Special Feature 1
→実習で身につけるSQL
Special Feature 2
→コンピュータは割り算が弱点

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

November 2017

11

2017年11月18日発行
毎月1回18日発行
通巻391号
（発刊325号）
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価
本体 1,220円
+税

Special Feature 1

データ分析に効くSQL 50本ノック

現場で培われた実践ノウハウ

SQL初学者のエンジニアも
営業担当も
自分でデータ分析が
できるようになる！

そう言われてみれば
うまく説明できない……

Special Feature 2

なぜ、コンピュータは割り算が下手なのか？

CPUの中の動きからわかる、
効率の良いプログラミングの
手がかり

Extra Feature

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

目次

Contents

Software Design
November, 2017
11

目次

Contents

Software Design
November, 2017
11

Special Feature 2 → 第2特集

Extra Feature → 一般記事

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

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

Software Design plus

最新刊！

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

データサイエンティスト養成読本 登竜門編
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-8877-5

改訂2版 データサイエンティスト養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-8360-2

改訂3版 Linuxエンジニア養成読本
養成読本編集部 編
定価 2,080円+税　ISBN 978-4-7741-8385-5

改訂3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 2,080円+税　ISBN 978-4-7741-8034-2

【改訂新版】サーバ構築の実例がわかるSamba［実践］入門
髙橋基信 著
定価 2,680円+税　ISBN 978-4-7741-8000-7

AWSエキスパート養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-7992-6

サーバ／インフラエンジニア養成読本 DevOps編
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-7993-3

［増補改訂版］クラウド時代のネットワーク技術 OpenFlow実践入門
高宮安仁、鈴木一哉、松井暢之、村木暢哉、山崎泰宏 著
定価 3,200円+税　ISBN 978-4-7741-7983-4

Unreal Engine&Unityエンジニア養成読本
養成読本編集部 編
定価 2,280円+税　ISBN 978-4-7741-7962-9

ソフトウェアエンジニアのためのITインフラ監視［実践］入門
斎藤祐一郎 著
定価 2,280円+税　ISBN 978-4-7741-7865-3

Unityエキスパート養成読本
養成読本編集部 編
定価 2,480円+税　ISBN 978-4-7741-7858-5

Android Wearアプリ開発入
神原健一 著
定価 2,580円+税　ISBN 978-4-7741-7749-6

Docker実践入門
中井悦司 著
定価 2,680円+税　ISBN 978-4-7741-7654-3

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

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

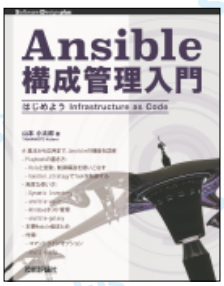
常田秀明、水津幸太、大島騎頼 著、Bluemix User Group 監修
B5変形判・288ページ
定価 2,800円（本体）+税
ISBN 978-4-7741-9084-6

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

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

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

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



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

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

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

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

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

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

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

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

BOOKS INFOMATION

現場で役立つシステム設計の原則

変更を楽で安全にするオブジェクト指向の実践技法

増田亨 著

A5判／320ページ
定価（本体2940円+税）

ISBN978-4-7741-9087-7

設計次第でソフトウェアの変更作業は楽で安全なものに変わる！
「ソースがごちゃごちゃしていて、どこに何が書いてあるのか理解するまでがたいへん」「1つの修正のために、あっちもこっちも書きなおす必要がある」「ちょっとした変更のはずが、本来はありえない場所にまで影響して、大幅なやり直しになってしまった」といったトラブルが起こるのは、ソフトウェアの設計に問題があるから。日本最大級となる求人情報サイト「イーキャリアJobSearch」の主任設計者であり、システム設計のベテランである著者が、コードの具体例を示しながら、良い設計のやり方と考え方を解説します。

コンテンツ・デザインパターン

吉澤浩一郎 著

B5変形判／208ページ
定価（本体2020円+税）

ISBN978-4-7741-9063-1

SEOにはじまり、SNSやコミュニティにおけるコミュニケーションにおいて必須なものとしてコンテンツの価値に注目が集まっています。しかし、「どうすればコンテンツをつくれるのか？」「どのようなコンテンツをつくればいいのか？」とお悩みの方も多いのではないでしょうか。
本書では、自社商品やサービスを売れるようにするために「誰に・何を・どのように」伝えていくべきかという流れとパターンを体系化し、具体的なコンテンツのつくり方をまとめています。国内外の豊富な事例を収録し、わかりやすく解説。Webマーケティング担当者必携の1冊です。

IBM Bluemix クラウド開発入門

B5変形判／288ページ
定価（本体2800円+税）

ISBN978-4-7741-9084-6

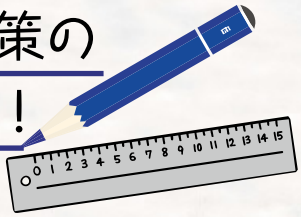
常田秀明・水津幸太・大島騎頼 著
Bluemix User Group 監修

IBMのクラウドサービスであるBluemixを基本的な導入方法から実際のアプリケーションを作る方法まで本書では紹介します。Bluemixの特徴は豊富なサービス群です。データを格納するための各種データストアサービス（RDBMS、NoSQL等）があります。さらにアプリケーションを開発するためのDevOpsサービス群、運用監視を行うための監視機能や、最近話題になっている人工知能利用のためのWatsonAPI等が提供されてます。これらをAPI経由で利用することによりシステムをより効率的に開発できます。

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

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）
ISBN978-4-7741-8508-8

金子則彦　著
A5判／688ページ
定価（本体3300円+税）
ISBN978-4-7741-8749-5

岡嶋裕史　著
A5判／680ページ
定価（本体2880円+税）
ISBN978-4-7741-8502-6

エディフィストラーニング株式会社　著
B5判／400ページ
定価（本体2980円+税）
ISBN978-4-7741-9055-6

岡嶋裕史　著
A5判／432ページ
定価（本体1880円+税）
ISBN978-4-7741-8501-9

庄司勝哉・吉川允樹　著
B5判／328ページ
定価（本体1480円+税）
ISBN978-4-7741-9054-9

大滝みや子・岡嶋裕史　著
A5判／744ページ
定価（本体2980円+税）
ISBN978-4-7741-8500-2

加藤昭・高見澤秀幸・矢野龍王　著
B5判／464ページ
定価（本体1780円+税）
ISBN978-4-7741-9053-2

角谷一成・イエローテールコンピュータ　著
A5判／544ページ
定価（本体1680円+税）
ISBN978-4-7741-8495-1

山本三雄　著
B5判／592ページ
定価（本体1480円+税）
ISBN978-4-7741-9052-5

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107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。

NGINX Application Platform

NGINXによる新しいアプリケーションプラットフォーム

「NGINX」は、軽量かつ安定性の高いWebサーバとして高いシェアをもつオープンソースソフトウェアです。そのNGINXの開発を主導しているNGINX社が、「NGINX Application Platform」という新しいプラットフォームを発表しました。

NGINX社は従来より、NGINXのエンタープライズ向けの商用版である「NGINX Plus」の提供を行ってきました。NGINX Application Platformは、そのNGINX Plusの技術やノウハウを軸として、モダンなアプリケーションの開発や管理を行うためのワンストップのプラットホームを提供するものです。クラウドやコンテナ、マイクロサービスといった要件をサポートし、市場の急激な変化に対応する柔軟性を備えた近代的なプラットフォームになるとのことです。

NGINX Application Platformは、NGINX Plusに加えて以下の3つの新しいツールから構成されます。

NGINX Unit

NGINX Unitは動的制御ができるように設計された軽量なWebアプリケーションサーバです。NGINXは多彩な用途に利用できる汎用的なWebサーバとして設計されていますが、一方で単体でアプリケーションを実行する環境は持っていません。NGINX Unitはその部分のニーズをカバーする存在になります。初期段階ではPHP、GoおよびPythonをサポートしており、将来的にはJavaやNode.jsなどほかの言語にも対応する予定だそうです。

NGINX Unitの最大の特徴は、可変的な環境でアプリケーションを動作させることを前提に設計されているという点です。アプリケーションのバージョンアップなどの保守作業を、プロセスを再起動することなくシームレスに実施できるほか、コンフィグレーションの変更もREST API経由で動的に行うことができます。また、サービスメッシュを構成するためのネイティブのロードバランサなども備えています。

NGINX Controller

NGINX Controllerは、NGINXのプラットフォームを集中的に管理するツールです。NGINXインスタンスの作成からコンフィグレーション、URLルーティング、SSL接続などの管理を、GUIから簡単に行えます。また、ワークフローやポリシー管理といった機能を備えるほか、マルチクラウドやマルチテナント環境に対応するなど、近代的なアプリケーションインフラに求められる要件をカバーしている点も大きな特徴です。

NGINX PlusとNGINX Unitの両方に対応し、両者の環境を統合して管理することもできるとのことです。

NGINX Web Application Firewall

NGINX Web Application Firewallは、NGINXプラットフォームをさまざまな外的リスクから保護するためのファイアウォール・ツールです。SQLインジェクションやクロスサイトスクリプティングをはじめとするレイヤー7攻撃の防止、DDoS攻撃への自動対応、IPレピュテーションといった機能が提供されます。

◆ ◆ ◆

上記のツール群のうち、NGINX ControllerやNGINX Web Application FirewallはNGINX Plusと同様に有償のツールですが、NGINX Unitについてはオープンソースで公開されており、無償で利用することができます。NGINX社ではそのほかにも、軽量なモニタリングおよび分析ツールであるNGINX Amplifyを正式リリースするなど、NGINXを中心としたエコシステムの強化に力を入れています。

NGINX Application Platformの立ち上げは、マイクロサービスに代表されるモダンなエンタープライズWebアプリケーションへの対応を推進していくというNGINX陣営の方針を改めて明確にしたものであり、今後の大きな改革を予感させるものと言えるでしょう。SD

The NGINX Application Platform
https://www.nginx.com/products/

DIGITAL GADGET vol.227

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

コンピュータグラフィックスの祭典SIGGRAPH2017［後編］
～映画の街ロサンジェルス。VRと映像技術編

技術の進化と映像の進化

コンピュータグラフィックスとインタラクティブ技術に関する世界最大の学会・展示会である第44回SIGGRAPH 2017が7月30日から8月4日の5日間、米国ロサンジェルスコンベンションセンターで開催されました（**pic.1**、**2**）。先月号に続いて、デジタルガジェット視点でレポートをお届けします。

今年のSIGGRAPHの注目は、昨年から大きな盛り上がりを見せつつある、VR、ARのコンテンツや応用技術です。VRと組み合わせた触覚デバイスや、各種センサーなどの研究開発も進みつつ、PlayStation VR、Oculus、HTC Vive、Sumsung GearVR、Google Daydreamといった、平易に手に入る市販VRデバイスを活用した、VR産業が大きく花開いてきている印象が強く感じられました。またSIGGRAPHの出展や発表でも、VR専業企業の活躍が多くみられました（**pic.3**）。

単なる物珍しい体験としてのVRから、映画に次ぐ映像コンテンツとして、ストーリー性のあるVR映像に、豊富な予算が投下され、経験豊富なメンバーが参画しつつあり、そういったVR作品のメイキングセッションもいくつか開催されました。

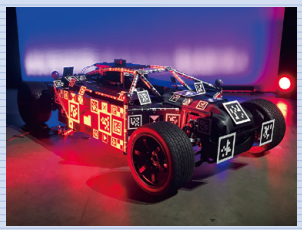
Sonaria

ストーリーが秀でたVR作品を配信し続けているGoogle Spotlightプロジェクトの作品の1つ。音を中心とした実験的作品で、縦横無尽に広がる三次元空間のための音の最適化を検討するためにいろいろな要素が加えられたものです（**pic.4**）。11の動物と、8つのセット（背景）が用意され、制作時には360度全方向を考慮したストーリーボードが描かれ、検討されたとのこと。たとえば、水の中のようなボコボコ、ポコポコといった音は実際に水の中に防水マイクを落として集録したり、空気中にいる場合、水中にいる場合、その境い目にいる場合と、それぞれ異なる環境音が考慮されました。音がどの方向から聞こえているかを常に考えながら編集された、とても手間のかかった作品です。

Son of Jaguar

同じくGoogle Spotlightの作品。義足の闘牛士から発想を得た、片足の覆面レスラーという独得のキャラクタが特徴的な作品です（**pic.5**）。アイルランドの詩人、オスカー・ワイルドの「素顔で語るとき、人はもっとも本音から遠ざかる。仮面を与えれば真実を語り出す」という言葉から着想を得たストーリーで、VR空間ならではのキャラクタ設定や、環境設定などが細かく考えられた作品です。実際のVR映像制作のまえに、数多くのスケッチや、カ

pic.1 SIGGRAPH会場に設置された、ロゴのオブジェ


pic.2 認識用の二次元バーコードがいたるところに貼り付けられた撮影用ダミー車


pic.3 脳波でVR映像をコントロール。HTC Viveを改造したNeurableの脳波VRヘッドセット

pic.4 Sonariaの1画面

※本記事で紹介しているものは
国内未発表・未発売のものを含んでいます。

ラースクリプトと呼ばれるストーリーに基づいた色設計、プロレス興行の調査などが行われました。キャラクタの形状をあえて、ポリゴン数が少ないゴツゴツした感じで表現し、その確認のために3Dプリンタも活用したそう。

DEAR ANGELICA

残念ながら現在は解散してしまった、VR作品専門の映像制作スタジオOculus Story Studioの作品です。Oculusの独自開発ツール「Quill」（https://www.oculus.com/story-studio/quill/）を使用してVR空間の中に手作業で描かれた素材をもとに作られたアート要素の強いVR作品です（**pic.6**）。作品の制作の流れとしては、Quillで描いた大量のデータを特殊効果の扱いに強い3DCGツール、Houdiniで編集、特殊効果、最適化、タイミングの調整を行い、VRにも対応したUnreal Engine 4というゲームエンジンでの映像再生を実現させたそう。人間が描いた手書きの順番や、描画の強さ、弱さといったものをうまくデジタル環境で再現し、表現することで、描く人の情熱や勢いみたいなものも伝えたかったとのこと。

◆ ◆ ◆

今年のSIGGRAPH会場には、参加者は無料で入場できるVR専用のシアターが設置され、会場内では20組のVRヘッドセットと視聴用のイス、サポートスタッフが待ち構えています（**pic.7**）。連日上映チケットがソールドアウトしてしまうという、人気の催しでした。VRシアターをはじめ、各所のVRの展示には「体験」のための工夫が見られたのも今年の特徴です。

- セッティングの補助が必要。動き回ると思わぬケガをしやすいので椅子に座って視聴する
- VR作品は見ている本人しかわからない。視聴を待っている人のためにデモ上映のディスプレイを設置
- デモ用の視聴映像は、意図的にゆっくりと動かないと、酔うくらいの映像になってしまうため要注意
- VR作品は長い時間集中して見続けることが難しいので、数分で完結するストーリー
- VR空間はどこを見ればいいのかわからない自由視点のため、演出で視線を導く工夫が必要（**pic.8**）

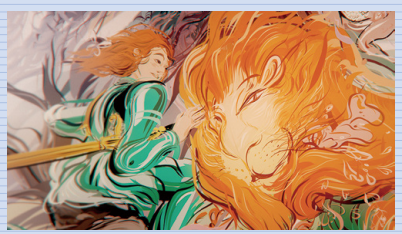
VRだけでない、先端技術

流行のVR技術に限らず、SIGGRAPHでは、数多くの先端技術が紹介されています。その中からいくつか特徴的なものを紹介しておきましょう（**pic.9〜13**）。

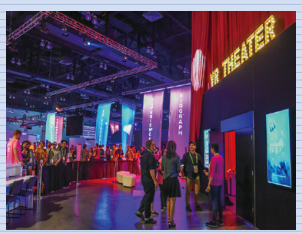
これからのコンピュータグラフィックスの進化

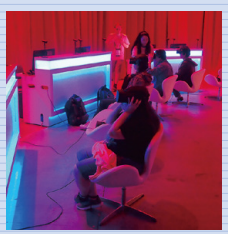
何度目かのブーム到来で、こんどこそ一般に定着するとも言われているVR。一方、スマートフォンの進化が頭打ちになりつつある現在、次の進化はVRやARの方向へ行くと言われつつも、まだまだ先が見通せていないのが現状です。ただ、昨年のSIGGRAPHと今年のSIGGRAPHとを比べてみると、VR関連の盛り上がりの大きな差異は、より多くの予算が動き、より多くの映像制作者が興味をもってVR映像の制作に参画してきていることです。

pic.5 Son of Jaguarに登場するキャラクタ制作中のスケッチ



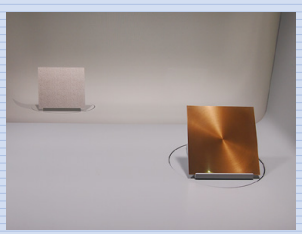
pic.6 DEAR ANGELICAのワンシーン



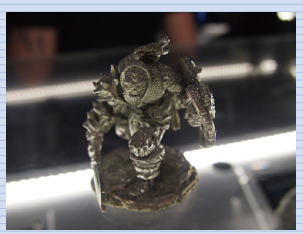


pic.7 左の写真は会場内に設置されたVR専用シアター。VR短編作品を何本かまとめて観ることができる。写真右はVRシアター内部。VRヘッドセットの装着を手助けしてくれるスタッフがおり、安全のため椅子に座って視聴する。会場の中央部には、応年のVRデバイス「任天堂バーチャルボーイ」の展示も

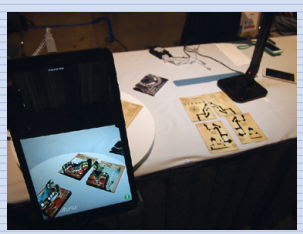
pic.8 参加者それぞれが、違う方向を見ている、VR体験中の奇妙な風景



pic.9 世界的に使われているカラーサンプルで知られるPantone社のx-riteは、実物とCGとを比較する質感マネジメント技術



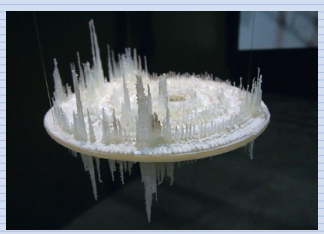
pic.10 formlabsという比較的安価に購入できる高性能3Dプリンタによる出力サンプル。ボードゲームのフィギュアのような金属質感の3Dプリントサンプル



pic.11 VRライブラリvuforiaの技術を活用したVRカードゲーム

また、今までは企業のサイドビジネスとして行われてきたVR関連の機材開発、VR映像制作も、専業の企業、専業のスタートアップ、専業の映像チーム、専業アーティストなどが生まれてきており、確実に裾野が広がっていることが実感されます。ただ、まだまだ先進的なものとしてとらえられることが多く、一般にはまだまだ浸透しておらず、ゲームに似た何かだと思われている場合もあります。より広くたくさんの人がVR映像を楽しみ、さまざまなVR映像が作られ、撮影され、映画に次ぐ一大産業になるのはまだまだ先ですが、その片鱗がうかがえる今年のSIGGRAPHでした。

来年夏のSIGGRAPH 2018は8月12日から16日の5日間、カナダのバンクーバーで開催されます。また、来年2018年冬のSIGGRAPH ASIAは東京（有楽町の東京国際フォーラム）での開催が決定しています。各国から集まるCG作品や、最新技術展示に多くの期待が集まっており、来年、また何が見られるのか、体験できるのか、今から楽しみです。3D



pic.12 地震の振動を目で見える形に3Dプリントしたアート作品。写真はエクアドルの高地とガラパゴス諸島の4つの活火山での地震活動を記録したもの

pic.13 i am aiという顔がタブレット端末のAIロボット。人間ふうに受け答え、流れ作業やパズルゲームを代行してやってくれる。ユーモラスでありながら、少し頼りない

Gadget 1

Echolocalizator

※イベント展示品のため参照URLなし

視覚遮断音波メガネ

SIGGRAPHアートギャラリー展示より。超音波の反射を聞き取りながら飛び回るコウモリのような体験ができる特殊な眼鏡。目が塞がれていても、潜水艦の「ピーン、ピーン」という音のように、周りの障害物を音に変換して聞くことで、周辺情報を把握することができる感覚デバイスです。この眼鏡を装着したばかりだと視覚が奪われて妙な感覚になりますが、徐々に音で周辺状況を把握しようと耳を中心とするさまざまな感覚が研ぎすまされてくることがわかります。

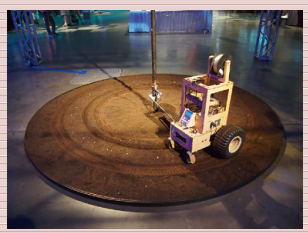
Gadget 2

Milpa Polimera

http://www.marcelaarmas.net/

自己増殖3Dプリンタ

SIGGRAPHアートギャラリー展示より。トウモロコシを原材料とするフィラメント（3Dプリンタの素材となるコイル状の樹脂）を用い、畑に模した円形状の土の上をグルグルとトラクターのように延々と動き続けるアート作品。移動型の3Dプリンターが動きながら出力し続けているのは、トウモロコシの形状をした種子。つまりは素材もトウモロコシ、生産しているのもトウモロコシ、生み育てようとしているのもトウモロコシという、ある閉じた生態系の中での循環をテーマに表現した作品です。



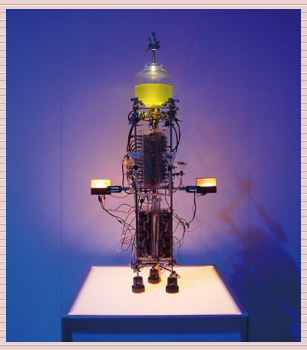
Gadget 3

BioSoNot 1.2

http://gilbertoesparza.net/

水の浄化装置

SIGGRAPHアートギャラリー展示より。汚水を微生物の働きで浄化する装置の生物的活性を、音に変換してその動きを感じるための装置。燃料電池のしくみを用いたバイオセンサーでは、微生物の代謝の際に発生する電子を捕捉して感知することができます。その情報を音に加工し、細菌の生活を描いた「交響曲」を奏でているとのことです。



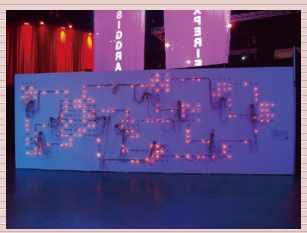
Gadget 4

Dispersiones

http://www.leonunez.com.ar/

生命のリレー

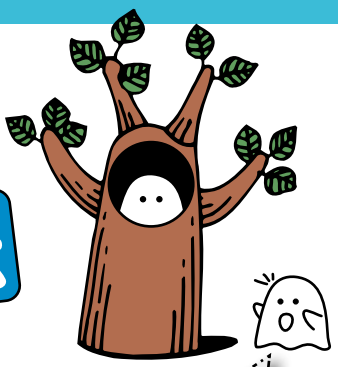
SIGGRAPHアートギャラリー展示より。音と光で人工生命を表現したインタラクティブな装置。物理的に相互接続された照明機器が光る様子を、生命の誕生、進化、淘汰を表現したライフゲームのような人工生命アルゴリズムで表現しています。伝搬していく様子を表す、スイッチで切り替わる機械式リレー端子と照明機器、その接続ケーブルで構成され、作品の前に立った人の動きに反応します。巨大な人工生物というよりも、ネットワークそのものが生命体のように感じさせる作品です。



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

54

結城浩の再発見の発想法



Backtrack ―― バックトラック

バックトラックとは

バックトラック(Backtrack)とは、**解空間の中を行きつ戻りつして目的の解を求める、深さ優先探索アルゴリズムの一種**です。**部分解**を少しずつ組み立てていき、途中で「これでは最終的な解に行き着けない」と判断したら、そこから深くなる解の探索をばっさり捨てて別の道を探る(バックトラックする)のが特徴です。バックトラックは、特定の問題を解くための具体的なアルゴリズムというより、もう少し抽象度が高い発見的な戦略と言えます。

私たちが何かの問題を解こうとするとき、「起こり得るたくさんの可能性から条件に合った解を探索する」という状況になる場合があります。

この場合、もっとも単純な解の探索方法は「1列に並んだすべての可能性を端から順に見ていって、条件に合うかどうかを調べる」というものです。これは「しらみつぶし」や「ブルートフォース」と呼ばれます。

今回のバックトラックも「すべての可能性を順に見ていく」という点ではブルートフォースと同じですが、バックトラックは単純に「1列に並んだものを端から見ていく」のではありません。バックトラックでは**木構造**として表現された可能性を**枝刈り**しながら見ていくのです。

8クイーン問題

バックトラックの説明をするとき、多くのアルゴリズムの教科書では「8クイーン問題」が例として挙げられます。

8クイーン問題とは、8×8のチェス盤に8個のクイーンを置く問題で、どのクイーンを見ても、ほかのクイーンの効き筋にいないという条件を満たす必要があります。チェスのクイーンは縦横斜めのすべてが効き筋になっており、飛車と角を足したような動きができますので、8×8の盤に8個のクイーンを置くのはなかなか難しい問題となります(**図1**)。

自明なこととして、8個のクイーンは列(aからhまで)と行(1から8まで)ごとに1個しか存在できません。ですから「a列はどの行にクイーンを置くかを決める」「b列はどの行にクイーンを置くかを決める」……のようにして、少しずつ部分解を作っていく方法が考えられます。その部分解の集合は**図2**のような木構造を作ることになり、最終的な解を求めるのは、この木構造を巡回(traverse)することに対応するのです。

a列1行にクイーンを置き、b列3行にクイーンを置き、c列……と木構造を下に進むと、やがて置けなくなる状態に至ります。そうなったら、それより下方向への探索を行うのは無駄です。残りのクイーンの位置を検討する必要はありません。そこでバックトラックが行われます。つまり、最後に置くことができたクイーンを別の位置に動かす部分解へと進むのです。言葉で説明するとややこしいですが、**図2**のような木構造で想像するなら状況は理解できるでしょう。

バックトラックでは木構造になった解空間の探索において「枝刈り」が行われ、それによって

効率的な解空間の探索を行うことができます。

「それならブルートフォースなんかせずに、いつもバックトラックを使えばいい」と考えたくなりますが、そうはいきません。バックトラックを使うためには、「現在のところまでは条件を満たしている」という部分解を作ることができる必要があるからです。

たとえば、暗号解読で手がかりが何もなければ、バックトラックは使えません。「326421479の鍵がダメだから、320000000番台の鍵はすべてダメ」のような枝刈りはできないからです。

バックトラックを行うためには、解空間の構造がわかっていなければなりませんが、効率を高めるため、とくにどのような順番で部分解を作っていくかは重要です。厳しい条件を先に確認すれば、探索する空間を小さくできるからです。

日常生活とバックトラック

日常生活で私たちは自然にバックトラックを行っています。たとえば、入り組んだ路地で目的の**家を探す**ときを考えます。1つの道を選び、進んでいきます。そして「いや、こんな遠くまで来るはずないから、この道の先にはない」と判断して後戻りし、別の道へ進みます。これはまさにバックトラックです。ここでも、部分解をどう作っていくかは重要です。「駅より東にあるはず」のような大きな条件を先に確認すれば、探索する空間を小さくできるからです。

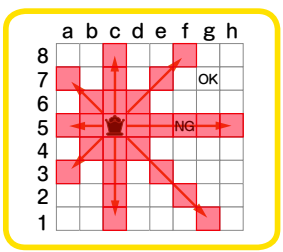
あるいは、数学の**試験問題を解く**ときを考えます。与えられた問題に対していくつかの解法を思いついたとします。式変形で知っている問題に帰着させるか、そのままごりごり計算するか、問題の性質を生かして単純化するか……どの解法を選んだとしても、さらにその先まで進まなければ本当に解けるかどうかはわかりません。最終的な解に行き着くまでの筋道は、抽象化して考えるなら木構造になるでしょう。1つの解法を選び、先に進んだけれど行き詰まったら、そこから先の作業はバッサリ捨てて、バックトラックすることになります。

数独を解く際にもバックトラックを使えます。「このマス目に入り得る数字は4と7と9のどれかだけど、まだどれになるかはわからない。もし4だとしたら……」というのは、数独の解空間を探索していることにほかなりません。矛盾が見つかるまで進み、矛盾が見つかったらバックトラックして別の部分解を試すことになります。数独を解くのがうまい人は、条件が最も厳しいマス目をすばやく見つけることができるのではないでしょうか。

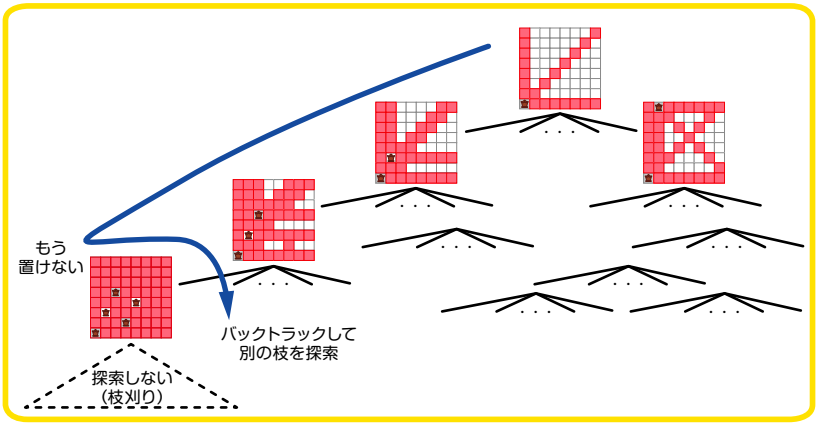
あなたの周りを見回して、多くの候補から条件に合うものを求める状況があるかどうか探してください。そのとき、バックトラックをしているでしょうか。探索するときの順番を変えることで、効率は変わるでしょうか。

ぜひ、考えてみてください。SD

▼図1　クイーンの効き筋



▼図2　8クイーンの木構造と、バックトラックでの枝刈り



及川卓也の プロダクト開発の道しるべ

品質を高めるプロダクトマネージャーの仕事とは？

第13回 プロダクトアイデアの生み出し方

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

前回（2017年10月号）はプロダクトマネージャーの実務を、会社や組織のビジョンやミッションから機能要求まで落とし込んでいくまでの過程を用いて説明しました。

その中でも触れましたが、すべてのプロダクトマネージャーに共通する役割にアイデアを生み出すことがあります。

良いアイデアを生みだすには

アイデアを整理し、製品の要求仕様に落とし込んでいくために必要な考え方について説明しましょう。

まず、最初に製品やサービスは基本的に2つの目的しかないことを理解する必要があります。

① ユーザの課題を解決すること
② ユーザに新しい価値を提供すること

誰にも使われないものを作りたいのでなければ、この2つのどちらかを実現する必要があります。良いアイデアとはこれらを実現するベースとなる考えなのです。

最初のアイデアが優れているとは限らない

前回もお話しましたが、最初のアイデアのほとんどはたいしたアイデアではありません。最初のアイデアは後から冷静になって見てみると、どうでもよいアイデアであることがほとんどです。ビジネスコンテストなどで大学生が「すばらしいアイデアを考えつきました！」とばかりにアイデアを発表したりするのを聞いていると、自分に酔ってしまっているんじゃないかなと感じることがあります。これは大学生などの若い人に限りません。どんなに経験の長い人も、最初のアイデアを思いついた瞬間に、それに惚れ込んでしまっていることがあります。これでは冷静な判断はできません。

アイデアを分解する

では、良いアイデアを生み出すためにはどうすれば良いでしょう？

それは、アイデアを分解し、分解された要素それぞれについての検証を行うことです。

先ほど、アイデアは①ユーザの課題を解決することか②ユーザに新しい価値を提供することのどちらかだとお話しましたが、アイデアをまずこの形に分解してみるのです。

②のユーザに新しい価値を提供することも①に吸収できたり、①での考え方をそのまま応用できるので、ここでは①のユーザの課題を解決することを考えてみましょう。

この「ユーザの課題を解決すること」というアイデアは「ユーザの課題」と「その解決策」という2つに分解できます。そして、良いアイデアというのは「ターゲットとなるユーザが抱える重大な課題」と「その最善の解決策」の組み合わせなのです（**図1**）。

アイデアをこの2つの要素に分解した後は、それぞれを検証することになります。アイデアを思いついた段階では、それぞれはあくまでも仮説

に過ぎません。課題仮説を検証し、そのあとに解決策の検証を行います(**図2**)。その検証2つが終了したあとに、初めてそのアイデアは実用化して成功の可能性があるアイデアとなるのです。

課題仮説の検証も解決策の検証も実際のユーザに当たるしかありません。ユーザへのヒアリングや行動分析などを行い、検証を行います。もし、すでに存在する製品から派生した製品であったり、その製品の新機能として提供することを考えているアイデアならば、ログの分析などからもユーザの課題を抽出できるでしょう。解決策もユーザへのヒアリングやモックやプロトタイプを制作し、ユーザに試してもらうなどして、その解決策で本当に課題を解決できるかを確認する必要があります。

この仮説検証で使えるのが、この連載の第10回(2017年8月号)で解説したバリュープロポジションキャンバスになります。その使い方については、第10回を参照していただくとして、ここでは最初に思いついたアイデアへの執着の捨て方について1つの方法を説明したいと思います。

解決策の千本ノック

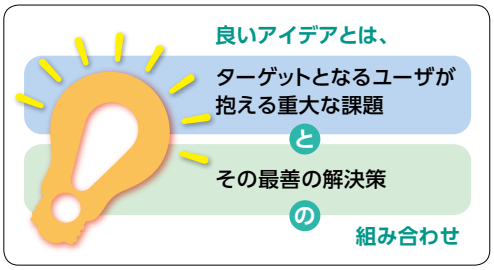
先ほど説明したように、最初にアイデアを思いついたとき、多くの人はそれに対しての愛着を持っています。その愛着は良いアイデアを考え出すには邪魔(じゃま)なこともあります。

執着を捨てるには、アイデアを何個も考え出すことです。ターゲットユーザが抱える重大な課題であることが検証された課題に対しての、解決策をいくつも考えてみましょう。10個と言わず、100個考えてみましょう。1,000個まで考えつくのはさすがに難しいと思いますが、大量の解決策を自分以外のチームメンバーも巻き込み考えてみるのが良いでしょう(**図3**)。

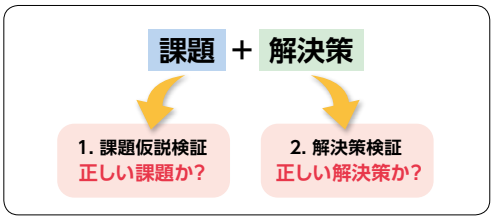
10個も100個も案が出ないと思うかもしれませんが、それでもひねり出してみましょう。数を出して並べてみると、案外99個目のものが一番良かったなどということもあるものです。なにより、このように案をひねり出すことで必然的に最初の案に対する執着が薄れていきます。結果として、実際にユーザに試してもらい評価が低かった場合でも、素直にそれを認め、その案を諦めることもできるのです。

数多くの案を出す工夫はいろいろありますが、ある程度案が出たら、それを机の上に並べてみることをお勧めします[注1]。KJ法と呼ばれる、ブレーンストーミングなどで出たアイデアを整理する方法がありますが、同じように案を並べ、グルーピング化してみます。その際のグループ化もいろいろな軸(＝側面)から行うことが大事

▼図1　良いアイデアとは

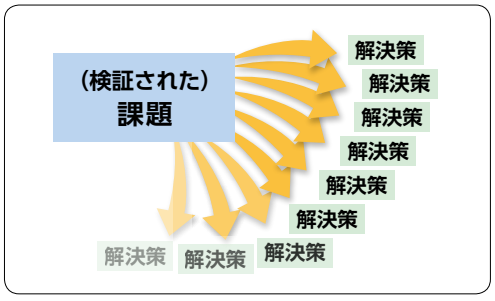


▼図2　アイデアを分解したそれぞれの要素について検証する



注1) 付箋紙に案を書くと、並べ替えが楽なのでお勧めです。

▼図3　アイデア(解決策)の千本ノックのように、多くの解決策をひねり出す。これにより、最初のアイデアへの執着を捨てるとともに、最善の解決策を考えつくことができる



です。

軸の持ち方の例として、5W1Hで考えてみる方法があります。解決策の案の場合、「誰に(＝Who)」と「なぜ(＝Why)」はすでに課題に含まれていることが多いですが、「誰が」や「誰と」という形でのWhoや、「どこで(＝Where)」、「どうやって(＝How)」、「いつ(＝When)」というのを軸に考えてみると案が生まれてくることがあります。

おもしろいのは、出ている案がバランス良く分散していない場合の考え方です。もちろん足りない要素を増やすように考えてみることもできますが、あえて、この課題は偏った解決策しかないのではないかとして、偏った要素をより尖らせていくというのもあります。

いずれにしろ、物事を上から見たり、横から見たり、斜めから見たりするときの切り口はいくらでもありますので、工夫してみることが大事です。

切り口の持ち方としては、ブレーンストーミングの考案者、A.F.Osborne氏によるオズボーンのチェックリストと呼ばれるものが有名です。これは次の9つの法則からなります。

1. ほかに使い道はないか？(転用)
2. ほかに似たものはないか？(応用)
3. 変えてみたらどうか？(変更)
4. 大きくしてみたらどうか？(拡大)
5. 小さくしてみたらどうか？(縮小)
6. ほかのものでは代えられないか？(代用)
7. ほかのものと入れ替えられないか？(置換)
8. 逆にしてみたらどうか？(逆転)
9. ほかのものと組み合わせられないか？(結合)

また、デザイン思考で有名なIDEOのCEOのTim Brown氏は商品(商品を提供する要素技術とここでは考えたほうが良いでしょう)と消費者をそれぞれ縦軸と横軸に取り、取り得るアプローチを4つに分割して考えられるとしています(**図4**)。

たとえば、既存技術を新しい領域に展開すること(**図4**右下「適応」のマス)でも、十分新たな顧客を開拓することが可能となります。Flickrという写真共有サービスがある中で、同じアイデアを動画に展開したYouTubeがユーザの支持を得たことなどは、既存アイデアを新しい領域に適応させたものと考えられます。イノベーティブなものを生み出そうとすると、つい新規性のある技術で新規ユーザの獲得を目指すことばかり考えがちですが、この図が示すようにほかにもアプローチはあります。

チームでのアイデアディスカッション

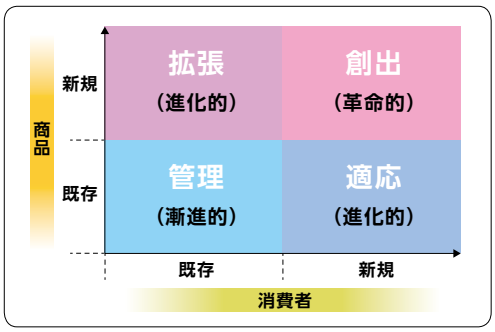
ここまでプロダクトマネージャー個人として、どのようにアイデア出しを行うかについて説明しました。アイデアはプロダクトマネージャーから出されることも多いですが、プロダクトチームの誰もが出せることが理想です。

プロダクトアイデアや製品要求をチームで議論することには、2つの利点があります。まず、多くのアイデアが集まることがあげられます。個人では考えもつかなかったアイデアが出てくることが期待されます。

次に、個人的にはこちらのほうがより重要と思われますが、チームメンバーの納得感が高まるということがあります。人は他人から言われたことをやることに心理的な抵抗感を持ってしまいがちです。同じ結論であっても、その結論が出る場

注2) Tim Brown 著、千葉敏生 訳、早川書房 刊、2014/05/10発行

▼図4 『デザイン思考が世界を変える』[注2]の中で紹介されているTim Brown氏によるイノベーションの分類



にいること、もっと言うと、その結論を自らが出した場合にコミットメントが高まります。

プロダクトマネージャーとして自信のあるアイデアがあったとしても、あえてそれを押しとどめ、チームからの意見を募るというのは、チームの結束を高める、チームビルディングとしても有効なのです。

リーダーシップのあり方

チームでのディスカッションとも関係するので、ここでプロダクトマネージャーのリーダーシップのあり方についても考えてみましょう。最初にお断りしますが、成功するプロダクトマネージャーに、これと言った型があるわけではありません。禅問答のようになってしまいますが、成功するプロダクトマネージャーが成功するのです。つまり、リーダーシップについても、最終的にはプロダクトを成功に導きさえすれば、どんな形であってもかまわないのです。

筆者はSteve Jobs氏をプロダクトマネージャーの成功例としてあげることがあります。Jobs氏は良い意味でも悪い意味でも極端なプロダクトマネージャーです。プロダクトに対する妥協が一切ない姿勢などは見習うべきところですが、二度と一緒に働きたくないという元同僚や部下が数多くいるなど、人格的には模範としづらいところも多くあります。ですが、iPhoneがそうであるように、あれだけ成功するプロダクトを世に出したというだけでも、紛れもない史上最高のプロダクトマネージャーの1人なのです。

ただ、Jobs氏になれる人は世の中にはほとんどいません。凡人とまでは言いませんが、一般のプロダクトマネージャーは個人ではなく、チームの力を使ってプロダクトを成功に導く必要があります。そこで気に留めないといけないのがリーダーシップのスタイルです。

リーダーシップの分類の1つに、トランザクティブリーダーシップとトランスフォーメーショナルリーダーシップという分類があります。従来型の上意下達で物事を決めていくスタイルはトランザクティブリーダーシップと呼ばれます。トランザクティブという言葉からわかるように、部下との関係を取引(＝トランザクション)のようにとらえ、高い成果をあげた部下には報酬を、そうではない部下には罰を与えるというスタイルです。昭和の時代のリーダー像と言うと語弊があるかもしれませんが、一昔前のリーダーはほとんどがこのスタイルでした。

一方のトランスフォーメーショナルリーダーシップは変革を指導するリーダーシップのことです。ビジョンを共有する、ビジョナリーのスタイルで、部下のモチベーションを向上させることで自覚を促します。

別のリーダーシップのスタイルとしては、サーバントリーダーシップというものもあります。こちらはサーバント(＝使用人・召使い)という言葉からもわかるように、相手に奉仕することで導くスタイルのリーダー像です。傾聴や共感、納得などが特徴となります。

ここでは、便宜上、部下という言い方をしていますが、プロダクトマネージャーの場合は、チームのメンバーと置き換えてかまいません。人事権を持っているかどうかは重要ではありません。むしろ、人事権を持っていないがゆえに、これらのリーダーシップのスタイルの中で、トランスフォーメーショナルリーダーシップやサーバントリーダーシップというのがプロダクトマネージャーには相応(ふさわ)しいものとして考えられています。

チームでのディスカッションなどを通じてアイデアやさまざまな意見を集約するには、このトランスフォーメーショナルリーダーシップやサーバントリーダーシップというのを意識するようにしてください。

◆ ◆ ◆

今回はアイデアの生み出し方とそれから派生し、プロダクトマネージャーのリーダーシップについて解説しました。次回は企業や組織、事業などの目標設定とその管理に使われる、OKR(Objectives and Key Results)について解説します。SD

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

宮原徹の

オープンソース放浪記

第21回 オープンデベロッパーズカンファレンスを開催しました

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

「開発」をテーマにイベントを開催

本連載は全国各地で開催されているオープンソースカンファレンス（OSC）の様子をおもにお届けしてきましたが、OSCはどちらかというとインフラエンジニア向けのイベントでした。

しかし世の中ではDevOpsやSREといったキーワードが流行しているように、インフラエンジニアも開発者とのコミュニケーションをとる機会が増え、構築作業自体も自動化スクリプトで実現するなど「開発」に積極的に関わる必要が出てきています。そこで8月19、20日の2日間、「オープンデベロッパーズカンファレンス」（ODC）を開催しました。

久しぶりの日本工学院専門学校

ODCの会場となった日本工学院専門学校はJR蒲田駅すぐ近くと絶好のロケーション。校舎も建て替わって数年というすばらしい会場です。建物は違いますが、以前にもOSCの会場として利用させていただいたこともありましたので、久しぶりの会場です。

ODCは初開催ということもあって人が集まるのかどうか不安なところもありましたが、朝からたくさんの方が来場してくれてホッと一安心。最終的に、初日の土曜日が250名、2日目の日曜日が150名と、想定の400名ちょうど参加していただけたので、見込みどおりでした。

セッションの内容ですが、普段のOSCではあまり話すことができないOSSがどのように開発されているのか、コミュニティ開発の裏側について説明するセッションが複数あったのが印象的でした。私自身が企画したMySQLとPostgreSQLの開発の比較セッションも、たとえばPostgreSQLはRedmineなどのチケット管理ではなく、メーリングリストで開発の課題が管理されているというのは驚きでした。そのほか、システム設計やDevOpsに関わるセッション（**写真1**）など、私自身が今興味を持っていること（**写真2**）についてジックリ聴くことができたので、企画運営している私自身が楽しめるイベントになりました。こういう収穫があると、今後も頑張って継続していくぞ、というモチベーションアップになりますね。今回は東京での開催でしたが、今後の各地のOSCの中にも開発をテーマにしたセミナーを積極的に取り入れていくつもりです。

2日目のメインはLLイベント

今回のODCの2日目は、メインがLLイベントです。これまではLLは「Lightweight Language」でしたが、今回は「Learn Language」とい

▼写真1 私のDevOps超入門の講演。本連載のページをスクリーンに映して宣伝中

▼写真2 『スラスラわかるPython』（翔泳社）の監修をした寺田学氏と。機械学習の講演も大好評でした

▼写真3　LLイベントの竹迫良範氏の基調講演の様子。日曜日にもかかわらず100名以上が参加しました

写真4▶
日本酒立ち飲みの店「日本酒人」。次々と日本酒が入れ替わるので、つい通ってしまいそうです

うことで、基調講演の「ハッカーになるためには何の言語を勉強したらいいですか？」を始め、開発言語の学習がテーマとなりました(**写真3**)。とくに最近注目されている関数型言語のセッションなど、私も興味があったのですが、私は私で裏側でRaspberry Piのハンズオンなどをやっていたので聞けずじまい。それでも、あらためて自分の中で気になっていることを発見できるのも、このようなイベントの醍醐味ですね。

蒲田の夜は更けていく

さて、会場のあった蒲田はたくさんの飲み屋さんがありますので、当然懇親会も盛り上がります。前日準備が終わったあと、蒲田といえば餃子だろ、ということで餃子を堪能したあと、私は1人で翌日の2次会のお店を探しに1人で夜の蒲田をウロウロと。一応目星を付けておいた日本酒立ち飲みのお店(**写真4**)に行くと、いい感じだったので突撃して店主お勧め3種飲み比べを注文。それぞれ90ml(0.5合)が3つですので、1.5合につまみ2品が付いて2,000円は安い！　隣にいる常連のお兄さんたちの会話を楽しみながら、明日はここに何人来るかな〜と想像しながらほどほどで帰宅しました。

さて、翌日は懇親会1次会が終わったあと、有志でまた日本酒立ち飲みのお店へ、10名ほどでゾロゾロと。土曜日のせいか少し混んでいましたが、奥の方に陣取らせてもらって、思い思いに注文します。どれもこれも安いので、どんどん注文が入っていくのが怖いところ。明日もあるんですよ〜、と思いつつ、私も少し飲み過ぎてしまいました。しかし、**写真5**のように揃いのTシャツを着て入っていったので、いったい何者かと思われたに違いありません。SD

▼写真5　講師陣と個人スポンサーに配布した「開発」Tシャツで記念撮影

Report

まさに昭和！ 蒲田の「鳥まん 本店」

2日目は有志のみなさんが会場の片付けを手伝ってくれたので、その後そのまま打ち上げに。向かったのは会場ほど近くの「鳥まん 本店」です。昭和39年(1964年)創業の老舗、安くて美味しい大衆居酒屋です。とくに名前に鳥が入っているとおり、若鶏の唐揚げがサイズも大きくて、揚げたて熱々で美味しいです。また、こういうお店はビールももちろんいいですが、焼酎ボトルにホッピーが似合います。みんなで美味しく、2日間の疲れを癒すことができました。

▶鳥まんの鶏の唐揚げ。1人1皿ぐらいがちょうどいいです。火傷注意

◀左の赤いのは「バイス」と言うらしい。紫蘇風味で焼酎割りで飲むそうです。初めて見ました

ツボイのなんでもネットにつなげちまえ道場

第29回

LoRaWANを使ってみる（前編）

Author 坪井 義浩（つぼい よしひろ） @ytsuboi

はじめに

今回は、今話題のLPWA（Low Power Wide Area）の中でもとくに注目されているLoRaWANを使ってみたいと思います。普段はLoRaと呼ばれていますが、LoRaというのは無線の変調方式の呼称で、「LoRaWAN」というのが仕様全体を指す呼称です。つまり、LoRaとは、LoRaWANのうち物理層の部分の呼び名なのです。

変調とは電波に情報を載せるときの変換のことです。変調について詳しく説明をすると、4ページまるごと埋まってしまいますので、詳しく知りたい方は、サイレックスさんの記事[注1]などを参照してください。

今回は、日本国内でLoRaWANを使ってみるための最も手っ取り早い方法を提供しているであろう、SORACOMさんのソリューションで、LoRaWANの技術の説明をしつつ実際に動かしてみることにします。

LPWA

LoRaWANは、Low Power（低消費電力）、Wide Area（広域）というのが特徴です。無線ネットワークと言えば、ケータイのLTEや、Wi-Fiがすぐに思いつく規格だと思いますが、これら2つは高速で通信できる代わりに、とても多くの電力を必要とします[注2]。より消費電力の少ないところでは、BluetoothやBluetooth Low Energyといった規格がありますが、これらはせいぜい10mくらいが通信距離でしょう。EnOcean[注3]は、エナジーハーベスティング（環境発電技術）で動かすくらい低消費電力で、筆者が試したことがある範囲では展示会場で60mくらい飛びました。でも、屋外で無線通信をしようとするには少し心許ない距離です。

こういった、屋外にセンサノードの設置を考えた場合、電池で運用をしたくなり（つまり低消費電力が要求され）、離れた場所との通信がしたくなり（つまり広域の通信が要求され）ます。IoTというものが流行りつつありますが、こういった要求を満たす技術や規格がLPWAです。

LPWAには第21回でも紹介したNB-IoTなども含まれますが、NB-IoTの国内サポートはまだまだ時間を要しそうです。LPWAとして頻出の規格としては、ほかにSigfoxがありますが、こちらについては近いうちに紹介をすることにします。

LoRa変調（物理層）

さて、LoRaWANの物理層について説明をしたいと思います。まず、LoRaWANはいわゆるサブギガ（1GHz以下の周波数帯）の無線を使って通信をします。サブギガといえば、ソフトバンクが5年ちょっと前に割り当てを希望していた「プラチナバンド」を含む周波数帯です。周波数の低い電波は、回折（かいせつ）と言って、障害物の影に回り込んで伝わりやすいという特徴があります。

注1） http://www.silex.jp/blog/wireless/2013/03/post-4.html

注2） LTEについては、本連載の2017年3月号の第21回と4月号の第22回で扱いました。

注3） EnOceanは本連載の2017年7月号の第25回で扱いました。

Wi-FiやBluetoothに使われている2.4GHz帯は、世界共通で免許不要で使える周波数帯です。ケータイやWi-Fiなど、無線通信を多用しているので忘れがちですが、本来、ほとんどの周波数帯域は使用免許を受けずに利用してはいけないのです[注4]。

ケータイについては携帯電話事業者が包括免許を受けていますし、Wi-FiやBluetoothについては先述の2.4GHz帯を利用しています。LoRaWANも免許不要の周波数帯を使うのですが、サブギガについては国によって免許を受けずに快適に使用できる周波数帯が異なります。このあたりは2017年7月号の本連載第25回でも少し触れましたが、LoRaWANの仕様でも日本では923MHz帯なのに対して、アメリカでは915MHz帯、ヨーロッパでは433MHz帯と868MHz帯といった具合に国や地域によって使用する周波数帯域が異なります。ですので、海外用のLoRaWANの無線機は工事設計認証（いわゆる技適）以前に、そのままでは国内で運用できません。

また、一般社団法人電波産業会（通称ARIB、アライブ）の標準規格で、920MHz帯にはデータの「送信休止時間」が規定されています。このため、国内で920MHz帯を使用するときには、この送信休止時間を設ける必要があることからも、他国向けの機器やソフトウェアをそのまま使用できないといった事情もあります。

LoRaWAN（MAC層）

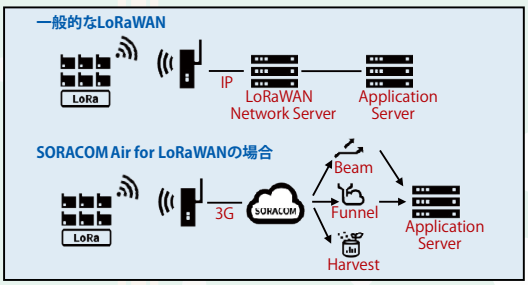
冒頭で、LoRaWANは仕様全体を指す名称だと記したように、LoRaWANはLoRaより上のレイヤの仕様を含みます（**表1**）。

LoRaWANの変わっているところは、LoRaWANのゲートウェイ（エンドデバイスが無線通信をする相手）は単なる中継機器であることです。ゲートウェイは、エンドデバイスとネットワークサーバ（バックエンドとも呼びます）の間でやりとりされるLoRaパケットをIPパケットに変換して転送をするだけの装置で、通信制御はネットワークサーバが行います（**図1**）。

たとえば、LoRaのエンドデバイスには固有のIDがあるのですが、このIDの管理やデータの暗号化・復号といった制御はネットワークサーバが行うのです。ですので、LoRaWANにはネットワークサーバが不可欠なのです（LoRa変調でエンドデバイスどうしが直接通信することも可能で、これもLoRaと呼ばれるために多くの人が混乱をしています）。ゲートウェイとネットワークサーバは、IPネットワークで接続されますが、LoRaWANのエンドデバイスはIPで通信をするわけではありません。

LoRaWANには、3つの通信クラスがあります。一般的に用いられるのはClass Aで、エンドデバイスから通信を始める方式です。エンドデバイスは必要なときに無線機の電源を入れるため、消費電力が少ないというメリットがあります。定期的にエンドデバイス側の無線機の電源を入れ、ゲートウェイからのビーコンを受信したのをきっかけに通信を行う方式がClass Bです。エンドデバイスの無線機の電源を入れっぱなしにして、サーバから通信を開始できる方

▼図1　LoRaWANのネットワーク



▼表1　LoRaWANとレイヤ

	LoRaWAN	Ethernet
アプリケーション層	-	HTTPなど
ネットワーク・トランスポート層	-	IP、TCP、UDP、TLS
データリンク層	LoRaWAN（LoRaWANは、下位層のLoRaを含む）	Ethernet
物理層	LoRa変調	1000Base-T

注4）電波が著しく微弱な機器については、この限りではありません。

式がClass Cです。無線機の電源を入れているとその分電力を消費するため、Class AよりClass Bが、Class BよりもClass Cのほうが電力を消費するという特徴があります。

LoRaWANにはData Rate(データレート)というしくみもあります。センサーとゲートウェイが近くにあるなど電波条件が良い場合は高速なデータレートを選び、逆に電波条件があまり良くない場合は低速ながらノイズに強いデータレートを選ぶことができます。通信できる距離と通信速度はトレードオフなのです。よくLoRaの通信速度は、と質問を受けますが、データレートしだいだというのが答えになってしまいます。すごく乱暴に概算の数字を挙げると、LoRaの通信速度は20bpsくらいです。キロとかメガを付け忘れたわけではありません。秒20bitと極端に低速ですので、LoRaやLoRaWANで通信することができるのは温湿度などそういった少量の数値データだと考えておくとよいでしょう。画像データなどを送るのには向きません。なお、エンドデバイスの設置場所が移動する場合などは、ADR(Adaptive Data Rate)というしくみで、データレートの切り替えを行うことができます。

通信速度と通信距離のほかのFAQとして、暗号化が挙げられます。先ほどLoRaWANのネットワークサーバのところでデータの暗号化と復号に触れたとおり、LoRaWANには暗号が仕様に含まれています。LoRaWANでは、MAC層とその上のアプリケーション層それぞれでAESによる暗号化が行われます。MAC層とアプリケーション層それぞれ別の鍵が使われるのには理由があります。先ほど述べたように、LoRaWANではゲートウェイは中継機器で、ネットワークサーバの存在が前提です。ネットワークサーバは、MAC層の暗号を使って不正なデバイスからのデータを排除します。このMAC層の暗号の鍵はネットワークセッションキー、アプリケーション層の暗号鍵はアプリケーションセッションキーと呼ばれています。この暗号における共有鍵は、エンドデバイスの工場出荷時にあらかじめ書き込んでおく(ABP：Activation by Personalization)か、エンドデバイスに工場出荷時に仮の鍵を書き込んでおき、エンドデバイスがLoRaネットワークに接続するJOINという手順の際に仮の鍵を使い認証を行い、そのあとに本番の鍵を発行するOTAA(Over the air activation)という方法で発行します。たいていはエンドデバイスを製造する者とネットワークサーバを提供する者は別ですので、OTAAが便利でしょう。

使ってみる

概説がとても長くなってしまいました。何事も実際に動かさないと身につきませんので、実際にSORACOMさんのソリューションで動かしてみましょう。まずゲートウェイについては、共有サービスモデルゲートウェイであれば「LoRa インドアゲートウェイ AL-020」を購入して、受け取ったときにユーザコンソールの「発注」画面で「受け取り確認」ボタンをクリック、開梱して電源につないで設置するだけです。SORACOMのゲートウェイには「所有モデル」と「共有サービスモデル」の2種類が存在し、共有サービスモデルは電気通信事業法の関係か、設置場所を気軽に変更することはできませんし、常時接続状態の維持が必要となっています。

幸運にもオフィスや自宅の近くに共有サービスモデルのゲートウェイがある方は、このゲートウェイを利用してエンドデバイスだけでLoRaWANを試用してみることができます。

▼写真1　LoRa インドアゲートウェイ AL-020を設置

SORACOMの共有サービスのゲートウェイの設置場所は、LoRa Space[注5]で確認できます。ここには「通信ログ」として接続実績のデータが掲載されていますので、ゲートウェイと通信ログの間の地点であれば恐らくLoRaWANに接続できるでしょう。

SORACOMが販売しているエンドデバイスは、「LoRa Arduino開発シールドAL-050」と「LoRa GPSトラッカーLT-100」の2種類です。現在のところ、デバイスID（MAC層の項で触れたIDです）をユーザコンソールに登録するには、SORACOMにデバイスを注文して「受け取り確認」ボタンをクリックするのが唯一の方法です。ですので、今のところSORACOMのソリューションでLoRaWANを使うには、この2種のどちらかを購入するしかありません。GPSトラッカーはあとあとできることに限りがありそうですので、Arduino開発シールドを使ってみることにします。

エンドデバイスを試しに動かしてみましょう。実験するアプリケーションは、LoRa Spaceに接続ポイントを登録するものにします。これはArduino開発シールドを使うもので、SORACOMが方法を詳しく記したページ[注6]を公開しています。ここで使用するGPSモジュールは、筆者の手元にあったGroveのGPSモジュールに置き換えて実際に試してみました。用意したものは、「LoRa Arduino開発シールドAL-050」とArduino Uno R3、それからBase Shield V2[注7]です。

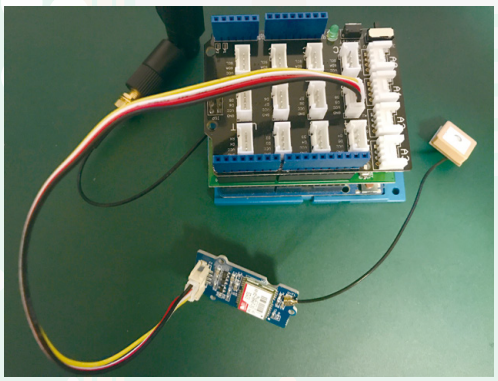
Arduino Unoで使いますので、Base Shieldのスイッチが「5V」側になっていることを確認し、Arduino UnoにLoRa Arduino開発シールドとBase Shieldを取り付けます。また、Grove - GPSにGroveのケーブルを挿し、反対側はBase ShieldのD8の端子に挿しました（**写真2**）。

SORACOMのArduino library for LoRaWAN Deviceに含まれるサンプルスケッチ、SLS_GPS_LoRa.inoはとくに書き換える必要はありません。

組み立ててスケッチを書き込んだところで、外に出てGPSを受信できているか確認してみました（**写真3**）。こういったGPSは、スマートフォンのGPSのようにWi-Fiやセルラーのアシストがないため、空が見えるところでなければ位置情報を得ることができない場合がほとんどです。

筆者は**写真3**のように基板を円筒状のタッパーに納めてみました。LoRaのアンテナを立てておきたかったのと、LoRa Spaceへの登録の際に基板を持って街中をウロウロするので、怪しい人に思われないためです。実際のところ、ノートパソコンとタッパーを持って歩いていたら、やはり人々の視線が少々刺さりました。SD

▼写真2　Grove - GPSを接続



▼写真3　GPSの受信テストの様子

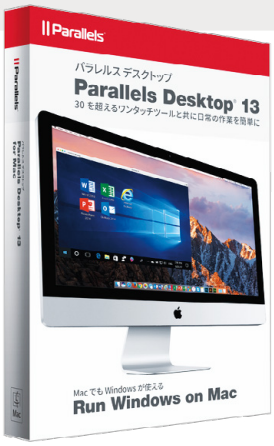
注5）https://lora-space.soracom.jp/map
注6）https://dev.soracom.io/jp/start/loraspace-register-point/
注7）https://www.seeedstudio.com/Base-Shield-V2-p-1378.html ）、Grove - GPS(URL: https://www.seeedstudio.com/Grove-GPS-p-959.html

読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」にアクセスし、アンケートにご協力ください（アンケートに回答するにはgihyo.jpへのお名前と住所のアカウント登録が必要となります）。ご希望のプレゼント番号を記入いただいた方の中から抽選でプレゼントを差し上げます。締め切りは**2017年11月16日**です。プレゼントの発送まで日数がかかる場合がありますが、ご容赦ください。

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

01 Parallels Desktop 13 for Mac



3名

Macマシン上でWindowsやmacOSなどの仮想マシンを実行できる仮想化ソフトです。最新のWindows 10 Fall Creators UpdateおよびmacOS High Sierraに対応し、Windowsの仮想マシン上で動くアプリと機能を「Mac Touch Bar」から起動できるようになりました。通常版を2名様に、Visual Studio、Vagrant、Chef、Jenkinsなどの開発ツールをサポートするPro Edition（1年更新版）を1名様にプレゼントします。

提供元 パラレルス https://www.parallels.com/jp

02 オゾンの力 for トイレ

トイレやお部屋の気になるニオイを、オゾンで強力分解・脱臭する装置です。消臭剤などと違って薬品を使わないので、お手軽に使用できます。またフィルタ交換などは必要なく、コンセントに接続するだけで使用できます。人の動きを感知して動作を停止する人感センサーも搭載。

提供元 フォースメディア
http://www.j-force.net

1名

03 Bluemix ノベルティTシャツ

人工拡張知能「Watson」や、GUIでアプリを開発できる「Node-RED」などが特徴的な、IBMが提供するPaaS製品「IBM Bluemix」のノベルティTシャツです。Lサイズをプレゼント。

提供元 日本アイ・ビー・エム
https://www.ibm.com/ibm/jp/ja

1名

04 クローリングハック

竹添 直樹、田所 駿佑 ほか 著

サイトを巡回してWebページの内容を取得する「クローラー」の開発において押さえておくべき知見——文字コード、HTML、認証、Ajax/JSON、認証、クローリングマナーについてまとめられています。

提供元 翔泳社
http://www.shoeisha.co.jp

2名

05 SREサイトリライアビリティエンジニアリング

Betsy Beyer ほか 編

SREとは、Googleでサービスの信頼性を支えるエンジニアチームが蓄積してきた、システム運用に関するベストプラクティス。各分野ごとに、Googleの最先端のノウハウを学べます。

提供元 オライリー・ジャパン
https://www.oreilly.co.jp

2名

06 マジメだけどおもしろいセキュリティ講義

すずきひろのぶ 著

過去のセキュリティ事件を題材に、その事件が起きた背景や真の原因を解説した1冊。サイバー攻撃の巧妙な手口は興味深いと同時に、知ることで企業や家庭のセキュリティを考えるヒントにもなります。

提供元 技術評論社
http://gihyo.jp

2名

07 Amazon Web Services負荷試験入門

仲川 樽八、森下 健 著

クラウド環境（Amazon Web Services）での負荷試験のノウハウを紹介しています。システムにかかる負荷を正しく見積もることで、本番でも想定どおりのパフォーマンスを発揮させることができます。

提供元 技術評論社
http://gihyo.jp

2名

第1特集

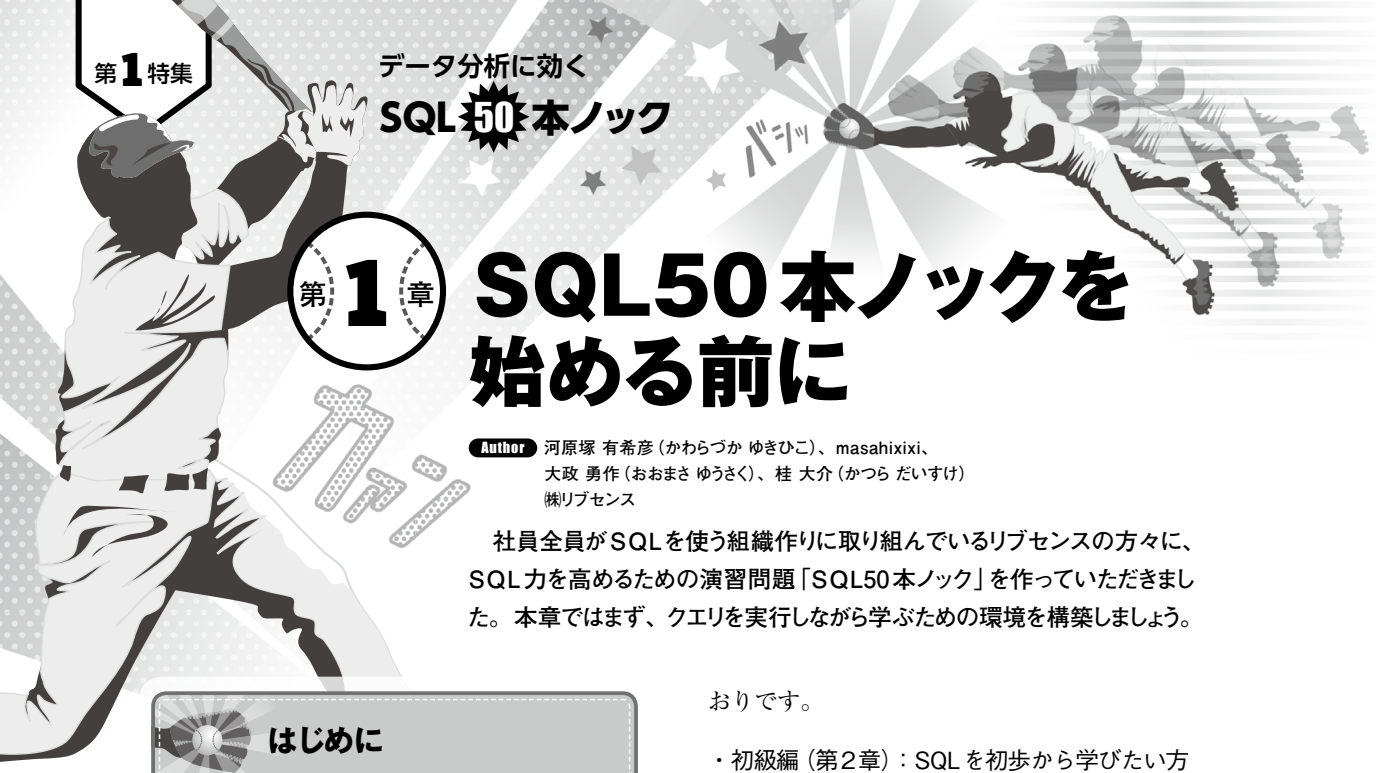
データ分析に効く SQL 50本ノック

Author 河原塚 有希彦、masahixixi、大政 勇作、桂 大介
㈱リブセンス

近年、SQLを利用する機会が増えています。顧客情報・販売情報などの重要データはデータベースに集まるため、営業やマーケティング担当者からデータの抽出・集計の依頼を受けるエンジニアも多いことでしょう。その際にSQLが必要になってきます。もはや開発で使うだけの時代ではないのです。

そこで、SQLでデータを抽出するところから始めて、累積比率・移動平均を求められるようになるまでの演習問題「SQL50本ノック」を用意しました。学習環境を簡単に構築できるスクリプトもあります。

自分のSQL力を磨くために使うも良し、社内のSQL勉強会の教材に使うも良し。本特集をデータ・ドリブンな組織づくりに役立ててください。



第1章 SQL50本ノックを始める前に

Author 河原塚 有希彦（かわらづか ゆきひこ）、masahixixi、
大政 勇作（おおまさ ゆうさく）、桂 大介（かつら だいすけ）
㈱リブセンス

社員全員がSQLを使う組織作りに取り組んでいるリブセンスの方々に、SQL力を高めるための演習問題「SQL50本ノック」を作っていただきました。本章ではまず、クエリを実行しながら学ぶための環境を構築しましょう。

はじめに

今日、データ・ドリブンな意思決定が営業やマーケティングの現場でも求められています。「SQL力」はそれを加速させる強力な武器であるにもかかわらず、SQL活用の推進は十分とは言えず、データ抽出はいまだエンジニアの仕事となっているのが、多くの現場の状況ではないでしょうか。

本特集ではSQL力を確実に身に付けるべく、初歩から応用まで幅広いノック（問題）を用意しました。また別立てのコラムとして、組織内でのSQL力の活用方法についても紹介します。

初学者の教科書としてはもちろん、周囲へ教えるときの参考書や、組織でのSQL浸透のヒントとして、幅広く本特集をご活用ください。エンジニアが起点となって、社内にデータ・ドリブンの文化が根づいていく――本特集がその1つのきっかけとなれば幸いです。

想定読者

本特集のSQLノックは初級編・中級編・応用編に分かれています。ノックは徐々に難しくなるように作られており、1本ごとに独立していますので、自分のスキルに合わせて読み進めてください。各章のレベルと想定読者は次のとおりです。

- 初級編（第2章）：SQLを初歩から学びたい方
- 中級編（第3章）：複数テーブルを組み合わせてデータ抽出する方法を学びたい方
- 応用編（第4章）：SQLを使いこなしてさらに多様な使い方を身に付けたい方。とくに、自由自在に分析を行う際に便利な機能やノウハウを知りたい方

それぞれのノックでは、説明用のサンプルクエリを載せています。まず1周目はサンプルクエリを見ながら読み進めて、2周目からはノックだけを読んで解けるか挑戦しましょう。

実行環境を用意しよう

本特集の実行環境として、PostgreSQLを用意します。もちろん、業務で使うデータベースがMySQLやOracleの読者にも参考になるよう、一部を除いてSQL標準に準拠したクエリで説明します。インストールの手間もないように、簡単に環境構築を行えるスクリプトも用意しています。

データベースサーバ（PostgreSQL）を用意しよう

本特集では実行環境を簡単に用意できるよう、シェルスクリプト（Windows 10ではバッチファイル）を用意しています。このスクリプトは、

Dockerを用いて「コンテナ」と呼ばれる仮想環境上でデータベースサーバを構築するものです。利用環境はLinuxの各ディストリビューション、macOS、Windows 10を想定しています。

Dockerのインストール

Dockerは各OS向けにプログラムが提供されていますので、Dockerの公式サイト[注1]から環境に応じてダウンロード、インストールしてください。Community EditionとEnterprise Editionがありますが、今回は学習用途ですのでCommunity Editionを選択してください。

WindowsでDocker Community Editionを動作させるためには、Windows 10 Enterprise/Professional/Educationのいずれかのエディション、かつHyper-Vに対応したCPUが必要です。さらに、コントロールパネル［プログラムと機能］-［Windowsの機能の有効化または無効化］からHyper-Vを有効化する必要があります（**図1**）。

詳細なインストール手順は、Docker公式サイトの次のページを確認してください。

- Ubuntu：https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/
- CentOS：https://docs.docker.com/engine/installation/linux/docker-ce/centos/
- Windows 10：https://store.docker.com/editions/community/docker-ce-desktop-windows
- macOS：https://store.docker.com/editions/community/docker-ce-desktop-mac

スクリプトの実行

次に、本特集のサポートページ[注2]から構築用スクリプト一式「env_script.zip」をダウンロードし、展開したうえで実行してください。

注1）URL https://www.docker.com/get-docker
注2）URL http://gihyo.jp/magazine/SD/archive/2017/201711/support

・Linux/macOS

```
$ sh postgres_initialize.sh
```

（環境によってはsudoが必要です）

・Windows 10（コマンドプロンプト）

```
> postgres_initialize.bat
```

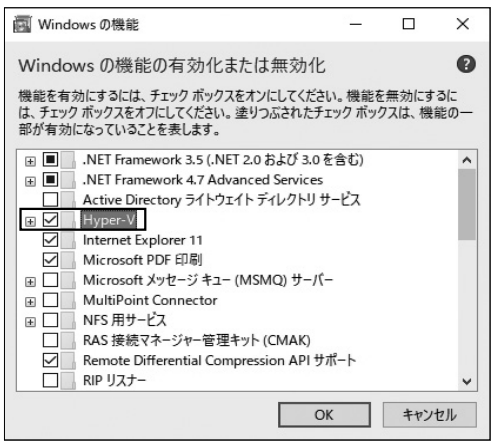
実行すると、コンソールには数分間にわたってサーバのダウンロードとインストールのメッセージが流れます。この間、Debianイメージ上でPostgreSQLのサーバが立ち上がり、その中に今回使うサンプルデータ（後述）が展開されます。しばらく待つと、PostgreSQLのクライアントである「psql」が立ち上がり、データベースサーバに接続します。コンソール上に、

```
#================
# login postgres
#================
psql (9.6.5)
Type "help" for help.

postgres=#
```

のように表示されれば、準備完了です。

psqlを終了する際は、\qと入力してください。なお、データベースサーバはDocker上で終了するまで起動したままとなります。再度psqlを使う場合、もう一度シェルスクリプトを実行してください。これにより、データベースサー

▼図1　Hyper-Vの有効化



バが起動していてもしていなくても、データベースサーバの再起動とデータ初期化を行って、クリーンな環境が用意されます。

サンプルとして扱う「Pagila」について

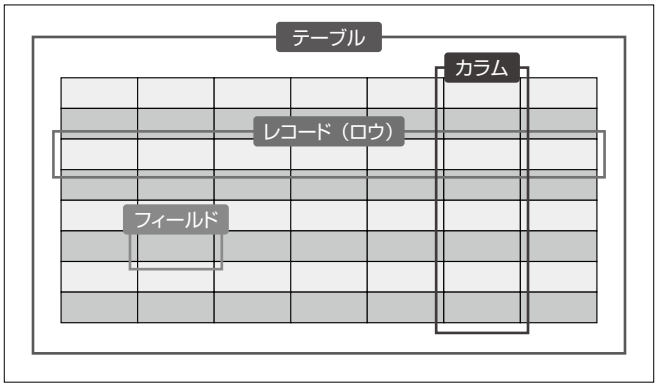
Pagila[注3]は、PostgreSQLで用意されているサンプルデータで、同じくMySQLで提供されている「Sakila」をポーティング（移植）したものです。PagilaはDVDレンタルショップで取り扱うデータをモデリングしたもので、出演俳優をはじめとする作品メタ情報からカスタマー情報、そして売上情報まで含まれたデータベースとなっています。全体のER図など詳しい情報を知りたい方は、MySQLの公式サイト[注4]をご覧ください。

データベースとテーブルの構成

第1章の最後に、データベースにおける基本的な用語をおさらいしておきます。

一般的に、データベースは複数のテーブルで構成されています。Pagilaのように20テーブル程度のものもありますし、エンタープライズシステムでは数百テーブル以上から構成されるものもあります。

▼図2　テーブルの構成



テーブルは、値が2次元に配置されたものです。言葉で説明するよりも、**図2**を見てもらうのが早いでしょう。テーブルはレコード（行）とカラム（列）で構成されていて、レコードとカラムが交わったところをフィールドと言い、そこに値が格納されています。SD

注3）URL https://www.postgresql.org/ftp/projects/pgFoundry/dbsamples/pagila

注4）URL https://dev.mysql.com/doc/sakila/en/sakila-structure.html

column　SQL上達の秘訣は?

SQL上達の秘訣はズバリ、「自分の興味がある、もしくは必要に迫られているデータを利用すること」です。日々業務で利用しているデータの抽出を、自分自身の手でクエリを書いて行うことが大きなモチベーションになります。日々業務で使う必要に迫られたデータだからこそ、取得条件の間違いや甘さにも気づけ、修正するための創意工夫も苦なく行えます。こうしていつの間にか反復練習が行え、スキルアップが成されるのです。

column　本番では分析用のデータベースを準備しよう

本特集でSQLの基礎を学んだあと、いざ現場で分析をしようとなったときに最初にやるべきことは、分析用のデータベースを準備することです。アドホックな分析クエリは高負荷になりがちです。トラブルを避けるために、たとえ参照系のクエリしか実行しないとしても、本番環境に分析クエリを投げることは可能な限り避けるべきです。

データベースは、万一に備えて定期的にバックアップが取得されているはずです。そのバックアップデータから分析用のデータベースを準備しましょう。データベースに個人情報が含まれている場合は、法令・ガイドライン[注A]に従って適切なアクセス制御やデータの匿名化などの対応を行ってください。

注A）「個人情報保護委員会」URL https://www.ppc.go.jp

第2章 初級編
SQLの基本を学ぶ

Author masahixixi、大政 勇作（おおまさ ゆうさく）、
河原塚 有希彦（かわらづか ゆきひこ）
㈱リブセンス

いよいよSQL50本ノックの始まりです。ノックという形でデータの抽出・集計などの問題を出していきますので、それを実現するクエリを一緒に考えながら、SQLを学んでいきましょう。

customer、paymentテーブルを使ってSQLの基本を学ぶ

初級編では、おもにcustomerテーブルとpaymentテーブルを使ってSQLの基本を学んでいきましょう。

はじめにそれぞれのテーブルに存在するカラムを簡単に整理しておきます（**図1**）。

customerテーブル

顧客情報が格納されたテーブルです。顧客ID（customer_id）、氏名（last_name、first_name）などが格納されています。

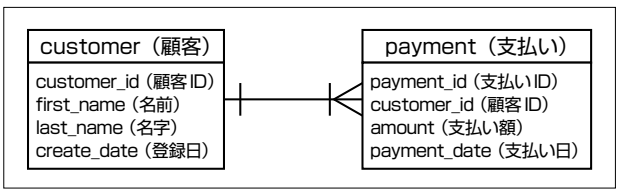
paymentテーブル

支払い情報が格納されたテーブルです。支払いID（payment_id）、顧客ID（customer_id）、支払い額（amount）、支払い日（payment_date）などが格納されています。1人の顧客が複数回利用することがあるため、顧客と支払いは1対多の関係にあります。また、顧客なしで支払いはできないため、支払いテーブルには必ず顧客が関連づきます。

◆ ◆ ◆

それでは実際にクエリを書きながら学んでいきましょう。

▼図1 customerテーブルとpaymentテーブルの関係



データを抽出する - SELECT

ここから先では、第1章で準備した環境でクエリを実行しながら進めてください。

それでは、SELECTから学んでいきます。テーブルからデータを取り出すときに使用するのがSELECTです。まず、次のノックに挑戦してみてください。

paymentテーブルの一覧を表示する

答えは、次のクエリになります。実際に、実行してみてください。

```
SELECT * FROM payment;
```

いかがでしょうか。**図2**のように何やらたくさん結果が出てきました。psql（PostgreSQLのターミナル型フロントエンド）でのクエリ実行結果は、moreコマンドと同じ操作で移動できます[注1]。/rowsと入力すれば実行結果のレコード数が見つかりますので、(16049 rows)と表示されていることを確認してください。

続いて次も実行してください。

```
SELECT payment_id FROM payment;

payment_id
------------
     16050
     16051
     16052
     16053
     16054
(..略..)
(16049 rows)
```

注1）「Enterキーで次の1行を表示」「Spaceキーで次の1画面を表示」「/文字列 で指定した文字列を検索し、その場所に移動」など。

さらに次も実行してみましょう。

```
SELECT
  payment_id,
  customer_id
FROM
  payment
;

payment_id | customer_id
-----------+-------------
     16050 |         269
     16051 |         269
     16052 |         269
     16053 |         269
     16054 |         269
(..略..)
(16049 rows)
```

ここまで連続で実行し、少しイメージが湧いたでしょうか。どのクエリもテーブルからデータを取り出していることがわかるかと思います。

SELECTとFROMの間に*を指定するとすべてのカラムを抽出、カラム名を指定すると指定したカラムのみ抽出、複数指定したいときには,で区切る。

以上のことをふまえて次に進みましょう。

▼図2 「SELECT * FROM payment;」の実行結果

```
 payment_id | customer_id | staff_id | rental_id | amount |        payment_date
------------+-------------+----------+-----------+--------+----------------------------
      16050 |         269 |        2 |         7 |   1.99 | 2007-01-24 21:40:19.996577
      16051 |         269 |        1 |        98 |   0.99 | 2007-01-25 15:16:50.996577
      16052 |         269 |        2 |       678 |   6.99 | 2007-01-28 21:44:14.996577
      16053 |         269 |        2 |       703 |   0.99 | 2007-01-29 00:58:02.996577
      16054 |         269 |        1 |       750 |   4.99 | 2007-01-29 08:10:06.996577
      16055 |         269 |        2 |      1099 |   2.99 | 2007-01-31 12:23:14.996577
      16056 |         270 |        1 |       193 |   1.99 | 2007-01-26 05:10:14.996577
:
```

条件を指定する - WHERE

さて、SELECTが使えるようになりテーブルからデータを抽出できるようになりました。

抽出したデータを眺めると感じるかもしれませんが、このままではデータが膨大であり、ほしい情報をピンポイントで見ることができません。ここではほしいデータのみ抽出できるWHEREについて学びます。

ノック02 paymentテーブルのcustomer_idが1のレコードを抽出する

まずは、次のクエリを実行してみてください。

```
SELECT
  payment_id,
  customer_id
FROM
  payment
WHERE
  customer_id = 1
;

payment_id | customer_id
-----------+-------------
     16677 |           1
     16678 |           1
     18495 |           1
(..略..)
(32 rows)
```

実行結果を見ると、customer_idが1のレコードで絞られていることがわかります。

続いて次も実行してみましょう。

customerテーブルの名前(first_name)がKELLYのレコードを抽出する

今度はcustomerテーブルも使ってみます。

```
SELECT
  first_name,
  last_name
FROM
  customer
WHERE
  first_name = 'KELLY'
;

first_name | last_name
-----------+-----------
KELLY      | TORRES
KELLY      | KNOTT
(2 rows)
```

first_nameがKELLYで絞られました。

演算子

WHEREではさまざまな演算子が利用できます。必須と言えるので押さえておきましょう。

▼表1　論理演算子

演算子	説明
条件1 AND 条件2	どちらの条件にも一致
条件1 OR 条件2	どちらかの条件に一致
NOT 条件	条件に一致しない

論理演算子

SQLでは**表1**のような論理演算子が使えます。実際にSQLを実行して結果を見ていきます。

customerテーブルから、名字(last_name)がKNOTTで名前がKELLYの顧客を抽出する

まずはANDを試してみます。

```
SELECT
  first_name,
  last_name
FROM
  customer
WHERE
  first_name = 'KELLY'
  AND last_name = 'KNOTT'
;

first_name | last_name
-----------+-----------
KELLY      | KNOTT
(1 row)
```

first_nameをKELLYとした場合は2名抽出されましたが、last_nameも指定することで1名に絞られました。

customerテーブルから、名前がKELLYもしくはMARIAの顧客を抽出する

次にORを試してみます。クエリの実行結果にMARIAとKELLYのレコードが含まれていることを確認してください。

```
SELECT
  first_name,
  last_name
FROM
  customer
WHERE
  first_name = 'KELLY'
  OR first_name = 'MARIA'
;

first_name | last_name
-----------+-----------
MARIA      | MILLER
KELLY      | TORRES
KELLY      | KNOTT
(3 rows)
```

ノック06 customerテーブルから、名前がKELLYやMARIA以外の顧客を抽出する

続いて、NOTを試します。

```
SELECT
  first_name,
  last_name
FROM
  customer
WHERE
  NOT (
    first_name = 'KELLY'
    OR first_name = 'MARIA'
  )
;

first_name  |  last_name
------------+-------------
MARY        | SMITH
PATRICIA    | JOHNSON
LINDA       | WILLIAMS
(..略..)
(596 rows)
```

NOTは条件の否定です。WHERE NOT(条件)とすることで、括弧内の条件を否定することができます。注意点は、括弧を付けることでNOTが影響する範囲を明確化している点です。括弧を付けなかった場合、演算子の優先順位によって思ったとおりの結果が得られないことがあります。今回のクエリも、括弧をはずしてしまうと「first_nameがKELLY以外、もしくはfirst_nameがMARIAの顧客」となってしまい期待した結果が得られません。試しに括弧をはずして実行してみてください。合計件数が変わってきます。

実行結果と照らし合わせることで理解が深まると思いますので、さまざまな条件でWHEREを書いてみてください。

▼表2　比較演算子

演算子	説明
A < B	AはBよりも小さい
A > B	AはBよりも大きい
A <= B	AはB以下
A >= B	AはB以上
A = B	AとBは等しい
A <> B	AとBは等しくない
A != B	

OR条件を、列挙して記述できるIN句

IN句を使うと、複数のOR条件をシンプルに記述することができます。

ノック07 customerテーブルから、名前がAARON、ADAM、ANNの顧客を抽出する

複数の値で抽出するにはWHERE カラム名 IN（カンマ区切りの値）と書きます。

```
SELECT
  first_name,
  last_name
FROM
  customer
WHERE
  first_name IN ('AARON', 'ADAM', 'ANN')
;

first_name | last_name
-----------+-----------
ANN        | EVANS
ADAM       | GOOCH
AARON      | SELBY
(3 rows)
```

無事、括弧の中で列挙した名前で絞り込めました。これだけでは「ORと大差ないな」、「ORだけ覚えればいいや」と感じるかもしれません。しかし、この括弧の中に、実はクエリを書くことができます。別のクエリを呼び出した結果を使って絞り込みをかけられるため、IN句はORに比べて複雑な条件を柔軟に扱える強みがあります。第4章の応用編ではIN（クエリ）を使ってデータを抽出する例を紹介しますので、楽しみに読み進めてください。

比較演算子

続いて比較演算子を見ていきます。先ほどの名前の抽出では=演算子を使いました。クエリでは一般的なプログラミング言語と同様、さまざまな比較演算子が使えます（**表2**）。とくに数値データ型のカラムで力を発揮しますので、

paymentテーブルを使って練習してみましょう。

paymentテーブルから、支払い額(amount)が6.99ドル以上のレコードを抽出する

次のクエリを実行してみましょう。カラム名 演算子 値 の形式で値の比較を行えます。

```
SELECT
  payment_id,
  amount
FROM
  payment
WHERE
  amount >= 6.99
;

payment_id | amount
-----------+--------
     16052 |   6.99
     16058 |   8.99
     16060 |   6.99
(..略..)
(2651 rows)
```

paymentテーブルから、支払い額(amount)が0.99ドル以外のレコードを抽出する

条件に一致しないレコードの抽出には!=演算子が使えます。

```
SELECT
  payment_id,
  amount
FROM
  payment
WHERE
  amount != 0.99
;

payment_id | amount
-----------+--------
     16050 |   1.99
     16052 |   6.99
     16054 |   4.99
(..略..)
(13070 rows)
```

NULL演算子

SQLにもNULLが存在します。

IS NULL

値のないフィールドを抽出する際には、IS NULLを使ってWHERE句を組み立てます。IS NULLは「NULLかどうか」を判定します。

rentalテーブルのreturn_dateがNULLのレコードを抽出する

次のクエリを実行してください。

```
SELECT
  rental_id,
  return_date
FROM
  rental
WHERE
  return_date IS NULL
;

rental_id | return_date
----------+-------------
    11496 |
    11541 |
    12101 |
(..略..)
(183 rows)
```

return_dateがNULLのものだけが抽出されていることがわかります。

IS NOT NULL

逆に「NULLでないか」の判定はIS NOT NULLを使います。

rentalテーブルのreturn_dateがNULLではないレコードを抽出する

次のクエリを実行してください。

```
SELECT
  rental_id,
  return_date
FROM
  rental
WHERE
  return_date IS NOT NULL
;
```

```
rental_id |     return_date
----------+---------------------
        2 | 2005-05-28 19:40:33
        3 | 2005-06-01 22:12:39
        4 | 2005-06-03 01:43:41
(..略..)
(15861 rows)
```

今度は、return_dateがNULL以外のレコードが抽出されていることがわかります。

BETWEEN演算子

範囲を指定する条件には、BETWEEN演算子が使えます。>=や<=で条件指定するよりも、簡便に読みやすく範囲条件を記述できます。

ノック12 customerテーブルから、顧客IDが11から13の顧客をBETWEENを使って抽出する

次のクエリを実行してください。

```
SELECT
  customer_id,
  first_name,
  last_name
FROM
  customer
WHERE
  customer_id BETWEEN 11 AND 13
;

customer_id | first_name | last_name
------------+------------+-----------
         11 | LISA       | ANDERSON
         12 | NANCY      | THOMAS
         13 | KAREN      | JACKSON
(3 rows)
```

BETWEEN a AND bという記法で、範囲が指定できます。BETWEENを使った場合、境界値が含まれることに注意してください。

LIKE演算子

あいまいな検索にはLIKE演算子を使います。

ノック13 filmテーブルのdescriptionにAmazingが含まれているレコードを抽出する

次のクエリを実行します。

```
SELECT
  title,
  description
FROM
  film
WHERE
  description LIKE '%Amazing%'
;

      title        |   description
-------------------+------------------
ANNIE IDENTITY     | A Amazing Panoram...
ANONYMOUS HUMAN    | A Amazing Reflect...
BRANNIGAN SUNRISE  | A Amazing Epistle...
(..略..)
(48 rows)
```

%は任意の文字列を表し、上の例だとフィールドのどこかにAmazingという文字列が含まれていれば抽出します。

それでは、先頭と末尾から%をそれぞれ取り除いた結果はどうでしょうか。

まずは先頭の%を取ってみます。

```
SELECT
  title,
  description
FROM
  film
WHERE
  description LIKE 'Amazing%'
;

title | description
------+-------------
(0 rows)
```

次に末尾の%を取ってみます。

```
SELECT
  title,
  description
FROM
  film
WHERE
  description LIKE '%Amazing'
;

title | description
------+-------------
(0 rows)
```

それぞれ、「Amazingから始まるdescription」と「Amazingで終わるdescription」が抽出されます。しかし、結果を見ると、Amazingから始

まるものも終わるものも含まれていないことがわかります。

ちなみに、否定の意味でNOT LIKEも使えるようになっておきましょう。

ノック14 filmテーブルのdescriptionにAmazingが含まれていないレコードを抽出する

次のように書きます。

```
SELECT
  title,
  description
FROM
  film
WHERE
  description NOT LIKE '%Amazing%';

title            |   description
-----------------+---------------------
ACADEMY DINOSAUR | A Epic Drama of a Fe...
ACE GOLDFINGER   | A Astounding Epistle...
(..略..)
(952 rows)
```

レコード総数を出す - COUNT

レコードの数を数えたいときにはCOUNTが使えます。

ノック15 paymentテーブルの総件数を求める

次のクエリを実行してください。

```
SELECT COUNT(*) FROM payment;

 count
-------
 16049
(1 row)
```

paymentテーブルには、16049レコードあることがわかります。COUNT(*)はすべてのレコード数を取得という意味です。

COUNT(customer_id)と書くとcustomer_idカラムのNULLではない件数を返してくれます。今回customer_idにはNULLが含まれていないため、COUNT(*)の結果と件数は変わりません。しかし、実際の業務で使うデータベースには、しばしばNULLが含まれるテーブルもあるので気をつけましょう。

COUNTのように「複数行の情報を集計し1行にまとめる」ものを「集計関数」と呼びます。合計・平均・最大値・最小値を求める集計関数もありますので、第3章で詳しく取り上げます。

重複を取り除く - DISTINCT

重複を取り除きたいときはDISTINCTを使います。たとえば、支払いを行ったユニークな顧客IDリストを作成する場合にはcustomer_idが同じ人を支払い数に関係なく1名と扱いたいです。

ノック16 paymentテーブルから、支払いを行ったユニークな顧客IDを求める

次のクエリを実行してください。

```
SELECT
  DISTINCT customer_id
FROM
  payment
;

customer_id
-------------
         251
         106
         120
(..略..)
(599 rows)
```

customer_idカラムを対象に重複を取り除いた結果なので、つまり支払いを行ったユニークな顧客IDとなります。

また、次のようにCOUNTと合わせて使うこともできます。

ノック17 paymentテーブルから、支払いを行ったユニークな顧客数を求める

次のクエリを実行してください。

```
SELECT
  COUNT(DISTINCT customer_id)
FROM
  payment
;

count
------
  599
(1 row)
```

このようにCOUNTとDISTINCTはユニークユーザを集計したいときに重宝するので合わせて覚えておきましょう。

順序を指定する - ORDER BY

ここまでのノックで、さまざまな条件でデータを抽出できるようになりました。次は抽出したデータを並び替える方法を学びましょう。

ノック18 customerテーブルから、顧客の名字を昇順に出力する

指定した順序によってレコードを抽出したい場合には、ORDER BYを使います。

```
SELECT
  customer_id,
  last_name
FROM
  customer
ORDER BY
  last_name
;

customer_id |  last_name
------------+--------------
        505 | ABNEY
        504 | ADAM
         36 | ADAMS
         96 | ALEXANDER
        470 | ALLARD
         27 | ALLEN
        220 | ALVAREZ
         11 | ANDERSON
        326 | ANDREW
        183 | ANDREWS
(..略..)
(599 rows)
```

ORDER BY last_nameと指定することで、昇順にデータを取得できました。ただし、同じ名字の顧客がいた場合にはどちらが先に表示されるかわかりません。「同じ名字の場合は、顧客IDが若い順に並べる」ように、その順序も指定する場合には、ORDER BY last_name, customer_idのようにカンマ区切りでカラム名を指定します。

では、降順に出力するにはどうすると良いでしょうか。

ノック19 customerテーブルから、最近登録された顧客を順に（customer_idの降順に）並べて出力する

降順で出力するには、カラム名のあとにDESCキーワード（DESCendent：降下する）を使います。

```
SELECT
  customer_id,
  first_name,
  last_name
FROM
  customer
ORDER BY
  customer_id DESC
;
```

```
customer_id | first_name  |  last_name
------------+-------------+-------------
        599 | AUSTIN      | CINTRON
        598 | WADE        | DELVALLE
        597 | FREDDIE     | DUGGAN
        596 | ENRIQUE     | FORSYTHE
(..略..)
(599 rows)
```

昇順を表すASCキーワード（ASCendant：上昇する）もありますので、ご自身で書き換えて実行し、確かめてみてください。ASCキーワードは省略可能ですが、明示的に記述することで、クエリの意図がより明確になります。

抽出したデータの件数を絞る場合はLIMITを使います。

customerテーブルから、最近登録された顧客3名を抽出する

次のクエリを実行します。

```
SELECT
  first_name,
  last_name
FROM
  customer
ORDER BY
  customer_id DESC
LIMIT 3
;

first_name | last_name
-----------+-----------
AUSTIN     | CINTRON
WADE       | DELVALLE
FREDDIE    | DUGGAN
(3 rows)
```

いかがでしょうか。LIMIT 件数とすることで、データの抽出件数を指定することができます。実務では「新規登録者N人にキャンペーンを打つ」ようなケースが頻繁にあると思いますので、繰り返しクエリを書いて練習しましょう。

SELECTするときはLIMITをつける癖をつけよう

今回の教材には含まれませんが、実際の業務では巨大なテーブルを取り扱うことも多いかと思います。たとえば、アクセスログを格納したテーブルや、あるテーブルのスナップショットを日々記録したテーブルなどは数十万から数億レコードに達することも珍しくありません。

このような巨大なテーブルに対して、次のように一覧表示するクエリを実行したとします。

```
SELECT
  *
FROM
  アクセスログ
ORDER BY
  更新日 DESC
;
```

このようなクエリを実行すると多くの場合、結果が出力されるまでに長い処理時間がかかってしまいます。この間はI/Oをはじめとする計算機資源を消費してしまっていますので、同時アクセスしているほかのクエリの実行も待たされてしまいます。つまり、ほかの人から見るとデータベースサーバが落ちたように感じられてしまいます。

LIMITを付けた場合は、付けない場合に比べてデータ転送時間を「LIMITで指定した件数 / 全レコード件数」に短くでき、こうしたトラブルを予防できますので、SELECTクエリには積極的にLIMITを付けるように心がけましょう。

同じく、SELECT *ではなくSELECT カラム名とするのも良い方法です。

```
SELECT
  カラム1, カラム2
FROM
  アクセスログ
ORDER BY
  更新日 DESC
LIMIT 10
;
```

もちろん、クエリの実行に処理時間がかからないようにするのもデータベースエンジニアの腕の見せどころではありますが、さまざまなクエリに対応できるわけではありません。

カラムの値ごとに集計する - GROUP BY

顧客の中からロイヤルカスタマーを抽出し、綿密な営業やプロモーションを行うことは、ビジネスの現場でよくあるのではないでしょうか。そんなときに役立つのが、GROUP BY句です。GROUP BY カラム名とすることで、特定の値ごとに集計を行えます。

ノック21 paymentテーブルから、これまでの累計で支払い回数が多い顧客の上位3人の顧客IDを抽出する

次のようなクエリで抽出できます。

```
SELECT
  customer_id,
  COUNT(*) AS payment_count
FROM
  payment
GROUP BY
  customer_id
ORDER BY
  payment_count DESC
LIMIT 3
;

customer_id | payment_count
------------+---------------
        148 |            46
        526 |            45
        236 |            42
(3 rows)
```

GROUP BY customer_idにより、顧客IDごとに取りまとめられます。そしてCOUNT(*)によって、顧客IDごとの支払い回数を集計できます。

集計した結果のカラム名は、AS句で設定できます。AS 名称とすると、集計結果カラムに名称を付与できます。何も設定せずともcountというデフォルト名称が付与されますが、COUNTを複数使ったときにみんな同じ名称になってしまいますので、ASを付与する癖をつけておきましょう。今回はpayment_countという名称をつけました。

支払い回数の多い順にソートするには、先ほど学んだORDER BYにpayment_countを指定します。ORDER BYをはじめとするカラム名を指定する句では、先ほどAS句で指定した別名を使うことになります。今回のクエリでは、ORDER BY payment_count DESCと指定することで支払い回数順にソートを行えます。

データ型と関数

これまで抽出したデータには数字や文字列、時間などが含まれていましたが、データベースにも型があります。データ型は、データベース固有のものもあるため、代表的なものを紹介して、データの型変換まで行ってみます。

データ型には、VARCHAR型などの文字列、INTEGER型などの数値、TIMESTAMP型などの日時を指定するものがあります。

データの保存時には型チェックが行われ、そのデータが妥当なものかどうか判定されます。

おもなデータ型は**表3**のとおりです。ほかのプログラミング言語では、データの型によって使える関数や演算子の振る舞いが違ったりしますが、それはSQLでも同様です。ここからしばらく、関数を使ったノックが続きますのでデータ型と関数の扱いに慣れていきましょう。

▼表3　データベースの代表的なデータ型

データ型	データ内容
CHARACTER	固定長文字列
VARCHAR	最大長付き可変長文字列
BINARY(n)	バイナリ文字列
BOOLEAN	真偽値
INTEGER	整数値
DECIMAL(p,s)	固定小数点数
NUMERIC(p,s)	
REAL	浮動小数点数
FLOAT	
DATE	日付(年、月、日)
TIME	時間(時、分、秒)
TIMESTAMP	日時(年、月、日、時、分、秒)

paymentテーブルの売上金額（amount）をドルから円に変換して、“109”のように小数点以下を四捨五入した形で抽出する（1ドル110円とする）

数値のデータ型であれば四則演算ができます。たいていのデータベースでは、テーブル定義を参照して、カラムのデータ型を調べることができます（調べ方は次頁のコラム「中身を知らないデータベースで分析する」をご覧ください）。

まずは、1ドル110円としてデータを抽出してみます。

```
SELECT
  amount * 110 AS amount_yen
FROM
  payment
LIMIT 3
;

amount_yen
-----------
    218.90
    108.90
    768.90
(3 rows)
```

ドルから円に変換できましたが、日本円は通常1円単位のため、小数値は使いません。小数を四捨五入するにはROUND()関数を使います。

SQLの関数は、カラムを引数にとり、ROUND(amount)のように使います。この関数は、引数にとったカラムのデータにそれぞれ作用します。今回は、円に変換したあとに四捨五入することにしましょう。

```
SELECT
  ROUND(amount * 110) AS amount_yen
FROM
  payment
LIMIT 3
;

amount_yen
-----------
       219
       109
       769
(3 rows)
```

paymentテーブルの売上金額（amount）をドルから円に変換して、“109yen”のように小数点以下を四捨五入し単位を付けて抽出する

最後に109yenのような形で出力することにしましょう。文字列を結合するにはCONCAT()関数を使います。

```
SELECT
  CONCAT(
    ROUND(amount * 110),
    'yen'
  ) AS amount_yen
FROM
  payment
LIMIT 3
;

amount_yen
-----------
219yen
109yen
769yen
(3 rows)
```

目的のデータを抽出できました。しかし、CONCAT()関数が数値と文字列をよしなに結合したので良かったものの、明示的にデータの型を変換しなければならないケースもあります。

その場合はCAST()関数を使います。CAST(変換したいカラム AS 変換後のデータ型)のように書きます。

```
SELECT
  CONCAT(
    CAST(ROUND(amount * 110) AS VARCHAR),
    'yen'
  ) AS amount_yen
FROM
  payment
LIMIT 3
;
(実行結果は省略)
```

少々複雑になってしまいましたが、ここで扱った内容を理解していると、クエリでさまざまなデータ加工ができるようになります。CASTは次の中級編でも詳しく解説します。SD

column

中身を知らないデータベースで分析する

本特集を読んで、自分が携わる事業やサービスのデータを分析しようとしたときに、どんなテーブルやカラムがあるかがわからなくて、どこから分析をしたら良いか困ってしまうことがあるかもしれません。

テーブル定義のドキュメントが読めると、どんなデータが保存されているかが、ビジネスやサービス内容と絡めて理解できるので理想的です。

ただし、度重なるシステム改修によってドキュメントの更新が追いついていなかったり、そういったドキュメントが準備されていない、といったこともあるでしょう。そうした場合は、データベースのテーブルの定義を調べるところから始めると良いでしょう。

今回の題材であるPagilaで試してみます。まずは、クエリを使って対象のデータベースのスキーマを見てみます。

```
SELECT
  DISTINCT table_schema
FROM
  information_schema.columns
;

    table_schema
--------------------
 information_schema
 pg_catalog
 public
(3 rows)
```

結果を見るとpublicスキーマというものがあります。Pagilaのデータはpublicスキーマの中に入っています。では、publicスキーマに所属しているテーブルを見てみましょう。

```
SELECT
  DISTINCT table_name
FROM
  information_schema.columns
WHERE
  table_schema = 'public'
ORDER BY
  table_name
;
```

```
         table_name
----------------------------
 actor
 actor_info
 address
 category
 city
 country
 customer
 customer_list
 film
 film_actor
 film_category
 film_list
 inventory
(..略..)
(28 rows)
```

このテーブル一覧を元に、テーブル名から中身を推測しつつ、テーブルのカラム定義を見ていきます。たとえば、今回のノックでもよく使われたcustomerテーブルを調べてみます。次のクエリを実行すると、

```
SELECT
  column_name,
  data_type,
  column_default,
  is_nullable
FROM
  information_schema.columns
WHERE
  table_schema = 'public'
  AND table_name='customer'
;
```

カラム名、データ型、カラムのデフォルト値、NULL許容について知ることができます（**図3**）。

この結果を踏まえて、データをざっと把握するためにLIMITを付けてクエリを投げてみます。カラムの定義と合わせて、データの中身を見ることで、ざっくりとしたデータの偏りや出現頻度を把握することができます。

```
SELECT * FROM customer LIMIT 100;
(実行結果は省略)
```

また、paymentテーブルとcustomerテーブルのように、customer_idカラムで関連している場合があります。そういったテーブル同士の関連は、外部キー（FOREIGN KEY）で定義されていることが多いです。外部キーは**図4**のようにして調べられます[注A]。

これらのテーブル情報を元に、さまざまな観点から分析をしていけることでしょう。

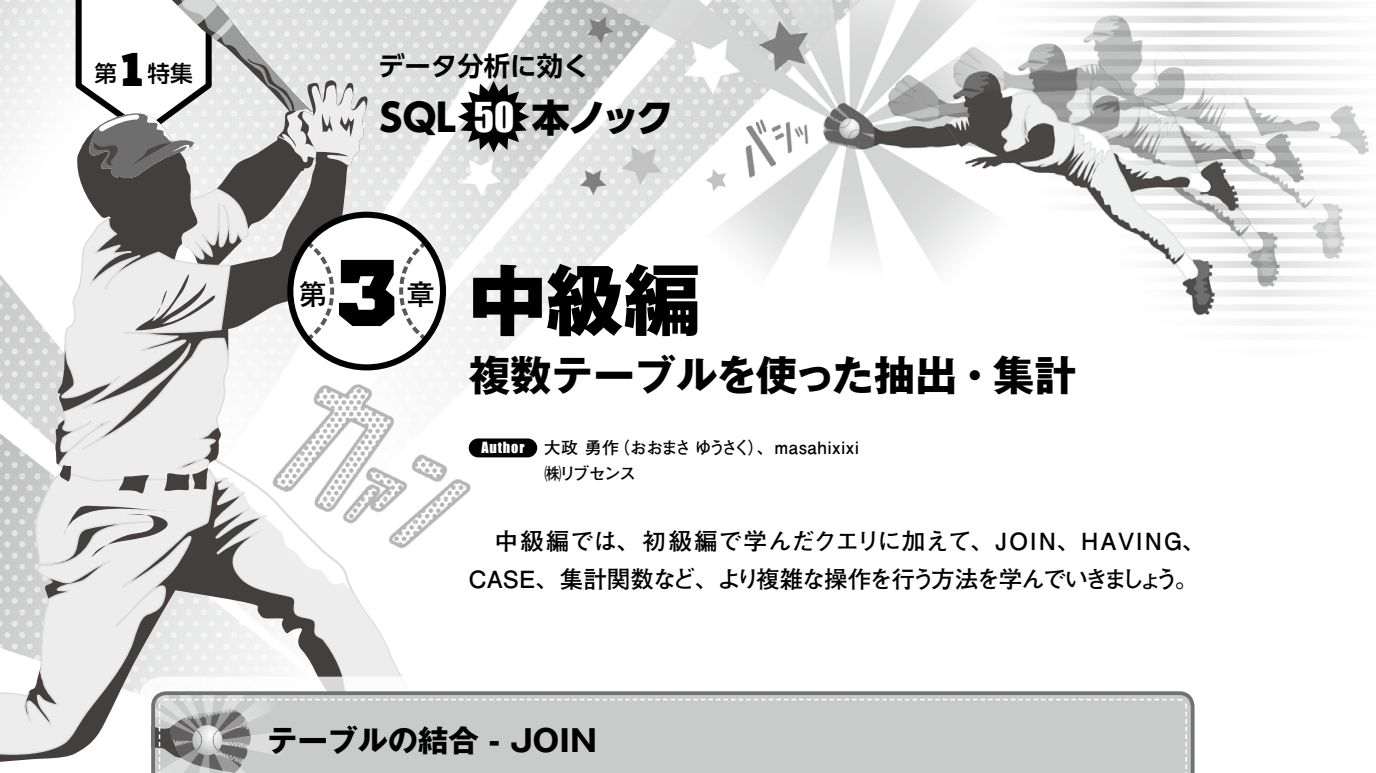
▼図3　customerテーブルのカラム定義の出力結果

```
column_name |          data_type          |                column_default                | is_nullable
------------+-----------------------------+----------------------------------------------+-------------
customer_id | integer                     | nextval('customer_customer_id_seq'::regclass) | NO
store_id    | smallint                    |                                              | NO
first_name  | character varying           |                                              | NO
last_name   | character varying           |                                              | NO
email       | character varying           |                                              | YES
address_id  | smallint                    |                                              | NO
activebool  | boolean                     | true                                         | NO
create_date | date                        | ('now'::text)::date                          | NO
last_update | timestamp without time zone | now()                                        | YES
active      | integer                     |                                              | YES
(10 rows)
```

▼図4　外部キーを調べる

```
SELECT
  k1.table_name AS fk_table,
  k1.column_name AS fk_column,
  k2.table_name AS ref_table,
  k2.column_name AS ref_column
FROM
  information_schema.referential_constraints AS rc
  INNER JOIN information_schema.key_column_usage AS k1
    USING(
      constraint_catalog,
      constraint_schema,
      constraint_name
    )
  INNER JOIN information_schema.key_column_usage AS k2
    ON k2.constraint_catalog = rc.unique_constraint_catalog
      AND k2.constraint_schema = rc.unique_constraint_schema
      AND k2.constraint_name = rc.unique_constraint_name
      AND k2.ordinal_position = k1.ordinal_position
;

   fk_table       |    fk_column     | ref_table |  ref_column
------------------+------------------+-----------+--------------
 inventory        | film_id          | film      | film_id
 film_category    | film_id          | film      | film_id
 film_actor       | film_id          | film      | film_id
 rental           | inventory_id     | inventory | inventory_id
 store            | manager_staff_id | staff     | staff_id
 rental           | staff_id         | staff     | staff_id
 payment          | staff_id         | staff     | staff_id
 payment_p2007_06 | staff_id         | staff     | staff_id
(..略..)
(40 rows)
```

注A）図4のクエリの中に出てくるJOINやUSINGなどについては、中級編以降で解説します。



第3章 中級編
複数テーブルを使った抽出・集計

Author 大政 勇作（おおまさ ゆうさく）、masahixixi
㈱リブセンス

中級編では、初級編で学んだクエリに加えて、JOIN、HAVING、CASE、集計関数など、より複雑な操作を行う方法を学んでいきましょう。

テーブルの結合 - JOIN

paymentテーブルとcustomerテーブルは、ともにcustomer_idカラムを持っているため、2つのテーブルのデータがひも付きます。このひも付きを使って、paymentテーブルをcustomerテーブルの顧客氏名とともに表示させたい場合は、テーブルとテーブルを結合するJOINを使います。

テーブル同士の結合にはいくつか種類がありますが（**図1**）、ここではとくに利用頻度の高いLEFT JOINとINNER JOINについて扱います。

▼図1　JOINの種類

JOINの種類	イメージ
INNER JOIN	左テーブル ○ 右テーブル
LEFT OUTER JOIN	左テーブル ◐ 右テーブル
RIGHT OUTER JOIN	左テーブル ◑ 右テーブル
FULL OUTER JOIN	左テーブル ● 右テーブル
CROSS JOIN	左テーブル × 右テーブル

LEFT JOIN

paymentテーブルにcustomer_idでひも付くcustomerテーブルを結合し、payment_id、last_name、first_nameカラムのデータを抽出する

paymentテーブルを元に、customerテーブルの一致するデータを結合して抽出するには、LEFT JOINを使います[注1]。

なぜこんなにJOINの種類があるかというと、2つのテーブルの間に結合できないデータがあった場合の取り扱いが異なるからです。今回使っているPagilaのデータではJOINの違いがわかりにくいため、説明では一部データが欠損した、不完全なデータを使います。paymentおよびcustomerテーブルの抜粋を用意しましたので、データベースの中にこのデータが入っていると思って読み進めてください（**表1**、**表2**）。

LEFT JOINは日本語で左外部結合といい、左側のテーブルを元に右側のテーブルを結合します。今回のノックではpaymentテーブルを優先して結合したいため、FROM句にpaymentテー

注1）LEFT JOINやRIGHT JOINは外部結合と呼ばれ、LEFT OUTER JOINのように、OUTERを明示的に書くこともできます。

ブルを指定します。テーブルの結合条件を指定する場合は、ONのあとに条件を書きます。

```
SELECT
  payment_id,
  first_name,
  last_name
FROM
  payment
  LEFT JOIN customer
    ON payment.customer_id
      = customer.customer_id
;
 payment_id | first_name | last_name
------------+------------+-----------
      31917 | MARGIE     | WADE
      31918 | MARGIE     | WADE
      31919 | CASSANDRA  | WALTERS
      31921 | NAOMI      | JENNINGS
      31922 |            |
```

LEFT JOINは、paymentテーブルとcustomerテーブルのデータがひも付かない場合でも、paymentテーブルのデータは優先されてすべて抽出されます。customerテーブル側にひも付くcustomer_idがなかった場合は、データにNULLが入ります（**図2**）。

なお、結合に使うカラム名が2つのテーブルで同じ場合、USING(custom_id)と短い表現で書くこともできます。

```
SELECT
  payment_id,
  first_name,
  last_name
FROM
  payment
  LEFT JOIN customer
    USING(customer_id)
;
```
（実行結果は省略）

INNER JOIN

paymentテーブルとcustomerテーブルで、データのひも付きがあるものだけを抽出するには、INNER JOINを使います。

```
SELECT
  payment_id,
  first_name,
  last_name
FROM
  payment
  INNER JOIN customer
    ON payment.customer_id
      = customer.customer_id
;
 payment_id | first_name | last_name
------------+------------+-----------
      31917 | MARGIE     | WADE
      31918 | MARGIE     | WADE
      31919 | CASSANDRA  | WALTERS
      31921 | NAOMI      | JENNINGS
```

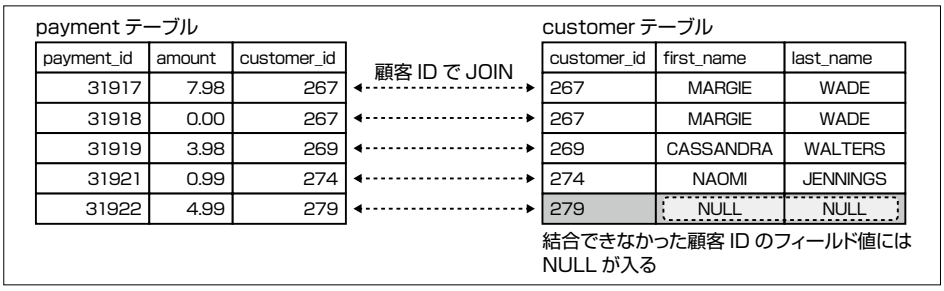
▼表1　paymentテーブルの抜粋

payment_id	customer_id	amount
31917	267	7.98
31918	267	0.00
31919	269	3.98
31921	274	0.99
31922	279	4.99

▼表2　customerテーブルの抜粋

customer_id	first_name	last_name
267	MARGIE	WADE
269	CASSANDRA	WALTERS
274	NAOMI	JENNINGS
275	CAROLE	BARNETT

▼図2　LEFT OUTER JOIN



INNER JOINは、2つのテーブルで結合できるデータのみに絞り込まれるため、WHERE句のような働きも併せ持ちます。今回のサンプルでは、paymentテーブルにあったcustomer_id = 279のデータがINNER JOINによって除外されました（**図3**）。

以降では再度、Docker環境のPagilaのデータを使ってクエリを実行しながら進めます。

ノック25 paymentテーブルから、顧客名がBRIAN WYMANの支払いデータを抽出する

JOINで結合したテーブルは、WHEREやGROUP BYなどで使うことができます。このノックでは顧客名で絞り込みをしたいので、WHERE句でfirst_nameとlast_nameをAND条件で指定します。カラムは、payment_id、customer_id、amountを取り出すことにしましょう。

```
SELECT
  payment_id,
  customer_id,
  amount
FROM
  payment
  INNER JOIN customer
    ON payment.customer_id
      = customer.customer_id
WHERE
  first_name = 'BRIAN'
  AND last_name = 'WYMAN'
;

-- ERROR:  column reference "customer_id" is ambiguous
-- LINE 1: SELECT payment_id, customer_id, amount
```

おかしいですね、エラーになってしまいました。このエラーはSELECTで指定したcustomer_idが、paymentテーブルのものかcustomerテーブルのものかが判別できないために発生したものです。

どのテーブルのものか明示するためには、テーブル名.カラム名のようにテーブル名をドットでつなぎ、指定します。ここでは、payment.customer_idという形で明示しましょう。

```
SELECT
  payment_id,
  payment.customer_id,
  amount
FROM
  payment
  INNER JOIN customer
    ON payment.customer_id
      = customer.customer_id
WHERE
  first_name = 'BRIAN'
  AND last_name = 'WYMAN'
;

 payment_id | customer_id | amount
------------+-------------+--------
      16160 |         318 |   9.99
      17400 |         318 |   2.99
      17401 |         318 |   2.99
      17402 |         318 |   0.99
      17403 |         318 |   7.99
      20001 |         318 |   2.99
      25744 |         318 |   4.99
      25745 |         318 |   2.99
      25746 |         318 |   8.99
      25747 |         318 |   0.99
      25748 |         318 |   0.99
      25749 |         318 |   5.99
(12 rows)
```

念のため、customer_id = 318がBRIAN WYMANであることを確認しましょう。

▼図3　INNER JOIN

paymentテーブル

payment_id	amount	customer_id
31917	7.98	267
31918	0.00	267
31919	3.98	269
31921	0.99	274
31922	4.99	279

顧客IDでJOIN

customerテーブル

customer_id	first_name	last_name
267	MARGIE	WADE
267	MARGIE	WADE
269	CASSANDRA	WALTERS
274	NAOMI	JENNINGS

× 結合できなかった顧客IDのレコードは含まれない

```
SELECT
  customer_id, first_name, last_name
FROM
  customer
WHERE
  customer_id = 318;

 customer_id | first_name | last_name
-------------+------------+-----------
         318 | BRIAN      | WYMAN
(1 row)
```

customer_id = 318はBRIAN WYMANであることがわかります。また、先ほど書いたクエリを、

```
SELECT
  payment_id,
  payment.customer_id,
  customer.first_name,
  customer.last_name,
  amount
FROM
  payment
  INNER JOIN customer
    ON payment.customer_id
      = customer.customer_id
WHERE
  first_name = 'BRIAN'
  AND last_name = 'WYMAN'
;
(実行結果は省略)
```

のように書き換えても確認できますので、ぜひこちらも手元で実行してみてください。

テーブル名が長い場合は、カラム名と同様にASを使ってテーブルに別名を付けることで短くできます。

```
SELECT
  payment_id,
  p.customer_id,
  amount
FROM
  payment AS p
  INNER JOIN customer AS c
    ON p.customer_id = c.customer_id
WHERE
  first_name = 'BRIAN'
  AND last_name = 'WYMAN'
;
(実行結果は省略)
```

これでデータを抽出できました。なおUSINGを使った場合は、どちらのテーブルのカラムであるかの明示は不要です。

```
SELECT
  payment_id,
  customer_id,
  amount
FROM
  payment
  INNER JOIN customer USING(customer_id)
WHERE
  first_name = 'BRIAN'
  AND last_name = 'WYMAN'
;
(実行結果は省略)
```

これらのJOINに慣れてきたら、RIGHT JOINといった、ほかの結合方法も試してみてください。

GROUP BYした結果で絞り込む - HAVING

ここでは、filmテーブル、film_categoryテーブル、categoryテーブルを使い、HAVINGについて学んでいきます。HAVINGを使うと、GROUP BYでの結果に対して、さらに条件を付けて絞り込むことができます。初級編で学習したとおり、GROUP BYを使うとカラムの値ごとに集計できます。

filmテーブルは映画情報が格納されたテーブルです。映画ID（film_id）、タイトル（title）、説明（description）、リリース年（release_year）などが格納されています。

film_categoryテーブルには映画ID（film_id）、カテゴリID（category_id）などが入っています。

categoryテーブルにはカテゴリID（category_id）、カテゴリ名（name）などが入っています。

ノック26 filmテーブル・film_categoryテーブル・categoryテーブルから、カテゴリ名ごとに映画の作品数を集計し、65件以上のものを抽出する

先にクエリと実行結果を見ていきましょう。

```
SELECT
  category.name AS name,
  COUNT(category.name) AS film_cnt
FROM
  film
  INNER JOIN film_category
    USING(film_id)
  INNER JOIN category
    USING(category_id)
GROUP BY
  category.name
HAVING
  COUNT(category.name) >= 65
ORDER BY
  film_cnt DESC
;

    name     | film_cnt
-------------+----------
 Sports      |       74
 Foreign     |       73
 Family      |       69
 Documentary |       68
 Animation   |       66
(5 rows)
```

GROUP BYを使ってカテゴリ名ごとに集計したうえで、HAVING COUNT〜の部分で条件に合うレコードを抽出しているように見えます。

ここまで進めてきた方であれば、WHEREとどう違うのかという疑問が浮かんでくると思います。まずはその違いについて解説します。

HAVINGとWHEREの違いについて

適用される順番が違うのがポイントです。クエリはWHERE→GROUP BY→HAVINGという順に適用されていきます。WHEREはGROUP BYの前、HAVINGはGROUP BYの後と覚えておきましょう。

WHEREを用いると、適用順が後であるGROUP BYで集計した結果に対して絞り込むことができません。一方で、HAVINGを用いればGROUP BYでの集計結果を絞り込むことができます。

いかがでしょうか。ちなみに、1つのクエリ内で両方を使用することもできます。

```
SELECT
  category.name AS name,
  COUNT(category.name) AS film_cnt
FROM
  film
  INNER JOIN film_category
    USING(film_id)
  INNER JOIN category
    USING(category_id)
WHERE                          ┐
  category.name IN(            │
    'Sports',                  ├ WHERE句
    'Games',                   │
    'Travel'                   │
  )                            ┘
GROUP BY
  category.name
HAVING                         ┐
  COUNT(category.name) > 60    ┘ HAVING句
ORDER BY
  film_cnt DESC
;

  name  | film_cnt
--------+----------
 Sports |       74
 Games  |       61
(2 rows)
```

上のクエリで、次の3つのケースをそれぞれ実行してみてください。違いが実感できると思います。

・全体を実行（上記のクエリ）
・HAVING句を削って実行
・WHERE句を削って実行

なお、SELECTクエリの評価順序は次のとおりです。

(1) FROM
(2) JOIN
(3) WHERE
(4) GROUP BY
(5) HAVING
(6) SELECT
(7) ORDER BY
(8) LIMIT

条件分岐 - CASE

プログラミングにおける条件分岐にあたるものがCASEです。長くなりがちで多少読みづらさがありますが、単に「○○という結果のとき××と表示するもの」と覚えておくと良いです。

paymentテーブルで、支払い額が5を超える場合はexpensive、1を超える場合はmodest、そうでなければcheapとして一覧表示する

それではCASEを使ったクエリを実行してみましょう。CASE WHEN 条件1 THEN 値1 ELSE デフォルト END値という構文で、条件ごとに値を振り分けることができます。

```
SELECT
  payment_id,
  amount,
  CASE
    WHEN amount > 5 THEN 'expensive'
    WHEN amount > 1 THEN 'modest'
    ELSE 'cheap'
  END AS price_range
FROM
  payment
;

 payment_id | amount | price_range
------------+--------+-------------
      16050 |   1.99 | modest
      16051 |   0.99 | cheap
      16052 |   6.99 | expensive
(..略..)
(16049 rows)
```

POSIX正規表現

SQLでは正規表現を扱うこともできます。こちらに関してはPostgreSQL、MySQLでそれぞれ表現方法が違うので、MySQLでの表現方法についても軽く触れておきたいと思います。

filmテーブルのdescriptionに入っているThoughtfulまたはInsightfulの数を求める

先にクエリを実行してみましょう。

```
SELECT
  COUNT(*)
FROM
  film
WHERE
  description ~ '(Thou|Insi)ghtful'
;

 count
-------
    91
(1 row)
```

正規表現のマッチ演算子は**表3**のとおりです。ここで紹介したのはPostgreSQLの正規表現ですが、MySQLでは次のように書けます。

```
SELECT
  COUNT(*)
FROM
  film
WHERE
  description REGEXP '(Thou|Insi)ghtful'
;
(実行結果は省略)
```

どちらの環境にも対応できるよう、それぞれ覚えておくと良いと思います。

▼表3　PostgreSQLの正規表現

演算子	説明
~	正規表現に一致し、大文字小文字を区別する
~*	正規表現に一致し、大文字小文字を区別しない
!~	正規表現に一致しない、大文字小文字を区別する
!~*	正規表現に一致しない、大文字小文字を区別しない

集計関数

ここからは、集計で使われるクエリを学んでいきましょう。初級編ではCOUNTを使いましたが、これはデータの個数を数えるものでした。たとえば売上の合計金額を知りたい場合は、SUM関数を使ってamountの値をすべて足し上げます。

```
SELECT
  SUM(amount) AS total_sales
FROM
  payment
;

 total_sales
-------------
    67416.51
(1 row)
```

COUNTやSUMは、値の集まりを1つに集計するため、集計関数とも呼ばれます。

GROUP BYと組み合わせる

集計関数はGROUP BYと組み合わせることで、クエリの表現力が一気に広がります。たとえば、顧客ごとの売上の金額を知りたい場合は次のように書きます。

```
SELECT
  customer_id,
  SUM(amount) AS total_sales
FROM
  payment
GROUP BY
  customer_id
;

 customer_id | total_sales
-------------+-------------
         251 |      120.69
         106 |      100.77
         120 |      143.68
         285 |      135.74
         264 |       98.75
         497 |      129.72
         452 |      107.68
         496 |       88.79
(..略..)
(599 rows)
```

▼表4　集計関数

関数	集計処理
COUNT	レコードの件数
SUM	合計
AVG	平均値
MAX	最大値
MIN	最小値

お店ごとであったり、日付ごとであったり、クエリのGROUP BYの指定を変えるだけで、さまざまな観点で集計をすることができます。おもな集計関数には**表4**のものがあります。

ノック29　paymentテーブルから、支払い額上位5名の顧客データを抽出する

SUMなどの関数の処理結果は、SELECTの結果だけではなく、WHERE句やORDER BYの中でも使うことができます。

customer_idをGROUP BYでまとめて、ORDER BY SUM(amount) DESCとすると、支払い額の降順で顧客情報を並べることができます。

```
SELECT
  customer_id,
  SUM(amount) AS total_sales
FROM
  payment
GROUP BY
  customer_id
ORDER BY
  total_sales DESC
LIMIT 5
;

 customer_id | total_sales
-------------+-------------
         526 |      221.55
         148 |      216.54
         144 |      195.58
         137 |      194.61
         178 |      194.61
(5 rows)
```

日付の条件指定

中級編の最後では、日別や月別といったデータ集計の方法を見ていきます。

日付型に変更する - CAST

ノック30

paymentテーブルから、日付ごとの売上金額を集計する

では、日付ごとの売上を見ていきましょう。paymentテーブルのpayment_dateに日時が入っているので、GROUP BYで指定してみます。

```
SELECT
  payment_date,
  SUM(amount) AS total_sales
FROM
  payment
GROUP BY
  payment_date
ORDER BY
  payment_date
;

        payment_date         | total_sales
-----------------------------+-------------
 2007-01-24 21:21:56.996577 |        2.99
 2007-01-24 21:22:59.996577 |        2.99
 2007-01-24 21:32:05.996577 |        3.99
 2007-01-24 21:33:07.996577 |        4.99
 2007-01-24 21:33:47.996577 |        6.99
 2007-01-24 21:36:33.996577 |        0.99
 2007-01-24 21:40:19.996577 |        1.99
 2007-01-24 22:00:12.996577 |        4.99
 2007-01-24 22:29:06.996577 |        4.99
(..略..)
(15817 rows)
```

payment_dateがtimestamp型のため、日付ごとではうまく集計できませんでした。

timestampからdateのようにデータの型を変更するにはCASTを使います。初級編の最後で説明したように、CAST(変換したいカラム AS 変換後のデータ型)という形式で型を変換できますので、payment_dateをdate型に変換するように書き換えます。

```
SELECT
  CAST(payment_date AS DATE) AS p_date,
  SUM(amount) AS total_sales
FROM
  payment
GROUP BY
  p_date
ORDER BY
  p_date
;

   p_date   | total_sales
------------+-------------
 2007-01-24 |       86.81
 2007-01-25 |      568.61
 2007-01-26 |      743.30
 2007-01-27 |      708.27
 2007-01-28 |      793.10
(..略..)
(40 rows)
```

CASTは整数を文字列に変換したり、整数を小数に変換したりする場合にも使うことができます。データ型に関しては、RDBMSによって種類や使える関数が異なるためご注意ください。

日付から月だけ抽出 - EXTRACT

ノック31

paymentテーブルから、月別の売上金額を集計する

先ほどは日別の集計でしたが、次は月別の集計をしてみます。日付から特定の部分だけを取り出すにはEXTRACTを使用します。

```
SELECT
  EXTRACT(MONTH FROM payment_date)
    AS p_month,
  SUM(amount) AS total_sales
FROM
  payment
GROUP BY
  p_month
ORDER BY
  p_month
;
```

```
 p_month | total_sales
---------+-------------
       1 |     4824.43
       2 |     9631.88
       3 |    23886.56
       4 |    28559.46
       5 |      514.18
(5 rows)
```

ただし、この集計の方法は誤解を招きやすいものになっています。取り出した1月が去年なのか今年なのかがわかりません。あらためて年、月の2項目を使って集計してみましょう。

```
SELECT
  EXTRACT(YEAR FROM payment_date) AS yyyy,
  EXTRACT(MONTH FROM payment_date) AS mm,
  SUM(amount) AS total_sales
FROM
  payment
GROUP BY
  yyyy,
  mm
ORDER BY
  mm
;

yyyy | mm | total_sales
-----+----+-------------
2007 |  1 |     4824.43
2007 |  2 |     9631.88
2007 |  3 |    23886.56
2007 |  4 |    28559.46
2007 |  5 |      514.18
(5 rows)
```

今回のデータでは問題なかったのですが、1年分以上のデータが蓄積されているデータベースの場合、月だけで集計すると別の年のデータ（たとえば2007年1月と2006年1月のデータ）も集計してしまう恐れがあります。先ほどのようなミスで意思決定を間違えないように、日頃から数値感の把握をしたり、複数の書き方でクエリを書いて結果のダブルチェックをしてみたりするといった工夫ができれば理想的です。また、こうした問題は表計算ソフトなどを使ってグラフにしてみると、気付けることも多いです。

EXTRACTを使わずに年、月を取り出すにはどうしたら良いでしょうか。日付情報を文字列として扱って、左から7文字を取得すると、yyyy-mmの形でデータを取得できます[注2]。

```
SELECT
  LEFT(
    CAST(payment_date AS VARCHAR),
    7
  ) AS yyyymm,
  SUM(amount) AS total_sales
FROM
  payment
GROUP BY
  yyyymm
ORDER BY
  yyyymm
;

 yyyymm  | total_sales
---------+-------------
 2007-01 |     4824.43
 2007-02 |     9631.88
 2007-03 |    23886.56
 2007-04 |    28559.46
 2007-05 |      514.18
(5 rows)
```

LEFT(文字列 , 文字数)は、「文字列」から「文字数分」の文字を左から切り出す関数です。この関数を使うことで、timestamp型のpayment_dateの値を文字列変換し、2007-01のような文字列を作ることができます。せっかく日付用の型でデータが格納されているのに文字列として操作するという、少々乱暴な書き方ではありますが、この方法でもEXTRACTと同様の結果を得ることができました。

このほかにも、GROUP BYを使ったときの合計値と、使わなかったときの結果が一致するかなど、さまざまなチェック方法が考えられます。

日付の条件指定

paymentテーブルから、2007年1月の売上データを抽出する

次は日付の条件指定の書き方です。初級編で説明したように、WHERE句に1月の開始日と終

注2）データベースエンジンによっては、TO_CHARやCONVERTなどの、日付情報をフォーマット指定して文字列で取り出す関数が提供されています。

了日をAND条件で指定してみます。

```
SELECT
  SUM(amount) AS total_sales
FROM
  payment
WHERE
  payment_date >= '2007-01-01'
  AND payment_date <= '2007-01-31'
;

total_sales
-------------
    4177.92
(1 row)
```

おや、おかしいですね。月別で集計した結果(4824.43)よりも売上が少なくなってしまいました。これはクエリの学び始めで誰しもがつまづくポイントなのですが、実は終了日の指定のしかたに問題があります。

payment_date <= '2007-01-31'と書くと、「2007-01-31 00:00:00」までのデータが取得されます。つまり、「2007-01-31」のほとんどのデータが対象外になってしまうのです。

日付の範囲指定を正しく書くにはいくつか方法があります。代表的なものとしては、条件を「2007-02-01」未満にする方法と、payment_dateをCASTしてdate型で扱う方法です(**図4**)。

2007年1月の売上データは次の3通りの方法で表現できます。

①payment_date >= '2007-01-01' AND payment_date < '2007-02-01'
②payment_date >= '2007-01-01' AND CAST(payment_date AS DATE) <= '2007-01-31'
③CAST(payment_date AS DATE) BETWEEN '2007-01-01' AND '2007-01-31'

③を使ったクエリは次のとおりです。

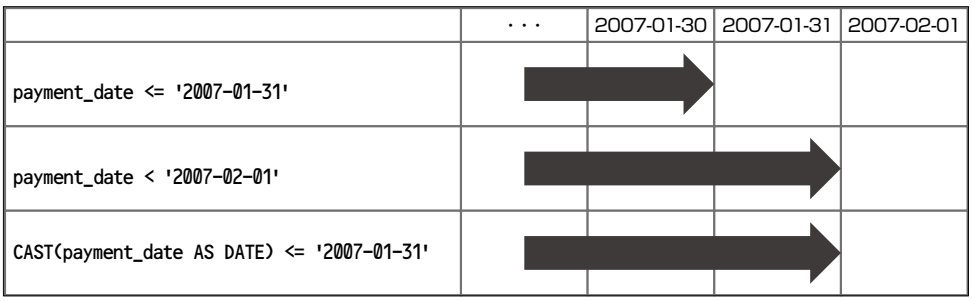
```
SELECT
  SUM(amount) AS total_sales
FROM
  payment
WHERE
  CAST(payment_date AS DATE)
    BETWEEN '2007-01-01' AND '2007-01-31'
;

total_sales
-------------
    4824.43
(1 row)
```

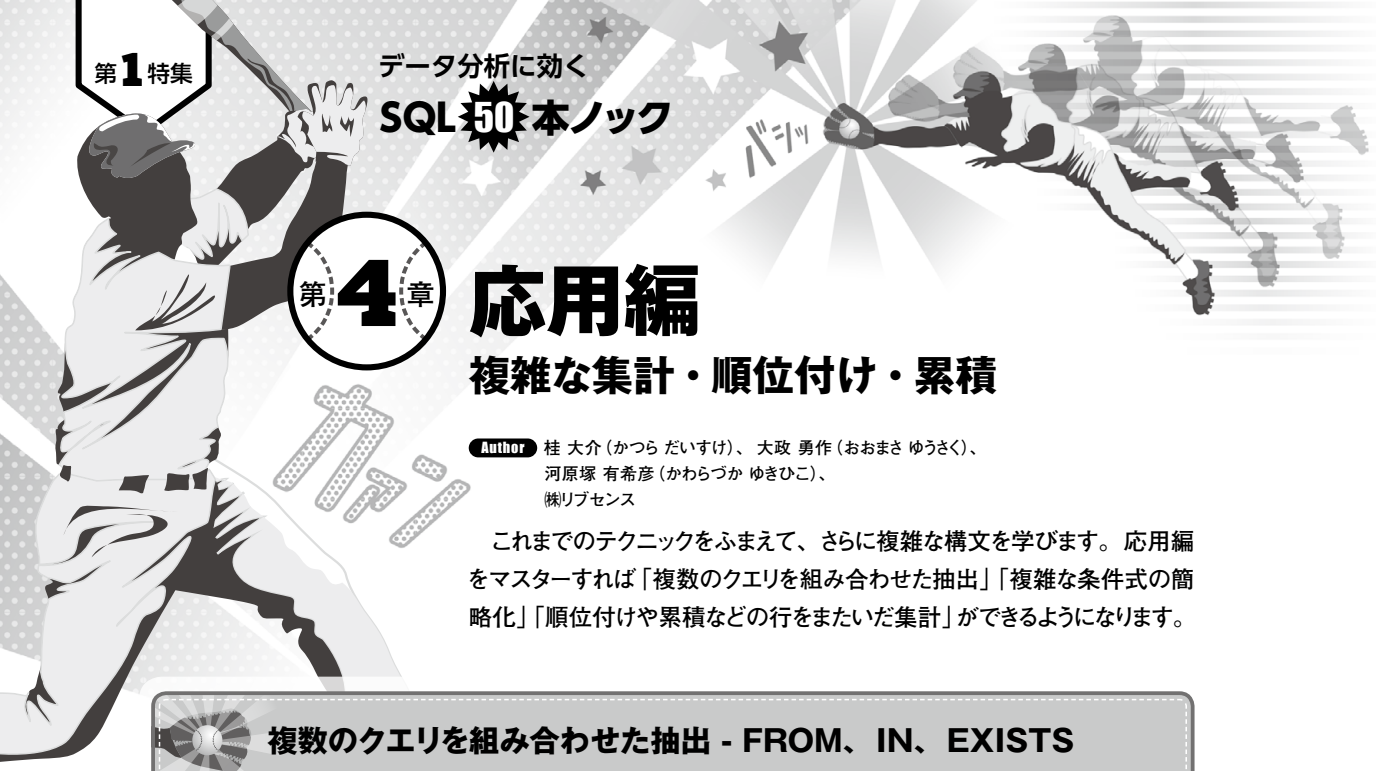
今回のクエリは、無事結果が一致しました。もちろん、先ほど使ったEXTRACTを使って年、月が一致するように条件指定しても大丈夫です。

```
SELECT
  SUM(amount) AS total_sales
FROM
  payment
WHERE
  EXTRACT(YEAR FROM payment_date) = 2007
  AND EXTRACT(MONTH FROM payment_date) = 1
;

total_sales
-------------
    4824.43
(1 row)
```

ただし、こちらの書き方は月初から月末までのデータを見たいときにしか使えないので、先ほどの書き方を覚えておくと応用しやすいです。SD

▼図4　日付の範囲指定





第4章 応用編
複雑な集計・順位付け・累積

Author 桂 大介（かつら だいすけ）、大政 勇作（おおまさ ゆうさく）、
河原塚 有希彦（かわらづか ゆきひこ）、
㈱リブセンス

これまでのテクニックをふまえて、さらに複雑な構文を学びます。応用編をマスターすれば「複数のクエリを組み合わせた抽出」「複雑な条件式の簡略化」「順位付けや累積などの行をまたいだ集計」ができるようになります。

複数のクエリを組み合わせた抽出 - FROM、IN、EXISTS

「集計処理を行った結果に対して、さらに集計処理を行う」ような、複数のクエリを組み合わせた抽出を行う場合には「サブクエリ」というしくみを利用します。

別のクエリ結果をテーブルとして使う - FROM

サブクエリで抽出した結果をテーブルとして使いFROMに引き渡すと、クエリ結果に対する集計を行えます。文章にすると難しそうに感じますが、「大きな問題を小さな問題に分割して解いていく」というイメージさえ持てれば解けたも同然です。

ノック33 paymentテーブルから顧客IDごとに累計売上を合計し、1顧客あたりの平均売上、最低売上、最高売上を求める

さっそく今回のノックを小さな問題に分割して解いていきましょう。

①「**顧客IDごとに累計売上を合計し……**」。ここまでは、初級編で解説した内容の復習で解けます。GROUP BY customer_idとすることで、顧客IDごとの集計を行うことができましたね。合計値には、何の合計値かわかるようにtotal_salesと名付けましょう。

```
SELECT
  customer_id,
  SUM(amount) AS total_sales
FROM
  payment
GROUP BY customer_id;

 customer_id | total_sales
-------------+-------------
         251 |      120.69
         106 |      100.77
         120 |      143.68
(..略..)
(599 rows)
```

②「**1顧客あたりの平均売上、最低売上、最高売上を求める**」。次に、先ほど学んだ集計関数を使って平均、最低、最高をクエリにします。①で作った顧客IDごとに売上を合計したテーブルがcustomer_paymentというテーブル名で存在すると想像しながら、書いてみましょう。

```
SELECT
  AVG(total_sales),
  MIN(total_sales),
  MAX(total_sales)
FROM
  customer_payment;

ERROR:  relation "customer_payment" ↵
does not exist
```

customer_paymentテーブルは存在しないのでエラーとなりますが、イメージはつかめると

思います。このエラーを解消するために、括弧で囲った最初のクエリを書き、AS customer_payementとして辻褄を合わせてみましょう。

```
SELECT
  AVG(total_sales),
  MIN(total_sales),
  MAX(total_sales)
FROM
  (SELECT
    customer_id,
    SUM(amount) AS total_sales
  FROM
    payment
  GROUP BY
    customer_id
  ) AS customer_payment;

         avg          |  min  |  max
----------------------+-------+--------
 112.5484307178631052 | 50.85 | 221.55
(1 row)
```

これで1顧客あたりの平均売上、最低売上、最高売上を求めることができました。

問題を分割して考えた際に、結果のテーブルを使ってさらに結果を求めるイメージができるのなら、そこがサブクエリの使いどころです。サブクエリ単位で実行し、結果を確認しながらクエリを構築していくと良いでしょう。

JOINとサブクエリの使い分け - なるべくJOINを使おう

サブクエリを使うクエリは、しばしばJOINを用いても実現可能です。たとえば「直近の支払い3件を、顧客の氏名とともに抽出する」というクエリは、JOINでもサブクエリでも結果を得られます。しかしサブクエリを用いたクエリは重くなりがちです。

1つのクエリの中に複数のクエリがあり、それぞれを実行しなければ結果が得られないという特性と、サブクエリの結果テーブルにはインデックスと呼ばれる索引機能が働かないためです。

一方、JOINを用いた場合は（設定にもよりますが）インデックスと呼ばれる索引機能が働くため高速にテーブル同士の結合が行われます。また、サブクエリを使ったクエリはネストが深くなり、可読性が落ちがちです。ですから、JOINで済む場合はなるべくJOINを使うようにしましょう。

応用編の後半では、重くなりがち、可読性が落ちがちなサブクエリの欠点を解消するWITHを紹介していますので、そちらもご覧ください。

別のクエリ結果を値として使う - IN

初級編では、IN（値の列挙）という絞り込みの方法を学びました。ここではその応用として、()内でサブクエリを使う方法を学びます。

まずは2007年5月に支払いのあった顧客のlast_nameを取り出してみましょう。ここではpayment_p2007_05テーブルを扱います。これはpaymentテーブルのうち、2007年5月のデータのみを取り出したものです（今後もpayment_p2007_XXというテーブルをいくつか扱うので、覚えておいてください）。

まずはサブクエリを用いず、中級編で習ったJOINを使って素直に書いてみましょう。

ノック34 customerテーブルとpayment_p2007_05テーブルからJOINを用いて、2007年5月に支払いのあった顧客のlast_nameを抽出する

次のようなクエリになります。

```
SELECT
  last_name
FROM
  customer AS c
  INNER JOIN payment_p2007_05 AS p
    ON c.customer_id = p.customer_id
;

 last_name
-------------
BROWN
MOORE
ANDERSON
WHITE
HARRIS
HARRIS
(..略..)
(182 rows)
```

一見うまくいったように見えますが、この方法だと2回購入した人（HARRIS）は2回出てきてしまいます。もちろんDISTINCTを使って重複を削除することもできますが、ここからさらにJOINをつなげていくケースなどでは、JOINするごとにどんどん行が膨れあがってしまい、処理時間も膨大にかかります。

それではこれをINを用いて書き直してみます。

ノック35 INを用いて、2007年5月に支払いのあった顧客のlast_nameを抽出する

次のようなクエリになります。

```
SELECT
  last_name
FROM
  customer
WHERE
  customer_id IN
    (SELECT
      customer_id
    FROM
      payment_p2007_05
    )
;

last_name
-------------
BROWN
MOORE
ANDERSON
WHITE
HARRIS
CLARK
(..略..)
(158 rows)
```

先ほど2回出てきていたHARRISは、無事に1回だけ表示されるようになりました。

条件はcolumn IN（query）という形になり、queryのところにSELECTクエリが書かれます。ここでは5月に支払いのあったcustomer_idを抽出しており、その結果を値として親クエリではINを用いて絞り込みを行っています。試しにINの内部のSELECTクエリだけを実行して、customer_id一覧が表示されることをご自身で確認してみてください。

別のテーブルに存在する値だけ抽出する - EXISTS

実は先ほどのINで扱ったクエリは、EXISTSで書き直すことも可能です。EXISTSはその名のとおり、該当する行が存在するかどうかを判定する構文です。

ノック36 payment_p2007_05テーブルから、EXISTSを用いて、2007年5月に支払いのあった顧客のlast_nameを抽出する

実際のクエリを見てみましょう。

```
SELECT
  last_name
FROM
  customer AS c
WHERE
  EXISTS (
    SELECT
      1
    FROM
      payment_p2007_05 AS p
    WHERE
      c.customer_id = p.customer_id
  )
;

last_name
-------------
BROWN
MOORE
ANDERSON
WHITE
HARRIS
CLARK
(..略..)
(158 rows)
```

EXISTSキーワードの後ろにはサブクエリを書きますが、先ほどのINと異なり、EXISTSのクエリは単独では実行できません。サブクエリのWHERE句にはJOIN同様に結合条件が書かれます。内部ではJOINと似た結合が行われますが、あくまでEXISTSは該当するレコードがあるかないかを確認するだけであり、実際に複数レコードとの結合が行われることはありません。

EXISTSの特徴をおさらいしておきましょう。

・ある外部キーがほかのテーブルに存在するかどうかを判定する

・一般的にJOINしてからDISTINCTを行うのに比べ高速である
・SELECT句のカラムは結果に影響せず、慣例的に1が使われることが多い
・これまでの条件演算と同様にNOT EXISTSとして、否定条件で使うこともできる

EXISTSとINの使い分け

EXISTSとINでは似た動きができることがわかりました。利用するデータベースにもよりますが、一般的には、

・INはまずサブクエリを実行する
・EXISTSは内部でJOINに近い動きをする

という違いがあります。

使い分けについてはまずは、

・サブクエリの結果が小さくなる場合は、INを使う
・親クエリの結果が小さくなる場合は、EXISTSを使う

と覚えておきましょう。

問い合わせの結合 - UNION、INTERSECT、EXCEPT

次は問い合わせの結合です。2つ以上のクエリの結果を組み合わせる方法を学びます。図で表すと**図1**のようなイメージになります。

結果の和集合をとる - UNION

2つ以上のクエリから、その結果の和集合（どちらか、もしくは両方に含まれるもの）を取り出したいときはUNIONでクエリをつなぎます。

ノック37 1月と5月の支払い履歴（payment_p2007_01/05テーブル）から、どちらかに含まれるcustomer_idを抽出する

次のようなクエリで実現できます。

```
SELECT                          ┐
  DISTINCT customer_id          ├ クエリA
FROM                            │
  payment_p2007_01              ┘
UNION
SELECT                          ┐
  DISTINCT customer_id          ├ クエリB
FROM                            │
  payment_p2007_05              ┘
;

customer_id
-------------
         251
         106
         120
(..略..)
(539 rows)
```

▼図1　問い合わせの結合イメージ

結合の種類	イメージ
UNION	クエリA ◯◯ クエリB
INTERSECT	クエリA ◯◯ クエリB
EXCEPT	クエリA ◯◯ クエリB

このように最低どちらかに入っているcustomer_idが表示されます。このとき重複はデフォルトで削除されますが、削除を避けたい場合はキーワードをUNION ALLに変更しましょう。

結果の積集合をとる - INTERSECT

同様に積集合（両方に含まれるもの）をとりたい場合は、INTERSECTを使います。

ノック38 1月と5月の支払い履歴（payment_p2007_01/05テーブル）から、両方に含まれるcustomer_idを抽出する

次のようなクエリになります。

```
SELECT
  DISTINCT customer_id
FROM
  payment_p2007_05
INTERSECT
SELECT
  DISTINCT customer_id
FROM
  payment_p2007_01
;

customer_id
-------------
         251
         120
         337
(..略..)
(139 rows)
```

5月と1月、どちらでも利用のあった顧客が139人いたことがわかります。

問い合わせの結合はどのキーワードでも連続させることが可能です。1、2、3月と続けて利用した顧客を見てみましょう。

1、2、3月の支払い履歴（payment_p2007_01/02/03テーブル）から、すべてに含まれるcustomer_idを抽出する

```
SELECT
  DISTINCT customer_id
FROM
  payment_p2007_01
INTERSECT
SELECT
  DISTINCT customer_id
FROM
  payment_p2007_02
INTERSECT
SELECT
  DISTINCT customer_id
FROM
  payment_p2007_03
;

Fcustomer_id
-------------
         251
         106
         120
(..略..)
(512 rows)
```

これは実際には次のクエリと同義で、内部では先に前の2つのクエリが実行されています。

```
(
  SELECT
    DISTINCT customer_id
  FROM
    payment_p2007_01
  INTERSECT
  SELECT
    DISTINCT customer_id
  FROM
    payment_p2007_02
)
INTERSECT
SELECT
  DISTINCT customer_id
FROM
  payment_p2007_03
;
(実行結果は省略)
```

結果の差集合をとる - EXCEPT

それでは、最後に差集合（片方にあって、もう片方にはないもの）を取り出すEXCEPTを実行してみましょう。

payment_p2007_01/05テーブルから、5月に支払いがあって、1月には支払いがなかったcustomer_idを抽出する

次のようなクエリになります。

```
SELECT
  DISTINCT customer_id
FROM
  payment_p2007_05
EXCEPT
SELECT
  DISTINCT customer_id
FROM
  payment_p2007_01
;

customer_id
-------------
         264
         410
          80
(..略..)
(19 rows)
```

和集合のUNION、積集合のINTERSECTと異なり、差集合EXCEPTでは結合するクエリの順

番によって結果が異なることに注意しましょう。上のクエリでは、5月にいて1月にはいなかった顧客を抽出していますが、以下のクエリでは1月にはいて5月にはいなくなってしまった顧客を抽出しています。

```
SELECT
  DISTINCT customer_id
FROM
  payment_p2007_01
EXCEPT
SELECT
  DISTINCT customer_id
FROM
  payment_p2007_05
;

customer_id
-------------
         106
         455
         209
(..略..)
(381 rows)
```

実行してみて結果セットが異なることを確認しましょう。

結合結果に対しての操作

UNIONなどで結合された結果に対しては、通常のクエリと同様GROUP BYやORDER BYを適用することが可能です。

payment_p2007_01/05テーブルから、1月または5月に支払いがあった顧客を、customer_idの昇順で3件のみ抽出する

次のようなクエリになります。

```
SELECT
  DISTINCT customer_id
FROM
  payment_p2007_01
UNION
SELECT
  DISTINCT customer_id
FROM
  payment_p2007_05
ORDER BY
  customer_id ASC
LIMIT
  3
;

customer_id
-------------
          1
          2
          3
(3 rows)
```

複雑なサブクエリを簡略化する - WITH [注1]

複雑な分析をしていると、集計した結果を結合して、それと別の集計テーブルを結合……と、どんどん構想が広がっていきます。もちろんこれまで使ったテクニックを駆使してそれを実現しても良いのですが、テーブルが増えていくとクエリの見通しが悪くなってしまいます。

WITHを使えば、実行結果を1つの仮想テーブルとして扱うことができます。たとえば1月に7回以上レンタルしてくれたロイヤルカスタマーにメールを送りたいとします。まず、payment_p2007_01テーブルを集計して7回以上支払いのあるcustomer_idを抽出し、その後customerテーブルと連結してみましょう。

WITH句を使って、payment_p2007_01テーブルから1月に7回以上支払いのあったアクティブなcustomerのemailを抽出する

構文はWITH table_name AS (query)となります。queryの結果をtable_nameとして、それ以降のクエリで利用することができます。では、実際に次のクエリを実行してみてください。

注1）MySQLでは8.0から利用可能です。

```
WITH loyal_customers AS (
  SELECT
    customer_id,
    COUNT(*) AS cnt
  FROM
    payment_p2007_01
  GROUP BY
    customer_id
  HAVING
    COUNT(*) >= 7
)
SELECT
  email
FROM
  customer AS c
  INNER JOIN loyal_customers AS lc
    ON c.customer_id = lc.customer_id
WHERE
  c.active = 1
;
```

```
email
---------------------------------
EDNA.WEST@sakilacustomer.org
SUE.PETERS@sakilacustomer.org
LESLIE.SEWARD@sakilacustomer.org
(3 rows)
```

また、WITHは2つ以上同時に使うこともできます。その場合は、

```
WITH table1 AS (
  query
),
tabel2 AS (
  query
)
```

と、カンマで区切ります。

部分的に集計関数を適用する - ウィンドウ関数 注2

先ほどはロイヤルカスタマーを抽出しましたが、ここで各顧客に対してそのロイヤルティを購入回数に応じて順位付けしたいとしましょう。各行に対して、全体と比較した評価を行いたい場合はウィンドウ関数を使用します。

まずはサンプルを見てみます。1月の利用回数が多かった顧客と順位を一緒に表示しましょう。

ノック43 ウィンドウ関数を使って、payment_p2007_01テーブルとcustomer_listテーブルから1月の利用回数が多かった顧客をその順位と一緒に表示する

次のようなクエリになります。

```
SELECT
  cl.name,
  COUNT(*) AS cnt,
  RANK() OVER (
    ORDER BY COUNT(*) DESC
  ) AS ranking
FROM
  payment_p2007_01 AS p
  INNER JOIN customer_list AS cl
    ON p.customer_id = cl.id
GROUP BY
  cl.name
;
```

```
name                 | cnt | ranking
---------------------+-----+---------
SUE PETERS           |   8 |       1
EDNA WEST            |   7 |       2
LESLIE SEWARD        |   7 |       2
MINNIE ROMERO        |   6 |       4
BILLY POULIN         |   6 |       4
(..略..)
(520 rows)
```

このクエリではcustomerではなく、customer_listテーブルを使って顧客情報を取得しています。customer_listテーブルは「ビュー(view)」と呼ばれる特殊なテーブルです。ビューはSELECTクエリの実行結果を保持しているものでcustomerテーブルのfirst_nameとlast_nameをつなげたnameカラムや、address、countryテーブルをJOINして得られたcountry(居住国)カラムなどがあります。応用編ではこのcustomer_listビューを活用してノックを進めます。

さて、SELECT句の3行目から5行目に見慣れない関数があると思いますが、これがウィン

注2) MySQLでは8.0から利用可能です。

ドウ関数です。

構文はfunction_name() OVER (ORDER BY column)となっています。順番に見ていきましょう。まず、RANK()は順位を表示する関数であることを示しています（それ以外のウィンドウ関数についても、のちほど扱います）。次のOVER (ORDER BY column)にて順序を指定します。今回は回数の降順で順位付けしたいのでORDER BY COUNT(*) DESCを指定しています。

範囲（パーティション）を指定する - PARTITION

次に順位を国別に取得してみましょう。今回は先ほどのRANK()とORDER BYに加えて、順位付けする範囲をOVER()の中でPARTITION BY columnとして指定します。実行結果の横幅が長くなってきましたので、ここからは顧客の名前ではなくIDを表示します。

ノック44 payment_p2007_01テーブルとcustomer_listテーブルから、1月の顧客の利用回数順位を国別に表示する

次のクエリを実行してみてください。

```
SELECT
  cl.id,
  cl.country,
  COUNT(*) AS cnt,
  RANK() OVER (
    PARTITION BY cl.country
    ORDER BY COUNT(*) DESC
  ) AS rank
FROM
  payment_p2007_01 AS p
  INNER JOIN customer_list AS cl
    ON p.customer_id = cl.id
GROUP BY
  cl.id, cl.country
;

id  | country  | cnt | rank
----+----------+-----+------
176 | Algeria  |   5 |    1
 69 | Algeria  |   2 |    2
441 | Algeria  |   1 |    3
528 | Angola   |   5 |    1
383 | Angola   |   2 |    2
381 | Anguilla |   3 |    1
(..略..)
(520 rows)
```

今回は国ごとに順位がついているのがわかると思います。

このようにウィンドウ関数は大きく分けて、

・関数
・範囲
・順序

からなっています。

ウィンドウ関数で平均を求める - AVG

先ほどは国別の順位を表示しました。次に国別の平均値も併せて表示してみましょう。

ノック45 ノック44の結果にあわせて、国別の平均利用回数も表示する

使う関数は平均値を計算するAVGです。クエリは次のようになります。

```
SELECT
  cl.id,
  cl.country,
  COUNT(*) AS cnt,
  ROUND(AVG(COUNT(*)) OVER (
    PARTITION BY cl.country
  ), 2) AS avg_pay,
  RANK() OVER (
    PARTITION BY cl.country
    ORDER BY COUNT(*) DESC
  ) AS rank
FROM
  payment_p2007_01 AS p
  INNER JOIN customer_list AS cl
    ON p.customer_id = cl.id
GROUP BY
  cl.id, cl.country
;

id  | country  | cnt | avg_pay | rank
----+----------+-----+---------+------
176 | Algeria  |   5 |    2.67 |    1
 69 | Algeria  |   2 |    2.67 |    2
441 | Algeria  |   1 |    2.67 |    3
528 | Angola   |   5 |    3.50 |    1
383 | Angola   |   2 |    3.50 |    2
381 | Anguilla |   3 |    3.00 |    1
(..略..)
(520 rows)
```

先ほどのRANKと異なり、AVGの場合は何の平均を表示するかをAVG(column)として指定

します。今回は国別の利用回数の平均なので、COUNT(*)を入れてAVG(COUNT(*))とします。

OVER句は先ほどと同様にPARTITION BY customer_list.countryを指定。小数点の表示が長くなってしまうので、ROUND関数でくくっておきましょう。

ウィンドウ関数で合計を求める - SUM

同様に国ごとの合計を表示することも可能です。

ノック46 ノック45のクエリを変更し、平均回数の代わりに国ごとの合計利用回数を表示する

SUM(COUNT(*))を用いて、国ごとの合計利用回数を並べて表示してみましょう。

```
SELECT
  cl.id,
  cl.country,
  COUNT(*) AS cnt,
  SUM(COUNT(*)) OVER (
    PARTITION BY cl.country
  ) AS total_pay,
  RANK() OVER (
    PARTITION BY cl.country
    ORDER BY COUNT(*) DESC
  ) AS rank
FROM
  payment_p2007_01 AS p
  INNER JOIN customer_list AS cl
    ON p.customer_id = cl.id
GROUP BY
  cl.id, cl.country
;

id  | country  | cnt | total_pay | rank
----+----------+-----+-----------+------
176 | Algeria  |   5 |         8 |    1
 69 | Algeria  |   2 |         8 |    2
441 | Algeria  |   1 |         8 |    3
528 | Angola   |   5 |         7 |    1
383 | Angola   |   2 |         7 |    2
381 | Anguilla |   3 |         3 |    1
(..略..)
(520 rows)
```

累積比率のための累積回数

ここまで国別の平均や合計を出してきましたが、パレート分析などを行う際には累積の比率が重要になります。そこで、国ごとの累積比率の算出に挑戦してみましょう。

累積比率は「累積比率＝累積利用回数÷全体の利用回数」として算出するため、まず累積の利用回数が必要です。

あるパーティションの中で集計する範囲を限定するためにはROWS BETWEENオプションを利用します。こちらはパーティションと区別してフレームと呼ばれます。

ノック47 1月の国別の利用回数を降順に抽出し、累積回数と併せて表示する

次のクエリを実行してみましょう。

```
SELECT
  cl.country,
  COUNT(*) AS count,
  SUM(COUNT(*)) OVER (
    ORDER BY COUNT(*) DESC
    ROWS BETWEEN
      UNBOUNDED PRECEDING
      AND CURRENT ROW
  ) AS cumulative_count
FROM
  payment_p2007_01 AS p
  INNER JOIN customer_list AS cl
    ON p.customer_id = cl.id
GROUP BY
  cl.country
ORDER BY
  count DESC
;

country       | count | cumulative_count
--------------+-------+------------------
India         |   111 |              111
China         |   109 |              220
United States |    70 |              290
Japan         |    65 |              355
Mexico        |    63 |              418
(..略..)
(99 rows)
```

cumulative_countに、最初の行から現在の行までのcountを合計した値が入っていることがわかります。

6行目の見慣れないROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROWがウィンドウの範囲（ウィンドウフレーム）の指定となります。これはROWS BETWEEN frame_start AND frame

_endという文法になっており、それぞれフレームの始点と終点を設定します。frame_startとframe_endには、**表1**の値が設定できます。

今回のケースではパーティション内を降順に並べたうえで、パーティションの最初の行から現在の行までを合計することで、累積回数を得ています。

frame_endは省略が可能であり、デフォルトはCURRENT ROWとなるため、今回のケースでは単にROWS UNBOUNDED PRECEDINGと記述することも可能です。

累積比率

それでは累積比率の計算に移ります。「累積比率＝累積利用回数÷全体の利用回数」という式をそのままクエリ上で表現します。累積利用回数は先ほど（ノック47）のとおり。全体の利用回数は、こちらもウィンドウ関数のSUM(COUNT(*)) OVER ()を利用します。

ノック48 payment_p2007_01テーブルとcustomer_listテーブルから1月の国別の利用比率を降順に並べ、累積で表示する

次のようなクエリになります。

```
SELECT
  cl.country,
  ROUND(
    SUM(COUNT(*)) OVER (
      ORDER BY COUNT(*) DESC
      ROWS BETWEEN
        UNBOUNDED PRECEDING
        AND CURRENT ROW
    ) / SUM(COUNT(*)) OVER (),
    2
  ) AS cumulative_percent
FROM
  payment_p2007_01 AS p
  INNER JOIN customer_list AS cl
    ON p.customer_id = cl.id
GROUP BY
  cl.country
ORDER BY
  COUNT(*) DESC
;
```

▼表1　ROWS BETWEENで指定できる値

キーワード	意味
UNBOUNDED PRECEDING	パーティションの最初の行
n PRECEDING	n行前
CURRENT ROW	現在の行
n FOLLOWING	n行後
UNBOUNDED FOLLOWING	パーティションの最後の行

```
country            | cumulative_percent
-------------------+--------------------
India              |               0.10
China              |               0.19
United States      |               0.25
Japan              |               0.31
Mexico             |               0.36
Russian Federation |               0.41
Brazil             |               0.46
Philippines        |               0.49
Indonesia          |               0.52
(..略..)
(99 rows)
```

全体として99ヵ国ありますが、上位9ヵ国で利用の過半数を占めていることがわかります。

ウィンドウ関数で移動平均を求める

最後はこれまで学んだテクニックを活用して移動平均を計算してみましょう。

移動平均とはおもに時系列データなどに対し、ある一定区間ごとの平均値を、区間をずらしながら求めるものです。たとえば曜日ごとの変動の大きいデータに対して7日間移動平均を計算することで、曜日変動を抑えた傾向を見ることができます。

今回はpaymentテーブルを使い、4月6日から12日の利用回数の3日移動平均を見てみましょう。まずは当該期間について、日別の利用回数を計算します。

ノック49 paymentテーブルを使い、2007年4月6日から12日の日別利用回数を集計する

次のクエリを実行してみてください。

```
SELECT
  CAST(payment_date AS DATE) AS d,
  COUNT(*)
FROM
  payment AS p
WHERE
  CAST(payment_date AS DATE)
    BETWEEN '2007-04-06' AND '2007-04-12'
GROUP BY
  d
ORDER BY
  d ASC
;

     d      | count
------------+-------
 2007-04-06 |   486
 2007-04-07 |   472
 2007-04-08 |   516
 2007-04-09 |   514
 2007-04-10 |   482
 2007-04-11 |   468
 2007-04-12 |   452
(7 rows)
```

ここまでは大丈夫ですね。次にこのクエリを改変して移動平均を計算します。

paymentテーブルを使い、2007年4月6日から12日の利用回数の3日移動平均を計算する

利用回数の移動平均を出すためウィンドウ関数にはAVG(COUNT(*))を使い、順序は日付順、範囲（フレーム）のスタートには2行前（2 PRECEDING）を指定します。こうして2行前から現在行までの計3行の平均が計算できます。結果を見やすくするため、ROUND関数で小数点桁数も指定しておきましょう。実際のクエリは次のようになります。

```
SELECT
  CAST(payment_date AS DATE) AS d,
  COUNT(*),
  ROUND(AVG(COUNT(*)) OVER (
    ORDER BY
      CAST(payment_date AS DATE) ASC
    ROWS BETWEEN
      2 PRECEDING
      AND CURRENT ROW
  ), 2) AS moving_avg
FROM
  payment AS p
WHERE
  CAST(payment_date AS DATE)
    BETWEEN '2007-04-06' AND '2007-04-12'
GROUP BY
  d
ORDER BY
  d ASC
;

d          | count | moving_avg
-----------+-------+------------
2007-04-06 |   486 |     486.00
2007-04-07 |   472 |     479.00
2007-04-08 |   516 |     491.33
2007-04-09 |   514 |     500.67
2007-04-10 |   482 |     504.00
2007-04-11 |   468 |     488.00
2007-04-12 |   452 |     467.33
(7 rows)
```

結果を検証してみましょう。9日のmoving_avg値を見てみると、前2日の結果との平均値「(472 + 516 + 514) ÷ 3」と等しくなっています。

あとはフレーム句を6 PRECEDINGにして1週間の移動平均を計算したり、1 PRECEDING AND 1 FOLLOWINGにして前後1行を使った移動平均を算出したりもできます。先述の曜日変動に限らず、日々のバラつきが大きいとき、長期的な傾向を把握したいときにも有効な手法です。

終わりに

いかがでしたか？　初級編でSELECTを使ってテーブルの内容を抽出するところからスタートして、応用編の最後には移動平均を使って期間ごとの数値変動を求めるところまで学びました。

本特集で取り上げたノックをマスターしたなら、データ・ドリブンな意思決定のために必要なSQL力を確実に身につけたと言えます。

一度では理解しきれない部分もあると思いますが、クエリを書き換えながら実行することで理解が深まっていきます。繰り返しノックを解いて、自分の力にしていってください。SD

SQL勉強会を開いて、チームのSQL力を高める5つのポイント

本特集を読んでくださっている読者の方は、組織内で仕事をしている方が多いかと思います。働いている中で、「営業から数字出しや分析の依頼を急にされて困った」とか、「クライアントから、顧客の購入一覧を出してと急かされた」などの依頼を受けた経験は、誰しもあるものではないでしょうか？

本特集をここまで読んだ方ならお気づきかと思いますが、データを必要とした人がクエリを書いて、試行錯誤しながらデータを取得しないと納得のいくデータは取得できません。依頼されてデータ出しを行うと「これじゃない……」と手戻りも起こりがちです。

もし「みんなSQLが書ければもっと円滑にコミュニケーションが取れるのに……」そんな環境にいるのなら、SQL勉強会を開くことは良い解決法になるかもしれません。

筆者らの所属する㈱リブセンスでも前述のような期間がありましたが、SQL勉強会が各部署ごとに開かれてチームの「SQL力」が上がった結果、各々自分自身でデータ取得する文化になりました。SQL勉強会を開催する際のポイントは次の5点です。

①市販の参考書を使うのではなく、実務に即した資料を用意する

参加者が自学自習しやすくなるよう、その場での口頭説明だけではなく繰り返し閲覧できる資料を用意しましょう。

また、参考書では興味を持てなくても、実務データを使うことで俄然（がぜん）興味が湧く参加者は非常に多いです。さらに、実務のデータと実行環境を工夫して用意すれば、仕事をしながら勉強してもらえます（！）。

ただし、資料作りは結構たいへんですので、そこでエネルギーを使い切ってしまっては元も子もありません。最初は実務で使うテーマを用いたノック（問題）だけを用意して、参加者に解いてもらう形でも勉強会は成り立ちます。

②講師のコミュニティを作る

何よりもまず教える側のコミュニティづくりが大切です。もし複数人の講師が確保できない場合は、まず1～2名にノックを解けるように教え、教えた内容の説明を書いてもらうなどして講師のサポートをしてもらいましょう。

講師の負担は大きいので、講師同士で資料をシェアしあい、盗み合いましょう。その過程で、組織内で伝わりやすい教材が育まれていきます。

③長期型ではなく、短期集中型で実施する

「長期間かけてじっくり」というケースですと、徐々に参加者が減りがちです。仕事の繁忙期などとかぶり、継続もたいへんです。

ですから短期集中型で、参加しきれる勉強会設計が大事です。週に1回を数ヵ月やるよりも、毎日開催して2週間で終了とするなど期間を区切ってしまいましょう。

内容は一度ですべてを盛り込んでしまうと講師も参加者もたいへんですし、脱落者も出やすくなりますので、参加者の興味を引くポイントを絞って実施しましょう。弊社では、参加者が増えてきたタイミングで、白帯コース・茶帯コース・黒帯コースのようにレベル分けして開催する方法を取りました。

④講師のサポートは手厚く

質問を投げると返してくれる土壌を用意しておくことは重要です。チャットや口頭で気軽に聞ける人間関係を作ることも、（一見、技術的には見えなくとも）組織の技術レベルを高めるために重要です。

⑤毎回、宿題を出す

反復だけが、「SQL力」を高めます。

以上が、SQL勉強会の開催ノウハウです。講師の技術レベルは、必ずしも高い必要はありません。講師には、知識よりも「参加者に寄り添う気持ち」が重要だからです。

本特集をここまで読んだ方なら、十分講師をできるレベルです。ぜひSQL勉強会を開いてチームのSQL力を高めてください。

column クエリの共有方法――社内WikiとRedash

分析クエリは、仮説を検証するためにアドホックに1回だけ使われるものもあれば、KPI（Key Performance Indicator、重要業績評価指標）のチェックなどを目的として繰り返し使われるものもあります。

当社ではそれぞれの目的によって共有のしかたを分けています。

前者のケースは、社内のWikiに仮説とその検証結果をクエリとともに記載して共有しています。現実世界のデータは、しばしば複雑な要件が入り混じっていることがあり、分析者が気づかない条件の考慮もれをしているケースがあったりします。結果だけでなくクエリも共有しているのは、Wikiの記事を見たほかのメンバーがミスを発見できるようにしたり、クエリを再実行してみて再検証できるようにしたりするためです。また、このWikiの記事を元に新たな仮説が見つかるかもしれません。

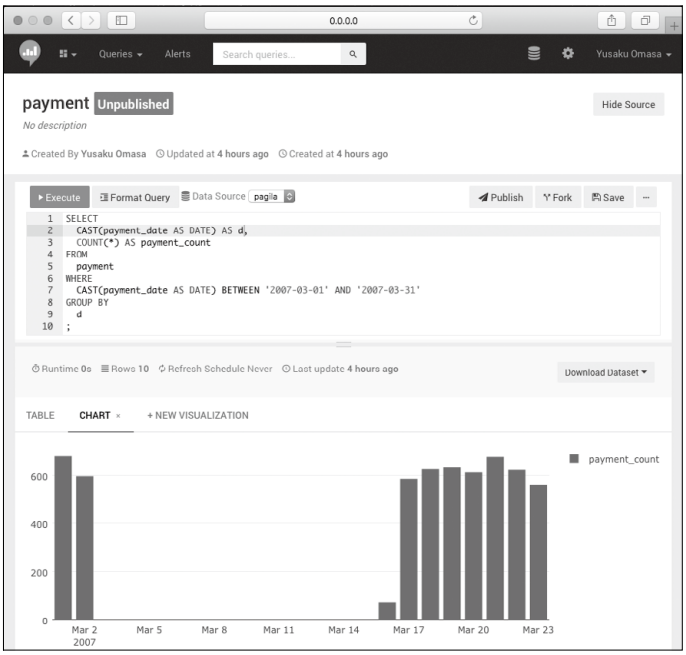
一方、後者のケースは「Redash」[注A]などのツールで共有しています。

Redashは、データベースを含む多種多様なデータ・ソースに対して、クエリを実行することができるWebアプリケーションです。クエリを保存して共有できるだけではなく、スケジューラを使って定期的にクエリを実行したり、実行結果をグラフ描画したりすることができます（**図2**）。また、複数のグラフをダッシュボードとして1画面にまとめることもできるので、KPIのチェックにとても便利です。

なお、どちらの共有をする場合でも、クエリの実行結果（数値など）からグラフを描画することをお勧めします。グラフにすることで数値を見るだけでは見えなかった、分布の偏りや傾向が見えることがあります。**図2**のpaymentの日別の支払い回数を集計したグラフを見ると、月初から月中まで支払いのデータがないことが一目瞭然でわかります。

クエリの使われ方を意識しながら、より効果的な共有ができるように工夫してみてください。

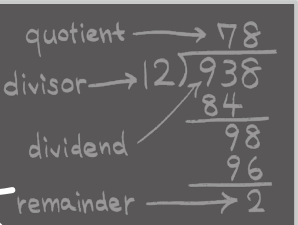
▼図2　Redash



注A）URL https://redash.io/

第2特集

なぜ、コンピュータは割り算が下手なのか!?



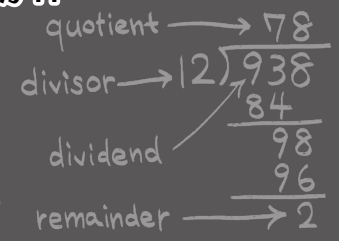
日ごろ、みなさんはコンピュータを何に使っていますか？　インターネットの情報を入手するために使っていますか？　SNSで友人知人とコミュニケーションをとるために使っていますか？　レポートや報告書を書いたり、売上を集計したり、プレゼンテーションの資料を作ったり、お絵かきをしたり、動画編集をしたり、コンピュータの使い方は千差万別です。本誌の読者であれば、プログラミングでソフトウェアやシステムの開発に携わっている方も多いことでしょう。

しかし、情報処理装置としてのコンピュータも、その基本は計算を繰り返すことによって情報を処理します。テキストデータや動画、音声のデータであっても、その基本は文字コードでありコード化された音声情報、映像情報です。その実態は数値データにすぎません。コンピュータは中央処理装置（Central Processing Unit：CPU）で超高速な数値演算を繰り返し、あらゆる情報処理を実現しています。

特集では、どうしてコンピュータは割り算が苦手なのか、その秘密を紐解（ひもと）いていくことにします。

Author 飯尾 淳（いいお じゅん）　中央大学
Mail iiojun@tamacc.chuo-u.ac.jp
Twitter @iiojun

第1章 割り算はなぜ難しい？



Author 飯尾 淳(いいお じゅん)　中央大学
Mail iiojun@tamacc.chuo-u.ac.jp
Twitter @iiojun

みなさんは、小学校で四則演算を習ったことでしょう。加減乗除、ひらたく言えば、足し算、引き算、掛け算、割り算です。いずれも基本的な数値演算です。ところが、コンピュータは割り算を苦手としています。

そこでまず、そもそも割り算は難しいものだということを理解してください。本章では簡単な例題として、整数の加減乗除から考えていきます。

整数の加減乗除

四則演算の中で、割り算というのはやや特別な演算です。割り算の説明に入る前に、まずは、自然数の計算から始めましょう。小学校の算数に遡(さかのぼ)ります。

自然数の集合上における演算

「自然数」とは、1から始まる、1，2，3，……という数のことです。定義によっては、0を含めることもあります。本稿では、説明の便宜上、前者の定義を用いることにします。

さて、まずは自然数の集合を考えましょう。自然数の集合$\boldsymbol{N}$とは、自然数のすべてを含む集合です。自然数はいくらでも大きな数を考えることができるので、集合$\boldsymbol{N}$の大きさは無限大です。集合論を考えるとき、どのくらいの程度の無限大なのか、それを集合の濃さ(濃度)と考えて議論することがありますが、ここではそこまでは踏み込みません。とにかく無限大の個数を持つ自然数の集まりと考えてください。

ここで、自然数の集合$\boldsymbol{N}$から任意の要素$\boldsymbol{x}$と$\boldsymbol{y}$を取り出します。$\boldsymbol{x} \in \boldsymbol{N}$[注1]、$\boldsymbol{y} \in \boldsymbol{N}$です。$\boldsymbol{x}$と$\boldsymbol{y}$は同じ値でもかまいません。そのうえで、$\boldsymbol{x}$と$\boldsymbol{y}$に関する演算「☆」を考えましょう。$\boldsymbol{x}$☆$\boldsymbol{y} \rightarrow \boldsymbol{z}$となるような操作を考えます。この表現は、「演算☆は$\boldsymbol{x}$と$\boldsymbol{y}$という2つの要素を対象とし、その結果が$\boldsymbol{z}$となる」ということを表しています。

それでは、演算☆の実態として、まずは足し算を考えてみます。任意の自然数$\boldsymbol{x}$, $\boldsymbol{y}$を考えたときに、足し算の結果得られる$\boldsymbol{z}$は$\boldsymbol{N}$に含まれるでしょうか。

その答えは自明ですね。2つの自然数を足し算した結果は、自然数です。つまり、$\boldsymbol{x}+\boldsymbol{y} \rightarrow \boldsymbol{z}$としたとき、zは必ず自然数になります。すなわち、自然数の集合Nは、足し算について閉じている[注2]といえます。

では、引き算についてはちょっと脇に置いておくことにして、掛け算はどうでしょうか。掛け算についても足し算と同じことがいえます。$\boldsymbol{x} \times \boldsymbol{y} \rightarrow \boldsymbol{z}$で得られる$\boldsymbol{z}$は必ず自然数になります。自然数の定義として0を含めた場合でもそれは同様です。自然数の集合$\boldsymbol{N}$は掛け算についても閉じています。

ところで、先ほど脇に置かれた引き算はどうでしょうか。$\boldsymbol{x}$-$\boldsymbol{y} \rightarrow \boldsymbol{z}$のとき、$\boldsymbol{z}$は必ず自然数と

注1) 「∈」は「集合に属する」を示す記号で、この場合は「$\boldsymbol{x}$は$\boldsymbol{N}$に属する」という意味。

注2) ある集合がその演算について閉じているとは、任意の要素に対する演算の結果が、必ずその集合に含まれることをいいます。

なることが保証されているでしょうか？　なお、負の数を習うのは、中学生からですね。ここから先は小学校の四則演算を少しだけ超えた話に進みます。

任意の自然数x, yを考えたときに、x-y→zの結果、zが自然数になるとは限りません。最も単純な場合として、xとyが同じもののとき引き算の結果は0となり、自然数の範囲を逸脱してしまいます。0を含むような自然数の定義を用いたとしても、x(引かれる数)がy(引く数)よりも小

コラム　逆ポーランド記法

先ほどの例では、演算の表現として「演算☆」を考えました。この演算☆はxとyの2つを対象とした計算操作を表現していました。そしてその表記は「x☆y」というように2つの演算対象の間に演算子を配置します。

2つの対象の間に演算子を配置したのは、我々が習ってきた四則演算に対応させやすいと考えたからというだけの理由です。x☆y→zという操作を、2つのパラメータxとyをとる関数fが計算結果としてzを返すという状況に見立てることだってできるでしょう。つまり、関数fの実態が演算☆であるという状況です。このとき、$f(x,y)$→zと表現できます。x,yという引数にfを適用するということをシンプルに表すならば、fxy→zと書いてもかまいません。演算子のままで表現すれば、☆xy→zという表現もできますね。

x☆y→zという表現を、演算子を中に置くので中置記法、☆xy→zという表現を、演算子を前に置くので前置記法と呼びます。前置記法は「ポーランド記法(Polish Notation：PN)」とも呼ばれます。これはポーランドの論理学者であるヤン・ウカシェヴィチ(Jan tukasiewicz)に由来するとのことです。前置記法は、先の関数の例をみれば明らかなように、コンピュータのプログラム表記に親和性が高い表現です。

前置記法と中置記法があるならば、後置記法があるだろうという想像は難くありません。そのとおり、「逆ポーランド記法(Reverse Polish Notation：RPN)」と呼ばれる表現も存在します。先の例で表現すれば、xy☆→zという表現になります。

PNもRPNも、複雑な演算を括弧なしで表現できるというメリットがあります。たとえば、「$(3+4)\times(5-2)$」という計算を考えてみましょう。普通に計算すると、「$(3+4)\times(5-2)=7\times3=21$」となりますね。これをRPNで表現すると、「3 4 + 5 2 − × =」と書けます。括弧を使う必要はありません。

RPNに関するメリットの1つに、そのまま読めば計算できる、というメリットがあります。「3 4 + 5 2 − × =」は、「3と4を足して(+)、5から2を引いて(−)、それらを掛けた(×)ものは何(=)？」と素直に読めばよいのです。日本語に素直に対応しています。前置記法は「add three and four … 」と考えると、英語など動詞が先に置かれる言語に対応しているとも言えるでしょう。

さらに、RPNはスタックを用いてごく自然に計算処理を実装できます。頭から順番にパースしていき、「数値が来たらスタックに積む、演算子が来たらスタックの上2つを取り出して演算子、その結果をスタックに積む」という単純なルールを適用するだけで計算できるわけです。

昔の電卓は逆ポーランド記法のものがありました(**写真1**)。今でも、Unixコマンドのdc(digital calculator)は逆ポーランド記法で記述した式を計算してくれます。憶えておくと便利に使えるでしょう[注3](**図1**)。

▼写真1　RPN計算ができる電卓の例(HP 50g：ヒューレット・パッカード)

▼図1　Unixのdcコマンドの実行例

```
iiojun — dc — 67×6
mac3:~ iiojun$ dc
3 4 + 5 2 - * p
21
```

注3) なおdcでは計算結果の出力として「=」の代わりに「p」を使います。pコマンドは、スタックの一番上を出力するという命令です。

さかったとしたら、結果は負の数となり、自然数ではなくなってしまいます。つまり、自然数の集合Nは引き算について閉じていないのです。

割り算の話はさらにややこしくなるのですが、先に結論を言っておくと、割り算も自然数の範囲で語ることはできません。その詳細については後述します。

整数の集合上における演算

さて次は、扱う数の範囲を少しだけ拡大します。これまで自然数を扱ってきましたが、次は整数の集合Zを考えてみましょう。整数の集合Zも、無限個の要素を持つ集合です。

なお、自然数の集合Nと整数の集合Zの大きさは等しい(濃度が等しい)ということが知られています。直感的には「自然数の集合である{1, 2, 3, ……}という集合に0と{-1, -2, -3, ……}という集合を加えたものが整数の集合なんだから、それは自然数の集合の2倍の大きさがあるんじゃないの？」と感じるかもしれません[注4]。

数学では、任意の自然数nを選んだときにそれに対応する整数mが一対一に決まれば、NとZの大きさは同じだということが言えます。そして、次の式を考えると、Nの要素(自然数)であるnと、Zの要素(整数)であるmが一対一対応する[注5]ことは明らかです(図2)。

- $m = n / 2$(n が偶数のとき)
- $m = -(n-1) / 2$(nが奇数のとき)

この対応を考えることにより、自然数の集合Nと整数の集合Zは同じ濃度を持つということが示されます。無限大のトリックとも言えるかもしれませんが、自然数と整数、それぞれの集合は同じ大きさを持つのです。

さて、本題に戻りましょう。自然数は引き算について閉じていない、つまり、「任意の自然数を2つ持ってきて引き算をした結果は、必ずしも自然数とはならない場合がある」ということを先ほどは確認しました。では、整数の集合Zは引き算について閉じているでしょうか？

この答えは明らかですね。整数は自然数を拡張して負の数も含んでいます。よって、引き算の結果として値が負の数になったとしても、その答えは整数に含まれます。演算の対象として負の数を持ってきたとしても、整数であるという事実には変わりありません。ですから整数の集合Zは減算について閉じています。

自然数は足し算と掛け算について閉じていましたが、整数の場合はどうでしょうか。これも、例を示すまでもありません。負の数まで含めたとしても、整数どうしの足し算と掛け算は、必ず整数となるでしょう。すなわち、整数の集合Zは足し算や掛け算についても閉じています。

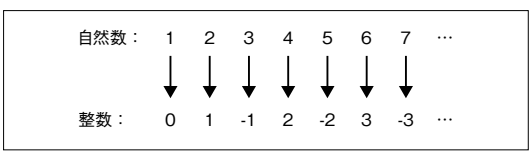
割り算の難しさ

さて、いよいよ割り算です。四則演算のうち、割り算はさらに複雑な演算と考えることができます。なぜなら、割り算の結果は、整数の集合をはみ出してしまう場合が多いからです。最も簡単そうな例を示しましょう。$x \in Z$、$y \in Z$として、$x = 9$、$y = 4$の場合を考えます。$9 / 4 = 2.25$ですね。あるいは分数で表せば$\frac{9}{4}$でしょうか。この答えはすでに整数ではありません。

数学では、さらに実数だとどうなるか、有理数[注6]と無理数[注7]ではどうなるか、虚数[注8]の導入……というように議論が進んでいきますが、本稿ではとりあえずここでいったん立ち止まります。小中学校レベルの四則演算を考えただけで

注4) そもそも自然数の集合が無限大の大きさを持つので、「0」を無視できるということは、なんとなく理解できるでしょう。
注5) この状態を「全単射が存在する」といいます。

▼図2 自然数と整数の全単射の関係



注6) 2つの整数を使った分数形式で表現できる値。
注7) 有理数以外の数値。小数点以下が循環しないものや、πや$\sqrt{2}$なども無理数である。
注8) 自乗してマイナスになる数。

も、四則演算のうち割り算は特別な計算なんだということを理解していただけたでしょうか。

本節のまとめ

- 自然数の集合Nは、四則演算のうち足し算と掛け算にのみ閉じている
- 整数の集合Zは、四則演算のうち足し算、引き算、掛け算に関して閉じている
- 整数を対象とした割り算の結果は、必ずしも整数となるとは限らない

%(余り)計算

小学校の四則演算から考え始めたのに、だいぶややこしい話になってしまいました。ここでまた、小学校で習う範囲の割り算に戻って考えましょう。

割り算と余り

先ほどの9を4で割る計算について、少数や分数をまだ習っていない小学生は、何と答えるでしょうか。彼ら彼女らは、「9割る4は、2、余り1」と答えます。

ここで、何やら怪しい「余り」という言葉が出てきました。小学校の算数では、「9は4で割り切れない」と考えます。9より小さい数のうち4で割り切れる最大の数は8であり、その8を4で割った答えが2、そして、9から8を引いた残りの「余り」が1である、そのような割り算を小学校では行うのでした。

この「余り」、硬い表現でいえば剰余を求める演算は、プログラミングではしばしば使われるのでみなさんにも馴染みのある演算かもしれません。言語によりますが、%という演算子を用いたり、moduloの意味でmodという記号を用いたりしますね。

よくある例題として、2次元のデータを1次元に並べ直すときに使うケースを考えてみましょう。(x, y)の画素値で表現される2次元画像データを、1次元配列に格納しなければならないようなケースです。

画像の幅をw、高さをhとすると、1次元配列のインデックスiは、$i = y \cdot w + x$(ただし、$0 \leqq x < w$, $0 \leqq y < h$)で計算されます。画素値の座標(x, y)も、幅や高さを表すwとhも、そしてインデックスiも、すべて整数であることがミソです。x, yとiの間に上記の関係があるときに、iが与えられれば、次の式でxとyを求めることができます。

$$x = i \,\%\, w$$
$$y = i / w$$

ただし、ここでの割り算は「整数の範囲での割り算(余りを無視する)」です。

整数の範囲での計算方法

ところで、「9割る4は、2余り1」という答えは、そもそも、x☆$y \rightarrow z$という式に合致していません。この形式に適合させるにはどうすればよいでしょうか。

まず、「余り」を無視するという方針が考えられます。最も単純な解決方法です。「9割る4は2。以上！」としてしまう方法です。余りは考えない、ある意味で潔い方法です。ただし、この方法には大きなデメリットがあります。「8割る4」、「9割る4」、「10割る4」、「11割る4」、いずれも答えは2であり、答えから割られる数を求めることができません。

整数の範囲で可逆演算とするには、「**整数 / 整数 → {整数、整数}**」としなければなりません。すなわち、先に示した例のように、余りを無視した割り算と余りの結果を組みにして扱う方法です。この方法を用いれば、2つの数字と割る数から、割られる数を再現することができます。整数の範囲でなんとか割り算をうまく取り扱うことができるようにしようという苦労が偲(しの)ばれる方法でもあります。

本節のまとめ

・整数の割り算では「余り」が生じる
・整数の範囲で割り算を定義することは可能だが、足し算、引き算、掛け算と異なり元の数を再現するような演算を定義するためには余りとセットで考えないといけない

割る数と割られる数

前項で、前説明なく「割られる数」とか「割る数」などという言葉を使いました。x☆$y \rightarrow z$という演算を考えたときに、xを「演算☆を適用される数」、yを「演算☆を適用する数」とするのはごく自然な考えでしょう。足し算だったら「(足される数)xに(足す数)yを足す」、引き算だったら「(引かれる数)xから(引く数)yを引く」ですね。

それをふまえて、四則演算の中でも割り算は特別だということを示す議論の最後に、2つの要素を対象とした演算のあり方に目を向けてみることにしましょう。

小学校の復習：割り算の意味

小学生でも、割り算を最初からすんなりと理解するのは難しいのだそうです。小学校では、割り算は2つの意味があると教えるので、それで混乱する児童が出てきてしまうのだとか。その2つの意味とは、次の2つです。

・全体をいくつかに分けたときに、1つあたりの数を求める計算
・全体を1つあたりの数を決めて分けたときに、いくつに分けられるかを求める計算

前者の考え方は、このようなものです。「キャンディが15個あったとします。そしてそのキャンディを配ろうとしている子供たちが5人います。さて、子供たちは何個のキャンディをもらえるでしょうか？」

15個のキャンディを、まず1つずつ5人の子

コラム 時差を簡単に考える方法

剰余計算の応用として、時差を簡単に理解する方法を考えてみましょう。

近隣諸国との時差というのは比較的わかりやすいものです。たとえば中国やマレーシアの時間は、日本から1時間だけ遅れています。ベトナムは2時間遅れです。1時間遅れているということは、日本でお昼を食べ終わって午後の仕事を始めようかという日本時間の13時に、向こうは12時で、今からお昼休みにしようかと考えているだろう、そのような感覚でとらえることができます。

欧州だと(時期にもよりますが)8時間ないしは9時間遅れです。この時間差は、日本の夕方がちょうど朝に相当すると考えれば難しくありません。夕方、ひと仕事終えたなと感じている時間に、向こうではこれから仕事が始まるんだな、という感じです。したがって、欧州と電話会議をしましょうなどというときには、日本は夜、向こうは昼とするパターンが多いようですね。

難しいのが米国です。一番東のEastern Standard Time(EST)でGMT-5ですので、日本のGMT+9から比較すると14時間、遅れていることになります。西海岸のPacific Standard Time(PST)だとさらに3時間遅れて17時間のズレが生じます。

ところが、地球の時差はしょせん24時間以内ということを考えると、割と単純に考えることができます。米国西海岸が日本より17時間遅れているという状況で、日付を無視して考えてみましょう。日付を無視して、日本より何時間先んじているかを考えると、**24 - 17 = 7**です。つまり、日本が7時間、遅れているということになるわけです。これは、日本と欧州の関係性と似ているでしょう。東海岸でも、日本が10時間遅れているという状況です。そう考えると、向こうが夜のときに日本が朝なのだとすぐにわかります。

供に配ります。配り終えると10個のキャンディが残っています。まだ配ることができますね。2巡めとしてまた1つずつ5人の子供に配ります。子供たちはこの時点でキャンディを2個ずつ手にしています。配り終えて、まだ5個残っています。これも配ってしまいましょう。3巡して、ようやくキャンディはなくなりました。子供たちが手にしたキャンディは、それぞれ3つです。

同じ「**15／5＝3**」でも、後者の立場では次のような解釈がなされます。

15個のキャンディから5個取り出したものを用意します。これを、1人めの子供に配ることにします。残りは10個あるので、そこからさらに5個取り出します。これも、次の子供に配りましょう。さて最後に5個残ったものがあるので、それももう1人の子供に配ることができますね。こうして、5個セットのキャンディを、3人の子供に配ることができました。

いずれの計算も、**5 × 3＝15**という計算を逆から見ているだけにすぎません。5行3列に並べたキャンディを、横から分けているか縦から分けているかの違いですが、異なる解釈ができてしまうために混乱が生じてしまうようです。

「20分／500円」という表記は正しいか？

この混乱を避けるために、大人は「単位」を考えます。先の例では前者だと「15[個]／5[人]＝3[個／人]」、後者だと「15[個]／5[個／人]＝3[人]」ということになるでしょう。15「個」を5「人」で分ければ、1人あたり3個「＝個／人」という考えなのか、15「個」を1人あたり5個（「個／人」）で分ければ3「人」に配れるという考えなのか、単位を付けて計算すれば、明確です。

単位だけで、きちんと次元がそろっていることを確認してみましょう。答えの単位は、前者だとそのまま[個／人]となっていますね。後者でも、[個]を[個／人]で割っているので、[個]が約分されて[人]が残ります。

コインパーキングの料金表示は、多くの駐車場で「x分／y円」という書き方になっています（**写真2**）。利用者目線で考えたときに、はたしてこれは正しい書き方でしょうか。

これは「y円／x分」と書くべきではないでしょうか？「あぁ、今いくら持ってるから何分間だけ駐車できるな……」などと、普通は考えません。「用事を済ませるのに何分停めないといけないから、いくらかかるかな？」と考えませんか？

「なんでこんな書き方なんだろう？」と調べてみたら、一般社団法人 日本パーキングビジネス協会（JPB）の時間貸駐車場における表示検討委員会という集まりが、平成26年9月17日付けで「時間貸駐車場における表示・運用に関するガイドライン」という文書を出していました。そのガイドラインに、「看板の料金表示はこのように書くべし」と出ているのが理由でした。みなさんそのガイドラインにしたがって料金表示をしているのですね。

おそらく、100円パーキング時代の名残りだったのでしょう。コインパーキングといえば、かつては「100円を入れたら何分間だけ停められる」というものでした。その感覚であれば時間を100円で割って表示する方法で示す気持ちも、わからぬでもありません。しかし、単価が上がったり、逆に競争激化で価格競争になったりと、さまざまな変化が起こった結果として、このようなヘンな表示になってしまったのでしょう。我々は、この問題から「割る数と割られる数は適切に

▼写真2　あるコインパーキングの料金表示

考えるべし」との教訓を得ることができます。

それにしてもこの駐車場、1時間、車を停めとておくと1,500円もかかります。むちゃくちゃ高いですね(苦笑)。

「0で割ってはいけない」ということ

さて、割られる数と割る数について考えてきましたが、割り算の場合、割る数には1つ、制約条件があります。それは、「0(ゼロ)で割ってはいけない」という条件です。

図3は、Pythonで「5/0」を計算してみた結果です。「ZeroDivisionError: division by zero」というエラーメッセージが表示されています。では、なぜ、ゼロで割ってはいけないのでしょうか。

先に示したキャンディの例をもう一度考えてみましょう。あのとき「$15 / 5 = 3$」は「$5 \times 3 = 15$」という計算を逆から見ているだけ、という話をしました。すなわち、(割られる数)xを、(割る数)yで割ったときに、割り算の答え、つまり、商としてzが得られる、その関係は「$x / y = z$」でもあり「$x = z \times y$」でもあるのです。

ところが、ここで、割る数yが、0だったらどういうことになるでしょう。なお、便宜上、割られる数xは0ではない数としておきます。

先の説明によれば、「$x / 0 = z$」は「$x = z \times 0$」と同じです。ところが、後者の右辺は「0」ですね。なぜなら、いかなる数に0を掛けても、その結果は0だからです。しかし、xは0ではありません。おかしいですね? xが0だとすると、zは「なんでもいい」(不定)ということになり、それもおかしなことになります。

このような関係性を満たすことができないため、割り算には「ゼロで割ってはならぬ」という鉄の掟があるのです。

ところで、よくある誤りの計算例として、「2は1に等しい」という有名な(誤った)証明があります。その証明とは、「aとbが等しいとする。両辺にaを掛けて、そこからb^2を引く。両辺を因数分解して($a - b$)で割ると、$a + b = b$となる。$a = b$だったので左辺は$2b$、すなわち$2b = b$となり、bで割ると$2 = 1$」というものですが(**図4**)、どこに誤りがあるか、わかりますか?

本節のまとめ

- 小学校では割り算の意味を2種類教えるため混乱が生じることがある
- 単位を考えながら計算すると混乱を避けることができる
- ゼロで割ることは定義されていない。どのような状況でもゼロで割るとエラーになってしまう

本章では、「なぜ、コンピュータは割り算が下手なのか!?」の前段階として、小学校時代を思い出しながら「割り算はなぜ難しい?」のかを考えてみました。次章からは、いよいよコンピュータの数値の内部表現を含めて、割り算を考えてみます。SD

▼図3 Pythonでゼロ除算を試したところ

```
iiojun — python — 80×24
Last login: Wed Aug 30 10:01:10 on ttys000
mac3:~ iiojun$ python
Python 3.5.1 (default, Dec  4 2016, 18:02:26)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.38)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 5 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>>
```

▼図4 矛盾した計算

$$a = b$$
$$a^2 = ab$$
$$a^2 - b^2 = ab - b^2$$
$$(a + b)(a - b) = b(a-b)$$
$$a + b = b$$
$$2b = b$$
$$2 = 1$$

第2章 コンピュータ内部の取り扱い

Author 飯尾 淳(いいお じゅん) 中央大学
Mail iiojun@tamacc.chuo-u.ac.jp
Twitter @iiojun

前章では、そもそも割り算という計算それ自体が、ほかの演算に比べると厄介なのだということを確認しました。本章では、コンピュータ内部での数値の取り扱いについておさらいすることで、コンピュータでも割り算を扱うのは難しいのだなということを確認していきましょう。

そもそも2進数とは

ご存じのとおり、現在のコンピュータはすべての情報を0と1で表します。電圧のレベルに対応する0と1による表現が、コンピュータの基礎となっているわけですね。したがって、数値も0と1で表現しなければなりません。

本節では、整数を2進数でどのように表現するか、そしてその演算をどう実現するかについて、まず、簡単な例から考えてみます。

整数の2進数による表現

みなさんが日ごろ生活で利用している数値の表現は、0，1，2から9まで、10個のアラビア数字を用いて表現されています。なお、数字の表現はそのほかにもローマ数字を用いたものや、「1つ、2つ、……」などの序数による表現など、いくつかのバリエーションがありますが、本稿で扱う数字の表現は、通常のアラビア数字による表現を対象とします[注1]。そして、10個の数字を使って表す数字の数え方を、10進法というのでした。

一方、現代のコンピュータは、その内部では0と1で数値を表現します。つまり、2進法です。10進法で表された数値nを2進数に変換するのは簡単です。次の手順で右から順に0か1を並べていけば、その結果が2進数の表現になります。

❶偶数なら0、奇数なら1を置く
❷その数を半分にする。奇数の場合は1を引いて2で割る
❸その数が1なら最上位の桁に1を置いて終了。それ以外なら❶に戻る

例として、10進数で表した「25」を2進数の表現に変換してみましょう。

図1を見てください。まず、25は奇数なので、最下位の桁は「**1**」です。奇数なので1を引いて半分にすると、12ですね。手順を頭に戻って、12は偶数なので0を置くと、「**01**」になりました。さらに半分にします。6になりました。また手順を頭に戻って、6は偶数ですから「**001**」ですね。半分にすると3です。今度は奇数なので、「**1001**」となりました。3は奇数ですから1を引いて半分にすると、1になります。終了条件を満たすので、最上位の桁に1を加えると「**1 1001**」、これで終わりです[注2]。

注1) 小数点は「.(ピリオド)」で表します。ケタの位取りを示す「,(カンマ)」は使いません。余談ですが、スペイン語のように、小数点を「,(カンマ)」で表し位取りを「.(ピリオド)」で表す言語もあります。世界は広いですね。

注2) 以後、5桁以上の2進数には4桁の単位で空白による区切りを入れます。

以上の計算手順を数式的に表すと図2のようになります。n進数においてm桁めの数字がx_mであるとき、その数字は、$\sum n^m \cdot x_m$として表されることを理解していれば、このような手順で正しく変換できていることがわかるでしょう。

ところで、コンピュータで扱う数として、ほかにも8進数や16進数がよく使われます。8進数は、0から7までの8種類の数値を使います。16進数は、0から9までの10個に「a, b, c, d, e, f」の6文字を加えた16種類の文字を数字として使います。8進数や16進数が好まれるのは、8は2の3乗、16は2の4乗であり、それぞれ3桁、4桁の2進数とそれぞれの1桁がちょうど対応するからです。10進数と比べると2進数との親和性が高いというわけですね。

人間が10進数で数えるようになったのは両手の指が10本あったから、という説があるそうです。もし、人間の指がそれぞれ4本ずつだった

▼図1 「25」を2進数にしてみる

```
25→奇数(1)
(25 - 1) ÷ 2 = 12
12→偶数(01)
12 ÷ 2 = 6
6→偶数(001)
6 ÷ 2 = 3
3→奇数(1001)
(3 - 1) ÷ 2 = 1
1→奇数(1 1001)
終了
```

▼図2 2進数に変換する手順の意味

$$
\begin{aligned}
&25\\
&= 24 + 1\\
&= 2\,(12 + 0) + 1\\
&= 2\,(2\,(6 + 0) + 0) + 1\\
&= 2\,(2\,(2\,(2 + 1) + 0) + 0) + 1\\
&= 2\,(2\,(2\,(2 * 1 + 1) + 0) + 0) + 1\\
&= 2^4 * 1 + 2^3 * 1 + 2^2 * 0 + 2^1 * 0 + 1
\end{aligned}
$$

1 1 0 0 1

コラム 生活の中の○○進数

本文で「日ごろよく利用している数字の数え方は10進数」と説明しました。しかし、実はほかの数え方もそれなりに利用しています。

ほとんどのみなさんが、ほぼ毎日接しているのは12進数でしょう。1ダースを単位にした数え方もさることながら、時間の進み方は12進数そのものです。16進数のように新たな記号を採用せず、11、12という表現をしていますが、考え方は、1から12まで進んだらまた1に戻る、12のサイクルで数えられています。24時間時計であれば、24進数ととらえることもできますね。

1月から12月まで、年間を通した月日の流れも、12進数を踏襲しています。逆に短い単位はどうでしょう。1時間は60分なので、分単位、そして秒単位でみれば、60進数で数えられているともいえます。分と秒は0から始まるので、59分の次、59秒の次は、それぞれ桁の繰り上がりが生じて0分、0秒に戻ります。

このように考えていくと、実は、n進数の考え方と、前章で示した整数の割り算は密接な関係にあるということがわかります。トータルの量を、nで割った剰余が一番小さい桁の値になっています。割り算の答えである商がnよりも大きければ、さらに桁溢れが生じて、再度、割り算と剰余の計算を進めていくことになりますね。

たとえば、「156」という数を10進数で表す(もうすでに表されていますが……)ことを考えてみましょう。まず、156を10で割ります。すると、答えは15あまり6ですね。したがって、一番小さい桁の数は「6」です。次に、15をさらに10で割ります。答えは1あまり5。ですので、10の位は5となり、100の位は1です。理解を助けるために10進数でやってみましたが、n進数($n \neq 10$)の場合でも同じです。先に示した2進数への変換も、同じ理屈となっているはずです。確認してみてください。

ら、8進数で数えているようになっていたかもしれません。そうなっていたら、コンピュータサイエンスやIT業界に対する世間の理解も、今よりは進んでいたでしょうか？

簡単な計算

整数の2進数表記がわかったので、次は簡単な演算を試してみましょう。**12 + 13 = 25**という演算を2進数で確認してみます。

先の変換式に従えば、12と13はそれぞれ2進数で「**1100**」と「**1101**」です。足し算は、10進数の計算となんら異なるところはありません。筆算で計算してみましょう（**図3**）。

小学校で習ったように、下の桁から計算していけばOKです。0と1を足すと1、これはいいですね。次の桁、0と0を足して0、これも問題ありません。3つめの桁は1と1の足し算です。1と1を足すと、10進数では2になるところですが、2進数だと2は使えません。したがって、繰り上がりが発生します。1繰り上がり、10となります。下の桁がそのまま降りてきて、3つめの桁は0です。

4桁めも1足す1ですが、さらに繰り上がりの1も足さなければなりません。その結果は11となるので、下の桁の1が4桁めの数値になり、さらに繰り上げの1が5桁めとなります。2進数で**1100**と**1101**の足し算を計算した結果は、**1 1001**、すなわち、10進数で25です。正しく**12 + 13 = 25**の結果と合致していることを確かめてください。

結果が負になる計算をどうするか

では、引き算はどうでしょうか。13から12を引くのは簡単ですね。**1101**から**1100**を引いたら残りは1、これは10進法で13から12を引いたら1となるのと合致します。では12から13を引いたら、どうなるでしょう？

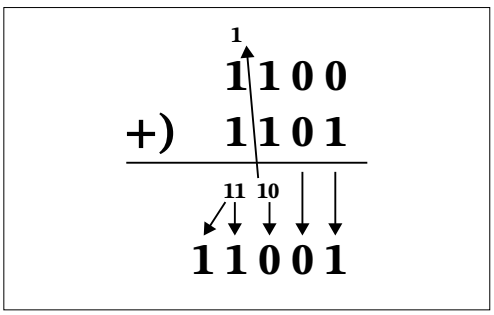
実は、ここまで「整数の2進数表記」と書いてきましたが、整数ではなく正の数と0、あるいは、0を含む定義の自然数を2進数で表現したらどうなるか、という話しかしていませんでした。

そして、整数です。当然ながら、負の数も扱います。しかし、コンピュータの内部で表現できる記号としては「0」と「1」しか扱えません。マイナスの記号は扱えないのです。さて、どうしましょう？

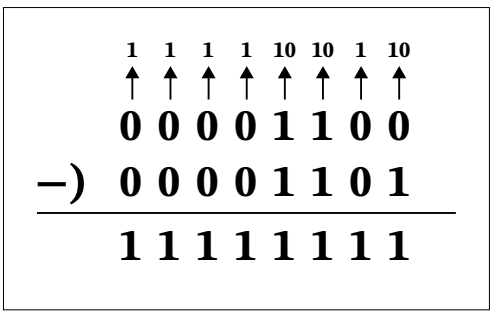
マイナスになる引き算の例として、**12 - 13**を考えてみます。なお、ここで扱う整数は8ビットで表現できる範囲のものとして考えることにしましょう。Cでいえばchar型に相当する大きさです。前章で解説したとおり、理論上は、整数の集合として無限大の大きさを持つものを考えますが、コンピュータで処理する際の一般的な処理系では、数を有限なものとして扱います。ほとんどの計算はそれで用が足りるからです。

さて、8ビットで表現した**12 - 13**を筆算で計算してみましょう（**図4**）。一番下の桁は0から1を引きたいところですが0よりも1のほうが大きいのでそのままでは引けません。10進数での引き算と同様に、上の桁から借りてきましょう。

▼図3　2進数の足し算



▼図4　結果が負になる2進数の引き算



10 - 1は計算できますね。その答えは9……ではなく2進数なので、1です。

次の桁、0から0を引く、ではなく、既に負債があります。下の桁に1を貸した状態でした。しかし、実は自分自身はもともと0だったので、さらに上の桁から借りてこなければなりません。上の桁から借りてきて、10となりますが、そこから下の桁に融通したぶんの1を引くと、**10 - 1 = 1**となります。その状態で、やっと、1から0を引くことができるようになりました。その答えは1です。

下から3桁め、下の桁に融通してやったので0になっています。0から1は引けません。一番下の桁と同様の手順で上の桁から1を借りてきて、**10 - 1 = 1**と計算します。その次の桁も同様ですね。5桁め以降は、下から2桁めと同じロジックです。最後の桁だけは、もはやどうしようもありません。架空の9桁めを想定し、そこから借りてくることにしましょう。その結果、**12 - 13 = -1**の結果は、2進数表記で「**1111 1111**」ということになりました。

なお、上記の計算は、仮の9桁めが存在するような状態、すなわち、2進数での**1 0000 0000**、10進数では256を足した状態で引き算を計算するとして、**図5**のような計算を行っていることに相当します。ここで、$2^2 = 2^1 + 2$や$2^3 = 2 \times 2^2$、また$2^8 = 2^7 + 2^6 + 2^5 + 2^4 + 2 \times 2^3$と展開していることに注意してください。上の桁から借りてきたり下の桁へ融通したりというやりとりは、数式上、**図5**のように表すことができます。

本節のまとめ

- コンピュータのなかでは数値は2進数で表される
- 我々が使っている10進数の整数を2進数の表現に変換する手順は簡単である
- 足し算と引き算は10進数での計算と同様の方法で実現可能。結果が負の数になるような引き算は、最上位のもう1つ上の桁を仮定して計算する

負の数の取り扱いと乗除算

足し算と引き算ができるようになったので、本節では掛け算と割り算までの解説を試みます。その前に、負の数の取り扱いをきちんと整理しておきましょう。

負の数の補数表現

前節では、12から13を引く計算をしてみたことで、-1は「**1111 1111**」と表現されることがわかりました。余裕があれば、**1111 1111**(10進数で-1)と**0000 1101**(10進数で13)を足したら**0000 1100**(10進数で12)となることを確認してみましょう。なお、8ビットという制限がなければ、**1111 1111**と**1101**を足した結果は、**1 0000 1100**となるはずです。

それにしても、-1を**1111 1111**で表すのは、一見、奇妙に見えませんか。実はこれは「補数」という概念を用いることにより、マイナスの記号を使わずに負の数を表現できるようにする工夫なのです。nビットの数値表現において、ある数xの補数とは、xと加算すると2^nとなるような数のことをいいます。8ビットの整数であれば、xの補数とxを足すと、$28 = 256$となるような関係です。

さてここで、2^8は2進数で**1 0000 0000**と表されることを思い出してください。8桁の2進数を

▼図5　上の桁から借りてくる計算の数式的なイメージ

$$
\begin{aligned}
256 + 12 - 13 &= (2^8 + 2^3 + 2^2) - (2^3 + 2^2 + 1) \\
&= (2^7 + 2^6 + 2^5 + 2^4 + 2 \times 2^3 + 2 \times 2^2 + 2^1 + 2) - (2^3 + 2^2 + 1) \\
&= 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 1
\end{aligned}
$$

取り扱う処理系では、桁溢れが生じ、**1 0000 0000**は**0000 0000**すなわち0と同一視されます。ということは、yをxの補数、つまりxに足して0となるような数とすれば、$x+y=0$という関係性から$y=-x$が導かれるということです。このように、xの正負を反転させた$-x$をxの補数で表すのは、極めて妥当な選択なのだということがわかります。

補数の求め方は簡単です。8ビットで表された数値の、1と0を反転させたものを考えます。たとえば、10進数の25であれば、2進数では**0001 1001**と表現されるので、そのビットを反転させた数値は**1110 0110**です。この2つの数を足すと、**1111 1111**になります。これに1を足すと桁溢れが生じて**0000 0000**になりますね。すなわち、ある数xの補数は、「ビットを反転させたものに1を足す」という操作で求めることができます。25の補数は、**1110 0111**です。**0001 1001**と足し合わせて**1 0000 0000**になることを確認してみてください。

符号付き整数の実態

8ビットの符号付き整数が扱うことのできる範囲は、-128から127までです。なぜなら、128以上の数は最上位ビットが1なので、その補数の最上位ビットは0でなければなりません。しかし、最上位ビットが0ということはその数は127以下のいずれかの数値と同じになってしまいます。

▼リスト1　file1.c

```
#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char** argv) {
  unsigned char x = 0;

  while(1) {
    printf("unsigned: %4d, signed: %4d\n", x, (char)x);
    if (x++ == 255) break;
  }

  exit(0);
}
```

このような理由から、符号付き整数の最上位ビットは正負のいずれかを表していると考えることができるでしょう。最上位ビットが0のときは正の数、最上位ビットが1のときは負の数を表していると考えられます。

リスト1のプログラムは、0から255までの符号なし整数(unsigned char)を順番にインクリメントし、符号なし整数と符号付き整数でどのように扱われるかを確認するためのものです。

このプログラムをコンパイルして実行してみると、**図6**のような出力が得られます(長いので、一部、省略しています)。

0から127までは符号なしでも符号付きでも同じとなり、符号なしで128から255までは、符号付きだと負の数になっていることがわかります。

掛け算と割り算

長くなってきましたが、いよいよ、掛け算と割り算です。これらの計算は、ビットシフトと加減算の組み合わせで実現できます。

ビットシフト演算とは、2進数で表現されたビット列を、右左に何桁かずらす操作のことで

▼図6　コンパイルして実行してみる

```
$ gcc file1.c -o file1
$ ./file1
unsigned:   0, signed:   0
unsigned:   1, signed:   1
unsigned:   2, signed:   2
unsigned:   3, signed:   3
unsigned:   4, signed:   4
(..省略..)
unsigned: 123, signed: 123
unsigned: 124, signed: 124
unsigned: 125, signed: 125
unsigned: 126, signed: 126
unsigned: 127, signed: 127
unsigned: 128, signed: -128
unsigned: 129, signed: -127
unsigned: 130, signed: -126
unsigned: 131, signed: -125
unsigned: 132, signed: -124
(..省略..)
unsigned: 251, signed:  -5
unsigned: 252, signed:  -4
unsigned: 253, signed:  -3
unsigned: 254, signed:  -2
unsigned: 255, signed:  -1
$
```

す。これまで何度も例題に出した25を8ビットの2進数で表現すると、**0001 1001**でした。これを左に1ビットシフトさせると、**0011 0010**です。

ところで、**0001 1001**は、十進数において2のべき乗で表現すると、$25 = 2^4 + 2^3 + 2^0$で表されるというわけでした。すなわち、各nビットめの1が立っているところだけ、2^nを足し合わせた数になっているということです。これを左に1ビットずらすということは、4を5に、3を4に、0を1にするという状況と考えることができるでしょう。つまり、次のように操作するということです。

$2^4 + 2^3 + 2^0$ —(1ビット左にシフト)
→ $2^5 + 2^4 + 2^1$

ところが、1ビット左にシフトした値は、$2^5 + 2^4 + 2^1 = 2 \times (2^4 + 2^3 + 2^0)$にほかなりません。すなわち、左に1ビットずらすという操作は、その数を2倍するということに相当します。また、右に1ビットずらす操作は2で割るという計算を行うことに相当します。

例題として、**25 × 6**を考えてみましょう(**図7**)。それぞれ、2進数の表現は**1 1001**と**110**です。掛ける数6を、一番下の桁から順番に走査していきます。0であれば何もしません。1であれば、その桁の数だけ左にビットシフトした数値を足していきます。最後まで走査し終えたら、すべて足し合わせた数が答えです。$6 = 2^2 + 2^1$

▼図7 2進数の掛け算

```
   11001
×)   110
-------
  110010   ← 左に1ビットシフト
 1100100   ← 左に2ビットシフト
-------
 10010110
```

ですので、6を掛けるということは、掛けられる数の25を左に2ビットシフトしたものと1ビットシフトしたものを足し合わせればよいわけですね。

割り算はその逆です。割る数を、あらかじめ割られる数と同じ長さになるまで左シフトしておき、割られる数よりも大きかったら0、小さかったら1を立てて引き算をして、順次、右にシフトしていけばよいわけです。やり方は10進数での除算とまったく同じで、計算途中で引いていく値を求めるのに部分的な乗算を行う必要もなく、右方向のビットシフト演算をするだけですので、10進数の割り算より簡単かもしれません。

本節のまとめ

- 負の数は補数表現で表すことができ、その求め方は「各ビットを反転したものに1を足す」である
- 符号付き整数の最上位ビットが1のとき、その数は負の数である
- 乗算と除算は左右のビットシフトと加減算の組み合わせで実現可能である

浮動小数点

コンピュータの中で扱われている2進数を対象にした四則演算がどのように実現されているのかを、ここまで駆け足でおさらいしてきました。しかし、いずれも整数を対象とした演算にとどまっています。前章で、割り算の結果は整数とは限らない、だから割り算は難しい計算なのだという話をしました。コンピュータが整数しか扱えないのでは、割り算はおろか、複雑な計算などできようもありません。

もちろん、2進数で実数を扱う枠組みは用意されています。浮動小数点というしくみです。ここでは、その中で最もシンプルな単精度の浮動小数点を扱います。

固定小数点による実数とその課題

2進数だからといって、整数だけしか表現できないというわけではありません。例として、10進数で13.25という数を考えてみましょう。

10進数の13は、2進数では**1101**でした。これは、$13 = 8 + 4 + 1 = 2^3 + 2^2 + 2^0$ということを意味しているのでしたね。同じように、0.25はどう表されるでしょうか。0.25は、$\frac{1}{2^2}$、つまり$2^{(-2)}$と表すことができます。したがって、13.25は2のべき乗を組み合わせた表現によれば、次のように表現されるということがわかります。

$$13.25 = 2^3 + 2^2 + 2^0 + 2^{(-2)}$$

これを、素直に2進数で表記すると、**1101.01**と表現できるでしょう。しかし、ちょっと待ってください。コンピュータで扱える数値は0と1のみ。小数点を表現する手段はありません。

その問題を解決する方法として、小数点の場所をあらかじめ決めておくという考え方があるでしょう。これが、固定小数点の考え方です。

8ビットの実数を想定したとして、仮に右から3桁を小数部に使うと決めたとします。しか

コラム 2進数で考える(部分集合の数を求める問題)

「配列を任意の部分集合に分割する。配列の要素数をn個としたとき、組合せの数は何通りになるか？」という問題の答えをすぐに求めることができますか？

問題は、図8の例に示すように、一列に並べられた何かをグループに分けるやり方は何通りあるでしょうかというものです。

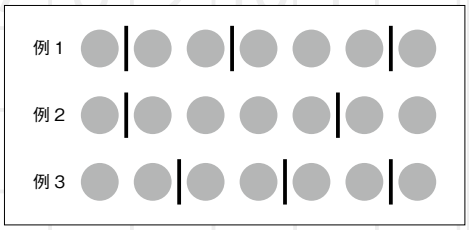
いろいろな考え方があるでしょう。

プログラマならわりとすぐに思いつくのではないかという簡単な方法は、再帰法によって求めるやり方です(図9)。単純化のためにまったく分割しないケースも含むこととして要素数がnのときの組合せの数を$f(n)$とすると、$f(n)$は$f(n-1)$のときの2倍となることがわかります。したがって、再帰的に計算していくと、$2^{(n-1)}$となります。まったく分割しないケースは除くことにすれば、組合せの数は$2^{(n-1)}-1$です。

ところで、この問題は、2進数の応用問題と考えると、一瞬で解答に辿り着くことができるのです(図10)。

一列に並べた配列の要素に、区切りを入れてグループに分割すると考えましょう。区切るところを「1」、区切らないところを「0」とコーディングします。n個の要素が並べられているとき、区切る個所は間の$n-1$個なので、これは「$n-1$桁の2進数のうち1つでもビットが立っている数は何個あるか？」という問題と同じです。$n-1$桁の2進数は$2^{(n-1)}$個あるので、1つでもビットが立っている数の個数は、そこから0(すべてのビットが0)を除いた$2^{(n-1)}-1$個です。

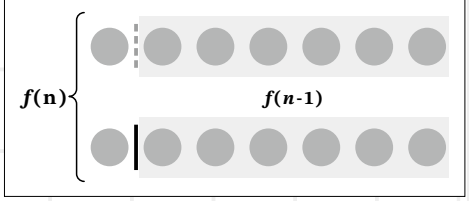
▼図8 部分集合に分ける例



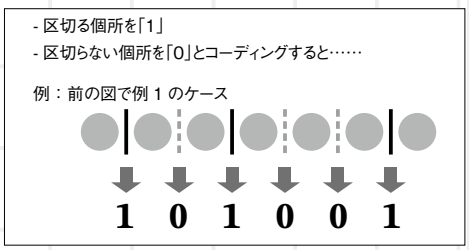
▼図9 再帰的に考える方法

$f(n) = 2 \times f(n-1) = 2 \times 2 \times f(n-2) - \cdots = 2^{n-1} f(1)$

$f(1) = 1$



▼図10 2進数で考える方法



し、このやり方はあまりうまくありません。小数点以下に3ビットもとってしまうと、残りは5ビットしかありません。5ビットで表すことができる数値はたかだか32個。符号付きにしたとしたら、正の数としては15までしか表すことができません。

小数点以下にも課題が残ります。3ビットで表せる最小単位は$2^{(-3)}$、これはたかだか0.125です。小数部が3ビットしかないような表現では、0.125よりも細かい数字を表すことができません。

もちろん、全体が8ビットしかないことが問題なので、一般的な32ビットやその倍の64ビットにすればある程度は解決する制約ですが、数値が取ることのできる範囲を大きくしようと、小数点を右側に設定すれば小数点以下の粒度が荒くなり、細かい粒度で数を表現したいと小数点を左側に設定すれば、表すことができる数の大きさが限られるというトレードオフが発生してしまいます。そのため、現在のコンピュータにおけるほとんどの処理系では、固定小数点ではなく浮動小数点の考え方によって実数を表現しています。

指数表記と浮動小数点

浮動小数点の考え方は、数値の指数表記に基づいています。指数表記とは、数値を「仮数、基数、指数」の組み合わせで表現するやり方です。

具体例で示しましょう。123.45という数を考えます。小数点を移動して、**$123.45 = 1.2345 \times 10^2$**という表記にします。ここで、1.2345を仮数部、10が基数、2乗しているところの2を指数部といいます。この表記は、非常に大きな数や、逆にとても小さな数でもそれなりの精度で表現できるという特徴をもっています。電卓で計算結果がとても大きくなってしまったときや、小さくなってしまったときに、**1.23E+15**とか、**4.56E-15**などと表されることがありますが、これは、基数が10であることを省略した表現です[注3]。なお、仮数部の最上位は10進数の一桁で表すため、1から9の数字でなければなりません[注4]。

さて、先の例では10進数の指数表記を考えました。コンピュータでは2進数を扱います。2進数でも同様に指数表記を考えることができます。10進数の25は2進数で**1 1001**でした。右にシフトさせるという操作は2で割ることと同じだったことを思い出しましょう。右に4回シフトしたものに2^4を掛ければ元どおりになります。つまり、**$1\ 1001 = 1.1001 \times 2^4$**と指数表記で表すことができました[注5]。

先ほどの13.25はどうでしょうか。13.25を(固定小数点で)2進数に直すと**1101.01**だったことがわかっていれば、あとはほぼ同じです。**1101.01**を右に3回シフトすれば**1.10101**となるので、**$1101.01 = 1.10101 \times 2^3$**と表現されました。

この指数表記の考え方に基づいて、コンピュータの内部での実数の取り扱いが定められています。小数点があちこち移動するので、固定小数点に対して浮動小数点と呼ばれます。

単精度浮動小数点(IEEE 754)

浮動小数点をどのように表記するか、コンピュータの黎明期にはさまざまな方法が提案されていたそうですが、現在は、ほとんどの処理系でIEEEという団体が定めたIEEE 754という標準にしたがって実装されています(**図11**)。IEEE 754では、何ビット用いるかによって単精度や倍精度などいくつかの形式が定義されています。本稿では32ビットを使う単精度で解説しますが、倍精度であっても考え方は同じです。

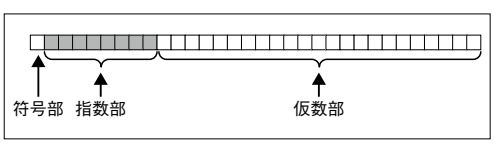
単精度浮動小数点は4バイト、すなわち、32

注3) EはExponentの頭文字です。

注4) 0は使いません。その理由はなぜでしょうか、少し考えると自明ですので、頭の体操と思って考えてみてください。

注5) 仮数部のみ2進数表現としています。

▼図11 IEEE 754による単精度浮動小数点の定義



ビットを使います。先頭は符号部(1ビット)、正なら0、負の数なら1です。

続く8ビットは指数部です。ただし、指数部に収める数値は指数部の値に127を足した数にします。この127をバイアス値といい、127を足すことによって-127から128までの数値を表現します。つまり、8ビットの整数は0～255まで表現できるので、指数部に記録された数から127を引き実際の値に戻すことで、-127～128の値を表現できるようにしているのです。

残りの23ビットが仮数部です。ただし、仮数部の先頭の1は省略します。なぜならば、仮数部の先頭は必ず1になるからです[注6]。

以上のルールを適用して、13.25の浮動小数点表現を考えてみましょう。13.25は、2進数の指数表現では**1.10101 × 2^3**と表されるのでした。正の数ですので符号部は0です。指数部は3ですが、127を足すと130です。130を2進数で表すと**1000 0010**です。仮数部から先頭の1を省いたものは**1 0101**、残りは0で埋めて23ビットにすると、**101 0100 0000 0000 0000 0000**となりますね。以上を並べて13.25の単精度浮動小数点による2進数表記は、**0100 0001 0101 0100 0000 0000 0000 0000**となるはずです。

では、実際にプログラムを動かして確認してみましょう。**リスト2**のプログラムは、float型で定義された変数を表示するだけのほとんど意味のないプログラムですが、とりあえずこれを使います。

これを、gccでコンパイルしたあと、lldbデバッガ(gdbでも同様)で中身を覗いてみます(**図12**)。

デバッガを起動したら、7行めにブレークポイントを設定します(`b 7`)。実行(`r`)し、ブレークポイントで停止したら、変数***x***の中身をみてみましょう。pコマンドを使い、「`p x`」とすれば中身が13.25であることを確認できます。次に、pコマンドのスイッチで「`/t`」を指定します。これは「値を2進数で表示せよ」という指定です。「`p/t x`」とすると、変数***x***の内容を2進数で表示してくれます。先頭の`0b`は2進数であることを示しています。先ほどの結果と比べてみましょう。同じ結果になっていますね？SD

本節のまとめ

- 固定小数点はシンプルだがいろいろと課題も多く、実際には使われていない
- 浮動小数点は指数表記の考え方に基づいている。なお、2進数なので基数は2である
- IEEE 754のフォーマットに従い、符号部、指数部、仮数部を並べることで、実数を2進数で表すことができる

注6) 10進数の指数表現において、仮数部の先頭は1～9であって0は使わなかったことを思い出してください。2進数だと、0を使わない仮数部の先頭は必ず1となるでしょう。

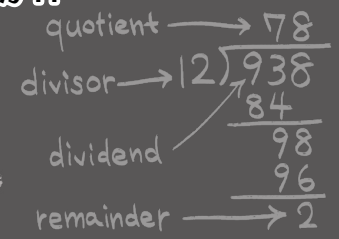
▼リスト2　file2.c

```
#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char** argv) {
  float x = 13.25;
  printf ("%f¥n", x);
  exit (0);
}
```

▼図12　デバッガを用いた浮動小数点の確認

```
tmp — lldb -f file2 • file2 — 80×25
mac3:tmp iiojun$ gcc file2.c -g -o file2
mac3:tmp iiojun$ lldb -f file2
(lldb) target create "file2"
Current executable set to 'file2' (x86_64).
(lldb) b 7
Breakpoint 1: where = file2`main + 42 at file2.c:7, address = 0x0000000100000f5a
(lldb) r
Process 10018 launched: '/private/var/tmp/file2' (x86_64)
Process 10018 stopped
* thread #1: tid = 0xd3005, 0x0000000100000f5a file2`main(argc=1, argv=0x00007ff
f5fbff550) + 42 at file2.c:7, queue = 'com.apple.main-thread', stop reason = bre
akpoint 1.1
    frame #0: 0x0000000100000f5a file2`main(argc=1, argv=0x00007fff5fbff550) + 4
2 at file2.c:7
   4    int
   5    main(int argc, char** argv) {
   6      float x = 13.25;
-> 7      printf ("%f\n", x);
   8      exit (0);
   9    }
(lldb) p x
(float) $0 = 13.25
(lldb) p/t x
(float) $1 = 0b01000001010101000000000000000000
(lldb)
```

第3章 CPUレベルで考える実装上の話題



Author 飯尾 淳(いいお じゅん) 中央大学
Mail iiojun@tamacc.chuo-u.ac.jp
Twitter @iiojun

四則演算の中でも割り算はとくに難しい計算だという話と、コンピュータで演算処理をどう扱うかという基礎がわかったところで、いよいよ、コンピュータは割り算が苦手というその理由に踏み込んでいきましょう。本章では、CPU(Central Processing Unit:中央演算装置)が数値演算をどう扱うかという話と、実際に割り算は遅いのかという話の核心に迫ります。

CPUにおける演算

結局のところ、実際の演算を処理するのはCPUの命令(機械語)レベルの話です。そのレベルから、割り算は遅いという事実を受け止め、対策を考えるのが得策というものでしょう。

DIV命令とさまざまな工夫

何はともあれ、まずはCPUの命令レベルで「割り算はコスト[注1]のかかる処理だ」ということを確認しましょう。Intelのx86系CPU[注2]の浮動小数点関係のサブセットであるx87命令セットの仕様をみてみましょう。**表1**は浮動小数点命令に関する記載から四則演算部分を抜粋したものです。

注1) 費用ではなく、時間・労力などの負担と捉えてください。
注2) 本章ではおもにIntel x86系CPUを元に、話を進めます。

▼表1 四則演算に関する浮動小数点命令の例

命令	動作	レイテンシ	スループット
fadd	浮動小数点加算	5	1
fsub	浮動小数点減算	5	1
fmul	浮動小数点乗算	7	2
fdiv(単精度)	浮動小数点除算	23	23
fdiv(倍精度)		38	38
fdiv(拡張精度)		43	43

表中のレイテンシの数値は命令の実行の開始から完了までにかかる時間[注3]のこと。スループット[注4]の数値はCPUのパイプラインを占有することによって後続の命令をブロックする時間です。これを見ると、レイテンシ、スループットともに、割り算の処理は極めて時間がかかることが明白です。加減算、掛け算と比較すると、3倍から8倍以上もの手間がかかるようです。

一方で、CPUには逆数を求める命令も用意されています。$\boldsymbol{x}$を$\boldsymbol{y}$で割るという演算は、$\boldsymbol{x}$に$\boldsymbol{y}$の逆数($\frac{\boldsymbol{1}}{\boldsymbol{y}}$)を掛けると解釈できます。つまり、逆数を求める演算を簡易に実現できれば、逆数を求めて掛け算を行うことで、割り算を比較的低コストに実現できるというわけです。実際の計算においては、逆数を必要とする場面も少なくないため、逆数を求める演算を独立して用意しておくことにはそれなりの意義があると言えるでしょう。

逆数を求める演算は、パラメータが1つであることから問題もシンプルで、低コストなアルゴリズムがいくつか提案されています。平方根の逆数を求めてそれを2乗する方法(Fast inverse square root algorithm)

注3) クロック数と呼ばれ、CPUの動作周波数によって1クロックの時間が計算できる。
注4) Intelの仕様書による用語定義より。

や、除算テーブルを用いて逆数の近似値を求める方法、ニュートン法[注5]で求める方法などが利用されているようです。

数値演算プロセッサとGPU

そもそも、複雑な数値演算には特別なハードウェアが必要だという認識は、CPUというものが作られた当初から考えられていたことでした。現在のIntelアーキテクチャの基礎を成す8086 CPUの時代から、その考え方は存在していました。当時、コプロセッサ(数値演算プロセッサ。浮動小数点処理装置(Floating Point Unit:FPU))として8087という型番のハードウェアが、8086のCPUとは別にチップとして存在していました。ややこしい数値演算はコプロセッサが行っていたのです。

今回参照したインテルの仕様書にも、x86ではなく、x87浮動小数点命令という名称で記載されており、その名残を確認できます。CPUが拡張されるとともに、コプロセッサの機能はCPUの内部に取り込まれていきました。

しかし、昨今の状況はどうでしょうか。そう、GPU(Graphics Processing Unit)です。GPUはそもそも画像処理用として誕生しました。グラフィクスのための演算を処理するためのデバイスです。しかし、世間には意外にもグラフィクス演算と類似の処理をする需要が溢(あふ)れていたようです。最近では、GPGPU(General Purpose GPU)と称し、汎用演算をGPUで処理しようというやり方が当たり前になりました。そのためのライブラリなども整備されています。もはやGPUというよりはコプロセッサと呼ぶべきなのではないでしょうか。

最近では、AIブームに乗って、ディープラーニング(深層学習)に特化した専用のプロセッサ「Tensor Processing Unit(TPU)」をGoogleが開発しています。ディープラーニングで用いる多層・多ノードのDeep Neural Network(DNN)では、多数の積和演算とアクティベーション関数[注6]の組み合わせという典型的な計算パターンを多用します。このような定型化された演算が必要なシーンは、専用のハードウェア化によって処理の高速化を図る手法が適しています。

かつて、信号処理では同じように積和演算を多用するシーンがありました。信号処理装置に使われていたDigital Signal Processor(DSP)では、CPUと同様の機能に加えて、多数のデータを対象として積和演算を1命令で実施できるような専用の命令が備えられていたものです。このような技術の変遷を見ていると、歴史は繰り返すという言葉が、頭に浮かんできますね。

本節のまとめ

- CPUレベルでDIV命令は遅い。そのためにいろいろな工夫が重ねられている
- 複雑な計算をするコプロセッサ機能はCPUに取り込まれた。しかし、似たような考え方はGPUやTPUなどにしぶとく生き残っている

試してみよう

それでは実際に、割り算はどのくらい遅いのかを試してみることにしましょう。**リスト1**は、1から9,999まで2組の数をそれぞれ計算する「だけ」のC言語で書かれたプログラムです。

file3_add.cをコピーしたファイルを作り、そ

注5) aの逆数xを求める漸化式。$x_{n+1}=x_n(2-ax_n)$

注6) ニューラルネットワークで使用される関数。

▼リスト1 file3_add.c

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_ITER 10000

int
main (int argc, char** argv) {
    int i, j;
    double x, y, z;
    for (i = 1; i < MAX_ITER; i++)
        for (j = 1; j < MAX_ITER; j++) {
            x = (double)i;
            y = (double)j;
            z = x + y;
        }          ↑この演算子を加減乗除で作成
    exit (0);
}
```

れぞれにおいて足し算をしている部分($z = x + y$の部分)を、引き算、掛け算、割り算に変更したものを、それぞれfile3_sub.c、file3_mul.c、file3_div.cとしておきます。

それぞれ、コンパイルして実行させてみましょう(**図1**)。timeコマンド[注7]で実行時間を測ります。科学的に計測するには何回か試行して平均値をとるべきですが、今回はとりあえず1回の試行結果を示します。みなさんもご自身で何回か試して確認してみてください。

このように、割り算を使ったプログラムだけ、ほぼ倍の時間がかかっています。もちろん割り算以外の部分に関するオーバーヘッドがあるので割り算は倍程度遅いと結論付けることはできませんが、割り算だけとくに時間がかかる処理であるということは理解できるでしょう。

file3_div.cがどのようなコードに変換されているか、もう少し詳細に確認してみます。**gcc -S file3_div.c**としてアセンブラのコードを出力すると、**リスト2**のようなコードが生成されていました。

注7) 指定したコマンドの実行時間を表示するコマンド。realは呼び出しから終了まで、userはコマンド自体の処理時間、sysはOSが処理した時間。

file3_add.s、file3_sub.s、file3_mul.sも同様にして作成し、比較すると、それぞれのファイルでは**divsd**命令の代わりに**addsd**、**subsd**、**mulsd**命令が使われていることがわかります。それ以外の部分はすべて同じです。前節で示し

▼図1 各演算のベンチマーク

```
$ gcc file3_add.c -o file3_add
$ gcc file3_sub.c -o file3_sub
$ gcc file3_mul.c -o file3_mul
$ gcc file3_div.c -o file3_div
$ time ./file3_add

real    0m0.499s
user    0m0.492s    ←足し算にかかった時間
sys     0m0.003s
$ time ./file3_sub

real    0m0.501s
user    0m0.494s    ←引き算にかかった時間
sys     0m0.003s
$ time ./file3_mul

real    0m0.558s
user    0m0.552s    ←掛け算にかかった時間
sys     0m0.003s
$ time ./file3_div

real    0m1.076s
user    0m1.069s    ←割り算にかかった時間
sys     0m0.003s
$
```

コラム Pentium FDIVバグの話

1994年の10月、当時インテルの主力製品であったPentiumプロセッサにバグがあることが報じられました。そのバグとは、x87浮動小数点命令の除算命令、FDIVを実現する回路に誤りがあり、ある数値で割り算を実施するとその結果として誤った数値を返すというものでした。FDIV命令にバグがあるということで、Pentium FDIVバグとも呼ばれています。

このバグそれ自体は、ごくまれなケースで発生するものとして、些細なものであるとインテルは主張しました。確かにその状況が発生する確率は非常に低いものだったようです。しかし、完全性が求められるコンピュータとしては、信頼性に不安を抱かせるに値する大きな問題でした。90年代半ばといえば、インターネットの普及とともにパソコンそれ自体が一般消費者に受け入れられるようになっていた時代です。日本ではWindows 95が牽引役となり、パソコンが家電製品並みに扱われるようになってきたタイミングでした。そのような状況に水を差す出来事だったと記憶しています。

このバグ(エラッタ)は、その後、ソフトウェア的に回避するという対応が選ばれました。この失敗に懲りたのかどうかは知りませんが、今ではCPU内部のマイクロコードを修正できるような手段が提供されるようになっています。ハードウェア的なエラッタが生じたとしても、BIOSやOSがパッチを適用することによりそれを回避できるようなしくみが用意されているのです。

▼リスト2 file3_div.s

```
        .section        __TEXT,__text,regular,pure_instructions
        .macosx_version_min 10, 12
        .globl  _main
        .align  4, 0x90
_main:                                  ## @main
        .cfi_startproc
## BB#0:
        pushq   %rbp
Ltmp0:
        .cfi_def_cfa_offset 16
Ltmp1:
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
Ltmp2:
        .cfi_def_cfa_register %rbp
        subq    $48, %rsp
        movl    $0, -4(%rbp)
        movl    %edi, -8(%rbp)
        movq    %rsi, -16(%rbp)
        movl    $1, -20(%rbp)           i = 1
LBB0_1:

        cmpl    $10000, -20(%rbp)       i < MAX_ITER
        jge     LBB0_8
## BB#2:
        movl    $1, -24(%rbp)           j = 1
LBB0_3:

        cmpl    $10000, -24(%rbp)       j < MAX_ITER
        jge     LBB0_6
## BB#4:
        cvtsi2sdl       -20(%rbp), %xmm0 xmm = (double)i
        movsd   %xmm0,  -32(%rbp)       x = xmm
        cvtsi2sdl       -24(%rbp), %xmm0 xmm = (double)j
        movsd   %xmm0,  -40(%rbp)       y = xmm
        movsd   -32(%rbp), %xmm0        xmm = x
        divsd   -40(%rbp), %xmm0        xmm = xmm / y
        movsd   %xmm0, -48(%rbp)        z = xmm
## BB#5:
        movl    -24(%rbp), %eax         EAX = j
        addl    $1, %eax                EAX = EAX + 1
        movl    %eax, -24(%rbp)         j = EAX
        jmp     LBB0_3
LBB0_6:
        jmp     LBB0_7
LBB0_7:
        movl    -20(%rbp), %eax         EAX = i
        addl    $1, %eax                EAX = EAX + 1
        movl    %eax, -20(%rbp)         i = EAX
        jmp     LBB0_1
LBB0_8:
        xorl    %edi, %edi              EDI = 0
        callq   _exit                   exit( EDI )
        .cfi_endproc

.subsections_via_symbols
```

※アセンブラコードのコメント部分は、解説のため省略しました。
※コード左側がラベル(ジャンプ先)、インデントされた順に、オペランド(コマンド)、オペコード(複数の場合はカンマ区切り)です。
※反転部分はC言語プログラムの対応部分です。解説上、CPU内部メモリ(レジスタ)は、xmm、EAX、EDIとしてあります。
※オペランドのjmp(無条件ジャンプ)、jge(Jump Greater than or Equal：大きいか等しい場合ジャンプ)には矢印を入れました。

たfaddやfdivとは違う命令[注8]ですが、仕様によれば、divsd命令もレイテンシとスループットともに、それ以外とは格段に大きな値になっていることを確認することができます。SD

注8）厳密には、fadd、fsub、fmul、fdivなどはx87命令セットの命令です。addsd、subsd、mulsd、divsdなどはSSE（Streaming SIMD Extensions）のSIMD拡張命令で、Pentium Ⅲから実装されました。

本節のまとめ

- 割り算は、インストラクションレベルでコストの高い計算である、ということを改めて確認した
- C言語で作成されたプログラムが、実際にどのようにアセンブラで展開されているかを確認した

コラム コンパイラによる最適化

割り算はコストがかかる計算であるということを確認したので、最適化を考えてみましょう。gccに-Oオプションを付けて、コンパイル時に最適化処理を加えてみます。

```
$ gcc -O file3_div.c -o file3_div_opt
$ time ./file3_div_opt

real    0m0.006s
user    0m0.001s
sys     0m0.002s
$
```

おやおや？　ものすごく最適化されましたよ。圧倒的な速さです。生成されたコードを見てみましょう（リスト3）。

実は、「何もしない」コードに変換されてしまっていたのでした。もともと、file3_div.cのなかで、計算結果は変数zに格納されるものの、その後まったく利用されず使い捨てにされていました。コンパイラの最適化によって、「なかったこと」にされたわけですね。

▼リスト3　file3_div_opt.s

```
        .section        __TEXT,__text,regular,pure_instructions
        .macosx_version_min 10, 12
        .globl  _main
        .align  4, 0x90
_main:                                  ## @main
        .cfi_startproc
## BB#0:
        pushq   %rbp
Ltmp0:
        .cfi_def_cfa_offset 16
Ltmp1:
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
Ltmp2:
        .cfi_def_cfa_register %rbp
        この間のコードがすべてなくなっている
        xorl    %edi, %edi
        callq   _exit
        .cfi_endproc

.subsections_via_symbols
```

バックナンバーのお知らせ　BACKNUMBER

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

本誌バックナンバー（紙版）はお近くの書店に注文されるか、下記のバックナンバー常備取り扱い店にてお求めいただけます。また、「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）や、e-hon（http://www.e-hon.ne.jp）にて、Webから注文し、ご自宅やお近くの書店へお届けするサービスもございます。

2017年10月号

第1特集
Gitのキホン
GitHub、Bitbucket、GitLab、GitBucket

第2特集
脆弱性スキャナVuls入門
システムのセキュリティチェックをもっと楽に

一般記事
・ネットワークエンジニア　その技術の極め方
・身近なメールで使われている技術をネットワークコマンドで体験してみよう

定価（本体1,220円+税）

2017年9月号

第1特集
Web技術【超】入門
いま一度振り返るWebのしくみと開発方法

第2特集
tmux&Byobu
開発効率アップのターミナル改造術

一般記事
・認証を支える技術
・Ejectコマンドで遊んでみませんか?

定価（本体1,220円+税）

2017年8月号

第1特集
私も機械学習エンジニアになりたい!
先端Web企業の取り組み方は?

第2特集
エンジニアのための うけるプレゼン・すべるプレゼン
——あなたの思いを伝える技術、教えます

一般記事
・「Mastodon」旋風からわかるSNSの未来

定価（本体1,220円+税）

2017年7月号

第1特集
もっとbashを使いこなしませんか?
理論&応用でシェル力の幅を広げる

第2特集
データの抽出・加工に強くなる!
MySQL[SELECT文]集中講座

一般記事
・ハッシュ関数を使いこなしていますか?（後編）
・Jamesのセキュリティレッスン【10】
・Windows Server 2016で構築する最新ファイアサーバ（後編）

定価（本体1,220円+税）

2017年6月号

第1特集
Vim、Emacs、Atom、Visual Studio Code
あなたのプログラミングを 加速させるエディタ

第2特集
多用途に使いこなせ、コードが読みやすく保守しやすい
今すぐはじめるPython

一般記事
・ハッシュ関数を使いこなしていますか?（前編）
・Windows Server 2016で構築する最新ファイアサーバ（前編）

定価（本体1,220円+税）

2017年5月号

第1特集
先輩が教える得ノウハウ
Linux入門[UNIXネットワーク編]

第2特集
サイボウズ流　サービス改善につなげる
ドッグフーディング環境の作り方

第3特集
いまから学ぶブロックチェーンのしくみ

定価（本体1,220円+税）

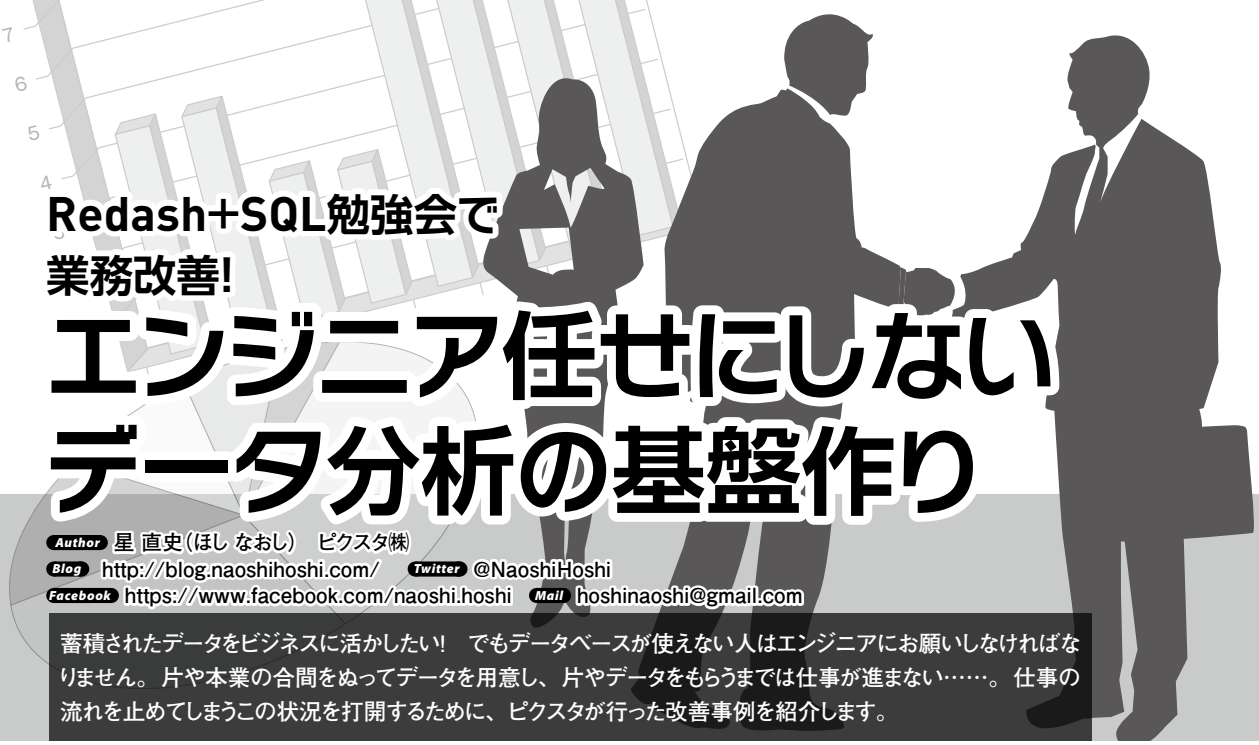
Software Design　バックナンバー常備取り扱い店							
北海道	札幌市中央区	MARUZEN＆ジュンク堂書店　札幌店	011-223-1911	神奈川県	川崎市高津区	文教堂書店　溝の口本店	044-812-0063
	札幌市中央区	紀伊國屋書店　札幌本店	011-231-2131	静岡県	静岡市葵区	戸田書店　静岡本店	054-205-6111
東京都	豊島区	ジュンク堂書店　池袋本店	03-5956-6111	愛知県	名古屋市中区	三洋堂書店　上前津店	052-251-8334
	新宿区	紀伊國屋書店　新宿本店	03-3354-0131	大阪府	大阪市北区	ジュンク堂書店　大阪本店	06-4799-1090
	千代田区	書泉ブックタワー	03-5296-0051	兵庫県	神戸市中央区	ジュンク堂書店　三宮店	078-392-1001
	千代田区	丸善　丸の内本店	03-5288-8881	広島県	広島市南区	ジュンク堂書店　広島駅前店	082-568-3000
	中央区	八重洲ブックセンター本店	03-3281-1811		広島市中区	丸善　広島店	082-504-6210
	渋谷区	MARUZEN＆ジュンク堂書店　渋谷店	03-5456-2111	福岡県	福岡市中央区	ジュンク堂書店　福岡店	092-738-3322

※店舗によってバックナンバー取り扱い期間などが異なります。在庫などは各書店にご確認ください。

デジタル版のお知らせ　DIGITAL

デジタル版Software Design 好評発売中！紙版で在庫切れのバックナンバーも購入できます

本誌デジタル版は「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）、「雑誌オンライン.com」（http://www.zasshi-online.com/）、「Gihyo Digital Publishing」（https://gihyo.jp/dp）で購入できます。最新号、バックナンバーとも1冊のみの購入はもちろん、定期購読も対応。1年間定期購読なら5%強の割引になります。デジタル版はPCのほかにiPad／iPhoneにも対応し、購入するとどちらでも追加料金を払うことなく、読むことができます。



Redash+SQL勉強会で業務改善!

エンジニア任せにしないデータ分析の基盤作り

Author 星 直史(ほし なおし) ピクスタ㈱
Blog http://blog.naoshihoshi.com/ Twitter @NaoshiHoshi
Facebook https://www.facebook.com/naoshi.hoshi Mail hoshinaoshi@gmail.com

蓄積されたデータをビジネスに活かしたい! でもデータベースが使えない人はエンジニアにお願いしなければなりません。片や本業の合間をぬってデータを用意し、片やデータをもらうまでは仕事が進まない……。仕事の流れを止めてしまうこの状況を打開するために、ピクスタが行った改善事例を紹介します。

業務改善に至る背景

ビッグデータ、データ分析・解析、BI(Business Intelligence)ツールなど、蓄積した大規模データを使用し、ビジネスに活かすシーンが増えてきました。この潮流は、データの収集や可視化を効率的に行うことで、「原因・要因の発見や、情報・状況の迅速な把握を可能にし、より精度の高い具体的なアクションの意思決定をする」というニーズの高まりによるものなのではないかと思います。

以前はこのようなことを実現するためには、環境を準備するだけでも多大な工数が必要とされていました。しかし昨今は、安価で容易に実現できるさまざまなサービスが台頭してきています。また、それらを使用することによって専任のエンジニアではなくても扱えるようになるため、エンジニアの工数低減や、作業効率、情報の把握、アクションの精度が向上しやすい環境が整いつつあります。

弊社でもPIXTA[注1]を運営する中で、次のような取り組みが頻繁に行われるようになりました。

- 蓄積したデータの分析・解析を行い、施策を行ううえでの根拠とする
- 施策実施中、実施後の計測や数値のビジュアライズをする
- 計測などをした結果を元に、さらに良い結果が導けるか検討をする

弊社では、全社員の内わけとしてエンジニア比率が20%前後とけっして多くはなく、プロダクト改善の合間に上記のようなデータ抽出依頼をこなしています。また、直接施策に影響があるようなものがすべてではなく、ビジネス側のメンバーが作成する週報や月報のデータ抽出依頼もこなしています。これはデータの所在がMySQLやTreasure Dataなど、おもにエンジニアが操作しなければ取得することが難しいような構成になっていたためです。

このような体制だったことから、エンジニアではないメンバーが増えるにしたがってデータ抽出依頼の量も増え、次第にプロダクト改善のスピード感が損なわれてしまう問題が出てきてしまいました。具体的には次のような問題です。

エンジニアがデータ抽出依頼対応に割く工数が増える

先述のとおり、エンジニアではないメンバーが増えていくにつれてデータ抽出依頼が多くな

注1) https://pixta.jp

り、本来優先度を高くして取り組みたいプロダクト改善が思ったようなスピード感で進められなくなってきました。

データ抽出依頼が増えてしまう要因の1つでもありますが、以前抽出したデータの“日付の範囲を変更しただけ”のものも定期的に依頼されます。また、使用したクエリがエンジニア間で共有されておらず、別のエンジニアが対応した場合にはクエリをはじめから作成してしまうといった無駄な工数が発生していました。

データ抽出完了待ちが発生する

データ抽出がエンジニアにしかできない作業であったため、依頼者が抽出されたデータをもとに別の行動を起こす場合、対応が完了するまで待ち続けなければなりませんでした。プロダクト改善においてはPDCAを回すことが重要な活動の1つですが、これを高速に回すことができず、エンジニア以外のメンバーのスピード感も失われつつありました。

機械的にできることを手作業で行ってしまう

エンジニアはすべてのデータ抽出を最優先で行うわけではなく、ほかのタスクを優先して対応しなければならない場合もあります。しかし、ほかのタスクを優先することが続いてしまうとデータ抽出依頼者は待ちきれず、クエリを実行すれば一瞬で終わるような処理を、膨大なデータを目視で確認し、手作業で計算、計測を行うようになってしまいました。

また、データ抽出依頼者も「エンジニアが忙しそうだから依頼しにくい」と思ってしまい、手作業でやってしまおうという空気感ができてしまいました。エンジニアと非エンジニアの間に余計な気遣いが発生してしまい、結果的に本来行うべき作業に集中できないといった問題が発生してしまいました。

◆ ◆ ◆

このような問題を抱えていると、メンバー各個人の生産性が向上せず、ユーザに対する価値提供のスピードや質が低下してしまいます。限られた人員でプロダクトを作っている弊社にとって、これは非常に重大な問題だととらえました。

そこで、まずはこれらの問題が改善している状態を思い描き、その状態から逆算して、対応を進めることにしました。改善している状態は次のとおりに定義しました。

- MySQLからのデータ抽出をエンジニアではない職種の方でも抽出できている状態
- 抽出したデータを容易にビジュアライズできている状態
- 結果的にエンジニアのデータ抽出依頼が減った状態

本記事では、上記の問題を改善するために、データ可視化ツールであるRedash[注2]を導入し、非エンジニア自らが、データの閲覧とクエリの修正を行えるようになるまでに、弊社で取り組んだことについて紹介します。

まず理想の状態を実現するまでの障壁を分解しました。

実データをエンジニアではない職種の方でも抽出できる

エンジニアの場合は、MySQLサーバに対して接続が許可されたサーバに接続し、CLIでSQLを実行し、欲しいデータを取得することができますが、非エンジニアの場合は、そもそもサーバに接続すること自体、難易度が高く、データ取得の障壁になってしまうのではないかと考えました。そのため、使用者(非エンジニア)に“サーバに接続”ということを意識させないようなしくみが必要だと考えました。

抽出したデータを容易にビジュアライズできる

これまでのデータ抽出依頼のフローは、

注2) https://redash.io/

1. エンジニアにデータ抽出を依頼
2. エンジニアがデータを抽出。依頼者は抽出されるまで待つ
3. 依頼者がエンジニアから欲しいデータをCSVなどで受け取る

という手順でした。抽出結果をCSVなどで受け取ったあとは、多くの場合はGoogleスプレッドシートやExcelにデータを貼り付け、グラフ化をしてビジュアライズしていました。さらには、そのグラフのスクリーンショットを取り、社内で使用する資料(週報)などに貼り付けるという作業をしていました。

そのため、データ取得、可視化、共有を行うまでに、異なるツールを使用して何ステップも踏まなければならなかったので、これを一元化できるものが必要だと考えました。

何ができれば必要十分か

上記の障壁を勘案すると、次の項目が満たされると理想の状態を作るための環境が整うのではないかと考えました。

・データ取得の容易性
・可視化(ビジュアライズ)の容易性
・共有することの容易性

データ可視化ツールの検討

冒頭でも述べたとおり、昨今では多様なデータをビジネスに活かすニーズが高まってきており、商用BIツールや、オープンソースソフトウェア(OSS)のデータ可視化ツールが台頭してきました。まず、弊社では導入を検討する際に**表1**の観点で商用かOSSかを比較しました。

導入と撤退の容易性

商用の場合は、サービスの契約先の営業さんに話を聞いたり、ライセンスの契約を結んだり、入金するための社内稟議を通さなければならなかったりするなど、気軽に導入ができないのではないかと思います。

OSSの場合、自分たちで環境構築しなければならない手間こそはありますが、Amazon Web Services(AWS) AMI、Google Compute EngineのCustom Image、Docker Composeなどを使用すれば瞬時に動作する環境が手に入ります。

また、撤退の容易性についても検討しました。商用の場合はライセンスや契約期間といった縛りがあるため、仮に導入が失敗した場合も簡単に契約を打ち切ることができません。OSSの場合は導入が容易であることと同時に、社内で構築した環境を捨てるだけで瞬時に撤退できます。

このようなことから、導入と撤退の容易性という点においては、OSSのほうが有利であると判断しました。

コスト(金銭面)

商用の場合ライセンス料金を支払う必要がありますが、OSSの場合ライセンス自体に料金はかかりません。仮に導入に失敗した場合でも、OSSならば金銭的な痛手はほぼかかりません。かかるとしたら導入時に使用したインスタンス料金くらいでしょうか。ただ、これについても最小のインスタンスや構成で環境を構築すればローコストで運用できるため、懸念として挙げるほどではないと言えます。

サポート体制

サポート体制について比較するのはやや難しいですが、商用の場合はメーカーからサポートが受けられる一方、問い合わせから回答まで時間がかかることがあります。また、機能のカスタマイズや使用するメンバーが増えた場合に再

▼表1　データ可視化ツールの選択：商用とOSSをピクスタの視点で比較

	導入と撤退の容易性	コスト(金銭面)	サポート体制
商用	×	×	△
OSS	○	○	×(?)

度契約を結ぶ必要があるなど、柔軟な対応ができないことが考えられます。

一方OSSでは、基本的にはすべて自分たちで解決しなければならないのでサポートという一般的な観点では×でしょう。しかし、OSSのコミュニティ活動が活発であったり、問題が発生した場合でも公開されているソースコードを読めば原因と対策を打てる場合があります。**表1**の欄に「？」をつけたのはこのためです。

大きく分けて商用かOSSかで比較した結果、運用まで持っていけるような確実性がないことと、すばやくツールを導入して状況が改善するかの検証が重要であると判断したことから、導入と撤退の容易性とコストの観点でOSSのデータ可視化ツールを採用することにしました。

OSSのデータ可視化ツールで検討したのはSuperset[注3]とRedashです。比較検討した項目は**表2**のとおりです。

環境構築の敷居と運用の容易性

ここでいう環境構築の敷居と運用の容易性とは、技術的な導入の敷居と、導入者だけではなくそれ以外のメンバーも容易に運用できるかです。これらについて検討しました。Supersetは環境構築の手段としてDocker Composeしか対応していませんでした[注4]。一方Redashは、AWS AMI、Google Compute EngineのCustom Image、Docker Composeと構築の方法が充実していました。

弊社のエンジニア全体のスキルを俯瞰し、AWSとDockerについて、それぞれ扱えるメンバー数とその知識の深さを比較した結果、AWSのほうがはるかに優勢でした。また、今後の展開を考えた場合、非エンジニアを含めて全社的に導入することが最終目標であったため、非エンジニアからの質問が多くなることが予想されました。その場合に、メンテナンスできるメンバーが少ないことは持続可能性のリスクであると考えたため、環境構築においてはAWS AMIが使用できるRedashに軍配が上がりました。

データソースの充実

SupersetとRedashはMySQLやPostgreSQLなどといったデータソースからデータを取得し、可視化をします。SupersetとRedashが扱えるデータベース(DB)・データウェアハウス(DWH)に大きな差はありません[注5]。

しかし、SupersetとRedashの差はDB・DWH以外のデータソースが扱えるかどうかにあります。Redashでは、Google Analytics、Googleスプレッドシートなど、基本的なDB・DWH以外のサービスも扱えることが大きな特徴です。弊社では、Googleスプレッドシートを利用してデータの加工、分析、ビジュアライズなどを行っている非エンジニアのメンバーが多く、データ可視化ツールでGoogleスプレッドシートが扱えることで、業務効率化が図れる可能性がありました。

◆ ◆ ◆

ほかにも比較検討したことや機能差分はありますが、弊社での大きな判断ポイントは上記の2点でした。また、両判断ポイントにおいてRedashはSupersetと比較して優位であったため、弊社ではデータ可視化ツールとしてRedashを導入することにしました。

注3) https://github.com/apache/incubator-superset
注4) 2017年1月当時。

▼表2　データ可視化ツールの選択：SupersetとRedashをピクスタの視点で比較

	環境構築の敷居と運用の容易性	データソースの充実
Superset	×	△
Redash	○	○

注5) MySQL、PostgreSQL、Redshift、Treasure Dataなど、比較的シェアが大きいものについては両ツールともに使用することが可能です。

データ可視化ツールの導入と活用方法

データ可視化ツールがRedashに決定しましたので、実際に環境構築の方法と基本的な使い方について説明します。解説にあたり前提は次のとおりです。

- AWS AMIを使用[注6]
- Redashのバージョンは上記AMIを使用するため2.0.0+b2990を使用
- データ取得のターゲットはMySQL

先述のとおりRedashはAWS AMIから立ち上げることが可能です。各リージョンのAMIはRedash Help Centerから取得できます[注7]。基本的には通常のAMIの立ち上げと同様ですが、RedashはWebからアクセスすることと、各種設定、トラブルシューティング時にインスタンスにSSH接続するため、セキュリティグループで80番、443番、22番を許容するように設定してください(**図1**)。

また、Webからアクセスするため、Public IPを取得する必要があります。EC2インスタンスの起動設定「Step 3: Configure Instance Details」のAuto-assign Public IPをEnableに設定します(**図2**)。

注6) ap-notheast-1 AMI ID: ami-fde8199bを使用

注7) Setting up a Redash instance URL https://redash.io/help-onpremise/setup/setting-up-redash-instance.html

COLUMN

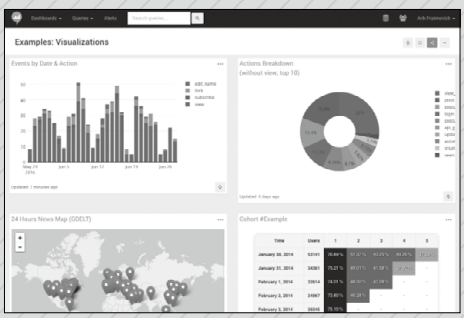
Supersetの特徴

目的や組織によってはRedash(**図A**)よりSuperset(**図B**)のほうが良い可能性もあります。弊社では大きな判断ポイントとして「環境構築の敷居と運用の容易性」と「データソースの充実」という2点を挙げましたが、弊社の事情をふまえず、単純にツールとして比較した場合、次のような違いがあります。

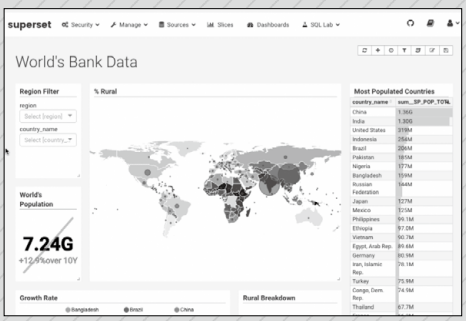
- UIによる操作
- 柔軟なダッシュボードの見せ方

おもにUIによる操作と、描画したグラフの種類とその見せ方に大きな特徴があります。RedashはSQLを書くことが前提であることに対して、Supersetはグラフを描画する際、UI(プルダウンリストなど)でどこの情報の何をどうやって描画するかを選択し、データを表示させることができます。また、ダッシュボードにおいては、Redashは1行が2カラムに分かれていて、グラフの表示領域を1カラムまたは2カラムしか指定できないのに対して、SupersetはSlice(描画するグラフの単位のようなもの)を定義してそれを自由にダッシュボードに配置することができます。整理されたダッシュボードにしたい場合や、UIからのデータ操作はSupersetのほうがやや扱いやすいと思います。とはいえ、UI操作であっても、SQLでいうWHERE句、ORDER BY句、GROUP BY句などの概念は出てくるので、ある程度SQLを知っていないと扱いが難しいという印象です。

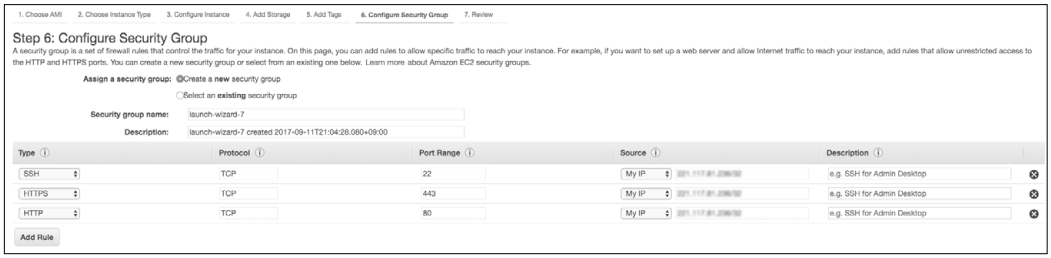
▼図A　RedashのDemo画面



▼図B　SupersetのDemo画面



▼図1　セキュリティグループの設定

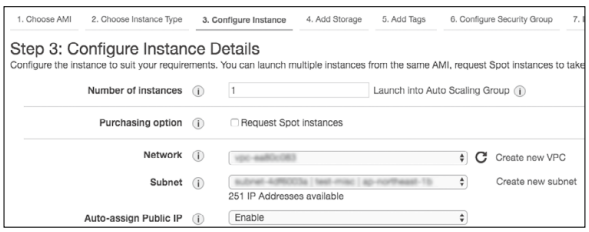


EC2の起動設定は以上です。

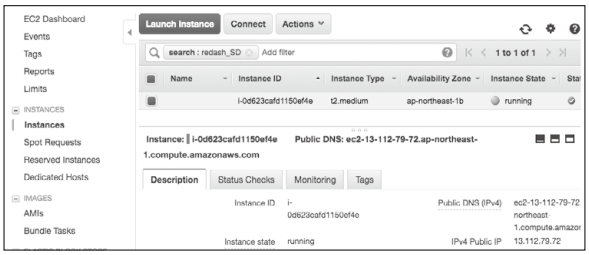
最後にPublic IPを確認するために、AWSマネジメントコンソールから[EC2]→[Instances]を開き、起動したインスタンスのステータスがrunningになったら、[Description]タブを表示してPublic IPアドレスを確認し、ブラウザからアクセスします(**図3**)。

Public IPアドレスにアクセスすると、はじめに管理者権限の情報設定画面が表示されるので、管理者の情報を入力します。入力が完了すると、いよいよRedashを使用できるようになります。

▼図2　Public IPの取得設定



▼図3　EC2インスタンスのIP確認



COLUMN

AWS EC2インスタンスのセキュリティグループの設定

「データ可視化ツールの導入と活用方法」でセキュリティグループの設定について触れました。セキュリティグループの設定において80番、443番、22番というのはそれぞれ**表A**のような意味を持ちます。

RedashはWeb UIですので、80番(HTTP)を許可していなければアクセスすらできないため指定する必要があります。また、Redashで機密情報を扱う場合は443番(HTTPS)を指定し、通信を暗号化しなければ、セキュリティ上不安が残ってしまいます[注A]。最後に22番(SSH)ですが、Redashは環境変数などで設定を変更したり、Pythonでコードを書いてデータソースにしたりすることもできます。つまり、Redashをホストしているインスタンスにリモート接続し、操作する必要が出てくるため、22番(SSH)を開けておく必要があります。

注A) あわせてアクセスを許可するIPアドレスも絞ると、より堅牢性が向上します。

▼表A　AWS EC2インスタンスに設定したポートの意味

ポート番号	プロトコル	意味
80	HTTP	http://で始まるURLでブラウザからのアクセスを許可する
443	HTTPS	https://で始まるURLでブラウザからのアクセスを許可する。HTTPとの違いはSSL/TLSを使用したサーバの認証・通信内容の暗号化・改ざん検出などを行う点
22	SSH	暗号や認証の技術を利用して、安全にリモートサーバと通信するためのプロトコル

データソースの追加(MySQL)

Redashは外部のデータソースからデータを取得し、可視化を行います。そのため最初に行うべきことはデータソースの登録です。ここではRDBMSの1つであるMySQLを例にして登録の手順を紹介します。

手順1： Redashに管理者でログイン後、画面右上のDBアイコンをクリックする(**図4**)
手順2： [New Data Source]をクリック
手順3： TypeにはMysqlを選択し、各種認証情報を入力する
手順4： 作成したデータソースに正常に接続できるか確認

認証情報を入力後、[Test Connection]ボタンを押下しRedashからMySQLに正常に接続できていれば設定完了です。

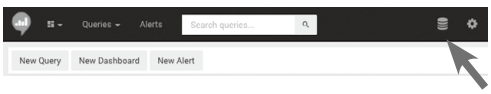
データソースの追加(Googleスプレッドシート)

弊社では非エンジニアのメンバーが、Googleスプレッドシートを使ってデータの蓄積、分析、可視化、共有を行っていることが多かったため、SupersetとRedashの違いで挙げた、Googleスプレッドシートがデータソースに適用できることがRedash選択の判断ポイントとなりました。

Googleスプレッドシートの値をデータソースに追加する場合、手順がMySQLとは異なるため、その手順について説明します。

手順1： Google API Console[注8]にアクセス
手順2： 新規プロジェクトを作成しRedash用のプロジェクトを作成
手順3： API Keyを作成
3-1：画面左のサイドナビより[APIとサービス]→[認証情報]を押下
3-2：[認証情報を作成]→[サービスアカウントキー]を選択
3-3：サービスアカウントの種別に[新しいサービスアカウント]を選択し、各項目を入力。キーのタイプはJSONを選択し作成ボタンを押下[注9](**図5-1**)
3-4：画面左のサイドナビより[ダッシュボード]をクリック→[APIとサービスの有効化]をクリック
3-5：検索フォームに"Google Drive API"と入力すると出てくる「Google Drive API」を選択
3-6：[有効にする]をクリック
手順4： Googleスプレッドシートで API連携
4-1：API Keyを作成する過程(3-3)でダウンロードしたJSONを開くと"client_email"というキーが存在するので、それをコピー
4-2：Googleスプレッドシートの共有設定で、4-1でコピーしたE-mailと共有する
手順5： Redashでデータソースを追加
5-1：Redashのデータソース追加画面に遷移
5-2：Typeは「GoogleSpreadsheet」を選択

▼図4　データソースを追加



COLUMN

AWS RDSインスタンスのセキュリティグループの設定

AWS EC2でRedashインスタンスを立ち上げる際に設定したセキュリティグループですが、AWS RDSのMySQLを使用する場合は、RDSインスタンスに対してもセキュリティグループを設定する必要があります。

データソースとしてMySQLを設定する場合、MySQLはデフォルトのポートが3306番です。また、許可するIPアドレスもEC2インスタンスのIPを指定することで"EC2からRDSに対して3306番でアクセスを許可する"という設定をし、データの取得を可能にします。

注8) https:// console.cloud.google.com
注9) JSONファイルがダウンロードされます。認証情報が書かれている重要なファイルであるため、紛失・削除しないように注意してください。

し、JSON Key File に3-3で取得したJSONファイル選択して保存(**図5-2**)

Google スプレッドシートとの連携は以上です。

ただし、クエリ作成はさらに少し手順を踏む必要があります。Redash に取り込みたいGoogle スプレッドシートのURLが"https://docs.google.com/spreadsheets/d/hogehoge/edit#gid=12345"だと仮定します(**図5-3**)。このスプレッドシートが2つのシートで構成されていた場合、左から2番めのシートを取り込む場合のクエリは"hogehoge|1"となります(**図5-4**)。注意することは2つあります。

注意点1:スプレッドシートのURLとシートの番号は"|"で区切る

注意点2:シートの番号は左から0、1、2、……となる[注10]

取得したデータの可視化

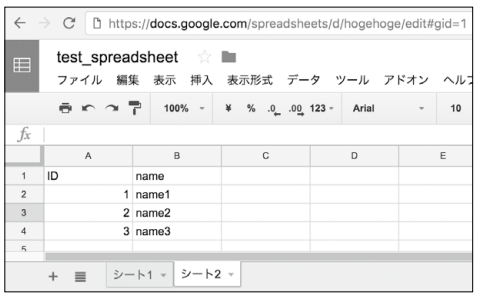
取得したデータはグラフで可視化することが可能です。可視化までの手順は次のとおりです。

手順1:クエリを実行するとTABLEタブが表示されるので、その横にある[+ NEW VISUALIZATION]ボタンを押下

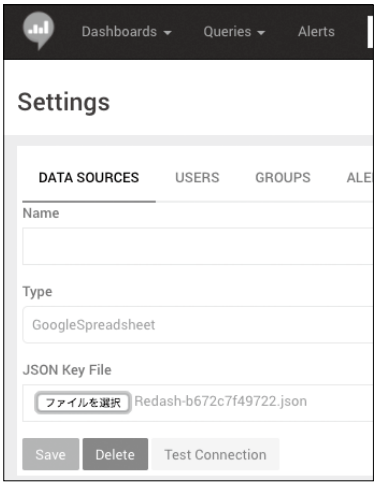
注10) したがって、シートの順番を入れ替えてしまうと取得するデータも異なってしまいます。

▼図5-1 API Keyを作成

▼図5-3 データソースとするGoogle スプレッドシート例



▼図5-2 Redashへのデータソース追加



▼図5-4 図5-3のデータを取り込むRedashのクエリ

手順2： 各項目を埋める

これだけの操作で取得したデータを可視化することができます。また、作成したグラフはダッシュボードに追加できます。

クエリの再利用と共有

データ可視化ツールに必要な要件として、共有することの容易性を挙げました。共有については、Redashでは2つの機能があります。

1点目はダッシュボード機能で、可視化したグラフやデータをほかのメンバーに共有する際、1つずつURLを送るのではなく、あるまとまりの単位で共有することを可能にする機能です。

2点目はエンジニアには馴染みがあるFork機能です。これは、あるクエリをもとにしたクエリを作成する場合に使用します。

ダッシュボードの作成と設定

ダッシュボードの作成と設定の手順は次のとおりです。

手順1： グローバルヘッダから[Dashboards]のプルダウンにある[New Dashboards]を押下し、ダッシュボード名を入力
手順2： ダッシュボードページに遷移後、画面右上の設定アイコンから[Add Widget]を押下(**図6-1**)
手順3： 追加したいクエリ名を検索し(**図6-2**)グラフを選択

この操作を繰り返し行い、あるまとまり単位のダッシュボードを共有すると一覧性が向上し、同じ共有のしかたでもわかりやすい画面を共有できます。

クエリのFork

クエリ編集画面を開くとSaveボタンの横にForkボタンがあります。Fork機能はこのボタンを押下するだけで、もとのクエリをForkすることができます。

クエリの定期実行

Redashではデータソースからクエリでデータを抽出、表示するだけではなく、登録したクエリを定期的に実行することができます。毎日決まった時刻に実行することや、決まった時間や日付の間隔で実行することが可能です。

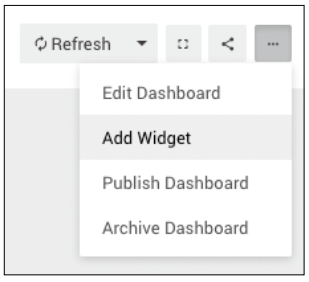
設定はとても簡単です。

手順1： クエリ画面にアクセス
手順2： [Refresh Schedule]のリンクを押下(**図7-1**)
手順3： 実行スケジュールを設定(**図7-2**)

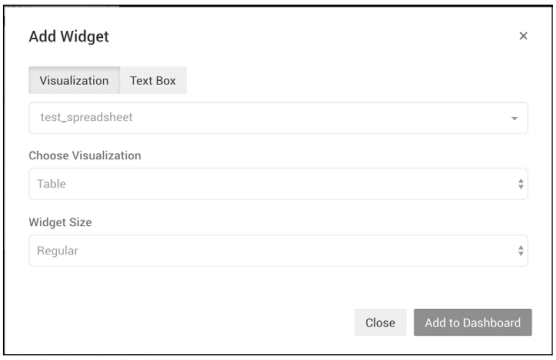
以上の設定で、決まった時間、決まった間隔で実行させることができます。

この定期実行機能は、週報など、定期的に実行する必要があるデータの取得に役立つことはもちろんですが、データ取得量が多く、レスポンスに時間がかかってしまうようなクエリの時

▼図6-1　ダッシュボード作成



▼図6-2　ダッシュボード作成



間短縮にも効果的に使えます。Redashのしくみとして、同一のデータであれば(都度データソースにアクセスして取得するのではなく)Redash内にキャッシュのような形で残してレンダリングを高速化していることを利用します。たとえば始業前に定期実行を設定することで、始業後、実際にデータを閲覧する際にスムーズに閲覧することが可能になります。

クエリ文字列から動的にSQLを組み立てる

Redashは実行するクエリをURLのクエリ文字列によって動的に実行することができます。設定は、実行するクエリ内に次の記述を追加するだけです。

```
{{ user_id }}    user_idは任意の識別子
```

たとえば、次のクエリでURLが生成された場合、

```
# http://ip_adress/queries/1/source?p_user_id=1
SELECT * FROM users where id = {{ user_id }}
```

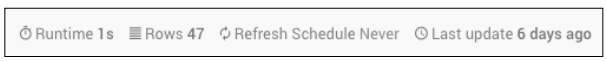
URLにクエリ文字列として"p_user_id"を加えると、実行したいクエリの{{ user_id }}の部分にクエリ文字列で指定したuser_idを適用するといった具合です。

また、RedashはWeb UIですので、SELECT文の中にHTML要素(たとえばリンクなど)を含めれば、一覧ページと詳細ページの関係を作ることもできます[注11]。

```
SELECT CONCAT('<a href="http://ip_adress
/queries/1?p_id=', users.id, '">', 
users.id, "</a>") as "詳細ページのリンク",
       last_name,
       first_name
FROM users
```

注11) 便利ではありますが、乱用してしまうとコードがたいへん読みにくくなってしまい、クエリ作成者以外の人が読めなくなってしまう可能性もあるので、慎重に使いたいところです。

▼図7-1　定期実行

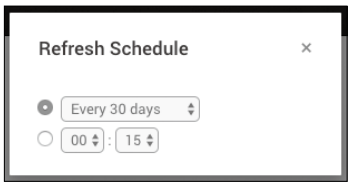


ここまでが基本的なRedashの使い方の紹介です。データ可視化ツール導入にあたっての必要条件を再確認すると、Redashは掲げていた条件を次のように満たすことができました。

- **データ取得の容易性**……導入初期にエンジニアが一度データソースの設定さえしてしまえば、非エンジニアはデータベースへの接続といったことを考える必要がなくなる。またRedashはWeb UIであるため、ブラウザから誰でもアクセスすることが可能に
- **可視化(ビジュアライズ)の容易性**……取得したデータに対して適用したいグラフを選択するだけで容易にデータを可視化することができる。また、作成したデータをピンチイン／ピンチアウトで拡大縮小し、詳細データを閲覧することが可能に
- **共有することの容易性**……作成したクエリをダッシュボードにすることにより、URLを伝えるだけで閲覧したいデータやグラフを容易に共有できる。また、一度作成したクエリをForkすることによってクエリの再利用ができ、クエリを初めから作る手間を減らすことが可能に

Redashによるデータ可視化環境構築ができれば、残りの問題点はエンジニアへのデータ抽出依頼の削減のみです。この問題を改善すればエンジニアだけではなく、会社全体の生産性の向上につながり、ひいてはプロダクト改善プロセスを高速化させ、ユーザへの価値提供が促進されることになります。

▼図7-2　定期実行



COLUMN

Redashの認証機能

Redashの認証はデフォルトだとE-mailとパスワードでの認証となります。この場合、管理者が事前に招待メールを送信しなければなりません。使用するユーザの規模が大きくなった場合、Redashの認証設定だけで苦労してしまいます。これを簡易にするために、RedashではGoogleアカウントでの認証機能も利用できます。手順は次のとおりです。

手順1：Google API Console[注B]にアクセス

1-1 画面左のサイドナビより[認証情報]→[OAuth同意画面]タブを選択し、項目を入力後に保存ボタンを押下 1-2 画面左のサイドナビより[認証情報]→[認証情報]タブを選択し、[認証情報を作成]プルダウンを押下後、[OAuthクライアントID]を選択 1-3 [承認済みのJavaScript生成元]にURLを入力。[承認済みのリダイレクトURI]にURL＋/oauth/google_callbackを入力し、作成ボタンを押下（図C） 1-4 ポップアップで表示されるクライアントIDとクライアントシークレットを控える

▼図C　Redashの認証にGoogleアカウントを利用する

手順2：RedashのEC2インスタンスにSSH接続し、次のコマンドを実行

```
$ cd /opt/redash/current
$ sudo -u redash bin/run ./manage.py org set_google_apps_domains {{許可するドメイン}}
```

手順3：/opt/redash/.envに次を追記

```
export REDASH_GOOGLE_CLIENT_ID="手順[1-4]でメモしたクライアントID"
export REDASH_GOOGLE_CLIENT_SECRET="手順[1-4]でメモしたクライアントシークレット"
export REDASH_PASSWORD_LOGIN_ENABLED=false # パスワードログインを禁止
```

手順4：Redashの再起動

```
$ sudo supervisorctl restart redash_server
```

注B）https://console.cloud.google.com/

非エンジニア向けのSQL勉強会の実施

Redashによりデータ可視化環境ができたので、残る課題はエンジニアのデータ抽出工数の削減となります。Redashの Fork機能と、クエリの共有を行っていけばある程度は削減できると思ったのですが、「そもそもエンジニアにデータ抽出依頼が来ない」ことについても考えてみました。つまり、非エンジニアがSQLを扱い、自らデータ抽出を行うことができている状態です。

弊社ではWebディレクターなど、ごく一部、SQLを書けるメンバーはいましたが、データ抽出依頼をするメンバーの大半はSQLやデータベースの存在は知っているものの、何に使用するかわからない状態でした。また、次のような思い込みをしているメンバーも多数いました。

・SQLはエンジニアにしか扱えない特殊な技能
・自分がSQLでデータを触ったら壊れてしまう（ユーザに影響を与えてしまう）

これらの苦手意識を持った非エンジニアがSQLを扱える状態にすることが理想的な状態であり、これを実現するために、実際に触ってもらい、苦手意識を克服し、誰でも手軽にデータ抽出できるという認識になってもらうための勉強会を実施することにしました。

SQL勉強会実施に向けての戦略

SQLに苦手意識を持つメンバーが多い現状であったため、SQL勉強会の実施や、Redashを運用レベルに持っていくまでの戦略を考える必要がありました。そこで、SQLをまったく知らない人に、なるべくたくさん勉強会に参加してもらえるように、ハードルを下げるためのしくみも用意しました。

❖ 参加者は事前準備なしで参加してもらう

これまで、SQLが扱える一部の非エンジニアは、アプリ型のSQLクライアントをインストールし、MySQLの認証情報を手で入力していました。しかし、これは大半の非エンジニアにとってはハードルが高いものでした。今回は先述のとおり、Redashのデータソースに弊社で使用しているDBをあらかじめ登録しておくことで、参加者は“サーバへ接続”といったことを考える必要がなくなり、事前準備が不要になりました。また、Redash自体への認証をGoogleアカウントを使用しての認証方式に切り替え、認証の手間も減らすことで、Redashにアクセスすればすぐにを打ち込むことができる環境を整えました。

❖ 運営側への協力を仰ぐ

SQL勉強会実施の告知をしたところ、非エンジニアから40人強の参加が決まりました。この人数になると、講師1人では質問のすべてを受け答えできず、結局参加者の理解が深まらないままに終了してしまい、次回参加の熱量が失われてしまうことが懸念されました。そこで、運営を手伝ってくれるサポーターを社内エンジニアから募ることにしました。さらに念のため、講師のプレゼン内容を事前にサポーターに周知させ、スムーズに質問に答えられるように配慮しました。勉強会1回あたり、講師となるメインプレゼンター1名、参加者側の席で質問などに瞬時に受け答えできるサポーターを最低3名募り、勉強会を実施しました。

また、講師もなるべく1人ではなく複数名に協力してもらうことで、講師自体の負荷を分散でき、発表内容の精査も行え、リスケのリスクも防ぎつつ、継続的に安定して勉強会を運営できました。

❖ SQL勉強会の内容

人のアサインができたら、次は肝心の内容決めです。運営側のエンジニア（講師とサポーター）が最初に集まり、非エンジニアがどれくらいのレベルでSQLを理解してくれればデータ抽出の負荷が減っていくかの現実的なラインを決定し、

そこに向けて勉強会の実施内容を決めていきました。

結論としては、勉強会実施後の状態の期待値を次のとおりに設定しました。

最低限の目標

・やりたいことの要件と、おおよそどのテーブルからデータを取得するかあたりがついていること
・クエリを完全に組み立てていなくても、わからないなりにクエリができあがっていること

ストレッチした目標

・抽出条件に使うカラムを理解していること
・JOIN句を使い、テーブル結合ができること

SQL勉強会実施後、最低限の目標を達成していた場合には、次のような業務上のやりとりができることを想定しました。

> 非エンジニア「素材数をn点以上持っているユーザを出したいのですが(要件の理解)、取ってくるテーブルは○○テーブルで合っていますか？(テーブルの特定)」
> エンジニア「はい、合っています！　試しにクエリをちょっと作ってみてもらえますか？」

また、ストレッチした目標は、ここまでできていたらエンジニアの負荷が下がるだろうという目標にしました。業務上想定できるやりとりは次のとおりです。

> 非エンジニア「退会していないユーザが持つ素材データのうち(JOIN句の使用)、販売中のデータを月別に出したいのですが(抽出条件の理解)、このクエリで合っていますか？」
> エンジニア「はい、合っています！　販売中かどうかは素材テーブルの○○というステータスが△△の場合が販売中になるので、WHERE句で指定してください。」

最低限の目標の場合、非エンジニアがSQLを知ってはいて、ある程度クエリを組み立てることはできるが、エンジニアのサポートは要するといった状態です。エンジニアの負荷が下がることはそこまで期待できませんが、ストレッチした目標を達成した場合には、エンジニアは、そこそこできあがったクエリを確認、あるいは少し手直しの指示を出すだけで終わるので、目に見えた工数の低下につながるだろうと考えました。

ストレッチした内容はハードルが高いかと思いましたが、有志で集まったエンジニアのフォロー体制があれば達成できるだろうと考えました。そこでSQL勉強会の内容は、思い切ってストレッチした内容まで実施することにしました。

勉強会の実施

勉強会は全4回にしました。

第1回　イントロダクションとRedashの画面にたどり着く
第2回　基本構文(SELECT、FROM、WHERE)を実行してみる
第3回　集計関数(COUNT、SUM、AVG、MAX、MIN)を実行してみる
第4回　JOIN句を使ってテーブルを結合する

終わってから振り返ってみると、第1回～第4回の中で、もっとも重要だと感じたのは第1回めだと思います。第1回めはエンジニアが参加者に対して積極的にフォローすることや、SQLはそれほど難しくないこと、最初から完璧を目指さないこと、間違ったクエリを流してもユーザに何の影響もないことなどを説明し、参加者の心理的なハードルを下げることに徹しました(**図8**)。

第2回～第4回は基本的にハンズオン形式で

▼図8　SQL勉強会 発表スライドより抜粋

心意気と習得のコツ(お約束)

- いきなり完成を目指さない
- とりあえず、データ眺めてみる
- データを見て考えながら少しづつ完成に近づける
- エラーメッセージを見てみる
- 15分考えてわからなかったら聞く

エンジニアもデータ出しは毎回この手順でやってます。

行い、習うより慣れろの精神で学んでもらうことにしました。また、参加者は40人強にもなるので、同じ内容でも習得スピードに差が出ることも考慮し、ハンズオン時の進め方は基本編と応用編に分けました。具体的な内容と時間配分は次のとおりです。

・何を学ぶか発表、重要概念の説明(5分)
・お題1の発表(15分)
——基本問題の出題
——応用問題の出題
——基本問題の正解例の発表
——応用問題の正解例の発表
・お題2の発表(15分)
——基本問題の出題
——以下お題1の進め方と同様
・お題3の発表(15分)
——基本問題の出題
——以下お題1の進め方と同様
・今回の学びのふりかえり(5分)
・次回予告(5分)

応用編を実施するメンバーは、毎回おおよそ1〜2割はいました。応用編を用意することによって、手持ち無沙汰になることを防ぐだけではなく、基本編ができたメンバーが応用編の解説を聞くことで、問題に対してさまざまな解があることを発見し、より理解を深めてもらうような場面が多々ありました(**写真1**)。

▼写真1　SQL勉強会の様子

勉強会実施後の振り返り

SQL勉強会実施後の振り返りとして、良かったこと、悪かったことさまざまありましたが、その中でも重要なものをピックアップして紹介します。

復習の回を準備して勉強会の脱落を防ぐ

エンジニアを何名かサポート役として協力を仰ぎましたが、どうしても習得の早さに個人差があるため、理解が追いつかずに脱落してしまうメンバーも一定数いました。勉強会の実施中の様子を見て、あまり理解が進んでいないメンバーをチェックすることや、チェックしたメンバーに対して実施後に感触を聞き、復習の回を行うか判断することは重要です。

実際に弊社では、第2回と第3回の間に第2.5回と称して、復習の回を設けることにしました。復習回に参加したメンバーは数名だったので、その分メンバーの理解のスピードに合わせて丁寧に教えることができ、結果的に第4回までたどりつくことができたのだと思います。

テーブルの物理名と論理名のマッピング表を用意する

エンジニアであれば、DBにどんなテーブルが存在していて、そのテーブルにどのようなカラムがあるかを把握していることが多いと思います。そもそも**DESC テーブル名**でカラムの物理名を調べたり、物理名や既存のプログラムからある程度論理名を推測することなどが可能です。しかし、非エンジニアにとっては物理名までたどり着くことはできても、論理名まで把握するのは難しいでしょう。

そこで勉強会の前に、最低限、勉強会で使用するテーブルとカラムの物理名、論理名の一覧を作り、参加者に共有しました。そうすることで、参加者には、やりたいこと(抽出条件)から論理名を探し、物理名を特定したのちにSQLを組み立てる、というリズムが生まれました。また、この一覧表を作成することで、非エンジニ

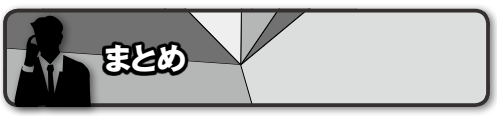
ア自らテーブルの構造と意味を探しにいき、エンジニアに聞かずともクエリを作成できるケースが多くなりました。勉強会の事前・事後ともに役立つ資料になったと振り返っています。

Redashの浸透

Redashは勉強会でも大いに活躍しましたが、真価を発揮するのは運用に入ってからだと思います。Redash導入前の問題点として述べた、毎回エンジニアに依頼がくる状態、エンジニアの○○さんしか知らない秘伝のクエリの存在などを解消するには、組織全体にRedashが浸透している状態を作らなければならないと考えました。

何かを普及させる際に筆者はよく「普及曲線」を考えます[注12]。全体に一気に浸透させる前に、非エンジニアの中でも、SQLを理解している人や週報などの作成者(弊社ではリーダー層が多いです)に、先行してRedashの概要や操作方法などを布教していきました。Redashの良さを先にリーダー層に広めることで、データ可視化の基盤として広める際、その後の展開が楽になるからと睨(にら)んだためです。リーダーが率先して使うことやリーダーが操作に慣れている状態を先に構築することで、結果的にメンバーに指導がしやすくなり、勉強会後、一気にRedashが広まっていきました。

また、エンジニア同士のクエリの共有は「それRedashでお願いします」の一言を添えるだけで、秘伝のタレ化[注13]することは防げているように思います。



冒頭で述べた問題を打破するためにデータ可視化ツールの検討、導入をし、組織全体にSQLの操作を教えることで業務改善の足がかりになりました。SQL勉強会実施後、非エンジニアからSQLの質問があっても、テーブル名やカラム名を教えるとその場で理解を得られる場面が増え、エンジニアの作業工数削減が実現できました。その結果データ抽出依頼の件数が半分になりました。筆者がよく知るメンバーの中には、週報と月報の作成をRedashに移行し定期実行することで、毎週2時間かかっていた作成コストがほぼ0になったと聞きました。もちろん、すべてをRedashで代替できるわけではありませんが、Redash導入以前の次の問題は改善傾向にあるのではないかと思います。

- エンジニアがデータ抽出対応に割く工数が増えてしまう
- データ抽出完了待ちが発生する
- 機械的にできることを手作業で行ってしまう

また、SQLの習得の早いメンバーにはサブクエリやUNIONなどといった関数を使用し、エンジニアのサポートがなくても自ら学び、クエリを書き、実践している方も出てきました。

エンジニアとしては、クエリのFork機能を利用することでアドホックなデータ抽出依頼にも迅速に対応できています。また、○○さんしか知らないクエリもRedashで共有することで、少なくなってきています。Redashはエンジニアのエコシステムのなかでも利用する価値があるといえます。

このように、当初は「エンジニアしか扱えないもの」と思われていたSQLでしたが、「技術的なところを考えなくても実行できる環境であるRedashの導入」「非エンジニアの目線に立った勉強会」「運用を見据えたRedashの浸透戦略」これらを1つずつ考え抜き、着実に実行していくことで自分たちの思い描く理想とする組織像に一歩近づくことができました。SD

注12)「普及曲線」とは、まだ社会(今回だと組織)に普及していない新しいモノ(今回はRedash)などがどのような過程をへて普及していくかを分析した曲線です。新しいモノの採用者を5つ(イノベーター、アーリーアダプター、アーリーマジョリティ、レイトマジョリティ、ラガード)に分類し、その分布を示したものです。(エベレット・M・ロジャース著『Diffusion of Innovations』(1962年、邦題『イノベーション普及学』)より)

注13) 特定のエンジニアしか知らないクエリ。

クローリングハック

竹添 直樹、島本 多可子、田所 駿佑、萩野 貴拓、川上 桃子 著
A5判／336ページ
2,680円＋税
翔泳社
ISBN＝978-4-7981-5051-2

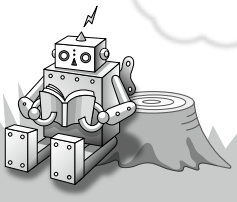
サイトを巡回してWebページの内容を取得する「クローラー」は、複数のサイトの情報をまとめたり、比較したりといったWebサービス運用には必須のプログラム。クローラーはさまざまなサイトを巡回する以上、その開発は一筋縄にはいかず、技術の面、マナーの面で押さえておくべき事柄が多くある。本書はそんな複雑なクローラーの開発について、基本知識、HTTP、文字化け対策、スクレイピング、認証の突破方法、JavaScriptが駆使されたリッチなサイトへのクローリング方法、クローリングマナーと、多面的に解説している。とくにマナーについては、robot.txtの読み方やアカウント情報の取り扱い方など、トラブルを避けるための注意喚起が豊富で、開発の先の運用やサービス運営も見据えた、誠実なつくりになっている。

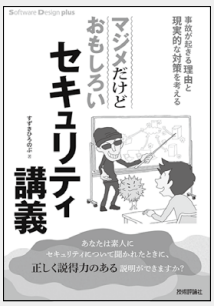
SRE サイトリライアビリティエンジニアリング

Betsy Beyer、Chris Jones、Jennifer Petoff、Niall Richard Murphy 編／澤田 武男 ほか 監訳／Sky株式会社 玉川 竜司 訳
B5変形判／590ページ
4,800円＋税
オライリー・ジャパン
ISBN＝978-4-87311-791-1

SREとは、Googleの「サイトリライアビリティエンジニア」が培ってきた、システム管理とサービス運用のためのベストプラクティスである。サイトリライアビリティエンジニアの任務は「システムの信頼性を高めること」にあるが、ソフト開発者とともにコードを書くこともあれば、運用のための各種ミドルウェアを構築することもあるなど、職能が決まっているわけではない。そのためSREについて記した本書も、SLO(サービス品質保証)、ロードバランシング、ジョブ管理、テスト、さらには新人への教育・引継など、ジャンルは多岐に渡り、それぞれの分野でGoogleのノウハウを吸収できる。従来のインフラエンジニアチームをSREチームとして統合しなおすという試みが日本のWeb企業でも始まっており、今後の大きな流れになるかもしれない。

SD BOOK REVIEW





マジメだけどおもしろいセキュリティ講義

すずきひろのぶ 著
A5判／416ページ
2,600円＋税
技術評論社
ISBN＝978-4-7741-9322-9

本誌連載「セキュリティ実践の基本定石」の中から、今なお学ぶところの多いセキュリティ事件を扱った回を厳選し、1冊に収録した本。世界各地で起きた事件の背景を探り、現実的な対策を考えるのが本書の主旨だ。サイバー攻撃の多くは人の無知・油断・思い込みにつけ込んでコンピュータに侵入してくるため、技術的対策だけでは不十分だということが、本書を読むとよくわかる。攻撃者の実体や攻撃の手口は、私たちが思う以上に多様だ。世界中のコンピュータに脆弱性を探るアクセスが毎日のように来ていること、国家ぐるみで盗聴が行われていること、自分のPCや口座が知らぬ間に不正利用されている可能性があることをあなたは知っているだろうか。効果的な対策のために、本書で事実を知るところから始めてはどうだろう。

Amazon Web Services負荷試験入門

仲川 樽八、森下 健 著
B5変形判／368ページ
3,800円＋税
技術評論社
ISBN＝978-4-7741-9262-8

クラウドで自社システムを構築すればきっと高性能に違いない。それは誤解だ。システム開発して実際に運用してみると、思ったとおりのパフォーマンスが出ないことはよくある話だ。機能要件は満たすけど、非機能要件は残念な結果と言い換えていいかもしれない。期待どおりの性能が出なければ、ユーザがシステムの価値を認めない時代。結局、開発に投じた人月も無駄になり、開発者の地位も下がってしまう。しかし、負荷試験を事前に行い、その結果を開発にフィードバックすることでこうした事態を避けることができる。手戻りが増えることになるかもしれないが、ちゃんと負荷試験を行い結果を出すことで、エンジニアへの信頼感も上がる。本書で具体的な負荷試験の方法を学び、日々の開発を充実したものにしてほしい。

ホワイトボックススイッチって何？

ユーザがソフトウェアを自作する新しいスイッチの形

Author 伊東 宏起（いとう ひろき）
さくらインターネット㈱
Mail contact@hekki.info Twitter @_nihi

Author 井上 喬視（いのうえ たかし）
さくらインターネット㈱
Mail t-inoue@sakura.ad.jp

近年登場した「ホワイトボックススイッチ」は、ネットワークの世界にソフトウェアのノウハウを持ち込んで開発・運用を効率化できる、領域横断的な特徴を持つ新しいスイッチの形です。本記事では、そもそもスイッチとは何をする装置なのかというところからはじめ、現状のスイッチの問題点、ホワイトボックススイッチのメリット、そしてホワイトボックススイッチにおける開発について解説します。

そもそもスイッチって何？

ホワイトボックススイッチのお話をする前に、まずは一般的なネットワークスイッチのことから理解を深めたいと思います。

ネットワークスイッチ

ネットワークスイッチは、ひとことでいうと「複数のデバイスがネットワークケーブルを介して互いに通信するための装置」で、正確にはスイッチングハブ、または略して単にスイッチと呼ばれています。

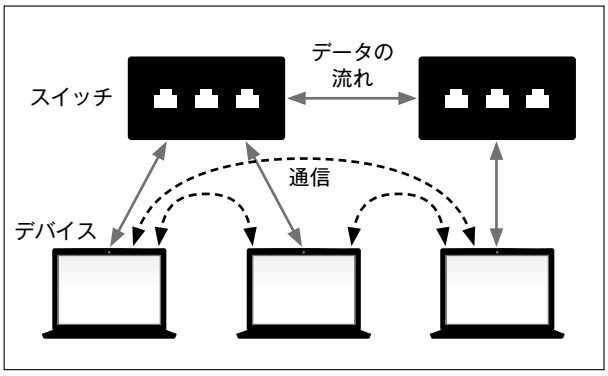
ネットワークに接続されるデバイスはすべてMACアドレス[注1]という固有の識別子を持っており、デバイスからネットワークにデータが送信されるときは、デバイス自身が持つ送信元のMACアドレスと、データを送信したい宛先のMACアドレスが付与されます。スイッチは、デバイスから送信されてきたデータから宛先MACアドレスを読み取り、適切な宛先へ転送するというのが基本的な役割です（図1）。

ネットワークプロトコル

ネットワークの世界では、異なるデバイス間の相互通信を可能にするため、「ネットワークプロトコル」という通信におけるルール（規約）が定められています。代表的なプロトコルとしては、HTTPやIP、TCP、Ethernetといったものが挙げられます。はじめに述べたMACアドレスは、Ethernetで取り決められているもののうちの1つです。

ネットワークプロトコルは「レイヤ」と呼ばれる、機能ごとに分けられた階層構造で分類されています。これはOSI参照モデルという考え方によるものです。OSI参照モデルは、ISO（国際

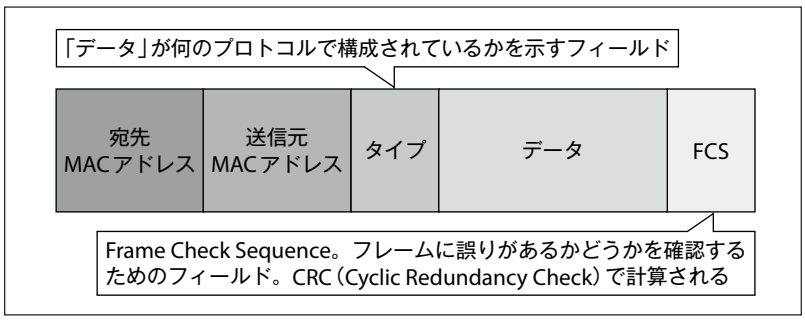
▼図1　スイッチとデバイスの接続



注1）IEEEによって管理されている、デバイスを一意に識別可能な48bitの識別子のこと。基本的には世界中のデバイスで重複しないように管理されている。

標準化機構)によって策定された、ネットワークが持つ機能を分類したモデルです。現実のネットワークのしくみとはやや乖離(かいり)もあるのですが[注2]、ネットワークのしくみが階層構造で構成されていることを理解するのに役立つため、ネットワークの世界においては基本中の基本とされています。

▼図2　Ethernetフレームの構造



・レイヤ7：アプリケーション層
・レイヤ6：プレゼンテーション層
・レイヤ5：セッション層
・レイヤ4：トランスポート層
・レイヤ3：ネットワーク層
・レイヤ2：データリンク層
・レイヤ1：物理層

ここでは7層あるレイヤのうち、スイッチにとくに関係しているレイヤ1～3を取り上げて解説します。

❖ レイヤ1(L1)――物理層

物理層は、ケーブルや無線といった伝送媒体を伝わる電気信号や光信号などについて取り決めを行うレイヤです。レイヤの中では最下層にあたり、伝送距離や通信速度が決定される、重要かつ通信の基礎となるレイヤです。Ethernetにおいては、LANケーブル(銅線)を用いた1Gbpsの通信速度を持つ1000BASE-Tといった規格や、光ファイバーケーブルを用いた10Gbpsの通信速度10GBASE-SRといった規格があります。また近年高速化が著しく、400Gbpsの通信速度を持つ規格も検討されています。

❖ レイヤ2(L2)――データリンク層

データリンク層は、ネットワークデバイス間で相互に通信するための手順について取り決めを行うレイヤです。Ethernetでは、Ethernetフレームといったデータ構造でやりとりを行う決まりとなっています(**図2**)。Ethernetフレームには送信元MACアドレスと宛先MACアドレスが含まれており、データがどのデバイスからどのデバイスに転送されるべきかを判別できるようになっています。冒頭でも述べましたが、このEthernetフレームを判別して適切な宛先へと転送する装置がスイッチです。

❖ レイヤ3(L3)――ネットワーク層

ネットワーク層は、複数のデータリンクを接続して相互に通信を行えるようにする手順を取り決めるレイヤです。前述のデータリンク層のみで通信を行う場合、デバイス数が増えたときのスケール面で問題があるため、ある程度のサイズで分割することが一般的な構成です。ネットワーク層の代表的なプロトコルはIP(インターネットプロトコル)です。IPは複数のデバイスをセグメントと呼ばれるグループに分割し、セグメントを越えて通信する場合の手順(この手順をルーティングと呼んでいます)を提供します。ルーティングを行う装置のことを、スイッチと区別してルータと呼んでいます(**図3**)。

基本的に、スイッチはレイヤ2までの機能までに対応した製品が一般的ですが、ミドルエンドからハイエンドの製品になると、レイヤ3(IP)

注2) たとえば、Ethernetでは1層と2層の両方が規定されている。

の処理機能を持つスイッチがあります。これらは通常のスイッチと区別して、「L3スイッチ」と呼ばれます。L3スイッチという呼称から、一般的なスイッチのことをL2スイッチとも呼んでいます。

そのほかのスイッチ

通常、スイッチというとL2スイッチ、もしくはL3スイッチのことを指しますが、それより上位レイヤのプロトコルに対応したスイッチも存在します。いずれもソフトウェアではなく、ハードウェアで処理するといった特徴があります。

❖ L4スイッチ

L4スイッチは、レイヤ4のプロトコルであるTCPをもとに転送先を決められるスイッチです。たとえば、新しいTCPセッションが到着したときにどのサーバに転送するかといった、ロードバランサの役割を果たします。

❖ L7スイッチ

L7スイッチは、レイヤ7のプロトコルをもとに転送先を決めることができるスイッチで、HTTPのURLをベースとしたものが一般的です。リバースプロキシのようなものをイメージするとわかりやすいでしょう。L7スイッチはアプリケーションデリバリコントローラ(ADC)と呼ばれることもあります。

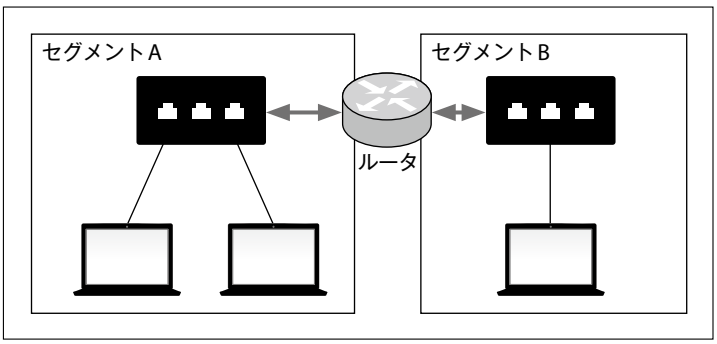
現状のスイッチの問題点

ネットワークスイッチについて簡単におさらいしましたが、本節では現状のスイッチにおける問題点について述べます。

ハードウェアの特殊性

ネットワークスイッチは、デバイスから送信

▼図3　ルーティング



▼表1　スイッチベンダと独自開発のOS

スイッチベンダ	OS
Cisco Systems	IOS、NX-OS
Juniper Networks	JUNOS
Arista Networks	EOS

される多量のフレームを高速に処理する必要があります。そのためスイッチの多くには、フレームの転送処理専用に開発されたスイッチングチップと呼ばれるASIC[注3]や、フレームの宛先を高速に参照するためのCAM[注4]と呼ばれる特殊なメモリを搭載しています。こういった特殊なハードウェアを搭載する都合上、スイッチベンダ各社でASIC、およびその上に載せるOSを独自開発しているケースが多くみられます(**表1**)。とくにASICはスイッチの肝となる部分であるため、これまではスイッチベンダが独自に開発を行うのが一般的でした[注5]。このように、スイッチではハードウェアとOSが密になっているため、「ハードウェアはベンダAにして、OSはベンダBにしよう」というような選択肢は基本的にありませんでした。

また、ハードウェアやOSはベンダそれぞれに独自色があり、運用のルールやクセが違うた

注3) Application Specific Integrated Circuit。特定用途のために設計されたLSI。ネットワーク機器ではおもにパケットの転送を担う。
注4) Content Addressable Memory。メモリ内のデータを検索して値を取得できる、連想配列のようなしくみを持つメモリのこと。
注5) 汎用的なスイッチングチップを開発するベンダもあり、各スイッチベンダからは汎用チップを搭載したモデルも販売されている。しかしながら、OSはスイッチベンダ独自のものが搭載されることが一般的。

▼図4　スイッチごとのコマンド例（設定モードの開始と終了、設定の参照）

```
# ベンダAのスイッチ
SwitchA>enable
SwitchA#configure terminal
SwitchA(config)#exit
SwitchA#show running-config

# ベンダBのスイッチ
user@SwitchB> configure
Entering configuration mode

{master:0}[edit]
user@SwitchB# show
```

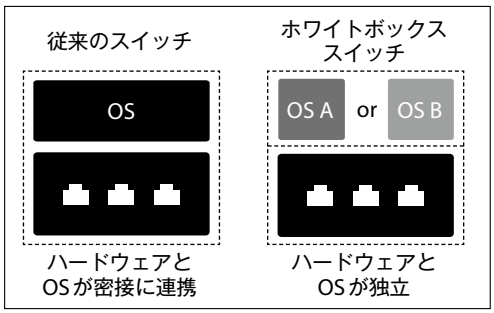
め、ネットワークエンジニアにとっては大きな負荷となります。複数ベンダのスイッチ同士で通信をさせる場合、基本的なプロトコルで問題が起こることは少ないのですが、プロトコルが複雑な場合、仕様にベンダ独自の解釈が含まれていたり、独自の拡張が加えられたりしている場合も少なくありません。その独自の拡張がデファクトスタンダードとなっているケースもあります。そのため、別ベンダ同士のスイッチを接続する場合には相互接続検証を行う必要があったり、通信障害などのトラブルが発生したときの切り分けに難航したりすることがあります。

自動化しにくい問題

一般的にネットワークスイッチでは、おもにCLIでコマンドを入力して設定を行います。入力するコマンドは各ベンダごとに違いがあり、統一されていません（**図4**）。最近では各スイッチベンダのOSにもAPI機能の実装が進みつつありますが、仕様は統一化されていないため、各ベンダごとに実装を変更する必要があるのが現状です[注6]。APIがない場合、対話的にコマンドを自動実行できるexpectやTera Termのマクロ機能といったツールを用いて、苦労しながらも運用の自動化をすることになります。

注6）OpenConfigという、ベンダに依存しないAPIを策定しようという動きもあるが、現時点で利用できるスイッチは少ない。

▼図5　ハードウェアとOSの関係



ホワイトボックススイッチのメリット

現状のスイッチには、ベンダごとの設計の違いから、運用の自動化がしにくいという問題がありました。ホワイトボックススイッチは、その問題をどのように解決するのでしょうか。ここからはホワイトボックススイッチのメリットについてみていきます。

そもそもホワイトボックススイッチって何？

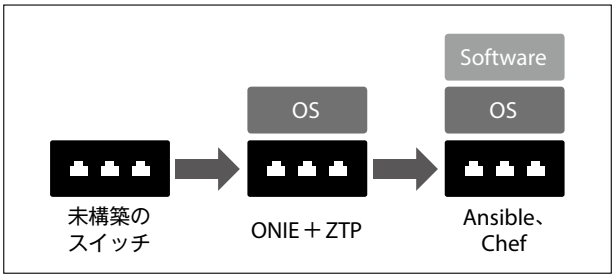
ホワイトボックススイッチのメリットについて紹介する前に、そもそもホワイトボックススイッチとは何かについて紹介します。

前節で述べたように、従来のスイッチはハードウェアとOSが密接に連携していました。それに対してホワイトボックススイッチは、一般的な汎用サーバのように扱うことができます。たとえば従来のスイッチの場合、あるベンダ製ハードウェアにはそのベンダ製のOSがインストールされていて、それ以外のOSを利用することはできません。これがホワイトボックススイッチの場合は、A社製のハードウェアにA社製のOS（A）だけでなくB社製のOS（B）もインストールできます（**図5**）。このように、ユーザがハードウェアとOSの組み合わせを選択して利用できるという特徴があります。

現在、ホワイトボックススイッチ用のハードウェアは台湾のベンダを中心に複数のベンダがラインナップをそろえており、サーバ収容部分

ホワイトボックススイッチって何？

▼図6　ホワイトボックススイッチ構築の流れ



などで利用する1G多ポートの低価格なモデルから、コアネットワーク向けの100G多ポートの高価なモデルまで、数多くの製品が登場しています。OSに関してはホワイトボックススイッチ用OSをメインに取り扱うベンダもいくつか登場していますが、従来のスイッチ向けのルーティングエンジンを開発しているベンダの参入や、MicrosoftやFacebookといったハイパージャイアントと呼ばれる大企業が、自社データセンター向けに独自のOSを開発する例も見られます。こういった多種多様な製品群の中から、ユーザはネットワークの規模感、利用したいプロトコル、周辺機能、サポート、価格など、いろいろなことを吟味しながら、ハードウェアとOSの組み合わせを選ぶことができます。

それでは、ホワイトボックススイッチの特徴をふまえたうえで、ホワイトボックススイッチのメリットを紹介します。

メリット1 スイッチがサーバのように使える

ホワイトボックススイッチ用OSは、大半のものがLinuxベースで実装されており、まるでNICがたくさん搭載されているLinuxのようにも見えます。筆者が調査した限りではDebianをベースにしたものが多く、通常のLinuxを操作したことのある方なら、それほど違和感なく使い始められる印象があります。また、Linuxをベースにしているということは、Linux用に開発されたソフトウェアをスイッチ上で実行できる[注7]ということになります。具体的な例として、監視ソフトウェアで有名なZabbixのagentを導入すれば、サーバと同じしくみで監視やアラート発報などができるようになります。

また構築に関しても、ホワイトボックススイッチ独自の機能であるONIE[注8]を利用してOSのインストールを行い、ホワイトボックススイッチに限らずさまざまなスイッチで利用できるZTP[注9]をベースにOS上の初期設定を行ったあとは、AnsibleやChefといったプロビジョニングツールを使えば、サーバと同じように機器の役割ごとの共通設定を行うことができます(**図6**)。

プロビジョニングツールを利用して構築を行っていれば、運用中に設定を変更する場合もプロビジョニングツールのコンフィグを変更して差分を適用するといった、サーバと同じフローのメンテナンスを行うことができます。さらに、機器故障に伴うハードウェア交換のような障害対応の場面でも、サーバと同じような手順で障害対応を行えるようにしくみを作ることもできるので、運用者の負荷を下げることが期待できます。

メリット2 自分でソフトウェアを実装できる

前項で紹介したとおり、ホワイトボックススイッチ用のOSは大半のものがLinuxベースで実装されていますので、一般的なLinuxのようにシェルスクリプトを実行したり、各種プログラミング言語の実行環境を導入したりすることができます。それにより、スイッチ上で動作するソフトウェア(後述)を、ユーザが自分で実装して導入できるようになりました。また、これらが可能になったことで、異なるOSが動作す

注7) あくまでもLinuxをベースにカスタマイズされたOSなので、すべてのソフトウェアが利用できるわけではない。
注8) Open Network Install Environment。ホワイトボックススイッチ特有の機能で、OSのインストールを実行するためのブートローダー。
注9) Zero Touch Provisioning。ユーザが手動で操作をすることなく、機器の初期設定などを自動で行える機能。

るスイッチ上で同じソフトウェアを実行できるようになりました。そのソフトウェアのレイヤでOSごとの違いを吸収してしまえば、結果として、どのスイッチで何のOSが動作しているのかを意識する必要がない状態を作り出すことができます(**図7**)。

弊社ではこのメリットを活かして、ホワイトボックススイッチ用の制御フレームワークを実装し、次のようなオペレーションを完全自動化したうえで商用サービスに導入しています。

①ホワイトボックススイッチの下記項目の設定変更
- リンクのup/down
- access/trunk VLAN ID
- Policing/Shaping rate
- ACL(Source IP/MAC address filter)

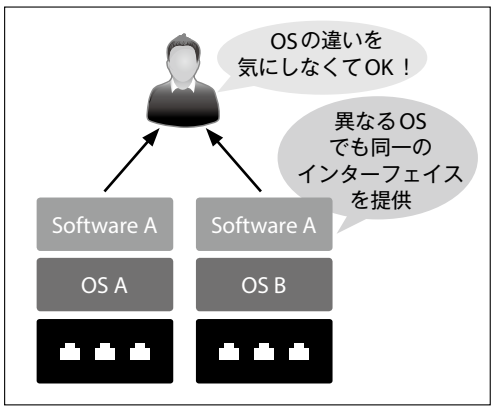
etc...

②上記の設定変更のタイミングに合わせた最新の設定バックアップ

さらに弊社では、PythonのREST API用フレームワークであるFlask-RESTfulを利用したホワイトボックススイッチ用のREST APIサーバを開発し、「異なるホワイトボックススイッチ用OSでも同じエンドポイントを叩けば同じJSONオブジェクトが取得できる」というコンセプトの、スイッチ向けREST APIの開発にも取り組んでいます。このREST APIサーバは次節でサンプルコードを交えつつ詳しく紹介をしたいと思います。

サーバの世界では自分で実装したソフトウェアを各サーバで導入し、システムを構築することはこれまで当たり前のように行われてきました。ホワイトボックススイッチの例から、そんな常識がネットワークの世界にもやってきたと感じます。また、SDN[注10]というコンセプトが現れて数年経ちますが、ホワイトボックススイッチの登場によって、ますますプログラミングの

▼図7　OSとソフトウェアの関係



知識やスキルがネットワークエンジニアにも求められるようになり、ネットワークエンジニアやソフトウェアエンジニアといった垣根がどんどんあいまいになっていくのでは、と感じています。

ホワイトボックススイッチ向けソフトウェアの開発

ここからは、ホワイトボックススイッチに導入するソフトウェアの実際の開発について解説します。

スイッチソフトウェアの種類

ホワイトボックススイッチ向けソフトウェアの開発といっても、何を作るのでしょうか？ホワイトボックススイッチでのソフトウェア開発には、大きく分けて2つの種類があります。

1つ目は、ASICを制御してパケットの転送を担うソフトウェアの開発です。従来のスイッチでは、これらソフトウェアはネットワーク機器ベンダが開発を行って製品に組み込みますので、ユーザ自身が追加で実装を行うことはできませんでした。仮に新しいプロトコルなどに対応したバージョンが必要な場合は、ユーザがネットワーク機器ベンダに実装の依頼をして、できあがりを待つという選択肢を選ぶしかないということになります。ホワイトボックススイッチでも同様に、ホワイトボックススイッチ用OSベ

注10) Software Defined Networking。ソフトウェアによって、ネットワークを定義および実現するしくみ。

ンダが開発を行います。しかしホワイトボックススイッチ用OSでは、ユーザに対してASICを制御するためのAPIやSDKが公開されており、ユーザ自身で開発を行うこともできます[注11]。

2つ目は、ホワイトボックススイッチ用OSを一般的なLinuxとみなしたソフトウェアの開発です。一例として本節では、「メリット2」で紹介した「ホワイトボックススイッチ用のREST APIサーバ」を題材にして、スイッチポートのステータスをJSONで取得するホワイトボックススイッチ向けソフトウェアを開発するうえでの開発の流れを紹介します。

注意点

開発の流れに入る前に、ホワイトボックススイッチ向けソフトウェアを開発するうえでの注意点を挙げたいと思います。

ホワイトボックススイッチには、ASICのようなパケットの転送に特化したリソースだけでなく、普段私達がCPU、メモリ、ストレージと呼んでいるような、汎用的なコンピュータリソースも搭載されています。しかし、それらは低スペックな場合があり、実装するソフトウェアのリソース消費量について注意する必要があります。また、CPUはx86_64のものばかりではなく、ARMやPowerPCなどが搭載されていることがあります[注12]。その場合、C言語やGo言語といったコンパイラ型言語では、CPUアーキテクチャを意識した開発やコンパイルを行う必要があります。

開発環境を作る

❖ 開発環境の紹介

それでは、開発環境について紹介します。ここでは、Cumulus Networks社[注13]が提供しているホワイトボックススイッチ用OSの1つである「Cumulus Linux」[注14]を題材とし、その上で動作するソフトウェアを開発します。とは言え、Cumulus Linuxを動作させるためのホワイトボックススイッチ用のハードウェアやライセンスをすぐに用意するのは難しいので、同社が提供する仮想アプライアンス「Cumulus VX」[注15]を利用します。Cumulus Networks社以外にも仮想アプライアンスを提供するベンダはいくつかありますが[注16]、「無料で利用できる」「主要ハイパーバイザをサポートしている」といった点から、幅広い読者の方が利用できると考えて本製品を採用しました。

また本記事では紹介しませんが、それぞれの仮想アプライアンスとGitHub、Jenkins、Slackなどを組み合わせれば、継続的インテグレーションなどソフトウェア開発の世界では当たり前に行われている開発手法を、そのまま利用することができます。

❖ Cumulus VXを立ち上げる

前置きはここまでにして、さっそくCumulus VXを立ち上げましょう。Cumulus Networks社のサイトからCumulus Networks accountを作成すれば仮想アプライアンスのイメージがダウンロードできますので、手元の環境に応じて最適なイメージをダウンロードしてください[注17]。なお、イメージの詳細な仕様はTechnical Documentation[注18]の[Cumulus VX]-[Getting Started]に記載されています。

イメージから仮想アプライアンスを作成すると、Cumulus VX上でeth0として認識されるマネージメントポート(アダプター1)と、swp1〜7

注11) 可能ではあるものの、ネットワークの基本的な知識はもちろん各種プロトコルに関しての深い知識、プログラミングスキルも必要なため、難易度は非常に高い。また、ベンダよってはASICの仕様がオープンになっておらず、NDA(秘密保持契約)を結ばなければならないなどの制約もあるため、始めるまでのハードルもある。

注12) ホワイトボックススイッチが登場したころはPowerPCを搭載したモデルが主流だったが、最近はx86/x86_64やARMを採用したハードウェアが多くなってきた。

注13) URL https://cumulusnetworks.com

注14) URL https://cumulusnetworks.com/products/cumulus-linux

注15) URL https://cumulusnetworks.com/products/cumulus-vx

注16) ホワイトボックススイッチOS用ベンダだけでなく、従来のネットワーク機器ベンダでも提供され始めている。

注17) 本記事ではCumulus VX 3.4.1、VirtualBox用イメージを利用。

注18) URL https://docs.cumulusnetworks.com/category/docs

▼図8　仮想マシンの構成

として認識されるスイッチポート用の仮想ネットワークインターフェース(アダプター2〜8)が定義されていることがわかります(**図8**)。なお、アダプター1はデフォルトのNATからブリッジアダプターに変更していますが、それ以外は変更せずに利用しています。

準備が整ったら仮想アプライアンスを起動します。Cumulus VXの基本的な機能はCumulus Linuxと同じと考えて問題ありませんので、ダウンロードしたものと同じバージョンのCumulus Linuxのドキュメントを参照してください。

❖ 実行環境の構築

Cumulus VXにログインしたら、REST APIサーバの実行環境を作成します。

まずは、swp1〜7として定義されているスイッチポートのコンフィグを作成します(**図9**)。この手順はCumulus Linux固有のものですので、最初はあまり深く考えずに写経してしまっても問題ありません。

次にPythonのバージョンを確認します。

▼図10　Pythonのバージョンを確認

```
# Python 2.x系のバージョン確認
$ python --version
Python 2.7.9

# Python 3.x系のバージョン確認
$ python3 --version
Python 3.4.2
```

筆者の環境ではPython 2.7.9とPython 3.4.2がインストールされていましたので(**図10**)、Python 3.4.2を利用することとします。

前節で紹介したとおり、PythonのREST API用フレームワークであるFlask-RESTfulを利用します。また、Pythonのライブラリのインストールにはpipを利用しますので、あらかじめインストールしておきます。pipのインストールが終わったら、Flask-RESTfulをインストールしましょう(**図11**)。

ここまでででCumulus VXの起動とFlask-RESTfulのインストールが完了しました。Pythonを利用したことのある方ならおわかりかと思いますが、通常のLinux上でPythonを利用する場合と同じように環境構築ができます。

▼図9　スイッチポートのconfigの作成

```
# swp用のconfigの作成
$ for i in $(seq 1 7); do
> echo -e "auto swp${i}\niface swp${i}\n" | \
> sudo tee -a /etc/network/interfaces.d/swp.intf
> done

# ネットワークインターフェースのリロード
$ sudo ifreload -a
```

▼図11　pipとFlask-RESTfulのインストール

```
# Python 3に紐づくpipのインストール
$ wget https://bootstrap.pypa.io/get-pip.py
$ sudo python3 get-pip.py

# pipからFlask-RESTfulをインストール
$ sudo python3 -m pip install flask_restful
```

▼リスト1　api.py

```
from flask import Flask
from flask_restful import Resource, Api

from swp import SwitchPort

app = Flask(__name__)
api = Api(app, catch_all_404s=True)

class SwitchPortView(Resource):
    def get(self, swp_id):
        port_data = {}
        port_data['port'] = swp_id

        try:
            swp = SwitchPort(swp_id)
        except ValueError as e:
            port_data['message'] = str(e)
            return port_data

        port_data['link'] = swp.get_link()
        port_data['mtu'] = swp.get_mtu()

        port_data.update(swp.get_rx_state_dict())
        port_data.update(swp.get_tx_state_dict())

        return port_data

api.add_resource(SwitchPortView, '/api/v1/swp/<string:swp_id>/')

if __name__ == '__main__':
    app.run(debug=True,host='0.0.0.0')
```

「APIのエンドポイントの定義」に対応するViewの定義

← APIのエンドポイントの定義

❖ スイッチ上で動作するREST APIサンプルを動かす

それではREST APIのサンプルコードを動かしてみましょう。サンプルコードの簡単な仕様を次に示します。

- http://host:port/api/v1/swp/:id/ にGETリクエストを発行すると:idで指定した各スイッチポートの情報が取得できる
- hostはCumulus VXのeth0のIPアドレス
- port/はFlask-RESTfulのデフォルトポートである5000
- 取得できる情報は「インターフェースのリンク状態」「RX/TX（受信／送信）のパケットカウンタの値」「MTU」

任意のディレクトリにapi.py（**リスト1**）とswp.py（**リスト2**）を作成します。api.pyはFlask-RESTfulを制御する処理が記述されたファイルで、コード全体では中心的な処理を担います。本ファイルはCumulus Linuxに依存する部分はありませんので、一般的なFlask-RESTfulの記法のみで構成されています。swp.pyは、仕様の

▼図12　REST APIの実行

```
# Cumulus VX上で実行
$ python3 api.py

# Cumulus VXとは別の端末で実行
# host、portは各自の環境に合わせて読み替えてください
$ curl -s http://host:port/api/v1/swp/1/
{
  "txOk": "288",
  "txDrp": "0",
  "rxOk": "1757",
  "link": "up",
  "port": "1",
  "txErr": "0",
  "rxDrp": "123",
  "rxErr": "0",
  "txOvr": "0",
  "rxOvr": "0",
  "mtu": "1500"
}
```

▼リスト2　swp.py

```
from json import loads
from subprocess import getoutput

class SwitchPort(object):
    def __init__(self, swp_id):
        self.swp = 'swp{}'.format(str(swp_id))

        netstat_dict = loads(getoutput("cl-netstat -j"))

        if self.swp in netstat_dict:
            self.netstat_swp_dict = netstat_dict[self.swp]
        else:
            raise ValueError('{} is not fonund.'.format(self.swp))

    def get_link(self):
        is_link_down = 'state DOWN' in getoutput('ip link show dev {}'.format(self.swp))

        if is_link_donw:
            return 'down'
        else:
            return 'up'

    def get_mtu(self):
        return str(self.netstat_swp_dict['MTU'])

    def get_rx_state_dict(self):
        rx_state = {}

        rx_state['rxOk'] = str(self.netstat_swp_dict['RX_OK'])
        rx_state['rxDrp'] = str(self.netstat_swp_dict['RX_DRP'])
        rx_state['rxErr'] = str(self.netstat_swp_dict['RX_ERR'])
        rx_state['rxOvr'] = str(self.netstat_swp_dict['RX_OVR'])
        return rx_state

    def get_tx_state_dict(self):
        tx_state = {}

        tx_state['txOk'] = str(self.netstat_swp_dict['TX_OK'])
        tx_state['txDrp'] = str(self.netstat_swp_dict['TX_DRP'])
        tx_state['txErr'] = str(self.netstat_swp_dict['TX_ERR'])
        tx_state['txOvr'] = str(self.netstat_swp_dict['TX_OVR'])
        return tx_state
```

Cumulus Linuxに依存する処理を行い、データを取得

Flask-RESTfulに合わせたデータ構造に変換

中で紹介した「取得できる情報」の元になるデータを、Cumulus Linuxに依存する処理を行って取得し、Flask-RESTfulに合わせたデータ構造に変換する処理を担います。

サンプルコードを作成したらREST APIサーバ(api.py)を起動して、別の端末からエンドポイントにリクエストを発行してみましょう(**図12**)。ここではswp1の状態を取得しています。ここまでできたら、一通り完成です。

◆ ◆ ◆

いかがでしたでしょうか。本記事で従来のスイッチとホワイトボックススイッチの違い、ホワイトボックススイッチの拡張性などについて知っていただけましたでしょうか。筆者は、これからのネットワークの制御や運用には、ネットワークに関する知識やスキルだけではなく、プログラミングのスキルも必要不可欠になってくると感じています。それは言い換えれば、プログラマの力がネットワークの分野で必要になってきている、ということです。本記事がプログラマのみなさんの目に止まり、新しいネットワーク技術を作り出す一歩になれば幸いです。SD

速攻で仕事を片づける
CLI力をマスターせよ!

シェル芸人からの挑戦状

今回のシェル芸人 上田 隆一、田代 勝也、山田 泰宏、eban
編者 山田 泰宏

第3回 ネットワーク(その1)

端末同士での通信

本連載も3回目になりました。今回のテーマは「ネットワーク」です。すなわち、端末同士の通信に関連する問題を出題します。CLI端末からネットワークに関する操作ができると、ほかの端末同士の疎通確認やスイッチ、APIの動作テストをはじめ、何かと役に立ちます。

HTTP/HTTPS通信の疎通確認のみであれば、GUIで簡単に使えるWeb APIテストツールの力を借りるのが手ごろかもしれません。しかし、UNIXのコマンドには、低いレイヤでの通信の操作を可能にする、自由度の高いコマンドが数多く用意されています(その分、ちょっと応用すると悪いことにも使えてしまうのですが……)。

環境

今回の問題の検証に使ったOSはUbuntu Server 16.04 LTSです。コマンドはさまざまなものを試すのでその都度インストールします。解答の解説においてインストールが必要なことを説明します。ただ、aptで簡単にインストールできるものについてはインストールする手順を省略することがあります。Ubuntuですので、基本的なコマンドについては、GNU core utilitiesの使用を前提にしています。シェルは前回に引き続きbashを使います。バージョンは4.3です。

問題1

ネットワークデバイス一覧

出題、解答、解説:上田

初級 ★

リスト1-1のように、OSから認識されているネットワークデバイス(eth0など)の一覧を作ってみましょう。

▼リスト1-1 出力の例

```
lo
eth0
wlan0
```

解答

ifconfigコマンドを使う方法がまず思いついてしまうのですが、Linuxでは非推奨なのでipコマンドを使うことにします。`ip link`と打つと、**図1-1**のように認識されているデバイス一覧が表示されます(IPアドレスも見たい場合には`ip addr`です)。

あとは**図1-2**のように、行頭が数字の行を抽出し、awkで2列目を抜き出して、tr -d :でコロン(:)を消せば答えになります。

このような文字列の切り出しができると、報告書か何かを作るときに便利ですので、普段から手遊びで練習しておくと良いでしょう。

別解

procの下にあるファイルやtcpdumpコマンドを使う方法もあります(**図1-3**)。

▼図1-1 ipコマンドでデバイス一覧を出力

```
↓出力が長いので各行の右側は省略
$ ip link
1: lo: <LOOPBACK,UP,LOWER ...
    link/loopback 00:00:0 ...
2: eno1: <BROADCAST,MULTI ...
    link/ether 0c:c4:7a:0 ...
3: enp4s0: <NO-CARRIER,BR ...
    link/ether 0c:c4:7a:0 ...
9: eth0: <NO-CARRIER,BROA ...
    link/ether 22:3c:ae:7 ...
```

tcpdumpを使うほうは、ip linkでは出てこないusbmon1やnfqueueなど、筆者(上田)にとっては見慣れないものも表示されて興味が出てしまいました。

ほかにも解答例あれば、Twitterなどで教えていただければと思います。

▼図1-2 解答

```
$ ip link | grep ^[0-9]
1: lo: <LOOPBACK,UP,LOWER_ ...
2: eno1: <BROADCAST,MULTIC ...
(..略..)

$ ip link | grep ^[0-9] | awk '{print $2}'
lo:
eno1:
(..略..)

↓これが答え
$ ip link | grep ^[0-9] | awk '{print $2}' | tr -d :
lo
eno1
enp4s0
eth0
```

▼図1-3 別解(田代)

```
↓procを参照する方法
$ cat /proc/net/dev | awk 'NR>=3{print $1}' | sed 's/:$//'
↓tcpdumpを使う方法
$ tcpdump -D | awk '{print $1}' | sed 's/^[0-9][0-9]*\.//'
```

問題2

telnetコマンドでHTTP通信

出題、解答、解説:田代

初級 ★

telnetコマンドでWebサーバに接続すると、HTTPリクエスト(以下、リクエスト)を手入力して直接会話ができます。

図2-1は「www.google.co.jp」にHEADリクエストを送る操作です。リクエストは最後に空行が必要なことに注意してください。

図2-1と同様に、任意の文字列をそのままHTTPリクエスト(以下、リクエスト)として送る操作をワンライナーで実現し、HTTPレスポンス(以下、レスポンス)を取得してください。

また、HTTPSで接続する場合はどうすればいいでしょうか。

▼図2-1 telnetでHTTP通信をする

```
$ telnet www.google.co.jp 80
Trying 172.217.26.99...
Connected to www.google.co.jp.
Escape character is '^]'.
HEAD / HTTP/1.1              ←入力
Host: www.google.co.jp       ←入力
                             ←入力(空行)
HTTP/1.1 200 OK              ┐
(..略..)                     ├ レスポンス
Vary: Accept-Encoding        ┘

^]                           ←入力(Ctrl-]のあと、Enterを押下

telnet> quit                 ←入力
Connection closed.
```

解答(HTTPの場合)

まずはリクエストの文字列を出力するコマンドを作成します。ここではprintfコマンドを使います(図2-2)。改行は\nで指定します。最後に空行を出力していることに注意してください。

この文字列をtelnetコマンドに標準入力から流し込んでやればいいのですが、図2-3のようにパイプでつなげただけではレスポンスが表示されません。これはtelnetコマンドがレスポンスを表示する前に終了してしまうためです。

そこでリクエスト送信後に1秒程度待ち時間を入れると、レスポンスが表示されます(図2-4)。待ち時間を入れるにはsleep 1のコマンドを使い、全体を()で囲ってグループ化します。待ち時間の1秒は目安であって、実際にはもっと短くても大丈夫なことが多いでしょう。何らかの理由で1秒以上必要な場合もあり得ます(回線品質が悪い、サーバの負荷が大きいなど)。

▼図2-2 リクエストの文字列を出力する

```
$ printf 'HEAD / HTTP/1.1\nHost: www.google.co.jp\n\n'
HEAD / HTTP/1.1
Host: www.google.co.jp
                          ←空行
```

▼図2-3 リクエスト文字列をtelnetの標準入力に渡す

```
$ 図2-2のコマンド | telnet www.google.co.jp 80
Trying 172.217.26.99...
Connected to www.google.co.jp.
Escape character is '^]'.
Connection closed by foreign host.
 ↑レスポンスが表示されずに切断される
```

▼図2-4 解答(HTTPの場合)

```
$ ( printf 'HEAD / HTTP/1.1\nHost: www.google.co.jp\n\n'; sleep 1; ) | ↵
telnet www.google.co.jp 80
Trying 172.217.26.99...
Connected to www.google.co.jp.
Escape character is '^]'.
HTTP/1.1 200 OK
(..略..)
Vary: Accept-Encoding

Connection closed by foreign host.
```

解答(HTTPSの場合)

HTTPS接続の場合は、telnetの代わりに、openssl コマンドのs_clientサブコマンドを使います(図2-5)。標準エラー出力に出力されるサーバ証明書の検証情報は/dev/nullにリダイレクトして消しています。

図2-5では、サーバ証明書や暗号化に関する情報も出力されます。HTTPレスポンスのみを出力するには-quietオプションを使います(図2-6)。こ

▼図2-5 openssl コマンドでHTTPS通信をする

```
$ ( printf 'HEAD / HTTP/1.1\nHost: www.google.co.jp\n\n'; sleep 1; ) | ↵
openssl s_client -connect www.google.co.jp:443 2> /dev/null
CONNECTED(00000003)
---
Certificate chain
(..略..)
HTTP/1.1 200 OK
Date: Mon, 21 Aug 2017 15:15:49 GMT
(..略..)
Transfer-Encoding: chunked
Accept-Ranges: none
Vary: Accept-Encoding
```

▼図2-6 解答(HTTPSの場合)

```
$ ( printf 'HEAD / HTTP/1.1\nHost: www.google.co.jp\n\n'; sleep 1; ) | ↵
openssl s_client -connect www.google.co.jp:443 -quiet -no_ign_eof 2> /dev/null
HTTP/1.1 200 OK
Date: Mon, 21 Aug 2017 15:24:26 GMT
(..略..)
Transfer-Encoding: chunked
Accept-Ranges: none
Vary: Accept-Encoding
```

の場合は-no_ign_eofオプションを併用しないと、入力が終了したあとに切断されません[注1]。

補足

図2-7のようにtelnetコマンドの代わりにncコマンド(netcat)を使うと、リクエスト送信後の待ち時間(sleep 1)が不要です。

別解(HTTPの場合)

図2-8は山田さんからいただいた別解です。HTTPであれば、bashに備わっている機能を利用して同様のことができます。「/dev/tcp/ホスト名/ポート名」というパスとリダイレクトを使うとTCP通信を可能にするファイルディスクリプタ[注2]が得られます。これに対して標準入力を与えると、その入力内容のリクエストを送ってくれます。

また、入力を与えたあとにcatなどで内容を読み出すとレスポンスが得られます。誌面の関係上詳しい説明は割愛しますが、興味のある方は検索エンジンで「bash TCP通信」を検索して調べてみてください。

注1) man s_clientのCONNECTED COMMANDSの章を参照。

注2) ファイルディスクリプタというのは、OSがファイルや標準入出力の参照として使うキーです。たとえば標準入出力は、ファイルディスクリプタを経由しているため、データの出力先や入力元としての利用もできます。

▼図2-7 ncコマンドでHTTP通信をする

```
$ printf 'HEAD / HTTP/1.1\nHost: www.google.co.jp\n\n' | nc www.google.co.jp 80
```

▼図2-8 別解(HTTPの場合)(山田)

```
$ exec 3<> /dev/tcp/www.google.co.jp/80 && (echo 'HEAD / HTTP/1.1'; echo 'Host: www.google.co.jp';
echo;) >&3; (cat & sleep 1; kill $!) <&3
```

問題3

Content-Lengthの改ざん

出題、解答、解説:山田

中級 ★★

Webサーバのようにマシンに何かのHTTPのリクエストが来たら、レスポンスを返すワンライナーを考えてください。

基本的にWebサーバはレスポンスするデータのバイト数をContent-Lengthというヘッダに含めます。この問題でも、レスポンスを返す際にはヘッダにContent-Lengthを含めることとします。ただしこの際、ヘッダのContent-Lengthに記述する値を、実際のデータの量よりも大きくしてください。

そして、CLI端末をもう1つ開き、curlコマンドでリクエストを投げて、そのレスポンスを受け取ってみてください。そのとき、何が起こるかも確認してください。

解答

問題2の最後に出てきたncコマンドは、リクエストを送るのにも、受け付けるのにも使える便利なコマンドです。-l(listen)オプションに続けてポート番号を指定すると、そのポート番号でリクエストを待ち受けます。ncに標準入力から文字列を与えると、その文字列が受けたリクエストに対するレスポンスとして使われます。

ここでは**リスト3-1**の文字列の内容を、リク

▼リスト3-1 レスポンス内容

```
HTTP/1.1 200 Ok          ←ヘッダ部分
Content-Length: 5        ←ヘッダ部分(実際のレスポンスボディより1バイト多い値)
                         ←空行(改行)が入る
test                     ←ここがデータの内容(レスポンスボディ、末尾に改行はなし)
```

エストに対応するレスポンスとして返そうと思います。testという文字列は4バイトですが、ヘッダ部分のContent-Lengthの値は5(バイト)となっていますね。つまり、実際のデータよりも1バイト分大きな数字が指定されています。

リクエストを受けたら**リスト3-1**の内容のレスポンスを返すコマンドは、**図3-1**になります。これを実行すると、8080ポートでリクエストの待ち受け状態になります。

ここでもう1つCLI端末を開き、待ち受けている8080ポートに対してリクエストを送ってみます(**図3-2**)。ただし、レスポンスとして返ってくるデータは先ほどの不正な値を持ったデータ(**リスト3-1**)です。curlはこのようなデータを受け取ると受信データが1バイト足りない旨を伝えるエラーメッセージを表示します。

ちなみに、なぜcurlは残り1バイトを受信するために待機状態にならず、すぐにエラーが表示されてコマンドが終了するのでしょう?

それはncコマンドが標準入力から受けた内容をすべて送信し終わったあとに、TCP通信の終了を意味するFIN + ACKパケットを返して通信を終了させているからのようです。

別解

socatという、サーバ側としてもクライアント側としてもソケット通信ができるコマンドがあり、これを使って解くこともできます。詳細は割愛しますが、**図3-3**のコマンドでできます。

補足

curlには--ignore-content-lengthというオプションがあり、**図3-4**のようにリクエストを送ると、エラーが出なくなるようです。

◆ ◆ ◆

ところで、このような変なレスポンスを受信した際のほかのHTTPクライアントの挙動も気になりませんか? curl以外では、たとえばみなさんが普段お使いのWebブラウザが挙げられます。

図3-1のワンライナー実行後に、ブラウザのアドレス欄に「http://localhost:8080」と入力して試してみましたが、「接続が切断されました」のようなエラーメッセージが表示されるものもあれば、何事もなかったかのように「test」と表示するものもあるようです。

▼図3-1 解答(リクエストを待ち受けて、レスポンスを返すワンライナー)

```
$ ( echo -e "HTTP/1.1 200 Ok\nContent-Length: 5"; echo; printf test ) | nc -l 8080
```

▼図3-2 解答(もう1つの端末でリクエストを送信する)

```
$ curl http://localhost:8080/
test  ←レスポンス
curl: (18) transfer closed with 1 bytes remaining to read  ←エラーメッセージ
```

▼図3-3 別解 socatを使った解答(eban)

```
$ socat -v tcp-listen:8080,crlf,reuseaddr,fork system:'echo HTTP/1.1 200 OK;
echo "Content-Length: 5"; echo; printf test'
```

▼図3-4 Content-Lengthのエラーが出力されないようにする方法

```
$ curl --ignore-content-length http://localhost:8080/
test
```

問題4

pingのパケット解析

出題、解答、解説：山田

上級 ★★★

この問題でもCLI端末を2つ使います。片方の端末はgihyo.jpにpingコマンド[注3]でICMPパケットを投げるために使います。そして、もう片方の端末も使って次に示す問題を解いてください。

【小問1】

pingコマンドは、ICMP Echo Requestというパケットを、引数として指定されたホストに送ります。そのパケットを受け取ったホストは、ICMP Echo Replyというパケットを返します[注4]。pingを動作させ、最初に送信する1回分のICMP Echo Requestのパケットの内容を16進数で出力してください。内容に多少余計なデータが入っていてもかまいません。

【小問2】

表4-1はpingのICMP Echo Requestのパケットの構造です[注5]。0xNNという表現は、その個所のデータが16進数の数値でNNで表現されるという意味です。ちなみに、8ビット＝1バイトとします[注6]。**表4-1**に「データ(可変長)」とあるように、pingのパケットには自由に可変長のデータを含めることができます。

pingでパケットを送ったとき、この可変長のデータ部分のみを16進数表記で出力するワンライナーを考えてください。

▼表4-1　ICMP Echo Requestのパケット構造

データの内容	データのサイズ
タイプ(0x08)	1バイト
コード(0x00)	1バイト
チェックサム	2バイト
識別子	2バイト
シーケンス番号	2バイト
データ(可変長)	可変長

注3) ご存じの方も多いとは思いますが、pingはネットワーク上のあるホストまでパケットが到達可能かどうか、あるいはホストが生きているかどうかを確認するためによく使われるコマンドです。

注4) パケットのフィルタリングの設定で制限されている場合などもあるため、常に返せるというわけではありません。

注5) RFC 792で定められています。https://tools.ietf.org/html/rfc792#page-14

注6) 厳密に言うと、常に1バイト＝8ビットとも限らないので、このような話のときは本来「オクテット」と言うべきですが、「バイト」のほうがなじみのある読者のほうが多い気がするので、バイトを使います。

解答(小問1)

パケットの生データの監視にはtcpdumpというコマンドがよく使われます。プロトコルや監視対象のネットワークデバイスの名前など、パケットを監視する条件を細かく指定すると、その条件に合致したパケットを標準出力やファイルに書き出せます。

端末を2つ開きます。まず片方の端末で、tcpdumpを実行してICMPプロトコルのパケットの監視を開始します(**図4-1**)。-xオプションで、受信したデータを16進数表記で表示できます。そして-c 1オプションを付けることで、1つパケットを受け取ったら動作が終了します。また、icmpという引数が指定されていますが、これを指定することでpingが送るICMPプロトコルのパケットのみを監視できます。

このコマンドに続き、grepが指定されています。tcpdumpの出力はIPアドレスやパケットの種類の情報などを含みます。16進数の表記の個所は[タブ文字]0xという文字列が含まれています。そこでタブ文字を含む行をgrepで抽出することで、16進数表記の数値を含んだ行のみを出力します。grepの引数となっている$'\t'はタブ文字として展開されます。この展開はbashの機能により行われます[注7]。

▼図4-1　tcpdumpを使ってパケットを監視する

```
$ sudo tcpdump -x -c 1 icmp 2>/dev/null | grep $'\t'
```

注7) bashには、$'文字列'という表現を特殊文字に展開する「ANSI-C Quoting」という機能があります。GNUのWebサイト(https://www.gnu.org/software/bash/manual/html_node/ANSI_002dC-Quoting.html)に説明があります。

図4-1の端末で監視をしつつ、もう1つの端末でpingを動作させましょう(図4-2)。こちらにも-c 1が指定されていますが、pingはこれにより、1つパケットを送信し、それに対するレスポンスを受けたら動作を終了します。

図4-2のコマンドを実行すると、図4-1の端末に図4-3のような結果が出力されます。これがICMP Echo Requestの内容を16進数で表記したデータとなります。

解答(小問2)

図4-3の出力から、可変長のデータ部分のみを取り出すことができれば、小問2の答えとなります。ここであらためて表4-1のICMP Echo Requestの構造を見てみましょう。ICMP Echo Requestのパケットは、0x08と0x00が出現することがわかっています。よって0800という表記が最初にパケットの先頭に現れ、そこから6バイト分はチェックサムなどに使われるはずです。

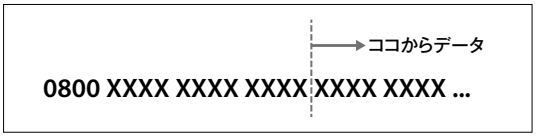
▼図4-2　pingで1つだけパケットを送る

```
$ ping -c 1 gihyo.jp
```

▼図4-3　小問1の解答(ICMP Echo Requestの内容)

```
各行の文頭にはタブ文字が含まれる
  0x0000:  4500 0054 186d 4000 4001 ebf5 ac11 0002
  0x0010:  6814 221f 0800 a0e4 0563 0001 84e4 a259
  0x0020:  0000 0000 60a6 0b00 0000 0000 1011 1213
  0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
  0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
  0x0050:  3435 3637
```

▼図4-4　ICMPパケットの可変長データ



▼図4-5　小問2の解答(ICMP Echo Requestから可変長データ部分のみを抽出)

```
$ sudo tcpdump -x -c 1 icmp 2>/dev/null | grep $'\t' | ↵
sed 's/.*://' | xargs | sed -r 's/^.*0800 (.{4} ){3}//'
84e4 a259 0000 0000 60a6 0b00 0000 0000 1011 1213 1415 ↵
1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 2627 2829 2a2b ↵
2c2d 2e2f 3031 3233 3435 3637
```

それ以降のデータが、ICMPパケットの可変長データと判断して良さそうです(図4-4)。

なお、0800よりも前にあるデータはIPヘッダです。送信元と送信先のIPアドレスのような通信に関する情報が含まれていますが、今回は不要なので削ってしまいます。

図4-3の結果から、データ部分のみを抽出する解答が図4-5です。ここではまず、sedで「0x数字4桁:」のデータの出現位置を表す表記を削除するため、文頭からコロン(:)までを削除しています。xargsコマンドを単体で使うと改行を除いてくれるので、これを利用すると2バイトごとに空白区切りの文字列になります。

最後にsedの-rオプションで拡張正規表現を有効にし、「文頭から0800まで & そこから6バイト分」を削ります。すると図4-5の結果が得られます。これが小問2の解答となります。

補足

問題自体はこれで解けましたが、せっかくなので抽出した可変長部分の内容を読み解いてみましょう。最初の16バイトはパケットを送信した時刻のUNIX時刻とマイクロ秒になります(図4-6)。詳しい説明は割愛しますが、リトルエンディアンというルールでバイト列が並んでいます。これは順番を並べ替えて10進数に変換することができます。図4-7は、dateコマンドで、パケットが送信された時刻を確認する例です。興味のある方は試してみてください。

それ以降のバイト列には、16進数で0x10～0x37のレンジで表される、データのかさ増しのための適当なデータが入っています。

別解(小問2)

図4-8は田代さんからいただいた別解です。「可変長データ」以外のバイト

▼図4-6　可変長データの最初の16バイト

```
84e4 a259 0000 0000  ←UNIX時刻
60a6 0b00 0000 0000  ←マイクロ秒
```

列の長さは固定長であることを利用し、固定のバイト数をcutで抽出するアプローチを取っています。

Ubuntuのpingは、デフォルトでは可変長データ部分が56バイトのパケットを送出します。これにIPヘッダ(20バイト)とICMPヘッダ(8バイト)が付いて、パケット全体としては合計84バイトになります。可変長データ部分を取り出すには、IPヘッダとICMPヘッダの合計28バイトを先頭から取り除けば良いです。

図4-8ではcutコマンドの-cオプションに57-という数字とハイフンで終わる引数が指定されています。これにより「文頭から57バイト目の文字」から文末までの文字列を抽出できます。結果的に文頭の28バイト(84 − 28 = 56)を除くことになります。

ちなみに、pingの-sオプションで送信するパケットの大きさを変更できます。この数値を大きくすると、可変長データ部分が大きくなります。その際は「補足」で述べた「かさ増しのための適当なデータ」(0x10〜0x37のレンジで表されていたもの)の範囲がさらに広くなり、0x37以降もインクリメントが続いて0xffまで続きます。0xff以上は1バイト中に収まらないので、値がいったん0x00にリセットされ、再度インクリメントが続いて同じパターンが繰り返されます。

図4-9はpingの送信するパケットを大きくしていき、そのときのパケットの内容を比較したものです。

終わりに

ネットワーク関連のコマンドにはなじみが全然ない読者の方が多いかもしれません。しかし、使い方を覚えると、何かの場面できっと強力な武器になると思います。

たとえば、今回出てきたパケットキャプチャの方法は、使い方を覚えればpingだけでなく、HTTPをはじめ、いろいろな種類のパケットの内容の確認にも使えます。筆者(山田)は実際に解答にあったのと同様の操作が役立ったことがあったので、今回出題してみました。

ネットワーク系の操作は奥が深く、1回の連載で終わらせるのはもったいないと思っています。というわけで次回の連載でもネットワークについての問題を出題する予定です。SD

▼図4-7 時刻情報を読み取る処理

```
84e4 a259 0000 0000 というリトルエンディアンのバイト列を並び替えて処理する
$ echo 00000059a2e484 | tr '[:lower:]' '[:upper:]' | ↵
sed 's/^/obase=10;ibase=16;/' | bc
1503847556  ←パケットの送信日時(UNIX時刻)

UNIX時刻をdateコマンドで読むと日付がわかる
$ date -d @1503847556
2017年  8月 27日 日曜日 15:25:56 UTC
```

▼図4-8　別解(田代)

```
$ sudo tcpdump -x -c 1 icmp 2>/dev/null | grep $'\t' | ↵
awk '{$1="";print}' | tr -d ' \n' | cut -c57-
```

▼図4-9　pingの-sオプションで送信するパケットの大きさを変更

```
デフォルトのpingが送る56バイトのかさ増しデータ。0x10から0x37までが並んでいる
$ ping -c 1 gihyo.jp  ←別端末で実行
$ sudo tcpdump -x -c 1 icmp 2>/dev/null | grep $'\t' | awk '{$1="";print}' | tr -d ' \n' | cut -c89-
101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637

57バイトで指定。末尾に0x38が追加されている
$ ping -c 1 -s 57 gihyo.jp  ←別端末で実行
$ sudo tcpdump -x -c 1 icmp 2>/dev/null | grep $'\t' | awk '{$1="";print}' | tr -d ' \n' | cut -c89-
101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f303132333435363738

300バイトで指定。0x10から0xffまで並んだあと、0x00から順にデータが並んでいる
$ ping -c 1 -s 300 gihyo.jp  ←別端末で実行
$ sudo tcpdump -x -c 1 icmp 2>/dev/null | grep $'\t' | awk '{$1="";print}' | tr -d ' \n' | cut -c89-
101112131415161718191a(中略)feff000102030405060708090(中略)2122232425262728292a2b
```

presented by
Japan Android Group
http://www.
android-group.jp/

コミュニティメンバーが伝える Androidで広がるエンジニアの愉しみ

第20回 Androidエンタープライズの世界

Androidは世界で出荷される約9割のスマートフォンに搭載される標準OSです※。そのため、多くのAndroidアプリが開発され続けており、そして多くのエンジニアが活躍しています。Androidで広がる新しい技術に魅了されたエンジニアが集うコミュニティもあり、そこでは自分が愉しむための技術を見つけては発信しています。その技術の一幕をここで紹介します。

※Gartner Worldwide Smartphone Sales to End Users by Operating System in 3Q16

重村 浩二（しげむら こうじ）
日本Androidの会 中国支部長

Androidエンタープライズとは何か

Androidの最初のベータ版がリリースされてから今年で10年を迎えます。スマホは日常に溶け込み、私たちの生活になくてはならないものとなっています。その利用範囲は趣味や娯楽にとどまらず、業務にも活用できるアプリケーション（以後アプリ）がGoogle Play Storeには公開され、各企業で開発が進められています。そんな状況から、Googleが展開するAndroidもビジネス向けの利用を促進するために、2014年のGoogle I/Oで「Android for Work[注1]」というサービスを発表しました。

Android for Workでは、個人所有の端末を業務でも使うBYOD（Bring Your Own Device）を推進しています。リリース後に改善が続けられてすべてのAndroid端末を仕事で利用できるようになったため、現在は「Android」と統合され、「Andorid for Work」という呼称は使われなくなりました。ビジネス向けの文脈で扱う際には「Androidエンタープライズ」「ビジネス向けAndroid」と呼ばれているので、本稿では前者にならうことにします。

モバイルアプリでビジネス向けというと、あまり馴染みのない読者の方も多いかもしれませんが、世界中に普及した今だからこそ、ビジネスに使われることを目的としたAndroidエンタープライズがどのようなものなのか、本稿で触れてもらえたらと思います。

注1）https://www.android.com/intl/ja_jp/work/

Androidエンタープライズのしくみ

個人所有の端末でBYODを実現するためには、ビジネスで扱うデータと個人のデータを明確に区別し、ビジネス向けの用途で扱うデータは、プライベートのデータと隔離するようなしくみが必要となります。そして、ビジネス向けのデータを扱う領域は企業側でワイプ（データ消去）や使用不可にするなどの管理を行えることが望ましいでしょう。

この実現のために、Androidエンタープライズでは端末内に仕事用プロファイルを作成し、業務で利用するアプリとそれに付随するユーザデータを格納するしくみを提供しています（**図1**）。

Lollipop（Android 5.0）からユーザ管理の概念が導入されましたが、このユーザ管理ではアクティブとなっているアカウントへのプッシュ通知しか通知表示をすることができません。そのため、個人用とビジネス用のアカウント切り替えが面倒でした。

Androidエンタープライズでは個人アカウント内で個人用プロファイルと仕事用プロファイルとで領域が分けて管理され、同時に実行が行

▼図1　Androidエンタープライズでの領域管理

われているため、個人用アプリ向けとビジネス用アプリ向けのプッシュ通知を同時に受け取れるメリットがあります。

個人用アプリとビジネス用アプリを区別するUI

1つのアカウント内で2つのプロファイルが同時に実行できる状態だと、個人向けとビジネス向けの区別を付けるUIが必要となります。Androidエンタープライズでは**図2**にあるように、ビジネス用アプリの区別を付けるためのマークがアイコン右下に表示され、ビジネス用アプリ実行中にはステータスバー上にも同じマークが表示されることで、ユーザが間違わないようなしくみが提供されています。

◀図2
ビジネス用アプリを示すアイコン

Androidエンタープライズを試すには

Androidエンタープライズを試すには、モバイル管理製品の設定と、各Android端末の設定が必要となります。

モバイル管理製品の設定

モバイル管理製品として、G SuiteなどGoogleが提供するサービスを通して管理コンソールから設定するか、もしくはサードパーティ製のモバイル管理製品(SAPやIBM、Citrixなど)を導入し、Androidエンタープライズを利用できるように設定する必要があります。この設定については、各社の製品によって設定方法が異なるため、各社のマニュアルを参照して設定を行ってください。

各Android端末の設定

G Suiteで試す場合は、Google Play Storeから端末管理ポリシーをインストールし、仕事用プロファイルの作成を行い、個人用プロファイルと分離する必要があります[注2]。

仕事用プロファイルが作成されると、先ほどの**図1**にあるように、仕事用プロファイルの領域が用意されます。そのあとは、管理者から許可されたアプリをビジネス向けGoogle Play Store[注3]よりインストールして環境を構築することになります。

Androidエンタープライズに対応したアプリを作成するには

Androidエンタープライズに対応したアプリを開発するには、仕事用プロファイル環境にアプリを適用していく必要があります。公式のAndroid in the Enterprise[注4]を見ながら、情報

注2）https://play.google.com/store/apps/details?id=com.google.android.apps.enterprise.dmagent
注3）https://play.google.com/work?hl=ja
注4）https://developer.android.com/work/index.html

を収集してください。

adb（またはAndroid Studio）からのインストールであれば個人用プロファイル向けと同時にインストールされて利用できるようになりますが、いくつか開発するうえで注意点があります。最新の情報は脚注4のサイトを参考にしてもらいつつ、筆者が実際に業務で実施した対応について紹介したいと思います。

startActivityの呼び出しでクラッシュする

Androidの特徴的な機能であるIntentを用いてstartActivityを実行すると、クラッシュが発生する可能性があります。仕事用プロファイルでは、基本的に情報システム部門の管理者が許可したアプリしかインストールすることができません。そのため、場合によってはAndroid端末であればプリインストールされている“ブラウザ”アプリや“Gmail”アプリなどがない可能性があります。すると、暗黙的Intentをstart Activityメソッドで呼び出したとき、Intentに指定した対象のアプリが見つからない可能性があります。

startActivityメソッド呼び出し時にアプリが見つからない場合は、**リスト1**にあるようにActivityNotFoundExceptionが返ってくるので、try-catch文で例外処理を行ってやる方法が、取れる対策の1つめとなります。

しかし、理想としてはそもそも機能が利用できないのであれば、ユーザに機能のアクションが実行できないように、startActivity呼び出し前に無効化することが望ましいでしょう。

リスト2にあるように、Intent#resolveActivityメソッドでIntentの対象Activityが存在するかどうかを事前にチェックすることができるため、startActivityメソッドを呼び出す前に、機能の呼び出しを無効化しておくことをお勧めします。

仕事用プロファイルの動作確認

Androidエンタープライズを本番運用する際には上記の端末管理ポリシーを用いていくことになりますが、アプリの開発を行う際にはさまざまな状態を検証するために、開発用の環境構築が必要になります。Googleは開発用の環境構築のために、「Test DPC」というアプリをGoogle Play Store、GitHubに公開しています[注5]。こちらをもとにインストールして環境構築することで、アプリ上から基本的なシステムアプリなどのインストールや有効化・無効化を制御することが可能です。

注5）（Google Play Store）https://play.google.com/store/apps/details?id=com.afwsamples.testdpc
（GitHub）https://github.com/googlesamples/android-testdpc

▼リスト1　ActivityNotFoundExceptionをtry-catchで処理する例

```
try {
  Intent intent = new Intent(Intent.ACTION_VIEW, "http://www.gihyo.co.jp");
  startActivity(intent);
} catch(ActivityNotFoundException ignored) {
  // 例外処理
}
```

▼リスト2　resolveActivityメソッドを利用した機能無効化の例

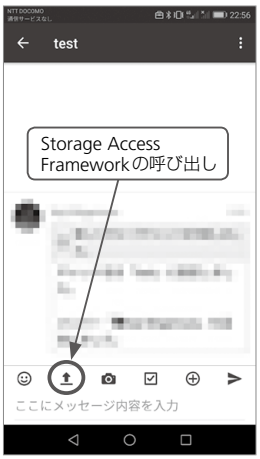
```
Intent intent = new Intent(Intent.ACTION_VIEW, "http://www.gihyo.co.jp");
if (intent.resolveActivity(getPackageManager()) == null) {
  // 機能の無効化を行う（例：ボタンの無効化）
  button.setEnabled(false);
}
```

個人用プロファイルのデータを参照するには

ビジネス用アプリを利用していると、プライベートで撮影した写真や、ダウンロードしたデータにアクセスしたいケースがあります。独自の実装でファイル管理のピッカーなど用意していると、仕事用プロファイル内のデータにしかアクセスできず、少々不便な状況が発生してしまいます。そんな要件に対応できるように、仕事用プロファイルから個人用プロファイルのデータにアクセスする手段が提供されています。

これを実現するには、アプリを開発する際にStorage Access Frameworkを通してデータにアクセスするように作るのがお勧めです。

Storage Access Frameworkを使ってデータにアクセスする仕事用プロファイルのアプリを作成すれば、**図3**のアイコンから遷移する**図4**のメニュー内に「プライベートアプリ」という項目が追加できます。このようにメニューからのタップで、個人用プロファイルの領域にあるデータにアクセスできるようにすると良いでしょう。



◀**図3 Storage Access Frameworkを用いたビジネス用アプリ例:手順1**

◀**図4 Storage Access Frameworkを用いたビジネス用アプリ例:手順2**

さらに情報を得るために

Androidエンタープライズに関する情報を日本語サイトで取り扱っているところはほとんど見受けられません。基本的には公式サイトやStack Overflowのような海外のコミュニティサイトから情報を仕入れることが中心となるでしょう。米Googleが運営しているコミュニティサイトもあります[注6]。

まとめ

ビジネス用アプリは世に出ているアプリの中ではおそらく少数派に入ることでしょう。しかし、世界に個人向けとしてAndroid端末が普及した今なら、ビジネス向けとしてのニーズも高まっていく可能性はあるのではないでしょうか？今回紹介したAndroidエンタープライズはBYODを実現するためにGoogleがAndroidの標準機能として追加した機能です。Android for Workが発表されてからあとにGoogleが発表したOS(Marshmallow(6.0)、Nougat(7.0)、Oreo(8.0))では、毎回Androidエンタープライズに関する修正が行われてきています。それだけGoogleも力を入れている領域であると筆者は考えています。

今後のアプリ開発で上記の情報が読者の役に立てることを願っています。SD

注6) https://www.android.com/enterprise/
以前はAndroid DevHubというコミュニティへの登録ができたのですが、本稿制作中にサイトが更新され、現状のコミュニティサイトに登録申請のページは見当たりませんでした。

一歩進んだ使い方のためのイロハ

Vimの細道

第23回

foldでテキストを折り畳む（後編）

mattn
twitter:@mattn_jp

Vimの折り畳み機能「fold」について、前後編で解説します。後編では折り畳みを行う関数を自分で作り、タブ文字でインデントされたテキストファイルをツリー状に折り畳みます。

前回のおさらい

前回（本誌2017年10月号）は折り畳みの基本を紹介しました。折り畳みを使うことで、構造化されたテキストに一手間加えるだけで、単なるテキストファイルが抜群に扱いやすくなります。前号をお読みになった方も再度読み返しながら、fold機能の使い道を探ってみてください。

今回は、前回に紹介しきれなかったfoldexprという機能を紹介したいと思います。Vimのfold機能は、Vimの動作を制御する専用の言語「Vim script」によってさらに拡張できるようになっています。基本機能だけでは実現できない、いろいろな折り畳み方法を独自に定義できるようになります。筆者が知る限り、ほかのテキストエディタで折り畳みの拡張機能が提供されているものはほとんどありません。

foldexprのしくみ

foldexprを有効にするには、次のようにfoldmethodオプションを設定します。

```
setlocal foldmethod=expr
```

これにより、foldexprオプションの値が読み込まれるようになります。

foldexprオプションの値は式の形式で記述します。その式が評価され、結果の値しだいで折り畳みの挙動が変わります。またfoldexprオプションの式の中（もしくはそこから呼ばれる関数の中）では、v:lnumという変数が有効になります。Vimは対象の行が折り畳まれるべきかを、画面の上部から下部に向かって行ごとに確認していきますが、その対象の行がv:lnumで示されます。そして、foldexprで示される式の評価結果により、「折り畳まれるかどうか」を判定します。

まずは簡単な例を見てみましょう。

```
見出し1
    これは
    本文1です
見出し2
    これは
    本文2です
```

※空白部分はタブ文字（以降同）

まずはこのテキストをexample1.txtとして保存してください。次に新しくバッファを開き、**リスト1**のソースコードを入力し、myfold.vimとして保存してください。ここでは、foldexprオプションにMyFoldLevelという関数の呼び出

▼リスト1　myfold.vim

```
function! MyFoldLevel(lnum)
  return getline(a:lnum)[0]=="\t"
endfunction

setlocal foldmethod=expr
setlocal foldexpr=MyFoldLevel(v:lnum)
```

しを、引数v:lnumで与えています。この式が各行ごとに呼び出されます。ではMyFoldLevelの中身を見ていきます。この関数では条件式の結果をreturnで返却しています。getline()関数は引数で与えられた行の文字列を返す関数ですので、getline(a:lnum)は、現在折り畳みの判断が行われている行の文字列となります[注1]。続けて[0]で添え字の0番目を参照しているので、getline(a:lnum)という式全体は行の一番始めの文字を表しています。そして== "\t"で、この式とタブ文字との比較を行っています。つまりこの関数MyFoldLevelは、引数で示される行の先頭がタブ文字であれば1を、そうでないならば0を返すという動作になります。

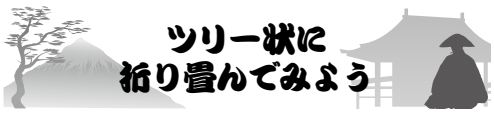
実際に動作を見てみましょう。example1.txtのバッファを開いた状態で、

```
:source myfold.vim
```

と実行します。行の先頭がタブ文字で始まっている行が折り畳まれ、次のようになります。

```
見出し1
+-  2 lines: これは---------------
見出し2
+-  2 lines: これは---------------
```

なお折り畳みのレベル0の行、つまり「見出し」で始まっている行は折り畳まれません。また現状のMyFoldLevelでは、同じ折り畳みレベルの行が続いていない場合は折り畳まれません。このしくみについては次節で説明します。



折り畳みレベルをカスタマイズ

foldexprは、評価結果の値によって**表1**の動作を行います。前述のMyFoldLevelは0と1しか返しませんでした。これを少し修正して、タブ文字でインデントされたテキストがツリー状に折り畳まれる設定を作ってみましょう。この実現には、行頭に続くタブ文字の数だけ折り畳みレベルが大きくなる関数を作れば良いですね。

前回説明したとおり、Vimの折り畳みは「同じ折り畳みレベル以上の行が並んでいる」場合に折り畳まれるという動作になっています。

```
ルート
    アイテム1
    アイテム2
        アイテム3
        アイテム4
    アイテム5
```

のようなタブ文字でインデントされたテキストでは、「アイテム1」は隣り合う同じ折り畳みレベルの「アイテム2」と一緒にまとめられてしまい、ツリー状にはなりません。「アイテム1」と「アイテム5」は折りたたまれず、「アイテム2」「アイテム3」「アイテム4」が折り畳まれるというのが目標の形です。

そこで使用するのが、**表1**でも示した>です。foldexprに返してほしい値は次のようになります。

```
>1
    >2
    >2
        >3
        >3
    >2
```

よって、各行のインデントレベルに>を付けたものを返せば良いことになります(**リスト2**)。matchstr()関数は、第1引数の文字列のうち

▼表1　foldexprの動作

値	意味
0	折り畳まない
1、2、…	このレベルで折り畳む
-1	折り畳みを定義せず、前後の行のレベル(低いほう)に従う
=	直前の行と同じレベルで折り畳む
a1、a2、…	直前の行のレベルに数字分足したレベルで折り畳む
s1、s2、…	直前の行のレベルから数字分引いたレベルで折り畳む
>1、>2、…	このレベルから折り畳みを始める
<1、<2、…	このレベルで折り畳みを終える

注1) v:～はVimの組込み変数。a:～はfunction内のみで使える変数。

▼リスト2　myfold.vimのMyFoldLevelを修正

```
function! MyFoldLevel(lnum)
  return '>' . (len(matchstr(getline(a:lnum),'^\t*'))+1)
endfunction
```

第2引数の正規表現でマッチしている部分の文字列を返します。ここではタブ文字にマッチさせ、len()関数でタブ文字の個数を返しています。「ルート」にはインデント幅がありませんが、全体の折り畳みレベルを+1することで「ルート」も折り畳まれるようにしています。

折り畳みの見栄えをカスタマイズ

次は折り畳みの見栄えを変えてみます。通常、折り畳まれた行は次のように表示されます。

```
+- 2 lines: これは----------
```

折り畳まれた際のテキストは、foldtextというオプションで変更できます。これもfold exprオプションと同様に式を設定します。

```
setlocal foldtext=v:folddashes.'折り畳み'
```

このように設定すると、折り畳まれた行が次のように表示されます。

```
--折り畳み----------
```

このfoldtextの式として、自作関数を渡すことができます。

foldtextで使える関数や変数を**表2**、**表3**に示します。これらを踏まえ、タブ文字でインデントされたテキストをツリー状に折り畳んでフォルダのように見せる**リスト3**のうち、MyFoldTextの説明をします。

getline()で得た折り畳み開始行をmatchlist()という関数でタブ文字とそれ以外に分解しています。matchlist()の使い方は:help matchlist()を参照してください。続いてタブ文字の幅(見た目の幅)をstrdisplaywidth()という関数で取得し、その幅に合わせた空白を作っています。タブ文字はtabstopオプションで見た目が変わってしまいますから、どのユーザも同じ見た目にするためにはこのようにします。また、折り畳まれたテキストの後ろの部分にも「-」が付くので、画面幅いっぱいに空白を足しています。これにより、折り畳まれた行はフォルダのように扱われ、カラースキームが適用されていれば、通常のテキストとは異なる色で表示されます(**図1**)。

折り畳みのその他の適用方法

こういったfoldexprやfoldtextの設定を適用するには、先に説明した:sourceコマンドのほかに次の方法があります。

・モードラインで指定
・ファイルタイプでスクリプトを実行

モードラインによる指定は1行で書かなければならず、しかも使えない文字をエスケープしながら記述する必要があってとても困難です。このような複雑な折り畳みを行うには、拡張子.vimのファイルに処

▼表2　foldtextで使える関数

関数	説明
foldclosed({lnum})	行{lnum}が閉じられている場合は折り畳み開始行を返す
foldclosedend({lnum})	行{lnum}が閉じられている場合は折り畳み開始行を返す
foldlevel({lnum})	行{lnum}の折り畳みレベルを返す
foldtext()	式を評価した際の折り畳み表示内容を返す
foldtextresult({lnum})	折り畳まれた行{lnum}のテキストを返す

※{lnum}は対象の行番号を指定。省略した場合は現在の対象行が入る。

▼表3　foldtextで使える変数

変数	説明
v:folddashes	折り畳まれたテキストの前に付く「-」文字
v:foldlevel	折り畳みレベル
v:foldstart	折り畳みの開始行
v:foldend	折り畳みの終了行

▼リスト3　ツリー状に折り畳まれた行をフォルダのように見せる

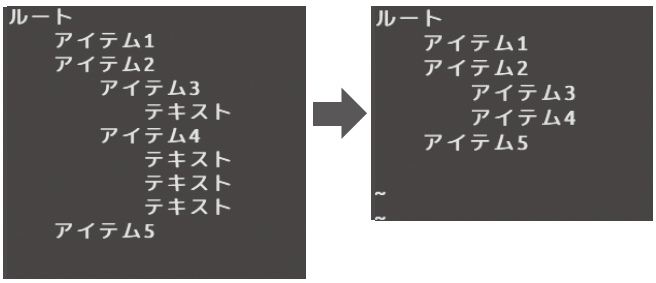
```
function! MyFoldText(foldstart)
  let [_, tabs, text; _] = matchlist(getline(a:foldstart), '^\(\t*\)\(.*\)')
  let width = strdisplaywidth(tabs)
  return repeat(' ', width) . text . repeat(' ', winwidth(''))
endfunction

function! MyFoldLevel(lnum)
  return '>' . (len(matchstr(getline(a:lnum),'^\t*'))+1)
endfunction

setlocal foldmethod=expr
setlocal foldexpr=MyFoldLevel(v:lnum)
setlocal foldtext=MyFoldText(v:foldstart)
```

理を記述し、「~/.vim/ftplugin/<ファイルタイプ>/<カテゴリ>.vim」というファイル名で配置します。たとえばテキストファイルであれば、「~/.vim/ftplugin/text/myfold.vim」、メモを記述するための自作ファイルタイプmemoであれば「~/.vim/ftplugin/memo/myfold.vim」というファイル名になります。

◆ ◆ ◆

前回と今回で、折り畳みの基本とVim scriptを使った独自の折り畳みルールを作る方法を説明しました。タブ文字だけでなく空白にも対応させたり、Markdownの見出しを折り畳む関数を作ったりするのも、それほど難しくないと思います。ぜひいろいろな折り畳み関数にチャレンジしてみてください。SD

▼図1　リスト3による折り畳み



Vim自報

Vimと日本語入力

Vimの長い開発の歴史の中で、我々日本人開発者がどうしても解決できない問題がありました。仮入力の問題とiminsertオプションのデフォルト値の問題です。前者は現状のオンザスポット入力の独自実装がバギー（バグだらけ）であること、後者はiminsertのデフォルト値が原因でインサートモードに入るたびにIMEをアクティブにしてしまうという問題です。ほかの国のIMEと協調した結果、このような問題のある現状の動作になってしまっていました。

長い間この問題が解決できず「Vimで日本語を入力するのは難しい」と言われ続けてきましたが、patch 8.0.1026により、仮入力方式がオーバーザスポット（仮入力を独自実装に頼らずIME自身に任せる方式）に、そしてpatch 8.0.1114でiminsertのデフォルト値が「0」に、つまりインサートモードに入っても勝手にIMEがアクティブにならないように修正されました。日本人Vimmerが長い間苦しんできた問題でしたので、一日本人としてとても喜んでいます。ぜひ日本語が入力しやすくなった最新版のVimを試してみてください。この話の詳細は筆者のブログ[注A]を参照してください。

注A）URL https://mattn.kaoriya.net/software/vim/20170905113330.htm

書いて覚える Swift 入門

第31回　収穫の秋にWatchすべきなのは？

Author 小飼弾（こがい だん）　twitter @dankogai

人類にはまだ早すぎるApple Watch

もはや恒例となった「Apple秋の収穫祭」、今年2017年9月12日[注1]のイベントで読者のみなさんが最も注目したのはどの製品でしたか？　全面ガラス張りのSteve Jobs Theater？　全面ディスプレイのiPhone X？　まさか4K HDR対応したApple TV？

筆者が最も注目したのは、フォームファクターが一番変化していない製品。そう、Apple Watchです（**写真1**）。実質iPhone 7sなiPhone 8でさえ、背面がアルミニウムからガラスに変わったというのに、Apple Watchときたらそれすらない。フォームファクターが変わる都度、iPhoneはケースやバンパーといったアクセサリーも変化しますが、Apple Watchのバンドは初代のものがそのままSeries 3でも使えますし、最新のものが初代でも使えます。そうでないと困ります。何しろ今回「廃盤」になったのは初代ではなくSeries 2だったのですから。

しかし、今回のSeries 3のCellular Modelは、ほかのApple WatchとはiPhoneとiPod Touch以上の違いがあります。Series 3 CellularはiPhoneがなくてもオンラインなのです。これは何を意味するか？

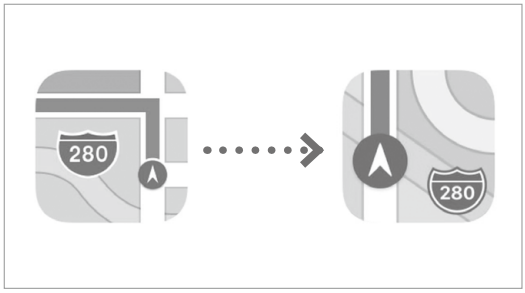
真の全裸コンピューティングが実現するのです！　実際に露天風呂でSiriにたずねてみたら、このように答えてくれるのです（**図2**）。

Wi-FiからCellularへの切り替えはとてもスムーズ。iPhoneを見失うと✕が表示されますが（**図3**）、数秒でセルラー接続に切り替わります（**図4**）。

全裸コンピューティングは人類には早すぎに

▼写真1　Apple Watch Series 3

▼図1　マップのアイコンもApple Parkに引っ越し



注1）URL https://www.apple.com/jp/apple-events/september-2017/

▼図2　siriに「ここはどこ」と問うと……

▼図3　iPhoneとの接続が切れたときの表示

▼図4　わずか数秒でセルラー接続

▼図5　iPhoneなしでも

▼図6　メッセージ送信

▼図7　もちろん電話も!

しても、脱スマホコンピューティングなら筆者を含め少なからぬ人が待ちわびていたのではないのでしょうか。ジョギングやワークアウトのとき、あるいは温泉街を浴衣で練り歩くとき……スマホですらかさばり過ぎて大き過ぎると感じるシーンは決して少なくありません。そんなときでもメッセージは出せますし(**図5、図6**)……。

電話もできますし(**図7**)……。

支払いだって問題ありません(**図8**)。

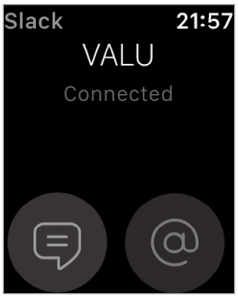
まあ、どこでもオンラインってことは……、どこでも仕事が追いかけてくるということ(**図9**)でもありはするのですが。

もちろんApple WatchがiPhoneの代わりを100%つとめるというのには無理があります。何と言っても世界で一番使われているとAppleが豪語するカメラがありませんし、セルラー契約の1つをとっても、初期設定にはiPhoneが必須です。しかし今回の「どこでもオンライン」と同じぐらい感心したのは、複数のデバイスをどう組み合わせるか。たとえばAirPodsは、iPhoneがあるときにはiPhoneの、Apple Watchしかないときには Apple Watchの(イヤ｜マイクロ)フォンに全自動で切り替わります(**図10**)。

だとしたら、Apple Watch、いやユーザのコンピューティング環境が全自動で切り替えられる道理がないわけがありません。デバイス自身が今、自分が使われていることを判定し、自動的に切り替わる。音声に関してはAirPodsですでにそれが実現しています。映像だってそうならない道理はありません。すでにApple TVはiPhoneやiPadやMacのディスプレイとしても機能していますが、ワン(クリック｜タップ)必要だという点でAirPodsに及びません。それがAirPodsなみにシームレスになる

▼図8　Apple Payも OK!

▼図9　Slackの通知も OK!



▼図10　自動的にAirPodsも接続される

のだとしたら？

次はARディスプレイ+カメラということになるでしょう。Google GlassにせよHololensにせよはたまた PS VRにせよ、VR/MRがマスマーケットに来ていない最大の理由は、Steve Jobsが言っていたところの“Connecting the dots”に成功していないこと。今回iOS 11にARKitが搭載され、iPhoneのカメラとCPUがARに十分な性能があることを示したAppleが、いまだヘッドアップディスプレイを欠いている状況を放置するとは筆者には考えられません。

そのときの「ハブ」は、いまだiPhoneでしょうか？　それともApple Watchでしょうか？

Apple Watch Series 3で後者の可能性が見えてきました。iPhone以上に身近になりうるデバイスは今のところそれだけなのですから。

どう転んでもSwift

しかし未来がどちらに進んでも、これだけは変わらないだろうということが1つあります。それが開発言語。iOSもwatchOSもtvOSも、メインの開発言語はSwift。Linuxへの移植とSwift Playgrounds for iPadでSwiftはXcodeの専売特許ではなくなって久しいのですが、Swiftが「言語ハブ」であり続けることだけは確かだと弾言しておきます(断言までするのはちょっとこわい)。

とはいえ、2017年秋現在の主力開発環境はXcode on Mac。iOS 11と同じく日本時間9月20日に9が正式リリースされています。Swiftも4に上がりました。主な新機能および変更点は第28回(本誌2017年8月号)のWWDC 2017特集で紹介したとおりですが、同記事で紹介しきれなかった分もあらためて紹介します。

総称型のsubscript

subscriptで総称型が扱えるようになりました。たとえばこんな書き方ができます。

```
struct JSON {
    var value: [String:Any]
    init(_ value: [String:Any]) {
        self.value = value
    }
    subscript<T>(k: String) -> T? {
        return value[k] as? T
    }
}

let json = JSON([
    "bool":true,
    "number":42.195,
    "string":"Swift",
    ]
)

var bool:Bool?     = json["bool"]
var number:Double? = json["number"]
var string:String? = json["string"]
```

最終的な型はコンパイル時に決まる必要があるため、左辺は型を明示しないとコンパイル時エラーになります。

Dictionaryのデフォルト値

Dictionaryの要素はOptionalなので、dict[key]でkeyが存在しない場合の値はnilですが、dict[key, default:val]と書くことで、nilの代わりにvalを返すようになりました。

これは統計を取るときなどにとても便利で、今まで、

```
var count = [String:Int]()

for k in words {
    if count[k] == nil {
        count[k] = 0
    } else {
        count[k]! += 1
    }
}
```

のように初期化の有無で必要だった条件分岐が、

```
for k in words {
    count[k, default:0] += 1
}
```

のように不要になります。

"""複数行"""引用符

"""で複数行に渡る文字列リテラルが書けるようになること自体は同記事でも軽くふれたのですが、この機能が最も威力を発揮するのは、Swift内でソースコードを表記するときでしょう。

たとえばSwift 3でSwiftのソースコードを書くには、

```
let howmany = 42
let fizzBuzzSwift = [
    "#!/usr/local/bin/swift",
    "extension Int {",
    "    var fizzbuzz:String {",
    "        let fb = (\"Fizz\", \"Buzz\")",
    "        switch (self % 3, self % 5) {",
    "        case (0, 0) : return fb.0 + fb.1",
    "        case (0, _) : return fb.0",
    "        case (_, 0) : return fb.1",
    "        default     : return \"\\(self)\"",
    "        }",
    "    }",
    "}",
    "_ = (1...\(howmany)).map{ print($0.fizzbuzz) }"
].joined(separator: "\n")
```

のように「分かち書き」する必要がありましたが、

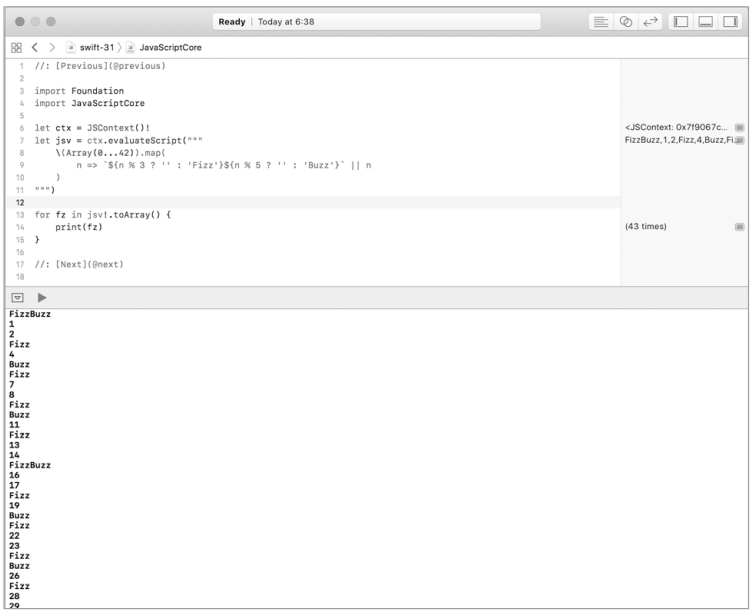
```
let howmany = 42
let fizzBuzzSwift = """
#!/usr/local/bin/swift
extension Int {
    var fizzbuzz:String {
        let fb = ("Fizz", "Buzz")
        switch (self % 3, self % 5) {
        case (0, 0) : return fb.0 + fb.1
        case (0, _) : return fb.0
        case (_, 0) : return fb.1
        default     : return "\\(self)"
        }
    }
}
_ = (1...\(howmany)).map{ print($0.fizzbuzz)
}
"""
```

とずいぶんとスッキリ書けるようになります。もう一例として、Swift Playgroundでちょっと変わったFizzBuzzをこさえてみました。見てのとおり、"""でくくられているJavaScriptを実行しています(**図11**)。

"""も変数展開(interpolate)する点は注意してください。この例では\(howmany)が変数展開されていますし、\\(self)は変数展開されないよう\をエスケープしています。

Swift 4でもなお変数展開なしの引用符(PerlやRubyでは')はサポートされていません。

▼図11　FizzBuzzの実行例



次回予告

そろそろ紙幅も尽きるので今回はここまで。次回はSwift 4で大いに整理された整数型のプロトコルをおもに取り上げます。SD

RDBアンチパターン

PostgreSQLとMySQLの失敗と対策

Author 曽根 壮大（そね たけとも） ㈱はてな Twitter @soudai1025

第1回 隠された状態

本連載では、開発の現場で発生しやすいリレーショナルデータベース（RDB）全般の問題をRDBアンチパターンとして紹介していきます。今回のアンチパターンの主人公は、DB設計者の作ったテーブルによって苦しむアプリ開発者と運用者です。

隠された状態

前回（本誌2017年10月号）は「ソートの依存」と題して、RDBが苦手なソートについて説明しました。今回は「隠された状態」という題名で、DB設計についてのアンチパターンについてお話します。

RDBに状態を持たせるのはたいへん危険です。第2回「失われた事実」（同年6月号）や第5回「フラグの闇」（同年9月号）でも、RDBに事実のみを保存することの大切さと難しさをお話しました。なぜ複数回に渡って強調してきたかというと、隠された状態には多くの罠が潜んでいるからです。今回はその理由と、それによって引き起こされる問題について説明します。

事の始まり

DB設計者：すごいことを思いついたぞ。IDの先頭が9なら管理者、1なら一般ユーザとすれば、is_adminカラム[注1]がいらないぞ！　しかもこれなら、ユーザの権限が増えたときにいちいちroleを示すカラムを追加しなくても、2〜8までの値で表現できるし便利だ！

——時間は経ち、実装フェーズへ。

アプリ開発者：このテーブル、user_id見ないと管理者なのか一般ユーザなのかわかんないじゃん。ふぅ……、意味を含んだIDを使うと結局こういうコード（**リスト1**）を書くハメになるんだよなぁ。これならカラムを分けてるほうが、$user->is_adminを見るだけだからコードもわかりやすいし楽だよ。

——それから3日後。

PM：管理する権限、一般ユーザと管理者だけだったけど、「閲覧のみユーザ」も追加すること

▼リスト1　user_idを見てユーザを判別

```
if($this->is_admin($user->user_id))
{
  // 管理者用の処理へ
}

// user_idの先頭が9だった場合は管理者
function is_admin($user_id)
{
  $role_id = mb_substr($user_id, 0, 1);
  return ($role_id == 9) ;
}
```

注1）ユーザが管理者かどうかを判別するためのフラグを格納するカラム。

▼リスト2　リスト1に「閲覧のみユーザ」用の処理を追加

```
$role_id = $this->get_role_id($user->user_id)

if (role_id == 9) {
    // 管理者用の処理へ
} elseif (role_id == 8) {
    // 閲覧のみユーザの処理へ
}

function get_role_id($user_id)
{
  return $role_id = mb_substr($user_id, 0, 1);
}
```

になったから。

アプリ開発者：え？　閲覧のみユーザはどうやって見分けるんですか？

PM：user_idの先頭が8のユーザを閲覧のみユーザとする。

アプリ開発者：はい……（書き直しになるぞ）。

リスト2で運用を開始し、それから3年後。

社長：今の会員数と管理者の人数って何人？

運用者：すぐには出せません。明日でも良いですか？

社長：え？　count()で会員数を数えるだけでしょ。

運用者：それが、user_idから権限を判断してるので検索用のSQL関数が必要です。つまりはテーブルフルスキャンになるので、時間がかかるんです……。

社長：なんだって……。

今回のアンチパターン

今回のアンチパターンにはデータの保存方法に問題があります。今回の例ではわかりやすく、user_idの先頭文字のみでしたが、次のような場合はどうでしょう？

・県番号（2桁）部門ID（3桁）支店ID（4桁）
（例：330010012）

このIDの例は後にも出てきますが、このような場合には県別や部門別のGROUP BYができないため、集計のためにsubstr(id, 3, 3)などで整形してから検索する必要があります。また、第4回「効かないINDEX」（2017年8月号）でもお話しましたが、基本的に関数を利用したWHEREやGROUP BYは、関数を実行してみるまでRDBMS側では結果がわからないので、INDEXを利用できません。そのため開発面の工数も上がりますし、運用面のコストも格段に上がります。

意味を含んだID

このように、意味を持たせたIDを「意味を含んだID」、あるいは「論理ID」「スマートカラム」と言ったりします。IDとはidentificationのことですから、識別できる一意の値以上の意味を持たせてはいけません。アンチパターンのように、データにビジネスロジックを持たせたり、複数の意味を持たせたりすると、一見しただけでは本来の状態を読み取れないデータになってしまいます。これが隠された状態です。

意味を持つIDについては、奥野幹也さんのブログ「漢のコンピュータ道」でも取り上げられたテーマですので、ぜひこちらも見てみてください[注2]。

隠された状態はこのほかにもある

ここまでは意味を含んだIDを取り上げて隠された状態の話をしましたが、実は隠された状態はこれだけではありません。RDBの設計の際、汎用性を高める設計を目指した場合にたびたび陥る罠として、次の2つのSQLアンチパターンがあります。

EAV（エンティティ・アトリビュート・バリュー）

複数の目的に使われるカラムを用意する設計

注2）「リレーショナルモデルのドメイン設計についての議論」
URL http://nippondanji.blogspot.jp/2013/12/blog-post_8.html
「IDの設計についてのさらに突っ込んだ議論」
URL http://nippondanji.blogspot.jp/2013/12/id.html

▼図1　EAV

会員テーブル

会員id	名前
1	曽根 壮大
2	曽根 徠楽
3	曽根 煌楽
4	曽根 朔楽

会員情報テーブル

会員id	属性名	値
1	年齢	32
1	特技	格闘ゲーム
1	職業	CRE
2	年齢	9
2	特技	バレエ
2	好きな食べ物	抹茶アイス

会員情報テーブルには情報を柔軟にINSERTできるが、何が入っているかは取り出してみないとわからない

です。たとえば、**図1**の会員情報テーブルのような設計です。この設計では、どのような属性名と値の組み合わせのデータでも保存できる反面、次のようなデータの状態は実際に取り出すまでわかりません。

・その属性名に対する値があるのかないのか
・その属性名があるのかないのか
・属性名と値の組み合わせは正しいのか
・属性名の一覧

また次のような設計上の問題もあります。

・必須属性を設定できない

たとえば、属性名＝「年齢」にはNOT NULL制約を設定して必須属性にしたいところですが、「電話番号」が保存されることもあり、そのときにはNULLが入る場合があります。ですので、ひとくくりに必須属性を設定できません。本来であれば、年齢カラムにNOT NULLを設定するだけで実現できるしくみも、EAVではできないのです。

・データ型を指定できない

たとえば、属性名＝「作成日」の値は当然日付でないとダメですが、「年齢」や「電話番号」も保存され得るため、ひとくくりに日付型を設定できません。加えて、「作成日」＝ yyyy/mm/dd、「更新日」＝ yyyy-mm-ddのように、同じ日付でもフォーマットが違うデータが生まれてしまいます。これにより、範囲検索が難しくなるなどの問題が起きます。

・正規化されていないため外部キー制約（参照整合性制約）を強制できない

たとえば、属性名＝「都道府県」の値には「東京都」と「東京」の両方が保存される可能性がありま

RDB TIPS　EAVの代替案になり得るJSON型

スキーマレスな設計ができることがEAVのメリットです。しかし、多くのデメリットがあることは理解できたかと思います。そこで近年、MySQL 5.7やPostgreSQL 9.3以降のRDBMSではその代替案として、JSON型を採用するケースがあります。JSONを利用することで、Key-Valueな値をそのまま保存できるうえに、Keyでの検索やValueの更新をRDBがサポートしてくれる場合もあります。これにより、EAVを採用するケースはほぼなくなったと言っても良いでしょう。

ただし、JSON型にもデメリットはあります。そのお話は次回のテーマとなっていますのでお楽しみに！

▼図2　Polymorphic Associations

Propertiesテーブル（子）

properties_id	名前	住所	参照先
1	藤原とうふ店	群馬県…	Shop
2	篠原重工	東京都…	Shop
3	曽根 朔楽	広島県…	User
4	曽根 壮大	広島県…	User
5	アナハイム・エレクトロニクス	京都府…	Shop
6	曽根 煌楽	広島県…	User
7	曽根 徠楽	広島県…	User

Propertiesテーブルの参照先によって親テーブルが変わるため、外部キー制約が貼れない

Shopテーブル（親）

shop_id	properties_id	従業員
1	1	2
2	2	50
3	5	100

Userテーブル（親）

user_id	properties_id	年齢
1	3	4
2	4	32
3	6	7
4	7	9

す。本来であれば、正規化して都道府県テーブルを作り、idを指定することで、表記揺れは発生せず、また存在しない都道府県を指定されることもないのですが、EAVではこのような問題も発生します。

・属性名を補わなくてはならない

上記項目と同じく、属性名も表記揺れの影響を受けます。たとえば、属性名が「updated_at」なのに日付が入る可能性もありますし、場合によっては「update_at」のようにタイピングミスが生まれる可能性もあります。

このように、汎用性を高めた反面、RDBが本来持っている多くのメリットを失い、RDBの責務であるデータを守ることが難しくなります。

Polymorphic Associations（ポリモーフィック関連）

子テーブルが複数の親テーブルを持つような設計です。たとえば**図2**のような設計です。この場合、子テーブルの属性によって紐づく親テーブルが変わります。そのため外部キー制約は使えず、アプリ側や運用者としてはJOINする対象がデータを取り出すまでわかりません。結果、親テーブルの両方をJOINしてからNULLの場所によって対象データを判断するような運用が行われるようになり、非効率なクエリが実行されることになります。

またEAVとは違い、テーブルをそれぞれ用意することで必須属性やデータ型を利用できますが、EAVと同じく外部キー制約は使えないために、参照整合性は担保できません。

◆ ◆ ◆

まとめると、EAVは1つのカラムに状態を詰め込んでレコード単位で状態を隠す手法ですが、Polymorphic Associationsはテーブル単位で状態を隠す手法と言えるでしょう。EAVとPolymorphic Associationsについては、本連載でもたびたび紹介している名著『SQLアンチパターン』[注3]で解説されています。使いたくなるケース、その場合の解決策などが細かく書かれていますので、ぜひご一読ください。

複数の目的に使われるテーブル

もっとシンプルに考えると、テーブルそのものに問題があることが考えられます。

今回のアンチパターンでは、ユーザの属性として一般ユーザと管理者を1つのテーブルに格納したことが問題です。データの属性によって入る値が変わるカラムやNULLが入るようなカ

注3）Bill Karwin 著、和田 卓人、和田 省二 監訳、児島 修 訳、2013年発行、オライリー・ジャパン、ISBN＝978-4-87311-589-4。

ラムがある場合は、複数の目的で使われるテーブルになっていないか調査しましょう。

たとえば、管理者テーブルと一般ユーザテーブルに分ければこのような問題は発生せず、アプリ側は参照するテーブルを分けることで、問題を局所化することができます。ドメインとしても分けやすくなるので、モデルとしてシンプルになるでしょう。似たような属性のデータの場合、1つのテーブルに保存してしまいがちですが、パフォーマンスの面を考えても、テーブルを分けたほうが良いケースが圧倒的に多いです。加えて、テーブルを分けることで、「管理者の場合は値が入るが一般ユーザの場合はNULLになる」といったカラムの出現も防ぐことができます。

コードやデータから見えない状態の辛さ

隠された状態は、パフォーマンスの問題も当然ありますが、何よりも運用コストが跳ね上がることが問題です。

たとえば今回の例のように、権限別の集計SQLは難しくなりますし、仕様変更などの影響範囲も大きくなります。EAVやPolymorphic Associationsといった、ほかの隠された状態の例のいずれでも、アプリ側では一見して状態を判断できず、仕様変更があったときにはたいへんです。運用者も、データが大きくなればなるほどたいへんになります。

また、バグが起きて不正なデータが生まれた場合に整理が難しいのも、このアンチパターンの特徴です。アプリ側はコードからデータが判断できないため、バグを生みやすい構造になっています。そのため不正なデータも入りやすく、開発者も運用者も苦しい状態になるのです。

失われた制約

不正なデータが入りやすい問題に関連して、隠された状態を持つ設計では制約が使えなくなる、つまりアプリのバグやオペレーションの際に不正なデータが投入されることを防げない、という問題もあります。

先ほどの説明のとおり、アプリ側で判断できないため不正なデータが投入されやすいです。これを制約で防げれば問題ないのですが、隠された状態の設計ではCHECK制約や外部キー制約が使えないため、RDB側の制約で防ぐことができません。加えて、ドキュメントに正しい仕様が残されていない場合、どの値が何を表すのかが正しく読み取れなくなります。そんなケースでの運用者は、それが正しいデータなのか不正なデータなのかを判断することが難しくなり、最終的にはコードを読み解くことになります。

例を挙げますと、何らかの理由で「IDが7から始まる会員のデータ」を作り、その背景をドキュメント化していない場合、7から始まる会員がどのような権限なのか、コードを読む以外に判断できなくなります。会員IDのような値は多くのケースで主キーとなっていることから、多くのテーブルから外部キー制約として参照され、そのため一度設定されると最後、更新することが難しいのも特徴です。

次のようなことを想像してみてください。もしも権限が1桁では足りず、2桁になったら？　冒頭で見せた「県番号(2桁)部門ID(3桁)支店ID(4桁)」といったIDの場合に部門IDが3桁で足りなくなったら？　もうおわかりでしょう。すべての関連するテーブルに影響を与える大改修になり、もちろんアプリ側も影響を受けます。

このように制約はデータを守り、アプリを守り、そしてシステムを守ることで、運用コストや開発コストを下げてくれているのです。それを失うことは大きなデメリットと言えます。

このアンチパターンのポイント

今回のアンチパターンは「1つのデータ(テーブル・カラム・レコードなど)に複数の属性を与えた」ことで発生しました。何度も言いますが、RDBには事実のみを保存するのが基本です。ですので、そのデータの属性や責務が複数ある場合は、もちろんその保存先を分けていくのが妥

▼図3　意味を含んだIDのリファクタリング

会員テーブル

会員id	名前
900001	曽根 壮大
900002	曽根 徠楽
100003	曽根 煌楽
100004	曽根 朔楽

会員情報テーブル

会員id	名前	権限
1	曽根 壮大	管理者
2	曽根 徠楽	管理者
3	曽根 煌楽	一般ユーザ
4	曽根 朔楽	一般ユーザ

当です。

たとえば、意味を含んだIDの場合は**図3**のように正しく正規化するのが良いでしょう。これで外部キー制約も設定可能ですし、GROUP BYによる検索も容易です。場合によっては権限が増えることもあるでしょう。そのときはINSERTするだけで正しく追加できます。

Polymorphic Associationsの場合は、先ほど紹介したSQLアンチパターンにも記載してありますが、交差テーブルを用意するのが良いです（**図4**）。

また、複数の目的で使われるテーブルの場合やEAVでは、責務を分けて**図5**のようにしましょう。

RDB TIPS　トリガーのメリットとデメリット

隠された状態に近い機能がRDBMSには用意されています。それがトリガー[注A]です。トリガーはアプリや運用者側からは見えませんし、振る舞いが予想できません。それがメリットでもありデメリットでもあります。たとえば、一時的にDELETEされた会員のレコードを、アプリに影響を与えることなく保存したいといった場合にトリガーは有能です。しかし、永続的に影響を与えるような場所でトリガーを使うと、隠された状態と同等の問題を生むことがあります。

設計における隠された状態と違うのは、制約を犠牲にしないという点です。そこで筆者は、アプリ側で機能を実装するのと比較したうえで次のようなメリットがある場合のみ、トリガーの採用を検討しています。

①パフォーマンス的メリット
②アプリ側の実装が大幅に削減できる
③既存のアプリの振る舞いを維持したまま、仕様を変更できる

①の例としては、INSERTされた数をカウントするテーブルを作成する場合などです。②や③の例としては、先ほどのような会員を削除する場合などです。

また、複数のアプリからデータベースが参照されているケースでのデータベースリファクタリングは、アプリの振る舞いを維持したまま行いたいので、トリガーは重宝します。

しかしここに書いたような採用ケースは、「顕著なメリット」がある場合のみの話です。メリットと隠された状態が生むデメリットを比較して悩むようであれば、採用を見送るのが妥当でしょう。筆者も、多少メリットのほうが大きいかも、という程度であれば採用を見送っています。このバランス感覚は、経験に基づく設計力が試される部分でもありますので、ぜひ養っていただければと思います。

注A）テーブルに対するある操作に反応して、別の操作を実行する機能。

▼図4　Polymorphic Associations（図2）のリファクタリング

Propertiesテーブル

properties_id	名前	住所
1	藤原とうふ店	群馬県…
2	篠原重工	東京都…
3	曽根 朔楽	広島県…
4	曽根 壮大	広島県…
5	アナハイム・エレクトロニクス	京都府…
6	曽根 煌楽	広島県…
7	曽根 徠楽	広島県…

交差テーブル

shop_id	properties_id
1	1
2	2
3	5

user_id	properties_id
1	3
2	4
3	6
4	7

Shopテーブル

shop_id	従業員
1	2
2	50
3	100

Userテーブル

user_id	年齢
1	4
2	32
3	7
4	9

▼図5　複数の目的を持つテーブルのリファクタリング

会員テーブル

会員id	名前	権限
1	曽根 壮大	管理者
2	曽根 徠楽	管理者
3	曽根 煌楽	一般ユーザ
4	曽根 朔楽	一般ユーザ

管理者テーブル

会員id	名前
1	曽根 壮大
2	曽根 徠楽

一般ユーザテーブル

会員id	名前
3	曽根 煌楽
4	曽根 朔楽

このように隠された状態は、汎用性を上げることを目的としたものである場合は正しく正規化できます。とくに交差テーブルは、作ることが面倒なため忌避する人もいますが、それは逆効果です。外部キー制約やトランザクションのパフォーマンスの問題が顕在化して初めて対策するのでは遅いですし、もとのテーブル設計はそのままに解決できることが多いからです。

以上のように、このアンチパターンを防ぐコツは次の3つです。

・データに複数の意味を持たせない
・1つのデータの責務を小さくする
・常に状態が見えるようにするために事実のみを保存する

これらを守り、より良い設計を心がけましょう。

次回のRDBアンチパターン

今回のRDBアンチパターンはいかがでしたでしょうか？　隠された状態はアプリ側にも影響が大きいアンチパターンです。一度導入すると、その問題は多くのアプリに波及し、いずれは破綻することでしょう。そうなる前に、早めにリファクタリングをご検討ください。

次回は、今やデファクト・スタンダードとも言えるJSONに関するアンチパターンです。次回の「JSONの甘い罠」もお楽しみに！ SD

50

Debian Developer　やまねひでき　henrich@debian.org

DebConf17レポート（前編）

Debian Hot Topics

DebConf17開催

今回は、カナダ・モントリオールのCollège de Maisonneuveで8月6日～12日に渡って開かれたDebConf17（**写真1**）の様子をお送りします。多くのセッションが録画されていますので、サイトのスケジュール一覧[注1]から興味のあるセッションを選んでご覧になってください。ここではその中のいくつかをかい摘んで紹介します。

DebConfとは?

DebConf（Debian Conference）は毎年開催されているDebian開発者（とその他の人たち）向けのイベントで、開催地を固定せず世界各地で開催されています。おおよそ1週間（+前乗りしてくる人たち向けのDebCampでさらに1週間）のイベントで、朝から晩までセッションやBoF[注2]・Hack、あるいは（ときにはビールを飲みつつの）ディスカッションを行います。参加者は2～500名程度で、北米・ヨーロッパが中心ではあるものの、南米やアジア・アフリカ地域からの参加者もいます。

▼写真1
DebConf17の案内板

注1）URL https://debconf17.debconf.org/schedule/
注2）Birds of Featherの略で「非公式の議論グループ」のこと。「とにかく集まって話をしよう」ということですね。

Debian初心者による初心者向けDebianの解説

DebConfでは「Open Day」という外部への公開日がありますが、この日に行われたDebConfスポンサー企業・Collabora社のHelen Koikeさんによる「Debianとは？」というテーマの発表「A newbie's newbie guide to Debian」[注3]の資料が非常によくまとまっています。Debian自体とそれを取り巻くプロジェクトの概要について把握・理解したいという方にお勧めです。

「The Machine」とそれを支えるDebian

DebianではおもにX.org周りのメンテナをしているKeith Packardさんによる HPE（Hewlett Packard Enterprise）のスポンサートークでは、「The Machine」[注4]という今までのコンピュータからアーキテクチャを見直してメモリを根本に据えた「メモリドリブン」なシステムの開発話が行われました。不揮発性の「メモリプール」を中心に各ノードが超高速でメモリにアクセスすることにより、「超」が付くほどの大量のデータを従来の数十分の1の消費電力と時間で処理する

注3）URL https://debconf17.debconf.org/talks/68/
注4）URL https://www.labs.hpe.com/the-machine

ことが可能になるのがウリのようです。

ちなみにHPEというと、昨年SUSEがHPEのOpenStackおよびCloud Foundry関連資産を買収しました。そうなると、今までHPEがOpenStack「Helion」関連で使っていたDebianベースのHPE社内ディストリビューション「hLinux」の行方が気になるところですが、このThe Machineのインフラ管理周りに利用しているとのことです(とはいえ、人員についてはリストラも行われたようです……)。

aptの改善とさらなるアイデア

今年もぬいぐるみの「Cow」と一緒に(**写真2**)、aptメンテナのJulian Andres Klodeさんによるaptの改善進捗報告として「An apt talk」という発表が行われました[注5]。

Debianの重要なピースであるパッケージマネージャー「apt」ですが、まだまだ改善の手が加えられています。異なるアーキテクチャに移植作業をすることでバグを洗い出せるため、現在はaptを*BSDに移植しようと考えているそうです。ただ、現在CI(継続的インテグレーション)として利用しているのがTravis CIで、これはWindowsとLinuxとmacOSはサポートしていますが、*BSDの対応はありません。そこでまずはmacOSへの移植を試みているとのこと。実機は持っていないのと、まだいくつかパッチを書いただけで、まだ完了していないらしいですが、将来的にmacOSでもaptがネイティブで動作するかもしれません[注6]。

▼写真2
Julian Andres Klodeさんとともに登壇したCow

注5) URL https://debconf17.debconf.org/talks/58/

注6) パッケージがきちんと動作するかどうかはまた別問題です。

unattended-upgradesの動作変更

Windows Updateのようにバックグラウンドで自動的にパッケージのセキュリティアップデートを行う「unattended-upgrades」ですが、動作にいくつか変更が実施されました。

- aptのバージョン1.2.9までは、日時cronジョブがAM6〜7時の間に動作していた。その際、ミラーの負荷を減らすために30分までのランダムスリープを付加していた。しかし、Ubuntuでミラーサーバの負荷が耐えられないほどになっていたので改善を迫られていた
- 1.2.10でcronからsystemd timerに乗り換え、AM6時から前後12時間(=つまり適当な時間)に実行するようにした……のだが、パッケージアップデートが日中に実行されシステムを壊すことが発生。また、起動時やネットワークの接続を待たないで実行されるようになってしまった
- バージョン1.4.1〜1.4.6の間でupdateとupgradeにジョブを分け、updateは適当な時間に実行、upgradeをAM6〜7時の間に動作するようにして日中にシステムが破壊されないようにした。また、systemd timerはnetwork-online target依存にして起動時にネットワークがつながっていない場合は実行されないように担保した。ただ、これでも不十分でシステムがスリープから復帰したときにはネットワークがつながっていない状態で動作してしまった。ちなみに今も改善作業中の様子で、本稿執筆時のapt 1.5に至るまでも微調整を続けている様子が見て取れます

ネイティブでのHTTPSサポート

aptのHTTPSサポートについては、2006年時点でcurlベースで実現していましたが、HTTPのリポジトリへ接続する際にHTTPSプロキシ経由ではつながらなかったそうです。これを2017年のapt 1.5で新しく書き直し、きちん

とサポートするようにしています。

apt-keyコマンドが非推奨に

aptリポジトリを追加する際、リポジトリのデジタル署名に使われているGPG鍵情報を信頼する対象として追加するapt-keyコマンドについて、必要となるgnupgパッケージの依存関係が「Suggests」に変更されていて明示的にインストールしないと利用できないなどしており、「非推奨になっているので使わないようにしてほしい」とのこと。ただ、会場からは「Too late!」と苦情が出たり、代わりに示された手法がそれなりに面倒そうだったりと、今後何かしらの改善を期待したいところです。

その他の変更点

apt 1.5からは、ユーザがapt line[注7]で「stable」を指定している場合に、パッケージのリリースが行われてリポジトリのReleaseファイル内のコードネームが変わったら、ユーザに確認するようになったようです(意図せずにバージョンをメジャーアップグレードすることが避けられるでしょう)。

また、カンファレンスならではの話として、食事の際に周りと雑談していたことがそのまま開発につながることがあります。aptについても、差分パッケージ[注8]について、「現在debdeltaというツールはあるが、apt/dpkgに統合されていないし、別のアプローチがあるのではないか?」という雑談から試行錯誤が始まりました[注9]。まだまだaptの進化は続きそうです。

Googleの社内ディストリビューションがDebianベースに

Googleに勤めるDebian開発者のMargarita Manteloraさんのライトニングトークで、Google社内デスクトップ環境として利用していたLinuxを、Ubuntuベースの「goobuntu」と呼ばれていたものからDebianベースの「gLinux」に変更することが明かされました(**写真3**)。

gLinuxは次のような特徴を持ちます。

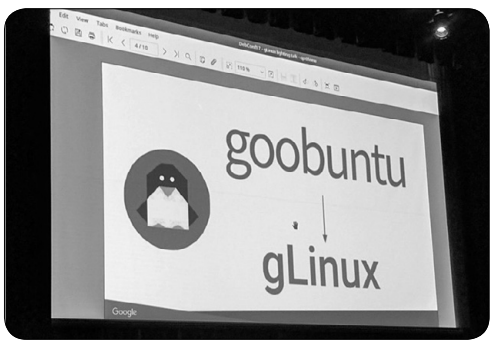
- Debianテスト版をベースとして開発
- 常に変更を取り入れる「ローリングリリース」
- 問題があるバージョンについてはパッケージを「ホールド」してリリース対象にならないようにする
- パッケージはすべてGoogle社内で再ビルドを行う

発表時点ではアルファ版、8月16日にベータ版をリリースとのことでした。Google社内での利用からDebianへのフィードバックが多くなれば良いですね。

医用分析機器でもDebianが動作

DebConf17には、スポンサーとして世界的な製薬・ヘルスケア企業であるRoche社[注11]が参加していました。Rocheに勤務するAndre Rothさんによる「Debian for Medical Software」というセッションでは、Rocheで利用している医用分析機器の内部でDebianをベースにしたディストリビューション「Roche Linux」が動いてい

▼写真3　Google社内デスクトップ環境はgLinux(Debianベース)に[注10]



注7) パッケージの入手先の設定。具体的には/etc/apt/sources.listなど。
注8) RPMでいうところのdrpmのような、変更分の差分(delta)のみを配布してダウンロード量を減らす手法。
注9) URL http://deb.li/3eHGE
注10) 写真の出典:URL https://twitter.com/mbanck/status/896029270136553472　CC-BY-SA 3.0の下で利用。
注11) URL http://www.roche.com/

る話が披露されています[注12]。

Rocheでの医用分析機器の初期はリアルタイムオペレーティングシステムを使っていたが、より複雑な処理をする必要が出てきたためLinuxに移行したとのこと。2007年当時のRoche Linux1とRoche Linux2ではPengutronixというベンダのPTXdistという、ソースからすべてビルドするシステムを使っていたそうです。しかし、さらに新しいハードウェアをサポートしたい、というところから分析したところ、既存システムでは満たされていない次の要望が上がってきました。

- さまざまな内部プロジェクトで利用できるようにしたい(モジュラー形式で自由に追加と削除ができるように)
- 実運用環境だけではなく、開発やテストにも使いたい
- セキュリティの長期サポートがほしい
- より多くのライブラリやツールを利用できるようにしたい
- そのうえでメンテナンスの労力は下げたい

そこで**表1**のような評価項目について検討し、「Make / Take / Buy」(作るか・持ってくるか・それとも買うか)という判断の結果、メインラインのディストリビューションであるDebianを「Take」(持ってくる)という決断を下し、2013年のRoche Linux3からDebianをベースにしているとのこと。

医用ソフトウェアは、法律の関係から監査や記録が要求され、再現性が必要となるためにビルド時に変更が意図せずに行われるインターネット上の外部リポジトリには依存せず、内部でリポジトリを持たねばならないなど、さまざまな制限があるようです。とくにディストリビューションとして「Reproducibility(再現可能性)」を重視しているDebianを採用することで、このあたりが楽になっているそうです。内部リポジトリはaptlyを、設定にはAnsibleを使い、ビルドイメージまたは開発用のLinux Container(LXC)として提供しており、aptlyには自社内での利用をふまえてcontributionを行うなどしている模様です。

また、社内クライアントとして「RLC」があり、こちらはUbuntuベースです。現在のところ同社内にはWindowsが120,000台、macOSが15,000台に対してRLCは300台ほどでしかありませんが、まだまだ増え続けているそうです。

日本からの参加者の発表も多数

今回のDebConfでは、多くの日本からの参加者によるセッションが開かれました。

Debianのドキュメントの取りまとめやインプットメソッド周りで活動されている青木修さんは「Modernized packaging tutorial and practical challenge of DEP-5」と題した発表を行いました[注13]。発表では、ソースパッケージ内のdebian/copyrightファイルの書式「DEP5」に触れ、「debmake」であればDEP5に準拠した形でのテンプレートをある程度自動生成してくれるなどの優位性を挙げ、パッケージングに使われているツールとしてdh-makeからdebmakeへの移行を促しました。青木さんは長いDebian開発者歴にもかかわらず、意外にも初めてのDebConf参加ということでしたが、発表外の時間は手つ

注12) URL https://debconf17.debconf.org/talks/165/

注13) URL https://debconf17.debconf.org/talks/48/

▼表1　Roche Linuxでの評価項目

ポイント	各項目
効果対コスト	移行労力、ライセンス、商用サポート、開発者のトレーニング
柔軟性	モジュラリティ、ハードウェアサポート、フットプリントのサイズ、設定変更が可能かどうか
ユーザビリティ	利用可能なツールの種類、簡易に使えるかどうか
長期性	長期サポート、再現可能性(Reproducibility)、コミュニティ
信頼性	アップグレードが容易、セキュリティサポート

かずになっているドキュメントの扱いについて精力的に関係者に働きかけて合意を得るなどされていたようです。

前回のDebConf16に引き続きの発表となるロジャー清水さんは、自身の出身地である中国の金盾(グレートファイアウォール)による検閲をどのようにしてくぐり抜けて利用するかという発表「Make use of Debian to fight with censorship -- alternative way other than tor」を行いました[注14]。速度の問題からこの手の話題でよく名前が出るTorではなく、ほかのやり方を試みているとのこと。

暗号化の重要なソフトウェア「GnuPG」のupstream developerでもある新部裕さんの発表「GnuPG 2.1 Explained for Everyone」では、一緒に参加されていたご家族とともに「GPGの動作の寸劇」を行って会場を沸かせました[注15]。続けての発表「Let's use Ed25519 with GnuPG 2.1 and Gnuk Token!」ではGPG 2.1で楕円曲線暗号の1つ「Ed25519」[注16]の利用を提案しました[注17]。

Debian開発者以外では、Linux Foundationの「最も保守的」なプロジェクトであるCivil Infrastructure Platform(CIP)から、小林良岳さんが「Debian on Civil Infrastructure Systems」という発表(**写真4**)にて、交通・エネルギー・工場などのさまざまな社会生活にとってクリティカルな分野の製品のベースにDebianが使われていることを明かしました[注18]。そして、25〜50年のライフサイクルを持つ鉄道などの産業システムを例に挙げ、このようなシステムを運用していくための超長期サポート(SLTS、Super Long Term Support)の必要性を訴えました。現在のDebianではLTSでも5年ですから、まずは7〜10年あたりをターゲットに相談ができると良いですね。

なお、CIPはLinuxカーネルとして4.4シリーズを選択したこと、そのメンテナとしてDebianのカーネルパッケージメンテナでもあるBen Hutchingsさんが担当することが明かされました。LinuxカーネルについてはDebianとCIPで同一人物がメンテナンスを担当していることで、スムースに協力が築けそうです。

注14) URL https://debconf17.debconf.org/talks/90/
注15) URL https://debconf17.debconf.org/talks/32/
注16) URL https://ed25519.cr.yp.to/ djbdnsなどで有名なDaniel J. Bernsteinらにより開発された。
注17) URL https://debconf17.debconf.org/talks/162/

▼写真4 小林氏のCivil Infrastructure Platformの発表

たまには ハズレの発表も……

カンファレンスに参加すると発表のクオリティは玉石混交で、当たりばかりとは限りません。筆者が参加した「whalebuilder: building packages using Docker」というセッション[注19]は、今流行りのDockerを使ってパッケージをビルドするというので、何か大きく改善される話が出てくるのか……と思いきや、既存のpbuilder/cowbuilderベースのビルドとの違いがあまりわからなかったので拍子抜けでした。デモもあまり練ってきていないようでエラー連発で締まらない感じとなり、あらためて発表の際の準備が重要だと思わされました。

次号へ続きます

まだまだ話題は尽きませんので、次回も引き続きDebConf17の話題をお届けしたいと思います。SD

注18) URL https://debconf17.debconf.org/talks/101/
注19) URL https://debconf17.debconf.org/talks/84/

SOURCES

レッドハット系ソフトウェア最新解説

第14回

ソフトウェアリポジトリのライフサイクル管理

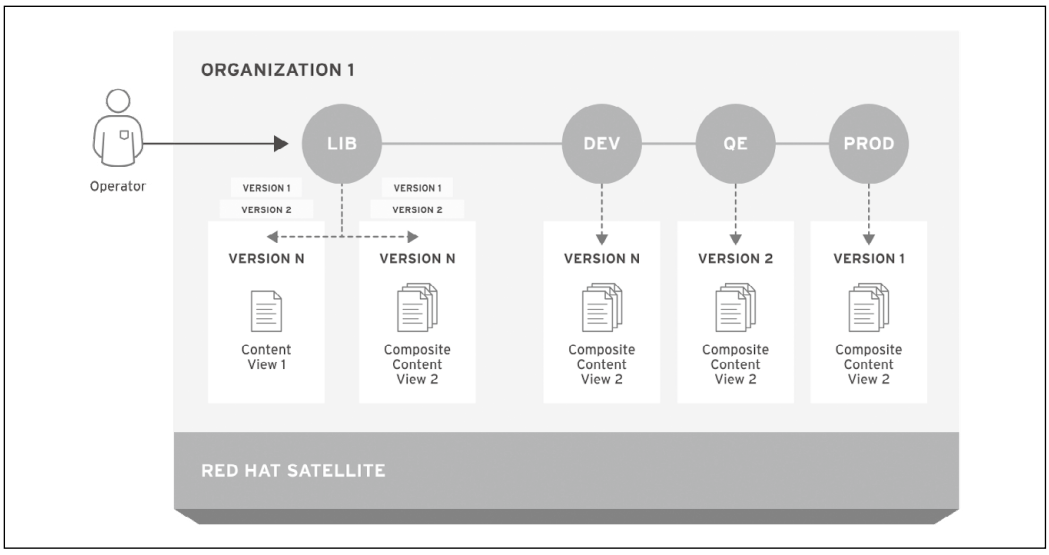
アップデート検証から本番適用までのワークフローを確立する手助けとなる、ソフトウェアリポジトリのライフサイクル管理方法を紹介します。

Author 小島 啓史(こじまひろふみ)
mail : hkojima@redhat.com
レッドハット㈱ テクニカルセールス本部 ソリューションアーキテクト

なぜライフサイクル管理が必要か

ITシステムの運用をしていくなかで頭を悩ませることが多いのは、おそらくアップデート関連の話です。とくに個人情報を取り扱うような重要度の高いシステムでは、定期的に脆弱性を潰すためのアップデートが求められる可能性が高くなります。そのようなケースに対応する、アップデート検証から本番環境への適用までのワークフローのイメージが**図1**となります。**図1**ではライブラリ／開発／QE／本番環境の4つの環境と複数のバージョンがあるリポジトリが存在し、ライブラリと開発環境には最新版のリポジトリ(図中では後述する「コンテンツビュー」と記載しています)を割り当てており、QE環境にはバージョン2のリポジトリを、本番環境にはバージョン1と最も古いバージョンのリポジトリを割り当てています。これは各バージョンのリポジトリを、開発→QE→本番の順番で検

▼図1　リポジトリのライフサイクル管理を利用したワークフローのイメージ



証・割り当てしていった結果となります。こうしたライフサイクル管理により、アップデートだけでなく新機能追加などでも一貫性を持たせられるようになります。

では、このようなライフサイクル管理を導入するにはどうすべきでしょうか。1つの方法として、wget/rsyncなどを活用してカスタムソフトウェアリポジトリを作成し、アップデートの際にはリポジトリのスナップショットをcpなどで作成して各環境での検証を順番に行うことが考えられます。Red Hatでは、こうした作業の自動化やアップデート前後のシステム状況を確認するためのインベントリ機能を備えているRed Hat Satelliteの利用を推奨しています。以降はこのようなライフサイクル管理を想定した、Red Hat Satelliteの利用例を紹介していきます。

Red Hat Satellite 6の準備

次の手順はRed Hat製品であるRed Hat Satellite 6(以下Satelliteと記載)の利用を前提としますが、もしOSSコミュニティ版のソフトウェアを利用して試してみたい場合は、Satelliteのベースになっている Katello[注1] やForeman[注2] を利用して同様の手順を実施してみてください(**図2**)。その際にはインストールガイド[注3] も参照できます。

まずはSatelliteをインストールする最新のRHEL7サーバを1台用意して、Satelliteをインストールするための製品リポジトリを有効にします。このとき利用するポート[注4] も忘れずに開いておいてください。

続いてSatellite関連パッケージをインストールし、インストールスクリプトを実行します。しばらく待つとインストールが完了し、Satelliteサーバにアクセスするための URL やユーザ情報が表示されますので、適当なWebブラウザを利用してSatelliteサーバにログインします。サポート対象のWebブラウザを利用する場合は、Firefox 35+かChrome 28+を利用してください。

▼図2　RHELの登録とSatellite製品リポジトリの有効化

```
# subscription-manager register --auto-attach
# subscription-manager repos --disable="*"
# subscription-manager repos c
    --enable=rhel-7-server-rpms \
    --enable=rhel-7-server-satellite-6.2-rpms \
    --enable=rhel-server-rhscl-7-rpms \
```

▼図3　hammerコマンドを使うための認証情報の設定

```
# cat << EOF > ~/.hammer/cli_config.yml
:foreman:
 :host: 'https://FQDN_OF_SATELLITE/'
 :username: 'admin'
 :password: 'PASSWORD_OF_ADMIN_USER'
EOF
```

```
# yum -y install satellite
# satellite-installer --scenario satellite
```

ちなみに、Satelliteをインストールすると管理ユーザとしてadminユーザが作成されます。今回は作業を単純化するために、このadminユーザ権限でSatelliteのCLIツールであるHammerを使って管理作業をしていきます。なお、hammerコマンドを実行する際の認証情報もユーザのホームディレクトリに保存しておきます(**図3**)。これでSatelliteの準備は完了です。

コンテンツビューの作成

コンテンツビューは、Satelliteサーバが保持しているソフトウェアリポジトリの集合です。Satelliteが保持できるソフトウェアリポジトリの種類は、Yum、Docker、Puppetとなります。コンテンツビューを公開してクライアントが利用できるようにするには、公開時点でのスナップショットを作成する必要があります。各クライアントにはそれぞれスナップショット取得時

注1) URL http://www.katello.org/
注2) URL https://theforeman.org/
注3) URL https://theforeman.org/plugins/katello/3.4/installation/index.html
注4) URL http://red.ht/2xkly0N

▼図4　RHEL7.3のRPMパッケージのインポート

```
# hammer product create --name "rhel7-rpms"
# hammer repository create --name "rhel7u3-rpms" --content-type "yum" --publish-via-http “true”
--url “http://WEB_SERVER/RHEL/7/3/” --product "rhel7-rpms"
# hammer repository synchronize --name rhel7u3-rpms --product rhel7-rpms
```

▼図5　RHEL7.3のRPMパッケージが含まれるコンテンツビューの公開

```
# hammer content-view create --name "rhel7-content-view"
# hammer content-view add-repository --name "rhel7-content-view" --repository "rhel7u3-rpms"
--product “rhel7-rpms”
# hammer content-view publish --name "rhel7-content-view"
```

▼図6　作成したコンテンツビューのプロモート

```
# hammer lifecycle-environment create --name "Production" --prior "Library"
# hammer content-view version promote --content-view "rhel7-content-view" --version 1 --to-
lifecycle-environment "Production"
```

▼図7　公開およびプロモートされたコンテンツビューのイメージ

点のコンテンツビューが割り当てられ、基本的にはそれに含まれるソフトウェアを利用するしくみとなります。最初に記載したリポジトリのライフサイクル管理には、コンテンツビューのスナップショットを活用することになります。

コンテンツビューを作成するには、まずソフトウェアリポジトリを作成する必要があります。ここではRHEL7.3のISOイメージに含まれるRPMパッケージをインポートしてみる例を紹介します[注5]。まず「rhel7-rpms」という名前の製品（ソフトウェアリポジトリを置く論理リソース）を作成したあとに、「rhel7-rpms」に紐付けるリポジトリ「rhel7u3-rpms」を作成します[注6]。リポジトリを作成したあとには、synchronizeサブコマンドを実行して、WebサーバからRHEL7.3のRPMパッケージを同期します（**図4**）。

同期完了後に、コンテンツビュー「rhel7-content-view」を作成して「rhel7u3-rpms」リポジトリを紐付けます。そしてpublishサブコマンドを実行して、「rhel7-content-view」を公開します。publishサブコマンドを実行すると、コンテンツビューのその時点のスナップショットが取得・公開されます（**図5**）。

公開した「rhel7-content-view」のスナップショットを、「Library」と「Production」の2つの環境で利用することを想定してみます。デフォルトで作成済みの「Library」環境に割り当てられたコンテンツビューのスナップショットのプロモート先として「Production」環境を作成し、前の手順で公開した「rhel7-content-view」のスナップショット（ここではバージョン番号1が付与されています）を「Production」環境にプロモートします（**図6**）。この結果、「rhel7-content-view」のスナップショットを「Library」環境と「Production」環境で利用できるようになります（**図7**）。

ここまでの手順が完了すると、コンテンツ

注5） 以降のhammerコマンドは、Satelliteでデフォルトで作成される組織名の指定「--organization “Default Organization”」の記載を省略しています。

注6） この例では、RHEL7.3のISOイメージを適当なディレクトリにマウントして、Webサーバとして公開していることを想定しています。

▼図8　Satelliteへのシステム登録と“Production”環境のコンテンツビュー紐付け

```
# yum -y install http://satellite.example.com/pub/katello-ca-consumer-latest.noarch.rpm
# subscription-manager register --org="Default_Organization" --environment="Production/rhel7-content-view" --username admin --auto-attach
```

▼図9　登録したRHELシステムの情報表示イメージ

▼図10　コンテンツビューの新しいスナップショット公開イメージ

バージョン	状態	環境	コンテンツ	説明	アクション
バージョン 4.0	公開 (2017-09-20 08:22:15 UTC)	Library	9737 パッケージ		プロモート 削除
バージョン 1.0	Production にプロモート (2017-09-20 07:20:13 UTC)	Production	4751 パッケージ		プロモート 削除

ビューの利用準備が整ったことになります。RHEL7システムを1台用意して、Satelliteサーバへ登録してみます。Satelliteサーバに登録するための情報を含んだRPMパッケージをインストールし、「subscription-manager register」コマンドを実行して登録します(**図8**)。この登録の際に、「Production」環境の「rhel7-content-view」コンテンツビューを--environmentオプションで紐付けておきます。登録したRHELシステムは、**図9**のような形式で確認できます。

そして、このシステムで「yum repolist」コマンドなどを実行して、RHEL7.3のRPMパッケージが参照できることを確認できます。さらに同様の手順で新たにRHEL7.4のRPMパッケージを追加して、コンテンツビューの新しいスナップショットとして公開し、「Library」環境に紐付けてみます(**図10**)。この場合、「Library」環境のコンテンツビューに紐付けられたシステムのみが、RHEL7.3とRHEL7.4のRPMパッケージを参照できるようになります。先ほど「Production」環境に紐付けたシステムが追加されたRHEL7.4のRPMパッケージを参照するには、コンテンツビューの新しいスナップショット(**図10**ではバージョン4.0と記載)をプロモートする必要があります。

今回で紹介したソフトウェアリポジトリのライフサイクル機能を活用すると、テスト環境と本番環境で同一のリポジトリを使うことができますので、アップデートや新機能追加などの過程でパッケージの差異による不具合が発生する可能性を減らすことができます。ぜひともご参考にしてください。SD

Ubuntu 17.10の変更点

Ubuntu Japanese Team
あわしろいくや

今回は10月19日にリリース予定のUbuntu 17.10とそのフレーバーの変更点をお知らせします。

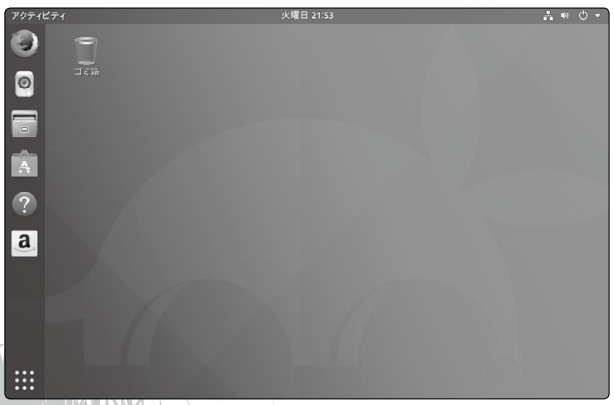
概要

リリース日とコードネーム

Ubuntu 17.10は10月19日にリリース予定で、コードネームは“Artful Aardvark”「巧妙なツチブタ」という意味です。17.04のZのあと、順当にAに戻ってきましたが、正確にはUbuntuのコードネームはAから始まっているわけではなく、実はAから始まるのは今回が初めて、というやや複雑なことになっています[注1]。サポート期間はすべてのリリースで9ヵ月間です。

注1） Aは今回が初めて、Bは次で2回目（前は5.10）、Cはまだなく、Dからは2回目が始まりますが、Wのようにすでに2回使用しているものもあり（4.10と15.10）、歴史的経緯でこのあたりのことを真面目に考えてしょうがないことになっています。

図1 Ubuntu 17.10のデスクトップ



UnityからGNOME Shellへ

2017年6月号の本連載第86回で既報ですが[注2]、このUbuntu 17.10からデフォルトのデスクトップシェルがUnity 7からGNOME Shellに変更になりました。

今回はリリースの1ヵ月前時点での情報ですので、実際のリリースとは異なる可能性はありますが、ひとまずは**図1**のUbuntu 17.10のスクリーンショットをご覧ください。Unityに見えるかもしれませんが、これはれっきとしたGNOME Shellです。

フレーバー

UbuntuがGNOME Shellを採用したということで、Ubuntu GNOMEのリリースは行われません。Ubuntu GNOMEをアップグレードするとUbuntuになる見込みですが、Ubuntu GNOMEの構成のままになる可能性もあります。一方、LXDEを採用するLubuntuは新たにLXQtを採用するLubuntu Nextもリリースすることになりました。

共通の変更点

カーネルのバージョンは4.13で、リリース時点で最新版ですが、X.Orgのバージョンは1.19.3であり、17.04と同じバージョンです。これは単純にこのバージョンが最新だからです。グラフィックスライ

注2） 2017年7月号の第87回ではUnityの歴史や機能を解説し、第88回ではUnityからGNOMEへの移行状況が解説されているので、併せてご一読ください。

ブラリであるMesaは17.2.1になる見込みです。

ネットワークの設定を抽象化するnetplan（パッケージ名はnplan）がデフォルトでインストールされるようになりました。ネットワークデーモンがNetwork Managerであってもsystemdのnetworkdでも設定を共通化できるというものですが、普段デスクトップのみ、あるいはサーバのみで使用しているのであれば意識することはないかもしれません。

Ubuntu

Ubuntu版GNOME Shellの特徴

Ubuntu版GNOME Shellは素のGNOME Shellとは大きな違いがあります。素のGNOME Shellとは何かというのは定義が難しいですが、ここではUbuntu GNOME 17.04のGNOME Shellとします。

Ubuntu版GNOME Shellはテーマを Unityで使用していたAmbianceにしたほか、デスクトップにアイコンを表示するといった小さな変更や、2つの拡張機能をデフォルトで有効にしているという変更点があり、これが特徴となっています。

うち1つはUbuntu Dockで、デスクトップの左側に表示しているバーです。Unity Launcherに似せた表示になっていますが、アプリケーションを起動する場合は一番下にある格子状のアイコン（アプリケーションボタン）をクリックします。Unity Dashに相当する、画面左上にあるアクティビティをクリックしてもアプリケーションの起動はできますが、この場合は検索主体になり、アプリケーションのアイコンから選択したい場合はやはりアプリケーションボタンを押すことになるので、であれば最初からアプリケーションボタンをクリックしたほうがいいでしょう。

Ubuntu Dockの設定はGNOMEの設定画面から変更できます。Dockの常時表示をやめたり、Dockの表示位置を変更したりといったことなどができます。

もう1つはUbuntu appindicatorsで、これはGNOME ShellでUnityのAppindicatorをサポートするための拡張機能です。GNOME Shellで右上にアイコンを表示しようとすると拡張機能にするしかないため、この拡張機能が有効になっています。Appindicatorに対応しているアプリケーション、たとえばRemminaを起動してみると、右上にアイコンが表示されるのがわかります。

ほかにも注目すべき点として、ウィンドウのボタンが右に移動したことです。これはGNOME Shellを踏襲した形ですが、設定上はUnityと同じく左にすることもできます。それをこの機に右のままにしたということは、さほどの重要性がなかったということでしょう。もともとのプランでは空いている右側で何かをするということでしたが、結局何も行われませんでした。

GDM

17.04まではUbuntuのデスクトップマネージャーはLightDMでしたが、17.10ではGDMになりました。これはGNOME ShellとLightDMの組み合わせでは不都合があるからですが、LightDMにあったゲストセッションなどの機能はGDMにはなく、また後述する各種セッションをインストールした場合、GDMだとGDM自体を再起動しないとそのセッションを選択できないといった不便な点があります。

Wayland

GNOME Shellに移行することによる最大のメリットは、おそらくディスプレイマネージャーにWaylandを選択できることでしょう。17.10の段階ですでにデフォルトのセッションになっており、NVIDIA/AMDのプロプライエタリなドライバを使用しているといった例外を除けばWaylandを使用することになります。ログイン時にX.Orgのセッションも選択できるようになっているため、Waylandでは不都合がある場合はそちらを選択してください。

今のところWaylandだからといって明確なメリットがあるわけではありませんし、VNCやFcitxが使用できないなどの制限もあります。今後はディスプレイごとにDPIの設定を行えるなどといったWaylandのメリットも享受できるようになるため、自分の使用環境をWaylandにした場合どういった問題が出るのかの洗い出しは早めにやっておいたほうがいいよう

に思います。

GNOME 3.26

17.10は9月13日にリリースされたばかりのGNOME 3.26 “Manchester”にアップデートしています。大きな変更点は設定ツールのルック＆フィールが変更されたことや検索インターフェースの変更ですが、ほかにもさまざまな変更点があります。

これまでレガシートレイアイコンとして左下にアイコンが表示されることがありましたが、この機能はなくなりました。どうしてもレガシートレイアイコンを表示したい場合は、Topiconsという拡張機能をインストールするといいようです。

Unityセッションと GNOMEセッション

Ubuntu 17.04からアップグレードした場合、あるいはunity-sessionパッケージをインストール後GDMを再起動すると、Unity 7セッションが使用できます。システム設定の「オンラインアカウント」や「セキュリティとプライバシー」が使用できなくなったなどの制限がありますが、GNOME Shellに慣れない場合はこちらを使用するのも手です。とはいえ、おそらく18.04 LTSでは残るものの、それ以降の早い段階でリポジトリから削除されることが予想されるため、早めにGNOME Shellに慣れておくのがいいでしょう。

gnome-sessionパッケージとして素のGNOME Shell用のセッションもありますが、前述の拡張機能が有効な状態になっており、Ubuntuセッションと大差ないものになっています。ただしUbuntu GNOME 17.04からアップグレードした場合は、素のGNOME Shellになります。これはUbuntu用のセッションがインストールされていないからだと思われます。

日本語入力

この記事を執筆している9月中旬現在では、日本語入力がどうなるのか決まっていません。このままリリースされたとすると、ログイン後設定の「地域と言語」を開いて「入力ソース」で「日本語（Mozc）」を追加するという手間がかかることになりそうです。

Kubuntu

現在KDEはKDE Software Compilation（SC）として3つに分割してリリースされています。Plasmaが5.10.5、Frameworksが5.38.0、Applicationsが17.04.3というバージョンです。リリースまでにアップデートされる可能性はあります。

17.10ではメディアプレイヤー関連に大きな変更があり、動画再生のDragon Playerが著名なVLC Media Playerに、音楽再生のAmarokがCantataに置き換えられました。Amarokも著名な音楽プレイヤーでしたが、KDE Frameworks 5にポーティングしたバージョンがリリースされておらず、置き換えられることになったようです。現在作業中のようなので、リリースされればまたデフォルトの音楽プレイヤーに戻るのかもしれません。

Xubuntu

Xubuntuはあまり大きな変更はなく、順当なバージョンアップとなっています。Xfceは現在GTK+ 3へのポーティングを行う4.14を目指して開発中ですが、このままの開発速度で行くと来年にはリリースできるのではないでしょうか。

Lubuntuと Lubuntu Next

Lubuntuはメディアプレイヤーがgnome-mplayerからgnome-mpvに変わったくらいで大きな変更はありません。Lubuntu Nextは前述のとおりデスクトップ環境をLXDEからLXQtに変更し、それに伴い構成するアプリケーションにも大幅な変更があります。

Ubuntu MATE

MATEのバージョンはマイナーバージョンアップは行われているものの1.18で、17.04と変わっていません。Ubuntu MATEはMATEの強力なカスタマイズ性を活かし、Unity 7ライクなルック＆フィールを

実装してしまいました。正確にいえば前からあったのですが、17.10でその完成度を大幅に高めたのです。[システム]-[設定]-[ルック&フィール]-[MATE Tweak]でMATE Tweakを起動し、「パネル」の「Traditional」を「Munity」に変更してください。この際、パネルに施されている変更はすべて破棄されますので、ご注意ください。

MunityでPlumaを起動し、Altキーを押したところが**図2**ですが、Unity風のレイアウト、グローバルメニュー、Heads-Up Display(HUD)、Appindicatorが実現されています。残念ながらUnity Dashの再現はできていませんが、いずれリポジトリから削除されてしまうであろうUnity 7よりも、より長く残ると思われるMunityに乗り換えてしまうほうが、実はいいのかもしれません。

これまでUbuntuのセカンドチョイスとして存在感のあったUbuntu GNOMEがなくなったことにより、その役割を担うのはUbuntu MATEが有力だと思われるので、そういう点からも一度使ってみるといいのではないでしょうか。Fcitxが満足に動作するというのもポイントです。パネルレイアウトもTraditionalやMunityのほかにもWindowsふう(Redmond)やmacOSふう(Cupertino)もあるので、気に入るものが見つかることでしょう。

Ubuntu Budgie

17.10のUbuntu Budgieは、Budgie 10.4にアップデートされています。これまで右側に表示されるメニューのRavenで設定を行っていましたが、このバージョンからは設定画面が独立しました(**図3**)。

Budgie Welcomeを起動し、[Getting Started]-[Language & Input]にIBusとFcitxを切り替える機能と、CJK関連パッケージをインストールする機能が追加されました。しかし前者の機能は動作せず、後者も日本語関連でインストールするのがibus-anthyと、あまり理解せずに追加された機能と思われ、今いち使いどころがありません。Ubuntu Budgieユーザは修正に挑戦してみるのもいいでしょう。

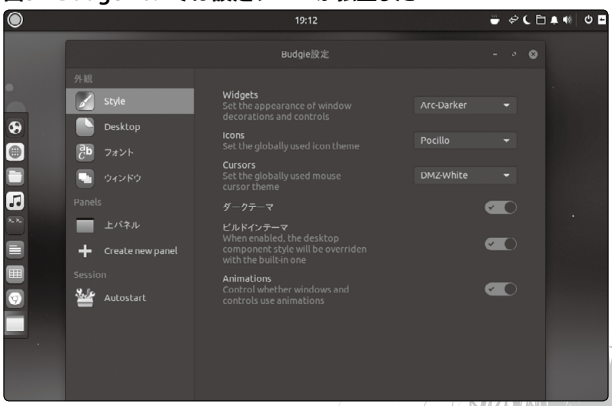
Ubuntu Budgieで日本語入力を行いたい場合は、17.04での方法が有効であると思われます。次のコマンドを実行してください。

```
$ gsettings set org.freedesktop.ibus.panel ↵
show-icon-on-systray true
$ gsettings set org.freedesktop.ibus.panel ↵
xkb-icon-rgba '#ffffff'
$ gsettings set org.freedesktop.ibus.general.↵
hotkey triggers "['<Control>space']"
```

これでIBusのステータスアイコンが表示されるようになりますが、BudgieのAppindicatorアプレットは安定しているとはいい難く、クリックするとフリーズしてしまうことがあります。これはFcitxでも同様の動きでした。よって、よりクリックする回数が少ないと思われるIBusのほうが安定して動作するといえます。SD

図2　Ubuntu MATEでUnity風のUIにし、HUDを有効にしたところ

図3　Budge 10.4では設定ツールが独立した



Unixコマンドライン探検隊
Shellを知ればOSを理解できる

Author 中島 雅弘(なかじま まさひろ) ㈱アーヴァイン・システムズ

複数のホストを縦横無尽に行き来する場面で役立つ、sshの応用を紹介します。

第19回 ssh(その3)

接続時のパスワード入力を省略するいくつかの施策

sshについては、本連載第15回、16回(2017年7月号、8月号)に続いての3回目です。

パスワードもパスフレーズも使わずにリモートホストにログインできれば、バッチ処理内などでリモートコマンドやリモートコピーをおこなうときに便利です。

パスフレーズなしの秘密鍵

あたりまえですが、パスフレーズなしの秘密鍵ならパスフレーズの入力は求められません。sshでパスフレーズとパスワードの入力をしない運用としては、このあとに説明するほかの方式に対して簡単な準備で対応できることや、安全度の高い公開鍵認証であることから広く使われている方式です。しかし、パスフレーズを指定しない秘密鍵は、機密性は下がりますので取り扱いには注意しましょう。

秘密鍵のパスフレーズは、ssh-keygen -pを実行すれば変更できます。

STEP UP! ホストベース認証

ホストベース認証とは、sshサーバとクライアントが、ユーザレベルの認証を省略して、直接認証を行う認証方式です。この方式では、サーバに登録済みのクライアントから接続があった場合は、そのクライアントのユーザ情報についても信用して接続を許可します。各ホストに同一のユーザアカウントが存在する環境において、ユーザがパスワードやパスフレーズを入力せずに各ホストにログインできるようになり、運用の効率が高まります。

たとえばシステム管理者が複数のクライアントを一元管理している環境においても、sshなら(古典的な手法の)rコマンドに対して、より安全にホストベース認証ができます。とはいえ、適切な機密性管理がされていないと、信頼できないホストからの接続が許されてしまうなどの危険があります。一般の利用者が不注意で機密を漏えいしてしまうことを避けるためにも、利用可能な環境を管理者に限定し、接続環境も、プライベートネットワーク内とする運用がよいでしょう。

ここからは、SSH2プロトコルでのホストベース認証の設定方法の流れを紹介します。

クライアント側の準備

①クライアントの設定ファイル(/etc/ssh/ssh_configもしくは、~/.ssh/config)で“HostbasedAuthentication yes”と記述する
②クライアントの設定ファイル(/etc/ssh/ssh_config)で“EnableSSHKeysign yes”と記述する
③ssh-keysignのオーナーをroot、パーミッショ

ンを4711(set-uid[注1]付きで、グループとそのほかのユーザは実行権限のみ)に設定

手順②と③の設定は、ホストベース認証の際に、**ssh-keysign**コマンドによって、ローカルホストの秘密鍵にアクセスしてデジタルシグニチャ(電子署名)を生成するための設定です(ユーザがssh-keysignを直接実行する必要はありません)。sshでクライアントに接続した際に、

```
ssh-keysign not enabled in /etc/ssh/ssh_config
```

というエラーが表示されるようなら、EnableSSHKeysignをyesにし忘れていないか確認しましょう。

ssh-keysignは、デフォルトでは手順③のパーミッションになっていないこともあります。また、環境によって、ssh-keysignのある場所が異なりますので注意してください(**表1**)。sshでクライアントに接続した際に、

```
initgroups masa 20: Operation not permitted
ssh_keysign: no reply
```

のようなエラーを見たら、ssh-keysignのオーナーとパーミッションを確認してください。

macOS SIPは、disableする必要がある

macOS(El Capitan以降)では、System Integrity Protectionがかかっていて、rootであっても変更が加えられないファイルやディレクトリがあります。SIPをdisableにしないと、ssh-keysignのパーミッションを変更できません。

```
$ sudo chmod u+s /usr/libexec/ssh-keysign
chmod: Unable to change file mode on /usr/libexec/ssh-keysign: Operation not permitted
```

SIPをdisableにする方法は、次の手順です。

1. システムを再起動
2. リンゴマークが出るまでComand+Rキーを押して、リカバリーモードで起動させる
3. UtilitiesメニューのTerminal(あなたの環境では日本語表記かもしれません)で、ターミナルを立ち上げ
4. csrutil disableとコマンドを入力し、成功のメッセージを見届ける
5. システムを再起動すれば、SIPはoffになる

パーミッションの変更が済んだら、同様の手順でにSIPをenableにしておきましょう。

注1) set-uidの解説は連載第7回(2016年11月号)を参照。

サーバ側での対応

①サーバの設定ファイル(/etc/ssh/sshd_config)で"HostbasedAuthentication yes"と記述する
②サーバの/etc/sshディレクトリの下のssh_known_hostsに、クライアントの公開鍵をコピーしておく
③サーバの/etc/hosts.equivにクライアントのホスト名または、IPアドレスを記述しておく
④サーバ側のsshd_configを変更したら、sshdに読み込ませる

手順②にあるサーバのssh_known_hostsへは、クライアント上の/etc/ssh/にあるホスト共通の公開鍵(ssh_host_rsa_key.pubなど)を追記します(単純に配置するだけでは動作しないこともありますので、記述形式についてはmanを参照してください)。

クライアントの/etc/sshにssh_host_*がない場合の作り方

使っている環境によっては、システム共通の鍵がない場合があります。そうした場合は、管理者権限で、次のようにssh-kegenコマンドを実行しましょう。

システムで使えるすべての鍵を作る場合

```
$ sudo ssh-keygen -A
```

特定の暗号方式の鍵だけを指定して作る場合には、明示的に暗号タイプを指定します。

rootでrsa鍵だけを作る例

```
# ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key -C '' -N ''
```

サーバの設定ファイル(/etc/ssh/sshd_config)に"IgnoreUserKnownHosts no"と記述してあれば、~/.ssh/known_hostsのホスト公開鍵も参照されます。サーバの~/.ssh/known_hostsが

▼表1 参考:ssh-keysignのあるところ(例)

環境	パス
macOS 10.12.6	/usr/libexec/ssh-keysign
Ubuntu 16.04	/usr/lib/openssh/ssh-keysign
CentOS 7.3	/usr/libexec/openssh/ssh-keysign

有効に登録されていれば、システム共通の鍵の登録操作はしなくても接続できます。

手順③で指定している/etc/hosts.equivは、許可するホストとユーザを1行に1組で記述したもので、rshによるホストベース認証のときから用いられています。host.equivはシステム全体で使われ、各ユーザは~/.rhostsに同様の形式で個別の情報を記述できます。sshでは、rコマンドで使うこれらのファイルのほかに、sshのみで使う/etc/shosts.equivや~/.shostsも参照します。サーバの設定ファイル(/etc/ssh/sshd_config)に"IgnoreRhosts no"と記述すれば、~/.rhostsまたは~/.shostsに書かれたホスト名も有効になります。

rshからの伝統で、ユーザがrootの場合は/etc/hosts.equivは参照されませんので、rootの~/.rhostsもしくは~/.shostsに記述をしておきます。rootのloginを許可するためには、さらに"PermitRootLogin yes"も記述します。

ホストベース認証は、ホスト名が接続許可の判断基準になります。サーバ側がクライアントのホストを認識している名前をhosts.equivやssh_known_host中に記述します。

hosts.equivに書くべきホスト名が不適切で接続できないときに、サーバが接続を試みているホストをどのように認識しているかは、ログを見るとわかるでしょう。サーバ側はtail -fで、ログの状態を監視しながらクライアントから接続してみます。

```
Ubuntuの例：随時logに追加される情報が表示されるので、接続元のホスト名を確認
$ sudo tail -f /var/log/auth.log
tail -fを終了するには、Control-Cを入力
```

/etc/ssh/sshd_configを編集したらsshdにSIGHUPを送って、設定を読み込ませます[注2]。

ホストベース認証のまとめ

誌面で紹介した範囲では、現実のさまざまな環境で生じる課題をすべてカバーできません。実際にはいろいろと試行錯誤[注3]が必要です。

そのほかのパスワード入力を省く施策

STEP UP! SSH_ASKPASS

SSH_ASKPASS環境変数は、sshでX Window Systemを使うためのしくみですが、SSH_ASKPASSを用いると、パスワード入力を自動化できます。Linuxでは、setsidというコマンド[注4]を活用します(**リスト1**)。

リスト1では、gpというスクリプト(**リスト2**)を呼び出して、plist(パスワードテーブル。**リスト3**)からホスト名に対応したパスワードを取得します。ここでは、単純にするため平文のパスワードをplist内に記述していますが、実用には何らかの方法で暗号化しておくのが良いでしょう。

この方式は、公開鍵認証のパスフレーズを省略したり、ホストベース認証のようにさまざまな設定をしなくてもパスワード入力を省略して接続できるのが利点ですが、認証に失敗した場合に適切なエラー処理がしにくいのが欠点です。

RubyのNet::SSHを使う

Rubyが使える環境なら、RubyのNet::SSHを使えば、sshをスクリプト中から自在に操ることができます。Net::SSHは標準では入っていないので、gemでインストールします。

```
$ sudo gem install net-ssh
```

詳細は、本家[注5]のドキュメントを参照してください。ここでは、パスワード認証と、公開鍵認証での簡単な例を**リスト4**に掲載しておきます。

この方式なら、簡単にsshを使ってバッチ処

注2) macOSについては不要です。連載第16回「ssh(その2)」(2017年8月号)を参照してください。

注3) ssh(その2)のsshのまとめとトラブルシュートなどを参考に。

注4) macOSには、このコマンドがありません。setsid(2)システムコールはありますので、C言語を使ってsetsidコマンドを作ることはできます。

注5) URL https://github.com/net-ssh/net-ssh

▼リスト1　SSH_ASKPASSを使ってパスワードを省略して、複数のシステムに自動で接続するスクリプト

```
#!/bin/bash

remote_cmd='ls -a'
hosts=(my-ubuntu.sonzaishinai.jp her-ubuntu.sonzaishinai.jp)
user=bot
hostinfo="./host.crr"

export DISPLAY=fake:0
export SSH_ASKPASS="./gp"

pids=()
for host in ${hosts[@]} ; do
  while [ -f ${hostinfo} ] ; do # 前の処理が終わるのを待つ
    sleep 1
  done
  echo ${host} > ${hostinfo}

  setsid ssh ${user}@${host} ${remote_cmd} &

  pids+=($!)
  echo "$host:${pids[@]}"
done

echo "----"

while (( ${#pids[@]} > 0 )) ; do
  echo ${pids[*]}
  pids=( $(ps ax | awk -v "p=${pids[*]}" 'BEGIN{split(p, pids)} {for (i in pids) ➥
{if ($1==pids[i]) left=left " " $1}} END{print left}') )
  sleep 2
done
echo "----"

echo "all remote tasks are done"
```

▼リスト2　パスワードデータベースplistからパスワードを抽出するスクリプトgp

```
#!/bin/bash
hostinfo="./host.crr"
host=$(cat ${hostinfo})
/usr/bin/awk "\$1==\"${host}\" {print \$2}" plist
rm  ${hostinfo}
exit 0
```

▼リスト3　plistにリモートホスト名とパスワードの組を記述しておく

```
my-ubuntu.sonzaishinai.jp  ore.ore.oredayo
her-ubuntu.sonzaishinai.jp  kiminonaha.➥
michikodesu
```

▼リスト4　Net::SSHを使ったパスワード認証と公開鍵認証のコーディング例

```
require 'rubygems'
require 'net/ssh'

# パスワード認証の場合
Net::SSH.start('my-ubuntu', 'ore', :password ➥
=> 'ore.ore.oredayo') do |ssh|
  puts ssh.exec! 'ls -a'
end

# 公開鍵認証
Net::SSH.start('her-ubuntu', 'ore', :keys => ➥
['~/.ssh/id_rsa'], :passphrase => ➥
'himitsunokagi') do |ssh|
  puts ssh.exec! 'ls -a'
end
```

理を実現できます。

sshをもっと便利に

configで面倒な設定をまとめる

接続先によって、ローカルの環境とアカウント名が違う場合など、毎回入力するのが面倒なことや、sshの挙動を決める設定は、.ssh/configに書いておくことができます。configファイルの

フォーマットは、man[注6]で確認してください。

configファイルの例
```
ServerAliveInterval 30
ServerAliveCountMax 4

Host aru.irvine.jp
    Port 10022
    ForwardX11 yes
    ForwardX11Trusted yes
    ForwardAgent yes

Host irvine.jp
    User ore

Host *
    User masa
```

ssh接続時の自動処理について

sshでログインする際、bashでの.bashrcや.bash_profileなどと同様に動作環境を設定することができます。~/.ssh/environmentは、追加の環境変数を定義するのに使います。リモートホスト側でsshd_configでPermitUserRCがセットされていて、リモートホストの~/.ssh/rcが存在すれば、ログイン時に実行されます。これがなければ、/etc/ssh/sshrcが実行されます。

~/.ssh/rcで無用な表示などをすると、バッチ処理やscpなどsshを使うコマンドなどが期待どおりの動作をしないことがありますので、sshログイン時にどうしても必要な個別の設定のみをするのがよいでしょう[注7]。

ポートフォワードを便利にする工夫

~/.profile、~/.bash_profileで多段で接続する設定を便利にする

連載第16回のssh(その2)で解説した、リモート／ローカルポートフォワードと、autosshを使ったトンネリングの方式では、踏み台のホストに接続して、そこでローカルポートに接続することで目的のホストに接続しました。この一連の手順を簡便にするために、**リスト5**のような定義をしておくと便利になります。

.profileなどで定義しておけば、端末を立ち上げたときに(1つだけ)autosshが自動で実行されます。リモート側でも同様にトンネルを作っておけば、achiraとコマンドプロンプトで入力するだけで、踏み台経由でリモートのプライベートネットワーク内のホストに接続できます。

autosshのモニタリングポート

autosshでは、接続状態を確認するためのモニタリングポートを-Mで指定する必要があります。ポート番号には、使っていないポートを指定するのですが、ポート番号0を指定すると、モニタリングポートを使用しないで、切断のシグナルをconfigファイル中で(OpenSSHの比較的新しいバージョンにある設定)ServerAliveInterval(sec)、ServerAliveCountMaxi(回数)によって受け取ります。この方法のほうが、モニタリングポートを使うよりもよいとmanpageでは推奨しています。

scpも踏み台経由で使いたい

連載第16回のssh(その2)で解説したリモートポートフォワード(-Rオプション)の設定ができている環境で、sshだけでなくscpやrsyncなども踏み台を経由して接続する場合の設定方法を確認しておきましょう。

次のように、~/.ssh/config中に、ProxyCommandをssh -Wを使って定義します(%hがHostName、%pがPortに変換されます)。

~/.ssh/configへ追記
```
Host my-macmini
    HostName localhost
    Port 9999
    ProxyCommand ssh -W %h:%p my-ubuntu.irvine.jp
```

▼リスト5　こちら側の設定(「あちら側」にも対称に、ローカル接続のポート番号を9999、autosshの接続時のポートを9998としてトンネルを作っておく)

```
alias achira='ssh fumidai.irvine.jp -t ssh -p 9998 localhost'
alias psautossh='ps ax | grep autossh | grep -v grep'
alias tunnel='psautossh > /dev/null || autossh -f -M 0 -N -R 9999:localhost:22 fumidai.irvine.jp && psautossh'
tunnel
```

注6) ssh_config(5)、sshd_config(5)
注7) rcが存在していると、Xフォワードがうまくいきません。

加えて、接続先に、接続元の公開鍵を登録しておきます。これで、お互いが異なるプライベートネットワーク内にあるホスト間でscp、rsyncもできるようになります。

対象のホストにmy-macminiを指定できるようになる
```
his-centos: masa$ scp -r logfiles/ ↵
my-macmini:logpool/centos/
```

接続先に公開鍵が登録されていない場合
```
The authenticity of host '[localhost]:9999 ↵
(<no hostip for proxy command>)' can't be ↵
established.
ECDSA key fingerprint is SHA256:↵
n0vLDdl4FHthxUtbbZIEHVR093YQlJFMukJB2Y/MPig.
Are you sure you want to continue connecting ↵
(yes/no)? yes
Warning: Permanently added '[localhost]:9999' ↵
(ECDSA) to the list of known hosts.
Password:
```

sshをさらに便利にするコマンド

STEP UP! ssh-agent

秘密鍵には、パスフレーズを付けておくのが安全です。しかし、頻繁に(複数の)サーバとローカルで行き来しながら作業する際、冒頭から見てきたように毎回のパスフレーズ入力は省きたいときがあります。ssh-agentは、あらかじめパスフレーズを入力して秘密鍵を登録しておけば、サーバへの接続の際に登録済みの秘密鍵を取り出して使ってくれるしくみです。ssh-agentは複数の秘密鍵を扱うことができ、それぞれの鍵の登録のときに一度だけパスフレーズを入力するだけで、ssh-agentが動いている限り秘密鍵を解くパスフレーズの入力を省略することができます。

ssh-agentは、環境変数を使って実行状態を各プロセスとの間で管理・共有しますので、直接実行しないでください。連載第8回(2016年12月号)で確認したように、**子プロセスの環境変数は、今実行中のシェル(親プロセス)には反映されません**。evalによって実行すれば、ssh-agentプロセスが設定する環境変数を、現在実行中のシェルに反映することができます。

```
$ eval $(ssh-agent)
```

秘密鍵の登録には、**ssh-add**コマンドを使います。

```
$ ssh-add ~/.ssh/id_rsa
```

登録されている識別子のリストは、ssh-add -lで確認できます。情報の保存先は、${TMPDIR}/ssh-XXXXXXXXXX/agent.<ppid>です。この情報は実行者にしか閲覧できない権限になっています。

登録されている識別子を削除するには、次のように-dオプションで削除したい秘密鍵を指定します。ssh-add -Dは、すべての識別子の情報を削除します。

```
$ ssh-add -d ~/.ssh/id_rsa
```

ssh-agentの停止
```
$ eval $(ssh-agent -k)
```

ssh-agentを停止するときもevalを使います。この処理をせず、現在実行中のシェルからログアウトしてもssh-agentは動き続けます。うっかり接続が切れてしまっても、再度接続して、環境変数SSH_AUTH_SOCKを、接続前の値に設定すればssh-agentサービスを切断前の状態と同じように使うことができます。意図せずssh-agentを残してしまった場合は、ssh-agentが動いているホストに再度接続して、killコマンドでプロセスを停止しましょう。

evalを使ったssh-agentの扱いは、.profileや.logoutに記述する場合に向いています。専有できるワークステーションにログインして、作業し続けるなら便利です。一方で、一時的な作業などでは、エージェントプロセスや一時ファイルが残らないように管理するのは面倒です。そのような場面では、ssh-agentを親プロセスとして、シェルを実行することができます。これなら、シェルを終了(exit)すればエージェントも一緒に停止します。

```
$ ssh-agent bash
```

Ubuntu の Unity は、agent 機能を gnome-keyring によって提供する

Ubuntu の Unity は、gnome-keyring というプログラムによって鍵の管理をしています。ターミナル内で ssh を実行する際、秘密鍵にパスフレーズが付けてあると gnome-keyring のダイアログが表示され、そこにパスフレーズを入力します。パスフレーズ入力時のダイアログのチェックボックスにチェックしておけば、永続的に記憶させておくこともできます。

gnome-keyring ではなく、ssh-agent を使いたいときは、次のように Keyring デーモンの SSH コンポーネントを無効化します。

```
# ln -sf /dev/null /etc/xdg/autostart/gnome-keyring-ssh.desktop
```

ssh-agent とポートフォワードの組み合わせの例

my-mac → my-ubuntu → her-centos と、ssh-agent の機能を使って接続してみましょう。秘密鍵の情報はフォワードされるので、my-ubuntu の公開鍵を her-centos に登録しておく必要はありません。中継先サーバ(my-ubuntu)では、sshd が ssh-agent の役割を担います。接続の開始は、ssh に -A オプションを指定して ssh-agent を使っていることを明示します(**図1**)。~/.ssh/config に次のように記述しておけば、her-centos に接続するとき、-A オプションを省略できます。

```
Host her-centos
  ForwardAgent yes
```

ssh-copy-id

ssh-copy-id を使うと、scp して authorized_keys に追記、といった一連の処理を一度にできます。

```
$ ssh-copy-id -i ~/.ssh/id_rsa.my-mac her-ubuntu
```

指定する公開鍵のファイル名は .pub で終わるようにします。-i オプションに続くファイル名に、.pub を付けずに指定しても、.pub がファイル名の後ろに補完されます。上の例で ssh-copy-id は、id_rsa.my-mac.pub というファイルが存在していると期待します。

ssh-copy-id でも、公開鍵が登録される前の接続には、パスワード認証が用いられることに注意しましょう。

ssh-keyscan

連載第15回の ssh(その1)で解説したように、

▼図1　参考：ssh-agent を使った踏み台アクセス一連の流れ

```
ore@my-mac:~$ eval $(ssh-agent)
Agent pid 4041
ore@my-mac:~$ printenv SSH_AUTH_SOCK
/tmp/ssh-rFynS73CXxTD/agent.4041
ore@my-mac:~$ ssh-add ~/.ssh/id_rsa
Enter passphrase for /home/ore/.ssh/id_rsa:
Identity added: /home/ore/.ssh/id_rsa (/home/ore/.ssh/id_rsa)
ore@my-mac:~$ ssh-add -l
4096 92:89:82:b9:cb:82:9e:97:1a:3b:3d:47:d8:62:70:72 /home/ore/.ssh/id_rsa (RSA)

ore@my-mac:~$ ssh -A my-ubuntu
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-92-generic x86_64)

ore@my-ubuntu:~$ ssh her-centos
Last login: Thu Aug 24 12:25:45 2017 from his-solaris

[ore@her-centos ~]$ exit
logout
Connection to her-centos
ore@my-ubuntu:~$ exit
logout
Connection to my-ubuntu closed.
ore@my-mac:~$

ore@my-mac:~$ eval $(ssh-agent -k)
Agent pid 4042 killed
```

sshは信頼済みの接続先マシンのフィンガープリントを~/.ssh/known_hostsに保存して、再度接続するときに得るフィンガープリントと照合して、サーバが変更になっていることや、なりすましているということを判定します。

新しいクライアントを追加したとき、SSHサーバが移設したりIPアドレスが変更になったりしたときなど、複数の接続先があれば、それぞれにsshで実際にログインしてフィンガープリントを登録するのはたいへんです。こんなときに役立つのが、公開鍵を収集してくれるssh-keyscanです。接続先が複数ある場合でも、複数ホストに並行で接続してフィンガープリントを効率よく取得します。sshdをクライアント側で立ち上げておけば、先述のホストベース認証でサーバ側にクライアントのホスト公開鍵を登録するときにも便利です。たとえば図2のように実行します。-Hオプションは、フィンガープリントをハッシュ化して出力するオプションです。

今回の技術が活躍するところ

sshに主要な認証方式(パスワード認証、公開鍵認証、ホストベース認証)のうち、ホストベース認証を解説しました。また、(比較的)安全に複数のサーバを行き来するのに便利な機能やしくみを紹介しました。

STEP UP! SSHと暗号系

パスワードによる秘密鍵暗号系は、暗号化にも復号にも共通の秘密鍵(パスワード)を使います。パスワードが外部に漏れることは、すなわち丸裸になるということです。

一方、公開鍵暗号系では、公開鍵を使って暗号した情報を秘密鍵によって復号します。復号は秘密鍵でしかできず、公開鍵を持っていても復号できないことと、公開鍵から秘密鍵を推察できないしくみになっていることがミソです。認証時には、公開鍵暗号系によって安全が確保できます。

しかし、公開鍵暗号系は処理のための計算量が秘密鍵方式に比べて膨大で、大量の情報をやりとりするには適していません。そのため、共通に使う秘密鍵を公開鍵暗号系で送信し、データ通信はその鍵を使っておこなう方式をSSH1プロトコルで採用しました。SSH2プロトコルでは、SSH1よりも安全と考えられている鍵交換方式(Diffie-Hellman鍵共有)によって、それぞれのホストが外部に情報を漏らさずに共通鍵を入手します。

▼図2 a-serverとb-serverからフィンガープリントを取得してknown_hostsに追記する例

```
$ ssh-keyscan -H -t rsa a-server.gihyo.jp
b-server.gihyo.jp >> ~/.ssh/known_hosts
```

最後に、sshを安全に運用するための基本プラクティスをまとめておきます。

1 パスワード認証を避け、公開鍵認証を使う

公開鍵の受け渡し、フィンガープリントの受け入れができて、公開鍵認証ができるようになったら、パスワード認証によるログインを禁止して、公開鍵認証だけを承認すれば、安全度が高まります。

2 ファイアウォールによって、sshへの接続元を制限する

ファイアウォールやTcpwrapperで、sshサービスへのアクセス元を制限できれば、安全度が上がります。

3 デフォルトポートを使わない

インターネットに露出しているホストでは、よく知られた22番ポートをそのまま使うと、見知らぬ訪問客がたくさん来ます。可能なら、sshデフォルトの22番ポートでのサービス提供をやめて、別のポートでsshdを起動すれば安全度が少し高まります。

次回について

次回は、テキスト処理(その4)を予定しています。SD

今回の確認コマンド

【manで調べるもの(括弧内はセクション番号)】
ssh(1), ssh-keygen(1), sshd(8), ssh-keysign(8), ssh_config(5), sshd_config(5), ruby(1), gem(1), scp(1), rsync(1), ssh-agent(1), ssh-add(1), ssh-keyscan(1), ssh-copy-id(1), 【Linuxのみ】setsid(1)

第67回

btrfsにおける新しい空き領域管理方法 free space b-tree

Text：青田 直大　AOTA Naohiro

btrfsの新しいspace cache

ディスク領域の管理は、ファイルシステムの重要な仕事の1つです。ディスク上のあるブロックが使用されているのかどうか、どのように使用されているのかがわからなければ、データを適切に読み書きすることはできません。

ここで「どの領域をどのように使っているか」だけでなく「どの領域が空いているか」を知っていることが重要です。同一のファイル内のデータはなるべくシーケンシャルに置いたほうがパフォーマンスが向上します。そういうときには、今書こうとしているデータがすっぽり入る空き領域を見つけたくなります。この探索を効率的に行うためには、ディスク上に空き領域の情報を置いておく必要があります。

実際にファイルシステムはディスク上でどのようにデータ領域を管理しているのでしょうか。Ext4ではbitmapを使ってデータ領域を管理します。すなわち、bitが立っていれば対応するブロックを使用している、立っていなければ使用していないと、「使っていること」と「使っていないこと」を同時に管理していることになります。

XFSでは、2種類のfree space B+treeを使って空き領域を管理しています。どちらのtreeも、ノードには「どのアドレスからどれだけのブロックが空き領域なのか」を記録しています。2つの違いは、アドレスでソートされているか、サイズでソートされているかの違いだけです。この2つのツリーがあることで、ファイルに追記した場合には以前のアドレスの近くを探すことも、あるいは新しいファイルにはそのサイズに合うような空き領域を探すことも、どちらの作業も効率的に行えるようになっています。このようにXFSは、直接的には「使っていないこと」だけを管理しています。

では、btrfsではどのように空き領域を管理しているのでしょうか。実は、btrfsはもともとは直接空き領域を管理するデータ構造をディスク上には持っていませんでした。btrfsでは、ディスクの使用状況を適宜スキャンして、そこからメモリ上に空き領域情報を構築していました。すなわちbtrfsは、ディスク上の「使っていること」だけを管理していたのです。

一度スキャンしてしまえばメモリ上に空き領域情報があるとはいえ、ディスクのあちこちを読んで、空き容量管理のキャッシュを構築するのはたいへんなことです。そこで、Linux 2.6.37からこのキャッシュをディスクに書き出すよう

になりました。さらに、Linux 4.5からは、free space treeといってキャッシュの書き出しの方法が変わりました。今回は、btrfsの空き領域管理を、メモリ上のキャッシュ、ディスクへの書き込み、そしてfree space treeの順番に見ていきます。

extent空間とBlock Group

本題に入る前にbtrfsのファイルがどのようにディスクまでマッピングされるかを通して、btrfsで使われている用語を紹介します。

多くのファイルシステムでは、**図1**のようにファイルデータを直接ディスクにマッピングしています。すなわち(**inode番号，ファイル内offset，サイズ，パーティション上のアドレス**)のようなマップ情報を保持しています。

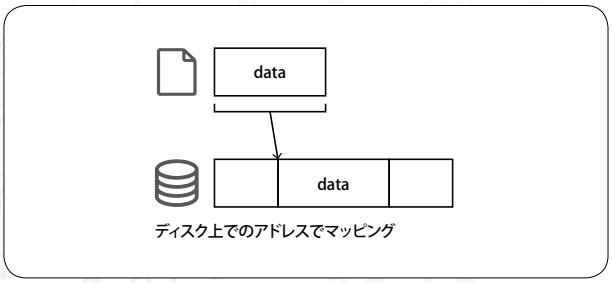
このような、ディスクへの直接のマッピングをbtrfsでも使えるでしょうか？ 理論的にできないことはありませんが、効率が悪くなったり、将来的に扱いにくくなります。たとえば、btrfsでは当初RAID1をサポートしていました。RAID1を構成するためには2つのデバイスが必要となります。もし直接のマッピングを行うなら(**inode，ファイルoffset，サイズ，ディスク1のアドレス，(optional)ディスク2のアドレス**)といったマップ情報を持つことになるでしょう。当然ながら、この方法ではRAID1を使わないシステムで、最後のアドレス部分の容量を無駄にすることになります。さらには、もしこんなマッピングをしていれば、RAID5/6対応するには、エントリを追加することになります。すると、すべてのマップ情報の形式を書き換える必要が出てくるなど、機能拡張への柔軟性が失われてしまいます。

そこで、btrfsではextent空間という最大2^{64}byteの巨大な仮想アドレス空間を導入しています。btrfsのファイルデータは一度extent空間にマップされ、さらにextent空間上の領域が実際のディスクにマップされるというように、2段階のマッピングをとっています(**図2**)。このとき、使用領域ごとにマッピングをするのではなく、extent空間を1GBごとなどの、ある程度の領域に切り分けて、その単位でディスクへのマッピングを行います。このときのextent空間上の領域をBlock Groupと言い、ディスク上の領域をchunkと呼びます。すなわち、ファイルデータは、まず(**inode番号，ファイル内offset，サイズ，extent空間上のアドレス**)のようにextent空間にマップされ、その領域が属するBlock Groupが(**extent空間上のアドレス，Block Groupのサイズ，ディスクのID，ディスク上のアドレス……**)という形式でBlock GroupがChunkにマップされます。

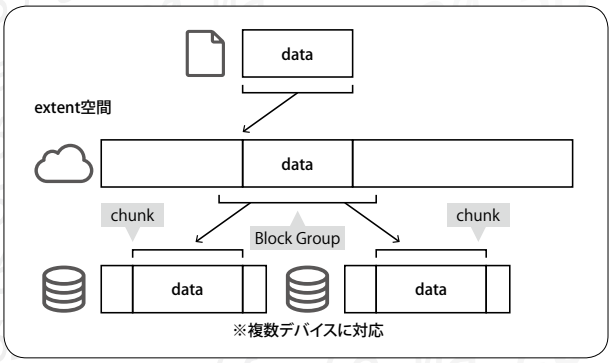
extent tree

extent treeは、btrfsにおいてextent空間の使用領域を管理するB-treeです。このtree上にextent空間上で連続した1つの使用領域につき、1つ(extent空間上のアドレス、EXTENT_ITEM

▼図1　多くのファイルシステムでのマッピング



▼図2　Btrfsのマッピング



(定数)、領域のサイズ)をキーとするextentアイテムが作られ、そこにextentの管理情報が保持されます。たとえば、この領域がどのファイルから参照されているのか、といった情報が入っています。

このツリーのアイテムを読んでいくことで、あるBlock Group中で使用されている領域がわかります。そこで参照されていない部分を空き領域としてfree space cacheを構築します。

free space cache

free space cacheは、前述したようにextentツリーを読んでメモリ上に構築される空き領域を管理する赤黒木[注1]です。このツリーは、Block Group1つにつき、1つ作られます。基本的には、ツリーのノード1つが1つの空き領域に対応し、データとして空き領域の開始アドレスとサイズとを持ちます。

ただし、この方法だけでは空き領域が断片化したときに、メモリ効率が悪化してしまいます。たとえばある128MBの領域が、最初の4KBは使っている、次の4KBは使っていない、その次はまた使っている……というように使用中の4KBと使用していない4KBが交互に存在する状態になったとしましょう。この128MBには16,384個の空き領域が存在することになります。1つのノードが72byteですので、この領域を表現するために1MB以上のメモリを割くことになります。

このように断片化してしまった領域を、より効率的に管理するため、free space cacheは先ほどの連続領域ノード(extentノード)に加えて、bitmapノードを併用します。bitmapのサイズが4KBであり、1つのbitがbtrfsの論理ブロックサイズ(基本的にPAGE_SIZE =ほとんどの場合4KB)分の空白領域を表現します。したがって、1つのbitmapによって4KB×8×4KB=128MB分の領域分の空き状況を管理できます。すなわち、先ほどのひどく断片化した128MBの領域は、bitmapノードを使って、ノードのサイズ:72byte+bitmapのサイズ:4KB分のメモリで表現され、extentノードだけの場合に比べて消費メモリが280倍以上も小さくなります。

cacheの保存 space_cache

さて、このようにextent treeを読んでいくことで、Block Groupごとに1つfree space cacheが作られるわけですが、cacheの製作にはどの程度のI/Oが必要でしょうか。extentアイテムのサイズはスナップショットなどの使用状況によって変わりますし、そもそもファイルシステムの使用状況によって1つのBlock Groupの中のextentの数は変わってきます。

したがって、一概に言うことはできませんが、一例として筆者が日常的に使用しているノートパソコンで見てみることにします。このファイルシステムには471のBlock Groupがあり、36,758個のextent treeのleafがあります。平均すれば、1つのcacheを作るのに78個のleafを読む必要があるといえます。1つのleafは(デフォルトでは)16KBですので、おおよそ1.2MBほどのI/Oが必要です。しかも、これらのleafはツリー上で隣接しているからといって、extent空間でも、当然ディスク上でも連続しているわけではないので、コストはより高くなります。

そこで、一度作ったcacheをディスクに保存しておいて、extent treeをスキャンする手間をはぶくという機能がLinux 2.6.37で追加され、今ではデフォルトでこの機能が使われるようになっています。

では、このcacheはディスクのどこに書かれているのでしょうか。btrfsには"root tree"という、btrfs内のすべてのツリーのrootを管理するtreeがあります。ここに、たとえばextent treeのrootがextent空間上のどこにあるのかを記録したり、snapshotの情報を記録したりしています。ここにはファイルシステムのrootディレクトリ

注1) https://en.wikipedia.org/wiki/Red-black_tree

のinodeも記録されています。btrfsは、このtreeの中にcacheを保存する「見えないファイル」(以後cache fileと呼ぶ)用のinodeを作成します。また、このtreeの中に、cacheのmetadataを保持するアイテムも記録します。

具体的に、treeをダンプして見てみましょう。**図3**のように"btrfs.img"上のroot treeをダンプします。cacheのmetadataは、**(FREE_SPACE, UNTYPED, <Block Groupの開始アドレス>)**というキーに保管されています。**図3**でいえば、"item 61"から"item 64"ですね。このアイテムには、"location"にcacheを保存しているinodeの位置が記録されています。ほかにもcacheのgeneration、記録されているノードの数、bitmapノードの数が保存されています。generationというのは、btrfsの1回のトランザクションのたびにインクリメントされる数値です。後述するようにgenerationは、cache fileにも記録されています。すなわち、この2つのgenerationが一致しなければ、システムのクラッシュなどでcacheが正しく書き出せていないことを意味しています。

"item 64"に対応するcache fileについて見ていきましょう。cache fileのinodeは**(302 INODE_ITEM 0)**であり、すなわちダンプ中の"item 59"であることがわかります(**図3**)。ここにはcache fileのinode情報が記録されています。注目したいのは、flagsの部分です。**"NODATASUM|NODATACOW"**とchecksumもCopy-on-Writeも行わない、とあまりbtrfsらしくはないデータ領域の確保の方法になっています。これにはもちろん理由があります。checksumを有効にしていると、cache fileの更新によってchecksumの書き換えが発生します。このchecksumを保存しているmetadata領域もまた、何かしらのBlock Groupに所属しています。すると、checksumを保存するために、Block Groupから新しい領域を使うかもしれません。もし新しい領域を使えば、今度はこちらのBlockGroupのcacheを更新することになります。そして、またchecksumが更新され……というように書き換えが連鎖してしまいます。そこで通常のchecksumは無効化され、後述するようにcache fileの中にCRC (Cyclic Redundancy

▼図3　cacheが保管されるinodeのダンプ

```
$ btrfs-debug-tree -t ROOT_TREE btrfs.img
...
        item 59 key (302 INODE_ITEM 0) itemoff 3271 itemsize 160
                inode generation 79 transid 79 size 262144 nbytes 786432
                block group 0 mode 100600 links 1 uid 0 gid 0 rdev 0
                sequence 27 flags 0x3(NODATASUM|NODATACOW|NOCOMPRESS|PREALLOC)
                atime 0.0 (1970-01-01 09:00:00)
                ctime 1506267367.5870589 (2017-09-25 00:36:07)
                mtime 0.0 (1970-01-01 09:00:00)
                otime 0.0 (1970-01-01 09:00:00)
        item 60 key (302 EXTENT_DATA 0) itemoff 3218 itemsize 53
                generation 79 type 1 (regular)
                extent data disk byte 4086525952 nr 262144
                extent data offset 0 nr 262144 ram 262144
                extent compression 0 (none)
        item 61 key (FREE_SPACE UNTYPED 29360128) itemoff 3177 itemsize 41
                location key (256 INODE_ITEM 0)
                cache generation 79 entries 210 bitmaps 3
        item 62 key (FREE_SPACE UNTYPED 1103101952) itemoff 3136 itemsize 41
                location key (300 INODE_ITEM 0)
                cache generation 77 entries 0 bitmaps 0
        item 63 key (FREE_SPACE UNTYPED 2176843776) itemoff 3095 itemsize 41
                location key (301 INODE_ITEM 0)
                cache generation 79 entries 1 bitmaps 0
        item 64 key (FREE_SPACE UNTYPED 3250585600) itemoff 3054 itemsize 41
                location key (302 INODE_ITEM 0)
                cache generation 79 entries 2 bitmaps 0
```

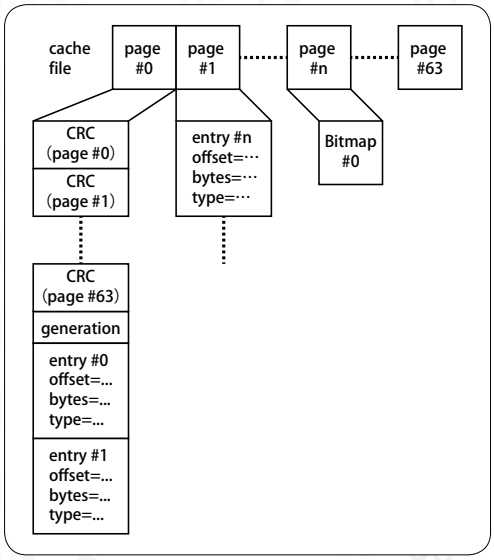
Check)を持つようにしています。

また、inodeのsizeが262,144 = 256KBと、ノード2個、bitmap 0個を保存するには大き過ぎる領域がとられているように思います。これは、Block Groupのcacheを保存するのに十分なデータ領域を常に確保しておくためです。また、nbytesが786,432とずいぶんと大きくなっています。本来このファイルがディスク上で実際に確保している領域のサイズを示し、cache fileならsizeと同じ262,144になっているはずです。これはおそらく、cacheの更新時にsizeだけをリセットして、nbytesをリセットし忘れているバグでしょう。実際、786,432は262,144の整数倍になっており、cacheが書かれるたびに262,144ずつ増えてしまっているのだと思われます。

INODE_ITEMの次"item 60"に、cache fileのデータ位置が記録されています。すなわち、extent空間上の4,086,525,952byte目から262,144byteの領域にcache fileのデータが保存されています。cache fileの構造を見ていきましょう。

cache fileの先頭には前述したようにcheck sumを保存するエリアがあります(**図4**)。ここは、cacheデータの1ページ分ごとにCRCが計算され保存されています。そのあとに、cacheのgenerationが保存され、さらにメモリ上のcache treeのノードごとに1つのエントリが記録されます。エントリには、そのノードが担当する領域のアドレスとサイズ、そしてbitmapかどうかが記録されます。実際のbitmapのデータは、エントリをすべて記録した後ろに保存されます。bitmapのサイズはすべて1ページですので、ここはbitmapを単純に並べているだけですね。

▼図4　ディスク上のcacheの構造



このように、cache fileは事前に予約した256KBなどの連続領域に、一度構築したcacheを書いてしまいます。extent treeからの読み込みであれば、最大70ヵ所ほどにばらついた合計1.2MBのI/Oを必要としたのに対して、1ヵ所から256KB読み込んでくればいいので、かなりコストが削減されています。

free space cache v2

space cacheの導入により、一度作ったcacheを256KBにぐっと固めて読み込めるようになりました。ところが、巨大で頻繁に更新されるようなファイルシステムでは、このcacheが逆にパフォーマンスを悪化させてしまいます。

Facebook社では、数十TBもの巨大、かつ更新の激しいbtrfsを使っているようです。データ用のBlock Groupが1GBですので、10TBのbtrfsにはBlock Groupが1万個近くは作られていることになります。これらがすべて更新されたとすれば、cacheの書き込みで合計で2GB以上の書き込みが1万ヵ所に発生することになります。さすがに、ここまでいくことはないでしょうが、cache fileは少しでも更新があれば、cache fileの256KBをまるっと書き換えてしまう性質を持つため、大規模なファイルシステムではI/O負荷が大きくなってしまうのです。さらに、この書き込みがトランザクションのコミットのタイミングで起きてしまうと、その間ファイルシステムは新たな書き込みを止めてしまいます。Facebookでは、数秒間I/Oが止まってしまうこともあったようです。

cache fileの問題は、ファイルデータとしてほぼ構造なくダンプするために、少しでも更新があればすべて書き出してしまう点、そしてBlock Groupごとに1つのファイルを使ってしまうという点でした。更新された部分だけを書き換え、複数のBlock Groupを一度に扱う……ということで結局free space treeという、やっぱりb-treeが導入されることになりました。

free space treeはいわば、extent treeを「反転」したb-treeです。extent treeが使っている領域を管理していたのに対して、free space treeは使っていない領域を管理します。

free space treeとextent treeとのダンプを見てみましょう(**図5**)。btrfsをspace_cache=v2のmount optionをつけてmountし、free space treeを作らせます。するとリストのようなダンプを取得できます。free space treeには3種類のアイテムがあります。1つは(**Block Groupの開始アドレス、FREE_SPACE_INFO、Block Groupのサイズ**)のキーを持つアイテムです。これは、そのBlock Group中の空き領域の個数、そしてこのBlock Groupでは空き領域をextentで表現しているのか、あるいはbitmapで表現するのかを記録しています。

残りの2つのアイテムは、**(空き領域の開始アドレス, FREE_SPACE_EXTENT(またはFREE_SPACE_BITMAP), 空き領域のサイズ)**というキーを持つアイテムです。こちら2つは、cache treeの場合と同様にextentとして、あるいはbitmapとして空き領域を示すことになります。

まず、**図5**中のFREE_SPACE_TREEのダンプのitem 4を見てください。29,360,128byte

▼図5　free space treeとextent treeのダンプ

```
$ mount -o space_cache=v2 btrfs.img /mnt/tmp; umount /mnt/tmp
$ btrfs-debug-tree -t FREE_SPACE_TREE btrfs.img
free space tree key (FREE_SPACE_TREE ROOT_ITEM 0)
leaf 32276480 items 376 free space 6835 generation 84 owner 10
leaf 32276480 flags 0x1(WRITTEN) backref revision 1
fs uuid b909e37e-ab55-412e-b20a-3eb643d5004b
chunk uuid 5da683f7-b332-4797-ab75-edb4db3c08b4
        item 0 key (12582912 FREE_SPACE_INFO 8388608) itemoff 16275 itemsize 8
                free space info extent count 0 flags 0
        item 1 key (20971520 FREE_SPACE_INFO 8388608) itemoff 16267 itemsize 8
                free space info extent count 1 flags 0
        item 2 key (20987904 FREE_SPACE_EXTENT 8372224) itemoff 16267 itemsize 0
                free space extent
        item 3 key (29360128 FREE_SPACE_INFO 1073741824) itemoff 16259 itemsize 8
                free space info extent count 364 flags 0
        item 4 key (29360128 FREE_SPACE_EXTENT 32768) itemoff 16259 itemsize 0
                free space extent
        item 5 key (29425664 FREE_SPACE_EXTENT 16384) itemoff 16259 itemsize 0
                free space extent
...
$ btrfs-debug-tree -t EXTENT_TREE btrfs.img
...
        item 0 key (29392896 METADATA_ITEM 0) itemoff 16250 itemsize 33
                extent refs 1 gen 36 flags TREE_BLOCK
                tree block skinny level 0
                tree block backref root 280
        item 1 key (29409280 METADATA_ITEM 0) itemoff 16217 itemsize 33
                extent refs 1 gen 47 flags TREE_BLOCK
                tree block skinny level 0
                tree block backref root 290
        item 2 key (29442048 METADATA_ITEM 0) itemoff 16184 itemsize 33
                extent refs 1 gen 4 flags TREE_BLOCK
                tree block skinny level 0
                tree block backref root 18446744073709551607
...
```

目から32,768byteが空いていることを示しています。29,360,128 + 32,768 = 29,392,896ということで、extent treeを見るとたしかにitem 0の部分に該当するMETADATA_ITEMがあります。**METADATA_ITEM**に割り当てられるextentのサイズはデフォルトで16KBです。すると、extent treeのitem 1とitem 2との間は32KB分あり、item 1に対応するextentの後ろに16KB分の空きスペースがあることが見てとれます。そこで、free space treeに戻ると、たしかに29,409,280+16KB=29,425,664の位置を示すFREE_SPACE_EXTENTアイテムのitem 5があるのが見てとれます。

free space b-treeの導入は、どの程度の効果があるのでしょうか。ここではFacebookによる実験結果を紹介します。この実験では、50TB分のBlock Groupに変更を加えて、コミットにかかる時間を計測しています。その結果、cache fileでは5秒から最悪の場合40秒ほどもかかっていたコミットが、b-treeによってcacheを保存しない場合とほぼ同等の0.3ミリ秒ほどしかかからなくなりました。一方、cache fileのように一ヵ所に固まったデータを読むのではなく、b-treeではあちこちのノードを読んでくるので、cacheの読み込みはcache fileの10ミリ秒ほどから30ミリ秒ほどと悪化を見せています。しかし、cacheの読み込みは最初の一度だけですので、これで残りのすべての書き込みが速くなるならなんの問題もないでしょう。また、cache fileのときは問題になっていたchecksumも、b-treeを使っているので既存のb-tree用のchecksum機構をそのまま活用できています。

まとめ

今回は、btrfsにおける新しい空き領域管理方法となるfree space b-treeを紹介しました。Facebookのように巨大なファイルシステムでなければ、なかなか差は出ないかもしれませんが、一度free space b-treeも試してはいかがでしょうか。SD

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第45回 wall de talk

ポケベル世代の俺様にはさっぱりわからんとぼける担当編集。とぼけているんじゃなくてボケてます。編集部高齢化の余波でナニがアレしてほげコロgeです。

to be Continued

ポケベル、CASIOのPHSでの発信者メッセージ、MMS、写メール、ICQ、MSN Messenger、Twitter、Google トーク、Facebookメッセンジャー、ハングアウト、LINE……とメッセージを交換するプラットフォームを渡り歩いてきました(前の方を知ってるほどオッサン度が高し)。Linuxサーバ内でも会話できるツールはいくつかありますが、その1つは`wall`コマンドです。ほら、`shutdown`するときに全部の端末に音と共にメッセージが来るアレです。同じサーバにログインして作業しているメンバー全員にメッセージを送るのに重宝するのでみんなも試してみましょう。ちなみに「`mesg n`」を実行した端末では`wall`メッセージを受け取らないようになります。

November 2017

NO.73

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

経験者だから話せるコミュニティ運営の深い話

今回は、7月に2回行ったコミュニティ運営セッションの模様をお届けします。

jus研究会 札幌大会

■ITコミュニティの運営を考える
【講師】小山哲志（jus）、法林浩之（jus）
【日時】2017年7月15日（土）14:00～14:45
【会場】札幌コンベンションセンター 207会議室

札幌にてITコミュニティの運営を考えるセッションを実施しました。参加者は9人でした。参加者を2グループに分けてグループディスカッションを行いました。その後、各グループから代表の方が議論の内容を報告しました。

片方のグループは、Sapporo.beamというElixirのコミュニティを運営している方から報告がありました。Sapporo.beamは参加者が数人ということもあり、毎週、カフェの片隅で会合を開いています。すでに170回ほどの開催実績があり、この手法を用いることで手軽にかつ継続的に会合を開けているとのことです。それから、「札幌も、ひと頃に比べるとコミュニティ活動が沈静化している」「東京からUターンで戻ってきた人から見ると、非IT寄りのコミュニティが少ない」という話題や、コミュニティの終わらせ方といった話題も出ました。

もう一方のグループからは、おもに参加者としてコミュニティに関わっている学生から報告がありました。初心者の立場から見るとITのコミュニティはまだまだハードルが高いと感じているが、受け入れる運営側はとくにハードルを上げているわけではないという話がありました。また、学生でコミュニティに参加している人はもともとのスキルが高いと思い込みがちですが、受け入れる側はスキルの高い人ばかりを歓迎しているわけではなく、多様な人を受け入れることが大事だと考えているようです。

今回はグループ分けの際に、コミュニティ運営経験のある人とない人が混在するように配慮してみました。その結果、どちらのグループも運営経験のない人からの質問に経験者が答えるという図式ができあがりました。表立って伝える機会の少ないコミュニティ運営者の苦労や配慮といったものが伝わる、良い機会になったのではないかと考えています。

jus定期総会併設勉強会

■ITコミュニティの運営を考える
【出演】岡田良太郎（OWASP）、
白石俊平（TechFeed／元html5jリーダー）、
須藤功平（OSS Gate）、関治之（Code for Japan）、
日高正博（DroidKaigi／techbooster）、
宮原徹（OSC事務局）
【司会】榎真治（jus/LibreOffice日本語チーム）、
法林浩之（jus）
【日時】2017年7月22日（土）15:30～17:30
【会場】サイボウズ㈱ 東京オフィス

jusでは毎年7月に定期総会を行っていますが、総会の併設行事として「ITコミュニティの運営を考える」のセッションを行いました。今回は6人のコミュ

ニティ運営者を招き、司会も含めて8人体制でディスカッションしました。参加者は34人でした。以下、登壇者から出されたお題と議論を紹介します。

■どういうコミュニティを目指して運営しているか？（日高さん）

目指す目標が明確なコミュニティとそうでないコミュニティがありますが、たとえばOSCは地方に点在するコミュニティをつなげるメタコミュニティ的なものを目指しています。Code forコミュニティも団体としての明確な目標はありませんが、各地域のIT人材をつなげるという点ではOSCと共通するものがあるようです。この話題から派生して、コミュニティ運営にどれぐらいの時間を割いているかや、DroidKaigiは若年層が多いといった話も出ました。

■継続の秘訣は？（須藤さん）

OSS Gateは「継続的にOSS開発者を輩出する」という目標を持っているため、継続するしくみ作りが大切と考えています。とくに今は各地で実施しているワークショップ運営を無理なく続けられる方法を模索しています。これに対してほかの登壇者からは、目的を明確に決め過ぎないことで技術や時代の移り変わりに対応できる、若者を重要な役割に登用することでコミュニティを引き継いでくれる人が育つ、運営メンバーを毎年大幅に入れ替えて新陳代謝を促す、などのアドバイスがありました。

■コミュニティ運営の失敗談を聞きたい（白石さん）

宮原さんからは、各地でOSCを立ち上げる中でうまくいかなかったケースとして、地元コミュニティの立ち上げにOSCが利用されただけでOSCとしては根づかなかった例が紹介されました。関さんからは、各地のCode forコミュニティを見ていく中で、リーダーに熱量があり過ぎるとメンバーがついていけなくなるという話がありました。この話題から派生して、コミュニティ運営においてルールはどこまで厳密に定めるべきかや、最近多くのイベントで定められている行動規範についての議論もありました。

■ダイバーシティについてどう思うか（関さん）

関さんによると、米国のCode forコミュニティはダイバーシティに対する意識が非常に高く、たとえば参加者層における白人・黒人などのバランスも考慮しているそうです。日本でもダイバーシティに対する関心は高まっていますが、IT業界は女性がもともと少ないのでバランスを取りにくく、作り込もうとすると手間がかかるという意見が出ました。また、コミュニティの本質は多様性ではなく同質性であり、たとえば学校や職場ではマイノリティの立場にある人がコミュニティに来ると仲間が見つかるのが良い点である、というコメントもありました。

■勉強会多過ぎない？（宮原さん）

以前に比べて勉強会を実施する環境が整備され、開催数が増えたのは良いことだが、多くなり過ぎて参加者にとっては供給過剰ではないかという問題提起です。これに関しては、参加登録者のキャンセル率が高くなり運営側としては人数が読みづらい、勉強会で話を聞くだけで帰ってしまいコミュニティに入らない人が増えている、PHP勉強会では参加者も全員自己紹介をすることでコミュニティへの加入度を高めている、などの話がありました。

■最近、変わったことは？（岡田さん）

宮原さんからは、自分が興味を持つ範囲の外側を見ない人が増えたためにOSCのようなメタコミュニティ的なイベントの集客が頭打ちになっているという話がありました。一方で、日高さんが運営する技術書典では業務と関係なさそうな冊子が売れるという現象があります。参加者は即効性を求めるイベントと趣味で楽しむイベントを分けて考えているのかもしれません。また、イベントを通して技術者を採用したい企業が増えたという話題も出ました。

◆ ◆ ◆

今回は登壇者が多く、大きなコミュニティや年数の長いコミュニティを運営している方もおり、豊富な経験に基づくコメントが多く聞けました。座談会的な雰囲気もあり、盛り上がって良かったです。SD

温故知新
IT むかしばなし

第71回

初期のWindows ～誕生からWindows 95登場までの困難な道のり～

速水 祐（はやみ ゆう） http://zob.club/ twitter @yyhayami

はじめに

Windowsは、現在でもPCの主流なOSの地位を確保し続けています。1995年のWindows 95の登場からその地位が築かれたのですが、そこまでに至る9年間の過程は、混乱を極めた苦難の道のりでした。今回は、初期のWindowsについてのお話をしましょう。

Windowsの誕生

延期に延期を重ねて、Windows 1.0は1985年の11月に完成します。ここからWindowsの困難な道のりが始まります(**図1**)。

長い開発期間をかけたWindows 1.0ですが、複数のウィンドウを重ねることができない縦横に並べるだけのタイルウィンドウでした。さらにi8086/8088 CPUのメモリ領域の限界である1MB(MS-DOSが利用できるのは640KB)の壁とIPC(プロセス間通信)のグラフィック環境の問題から、見るにも操作するにも苦痛を伴うようなものでしかなかったのです。

その後、実用的なGUI環境の実現を目指して開発は続けられ、1987年11月にWindows 2.0が登場します。Windows 2.0は、オーバーラップウィンドウを実現しており、操作性の快適性やスピードも1.0に比べれば大幅にアップしたものの、まだ満足できるものではありませんでした。

Windows 2.0は、リアルモードで動作するため、高速・高機能のi80286 CPUで動かしても1MB以上のメモリ領域は使用できなかったのですが、EMSメモリによるメモリの拡張はできたため、なんとか複数のアプリケーションの動作が可能になりました。そのためMacintoshで動いていた憧れの表計算ソフト「Excel」がWindows上で動作するようになりました。しかし、Excelを動かすためには拡張メモリ以外にもハードディスクなどの高価なハードウェアを追加する必要があり、その対価に対して十分な操作性を提供できるものではありませんでした。

Windows 2.1の登場

Windows 2.0が発売されて半年後に、Windows 2.1がリリースされます。2.0に比べて、よりメモリ管理機能を充実させて、i80286向けを「Windows/286 2.1」、i80386向けを「Windows/386 2.1」と改称し、2つのパッケージとして販売されました。

IBM-PC版のWindows 2.1/286は、i8086/8088でも動作可能であり、リアルモードで動作します。XMS(eXtended Memory Specification：メモリ拡張管理方法)の規格に含まれるHMA(High Memory Area)へ対応することで、i80286であれば1MBの上64KBのメモリ領域の使用ができるようになりました。しかし、この時点では、1MB＋64KB以上の上位のプロテクトメモリであるEMB(Extended Memory Block)は使えなかったようです。

もう1つのWindows/386は、Windows 2.0の登場のあと、数ヵ月でWindows/386 2.0として登場しています。実用的なOSとしては初めて仮想86モードを使い、DOSアプリを複数同時に動作させ、ほかのプログラムやメモリの保護も行うことができる画期的なものでした。内部的にはi80286互換の、16bitのi80386プロテクトモードで動作するカーネル部分も初めて作られたのです。

Windows 3.0の登場

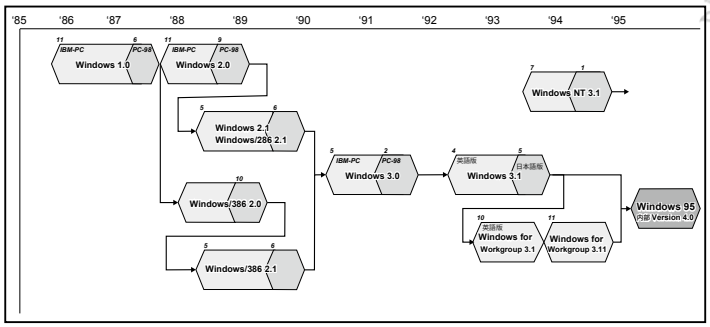
2年後の1990年になると、Windows/286 と Windows/386を統合し、GUIを洗練化して操作性が大幅に高まったWindows 3.0が登場します。Windows 3.0のGUI操作が、現在のWindowsの操作とほぼ同じになり、その後のWindowsの基本になったバージョンでした。Windows 3.0は、次の3つのモードで動作します。

①リアルモード：i8086/8088で動作するためのもの
②スタンダードモード：i80286のプロテクトモードで動作する
③エンハンストモード：i80386の仮想86モードとプロテクトモードを効果的に利用

スタンダードは16bitプロテクトモードで動作します。ついにメインのWindowsもプロテクトモードで動くことになりました。しかしこのモードでは、DOSアプリケーションを動作させるためにはリアルモードに切り替えることになり、MS-DOSのシステムコールもリアルモードになってから利用しているため、安定な動作を期待することはできなかったのです。それに対してエンハンストモードは、DOSアプリケーションもシステムコールも仮想86モードで実行するため、安定度もマルチタスクとしての動作もはるかに優れていると思われました。

1991年のパソコン雑誌では大々的にWindows 3.0を取り上げ、時代は完全にWindowsになるような記事が満載でしたが、想定されたような安定した動作ではなく、当時のPCでは快適と言える操作スピードが実現できなかったため、国内での利用は広がらず、1992年に入るとWindowsの記事は影を潜めます。

▼図1　Windowsの道のり



Windows 3.1

1993年に日本語Windows 3.1が登場します。PC-9801版のNEC製Windowsだけでなく、Microsoft社からDOS/V版の日本語Windows 3.1も発売されたのです。

日本語Windows 3.1は、見た目に3.0と同様ですが、i80386の使用を前提とした[注1]もので、エンハンストモードで動くWindowsでした。MS-DOSのシステムコールを利用していたディスクアクセス処理は、プロテクトモードの処理となり、内部システムの大幅なビルドアップにより、安定度が格段に上がり、パソコンの高速化と相まって、快適な動作が実現できたのです。

注1）英語版Windows 3.1は、スタンダードモードを備え、i80286での動作も可能。

そのため日本でも一気にWindowsの普及が広がります。

Windows for Workgroup

海外ではWindows 3.1にネットワーク機能を追加[注2]して、ネットワーク上で複数の端末で作業を行うことを狙ったWindows for Workgroupが登場します。Windows for Workgroupは日本語化されなかったのですが、1994年当時、DOS/Vユーザは、WIN/V[注3]を使って日本語化して利用していました。筆者も使っており、付属しているネットワーク対応のトランプゲーム「ハーツ」による4人でのネットワーク対戦や複数人チャットでは複数で同じ作業を行っているのだという実感を味わいました。

約9年間のバージョンアップを繰り返す困難な道のりのあと、標準OSの地位に導いたWindows 95の登場となります。SD

注2）Windows 3.1でネットワークを利用するためには、LAN ManagerやTCP/IPスタック・アプリケーションなどのLAN構成ソフトウェアのインストールが必要。
注3）西川和久氏が率いるC.F.Computingによって発売された、英語版Windowsを日本語化するソフトウェアで、かな漢字変換（IME）、漢字True Type/WIFEフォントが使用可能になる。

あなたのスキルは社会に役立つ

2011年3月11日の東日本大震災発生の直後にHack For Japanは発足しました。今後発生しうる災害に対して過去の経験を活かすためにも、エンジニアがつながり続けるためのコミュニティとして継続しています。防災や減災、被災地の活性化や人材育成など、「エンジニアができる社会貢献」をテーマにした記事をお届けします。

第71回

第6回 石巻ハッカソン

●Hack For Japanスタッフ　高橋 憲一（たかはし けんいち）　Twitter @ken1_taka

2017年7月28日から30日にかけて毎年夏恒例の石巻ハッカソンが開催されました。開発を通して若者がITに興味を持ったり、自分の夢を持ったりするきっかけになってほしいという趣旨で2012年に始まり、今回で6回目を数えるこのハッカソン、毎回会えるおなじみの顔ぶれだけでなく、初めて参加するという人の比率が高かったことが特徴的です。石巻市がある宮城県内のみなさんはもちろんのこと、福島や東京などさまざまな地域から100人を超える方が参加しました。

今回のテーマは「ひみつ道具」

石巻ハッカソンでは毎回テーマが設定され、それに沿うものを開発します。これまでは「海」、「青春」というテーマが設定されてきたのですが、今回のテーマは「ひみつ道具」です。ひみつ道具というと、やはりあの青色の猫型ロボットがポケットから出してくれるいろいろな道具を思い浮かべてしまう人が多かったようです。主催者側の意図もまさにそこで、あなたの夢のひみつ道具をITで表現しましょうというものでした。

ITブートキャンプ部門

プログラミング初心者のための部門で、この部門が用意されていることは石巻ハッカソンの伝統といっても良いでしょう。小学生のための部門とおもに高校生のための部門に分かれて進められました。小学生は始めにOzobot[注1]という、紙の上に線を描くとその上を線に沿って動く小さなロボット型のプログラミング教材を用いて、まずは興味を持ってもらうことが優先されました。そのあとに、Scratch JrというScratchよりも文字を減らしてイラストアイコンだけでプログラミングを始められるビジュアルプログラミングツールが使われていました。

高校生部門で使われたのは2012年の第1回目のときから使われているCorona SDK[注2]です。講師の指導のもとで基礎から始めて、最終日にはハッカソン部門と同様に自分のアイデアで作ったものを発表してもらいました。Corona SDKには物理エンジンが組み込まれており、2Dのアニメーションは比較的手軽に実現できるようになっているため、高校生たちの自由な発想でさまざまな作品ができあがりました。

ハッカソン部門

初日にアイデアピッチとチームビルディングを行い、全部で19のチームで開発がスタートしました。

大人げない人たち

とくにそのようなカテゴリがあるわけではないのですが、なりふりかまわず高い技術力でぶっちぎる大人たちを、いつの間にかそう呼ぶようになりました。筆者の場合、3回目までは初心者のサポートに

注1　https://www.ozobot.jp/
注2　https://coronalabs.com/

奔走していたのですが、4回目からは「そろそろ自分もガチで開発したい」と思い、年に一度の石巻ハッカソンだからとハードルを高く設定して臨むようになりました。この大人げない人たちはそうすることで、このハッカソンに参加する大人の役目でもある「かっこいい背中」を若者に見せているつもりです。後述する「ヤバい賞」の常連で、大人げない人たちの筆頭のような存在の友人とは「ハードルを高く設定し、短時間で必死に開発するのはエンジニアにとって筋トレのようなもの」と話しています。

恒例のお昼のカレー

2日目のお昼ご飯にはカレーが提供されるのが毎年の恒例となっており、「開発はできないけどカレー作りのお手伝いとして参加します！」という方もいます。今年はシーフードカレーで、石巻の名産であるホヤが入っていました（**写真1**）。石巻は何度も訪れており、この地の美味しい食材を愛してやまない筆者ではありますが、実はホヤだけは苦手なのです。しかし、このカレーはとても美味しく食べることができました。

成果発表

最終日には各部門ごとに成果発表が行われ、とくにすばらしい成果を上げた作品には賞が贈られました。

ITブートキャンプ高校生部門

12人からの発表があり、優秀賞が2名に贈られました。

●アプリ名：大胸筋

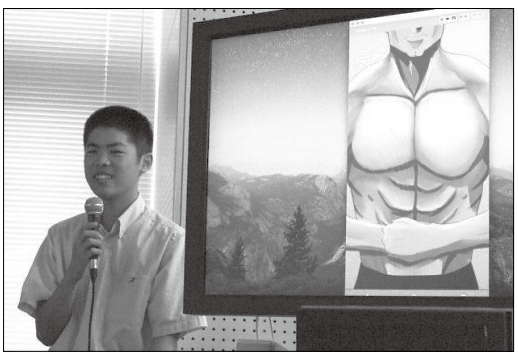
石巻工業高校2年生の作品で、大胸筋を動かしてそのすばらしさを知ろうという意図で作られたアプリです（**写真2**）。起動するとBGMが流れ、タップすると効果音とともに表示された絵の大胸筋が動きます。アプリの完成度の高さとインパクトの強さを評価されての受賞となりました。普段意識していないけれど大胸筋はいろいろな場面で使われるので、実はひみつ道具なのではないかと思ったという着眼点がおもしろく、会場は爆笑の渦に包まれていました。

●アプリ名：ブレークアバウトゼロ

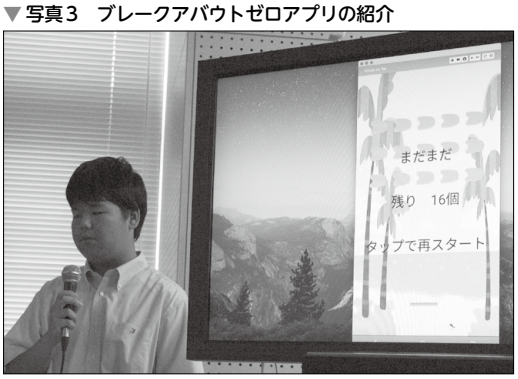
石巻工業高校1年生の作品で、ブロック崩しふうのゲームです（**写真3**）。失敗するとガラスの割れる音が出るなど、効果音にも凝ったものになっています。すべてのブロックを消すことができた際のコン

▼写真1　石巻名産のホヤ入りカレー

▼写真2　大胸筋アプリの紹介



▼写真3　ブレークアバウトゼロアプリの紹介



プリート画面への遷移や、ゲームオーバー時にすべて消せなかった場合の残りのブロック数表示などもしっかりと作ってありました。

ハッカソン部門

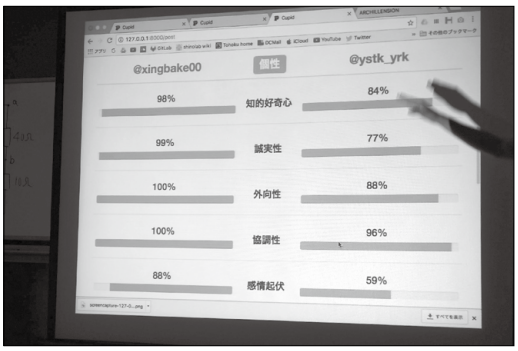
●特別賞▶声ハック、Cupid

ハッカソン部門の審査員のお二人からはそれぞれ特別賞が贈られました。1つは「声ハック」のチームで、このチームが作ったものはGoogle HomeとAPI.AIを活用して、音声で家電を操作するアプリです。たとえば「Turn on light」と話しかけると明かりを点けるといったようなことができます。API.AIは自然言語を解釈して会話するボットを作ることができるAPIで、ほかにはGoogle Assistanceと連携するためのActions on Google、Google Cloud Functionsなども使用しています。さらにIRKit[注3]を使用することで、赤外線リモコンで操作できる家電ならどのようなものにでも対応できるようにしたのが特徴です。

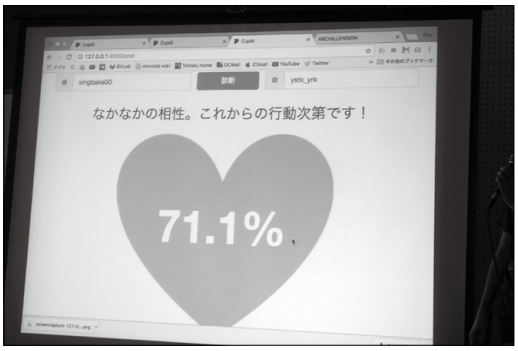
もう1つは「Cupid」のチームで、こちらのチームはIBM WatsonのPersonality Insightsを活用して恋のひみつ道具、相性診断アプリCupidを開発しました。お互いのこれまでのTwitterへの投稿内容を取得し、Personality Insightsで性格を分析(**写真4**)、それをもとに相性を算出するしくみです(**写真5**)。処理の実装にはPython、フレームワークとしてはFlaskが使われています。東北大学の2人によるチームで、実はHTMLを書くのもハッカソンに参加するのも初めてということだったのですが、審査員の方から「このままWatsonのデモに使えるのではないか」というコメントが出るほどのすばらしい成果でした。

注3　http://getirkit.com/

▼写真4　性格分析結果



▼写真5　相性診断結果



●ひみつ道具賞▶RAKUGAKI

今回のテーマである「ひみつ道具」と呼ぶに最もふさわしいものを開発したチームに贈られました。受賞したのは「RAKUGAKI」チームで、GoogleのTangoに対応した端末を使って周囲の壁や床などの環境を認識し、その上にARで落書きができるアプリを開発しました。端末を振ると「カラカラ」と音が出るようになっており、絵を書くときはシューというような音も出てスプレー缶でペイントアートをする感覚を再現しています。開発にはUnityを使い、当初は塗りつぶしのためにたくさんのオブジェクトを置いていたところ、それでは動作が重くなることが判明したため、ダイナミックメッシュの頂点で処理するように工夫したとのことです。

このチームは参加者全員の投票によって選ばれたオーディエンス賞も受賞しています。

●ヤバい賞▶「チャールズ」チーム

この賞はヤバいほどに高い技術力を出したチームに贈られました。

受賞したのは「チャールズ」チーム。「ザックリなお題をなんとかしたい」というところから始まり、進化型ニューラルネットワークと遺伝的アルゴリズムを使用して、あいまいな課題に対して人工生命のアプローチでいい感じにできるかもしれない糸口を

得たというものでした。デモでは「黒い玉をたくさん取ってきて」というあいまいな要求が、「この部屋の中の黒い玉を遠いほうから20個取ってきて」というプログラム可能な詳細な仕様に近づいていく過程を見ることができました。実装にはTensorFlowが使われています。

大まかな流れとしては、学習データが何もないところから、人工生命(エージェント)に報酬を与える(褒(ほ)める)ことによってニューラルネットワークは強化されていきます。遺伝的アルゴリズムは遺伝子の進化を模しているので、報酬が与えられなければ死んでいき、報酬が与えられたものは残っていき進化してアルゴリズムを獲得していくというようなものになっています(**写真6**)。

石巻ハッカソンのレベルが毎年上がっているのは、毎年こういうガチな大人がいることで良い刺激を与えているのも1つの要因ではないかという審査員からの講評もありました。

●グランプリ▶「forfools」チーム

会場となった石巻工業高校の生徒を中心に構成されたチームです。自分たちの学校をもっとみんなに知ってほしいというところから始まったという、自分たちが学ぶ高校の校舎を忠実に再現したホラーゲームです(**写真7**)。コンソールゲーム機のコントローラを操作して一人称視点でその中を歩き回ることができるようになっており、周辺の森は風などの天候の変化も表現されていました。Unityを使って開発されており、3Dデータのモデリングにはblenderが使われました。実際に動作するデモの完成度の高さもさることながら、作品を仕上げていく過程でメンバーが成長していっている感が大いにあることも評価ポイントとなりました。

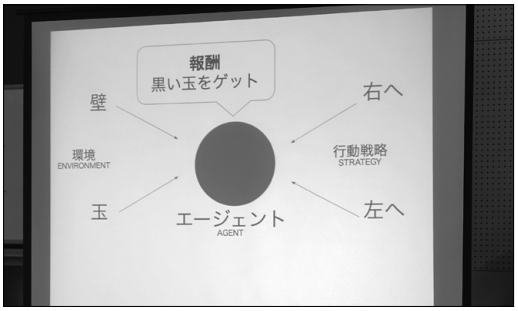
そのほかの作品

惜しくも賞は逃しましたが、ほかにもおもしろい作品がありましたのでいくつか紹介します。

●アプリ名：FreezeTag

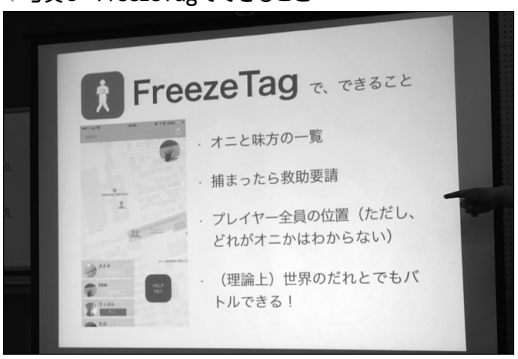
千葉県の柏から参加したチームが、アナログな遊

▼写真6　人工生命のエージェントが認識する環境と獲得していく行動戦略

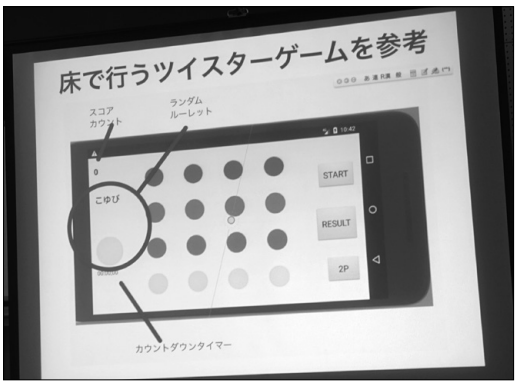


▼写真7　Unityで作られた教室のシーン

▼写真8　FreezeTagでできること



▼写真9　ハンドツイスター



びである「氷おに」とデジタルなものであるアプリを組み合わせたら「ひみつ道具」のおもしろい体験になるのではないかと考えて開発したものです（**写真8**）。Apple Watchも組み合わせて使うことができるようになっています。ひみつ道具賞の選定でRAKUGAKIと競っていたとのことで、最終的にはより「ひみつ道具」感のあるものということでRAKUGAKIのほうが選ばれましたが、こちらもすばらしい作品でした。

●アプリ名：ハンドツイスター

ご自身が手に障害を持つ参加者が自らの経験をもとに、障害を持つ人が楽しく機能訓練（トレーニング）できる道具を目指して企画、開発されたものです（**写真9**）。床で行うツイスターゲームを参考に、スマホの画面で手の指を使って行うように考えられています。

総評

```
while (Japan.recovering)
    we.hack();
}
```

非常にハイレベルなハッカソンだったということが言えると思います。2011年3月の東日本大震災で大きな被害を受けた石巻で、ITを新しい産業として盛り上げていくことを目的として設立されたイトナブ石巻[注4]の主催により、2012年から毎年開催されているこの石巻ハッカソン。もはや震災復興の域を超えて、開発に興味を持つたくさんの人を引きつけるイベントとして認知されつつあるのではないかと感じました。最初の数回こそHack For Japanは共催という形でサポートしていましたが、現在は完全に一参加者として楽しませてもらっています。2012年の第1回目のときにはまだ高校生でブートキャンプ部門に参加した若者が、ハッカソン部門で1つのチームを立派に取り仕切る姿や、当時アプリ開発は初めてと言っていた大学生が特別賞を受賞したりと、初回からずっと見てきた筆者にとって、みんなの成長を見ることも楽しみのひとつです（**写真10**）。

そしてもう1つ特筆したいこととして、このハッカソンを主催するイトナブ石巻の運営力のアップが挙げられます。高校生の発表に始まり、ハッカソン部門の19ものチームの発表をスケジュールどおりに終えることができたのはうれしい驚きでした。これまでは時間が伸び伸びになりがちで、遠方から参加してくれたみなさんに帰りの時間を心配させてしまうというような場面が多々ありました。しかし今回は滞りなく進行し、しかも表側にはそのスケジュール管理の厳しさは見せずにできていたことがすばらしかったと思います。

来年は8月3日から5日にかけての開催が予定されています。ぜひ石巻を訪れて開発に打ち込んでみませんか。SD

注4　http://itnav.jp

▼写真10　密度の濃い3日間をすごしたみんなで記念撮影

Report

ピクシブ、「pixiv MEETUP -10th Anniversary-」開催

9月9日、新宿セントラルパークシティホール（東京都新宿区）にて「pixiv MEETUP -10th Anniversary-」が開催された。本イベントはピクシブ㈱のイラストコミュニケーションサービス「pixiv」のサービス開始10周年を記念したイベント。計6つ行われたセッションのうち、エンジニアに関係する2つのセッションの模様をレポートする。

○「新規事業」はどのように生まれたのか――ピクシブでプロダクトを創るということ

▲左から山下鎮寛氏（ピクシブ㈱ pixivプレミアムプロダクトマネージャー）、清水智雄氏（ピクシブ㈱ pixiv・Sketch・Pawooプロダクトマネージャー）、道井俊介氏（ピクシブ㈱ ImageFlux事業責任者）

（山下）ピクシブは新規事業が多い会社ですが、お二人はなぜ新規事業を立ち上げたのか教えてください。

（清水）新規事業を立ち上げるにあたっては、創作活動を楽しむ人が増え、そして楽しみ続けてもらうため、という目的が一番にあります。そこを起点にして、クリエイターの日々の作業の中でまだサービス化されていないもの、未来にあってあたりまえのものを見つけ出し、実現するということを大切にしています。たとえばイラストレーターさんや漫画家さんは、絵を描きながら生放送配信をするといったことをよく行うのですが、それをサービス化したのが「pixiv Sketh Live」だったりします。

（道井）私が立ち上げに関わった「ImageFlux」は、pixivの画像処理技術を切り出して事業化したものです。画像処理はどの企業でも必須の技術ですので、収益化できるのではないかと思い、事業化しました。

（山下）新規事業の現場で必要となるマインドセットとはどのようなものでしょうか。

（清水）既存サービスへの分析と「妄想力」です。妄想力と想像力の違いはストーリー性にあります。サービスを考えるうえでストーリー、つまり流れ（時代の流れとユーザの行動の流れ）を考えることが重要です。

（道井）それに加えて大事なのが、365日それ（妄想）を続けることだと思います。妄想する内容の99%はどうでもいいことだとしても、考え続けていくことで、点と点がつながる瞬間があります。

○エンジニアの働き方――クリエイターファーストなエンジニアカルチャー

▲左から川田寛氏（ピクシブ㈱ エンジニアリングマネージャー）、高山温氏（ピクシブ㈱ 最高技術責任者）、小芝敏明氏（ピクシブ㈱ 開発本部長）

（川田）ピクシブは全員が「クリエイターファースト」のもとに働くという文化の会社ですが、開発体制と現場の雰囲気にはどのような特徴があるでしょうか。

（小芝）社員150名のうち、3分の2がサービス開発に携わっています。構成としては、5人ほどの小さなチームが数多く存在する組織です。ひとつひとつのチームは、ただタスクが割り当てられてそれをこなすというものではなく、そのチームだけで1つのサービスを上から下まで開発できるようにしています。

（高山）チームは数多く存在していますが、チーム同士の交流は厚く、風通しは良いと言えます。他チームとの交流を促す発表会を定期的に開催しています。

（小芝）プログラミング言語など、採用する技術に関してはプロダクトごと、チームごとに決定し、その知識の共有も発表会で行います。

（高山）メンバーの評価体制についてですが、チーム内のメンバーは基本フラットで、チームの外のマネージャが評価するしくみです。また1人のマネージャが独善的に判断するのではなく、マネージャ同士が集まって評価の検討をしています。

（川田）プロダクト開発の進め方はいかがでしょうか。

（高山）まず新規事業の立ち上げは、やりたい人ベースで進めることが多いです。エンジニアでも企画に携わることがままあります。

（小芝）Pawooの立ち上げは鮮烈でした。1人のエンジニアの声掛けからはじまり、Mastodonに興味のある仲間が集まっていき、翌日から開発が始まりました。

（高山）お祭り状態でしたね。「興味のある人は誰でも参加して良い、普段関わるプロジェクトはいったん止めても良い」と号令を出した覚えがあります。

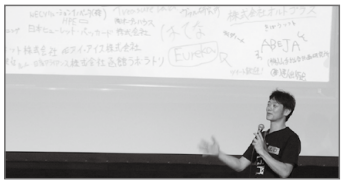
CONTACT
ピクシブ㈱ URL https://www.pixiv.co.jp

「エンジニアサポートCROSS 2017」開催

9月8日、横浜港大さん橋ホール（神奈川県横浜市）にて、「エンジニアサポートCROSS」がJAIPA（JApan Internet Providers Association）主催、横浜市経済局共催で開催された。

通算6回目の開催となる本イベント、国内最大級のIT系勉強会としてもはや定番的な秋の行事として認知している方も多いだろう。セッションの内容は、最近のソフトウェア開発の流れを受け、クラウドや機械学習、ブロックチェーンなどが多くなってきたが、エンジニアにとっての身の回りの問題であるキャリアパスやITコミュニティ運営、最新技術の習得など幅広い。テーマがごった煮状態ではあるが、そこには上からの押し付けがなく、ITエンジニア自らが楽しむ様子が感じられ、まるでお祭りのような雰囲気である。デイリーポータルZのお楽しみコーナーでは「巨大いらないものガチャ（エンジニア版）」と「顔が大きくなる箱」も、参加者の気分を盛り上げている。その中でも、イベントの終盤に行われた「先達に聞くこれからのエンジニア像2017」は、パネリストがさくらインターネット㈱伊勢幸一氏、楽天㈱よしおかひろたか氏、及川卓也氏による座談会形式（司会は森藤大地氏）で、来場者の注目を集めていた。登壇者はみな50歳台ということで、これまでのエンジニア人生を振り返って、何がキャリアにとって有効だったかざっくばらんに意見を述べた。その中で昨今の機械学習ブームのせいもあり、数学をしっかり学ぶことの重要性がさまざまな点から強調された。

多様なセッションを通じてエンジニアとして何をすべきか、いろいろな視点で手がかりを見つけられるのが本イベントの最大の特徴だと言えよう。問題意識を持っているエンジニアのみなさん、ぜひ来年は個人スポンサーやスタッフとして参加されてはどうだろうか。



▲業務参加した100社以上の企業名が書き込まれたホワイトボードを背景に、実行委員長山口亮介氏による閉会の挨拶

CONTACT
エンジニアサポートCROSS 2017
URL http://2017.cross-party.com

「U-22プログラミングコンテスト2017」開催

10月1日、秋葉原コンベンションホール（東京都千代田区）にて、「U-22プログラミングコンテスト2017」が開かれた。本コンテストには、22歳以下の若手から334のオリジナルのコンピュータプログラミング作品が集まった。経済産業大臣賞に輝いた4作品を紹介する。

○総合部門『Draw Near』：OMNISCIENCE（立教新座高等学校、西村太雅さん・中島正晴さん）

Draw Nearは、宇宙船内でサバイバルを行いながら、地球に帰還するためのロケットづくりを行うSFアクションゲーム。チームメンバーそれぞれの得意分野を活かし、モデルデータ、テクスチャ、アニメーション、BGMをほとんど自作したとのこと。

○プロダクト部門『Circuitor』：固有スキルせんたく板（越谷総合技術高等学校 情報技術科 29期生、山上翔さん・黒土直斗さん・永薗朋弥さん）

CircuitorはJavaのGUIライブラリSwingで実装した、PC上で動作する電子回路シミュレータ。GUIで電子部品のオブジェクトを配置し、コードをつなげることで、カウントアップ、ブリッジ、フリップフロップといった回路をシミュレートできる。

○テクノロジ部門『scopion』：小川広水さん（小石川中等教育学校）

scopionは「考えずにたくさん書ける言語」をテーマにしたプログラミング言語。強力な型推論、予約語の大まかな排除といった特徴がある。すぐに試せるように、Dockerを使ったオンラインコンパイラも用意した。

○アイデア部門『narratica』：菅野楓さん（早稲田実業学校中等部）

「narratica」は、映画のシナリオを入力するとキャラクタごとの感情の変化を時間軸で可視化できるシステム。形態素解析や感情分析をPHP+SQLで実装して実現した。スタジオジブリの宮崎監督に実際に会いにいき脚本について話を伺うなど、行動力の高さも注目された。

CONTACT
U22プログラミングコンテスト URL http://www.u22procon.com

Report

グレープシティ、「Excel方眼紙公開討論会」開催

9月30日、KFC Hall&Rooms（東京都墨田区）にて、「Excel方眼紙公開討論会」がグレープシティ㈱主催で行われた。Excel方眼紙とは、セルを正方形に設定してExcelをワープロのように利用する手法で、その是非については議論が絶えない。イベントで行われた、Excel方眼紙反対派、賛成派の登壇者によるセッションをレポートする。

○「ネ申タヒすべし：Excel方眼紙という時間泥棒」：立命館大学情報理工学部教授　上原哲太郎氏

上原氏はExcel方眼紙反対派陣営。ソフトウェアハウス、省庁、大学とキャリアを踏む中で、さまざまなExcelファイルと出会ってきたという。

最初に上原氏が語ったのは表計算ソフトの歴史について。世界初のPC向け表計算ソフトViscalc（VisiCorp社、1979年）から、Multiplan（Microsoft社、1982年）、Lotus 1-2-3（Lotus Development社、1983年）と変遷していき、1985年にMicrosoft社がMacintosh版のExcelを発売、1987年からはWindows版も提供された。MicrosoftがOSとOfficeソフトをバンドルして提供するようになると、各家庭にもOfficeが入るようになり、Excelの、表計算以外の多様な使われ方が生み出された。表計算ソフトを「表印刷ソフト」にしてしまったのはExcelだと、上原氏は指摘した。

それでは、企業でExcelがワープロのように使われるようになったのはなぜか。上原氏は和歌山大学や京都大学で業務のシステム化を推進していた時代、自治体とのメールにて、初めてExcel帳簿に出会った。時が経って総務省に入省したとき、上原氏はExcel帳簿の起源を知る。省の中をExcel帳簿が飛び交っており、各自治体のExcel帳簿もここから来ていることを発見したのだ。

このようなExcelの使い方に「ネ申エクセル」という名前が付いたのは、三重大学教育学部特任教授の奥村晴彦氏の2013年のツイートがきっかけ。なぜネ申Excelが今も使われ続けるかというと、とくに省庁での“紙”への信仰が根強いからだと上原氏は分析している。データのフローを帳票から考える昔からの慣習、証跡＝印鑑のある書類への絶対的信頼、そして電子データという見えないものへの恐怖が、ネ申Excelを生き延びさせているという。ネ申Excel問題解決の鍵は、生産性への着目にある、と上原氏は指摘。もしもネ申エクセルを受け取ったら、その送り主に対して、「データと体裁の分離」について解説し、そのデータは何に使うのか、そもそもその仕事は必要かと分析を促すのが解決への道だと提案した。


▲上原 哲太郎氏

▲長岡 慶一氏

○「Excel方眼紙がダメな理由を知りたい」：プログラマ　長岡慶一氏

長岡氏はExcel方眼紙賛成派陣営。2016年の長岡氏のブログの2016年のエントリ『教えてほしい。Excel方眼紙って何がそんなに悪いの？』が経緯で今回、登壇の依頼がきたそう。ただ立場としては「どちらかというと賛成派」で、1文字1セルといった極端なExcel方眼紙には反対している。

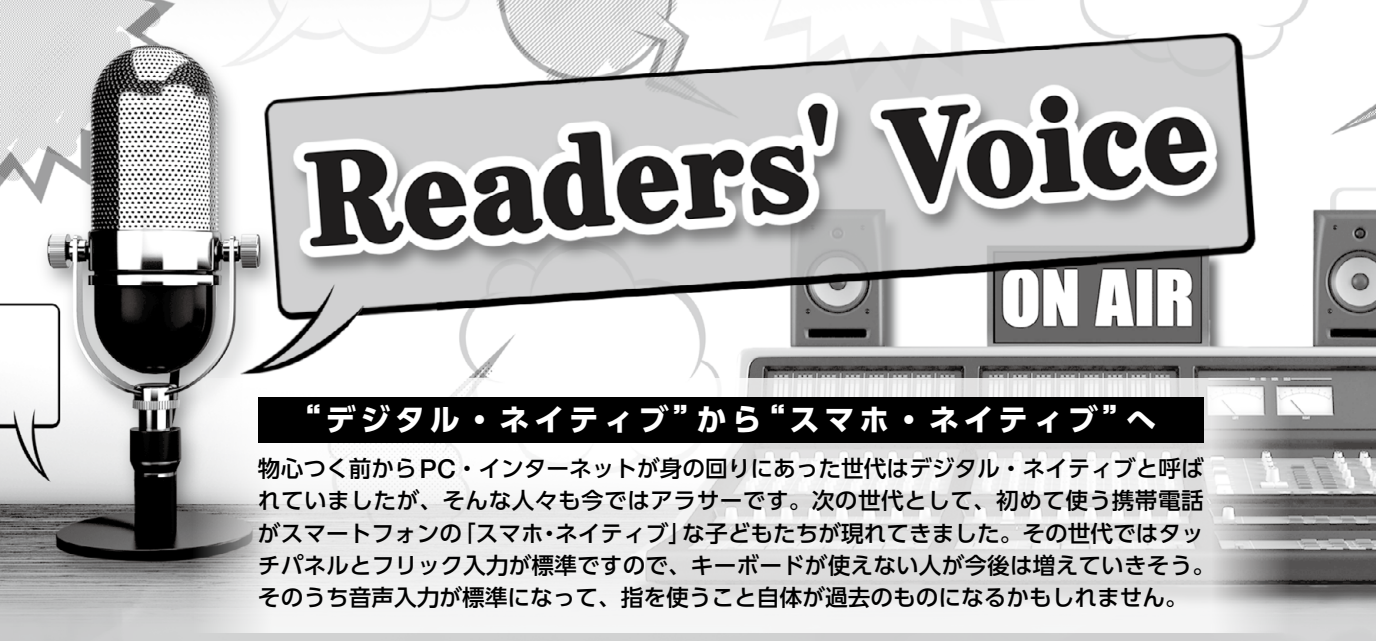
長岡氏が指摘したのは、Excel方眼紙賛成／反対の両派は、同じExcel方眼紙を見ていないのでは、ということ。不便な使い方がネ申エクセル、便利な使い方がExcel方眼紙なのであって、ネ申エクセル＝Excel方眼紙ではないのではと持論を展開した。長岡氏はまた、ネ申エクセル問題はExcel自体の問題、使い方の問題、方眼紙の問題に分解できるとし、便利に使えるはずのExcelをそうは使わないというところに問題の本質があると語った。

方眼紙を使用した便利なExcel方眼紙として、長岡氏は普段の業務で使っているDBのスキーマ定義書を挙げる。そこでは結合セルを部分部分で使って読みやすくしている一方、VBAやSQLと連携しやすい作りも心掛けているそう。そんな良いExcel方眼紙の条件として、「便利に使えること」「結合セルは極力使わない」「データを使い回すなら、そのしくみを考えておく」「印刷はおまけと考える」「ページや章の概念は捨てる」の5つを掲げた。最後に提案として、入力用・加工用・表示用シートと、工程でシートを分ける方法も紹介した。

◆ ◆ ◆

Excel方眼紙をテーマに取り上げた、恐らく史上初の企画だったが、Excelの苦労話には会場から同意の声が漏れたりなどと、強い一体感が感じられたイベントだった。

CONTACT
Excel方眼紙公開討論会
URL https://www.forguncy.com/information/events/excelforguncy



“デジタル・ネイティブ”から“スマホ・ネイティブ”へ

物心つく前からPC・インターネットが身の回りにあった世代はデジタル・ネイティブと呼ばれていましたが、そんな人々も今ではアラサーです。次の世代として、初めて使う携帯電話がスマートフォンの「スマホ・ネイティブ」な子どもたちが現れてきました。その世代ではタッチパネルとフリック入力が標準ですので、キーボードが使えない人が今後は増えていきそう。そのうち音声入力が標準になって、指を使うこと自体が過去のものになるかもしれません。

2017年9月号について、たくさんの声が届きました。

第1特集 Web技術【超】入門

移り変わりの激しいWeb技術の中でも、プロトコル、Webサーバ、Javaサーブレットなど、すぐには廃れない基礎的な技術をピックアップして解説しました。

初学者としては、こういった基本の基本から順序良く学ばせてくれるコラムは非常に助かりました。
LtKSKさん／神奈川県

クライアント専門のゲームプログラマにとって非常に有用な記事だと思います。 **ぐれちゃんさん／東京都**

表紙の「喰わず嫌いIIS」はそのとおりだと思いました。 **山下さん／東京都**

業務上、Apacheに慣れ親しんでいたため、なんとなくNginxを敬遠していたのですが、今回の記事で特徴がわかりました。機会があれば触ってみようと思います。 **たけしさん／埼玉県**

Javaサーブレットの歴史の流れが短くまとまっていたのが良かった。
勝木さん／東京都

JSPはオワコンというコラムにハッときました。 **ItSANgoさん／大阪府**

Web技術の入門に、そして復習に役立ったという声が多く寄せられました。また、今までは縁がなかったIISやNginxに興味を持ったという方も多くいらっしゃいました。

第2特集 開発効率アップのターミナル改造術

ターミナルを多重起動できるソフト「ターミナルマルチプレクサ」の特集です。カスタマブルなtmux、カジュアルに使えるByobuの2本立てで、作業効率アップの環境作りに取り組みました。

最近ターミナルに慣れてきてたので、こんなのもあるのかとためになった。
ホッシーさん／東京都

tmuxをさっそくインストールしてみました。裏技とか教えてほしい。
ももんがさん／静岡県

tmuxがとっつきにくくてByobuを使ってましたが、tmuxの良さを知ってしまったのでこちらも試してみようと思いました。 **くまくまさん／神奈川県**

この記事でByobuからtmuxに移行してしまった。vimrcの二の舞になりそうで恐ろしい(楽しみ)。
katsuyaさん／神奈川県

Byobuは途中で挫折した経験が過去にあるのですが、これを気にもう一度トライしてみようかなと思います。
n0tsさん／東京都

tmuxは使いこなしているつもりでしたが、sessionの切り替えができるなんて知りませんでした。また、一所懸命毎回たくさんのサーバを開き直ししていましたが、xpanesみたいな便利なコマンドがあったんですね。今回の記事を読んで、さらにtmuxが便利に使えそうです。 **今井さん／千葉県**

Byobuは便利な設定になっているので使い易そうですね。ただ、私は「Fn」キーが使いにくいキーボードを使用しているので、キーの設定の変更が簡単にできるのかどうか気になりました。
出玉のタマさん／大阪府

読者の間でも、tmux派とByobu派に分かれているようでした。記事でも触れられていましたが、設定に凝り過ぎると際限がなくなるというのがネックですね。

9月号のプレゼント当選者は、次の皆さまです

①HHKB Professional BT（無刻印）
曽我展世様（東京都）

②ドッキングステーション「PUD-PDC3L」
なまず様（神奈川県）

③ヌーラボ ノベルティTシャツ
張科様（東京都）、溝部寿一様（千葉県）、
黄掣様（大阪府）

④『UNIXプログラミング環境』
ken様（東京都）、和田健太郎様（熊本県）

⑤『Real World HTTP』
円満字竜也様（東京都）、増田渓介様（東京都）

⑥『現場で役立つシステム設計の原則』
toru様（愛知県）、韓徹様（福島県）

⑦『IBM Bluemixクラウド開発入門』
福田昌弘様（埼玉県）、石内博子様（福岡県）

※当選しているにもかかわらず、本誌発売日から1ヵ月経ってもプレゼントが届かない場合は、編集部（sd@gihyo.co.jp）までご連絡ください。アカウント登録の不備（本名が記入されていない場合、プレゼント応募後に住所が変更されている場合など）によって、お届けできないことがあります。2ヵ月以上、連絡がとれない場合は、再抽選させていただくことがあります。

一般記事 認証を支える技術

なりすまし・不正アクセスを防ぐためのしくみ「認証」について、初歩的な技術から攻撃の種類、シングルサインオン、さらには「認証・認可のファイアウォール」といった新しい概念まで紹介しました。

指紋認証は安全と思っていましたが、写真から指紋情報を取られて偽造に使われるなんて驚きです。
永作さん／東京都

二要素認証の重要性やそれぞれの認証の特徴がわかって良かった。
山本さん／東京都

複雑な認証について簡潔にまとめられていて良かったです。とくに「シングルサインオン」の解説は勉強になりました。
犬棟梁さん／埼玉県

社内のセキュリティで悩んでいる読者の方が多いようで、もう少し詳しく知りたいという声が多かったです。不審なサイトやメールはますます増えるばかりで、みなさん他人事ではないようです。

一般記事 Ejectコマンドで遊んでみませんか？

光学ドライブのトレイを開閉させるコマンド「Eject」で遊ぶ工作企画の第2弾です。今回は、ツイートするとスマートコンセントを経由して扇風機が回る装置を作りました。

光学ドライブの用途に笑った。
Gopherがブサカワイイさん／鳥取県

ギークな特集でおもしろかったです。
ぴょへいさん／大阪府

読むのにはおもしろかったが、自分でこれを実装する気力は今のところなさそうである。
cocoさん／宮城県

Ejectコマンドで遊ぶというのがとても懐かしく感じたので、大人の夏休みの課題のように読んだ。
hidewonさん／東京都

真面目にふざけた本企画、今回も好評のようです。自宅で試すというのはなかなかハードルが高そうですが、やってみたいという声もいくつかありました。

短期連載 人工知能時代のLispのススメ【3】

現在使われる多くのプログラミング言語に影響を与えた「Lisp」について、その画期的なしくみや思想に、あらためて注目する短期連載です。第3回では、Javaを使ってLispを再実装しながら、言語の深みに迫りました。

実践的で現場主義なところが大好きな連載でした。
オミオさん／宮城県

次はLispを使ったLisp処理系でしょうか。
下平さん／東京都

継続して読むことで、理解の薄かったLispについて理解が深まります。
Romeosheartさん／長崎県

JavaでLispを作る、というアプローチには驚きましたね。本連載はこれで最終回でしたが、情報学系の教員にはLisperが多いようで、毎回「学生時代、研究生時代を思い出しながら読んだ」という声が寄せられていました。

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
December 2017

2017年12月号
定価（本体1,220円+税）
184ページ

11月17日発売

［第1特集］学び方・仕事への生かし方

ITエンジニア・プログラマと数学

手を動かし納得しながらコーディング

数学とプログラミングの連携が求められている昨今、数学の基礎をきちんと固めてみましょう。数学を喰わず嫌いしている方でも、できるところから始めて成長する手がかりをゲットしてみませんか？

［第2特集］AIや機械学習で注目される

気になるGPU超入門

まさにAIブームで脚光を浴びる謎の会社――ではなく、みなさんおなじみのNVIDIA。機械学習だけでなくクラウドでの利用や仮想デスクトップでも注目を浴びているGPUテクノロジーを一気に解説します。

［特別付録］
・IIJ謹製「インターネット便利帳」
［特別企画］
・Python座談会

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

休載のお知らせ

「セキュリティ実践の基本定石」（第49回）は都合によりお休みさせていただきます。

SD Staff Room

●年上の従姉の話は聞くけど、親の注意をだんだん無視（奈良美智風）するようになってきた我が家の5歳児。そんな彼女に効くのが『たべてあげる』という絵本。この本を読んできかすと、嫌いなおかずもパクパク残さず食べてしまう。親の言うことよりも効くので、作者に感謝。ちなみにイタズラしたときも有効。（本）

●愛用のMacBookPro mid 2010が頻繁に再起動するようになってしまった。コアダンプではGPU系のエラー表示。冷却ファン故障かと、裏蓋開封して起動するも問題なし。とりあえずファンを取り出したら、エアフィルタがホコリで詰まっている。吸引。まだしばらく使えそうです。よかった。（幕）

●これを書いている翌日は、娘の小学校生活最後の運動会。まるで兄の背を追いかけるように応援団長に。本人にその気があるかはわからないけど、親としてはちょっと燃える（萌える）展開です。娘が出る競技に加えて、応援する姿もカメラに収めるべく、明日も校庭を奔走することになりそうです。（キ）

●ソフトウェアのUIが変わると操作を学びなおすのが大変です。それに比べると、シェルや正規表現のように基本機能はほとんど変更されないツールはとても重宝します。WindowsやExcelのように多くの人が使うソフトウェアもそうあってほしいものですが、ビジネス的にはなかなかそうはいかないか。（よし）

●エアコンからカビ臭い風が吹くようになり、クリーニングサービスを依頼。テキパキとエアコンの部品が外され、高圧洗浄機がかけられていきます。最後は清掃で出てきた真っ黒な汚水を見せられ、「こんなに汚れてたんですよ」「こんなに汚れてたんですね」と、システマティックなやりとりが交わされました。（な）

●2018年版の年賀状素材集が発売になりました。私も弊社の書籍「かんたん年賀状素材集」に来年の干支の犬のちびぬいたちの写真デザイン8点を提供させていただきました♪ @91頁です。ほかにもステキなデザインやイラストが満載です。これ1冊で家族みんなが使えますので、ぜひお手に取ってみてくださいませ。（ま）

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［E-mail］
sd@gihyo.co.jp

［FAX］
03-3513-6179

※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

Software Design
2017年11月号

発行日
2017年10月18日

●発行人
片岡 巌

●編集人
池本公平

●編集
金田富士男
菊池 猛
吉岡高弘
中田瑛人

●編集アシスタント
根岸真理子

●広告
松井竜馬
大橋 涼
目良直子

●発行所
㈱技術評論社
編集部
TEL：03-3513-6170
販売促進部
TEL：03-3513-6150
法人営業課（広告）
TEL：03-3513-6165

●印刷
図書印刷㈱

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

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