょう。契約者以外に使用されないように、IDとパスワードによる認証が求められることが一般的です。大きな組織であれば、組織内に専用のメールサーバが用意されていることもあると思います。

メールアプリはメールサーバに送信者メールアドレスと受信者メールアドレスを伝え、メッセージを送ります。この送信者メールアドレスと受信者メールアドレスを「エンベロープ送信者」、「エンベロープ受信者」と言います。メールアプリを使用した場合は、通常はアプリに登録済みの送信者メールアドレスと入力された宛先

▼図1 メールの送信



メールアドレスが使用されますが、プロトコル上は異なるメールアドレスを使用することもできます(詳細は第2章、第4章で解説します)。

エンベロープとは封筒(envelope)のことです。エンベロープ送信者／受信者は、郵便の場合の封筒の差出人と宛先に相当します。

メールの送信に使用されるプロトコルはSMTP(Simple Mail Transfer Protocol)です。TCP/IPポート番号は通常は587(submission)です。

メールを配送する

メールの配送は、郵便の場合で言うと、ポストから郵便物を回収し宛先の家まで配達するところまでに相当します。

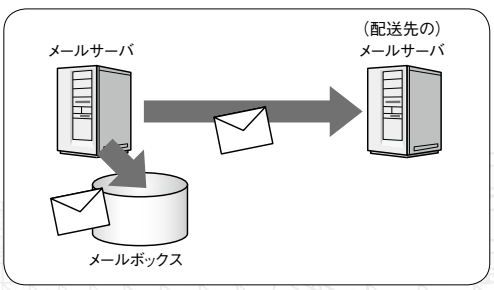
メールを受け取ったサーバは、エンベロープ受信者のメールアドレスが、自分が処理すべきものであれば、アドレスが示すユーザのメールボックスにメールを格納します。そうでなければ、宛先のアドレスごとに配送先のメールサーバにメールを送信します(**図2**)。メールの配送先はアドレスのドメイン部(@の右側)を使ってDNSのMXレコード[注1]から求めます。

このメールを受け付けて配送するプログラムをMTA(Mail Transfer Agent)と言います。オープンソースの代表的なMTAとしてはPostfix、Sendmail、qmailがあります。

メール配送に使われるプロトコルもSMTPですが、TCP/IPポート番号は25(smtp)です。

注1) 電子メールの配送先を決定する際に使用する情報。ドメインごとに配送先となるメールサーバのホスト名などが登録されている。

▼図2　メールの配送



MTAはエンベロープ受信者アドレスに従ってメールを配送するだけで、基本的にはメールのメッセージの内容については関与しません。ただし例外がいくつかあります。

- 自分がメールを中継したことをメッセージに記録する(Received)
- 足りない情報を補完する(Date、From、Message-Idなど)
- プロトコル上不正となる形式を整形する(8bitコードの扱いや長い行の折り返しなど)
- メッセージにエンベロープ送信者と受信者を記録する(Return-Path、Delivered-To)

これらの処理が行われるかどうかはMTAに依存します。

インターネットからメールを受け付けたあと、ウィルススキャンやスパムチェックを行うサーバを経由し、ユーザのメールボックスを持つサーバに配送するといったように、組織内で複数のメールサーバを経由することも一般的です。

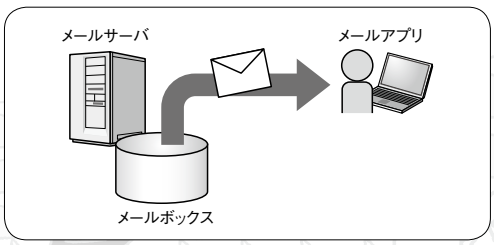
メールを受け取る

メールの受信は、郵便の場合、自宅ポストに配達された郵便物を取り出す行為に相当します。

メールボックスに配送されたメールはメールアプリによって読み出されます(**図3**)。メールアプリはメールメッセージ形式から送信者、受信者、件名、本文などを取り出して表示します。

プロトコルはPOP(Post Office Protocol)やIMAP(Internet Message Access Protocol)が使用されます。POPはメールボックスからメールを取り出すこととメールを削除することくら

▼図3　メールの受信



いしかできませんが、IMAPはメールの取り出し、削除のほかに検索や格納もできます。複数のメールボックスをフォルダとして扱うこともできます。

複数宛先

同じメールをCcやBccで複数の宛先に送った場合、メールの配送経路上ではできるだけ複数宛先を保ったまま送信されます(**図4**)。

配送先が異なる場合は配送先ごとにまとめて送信されますが、MTAによっては配送先が同一であってもまとめずにメールアドレスごとに送信するものもあります。たとえばPostfixは配送先ごとにまとめますが、qmailはまとめずにメールアドレスごとに送信します。

また配送先ごとにまとめる場合であっても、1つのメールに指定できる宛先メールアドレス数には上限があります。SMTPのことを規定しているRFC 5321には、最低でも100個は受け付けなくてはいけないという規定がありますが、上限はとくに定められていません。Postfixのデフォルトでは、1,000個までの宛先を受け付けますが、送信時の宛先数は最大50個です。

配送エラー

配送先メールサーバが応答しない、宛先のメールアドレスが存在しない、宛先のメールボックスが満杯で受け付けられないなどの理由により、配送途中でエラーになることがあります。

メールの配送エラーもメールとして元のメールのエンベロープ送信者に送られます(**図5**)。エラーメールもメールですので、通常のメールの配送と同じ経路で送られます。エラーメールのメッセージの形式はRFC 3461で規定されていますが、比較的新しい規格のため、これに従っていないMTAもあります。

不正中継防止

メールサーバはどこから送られた誰宛のメールでも中継していいというわけではありません。信頼できないクライアントから外部の宛先のメールを受け付けて送信してしまうと、スパマーによって利用され、スパム中継サーバとなってしまいます。メールを受け付ける際には最低限次のような制約を設けるべきです(**図6**)。

メール送信サーバの場合

・信頼できるクライアントからの接続であれば受け付ける
・そうでなければ受け付けない

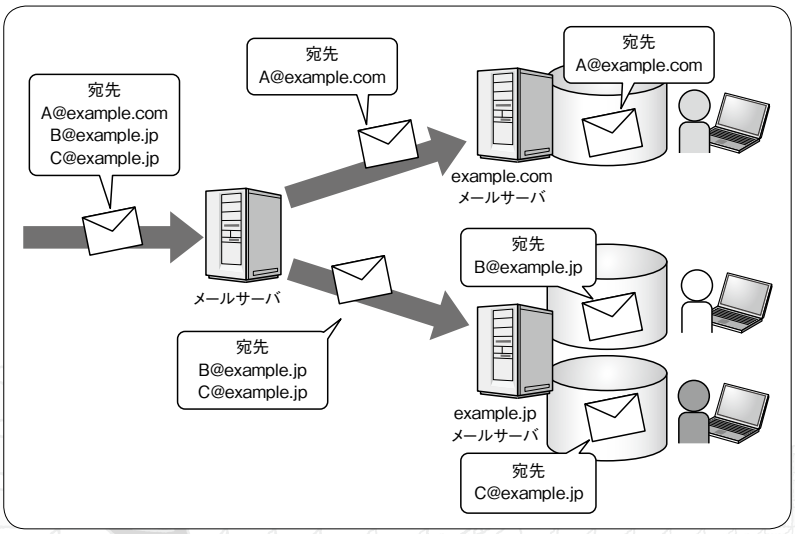
メール受信サーバの場合(DNSのMXレコードに登録されているサーバなど)

・自分が管理しているドメイン宛のメールであれば受け付ける
・そうでなければ受け付けない

送信と受信を兼ねているサーバの場合

・信頼できるクライアントからの接続であれば

▼図4　複数宛先への送信

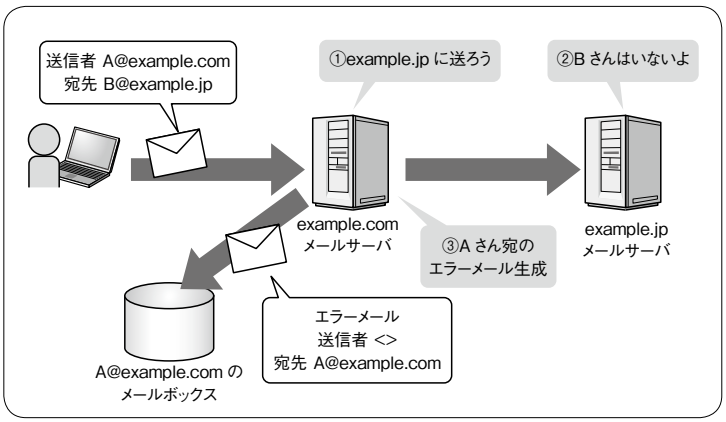


受け付ける
・自分が管理しているドメイン宛のメールであれば受け付ける
・そうでなければ受け付けない

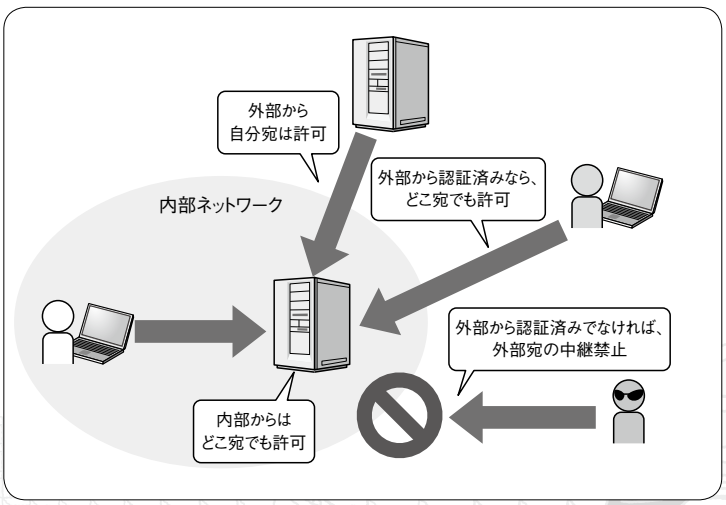
「信頼できるクライアント」とは、信頼できるIPアドレスのクライアント、または認証を通ったクライアントのことです。認証については第4章で詳しく説明します。

Postfixのデフォルト設定では、自分のサーバから発信されるメールは宛先がどこであっても受け付け、それ以外のクライアントからは自分のドメイン宛のメールしか受け付けないようになっているため安全です。

▼図5　配送エラー



▼図6　メールを受け付ける際の制約



メールに使われるプロトコル

前述したように、メールはDNS、SMTP、POP、IMAPというプロトコルが使用されます。それぞれについて詳しく見ていきます。

DNS

DNS（Domain Name System）はホスト名に対応するIPアドレスを得るためのプロトコルです。DNSはメールのためだけに使用されるものではありません。たとえば、ブラウザでインターネットのコンテンツを表示する際にも使用されています。

ですが、DNSにはメール配送専用の機能があります。MXレコードはメール配送のためだけに使用されます。メールは、宛先アドレスのドメイン部をDNSで検索し、そのMXレコードの値が示すサーバに送られます。

DNSはバイナリベースのプロトコルですので、人間が直接使うことはできません。そのため、配送先サーバなどの情報は次のようにdigコマンドなどを用いて調べます。

```
$ dig +short mx example.com
10 mx1.example.com
20 mx2.example.com
```

これはexample.comのメールの配送先はmx1.example.com、mx2.example.comの2つのサーバであることを表しています。数字は優先度を表し、数字が小さいほうが優先度が高くなります。この例ではmx1

のほうがmx2よりも優先度が高いことを表しています。

example.comドメイン宛にメールを配送するシステムは、まずmx1.example.comに配送を試み、失敗するとmx2.example.comに配送を試みます。

SMTP

SMTP(Simple Mail Transfer Protocol)はメールを送信するためのプロトコルでRFC 5321で規定されています。

SMTPはテキストベースのプロトコルですので、Telnetなどを使えば人間でもサーバとやりとりすることができます。コマンドや応答は改行で終了します。改行はCRLF(0x0D 0x0A)の2バイトが使用されます。

クライアントからコマンドを発行し、サーバがそれに対する応答を返します。サーバからの応答は、「3桁の数字」+「空白」+「文字列」の形式です。3桁の数字は先頭の1文字で**表1**の意味を表します。

応答の例
```
250 2.1.0 Ok
```

複数行の応答が返る場合は、最後の行以外は「3桁の数字」+「-」+「文字列」の形式で、最後の行は「3桁の数字」+「空白」+「文字列」の形式となります。

応答(複数行)の例
```
250-server.example.com
250-PIPELINING
250-SIZE 10240000
250-ETRN
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
```

▼表1　SMTPの応答コード

数字	意味
2xx	成功
3xx	成功(途中)
4xx	一時的な失敗
5xx	永続的な失敗

SMTPの基本的なコマンドはEHLO、HELO、MAIL、RCPT、DATA、RSET、VRFY、NOOP、QUIT、VRFYです。以降で各コマンドについて説明します(NOOP、VRFYについてはあまり使われないため割愛します)。

接続してから切断するまでのフローを**図7**に示します。

接続

MTAのポート(25か587)に接続すると、サーバから220応答が返ります。

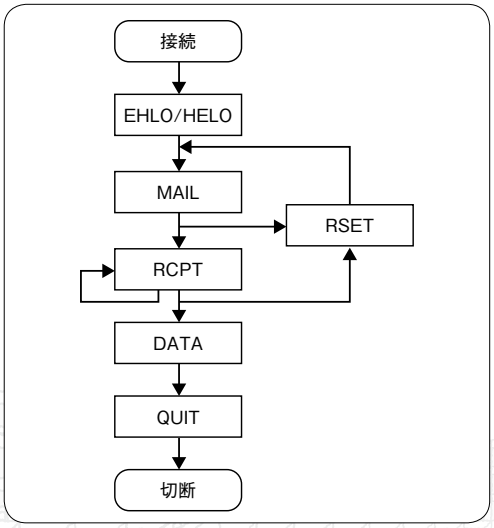
```
220 server.example.com ESMTP Postfix (Ubuntu)
```

応答の最初の文字列はサーバのホスト名で、それ以降はコメントです。

挨拶

接続後、クライアントからEHLO(Extended HELLO)コマンドを発行します。パラメータはクライアントのホスト名です。サーバの応答は通常は複数行形式です。

▼図7　SMTPの接続から切断までのフロー



```
EHLO client.example.com
250-server.example.com ←ホスト名
250-PIPELINING ←この行以降は拡張機能
250-SIZE 10240000
250-ETRN
250-STARTTLS
250-AUTH DIGEST-MD5 CRAM-MD5 NTLM LOGIN
PLAIN
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
```

応答の最初の値はサーバのホスト名、それ以降はサーバで有効な拡張機能を表します。各拡張機能はRFC 5321以外のRFCで規定されています。上の例の応答の意味を**表2**に示します。

EHLOがエラーになる場合は、拡張機能をサポートしないサーバです。EHLOがエラーになったらクライアントはHELOを使用します。HELOに対するサーバの応答は1行だけです。

```
HELO client.example.com
250 server.example.com
```

また、クライアントがEHLOではなくHELOを送信すると、サーバはクライアントが拡張機能をサポートしていないものとみなして処理します。

■ 送信者アドレス指定

MAILコマンドでメールのエンベロープ送信者アドレスを指定します。

```
MAIL FROM:<sender@example.com>
250 2.1.0 Ok
```

エラーメールなどのMTAからの通知メッセージの場合は空アドレスが指定されます。

```
MAIL FROM:<>
250 2.1.0 Ok
```

SIZE拡張が有効な場合は、これから送ろうとするメッセージの大きさをオプションで指定できます。

```
MAIL FROM:<sender@example.com> SIZE=5432
250 2.1.0 Ok
```

■ 受信者アドレス指定

RCPTコマンドでメールのエンベロープ受信者アドレスを指定します。

```
RCPT TO:<rcpt@example.com>
250 2.1.5 Ok
```

宛先が複数の場合は、RCPTコマンドを複数回発行します。なお、SMTPでは、To、Cc、Bccの区別はありません。

■ メッセージ送信

クライアントからDATAコマンドが送られると、サーバは354応答を返します。クライアントはサーバからの354応答を待ってからメッセージを送信し、最後に「.」だけの行を送ります。

```
DATA
354 End data with <CR><LF>.<CR><LF>
From: sender@example.com
To: rcpt@example.com
Subject: test
```

次ページに続く

▼表2　SMTPの拡張機能

拡張	RFC	内容
PIPELINING	RFC 2920	クライアントはサーバの応答を待たずに次のコマンドを送って良い
SIZE	RFC 1870	サーバが受け取れるメッセージサイズ、クライアントが送信するメッセージサイズを事前に通知
ETRN	RFC 1985	ETRNコマンドが有効
STARTTLS	RFC 3207	STARTTLSコマンドが有効
AUTH	RFC 4954	AUTHコマンドが有効
ENHANCEDSTATUSCODES	RFC 2034	拡張ステータスコード(RFC 3463)を返す
8BITMIME	RFC 6152	8bitメッセージを送信可能
DSN	RFC 3461	エラーメール(DSN)の詳細を指定可能

```
test
.
250 2.0.0 Ok: queued as 8FFBD6009D
```

メッセージ中に「.」で始まる行がある場合は、クライアントはその行の先頭に「.」を追加して送信する必要があります。サーバは行の先頭の「.」を無視します。

DATAコマンド発行後は接続を切断する以外にメールの送信を取りやめることはできません。

リセット

RSETコマンドを発行すると、それまでに指定したエンベロープ送信者アドレス、エンベロープ受信者アドレスをクリアし、EHLOが発行された直後の状態に戻します。

```
RSET
250 2.0.0 Ok
```

終了

QUITコマンドで接続を切断します。

```
QUIT
221 2.0.0 Bye
```

POP

POP(Post Office Protocol)はメールボックスからメッセージを取り出すプロトコルで、RFC 1939で規定されています。また、RFC 2449拡張機能をサポートしているPOPサーバもあります。TCP/IPのポート番号は110(pop3)です。

POPもSMTPと同じくテキストベースのプロトコルです。SMTPと同様に、コマンドや応答は改行で終了し、改行はCRLFです。

サーバからの応答は正常の場合は「+OK」から始まり、エラーの場合は「-ERR」から始まります。コマンドによっては応答が複数行になります。複数行の応答は「.」だけの行で終了します。複数行の応答の中に「.」で始まる行がある場合は、サーバはその前にさらに「.」を追加します。クライアントは「.」だけの行は応答の終了とみなしますが、後ろに何か文字が続く場合は先頭の「.」を無視します。

POPの基本的なコマンドは、USER、PASS、APOP、STAT、LIST、UIDL、RETR、TOP、DELE、NOOP、RSET、QUITです。拡張機能が有効な場合はCAPAコマンドも有効ですが、今回は拡張機能については割愛します。

接続

POPのポート(110)に接続するとサーバから応答が返されます(**図8**)。応答に< >でくくられた文字列がある場合は、サーバがAPOP認証(後述)に対応していることを表しています。

認証

USERコマンドとPASSコマンドで認証を行います。USERコマンドの引数はユーザ名、PASSコマンドの引数は平文のパスワードです。

```
USER hoge
+OK
PASS P@ssWord
+OK Logged in.
```

USER/PASSは平文認証ですので、通信経路が暗号化されていない場合はパスワードが盗聴される可能性があります。

APOPは直接パスワードを送信するのではなく、パスワードのMD5ハッシュ値を送信する認証方式です。引数はユーザ名とパスワードのハッシュ値です(**図9**)。APOPは、接続時にサーバから返される文字列に< >でくくられた文字列が含まれている場合に使用可能です。

ハッシュ値は、接続時の応答の< >でくくられた文字列(<と>も含みます)とパスワードを結合した文字列のMD5 digest値を16進数で表し

▼図8　POPのポートに接続した際のサーバからの応答

```
+OK Dovecot (Ubuntu) ready. <4392.3.55a3c764.WJKDtpgMuwPTSOM4r5uvmA==@pop.example.com>
```

たものです。Rubyでは**リスト1**のようにして求めることができます。

APOPではサーバもクライアントと同様にハッシュ値を求め、クライアントからのハッシュ値と同じ値であれば認証が成功します。そのためサーバでパスワードを平文(または復号可能な暗号文)で保持している必要があります。

■メッセージサイズ

LISTコマンドはメッセージのサイズを返します。引数はメッセージ番号です。メッセージ番号はログイン時に確定し、1から順番に番号が振られます。

```
LIST 2
+OK 2 5043
```

引数を指定しない場合は、「+OK」のあとに全メッセージのサイズを複数行で返します。

```
LIST
+OK scan listing follows
1 2044
2 5043
  (略)
17 2399
18 2448
.
```

■メッセージUID

UIDLコマンドはメッセージのUIDを返します。メッセージ番号はログインごとに振りなおされるので番号が変わりますが、UIDは一度割り当てられるとメッセージが削除されるまで変更されません。POPクライアントはこのUIDをもとに新着メールの有無を判定します。引数はメッセージ番号です。

```
UIDL 2
+OK 2 0000000255a3daf1
```

引数を指定しない場合は、「+OK」のあとに全メッセージのUIDを複数行で返します。

```
UIDL
+OK
1 0000000155a3daf1
2 0000000255a3daf1
  (略)
17 0000001155a3daf1
18 0000001255a3daf1
.
```

■メッセージ取り出し

RETRコマンドでメッセージを取り出します。引数はメッセージ番号です。「+OK」のあとにメッセージ全体を返します。

```
RETR 2
+OK
From: fuga@example.net
To: hoge@example.com
Subject: test

line1
line2
line3
.
```

■メッセージの一部取り出し

TOPコマンドでメッセージヘッダ全体と本文の先頭を返します。大きなメッセージ全体をダウンロードせずに送信者や件名を取り出す場合に使用されます。引数はメッセージ番号と本文の行数(0以上)です。

▼図9　APOP認証の様子

```
+OK Dovecot (Ubuntu) ready. <4392.3.55a3c764.WJKDtpgMuwPTSOM4r5uvmA==@pop.example.com>
APOP hoge aa63987281ea8816fac46564912fdc90
+OK Logged in.
```

▼リスト1　RubyでAPOPのハッシュ値を求める

```
require 'digest/md5'
Digest::MD5.hexdigest("<4392.3.55a3c764.WJKDtpgMuwPTSOM4r5uvmA==@pop.example.com>P@ssWord")
#=> "aa63987281ea8816fac46564912fdc90"
```

```
TOP 2 1
+OK
From: fuga@example.net
To: hoge@example.com
Subject: test

line1
.
```

■削除

DELEコマンドでメッセージを削除します。引数はメッセージ番号です。DELEを発行しても実際にはまだ削除されていません。QUITすることで実際に削除されます。

```
DELE 2
```

▼図10　RSETで削除を取り消す様子

```
LIST 2
+OK 2 2363 ←2番のメッセージが存在している
DELE 2
+OK Marked to be deleted. ←削除成功
LIST 2
-ERR Message is deleted. ←削除済みなのでエラー
RSET
+OK
LIST 2
+OK 2 2363 ←RSET後はメッセージが復活
```

▼図11　IMAPコマンド実行例（成功例）

```
A001 CREATE hoge
A001 OK Create completed.
```

▼図12　IMAPコマンド実行例（失敗例）

```
A002 SELECT hoge
A002 NO Mailbox doesn't exist: hoge
```

▼図13　IMAPコマンド実行例（プロトコルエラー例）

```
A003 HOGE
A003 BAD Error in IMAP command HOGE: Unknown command.
```

▼図14　IMAPコマンド実行例（複数データ返却例）

```
A004 SELECT INBOX
* FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
* OK [PERMANENTFLAGS (\Answered \Flagged \Deleted \Seen \Draft \*)] Flags permitted.
* 430 EXISTS
* 0 RECENT
* OK [UNSEEN 1] First unseen.
* OK [UIDVALIDITY 1436801777] UIDs valid
* OK [UIDNEXT 431] Predicted next UID
* OK [NOMODSEQ] No permanent modsequences
A004 OK [READ-WRITE] Select completed (0.000 secs).
```

```
+OK Marked to be deleted.
```

■リセット

RSETコマンドはDELEによる削除を取り消します（**図10**）。引数はありません。

■終了

QUITコマンドで接続を切断します。引数はありません。DELE済みのメッセージがある場合は実際にメールボックスから削除します。

```
QUIT
+OK Logging out.
```

IMAP

IMAP（Internet Message Access Protocol）はメールボックスからメッセージを取り出したり、格納、検索したりできるプロトコルで、RFC 3501で規定されています。TCP/IPのポート番号は143（imap）です。

POPはメールを取り出したらサーバから削除しクライアントでメールを管理するのが一般的ですが、IMAPは基本的にサーバにすべてのメールを置いて管理します。複数のクライアントから共通のメールボックスを扱えます。また、複数のメールボックスを扱うこともできます。メールボックスはメールアプリからはフォルダとして扱われます。

クライアントから発行するコマンドは、「タグ」+「空白」+「コマンド」の形式です。タグは「(」「)」「{」「%」「*」「+」「"」「\」を除く0x21～0x7EのASCII文字で、クライアン

トが自由に決めます。長さの制限はありません。

サーバからの応答は、成功時は「タグ」+「空白」+「OK」(**図11**)、失敗時は「タグ」+「空白」+「NO」(**図12**)、プロトコルエラー時は「タグ」+「空白」+「BAD」(**図13**)です。複数のデータを返す場合は「*」で始まる行を複数返し、最後にタグ付きの応答を返します(**図14**)。

IMAPはSMTPやPOPに比べてはるかに複雑なプロトコルでコマンドも多いので、以降はよく使用されるものだけ紹介します。

■ ログイン

LOGINコマンドでログインを行います。引数はユーザ名とパスワードです。

```
A001 LOGIN hoge P@ssWord
A001 OK LOGIN Ok.
```

LOGINは平文パスワードによる認証ですので、テストなどで手動で試すには良いのですが、メールアプリからは使用されることはあまりないでしょう。通常はSASL(Simple Authentication and Security Layer)による認証が行えるAUTHENTICATEコマンドが使用されます。今回は、AUTHENTICATEコマンドは説明しません。

■ メールボックス

LISTコマンドでメールボックスを一覧します。引数は階層名とメールボックス名です。メールボックス名にはワイルドカードが使用できます。

```
A001 LIST "" "*"
* LIST (\HasChildren) "." hoge
* LIST (\HasNoChildren) "." hoge.fuga
* LIST (\HasNoChildren) "." INBOX
A001 OK List completed.
```

メールボックスは階層構造にできます。階層の区切り記号はシステム依存です。上の例では「.」が区切り記号です。

ワイルドカード「*」は階層をまたいでマッチしますが、「%」は階層をまたぎません。

```
A002 LIST "" "%"
* LIST (¥HasChildren) "." hoge
* LIST (¥HasNoChildren) "." INBOX
A OK List completed.
```

CREATEコマンドでメールボックスを作成します。引数はメールボックス名です。

```
A001 CREATE hoge
A001 OK Create completed.
```

DELETEコマンドでメールボックスを削除します。引数はメールボックス名です。

```
A001 DELETE hoge
A001 OK Delete completed.
```

RENAMEコマンドでメールボックス名を変更します。引数は元のメールボックス名と新しいメールボックス名です。

```
A001 RENAME hoge fuga
A001 OK Rename completed.
```

■ メールボックス選択

SELECTまたはEXAMINEコマンドでメールボックスを選択します(**図15、16**)。引数は

▼図15　SELECTコマンド実行例

```
A001 SELECT INBOX
* FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
* OK [PERMANENTFLAGS (\Answered \Flagged \Deleted \Seen \Draft \*)] Flags permitted.
* 430 EXISTS
* 0 RECENT
* OK [UNSEEN 1] First unseen.
* OK [UIDVALIDITY 1436801777] UIDs valid
* OK [UIDNEXT 431] Predicted next UID
* OK [NOMODSEQ] No permanent modsequences
A001 OK [READ-WRITE] Select completed (0.000 secs).
```

メールボックス名です。SELECTは読み書き用、EXAMINEは読み込み専用です。

メッセージ情報取り出し

FETCHコマンドで現在のメールボックスの中のメッセージの情報を取り出します。

FETCHは多様な引数を取り、メッセージの情報をさまざまな形式で取り出せます。具体的には、メッセージの件名の一覧や、メッセージの構造、メッセージの生データなどです。

第1引数は**表3**のような形式で、対象のメッセージを指定します。第2引数で取り出したい項目を指定します。一部を**表4**で説明します。これらを( )でくくって複数指定することもできます。

サーバからの応答は、各メッセージについて

▼表3　FETCHの第1引数（対象メッセージ）

形式	意味
3	3番めのメッセージ
3:6	3〜6番めのメッセージ
3:*	3番め〜最後のメッセージ
3,5,6	3、5、6番めのメッセージ
1:3,5:*	1〜3番めと、5番め〜最後のメッセージ
1:*	すべてのメッセージ

▼図16　EXAMINEコマンド実行例

```
A001 EXAMINE INBOX
* FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
* OK [PERMANENTFLAGS ()] Read-only mailbox.
* 430 EXISTS
* 0 RECENT
* OK [UNSEEN 1] First unseen.
* OK [UIDVALIDITY 1436801777] UIDs valid
* OK [UIDNEXT 431] Predicted next UID
* OK [NOMODSEQ] No permanent modsequences
A001 OK [READ-ONLY] Examine completed (0.000 secs).
```

▼表4　FETCHの第2引数（取り出したい項目）

項目	意味
BODY	メッセージの構造
BODY[HEADER]	メッセージのヘッダ部[注2]
BODY[HEADER.FIELDS (Subject)]	件名[注2]
BODY[TEXT]	メッセージの本文部[注2]
RFC822	メッセージ全体
RFC822.SIZE	メッセージサイズ
ENVELOPE	メッセージの日付、件名、送信者、受信者などの情報
FLAGS	既読(\Seen)、返信済み(\Answered)などのフラグ
UID	メッセージの一意な識別子となる数値

注2） SELECTでメールボックスが選択された場合はメッセージが既読状態になります。

▼図17　FETCHコマンド実行例

```
A001 FETCH 1,2 BODY[HEADER.FIELDS (SUBJECT)]
* 1 FETCH (BODY[HEADER.FIELDS (SUBJECT)] {17}
Subject: test

)
* 2 FETCH (BODY[HEADER.FIELDS (SUBJECT)] {21}
Subject: hogehoge

)
a OK Fetch completed.
```

▼表5　STOREで設定できるフラグ

フラグ	説明
\Seen	既読。FETCHコマンドの指定によっては自動的に設定される
\Answered	返信済み
\Flagged	フラグ。Thunderbirdなどのメールアプリでは☆としてマークされる
\Deleted	削除。STOREではマークを付けるだけで実際には削除されない
\Draft	草稿

タグなし応答が返ります(**図17**)。

応答は、「メッセージ番号」+「FETCH」+「指定した項目とその値のペア」です。「{数字}」は文字列の特殊な表記で、次の行以降に書かれている文字列のバイト数を表します。

フラグ設定／削除

STOREコマンドで、メッセージごとに**表5**に示したフラグを設定できます。サーバの実装によってはこれら以外のフラグも設定できることがあります。

「+FLAGS」で指定フラグの追加、「-FLAGS」で指定フラグの削除、「FLAGS」で指定したフラグの置き換え、になります(**図18**)。EXPUNGEコマンドを実行すると\Deletedフラグが付いたメッセージが削除されます(**図19**)。

コピー

COPYコマンドでメッセージをほかのメールボックスにコピーできます。引数はメッセージ番号とコピー先のメールボックスです。

```
A001 COPY 5 Other
A001 OK Copy completed.
```

IMAPにはメッセージを移動するコマンドはありません。メールアプリがメッセージを移動した場合は、内部的にはコピーと削除が行われています。

メールボックスのクローズ

CLOSEコマンドで、SELECT/EXAMINEで選択したメールボックスをクローズします。

\Deletedフラグが付いたメッセージはCLOSEコマンド実行時に削除されます。\Deletedフラグを付けたままで削除したくない場合は、CLOSEコマンドを実行してはいけません。\Deletedフラグを付けたまま別のメールボックスを選択したい場合は、CLOSEせずにSELECT/EXAMINEを実行すれば良いです。

メッセージの格納

APPENDコマンドで、メッセージをメールボックスに格納できます。メールアプリで作成中のメッセージを草稿に格納したり、送信したメールを送信控フォルダに格納したりする際に使用されます。

引数はメールボックス、フラグ(オプション)、タイムスタンプ(オプション)、メッセージです。

```
A001 APPEND INBOX (\Seen) {300}
+OK
(...300バイトのメッセージ...)
A001 OK Append completed.
```

終了

LOGOUTコマンドで接続を切断します。

```
A001 LOGOUT
* BYE Logging out
A001 OK Logout completed.
```

◆ ◆ ◆

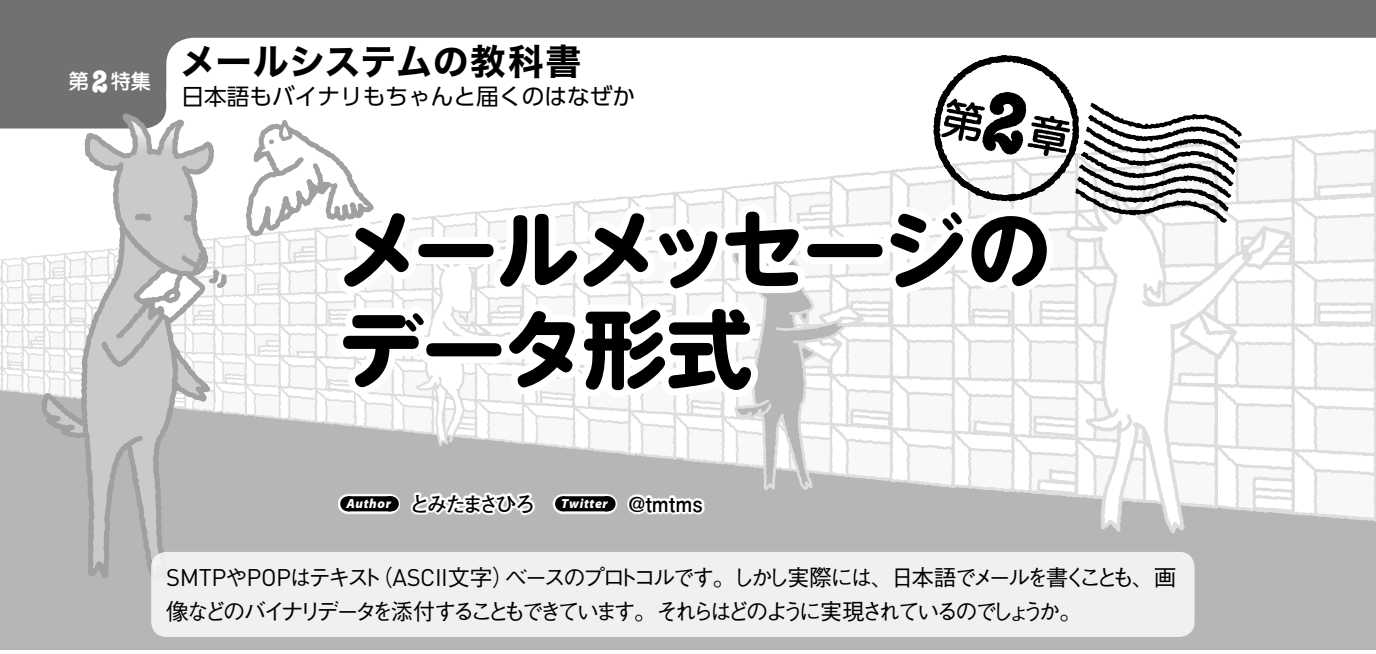
ここまでDNS、POP、SMTP、IMAPの概要を説明しました。メールアプリの裏側では、これらのコマンドを実行することでメールの送受信や管理が行われています。SD

▼図18　STOREコマンド実行例

```
A001 FETCH 1 FLAGS
* 1 FETCH (FLAGS (\Seen))  ←既読状態
A001 OK Fetch completed.
A002 STORE 1 -FLAGS (\Seen)  ←既読フラグを削除
* 1 FETCH (FLAGS ())
A002 OK Store completed.
A003 FETCH 1 FLAGS
* 1 FETCH (FLAGS ())  ←未読になった
A003 OK Fetch completed.
```

▼図19　EXPUNGEコマンド実行例

```
A001 STORE 1,3 +FLAGS (\Deleted)  ←削除フラグ設定
* 1 FETCH (FLAGS (\Deleted \Seen))
* 3 FETCH (FLAGS (\Deleted \Seen))
A001 OK Store completed.
A002 EXPUNGE  ←実際に削除
* 3 EXPUNGE
* 1 EXPUNGE
A002 OK Expunge completed.
```



第2章

メールメッセージのデータ形式

Author とみたまさひろ Twitter @tmtms

SMTPやPOPはテキスト（ASCII文字）ベースのプロトコルです。しかし実際には、日本語でメールを書くことも、画像などのバイナリデータを添付することもできています。それらはどのように実現されているのでしょうか。

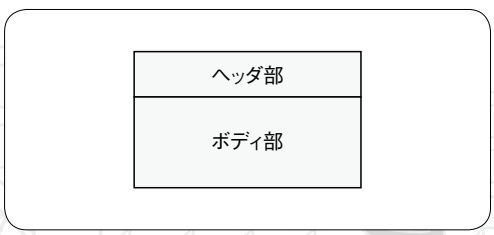
メールメッセージの構造

メールメッセージの基本的な形式はRFC 5322で規定されています。RFC 5322はかなり複雑です。日本語や添付ファイルを扱う場合はまた別のRFCがあり、さらに複雑になります。

メッセージの構造は大まかにヘッダ部とボディ部に分かれます。ヘッダ部には送信者名、受信者名、件名、日付など利用者に見えるものや、メッセージの構造、文字コード、通過したMTAなど通常利用者が意識しない情報が含まれます。ボディ部はメールの本文、添付ファイルのデータが含まれます。

添付ファイルが付いていないメッセージは単純な構造です（**図1**）。ヘッダ部とボディ部は空行で区切られます。とても単純なメッセージの例を**リスト1**に示します。添付ファイルが付いているメッセージについては後述します。

▼図1　メールメッセージの構造



ヘッダ部

ヘッダ部は複数のヘッダフィールドから構成されます。ヘッダフィールドは、「フィールド名」+「:」+「フィールド値」の形式です。CRLF（0x0D 0x0A）で終わります。

物理的な1行の長さは78バイト以下にすべきという制限がありますが、長い行は折り返すことで論理的に長い1行を表すことができます。折り返しは「CRLF」+「空白」（0x20または0x09（TAB））で、空白がある個所で行えます。次の2つの表現は同じ意味になります。

```
Subject: long long long
```

```
Subject: long
 long
 long
```

ヘッダ部にはASCII文字だけしか含めることができません。フィールド名の大文字小文字は区別されません。各フィールドの値は、それぞれ構造が決まっています。いろいろなフィール

▼リスト1　単純なメッセージの例

```
Date: Wed, 15 Jul 2015 23:20:10 +0900
Message-Id: <12345@example.com>
From: sender@example.com
To: recipient@example.net

Hello.
```

ドがありますが、必須のフィールドは、From、Date、Message-Idだけです。

メールアドレス

フィールドの前にメールアドレスの形式について説明します。メールアドレスはローカルパートとドメインを「@」で結合したものです。ローカルパートは、英数字と一部の記号を「.」で結合したもの、または「"」でくくられた任意の文字列です。「"」でくくられた文字列中に「\」「"」を含めたい場合は「\\」「\"」と記述します。ドメインは英数字と「-」を「.」で結合したものです。

次は正当なメールアドレスの例です。

- hoge.fuga@example.com
- hoge$fuga*@example.com
- "hoge..fuga"@example.com

次はメールアドレスとしては不当な例です。

- hoge..fuga@example.com
  ……ローカルパートに「.」が連続している
- hoge.@example.com
  ……ローカルパートが「.」で終わっている
- "hoge"fuga"@example.com
  ……「"」中に「"」がそのまま現れている
- hoge@exam_ple.com
  ……ドメイン中に「_」が含まれている

メールアドレスの厳密な形式については第1特集第1章の正規表現の記事(p.25～28)に書きましたので、そちらを見てください。

From/Sender/Reply-To

Fromはメッセージの作成者、Senderはメッセージの送信者を表します。作成者と送信者が同一の場合は、Senderは必要ありません。通常はFromだけのことが多いでしょう。Fromはメッセージに必ず必要なフィールドです。

作成者／送信者が1つの場合の形式は、次のいずれかの表現になります。

- メールアドレスそのまま
  例：mail@address
- < >でくくられたメールアドレス
  例：<mail@address>
- 表示名つきメールアドレス
  例：display name <mail@address>

あまり見かけませんが、実はFromは「,」で区切って複数のアドレスを含むことができます。その場合はSenderが必須となります。Senderには1つの送信者しか含むことができません。

Reply-Toはメッセージの受信者が返信をする際の宛先となるアドレスです。このフィールドがない場合は、Fromが返信先になります。Reply-Toの形式はTo/Cc/Bccと同じです。

To/Cc/Bcc

To、Cc、Bccはそれぞれ、おもな宛先、副の宛先、受信者に見えない宛先です。奇妙に思われるかもしれませんがTo、Cc、Bccはメッセージに必須ではありません。メールアプリからメールを送信する場合は、これらのフィールドに記述した宛先にメッセージが送信されるので、通常は一致するのですが、メッセージ上のこれらのフィールドと実際に送られる宛先は異なっている場合もあります。たとえば自分が参加しているメーリングリストに送信されたメールは、自分に届いたメッセージのToを見ても自分のメールアドレスは書かれていないでしょう。

実際に送られる宛先はSMTPのエンベロープ受信者アドレスです。第1章で説明したように、エンベロープ受信者はメッセージとは別に指定するので、異なる場合があり得ます。同様に、From/Senderとエンベロープ送信者が異なる場合もあります。

To、Cc、Bccフィールドには複数のアドレスやグループを含めることができます。値の形式は、Fromとほぼ同じですが、グループが指定できることが異なります。グループは「表示名」+「:」+「0個以上の宛先」+「;」という形式です。

・display name:;
・display name: mail@address;
・display name: mail@address, name <mail@address>;

宛先が0個のグループは、宛先としてエンベロープ受信者が指定されているけれども、それをメールの受信者に知らせたくない場合などに使用されます。

Date

Dateはメッセージの送信時刻が記録されます。「Thu, 16 Jul 2015 20:05:05 +0900」のような形式で記録されます。先頭の曜日は省略してもかまいません。

Message-Id/In-Reply-To/References

Message-Idはメッセージを一意に識別するためのフィールドです。形式は「<abc.def.hoge@example.com>」です。世界中で過去から未来まで含めて一意にする必要があるため、時刻やドメイン名から作られることが一般的です。

In-Reply-Toフィールドはメッセージに返信する際に付加されます。これは返信元のメッセージのMessage-Idの値を含み、どのメッセージに対して返信したかの情報を保持するためのものです。Referencesも同様ですが、直接の返信元だけでなく、返信を繰り返すたびに後ろにMessage-Idが追加されていきます(**リスト2**)。これらの情報はメールアプリ上でメッセージをスレッドツリー表示するのに使用されます。

▼リスト2　In-Reply-ToとReferencesが付加される様子

```
元メッセージ(A)
Message-Id: <A@example.com>

Aに対する返信(B)
Message-Id: <B@example.com>
In-Reply-To: <A@example.com>
References: <A@example.com>

Bに対する返信(C)
Message-Id: <C@example.com>
In-Reply-To: <B@example.com>
References: <A@example.com> <B@example.com>
```

Subject

Subjectフィールドは件名です。このフィールドはとくに決まった形式を持っていません。

空白とコメント

構造を持つフィールドの値には、空白を含めることができます。単語の途中以外、ほとんどの場所に空白を置くことができます。

また、空白を置けるところにはコメントを置くことも可能です。コメントは「(」と「)」でくくられた文字列で、コメントをさらに入れ子にすることもできます。たとえば、次のFromフィールド

```
From: hoge@example.com
```

は、次のように書いても同じ意味になります。

```
From: hoge    @ (comment) example.com ➥
(comment2)
```

任意のフィールド

決められているフィールド名以外のフィールドを自由に付けることができます。フィールド名は「:」以外の印字可能文字(0x21～0x7E)であれば何でも使えます。任意のフィールドであることを表すために「X-」で始まる名前が使われることが多いようです。

ボディ部

ボディ部はメッセージの本文です。ボディ部も通常はASCII文字だけで構成されます。

Unicode以前の日本語文字列の表現方法として、ASCII文字だけで表すことができるISO-2022-JPというエンコーディングがあります。これを使用すればメールメッセージの制約の中

でも日本語を記述できたため、メールで日本語を送る場合はISO-2022-JPを使用するのが一般的でした。現在でも、日本語のメッセージではISO-2022-JPが多く使用されています。

MIME(Multipurpose Internet Mail Extensions)

以前は、メールメッセージのヘッダにはASCII文字しか含められなかったので、日本語の件名や送信者名を記述できませんでした。ボディ部と違い、ESC(0x1B)を含められないのでISO-2022-JPも使用できません[注1]。

バイナリデータを送ることもできませんでした。バイナリデータをメールで送る場合は、uuencodeコマンドなどでテキストに変換したものを本文に貼り付けて送ったりしていました。

その後MIME規格が生まれ、多くのメールアプリがこの規格に対応することで、利用者が意識することなく、表示名や件名に非ASCII文字を使えるほか、バイナリデータも添付できるようになりました。

MIMEはRFC 2045、2046、2047、2048、2049で規定されています。MIMEを使用したメッセージは「MIME-Version: 1.0」というヘッダフィールドを付けて、MIMEメッセージであることを示します。MIMEに対応していない環境でも問題が起きないように、MIMEはRFC 5322と互換があるように作られています。

■ヘッダ中の非ASCII文字

MIMEでは文字列を特定の方法でASCII文字にエンコードすることで、ヘッダ中に非ASCII文字を含めることができます。**=?UTF-8?B?5pel5pys6Kqe?=**はUTF-8の文字列「**日本語**」をBエンコーディングで表現したものです。

Bエンコーディングは、3バイトのデータを4文字で表すエンコーディングです。エンコード後の文字はASCII英数字と「+」「/」「=」だけになります。Rubyの場合は**リスト3**のようにして変換できます。

エンコーディングはBエンコーディングのほかにQエンコーディングもあります。同じく「**日本語**」をQエンコーディングで表すと、**=?UTF-8?Q?=E6=97=A5=E6=9C=AC=E8=AA=9E?=**となります。Qエンコーディングは非ASCII文字、空白、「=」「?」「_」を1バイトずつ、「=」に続けて16進数2桁で表現したものです。空白は「_」に変換されます。これら以外のASCII文字はそのまま表現されます。

どちらのエンコーディングも特殊な記号は使われていないのでRFC 5322と互換があります。

注意すべき点は、これらのエンコーディングは空白で区切られた文字列の単位で行われるという点です。「**012 漢字 ABC**」は**012 =?UTF-8?B?5ryi5a2X?= ABC**と変換できますが、「**012漢字ABC**」を**012=?UTF-8?B?5ryi5a2X?=ABC**としてはいけません。そのような場合は、前後の**012**、**ABC**も含めてエンコーディングし、**=?UTF-8?B?MDEy5ryi5a2XQUJD?=**とする必要があります。

さらに、エンコードされた文字列が続くと間の空白が無視されて結合されるという規則があります。「**日本語 文字列**」を**=?UTF-8?B?5pel5pys6Kqe?= =?UTF-8?B?5paH5a2X5YiX?=**としてしまうと、デコードしたときに「**日本語文字列**」となってしまいます。このような場合は途中の空白も含めて**=?UTF-8?B?5pel5pys6KqeI0aWh+Wtl+WIlw==?=**とする必要があります。

また、「"」でくくられている場合はデコードしてはいけません。**"=?UTF-8?B?5pel5pys6Kqe?="**は「**" 日本語 "**」にデコードするのではなく、そのままの文字列として扱う必要があります。

このあたりの処理は非常に複雑なため実装者泣かせで、間違って実装されているプログラムも多いようです。

▼リスト3　RubyによるBエンコーディング

```
["日本語"].pack("m") #=> "5pel5pys6Kqe\n"
"5pel5pys6Kqe".unpack("m").first #=> "日本語"
```

注1）余談ですが、日本でメールの本文の先頭で自分の名前を名乗る習慣は、Fromに日本語で名前を書けなかったためではないかと個人的には思っています。

Content-Transfer-Encoding

MIMEエンコーディングによりヘッダ部に日本語を記述できるようになりました。さて次はボディ部です。

JISベースの日本語文字列の表現方法はISO-2022-JP以外にもSHIFT_JIS、EUC-JPがあります。UnicodeだとUTF-8が一般的です。これらはISO-2022-JP以外はいずれも8bitのデータです。また、そもそもテキストデータですらないバイナリデータはやはりテキストに変換する必要があります。

これらのデータをテキストに変換するための方式は、Base64とQuoted-Printableの2種類があります。ヘッダ部のContent-Transfer-Encodingフィールドで、ボディ部のエンコード方式を示します。Content-Transfer-Encodingの値はほかに「7bit」、「8bit」があります。Content-Transfer-Encodingフィールドを省略した場合は「7bit」が指定されたものとみなされます。これはとくに変換されていない7bitデータということを表しています。8bitについては後述します。

Base64

Base64はヘッダのBエンコーディングと同じ方式で、バイナリデータをテキストに変換するエンコーディングです。ヘッダ部には「Content-Transfer-Encoding: base64」と指定します。エンコード後の文字はASCII英数字と「+」「/」「=」だけであり、1行が76バイトよりも大きくならないように改行が挿入されます。デコード時には改行は単純に無視されます。

Quoted-Printable

Quoted-PrintableはヘッダのQエンコーディングと似た方式のエンコーディングです。ヘッダ部には「Content-Transfer-Encoding: quoted-printable」と指定します。空白(0x20)、タブ(0x09)、「=」を除く0x21～0x7Eの範囲のASCII文字はそのままですが、それら以外の文字は「=」に続けて16進数2桁で表現します。

改行コードはCRLFです。CRLFとして表れないCRやLFデータはそれぞれ=0D、=0Aに変換する必要があります。空白とタブ文字はそのまま記述できますが、行末には置けません。もし行末に空白かタブ文字がくる場合はエンコードして=20、=09とします。1行が76文字以内に収まらない場合は行末に「=」を置いて改行します。デコード時には行末の「=」は続く改行も含めて無視されます。

Content-Type

Content-Typeフィールドは「type/subtype」の形式でボディ部のデータの種類を表します。さらに「; attribute=value」を続けて属性を示すこともできます。**表1**に例をいくつか示します。Content-Typeフィールドが省略された場合は「text/plain; charset=US-ASCII」とみなされます。

type/subtypeのリストはIANA(Internet Assigned Numbers Authority)で管理されていて、Webで見られます[注2]。同様にcharsetのリストも見られます[注3]。

マルチパートメッセージ

Content-Transfer-EncodingとContent-Typeを適切に組み合わせることで、8bit表現の日本語テキストや画像ファイルなどのバイナリデー

注2) http://www.iana.org/assignments/media-types/media-types.xhtml
注3) http://www.iana.org/assignments/character-sets/character-sets.xhtml

▼表1 Content-Typeで指定できるデータの種類

種類	説明
text/plain	US-ASCIIテキスト
text/plain; charset=UTF-8	UTF-8テキスト
text/html	HTMLテキスト
image/jpeg	JPEG画像
application/octet-stream	バイナリデータ
application/javascript	JavaScript
application/vnd.ms-excel	Excel
message/rfc822	メールメッセージ

タをメールで送れるようになりました。ただ、バイナリデータを送れると言っても、1つのメッセージで1つのデータしか送れないようだと困ります。何らかのデータをメールで送りたい場合は、本文のテキストと1つ以上の添付ファイルという形式が一般的でしょう。

1つのメッセージに複数のデータを含めたメッセージをマルチパートメッセージと言います。マルチパートメッセージの構造はボディ部が複数のパートに分かれていて、各パートはメールメッセージと同じ構造を持ち、ヘッダ部とボディ部があります(**図2**)。マルチパートが入れ子になることもあります(**図3**)。

マルチパートメッセージの構造

マルチパートメッセージはContent-Typeフィールドに「multipart/mixed」を指定し、さらにboundary属性で各パートの区切り文字列を指定します。

boundary文字列の先頭に「--」を付加した文字列で各パートが区切られ、さらにその末尾に「--」を付加した文字列でパートの終了を表します(**リスト4**)。boundary文字列は、ボディ部に現れない文字列を選択する必要があります。

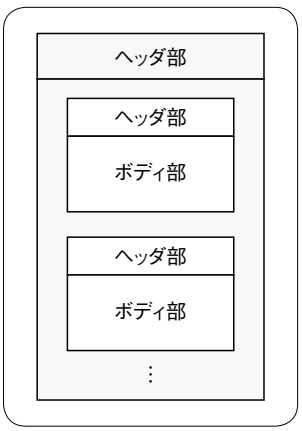
ファイル名

添付したデータのファイル名などの情報はContent-Dispositionヘッダフィールドに含みます(RFC 2183)。Content-Dispositionの値は「inline」(メッセージ本文中に表示)か「attachment」(添付ファイルとして表示)で、さらに「filename」などのパラメータを続けることができます(**リスト5**)。

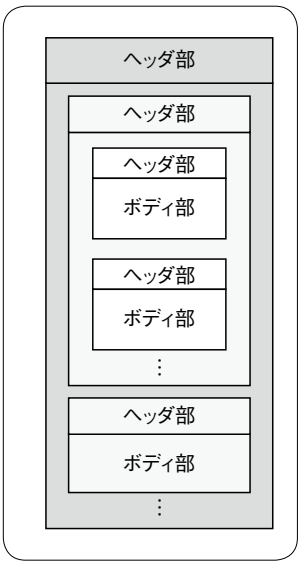
非ASCII文字を含むファイル名はMIMEエンコーディングでは記述できません。MIMEエンコーディングは必ず「=」「?」を含み、これらの文字は「"」でくくらなければパラメータの値には書けないのですが、MIMEヘッダでは「"」でくくられた文字列はデコードしてはいけないためです。そのためMIMEエンコーディングとは異なる方式でエンコードする必要があります(RFC 2231)。

エンコーディングの詳細は割愛しますが、

▼図2　マルチパートメッセージの構造



▼図3　マルチパートが入れ子になっている



▼リスト4　マルチパートメッセージの例

```
Content-Type: multipart/mixed; boundary="123456789.hoge"

(ここはマルチパートメッセージに対応したアプリでは表示されない)

--123456789.hoge
Content-Type: text/plain

(メッセージ本文)

--123456789.hoge
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64

(Base64で変換された添付ファイルデータ)

--123456789.hoge--

(ここもマルチパートメッセージに対応したアプリでは表示されない)
```

1つめのパート

2つめのパート

▼リスト5　Content-Dispositionの例

```
Content-Disposition: attachment; filename="hoge.data"
```

UTF-8で「**日本語ファイル名.txt**」をRFC 2231でエンコードした例を**リスト6**に示します。

8bitデータの扱い

SMTPは基本的に7bitのデータしか扱えませんが、拡張で8bitデータを扱うこともできます。SMTPのEHLOコマンドの応答で8BITMIMEが返される場合は、そのSMTPサーバは8bitデータを受信できます。

8bitデータを含むメッセージを送信したいクライアントは、EHLOコマンドの応答に8BITMIMEが含まれていることを確認し、MAILコマンドにBODY=8BITMIMEというオプションを渡します。さらにメッセージヘッダに「Content-Transfer-Encoding: 8bit」を指定することで、ボディ部に8bitデータを含めることができます。これでSHIFT_JISやUTF-8の本文をそのまま送信できます。

なお、8bitデータを送れるといっても、バイナリデータをそのまま送れるわけではありません。SMTPはテキストベースのプロトコルですし、メールメッセージもテキストデータです。

8bitテキストデータがバイナリデータと異なるのは、改行コード(CRLF)で区切られた各行の長さが最大998バイト以下であることと、NUL(0x00)データを含まないことです。この条件に当てはまらない場合は、Base64やQuoted-Printableでテキストに変換する必要があります。

文字化け

メールで文字化けに遭遇することが結構あります。たいていはメールを送信したアプリがメッセージを正しく作成していないためです。

▼リスト6　日本語ファイル名をRFC 2231でエンコードした例

```
Content-Disposition: attachment;
 filename*0*=utf-8''%E6%97%A5%E6%9C%AC%E8%AA%9E%E3%83%95;
 filename*1*=%E3%82%A1%E3%82%A4%E3%83%AB%E5%90%8D.txt
```

- **charsetが正しくない**
  テキストデータはSHIFT_JISなのに、Content-TypeのcharsetにISO-2022-JPと書かれていると、正しく表示できない
- **charsetに含まれない文字を使用している**
  たとえばISO-2022-JPは、半角カナ、丸囲み数字(「①」など)、括弧付き漢字(「㈱」など)を含むことができない。しかし、Windowsではこれらの文字をISO-2022-JPとして無理やりメールを送ってしまうため、Windows以外の環境でこのようなメールを受けると送り手の期待どおりに表示されないことがある。Windowsが作成したメッセージのISO-2022-JPは、ISO-2022-JPではなくCP50221として扱うことで文字化けを避けられる。同様にSHIFT_JISはCP932として扱えば良い
- **MIMEエンコーディングが間違っている**
  たとえばMIMEエンコーディングした文字列が「"」でくくられて**"=?UTF-8?B?5pel5pys6Kqe?="**となっていると、正しく実装されたアプリではそれをデコードしないため、利用者からすると文字化けのように見えてしまう。日本語を含む添付ファイル名が正しくRFC 2231でエンコーディングされていない場合も文字化けして見えることがある

◆ ◆ ◆

普段何気なく読み書きしているメールですが、メッセージの内部構造は非常に複雑になっています。メールは非常に古いプロトコルで、メールメッセージのRFCの初期の版のRFC 822は1982年に作られました。現在のRFC 5322でもそれと矛盾しないように規格が拡張されているのが、複雑になっている原因です。メールを読み書きするようなプログラムを作るような人以外は、この複雑なメッセージ構造を知っておく必要はないかもしれませんが、メールの文字化けなどの問題に遭遇したときにこの記事が一助になれば幸いです。SD

第3章

メールクライアントソフトのデータ管理

Author 櫻井 賢一（さくらい けんいち）　NYKソフトウェア／C Channel㈱
Mail Kenichi.Sakurai@cpa.com

本章では代表的な無料メールクライアントソフトである「Mozilla Thunderbird」を例に、メールクライアントソフトがどのようにメールデータを管理しているかについて説明します。

はじめに

近年、Gmailなどに代表されるいわゆる「Webメール」の機能が充実しているため、メールクライアントソフトを使っていないという人も増えてきました。一方、メールシステム自体はクラウド全盛の今の時代よりもずっと前から使われている技術で、第1章、第2章で述べられてきたプロトコルやサーバ側の技術とともに、メールクライアントソフトも進化・発展してきています。

本章では普段何気なく使っているメールクライアントソフトがどのようにメールデータを管理しているか、検索や読み込みの高速化のためにどのような工夫をしているかについて、代表的な無料メールクライアントソフトであるMozilla Thunderbird（以下、Thunderbird）を例に説明します。

Thunderbirdとは

Thunderbirdは非営利団体のMozilla Foundationを中心に開発が行われている、オープンソースの無料メールクライアントソフトです（**図1**）。バージョン1.0のリリースから10年以上の歴史があり、Windows、Mac OS X、Linuxで利用可能です。日本でも、個人／法人ともに広く使われているメールクライアントソフトの1つです。

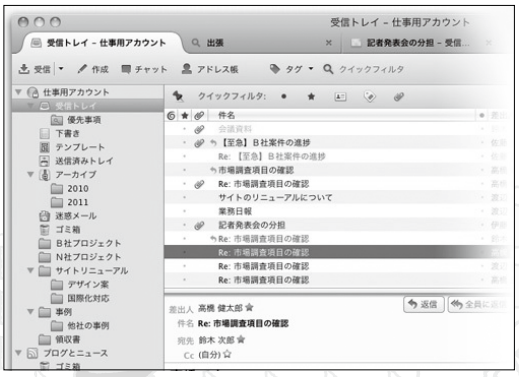
メールデータの保存

メールクライアントソフトを利用して送受信したメールは、通常ローカル（メールクライアントソフトがインストールされたPC）のファイルシステムに保存されています。ローカルに保存しているためネットに接続していないオフライン状態のときでもメールの閲覧・検索ができるわけですが、一体どのような形式で送受信したメールは保存されているのでしょうか。

メールデータの保存形式

メールクライアントソフトがどのようにメールを保存するかについての規約・制約は基本的

▼図1　Thunderbirdの画面イメージ



にありませんので、各メールクライアントソフトでそれぞれで異なっています。Thunderbirdでは「受信トレイ」、「送信済みトレイ」や、たとえば自分で作成した「仕事用」といった各フォルダごとに1つのファイルを作成し、そのファイルにフォルダ内の全メールを保存しています。

このファイルは、メールがネットワーク上で保存される際に使用されるmbox形式をベースに、独自の拡張を加えた形式になっています。mbox形式は**図2**に示すように「From␣」で区切られた各メールを、プレーンテキストで1ファイルに保存しています。保存される内容は送信されてきた形式そのままですので、たとえば本文がBase64でエンコードされたメールであれば、そのまま(エンコードされたまま)の状態で保存されます。

高速化のための工夫

単純に上記のmbox形式のファイルのみしか保持していなかったとすると数千、数万、あるいはそれ以上のメールの検索・閲覧・削除などを効率的に行うことはできません。Thunderbirdでは、この1つ1つのmbox形式のファイルに対応する索引ファイルをそれぞれ作成し、アクセス効率を高めています。また、すべてのフォルダのメールを効率よく検索するために検索用データベースも保持しています(**図3**)。

索引ファイル

索引ファイルは.msfの拡張子がついたファイルです。1つのmboxファイルに対して1つの索引ファイルが作成されます。索引ファイルは、フォルダ内の各メールやスレッドのサマリー情

▼図2　mbox形式(ThunderbirdのInboxファイルの例)

```
From
MIME-Version: 1.0
x-no-auto-attachment: 1
Received: by 10.28.22.137; Sun, 29 Mar 2015 06:12:51 -0700 (PDT)
Date: Sun, 29 Mar 2015 06:12:51 -0700
Message-ID: <CAPW8GLbVw0Hr_8==Pvhzgf6mq8YVq0Dqx959XKoSKc3a4=AqfA@mail.gmail.com>
Subject: =?UTF-8?B?R29vZ2xlIEFwcHMg44GnIEdtYWlsIOOCkuS9v+eUqOOBme0Ci+aWueazlQ==?=
From: =?UTF-8?B?R21haWwg440B4408440g?= <mail-noreply@google.com>
To: =?UTF-8?B?5qu75LqV44Kx44Oz44Kk44OB?= <kenichi@cchan.tv>
Content-Type: multipart/alternative; boundary=047d7bacb124681a4c05126d1f25

--047d7bacb124681a4c05126d1f25
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: base64

IEdvb2dsZSBBcHBzIOOBpyBHbWFpbCDjgpLkvb/nlKjjgZnjgovmlrnms5UNCltpbWFnZTogR29v

<<中略>>

aXY+PC9kaXY+PC9ib2R5PjwvaHRtbD4NCg==
--047d7bacb124681a4c05126d1f25--
From
MIME-Version: 1.0
x-no-auto-attachment: 1
Received: by 10.28.22.137; Sun, 29 Mar 2015 06:12:51 -0700 (PDT)
Date: Sun, 29 Mar 2015 06:12:51 -0700
Message-ID: <CAPW8GLYYEiFCu=2A05uY5Pq8-9J7Y94tQansT5z-k3LGcWCWZg@mail.gmail.com>
Subject: =?UTF-8?B?44Gp44GT44Gn44KCIEdtYWlsIOOCkuacgOWkp+mZkOOBq+a0u+eUqA==?=
From: =?UTF-8?B?R21haWwg440B4408440g?= <mail-noreply@google.com>
To: =?UTF-8?B?5qu75qu75LqV44Kx44Oz44Kk44OB?= <kenichi@cchan.tv>
<<以下略>>
```

送受信に利用したエンコーディングのまま保存

「From␣」で各メールを区切る

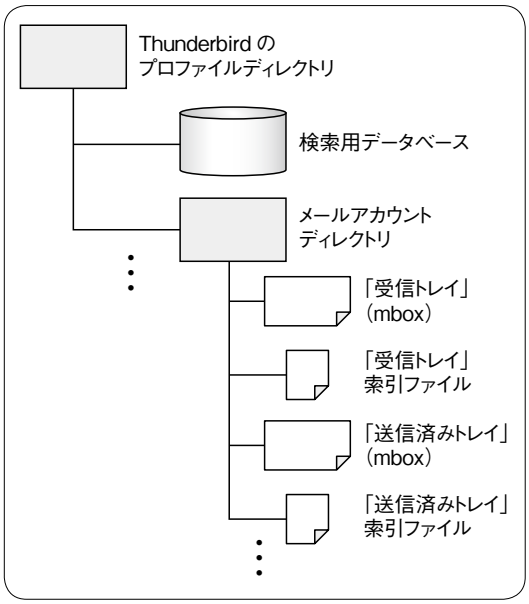
1通目のメール

2通目のメール

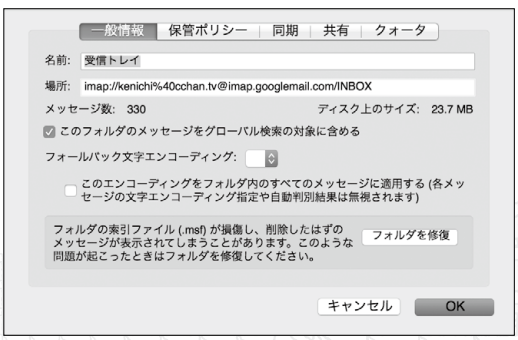
報と、フォルダのメタ情報などを保持しています。このファイルがあることにより各メールへのアクセスが素早く行えます。大量のメールがあるフォルダで表示をスレッドごとに切り替えたり、スレッド表示を解除したりということが高速に行えるのもこの索引ファイルのおかげです。まさに図鑑などの巻末についている「索引ページ」のような役割を果たしているわけです。

文字どおり「索引」情報で、メールの原本を保持しているわけではありません。ですのでmboxファイルがあれば、索引ファイルの再作成をすることが可能です。仮に索引ファイルが破損したという場合でも、対象フォルダの情報画面のメニューから、**図4**にある「フォルダを修復」ボタンを押すことにより索引ファイルを再作成できます。

▼図3　メールデータを保存しているファイル



▼図4　Thunderbirdのフォルダ情報画面



メールの削除処理

図4のフォルダ情報画面の記述内容を見て「あれっ？」と思われた方もいらしゃるかもしれませんね。『フォルダの索引ファイル(.msf)が損傷し、削除したはずのメッセージが表示されてしまうことがあります。』の部分です。

上で説明したとおり、Thunderbirdでは、メールの原本は各フォルダごとにmbox形式で1つのファイルにまとめて保存されています。そのため1つのフォルダに数千、数万あるいはそれ以上のメールが格納されている可能性があります。新規に受信・送信したメールはmboxファイルの末尾に追記されていきます。しかし、メールの削除やフォルダ間の移動の実行時に、mboxファイルの中から対象のメールを削除する(取り除いて詰める)ということは行いません。もしそうしてしまうとファイル内の更新対象個所が多くファイルシステムに対するオーバーヘッドが大きくなり、処理を実行するたびに時間がかかってしまうためです。

実際の削除の実行時には、mboxファイル内の対象メールの管理用ヘッダーのフラグを削除状態に変更し、あわせて索引ファイルの情報を更新するだけにとどめます[注1](**図5**)。mboxファイルからメールが削除されるわけではなく、「このメールはこのフォルダから削除されたメールだよ」と印をつけるだけですので、メールの削除を実施してもmboxファイルのサイズは変わりません。

索引ファイルが破損して不整合状態になったような場合に、「削除したはずのメッセージが表示」されるということが起こり得るのも、このようなしくみのためです。

注1) 設定、状況によりmboxファイルの更新はまったく行われない場合もあります。

フォルダの最適化

ということは、一見するといくら不要なメールをせっせと削除してもローカルディスクの使用領域の削減につながらなさそうです。「削除済み」のマークが付けられたメールを実際にmboxファイル内から削除し、ファイルサイズを小さく適正にする処理をフォルダの「最適化」と呼んでいます。ちょっと異なりますが、メモリ管理でたとえるとガベージコレクションをイメージしていただけると理解しやすくなるかもしれません。

現バージョンのThunderbirdのデフォルトの設定では、最適化実行によりディスク容量が20MB以上削減できる場合は自動で最適化が実行されるようになっています(**図6**)。また、手動で最適化を実行することも可能です。実際に数万通のメールを保存しているようなサイズの大きなフォルダで、いくつかメールを削除してから最適化を実行してみると、それなりに時間を要する処理だということを体感できます。

検索用データベース

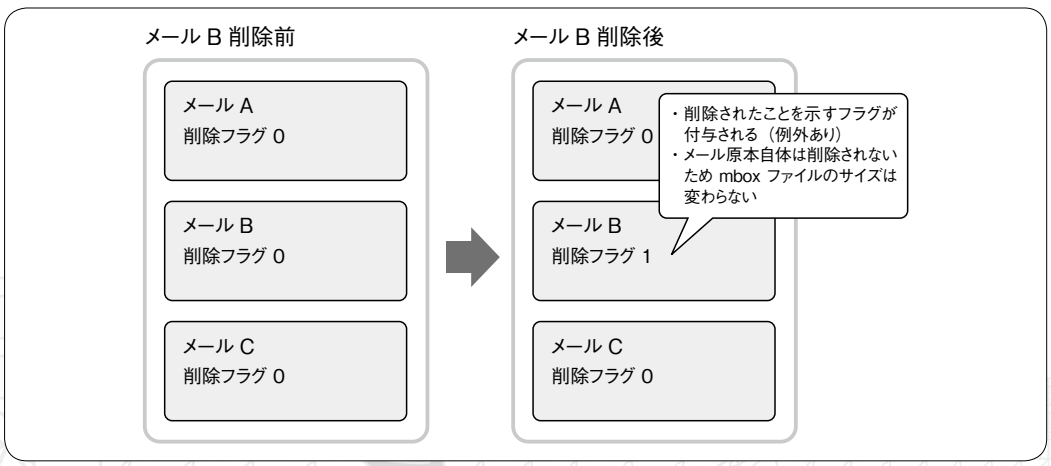
索引ファイルの利用によるアクセスの高速化に加えて、Thunderbirdでは検索専用のデータベースを持っていて検索処理の高速化を実現しています。「グローバル検索データベース(Gloda)」と呼ばれるこの検索用データベースは、SQLiteを用いて実装されています。軽量・高速なRDBMSとして実績のあるSQLite上でインデックス情報を管理することにより、メール原本自体は、そのメールが送信されたエンコードのまま保存しているにもかかわらず、(もちろん日本語も含めて)高速なフルテキストサーチが可能となっています。

このファイルもメール原本自体を管理しているものではありませんので、索引ファイル同様に再作成が可能[注2]です。

注2) 詳細手順はhttps://support.mozilla.org/ja/kb/rebuilding-global-databaseを参照。

▼図6 フォルダ最適化自動実施に関する設定(Thunderbirdの環境設定画面)

▼図5 メール削除実施時のmboxファイルの変化(メールBの削除を実行した場合)



メールのエクスポート

Thunderbirdでは個々のメールを選択して、eml形式のファイルとしてエクスポートすることが可能です。eml形式は多くのメールクライアントソフトでサポートされていて、インポートやエクスポートの際によく用いられています。1ファイルにメール1通の情報を、プレーンテキストのMIMEフォーマットで格納します。

メールデータのバックアップ・移行

これまでの説明のとおり、Thunderbirdの索引ファイルやグローバル検索用データベースは、元となるメールデータのファイル(mboxファイル)から再作成可能です。逆にmboxファイルはメールの原本となる情報を管理しているため、重要なフォルダのmboxファイルはしっかりとバックアップを取るべきです[注3]。

正式な移行手順ではありませんが[注4]、PCの買い替えなどでThunderbirdのデータを移行する場合、Thunderbirdのプロファイルディレクトリの適切な位置にmboxファイルさえ置けば、Thunderbirdを起動した際に認識し、適切に取り込まれて、索引ファイルや検索データベースの作成などの処理が行われます。

注3) POP3を利用していてサーバ側にメールデータが残っていない場合。
注4) 各ファイルの位置づけを説明するための記述です。正式な移行手順は、https://support.mozilla.org/ja/kb/moving-thunderbird-data-new-computerを参照してください。

Thunderbirdからほかのメールクライアントソフトに移行する場合にも、移行先のメールクライアントソフトが対応していれば、mbox形式のファイルそのものやエクスポートしたeml形式のファイルを利用して、メールデータを引き継ぐことが可能です。

Thunderbird以外のメールクライアントソフトのデータ管理

代表的なメールクライアントソフトの1つであるMicrosoft Outlookでは、Outlookデータファイル(.pst)にメールの情報が格納されています。このpstファイルのフォーマットはThunderbirdとは異なりプレーンテキスト形式ではないため、テキストエディタで開いて直接中身を確認することはできません。

また、Windows用の国産メールクライアントソフトとして歴史があり固定ファンも多い(有)リムアーツのBecky! Internet Mailでは、メールの原本データは.bmfファイルにテキスト形式で格納されますが、mbox形式ではありませんし、必ず1つのフォルダに対して1つのファイルというわけではないようです。

このようにメールデータの保存はメールクライアントソフトごとに異なっていることが多く、それぞれのソフトウェアごとに管理の効率化、処理の高速化のための工夫がなされています。

SD

Column オープンソースコミュニティへの参加

Thunderbirdはブラウザの Firefoxや、おもにモバイル機器向けのOSであるFirefox OSといった他のMozillaによるソフトウェアと同様にオープンソースのソフトウェアであり、世界中の多くのコミュニティメンバーの貢献により支えられています。コミュニティメンバーの貢献内容はプログラムの開発はもちろんですが、技術文書の翻訳やバグ報告・テスト支援など多岐にわたっています。これはほかのオープンソースコミュニティでも同様ですね。

筆者の知人で「オープンソースプロジェクトに参加してみたいが、自分にはハードルが高そう」と言っていた若いエンジニアの方がいます。確かにプログラム開発による貢献となるとハードルが高いと感じてしまう方もいると思います。でも、翻訳やバグ報告など自分が貢献できそうな領域を見つけて、そこをきっかけに徐々に使うだけの立場から、作り上げていく立場にもなっていくことはとても楽しいことだと思います。世界中の人が使ってくれますしね。最近では各地で翻訳イベントなども実施されています。オープンソースコミュニティへの参加、ぜひオススメしたいです。

メールクライアントソフトだからできること

S/MIMEを利用する

Webメールと比較してメールクライアントソフトを使うことのメリットの1つに、S/MIME(エスマイム)の利用があります。S/MIMEとはSecure/Multipurpose Internet Mail Extensionsの略でメールの暗号化とデジタル署名に関する標準規格です。S/MIMEを利用することによりメールの機密性・真正性・完全性・否認防止性を実現することができます。

●秘密にしたい情報をセキュアにやりとりする

Webの場合

本誌の読者であればブラウザを使ってWebサイトにアクセスする際に、たとえば個人情報を取り扱うサイトで「このサイトはSSL対応か？」ということを気にされる方が多いのではないかと思います。SSL対応のサイトにより実現できることは、通信の「エンド・ツー・エンドでの暗号化」(機密性)、「接続先のサイトが本当に目的のサイトかどうかの確認」(真正性)などいくつかありますが、メールのやりとりでも同様のことを気にされる方はそこまで多くはないかもしれません。

「エンド・ツー・エンドの暗号化」とは、Webブラウザとwebサーバの間の通信内容を暗号化することを指します。これにより通信経路上の第三者に内容を盗み見られることを防止できます。これは情報がどこの通信経路を通るか保証されないインターネットの世界ではとても重要なことです。

Webメールだと

メールの場合で考えてみましょう。たとえば取引先の会社と重要な機密情報をやりとりする必要があったとします。Webメールを利用して通信内容を完全にエンド・ツー・エンドで暗号化するためには、いろいろと難しい課題があります。

一例を挙げると、通常Webメールでは、受信したメール内の文章や添付ファイルの情報を利用者が読める形にデコード・整形しているのは、Webメールサービスの提供者です。この時点でWebメールサービスの提供者は平文のメールの内容を(原理的には)読むことが可能になり、エンド・ツー・エンドの暗号化は実現できていないことになります。暗号化されたメッセージの内容を復号化するために、自分の秘密鍵をWebメールサービスの提供者に預けておかなければならない、ということになるわけです。

メールクライアントソフトだと

ThunderbirdやOutlookなど主要なメールクライアントソフトの多くはS/MIMEに対応しています。メールクライアントソフトを利用することにより、送信者のメールクライアントソフトで送信時に暗号化されたメッセージを、受信者側のメールクライアントソフトで受信時に復号化するという「エンド・ツー・エンドの暗号化」を実現できます。各利用者の秘密鍵もWebメールのサービス提供者に預けたりする必要がないので、より厳格に機密性・真正性・完全性・否認防止性を実現することが可能になります。

●S/MIME利用時の注意事項

S/MIMEで暗号化、署名の対象になるのはメールの本文(ボディ)部分です。ヘッダ部分は対象にはなりません。つまり、S/MIMEを利用して暗号化してもメールの件名部分(サブジェクト)などは平文のままですので利用時には注意が必要です。

受信者側がS/MIMEに対応したメールクライアントソフトを使用していないと、メッセージの中身を確認できないという点も当然ですが注意事項の1つです。

●S/MIMEの普及について

S/MIMEの利用には、

①利用者が電子証明書を取得(購入)する必要がある
②秘密鍵をなくしてしまうと暗号化されたメールが読めなくなる

など利用面で不便な点もいくつかあり、少なくとも日本においては広く普及しているとは言えません。一方で、他国の事例では個人への電子証明書の普及が進んでいる国もあり、S/MIMEの利用率は国や地域により違いがあるようです。

S/MIMEは、昨今問題となっている成りすましメールによる特定の企業・組織への標的型メール攻撃の有力な防御手段の1つとなります。S/MIME自体は新しくはない技術ですが、日本でも普及してほしいと思います。なお、国内での普及活動は一般財団日本情報経済社会推進協会(JIPDEC)により行われています[注A]。

注A) http://jcan.jipdec.or.jp/smime/

第4章

メールの安全性はどう守るのか

Author 佐藤 潔（さとう きよし） ㈲ジーワークス

本章では、メールの送信者の認証や暗号化、迷惑メール対策について説明します。

なぜ送信者や送信先を偽装できるのか

日本年金機構の個人情報漏洩事件など、最近話題となっている標的型攻撃では、メールが攻撃の起点となっています。このようなウィルスメールや迷惑メールなどでは、送信者を偽装してメールを送付することが多いです。しかしなぜ送信者の偽装が可能なのでしょうか。

実はメール（SMTP）ではもともと、送信者を認証するしくみはありませんでした。そして現時点でも、メールアプリで「差出人」として表示される部分には認証するしくみはないのです。SMTPではSMTPセッション中で渡される送信先と送信者を、envelope-to（エンベロープ受信者）、envelope-from（エンベロープ送信者）と呼び、メールのヘッダ部分に記載されているものをheader-to、header-fromと呼びます。メールクライアントで「宛先」や「差出人」として表示されるのは、このheader-toとheader-fromのものです。しかし、実際に送信される先はenvelope-toにあたるアドレスです。

このenvelope-toとheader-toとは、実はまったく無関係のアドレスを表記することができるのです。そしてチェックもされません。これは「差出人」にあたるheader-fromとenvelope-fromでも同じで、後述のSPF/DKIM以外ではこれらのアドレスがチェックされることはありません。つまり、envelope-to以外はすべてウソの情報が書いてあってもメールを出すことができるのです。

メールの送信認証方法

このように、もともとメールのしくみには送信者を認証するための機構はありませんでした。しかし、ウィルスメールや迷惑メールのように、メールに送信認証がないことを悪用して大量のメールを出す人たちが出てきました。そのため、送信認証を行うことで、そういった悪用ができないしくみが徐々に取り入れられていきました。メールの送信認証がどのように変化してきたか、歴史を追いながら説明をしていきます。

Open relay

Open relayとは、メールサーバがなんの制限もなくメール送信を受け付ける状態になっていることを指します。今、メールアプリからメールを出すと、まず送信メールサーバに指定してある、利用しているISPや自社のメールサーバがメールを受け取り、そこから宛先のメールサーバへと転送されます。しかし、接続に使っていないISPや他社のメールサーバを指定して送ろうとしても、拒否されてしまい送れません。実は昔のメールサーバでは、なんの制限もなくすべての送信者からのメールを受け取り、それが

自分が扱う範囲のメールではない場合は、それを受け取るべきメールサーバに対してメールを再送するという動作をしていました。

しかし、迷惑メールを送る人たちが出てくると、このようなメールサーバに対して迷惑メール送付を押し付けることで、大量のメールを送るようになりました。そこで第1章でも説明があるように、自ネットワークからの接続や後述のSMTP-Authで認証が通った場合だけ、送信を許可するように設定するのが一般的となったのです。

POP before SMTP(PbS)

ホスティングサーバなどでOpen relayをしない設定をすると、なんらかの方法でメールアプリからの送信を接続元IP以外の方法で認証する必要が出てきます。そこで、POP before SMTP(PbS)という方法が使われるようになりました。これは(SMTP接続の)直前にメール受信でPOPの認証をしておくと、その接続元IPからのメール送信を一定時間だけ受け付ける、というものです。

もともとMTA(Mail Transfer Agent)とPOPサーバのようなMRA(Mail Retrieval Agent)とは直接は連携していないため、MRAで認証された接続元IPをなんらかの方法でMTAに参照させてやる必要があります。そのためにdracd[注1]などが利用されています。ただ、次で説明するOP25Bという制限が一般的になったため、今ではPbSが使われることはほとんどなくなっています[注2]。

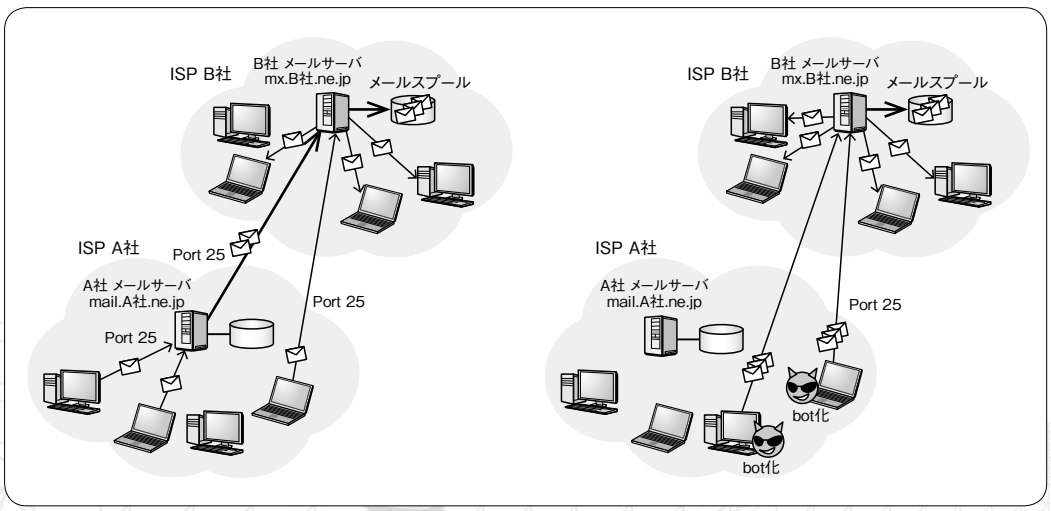
Outbound Port 25 Blocking (OP25B)

Open relayの対策が進んだため、迷惑メールの送信手段が変化し、ウィルスに感染させて遠隔操作可能にしたPC(bot)を利用して、直接送付先のメールサーバにメールを送る、という手法が一般的になりました。これは現在でも迷惑メール送信の主流となっています(**図1**)。

この、botからのメール送信自体を行わせないようにすることで、迷惑メールを出させなくするしくみがOutbound Port 25 Blocking(OP25B)です。botは直接、自ネットワーク外のメールサーバの25番ポートに接続を行い送信しようとします。そこでネットワークの出口で、メールサーバ以外のIPから外部の25番ポートへ接続しようとするものをすべて拒否してしまうこ

注1) http://mail.cc.umanitoba.ca/drac/index.html
注2) OP25Bにより自ネットワーク外へのSMTP接続自体ができなくなるため。

▼図1 通常のメール送信(左)とbotからのメール送信(右)の違い



とで、botからメールサーバへの接続をさせないようにしてしまうわけです(**図2**)。

通常、迷惑メール対策というとメールを受ける側の手法が思い浮かびますが、OP25Bの場合、迷惑メールを出させないことで全体の迷惑メールを減らすという手法であること、サーバではなくネットワークのフィルタで行うものであることが特徴的です。日本ではJEAGという迷惑メール対策を行っていた団体が強力に推進したため、ほとんどのISPでこのフィルタが行われるようになり、日本発の迷惑メールが大幅に減ることになりました[注3]。

Submissionポートと SMTP-Auth

OP25Bの制限があると、今までPbSにより可能だったホスティングサーバや外から自社メールサーバを使ったメール送信が利用できなくなってしまいます。そこで、Submissionポートと呼ばれるメールアプリからのメール送信を受け付ける専用のポート(587番ポート)が使われるようになりました。

Submissionポートから送信する際、ユーザ認証のない通常のSMTPではまた迷惑メール送信に悪用されてしまうので、送信に認証が必要なSMTP-Authというプロトコルが利用されるのが一般的となりました。

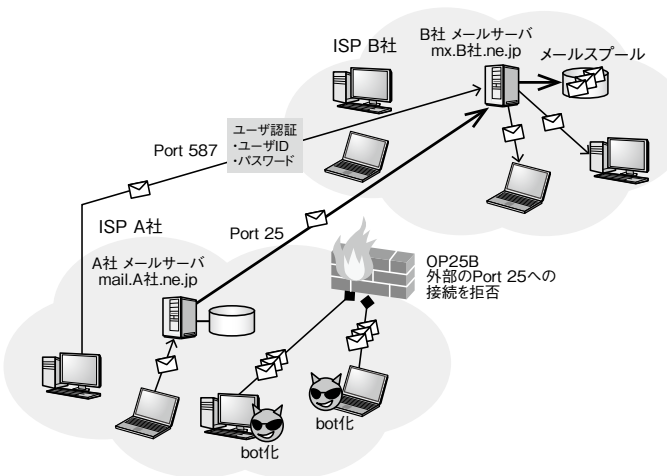
SPFとDKIM

SubmissionポートとSMTP-Authにより、自ユーザからの送信者認証は行えるようになりました。しかし最初に説明した、送信者のメールアドレスを偽装できてしまうという問題は解決されていません。ただ、送信者のメールアドレスの「ドメインが正しいか」を判定するしくみは作られました。それがSPF(Sender Policy Framework)と DKIM(DomainKeys Identified Mail)です。SPFやDKIMは、送信者として書かれているメールアドレスのドメイン部分から、そのドメインのDNSのTXTレコードなどを参照して、正規のメールサーバから出されたものかを判定できるようにするものです。

SPFは、DNSのTXTレコード(もしくはSPFレコード)に規定の書式にしたがって、MTAのIPを列挙します。この書式では、このドメインが送信者になっているメールはこのIPから出さ

注3) ただ現在は、このOP25Bを乗り越えるSubmissionスパムと呼ばれる手法で迷惑メールが出されることも起きている。

▼図2 OP25Bによりbotからのメール送信を防ぐ



れる、ということが示されています(**図3**)。

DKIMは、公開鍵暗号方式を用いてメール本文から署名を作り、メールヘッダに追加されます。受け取った側は送付元ドメインのDNSのTXTレコードから公開鍵を取得して、メールヘッダの署名が正しいかを判定します。

SPFは転送が行われる場合やエラーメールを返す場合に、DKIMはメーリングリストのように内容の一部が変更される場合に、問題が起きます。そのため、SPFとDKIMどちらも対応しておき、どちらか一方が通った場合には認証が通ったとするのがお勧めです。そのような柔軟な条件でSPFやDKIMの検証を行うには、milter manager[注4]やENMAなどのツールを利用するのが良いでしょう。

メールの暗号化

次にメールの暗号化はどのような手法があるのか、そしてその問題点を紹介します。

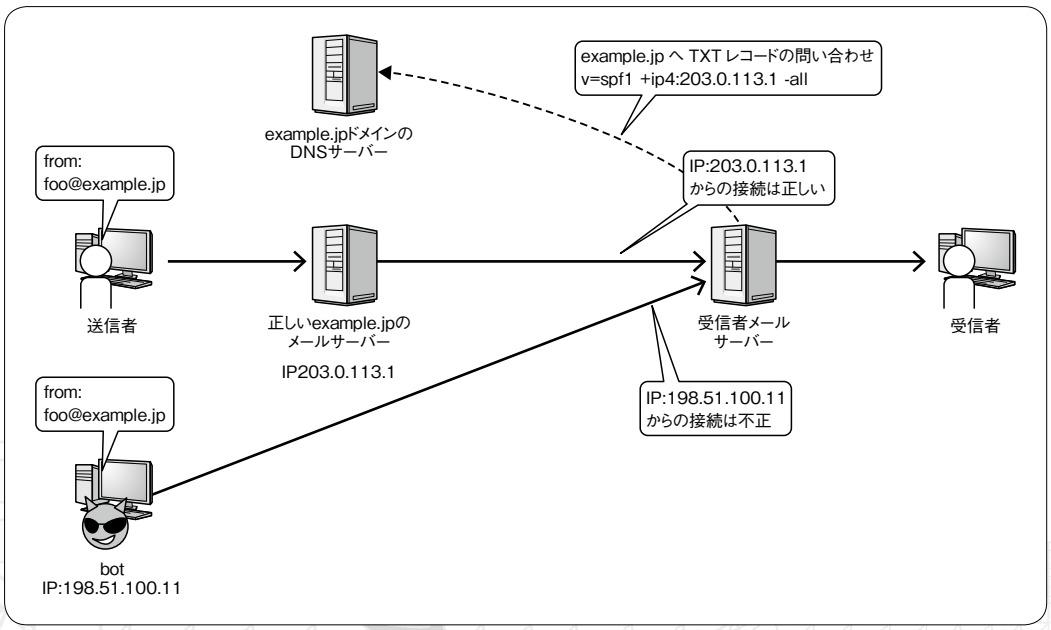
注4) http://milter-manager.sourceforge.net/index.html.ja

送信認証についてもそうでしたが、SMTPではもともと暗号化などは考慮されていませんでした。そのため、通信内容は基本的にすべて平文で流れるようになっていますし、POPのユーザ認証時などのパスワードでも平文で通信されます。なので、経路上でパケットキャプチャできる環境があると、そこでメールの内容やパスワードを盗み見ることができてしまいます。また通常、メールスプールに置いてあるファイルは暗号化されていないことが多く、メールスプールのメールを覗かれる危険性もあります。

SMTP over SSL/TLS(SMTPs)

SMTP over SSL/TLS(SMTPs)は、SSLまたはTLSで暗号化した通信経路上でSMTPの通信を行うというものです。さらに、最初からSSL/TLSでの接続を行うものと、最初はSMTPで接続してからSTARTTLSコマンドにより、SSL/TLSの通信に移行するものとがあります。またSMTPだけではなく、メール受信を暗号化して行うためのPOP over SSL/TLS(POPs)やIMAP4 over SSL/TLS(IMAP4s)

▼図3 SPFで送信元ドメインのチェックを行う



もあります。

ですが、これらの接続の暗号化が保証されるのは、あくまで自分のメールアプリとメールサーバ（MTAやMRA）との間の暗号化に過ぎません（**図4**）。送付元と送付先のメールサーバ間が暗号化通信しており、受信者もPOPsなどで受信していればすべての経路が暗号化されますが、通常は知るすべがありませんから、ちゃんと暗号化のことを考えるなら、ここだけ暗号化してもあまり意味がないでしょう。ただ、公衆Wi-Fiを利用して接続する場合など、メールサーバまでの経路の暗号化に意味がある場合は積極的に利用すべきです[注5]。

PGP（GPG）とS/MIME

SMTPsはあくまで通信経路の暗号化で、先の経路のことまでは保証されないため、限定的にしか暗号化の効果がありませんでした。そこでPGP（GPG）やS/MIMEでは、メールアプリで暗号化や署名をして、受け取った相手のメールアプリで暗号化を解くことで、完全な暗号化を実現しています。

PGPは公開鍵暗号方式を用いた暗号化ツールで、GPGはGnuPGとも呼ばれるPGPのGPL版のものです。PGPは商用利用は有料ですが、GPGはフリーソフトウェアなので無料で利用できます。S/MIMEも電子証明書を用いて暗号化や署名を行うしくみですが、SSLのように認証局から証明書を発行してもらう必要があります。詳しくは第3章のコラムを参照してください。

GPGやS/MIMEを利用すれば、完全な暗号化や送信者の認証を行うことができますが、これらを利用できる人がほとんどいない、というのが一番の問題となります。

暗号化zipファイルとパスワード問題

GPGのような完全な暗号化ではなくても、暗号化を行いたい場合は多くあります。そのようなときよく使われるのが、暗号化したzipファイルやパスワード付きのExcelファイルを添付して、別メールでパスワードを送る、というものです。

ですが、パケットキャプチャなどでメールが覗かれうる状況ならば、パスワードの書かれたメールも覗かれますから、この方法ではほぼ意味がないということがわかると思います。

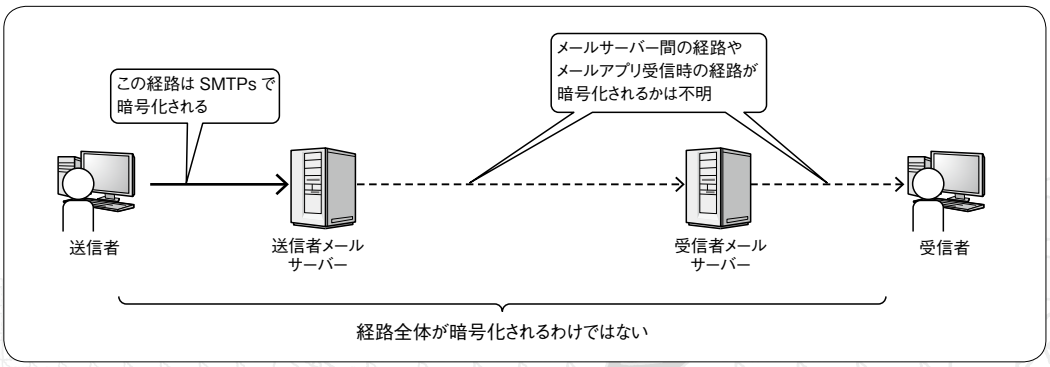
このように見ると、メールの暗号化には残念ながらこれといった解決法がないのが現状と言えるでしょう。

迷惑メールの対策

迷惑メールは2000年代後半をピークに徐々に減りつつありますが、それでもまだ大量の迷惑メールが出されています。本章の最後に、迷惑メール対策の代表的な手法と、利用方法を簡単

注5） 相手がGmailなどSMTPs対応であることがわかっている場合なども意味がある。

▼図4　SMTPsで暗号化される範囲



に紹介します。

迷惑メールの送信手法

メールの送信認証方法で説明したように、SMTPには送信者の認証機構がないため、送信者アドレスはまったくあてになりません。また、OP25Bの解説でも説明しましたが、多くの迷惑メールはbotを利用して出されているので、送信元の個々のbotを特定することはできても、遠隔操作している元にたどり着くのは困難です。そのため、真の迷惑メール送信者を捕まえることは難しくなっています。

ただ、botを用いて大量のメールを送るための特徴やクセがあり、それを使って迷惑メールを判定することができます。

DNSBL

DNSBL(DNS Blacklist)は RBL(Real-time Blackhole List)とも呼ばれ、迷惑メールを送付してきたIPアドレスのブラックリストを、DNSを用いて共有するサービスです。迷惑メールを送られたユーザからの通報や、ハニーポットという迷惑メールを受信するために仕掛けられた罠のメールアドレスに届いたメールなどから、迷惑メール送信元のIPアドレスをブラックリストに登録します。

利用者は、DNSBL提供元のDNSに対しメール送信元IPを問い合わせると、結果に応じて127.0.0.1などの値が返ってくるため、それでブラックリストに載っているか判断します。Spamhaus[注6]などのサービスが有名です。

正しいメールサーバでも、ウィルスなどからそれを使って迷惑メールを出されるとDNSBLに登録されてしまうため、その誤検出が問題になることも意外に多いです。ホスティングサービスなどで近くのIPアドレスのメールサーバから迷惑メールを出されることで、巻き添えでブラックリストに入れられてしまうこともあります。そのため、DNSBLだけの判定で迷惑メールと判定せず、複数の基準で総合して判定するシステムにすることをお勧めします。また、DNSBLのサービスが終了してしまうと、それが原因で誤検出を起こすことがありますので、利用しているDNSBLサービスが継続しているか注意しておく必要があります。

PostfixでDNSBLを利用するには、postscreenを利用するか、milter managerを使い、ほかの指標と組み合わせて利用するのが良いでしょう。

botからのメールが増えたため、DNSBLは結果的に動的IPのアドレス帯をブラックリストにすることと意味的に近くなります。そのような動的IPフィルタとしてはS25R[注7]という手法があります。

greylisting

botからのメールは短時間に大量のメールを送ることが目的であり、到達性は求められません。SMTPでは、メールサーバがなんらかの理由により一時的にメールが受け取れない場合、再送要求を出すことができるようになっています。再送要求を受け取ると、通常のメールサーバでは一定時間後に再送を行いますが、botでは再送せずにそのままあきらめてしまうものがほとんどなのです。greylistingは、わざといったんメールを「一時拒否」と返答し、きちんと再送されたものだけ受け取ることで迷惑メールをフィルタします。

Postfixでgreylistingを導入する場合、postgrey[注8]やmilter-greylist[注9]がよく使われます。

このようなSMTPセッションレベルでのクセを利用したフィルタはほかにも、tarpittingという返答の遅延を利用したものや、ウソの返答をしてその反応を見るもの、などいろいろなものがあります。

greylistingでは初めて受け取るメールサーバ

注6) https://www.spamhaus.org/

注7) http://gabacho.reto.jp/anti-spam/
注8) http://postgrey.schweikert.ch/
注9) http://hcpnet.free.fr/milter-greylist/

からのメールが遅延してしまうことや、まれに再送要求に応じない「正しい」メールサーバからのメールが受け取れないことがあります[注10]。tarpittingを使って遅延や誤検出を減らしたtaRgrey[注11]という手法もあります。

ベイジアンフィルタ

ベイジアンフィルタとは迷惑メール中に含まれる語句を、ベイズ学習という機械学習手法により学習させ、迷惑メールの判定を行うものです。ベイジアンフィルタでは、まず最初にどれが迷惑メールなのかを学習させる必要があります[注12]。届いたメールの内容を単語に分かち書きし、迷惑メールと判定されたメールの中に、たとえば「出会い」という単語が入っていた場合、「出会い」と入っているメールが迷惑メールだった確率、つまり単語ごとの迷惑メール確率を算出していきます。判定させる場合は、メールに含まれているすべての単語に、この単語ごとの迷惑メール確率を積み上げていき、閾値以上になったときに迷惑メールと判定します。

メールアプリでの迷惑メール判定でも多く使われており、Thunderbirdの迷惑メールフィルタやPOPFile[注13]などで使われています。サーバで利用する場合、SpamAssassin[注14]というよく使われているサーバ側の統合型迷惑メールフィルタでもベイジアンフィルタを利用することができますので、これを利用するのが良いでしょう。機械学習の前処理として分かち書きが必要となるため、言語によりローカライズが必要になります。SpamAssassinの日本語ローカライズはSpamAssassin日本語対応パッチ[注15]を利用します。SD

注10）Webの登録ページから登録確認メールなどをメールサーバを経由せずに直接送ろうとしている場合などに多い。
注11）http://k2net.hakuba.jp/targrey/ 筆者が考案したもの。
注12）ブレ学習されていても日本語圏でないところで学習されたデータは精度があまり良くないことも。
注13）http://getpopfile.org/docs/jp
注14）http://spamassassin.apache.org/
注15）http://emaillab.jp/spamassassin/ja-patch/

特別企画

なぜ俺の提案は通らないのか？

冷静と情熱の間には、お金の川が流れている

Author 土居 昭夫（どい あきお） ㈱NTT PCコミュニケーションズ

会社で進められているさまざまなプロジェクト。「なぜ、あの案件が決まってしまったのだろうか、俺のほうが優れているのに」と忸怩（じくじ）たる思いをしたことはありませんか。その決定の根拠は何でしょうか。答えはコスト＝お金です。しかし企業では、単に儲かる・損をする、という単純なモノサシで決定されているのではありません。そこにいたる共通解があるのです。本記事は、その気がつきにくいポイントを明らかにしていき、読者の皆さんの気持ちを晴らし、より実践的なエンジニアになる手がかりを共有したいと思います。

はじめに

本誌の読者は若手と中堅が多いと聞いています。立場はおそらく違いますが、自分の提案に対して周囲の共感を得ることが難しいなと感じていませんか。その困難は、両世代に共通していると思います。技術的な興味を引く説明はできても、それが自社にとってどのような影響を及ぼすのか……、事業の観点から説明するのが苦手――そういう方が多くないですか。

本記事では、事業での数字の扱い方をできる限りシンプルに解説します。そして会議などの提案資料の裏付けの1つとして「皆がわかる数字」を出せるようになることを目的としています。なお、本記事はDevelopers Summit 冬2015 20-E-1での講演および資料を元にしています。同Summitでも注意点として挙げましたが、本記事の内容をそのまま提案資料に活かすのは避けてください。なぜなら数字の費用計上のルールや、減価償却率の適応、固定資産への組み入れ方や除却の方針、そのほかについて、細かい部分での取り扱いは会社によって違いがあるからです。本記事を手がかりに、会計担当の方に確認を取ってから実践してください。

私が伝えたいこと

若い開発者に、わかりやすくお金の話を伝えたかったというのが最初の思いです。

会社にまつわるお金というのは、実は思っている以上にロジカルに成り立っています。麻生元総理曰く「帳簿っていうのを見れば、まず借り方と貸し方と、2つがあるでしょ、（中略）お金を100借りていれば、必ず、100貸している人がいないとおかしい。帳簿って言うのは左と右が必ずそろうことになってますから」――つまり入ってきたら必ず出ていく何かがあります。モノか、違う種類のお金に変わるだけなのです。

「帳尻が合う」の「帳」は帳簿の「帳」であり、項目を並べた最終行の合算値が貸方／借方でぴったり合うことなのです。

今後皆さんがどのような立場になろうが、お金の話は常に近くにあります。生きていくための手段は多いほうが良いですし、誰かに何かを伝えるための表現方法は手が多いほうが目的を達しやすいのです。また、以前とある会合で㈱さくらインターネットの田中社長のお話で印象に残ったものがあります。それは「事業には、作り手（技術）、売り手（営業・企画）、支え手（サポート）のバランスが大事」

ということです。これらのうち、どれかが突出しても、誰かが不幸になる構造になってしまい、その結果ユーザを幸せにできない、というものでした。筆者はこれら3つの関係性においてプロトコル(共有している概念)は、「モノ」と「お金」ではないかと気がつきました。モノがなくてはお金は入ってきませんが、入って残るような構造を作り込むには、3つの役割の方々が同じ感覚で説明できなくてはなりません。たとえば売り手が原価を度外視して「売れるから」という理由で売価を作ると、事業はうまくいきません。また、支え手が「お客様に完璧な体制を作りたい」と採算を度外視して投資しても、事業はうまく立ち行きません。もちろん「マインド」とか「プライド」とか「ビジョン」が共有できているのは大前提です。

でもお金はどうでしょうか。皆さん、会社に関する金銭感覚を共感できているのでしょうか?

そして、世の中には「原価○○円だから、自分でやればもっと安くできる」という主張もあります。これは「一見わかってる風」ですが、あまりにも浅薄な話です。こんな話をネタ以外でするのは少し格好悪いです。だいたい自分の仕事を勝手に値踏みされて外野に言われるのって気分は良くありません。ですから技術者には、そうした雑な議論をしてほしくないのです。

このような思いもあってDevelopers Summitでお話しさせていただいたわけですが、ひょんなことからこのような執筆の機会をもらいました。本記事が何かのためになれば幸いです。

なお、本稿の内容は個人の見解であり、その責任はすべて筆者個人に帰するもので、所属する組織を代表する意見ではありません。

商売で使う数字の基本

「ビジネス」というと堅苦しいので「商売」とここでは言います。しくみはいたってシンプルです。

仕入れ、加工し、宣伝し、販売する。販売後のケアも商品によっては含まれるでしょう(**図1**)。

ここで、やっていることは単純な足し算と引き算ですので「難しい」ということありません。**図1**にある費用には変動費と固定費の2種類があります。簡単に言うと、変動費は"顧客の伸びや売れ行きで変動する費用"のことで、固定費は、"売り上げにかかわらず運営していくこと自体でかかる費用"のことです。アカウントごとにライセンス費が出るならそれは変動費、設備の償却費やリース料、人件費は固定費、というとわかりやすいでしょうか。

現金の流れを押さえていますか?

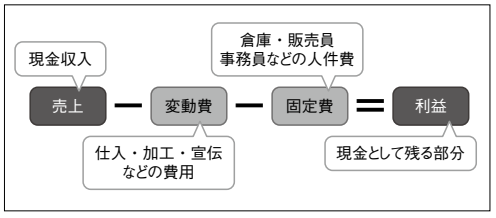
さて、事業の観点から見た場合、行為に紐づくキャッシュ(現金)の流れがとても重要です。

現金は各種の費用として日々流れていきます。実際のところ「ダラダラと流れている」というよりは、その多くに「締め日」があり、「決済日」があり、と慣例に従った日付があるので、特定のある日に膨大な決済をしていることと思います。

利益をきちんと確保した事業設計になっていない場合、いずれは資金が枯渇し事業運営に支障が出ます。極端な話をすると、「受注が突然大量に入ってきたけど、お金がないので製造できず納品できない」——みたいなことも起こり得るわけです。「うれしい悲鳴」とか言ってる場合ではありません。信用問題になりかねません。

そして、年度末には法人税などの納付がありますし、さらには毎月従業員の人件費や厚

▼図1　ビジネス(商売)のモデル図



生年金、住民税の支払い(会社が給与から差し引き自治体に納めています)など、出て行くお金はたくさんあります。ですので、筆者がいた会社では、仮に無収入になったとしても全従業員の2年分程度の人件費を支払える程度の現金が常にありました(このあたりの方針は企業によりまちまちです)。

個人と法人の違い

ここからほんの少し難しいのですが、個人と法人では大きく違うのが財務会計管理の方法です。たとえば個人であれば、家計簿などで現金の出入りを整理すれば、自ずとキャッシュフローが見えてきます。大半の方はこのくらいで十分、となりますが、法人の場合は財務の観点から、もっと精緻(せいち)にお金の動きを追います。これには資産や負債も含まれます。

なぜ法人がそうなのか、というと**税金の枠組みが違う**から、というのがざっくりとした回答になります(**図2**)。

費用と投資の違い、減価償却とは何か?

またもう1つ大事なことがあります。法人ではお金の使い道に「費用」、「投資」の2つの区別があります。皆さんも聞いたことがあるかもしれません。

会社内で何かしらの提案をするにあたり、これらを混同すると危険です。まず話がとおりません。法人においては大きく意味が違いますので、軽く整理しておきます。なお、ここで言う「投資」は有価証券取引などでイメージする「投資」とはニュアンスが違いますので気を付けてください。サーバやストレージ、ソフトウェアの開発委託などをおもに意味しています。

表1の中で「減価償却」という言葉が出てきましたが、世の中にあるさまざまな「モノ」について、ある種のモノには国税庁が定める耐用年数があり、買ったものはその年数に従って資産価値が減じていきます。そのため、ある年に買ったらそれでおしまい、というわけではありません。現金としてはその年に決済されますが、資産としては数年間残り続けることになります。そこで事業収支という観点では、直接の現金の支払いとは分けて考えねばなりません。このような要素が絡み合って、実態のキャッシュフローと、事業収支、というものが見えてくるわけです。

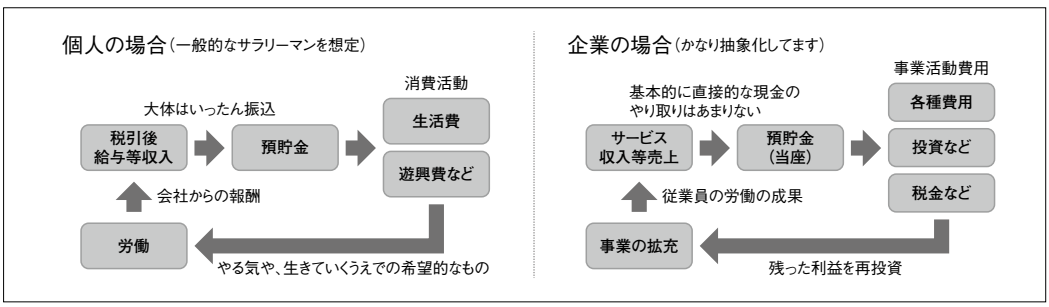
俺の提案はなぜ通らないのか

提案に限らないのかもしれませんが、ある

▼表1　費用と投資の表

	概要	例
費用	一般的な役務での人件費や小額の物品の購入など。サービス利用料、ライセンス料	・アルバイト、派遣社員の人件費(役務によります) ・事務用品、タブレット ・クラウドサービス利用料 ・データセンタ利用料 など
投資	高額な物品であり、耐用年数が決められており、減価償却の対象となる資産の購入	・ネットワークスイッチ ・サーバ ・ストレージ ・サーバラック(自社調達) ・自社開発ソフトウェア など

▼図2　お金の流れの個人と法人



Column クラウドでコストダウンと安易に言うなかれ

さまざまな広告で「クラウドにすればコストダウンできる」と安易に書かれていますが、実際のところそんなに簡単な話ではありません。

この言葉にはユーザの背景はいっさい考慮されておらず、今までやってきたオンプレミスで構築するプロセスを「単純に置き換えているシーンで安い」というのが実態とのギャップではないかと思います。

それを使うには、使いこなすまでのスタッフの教育コスト、システムを変更するためのスイッチングコスト(運用体制の切り替えや、ソフトウェアの更改などの変更にかかる費用)、また、既存システムを並行して運用するとなれば、移行完了までの期間は追加でクラウドのコストがかかります。それらを積算してイニシャルコストがどれだけかかり、いったい何年後からコスト効果が得られるのか、そういった部分を検討しなくては安易な結論は出せないのです。また、スイッチさせる場合は単にスイッチさせるのか、はたまたリニューアルして切り替えるのか、こういったことでもコストに差がでます。もっとも大きな罠は転送量による従量課金かもしれません。

このような論調でイノベーティブな技術が広がっていくのは、筆者にとって切ないものがありますが、これもまた本記事にあるような提案の難しさが要因になっていると思うのです。数文字のキャッチコピーで決済権者に直撃したい、となれば悲しいかな「コストダウン」が現在ではもっとも有効なワードなのかもしれません。

クラウド利用で得られるスピードや、柔軟性、ロバスト(堅牢)性を推してほしいと願ってやみません。

提案や議論がまとまらないのは、おおむねコミュニケーション上のズレに対して、意見が摺り合わさっていないことが原因ではないでしょうか。皆さんは、自分なりに課題を設定し、それを解決するために正しいことを進めています。しかし提案を受け承認を出す側は、皆さんと同じような課題意識を持っているのでしょうか? 現場で活動する皆さんと、承認者ではそもそもロール(役割)が違います。まずはそこを考えるのがいいと思います。

経営側と現場では優先度が違う

説明をわかりやすくするために、承認者は経営サイドの方(部長以上)とします。経営者は立場上事象を細かくとらえることをしません。また、提案にある技術には明るくないこともよくあります。それでも役割がそもそも違うので、それはそれでいいのです。

彼らは事業を通して企業を運営することに責任を負っていますので、提案がその観点から有用かどうかだけを判断します。もしかすると社会的に企業の価値を上げると思えば、収支が赤字だとしても短期的であればいいと割り切る——そういうこともたまにありますがおおよそイレギュラーな事例です。

たとえばOSSに対するドネーションやイベントの主催、スポンサード、このあたりはすべて費用です。しかし費用対効果を定量的に測ることは不可能です。ですが、その会社は周囲の方から感謝されますし、企業価値の向上の観点としては有用です。端的には費用が出て行っただけに見えますが、このようなお金の使い方をする会社は確実にあります。とは言え、これはとても難しいことで、実は筆者もうまくできたことがありません。費用対効果が測れない提案というのは非常に難易度が高いと感じています(**表2**)。

話はそれますが、筆者としてはこれからそういう活動がエンジニアに対する理解の深さを示す指標になり、また間接的なインセンティ

▼表2 経営者、管理者、現場エンジニアで優先順位が異なる(一例は筆者の個人的見解)

経営者、管理者	現場エンジニア
ビジネスプラン	安全性、可用性、機能性
話題性	プロダクト
プロダクト	ビジネスプラン

ブにもなり、そういった活動をする企業にエンジニアが集まるようになるのではないか、そのように予感しています。

基本的には収支が合っているかどうか、他社に比べて優位かどうか、そのあたりが大きな関心事です。経営側が技術そのものに造詣が深いのであれば、それは幸せなことかもしれませんが、それとこれとは判断軸が別ですので、主観的な期待はしないほうがいいでしょう。

事業規模の違いをとらえていますか?

提案内容がどのような事業モデルなのか、についても大きな違いがあります。SIなどの短期的な事業モデルなのか、Webサービスなど、長期化する事業モデルなのか。長期モデルに関しては設備更改のタイミングや、利用率における収支の観点が必要です(**表3**)。

たとえば、1,000万円の投資(開発で500万円、設備で500万円)、月々に50万円の運営費、1,000円／月のサブスクリションを販売という条件の事業モデルがあるとします。この当初設備では、1,000サブスクリプションが性能を担保する上での限界だとします。なお、シンプルにしたいので人件費は含みません。

この場合、いったいどれだけのサブスクリプションを販売すれば良いのでしょうか?はたまた、1,000円／月／サブスクリプション(契約)でいいのでしょうか?

これら2つを解決するには、シミュレーションを繰り返しながら売値を決めていく、というアプローチと、市場の価格感をベースに売値をまず決め、そこから導きだすアプローチがあります。

筆者の経験は前者が基本ですが、もちろん市場の価格感は根底に置く必要がありました。競合のある業界では、価格感のズレは商売の困難さと直結します。

緻密な作業が必要になってくる部分ですし、実際には売れ行きは予想とは異なるのが常ですが、ここで決めていった流れそのものがその後の軸になることですので、この作業は外せないプロセスです。前者でのシミュレーションをしてみましょう。

表4の内容はあえて非常にシンプルにしています。表中の項目は考慮する内容として重要な要素として挙げておきます。これらをベースにシミュレーションしてみましょう。

四半期ごと100ずつユーザが増え、リニアに10ヵ月目にキャパシティの限界を迎えます。ここで追加投資をせず、そのままユーザも増えないとしました。その結果、2年目のQ3でブレークイーブンを迎え、その後順調に利益が推移し、4年目Q2で累損を解消します(**図3**)。これまでの費用投下をカバーし、さらに純粋に利益を出すようになりました。

——少し待ってください。これはいくらなんでもユーザの伸びが理想的すぎるのではないでしょうか。予測どおりに売れないとか、解約のリスクをいっさい加味していません。というわけでもっと消極的に考え直しましょう。

全体のキャパシティに対して50%の利用率に到達して、それを維持した設定で再計算してみたところ、永遠に黒字化しない結果になっ

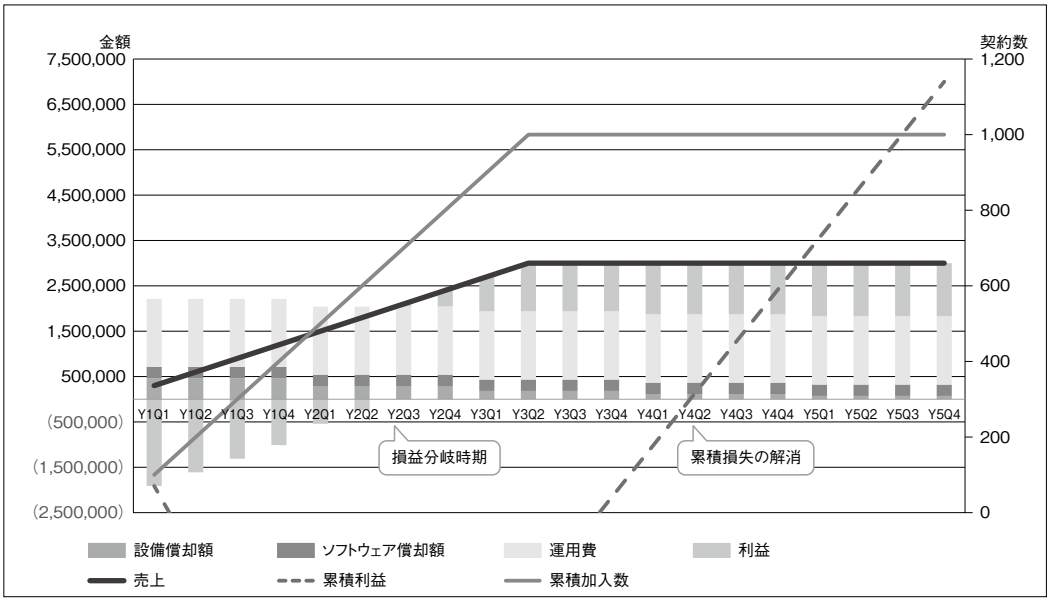
▼表3　収支予測で意識する諸要素

収入面	固定費用面	変動費用面
契約数	有形固定資産額	運用保守費用
平均顧客単価	無形固定資産額	広告宣伝費
キャパシティ	ライセンス	その他販売管理費
平均利用期間	データセンタ	回線利用料
広告の効果	その他販売管理費	

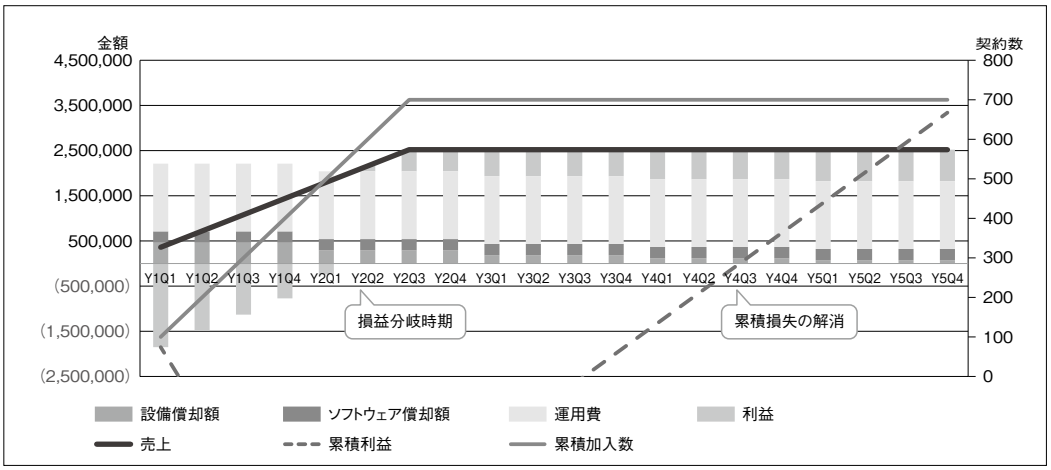
▼表4　サンプルモデルの条件(とあるITサービスついてこのような条件であるとします)

設備投資	500万円 償却法：定率法36.9%
開発したソフト	500万円 償却法：定額法5年間
運営費	50万円
キャパシティ	1,000契約／設備
販売単価	1,000円／月／契約

▼図3　1回目のシミュレーション



▼図4　2回目のシミュレーション（利用率70%、価格1,200円）



てしまいました。少々消極的過ぎたたようです。そこで70%までは到達できるとしてみたところ、2年目のQ3で**ブレークイーブン**に達したのち、ゆるやかな利益の蓄積により累損は解消されていきますが、5年のスパンでは解消しきれず、設備更改を考慮するとかなり厳しい結果といえます。投資に対して回収がしきれず、ズルズルと現金を吐き出し続けることになるでしょう。

そこで価格を1,000円から1,200円としたところ、2年目Q2でブレークイーブンを迎え、4年目Q3で累損解消となりました(**図4**)。価格の調整は難しい部分はありますが、リリース前ならなんとかなるかもしれません。

ブレークイーブンするポイントとは何か?

このようなシミュレーションから営業目標や稼働目標が導きだされ、投資計画が作られ

Column P/LとB/Sとは何か

P/L（Profit and Loss＝損益計算書）は収支を見るために作られるもので、売上に対しての費用・税・利益が明らかになります。

一方、B/S（Balance Sheet＝貸借対照表）は資産のバランスを見るために作られ、現金や現金同等物、債権などと、かつてそれらが交換された固定資産、債務などのバランスを表現したものです。これによって会社の現状の体力や資産状況を見ることができます。

本記事ではどちらかというとP/Lよりの話になりますし、大体現場ではこちらで考えるほうが話を進めやすくなります。

かつての職場であった興味深い議論はこういったものでした。あるサービスの設備を設計し、さて見積りとなった段階での出来事です。

筆者　「リースがいいのか、買ったほうがいいのか」

役員A「P/Lで見たときに投資（買う）すると初年度赤字スタートするのが嫌なので、リース[注a]でいこう」

役員B「リースにしたらB/Sには債務に載る。しかもトータルで支払いが増えるからだめだ。金ならあるよ！」

筆者　「どっちでもいいですよ……でも不必要な支払いはしないほうがいいから、買ったほうが気持ちはいいですね」

このときは、結局購入したんだったかな……忘れちゃいました。リースと購入は会社のキャッシュフローの状態によって判断すると思います。個人でいえば、ローンで買うか、現金一括で買うのか、という選択によく似ています。

さて、サーバやストレージは高価ですし、機能の陳腐化、費用対効果の向上が速いものです。利用するシーンにおいてそれを3年以上使うのかどうかが判断の目安です。購入した場合は減価償却費として、購入価格に対して一定の率がかかりながら価値が下落していきます。一定の率で償却されていきますので購入当年度がもっとも償却費が大きく、5〜6年かけてだんだんと低減していき、最終的には購入価格の5%程度まで落ちていきます。

一方、リースを活用した場合は、最初から一定の支払額になりますので、当初のコストは低いのです。しかし、投資をして購入した固定資産は、年々価値が減じていくことで、償却期間のだいたい半分でリースよりも有利な金額になります。3年後にある程度の売上規模があった場合、利益率はリースした場合よりもいいものになります（**図A**）。なお、リースについても契約期間が完了して再リースとなれば、費用がガクッと下がることもあります。

読者の皆さんは、**表A**に示すようなメリット、デメリットをふまえてより良い判断を心がけてください。

注a） ファイナンスリースです（リース会社による代理購入と考えてください）。

ることになります。このあたりは機器の購入タイミングを最適にするために行っている運用や予測がそのまま利用できます。

ソーシャルゲームなどのフリーミアムな世界では、もっとシビアかつ精緻なシミュレーションを行う必要がありますが、その話は専門の方にお任せすることにします。

さて、「ブレークイーブン」という言葉が出ましたが、これは「**損益分岐点**」を意味します（**図5**）。

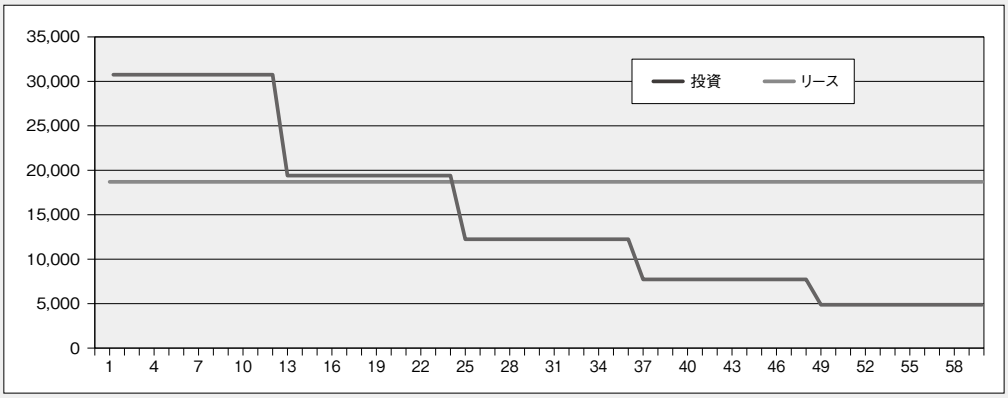
シミュレートを抽象化することでこのような図を作れますが、これによって「いくらで売ればいいのか」という指標が明らかになります。この指標を頭の中で計算してある程度作り出せるようになると、いろいろと提案することが楽になります。

経営側に届く提案をしていますか?

会社への提案というものは自社の成長において、さまざまな「仮説」が達成され、利益を産み、さらなる成長・安定の大きな因子になるという「有用な提案」でなければなりません。

つまるところ、単に「これがやりたい」「かっこいい」「○○社でも導入している」だけでは会

▼図A　購入・リース5年間の差
（60ヵ月、減価償却率36.9%、リース料率1.87%で計算。縦軸は月ごとの支払額）



▼表A　リース、投資のメリット／デメリット

	メリット	デメリット
リース	・初期投資が少なく、支払いが一定 ・損益分岐が早い ・いらなくなったら返却できる	・支払総額が大きくなる ・途中でやめても残存契約期間分が違約金として発生するので、キャッシュアウトが発生する ・改造などは契約上難しい場合が一般的
投資	・減価償却されるので徐々に原価が下がる ・支払い総額に特別な上乗せはない ・基本的には自分のものなので、好きにしていい	・初期投資が大きい ・税金がかかる ・処分する責任がある

※大きな違いでは所有権の有無が挙げられるが、メリットともデメリットとも言えるので外している。
※リース会社との契約にもよるのであくまでも目安である。

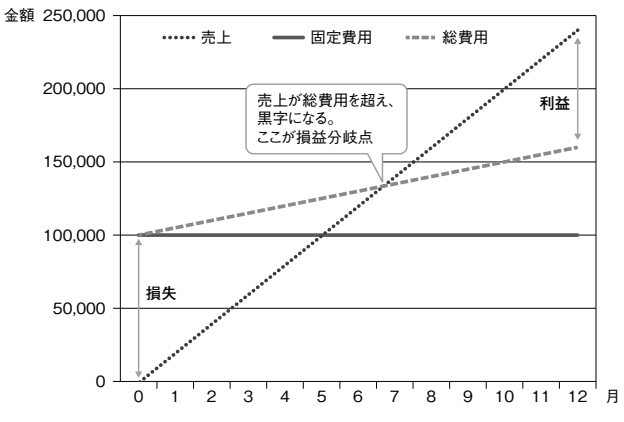
▼図5　損益分岐点の例

●損益分岐点の例

売値
1,000 円／契約
月ごとに 10 売れる

固定費
10 万円／月

変動費
4 契約ごとに
1,000 円



Column

講演時に心に残った質問

本記事では事業性の強い提案を想定して書きました。しかし提案には、このほかにも業務の効率化など、別の性格のものもあります。Developers Summitでも次のような質問がありました。

「社内のシステムを更改したい、その場合はどう説明すれば良いか」

――社内システムの場合は、基本的にはそもそも費用ですので、コストカットの側面で攻めるのが常道です。ですから何が効率化されるのか、具体的には、新システムへの更改により業務プロセスが改善されることで、コスト系人員を売上系人員に配置転換できる、とか、そうしたロジックが有効でしょう。単純なコストカットよりも、さらに売上寄与までする、となればかなり良い案になると思います。

「社員に資格や検定を受けさせたいのですが、有料のものはなかなか通しにくい。どうすれば良いか」

――そうしたケースについては本記事を応用し、その費用・投資が自社においてどのような結果をもたらすかを定量的に説明する、あるいは世の中の流れを外部の権威を用いつつ説明するといった作戦が取れます。投資を行って長期の利益を取る、というモデルであれば話もしやすいのですが、純粋に費用が流れる話だと少し難易度が高いかもしれません。パートナーになるために有資格者が○○人必要、とするとやりやすいですね。

しかし提案というアクションの肝は、相手の立場と達成すべき事柄は何なのかに尽きるので、めげずに進めてください。

社の上層部には「思い」が届きません。

「届く」提案におけるポイントをまとめるとこう言えます。

“**どう**”やるか、”**いつ**”やるか。”**勝算**”の根拠は何か。それらを個別にまとめていくと次のようになります。

- **「どう」：利用する技術は何か、販売方法とそれに即した広告法や販売促進策は何か、戦略・戦術**
- **「いつ」：リリースを含めた各行動の実施スケジュール**
- **「勝算」：競合他社との有利差の比較（販売企画や利用技術において）、トレンド、市場を調べ尽くしたか**

本記事では事業企画をベースに説明をしてきましたが、このような種類の提案だけでなく、単純なソフトウェア開発においても同じことです。何かの機能を追加しようと考えたとき、その「機能」というのははいったい誰のためなのか、何のためなのか、どのくらいの期間でリリースできるのか、そういうところから話はスタートします。事業にまつわる話も本質的には一緒です。お金の流れを踏まえた、承認者にとって納得のいくストーリーが必要なのです。

そしてお金の話は、日々の生活における自らの決断、家族への説得（家庭内稟議、なんていう言葉もありますね）、はたまた業務上の些細な決済ひとつひとつに必ずあることです。そういった身近なことからお金の流れを感じることを行ってみてはいかがでしょうか。

この記事で興味が湧きましたら、簿記検定を受験してみることもスキル向上に役立ちます。簿記というのは技術者も含めて、さまざまな立場で一生使える知識・技能といっても過言ではありません。

これまで解説してきた「**お金の話**」をふまえ、上記ポイントを軸にストーリーを作り、「**会社にとってプラス**」な提案を行って、決済者が「**共感**」してくれれば、きっと「**俺の提案も通る**」ようになるでしょう。SD

バックナンバーのお知らせ　BACKNUMBER

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

本誌バックナンバー（紙版）はお近くの書店に注文されるか、下記のバックナンバー常備取り扱い店にてお求めいただけます。また、「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）や、e-hon（http://www.e-hon.ne.jp）にて、Webから注文し、ご自宅やお近くの書店へお届けするサービスもございます。

2015年8月号

第1特集
Lispより始めよ、されば救われん!
なぜ関数型プログラミングは難しいのか？

第2特集
安全な通信を確保する
SSL/TLSの教科書

短期連載
・AWSで始めよう！　モダンなJavaアプリケーション開発

定価（本体1,220円+税）

2015年7月号

第1特集
あなたにもできる!
ログを読む技術［セキュリティ編］

第2特集
黒い画面（tmux）の使い方
プロになるためのターミナル活用術

第3特集
6人の先駆者に訊く
スペシャリストになる方法

定価（本体1,220円+税）

2015年6月号

第1特集
新人さん歓迎特集
Git&GitHub入門

第2特集
OpenLDAPの教科書
ユーザ／ネットワーク管理の基本と活用例

一般記事
・Sambaによる Active Directoryの機能性と移行性を検証する

定価（本体1,220円+税）

2015年5月号

第1特集
テキスト処理ベーシックレッスン
手を動かしてデータを操ろう!

第2特集
ファイル共有自由在在
［徹底入門］最新・Sambaの教科書

特別付録
・3分間 HTTP&メールプロトコル基礎講座［特別篇］

定価（本体1,300円+税）

2015年4月号

第1特集
トラブルシューティングの極意
達人に訊く問題解決のヒント

第2特集
［最新］DNSの教科書
ネットワークを支える本物のインフラを学ぶ

特別付録
・3分間ネットワーク基礎講座［特別篇］

定価（本体1,300円+税）

2015年3月号

第1特集
カンファレンスネットワークの作り方

第2特集
いまからでも遅くない!
Hadoop超²入門

一般記事
・Cisco VIRLでネットワークシミュレーション［前編］
・Snappy Ubuntu Core

定価（本体1,220円+税）

Software Design　バックナンバー常備取り扱い店							
北海道	札幌市中央区	MARUZEN &ジュンク堂書店　札幌店	011-223-1911	神奈川県	川崎市高津区	文教堂書店　溝の口本店	044-812-0063
	札幌市中央区	紀伊國屋書店　札幌本店	011-231-2131	静岡県	静岡市葵区	戸田書店　静岡本店	054-205-6111
東京都	豊島区	ジュンク堂書店　池袋本店	03-5956-6111	愛知県	名古屋市中区	三洋堂書店　上前津店	052-251-8334
	新宿区	紀伊國屋書店　新宿本店	03-3354-0131	大阪府	大阪市北区	ジュンク堂書店　大阪本店	06-4799-1090
	渋谷区	紀伊國屋書店　新宿南店	03-5361-3315	兵庫県	神戸市中央区	ジュンク堂書店　三宮店	078-392-1001
	千代田区	書泉ブックタワー	03-5296-0051	広島県	広島市南区	ジュンク堂書店　広島駅前店	082-568-3000
	千代田区	丸善　丸の内本店	03-5288-8881		広島市中区	丸善　広島店	082-504-6210
	中央区	八重洲ブックセンター本店	03-3281-1811	福岡県	福岡市中央区	ジュンク堂書店　福岡店	092-738-3322
	渋谷区	MARUZEN &ジュンク堂書店　渋谷店	03-5456-2111				

※店舗によってバックナンバー取り扱い期間などが異なります。在庫などは各書店にご確認ください。

デジタル版のお知らせ　DIGITAL

デジタル版Software Design 好評発売中！紙版で在庫切れのバックナンバーも購入できます

本誌デジタル版は「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）と、「雑誌オンライン.com」（http://www.zasshi-online.com/）で購入できます。最新号、バックナンバーとも1冊のみの購入はもちろん、定期購読も対応。1年間定期購読なら5%強の割引になります。デジタル版はPCのほかにiPad／iPhoneにも対応し、購入するとどちらでも追加料金を払うことなく、読むことができます。

短期集中連載

Jamesのセキュリティレッスン

パケットキャプチャWiresharkの新展開

第4回

pcap-ngのさまざまな情報をWiresharkで見てみよう!

Writer 吉田 英二(Eiji James Yoshida)
合同会社セキュリティ・プロフェッショナルズ・ネットワーク(http://www.sec-pro.net/)

はじめに

みなさん、お久しぶりです！　最近、Rick Jamesの「Super Freak」にハマっている、Eiji James Yoshidaです。30過ぎの人なら知っているであろうMCハマーの「U Can't Touch This」の原曲ですが、開口一発目に"She's a very kinky girl"って歌詞を持ってくるRick Jamesの変態的なセンスがお気に入りだったりします。さて、前回の連載(2014年11月号～2015年1月号)ではバイナリエディタを使ってpcapとpcap-ngの違いを説明しましたが、さすがに毎回バイナリエディタで読むのも辛いので、今回はpcap-ngから追加されたさまざまなフィールドの情報を、Wiresharkの機能を使って見ていきたいと思います。

環境説明

本稿を書く際に使用した環境はWindows 8.1とWireshark 1.12.5です。キャプチャファイルは下記の筆者のブログからicmp.pcapとicmp.pcapngをダウンロードしてください。

・Software Design短期集中連載「Jamesのセキュリティレッスン」用キャプチャファイル
http://d.hatena.ne.jp/EijiYoshida/20140907/1410071296

あとは何かBGMでも流しましょう。筆者はMichael Jacksonの生前最後のアルバム「Invincible」を聴きながら深夜に本稿を書いています。少々うるさいですが攻撃的な曲が多くて目と頭が冴えます。

環境が整ったら、さっそくWiresharkでicmp.pcapとicmp.pcapngを別々のウインドウで開いてみましょう。

まずはWiresharkのSummaryを見比べてみよう

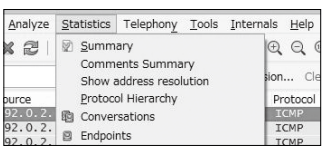
Wiresharkでicmp.pcapとicmp.pcapngを別々のウインドウで開いたら、両ファイルの[Statistics]メニュー(**図1**)にある[Summary]をクリックしてSummaryウインドウ(**図2**と**図3**)を表示し、並べてください。

このSummaryウインドウにはキャプチャファイルのさまざまな情報が表示されるため、解析前には必ず目を通しておくことをお勧めします。

icmp.pcapのSummaryウインドウ(**図2**)とicmp.pcapngのSummaryウインドウ(**図3**)を見比べると、片方にはない項目や機能していない項目が見つかるので、そのような個所を枠で囲ってみました。大まかに言ってしまえば、これがpcapとpcap-ngの情報量の違いとも言えます。

それではSummaryウインドウのFileとCaptureにある各項目の内容と、キャプチャファ

▼図1　[Statistics]メニュー



イルの該当するフィールドについて説明していきましょう。

● FileのName項目(図2の①、図3の①)

pcapとpcap-ngの両方に表示される項目です。FileのName項目には、キャプチャファイルのパスと名前が表示されます。まだキャプチャファイルに保存していない場合は一時ファイルのパスと名前が表示されます。

● FileのLength項目(図2の②、図3の②)

pcapとpcap-ngの両方に表示される項目です。長さがバイト単位で表示されます。

● FileのFormat項目(図2の③、図3の⑥)

pcapとpcap-ngの両方に表示される項目です。キャプチャファイルのファイル形式が表示されます。この項目はキャプチャファイルの拡張子ではなく、pcapのグローバルヘッダ・ブロックにあるマジックナンバー・フィールド(**図4の③**)の値や、pcap-ngのセクションヘッダ・ブロックにあるブロックタイプ・フィールド(**図5の⑥**)の値をもとに表示されます。

● FileのEncapsulation項目(図2の④、図3の⑦)

pcapとpcap-ngの両方に表示される項目です。リンク層のヘッダタイプを表すデータリンクタイプが表示されます。この項目はpcapのグローバルヘッダ・ブロックにあるデータリンクタイプ・フィールド(**図4の④**)の値や、pcap-ngのインターフェース概要・ブロックにあるデータリンクタイプ・フィールド(**図6の⑦**)の値をもとに表示されます。

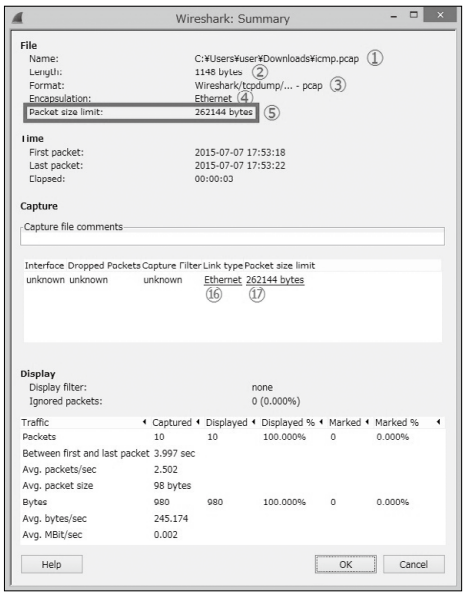
● FileのPacket size limit項目(図2の⑤)

pcapで表示される項目です。キャプチャするパケットの最大長を表すキャプチャリミットの値が表示されます。この項目はpcapのグローバルヘッダ・ブロックのキャプチャリミット・フィールド(**図4の⑤**)の値をもとに表示されます。

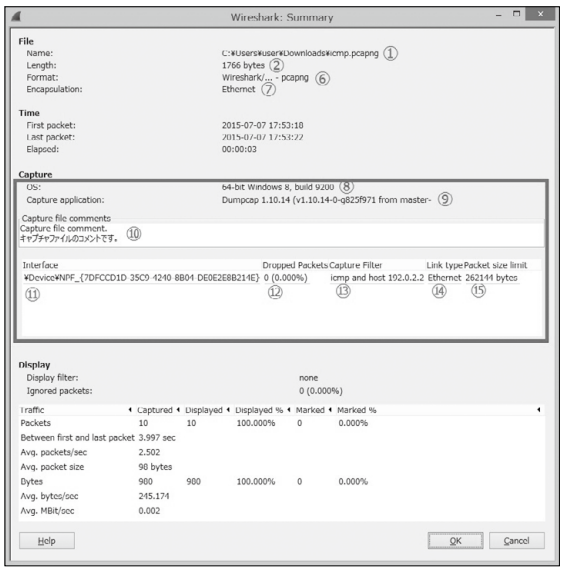
● CaptureのOS項目(図3の⑧)

pcap-ngで表示される項目です。パケットキャプチャで使用したOSの名前が表示されます。pcap-ngのセクションヘッダ・ブロックにあるオプション・フィールド(shb_os) (**図5の⑧**)の値をもとに表示されます。

▼図2　icmp.pcapのSummaryウインドウ



▼図3　icmp.pcapngのSummaryウインドウ



● CaptureのCapture application項目（図3の⑨）

pcap-ngで表示される項目です。パケットキャプチャで使用したアプリケーションの名前が表示されます。pcap-ngのセクションヘッダ・ブロックにあるオプション・フィールド（shb_userappl）（**図5の⑨**）の値をもとに表示されます。

● CaptureのCapture file comments項目（図3の⑩）

pcap-ngで機能する項目です。キャプチャファイルに設定されたコメントが表示されます。この項目はpcap-ngのセクションヘッダ・ブロックにあるオプション・フィールド（opt_comment）（**図5の⑩**）の値をもとに表示されます。

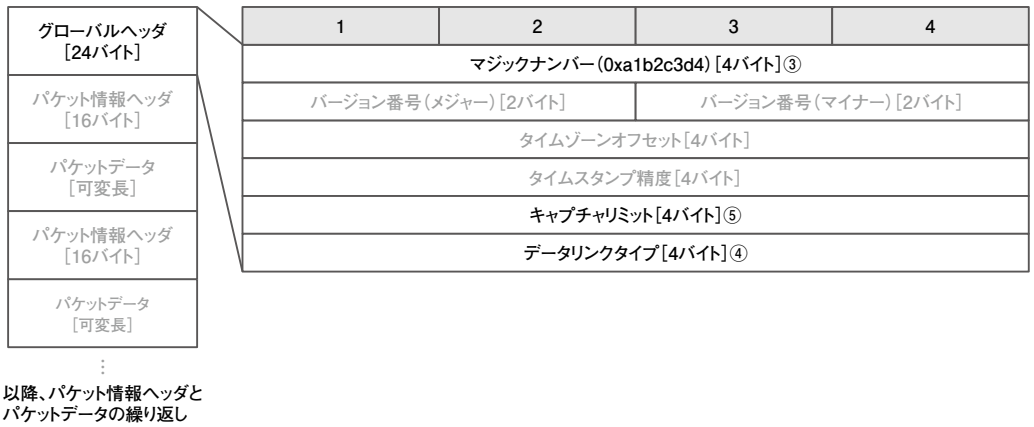
● CaptureのInterface項目（図3の⑪）

pcap-ngで機能する項目です。キャプチャで使用した各インターフェースの名前が表示されます。この項目はpcap-ngのインターフェース概要・ブロックにあるオプション・フィールド（if_name）（**図6の⑪**）の値をもとに表示されます。

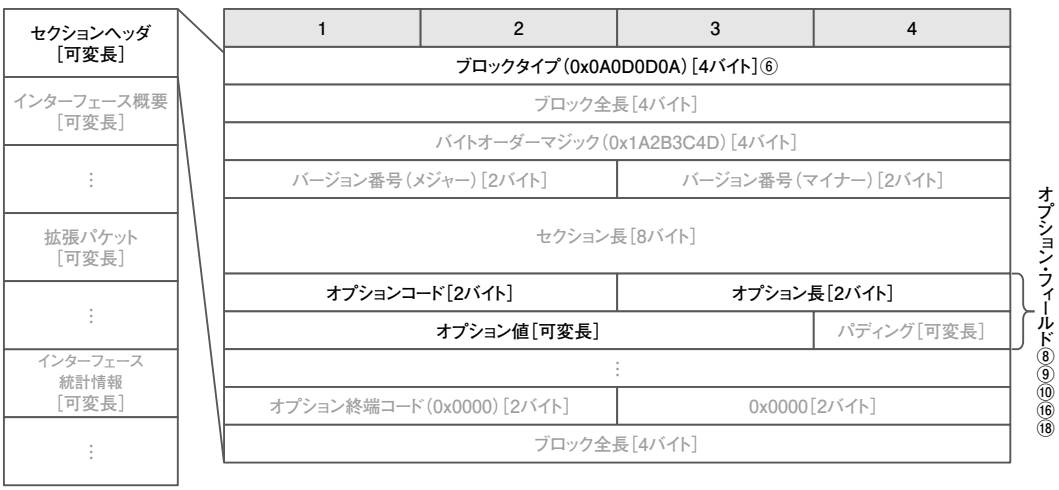
● CaptureのDropped Packets項目（図3の⑫）

pcap-ngで機能する項目です。キャプチャで

▼図4　pcapのグローバルヘッダ・ブロック



▼図5　pcap-ngのセクションヘッダ・ブロック



使用した各インターフェースがドロップしたパケット数が表示されます。インターフェース統計情報・ブロックのオプション・フィールド(isb_ifdrop)(**図7の⑫**)の値をもとに表示されます。

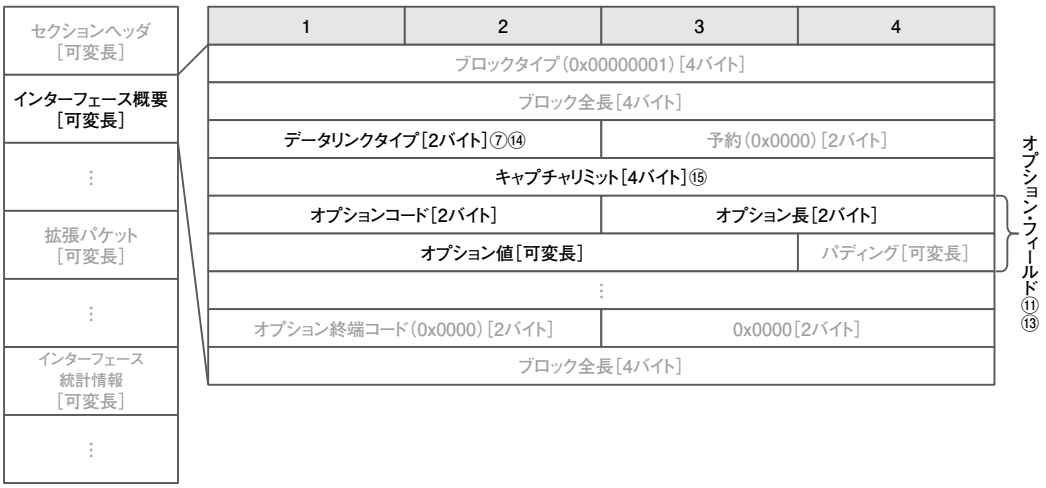
● CaptureのCapture Filter項目(図3の⑬)

pcap-ngで機能する項目です。キャプチャで使用した各インターフェースに設定されたキャプチャフィルタが表示されます。この項目はpcap-ngのインターフェース概要・ブロックにあるオプション・フィールド(if_filter)(**図6の⑬**)の値をもとに表示されます。

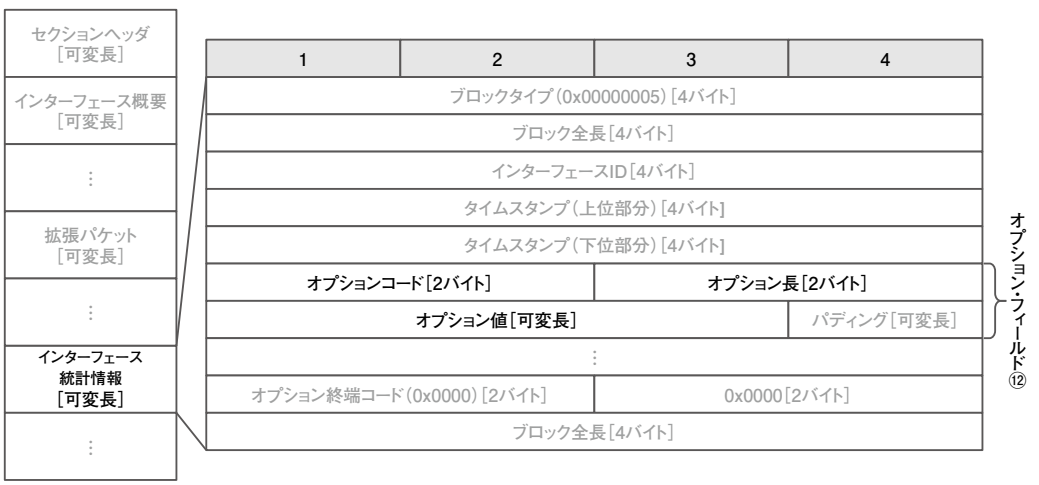
● CaptureのLink type項目(図2の⑯、図3の⑭)

pcapとpcap-ngの両方で機能する項目です。キャプチャで使用した各インターフェースのリンク層のヘッダタイプを表すデータリンクタイプが表示されます。この項目はpcapのグローバルヘッダ・ブロックにあるデータリンクタイプ・フィールド(**図4の④**)の値や、pcap-ngのインターフェース概要・ブロックにあるデータリンクタイプ・フィールド(**図6の⑭**)の値をもとに表示されます。

▼図6　pcap-ngのインターフェース概要・ブロック



▼図7　pcap-ngのインターフェース統計情報・ブロック



● CaptureのPacket size limit項目(図2の⑰、図3の⑮)

pcapとpcap-ngの両方で機能する項目です。各インターフェースのキャプチャするパケットの最大長を表すキャプチャリミットの値が表示されます。この項目はpcapのグローバルヘッダ・ブロックにあるキャプチャリミット・フィールド(**図4の⑤**)の値や、pcapngのインターフェース概要・ブロックにあるキャプチャリミット・フィールド(**図6の⑮**)の値をもとに表示されます。

◆ ◆ ◆

Summaryウインドウの各項目の内容と、キャプチャファイルの該当するフィールドについての説明は以上となります。両ファイルのSummaryウインドウは[Cancel]ボタンを押して閉じてください。

このようにpcap-ngから追加されたフィールドに含まれる情報の多くはSummaryウインドウに表示されるので、キャプチャファイルをpcap-ngで保存しているなら、このSummaryウインドウに表示される内容を知っておくと、インシデント対応などで役立ちます。

pcap-ngに保存されているコメントを見てみよう

pcap-ngにはキャプチャファイルに対するコメントだけではなく、パケットに対するコメントも保存できます。icmp.pcapngの[Statistics]メニュー(**図1**)にある[Comments Summary]をクリックしてComments Summaryウインドウ(**図8**)を表示してください。

これを見るとわかるように、Comments Summaryウインドウの内容はSummaryウインドウと大半同じですが、Statisticsの次を見るとCapture file comments(**図8の⑱**)のほかに、Summaryウインドウにはなかった内容があります。これがパケットに対して設定したコメントのPacket Comments(**図8の⑲**)です。このPacket Commentsはpcap-ngの拡張パケット・ブロックにあるオプション・フィールド(opt_comment)(**図9の⑲**)の値をもとに表示されます。

Packet Commentsを見るほかの方法としては、Packet Detailsペインで見る方法とEdit or Add Packet Commentsウインドウで見る方法などがあります(後述)。

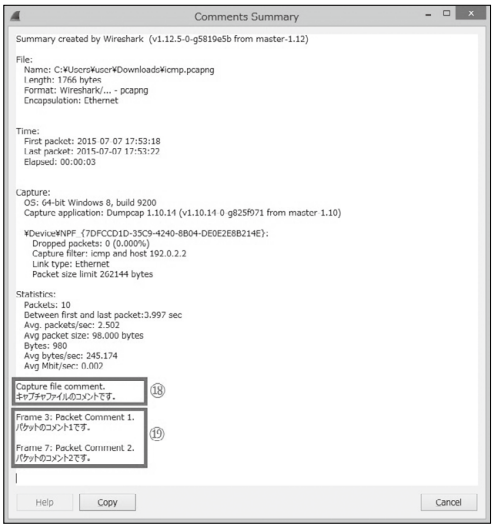
Packet DetailsペインでPacket Commentsを見てみよう

Packet DetailsペインでPacket Commentsを見るには、icmp.pcapngのPacket Listペインで No.3フレームをクリックします。するとPacket Detailsペインの一番上にPacket comments項目が表示されるので、それをクリックして展開するとコメントが表示されます(**図10**)。すぐにコメントが確認できるので便利ですが、今のところPacket Detailsペインで日本語を表示すると文字化けしてしまうのが欠点です。

Edit or Add Packet CommentsウインドウでPacket Commentsを見てみよう

Edit or Add Packet CommentsウインドウでPacket Commentsを見るには、icmp.pcapngのPacket Listペインで No.3フレームを右クリックして[Packet Comment...]を選択すると(**図11**)、Edit or Add Packet Commentsウインドウ(**図12**)にコメントが表示されます。

▼図8　icmp.pcapngのComments Summaryウインドウ



▼図9　pcap-ngの拡張パケット・ブロック

コメントを設定してみよう

Packet Commentsの表示でEdit or Add Packet Commentsウインドウを使ったので、今度は同じEdit or Add Packet Commentsウインドウで Packet Commentsを設定しましょう。

Packet Commentsを設定するには、まずPacket Listペインでコメントを設定したいパケットを右クリックして［Packet Comment...］を選択すると、Edit or Add Packet Commentsウインドウ(**図12**)が表示されます。あとは、このウインドウ内にコメントを入力して［OK］ボタンをクリックすれば設定されます。

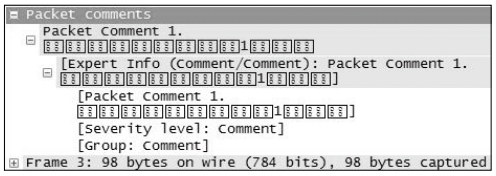
パケットではなくキャプチャファイルにコメントを設定したい場合は、［Statistics］メニュー(**図1**)にある［Summary］をクリックしてSummaryウインドウを表示して、あとはCaptureのCapture file comments項目(**図3の⑩**)の欄にコメントを入力して［OK］ボタンをクリックすれば設定されます。

◆ ◆ ◆

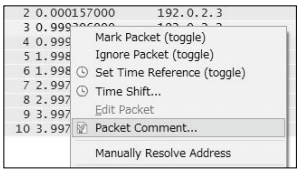
今回はWiresharkのSummaryウインドウやComments Summaryウインドウなどを使って、pcap-ngから追加されたさまざまなフィールドの情報を見てみました。前述のとおり、pcap-ngにはインシデント対応を行う際に役立つ情報を格納できるので、キャプチャファイルはpcap-ngで保存することをお勧めします。

本稿の内容が少しでもみなさんのお役に立てば幸いです。SD

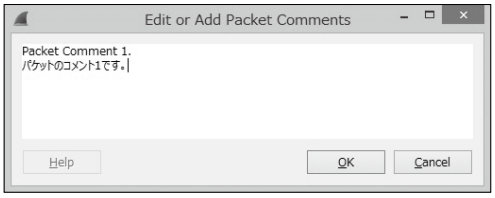
▼図10　Packet Detailsペインの Packet comments項目



▼図11　Packet Listペインのコンテキストメニュー



▼図12　Edit or Add Packet Commentsウインドウ



エンタープライズJavaの進化

DevOpsで始めよう!
モダンなJavaアプリケーション開発
しなやかで強いソフトウェアの作り方

Author 永瀬 泰一郎(ながせ たいいちろう) Twitter nagaseyasuhito
Mail nagase@nagaseyasuhito.net

ストップ属人化!
MavenとGitHubで安全なテストとスピーディなデプロイ

DevOps時代のいま、アプリケーションの開発はユーザからのフィードバックをもとにしたバグフィックスや、機能改修のイテレーションを矢継ぎ早に行い、安全かつスピーディにサービスを提供し続けることが重要です。中には1日に数百回もアプリケーションをリリース、デプロイしているサービスもあります。
そんな状況のなかで、継続的にサービスを提供し続けるためには、テストの環境はもちろん、リリースやデプロイまわりのしくみづくりも非常に重要になっています。そのようなしくみを整備するにあたって、アプリケーションのビルドツールは、今やなくてはならないものになりました。

Apache Mavenの逆襲

Apache Maven[注1]というツールをご存じでしょうか? 2002年から開発されているJava製のビルドツールで、多くのJavaプロダクトで使われているデファクトスタンダードの1つです。ビルドツールが普及する前は、独自のシェルスクリプトでjavacやjarコマンドを駆使しながら、クラスライブラリを自分でコピーしてきて依存関係を解決したり、EclipseなどのIDEの機能を使ってパッケージングしていました。しかし、このような方法は属人性が高くなりがちで、環境が少しでも変わるとビルドができなくなることが日常茶飯事でした。

問題解決のためのMaven

Apache Mavenはこのような問題を解決するためのビルドツールで、依存関係の解決からコンパイル、テストやパッケージングまで標準的なアプリケーションのビルドライフサイクルの大部分をサポートしています。そして実装はPure JavaですのでJavaがインストールされていればどのような環境でも動作するという"Write Once, Run Anywhere"[注2]を体現しています。

継続的なテストやデプロイのしくみを整備せずに、テスト用のデータベースの構築方法が不明だったり、稼働中のサービスがどの時点のソースコードがデプロイされているのかを誰も把握していなかったり、特定のスタッフだけが手動でコマンドを叩いたり、WebのUIを操作したりしてデプロイするようなしくみだったりすると、サービス継続において大きなリスクになります。

Mavenにはテストに関するプラグインも充実しているほか、標準的なリリース用のプラグインや、さまざまなアプリケーションサーバへアプリケーションをデプロイするプラグインなどが用意されています。そして複数人で1つのアプリケーションを開発するには、バージョン管理システム(VCS:Version Control System)もなくてはならないものの1つになっていると言えるでしょう。

今回はMavenと共用リポジトリサービスの1

注1) https://maven.apache.org/

注2) 「一度プログラムを書けばどのような環境でも動作する」という意味。Javaのスローガンとして広く知られていた。

▼表1　ビルドツールの比較

ビルドツール	リリース年	定義ファイル	特徴
Apache Ant	2000年	XML	コマンドの組み合わせでタスクを記述
Apache Maven	2002年	XML	プラグインの組み合わせでタスクを記述
Gradle	2007年	Groovy	Groovyスクリプトでタスクを記述

つのGitHubを使い、継続的にテストを実行する環境を整備し、属人化したデプロイを排除し、再利用性の高いデプロイのしくみを作ることで、安全かつスピーディーにサービスを提供する実践的な考え方と、その手法を紹介します。

Mavenのおさらい

Mavenにはビルドツールそのものだけではなく、個人で開発したクラスライブラリから企業が開発したアプリケーションサーバ、そしてMavenのプラグインまで、さまざまなJavaの成果物が登録されているMaven Central Repositoryと呼ばれるリポジトリシステムがあります。

Mavenが普及する前は、必要なクラスライブラリなどを自分で探し出してダウンロードしたあとに、クラスパスに追加する必要がありましたが、Maven Central Repositoryのお陰で、POM[注3]と呼ばれるMavenのビルドスクリプトに、利用したいクラスライブラリを記述するだけで、自動的にダウンロードしてクラスパスに追加できるようになりました。そして申請をすれば誰でも開発したクラスライブラリをMaven Central Repositoryへ登録できます。

このエコシステムを構築したことがMavenが大きく普及する引き金となったと言えるでしょう。Maven Central RepositoryはMavenだけでなく、ほかのビルドツールからも利用されていて、事実上Java標準のクラスライブラリリポジトリとなっています。

JavaのビルドツールにはMavenのほかにApache Ant[注4]やGradle[注5]といったビルドツールが存在します(**表1**)。AntはMavenよりも早く2000年にリリースされました。Mavenと同じようにXMLでビルドスクリプトを記述しますが、タスクごとにプラグインが用意されているMavenに対し、Antはjavacやjarなどコマンド単位でXMLタグが用意されていて、それらを組み合わせて比較的自由なスタイルでビルドスクリプトを記述します。またApache Ivy[注6]というAntのサブプロジェクトを使うことでMaven Central Repositoryにあるクラスライブラリのダウンロードにも対応できます。

Gradleは2007年にリリースされた比較的新しいビルドツールで、ビルドスクリプトにGroovyを採用したのが特徴です。Mavenはプラグインベースの宣言的なビルドスクリプトのため、条件による分岐やループなどの記述が困難でしたが、GradleはGroovyで非常に柔軟なビルドスクリプトを記述できます。

今回取り上げるMavenは、さまざまな種類のビルドレポートの出力から、テストに関するサポート、アプリケーションサーバへやクラウドプラットフォームへのデプロイに至るまで、目的に応じた非常に多くのプラグインが提供されています。これらのプラグインが、プロジェクトの要件にマッチするのであればMavenは非常に便利なビルドツールです。

反対にMavenのプラグインではカバーしきれなかったり、ビルドスクリプトで細かい制御が必要な場合などはGradleを使うといった使い分けをするといいでしょう。

ビルドライフサイクル

Mavenには**ビルドライフサイクル**、**フェーズ**、**ゴール**という概念があります。ビルドライフサイ

注3）Project Object Model
注4）http://ant.apache.org/
注5）https://gradle.org/
注6）http://ant.apache.org/ivy/

クルはフェーズを束ねる単位で、Clean、Default、Siteの3つが定義されています。フェーズは複数のゴールを束ねる単位で、3つのビルドライフサイクルで合計30個が定義されています(**表2**)。

コマンド実行時には`mvn [フェーズ]`のように引数にフェーズを指定して実行します。たとえば`mvn clean`を実行すると、Cleanビルドライフサイクルの`pre-clean`と`clean`、`mvn compile`を実行すると、Defaultビルドライフサイクルの`validate`から`compile`までのように、それぞれのビルドライフサイクルの該当するフェーズまでが対象となります。実行時には各フェーズに紐付(ひも)いたゴールが順番に実行されビルドが進行していきます。

Cleanはビルド成果物を削除し、プロジェクトのディレクトリをクリーンアップするフェーズが定義されています。ソースコードを削除しても、コンパイルされたクラスファイルなどが出力されるtargetディレクトリに、クラスファイルなどは残ってしまうのでビルドエラーの原因となる可能性があります。ビルドする際は必ず**clean**フェーズを指定するようにしましょう。

Defaultはコンパイルやテスト、パッケージングからインストール、デプロイまでの、いわゆる成果物を生成するフェーズが定義されています。

Siteはプロジェクトのドキュメントを生成するためのフェーズが定義されています。あまり活用されていませんが、Javadocをはじめ各種レポーティング用のプラグインを利用している場合には、ドキュメントがまとめてHTML化されるので、プロジェクトの概要を把握する際に重宝します。

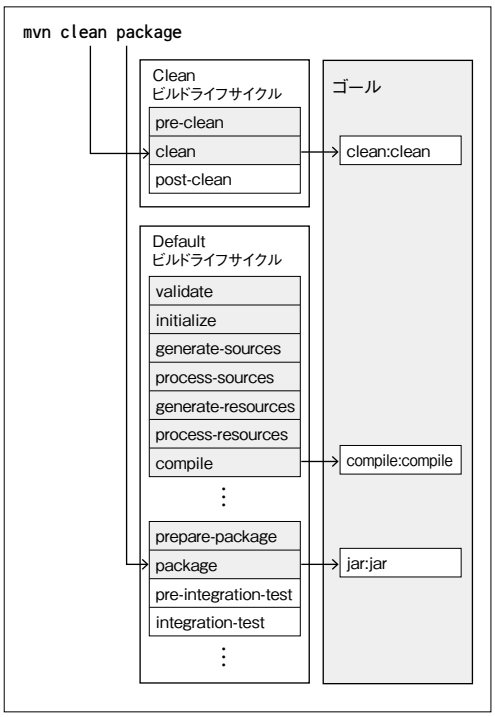
POMで定義しているパッケージングがjarのときは`jar:jar`ゴール、warのときは`war:war`を実行するなど、各フェーズで実行されるゴールはデフォルトでいくつか定義されている[注7]ほ

▼表2　Mavenのビルドライフサイクル

Clean	Default	Site
pre-clean	validate	pre-site
clean	initialize	site
post-clean	generate-sources	post-site
	process-sources	site-deploy
	generate-resources	
	process-resources	
	compile	
	process-classes	
	generate-test-sources	
	process-test-sources	
	generate-test-resources	
	process-test-resources	
	test-compile	
	process-test-classes	
	test	
	prepare-package	
	package	
	pre-integration-test	
	integration-test	
	post-integration-test	
	verify	
	install	
	deploy	

注7) http://maven.apache.org/ref/3.3.3/maven-core/default-bindings.html

▼図1　jarパッケージングでmvn clean packageを実行した際に呼び出されるフェーズとゴール



か、POMにプラグインの設定を書くことで、独自にどのフェーズでどのゴールを実行するかを定義できます。プロジェクトによってはprocess-resourcesフェーズでCSSの圧縮を行ったり、pre-integration-testフェーズでデータベースの初期化を行ったりするなど、必要なプラグインをPOMに定義してビルドの設定を作ります(**図1**)。

もちろんMavenのプラグインは独自に開発することもできるため、Maven Central Repositoryにはさまざまな用途に応じたプラグインが登録されています。標準のプラグインでは満たせないタスクも、すでにプラグインが公開されている可能性があるので、プラグインを開発する前にまず一度探してみるとよいでしょう。

Mavenプロジェクトのライフサイクル

単体テストと結合テスト

堅牢なアプリケーションを提供するために、ビルドツールはテストにまつわる機能がとても重要です。Mavenには単体テスト用のtestと、結合テスト用のintegration-testという2つのフェーズが定義されています。どのように使い分ければよいでしょうか?

まずはこの2つの違いを見てみましょう。testとintegration-testの違いの1つに、integration-testフェーズの前後には、結合テスト用の環境を準備するためのpre-integration-testと、後片付けをするpost-integration-testというフェーズが準備されていることが挙げられます。これはアプリケーションサーバやデータベースの初期化や起動、終了などの実行を想定して定義されたものです。

つまりtestフェーズではいわゆるクラスやメソッド単位のテストやモックオブジェクトなどを使った単体テストを実行し、テストをすべてパスしたもののみ、データベースやアプリケーションサーバなど結合テスト用の環境を準備してintegration-testフェーズを実行するといった使い方をします。

単体テストの実行時はmvn testでよいですが、結合テストの場合はmvn integration-testと実行するとpost-integration-testフェーズが実行されず後片付けができないため、代わりにmvn verifyを実行しましょう。

testフェーズはmaven-surefire-pluginというプラグインがデフォルトで使われます。このプラグインはクラス名がTestで終わるテストクラスをすべて実行します。

integration-testフェーズ用にはmaven-failsafe-pluginというプラグインが用意されていて、クラス名がIT(Integration Testの略)で終わるテストクラスをすべて実行します。つまりクラス名を書き分けるだけでテストのフェーズを変更できるのです。

なおmaven-failsafe-pluginはデフォルトで有効になっていないので、POMに定義する必要があります(**リスト1**)。

テスト設計で陥りがちな例として、事前にインストールされているソフトウェアや、定義されている環境変数に左右されるテストの作り方をして、特定のマシンや人だけしかテストが実行できないような設計にしてしまうと、どうしてもテストの実行回数が減っていき、いつしかテストを実行しなくなってしまうことがあります。

テストは**いつでも繰り返し実行できることが**

▼リスト1　maven-failsafe-pluginの設定

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-failsafe-plugin</artifactId>
  <version>2.18.1</version>
  <executions>
    <execution>
      <goals>
        <goal>integration-test</goal>
        <goal>verify</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

重要で、そのために極力どのような環境でも動作するような設計にすることがポイントです。

たとえば結合テスト用のアプリケーションサーバを事前に用意しておくのではなく、自動的にアプリケーションサーバを起動してテストを行うArquillian[注8]を使ったり、データベースもMaven実行時にMySQLサーバを構築するjcabi-mysql-maven-plugin[注9]というプラグインを使うなど、極力Mavenのみで環境を構築することも検討しましょう。

とくにデータベースに関しては、データが事前に準備されている共用のデータベースなどを使わずに、テーブルの作成やデータの準備などもテストの一部に含めることをお勧めします。

リリースとデプロイの違いを正しく理解していますか?

リリースとデプロイについてはさまざまな定義がありますが、違いを意識したことはあるでしょうか? 漠然と似たようなもの、という認識することも多いですが、ここではリリースは「ソースコードをタグ付けし、アプリケーションをデプロイ可能な状態にすること」、デプロイは「リリースされたアプリケーションをアプリケーションサーバに適用すること」と定義します。

ソースコードのタグ付けとは、GitやSVNなどのVCSの特定のコミットを識別するために名前付けするしくみのことで、タグの名前にはバージョンを指定するのが一般的です。

ここで重要なのがソースコードは単体テストや結合テストが、すべてパスしたものだけリリースするということです。古い機能や使わなくなったメソッドのテストが失敗するなど機能的には問題がなくても、テストが失敗しているものを本番環境にデプロイすることは、将来的に思わぬ事故の原因となるので避けるのが懸命です。

本番環境には必ずリリースされたアプリケーションだけをデプロイすることで、どの時点のソースコードが本番環境で動いているのかを把握するのに役立ちます。本番環境にどのリリースがデプロイされているのかわからない状況では、不具合が起きたときの原因究明に時間がかかってしまいます。

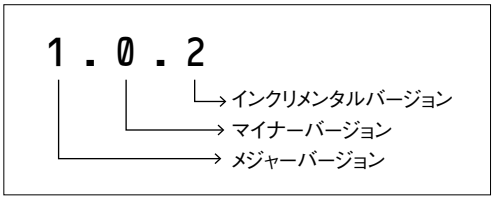
もちろん本番環境のみならず、開発環境でも常にどのバージョンがデプロイされているか把握できるようにしておくと良いでしょう。「あの人の環境では動くのに、自分の環境では動かない」というようなことは、サーバ通信のあるネイティブアプリケーションなど、クライアントとサーバのソースコードがそれぞれ独立している場合などはありがちです。そのような際にデプロイされているアプリケーションのバージョンを把握しておくことは、問題の切り分けの大きな手がかりになるでしょう。

悪い例を挙げますが、もしもリリースは手動でPOMのバージョンを変更後、単体テストや結合テストを行い、問題なかったらVCSにコミット、タグ付けを行い、POMのバージョンをインクリメントして再度コミットを行ったり、デプロイはリリースされたバージョンをチェックアウト、パッケージを作成したあと、手動でscpコマンドを叩きデプロイ先のサーバへコピー、サーバにログインしてからパッケージのパーミッションを変更、特定のディレクトリに移動したあと、アプリケーションサーバを再起動するコマンドを叩く——といった手順だといつか必ずミスが起こることは想像に難(かた)くないでしょう。

複雑な手順は作業が属人化しやすいため、リリースもデプロイも手動で複数のコマンドを実行するのではなく、1つのコマンドにまとめる

注8) http://arquillian.org/
注9) http://mysql.jcabi.com/

▼図2 バージョン表記の例



などシンプルに実行できるようにすることがとても重要です。

バージョンの付け方

Mavenプロジェクトのバージョン表記には1.0.2のようにピリオド区切りの3つの数字を使うことが一般的です。それぞれ、

- メジャーバージョン
- マイナーバージョン
- インクリメンタルバージョン

と呼ばれています(**図2**)

たとえば1.0.2というバージョンは、メジャーバージョンが1、マイナーバージョンが0、インクリメンタルバージョンが2という具合です。

それぞれのバージョン間で桁上りはせずバージョン1.0.9のインクリメンタルバージョンを上げた場合は1.1.0になるわけではなく、1.0.10になります。また2.3.10のメジャーバージョンを上げた場合は3.0.0のように下位のバージョンは0にリセットします。バージョンを上げる方針はプロダクトによってさまざまですが、一般的にインクリメンタルバージョンはAPIやインターフェース、メソッドシグネチャの変更を伴わない軽微な修正やバグフィックスの際に上げます。

マイナーバージョンはAPIやインターフェースの追加など後方互換性が損なわれない程度の修正や内部アルゴリズムの変更による性能改善、メジャーバージョンは大幅な機能の追加や削除、APIやインターフェースの変更などドラスティックな修正を行った際に上げましょう。

Mavenプロジェクトは開発中のバージョンには-SNAPSHOTという識別子がデフォルトで付加されています。たとえばバージョン1.5.2の開発中は1.5.2-SNAPSHOTのようになります。リリースの際にはこの-SNAPSHOTを外したものをVCS[注10]にコミットして、開発中のソースコードとリリースされたソースコードを明確に区別できるようにします。

GitHubとは

本誌の読者ならすでにGitHub[注11]をご存じの方も多いかもしれません。GitHubはGitをベースとした無料で利用できる共用リポジトリサービスです。同様のサービスにAtlassianが提供するBitbucket[注12]や、独自のサーバに環境を構築して使うGitLabやGitBucketといったOSSもあります。

単純なGitリポジトリとしても使えますが、これらのサービスには**プルリクエスト**(Pull Request)と呼ばれる複数人でアプリケーションを開発する際に非常に便利なしくみがあります。

プルリクエストとは元となるリポジトリを**フォーク**して、自分のリポジトリでソースコードの修正をし、差分をもとのリポジトリに対して取り込む(プル)ようにリクエストを通知するしくみのことです。機能追加やバグフィックスごとにプルリクエストを作成し、必要に応じてコードレビューや修正を加えたあとに元のリポジトリにマージできます。見やすいWebのUIでコードレビューが気軽にできるようになったことはGitHubが大きく広まった理由の1つです。多くのOSSの開発もGitHub上で行われるようになり、プルリクエストベースの開発フローはモダンな開発現場のデファクトスタンダードと言えるでしょう。

テストとプルリクエスト

GitHubにプルリクエストされたソースコードを、元のリポジトリにマージする前にテストを行うにはどうすれば良いでしょうか?

GitHubはプルリクエストが作られると、シーケンシャルなIDが振られ、そのIDに対応する

注10) Version Control System

注11) https://github.com/
注12) https://bitbucket.org/

▼図3　プルリクエストをローカルリポジトリにマージする

```
git config --add remote.origin.fetch '+refs/pull/*/head:refs/remotes/pull/*'
  プルリクエストのブランチ(refs/pull/*/head)をローカルブランチ(refs/remotes/pull/*)にマッピング

git remote update  リモートリポジトリをフェッチ

git checkout pull/14  #14のプルリクエストをチェックアウト
```

ブランチが生成されるので、ローカルにチェックアウトすることができます。

`git fetch origin pull/[プルリクエスト番号]/head:[ブランチ名]`でフェッチするか、もしくは**図3**のように`git config`でリモートリポジトリのブランチをローカルブランチにマッピングし、`git remote update`でフェッチしておくと、いつでもプルリクエストごとのブランチをチェックアウトできるので便利です。

チェックアウトしたらあとはいつもどおり、`mvn clean verify`コマンドなどでテストを実行し、問題がなければプルリクエストをマージすると良いでしょう。

実践!　リリース&デプロイ

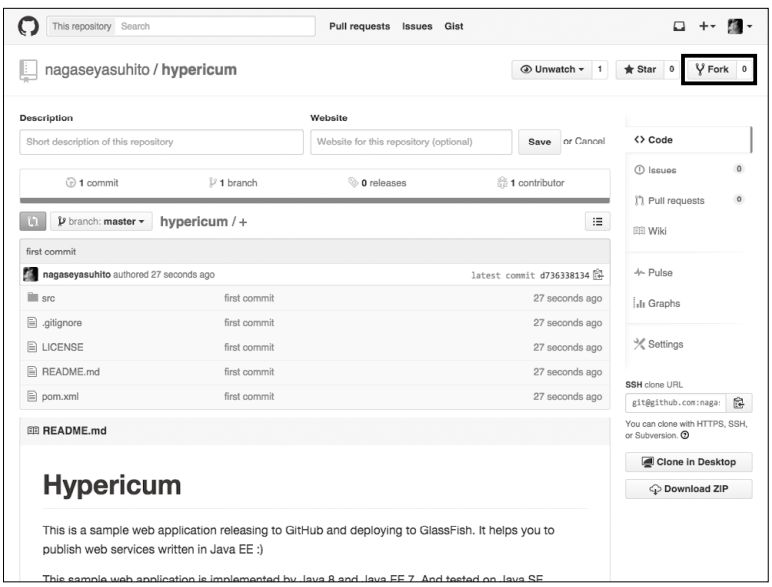
それではMavenを使ったシンプルなJava EEのアプリケーションをGlassFish[注13]にデプロイするケースを例に、実際にリリースとデプロイを体験してみましょう。

Mac OS Xの場合はHomebrew[注14]を使うとGlassFishを簡単にインストールできます。その他のプラットフォームの場合はオフィシャルサイトからアーカイブをダウンロードして適当なディレクトリに展開してください。

サンプルのソースコードは`https://github.com/nagaseyasuhito/hypericum`に公開しています。Java EEを使ったREST APIのサンプルで、kuromoji[注15]という日本語形態素解析エンジンを使い、リクエストのクエリ文字列をわかち書きし、JSONフォーマットに変換し

注13) https://glassfish.java.net/
注14) http://brew.sh/
注15) http://www.atilika.org/

▼図4　GitHubのForkボタン



て返します。GitHubにサインアップしてフォークしてみましょう。**図4**のリポジトリの画面右上の[Fork]ボタンをクリックすると自分のリポジトリにフォークされます。

プルリクエストベースのワークフロー

プルリクエストを体験するために、フォークしたリポジトリを、ローカルマシンにクローンして修正を行い、プルリクエストを投げてみましょう。通常はフォーク元のリポジトリに対してプルリクエストを行いますが、今回は説明を単純化するため同じリポジトリ内でプルリクエストを投げてみます。

POMのscm以下に記述されているリポジトリのURLを、自身のGitHubリポジトリのURLの修正してみましょう(**図5**)。

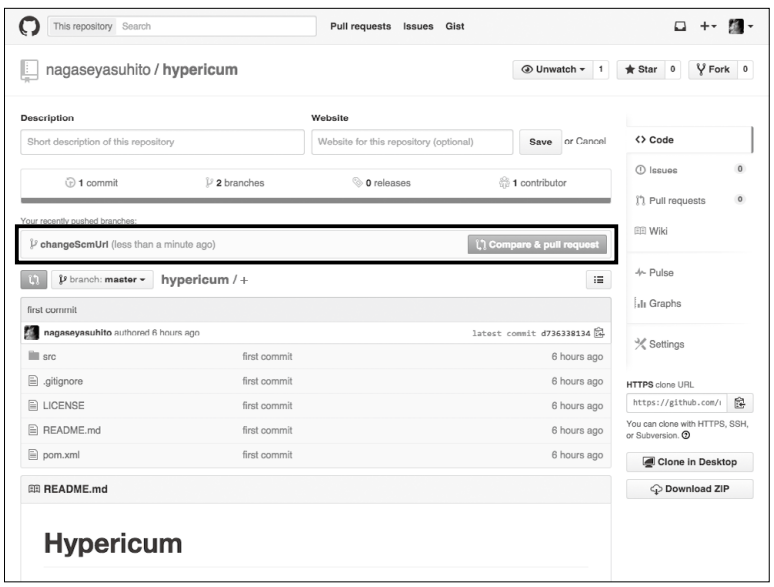
流れとしては、最初に`git clone`でローカルマシンにフォークしたリポジトリを持ってきたあと、`git checkout -b`で作業用のブランチを作成します。POM修正後に`mvn clean verify`でテストが正しく通るか確認し、`git push`で新しいリモートブランチにプッシュしています。新しく差分がプッシュされると、プルリクエスト用のボタンがWebのUIに表示されます(**図6**)。

このボタンをクリックするとプルリクエストの作成ができます。差分を確認して問題なければ[Create pull request]ボタンをクリックしましょう。複数人での開発の場合、プルリクエストの内容をチェックしながら、コードレビューやテストを行い、必要があれば追加で修正などをして再度プッシュします。このようにコードレビューと修正の履歴が可視化されるので、あとあとに修正の意図を知りたい場合に重宝します。

▼図5　Gitで編集する例

```
git clone git@github.com:[自身のGitHubアカウント名]/hypericum.git  ローカルマシンにクローン
git checkout -b changeScmUrl  changeScmUrlという名称のブランチを作成
vi pom.xml  POMを編集する
mvn clean verify  Mavenでテストを実行
git commit -a -m "change scm url to own repository url"  POMを修正後コミットしてoriginにプッシュ
git push origin changeScmUrl
```

▼図6　GitHubのプルリクエストの通知



maven-release-pluginでリリース

それでは次にリリースを行ってみましょう。Mavenにはmaven-release-pluginというリリースのためのプラグインが用意されていて、POMに記述しているプロジェクトのバージョンの変更やVCSへのタグ付けなどが行えます。

maven-release-pluginは**リスト2**のようにPOMにプラグインの設定を記述します。このプラグインにはprepareとperformという2つのゴールが含まれています。prepareゴールは、

①未コミットのファイルがないかチェック
②スナップショットの依存関係がないかチェック
③プロジェクトのバージョンから-SNAPSHOTを削除したものに変更(1.0.0-SNAPSHOT → 1.0.0)
④テストの実行
⑤変更したPOMをコミット
⑥VCSにタグ付け
⑦プロジェクトのバージョンをインクリメント(1.0.0 → 1.0.1-SNAPSHOT)
⑧変更したPOMをコミット

を実行します。

performゴールは、prepareゴールでVCSにタグ付けされたソースコードに対して、

▼リスト2　maven-release-pluginの設定

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-release-plugin</artifactId>
  <version>2.5.2</version>
  <configuration>
    <goals>package cargo:deploy</goals>
  </configuration>
</plugin>
```

▼リスト3　cargo-maven2-pluginの設定

```
<plugin>
  <groupId>org.codehaus.cargo</groupId>
  <artifactId>cargo-maven2-plugin</artifactId>
  <version>1.4.13</version>
  <configuration>
    <container>
      <containerId>glassfish4x</containerId>
      <type>remote</type>
    </container>
    <configuration>
      <type>runtime</type>
      <properties>
        <cargo.hostname>${cargo.hostname}</cargo.hostname>
        <cargo.glassfish.admin.port>${cargo.glassfish.admin.port}</cargo.glassfish.admin.port>
        <cargo.remote.username>${cargo.remote.username}</cargo.remote.username>
        <cargo.remote.password>${cargo.remote.password}</cargo.remote.password>
      </properties>
    </configuration>
  </configuration>
  <dependencies>
    <dependency>
      <groupId>org.glassfish.main.deployment</groupId>
      <artifactId>deployment-client</artifactId>
      <version>4.1</version>
    </dependency>
  </dependencies>
</plugin>
```

POMのconfigurationのgoalsで定義されたゴールを実行します。つまりgoalsにパッケージ生成とデプロイを行うゴールを記述しておくことで、リリースと同時にアプリケーションサーバにデプロイすることが可能になります。

cargo-maven2-pluginでGlassFishにデプロイ

アプリケーションサーバにデプロイするcargo-maven2-plugin[注16]というプラグインがあります。これはGlassFishだけでなくJBossやWildfly、WebLogicからTomcatやJettyまでさまざまなアプリケーションサーバに対応したデプロイ用のプラグインです。

リスト3はGlassFish4.1用の設定です。configuration以下のcontainerにはデプロイするコンテナの種類を記述し、configurationにはデプロイ先のホスト名やポート番号、管理ユーザのユーザ名やパスワードを記述します。

ホスト名などはMavenのプロパティにすることで、プロファイルやコマンドライン引数で指定できるようになります(**図7**)。

GlassFishの管理用コマンド`asadmin start-domain`でアプリケーションサーバを起動した状態で`mvn clean verify cargo:deploy`を実行するとデプロイが行われます。デプロイ後curlコマンドでREST APIにリクエストを送信すると、結果がJSONで返ってくることが確認できます(**図8**)。

先述したとおりmaven-release-pluginの`perform`を実行したときのゴールを`package cargo:deploy`にしておくことで、`mvn clean release:prepare release:perform`の実行で、リリースとデプロイを同時にできます。

まとめ

いかがでしたでしょうか？　Mavenには数多くの機能がプラグインとして提供されているため、宣言的に設定を記述できることがおわかりいただけたかと思います。またビルドライフサイクルの理解を深め、単体テスト、結合テストのユースケースの違いを把握することで効率的なテスト設計ができるようになります。

開発現場によってベストプラクティスはさまざまですが、この記事を参考にモダンなプルリクエストベースの開発フローと安全なリリース、デプロイのしくみを導入し、スピーディなサービスを提供できる体制を作るきっかけになれば幸いです。SD

注16) https://codehaus-cargo.github.io/cargo/Maven2+plugin.html

▼図7　コマンドライン引数でホスト名を指定する例

```
mvn clean verify cargo:deploy -Dcargo.hostname=127.0.0.1
```

▼図8　デプロイしたアプリケーションにクエリを発行する例

```
$ curl http://localhost:8080/hypericum/api/tokenize?query=本日は晴天なり
{
  "query":"本日は晴天なり",
  "tokens":{
    "token":[
      {"allFeatures":"名詞,副詞可能,*,*,*,*,本日,ホンジツ,ホンジツ","reading":"ホンジツ",
"surfaceForm":"本日"},
      {"allFeatures":"助詞,係助詞,*,*,*,*,は,ハ,ワ","reading":"ハ","surfaceForm":"は"},
      {"allFeatures":"名詞,一般,*,*,*,*,晴天,セイテン,セイテン","reading":"セイテン",
"surfaceForm":"晴天"},
      {"allFeatures":"助動詞,*,*,*,文語・ナリ,基本形,なり,ナリ,ナリ","reading":"ナリ",
"surfaceForm":"なり"}
    ]
  }
}
```

短期集中連載 最終回

プログラマに優しい現実指向JVM言語

Kotlin入門

第6回 KotlinでAndroidプログラミング

4月号から続いた本連載も今月で最終回です。今まで解説してきた内容をふまえて、KotlinによるAndroidプログラミングを解説します。

Author 長澤 太郎(ながさわ たろう) Twitter @ngsw_taro Mail taro.nagasawa@gmail.com

はじめに

Kotlinは、以前紹介したようにJava仮想マシンやJavaScriptだけでなくAndroidもターゲットとしています。Kotlinのシンプルで読みやすく、安全なコードでAndroidアプリケーションを開発できるのはとても魅力的です。そして、使いたくなるようなライブラリやツール群もいくつかあります。本記事では、開発環境の構築から始めて、Kotlin用Androidプロジェクトの作成、Kotlin活用のアイデアを解説したあと、ライブラリとツールの紹介をします。なお、Android開発経験のある人を対象とした構成となっています。

開発環境構築

連載第2回(2015年5月号)で紹介した開発環境の1つに、Kotlinプラグインを導入したIntelliJ IDEA[注1]がありました。このIntelliJ IDEAをベースとして作られているAndroid Studio[注2]にKotlinプラグインをインストールします。Android Studioが未インストールの人は、まずそちらをインストールしてください(インストール方法については誌面の都合上割愛させていただきます)。

Android Studioを起動した状態で、メニューから[Preferences...]を選択し表示します。そして[Plugins]の項目から画面下部にある[Install JetBrains Plugin...]をクリックします。するとインストール可能なプラグイン一覧が表示されるので[Kotlin]を選択し、[Install plugin]ボタンをクリックします。ダウンロードとインストールが始まります。検索ボックスに「Kotlin」と入力するとプラグインが名前でフィルタリングできるので便利です。インストールが完了したらAndroid Studioを再起動してプラグインを有効にしてください。

KotlinでAndroid開発を始める

まずはPhone用のプロジェクトを新規作成します。<プロジェクトルート>/app/src/main/java/<パッケージ>/MainActivity.javaが生成されているはずです(デフォルトの設定を使用した場合)。このMainActivity.javaファイルを開きます(**図1**)。

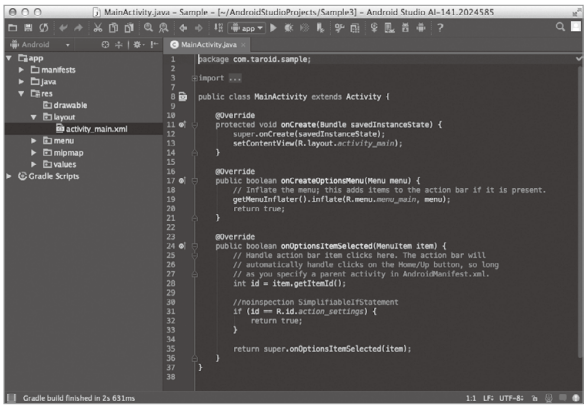
メニューから[Code]→[Convert Java File to Kotlin File]をクリックします。この操作によりMainActivity.javaに記述されているJavaコードが自動的にKotlinコードに変換され、ファイル名もMainActivity.ktに変わります(**図2**)。

次に、メニューから[Tools]→[Kotlin]→[Configure Kotlin in Project]をクリックします。すると使用するKotlinのバージョンについて質問するダイアログが表示されるので、そのま

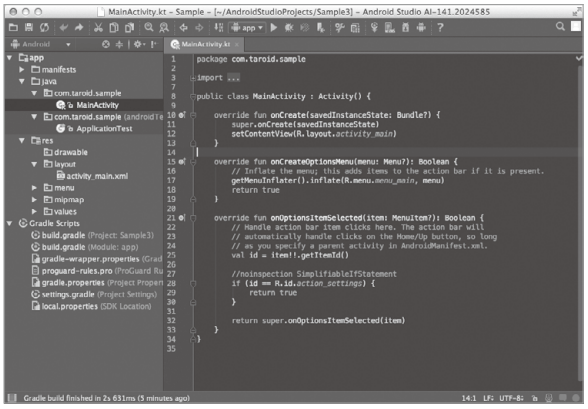
注1) https://www.jetbrains.com/idea/

注2) https://developer.android.com/intl/ja/sdk/index.html

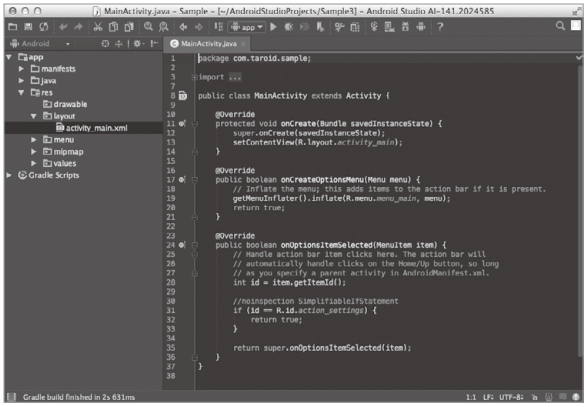
▼図1　自動生成されたMainActivity.java



▼図2　JavaからKotlinに自動変換される

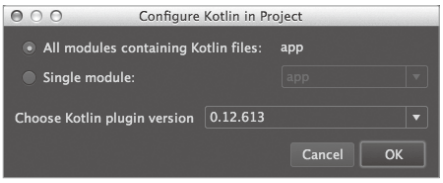


▼図4　build.gradleに必要な記述が自動的に追記される



ま「OK」ボタンをクリックします(**図3**)。**図4**のようにbuild.gradleに必要な記述が自動的に追記されます。この変更をAndroid Studioに反映するため画面上部の[Sync Now]をクリックしてください。

▼図3　Kotlinのバージョンを質問してくるダイアログ



最後に、/app/src/main/kotlinディレクトリを作成し、MainActivity.ktファイルを/app/src/main/javaからそこへ移動して完了です。これでAndroid開発を始められます！

MainActivity.ktを**リスト1**のように編集して、ビルド、実行してみてください。Kotlinコードが Android上で動いていることが確認できます。

Kotlin活用のアイデア

KotlinでAndroid開発を始めると、すぐにその快適さに気づくと思います。ここではその快適さをもたらしてくれるKotlin活用のヒントを紹介します。

関数リテラルでイベントリスナ

KotlinはJavaとの相互運用性が高いです。JavaコードをKotlinで呼び出すのがすごく簡単です。

Viewクラスのメソッド`setOnClickListener(View.OnClickListener)`の例を考えます。ビューのクリック時のアクションを登録するこのメソッドですが、クリック時のアクションは`View.OnClickListener`というインターフェースで表現されています。このインターフェースは`fun onClick(v: View): Unit`というメソッドを1つだけ持ちます。このように抽象メソッドをただ1つ持ったインターフェースを引数に取るメソッドに、Kotlinコードでは関数を渡すことができます！　この例では`(View) ->`

▼リスト1　Hello, Kotlin!

```
public class MainActivity : Activity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

      "Kotlin".hello()
   }

  fun String.hello() {
    val msg = "Hello, $this!"
    Toast.makeText(this@MainActivity, msg, Toast.
LENGTH_SHORT).show()
  }
}
```

▼リスト3　Javaで書くとノイズが多い

```
// Javaコード
button.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    Log.d(TAG, "Clicked");
  }
});
```

▼リスト4　Contextの拡張関数としてトースト表示機能を追加

```
fun Context.toast(msg: String) {
  Toast.makeText(this, msg, Toast.LENGTH_SHORT).show()
}
```

Unit型の関数をView#setOnClickListener(View.OnClickListener)の引数として渡せるということです。

リスト2ではクリック時のアクションを関数リテラルで表現してリスナ登録しています。参考までにJavaで書き直したものを**リスト3**に示します。

拡張関数で便利APIを作る

既存の型にメソッドを生やせる拡張関数がすごく便利なことは第5回で解説しました。この拡張関数を駆使してAndroid標準のAPIを使いやすく改造してみましょう。

まずはシンプルな例としてトーストを挙げます。**リスト4**のようなコードを書きました。この拡張関数toastにより、Contextのサブクラス内、たとえばActivity内でトースト表示する際にはtoast("こんにちは")と記述するだけで済みます。このような記述を得るためだけにBaseActivityのようなスーパークラスを導入していまいがちですが、いろいろ厄介なBaseActivityを避けられます。

▼リスト2　OnClickListenerの代わりに関数を渡せる

```
button.setOnClickListener {
  Log.d(TAG, "Clicked")
}
```

もう1つ面白い例を紹介します。**リスト5**では、プリファレンスの編集を便利にする拡張関数を定義しました。

呼び出し側のコードはどのようになるか想像できますか？　すでに取得しているプリファレンスprefに対して**リスト6**のようなコードが記述できるようになります。

まず**リスト5**の拡張関数editを見てください。この拡張関数の役目は、プリファレンスの編集が終わったあとにSharedPreferences.Editor#applyを実行することです。引数として受け取る関数fの中で実際に編集が行われ、fの呼び出しのあとにapplyを呼び出しているだけです。引数fの型に注目してください。初めて登場する記法です。SharedPreference.Editor.() -> Unitという関数型です。() -> Unitだけであれば「引数を取らずに何も返さない関数の型」と読めるのは、すでにご存じかもしれません。頭に付くSharedPreference.Editor.は、メソッドのレシーバとみなせます。つまりfは、クラスSharedPreference.Editorの() -> Unitなメソッドなのです！

このfは単なる関数ではなく、メソッドですので単体では呼び出せません。レシーバとなるオブジェクトが必要です。**リスト5**でeditor.f()のように呼び出しているのはそのためです。editorがレシーバとなりfを呼び出しています。

このような拡張関数、いや拡張メソッドは何の役に立つのでしょうか。実際、fの型を

▼リスト5　プリファレンスの編集を便利に

```
fun SharedPreferences.edit(f: SharedPreferences.Editor.() -> Unit) {
    val editor = this.edit()
    editor.f()
    editor.apply()
}

platformName("putLong")
fun SharedPreferences.Editor.put(pair: Pair<String, Long>) {
    putLong(pair.first, pair.second)
}

platformName("putString")
fun SharedPreferences.Editor.put(pair: Pair<String, String>) {
    putString(pair.first, pair.second)
}
```

▼リスト6　プリファレンス編集が簡単になった

```
pref edit {
  put("id" to 123L)
  put("name" to "たろう")
}
```

(SharedPreference.Editor) -> Unitとして、f(editor)と呼び出せば機能としては同じことができます。しかし呼び出し側(**リスト6**)のコードが変わってきます。it.put("id" to 123L)のようにレシーバを(それが明らかなのにもかかわらず)明示する必要が出てきます。fがSharedPreference.Editorの拡張メソッドであることでput("id" to 123L)のようにレシーバを省略することが可能になります。

それから**リスト5**の2つの拡張関数putについてです。この2つの関数は、Kotlin標準ライブラリに含まれるクラスPairを引数に取ります(説明のためにわざとらしくそうしています)。クラスPairはその名のとおり組、ペアを表現するクラスです。今回の場合、プリファレンスとして保存する対象となっているのでPairの第1要素をキー、第2要素を値としています。2つの関数putは同名ですが、Pair<String, String>とPair<String, Long>で引数の型が異なります。Kotlinではこれを区別できますが、Javaではできません。そのためJava用に別名を付けてやる必要があります。platformNameアノテーションを付けて、その引数にJava用の名前を付けるだけです。

最後に**リスト6**を見てください。putの引数を"id" to 123Lと記述しています。これはPairリテラルです。と言うとわかりやすいですが少し違います。toは任意の型に対する拡張関数で、Pairインスタンスを生成します。"id" to 123Lは"id".to(123L)ともPair("id", 123L)とも記述できます。

ライブラリ・ツール

Kotlinでの Android開発をより快適にしてくれるライブラリやツールを紹介します。

Kotter Knife

Kotter Knife[注3]はView Injectionライブラリです。頻繁に登場するfindViewByIdによるビューのマッピング作業から解放してくれるライブラリです。Java用のButter Knife[注4]のKotlin版と言えます。ちなみに開発者はAndroid界隈で名高い(どころかスーパースターの)Jake Whartonさんです。

導入は簡単です。**リスト7**のようにbuild.gradleを編集してください。

導入後、すぐに使い始められます。**リスト8**にKotter Knifeの簡単な使用例を示します。MainActivityのプロパティとして各ビュー(nameEditTextとsubmitButton)を保持しています。onCreateメソッド内でfindByViewIdメソッドを呼び出してビューのマッピングを行うのが通常の方法ですが、このコードには

注3) https://github.com/JakeWharton/kotterknife

注4) http://jakewharton.github.io/butterknife/

findByViewIdが登場しません。setContentView(R.layout.activity_main)を呼び出した後すぐにsubmitButtonに対してクリックリスナを登録しています。

これを可能にしているのは各ビューのプロパティの宣言時にby bindView(ID)と記述したおかげです。bindViewメソッドはKotter Knifeが提供するAPIです。プロパティ宣言のあとにbyと記述しているのはKotlinのDelegated Propertyという機能を使うためです。詳細は割愛しますが、Delegated Propertyとはプロパティへアクセスがあったときに、その処理を別のオブジェクトに委譲するしくみです。Kotter Knifeではこの Delegated Propertyを使って、プロパティとして保持しているビューに初めてアクセスがあったときに、ビューを取得するコードが発動するように作られています。

Anko

次に紹介するのはJetBrainsにより開発されているライブラリ、Anko[注5]です。Android開発を簡単にする便利なAPIが数多くそろっていますが、目玉機能はUIレイアウトを構築するDSL(Domain Specific Language：ドメイン特化言語)です！

Android開発では通常、XMLでレイアウトを組んでJavaコードからそれを利用するという流れが一般的なのはみなさんご存じのとおりです。Ankoはレイアウトを XMLファイルとしてではなく、Kotlinコード上で組み上げるアプローチを提案しています。Kotlinで作成されたDSLを使うので、Kotlinの恩恵をそのまま受けられます。つまり簡潔、型安全、NULL安全ということです。

まずは非常に簡単な例をご覧入れましょう。**リスト9**では、押すとトーストが表示されるボ

▼リスト7　Kotter Knife導入

```
dependencies {
  // (略)
  compile 'com.jakewharton:kotterknife:0.1.0-SNAPSHOT'
}
repositories {
  // (略)
  maven {
    url 'https://oss.sonatype.org/content/repositories/snapshots/'
  }
}
```

▼リスト8　Kotter Knife使用例

```
class MainActivity : Activity() {

  val nameEditText: EditText by bindView(R.id.name_edit_text)

  val submitButton: Button by bindView(R.id.submit_button)

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    submitButton.setOnClickListener {
      val name = nameEditText.getText().toString()
      Toast.makeText(this, name, Toast.LENGTH_SHORT).show()
    }
  }
}
```

▼リスト9　Ankoはこんな感じでで UIを表現する

```
class MainActivity : Activity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    verticalLayout {
      button("Click me!") {
        onClick {
          toast("Hello")
        }
      }
    }
  }
}
```

注5）https://github.com/JetBrains/anko 「小豆を煮詰めて作る、あの“あんこ”が名前の由来らしいです」

▼リスト10　UI構築部分を関数として切り出す

```
class MainActivity : Activity() {
  override fun onCreate(savedInstanceState:
Bundle?) {
    super.onCreate(savedInstanceState)
    val button = createLayout()
    button.setOnClickListener {
      toast("Hello")
    }
  }
}

fun Activity.createLayout(): Button {
  var button: Button? = null
  verticalLayout {
    // ボタンのテキストとしてリソースIDも指定できる
    button = button(R.string.click_me) {
      // ボタンのテキストサイズを指定
      textSize = 24f
    }.layoutParams {
      // マージンを指定
      margin = dip(20)
    }
  }
  // ボタンを返す
  return button ?: throw AssertionError()
}
```

タンが1つ表示されるようなアクティビティを作っています。

Kotlinコードですので、Kotlinでできることは何でもできます。たとえばレイアウトの使い回しはどうするのか、という問題はリスト10のように関数に出すのも1つの方法です。

拡張関数createLayoutで、Ankoを使ってレイアウトを構築しています。そしてAnkoを使って生成したボタンを返して、関数の呼び出し元でボタンにクリックリスナを登録しています。Ankoはまだ発展途上のライブラリで、筆者自身もベストプラクティスを模索中です。

肝心の導入方法ですが、簡単です。リスト11の1行をgradle.buildのdependenciesに追記するだけです。とても面白いライブラリですのでぜひ使ってみてください。

▼リスト11　Anko導入

```
compile 'org.jetbrains.anko:anko:0.6.2-15'
```

▼リスト12　activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.
android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent">

  <Button
    android:id="@+id/helloButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello"/>
</RelativeLayout>
```

Kotlin Extensions for Android

最後に紹介するのはKotlin Extensions for Androidというツールです。モチベーションとしてはKotter Knifeと同じくfindViewByIdの排除です。しかしKotlin Extendions for Androidはさらに一歩進めて、プログラマがビューのマッピングを行ったり指定したりするような記述すら必要ありません。

簡単な具体例を示します。リスト12はactivity_main.xmlという名前のレイアウトファイルです。そして、このレイアウトファイルを使うMainActivityの定義がリスト13です。

▼リスト13　ビューオブジェクトが自動生成されている

```
package com.taroid.sample

import android.app.Activity
import android.os.Bundle
import android.widget.Toast
import kotlinx.android.synthetic.activity_main.helloButton

public class MainActivity : Activity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    helloButton.setOnClickListener {
      Toast.makeText(this, "Hello", Toast.LENGTH_SHORT).show()
    }
  }
}
```

リスト13では、onCreate内でいきなり登場するhelloButtonに対してクリックリスナを登録しています。Kotter Knifeのときと異なり、helloButtonはプロパティに宣言されていなければDelegated Propertyも使用されていません。このhelloButton、どこからやってきたのかと言うとKotlin Extensions for Androidによって生成され、それをインポートすることでMainActivity内で使えるようにしています。自動生成されるビューの完全な名前を一般化すると「kotlinx.android.synthetic.<レイアウトファイル名>.<リソースID>」のような形式になります。

Kotlin Extensions for Androidを導入するには、まずプラグインをインストールします。同名のプラグインをAndroid Studioにインストールし、再起動します。そしてbuild.gradleをリスト14のように編集して、使えるようになります。

まとめ

Android Studioにプラグインを入れるだけで、すぐにKotlinによるAndroidプログラミングを体験できます。Kotlinコードからシームレスに AndroidのAPIを呼び出せます。Kotlinの独特な文法、たとえば拡張関数やNULL安全などもAndroid上で動きます。

ボタンのクリックリスナのような、抽象メソッドが1つしかないインターフェースとして関数リテラルを使うことができます。クリックリスナの登録のときに、匿名クラスを書いて、メソッドをオーバーライドするようなノイズの多い記述をKotlinではしなくて済みます。

Contextを引数に取るような便利メソッドの定義の際には、拡張関数が非常に威力を発揮するでしょう。Contextに対する拡張関数とすれば、呼び出し側のコードが目に優しく直感的なスタイルになります。

便利なライブラリ・ツールを3つ紹介しました。Kotter KnifeはButter KnifeのKotlin版で、Delegated PropertyというKotlinの機能をうまく利用して実現されているView Injectionライブラリです。AnkoはAndroid開発におけるDSLセットで、とくにUIレイアウトが興味深いです。Kotlinの簡潔で安全な特長をUIレイアウトに活かせるのはうれしいです。Kotlin Extensions for AndroidはView Injectionのためのツールで、ビューのマッピングが全自動なため作業が減り、コードもすっきりします。

おわりに

全6回に及ぶKotlin入門連載、いかがでしたか？　業務でKotlinを使う日は遠からず来るのではないかと期待しています。現時点でKotlinはβ版という位置づけですが、そろそろバージョン1.0がリリースされそうな気配を感じています。今後も筆者のブログでKotlin情報の発信は続けていくので、新しいマイルストーンがリリースされたときなどにはチェックしてください。

Kotlinでみなさまのプログラミングが少しでも快適に、そして今よりもっと楽しくなればエバンジェリスト[注6]冥利に尽きます。Let's enjoy Kotlin！ SD

▼リスト14　Kotlin Extensions for Android導入

```
buildscript {
  // (略)

  dependencies {
    // (略)
    classpath "org.jetbrains.kotlin:kotlin-android-extensions:$kotlin_version"
  }
}
```

注6）自称ですが。

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。

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

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

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

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

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

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

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

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

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

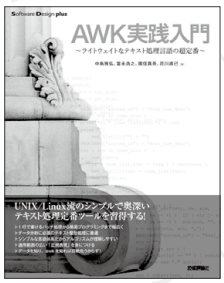
エンジニアのためのデータ可視化[実践]入門
森藤大地、あんちべ 著
定価 2,780円+税　ISBN 978-4-7741-6326-0

GPU並列図形処理入門
乾正知 著
定価 3,200円+税　ISBN 978-4-7741-6304-8

Zabbix統合監視徹底活用
TIS㈱ 池田大輔 著
定価 3,500円+税　ISBN 978-4-7741-6288-1

過負荷に耐えるWebの作り方
㈱バイブinfビッツ 著
定価 2,480円+税　ISBN 978-4-7741-6205-8

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

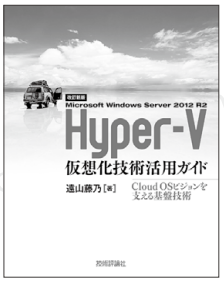
福田和宏、中村文則、竹本浩、木本裕紀 著
B5判・128ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7345-0

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

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

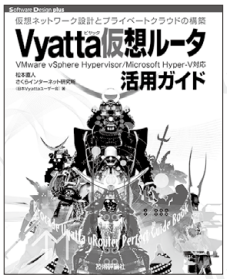
勝俣智成、佐伯昌樹、原田登志 著
A5判・288ページ
定価 3,300円(本体)+税
ISBN 978-4-7741-6709-1

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



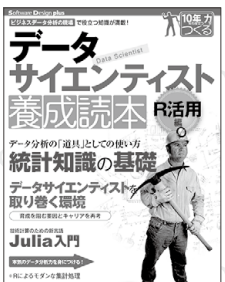
遠山勝乃 著
B5変形判・392ページ
定価 3,500円(本体)+税
ISBN 978-4-7741-6571-4

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



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

川瀬裕久、古川文生、松尾大、竹澤有貴、小山哲志、新原雅司 著
B5判・156ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7313-9



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

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
B5判・168ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6931-6

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

技術評論社　〒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

Erlangで学ぶ並行プログラミング

Author 力武健次技術士事務所 所長 力武 健次(りきたけ けんじ) http://rikitake.jp/

第6回 Erlang/OTPでのプロセス状態監視とテスト

この連載ではプログラミング言語Erlangとその並行プログラミングについて紹介していきます。今回はプロセス状態監視やテストの手法について紹介します。

最新版の状況

17.x系は17.5.6.2[1]が7月7日に、18.x系は18.0.2[2]が7月8日にリリースされました。GitHubリポジトリでタグを指定することでソースコードからビルドできます(詳細は連載第3回の注8で説明)。ランタイムシステム(ERTS)のバグ修正や時間の取り扱い変更に伴う修正が加えられています。

稼動プロセスをobserverで状態監視する

Erlangの仮想マシンでは多くのプロセスが同時並行に動いています。これを網羅的に状態監視するのは決して易しくはありませんが、幸いOTPには状態監視用の各種ツールが標準で用意されています。ここでは代表的なobserverというツールを見ていきます。

筆者は2014年にerltrek[3]というテキスト版スタートレックゲーム[注1]をErlang/OTPで開発しています(**図1**)。**図1**[注2]でerltrekのゲームを起動した直後に、observerを立ち上げると**図2**の画面が表示されます。ここでボタン[Applications]をクリックすると、仮想マシンに実行中の各アプリケーションとそれぞれに属するプロセスのツリー図が表示されます(**図3**)。各プロセスの状態はボタン[Processes]をクリックすると表示されます(**図4**)。**図4**では登録済みプロセスは登録された名前で、その他は起動した関数の情報が表示されています。

図4のプロセス一覧表で、詳細表示したいものをクリックすると、さらに細かい情報を得ることができます。試しにerltrek_galaxyという登録済みプロセスについて表示してみます(**図5**)。プロセスごとのメモリ使用状況やほかのプロセスとのリンクやモニタの関係が一目でわかります。このプロセスが保持している内部状態はボタン[State]をクリックすると表示できます(**図6**)[注3]。**図6**のラベルStateの項目を見ると、内部状態が省略された形で出ています。これは最下行の[Click to expand above term]をクリックするとさらに展開した形で見ることができます(**図7**)。

observerではこのほかにもスケジューラやI/O、メモリ消費の状態について表示できます。今回は同じBEAMの中で実行しましたが、別の

注1) 1970年代に作られた宇宙船間の戦闘ゲームです。FreeBSDではPortのgames/bsdgamesをインストールすれば"trek"でBSD UNIX版を起動できます。Linuxにも対応させたものは(https://github.com/jj1bdx/bsdtrek)にて公開しています。

注2) erltrekは筆者が発表した直後に、ErlangによるWebフレームワークの1つZotonic[4]のコア開発者Andreas Stenius氏によって、宇宙船やゲーム空間の銀河の状況が、すべてErlangのプロセスで表現されるように改良されました。なお、実行にあたってはWindowsでは表示ドライバが動作しないため、Linux/OS X/FreeBSDなどが必要です。

注3) プロセスの内部状態を表示するには、対象となるプロセスでgen_serverなどが動作していること、つまりsysモジュールに定義するシステム標準のメッセージに準拠したメッセージに対して送受信処理ができることが必要です。

▼図1　erltrekの実行とobserverの起動

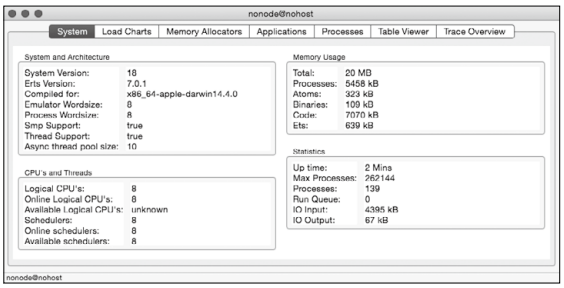
```
Erlang/OTP 18 [erts-7.0.1] [source] [64-bit] [
[smp:8:8] [async-threads:10] [hipe] [kernel-
poll:false] [dtrace]
↑実行は./game.shというコマンドで起動している

ErlTrek Shell (abort with ^G)
Command > Short range sensor scan
  0 1 2 3 4 5 6 7 8 9
0 . . * . . . . . H . 0  Stardate:      2000.00
1 . . . . . . K * . * 1  Position:      2,2/7,5
2 . . # . . @ . . . . 2  Condition:     GREEN
3 . . . . . . . . K . 3  Energy:        5000
4 . . . . . . . . . . 4  Shield:        1000
5 . . . . . . . . . . 5  Klingons:      44
6 H . . . * . . . . * 6
7 . . . . * E . . . . 7
8 . . . . . * . . . . 8
9 . K . . . . . . . . 9
  0 1 2 3 4 5 6 7 8 9
Starsystem Ardana
Condition changed to: RED
↑放っておいてもゲームは勝手に進む

Klingon hit with phasers from sector 3,7 level 62
Shield level down to 938
Klingon hit with phasers from sector 1,6 level 52
Shield level down to 886
 ……(中略)……
User switch command
↑ここでコントロールGを押して別のシェルを起動する
 --> ?
  c [nn]            - connect to job
  i [nn]            - interrupt job
```

右段へ続く↗

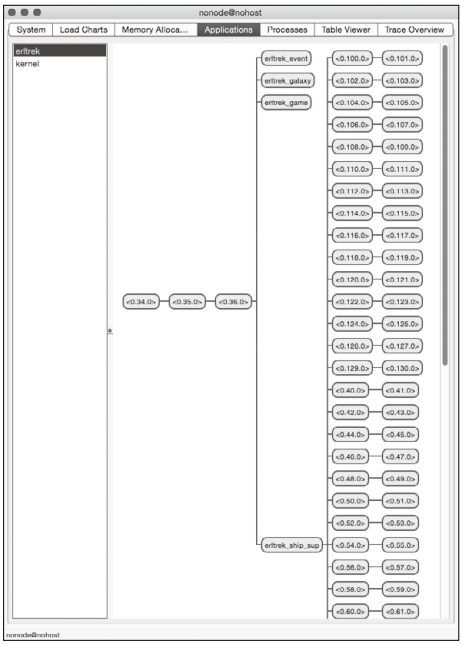
左段下から続く↙

```
  k [nn]            - kill job
  j                 - list all jobs
  s [shell]         - start local shell
  r [node [shell]]  - start remote shell
  q                 - quit erlang
  ? | h             - this message
 --> j ←現在のジョブの状況
   1* {erltrek_shell,start,[]}
 --> s ←新しくローカルシェルを起動する
 --> j ←ジョブの状況を見るとシェルが増えるのがわかる
   1  {erltrek_shell,start,[]}
   2* {shell,start,[]}
 --> c 2 ←2番のローカルシェルに接続する
Eshell V7.0.1  (abort with ^G)
1> observer:start(). observerを起動する
ok
2> ←ここでコントロールGを押してゲームに制御を戻す
User switch command
 --> c 1  ←【Enter】押すとゲームのプロンプトに戻る
Command > s
Short range sensor scan
  0 1 2 3 4 5 6 7 8 9
0 . . * . . . . . H . 0  Stardate:      2003.57
1 . . . . . . K * . * 1  Position:      2,2/7,5
2 . . # . . @ . . . . 2  Condition:     RED
3 . . . . . . . . . . 3  Energy:        5000
4 . . . . . . . . . . 4  Shield:        648
5 . . . . . . . . . . 5  Klingons:      44
6 H . . . * . . . . * 6
7 . . . . * E . . . . 7
8 . . . . . * . . . . 8
9 . . . . . . . . . . 9
  0 1 2 3 4 5 6 7 8 9
Starsystem Ardana
```

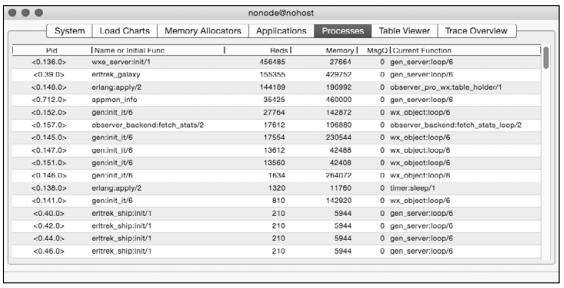
▼図2　observer起動時画面



▼図3　アプリケーションerltrekと関連プロセスのツリー図



▼図4　実行中プロセスの一覧



ノードから接続して状態監視することもできます。実行にはGUI環境が必要なのが難点ではありますが、BEAMの中で何が起こっているかを把握するには最初に試してみるべきツールでしょう。

observerで実行できることの多くは、前回紹介したsysモジュールの機能を使っても実現できます。sysモジュールはGUI環境がなくても実行できるので便利です。このほかにもプロファイリングツールとしてcprof/eprof/fprofの各モジュールがあり、これらを使って実行コードのうちに、どこに時間がかかっているかを調べることができます。詳しくは各モジュールのマニュアルを参照してください。

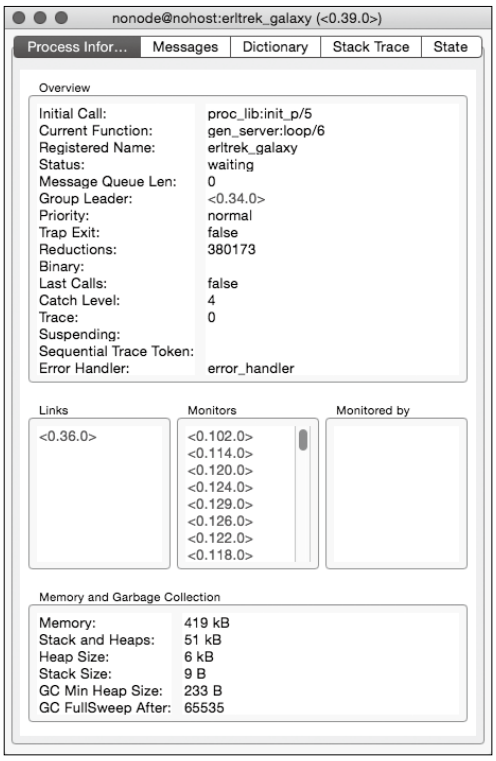
単体テストツールEUnit

OTPにある各種テストツールのうち代表的なものとして、EUnit[5]を紹介します。EUnitはプログラムの個々の関数が正しく動作しているかどうかをテストする単体テストのための自動化ツールです。EUnitは各種操作の実行結果が正しいかどうかを判定するマクロや、テスト前後の処理を自動化するための機能を備えています。

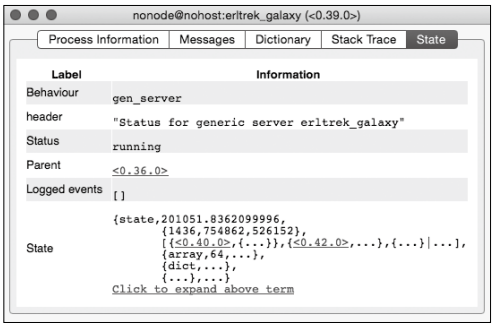
EUnitは独立したアプリケーションとしても使えますが、ここではビルドツールrebar[6]を使って単体テストを行う例を示します。rebarはErlangで書かれたビルドツールで、各種依存関係を解決したり、テストツールの起動やリリースやアプリケーションの作成を支援する機能を持っています。

早速、前回(第5回)で作成したカウンタのプログラムをテストしてみることにします。具体的なrebarの設定の仕方を**図8**に示します。rebarはテンプレートを生成する機能があるため、それに従ってapp.srcファイルを作り、ソー

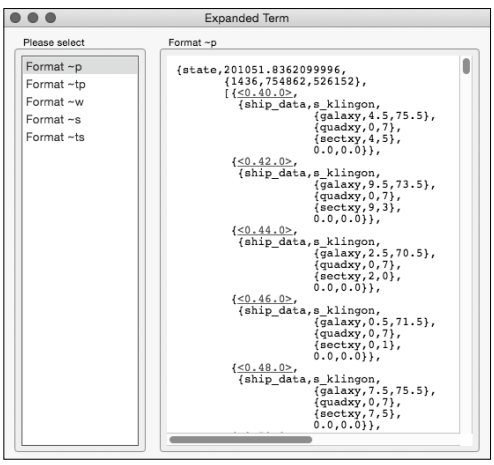
▼図5　登録済みプロセスerltrek_galaxyの詳細



▼図6　プロセスerltrek_galaxyの内部状態の概要



▼図7　プロセスerltrek_galaxyの内部状態の詳細表示



スの.erlファイルをsrc/の下に置けばコマンド1つでコンパイルとテストを実行できます。また、rebar.configに設定を書くことにより、EUnitの設定もできるようになっています。

テストをどのように書くかについては、**リスト1**を参照してください。このリストでは、テストの前準備と後片付けは直接実行する関数として定義し、テスト自身はテストセットを生成する関数を定義してそれをEUnitで実行する方法を取っています。このテストセットを生成する方法を取ることで、複数のテストセット生成関数を組み合わせたテストを行うことも可能になります。

Erlang/OTPではEUnitのほかにCommon Test[7]という単体テストだけでなく複数のノード間にわたるテストも可能なしくみが用意されており、OTP自身のテストはCommon Testで書かれています。

属性テストツールQuickCheck

プログラムのテスト手法として、属性テスト(property test)というのがあります。これは従来のテストツールでは発見し得なかったバグを見つけるのに役立ちます。EUnitなどのテストツールはあらかじめプログラムで列挙されたテ

▼図8　rebarの簡易的設定

```
git clone https://github.com/rebar/rebar.git
rebarをビルドする
cd rebar
./bootstrap
これでrebarという実行形式ファイルができるのでPATHの通っているディレクトリに置いておく

ターゲットディレクトリ(sd_ep06_eunitとする)に移動してrebarの実行準備をする

cd sd_ep06_eunit
mkdir src ebin

rebarでアプリケーション設定ファイルの雛形を作る

rebar create-lib
これでsd_ep06_eunit/src/mylib.app.srcというファイルができたのでこれをもとに次のとおりのsd_ep06_eunit/src/sd_ep06_eunit.app.srcというファイルを作る

sd_ep06_eunit/src/sd_ep06_eunit.app.src
{application, sd_ep06_eunit,
[
{description, "An Erlang mylib library"},
{vsn, "1"},
{modules, [
このリストにはコンパイルに必要なモジュール名を書いておく
          msgcounter_gen_server,
          sd_ep06_eunit
         ]},
{registered, []},
{applications, [
                kernel,
                stdlib
               ]},
{env, []}
]}.

sd_ep06_eunit/src/sd_ep06_eunit.app.src はここまで

sd_ep06_eunit/src/にはmsgcounter_gen_server.erlとsd_ep06_eunit.erlの2つのファイルがある。またsd_ep06_eunit/rebar.configにrebarでEUnitのカバレッジを有効にする次の設定をしておく
```

右段へ続く↗

左段下から続く↙

```
sd_ep06_eunit/rebar.config
{cover_enabled, true}.
sd_ep06_eunit/rebar.config はここまで

この状態でディレクトリsd_ep06_eunitに戻り次のコマンドを実行するとコンパイルの後にEUnitによるテストが実行される実行結果の例は次のとおり

rebar compile eunit
==> sd_ep06_eunit (compile)
Compiled src/sd_ep06_eunit.erl
Compiled src/msgcounter_gen_server.erl
==> sd_ep06_eunit (eunit)
Compiled src/msgcounter_gen_server.erl
Compiled src/sd_ep06_eunit.erl
  All 10 tests passed.
  Cover analysis: /home/example/src/sd_ep06_
eunit/.eunit/index.html

この状態で.eunit/index.htmlをWebブラウザで開くとテストカバレッジの状態がわかる。次にmsgcounter_gen_server.erlについての要旨を抜粋。縦線の左側は実行された回数

1..|      gen_server:start_link(?MODULE, [],
[]).
4..|      gen_server:call(Pid, inc).
2..|      gen_server:call(Pid, dec).
1..|      gen_server:call(Pid, zero).
3..|      gen_server:call(Pid, val).
1..|      gen_server:call(Pid, terminate).
1..|      {ok, #state{counter = 0}}.
4..|      {reply, Count + 1, #state{counter
= Count + 1}};
2..|      {reply, Count - 1, #state{counter
= Count - 1}};
1..|      {reply, ok, #state{counter = 0}};
3..|      {reply, Count, S};
1..|      {stop, normal, ok, S}.
1..|  terminate(normal, _S) -> ok.
カバレッジのない以下3行はブラウザでは赤字で表示される
0..|  handle_cast(_Msg, S) -> {noreply, S}.
0..|  handle_info(_Info, S) -> {noreply, S}.
0..|  code_change(_OldVsn, S, _Extra) ->
{ok, S}.
```

▼リスト1　sd_ep06_eunit.erl

```
EUnitによるテストケースのモジュールです
-module(sd_ep06_eunit).
-ifdef(マクロ) ... -endif. の間はマクロが定義されている場合のみコンパイルされる。EUnitではTESTというマクロを定義している
-ifdef(TEST).
EUnitに必要な定義ファイルをインクルードする
-include_lib("eunit/include/eunit.hrl").
-endif.
テストコード
-ifdef(TEST).
最後が"_test_"で終わっている関数はテストを生成する役割をする
counter_test_() ->
このsetupで始まるタプルで次の定義をする。*テスト全体はcounter_check/1で返されるテストセットで実行。テスト実行前にcounter_start_link/0を実行。テスト終了後はcounter_stop/1を実行
    {setup,
     fun counter_start_link/0,
     fun counter_stop/1,
     fun counter_check/1
    }.
カウンタのgen_serverを起動してPidを返す
counter_start_link() ->
    {ok, Pid} = msgcounter_gen_server:start_link(),
    Pid.
Pidで示されたカウンタを停止する
counter_stop(Pid) ->
    msgcounter_gen_server:stop(Pid).
Pidで示したカウンタに対するテストセットを返す
counter_check(Pid) ->
    [
      初期値が0であるかどうかのテストを返す
    ?_assertEqual(0, msgcounter_gen_server:val(Pid)),
      1つ増やしたら1になるかどうかのテストを返す
    ?_assertEqual(1, msgcounter_gen_server:inc(Pid)),
    ?_assertEqual(2, msgcounter_gen_server:inc(Pid)),
      ゼロに戻す関数の戻り値がokであるかどうかのテストを返す
    ?_assertEqual(ok, msgcounter_gen_server:zero(Pid)),
    ?_assertEqual(0, msgcounter_gen_server:val(Pid)),
    ?_assertEqual(1, msgcounter_gen_server:inc(Pid)),
    ?_assertEqual(0, msgcounter_gen_server:dec(Pid)),
    ?_assertEqual(-1, msgcounter_gen_server:dec(Pid)),
    ?_assertEqual(0, msgcounter_gen_server:inc(Pid)),
    ?_assertEqual(0, msgcounter_gen_server:val(Pid))
      ここまでで10個のテストを定義したことになる
    ].
 ここまででテストケースのモジュールは終了
-endif.
```

ストケースの実行結果を判定します。これに対し、属性テストではテストケースそのものを与えるのではなく、テストケースが満たすべき条件を与え、その範囲内での各種テストを大量かつランダムに実行してバグを見つけだそうとする点が違っています。その意味では、属性テストは処理を関数ごとに分けてその入出力を定義するという関数型プログラミングによく適したテスト手法と言えます。

属性テストツールを使うには、まず各関数の満たすべき属性を定義します。たとえば前述のカウンタのプログラムであれば、inc/1という関数の満たすべき属性は「実行後にカウンタの値が1増えていること」と定義できます。これらの属性の定義を必要な関数ごとに列挙し、かつそれぞれの関数の入力が取り得る値の範囲を定義して属性テストツールに与えることが、属性テストを書くという具体的作業になります。

今回紹介する属性テストツールQuickCheck[8]はもともとはHaskellのためのツールとして作られましたが、現在は同名のツールが他の各種言語向けに開発されています。今回はQuickCheck CIを使います[注4]。これを使うためには、GitあるいはMercurialの公開リポジトリにソースを置き、必要なライセンスファイルを置くことでライセンス条件に同意を示した上で、テストの指示の設定をするファイルを書く必要があります[注5]。

今回はカウンタの試験のために4つの関数についてテスト定義を書きました(**リスト2**)。**図9**に実行結果を示します。ここでは1,000回テストを実行して、すべて成功していることがわかります。また、どの関数をどのような頻度で実行したかもパーセント表示され、各テストがどのような関数の順序で実行されたという詳細の抜粋も表示されます。

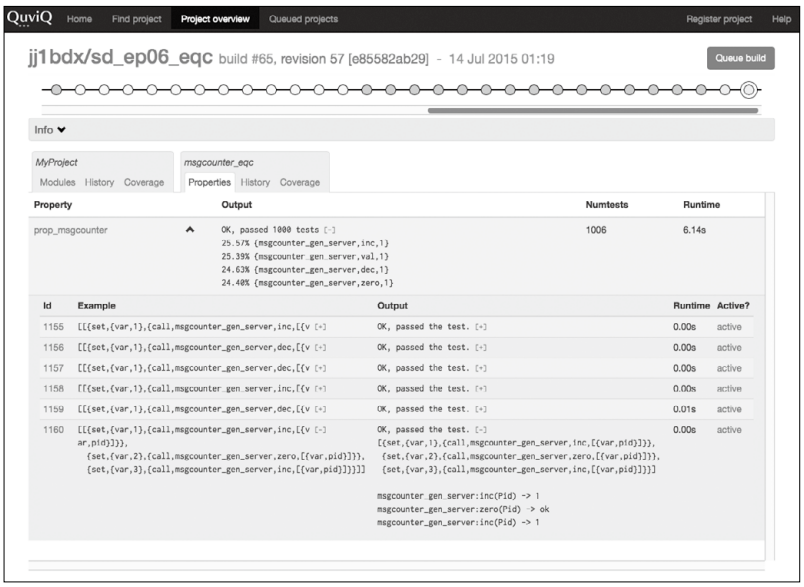
属性テストはテスト定義が関数の動作仕様そのものとなるため、実は関数の動作を理解したり記述していくうえでも有用な手法です。大変難しいテスト手法の1つ[注6]ですが、一度覚えると手放せなくなるのも事実です。

注4) Erlangでの QuickCheck実装はQuviq社のErlang QuickCheck[9](有償の非公開製品)そしてこれをOSSとして公開されているソフトウェアに対し無償で提供しているQuickCheck CI[10]、さらにQuickCheck互換の属性テストツールとしてOSSの形で実装したpropEr[11]やTriq[12]があります。

注5) 本稿ではQuickcheck CIの設定の詳細は割愛しますが、執筆に使った公開レポジトリ(サポートページ欄参考)がありますので、設定の詳細はそちらを参照してください。

注6) 筆者はQuickCheckの商品版のトレーニングを2015年3月末にQuviqとErlang Solutions両社のご厚意で米国サンフランシスコにて受ける機会がありましたが、テストケースを生成するためのルールを書くという発想に慣れるのは難しく、とても時間がかかりました。それでも3日間ホテルに缶詰になって学習した成果はあったと筆者は感じています。

▼図9　QuickCheck CIの実行結果



▼リスト2　QuickCheck CIに対する定義ファイル

左段下から続く↙

```
msgcounter_gen_serverモジュールの属性テストの例（抜粋のため各種定義は一部割愛）。QuickCheck CIの中で保持する内部状態を定義する。ここではカウンタのみとする
-record(state, {count = 0}).
テストの初期状態を定義する
initial_state() -> #state{}.

関数zero/1へのテストの定義。zero_command/1 では msgcounter_gen_server:zero/1を呼ぶ指示をする。{var,pid}と書いているのは後述のrun_commands/3の中で与えられるPidの値をzero/1の第1引数として与えよという意味
zero_command(_) ->
    {call, msgcounter_gen_server, zero, [{var, ▽
pid}]}.

zero_next/3では関数実行後の内部状態を定義する。ここではcountの値をゼロにするという内部状態を定義
zero_next(S, _, _) -> S#state{count = 0}.

zero_post/3では関数実行後に満たすべき条件を定義する。ここではカウンタがゼロに等しいことと返り値がokに等しいことを定義している
zero_post(S, _, Result) ->
    eq(S#state.count, 0),
    eq(Result, ok).

関数inc/1へのテストの定義
inc_command(_) ->
    {call, msgcounter_gen_server, inc, [{var, ▽
pid}]}.
ここではcountの値が前より1増えるという内部状態を定義している
inc_next(S, _, _) ->
    S#state{count = S#state.count + 1}.
ここではcountの値が実行後にはそれ以前より1増えるという条件を定義している
inc_post(S, _, Result) ->
    eq(Result, S#state.count + 1).
関数dec/1へのテストの定義
dec_command(_) ->
    {call, msgcounter_gen_server, dec, [{var, ▽
pid}]}.
ここではcountの値が前より1減るという内部状態を定義している
dec_next(S, _, _) ->
    S#state{count = S#state.count - 1}.
ここではcountの値が実行後にはそれ以前より1減るという条件を定義している
dec_post(S, _, Result) ->
    eq(Result, S#state.count - 1).
関数val/1へのテストの定義
val_command(_) ->
    {call, msgcounter_gen_server, val, [{var, ▽
pid}]}.
この関数の実行によって内部状態は変えないという定義をしている
val_next(S, _, _) -> S.
この関数の実行によって内部状態は変わっていないという条件を定義している
val_post(S, _, Result) ->
    eq(Result, S#state.count).

実際の属性テストのコード。詳細は省くが、numtests/2の第1引数で1000回テストを繰り返すことを指示し、このモジュールの中に書いてあるテスト定義（コマンド）のすべてに対し次のbegin - end節の中の内容をテストとして実行する
prop_msgcounter() ->
    numtests(1000, ?FORALL(Cmds, commands(?MODULE),
      begin
        カウンタのgen_serverを起動する
        {ok, Pid} = msgcounter_gen_server: ▽
start_link(),
        あらかじめ定義したコマンドをランダムに複数実行する。テスト定義の中でpidとして値を参照できるようにする
        {H, S, Res} = run_commands(?MODULE, Cmds, ▽
[{pid, Pid}]),
        コマンド群の実行後サーバを停止する
        msgcounter_gen_server:stop(Pid),
        結果がokでない場合は失敗したコマンドを集約して表示する
        pretty_commands(?MODULE, Cmds, {H, S, Res},
                        aggregate(command_ ▽
names(Cmds), Res == ok))
      end)).
以上で属性テストの定義の主要部分は終了
```

右段へ続く↗

まとめ

今回はErlang/OTPでのプロセス状態監視とテストのための各種ツールについて紹介しました。

ここまでの連載では、Erlang/OTPの持つ並行プログラミング支援の言語機能と各種ツールについてその概要を説明してきました。Erlang/OTPの世界はとても奥深く、すべてを紹介するのは困難ですが、この連載が皆様の理解の一助になれば幸いです。

ソースコードとサポートページ

連載の記事で紹介したソースコードなどGitHubのリポジトリに置いています(https://github.com/jj1bdx/sd-erlang-public/)。

また、今回はQuickCheck CIのためのリポジトリも用意しました(https://github.com/jj1bdx/sd_ep06_eqc/)。どうぞご活用ください。

SD

参考文献

[1] http://erlang.org/pipermail/erlang-questions/2015-July/085082.html
[2] http://erlang.org/pipermail/erlang-questions/2015-July/085097.html
[3] https://github.com/jj1bdx/erltrek
[4] http://zotonic.com/
[5] EUnit User's Guide, http://erlang.org/doc/apps/eunit/chapter.html
[6] https://github.com/rebar/rebar/
[7] Common Test User's Guide, http://erlang.org/doc/apps/common_test/basics_chapter.html
[8] https://en.wikipedia.org/wiki/QuickCheck
[9] http://www.quviq.com/products/erlang-quickcheck/
[10] http://quickcheck-ci.com/
[11] http://proper.softlab.ntua.gr/ ソースコードはhttps://github.com/manopapad/proper
[12] http://krestenkrab.github.io/triq/

思考をカタチにするエディタの使い方

るびきち流 Emacs超入門

Writer | るびきち | twitter@rubikitch | http://rubikitch.com/

第17回 “手遅れ”を防ぐ Emacsのセーフガードシステム

今回は、Emacsにおいて“いかに安全な編集を実現するか”に焦点を絞ってEmacsのセーフガード(安全装置)システムを紹介していきます。undo機能の強化、カーソル位置・ウィンドウ構成の引き戻しコマンド、ファイルの自動バックアップを設定することで、致命的なミスを格段に減らせます。

Emacsのセーフガードシステム

Emacsが歴史のあるソフトウェアであることは今さら言うまでもありません。歴史があるということは、それだけ多くの人に使われていて、長年に渡るノウハウが蓄積されていることを意味します。昔から多くの人が不満に思っている点は、たいてい解決されています。

今回はEmacsで使える多くのセーフガードシステムを標準・外部パッケージ問わず紹介していきます。人間は操作ミスをする生き物ですので、Emacsではその被害をなくしたり最小限に抑えたりするための方法が多くあります。昔から「保存し忘れたからフリーズしたときにデータが飛んだ」といった悲鳴をよく聞きますが、Emacsならばそんなことは太古の昔に解決されているのです。

Emacsの状態を元に戻す機能も大切なセーフガードです。けれども、せっかくセーフガードシステムが用意されていても知らなかったのでは意味がありません。今月は次のテーマを採り上げます。

- 間違った編集を元に戻す
- 迷子になったカーソルを戻す
- ウィンドウ構成を戻す
- 自動保存でデータ消失をなくす

編集を元に戻す

標準のundo

通常、Emacsのバッファは作成時からの編集履歴を記憶しています。間違った編集をしてバッファの内容がめちゃくちゃになったとしても、あわてずにC-/(undo)を押してください。直前の編集状態に戻ります。もっと前に戻したい場合は引き続きC-/を押します。もし戻し過ぎてしまった場合は、C-fなど編集を行わないコマンドを実行後、C-/を押します。

undoをカイゼンするredo+パッケージ

標準のC-/は戻し過ぎてしまった場合の挙動が使いやすくありません。というのは、過去方向だけでなく未来方向に進むことがあるからです。redo+パッケージはundoを過去方向のみ遡(さかのぼ)れるように再定義し、未来方向に進むM-x redoを定義します。これはMELPAに登録されているのでM-x package-install redo+でインストールして次の設定を加えてください。

```
(require 'redo+)
(global-set-key (kbd "C-M-/") 'redo)
```

この設定を加えた場合、C-/を押し過ぎてしまった場合にC-M-/で戻せるようになります。

編集履歴を永続化するundohistパッケージ

編集履歴は通常、バッファ作成時に初期化されます。つまり、バッファを削除したり、ファイルを開き直したり、Emacsを終了したときには失われてしまいます。

MELPAのundohistパッケージはバッファを削除する際に編集履歴をディスクに保存し、再びファイルが開かれるときにそれを復元します。インストール後、次の設定を加えてください。

```
(require 'undohist)
(undohist-initialize)
;;; 永続化を無視するファイル名の正規表現
(setq undohist-ignored-files
      '("/tmp/" "COMMIT_EDITMSG"))
```

編集履歴をツリー状に可視化するundo-tree

C-/で編集を戻して新しく編集しなおすと、編集履歴は分岐することになります。つまり、戻った時点から見て間違った古い履歴と、次の編集で作られる新しい履歴ができます。新しい履歴が作られたとき、通常のC-/では古い履歴にアクセスできません。編集履歴のデータには古い履歴も保持されているのに、もったいないですよね。

そこでMELPAのundo-treeパッケージを使って編集履歴の木構造を「見える化」します(**図1**)。もともとundoはC-x uとC-/に割り当てられていますが、undo-treeを導入すると両者に別の役割が与えられます。

C-/にはundo-tree版undoであるM-x undo-tree-undoが、C-x uには編集履歴の木構造にアクセスするM-x undo-tree-visualizeが割り当てられます。木構造では直観的に操作できます。C-p／C-nで履歴を時系列順にたどり(上下)、C-b／C-fで分岐を選択(左右)します。なお、M-x redoをキーに割り当てている場合は、そのキーにM-x undo-tree-redoが割り当てられます。インストール後、次の設定を加えてください。

```
(setq undo-tree-mode-lighter "")
(global-undo-tree-mode 1)
```

カーソル位置を戻す

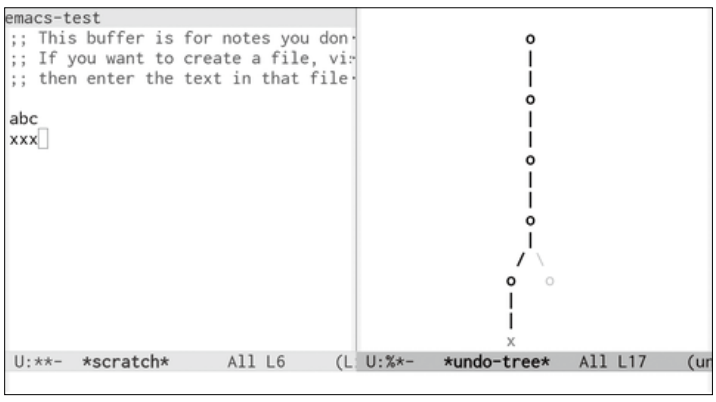
C-u C-SPCでマークを戻す

Emacsにはさまざまなカーソル移動コマンドがありますが、長距離を移動するコマンドを実行した場合に、すぐに元の位置に復帰できるしくみになっています。つまり、行きはよくても帰りがわからない——カーソルが迷子になる——ことがないようなセーフガードが設けられています。本節の内容は以前の連載(2015年6月号など)でも触れましたが、セーフガードの観点から再び採り上げることにします。

長距離移動とは、バッファ先頭・末尾への移動(M-<、M->)やインクリメンタルサーチ(C-r、C-s)や関数単位の移動(C-M-a、C-M-e)などが該当します。

これらのコマンドを実行するとき、あらかじ

▼図1　C-x uで編集履歴にアクセス



め“暗黙のマーク”によって元のカーソル位置を記憶します。そして、C-u C-SPCを押せば元の位置に戻れます。たとえば、バッファ先頭を見てからすぐ戻る場合、M-<のあとにC-u C-SPCを押せばいいだけです。C-SPC M-<と明示的にマークする必要はありません。

暗黙のマークの存在を知っていれば、作業がとても楽になります。たとえば、ソースコードにrequire(必要なライブラリの宣言)を書き加える場合、requireまでインクリメンタルサーチで移動し、書き加え、C-u C-SPCで戻れるのです。

何かしらの理由でカーソル位置が思いもよらない場所に移動してしまった場合、暗黙のマークがしてあればC-u C-SPCを押せば元の位置に戻れるのです。

マークは複数個記憶しているので、C-u C-SPCを繰り返せばどんどん過去のマークへ移動できます。何度もC-uを付けるのが面倒であれば次の設定を加えると良いでしょう。

```
(setq set-mark-command-repeat-pop t)
```

これによりC-u C-SPC C-SPC……とC-SPCを連打して遡れます。

カーソル位置を戻す point-undoパッケージ

C-u C-SPCはカーソル迷子の万能薬ではありません。C-v／M-vなどの画面スクロールは暗黙のマークを設定しません。また、外部パッケージによるコマンドも必ずしも暗黙のマークを設定するとは限りません。

MELPAにある拙作point-undoパッケージは、すべてのカーソル移動を記憶することで、暗黙のマークに関係なくカーソル位置を戻します。見た目上違和感のないように、カーソル位置だけでなくウィンドウの表示位置も復元します。

次の設定ではF7でカーソル位置を過去方向に戻し、M-f7で戻し過ぎたカーソル位置を未来方向へ進めます。

```
(require 'point-undo)
(global-set-key [f7] 'point-undo)
(global-set-key [M-f7] 'point-redo)
```

お使いの環境によってはF7やM-f7がEmacsで使えないこともあるので、そのときはほかのキーに割り当ててください。そのときはF7とM-f7のように対になるキーバインドをお勧めします。

編集履歴からカーソル位置をたどる goto-chgパッケージ

前節で示したように、Emacsは編集履歴を記憶しています。編集履歴とは、変更された場所とその内容の集まりです。

MELPAにあるgoto-chgパッケージは、編集履歴の中の変更された場所にアクセスすることで、変更された位置を遡ります。これは、慣れるまで動作がイメージしづらいかもしれませんが、バッファに変更を加えた時点で自動的に編集履歴に記録されることを理解すれば良いです。

手動のC-SPCに対して、自動のgoto-chgです。goto-chgは別な場所を編集してから元の場所を編集するときに効果を発揮します。たとえば文章を書いていて、ふと誤字脱字が目に付いて、そこを修正したあと、ふたたび元の位置に戻って文章の続きを書くといったケースです。

次の設定ではF8で編集履歴に記憶された場所を過去方向にたどり、M-f8で未来方向にたどります。

```
(require 'goto-chg)
(global-set-key [f8] 'goto-last-change)
(global-set-key [M-f8] 'goto-last-change-
reverse)
```

ウィンドウ構成を戻す

以前のウィンドウ構成に戻す 標準機能winner

Emacsを使っているとウィンドウ構成がめま

ぐるしく変わります。ウィンドウ構成とは、画面の分割状態とバッファの配置のことです。ウィンドウを分割したり、大きさを変更したり、別なバッファを表示させたときにウィンドウ構成が変更されます。

たとえば F1 f（関数の説明表示）などのヘルプを表示した場合、ヘルプバッファの内容を読んだあとにしたいことは、元のウィンドウ構成に戻すことです。ヘルプバッファはGUIにたとえればメッセージダイアログボックスがポップアップすることに相当します。ダイアログボックスは元の画面に重なるように表示され、「閉じる」を押せば消滅します。

M-x winner-undoは「閉じる」ボタンに相当するコマンドで、直前のウィンドウ構成に戻します。

ほかの「戻す」系パッケージと同様に逆方向のM-x winner-redoもあります。マイナーモードwinner-modeが有効のときのみ動作するので、**リスト1**の設定を記述する必要があります。

Emacsにポップアップウィンドウという概念を導入するpopwinパッケージが人気ですが、筆者はwinnerのほうが応用範囲が広いと感じています。popwinでは、popwinで表示されたウィンドウのみを閉じますが、winnerはすべてのウィンドウが対象です。popwinではC-gでポップアップウィンドウを閉じられますが、それを実現するためにかなり複雑なしくみになっています。ウィンドウを閉じることくらい、標準機能でも間に合います。

特殊状態を“取り止め”るには ESC(C-[)を3回叩け！

あなたは、ESC ESC ESCに割り当てられたコマンドをご存じですか？　Emacs的に意外なキーに割り当てられたkeyboard-escape-quitというコマンドはマルチな機能を発揮します。コマンド名からしてC-g（keyboard-quit）に似ていることは想像できますが、共通点と相異点があります。

▼リスト1　winner-modeの設定

```
(winner-mode 1)
(global-set-key (kbd "C-z") 'winner-undo)
(global-set-key (kbd "C-M-z") 'winner-redo)
;;; 後述のESC ESC ESCで使う場合
(setq buffer-quit-function 'winner-undo)
```

C-gとESC ESC ESCの共通点は次のとおりです。

・regionをキャンセルする
・ミニバッファから抜ける
・前置引数をキャンセルする

これらの用途では素直にC-gで間に合います。

一方、ESC ESC ESCの独自機能は次のとおりです。

・再帰編集から抜ける
・バッファを閉じる（閉じ方は設定可能）
・ウィンドウが分割されているときは分割を解除する

このうち「バッファを閉じる」アクションにwinner-undoを指定すれば、ESC ESC ESCでウィンドウ構成を戻せるようになります。バッファを閉じる関数はbuffer-quit-functionに指定します。デフォルトではnilになっていて、そのときはウィンドウ分割を解除、または隠しバッファ（バッファ名がスペースから始まる）を閉じるようになります。

「再帰編集」とは、コマンド実行中にバッファを編集できるようにする機能です。代表的な再帰編集は置換で起こります。M-%やC-M-%の途中でC-rを押せば置換は中断されて再帰編集に入り、ESC ESC ESCで抜けるまでEmacsの任意の操作ができます。再帰編集を抜けると、実行中のコマンドが再開されます。再帰編集は、モードラインのモード名が[]で囲まれているかどうかで判別できます。

前項でbuffer-quit-functionをwinner-undoに設定しているので、ESC ESC ESCは再帰編集から抜けるか、ウィンドウ構成を戻すコマンド

になります。単に前のバッファに戻りたいのであれば、buffer-quit-functionにprevious-bufferを設定してください。

ESC ESC ESCの定義はとてもシンプルなもので、cond式一発で構成されています。condは他言語でいうif～elseif～elseif～elseに相当する構文で、条件式を次々にチェックしていき、最初に一致した条件式にマッチする挙動を行います。elispが読める方は`M-x find-function-on-key ESC ESC ESC`でコードを読んだほうがより確実に理解できるでしょう。なお、標準添付のlispソースコードがインストールされていない環境ではエラーになるので、Emacsについて深く学びたい方はインストールしてください。Debian系列ではemacs24-elパッケージが必要です。

ウィンドウ構成を記憶させるいろいろな方法

Emacsであらゆるタスクを同時進行させている人ならば、ウィンドウ構成を戻すよりも、記憶させたくなると思います。ウィンドウ構成を記憶させる方法はいろいろあり、そのためのパッケージもたくさん存在します。本稿はあくまでもセーフガードを主題としているので軽く触れておくにとどめます。

標準の方法はレジスタにウィンドウ構成を記憶させることです。`C-x r w`で記憶し、`C-x r j`で復元します。

ウィンドウ構成を記憶する無難なパッケージはelscreenです。GNU Screenを模倣して作られていて、ウィンドウ構成をscreenに見立てて、切り替えられるようになっています。

helmにもelscreenに関するアクションが存在します。elscreen-persistパッケージでEmacsを終了してもscreenの状態を復元できます。elscreen-separate-buffer-listパッケージでscreenごとに独立したバッファリストを持てるようになります。elscreen + elscreen-persist + elscreen-separate-buffer-listと似たパッケージとして、persp-modeパッケージがあります。

大昔からあるwindows.el（未パッケージ化）は20年以上に渡っていまだにメンテナンスされています。ウィンドウ構成の情報や開いたバッファも永続化されます。

ウィンドウ構成に名前を付けるだけならばspacesパッケージやwindataパッケージがあります。

ぜひとも好みに応じてパッケージを選んでみてください。パッケージを自由に選べるのもEmacsの魅力です。

自動保存でデータ消失を防ぐ

標準のauto-save-mode

最後のテーマはデータ消失を防ぐ方法についてです。まさに「セーフガード」にぴったりな話題です。

Emacsでは標準でauto-save-modeという自動保存機能が有効になっています。この自動保存は編集中のファイルに直接保存されるのではなく、別のファイルに保存されます。一般に自動保存のファイル名は元のファイル名を#で囲んだものになります。たとえば、「foo.c」ならば「#foo.c#」となります。diredを使っていると、もしかしたらこのようなファイルを見掛けたことがあるかもしれません。普通のファイルならばカレントディレクトリに作られますが、Trampによるリモートファイル（sshやftpやsudoなど）は/tmpに置かれます。

自動保存ファイルは`C-x C-s`などで明示的に保存されたときに消去されます。保存せずにバッファを削除しようとしたり、Emacsを終了しようとしたりといったときには、本当に削除・終了するかわざわざ訊いてくるので、保存忘れによる最低限のセーフガードはできています。もし保存せずにバッファを削除した場合、自動保存ファイルが残っているので、そこからある程度の内容を復元できます。そのため、保存忘れ

に対しては二重のセーフガードになっています。

そして、何よりうれしいのがEmacsやOSがフリーズしたときに、自動保存に助けられるということです。自動保存ファイルが残っているファイルを開いたとき、

```
ファイル名 has auto save data; consider ➡
M-x recover-this-file
```

と教えてくれます。内容が古い場合はM-x recover-this-fileを実行してみましょう。

また、一度に複数のファイルを復元したい場合はM-x recover-sessionを実行します。自動保存ファイル名を記録したファイルがdiredで列挙されるので、C-c C-cで復元できます。

自動保存の間隔が頻繁であればあるほど、データ消失を防げます。今では次のように事実上即自動保存しても問題ない設定にしています。

```
;;; 4ストロークごとに自動保存(デフォルト300)
(setq auto-save-interval 4)
;;; 1秒のアイドルで自動保存(デフォルト30)
(setq auto-save-timeout 1)
```

バックアップファイル

標準に備わっているもうひとつのデータ消失対策セーフガードは、自動バックアップファイルの作成です。バッファが作成され、最初に保存したときにのみバックアップファイルが作成されます。バックアップファイル名はファイル名のあとに~が付きます。デフォルトでは、バージョン管理システム管理下のファイルや、一時ファイルのディレクトリ(/tmpなど)は対象外となります。

バックアップファイルは過保護で余計なファイルが混入されるせいで、しばしば嫌がられます。それならば、ファイルをバージョン管理システム管理下に置くのが一番です。きちんとコミットされている限り、いつでもすべてのリビジョンを取り出せるからです。バージョン管理システムと比べれば、バックアップファイルは原始的なしくみでしかありません。

バックアップファイルを作成しないのであれば、次の設定を加えればいいです。

```
(setq make-backup-files nil)
```

しかし、その分セーフガードがゆるんでいるので自己責任でお願いします。

ファイルに直接自動保存！

標準のauto-saveは別のファイルに保存しますが、より過激なアプローチとしてファイルに直接自動保存をさせるMELPAパッケージもあります。自動保存の間隔をコンマ数秒～数秒に設定すれば、手動でC-x C-sする必要がなくなります。つまり、“ファイルを保存する”という概念を消し飛ばしてしまいます。手動で保存するのが面倒と考えているならば導入の価値はあり

▼リスト2　auto-save-buffers-enhancedの設定

```
(require 'auto-save-buffers-enhanced)
;;; 自動保存の対象外となるファイル名正規表現のリスト
(setq auto-save-buffers-enhanced-exclude-regexps
      '("Org Src"))
;;; *scratch*バッファも保存対象にする
(setq auto-save-buffers-enhanced-save-scratch-buffer-to-file-p t)
;;; *scratch*バッファ保存時のファイル名
(setq auto-save-buffers-enhanced-file-related-with-scratch-buffer
      (locate-user-emacs-file "scratch"))
;;; 自動保存時にWroteというメッセージを出さないようにする
(setq auto-save-buffers-enhanced-quiet-save-p t)
 ;;; 3秒後に自動保存
(setq auto-save-buffers-enhanced-interval 3.0)
 ;;; 有効にする！
(auto-save-buffers-enhanced t)
```

▼リスト3 backup-each-saveの設定

```
(require 'backup-each-save)
;;; バックアップ先
(setq backup-each-save-mirror-location "/backup/backup-each-save")
;;; 日付の形式を指定
(setq backup-each-save-time-format "%y%m%d_%H%M%S")
;;; 元のメジャーモードで開くように設定する
(add-to-list 'auto-mode-alist '("-[0-9]¥¥{6¥¥}_[0-9]¥¥{6¥¥}$" nil t))
```

ます。「きりのいいところで保存するのではなく、きりのいいところでバージョン管理システムにコミットする」という考えを持っているなら、受け入れやすいです。

ただ、先月号でお話したように「利便性には代償が伴う」ことに注意してください。意図せずにファイルを変更してしまった場合、それに気づきにくくなるのです。バッファが変更されたならばモードラインに**という修正された印が表示されますが、自動保存であれ、ファイルが保存されたのならば常に無修正とみなされます。undohistやバージョン管理システムと連携させれば良いでしょう。

MELPAに登録されているauto-save-buffers-enhancedパッケージが古くから使われています（**リスト2**）。これは*scratch*バッファも自動保存・復元します。初回起動時はscratchファイルが存在しないのでエラーになりますが、*scratch*バッファに何か書き込むか、~/.emacs.d/scratchファイルをあらかじめ作成しておけば問題ありません。

シンプルで新しい実装としてreal-auto-saveパッケージも存在します。こちらのほうはマイナーモードとして実装されているので、有効無効を切り替えたり、特定のモードのみで有効にしたりできます。

保存時に自動スナップショット！

標準のバックアップファイルは最初の保存時に作られるものですが、MELPAにあるbackup-each-saveパッケージはより積極的なアプローチです。標準とは異なり、保存時に別ディレクトリに日付を含むファイル名で、毎回バックアップファイルとして書き出します。たとえば「/tmp/test.txt」ならば「/backup/backup-each-save/tmp/test.txt-150719_025939」のようなファイル名になります。履歴を見たり復元したければC-x d /backup/backup-each-save/tmp/test.txt-*のようにdiredを開けば良いです。**リスト3**が設定例になります。

前項のauto-save-buffers-enhancedと組み合わせると度が過ぎるほど冗長になりますが、現在のストレージ容量を考えれば、高々Emacsで編集する程度のテキストファイルがいくつあっても問題ありません。データ消失に対して、まさに鉄壁の守りになります！

おわりに

今回はセーフガード特集ということで、元に戻すことやデータ保護について触れました。「備えあれば憂いなし」というように、いざというときに役立つ知識をまとめました。

筆者は「日刊Emacs」以外にもEmacs病院兼メルマガのサービスを運営しています。Emacsに関すること関しないこと、わかる範囲でなんでも御答えします。「こんなパッケージ知らない？」「挙動がおかしいからなんとかしてよ！」はもちろんのこと、自作elispプログラムの添削もします。集中力を上げるなどのライフハック・マインド系も得意としています。SD

登録はこちら➡http://www.mag2.com/m/0001373131.html

Sphinxで始めるドキュメント作成術

山田 剛 Yamada Go Twitter @usaturn

第6回 Webサイトを作ろう（前編）

今回のテーマ

Sphinxで作成した静的HTMLをWebサーバにアップロードすれば、Webサイトとして使用できます。本稿では、見た目を変えるHTMLテーマや知っていると便利なこと、これまでにまだ扱っていない記法について取り扱います。

HTMLテーマを変更する

本連載ではこれまで、HTMLドキュメントを生成するためにmake htmlコマンドを実行してきました。この結果、**図1**のような画面のHTMLが生成されていました。

このHTMLの見た目を手軽に変更する機能が「HTMLテーマ」です。

▼図1 HTMLテーマ「alabaster」の例

HTMLテーマを変更するには作成したプロジェクトディレクトリのルートにあるconf.pyのhtml_themeを変更します。

sphinx-quickstartで生成されるconf.pyには、alabasterというHTMLテーマがデフォルト値として設定されています。別のHTMLテーマであるbizstyleに変更するにはconf.pyのhtml_themeをalabasterからbizstyleに変更します。

```
変更前
  html_theme = 'alabaster'
変更後
  html_theme = 'bizstyle'
```

HTMLテーマの設定を変更したらmake htmlコマンドでHTMLを生成しなおします。

```
$ make clean
$ make html
```

HTMLテーマがalabasterからbizstyleに変更され、生成されるHTMLファイルの見た目が変わります(**図2**)。

ほかにも、組み込みのテーマはsphinx_rtd_theme、classic、sphinxdoc、nature、haikuなど、十数種類あります。

また、各HTMLテーマにはそれぞれオプションが用意されています。オプションを変更するにはconf.pyのhtml_theme_optionsの記述を変更し、HTMLを生成しなおします。

```
変更前
  #html_theme_options = {}
```

変更後
```
html_theme_options = {
    'rightsidebar': 'true',
    'maincolor': 'black'
}
```

生成されたHTMLを確認すると左にあったサイドバーが右へ移り、ブルーからブラックを基調とした色彩に変更されます（**図3**）。

テーマごとに使えるhtml_theme_optionsは違いますので、ほかの組み込みのテーマや、テーマのオプションについての詳細は公式ドキュメント[注1]を参照してください。

注1）http://docs.sphinx-users.jp/theming.html#builtin-themes

ロゴとfaviconの追加

テーマを変更することにより見た目を大幅に変えることができました。次は「ロゴ」と「favicon」を追加してみましょう。ロゴで扱える画像データの種類はjpeg、png、gifです。ロゴはサイドバーの上部に配置され、横幅が200ピクセルを超える画像はリサイズされます（**図4**）。faviconは16×16か32×32（単位はいずれもピクセル）のWindowsのアイコンファイル形式（.ico）のデータを用意します。そしてconf.pyのhtml_logo、html_faviconが記載された行を書き換えます。

▼図2　HTMLテーマ「bizstyle」の例

▼図3　「bizstyle」のオプションを変更した例

COLUMN

HTMLの確認について

ブラウザは端末のローカルに保存しているHTMLファイルを読み込めますが、セキュリティ上の制限で一部のファイルが読めない場合があります。このような問題を避けるため、HTMLファイルを確認する際はローカルに簡易なhttpサーバを起動することをお勧めします。

すでにApacheやNginxが起動してあればそれらを使っても良いのですが、Sphinxを動かしているPythonを使い、すばやくhttpサーバを起動することができます。

Sphinxで生成したHTMLファイルの出力先（標準ではプロジェクトのルートの_build/html）で次のコマンドを実行してください。

Python2系の場合
```
$ python -m SimpleHTTPServer 8000
```
Python3系の場合
```
$ python -m http.server 8000
```

ブラウザでhttp://localhost:8000にアクセスするとドキュメントが確認できます。

```
変更前
#html_logo = None
#html_favicon = None
変更後
html_logo = 'logo.png'
html_favicon = 'favicon.ico'
```

ロゴ、faviconを保存しているファイルパス(conf.pyから見た相対パス)を指定します。conf.pyを書き換えたらmake htmlを実行し、HTMLを生成しなおします。

ファイルをダウンロードさせるリンク

Webサイトにファイルを設置し、そのファイルをユーザにダウンロードしてほしいというシーンでは、downloadロールが利用できます。downloadロールは、プロジェクト内に設置したファイルへのダウンロード用リンクを作成します。

▼図4　ロゴとfaviconを追加した例

▼リスト1　downloadロールの記述例

```
:download:`ダウンロードできます <example.zip>`
```

▼リスト2　脚注の記述例

```
Windowsのコマンドラインシェルには標準シェル [脚注1]_ 以外にオープンソースのbashやnyagos [脚注2]_ なども利用できる。

.. [脚注1] cmd、powershellがある
.. [脚注2] https://github.com/zetamatta/nyagos
```

▼図5　脚注の例(HTML)

Windowsのコマンドラインシェルには標準シェル [脚注1] 以外にオープンソースのbashやnyagos [脚注2] なども利用できる。

[脚注1] cmd、powershellがある

[脚注2] https://github.com/zetamatta/nyagos

example.zipのダウンロードリンクを作成する場合は、任意の場所にファイルを設置し、downloadロールでexample.zipの相対パス、あるいはプロジェクトのトップディレクトリをルートとした絶対パスで指定します。

リンクを張りたいドキュメント(reSTファイル)に**リスト1**のように記述します。ドキュメントを更新したらmake htmlを実行し、HTMLを生成しなおします。

脚注

「Webサイト」とは直接関係しませんが、リンクの一種である脚注について説明します。

reSTで脚注を作成するには、次のようにします。脚注を付けたい位置に「[キーワード]_」を記述します。指定するキーワードは、プロジェクト内で重複しないように指定します。生成されるHTMLには「[キーワード]」と表示され、脚注へのリンクが作成されます。

キーワードに対する脚注(説明文)は「.. [キーワード]説明文」と記述します。具体的には**リスト2**のように記述します。HTMLを生成すると**図5**のようになります。

また「[#foo]_」「[#hoge]_」と記述することにより自動採番することも可能です(**リスト3**、**図6**)。

サイトを公開する

Sphinxで作成されるHTMLドキュメントは静的HTMLですので、これらをWebサーバにアップロードすればコンテンツとして利用できます。

make html実行後のディレクトリ構成

プロジェクトをsphinx-quickstartで作成し、make htmlを実行した場合のディレクトリ構成は、**図7**のとおりです。

_build/htmlディレクトリ配下のファイル

図7のディレクトリの内、Webサイトの公開に必要なファイルは、_build/htmlの配下に生成されています。_build/html配下について簡単に説明をします。この中には、Webサイトとして利用するうえでは、不必要なファイルもありますが、セキュリティの観点では問題ないので、そのまま公開しても良いでしょう。

- **.buildinfo**
  設定情報などのハッシュ値を記録しておくファイル。存在しないとmake時にすべてのファイルを生成しなおす。Webサーバにアップロードする際は不要なため削除してもかまわない
- **genindex.html**
  索引をまとめたHTML
- **index.html**
  index.rstから生成されたHTML
- **objects.inv**
  オブジェクトのマッピング情報を保存したファイル
- **search.html**
  検索結果ページを表示するためのHTML
- **searchindex.js**
  検索するためのJavaScript

_sourcesディレクトリ配下のファイル

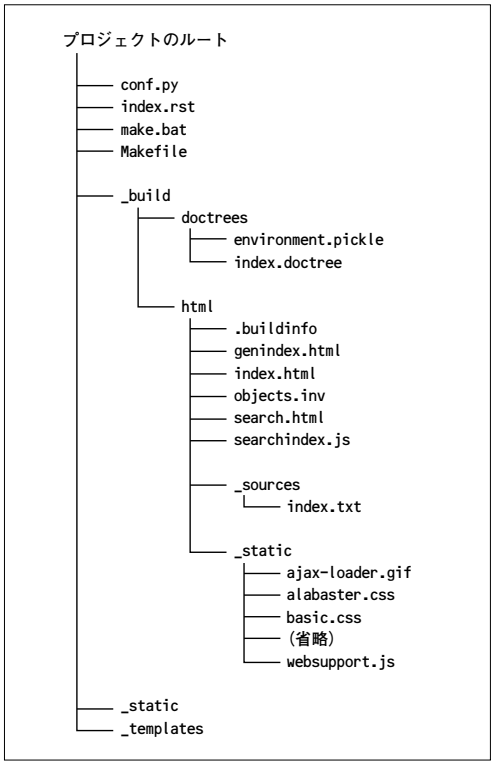
生成したHTMLのサイドバーにある「ソースコードを表示」で閲覧できるreSTのソースファイルを格納しています。

コンテンツとして公開したくない場合は、conf.pyのhtml_show_sourcelinkを次のように変更したうえで削除してください。

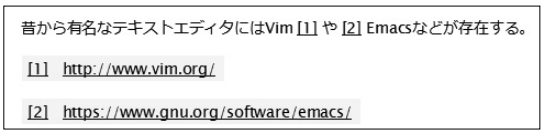
```
html_show_sourcelink = False
```

ただし、_sourcesディレクトリを削除した場

▼図7　sphinx-quickstart、make html後のディレクトリ構成



▼図6　脚注（自動採番）の適用例（HTML）



▼リスト3　脚注（自動採番）の記述例

```
昔から有名なテキストエディタにはVim [#vim]_ や [#emacs]_ Emacsなどが存在する。

.. [#vim] http://www.vim.org/
.. [#emacs] https://www.gnu.org/software/emacs/
```

合、検索結果画面にヒットした位置のコンテンツが表示されません。また、make htmlをするたびに_sourcesディレクトリは生成されます。

_staticディレクトリ配下のファイル

おもにHTMLテーマで使用されるcssファイルやjsファイルが格納されています。基本的に必要なため、すべてアップロードしてください。

サイトにHTMLドキュメントを設置する

Apache HTTP Server(以下、Apache)にHTMLドキュメントを設置する場合の手順を簡単に説明します。

サーバにApacheをインストールし、設定ファイル(httpd.conf、apache2.confなど)のDocument Rootを書き換えます。

```
Apacheのconfig設定
  書式
  DocumentRoot [HTMLドキュメントを設置する任意のパス]
  記述例
  DocumentRoot /var/www
```

Sphinxのプロジェクトをmake htmlしたあと、_build/html以下のファイルを/var/wwwへコピーします。Apacheを起動し、サーバにWebブラウザでアクセスすればSphinxで作成したコンテンツを閲覧できます。

Webサイトにrobots.txtや.htaccessを設置する

Sphinxでビルドした_build/htmlディレクトリの中身をすべてアップロードしてWebサイトの更新をしたい場合、robots.txtや.htaccessなどのドキュメントに直接関係ないファイルをあらかじめプロジェクトに含めておくことができます。

生成したHTMLに特定のファイルを含めるには、まずプロジェクト内の任意の場所に含めたいファイルを設置します。そして、conf.pyのhtml_extra_pathにそれらのファイルのファイルパス(conf.pyから見た相対パス)を指定します。

例として**リスト4**ではプロジェクトのルートにextrafilesというディレクトリを作成し、そこにrobots.txt、.htaccessを保存しています。

conf.pyを書き換えたらmake htmlを実行し、HTMLを生成しなおします。**リスト4**の場合はextrafilesに保存したファイルが_build/html配下にコピーされます。

▼リスト4　html_extra_pathの記述例

```
変更前
  #html_extra_path = []
変更後
  html_extra_path = ['extrafiles/robots.txt', 'extrafiles/.htaccess']
```

COLUMN

Webサイトのバックアップについて

Sphinxドキュメントはプロジェクトさえ残っていれば、HTMLファイルがなくなってもmake htmlを実行することで何度でも同じHTMLファイルを生成できます。

しかしながら、Sphinxの実行環境が変わってしまう場合、たとえば、使い方がよくわからない拡張を環境に追加したり、アップデートに失敗したりして意図しないHTMLが生成され元のHTMLが生成できなくなることがあります。

このような問題を避けるために、安易な生成済みHTMLの削除、環境の変更をしないようにしましょう。

また、Sphinxの実行環境はいつでも作りなおせるように自動化する、あるいは手順を残し、すでに公開しているサイトについてはバックアップを取ることをお勧めします。

◆ ◆ ◆

今回はSphinxドキュメントをWebサイトとして公開するために必要な基本的な情報と、これまでに触れていなかった記法について取り上げました。次回は応用編としてSphinxドキュメントをホスティングするためのいくつかの手段と、Sphinxと相性の良いバージョン管理について紹介します。SD

COLUMN

PyCon Singapore 2015

Author 清水川 貴之

本連載執筆陣の1人、清水川です。

2015年6月17日～19日にかけて、シンガポールでPyCon Singapore 2015[注A]（以下、PyCon SG）が行われました。PyCon SGは毎年シンガポールで行われるPythonカンファレンスで、今年で6年目です。シンガポールは、APAC（アジア太平洋）地域のPythonカンファレンス「PyCon APAC」を初めて開催した国です。PyCon APACは2010年から2012年までの3年間はシンガポールで開催され、2013年には日本で、昨年と今年は台湾で開催されました。

筆者は、前回のコラムでお知らせしたPyCon APAC 2015[注B]と同様に、Sphinxの多言語化機能を紹介する発表を行ってきました。今回は、PyCon SG 2015の様子と、Sphinxに対するカンファレンス参加者の様子を紹介したいと思います。

■シンガポールでのSphinxへの反応

筆者は、PyCon SGにて「Easy contributable internationalization process with Sphinx（Sphinxによる貢献しやすい翻訳プロセス）」[注C]の発表を行いました。この発表では、Sphinxのドキュメント翻訳サポート機能について紹介しました。

翻訳について、現地の人が実際に使っている言語をまじえて紹介しようと、事前にシンガポールの言語について調べたところ、公用語が「英語」だということがわかりました。シンガポールでは、学校を卒業するには英語の習得が必須なんだそうです。こういった事情もあり、筆者の「翻訳プロセス」についての発表は、参加者が13人でした。それでも、参加者にはSphinxの国際化機能について、その価値を提供できたのではないかと思います。

発表の初めに、どのくらいの人がOSSを使ったことがあるか聞いて手を挙げてもらったところ、10人ほどの人が手を挙げてくれました（**写真A**）。しかし、何かOSSへ貢献したことがあるかと聞くと手を挙げてくれたのは2人でした。そこで、「OSSを使うこと自体が最初の貢献だし、そのOSSをほかの人に伝えることも貢献ですよ」ということを伝えました。

▼写真A　発表の始めにいくつか質問しました

シンガポールは英語の国で、技術者も英語に慣れていますが、日本では英語に慣れ親しんでいる技術者はそれほど多くありません。そのため、翻訳ドキュメントはそのソフトウェアを広めるのにも、使い始めるのにもとても有用、ということを発表の中で伝えました。

発表のあとに、2人の方からSphinxの機能について質問を受けました。マレーシアから参加されたLucasさんは、「マレーシアには英語に不慣れな人が多いので、翻訳しないと読めない人が多い」という話をしてくれました。「教育に使うドキュメントの翻訳に今日聞いた話を役立てたい、さっそく使ってみたいのでスライドを共有してほしい」と、とても熱心な様子でした。こういう方に出会えるのはとてもうれしいですね。

今回のPyCon SGでは、多くの人にはSphinxを紹介できませんでしたが、ドキュメント翻訳をサポートするしくみを必要としている方に、Sphinxの手法を伝えられたと思います。

PyCon SG 2015の参加レポートをgihyo.jpに掲載しています[注D]。そちらもご参照ください。

注A） PyCon Singapore 2015　https://pycon.sg/

注B） PyCon APAC 2015
https://tw.pycon.org/2015apac/en/

注C） http://www.slideshare.net/shimizukawa/sphinx-autodoc-automated-api-documentation-pyconapac2015

注D） PyCon SG 2015参加レポートとSphinxに関する発表
http://gihyo.jp/news/report/01/overseas-pycon-presentation-training-2015/0002

Mackerelではじめるサーバ管理

Writer 田中 慎司(たなか しんじ) ㈱はてな
Twitter @stanaka

第7回 Mackerelでアラート通知を最適化しよう

サーバの状態を、思い思いのツールで知ることができると便利ですよね？ 今回はMackerelのアラートを外部サービスに通知する機能について、チャットツール「Slack」、インシデント管理ツール「PagerDuty」を例に解説します。開発チームごとにアラートを振り分ける「通知グループ」の設定についても紹介します。

柔軟なアラート通知

Mackerel[注1]では、発生したアラートをさまざまな手段(チャンネル)で通知することができます。またそれらの通知手段をアラート内容に合わせて使い分けることもできます。連載第7回目の今回は、アラート通知の詳細な設定方法について紹介します。アラートを発生させるための監視ルールの設定方法は、連載第3回(本誌2015年5月号)を参照してください。

通知チャンネル

アラートを適切な手段で受け取るために、まずはチャンネルの設定をしましょう。

各種チャンネル

チャンネルにはメールや、Slackなどのチャットサービスを指定することができます。執筆時点(2015年7月)では、チャンネルとしてメール、Webhook、Slack、HipChat、PagerDuty、im.kayac.com、Chatwork、Typetal、OpsGenieに対応しています。

ここでは、SlackとPagerDutyについて設定方法を紹介しますが、ほかのチャンネルについても似た手順で設定できますので、チームに合わせて最適なツールを利用してください。またWebhookを利用することで、単なる通知を越えたしくみを作ることもできますが、こちらについてはまた別の機会に紹介します。

Slackチャンネルを設定する

Slackチャンネルの設定をするには、まずはMackerelトップページのサイドバー内「Monitors」をクリックし監視ルール一覧に遷移します。次に、右上の「チャンネル設定」からチャンネル一覧に遷移します。そして右上の「通知グループ／通知チャンネルを追加」から「Slack」を選択します。

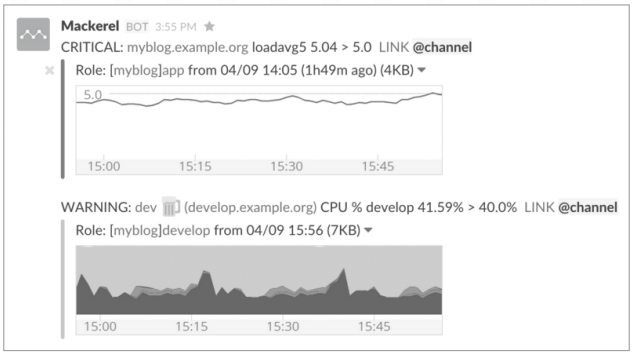
図1のようなダイアログが表示されますので、「通知チャンネル名」「URL」の2つを指定します。Slackチャンネルへの通知には、Slack APIの1つであるIncoming Webhooksを利用しています。**図1**でのURLの欄には、SlackのConfigure Integrationsの設定からIncoming WebHooksのWebhook URLを調べ、入力してください。

また、MackerelからSlackへのアラート通知を送る際に、監視のステータスに応じてメンションを送ることも可能です。メンションは「なし」「@everyone」「@channel」「@group」から選択できます。重要なアラートが発生した際にメンションを送ると、スマートフォンでもSlackアプリを利用することですぐに通知を受信することができます。それぞれの効果についてはSlackの

注1) URL https://mackerel.io

▼図1　Slackの設定ダイアログ

▼図2　チャットに、メトリックのグラフを画像として流す



ヘルプ[注2]を参照してください。

Slack以外のHipChat、Chatwork、Typetalkのチャットツール連携も同様の手順で設定できます(詳細は個別のヘルプを参照してください)。

グラフを画像として通知

チャットツールにアラートを流す際、対象メトリックを含むグラフを画像として合わせて通知することができます(**図2**)。

チャット上にアラートを通知する際にグラフ画像を合わせて投稿することにより、通知されたアラートがどの程度の緊急性をもっているか、数値が徐々に上って閾値(しきいち)を越えたのか、急激に上昇して閾値を越えたのか、直感的に判断することができます。それにより毎回Mackerelの画面を開く必要がなくなり、緊急性の低いアラートに作業を妨害されることがなくなります。

またチャット上のメンバが同じ画像を見ている前提で対応方針を相談できますので、「見ているグラフが実は別だった」といったミスを減らせます。またスマートフォン上のチャットアプリでも同様にグラフ画像を見ることができますので、出先でノートPCを広げる必要があるかかどうか、事前に判断することができるようになります。

PagerDutyチャンネルを設定する

PagerDuty[注3]は、Mackerelのような監視ツールからのアラートを集約し、PagerDuty上に登録した任意の通知ルールに従ってさまざまな通知を送ることができるインシデント管理サービスです。

たとえば、通知先を時間で変更するようなスケジューリング機能や、一定時間で反応がなければ次の通知先に通知するようなエスカレーション機能を持っています。通知の例として、音声、SMS、email、プッシュ型のアラート通知を送ることができます。

PagerDutyとの連携を行うことで、Mackerelで設定した監視ルールに従い発生したアラート通知をインシデント通知としてPagerDutyに送ることができます。アラートが発生したとき、アラートが解決されたときなどにPagerDutyに通知を送ります(**図3**)。

連携には、PagerDutyのIntegration APIを利用しています。設定方法の詳細はヘルプページ[注4]を参照してください。

またPagerDuty連携では、アラート状態に

▼図3　PageDutyへの通知

注2) URL https://slack.zendesk.com/hc/en-us/articles/202009646-Making-announcements
注3) URL http://www.pagerduty.com
注4) URL http://help-ja.mackerel.io/entry/howto/alerts/pagerduty

応じてPagerDutyへ通知するかどうか選択することができます。通知対象のアラート状態は連携設定フォームで「Warning & Critical」または「Critical only」のどちらかを選べます。それぞれ次のような挙動をします。

Warning & Critical

- Warning、Criticalいずれかのアラートが発生した際に、PagerDutyにインシデント通知を送る
- ステータスが正常に戻ったときに、PagerDutyのインシデントは自動的に解決される
- WarningとCritical間の状態遷移が生じた際にもPagerDutyに通知を送るが、一度発生したインシデントに対する経過通知を送るのみ

Critical only

- Warning時のアラートでは通知は送らず、Criticalのアラート時のみにPagerDutyにインシデント通知を送る
- Warning状態への遷移時にはPagerDutyに対しては何も行わない

いずれにしてもステータスが正常に戻ったときに、PagerDutyのインシデントは自動的に解決されます。

PagerDutyの類似サービスとしてOpsGenie[注5]にも対応していますので、チームによって適したほうを使ってください。

通知グループ

Mackerel上で扱うホスト数が増え、監視ルールが増えてくると、“すべてのアラートがすべてのチャンネルに通知される”状態では不都合がでてきます。典型的には、通知先のチャットのメンバによって関係のあるアラートと関係のないアラートがある、ということがあります。

これに対処するために、アラートが通知されるチャンネルをきめ細かく制御することができる通知グループというしくみがあります。通知グループを利用することで、特定のサービスや監視ルールで発生したアラートを異なるチャンネルに振り分けることができます。

通知グループの作り方

通知グループを作るには、通常のチャンネルと同様にトップページサイドバーの「Monitors」をクリックし監視ルール一覧に遷移します。次に、右上の「チャンネル設定」からチャンネル一覧に遷移します。そして右上の「通知グループ／通知チャンネルを追加」から「通知グループ」を新規設定します(**図4**)。通知グループ名とそのグループに入れるチャンネルと通知グループを選択します。次に、通知したいアラートの発生元のサービス、監視ルールを指定します。最後に通知チャンネルを選択します。

発生元のサービス、監視ルールはそれぞれ複数設定することができます。監視ルールの設定には「他の通知グループを無視」というオプションを指定することができます。

また複数の通知グループを設定することができます。「Default」の通知グループは、デフォルトではすべてのアラートが通知されますが、「他の通知グループを無視」と指定されているアラートを除きます。

▼図4　通知グループの設定

注5）URL https://www.opsgenie.com

通知グループの例

通知グループの通知先決定のアルゴリズムはすこし複雑ですので具体例を紹介します。サービスは2つ、ホストが3つあり、それぞれ次のように所属しているとします。

サービスA：ホストA、ホストB
サービスB：ホストA、ホストC

監視ルールは次の3つ。

監視ルール1：全サービスのCPU使用率
監視ルール2：サービスAのサービスメトリック
監視ルール3：サービスBのサービスメトリック

通知グループはデフォルトを含め、次の4つ。

通知グループ1：サービスA
通知グループ2：サービスB
通知グループ3：監視ルール3「他の通知グループを無視」が有効
通知グループDefault

このようなときに、ホストAのCPU使用率が高くなったとします。すると監視ルール1により、アラートが発生します。アラートの発生元のホストAはサービスAとサービスBの両方に所属していますので、通知グループ1と通知グループ2の両方に通知されます。また、デフォルトの通知グループDefaultにも通知されます。

もしCPU使用率が高くなったのがホストBの場合は、通知グループ1とデフォルトの通知グループDefaultの2つに通知されます。

またサービスAのサービスメトリックで監視ルール2によりアラートが発生した場合は、通知グループ1と通知グループDefaultの2つに通知されます。一方、サービスBのサービスメトリックで監視ルール3によりアラートが発生した場合は、通知グループ3のみに通知されます。

通知グループ3では「他の通知グループを無視」が有効になっていますので、通知グループ2および通知グループDefaultには通知されません。

通知グループは少し複雑になっていますが、柔軟な設定をすることができます。

お勧めの通知設定

Mackerelを活用するうえで、必要十分な通知を行うことはとても大事です。緊急対応の必要がないアラートをあまり関係のない人も含まれるチャンネルに大量に送ってしまったり、逆に緊急性の高いアラートを一部の人にしか届かないチャンネルに送ってしまったりすると、アラートが無視されやすくなったり、気づかれなかったりしてしまいます。

筆者が所属するはてなでは、次のような原則を用いてアラートを通知するようにしています。

- すべてのアラートを受け取るチャンネルとしてSlackの専用チャンネルを作る
  インフラ全体を見るチームのメンバのみがここを見ています。
- サービスごとのチャンネルに外形監視など、そのサービスの重要な監視のみを通知する
  各チャンネルにはディレクターなどエンジニア以外のメンバも参加していますので、緊急性の低いアラートは送らないようにしています。通知の際にメンションを入れるかどうかは、各チームに任せるようにしています。

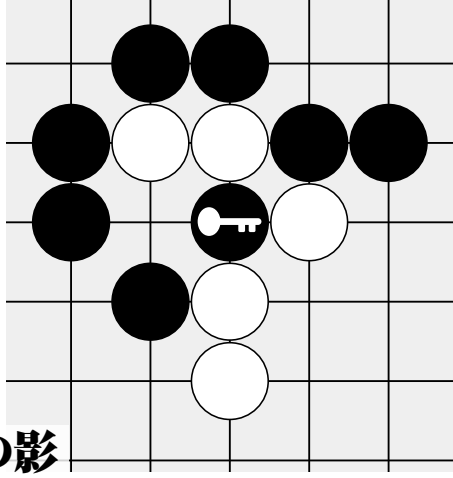
このようにアラートの重要性に応じて、受け取るメンバを切り分けるようにしています。

おわりに

今回は通知チャンネルと通知グループについて解説しました。Mackerelのようなサーバ監視を利用する際は、通知まわりをうまく設計することが重要となります。監視通知を日々受けとっていると感覚が麻痺してしまって、なかなか見直す機会がないかもしれませんが、たまには通知が最適化されているかどうか棚卸ししてみてはいかがでしょうか。SD

セキュリティ実践の基本定石

すずきひろのぶ
suzuki.hironobu@gmail.com



みんなでもう一度見つめなおそう

【第二四回】日本に忍び寄るランサムウェアの影

ランサムウェアとは、PCの中のデータを暗号化し、「それを戻したければ金銭を払え」と要求する「身代金目的」のマルウェアです。ランサムウェア自体は古くからあり、そんなに珍しいものではありません。しかし、近年は暗号技術、匿名通信路、デジタル通貨などの発展により高度化してきています。

ランサムウェア

ランサムウェア（Ransomware）のランサム（Ransom）とは「身代金」という意味です。辞書をひくと名詞では「身代金」のほかに「身請け」、動詞では「（身代金を払って）～を受け戻す」「（人質を）身代金を受け取って解放する」という説明があります。Ransomにしっくりくる言葉が見つからないので、本稿でも身代金を要求するためのマルウェアとして「ランサムウェア」という言葉を使います。

ランサムウェアの基本的なモデルは次のようなものです。

①まず相手のコンピュータにマルウェアを感染させる
②コンピュータの中のファイルを暗号化する
③復号するための鍵は攻撃者が持っている
④暗号化したファイルを戻すために金銭的な要求をする

ある意味、極めてシンプルなランサムのモデルです。

1989年のランサムウェア

最初のランサムウェアは1989年のPC Cyborg Trojan（以下、PC Cyborg ）だと言われています。PC CyborgはDOSで稼働するトロイの木馬です。AUTOEXEC.BATを書き換え、PCの立ち上げ回数を見ています。そして、一定の回数を越えると「PC Cyborg Corporationにリニューアル・ライセンスを払え」というメッセージを表示させて、Cドライブのファイルやディレクトリの名前をすべて暗号化してしまうものでした。要求する金額は189ドルで、指定された送り先は中南米パナマの私書箱でした。

PC Cyborgの作者はDr. Joseph Popp氏であることが英国のアンチウイルスベンダによって突き止められ、スコットランドヤード（ロンドン警視庁）によって逮捕されました。四半世紀前の事件で記憶が薄れたとはいえ、サイバー犯罪史においてはエポック・メイキングな事件でした。

2015年のランサムウェア

マルウェア対策の大手ベンダであるMcAfee社の2015年第1四半期に発行した脅威レポート“McAfee Labs Threats Report May 2015”[注1]は次のような文章で始まっています。

McAfee Labs saw almost twice the number of ransomware samples in Q1 than in any other quarter.

図1のグラフは同報告書からの引用ですが、2015年第1四半期に発見されたランサムウェアの数は、

注1） http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q1-2015.pdf

その前の2014年通年の数よりも多く、これまで四半期単位で最大だった2013年第2四半期の2倍になっているという驚異的な数字になっています。

現状においてランサムウェアは最大の脅威の1つととらえるべきものです。ランサムウェア被害の85%はアメリカとヨーロッパで、アジア地域全体では7%です。日本では感染率が低いためか関心が薄いと言わざるを得ません。

IPAの発表によれば、2015年段階で日本においてはランサムウェアの被害の報告は月に1桁とのこと[注2]ですので、蔓延(まんえん)している欧米の状況とは大きく異なります。

日本は空白地帯とも言えるわけですが、ターゲットが日本に振り向けられたならば、欧米がすでにそうであるように、一気に蔓延する可能性は極めて高いことを認識しておかなければならないでしょう。

今どきのランサムウェアの技術

基本的には何らかの方法でシステムに入り込むマルウェアによって引き起こされるので、感染のルートは以前に紹介したZeusといったいろいろな種類のマルウェアと変わりはありません。ですので、メールでマルウェアを送ったり、あるいはメールやメッセンジャーで送られたURLをクリックさせブラウザやブラウザのプラグインの脆弱性をついてシステムに入り込んだりとさまざまです。このように、いつでもランサムウェアが入ってくる可能性はあります。

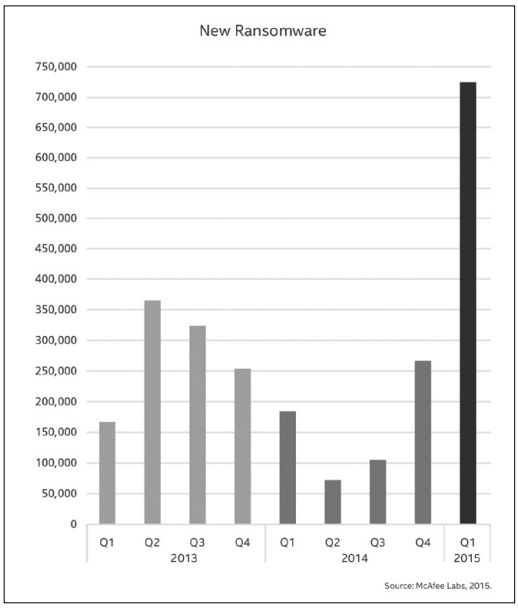
現在の形態のランサムウェアに変化していった背景には3つの技術が関わっています。

- **暗号技術の向上と普及**
- **ボットテクノロジの向上**
- **匿名化が進んだ支払い方法の普及**

暗号技術の向上と普及

「ランサム(身代金)」というぐらいですから、お金を払えば元に戻せることが条件です。もし、お金を払っても戻らないようだと誰もお金を払わなくなりますから、そもそも前提条件が成り立ちません。

◆**図1　新しく発見されたランサムウェア数**
(出典：McAfee Labs Threats Report May 2015)



◉言語の壁

LINEのプリペイド詐欺のケースを考えた場合、だまされた人もかなりの数いたようですが、言語の壁があり定形的な文言しか入力できずパターンが決まってしまっていて、だますにも限界がありました。

以前の標的型攻撃のメールの文面や、フィッシングの誘導メールや画面構成などで使われている日本語は、言い回しや文法があきらかにおかしいものがたくさんありました。そのころは言語の壁があったのは事実です。しかし、だんだんと完成度が高くなっており、最近は、普通の事務的文章と区別がつかないレベルにまで達してきています。

そして、それが今後は欧米で猛威を奮っているランサムウェアに振り向けられ、日本でも本格的な課題になることは時間の問題であろうと筆者は考えています。

注2) http://www.ipa.go.jp/security/txt/2015/06outline.html

いろいろな実装が考えられますが、ゼロから完全な暗号の実装を独自に進めるのは、相当な技術力が必要です。ですが、今や、そのようなことは必要ありません。今日においておもなオペレーティングシステムは暗号のためのAPIをデフォルトで用意しています。そのAPIを呼び出せば良いのですから、製作に必要な技術力やコストは大幅に下がります。

今から10年前の2005年に著名な暗号研究者Moti M. Yung氏とAdam L. Young氏が書いた“An Implementation of Cryptoviral Extortion Using Microsoft's Crypto API”[注3]という文書があります。

その文書は、題名にあるようにMicrosoft社の標準の暗号化APIを利用しランサムウェアの暗号化部分を作れることを示しています。つまりランサムウェアが使っている技術はMicrosoft社のオペレーティングシステムと同じ暗号学的な強度を持っているので、正当な方法[注4]以外では戻せないことがわかります。

まず公開鍵暗号の鍵ペアを生成し、暗号鍵を残し、復号鍵をボットのコレクションサーバにタグを付けてアップロードする（と同時に元の復号鍵を消去する）か、あるいはC&Cサーバ側で鍵ペアを作り感染先マルウェアが暗号鍵をダウンロードします。

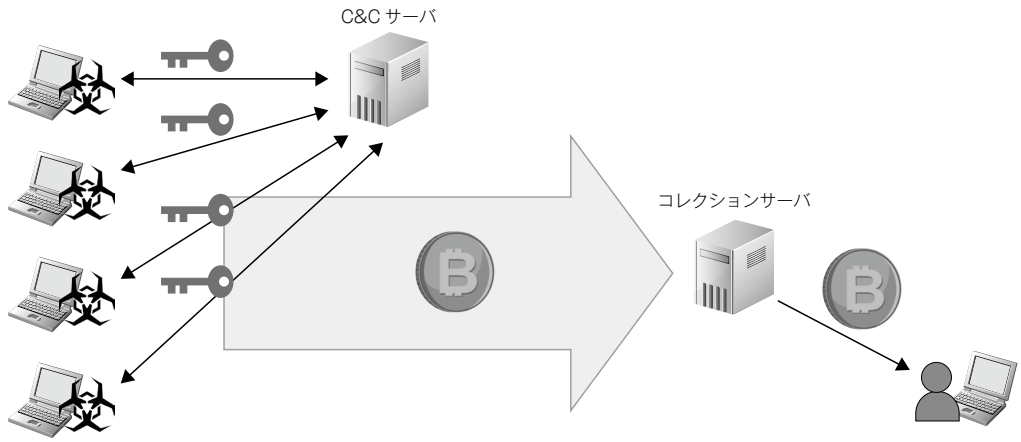
各々のファイルを暗号化する際はファイルごとに独立したセッション鍵を使い共通鍵暗号で暗号化し、そのセッション鍵を公開鍵暗号の暗号鍵で暗号化してしまいます。これは既存のファイル暗号化のアプリケーションのアプローチとなんら違いはありません。

こうなると正攻法での解読は不可能ですから、通常のアプリケーションと同じで製作者がミスをおかし、脆弱性を発生させた部分から攻めるしかありません。

ボットテクノロジの向上

マルウェアZeusを説明したときと同じく、命令を利用して公開鍵の情報を送り出す司令塔のC&Cサーバや、密かにデータ（この場合はビットコインなどの情報）を収集するコレクションサーバの技術は、そのままランサムウェアの基本的なインフラストラクチャーになります（**図2**）。C&Cサーバから暗号鍵が送られてくるタイプ、あるいは感染側で公開鍵ペアを生成し、復号鍵をC&Cサーバに通知するタイプ、初めからランサムウェアの中に暗号鍵が複

◆**図2　ランサムウェアのインフラストラクチャー**



ランサムウェアで使っているネットワーク・インフラの構造は、これまでのマルウェアのインフラとなんら変わりがない。コンピュータ内部の情報を盗むのではなく、C&C サーバから暗号化に使う公開鍵を送り、コレクションサーバに相当するセンターサーバにビットコインなどの情報を送る

注3）**http://www.cryptovirology.com/cryptovfiles/newbook/Chapter2.pdf**
注4）この場合、犯人から復号するための鍵を受け取ることを意味します。

数入っていて、どの暗号鍵を使うかC&Cサーバに指示されるタイプといろいろと考えられます。

理論上は最小で1KBにも満たないデータ量（使う暗号鍵のタグ）を送るだけというのも可能ですから、ファイアウォールでのトラフィック量の増減による異常検知といったことで発見するのもたいへん難しいことでしょう。

Gameover ZeuSのように情報流通のインフラをP2Pで構築しているとしたら、さらに把握するのは難しくなります。これらもランサムウェア側で新たに実装や検証をする必要はなく、ZeusやGemeover ZeuSでの実績のある環境を取り入れることが可能ですし、実際に取り入れているランサムウェアもあります。匿名通信経路を確保するためにTorネットワークを利用するものが現れるなど、速いスピードで進化しています。

匿名化が進んだ支払い方法の普及

デジタル通貨（Digital Currency）の登場が、ある意味、ランサムウェアに新時代をもたらしました。それまで最も難しかったのは、マネタイズする部分です。脅迫の際に銀行の振込先を教えているようでは、その銀行口座もあっというまに閉鎖されてしまいます。アメリカの同時多発テロ事件以降、世界中の銀行ではテロ資金のマネーロンダリングに神経を尖らせている状況です。このような状況でお金を動かすことは、以前にも増して難しくなっています。これまでは、ここがボトルネックでした。

しかし、ビットコインのようなデジタル通貨の登場で状況は一変します。ビットコインの特徴は身元を明かさずにビットコインを保有することが可能なことです。また、高速に、かつ複雑にビットコインを譲渡することが可能です。意図的に流通に複雑なルートを形成させることで、ビットコインの流れを追うことができなくなるような性質があるのも確かなようです。

2013年3月には全世界のビットコイン交換の7割を処理していたビットコイン交換所Mt. Gox社が、2014年初頭に大量のビットコインを盗まれていることが発覚し、倒産に追い込まれたのは記憶に新しいかと思います。ビットコインがどのような（所有者を表す）ビットコインアドレスをたどっていったか調べられそうな気もしますが、その盗まれたビットコインを追跡できたという話は、ついぞ聞きません。

また利用価値の高いプリペイカードやクーポンなどの登場も同様です。一時期、乗っ取ったLINEアカウントから、他人にプリペイカードを購入させ、その（金銭的な）価値を盗むという手口が流行りましたが、このように良い意味でも悪い意味でもインターネット時代の支払い方法は多様化し、かつ、瞬時にお金の受け渡しができる時代になっています。

CryptoLocker

CryptoLockerは2013年に現れたMicrosoft社のWindows向けランサムウェアです。「公開鍵暗号を使う」「Gameover ZeuSのボットネットに相乗りする」「ビットコインやプリペイカードで支払いを要求する」という今日のランサムウェアの特徴を持っており、ランサムウェアの代名詞的な存在になっています。感染する経路はおもにメールに付属した実行ファイルです。最初に感染した段階ではスタートアップ時に自動的に立ち上がりバックグラウンドジョブとしてC&Cサーバと通信をしています。

興味深いのは、RSA-2048の公開鍵ペアはサーバ上で用意されており、C&Cサーバから感染先コンピュータのCryptoLockerに暗号鍵（公開鍵）をダウンロードするよう指示を出し、ダウンロードさせることです。現状では2,048ビットのRSAを解読する方法はありません。近い将来でも無理です。

ZDNetの2013年12月の記事によれば、CryptoLockerの被害者数は25万、72時間以内に身代金を払えと要求し、おもにビットコインで平均300ドルを支払ったとあります[注5]。

注5）CryptoLocker's crimewave: A trail of millions in laundered Bitcoin
http://www.zdnet.com/article/cryptolockers-crimewave-a-trail-of-millions-in-laundered-bitcoin/

CTB-Locker

現在急激に増え、またさらに技術的に高度化しているランサムウェアがCTB-Lockerです。2014年7月に現れたと言われています。これまでのランサムウェアの最も進化したバージョンと言えるかもしれません。CTBの意味は次のとおりです。

- **C（Curve）：公開鍵暗号に楕円曲線暗号を使っている**
- **T（Tor）：インターネット上の接続経路を匿名化するTorを使う**
- **B（Bitcoin）：支払いはトラッキングが難しいBitcoinを要求する**

楕円曲線暗号は鍵ビット数が500ビット前後あればRSAの鍵長10,000ビット超の強度を持ちます。これは通常、処理の高速化あるいは実行されるプログラムのメモリ量のコンパクト化を狙うために使われます。しかし、公開鍵暗号を使った暗号化を考えた場合、RSAは暗号鍵のサイズが小さくて済み、現状の鍵ビット数程度であれば暗号化プロセスに関しては処理の不利益になるほどではありません。ですから、楕円曲線暗号を処理速度のためにあえて選ぶ必要はありません。強度を保つために使っていると考えるほうが合理的です。

Torは、普通は「途中で暗号化されたプロクシがたくさんあり、そこをいくつも経由することでサーバにアクセスする際にユーザを匿名化する」という説明になると思います。C&CサーバがTorのプロクシ網の外にある場合、つまり、最終アクセスホストとしてURLで指し示している場合には、C&Cサーバのある場所は明確になります。しかし、このプロクシの途中にC&Cサーバが隠されて置かれていた場合は、発見するのは困難、もしくは不可能でしょう。というのも、これを見つける技術は、ユーザの使っているコンピュータを見つける技術とほぼ同じだからです。今でも十分にTorの匿名性が確保されているわけですから、そこに隠れているC&Cサーバを見つけるのはほぼ不可能だと筆者は考えます。

最後にビットコインです。今でもビットコインは、トラッキングして本来の所有者を特定すること、キャッシュアウト時の保有者を特定することに成功していません。ですので、ビットコインで支払われればキャッシュアウトしても今はまだ足がつきません。

購入可能なランサムウェア

CTB-Lockerは販売されているランサムウェアとしても知られています。

malwareid.jpの「CTB Lockerランサムウェアまたは暗号化されたファイルの解読法」[注6]という記事から引用します。

CTB Lockerは誰でもオンラインで$3,000（米国ドル）で購入することができます。この金額で、全てを正しく設定するための基本的なキットや完全なサービスをCTB Lockerの開発者から受け取ることになります。

CTB-Lockerはただでも厄介なうえに、お金さえ払えば自分用にカスタマイズする親切丁寧な説明がついた開発キットが入手できるというランサムウェアです。つまり、今後もランサムウェアは、さらに加速度的に増えるということを意味しています。

非暗号ロック系ランサムウェア

こちらの場合は、人質を取るというより、脅迫するマルウェアと呼べるでしょう。いろいろなパターンがあるのですが、有名なのがRevetonというマルウェアです。Zeusの流れをくむマルウェアで、感染すると「違法な画像がコンピュータ内に存在しているのを発見した。警察に通報されたくなければ金を払え」と恐喝します。違法な画像の代わりに「違法な音楽や映画を発見したので、著作権管理団体に通報されたくなければ金を払え」というバリエーション

注6）http://www.malwarerid.jp/ctb-locker-ランサムウェアまたは暗号化されたファイルの/

もあるそうです。

筆者は本当にこんなもので引っかかるのか疑問ですが、このランサムウェアを真に受けて、それで観念して自ら警察に出頭した人がいるという記録があるのでそれなりに被害は出ているのでしょう。これまでに何度も繰り返してきた言葉――「コンピュータ・セキュリティを考えた場合、そのコンピュータ・システムのリソースの中で最も脆弱な部分は人間である」という言葉を思い出さずにはいられません。

初の日本語ランサムウェアの使用者は17才の少年

日本国内でランサムウェアを語るうえで、避けては通れない話題ですので言及したいと思います。

2015年7月1日、報道各社は、警視庁が不正アクセス禁止法違反と私電磁的記録不正作出・同供用容疑で神奈川県に住む17才の無職少年を逮捕したというニュースを流しました[注7]。

ネット上で「0Chiaki」と名乗り、技術評論社が利用している「さくらのVPS」のコントロールパネルへアクセスするためのアカウントとパスワードを盗み、サーバのOSを入れ替え、第三者サイトへリダイレクトするように設定したのも、この少年です。

ランサムウェアの定義が広いのはこれまでの説明のとおりですが、今日的な暗号でファイルをロックさせるタイプのランサムウェアを日本国内向けに日本語バージョンで作って配布したというのは、少なくとも筆者の調べた範囲では0Chiaki以前には事例を見つけられませんでした。

0ChiakiはYOMIURI ONLINEのインタビュー[注8]でTorLocker 2.0を使ったと言っています。

TorLockerのエコシステムは、TorLocker運営と、それを使う(相手に感染させる作業を行う)ユーザとの間で利益を分けるパートナーシップのモデルです。TorLocker運営はそれをアフィリエイトと呼んでいます。

TorLocker運営からユーザ(この場合は0Chiaki)にTorLockerのコントロールパネルのパスワード、TorLockerをカスタマイズするためのビルダ、必要なバイナリが送られます。ターゲット(被害者)のPCにTorLockerが感染し、身代金としてビットコインが支払われると、TorLocker運営が30%、ユーザが70%の取り分で分け合います。

ビジネスモデルと言っていいのかわかりませんが、アフィリエイト方式はビジネスモデルとして大きな収益が得られるチャンスがあります。あくまでも成功すれば、ですが。ですので今後、ランサムウェアは増えることはあっても減ることはないでしょう。

ただし、そうも世の中うまくは回りません。TorLockerのコントロールパネルのサーバが乗っ取られました。したがってTorLockerに感染した人がビットコインを払ったところで、元に戻す鍵を入手できません。つまり、TorLockerに感染する=ファイルを永遠に失うということになります。もちろんユーザにもTorLocker運営にもビットコインはまわりません。

またTorLockerの暗号化部分はScraperというマルウェアからの流用で、そのScraperに瑕疵(かし)があり、AES-256/RSA-2048という強力な暗号の組み合わせにもかかわらず、70%以上のファイルが復元できるという報告[注9]がカスペルスキー研究所(Kaspersky Lab ZAO)から出されています。

まとめ

ランサムウェアは欧米では蔓延しており、日本に本格的に上陸してくるのも時間の問題です。ランサムウェアをめぐる環境は、インフラを運営し開発環境を提供する側と、マルウェアをターゲット向けにカスタマイズし送る側とで分業する段階に達しています。そして、そのような環境では17才の少年でもランサムウェアを使ったサイバー犯罪パートナーとなる、そんな事例まで現れる時代になったことを我々は認識しておかなければなりません。SD

注7) http://www.asahi.com/articles/DA3S11836202.html
注8) http://www.yomiuri.co.jp/it/security/goshinjyutsu/20141219-OYT8T50085.html
注9) https://securelist.com/blog/research/69481/a-flawed-ransomware-encryptor/

ShowNetが示すネットワークの近未来

最終回 ShowNet 2015 Scratch & Rebuild the Internet Phase 2総括編

インターネット技術とビジネスが出会う国内最大のイベント「Interop Tokyo」。今年も6月10～12日に幕張メッセで開催され、昨年を約4,000名上回る136,341名の来場者を迎えて盛況の後閉幕しました。他展では類を見ないその最大の特徴である"ShowNet"は、会場全体に構築される最先端の技術を駆使したネットワークです。6回に渡ってお届けしてきた本連載もいよいよ最終回。今回はShowNet 2015での活動についてご紹介していきます。

Writer 櫨山 寛章（はぜやま ひろあき）
奈良先端科学技術大学院大学
Writer 大嶋 康彰（おおしま やすあき）
㈱ナノオプト・メディア
URL http://www.interop.jp

ShowNet 2015の取り組みについて

過去の連載で紹介してきたとおり、"今のインターネットを見直す"という3年がかりのプロジェクトの2年目となった今回は、Scratch & Rebuild the Internet Phase 2 ―ULTIMATE BALANCE―をテーマとして、簡潔性（simplicity）と柔軟性（flexibility）、高信頼性（reliability）といった一見相反する要素をバランスよく実現した近未来のネットワークを目指しました。

ShowNetはインターネット全体を模倣して作られることから、前述の3つの要素が通信事業者、サービスプロバイダ、データセンタ（DC）・クラウド事業者、エンタープライズのネットワークの世界に散りばめられています。ここからはそれぞれの要素における取り組みをいくつか紹介していきます。

簡潔性（simplicity）実現に向けての取り組み

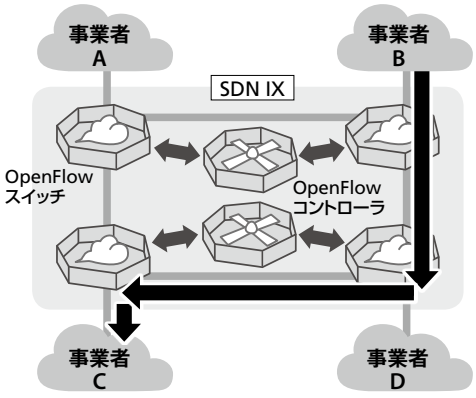
ShowNetではより簡素なネットワークを実現するため、昨年からルータやスイッチなどの機器の物理的な配置を見直すことをはじめ、論理的なネットワーク設計においても工夫をしています。仮想化、SDN（Software Defined Network）、NFV（Network Functions Virtualization）を採用することによって、経路制御における自動化を実現し、運用負荷を軽減することに成功しています。

出展社へのネットワークサービスではOpenStackを用いたセルフ仮想マシン環境を構築しました。また、ネットワークにBGP Flow specを実装し、DDoS対策などのセキュリティ対策機能を用いることによって必要機器を削減し、全体ネットワーク設計に簡潔性を持たせました。

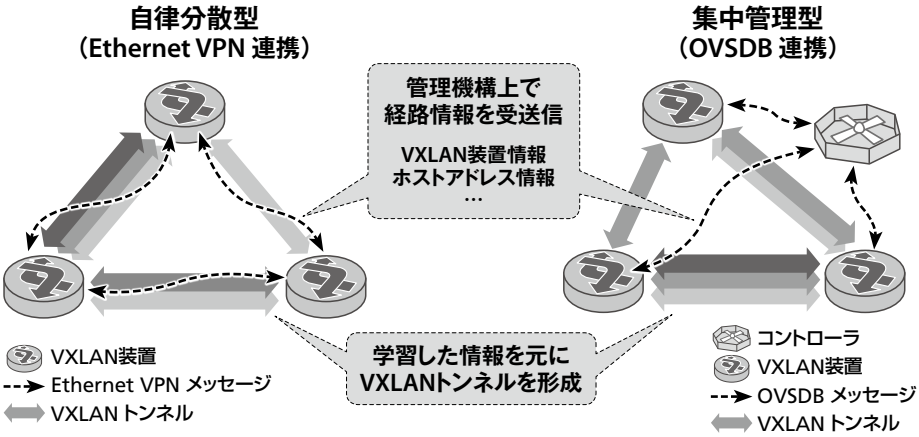
柔軟性（flexibility）実現に向けての取り組み

増え続けるトラフィックやインターネットにつながる端末の増加と多様化、それに伴うネットワークへの接続形態の変化などに対応するため、柔軟性のあるネットワーク構築の実現は今後のインターネット業界にとって大変重要な課

▼図1 事業者間をまたぐレイヤ2パスをOpenFlowを用いて自動でIX上に構築



▼図2　経路管理機構を用いたVXLAN相互接続実験



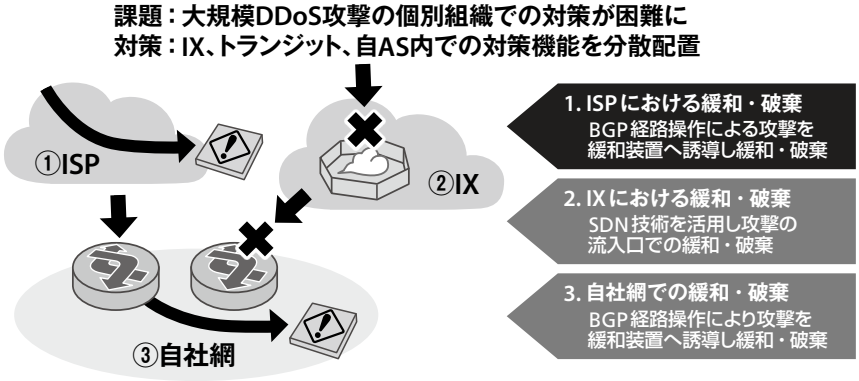
題です。今年のShowNetでは柔軟性を実現する技術として、SDNをIX（Internet eXchange）の基盤に採用しました。クラウド事業者間や通信事業者間で顧客ネットワークを接続する用途を想定し、事業者間をまたぐレイヤ2パスをOpenFlowを用いて自動でIX上に構築することによって、細かな経路制御や柔軟なパス交換を実現しました（**図1**）。また、ShowNet内のネットワークでは、柔軟性を維持しながら容易にスケールアウト可能なNFV環境を既存技術のみで構築しました。開催期間中は、実際に構築したNFVを用いて一部の展示会出展者を収容し、インターネット接続サービスを提供しました。

DC・クラウドネットワークでは、従来のネットワークに、より高い柔軟性を実現可能なVXLANの相互接続実験を実施しました。VXLANは現在主流であるVLANに対し、識別子空間が広いため、その拡張性の高さを期待されています。しかし、経路管理機構が不十分であることから、サービス網への適用にはいくつかの課題があります。今回のShowNetでは、Ethernet VPN、およびOVSDB（Open vSwitch Database）Management Protocol を VXLAN の経路管理機構として用い、各構成において相互接続性を確認しました（**図2**）。

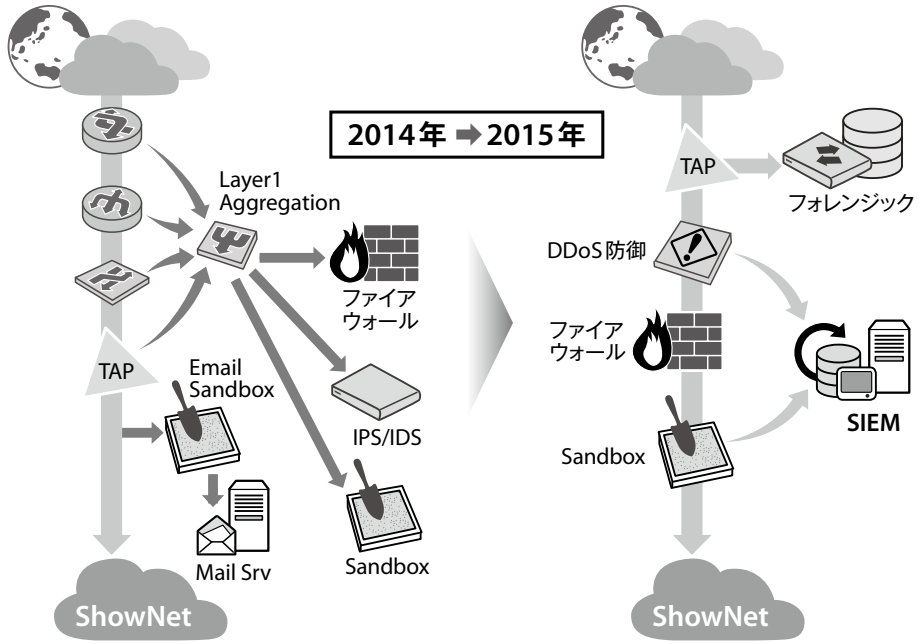
高信頼性（reliability）実現に向けての取り組み

ネットワーク利用者にとっての高信頼性という意味では、安全（セキュア）・快適（高パフォーマンス）ということが重要な要素になると考えられます。今年のShowNetでは、大規模なDDoS攻撃を個別組織で対策することが困難になってきていることから、DDoS対策機能をISP、IX、自社網に最適分散配置するという試

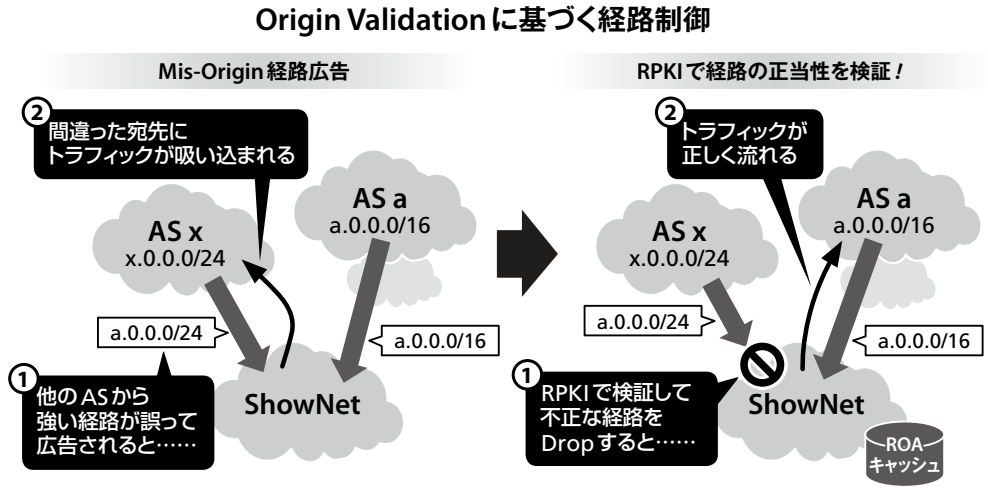
▼図3　DDoS対策機能の最適分散配置



▼図4　ShowNet 2015のセキュリティのポイント



▼図5　RPKI相互接続実証実験

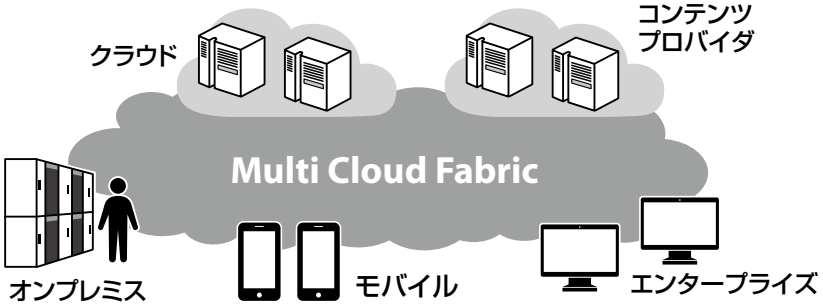


みを行いました(**図3**)。

また、さまざまな攻撃手段への対策としてインライン、多層防御、フォレンジック、SIEM(Security Information and Event Management)の運用といったセキュリティオーケストレーションモデルを実装することで、より安全なネットワークを構築しました(**図4**)。さらに、対外組織との接続部分では、通信事業者間での経路広告の信頼性と安全性を向上させるRPKIと呼ばれる公開鍵基盤(PKI)を用いて、正当なIPアドレス資源の所有者を証明するしくみを導入しました。このRPKIでは、実運用を行うとともに複数社のルータ間で相互接続実験を実施しました(**図5**)。

▼図6　Multi Cloud Fabric

さまざまなネットワークが接続可能なオープンな
プライベートコネクトプラットフォームを目指して



前述のIXや出展社ブース向けのネットワークにも採用されたSDN/NFVでは、柔軟なネットワークの実現に加えて、OpenFlowによる負荷分散やパケットI/O高速化技術の採用による高パフォーマンス化への取り組み、SDN技術を活用したDDoS攻撃流入口での緩和や破棄といったセキュリティ対策を実施しました。

DC・クラウドの分野では、企業やコンテンツプロバイダからの障害対策やオンプレミスとの併用、ワークロード分散などでニーズが高まっているマルチクラウド活用にも取り組みました。課題となっている品質の保証、障害時の切り分けの複雑性、セキュリティへの懸念などの解決として、Multi Cloud Fabricをコンセプトとした、さまざまなネットワークが持続可能でオープンなプライベート接続基盤の実現を目指しました(**図6**)。

Phase 2から何が見えたのか？

ここまでPhase 2の3つの要素における取り組みを紹介してきましたが、いくつかのテーマにまたがる技術キーワードが存在すること、バックボーンネットワークにもセキュリティ機能が実装されることなど、これまでの役割にプラスアルファがあります。これからのインターネットを支えていくためには各技術が単に連携するだけでなく、うまくバランスされた環境が必要となるでしょう。

たとえば自動化されたネットワークも、性能が出なくてはこれからも増え続けるネットワークトラフィックに対応することができませんし、多様化するユーザニーズやセキュリティの脅威にも柔軟に対応できなくてはなりません。今年のShowNetでは、仮想化とセキュリティ技術をネットワークで連携させることによって、簡潔性(simplicity)、柔軟性(flexibility)、高信頼性(reliability)の3要素をバランスよく実現した、近未来のネットワークの1つのモデルを提案できたのではないかと自負しています。

本連載でご紹介した各技術分野における詳細については、後日Interop Tokyo公式サイトに公開される報告書を参考にしてください。SD

COLUMN

Phase 3に向けて

2016年のInterop Tokyoは6月8～10日に幕張メッセでの開催が決定しています。今年の秋頃から次回に向けての設計が始まります。いよいよ次は3年越しのプロジェクトの最終回！　ぜひ次回に向けてもご期待ください。会合の模様など、ShowNet公式SNSアカウントより発信していますので、ご覧ください。

▶**Interop Tokyo公式ページ**　http://www.interop.jp

▶**公式FaceBook**　https://www.facebook.com/interop.shownet

▶**Twitter**　@ShowNet_NOCTeam

Red Hat Enterprise Linuxを極める・使いこなすヒント

.SPECs ドット・スペックス

第15回 Planner in JBoss BRMS 6.1

2015年4月にRed Hatミドルウェア製品であるJBoss BRMS 6.1がリリースされました。Business Rule Management Systemとして歴史を刻んできた同製品には、新たにPlannerと呼ばれる機能が追加されフルサポートが開始されました。今回はこのPlannerについて説明します。

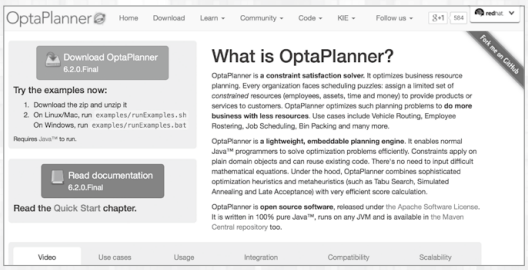
Writer レッドハット(株)サービス事業統括本部
プラットフォームソリューション統括部ソリューションアーキテクト部長 藤田 稜 (ふじた りょう)

OptaPlannerをいじってみよう

Plannerは Business Resource Planningを扱う機能です。限られたリソースを配分して最大の効率あるいは利益を上げるというと簡単に聞こえますが、現実には「完全解」を得られることはほとんどなく、「最適解」――つまり現実的には妥当で最善の解を得られれば十分であることが経験的にわかっています[注1]。

Plannerをきちんと理解するのはなかなか難しいのですが、Plannerのコミュニティ版であるOptaPlannerのサンプルを触ってみればこの機能の面白さが直感的にわかると思います。OptaPlannerのサンプルはoptaplanner.orgからダウンロードできます(**図1**)。

▼図1 optaplanner.orgのトップページ、左上にサンプルのダウンロードリンクがある



▼図2 サンプルプログラムのメインメニュー

ダウンロードしたファイルを解凍し、"examples"ディレクトリの下にあるrunExamples.sh[注2]を実行するとサンプルプログラムが起動します[注3](**図2**)。

サンプルを動かしてみる

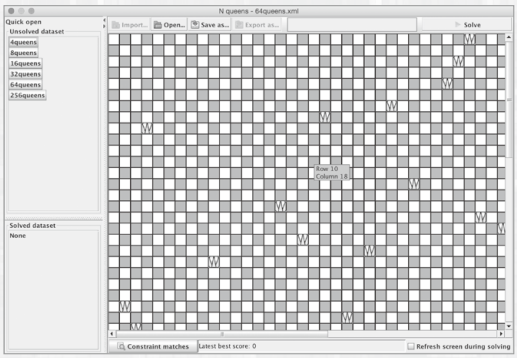
サンプルに含まれるのはいわゆるNP完全問題[注4]として代表的なものです。NP完全については次回に解説します。今回は簡単にサンプルを紹介します。

N queens

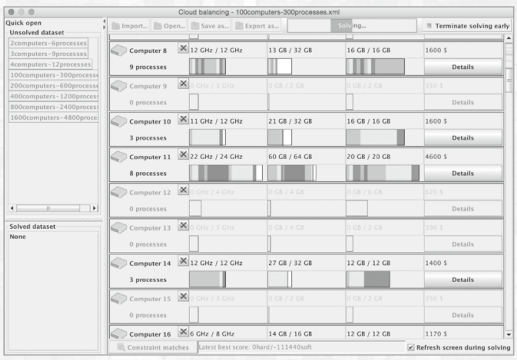
チェスの盤面に、いずれのクイーンからも取

注1) より正確には、現実的な時間内に完全解を得られない一群の問題に分類される。
注2) Windowsではrunexamples.bat
注3) 起動するにはJavaの実行環境が必要。OpenJDKあるいはOracle Javaを事前にインストールしておく。
注4) Non-deterministic Polynomial time、非決定性多項式問題。

▼図3　N queens



▼図4　Cloud Balancing



られないようにクイーンを配置する問題で、もともとはチェスの8×8の盤面に8つのクイーンを配置しますが、このサンプルでは256×256の盤面に配置する問題までを扱います(**図3**)。

Cloud Balancing

クラウドコンピューティングにおける、コンピューティングリソースの最適化問題で、広義にはBin Packing Problem、BPPと呼ばれる問題です。各プロセス(仮想マシン)が必要とするCPU、メモリ、ネットワーク帯域を満たし、かつコストが最小となるような組み合わせを求めます。マシンにトラブルがありシャットダウンした場合に再配置が行われる様子も見られるサンプルになっています(**図4**)。

Traveling Salesman

Traveling Salesman Problem[注5]、「巡回セールスマン問題」は有名ですね。サンプルにはアメリカ合衆国のいくつかの州の都市を最短距離で回るシナリオが含まれており**図5**に挙げたものはフロリダ州の巡回問題になっています。最短距離で巡回する＝最小の燃料消費という設定ですが、もちろん州間高速[注6]を利用して最短時間で巡回する、といった最適化問題も扱えます。

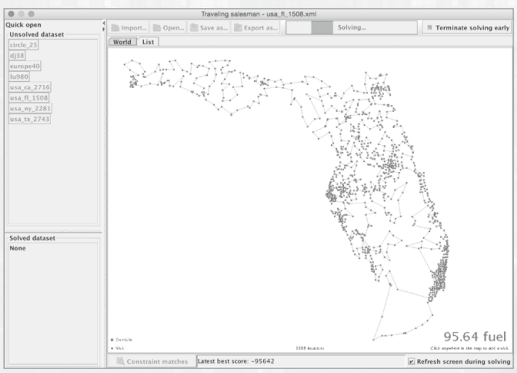
Course Timetabling

学校の時間割を最適化する問題で、Cloud Balancingと同じBPPに分類されます。小中高の時間割ぐらいであれば問題空間がそれほど広くないので、人力でも解くことができますが、大規模な総合大学であればかなり複雑な問題となり得ます。教室のキャパシティ、設備、常勤・非常勤講師の別など、さまざまな制約条件を可能な限り満たすように時間割を組むのは難しい問題です(**図6**)。

Vehicle Routing

TSPと似ていますが、こちらはVehicle Routing Problem、VRPと呼ばれ区別されます。1つの配送拠点から可能な限り少ない台数の配

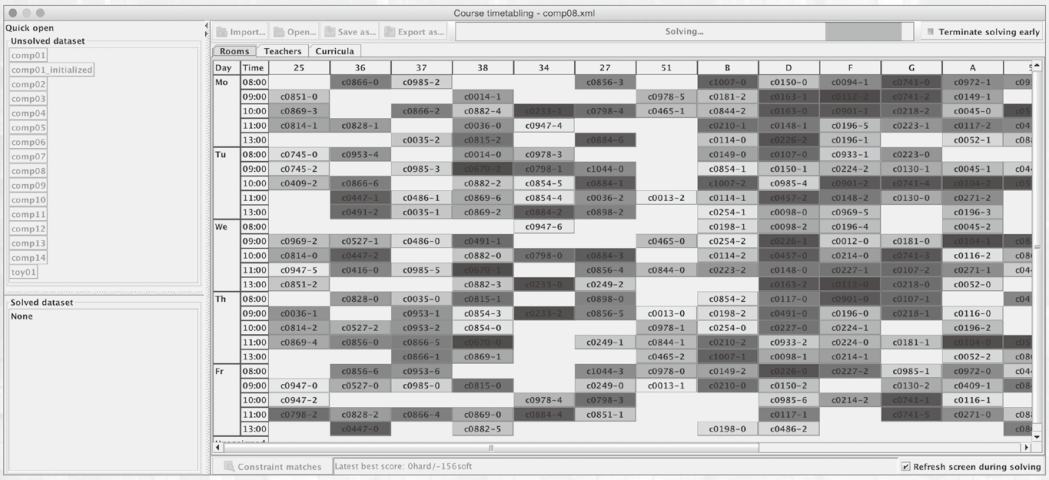
▼図5　Traveling Salesman Problem



注5) Political Correctnessを考慮するとSalespersonですが、いずれの場合もTSPと省略される。

注6) 州間高速はInterstateと呼ばれ、その表示に用いられる字形が同名のフォントになっており、Red Hatの製品ロゴに使われている。なお、"redhat"というロゴに用いられているフォントはMyriadという書体。

▼図6　Course Timetabling



送トラックで定められた時間内に多くの配送先をカバーする、というシナリオです。サンプルでは動作中に画面をクリックすることで配送先を増やすことが可能となっており、その際にそれまでに得られている最適解を棄却するのではないことがポイントです(**図7**)。

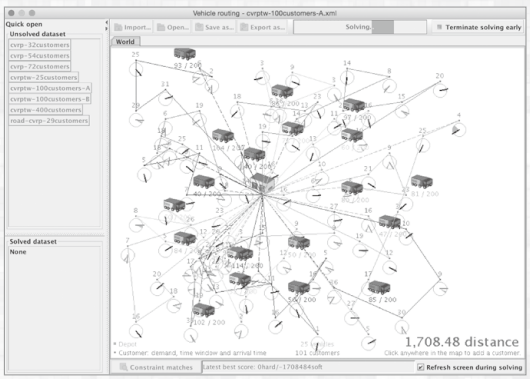
Cheap Power Time Scheduling

ICON Challenge on Forecasting and Scheduling[注7]を解くもので、各マシンに必要なリソースを配分しつつ、電力消費(コスト)を最小化する問題です(**図8**)。

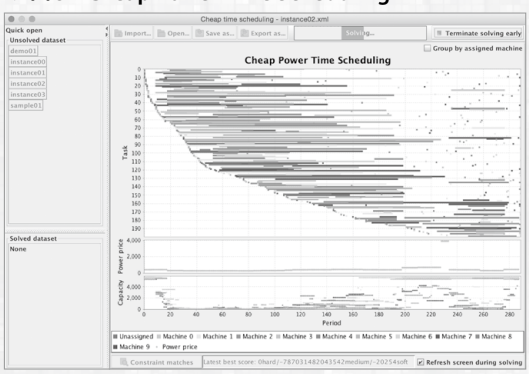
まとめ

紹介したサンプルの動作の様子を誌面で表現するのは難しく、その面白さがわかりにくいかもしれません。ぜひ、サンプルをダウンロードして自分のマシンで動かしてみてください。Plannerは現実世界の幅広い領域の問題を比較的簡単に解決することを目的としており、利用するエンドユーザに深い理解を要求しない点が魅力です。読者の皆さん、あるいはエンドユーザが直面している問題を解決できるかもしれません。

▼図7　Vehicle Routing



▼図8　Cheap Power Time Scheduling



次回はNP完全問題やアニーリングといった、Plannerの理解に必要な説明をする予定です。SD

注7）http://iconchallenge.insight-centre.org/challenge-energy

技術評論社

ゲームプログラマのための コーディング技術

多くの機能を持つゲームのプログラムは、巨大で複雑になりがちです。また、コードの保守、機能追加などの工程には複数のプログラマが関わることになります。
そのため、ゲームのプログラムには「わかりやすいコード」と「効率よく機能を追加できる設計」が求められます。これはゲームに限らず、職業プログラマとして必要なコーディング技術です。
本書ではC++のサンプルコードをもとに、すぐに実践できるコーディング技術を解説していきます。
まず、コードの抽象化を理解するため、複雑なコードを単純にして小さくするテクニックを紹介します。
続いて、オブジェクト指向設計の原則とパターンにふれながら、シンプルでより良いクラス設計とは何かひも解きます。

大圖衛玄 著
A5判／256ページ
定価（本体2,480円+税）
ISBN 978-4-7741-7413-6

大好評発売中！

こんな方におすすめ　脱プログラミング初心者、新人ゲームエンジニア、ゲーム開発初心者

Software Design plus

技術評論社

サーバ/インフラエンジニア 養成読本 基礎スキル編

Server/Infrastructure Engineer

クラウドコンピューティングの進化や各レイヤの複雑化など、サーバ／インフラエンジニアが習得すべき技術要素は多くなっています。さらに、これらを習得する以前に、Linuxやviエディタを自在に操作し、シェルやPerlでの簡単なプログラミングができることが必要条件となるため、若手には敷居の高い職種と言われます。
そこで本書では、仮想環境上でのLinux（CentOS 7）の構築から、基本コマンドの使い方、viエディタの習得、Perlでログをカスタマイズなど、一度マスターしてしまえば10年先にも必ず役立つ基本的な事柄をまとめました。
本書で、まずは基礎スキルを向上させましょう!

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

大好評発売中！

こんな方におすすめ
・これからインフラエンジニア／サーバ管理者になる人
・現場で利用されているツールを知りたい人、使いこなしてみたい人

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第23回 ❖次世代設定ファイル言語UCL

◉著者プロフィール
後藤 大地（ごとう だいち）
BSDコンサルティング（株） 取締役／（有）オングス 代表取締役／FreeBSD committer
エンタープライズシステムの設計、開発、運用、保守からIT系ニュースの執筆、IT系雑誌や書籍における執筆まで幅広く手がける。カーネルのカスタマイズから業務に特化したディストリビューションの構築、自動デプロイ、ネットワークの構築など、FreeBSD/Linuxを含め対応。BSDコンサルティングでは企業レベルで求められるFreeBSDの要求に柔軟に対応。

FreeBSDが使われるプラットフォームの拡大

FreeBSDはサーバオペレーティングシステムという側面から、さまざまなプロダクトの『プラットフォーム』という位置づけに、その必要とされる場所を変えつつあります。それは大規模なものであればペタバイトクラスのストレージシステム、高性能ネットワークアプライアンス、もちろんエッジサーバでも使われますし、BHyVeベースの仮想化プラットフォーム、サイズが小さくなってくると組込み機器やモバイルデバイスに至るまで、そうしたシーンの『プラットフォーム』として使われています。

このため、近年とくにFreeBSDに求められるものが「どのシーンにも必要に応じて簡単に対応できるようなしくみや構造」になってきています。たとえばすでに実装された機能や、今後実装される機能などを含めて、次のような取り組みが進められています。

- 小さいカーネルと多種多様なカーネルモジュール
- さまざまなアーキテクチャに対応するためのFDT（Flattened Device Tree）のサポート
- Web UIベースの管理アプリケーションとの親和性の向上
- 設定ファイルの管理を容易にするためのUCLの導入
- GPLを嫌うベンダ向けにベースシステムのコマンドをBSDライセンス実装へ置き換え

現在のカーネルはデフォルトで基本的なドライバや機能をスタティックに取り込んでいます。これを必要最小限のカーネルに変更すること、また起動時に必要なカーネルを読み込んで動作するように変更することが考えられています（メモリの少ないデバイスでも快適に動作でき、細かい機能調整がカーネルの再構築なしで可能になる）。

上記取り組みのなかでもユーザの視点から考えると、設定ファイルをUCL（Universal Configuration Language）へ移行させる取り組みが進んでいることは、そろそろ知っておいたほうがよいかもしれません。今回はUCLの基本的なアイディアと目的、UCLへの移行にともなって導入が検討されている操作コマンドの使い方などを紹介します。

人が扱いやすくてソフトウェアからも扱いやすい設定ファイル

/etc/にインストールされるシステムの設定ファイル――さまざまな設定ファイルがありますが――これらは“歴史的に随時”導入されてきたことから、フォーマットもさまざまです。UCLはこれらを統合するための設定ファイルフォーマットのようなものです。

実際にものを見てもらったほうが理解が早いでしょう。ログローテーションを実施するnewsyslog(8)の設定ファイルnewsyslog.conf(5)のフォーマットは**リスト1**のようになっています。

これをUCLで書き換えると**リスト2**のようにな

各コマンドのうしろについている括弧書きの数字は、manコマンドで見ることができるマニュアルに記載されている章番号を表しています。

ります。

1行目に書いてあるのはUCLのバージョン1であることを示す識別子です。UCLのフォーマットはNginxの設定ファイルによく似ています。これはUCLの取り組みを進めているAllan Jude氏がNginxの設定ファイルを扱いやすいフォーマットだと感じたことに理由があります[注1]。

UCLはJSONやYAMLのような書き方をもっと緩く書けるようにしたものと考えておくとよいと思います。ファイルのインクルードやオーバーレイなども可能であるほか、入れ子構造の記述もできます。/etc/の設定ファイルのいくつかはすでにUCLを使うように書き換わっていますし、pkg(8)コマンドもすでにUCLを使っています。

UCLへの移行はスムーズに

UCLへ移行する設定ファイルのいくつかはすでに目処が立っています。しばらくの間は徐々にUCLへの移行が進められる予定になっています。先ほどUCLファイルの先頭に**#fucl1**という表記がありましたが、この表記のないファイルは従来のフォーマットで記述された設定ファイルであるとして、従来のパーサが動作するように処理を切り替える予定でいます。

現在は/etc/rc.conf.local、/etc/rc.conf、/etc/defaults/rc.confのファイルが「システムのデフォルト設定」と「個別に変更した内容」といったようにベースシステムと個別の設定とを分離していますが、UCLへの移行にともなって、設定ファイルそのものに対しても差分だけを設定できるようにする予定になっています。これで従来よりもシステムのアップデートが簡単になります(究極的にはmergemaster(8)が必要ないようにすることを目指します)。

UCLはシェルスクリプトや管理ツールからも簡単に操作できる必要があります。これを実現するためのライブラリがlibucl、コマンドが**uclcmd**(1)です。

uclcmd(1)でUCLをマニピュレーション

libuclとuclcmdは開発段階にあります。古いバージョンのlibuclはFreeBSD 10.1-RELEASEにも導入されていますが、最新の機能を使用するには最新版が必要です。ここではuclcmdの最新版を取得してきて使ってみましょう。

libuclとuclcmdの最新版は**図1**および**図2**のようにgit(1)コマンドを実行して、Allan Jude氏のGitHubから持ってきます(gitは**pkg install git**のようにパッケージからインストールしておきます)。また、libuclおよびuclcmdのビルドにはautoconf、automake、libtoolが必要になりますので、

▼リスト2　UCLで記述したnewsyslog.conf(5)

```
#fucl1
all {
        file = /var/log/all.log
        mode = 600
        count = 7
        size = *
        when = @T00
        compress = bzip2
}
amd {
        file = /var/log/amd.log
        mode = 644
        count = 7
        size = 100kb
        when = *
        compress = bzip2
}
```

▼リスト1　現在のnewsyslog.conf(5)

```
# logfilename          [owner:group]    mode count size when  flags [/pid_file] [sig_num]
/var/log/all.log                        600  7     *    @T00  J
/var/log/amd.log                        644  7     100  *     J
```

注1　アイディアの最初の段階ではMac OS Xのlauchd(8)が使っているバイナリXML plistなどを使ってはどうか、という話もありましたが、すべてのデータを単一の設定ファイルにしたり、XMLを使うというのは、あまり人に優しいとは言い難いというのがAllan Jude氏の考えです。

図3のようにパッケージ経由でインストールしておきます。

最初にlibuclをビルドしてインストールします。クローンしたリポジトリへ移動して図4のようにビルドおよびインストールを実施します。libuclをインストールしたら、uclcmdのビルドとインストールを図5のように実施します。

uclcmdのビルドおよびインストールに成功すると、図6のようにコマンドが使用できるようになります。uclcmdでUCL形式データの値取得、値変更、フォーマット変更(JSON、縮小化したJSON、YAML、UCL)、マージ、削除などを実施できます。

uclcmdの実行サンプル

uclcmd(1)の操作はjq(1)コマンド[注2]の操作によく

▼図1　gitでlibuclのソースコードを取得

```
% git clone https://github.com/allanjude/libucl.git ↵
Cloning into 'libucl'...
remote: Counting objects: 3222, done.
remote: Total 3222 (delta 0), reused 0 (delta 0), pack-reused 3222
Receiving objects: 100% (3222/3222), 2.76 MiB | 260.00 KiB/s, done.
Resolving deltas: 100% (2143/2143), done.
Checking connectivity... done.
```

▼図2　gitでuclcmdのソースコードを取得

```
% git clone https://github.com/allanjude/uclcmd.git ↵
Cloning into 'uclcmd'...
remote: Counting objects: 287, done.
remote: Total 287 (delta 0), reused 0 (delta 0), pack-reused 287
Receiving objects: 100% (287/287), 94.07 KiB | 0 bytes/s, done.
Resolving deltas: 100% (160/160), done.
Checking connectivity... done.
```

▼図3　ビルドに必要になるツールをインストール

```
% pkg install autoconf ↵
% pkg install automake ↵
% pkg install libtool ↵
```

▼図4　libuclのビルドとインストール

```
% cd /pathto/libucl ↵
% ./autogen.sh ↵
% ./configure ↵
% make ↵
% make install ↵
```

▼図5　uclcmdのビルドとインストール

```
% cd /pathto/uclcmd/ ↵
% make ↵
% make install ↵
```

▼図6　uclcmdコマンド

```
% uclcmd ↵
Usage: uclcmd get [-cdejklnquy] [-D char] [-f filename] variable
       uclcmd set [-cdjuy] [-D char] [-f filename] [-i filename] variable [UCL]
       uclcmd merge [-cdjuy] [-D char] [-f filename] [-i filename] variable
       uclcmd remove [-cdjuy] [-D char] [-f filename] variable
中略
            array_1_name="value"

%
```

注2　JSONデータをコマンドから操作する場合に使うコマンド。

似ています。

個別の値を取得する場合には**図7**のようにgetサブコマンドを使います。

フォーマットを変更する場合は**図8**や**図9**のように値として「.」を指定して、あとはオプションで出力するフォーマットを指定します。

値を変更するには**図10**のようにsetサブコマンドを使います。

こうしたライブラリやコマンドのおかげで、Web UI系の管理アプリケーションではJSONデータを扱い、FreeBSD側に来た段階でUCLへ変更、そのまま設定ファイルを更新する、といった流れでシステムの設定ファイルをアップデートする作業がシームレスに実行できるようになります。SD

▼図8　uclcmd：UCLからJSON形式へ変換

```
% uclcmd get -f newsyslog.conf -j . ↵
{
    "all": {
        "file": "/var/log/all.log",
        "mode": 600,
        "count": 7,
        "size": "*",
        "when": "@T00",
        "compress": "bzip2"
    },
    "amd": {
        "file": "/var/log/amd.log",
        "mode": 644,
        "count": 7,
        "size": 102400,
        "when": "*",
        "compress": "bzip2"
    }
}
```

▼図7　uclcmd：値を取得

```
% uclcmd get -f newsyslog.conf amd.compress ↵
"bzip2"
% uclcmd get -f newsyslog.conf -k amd.compress ↵
amd.compress="bzip2"
```

▼図9　uclcmd：UCLからコンパクトなJSON形式へ変換

```
% uclcmd get -f newsyslog.conf -c . ↵
{"all":{"file":"/var/log/all.log","mode":600,"count":7,"size":"*","when":"@T00","compress":↵
"bzip2"},"amd":{"file":"/var/log/amd.log","mode":644,"count":7,"size":102400,"when":"*",↵
"compress":"bzip2"}}
```

▼図10　uclcmd：値を変更

```
% printf gzip | uclcmd set -f newsyslog.conf amd.compress -u ↵
all {
    file = "/var/log/all.log";
    mode = 600;
    count = 7;
    size = "*";
    when = "@T00";
    compress = "bzip2";
}
amd {
    file = "/var/log/amd.log";
    mode = 644;
    count = 7;
    size = 102400;
    when = "*";
    compress = "gzip";
}
```

LibreOffice 5.0の変更点

Ubuntu Japanese Team　あわしろいくや

今回は8月上旬にリリース予定のLibreOffice 5.0の変更点についてお知らせします。2年半ぶりのメジャーバージョンアップには、どんな理由があるのでしょうか。

LibreOffice 5.0

LibreOfficeの最初のバージョンは3.3でした。互換性の問題で、バージョニングはOpenOffice.orgを引き継ぐしかなかったのです。4.0がリリースされたのは2013年2月ですので、5.0は2年半ぶりのメジャーバージョンアップ、ということになります。ただし、LibreOfficeはタイムベースリリースであり、それはすなわち時期が来たらリリースされるということで、この機能を実装したらメジャーバージョンアップ、というルールにしているわけではありません。5.0も多数の機能が実装されたということはなく、単純にマーケティングの都合です。現に設定ファイルは4のままだったりします(**図1**)。

あえて大きな変更があった点を挙げるとすれば、それはユーザインターフェースです。とはいえ、これも5.0ではなく4.4からの路線ですので、むしろ4.4を5.0にすべきだったのかもしれません。サイドバーという機能自体はApache OpenOfficeのソースコードから取り込まれたものですが、LibreOfficeで徹底的に磨かれ、より洗練された使い勝手になっています。昨今のワイドディスプレイの普及を見ても、サイドバーの使い勝手を向上するには理にかなっているといえます。

もちろんそれ以外にも、やはり4.4からの流れではあるものの、コンテキストメニューの変更や、上部ツールバーの機能割り当ての変更など、歴史的経緯以外に変更しなかった理由を見いだせない部分にも手が入っています。

メジャーバージョンアップの際にはAPIを整理するという作業も行われています。4.0のときにはかなりアグレッシブな変更があったのですが[注1]、5.0ではいくつか追加されただけであり、非互換は発生しないものと思われます。

あらためて考えてみると、5.0というバージョニングはマーケティングの都合だったとしても、今後歩むLibreOfficeの独自路線のスタートラインに立ったバージョン、と言えるのかもしれません。

図1　設定ファイルの場所は［ツール］-［オプション］-［LibreOffice］-［パス］で確認できます

LibreOfficeによって使用されるパス

種類 ▲	パス
オートコレクト	/home/ikuya/.config/libreoffice/4/user/autocorr
ギャラリー	/home/ikuya/.config/libreoffice/4/user/gallery
テキストブロック	/home/ikuya/.config/libreoffice/4/user/autotext
テンプレート	/home/ikuya/.config/libreoffice/4/user/template
バックアップ	/home/ikuya/.config/libreoffice/4/user/backup
マイドキュメント	/home/ikuya
一時ファイル	/tmp
画像	/home/ikuya/.config/libreoffice/4/user/gallery
辞書	/home/ikuya/.config/libreoffice/4/user/wordbook

注1) もちろん大部分はOpenOffice.orgの負の遺産の棚卸しだったわけですが。

Writer

絵文字の入力

真っ先にオフにされることで有名なオートコレクト機能を使用し、絵文字が入力できるようになりました。入力方法も入力できる絵文字もおおむねEmoji cheat sheet[注2]に準じていますが[注3]、"_"（アンダースコア）の代わりに" "（スペース）が入ります。"Tofu on Fire"[注4]として一躍有名になった":name_badge:"は、LibreOfficeでは":name badge:"と入力します（**図2**）。入力できる絵文字は［ツール］-［オートコレクトオプション］の［置換と例外扱いの言語］を［英語（米国）］などにすると確認できますが、フォントはいわゆるトーフになってしまい、表示できません[注5]。もちろんある程度は推測できるので、とくに問題ないような気はします。

Ubuntu 14.04だと絵文字フォントがインストールされていないので、"ttf-ancient-fonts"パッケージをインストールしてください。15.04ではインストールされています。

そもそもからしてオートコレクト機能で絵文字入力というのは、インプットメソッドを使用しない言語向けの機能です。Mozcは絵文字変換に対応しているので、そちらで入力するほうが簡単です。しかし、どんな読みで変換すれば絵文字が表示されるのかわからない場合や、覚えてしまって直接入力してしまったほうが早い場合には、この機能は使えるのではないでしょうか。ベータ後に取り込まれた機能で未だに喧々諤々議論があるようですが、単純におもしろい機能だと思います。

サイドバーでスタイルのプレビュー

今までは書式設定ツールバーでしかできなかったスタイルのプレビューが、サイドバーの［スタイルと書式設定］でもできるようになりました。

Wordの蛍光ペンと文字の網かけ

Wordには、フォントの装飾に蛍光ペンと網かけ機能があります。一方、LibreOfficeには文字の背景しかありません。機能は似ていて、インポート／エクスポートもなんとなく行えてはいましたが、完全に同じものではないので十分なものではありませんでした。そこで、次のような変更が加えられました。

- Wordからインポートしたファイルに蛍光ペンと網かけがあった場合、変更を加えない場合はエクスポートしてもそのままとする
- 変更した場合はLibreOfficeの文字の背景とする
- 文字の背景をエクスポートした場合、蛍光ペンとするか、あるいは網かけとするかは設定（［ツール］-［オプション］-［読み込みと保存］-［Microsoft Office］）で変更できる

ついでに、［文字の背景］は［Text Highlighting］の訳語ですが、"Highlighting"を"背景"と訳すのは本来正しくありません。"強調"と訳すべきですが、文字を強調する方法は色を変える以外にもフォントを大きくするなどたくさんあります。というわけで、蛍光ペンよりも色数が多い［ラインマーカー］と訳すことにしました。

図2　絵文字の入力例。「:!: :bike: :atm: :bird: :100: :white flower: :name badge:」と入力しました

注2）http://www.emoji-cheat-sheet.com/

注3）もちろん一部ないものもあります。具体的にはGitHubでよく使われる:+1:などです。

注4）http://news.mynavi.jp/articles/2015/03/08/matsumura_apple/

注5）確認したところWindowでも同様でした。筆者が知る限り、絵文字を網羅したUI用のフォントは存在しないので、うまく調整しないとこのトーフを改善することはできないように思います。

画像のトリミング

マウスで簡単に画像のトリミングができるようになりました。大きめの画像を挿入し、ピンポイントで表示したい個所がある場合に便利でしょう。方法はいくつかあり、画像を選択した状態で、

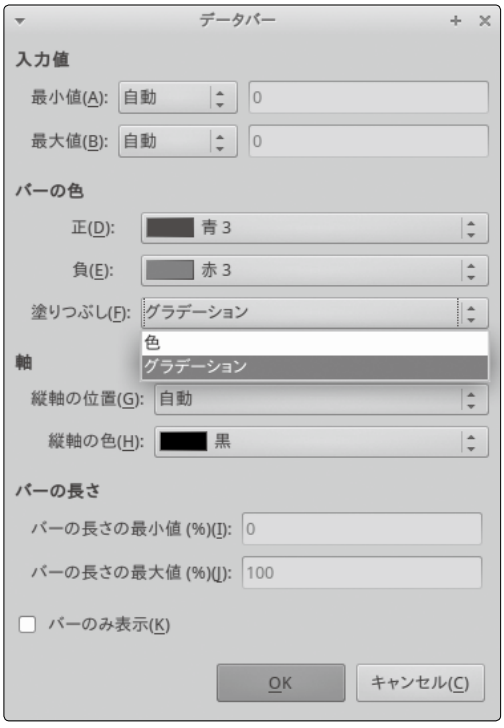
- 右クリックで［画像のトリミング］をクリックする
- ツールバーの［画像のトリミング］をクリックする
- ［書式］-［画像］-［画像のトリミング］をクリックする

のいずれかで実行してください。

ページ番号の表示

左下に表示されるページ番号ですが、ページをスクロールした場合、今までは次のページが完全に表示されるまでは前のページのままでした。たとえば2ページのドキュメントがあったとして、上から徐々にスクロールしていって、1ページを完全に表示しなくなってからページ数が2/2になっていました。ただ、これはあまり自然な動きではなく、いくらか2ページめを表示したらページ数を2/2にするべきです。というわけで、2ページめを43%表示したらページ数が2/2になるようになりました。おもしろいことに、この43%というのはソースコードにそのまま書いてありました。おそらく数字自体に意味はなく、だいたいこのくらい表示したらいいだろうというところから来ているものと思われます。

図3　強化されたデータバーの設定画面



表の管理機能の強化

たとえば5行の表があったとして、あと2行追加したいという場合、2行を選択して行を追加すると選択した分、すなわち2行を追加できるようになりました。いまいちどういうニーズがあるのかはよくわからなかったのですが、Calcと挙動を合わせたということのようです。あと、セルの右クリックから［挿入］-［上に行の挿入］［下に行の挿入］［左に列の挿入］［右に列の挿入］ができるようになりました。これはわかりやすく便利になったというか、今までできなかったのが不思議なくらいです。

OOXMLとの相互運用性向上

arcToを使用したWordのシェイプを正しく表示できるようになりました。LibreOfficeのサポート企業であるCollaboraの方が修正しているので、おそらく有償サポートで依頼があったのでしょう。

Calc

条件付き書式の強化

条件付き書式のデータバーが強化されました。今まではグラフはグラデーションになっていましたが、これをグラデーションとグラデーションなしの両方から選べるようになりました。また、数字なしのバーとバーの長さの上限も設定できるようになりました（**図3**）。

指数表記の強化

右クリック-［セルの書式設定］-［数値］タブ-［指数表記］で［エンジニア表記］が選択できるようになりま

した。通常の指数表記では、たとえば“1.11E+29”となるところを、［エンジニア表記］にチェックを入れると“111.11E+27”となります。筆者がこの手の表記にうといことを差し置いても、日本語ではあまり有用な情報を検索できませんでしたが、バグ報告をたどっていくと最初に報告された[注6]のは2002年6月17日とOpenOffice.org 1.0リリース直後で、13年越しで実装されたことになります。

関数の追加

FLOOR関数とCEILING関数は2番めのパラメータはオプションになりました。1番めと2番めのパラメータが必須の関数として、新たにFLOOR.XCLとCEILING.XCLが追加されました。ExcelのFLOOR.XCLあるいはCEILING.XCL関数をインポートした場合、CalcではFLOOR.XCLあるいはCEILING.XCLに置き換えられます。

ExcelにもあるFLOOR.MATHとCEILING.MATH関数がCalcでもサポートされました。FLOOR.XCLあるいはCEILING.XCL関数をExcel形式でエクスポートした場合、FLOOR.MATHあるいはCEILING.MATH関数に置き換えられます。FLOOR.XCLあるいはCEILING.XCL関数をExcel形式でエクスポートした場合は、FLOORあるいはCEILING関数で置き換えられます。非常にややこしいですが、自動的に処理するのであまり気にしなくてもよさそうです。

ほかにも、ENCODEURL関数とERROR.TYPE関数が追加されました。いずれもExcelとの相互運用性向上のためであり、またとても便利そうな関数です。

すべての行／すべての列の範囲指定

今までは、たとえばすべての行を指定する場合はA1:AMJ1、すべての列を指定する場合はA1:A1048576と範囲指定する必要がありましたが、5.0からは行の場合は1:1、列の場合はA:Aで範囲指定できるようになりました。もちろん今までの表記でも問題ありません。全部の列または全部の行を指定することはよくあることですので、とても便利になりました。

画像の扱い

Calcでも画像のトリミングや画像の置き換え、保存ができるようになりました。

OOXML/XLSXとの相互運用性向上

そのほか、OOXML/XLSXとの相互運用性を向上する改善が行われています。

塗りつぶしのカラーパレットの変更

LibreOffice 4.4から登場した新しいカラーパレットは、塗りつぶしの場合はサイドバーからしか使用できませんでした。しかし、5.0からはツールバーにある塗りつぶしアイコンからでも新しいカラーパレットが表示できるようになりました。

テキストでも背景色

テキストボックスでも背景色が設定できるようになりました。

PDFエクスポートの機能追加

PDFエクスポートで、タイムスタンププロトコルがサポートされました。ただし、残念ながら今回検証は行えませんでした。認証局に制限があったりするのでしょうか。

Adobe Swatch Exchange (.ase) サポート

aseはAdobe InDesignやPhotoshopで使用されているカラーパレットだそうです。これのインポートができるようになりました。興味深いのは機能そのものより、FreedomSponsorsというサービスを使用

注6）https://bz.apache.org/ooo/show_bug.cgi?id=5930

し、機能を実装した人に寄付を行ったことでしょう。日本にもこのようなマッチングサービスがあればいいのではないかと思います。もっとも、実装する人がいないかもしれませんが……。

インポートフィルタ

インポートフィルタの追加

WriterではApple Pages('09以前)、CalcではApple Numbers('09以前)、Lotus 1-2-3(wk3とwk4)、Quattro Pro(wq1とwq2)、DrawではClaris DrawとMacDraft(v. 1)のインポートフィルタが追加されました。残念ながら筆者の手元にはこれらのファイルはないため、確認できませんでした。MacDraftは初耳ですが、老舗のCADソフトだそうで、日本語版は発売されていないようでした。

その他

その他にもKeynote、MS Works、Adobe/Macromedia FreeHandインポートフィルタが強化されています。

GUI

大幅なデザインの変更(Impress)

Impressでは大幅にデザインが見直されました(**図4**)。2段あったツールバーは1段になり、縦に広く使えるようになった分、横に延びました。その余裕を利用して、レイアウトにもスペースが入ったりもしています。

図形描画ツールバー(Draw)

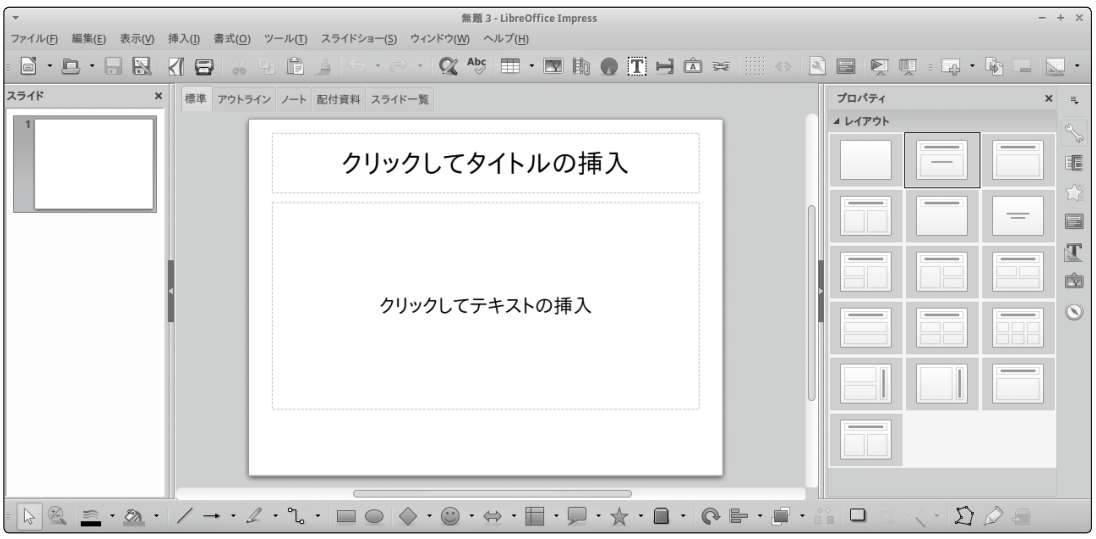
Drawでは、今まで下にあった図形描画ツールバーが縦に表示されるようになりました(**図5**)。たしかにDrawではページを縦置きで使用することが多いため、下にツールバーがあるとそれだけ表示面積が小さくなってしまいます。逆に横置きで使用することが多いImpressでは、今のままがベストでしょう。考えてみれば当然の変更です。

サイドバーの仕様変更

次のサイドバーに仕様変更がありました。

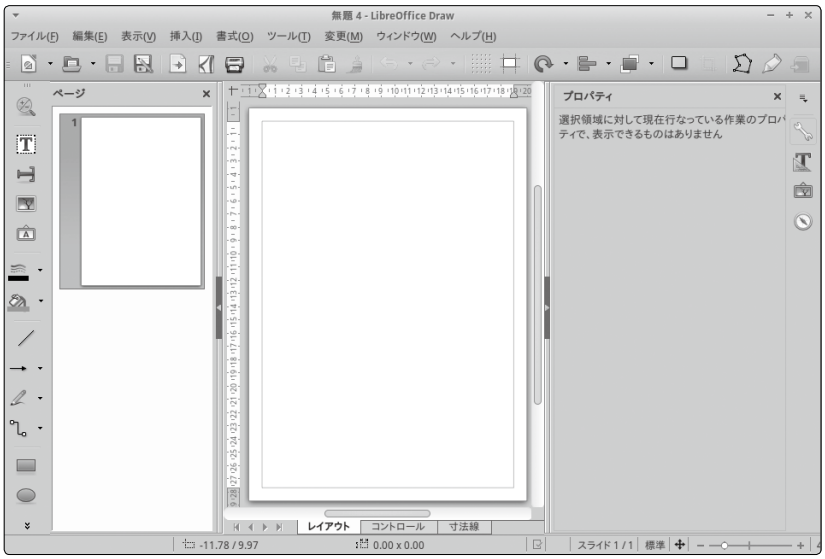
- Impressの画面切り替えタブ

図4　Impressは全体的に横幅を広く取るようになりました。また、サイドバーのレイアウトにもスペースが入っています



- Impressのタブの順番
- Calcのプロパティタブの配置セクション
- Impress/Drawのプロパティタブの［段落］セクションに［インデントを増やす］［インデントを減らす］ボタンの追加
- Impress/Drawのプロパティタブの［段落］セクションに［レベルを下げる］［レベルを上げる］［下に移動］［上に移動］ボタンの追加

図5　Drawは縦幅を圧縮するために横幅を広く取ったという感じです



タブの仕様変更

Calcのタブの仕様が変更になりました。まず、最初と最後のシートに飛ぶボタンがなくなり、シート移動のボタンが半減しました。最初か最後のシートに飛びたい場合は、Ctrl キーを押しながら左右のシート移動ボタンをクリックしてください。また、追加ボタンがタブの左側に移動したことにより、複数のシートを作る場合にそのままクリックすればよくなりました。今までは追加ボタンはタブの右側にあったため、タブを増加すると追加ボタンも右にずれたので、専属してタブを作成する場合はマウスを右にずらす必要がありました。タブをたくさん作ってスクロールが必要になった場合は、マウスホイールの回転によってタブの表示をスクロールできるようになりました。これらも、どうして今までこうならなかったのかが不思議な変更で、かなり使い勝手が向上すると思います。

機能削除

ユーザインターフェースにシステムフォントを使用

古くからのOpenOffice.orgユーザの中には見覚えがある人も多いであろう、［ツール］-［オプション］-［表示］-［ユーザインターフェースにシステムフォントを使用］機能が削除されました。常にシステムフォントを使用するため、設定する必要がないとのことです。

Windows版

64bit版の提供

Ubuntuにはまったく関係ありませんが、5.0からWindows用の64bit版インストーラも配布するようになりました。

インストール方法

10月にリリースされるUbuntu 15.10とそのフレーバーはLibreOffice 5.0がパッケージングされる見込みです。そのほかのUbuntuでは、PPA[注7]を確認してください。SD

注7）https://launchpad.net/~libreoffice

第42回

Linux 4.1の機能
～カーネルの動的更新livepatch

Text：青田 直大　AOTA Naohiro

Linux 4.2-rc3が7月19日にリリースされています。このまま順調にいけば、この雑誌が出るころにはLinux 4.2がリリースされているかもしれません。今月はLinux 4.1のlivepatch機能について紹介します。

カーネルの動的更新：livepatch

ユーザランドのセキュリティパッチに比べ、カーネルのセキュリティパッチの適用は困難です。パッチの適用には、新しいプログラムをインストールし、既存のプロセスを再起動するという2つのステップがあります。ユーザランドのプログラムであれば、そのプロセス単体の再起動でパッチの適用を完了できますが、カーネルの場合にはパッチの適用にはシステム全体の再起動が必要となり、サービスの停止時間が長くなってしまいます。

そこでカーネルを動的に書き換え、システム起動中にパッチの適用されたカーネルに切り替えるlivepatchという機能が開発されています。この機能を使うことで、カーネルを再起動することなくセキュリティパッチを適用できるようになります。

Linuxカーネルにおけるlivepatch機能は、現在3つの実装があります。最も歴史が古いのがKspliceです。現在はOracleにより管理され、Oracle が OracleLinux、Red Hat Enterprise Linux用のパッチを有料で、あるいはUbuntu、Fedoraデスクトップ用のパッチを無料で配布しています。残念ながらKspliceは、Linuxカーネル本体へのマージは行われませんでした。

そこでカーネル本体へのマージを考慮に入れて作られたのが、Red Hatのkpatchおよび SuSEのkGraftです。

これら2つの実装はそれぞれLKML（Linux Kernelの開発メーリングリスト）にほぼ同時期に投稿されました。そのあと、2つの実装間の比較や議論が行われ、ひとまずkpatchとkGraftの両方が使うコア部分のみを“livepatch”として、Linux 4.0にマージすることになりました。

livepatchモジュールの構造

まず、現在Linuxにマージされているlivepatchがどのように動いているのかを見ていきましょう。Linuxカーネルには、livepatch機能とともにlivepatchを使うサンプルコードもマージされています。このサンプルコードはサイズ

▼リスト1　livepatchのサンプルをビルドする

```
Kernel hacking --->
 [*] Sample kernel code  --->
  <M>   Build live patching sample -- loadable modules only
```

も小さく、livepatchの使い方を知るには最適です(**リスト1**)。kernel configのリストに示す設定(SAMPLE_LIVEPATCH)を有効にすることで、サンプルのpatchモジュールがビルドされます。このpatchを適用すると、本来カーネルへの引数を出力する/proc/cmdlineの出力が"this has been live patched"に置き換えられます。

patchモジュール(livepatch-sample)をmodprobeすることで、livepatchが適用されます(**図1**)。このとき、同時に/sys/kernel/livepatchの下に読み込まれたlivepatchを示すエントリが作られます。livepatch_sampleディレクトリの下の"**enable**"はpatchが有効かどうかを制御し、vmlinux/cmdline_proc_showは"**vmlinux**"(カーネル本体)の"**cmdline_proc_show**"という関数がpatchされたことが示されています。ここで"**enable**"に"**0**"を書くことで、livepatchがrevertされ、ふたたび/proc/cmdlineでカーネル引数が表示されるようになります。

それでは、サンプルのソースコード(**リスト2**)を見ていきましょう。このコードを理解する上で重要なのは、❶関数"**livepfatch_cmdline_proc_show**"、❷構造体"**struct klp_patch patch**"、❸関数"**livepatch_init**"の3つです。

まず、関数livepfatch_cmdline_proc_show()は

▼リスト2　サンプルpatchモジュールのソースコード(抜粋)

```
#include <linux/seq_file.h>
static int livepfatch_cmdline_proc_show(struct seq_file *m, void *v) ……❶
{
    seq_printf(m, "%s\n", "this has been live patched");
    return 0;
}

static struct klp_func funcs[] = {
    {
        .old_name = "cmdline_proc_show",
        .new_func = livepatch_cmdline_proc_show,
    }, { }
};

static struct klp_object objs[] = {
    {
        /* name being NULL means vmlinux */
        .funcs = funcs,
    }, { }
};

static struct klp_patch patch = { ……❷
    .mod = THIS_MODULE,
    .objs = objs,
};

static int livepatch_init(void) ……❸
{
    int ret;

    ret = klp_register_patch(&patch);
    if (ret)
        return ret;
    ret = klp_enable_patch(&patch);
    if (ret) {
        WARN_ON(klp_unregister_patch(&patch));
        return ret;
    }
    return 0;
}
```

▼図1　サンプルpatchのテスト

```
# cat /proc/cmdline                    # 本来のカーネル引数
BOOT_IMAGE=/boot/vmlinuz-4.1.1-gentoo-r1 root=UUID=0bb07f44-2cbe-402c-84c9-253e3d1d7ce6 ro init=/usr/lib/systemd/systemd
# modprobe livepatch-sample            # patchの読み込み
# cat /proc/cmdline                    # patchが適用され、カーネル引数が表示されなくなった
this has been live patched
# find /sys/kernel/livepatch/          # /sys/kernel/livepatch下にエントリができる
/sys/kernel/livepatch/
/sys/kernel/livepatch/livepatch_sample
/sys/kernel/livepatch/livepatch_sample/enabled
/sys/kernel/livepatch/livepatch_sample/vmlinux
/sys/kernel/livepatch/livepatch_sample/vmlinux/cmdline_proc_show
# cat /sys/kernel/livepatch/livepatch_sample/enabled        # livepatch_sampleが有効であるとわかる
1
# echo 0 > /sys/kernel/livepatch/livepatch_sample/enabled   # livepatch_sampleを無効にする
# cat /proc/cmdline                                         # カーネル引数が復活
BOOT_IMAGE=/boot/vmlinuz-4.1.1-gentoo-r1 root=UUID=0bb07f44-2cbe-402c-84c9-253e3d1d7ce6 ro init=/usr/lib/systemd/systemd zswap.enabled=1
# echo 1 > /sys/kernel/livepatch/livepatch_sample/enabled   # 再びlivepatch_sampleを有効にする
# cat /proc/cmdline                                         # カーネル引数が隠される
this has been live patched
```

置き換え後の/proc/cmdlineの読み出し関数です。**リスト3**の本来の読み出し関数であるcmdline_proc_showと同じ引数をとり、問題なく差し替え可能なようにできています。

次の"struct klp_patch patch"は、どのオブジェクト(カーネル本体、またはモジュール)のどの関数をどの関数に置き換えるかを記述する構造体になります。1つのpatchで複数のオブジェクトの複数の関数を置き換えられるように、**図2**のような少し複雑な構造になっています。"struct klp_patch"のobjsは、対象オブジェクトの配列となっています。各要素は"struct klp_object"であり、その"name"要素がpatch対象のモジュール名を示し(空文字の場合はカーネル本体)、"funcs"が書き換え対象の関数の配列を示しています。末尾のエントリはfuncsがNULLとなっています。"struct klp_func"はそれぞれ"old_name"により置き換え前の関数の名前を、"new_func"により置き換え後の関数のアドレスを指定しています。末尾のエントリは"old_name"をNULLとすることで示されます。

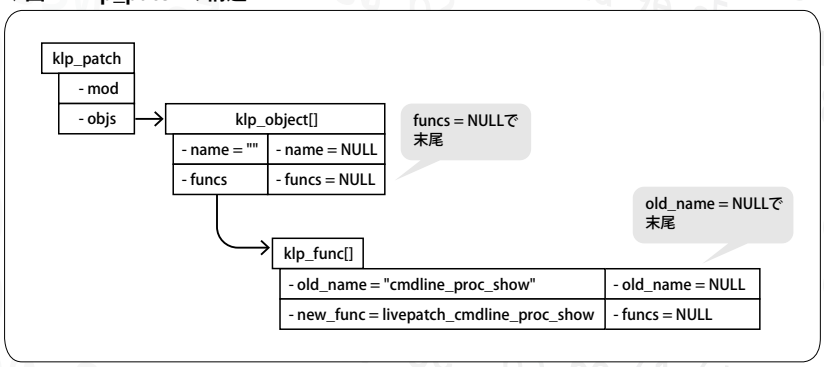
最後のlivepatch_initは、モジュールのロード時に実行される関数です。ここでlivepatchが登録(klp_register_patch)され、有効にされています(klp_enable_patch)。

livepatchにおけるpatchは、patchする対象の関数が何であるか、どのような関数に切り替えるのかのデータを持つカーネルモジュールとして作られています。

▼リスト3　本来の/proc/cmdline読み込み用関数

```
static int cmdline_proc_show(struct seq_file *m,
void *v)
{
    seq_printf(m, "%s\n", saved_command_line);
    return 0;
}
```

▼図2　klp_patchの構造



livepatch適用のしくみ

次にlivepatchによって古い関数を新しい関数に置き換えるしくみについて見ていきまょう。この部分にはkpatchもkGraftも、もちろんその共通基盤としてのlivepatchも、すでにカーネルに存在し長く使われているftraceという機能を利用しています。

ftraceとはFunction Traceの略で、Linuxカーネルの関数の動きをトレースするためのデバッグ用に用いられる機能です。この機能を使ってどの関数がいつ呼び出されたのかを調べることができます。

ftraceの実現にはコンパイラのプロファイル機能が使われています。コンパイラのプロファイル機能(-pg)を有効にすると関数呼び出しの前に関数"mcount"(または-mfentryをつけた場合は"__fentry__")の呼び出しが追加されます(**図3**)。本来はこの関数を使って、関数の呼び出し回数をカウントするといったプロファイル情報を収集します。Linuxカーネルでは、この5byteの領域を起動時にNOPに書き換えます。そして、ftraceを使うときに必要に応じてほかの関数の呼び出しに書き変え、トレースなどの機能を実現しています。

このftraceの機能を使って、livepatchは置き換え対象の関数の前に関数klp_ftrace_handlerの呼び出しを追加します(**図4**)。klp_ftrace_handlerは、ftrace

からの戻り先を新しい関数のアドレスに書き換えることで、関数の置き換えを実現します。

livepatchが危険な場合

さて、このlivepatchの機能を使うことで単純な関数の入れかえは実行することができますが、これだけでは危険な場合もあります。たとえば、**図5**のような関数f()を考えてみましょう。関数f()は、alloc()を呼び出し16byteの配列を確保し、その後、f()の中でその配列上のデータにアクセスします。

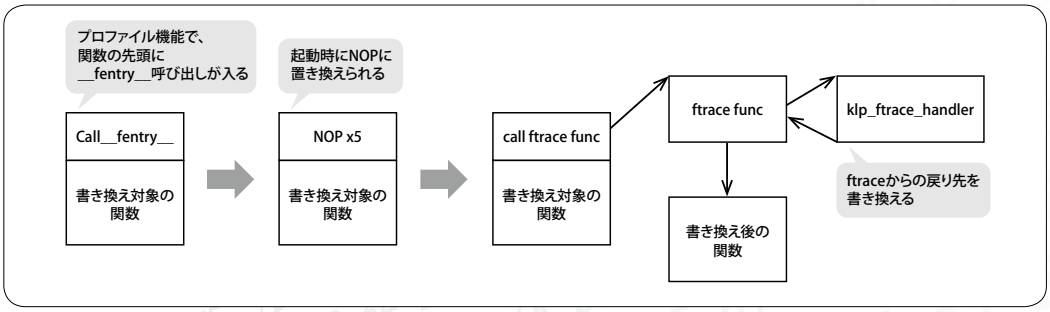
この関数が使う配列のサイズを16byteに変更するlivepatchには危険性があります。もしもf()を実行し始めて、alloc()を実行する前に、livepatchの適用により関数が入れかえられた場合、適用後のalloc()で確保する8byteの配列を、もとのf()が16byteの配列としてアクセスしてしまい、バッファオーバーフローが発生してしまいます。

このようなlivepatchを安全に適用するためには、patchの当て方を工夫し、livepatch適用前の関数と適用後の関数が同時に実行されないように保証する必要があります。このpatch適用方法の違いが、kpatchとkGraftの違いとなっています。

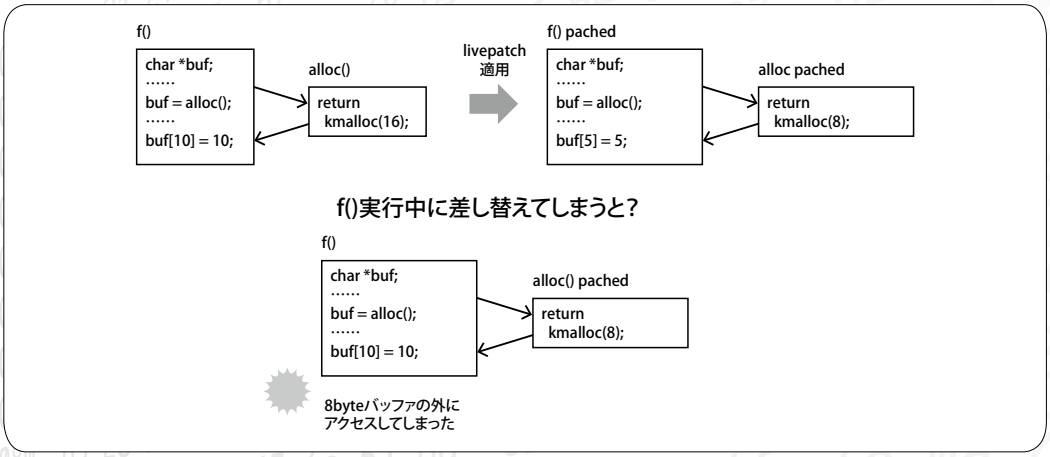
▼図3　プロファイルを有効にすると__fentry__が呼び出される

```
$ diff -u <(echo 'main(){}' | gcc -x c -S -o - -) <(echo
'main(){}' | gcc -x c -S -o - - -pg -mfentry)
--- /proc/self/fd/11    2015-07-20 18:43:43.353993325 +0900
+++ /proc/self/fd/12    2015-07-20 18:43:43.353993325 +0900
@@ -5,6 +5,7 @@
 main:
 .LFB0:
        .cfi_startproc
+       call    __fentry__
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
```

▼図4　livepatchの機能



▼図5　関数f()のlivepatchの危険性



kpatchのconsistency model

kpatchはlivepatchの適用にstop_machine()を使います。stop_machineは、すべての実行中のプロセスおよび割り込みを停止し、最高優先度で指定した関数を実行する機能です。

stop_machineから呼ばれるkpatchの関数であるkpatch_apply_patchは、ここですべてのプロセスのバックトレースを調べ、置き換え対象の関数が実行中でないことを調べます。どのプロセスのバックトレースにも置き換え対象となる関数がなければ、この時点で安全に関数を差し換えることが可能となります。

この方法はstop_machine()を使うことで「すべてのプロセスが停止し、割り込みも発生しない」という考えることの少ない状況を使えることから、比較的実装がシンプルとなります。その一方で、すべてのプロセスをpatch適用の間止めてしまうので、その間のサービスの停止につながるという問題点もあります。

kGraftのconsistency model

kGraftはkpatchに比べて、より複雑なpatch適用を行ないます。kpatchではシステム全体でatomicに古い関数から新しい関数への切り替えを行なっていましたが、kGraftはタスク単位で切り替えを行ないます(**図6**)。

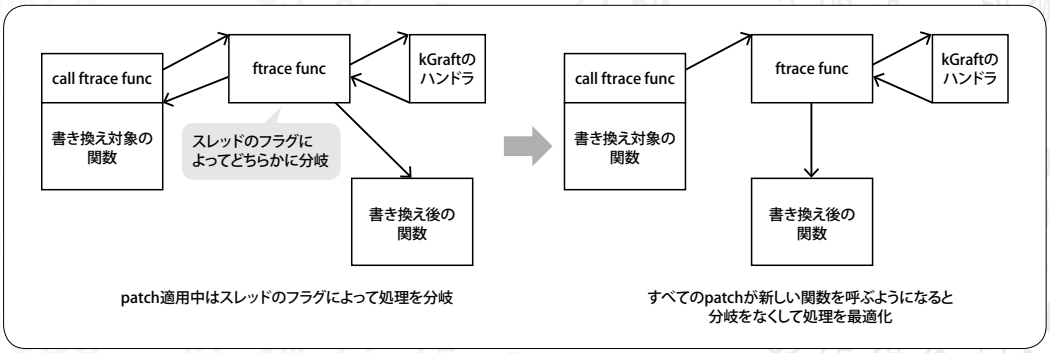
kGraftによるlivepatch適用を開始すると、システムの全スレッドにTIF_KGR_IN_PROGRESSというフラグが立ちます。そして、システムコールから抜けるタイミングやシグナルから返るタイミングなどで、TIF_KGR_IN_PROGRESSのフラグがクリアされます。

kpatchでは置換後の関数にジャンプするだけのシンプルな関数をftraceで実行していましたが、kGraftはTIF_KGR_IN_PROGRESSのフラグを見て呼び出す関数をpatch適用前の関数にするかpatch適用後の関数にするかを決定します。こうしてスレッドごとに順次patchが適用されていきます。すべてのスレッドがシステムコールなどでpatch適用後に切り替わると、ftraceによって呼び出される関数がkpatchと同様にシンプルにpatch適用後を呼び出すものに切り替えられ、patch適用中にはあったオーバーヘッドをなくしています。

patchの適用を完了するには、すべてのスレッドがシステムコールやシグナルから返ってくる必要があります。すなわちsleepしているスレッドがあると、そのスレッドが起きるまでpatchの適用が完了しないということになります。それでは困るので、kGraftはpatchの適用が完了していないスレッドに対してシグナルを送ってスレッドを起こします。

kGraftはkpatchに比べてstop_machine()を使わないため、プロセスが停止することがないというメリットがあります。その一方で、kpatchが独立したカーネルモジュールとして実装できているのに対して、kGraftではさまざまなカーネル本体への変更を必要としています。たとえば、スレッドの新しいフラグであるTIF_KGR_IN_

▼図6　kGraftの動作



PROGRESSや、カーネルスレッドがいつ安全にpatch後に切り替えられるかを示すためにカーネル本体のさまざまな場所に変更が必要となっています。

kpatchのuserland tool

最後にkpatchのuserland toolであるkpatch-buildを紹介します。前述したようにlivepatchはkernel moduleとして構成されています。しかし、実際にセキュリティパッチを当てたいというときには、patchからどのモジュールのどの関数をどのように書き換えるかを考えて、livepatchとして動作するカーネルモジュールを作成するというのは面倒なものです。kpatch-buildはこの作業を自動化し、patchを与えるとlivepatchとして動作するカーネルモジュールを作成してくれます。

では、**リスト4**のように/proc/cmdlineの出力に"`LIVE PATCHED: `"を追加するpatchからlive patch moduleを作ってみましょう。**図7**のようにkpatch-buildを使い"`-s`"でカーネルのソースコードを、次の引数にpatchのパスを指定します。kpatch-buildは指定されたpatchを用いて次のプロセスでlivepatch モジュールを作成します。

- 元のソースからカーネルをビルド
- patchを適用し、カーネルをリビルド
- objectファイルを比較し、差分を抽出
- 差分のobjectファイルをまとめと、livepatchモジュールを作成

差分を抽出するときに関数ごとに独立したセクションが作らるので、差分のobjectファイルは自然にリンクしてまとめることができます。実際、ここでモジュールのセクションを見てみると書き換えている関数である"`cmdline_proc_show`"に対応する"`.text.cmdline_proc_show`"というセクションができているのがわかります。なおビルドされたlivepatchモジュールはpatchのファイル名から自動的に名前が付けられます。

ビルドされたkpatch-patch-cmdline-overwrite.koを読み込むと、たしかに/proc/cmdlineがpatch適用された形に変わっていることがわかります。SD

▼リスト4　/proc/cmdlineへのpatch

```
--- a/fs/proc/cmdline.c 2015-07-21 10:50:52.293345753 +0900
+++ b/fs/proc/cmdline.c 2015-07-21 10:51:16.415734984 +0900
@@ -5,7 +5,7 @@

 static int cmdline_proc_show(struct seq_file *m, void *v)
 {
-       seq_printf(m, "%s\n", saved_command_line);
+       seq_printf(m, "LIVE PATCHED: %s\n", saved_command_line);
        return 0;
 }
```

▼図7　kpatch-buildによるlivepatchの作成と適用

```
$ sudo ./kpatch-build/kpatch-build -s /usr/src/linux-4.1.1-gentoo-r1 /tmp/patch-cmdline-overwrite
Using source directory at /usr/src/linux-4.1.1-gentoo-r1
Testing patch file
checking file fs/proc/cmdline.c
Building original kernel
Building patched kernel
Extracting new and modified ELF sections
cmdline.o: changed function: cmdline_proc_show
Patched objects: vmlinux
Building patch module: kpatch-patch-cmdline-overwrite.ko
SUCCESS
$ objdump -h kpatch-patch-cmdline-overwrite.ko|grep cmdline_proc_show
  8 .text.cmdline_proc_show 00000022  0000000000000000  0000000000000000  00000590  2**4
$ sudo insmod kpatch-patch-cmdline-overwrite.ko
$ dmesg|tail -n 1
[245544.727112] livepatch: enabling patch 'kpatch_patch_cmdline_overwrite'
$ cat /proc/cmdline
LIVE PATCHED: BOOT_IMAGE=/boot/vmlinuz-4.1.1-gentoo-r1 root=UUID=0bb07f44-2cbe-402c-84c9-
253e3d1d7ce6 ro init=/usr/lib/systemd/systemd zswap.enabled=1
```

September 2015

NO.47

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

公益のために……LibreOfficeのポータビリティへの挑戦

毎年、沖縄で行われるオープンソースカンファレンスにて研究会を開催しています。今回は沖縄在住でLibreOfficeなどの開発に携わっている安部さん（**写真1**）を講師にお迎えし、ソフトウェアのポータビリティに関する考察を語っていただきました。参加者は8人でした（**写真2**）。

jus研究会　沖縄大会

■古くて新しいLibreOfficeから見るポータビリティ

【講師】安部 武志（LibreOffice 日本語チーム）
【司会】法林 浩之（日本UNIXユーザ会）
【日時】2015年7月4日（土）15:15～16:00
【会場】沖縄コンベンションセンター
会議場B1（B棟2F）B会場

写真1　安部武志氏

写真2　研究会の様子

■ソフトウェアのポータビリティとは

はじめに、ソフトウェアのポータビリティには、可搬性、移植性、データの融通性という3つの意味合いがあることを示し、今回の発表ではその中からおもに移植性について話をするという前置きがありました。ソフトウェアの移植性は昔から重視されていて、1958年ぐらいにはすでにそれに関する論文が書かれていたようです。また、UNIXにとっても移植性は重要な特徴の1つであり、UNIXはCPUの種類を超えて移植可能な最初のOSでした。その結果、UNIXは多くの子孫を産み、現在に至るまで広く使われ続けています。

LibreOfficeのようなアプリケーションソフトウェアにとっても移植性は大事ですが、OSにとっての移植性がおもに対ハードウェアであるのに対して、アプリケーションの場合は対ハードウェア、対OS、対ミドルウェアなど多くの要素を考慮しなければならず、移植性を維持するのはより難しくなります。そのような厳しい条件下においてもLibreOfficeの移植性は高く、現在でもWindows XP/7/8、Mac OS X、Linux、FreeBSD/NetBSD/OpenBSD/DragonflyBSD、Androidなど幅広いアーキテクチャをサポートしています。さらにiOSもサポートしようとしていたり、

WebブラウザからLibreOfficeを使えるようにするしくみも開発中です。

■LibreOfficeの移植性に対する考え方

ここで、LibreOfficeの移植性に関する考え方がいくつか示されました。1つめは、モダンなC++コンパイラを使うことと、本当に必要なときだけクロスコンパイルすることです。ビルドスクリプトはGNU Makeベースで書かれており、bashなどのシェルが動く環境であればどこでも動作します。2つめは拡張機能の本体への取り込みを積極的に行っていて、その際にJavaではなくC++を使用していることです。Javaは移植性が高いように思われていますが、実際にはJavaのバージョンやOSベンダの数だけ実行環境のバリエーションが増えるので、移植性の維持という観点ではかえって面倒です。3つめは、LibreOfficeではバンドルしているライブラリが山のようにあるので、依存するライブラリもできるだけポータブルなものを選ぶようにすることです。

しかし、このような基本だけでは通用しないのがモバイル環境への移植です。たとえばAndroidではdynamic linkerが特殊だったり、アプリケーションの実行ファイルを50MBまでに抑えなければならないなどの制約があったりします。また、Touch対応、Tiled rendering、編集用のUIコンポーネントなど、モバイル環境ならではの移植ポイントも多々あります。Firefox for Androidという先行事例があるので、それを参考にしながら開発が進められていますが、ブラウザは基本的に閲覧機能だけで良いのに対して、LibreOfficeでは編集機能が必要なので、そこは自前で開発せざるを得ないようです。iOS版ではさらに厳しい制約があり、開発の難易度も高いとのことです。

■クロスプラットフォームであることの意義

さらに、LibreOfficeがクロスプラットフォームであることの意義について話がありました。開発者にとっての移植性は、次世代の開発者が新しい環境へ移植する手間をできるだけ少なくすることで開発効率を高めるためのものです。しかし、ユーザにとっては移植性の帰結であるクロスプラットフォーム性、すなわち多くの環境で同じソフトウェアが動き、同じデータを使い回せるというデータの融通性が重要です。

LibreOfficeにおけるクロスプラットフォームの意義は、従来のデスクトップ環境を所持していない人、たとえばパソコンがなくスマホしか持っていない人などにも使ってもらえる可能性を生み出す点にあります。とくにオフィススイートは公益性が大事で、たとえばイギリスでは内閣府が共有文書のフォーマットとしてODFを採用したり、スペインでは地域によって異なる公用語が使われているのをLibreOfficeでは両方サポートしている（**図1**）のは、公益性を維持するための努力と言えます。

最後に、移植性は昔も今も重要であること、LibreOfficeでは移植のノウハウは蓄積されているが今後も挑戦が続くこと、LibreOfficeがポータブルであることの意義は、ソフトウェアの開発効率向上だけでなく、公益への寄与という点でも重要であることを述べて講演を締めくくりました。

実際に開発者として活動している安部さんならではの実感のこもった、内容の濃い発表でした。SD

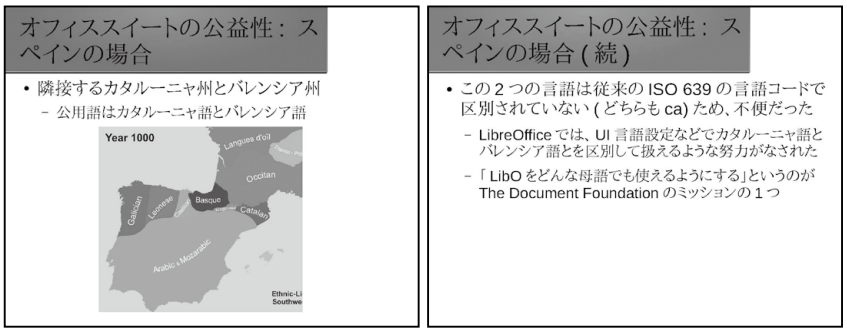


図1　発表資料「古くて新しいLibreOfficeから見るポータビリティ」より[注1]

注1）http://fixedpoint.jp/2015/07/04/

Hack For Japan
エンジニアだからこそできる復興への一歩

●Hack For Japanスタッフ
佐伯 幸治　SAEKI koji
Twitter @widesilverz

第45回 防災・減災とIT ～SNSとメディアの取り組み編～

Hack For Japanの活動においては、復興支援に加え、前号で取り上げた「国連防災世界会議」のような「防災・減災」をテーマにする機会も増えてきました。そこで今回の連載では「防災・減災とIT」をテーマにSNSとメディアにおけるトピックを取り上げて紹介します。

SNSにおける取り組み

東日本大震災以降、災害時におけるTwitterやFacebookといったSNSの有効性は多くの方が実感されているでしょう。この流れを受けて、各SNSは災害時に利用できる機能を実装する動きが見られます。またユーザ自身がSNSの機能を活かして災害に備えたり、復旧活動につなげるという利用方法も見受けられるようになりました。著者が把握している動きを以下にまとめてご紹介します。

Facebook

Facebookでは2014年10月に「災害時情報センター[注1]」を新機能として発表しました。この機能は「Facebook上でつながっている友達や家族に自分が無事であることを知らせる」、「災害の影響を受けた地域にいる人の安否を確認する」、「Facebook上でつながっている友達の無事を報告する」という3つがあり、最近ではネパール大震災で利用されました[注2]。紙幅の都合で、詳しくはFacebookの紹介ページをご覧いただきたいのですが、この機能は東日本大震災をきっかけにして生まれたものです。

また、Facebookは2015年3月に「Facebookを活用した災害対策と対応[注3]（**写真1**）」というガイドを発表しています。ガイドは「災害時対応と災害支援を担う組織のためのヒント」、「救助隊と行政機関のためのヒント」、「個人とコミュニティのためのヒント」の3つで構成されている内容で、それぞれの立場から投稿や写真・動画の利用目的がどういったものであるべきかが述べられており、「Facebookを災害時にどのように使うか」の指針として参考になる情報が盛り込まれています。

Twitter

Twitterでは2013年9月に新機能としてTwitterアラート[注4]を開始しました。この機能は、信頼性のある機関・団体からアラートのマークを付けて出された重要なツイートを、スマートフォンのプッシュ通知で受け取ることができるものです。

ここで定められている“信頼性のある機関・団体”とは、一般市民に向けて緊急情報を提供する国、地方、および国際機関としています。具体的には「警察および安全対策機関」、「緊急通報受理機関」、「地

注1　災害時情報センターのご案内
http://ja.newsroom.fb.com/news/2014/10/safety check/
注2　https://www.facebook.com/safetycheck/nepalearthquake
注3　http://ja.newsroom.fb.com/news/2015/03/facebook_disaster_response/
注4　https://about.twitter.com/ja/products/alerts

◆写真1　「Facebookを活用した災害対策と対応」表紙

方自治体およびその付属機関」、「地方自治体の業務を担う郡または地区の行政サービス機関」、「特定の政府官公庁、民間非営利団体」と限定され、原稿執筆時点（2015年6月）では104機関・団体のアカウントがTwitterアラートに登録されており、「警視庁警備部災害対策課」、「東京消防庁」、「新潟県長岡市危機管理防災本部」など、さまざまな機関・団体が確認できます。興味のある方は一度リストをご覧になって、気になる機関・団体をフォローしておくことをお勧めします[注5]。

このTwitterアラートは、Twitterで情報提供をする側にもメリットがあります。想定されるのは各機関・団体のTwitter運用担当者が被災したケースで、災害時、担当者は必ずしもオフィスにあるパソコンを使ってツイートできる場所にいるとは限りません。そのためTwitterアラートでは、パソコンと携帯電話・スマートフォンからの緊急情報を発信可能としており、また専用のツイート作成ボックスを使用してアラートツイートを作成、重要なツイートとしてタグ付けすることができます。

そもそもTwitterは、ほかのSNSに比べて東日本大震災時から多くのユーザに利用されていた経緯から、都道府県・市区町村といった自治体でのアカウント導入が進んでおり、平時から生活やイベント情報と同様の扱いで、大雨の際の注意を促すなど災害情報を市民へ向けて発信しているアカウントも見られます。Hack For Japanスタッフが参加して立ち上げた「減災インフォ（詳しくはコラム参照）」のTwitterアカウントでは、災害情報を発信している自治体のアカウントを活用しやすいように都道府県別にリスト化しています[注6]。

こうしたTwitterアラートやTwitterアカウントリストの利用以外にも、ハッシュタグやGPS情報を組み合わせた利用方法も検討されています。たとえば埼玉県和光市では、2014年2月に和光市の災害に関するハッシュタグを「#和光市災害」と定めることを公式発表し、防災訓練にてハッシュタグを活用

注5　https://about.twitter.com/ja/products/alerts/participating-organizations
注6　https://twitter.com/gensaiinfo/lists

Column 「減災インフォ」について

Hack For Japanスタッフがかかわっている「減災インフォ」というプロジェクトが始まっています。6月にはサイトをオープンしており、減災に関するさまざまな情報を発信していく予定です（**図A**）。現在サイトでは「ブログ」、「自治体と地域メディア」、「災害復興ボランティア」、「災害の種類とこれまで」、「勉強会」、「減災友達の輪☆」といったコンテンツを用意しています。ブログでは「災害時の情報まとめ（口永良部島噴火情報など）」、「イベント関連レポート（Twitter勉強会レポートなど）」、「メディアやSNSに関する減災情報まとめ（市町村のTwitter導入状況など）」といった記事を公開しています。

イベントの1つ、Twitter勉強会ではTwitter社の協力を得て災害時におけるTwitterの使い方を学ぶ場を開催したり、減災情報のまとめでは自治体のTwitterアカウント導入状況をまとめるといった活動を行いました。今後もイベントなどを考えていますので、興味ある方はご参加ください。

「自治体と地域メディア」では、信頼できる情報を収集して減災につなげていくため、都道府県別・市区町村別に信頼性のある情報発信元を調べてリンクで一覧にしています。お住まいの地域でどのような情報サービスが提供されているか、入手可能かを調べておくことができます。ぜひお住まいの地域について確認してみてください。

減災インフォはTwitterでも随時、情報発信しています。アカウントは@gensaiinfoです。皆さんのフォローお待ちしています！

◆図A 「減災インフォ」Webサイト

http://www.gensaiinfo.com/

する取り組みを始めています[注7]。この和光市の取り組みにならって市区町村レベルで災害時ハッシュタグの活用が広がりを見せており、「#宇部市災害」、「#北本市災害」、「#狭山市災害」、「#清瀬市災害」、「#金沢区災害」、「#御嵩町災害」といったハッシュタグが導入されています。

公的機関以外では、群馬県建設協会アカウントのTwitterでの発信が一例として挙げられます。『2014年2月の記録的な大雪の際、除雪の様子をツイッターで発信したところ大きな反響があったことから、一般向けの広報活動に生かせると判断。同協会の「災害情報共有システム」を刷新し、現場からの情報をソーシャル・ネットワーキング・サービス（SNS）で簡単に発信できる機能を追加した。（引用元：日経コンストラクション）』とあるように、群馬県建設協会は、平時からパトロールなどの状況をGPS情報付きで発信、災害時への備えとしています[注8]（**図1**）。

LINE & Instagram

現状、筆者の周囲での防災・減災情報のSNS活用としては「Twitterで情報発信・収集→Facebookグループでコメントをやりとりして情報を深掘りする」といった使い方が多いように思われ、LINEとInstagramについては、これから活用されていく印象を持っています。しかしながら平時からLINEは多くの方が利用しており、また海外ではInstagramの利用が多いということを考慮すると、これらも軽視できません。

LINEについては、オープンなパブリックアカウントとして運用可能なLINE@の自治体での導入が見受けられます。一例として挙げると埼玉県、滋賀県大津市危機・防災対策課、大阪府柏原市などがアカウントを取得しており、福岡県大野城市による「市民のみなさんへ、防災訓練をお知らせする連絡網」として活用されている事例がLINE@のブログで読むことができます[注9]。

Instagramについては国内での活用事例は目立っていないのですが、海外ではネパール大震災の際に、ネパールやインドの写真家たちが立ち上げた「ネパールフォトプロジェクト」がハフィントンポストやwiredで取り上げられており、原稿執筆時点で約6万人のフォロワーを集め継続的に投稿されています[注10]。また補足ですが、ネパール大震災ではドローンによる被災状況を空撮した動画がSNS経由で拡散されていったことも注目に値します。ドローンによる取り組みは日経新聞でも取り上げられ、今後は利用規制との折り合いをどのようにつけるかが課題として挙げられます[注11]。

注7　災害時におけるツイッターハッシュタグの利用について
http://www.city.wako.lg.jp/home/kurashi/bousai/bousaitaisaku/_13853.html
注8　http://www.gun-ken.or.jp/anshin.html
https://twitter.com/gunken000

◆図1　群馬県建設協会アカウントのツイート

注9　「防災訓練のお知らせ」福岡県『大野城市』の市民を守るメッセージ
http://blog.lineat.jp/archives/25032901.html
注10　HELPING NEPAL'S QUAKE SURVIVORS—WITH INSTAGRAM
http://www.wired.com/2015/05/tara-bedi-sumit-dayal-nepal-photo-project/
ネパール大地震でSNSを使った復興プロジェクト「写真や動画が災害時に機能するかどうかの試み」
http://www.huffingtonpost.jp/2015/05/13/nepal-instagram-plays-vital-role-_n_7280128.html
注11　Drone Films Bird's-Eye View of Nepal Quake Devastation　NBC News
https://www.facebook.com/NBCNews/videos/10707981796068 78/
ドローン、ネパールに集結 地震被災状況を空撮で把握、対応力欠く国を世界が支援
http://www.nikkei.com/article/DGKKZO86898300W5A510C1TZN000/

メディアにおける取り組み

メディアでは、NHKのITを利用した防災・減災への取り組みが目立っているようです。大きなトピックとしては3つ挙げられます。

1つめは5月に起きた口永良部島噴火でのネットとテレビの同時配信の試みです。NHK総合テレビで放送中の番組がNHKオンラインで同時配信されるというもので、テレビのない状況でもインターネットにつながってさえいれば、現地の避難状況を見ることができました[注12]。なお原稿執筆時点において、箱根山の噴火警戒レベルが2（火口周辺規制）から3（入山規制）へ引き上げられたことから、NHKはライブ映像を公開しリアルタイムでの状況を伝えています（**図2**）。

2つめは『データジャーナリズムの新番組「データなび」と、NHKが制作した報道のWebサイトを紹介するポータルサイト。ビッグデータの解析などでつかんだ新たな発見を分かりやすく伝えます』（番組紹介より）とした「データなび[注13]」サイトを4月にオープンしたことです。このサイトではテーマとして災害が設けられ、口永良部島噴火、御嶽山噴火、震災ビッグデータ、原発事故といった切り口でデータを元にしたコンテンツが見られます。

3つめとしてNHKは「ソーシャル・リスニング・チーム（Social Listening Team = SoLT：ソルト）」というプロジェクトを2013年に発足させています。これは複数人でTwitterのタイムラインを観測して、事故やトレンドの端緒を迅速につかみ報道につなげることを目的とした取り組みとして始まりました。現在も試行錯誤している段階のようですが、災害時での活用も想定されています[注14]。NHKの災害情報についてはTwitterアカウント「NHK生活・防災（@nhk_seikatsu）」で発信されています。

こうしたNHKの試みをきっかけにした民放各局、新聞、ネットメディアによる災害対応やそれぞれの連携も今後、期待されるところです。

◆ ◆ ◆

以上、「防災・減災とIT」として今回はSNSとメディアでの取り組みをご紹介しました。これら以外にもYahoo!防災速報[注15]や自治体の災害対応アプリ[注16]、JAXAによる標高データの無償公開[注17]、災害対応のための自治体サイトの改善[注18]など（技術評論社からも『［オープンデータ＋QGIS］統計・防災・環境情報がひと目でわかる地図の作り方[注19]』が発売されています！）、さまざまな分野で多様な取り組みが見られますので、またの機会に紹介したいと思います。SD

注12 NHK、テレビ放送をネット同時配信　口永良部島の噴火受け
http://www.itmedia.co.jp/news/articles/1505/29/news082.html
注13 **http://www.nhk.or.jp/d-navi/**
注14 震災ビッグデータからソーシャルリスニングへ
http://www.nhk.or.jp/bunken/book/regular/media/media11/2_06.pdf
NHK報道局が実践している「ソーシャル・リスニング・チーム」とは？
http://www.gensaiinfo.com/blog/2015/0710/2076/
注15 Yahoo!防災速報、「土砂災害警戒情報」「指定河川洪水予報」の提供を開始
http://news.mynavi.jp/news/2015/06/19/277/
注16 防災アプリ：好評　行政オープンデータ活用　埼玉・北本市
http://mainichi.jp/select/news/20150526k0000e040203000c.html
注17 世界最高水準の全世界標高データ（30m版）の無償公開について
http://www.jaxa.jp/press/2015/05/20150518_daichi_j.html
注18 災害対策＆セキュリティを強化、茨城県公式ホームページがリニューアル
http://www.rbbtoday.com/article/2015/04/03/130117.html
ホームページ一新 災害情報見やすく スマホ対応も 名張市
http://www.iga-younet.co.jp/news1/2015/04/post-877.html
注19 **http://gihyo.jp/book/2014/978-4-7741-6913-2**

◆図2　NHKによる箱根山ライブ映像

温故知新
IT むかしばなし

第46回

初代PC-9801のライン描画速度に魅せられて

速水 祐(HAYAMI You)　http://zob.club/　Twitter : @yyhayami

はじめに

1982年10月、IBM-PCに遅れること約1年。我が国の標準となる16bitパソコン「NEC PC-9801」が登場しました。その10年後の1992年には、筆者は「PC-9801スーパーテクニック」[注1]の一部を執筆しており、PC-9801の内部を解析することで、その特徴を活かしたソフトウェアを書いていました。ちょうど初代PC-9801が手元にありますので、今回はそれを動かしながら話を進めましょう。

初代PC-9801の3つの強み

1982年は、富士通のFM-7(6809 2MHz)、シャープのMZ-2000、X-1(Z80 4MHz)のような高性能な第2世代の8bitパソコンが登場してきており、NECのPC-8801の動作の遅さが目立ち始めたときでした。

そこで登場したのがPC-9801です。このマシンには、ほかのパソコンを超える次の3つの強みがありました。

注1)『PC-9801スーパーテクニック』、小高輝真、清水和文、速水祐共著(1992)、アスキー出版

❶16bit Intel 8086による高速性と広大なメモリ空間
❷グラフィックディスプレイコントローラ(GDC)を使用した高速な描画
❸漢字テキストVRAM

さらにNECが独自に開発したN88-BASIC(86)搭載によって、8bitマシンであるPC-8801のN-BASICとの互換性を保ち、過去のPC-8801のBASICソフトウェアが動作可能だったのです。

❶Intel 8086

IBM-PCは、データバスが8bitのIntel 8088を採用していましたが、PC-9801はデータバスが倍の16bit Intel 8086(i8086)を採用していました。動作周波数は、IBM-PCが4.77272MHzだったのに対し、PC-9801は4.9152MHz(約5MHz)でした。バス幅の増加とクロックの上昇により、高速化をかなり期待したのですが、第2世代の8bitパソコンは最適化が進んでいたため、実測してみると数十%程度の速度アップにすぎませんでした。

i8086の強みは、むしろメモリ空間の広大さです。標準的な8bit CPUの64KBに対してなんと16倍の1,024KBとなり、入り江から大海に放り出されたような大きさでした。

その後、筆者はこの感覚を二度感じています。1988年ごろの80386の32bitモードにおける1MB→4GB(約4,000倍)、そして最近のx86アーキテクチャの64bit化であるx86-64 4GB→16EB(約43億倍)です。最後の変化は、宇宙に放り出されたように倍率は大きく進歩していますが、最初の16倍の方がうれしく感じた記憶があります。

メモリマップを**図1**に示します。標準実装メモリは、128KBと全体の一部でしたが、64KBがシステムにほとんど占領されてわずかなフリーエリアを使っていた8bitマシンに比べると、かなり自由に使えるメモリエリアだと思いました。この後すぐに640KBのエリアも小さく思えてくるのですが……。

❷高速な描画を実現するGDC

i8086の実行スピードについては，少し期待外れでしたが、描画スピードに賭けるものがあ

りました。PC-9801には、NECの開発した描画機能を持つGDC（Graphics Display Controller）μPD7220が2つ搭載され、それぞれテキスト表示管理とグラフィック描画管理を行います。

それまでのパソコンでは、ライン描画が非常に遅く、当時ライン描画を含むゲームを作成していたとき、最適化したマシン語プログラムでも望むスピードが出ず、製作を断念したことがありました。ですので、GDCを使ったライン描画速度には興味があり実験してみました。

当時の最新の8bitパソコンだったMZ-2000（Z80 4MHz）で151秒かかったプログラムが、PC-9801では、なんと10秒！

圧倒的な速さでした。CPUの速度が数割しかアップしていないのに対して、約15倍であり、大きな速度変化を体感しました。これはBASICで作成したプログラムでもゲームとして利用できるレベルで、現在、実際に実行しても速度と描画の滑らかさがわかります。

GDCにはライン描画のほか、円／円弧／四辺形／グラフィック文字描画などの機能があり、グラフィックBIOSではこれらの機能がサポートされていました。当初のGDCは、グラフィックVRAMのプレーンごとの描画でしたがPC-9801VM以降に搭載されたGRCG（グラフィックチャージャ）による複数プレーン同時描画や2.5MHzから5MHzにクロックアップされ、描画スピードはさらに上がりました。

❸漢字テキストVRAM

漢字コードを漢字テキストVRAMに書き込むだけで描画が可能で、スクロールのスピードなども非常に高速でした[注2]。そして、この機能を使った実用的な日本語ワープロの、管理工学研究所の「松」やジャストシステムの「一太郎」が登場することになります。後にFreeBSDを日本語化する際にも、AT互換機に比べて設定が簡単で高速な表示／スクロールを実現できていました。

N88-BASIC(86)（互換性）

PC-8001、PC-8801と続くNECのパソコンのBASICは、Microsoft製だったのですが、PC-9801のN88-BASIC(86)は、NEC製なのです。その製作の過程は、互換性の確保と時間との壮絶な戦いだったようです[注3]。そのあとのPC-9801進歩は、その互換性を維持した状態での機能の拡大でした。ほかのコンピュータやゲーム機もこのように互換性を確保した機能拡張に成功したマシンがメジャーになっていったようです。

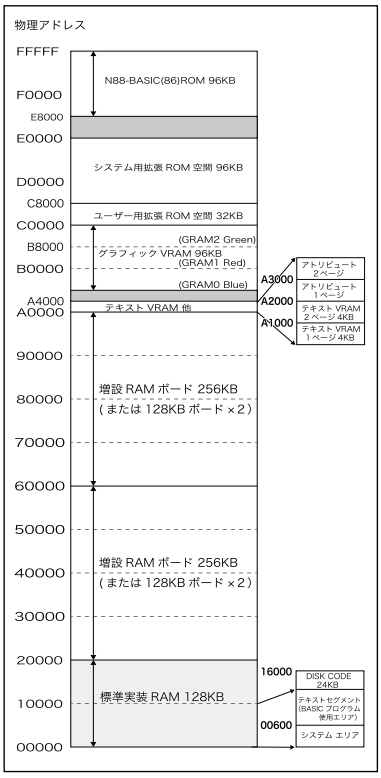
おわりに

Windowsの登場は、PC-9801の強みを覆すものでした。

Windowsでは、単機能なGDCによるグラフィック描画は使えないものであり、漢字テキストVRAMもインストール以外に使用することはなく、Windowsの互換性があれば、PC-9801の互換性は必要なくなりました。まさに強みが逆に作用して、21世紀になるとPC-98の終焉を迎えることになりました。

しかし、今、あらためてPC-9801を動かしてみると、エディタ、ワープロや単純なゲームなどは、現在のレベルで操作しても実用的に使用できます。レトロPCブームも盛り上がりつつありますが、みなさんも家で眠っているPC-98を動作させると、また新たな温故知新になるかもしれませんね。SD

▼図1　PC-9801のメモリマップ



注2） 初代PC-9801では、第1水準漢字ROMは、オプションで、専用のスロットに設置することで漢字が使用できるようになっていました。

注3） 『パソコン創世記』、富田倫生著（1994）、TBSブリタニカ

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

グレープシティ、「ActiveReports 9.0J Server」をリリース開始

グレープシティ㈱は7月16日、.NET 帳票開発コンポーネント「ActiveReports for .NET 9.0J Professionalエディション」の新機能として、帳票の管理運用を行うサーバ製品「ActiveReports 9.0J Server」をリリースした。

ActiveReportsは外観デザインの設定からデータ接続、印刷・PDFへの出力設定まで、あらゆる機能を備えた帳票開発コンポーネント。今回そのProfessionalエディションに追加されたActiveReports Serverは、ActiveReportsで作成した帳票の運用管理に特化したサーバ製品である。製品をWebサーバ(IIS)にインストールするだけで帳票運用環境を構築でき、使いやすいポータル画面で帳票のアップロード、プレビュー、印刷、スケジュール配信などの機能が利用できる。このActiveReports Serverの運用には、基本サーバライセンス(ActiveReports for .NET Professionalエディションユーザは無料で入手可能)とエージェントがインストールされたサーバのコア数に応じたコアサーバライセンスが必要となる。

●ActiveReports for .NET 9.0J Professional ライセンス価格

ライセンス種別	ライセンス数	価格(税込)
開発ライセンス	1ライセンス	302,400円
コアサーバライセンス	2コアライセンス	120,960円

CONTACT
グレープシティ㈱ URL http://www.grapecity.com

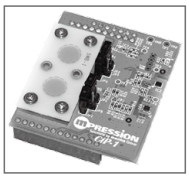
Hardware

マクニカ、IoT/M2Mセンサシールド「Mpression CiP-1」を発売

㈱マクニカは7月8日、非接触ひずみセンサ搭載のIoT/M2M開発ソリューション「Mpression CiP-1」を発売した。

Mpression CiP-1は、搭載されているインダクティブセンサが対象金属までの距離をサブミクロン単位で計測することで、非接触のひずみセンサを実現している。インダクティブセンサによる金属のひずみ検出や屋外・屋内でのねじ抜け情報などの検出を行いながら、周囲の照度・湿度・温度・ターゲット製品の熱源温度を計測できる。

同製品はテキサス・インスツルメンツ社の各種センサを搭載し、同社の開発評価キット「CC3200MOD LaunchPad」に接続することでWi-Fiを使用して、センサが取得した情報をモニタ確認できる。現在マクニカでは、これをゲートウェイやMobile Wi-Fiルータなどに接続し、AWSにセンサデータを転送する試験を実施している。今後は、同社子会社のマクニカネットワークスが取り扱うマシンデータ分析プラットフォーム「Splunk Enterprise」にアクセスすることで、蓄積データの表示・解析・分析を行うトータルソリューションを計画している。


▲Mpression CiP-1

CONTACT
㈱マクニカ URL http://www.macnica.co.jp

Service

クリエーションライン、Docker社公認トレーニングの提供を開始

クリエーションライン㈱は、米Docker社公認のトレーニングサービスの提供を開始する。

同社は今回、日本で初めてDocker社よりDocker Authorized Training Partnerとして認定を受けた。トレーニングはDocker社の提供する教材を日本語へローカライズし、日本で唯一のDocker Certified Trainer(Docker社公認トレーナー)が講師を務め、2日間のハンズオン形式として2015年8月末から実施される予定。トレーニング受講者にはDocker社公認の受講修了書が付与される。本トレーニングサービスは、同様の内容をユーザの希望の日程・場所で開催する個別対応サービスとしても提供していく用意があるとのこと。

同社は、Docker Authorized Consulting Partnerとしても認定を受けており、今後はコンサルティングパートナーとして、Docker社の各プロダクトの導入・設計支援サービスとともに、各社の提供するプロダクトとDockerとの連携検証支援も提供していく。

●トレーニング概要

学習形態	講習会(集合教育)、ハンズオン
開催言語	日本語
受講料	120,000円(税別)
期間	2日
初回実施日	2015年8月24日(月)、25日(火)

CONTACT
クリエーションライン㈱ URL http://www.creationline.com

Software

BlueMeme、「OutSystems Platform 9 Amsterdam」について発表

㈱BlueMemeは7月14日、同社が国内総代理店として提供中の大企業向けの高速開発ツール「OutSystems Platform」の新バージョン「OutSystems Platform 9 Amsterdam」について、報道者向けの発表会を行った。

「OutSystems Platform」は、モバイル／Webアプリの構築、展開、管理が簡単に実現できる、アプリケーションプラットフォーム。本製品の特徴は次のとおり。

・文字ベースのコード記述からの脱却

部品同士をGUIで組み合わせるように、グラフィカルな言語でアプリケーションのロジックを記述するので、JavaやC#の技術的な知識を身につけることなく、正しく動作する業務アプリケーションを作成できる

・サーバサイド型のモデル駆動型開発基盤

「Service studio」と呼ばれるクライアントツールでモデルデータを生成し、それがサーバに送信されることによってサーバサイドでソースコードが生成される。ソースコードの品質と安全性を確保しながら、集中型の構成管理を実現する

・ソースコードを管理しない構成管理

ソースコードではなくモデルデータをほかの環境へ移送するので、異なる環境間でのソフトウェア移送が可能。OSのバージョンなどのサーバの環境に合わせて、モデルデータからソースコードを自動生成する

・非機能要件の自動実装

インターフェース、セキュリティ対策、ユーザ認証といった非機能要件に該当する機能は自動実装される

BlueMemeの代表取締役である松岡真功氏は、「現状のスクラッチ開発はさまざまなフレームワーク・ライブラリを寄せ集めたパッチワーク･アーキテクチャであり、数年で機敏性を失い、リプレースの必要に迫られる。本製品はWebアプリ開発において必要な機能の多くを統合することでパッチワークアーキテクチャの解消を実現できる」と語った。

CONTACT
㈱BlueMeme URL http://www.bluememe.jp

Service

日本アイ・ビー・エム、日本での「IBM Watson」の展開について発表

日本アイ・ビー・エムは7月30日、IBM社が展開中のコグニティブ・コンピューティング・システム「IBM Watson」（以下、Watson）について、日本での事業展開に関する発表を行った。

Watsonはもともと、クイズ番組に出場して優勝することで「コンピュータが人間のように振る舞えるか」を証明するプロジェクトのために開発された。現在では、「自然言語を理解し、人間の意思決定を支援する」という機能を活かし、薬学や司法など幅広い分野で利用されている。

今回の発表では、Watsonの日本における本格的な事業展開が進んでいることが発表され、日本での2つの事例も紹介された。

○東京大学医科学研究所、がん研究に採用

東京大学医科学研究所（以下、東大医科研）は、「Watson Genomic Analytics」を活用して新たながん研究を開始している。特定された遺伝子変異情報を医学論文や遺伝子関連のデータベースなど、構造化・非構造化データとして存在する膨大ながん治療法の知識体系と照らし合わせることで、それぞれのがんに合った治療を提供することが可能になる。

○ソフトバンク、共同でエコシステムを整備

ソフトバンク㈱は、日本アイ・ビー・エムと共同で、各業界でWatsonを活用した新しいビジネスアイデアを展開するためのエコシステムプログラムを構築･提供し、2015年10月1日から「ビジネスパートナー」「テクノロジパートナー」の正式募集を開始する。これにより、起業家や開発者が日本において新しいWatsonのアプリケーションを開発し、採用することが可能になる。

発表会では、「ヤマダ電機にて、Watsonと接続されたロボット『Pepper』が、客と会話をしながら4Kテレビを買ってもらうためのセールストークを行う」ビデオが流された。流暢な受け答えからは、日本語へのローカライゼーションの進捗が好調であることが窺われた。

CONTACT
日本アイ・ビー・エム㈱ URL http://www.ibm.com/jp/ja

Hardware

ネットワールド、オランダRedSocks社とディストリビュータ契約を締結「RedSocks Malware Threat Defender」を販売開始

㈱ネットワールドは、オランダRedSocks社と日本で初めてディストリビュータ契約を締結し、次世代標的型攻撃対策製品「RedSocks Malware Threat Defender」を、7月16日より販売開始した。

「RedSocks Malware Threat Defender」は、すべてのアウトバウンド・トラフィック（企業のコンピュータシステムから外に出る通信）を監視し、標的型攻撃による情報漏洩の危機となる通信を、リアルタイムに、検知・通知できるアプライアンス製品である。

各端末に潜み、気がつかれないままデータを盗み続ける「マルウェア」は外部のC&Cサーバへ情報を送信する。この通信をリアルタイムでモニターし、C&Cサーバの情報と照らし合わせ感染を探し出すことができるのが本製品の特徴である。最近のマルウェアはセキュリティソフトのサンドボックス内を通過するようになっており、アウトバンドトラフィック分析に特化した同製品ならばその欠点を補える。しくみとしては、インターネットへの出口であるファイアウォールやルータなどのミラーポートに接続された同製品が、NetFlow/IPFIX（シスコシステムズ開発の、IPトラフィック情報を収集するためのネットワーク・プロトコル）により、すべてのアウトバウンドのパケットの中から必要なフロー情報（送信先のIPアドレスやURL、送信元IPアドレス、MACアドレス、ポート番号、プロトコル）を抽出して保有。この情報を、RedSocks社のセキュリティエキスパートチーム「The Malware Intelligent Team」から30分に1回の頻度で送られてくるC&Cサーバの情報と照合して、マルウェアによるC&Cサーバへの通信をリアルタイムに検知し、管理者に即時に通知する、というものだ。

価格は、150Mbpsまでの場合はアプライアンス＋初年度サブスクリプションで4,230,000円（税別）、次年度サブスクリプションで1,395,000円（税別）。250Mbps、1Gbps、10Gbpsについては、問い合わせの必要がある。

▲RedSocks Malware Threat Defender

CONTACT
㈱ネットワールド　URL http://www.networld.co.jp

Hardware

セキュアソフト、ネットワークセキュリティ対策製品「SecureSoft Sniper ONE」を発売

㈱セキュアソフトは、ネットワークセキュリティ機能強化を図った新製品「SecureSoft Sniper ONE」の販売を8月3日に開始した。

「SecureSoft Sniper ONE」は、同社主力製品のIPS技術を基盤とし、従来それぞれ専用機として提供されていた高度な検知・防御機能（DDoS攻撃対策、VoIP対策、DHCP対策、DNS対策）および高度化するセキュリティ対策に必要な機能（Regular Expression機能、HTTPS機能、Rate Limit機能）などの多彩なオプションを1台に搭載しながら高性能化を実現した。ユーザはこれらの中から必要なオプションだけを選択でき、コスト削減につなげられる。また、導入後の機能追加も可能で、ビジネスの変化・拡大にあわせて対策を拡充することができる。

製品ラインナップは3種類。価格は6,500,000円（税別）からとなっている。

●オプション概要

Anti-DDoS	TCP SSS、UDP SSS機能により、サーバへの不要な通信負荷を軽減
Rate Limit	通信の識別により静的、動的な帯域制御が可能
Regular Expression	正規化表現式を用いた高度なシグネチャの搭載が可能。推奨シグネチャなどの提供を予定
HTTPS	暗号化通信HTTPSを高速に復号し、その他機能との連携が可能
DHCP	DHCPサーバの防御やDHCPサービスの停止を狙う攻撃に対しMACアドレスベースでの対策が可能
VoIP	SIPをベースにした通話での盗聴、なりすまし、迷惑電話といった不正を検知防御し、通話セッションの管理も可能
DNS	DNSシステムの停止を狙った攻撃に対し、DNSサーバを防御する機能を搭載

●製品ラインナップ

機種名	ONE 2000	ONE 4000	ONE40G
筐体			
スループット	1.5/3Gbps	2/4Gbps	20/40Gbps
監視ポート	2/4ポート		
インターフェース	10/100/1000Base-T or 1000Base-SX or LX		10GBase-SR or LR
パワーサプライ	AC（冗長構成標準搭載）		AC/DC（冗長構成標準搭載）

CONTACT
㈱セキュアソフト　URL https://www.securesoft.co.jp

Software

トレンドマイクロ、「ウイルスバスタークラウド 10」を発売

トレンドマイクロ㈱は、総合セキュリティソフト「ウイルスバスタークラウド10」を7月29日に発売した。

最新版の「ウイルスバスタークラウド 10」は、Windows向けの「ウイルスバスタークラウド」、Mac向け「ウイルスバスター for Mac」、スマートフォン／タブレット端末向けの「ウイルスバスターモバイル」を統合した製品。1つの製品で、パソコン、スマートフォン、タブレット端末（Windows、Mac、Android、iOS、Kindle Fireシリーズ）を最大3台まで保護できる。同時発売の「ウイルスバスタークラウド 10＋デジタルライフサポートプレミアム」はパソコン／スマートフォンやインターネットの接続トラブルなどの問い合わせに365日対応する追加サービスが付いた製品となっている。おもな新機能・強化ポイントは次のとおり。

▲ウイルスバスタークラウド 10

ウイルスバスター クラウド（Windows）

・Windows 10に対応
7月29日にリリースされた、Microsoftの最新OS「Windows 10」に対応

・クラウドストレージスキャン
Microsoftの「OneDrive」に保存されているデータをスキャンし、ウィルスを検出。ウィルスのスキャンはトレンドマイクロのクラウド上にあるエンジンを用いて行うため、パソコンに負荷はかからない

・情報漏えい対策のマイナンバー対応
事前にマイナンバーの一部をウイルスバスタークラウド 10に設定しておくことで、インターネット利用時にマイナンバーが外部へ送信されることを防ぐ

・不要と思われるプログラム対策
ユーザの操作を中断する広告や不快な広告を大量に表示するプログラム、誇張もしくは偽の通知を表示するプログラムなどを検出する

・セキュリティ証明書チェッカー
アクセスするWebサイトのSSL証明書が偽装されている場合、アクセスを抑止する

・プライバシー設定チェッカーのFacebookアプリ対応
Facebookのプライバシー設定を確認し、プライバシー保護の観点から推奨の設定を提示する。最新版では、Facebook上で利用できるゲームなどのアプリが自動的に投稿するメッセージの公開範囲をチェックする

ウイルスバスター for Mac

・スマートスキャンの対応
ウィルスを検出するためのデータであるパターンファイルの一部をクラウドに移行。これによりクラウド上の最新パターンファイルを常に利用できることに加え、端末の負荷を下げられる。リアルタイムの保護と軽快さを同時に実現した

ウイルスバスターモバイル

・Android向け新機能
インストールしているアプリを一覧で管理できる新機能を追加。アプリのアンインストールや、ダウンロード済みアプリの確認が簡単にできる

・iPhone/iPad（iOS）向け強化機能
SNSのプライバシー設定を確認し、プライバシー保護の観点からお勧めの設定をアドバイスする機能が強化され、Facebookに加えTwitterにも対応。推奨する設定に変更することで、より安全にSNSを利用できる

●ダウンロード版製品の価格

製品	バージョン	価格（税込）
ウイルスバスタークラウド 10	1年版	5,380円
	2年版	9,680円
	3年版	12,780円
ウイルスバスタークラウド 10 ＋デジタルライフサポートプレミアム	1年版	7,980円
	2年版	13,800円
	3年版	18,580円

●パッケージ版製品の価格（9月4日発売）

製品	バージョン	価格（税込）
ウイルスバスタークラウド 10	1年版	6,460円
	3年版	13,810円

※パッケージ版の「ウイルスバスタークラウド 10＋デジタルライフサポートプレミアム」、および同時購入版はすべてオープン価格。

CONTACT
トレンドマイクロ㈱ URL http://www.trendmicro.co.jp

Event

9月2日、「SoftLayer Bluemix Summit 2015」開催

9月2日、ベルサール渋谷ファースト（東京渋谷区）にて「SoftLayer Bluemix Summit 2015」が開催される。

「SoftLayer Bluemix Summit 2015」は、IBMのIaaS型クラウド「SoftLayer」と、PaaS型クラウド「Bluemix」のユーザコミュニティが主催する、国内最大級の技術カンファレンス。2つのクラウドサービスについて、国内外の最新事例、普段聞けない話題、専門技術の話題、ハンズオンなどさまざまな企画がTrackA～Gに分かれて並行で執り行われる。基調講演として、「IBM Bluemixの最新動向と今後の方向性（日本アイ・ビー・エム㈱ 東京ソフトウェア＆システム開発研究所 クラウド開発 部長浦本直彦氏による）」「SoftLayerの最新動向と今後の方向性（IBM社SoftLayer CTO Marc Jones氏による）」などが行われる予定。

参加費は無料。申し込みはイベント支援サイトの「Compass」から（**http://softlayer.connpass.com/event/17037/**）。

▲Compass内のポスター画像

CONTACT
SoftLayer Bluemix Summit 2015
URL http://softlayer-bluemix-summit.jp

Software

サイオステクノロジー、機械学習搭載ITオペレーション分析製品「SIOS iQ」を販売開始

サイオステクノロジー㈱は7月28日、機械学習機能を搭載したITオペレーション分析ソフトウェア「SIOS iQ Standard Edition」の販売を開始した。

本製品は、2015年2月から提供中の無償版である「SIOS iQ Free Edition」の機能に加え、VMware仮想環境の性能問題の原因分析と予測をする機能を新たに搭載し、システムの性能問題の迅速な解決と未然防止に貢献する。

SIOS iQは、VMware仮想環境で稼働するシステムの最適化と問題解決を迅速に行うために、機械学習技術を用いて開発された。物理ホスト、ストレージ、仮想マシン、アプリケーションなど、システムを構成するすべての要素の振る舞いを、24時間×365日、包括的に監視し、その中で日常的な振る舞いと要素間の相関関係・相互依存性を自身で学習し続け、アノマリ（異常）を検出すると同時に原因分析を行う。これは、発生した問題の迅速な解決を可能にするだけでなく、システムへの影響も予測するので、深刻な問題の発生を未然に防止し、システムのサービス・レベルの向上にも寄与する。

今回新しく追加された性能問題の原因分析・予測機能は、従来の閾値（しきいち）による異常検出とはまったく異なるアプローチを採用し、異常値の判断基準となる閾値やポリシーを設定する必要がない。そのほかのおもな機能は次のとおり。

・フラッシュ・リード・キャッシュの効果分析・設定値の提案
vSphereのvFRCや各フラッシュ・ストレージ・メーカーが提供するフラッシュ・リード・キャッシュ機能における最適なキャッシュサイズとブロックサイズを導き出して提案する。製品自身でシステムの振る舞いを学習することにより、キャッシュサイズとブロックサイズの最適値を分析し、設定を変更した場合の効果（IOPS／レイテンシー）を予測する

・PERCダッシュボード（SIOS PERC Dashboard）
PERCダッシュボードは、性能／効率性／信頼性／キャパシティの4つの指標で、VMware仮想環境を評価し、迅速・明快にシステムの堅牢性を把握することを可能にする。また、シンプルで簡単な操作で、症状／影響範囲／原因／提案などの詳細な情報を参照することができ、俊敏に問題解決や最適化のアクションを取ることができる

・Microsoft SQL Server専用拡張機能
VMware仮想環境で、Microsoft SQL Serverを利用している場合、SQL Serverの振る舞いとそのほかの要素の相互関係を学習し、性能問題に関してさらに詳細な分析を行うことが可能

「SIOS iQ Standard Edition」は、1物理ホストに1ライセンス（240,000円／12ヵ月）を利用するサブスクリプション形式で提供、日本、米国、欧州の各市場向けに販売するとともに、Free Editionの提供も継続する。

CONTACT
サイオステクノロジー㈱ URL http://www.sios.com

ひみつのLinux通信

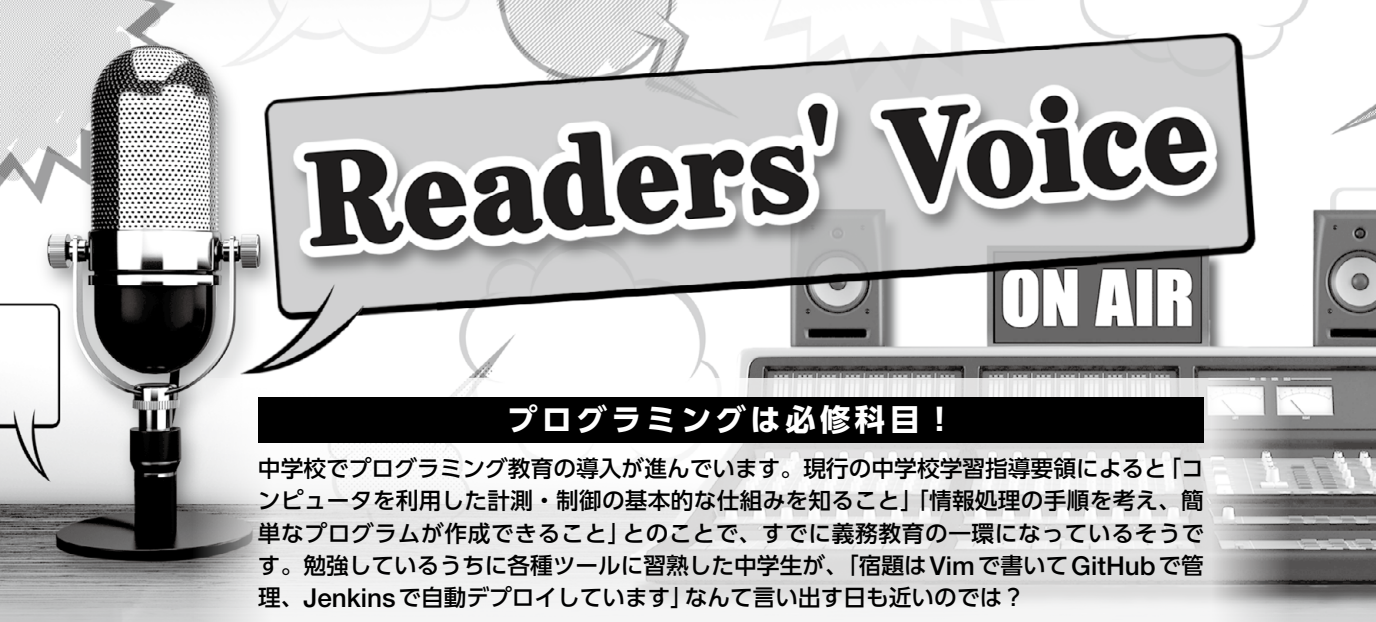
作)くつなりょうすけ
@ryosuke927

第20回 妖怪のせいなのね

FSS13巻を心待ちにしている妖怪はたくさんいると思うよ～（担当編集もその一人だけど）。

to be Continued

最近なんでも妖怪のせいにできてしまう風潮があります。眠気が目覚まし時計に勝って起きれなかったり、ココゾというときにスマホが再起動していろいろ手間取ったり、宝くじが当たらない、歯の被せモノがはずれる、出張なのにラップトップのACアダプタを忘れる、肝心な時にDNSサーバがメンテナンス落ちしてる、思わぬバグを踏んで作業時間がなくなる……。ま、妖怪のせいにしても怒られるのは自分ですけどね。ていねいに進めてても、そういうときはあるので、何にでも「ココロにゆとり」を持って行きましょう。そういえば、この業界には下手を踏ませる妖怪以外に、気づいたら良い方向に進めてくれる「妖精さん」ってのがいて……（妄想話はこの辺にしときます）。



プログラミングは必修科目！

中学校でプログラミング教育の導入が進んでいます。現行の中学校学習指導要領によると「コンピュータを利用した計測・制御の基本的な仕組みを知ること」「情報処理の手順を考え、簡単なプログラムが作成できること」とのことで、すでに義務教育の一環になっているそうです。勉強しているうちに各種ツールに習熟した中学生が、「宿題はVimで書いてGitHubで管理、Jenkinsで自動デプロイしています」なんて言い出す日も近いのでは？

2015年7月号について、たくさんの声が届きました。

第1特集 ログを読む技術[セキュリティ編]

脆弱性を突いた攻撃、DDoS攻撃といったサイバー攻撃の足跡「セキュリティログ」の分析についての特集です。OS標準のコマンドやツールを使った、分析手法・ツールを紹介しました。実際の攻撃が残したログを見ながら、手元でできるセキュリティ対策について学びました。

普段アプリ側の開発を行っているので、すごく勉強になった。ログの読み方・分析などすごく良い内容だと思った。
鈴木さん／埼玉県

Auditは知ってはいましたが、イマイチ使えていなかったので役立ちました！　ewiad420さん／神奈川県

知識が少ないもので、よくわからないかと思ったがサイバー攻撃がここまで来ているかということを知ることができて、いい勉強になった。
bunbunさん／大分県

日頃、ログに注視できていなかったのでとても参考になった。
カルマドさん／埼玉県

ログの読み方の基本を復習できたのはとてもよかった。なおかつ初心者ではなかなか見抜けないDDoS攻撃についても解説があり、長く読める特集だと思う。　romeosheartさん／長崎県

DDoS攻撃で"ゆっくり攻撃"する手法が増えてきていると載っていて驚きました。高速・大量アクセスへの対応が十分拡がったからだろうか？と、考えさせられました。
とーふやさん／神奈川県

日本年金機構の情報流出事件が報道され、情報セキュリティに対する意識が高まる中、今回の特集であるログ分析技術は興味深く参考になりました。サイバー攻撃があった際にいち早く検知できるようにさっそく実践してみます。
スマイルさん／茨城県

OSの標準機能を使って手元ですぐにできるセキュリティ対策、といったところが好評でした。相次ぐサイバー攻撃のニュースで不安になっていたなか、特集を見ながら実際にログをのぞいてみた読者の方も多いのでは？

第2特集 黒い画面(tmux)の使い方

tmux(ターミナルマルチプレクサ)は端末を仮想化して、ウィンドウの分割、プログラムの多重実行を可能にするツール。特集ではそのtmuxについて、導入方法と基礎知識・基本コマンド、カスタマイズ、現場での使われ方を解説しました。

Windowsマシンでも、Cygwinにtmuxいれてtailです！
きよしさん／大阪府

画面が暗くても将来は明るくいきたいものです(絶対違う……)。
tekitoizmさん／東京都

ターミナルを見るとgccしたくなる。ターミナルはプログラマの原点。
就活の巨人のさん／長崎県

さっそく、自分のすべての環境をtmux対応に整備してしまいました。おかげさまでマウスなしで快適な操作が可能となり、感謝感謝です。
psiさん／東京都

普段tmuxを使っているのですが、ググった設定をコピペしているだけで、使いこなしているとはとても言えない感じでしたので、第2特集の活用事例がとても参考になりました。
犬棟梁さん／埼玉県

7月号のプレゼント当選者は、次の皆さまです

①「Raspberry Pi B+」&「IR Camera module」セット
naoki様（千葉県）

②エアーサクセスミニ
佐藤法子様（千葉県）、さり様（愛知県）
平田妙子様（愛知県）

③Paragon Disk Wiper
向後久様（東京都）

④Amazon Web Servicesパターン別構築・運用ガイド
木下恵介様（神奈川県）

⑤「仮想化」実装の基礎知識
時武佑太様（東京都）、浅野浩史様（神奈川県）

⑥Javaパフォーマンス
xenserver様（東京都）、bina様（東京都）

⑦サーバ／インフラエンジニア養成読本 基礎スキル編
平川邦雄様（神奈川県）、永井一輝様（東京都）

※当選しているにもかかわらず、本誌発売日から1ヵ月経ってもプレゼントが届かない場合は、編集部（sd@gihyo.co.jp）までご連絡ください。アカウント登録の不備（本名が記入されていない場合、プレゼント応募後に住所が変更されている場合など）によって、お届けできないことがあります。2ヵ月以上、連絡がとれない場合は、再抽選させていただくことがあります。

真っ黒い画面を、マウスを使わずすべてキーボードで操作する姿はまさに凄腕のハッカー。tmuxの導入によって、利便性はもちろんモチベーションも高められそうです。

第3特集 スペシャリストになる方法

データセンタ、データベース、セキュリティ、ネットワーク、インフラ、ソフトウェアという6つの分野で活躍するベテランエンジニアに、「スペシャリストになる方法」を訊きました。若い人にはぜひ読んでもらいたい記事です。

先駆者の言葉は若手エンジニアにとって良い道しるべになると思います。
木檜さん／千葉県

それぞれのスペシャリストたちが、観点は違えど最後は勉強あるのみだったのが良い。　raihennさん／東京都

耳が痛い言葉もたくさん。
藤田さん／東京都

最新技術や流行っているものについて学ぶ、というのは共感した。
板垣さん／神奈川県

本誌では珍しい、読み物のみの特集でした。変化の速いIT業界では、分野によらず勉強し続けることが大事なようです。勉強の方法として、ひたすら手を動かす、技術書を読む、コミュニティに顔を出す、などが挙げられました。

短期連載 Kotlin入門【4】

Java仮想マシン上で動作するオブジェクト指向言語「Kotlin」についての短期連載。第4回ではオブジェクト指向を実現する「クラス」を解説しました。

今後ビジネスでも役に立てられそうな連載で続きが楽しみです。
ＹＹさん／神奈川県

こういう記事を読んでいると、そもそものプログラミングの足腰が鍛えられるような気がします。　Tayuさん／千葉県

新しい言語を学ぶことは、すでに使っている言語の良いところや悪いところを再発見する機会にもなります。読者の声にあるような、プログラミング一般のスキル向上も望めることでしょう。

連載

「Sphinxで始めるドキュメント作成術」のいろいろなテーブルの記述方法がとても興味深かったです。ただ、修正やほかのテキスト処理などを考えると、いろいろなアプリで共通化されたしくみがあればと感じました。
出玉のタマさん／大阪府

Sphinxの連載のおかげで、Sphinxを使ったドキュメントの書き方を少しずつではあるけれど覚えることができまし。おかげで、仕事の1つである「ソフトウェア設計書の作成」にSphinxを導入するのに成功しました。
福名　一さん／岡山県

Python製のドキュメンテーションツール「Sphinx」、少しずつ人気が出てきているようですね。エンジニアの方は、やはり文書や資料もプログラマブルに作成したいという思いがあるのでしょうか。

コメントを掲載させていただいた読者の方には、1,000円分のQUOカードをお送りしております。コメントは、本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」からアクセスできるアンケートにてご投稿ください。みなさまのご意見・ご感想をお待ちしています。

次号予告

Software Design
October 2015

2015年10月号
定価（本体1,220円+税）
192ページ

9月18日発売

［第1特集］多層防御や感染後対策を汎用サーバに実装

攻撃に強いネットワークの作り方

ShowNetの知見を盛り込んだネットワークセキュリティ最前線

最新機器を投入し、インターネット全体の安全性を高めるという試みがなされたShowNet 2015。一方、各企業のネットワーク担当者が現実的なレベルでセキュアなネットワークを構築するとしたらどうすればよいのでしょうか。

本特集では、ShowNetでの知見を盛り込み、“今すぐ取り組める”セキュアなネットワーク構築の指針を解説します。

［第2特集］機能、運用、セキュリティ……ベストな利用形態を探せ!

Webメールの教科書

クラウドサービス利用か?自社で構築か?

大規模Webメールサービスのインフラ基盤やスパムメール対策などを紹介。
また、自社メールシステムをOSSでWebメール化する方法も取り上げます。

［特別企画］「munkiによるMacクライアント一元管理」
［新連載］Vimの細道（仮）

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

休載のお知らせ

「書いて覚えるSwift入門」（第8回）、「Debian Hot Topics」（第30回）は都合によりお休みさせていただきます。

SD Staff Room

●関数型プログラミング特集を企画してやってみて思ったのは温故知新。ITエンジニアを目指して日々研鑽していくのならば、昔の本から読むといいかもしれない。コンパイラ作り、OS作り、MY言語作りといったところからパーサー作りなんかもよいでしょう。まずは自分のやれるところから。（本）

●「TVのチャンネルを回して」と言えば家で通じるが、編集部内で聞くと「変えて」でしょと言われる。電話もダイヤルを回すというのを知らない人が多い。それでも未だに保存のアイコンがフロッピーディスクだったりして、世の中古いんだか新しいんだか。捨ててあったラジオからバリコン取りしたのは大昔の話か。（幕）

●「じんましん」にやられました。発熱もあり、かなりキツかったです。熱が下がったと思われる夜に見た夢がすごく印象的で、それまでの混沌としたものから、突然絵本のようなやさしい線画に一変! ウィザードリィの3D迷路風な道を歩きながら懐かしい人とすれ違う内容（?）がビミョーでしたが。（キ）

●先日、セミの幼虫が数年の地下生活を終え地上に出てきているところを初めて目撃しました。羽化する様子も見たかったのですが、急いでいたので泣く泣くその場を離れました。あとでWebで調べたところ、羽化観察のコツは、幼虫を見つけたら家に持ち帰り、室内で観察すれば良いとのこと。結構ひどい。（よし）

●最近、クレイジーソルトにハマっています! 岩塩とスパイスのミックス調味料で、ふりかけるとなんでも本格的な料理になります（気のせい?）。チキンソテー、生野菜、ポテトサラダなどにかけて食べるのがお勧めです。1本（約100g）500円と強気の値段設定なので、豪快に使えないのがネックです。（な）

●母からきゅうりとトマトの苗をもらいました。といってもトマトはもう実がついている状態だったので、苗とは言えないかな。きゅうりは少し大きな鉢に植え替えたら、連日の強風にあおられ、抜けかけそうになるし、トマトは青い実はいっぱいできたけど、一向に赤くならないし……。収穫はいつになることやら。（ま）

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［E-mail］
sd@gihyo.co.jp

［FAX］
03-3513-6173

※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

Software Design
2015年9月号

発行日
2015年9月18日

●発行人
片岡 巌

●編集人
池本公平

●編集
金田富士男
菊池 猛
吉岡高弘
中田瑛人

●編集アシスタント
松本涼子

●広告
中島亮太
北川香織

●発行所
㈱技術評論社
編集部
TEL：03-3513-6170
販売促進部
TEL：03-3513-6150
広告企画部
TEL：03-3513-6165

●印刷
図書印刷㈱

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

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