Software Design
12
2016年12月18日発行
毎月1回18日発行
通巻380号
（発刊314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

》NoSQL活用していますか？ 》文字コードを極める

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

2016

適材適所で
活用して
いますか？

Special Feature 1

NoSQLの教科書

MongoDB
Couchbase
Redis
MySQLでNoSQL!

Special Feature 2 どう見えるのか、それが問題だ！

文字コード攻略マニュアル

HTML・Java・Ruby・MySQLの
ハマリどころ

年忘れ特別企画 思えば遠くにきたもんだ

先人たちの知恵と足跡に学ぶ
【温故知新】

ITむかしばなしスペシャル

OSとネットワーク、
IT環境を支えるエンジニアの総合誌

Software Design

毎月18日発売

PDF電子版
Gihyo Digital
Publishingにて
販売開始

年間定期購読と電子版販売のご案内

1年購読（12回）

14,880円（税込み、送料無料）1冊あたり1,240円（6%割引）

PDF電子版の購入については

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

をご覧ください。
PDF電子版の年間定期購読も受け付けております。

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

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

Fujisan.co.jpからのお申し込み方法

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

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

12
2016
［目次］
Software Design

contents

Special Feature 1 第1特集

NoSQLの教科書 適材適所で活用していますか?

MongoDB、Couchbase、Redis、MySQLでNoSQL! 017

Special Feature 2 第2特集

文字コード攻略マニュアル

HTML・Java・Ruby・MySQLのハマりどころ 059

12
2016
［目次］
Software Design

Special Feature 3　第3特集

年末特別企画

温故知新 先人たちの知恵と足跡に学ぶ ITむかしばなしスペシャル　083

Extra Feature　一般記事

Catch up trend

à la carte　アラカルト

contents

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.）
［表紙写真コラージュ］
藤井 耕志（Re:D Co.）
［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）
＊伊勢 歩、横山 慎昌（BUCH+）
＊藤井 耕志、萩村 美和（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。

技術評論社の 確定申告本

平成29年3月締切分

初めてでも大丈夫！ マネして書くだけ 確定申告 平成29年3月締切分

山本宏　監修／A4判／176ページ
定価（本体1,380円＋税）

ISBN978-4-7741-8442-5

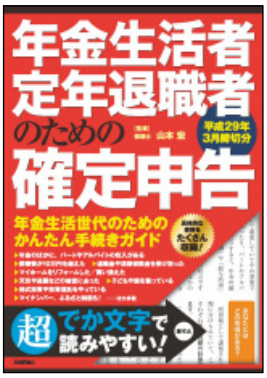
家族が働いていたり副業があるために確定申告が必要なサラリーマンや、主婦、パート、アルバイト、年金受給者はもちろん、個人事業主やフリーランサー、不動産オーナーまで、個別のケースごとに具体的な事例をたくさん掲載しています。自分の状況に近い事例を選んで、番号順にマネして書くだけで書類が記入できるので、忙しくて書類作成に時間をとれない方や、初めて確定申告を行う方などに、特におすすめしたい1冊です！
ふるさと納税をした方の申告方法や、確定申告の基礎知識もわかります。

フリーランス&個人事業主 確定申告でお金を残す！元国税調査官のウラ技 第3版

大村大次郎　著
A5判／224ページ
定価（本体1,580円＋税）

ISBN978-4-7741-8441-8

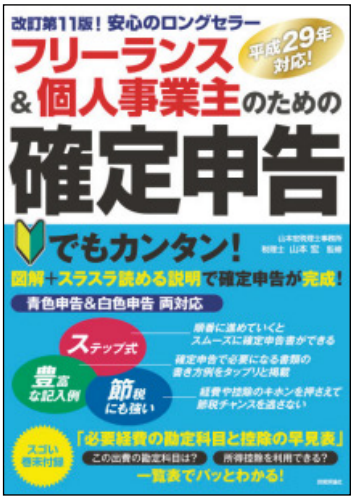
フリーランスや個人事業主の方が1年間の仕事の成果を書き入れる確定申告書。その申告書に何を書くのかによって、手元に残るお金の額は違ってきます。確定申告には、「こうしたらトクになる」というやり方がありますが、納税者に有利になる（納める税金が少なくなる）情報を税務署が教えてくれることはありません。節税にはとくに複雑な手続きは必要ないため、節税できるかどうかを分けるのは、ズバリ“知っているかどうか”です。確定申告を絶好の節税の機会にする方法を、元国税調査官の著者がお届けします！



年金生活者のための定年退職者のための確定申告 平成29年3月締切分

山本宏　監修
A4判／144ページ
定価（本体1,480円＋税）

ISBN978-4-7741-8443-2



フリーランス&個人事業のための確定申告 改訂第11版

山本宏　監修
A5判／256ページ
定価（本体1,480円＋税）

ISBN978-4-7741-8444-9

フリーランス&個人事業主として働く人の確定申告をサポートする定番書。
今回の第11版では、皆さんの「そうそう、それが知りたかったんだ」により応えられるようボリュームアップを行いました。ステップ式で確実にスピーディに手続きができるほか、勘定科目をさっと検索できる付録も充実しています。お手元にあることで事業を営む皆さまのお役に立てること請け合いです！

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

BOOKS INFOMATION

ISBN978-4-7741-8410-4
A5判／176ページ
定価（本体1680円+税）

コンピューターで「脳」がつくれるか

AIが恋に落ちる日

五木田 和也 著

囲碁で人間を打ち負かす人工知能ロボットや自動運転車が進歩を続けても「人間のようなロボット」にはなりません。世間は人工知能ブームに沸いていますが、人間のように自ら学習して成長していく「汎用人工知能」を実現するにはまだ道なかばです。そもそも汎用人工知能とディープラーニングやIBMのWatsonに代表されるようなふつうの人工知能は何が違うのでしょうか。
本書では、脳の各部位がどんな役割を持ち、どうやって情報を処理しているのかを解説し、人間らしい知能を実現するにはどのようなしくみが必要か解説します。

ISBN978-4-7741-8492-0
A5判／256ページ
定価（本体1980円+税）

『アポロ13』に学ぶITサービスマネジメント

映画を観るだけでITILの実践方法がわかる！

谷誠之・久納信之 著

本書では、映画『アポロ13』を題材にした好評のセミナーをベースに、ITサービスマネジメントの基礎を学んでいきます。アポロ13号の事故は40年以上も前の出来事ですが、その原因はITサービスの現場で発生するインシデントと共通しています。また、フィクションではなく史実に忠実に作られたストーリーなので、臨場感かつリアリティのある解説で、ITサービスマネジメントの本質と実践方法を楽しく学ぶことができます。さらに、映画『アポロ13』を鑑賞しながら本書を読めば、実体験のような高いモチベーションで学習できることでしょう。

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

紙面版
A4判・16頁
オールカラー

電脳会議

DENNOUKAIGI

一切無料

新規購読会員受付中!

『電脳会議』は情報の宝庫、世の中の動きに遅れるな!

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

今が旬の情報満載!

新規送付のお申し込みは…

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

電脳会議事務局　検索

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

一切無料!

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

毎号、厳選ブックガイドも付いてくる!!

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

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

Software Design plus

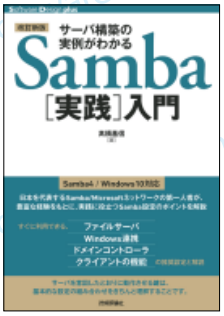
Software Design plusシリーズは、OSとネットワーク、IT環境を支えるエンジニアの総合誌『Software Design』編集部が自信を持ってお届けする書籍シリーズです。

最新刊！

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

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

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



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

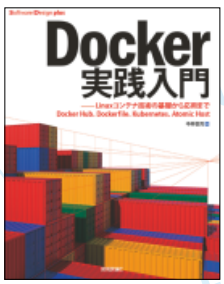
高宮安仁、鈴木一哉、松井暢之、村木暢哉、山崎泰宏 著
A5判・352ページ
定価 3,200円(本体)+税
ISBN 978-4-7741-7983-4

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

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

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



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

養成読本編集部 編
B5判・232ページ
定価2,080円(本体)+税
ISBN 978-4-7741-85-5

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

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

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

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

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

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

C#エンジニア養成読本
岩永信之、山田祥寛、井上章、伊藤伸裕、熊家賢治、神原淳史 著
定価 1,980円+税　ISBN 978-4-7741-7607-9

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

AWK実践入門
中島雅弘、富永浩之、國信真吾、花川直己 著
定価 2,980円+税　ISBN 978-4-7741-7369-6

シェルプログラミング実用テクニック
上田 隆一　著、USP研究所　監修
定価 2,980円+税　ISBN 978-4-7741-7344-3

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

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

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

事例から学ぶ情報セキュリティ
中村行宏、横田翔 著
定価 2,480円+税　ISBN 978-4-7741-7114-2

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

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

OpenSSH[実践]入門
川本安武 著
定価 2,980円+税　ISBN 978-4-7741-6807-4

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

技術評論社

新連載

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2回

Product Managerが日本を救う

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

ものづくり日本と言われるが

連載初回となる前回、その冒頭で、今日本でもProduct Manager（以降、PMと省略）の重要性が認識され始めているとお話しましたが、私がなぜこの連載を始めたかについて、今回はもう少しお話をさせていただきます。

昨今、人工知能（AI）が注目を集めています。日本ではこのようにある技術が注目されると、その技術者が不足していると危機感を煽（あお）る論調が聞かれるようになります。AIにしても、AI技術者が数万人規模で足りなくなるなどという話もあります。AI技術者というと何か別の職業のように聞こえるかもしれませんが、本誌読者ならおわかりのとおり、AI技術者もプログラマです。プログラマの中でAIを開発できたり、活用できたりする人がAI技術者であるだけです。

では、そのプログラマの日本での地位はどうでしょう？　大手IT企業の中には要求定義や工程管理しか行わず、実装は下請けや孫請けにいるプログラマが行っているところも多くあります。大手から仕事を請け負ったそのような会社の労働環境や待遇は大手より悪いため、日本ではプログラマになりたがる人はおらず、プログラミングをしなくなって一人前というような風潮さえあります。ものづくり日本と言われますが、そのものづくりにおいて現在一番重要な技術であるITにおいてさえ、このようなお寒い状況なのです。

数年前、ある全国紙にプログラムを自動生成する技術が大手IT企業によって開発されたという記事が掲載されました。その記事では、この技術でプログラマが行っていた仕事が不要になると書かれていました。これはこれで、すべてのプログラミングがそのように思われかねないミスリードな記事なのですが、それよりも私が残念に思ったのは、プログラマに用語説明が書かれていたことです。確か、「プログラマとはシステムエンジニア（SE）の用意したシステム設計書をプログラミング言語に書き換える技術者」などと説明されており、とても残念な気持ちになったことを覚えています。この説明もSEのほうが上級技術者であると誤解させますが、それよりもプログラマという言葉に解説が必要なことにショックを覚えました。

しかし、その状況も変わりつつあります。実装力がそのまま製品の良し悪しにつながることが、Webやスマートフォンなどでは認識されつつあります。また、政府主導で学校教育の中でのプログラミング教育が検討されるなど状況は改善してきています。プログラミングを理解した若者が多くなれば、その受け皿となる企業も変わらざるを得ません。笑い話ではないですが、学生の頃からプログラミング能力を高めた若者がいざ就職しようとしたとき、彼らの目に魅力的に映る日本企業がなかったならば、彼らは外資系企業に就職してしまうでしょう。

プロダクト開発に必要なもの

このようにソフトウェアプロダクトの開発に

おいてプログラミングを行うエンジニアの重要性は少しずつ認識されつつありますが、一方でプロダクト開発はプログラマだけで行えるものではありません。前回書いたように、プロダクトを世に出すためにはさまざまな役割を持つ専門家が必要です。

最高のユーザ体験(UX)を与えるためには、UXリサーチャーやUXデザイナーが必要でしょうし、市場調査をするためには企画担当者も必要です。完成したプロダクトを世に出すことを考えると、広報やマーケティング、さらにはユーザーサポートもいるでしょう。プログラミングだけ見ても、インフラからフロントエンドにバックエンドなどのエンジニア、プロダクトによってはデータベースに長けた専門家も必要かもしれません。

このような専門家で構成されたプロダクトチームを指揮するのがPMなのですが、PMは何の専門家かと言われると言葉に詰まってしまうことがあります。逆に言うと、一言でPMを説明しづらいからこそ、今までその重要性が認識されにくかったのかもしれません。

前回、PMの役割をスポーツチームのマネージャーに擬(なぞら)えましたが、メタファーとしてほかに私がよく使うのが映画のプロデューサーです。

映画は誰のものかと聞かれたら、皆さんはなんと答えるでしょうか？　有名監督の映画なら監督のものと思うかもしれません。人気俳優が出ていたら、その俳優かもしれません。原作がベストセラーになったものなら、その作家の名前を思い浮かべるかもしれません。脚本が良いものもあるでしょう。しかし、映画はこのような監督や脚本家、俳優だけでは作れません。プロデューサーが不可欠なのです。

多くの映画において、プロデューサーは商業作品として興行収入をあげることを目指します。映画によっては芸術性の高いものもあるでしょう。しかし、その場合でも純粋な芸術作品でもない限りは、作品が多くの人の目に触れるように、適切な尺を考え、1日に何回上映できるかなども考慮します。制作費を抑えるような努力もするでしょう。監督と衝突することも多くあると聞きます。監督や主役の交代をたまに耳にすることなどからもわかるように、必要と判断すれば、制作スタッフも変更します。

プロダクト開発におけるPMの役割もプロデューサーと似ています。縁の下の力持ち的な存在でありながら、最終的な決定はすべて下す。前回、PMはサーバントリーダーとして振る舞わなければならないが、同時に強いリーダーシップを持つ必要があるとお話したのはこのような理由からです。**合議制では良いプロダクトは生まれません。**プロダクトの方向性を決めたり、さまざまな局面においての決断を行うのは最終的にはPMになるのです。

IT立国を目指すには

今日、経済大国となっている国の多くは高いIT競争力があります。現代社会において、多くの経済活動はITによって支えられています。ITは昔はコスト削減のための技術でした。労働者の行っていた作業を機械化することが産業革命でしたが、ITも昔は事務作業や工場での生産を置き換えていくものでした。今でもその流れはあり、多くの企業においてITは不可欠なものとなっていますが、ITの活用はそれだけに留まらず、ITでないと生み出せないものを生み出した企業が成長し、活用できなかった企業は廃れていきます。

日本の国際競争力は未だに高いものを維持していますが、現在直面している少子化などの社会課題に対応するには、ITのさらなる活用は不可欠です。そんな状況下、**プログラムを書くエンジニアと同じように、ITでのものづくりを支えるのがPMである**と信じて疑っていません。エンジニアと同じように、PMを目指す人が増えてほしい。そんな思いが、私がこの連載を開始した理由となります。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。現在はプログラマーのための情報共有サービスQiitaのプロダクトマネージャーを勤める。

第96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。

cri-o

Kubernetesによる新コンテナランタイム「cri-o」

Dockerをはじめとしたコンテナ環境ためのオーケストレーションツール「Kubernetes」の開発チームが、インキュベータープロジェクトとして独自のコンテナランタイムの開発をスタートさせています。このランタイムは「cri-o」と名付けられており、コンテナ関連の2つの標準仕様である「OCI（Open Container Initiative）」と「CRI（Container Runtime Interface）」に準拠するとのことです。

OCIは、DockerやCoreOSを中心としたベンダー各社が発足させた「The Open Container Initiative」が推進するコンテナ標準仕様です。CoreOSやRed Hatなど複数のベンダーがOCIをサポートしたランタイムをリリースしているほか、リファレンス実装として「runC」があり、Dockerもバージョン1.11からはrunCベースとなっています。

CRIは、Kubernetesとランタイム間のインターフェースを標準化した仕様です。CRIに準拠していれば、Docker以外のランタイムでもKubernetesと組み合わせて利用できるようになっています。

開発チームによれば、cri-oプロジェクトのスコープはCRIのスコープと強く関連しているとのことで、機能面の具体的なターゲットとしては次のような内容が挙げられています。

- コンテナイメージの管理
- コンテナのプロセスライフサイクルの管理
- モニタリングおよびロギング
- リソース分割
- Docker Image Formatを含む複数のイメージフォーマットのサポート

一方で、イメージの作成やコマンドラインユーティリティによる管理はスコープの範囲外であり、cri-oではサポートする予定はないとのことです。

cri-o 登場までの経緯

実質的に、昨今のコンテナ型仮想化の隆盛はDockerの登場によって始まったと言えます。Kubernetesなどのサードパーティによる関連ツールも数多く作られ、Dockerを中心としたエコシステムは業界全体を巻き込んで急速に成長していきました。Dockerに特化したLinuxディストリビューションであるCoreOSの登場もその1つです。

風向きが変わってきたのは、そのCoreOSが、Dockerと決別して独自のコンテナランタイムの開発に着手したころからです。この決別の背景には、Docker一社の影響力が強くなり過ぎることに対する大きな懸念があったと言われています。最終的にDockerとCoreOSは再び和解しますが、この一件が、関連するベンダーを巻き込んでThe Open Container Initiativeの設立につながりました。

標準仕様としてのOCIが軌道に乗り始めたことで、各社は自由にOCI準拠のコンテナやランタイムを作れるようになりました。言い方を変えれば、Dockerに依存しないコンテナのライフサイクルが構築できるようになったということです。

このような流れの中で、Kubernetesプロジェクトでは Red Hatが中心となってOCIDと呼ばれる独自のランタイムの開発をスタートさせ、これが改名されてcri-oとなりました。Kubernetesの立場としては、コンテナランタイムを交換可能なコンポーネントとしてコモディティ化させることでイノベーションをさらに加速させたいという狙いがあるようです。

cri-oは、Kubernetesとの連携を前提にチューニングされたコンテナランタイムとして、ユーザに新しい選択肢を提供することになります。cri-oの開発チームは、プロジェクトのスコープはあくまでもオーケストレーションであり、Dockerと競合するものではないという点を強調しています。しかし、そうは言ってもcri-oの存在がコンテナ業界のエコシステムの主導権争いに大きな影響を与えることは明白であり、しばらくは目が離せない状況が続きそうです。

SD

cri-o
https://github.com/kubernetes-incubator/cri-o

DIGITAL GADGET vol.216

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

再び盛り上がる電子ブロック的アイデア

ブロックの歴史

1958年に登場したレゴブロック、最初の電子ブロックが発売されたのが1965年、それ以降、さまざまなブロックや電子ブロックが登場してきました。ブロックの拡張性と柔軟性のある考え方は、多くのハードウェア、ソフトウェアに影響を与えていると考えられます。ブロックが持つ本質で、ほかにも応用できる事柄は何でしょうか?

- 基本設計がシンプルで、拡張の余地があること
- 互換性を保ったうえで、個々の機能が豊富であること
- ある目的を達成するための方法が複数存在すること

一方、ブロック的な考えにもデメリットはあり、何かを試作したり、目的のものを作り上げるときに、それらのデメリットを考慮する必要もあります。

- 完成物ではなく、量産や複製が困難であること。永続性はないこと
- 限られた数と種類のブロックから作るため、発想にある種の制限を受けること
- 汎用的なブロックと、特殊なブロックが必要とする機能のバランスが難しいこと

※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。

これらのブロック的な考えは、オモチャだけではなく、電子楽器や建築の世界、家具など、モジュール化できるものであれば、何にでも役立つ要素を多数含んでいます。

電子ブロック的ブロックの数々

LittleBits

http://japanese.engadget.com/2015/06/19/2015-littlebits/

電子ブロックの話題をする際、米国のスタートアップ、LittleBitsの紹介を避けて通ることはできません(**pic.1**)。電子楽器メーカーであるKORGとの協業でシンセモジュールを出したり、NASAとの協業によるSpace Kitでは、NASAの科学者が記したサンプル回路の解説が付属します。そのほかにもクラウドモジュールcloudBitで、Web連携サービスIFTTTとつなぐこともできます。また、当初は高価だと敬遠されていた部分も、ハードウェアの回路図、仕様などをオープンソース化し、他社や個人がモジュールを作れる余地も出てきました。LittleBitsのWebページには、希望の新モジュールのアイデアを書き込むドリームビッツというページ(http://littlebits.cc/dreambits)が用意されており活況です。最近ではLEGOブロックと組み合わせて遊ぶ遊び方も広まり始めているようです。LittleBitsのCEOによると、既存のIoTデバイスの70〜80%は、LittleBitsの組み合わせで実現できるそう。

mCookie

https://www.microduino.cc/index-mcookie

mCookieはArduino互換ボードをリリースしているMicroduino Studioのモジュールです(**pic.2**)。レゴのパーツ

学研電子ブロック「EX-150」(写真は80年代発売当時のもの)

IoT化したレゴブロックの教育用製品「WeDo 2.0」

を取り付けることのできる小さな電子モジュールで、これらを簡単に着脱できます。形を整えたり、ケース的なものを作ったりするのは、LEGOブロックのパーツで平易に行えます。IoTブロックとも言えるmCookieです。

KOOV
https://www.koov.io

KOOVはプログラミング可能な、ロボット化ブロックです(**pic.3**)。おもにロボットやプログラミングの教材用に考えられたもので、ブロックの要素で形を作り上げたものを、プログラミングの要素で、動きをコントロールすることができます。教育カリキュラム「STEM101」のプロジェクトの1つとして推進されており、詳細は不明ですが、タブレット端末上で命令を組み合わせてプログラムを作成し、そのプログラムをブロックに送信して動かすようです。

Makeblock
http://www.makeblock.com

おもに教育用として考えられているMakeblockは、アルミのパーツや電子部品を組み合わせてロボットを作ることのできるブロック製品です(**pic.4**)。基本的なブロックを組み合わせて何かを作るというイメージよりも、3Dプリンターやレーザーカッター、プロッタ、ロボット、ロボットカーなど、ある完成物を組み立てるのに必要なパーツ群がそろっているというイメージです。

Brixo
http://www.getbrixo.com

BrixoはLEGOブロック互換のIoTモジュールです(**pic.5**)。電子回路の回線そのものもブロック化されているため、簡単な電子回路を組むことができます。モジュールとしては現状、Bluetooth接続、音センサー、光センサー、接近センサー、LEDライト、モーターのブロックが用意されています。ハンダ付けなしで回路を構築でき、非毒性の金属で安全だそうです。

ESLOV IoT Invention Kit
https://www.kickstarter.com/projects/iot-invention-kit/eslov-iot-invention-kit

ESLOV IoT Invention Kitは22ミリ角の正方形の電子回路で、センサーや機能モジュールを組み合わせて電子回路を作ります(**pic.6**)。基本となるWi-Fi機能を搭載したハブと、数個のモジュール、GUIベースの簡単なプログラミングで回路を作り上げます。モジュールは、各種センサー、小さなOLEDディスプレイ、ブザー、LED、物理ボタン、リレー、GPS、タッチセンサー、サーボモーター、マイクロフォン、自作用基盤など、ひととおりの部品がそろっています。

Google Project Bloks
https://projectbloks.withgoogle.com

Google Project Bloksは電子回路のためのブロックではなく、プログラミングの概念を学ぶための電子ブロックです(**pic.7**)。プログラムにおける分岐やループなどといった概念を学ぶことができるよう工夫されています。利用想定年齢は5歳以上とのこと。プログラミングによって物理的なものをコントロールできることを学べるそうです。オープンなハードウェアプラットフォームとして、広く多くの人がモジュールを提供できることを想定しているそうです。ブレインボードと呼ばれる頭脳の部分、パックと呼ばれるボタンやスイッチ、ダイヤルなどの操作部

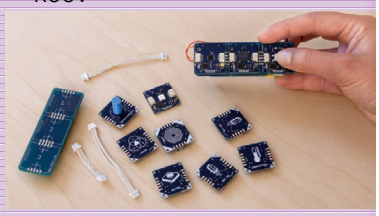
pic.1 LittleBits

pic.2 Microduino mCookie

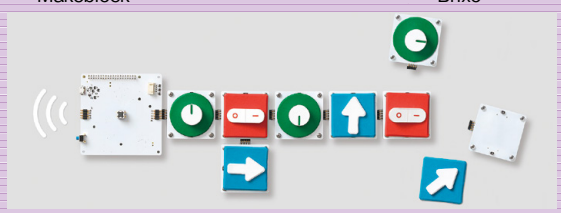
pic.3 KOOV

pic.4 Makeblock

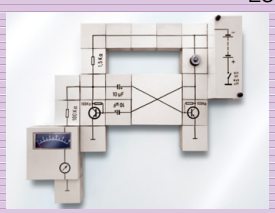
pic.5 Brixo



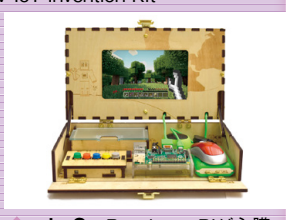
pic.6 ESLOV IoT Invention Kit



pic.7 Google Project Bloks



pic.8 ドイツの電子ブロック、LECTRON



pic.9 Raspberry Piが心臓部のPiperキット

分などに分かれます。

ブロックの本質

1970年代、電子ブロックが流行し、似たような製品も数多く登場しました。その後いったんブームは収束し、2000年代になってから、また電子ブロックの復刻版が登場し人気を博しました。新しい世代が飛びついたというよりも、子供のころ、高価すぎて手に入らなかった世代が大人になって、夢(というより欲望?)を叶えたのかもしれません。

電子ブロックと同じ時代に登場したLECTRON(http://lectron.info/)は、現在では、Webにある情報と、博物館に保存されているものでしか知る由もありません(**pic.8**)。現存しているものと、過去のものとの差異はどこにあるのでしょうか。永続性のある技術とは、どこがポイントなのでしょうか? 電子ブロックの栄枯盛衰にあわせて、ハードウェアやソフトウェアの栄枯盛衰も考えることができそうです。

さて、最近では新しい世代が電子ブロック的なオモチャやグッズを手に入れ、新しい発想のもとに、新しいものを作り上げるようになってきました。ネットやクラウドサービスさえも、ブロックの一部品でしかないのです。なにかあらかじめ決まったものしか完成しない「キット」的な、お膳立てがされたものではなく、各部品の特性や役割を知ったうえで、それらを最大限組み合わせて、今までにないものを作り上げるのがブロックの醍醐味です。発想の広がりを促進し、何か新しいしくみを発見するためのブロックであってほしいものです。

最近Microsoftに買収された子供達に人気のゲームMinecraftも、ある種ブロックの楽しみを仮想化したものです。実際にMinecraftの世界を使って電子工作とプログラミングを学ぶツール「Piper」(https://playpiper.com/)も販売されています(**pic.9**)。そうしたブロックによるある種の制限と、限界を知ったうえで、いつかはブロックの枠組みさえも超えて、新しいものを作り始めることになるのかもしれません。SD

Gadget 1

Moto Mods

http://www.motorola.com/us/moto-mods

ブロック的スマートフォン

Googleの組み替え式スマートフォンProject ARAは、残念ながら事情によりプロジェクトが中止されてしまいました。その一方、MotorolaのMoto Modsは、スマートフォン(Moto Z)の背面のモジュールを切り替えることで、さまざまな専用スマートフォンに切り替えることのできる確実で現実的なソリューションです。モジュールとして、スピーカーメーカーJBLのスマートフォン用スピーカー、小型プロジェクター、カメラメーカーHasselbladのズームレンズ付きカメラ、非接触充電バッテリーモジュールなどがあります。

Gadget 2

Linktz

http://www.linkitz.com

モジュール型腕時計

Linktzは追加モジュールによって、機能を持たせることのできる、ウェアラブルウォッチ型キットです。1つの腕時計に3個のモジュールを搭載でき、モーションセンサーとLEDを2個といった組み合わせや、マイクロフォン、スピーカー、USB端子などが用意されています。クラウドファンディングで目標を上回る10万ドルの資金を集め、現在追加の予約を受付中です。

Gadget 3

cubelets

http://www.modrobotics.com/cubelets/

サイコロ型ロボットブロック

cubeletsは磁気でくっつく立方体のブロックで、プログラミングも配線も必要なく、ブロックの組み合わせで、動きを作っていくことができます。Bluetoothユニットを追加すると、各ブロックの動きを再プログラミングすることもできます。LEGOブロック用のアダプタも用意されており、さらに拡張した形を作ることができます。

Gadget 4

Osmo

https://www.playosmo.com/en/

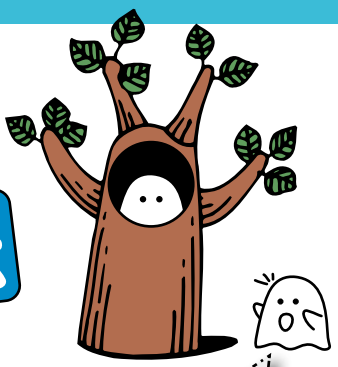
プログラミング学習ブロック

Osmoはプログラミングの概念を学ぶためのブロック。文字ではなく、単純な動作のしくみが絵で書かれた平面ブロックを組み合わせてゲームを作ります。画面の中ではなく、手で触れるモノの組み合わせでプログラミングを学ぶのが特徴です。iPadのアプリと専用スタンドの組み合わせで、ブロックを画像認識させて利用します。

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

43

結城浩の再発見の発想法



チューリングテスト

チューリングテストとは

チューリングテスト(Turing test)とは、数学者アラン・チューリングが提唱したテストで、**「機械は考えることができるか」という解答困難な問いを検証可能な問いに変換したもの**です。

「機械は考えることができるか」はとても難しい問いです。答えるのが難しいだけではなく、問いの意味を定義するのが難しいのです。それは「考える」という言葉が持つ意味の広さを想像すれば理解できるでしょう。そこで「機械に何ができたなら、考えたと言えるか」という発想の転換を行います。

チューリングは論文"Computing Machinery and Intelligence"[注1]の中で、「機械」や「考える」の定義をやめ、**イミテーション・ゲーム**という形で「機械は考えることができるか」の言い換えを行っています。

イミテーション・ゲームの概要はこうです(**図1**)。

- 登場人物は、男性Aと女性Bと質問者Cの3人
- 3人はそれぞれ別室に入る
- 「AとC」ならびに「BとC」は文字だけの通信ができる
- 質問者Cは通信相手のどちらが男性Aでどちらが女性Bかを知らないが、2人のどちらとも通信でコミュニケーションを取ることができる

そして、男性Aと女性Bと質問者Cには異なる目的が与えられます。

- 男性Aの目的は、質問者Cに対して「自分は女性Bであると思わせる」こと
- 女性Bの目的は、質問者Cに対して「自分は女性Bであると信じてもらう」こと
- 質問者Cの目的は、通信相手の性別を判断すること

男性Aが女性Bを模倣するのでイミテーション・ゲーム(模倣ゲーム)というわけです。

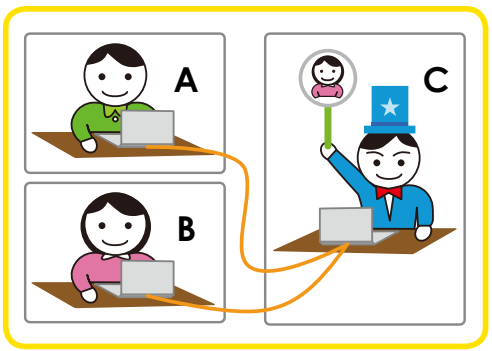
チューリングは「イミテーション・ゲームの男性Aを機械に変えたらどうなるか」と問います(**図2**)。そして、

「機械は考えることができるか」

という問いを、次のように言い換えます。

注1) 原文：URL http://www.loebner.net/Prizef/TuringArticle.html 和訳：URL http://www.unixuser.org/~euske/doc/turing-ja/index.html

▼図1 男性Aが女性Bのふりをするイミテーション・ゲーム



「男性Aを機械に変えた場合、質問者Cは、人間のときと同じくらい判断を誤るだろうか」

イミテーション・ゲームの男性Aを機械に変えたもの、これがチューリングテストです。

ある人は「イミテーション・ゲームで質問者Cをうまく惑わせる機械ができても、それは単に人間を模倣できる機械ができたに過ぎない。機械が考えているわけではない」と考えるかもしれません。でも、よく考えてみますと、私たちは普段たくさんの人とネットごしに会話しています。それは、日々イミテーション・ゲームと同じ舞台に立っているようなもの。通信相手に関して、性別・能力・信頼度……たくさんの判断を行っています。人間を模倣できるほどの機械でも「考えていない」と言うなら、私たちもまた考えてはいないことになってしまいます。

日常生活とチューリングテスト

チューリングテストにまつわる事象は、私たちの生活のあちこちで見つかります。

チューリングテストでは、**見た目**に影響されないようにコミュニケーションを文字に制限していました。実際、私たちは多くの場面で見た目に影響されている可能性があります。性別を隠した試験では男女で差がないのに、性別を明らかにした試験では男性が優位になると主張する人もおり、採用試験での性別の影響を研究している人もいます。その真偽はわかりませんが、人間同士のやりとりというのは繊細なものですから、試験官が自覚なく性差に影響を受けていてもおかしくはないでしょう。

消費者金融の自動契約機では、背後に人がいるにもかかわらず、あたかも無人で処理されるように見せています。これは恐らく、機械が処理しているように見せて、利用者の心理的抵抗を少なくしているのでしょう。いわば人間が機械のふりをしているので、逆向きのチューリングテストと言えるかもしれません。

チューリングテストで最も注目すべき点というのは「問題の言い換え」にあります。定義すら困難な問題に答えるのではなく、検証可能な問題に言い換えた点です。

入学試験などの**選抜試験**について考えてみましょう。あれは、人格を測定しているわけではありません。そのときに試験としてたまたま出された問題で、ほかの人よりも多く得点できた人を選んでいるわけです。定義すら困難な「入学するにふさわしい人を選択する」という問題を、「試験で多く得点できた人を選択する」という検証可能な問題に言い換えているのです。

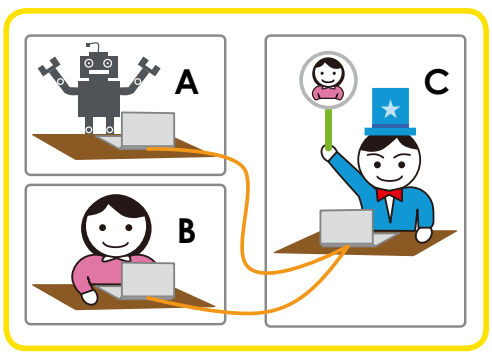
人工知能と試験と言えば、国立情報学研究所が行っている「**東ロボくん**」のプロジェクトを思い出します。そこでは、定義が困難な「ロボットが一定レベルの知能を持つ」という問題を、「ロボットが入試に合格する」という検証可能な問題に言い換え、東ロボくんの2021年度東大合格を目標に研究が進められています。

実は、東ロボくんの目的の1つは、ロボットに知能を持たせることの裏側にあります。東ロボくんの研究を通して、人間が得意な分野を見極めようというのです。ここでも「人間が得意な分野は何か」という問題を、「ロボットは何ができないか」という検証可能な問題に言い換えていますね。

あなたの周りを見回して、「定義すら難しいことを無理に解決しようとしている問題」はありませんか。その問題を、検証可能な別の問題に言い換えることはできないでしょうか。

ぜひ、考えてみてください。SD

▼図2　男性Aを機械に変えたらどうなるか



増井ラボノート

コロンブス日和

第14回　HashInfo

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

Author　増井 俊之（ますい としゆき）　慶應義塾大学

HashInfo

次のような作業をやりたいと思ったことはないでしょうか？

・同じファイルがあちこちにないかチェックしたい
・重要なファイルだけバックアップしたい
・自分のアイデアを日付つきで記録したい

実はこのような仕事は、1つの単純な方法を使って実行できます。

パソコンやWebには膨大なファイルがありますが、同じファイルがあちこちに冗長に置かれていることも多いでしょう。

私の場合、同じファイルを何度もダウンロードしてしまったり写真や動画をいろんな場所にコピーしてしまったりすることがよくありますし、バックアップのつもりでコピーしたデータをさらにバックアップ対象にしてしまった結果、バックアップのバックアップ（のバックアップの……以下同様）のような無駄なデータを作ってしまうことすらあります。

こういう問題を防ぐためには、同じファイルがすでに別のところに存在するか調べれば良いわけですが、すべてのファイルの中身を比較することは現実的ではありません。しかし、ファイルの「ハッシュ値」を利用すれば、あるファイルが独自の（ユニークな）ものなのかどうかを比較的簡単に判別できます。

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

ハッシュ関数とハッシュ値

デジタルデータを一定サイズの数値に変換する関数を**ハッシュ関数**と呼び、計算された値をそのデータの**ハッシュ値**と呼びます。

最近のプログラミング言語ではa['abc']のような**連想配列**が使えるのが普通ですが、これを通常の計算機上に実装する場合、"abc"のような文字列に対してハッシュ関数を計算し、その数値を添字として利用することによって通常の配列と同じように扱う手法がよく利用されています。

ハッシュ値のサイズが小さい場合は、異なるデータから同じハッシュ値が計算されてしまう（ハッシュ値が衝突する）ことがありますが、ある程度大きなハッシュ値を生成する適切なハッシュ関数を用意すれば、ハッシュ値が衝突する可能性はほぼゼロにできます。また、ハッシュ値からもとのデータを計算することもほぼ不可能にできます。

ハッシュ関数はさまざまなものが考えられますが、現在はMD5やsha1というハッシュ関数が広く利用されています。このようなハッシュ関数は次のような特徴を持つため、暗号化アルゴリズムなどで広く利用されています。

・ハッシュ値からもとのデータを知ることはできない
・異なるデータのハッシュ値が同じ値になることはない

ファイルデータから計算されるハッシュ値の情報（HashInfo）をうまく利用することにより、

最初に述べたようなさまざまな有用な機能を実現できます。

重複ファイルの視覚化

ハッシュ値の特徴を利用すれば、手持ちのすべてのファイルのハッシュ値をあらかじめ計算しておくことにより、あるファイルが別のところにすでに存在するかを調べることができます。たとえば、movies/abc.mp4という動画とbackup/xyz.mp4という動画のハッシュ値が同じであれば、abc.mp4とxyz.mp4は同じファイルだということがわかるので、たとえば両方をバックアップする意味はないと判断できます。

図1は、手持ちのパソコンのすべてのファイルのハッシュ値を計算し、それをTreemapという手法で視覚化してみたものです。同じファイルが複数ある場合、そのファイルを赤く表示しています(研究室に所属していた池滝俊太氏の卒業論文による)。

池滝氏は整理が得意なのか、重複するファイルは多くはないようですが、それでもかなりの部分に色がついている(ファイルが重複している)ことがわかります。

ハッシュ情報を複数の人間で共有すれば、自分が持っているファイルを他の人も持っているかどうかがわかります。**図2**は、研究室の数人の学生に協力してもらって、他人のパソコンまで比較対象を広げて同じ計算をしてみたものです。多くの学生はMacを利用しており、Macのシステムファイルは全員に共通であるため、**図1**に比べると左下部分が赤くなっています。

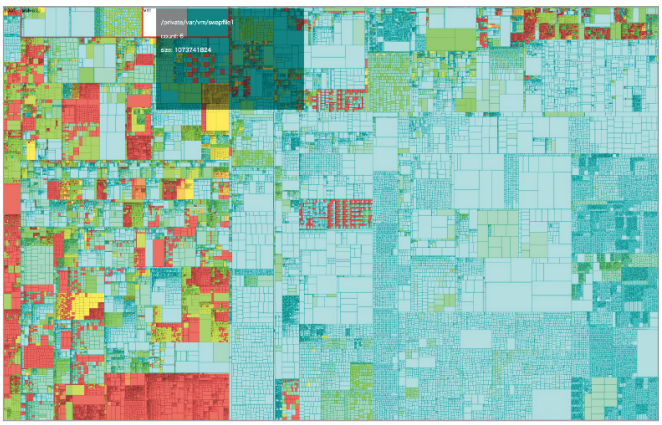
このように、ハッシュ関数を利用すると自分が持っているファイルの素性がなんとなく見えてくると言えるでしょう。ファイルの中身をいくら調べてもこういうことはわからないわけですが、他のファイルや他人のファイルとの比較によりファイルの特徴が見えてくることになります。

重要情報のバックアップ

前述の方法を使うと、複数のマシンでHashInfoを計算することによって、さまざまなマシンで同じファイルが共有されているかどうかがわかります。システムファイルやメジャーなアプリケーションのファイルは多数のマシンで共有されているはずですが、あまり重要でない大きなファイルでもいろんなマシンで共有さ

▼ 図1　自分のパソコン内で重複しているものを赤く表示したもの

▼ 図2　他人とのファイル共有の視覚化



れている可能性があります。

たとえばNode.jsでプログラム開発をする場合、node_modulesというフォルダの下にたくさんのライブラリがダウンロードされますが、このようなファイルはバックアップをする必要はありません。

ブラウザでWebからデータをダウンロードするとき、大きなアプリケーションのプログラムファイルも重要なファイルも同じフォルダに入ってしまい、広く配られているファイルと重要なファイルを区別することは困難ですが、HashiInfoを利用するとその違いが明白になるので、どのファイルが重要なのかがわかるようになります。

一方、コンパイル結果やログファイルなどは重要ではないにもかかわらず、他人が同じファイルを持っている可能性は低いと思われます。

「他人と共有していない」「自動生成されるファイルではない」といった条件を満たすファイルは重要である可能性が高いので、こういう性質をもつファイルだけをバックアップすることにしておけば、無駄なバックアップを防ぐことができるでしょう。

ファイルの存在証明

特許や論文になるかもしれない新しいアイデアを思いついたときなど、ある時点においてある情報を自分が持っていたことを証明したい場合があります。

情報がいつ作成されたかを個人的に覚えておくためにはファイル作成時刻を記録しておけば良いのですが、ファイルの中身や作成時刻は後から偽造できるので、確かにその時点でその情報が存在したということを他人に納得させることはできません。デジタル情報がいつ作成されたか証明するためには、信用できる外部の誰かに認証してもらう必要があります。

ある時点である書類が存在したことを証明するためには昔から「公証役場[注2]」が利用されています。公証役場の業務の1つに「確定日付の付与」というものがあります。これは、私文書に「確定日付」を付与し、その日付にその文書が存在したことを証明するものです。

デジタルデータを印刷して書類にしておけば、その存在を証明するために公証役場を利用できますが、情報を印刷して公証役場に持って行くのは面倒ですし費用もかかります。

最近はデジタル情報の存在を証明するために「電子公証システム[注3]」を利用できるようになったので、公証役場に行かなくても公的な認証を得られるようになりました。これは朗報なのですが、かなりの費用がかかるのが難点です。

ある時刻にあるデジタルデータが存在したことを証明する技術のことをデジタルタイムスタンプ[注4]技術と言います。電子公証システム利用は高価であり利便性の問題もあるので、Seiko[注5]やアマノ[注6]のような会社がデジタルタイムスタンプ技術を提供するサービスを提供しています。

しかし、大事なデータを民間企業に渡すのは心配ですし、サービスがどれほど信頼できるかわかりません。実際、現在運営を中止してしまったサービスもありますし、10年後／20年後も確実に利用できる保証はありません。

電子公証システムや民間の証明システムを利用する場合は証明したいデータの提示が必要ですが、情報が漏れることは心配かもしれません。その時点でその情報が存在することは証明したいけれども情報の中身は公開したくないかもしれません。簡単で良いアイデアを思いついたときなど、特許をとるまで内容は誰にも公開したくないでしょうが、そのアイデアを自分が思いついた日時については記録しておきたいでしょう。

注2）http://www.koshonin.gr.jp/a2.html
注3）http://www.koshonin.gr.jp/de2.html
注4）http://www.imes.boj.or.jp/japanese/kinyu/2000/kk19-b1-4.pdf
注5）https://www.seiko-cybertime.jp/product/easy_time_stamp/
注6）http://www.e-timing.ne.jp/product/timestamp/characteristic/typet/

つまり「情報を持っていることは証明するが、情報そのものは公開しない」方法が必要になります。

HashInfoによるデジタルタイムスタンプ

生のデジタル情報の存在証明を行う代わりに、その情報のハッシュ値の存在証明を行うことにすれば、情報そのものを渡す必要がないのでより安全だと考えられます。

たとえばある問題の解法を発見したとき、たとえば「問題AはBという方法で解決できる」といった文字列のハッシュ値を公証役場や電子公証サービスに登録しておけば、ハッシュ値からもとの文字列を計算することはほぼ不可能なので、文字列そのものを登録しなくても証明を行うことができます。

電子公証サービスや民間のデジタルタイムスタンプサービスを利用しなくても、一般的なブックマークサービスやブログサービスを利用することによって無料でハッシュ情報の存在を証明できるかもしれません。

たとえば秘密情報のハッシュ値がABCDEFという値になるとき、http://example.com/ABCDEFというURLをはてブのようなブックマーク登録サイトや任意のブログサービスに記録しておけば、ABCDEFという情報がどの時点で存在していたかがそれらのサイトで公開されることになりますから、Aという問題を解くための情報が、ある時点で存在したことが証明されることになります。

1つのサービスだけに登録すると、そのサイトの管理者に偽造される可能性もあるでしょうが、関連のないたくさんのサービスに同じ情報を登録しておけば、すべてを偽造することは不可能と考えられるので、その情報がその時間に存在したことをかなり確実に主張できると思われます。

つまり、時刻を証明したい情報があるときは必ず次の手順をとることにしておけば、確実にデータの存在証明を行うことができるでしょう。

・情報を秘密の場所に格納する
・情報のハッシュ値を複数のサイトに登録する

普通のテキストエディタを使っている場合でも、気合いを入れてセーブした場合には上記の処理を自動的に行うようにしておけばいいかもしれません。秘密にしておくべき元データを安全な場所に格納しておけば良いでしょうし、暗号化したデータをクラウドなどにもセーブしておけばさらに安全かもしれません。

HashInfo.net

今回はHashInfoを利用する3種類のアプリケーションを紹介しましたが、多数の人間でファイルのハッシュ値を共有するデータベースがあればさまざまな用途に利用できることが期待できるので、誰もがHashInfoを活用できるようにするためのHashInfo.netというサイトを運用したいと考えています。

一方、こういうサービスではプライバシーの問題に気を付ける必要があります。素性が怪しいファイルを他の誰かが持っていることがわかれば問題になる可能性もありますし、「名寄せ」的に個人が特定される可能性があるので、誰がどのハッシュを持っているかという情報は十分に注意して扱う必要があるでしょう。しかし、プライバシーやセキュリティに十分配慮すればHashInfoが有用な機会は多いはずです。

Googleは、ファイルの中身よりもファイル間のリンクを重視することで検索精度を向上させることに成功しましたが、ファイルの中身よりもファイルの重複を重視することによって大きな効果を得られる可能性もあるでしょう。HashInfo.netの運用などによって有益なシステムを構築できればと考えています。SD

第10回 その土地を知るには地酒から

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

今回はお酒の話、多めで

本連載も気がつけば第10回。毎回、全国各地のオープンソースカンファレンスなど、地域コミュニティでの交流の様子をお伝えしてきましたが、どうもイベントレポート中心になってしまってタイトルバックの絵にそぐわない感じですね。そこで今回は「お酒の話」を少し多めにお送りします。

日本酒のふるさと、島根へ

まずはOSC島根の話からです。今回は9月24日(土)の開催でした。TRONで有名な、坂村健 東京大学教授の基調講演、さらに地元の学生のみなさんが出展するブースが並ぶなど、普段の開催よりもたくさんの人が集まり、盛会となりました。

私自身、前日に松江工業高等専門学校を訪問させていただいたり、ブース出展をしてくれた学生さんを懇親会に招待したりして、日常の活動について話を聞かせてもらうなど、学生のみなさんとの交流を中心に楽しみました(**写真1**)。

また、OSC島根の運営を地元コミュニティのみなさんにお任せしているので、私も展示ブースでRaspberry Piを使ったデジタルオーディオプレーヤのデモをしたり、自社で行った夏の学生インターンシップの発表(Raspberry Piを使った電子工作)を手伝ったりと、出展側にまわることができ、たいへん充実した開催となりました。たまには一参加者としてOSCを楽しめるのも、小規模開催のよいところです。

酒どころ、新潟・長岡もいいところ

さらに翌週、10月1日(土)は新潟・長岡でOSCが開催されました。新潟市での開催が多いのですが、今回は久しぶりの長岡市での開催です。長岡駅からほど近いこともあって100名以上が参加し、和やかな中にも活気のある開催となりました。

この開催でも、私の主たる目的は学生のみなさんとの交流です。地元の長岡技術科学大学、長岡造形大学からの参加はもちろん、遠く山形県米沢市から山形大学の学生さんたちが4名参加してくれました(**写真2**)。OSCに参加してみたかったので、車ではるばる来てくれたのだそうです。これだけの行動力、若さってすばらしいですね！ 彼らはOSC東京にも参加することになったので、この欄に再登場するかも？

また、地元のエンジニアのみなさん向けに、前日にはOpenStack、当日はDockerのお話をさせていただきました。内容は超入門レベルですが、このような活動がどこかで成果に結びつくとうれしいですね。

その土地を知るには地酒から

それでは、お酒の話を。島根も新

▼写真1 松江高専のみなさんと。先輩後輩の仲がとてもいいですね

▼写真2 山形大学のみなさんと2次会。OSCを楽しく満喫してくれたようです

▼写真4　前から一度来てみたかった李白酒造さんを訪問。生酛造りを購入

▲写真3　OSC長岡の懇親会には、豪華な日本酒がずらりと並びます。写真を撮っているのは日本MySQLユーザ会の坂井恵氏

▼写真5　右手にひやおろし、左手に長期熟成酒。OSC長岡の前夜祭にて

潟も、どちらも日本酒の美味しい地域です(**写真3**)。島根は国造り神話が残るぐらい古くからの歴史のある地域ですから、当然お酒の歴史も古いです。ご存じの話として、「八岐大蛇を退治するためにお酒を飲ませて眠らせてしまった」という言い伝えが残っています。

さすがに神話の時代からのお酒造りは残っていませんが、今に残る酒蔵さんはだいたい江戸から明治・大正時代のころ、始まっているところが多いようです。

今回はお城から少し離れて、水の良い場所で酒造りをしている李白酒造さんを訪問、試飲させていただきましたが、こちらは明治15年の創業。「李白」の酒名は昭和3年、若槻礼次郎元首相の命名だとか(**写真4**)。地元の歴史を紐解くために酒蔵を訪問してみるのは楽しいものです。

日本酒の秋、ひやおろしの秋

日本酒の旬は1年中ですが、秋は「ひやおろし」の季節です。ひやおろしは、夏の間熟成させた日本酒を火入れ(加熱殺菌)せずに飲む、生酒に近い(貯蔵前に一度だけ火入れしている)日本酒です。9月から11月がひやおろしの季節ですが、2回目の火入れをしていないので、さらにゆっくりと熟成が進んでいくため、9月と11月に飲む味が異なってくるというおもしろさもあります。

いろいろな種類のひやおろしが出回るので、通常の銘柄と飲み比べをしてみるのも楽しいですね(**写真5**)。ひやおろしはコクのある味わいのものが多いので、秋の味覚、たとえばサンマに合わせて飲むとさらに味わい深いものがあります。本誌が出るころにはそろそろひやおろしも終わりの時期ですが、「晩秋旨酒」のひやおろしを楽しんでみてください。SD

Report 島根は出雲そば、新潟はへぎそば

島根と新潟は、日本酒だけでなくお蕎麦も美味しい土地です。

島根のお蕎麦は出雲そばと呼ばれますが、「割子」という丸くて小さなお重に入れて、いろいろな薬味をかけて食べるのが特長です。お重はだいたい3段(3枚)ですが、5段のメニューもあります。私も以前は5段でしたが、体重を気にするようになって3段になりました。

新潟のお蕎麦は、「布海苔」をつなぎに使った「へぎそば」が有名です。海苔のおかげで強いコシがあり、喉ごしがツルッとしているのが特長です。「へぎ」というのは漁で採った魚を入れる木の箱のことで、今でも頼むとへぎに載せて出してくれます。同じお蕎麦なのに、土地が変わるとずいぶんと違うものですね。

3枚割子蕎麦。真ん中は舞茸の天ぷら。下に薬味があります

小嶋屋さんのへぎそば4人前。いくらでも食べられる危険な味わい

ツボイのなんでもネットにつなげちまえ道場

第18回 mbed Device Connectorを使ってみる

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

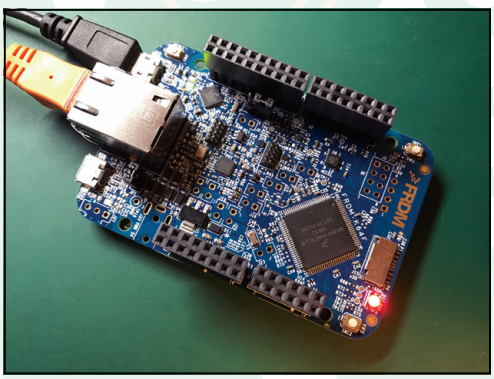
はじめに

今回はmbedをネットにつなげる方法について記したいと思います。mbedでは、「mbed Device Connector」というクラウドサービスが開発者向けに提供されていて、「100デバイス、1時間あたり10,000イベント」という制限はあるものの、この範囲であれば無償で利用できます。mbed Device Connectorに接続するクライアントはmbed Clientと呼ばれており、mbed OS 5のリリースに含まれています。

mbed Clientを使うサンプルは、mbed-os-example-clientとして公開されています。今回はこれを使って、mbed Device Connectorとはどんなものか試してみることにしましょう。

といっても、mbed-os-example-clientは、まだmbed LPC1768ではうまく動かせるようになっていません。サンプルのページに記されているFRDM-K64F（**写真1**）や、サンプルのmbed_app.jsonに記載のあるNUCLEO-F401REやNUCLEO-F411REでは動くようです。ですので、今回はFRDM-K64Fを使ってみます[注1]。ちなみに、mbed LPC1768のRAMは64KBですが、FRDM-K64Fは256KBと4倍です。新しいmbedをオンラインコンパイラで利用できるようにするには、ボードのページ[注2]の右カラムにある「Add to your mbed Compiler」というボタンをクリックします。

▼写真1　FRDM-K64F

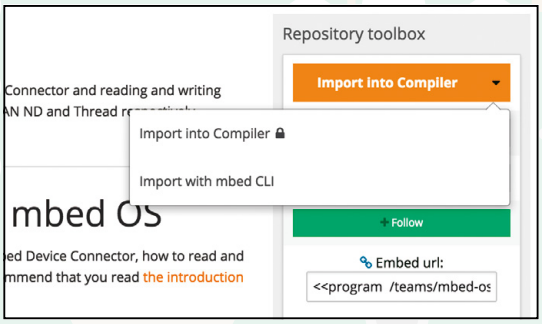


mbed-os-example-client

では、mbed-os-example-clientをオンラインコンパイラでビルドして、mbed Device Connectorを使ってみましょう。まず、mbed-os-example-clientのページ[注3]にアクセスし、オンラインコンパイラにインポートを行います。ページ右上にある「Import into Compiler」ボタン（**図1**）をクリックするだけでインポートができます。ボタンが「Import with CLI」という表記になっている場合、ボタン右の▼をクリックすれば、「Import into Compiler」に切り替えられます。

注1）http://ssci.to/1689　6,210円
注2）https://developer.mbed.org/platforms/FRDM-K64F/
注3）https://developer.mbed.org/teams/mbed-os-examples/code/mbed-os-example-client/

▼図1　インポートのボタン



こうしてサンプルプロジェクトをオンラインコンパイラにインポートしたところで、ソースコードの編集を行いましょう。このサンプル(**図2**)は、ネットワークのインターフェースとして、Ethernetのほかに Wi-Fiやメッシュネットワークをサポートしています。mbed_app.jsonを開くと、使用するインターフェースの選択や、Wi-Fiインターフェースを使う場合の設定、あるいはFRDM-K64F以外のmbedを使う場合のピンの設定が書かれているのが確認できます。今回は説明をシンプルにするため、FRDM-K64Fに標準でついているEthernetを使って接続をしますので、とくに設定の変更は必要なさそうです。

次に、security.hを見てみましょう。最後のほうの行に、

```
#error "You need to get security.h credentials from connector.mbed.com and replace the content of this security.h file"
```

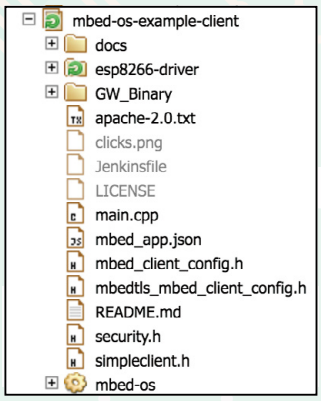
と記述があるように、証明書をここに書き込まなければコンパイル時にエラーが起きるようになっています。個人的には、IoTのエンドノードに書き込む証明書がハードコードされるという点に疑問を感じたりもしますが、あくまでこれはサンプルだと理解して、とりあえず動かしてみましょう。

説明のとおり、証明書はconnector.mbed.comにWebブラウザでアクセスをして入手します。developer.mbed.orgとアカウントは共通ですので、ログインしてください。ログインしたところで、左側にあるカラムの「My devices」の下、「Security credentials」をクリックします。次に、「GET MY DEVICE SECURITY CREDENTIALS」というボタンをクリックすると、**図3**のようにデバイスセキュリティ証明書が表示されます。このテキストをクリップボードにコピーして、先ほどのsecurity.hにペーストしてください。

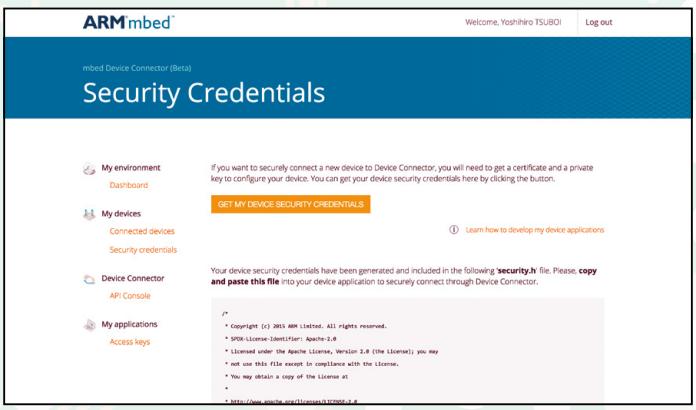
ここまで済ませたら、サンプルプログラムのビルドができるはずです。ビルドをしてダウンロードしたバイナリファイルをFRDM-K64FのMBEDドライブにコピーして、マイコンに書き込んでください。FRDM-K64Fをパソコンと接続するUSBコネクタは、**写真1**でケーブルが接続されている側のものです。FRDM-K64Fも、mbed LPC1768と同様に、USBで接続したパソコンからシリアルポートが認識されているはずです。このシリアルポートを、パソコンのシリアルターミナルソフトウェア(**図4**)から115,200bps、データビット8bit、パリティなし、ストップビット1bit、ハードウェアフロー制御なしの設定で開いてください。初めてWindows PCからmbedを使う場合、ドライバ[注4]のインストールが必要です。また、筆者はWindowsでは

注4) https://developer.mbed.org/handbook/Windows-serial-configuration

▼図2　インポートしたソースコード



▼図3　デバイスセキュリティ証明書



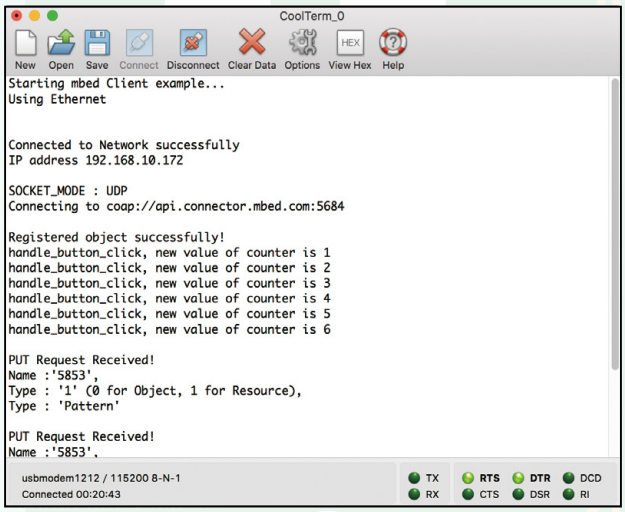
Tera Termを、MacではCoolTerm[注5]を使用しています。

ビルドしたバイナリファイルのFRDM-K64Fへの転送と、シリアルターミナルでシリアルポートへの接続、またEthernetコネクタを接続したところで、FRDM-K64Fのリセットボタン(パソコンと接続しているUSBコネクタのすぐ横のボタン)を押します。しばらくすると、**図4**のように、DHCPでIPアドレスを取ってきて、mbed Device Connectorに接続し、「Registered object successfully!」と表示されます。このメッセージから読み取れるように、mbed Clientは、mbed Device ConnetorにCoAPというRFC 7252として公開されているIoTやM2M(Machine to Machine)向けのプロトコルで接続しています。

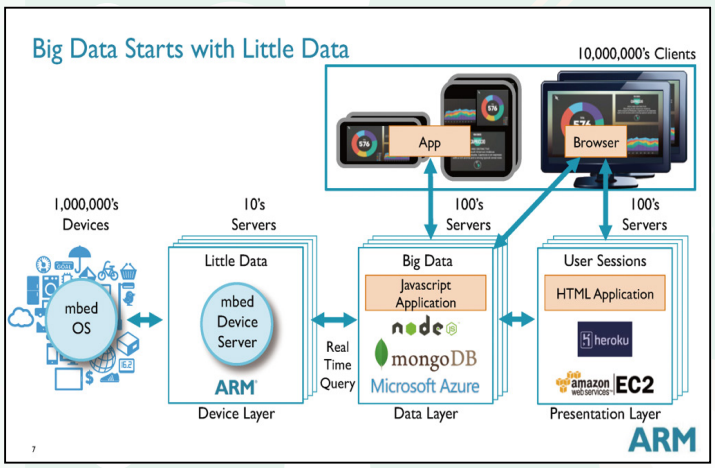
さて、ノードのほうは接続できたと言っていますが、サーバ側でも接続がなされているか確認してみましょう。先ほど証明書を取得したmbed Device ConnectorのWebサイトにアクセスをし、「Dashboard」というボタンをクリックします。こうしてダッシュボードを見ると、「My devices」というところが「1 of 100」となっています。冒頭で記したとおり、mbed Device Connectorは、ノードを100台まで接続できるのですが、うち1台がつながっているということが確認できます。また、「My devices」の下にある「Connected devices」をクリックすると、ノードに個別に付与されるIDと状態が一覧できます。

注5) http://freeware.the-meiers.org

▼図4 シリアルターミナルの画面



▼図5 mbed Device Connectorの位置づけ



※ARMのプレゼンテーションより抜粋

mbed Device Connector

mbed Device Connectorは、その名のとおり、デバイスを接続するためのものです。mbed Device Connectorでノード(デバイス)とサーバを接続することはできますが、データの蓄積や加工はARMのプレゼンテーションスライド[注6](**図5**)にもあるように、ユーザアプリケーションで行います。

先ほどノードのサンプルをビルドしてみましたので、こんどはアプリケーション(サーバ)側のサンプルを動かしてみましょう。mbed-connector-api-python-quickstart[注7]というPythonで書かれたサンプルが提供されています。これをGitHubからダウン

注6) このプレゼンテーションのころは、mbed Device Connectorは、mbed Device Serverと呼ばれていました。

注7) https://github.com/ARMmbed/mbed-connector-api-python-quickstart

ロードするか、gitコマンドでcloneし、Python(2.7系)をインストールしたパソコンで動かします。まず、Pythonのライブラリがいくつか必要ですので、

```
$ pip install -r requirements.txt
```

を実行し、サンプルプログラムと一緒に配布されているrequirements.txtに書かれているライブラリをインストールしておきます。

こちらも、mbed Device Connectorと接続するために認証キーが必要ですので、まずそれを用意しましょう。パソコンのWebブラウザでconnector.mbed.comに接続します。そこで、左側カラムにある「My applications」の下、「Access keys」をクリックしてください。ここで「CREATE NEW ACCESS KEY」というボタンを押してアクセスキーに名前を付け、「ADD」ボタンを押すと認証キーが発行されます(**図6**)。

この画面では、「ftf2016」という名前が付いていますが、これは筆者がFTFというイベントで行われたワークショップを冷やかしていたときにつけた名前ですので、深い意味はありません。

さて、このキーは、mbed-connector-python-quickstartのapp.pyに、

```
token = "ChangeMe"
```

と書かれた行がありますので、この「Change Me」に先ほど認証キーを発行して得られた文字列をペーストして書き換えます。

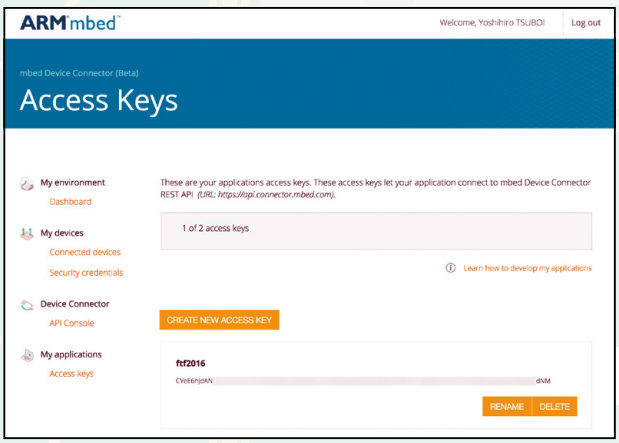
app.pyの編集を終えたら、保存し、“python ./app.py”などとして実行します。実行を開始したところで、同じパソコンのWebブラウザで、**http://localhost:8080**にアクセスをします。すると、**図7**のようなWebアプリケーションが開きます。

「Subscribe」にチェックを入れ、「GET」ボタンをクリックしてください。それから、FRDM-K64Fの「SW2 INT1」スイッチ(**写真1**で基板の右手奥にあります)を押すと、Webアプリケーションの「Presses」の右にあるカウントが増えていきます。シリアルコンソールを開いていれば、こういった操作をしたことはマイコンのシリアル出力でも確認できます。

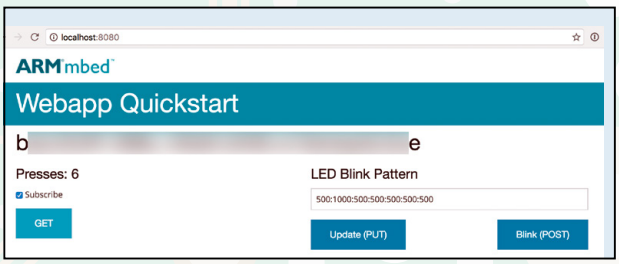
アプリケーションまで用意しなくとも、mbed Device ConnectorのWeb画面で、ちょっとしたAPIのテストを行うことができるように作られています。FRDM-K64Fをネットワークに接続して動かしているときに、connector.mbed.comの「API Console」にアクセスすると、APIを手軽に操作できます。

こうして、mbed OSのmbed Clientとmbed Device Connectorを使って、ちょっとしたエンドノードの実験をしてみました。とくに何も用意しなくても、mbedだけでIoTデバイスとサーバの実験ができるのはたいへん便利です。SD

▼図6　認証キーを発行する画面



▼図7　Web App Quickstart



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**2016年12月15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 Wi-Fi ホームルータ「Aterm WG2600HP2」 1名

Wi-Fiホームルータ「Aterm」シリーズのフラッグシップモデル。5GHz帯、2.4GHz帯ともに4本のアンテナを利用する4ストリームに対応し、業界最速※の実効スループットとなる約1,428Mbps（UDP）、約1,151Mbps（TCP）を実現しました。規格はIEEE 802.11ac。

※日本国内メーカーの家庭用Wi-Fiホームルータとして（2016年10月3日、NECプラットフォームズ調べ）。

提供元 NECプラットフォームズ　http://121ware.com/aterm

02 ウイルスバスター クラウド +デジタルライフサポート プレミアム 1年版

総合セキュリティソフト「ウイルスバスター」の最新版です。パソコン／スマフォ／タブレット（Windows、Mac、Android、iOS、Fireタブレット）を最大3台まで保護できます。24時間365日、ネット接続トラブルなどのサポートが受けられる「デジタルライフサポート プレミアム」付き。

提供元 トレンドマイクロ
http://www.trendmicro.co.jp

1名

03 Qiita ノベルティTシャツ（Mサイズ） 2名

プログラマのための技術情報共有サービス「Qiita」のノベルティTシャツです。ブルーかグレイ、どちらか1枚をプレゼント。

提供元 Increments　http://increments.co.jp

04 Ansible 徹底活用ガイド

平 初 ほか 著

構成管理ツールとしてAnsibleを選ぶべき理由から、導入方法の紹介、応用までを解説。Webメディア「Think IT」の連載記事「注目の構成管理ツールAnsibleを徹底活用する」を再編集した1冊です。

提供元 インプレス
https://www.impress.co.jp

2名

05 確かな力が身につくPHP「超」入門

松浦 健一郎、司 ゆき 著

PHPの入門書です。ショッピングカートやログイン処理、商品管理を行うDBの設定・制御法といったコマースサイトに必要となる機能の実装を中心に、サンプルコードを示しながら解説します。

提供元 SBクリエイティブ
http://www.sbcr.jp

2名

06 インフラエンジニアの教科書2

佐野 裕 著

実務経験を数年積んだインフラエンジニアをおもな対象に、プロトコル、データベース、セキュリティ、RFCの読み方まで、インフラエンジニアとしてのステップアップに必要となる知識をわかりやすく解説。

提供元 シーアンドアール研究所
http://www.c-r.com

1名

07 ポートとソケットがわかればインターネットがわかる！ 小川晃通 著

コンピュータ同士で通信をする際の重要な概念「ポートとソケット」を手がかりに、aicoさんの可愛いイラストで、Unix、TCP/IPといったインターネットのしくみをわかりやすく解説します。

提供元 技術評論社
http://gihyo.jp

2名

適材適所で活用していますか?

NoSQLの教科書

MongoDB、Couchbase、Redis、MySQLでNoSQL!

もはやRDBの対抗軸としてのNoSQLではなく、「適材適所でNoSQLを使う」ようになってきました。本特集では、NoSQLのしくみについてKey-Value StoreやドキュメントDBといった基礎技術の確認を最初に行います。その後、人気の高いMongoDB、ダークホースであるCouchbase Server、定番的なRedisを解説します。そしてMySQLでもNoSQLを実現できるmemcachedプラグインをMySQL Clusterとともに紹介します。これらNoSQLの実装を多面的に確認することでデータストアとしての活用方法を会得してください。

NoSQL 第1章

NoSQLの基本構造を理解する

ハッシュテーブルとドキュメント指向

Author 力武 健次（りきたけ けんじ）　力武健次技術士事務所 所長 http://rikitake.jp/

本記事ではNoSQLの主流として普及しているキーバリュー型ストレージ（KVS:Key-Value Store）の要素技術である「ハッシュテーブル」、分散ノードで運用するKVSである「分散ハッシュテーブル」とその要素技術である「コンシステントハッシュ」、そしてKVSのバリューの部分に自由な構造を持つことができる「ドキュメント指向」データベースについて紹介します。

原始的なKVSとしての配列とその限界

KVSとは、キーとキーに対応する値を組み合わせた表のことを言います（**図1**）。KVSでは、何らかのキーを与えると、テーブル（ストレージ中の表）の対応する位置にあるデータの読み書きができます。この条件を満たす最も単純なデータ構造は、キーをテーブル上の番地に限定した「配列」です。たとえばC言語の配列は、同じ型の要素をn個持ち、キーは0からn-1までの整数であるKVSと言い換えることができます。しかし、C言語の配列のような静的なデータ構造をKVSとして使うには、次の欠点があります。

・キーに文字列など任意の情報が使えない。キーと配列上の番地との対応づけを行うには、それ専用の配列が必要であり、なおかつキーに対応する番地を毎回探索する必要があるため、効率的ではない[注1]
・テーブルの大きさを自由に変えることができないか、著しく困難である
・1つのキーに1つの要素しか保持できないため、重複したキーを持つ情報を保存できない
・キーに対応する値のデータ構造を自由に変えることができない

▼図1　キーバリュー型ストレージ（KVS）の例

KVSはキーと対応する値を持つ表（テーブル）と考えることができる

キー	値
ユーザ名	jj1bdx
ディスク使用量	255.5
使用言語	Erlang

JSONによる表記

```
{
"ユーザ名": "jj1bdx",
"ディスク使用量": 255.5,
"使用言語": "Erlang"
}
```

より柔軟なKVSであるハッシュテーブル

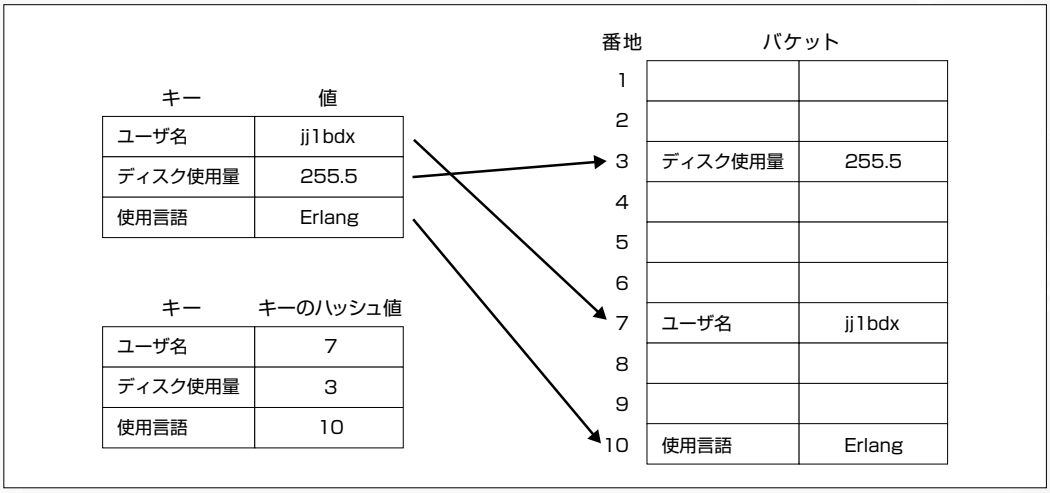
C言語の配列のようなデータ構造は定型的作業には有用ですが、データベースとして使うにはより自由度の高い文字列などをキーとして使える必要があります。そこで多くのKVSでは、「ハッシュ」という任意の情報を一定範囲の整数値に変換する手法を使うことで、任意の情報をキーとして使えるようにしています。このようなデータ構造のことを「ハッシュテーブル」

注1）線形探索ではn個の要素に対し最大でn回比較する必要があります。二分探索法を使って探索の時間を減らすことはできますが、二分探索法の場合はn個の要素に対して$\log_2 n$回の比較が必要になります。ハッシュ関数を使った場合はこれより速い結果が得られる可能性があります。

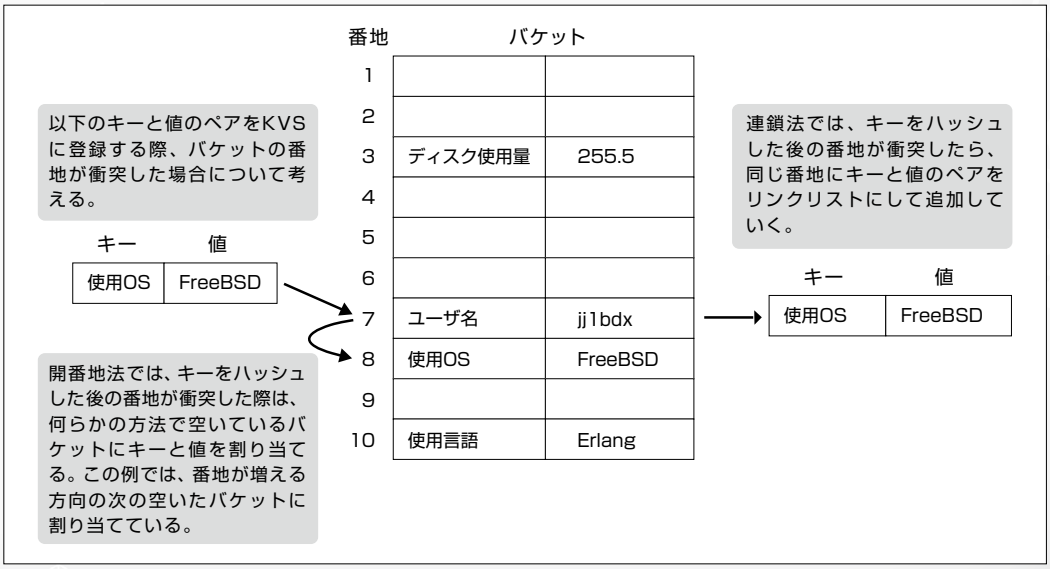
と呼びます。ハッシュテーブルは、AWKの連想配列、PerlのHash、Pythonの辞書、Luaのテーブル、JavaScriptのオブジェクト、ErlangやElixirのマップなど、多くの言語に標準機能として備わっています。

ハッシュテーブルは、中にキーと値の情報を保存する「バケット」を多数持っています。アクセスする際は、キーを一度「ハッシュ関数」で整数値に変換し、保存するバケットの番地を求めてから、要素の追加や書き換えなどの操作を行います（**図2**）。このハッシュ関数の役割は、キーを変換した後の値が同じになる（衝突する）確率をできるだけ下げることにあります。衝突がなければ、キーから番地は一度の計算で求まるため、高速にデータの読み書きができます。しかし、キーが衝突した場合は、同じバケットの番地に2つ以上のキーが存在することになるため、何らかの方法でキーと番地の対応づけを別途行うことが必要になります。代表的な方法としては次の2つがあります（**図3**）。

▼図2　KVSをハッシュテーブルで実装した例



▼図3　KVSでバケットが重複した際の回避方法



・同じ番地に属するキーと値の組を登録したリンクリストを各バケットに用意し、番地の重複するキーを持つデータを追加していく（連鎖法）
・衝突したバケットの番地を始点として、その番地から別の関数を使って空いているバケットを探し、そこにキーと値の組を追加する（開番地法）

どちらの方法でも、情報を書き込んでいけば空いているバケットがなくなってしまうため、書き込み効率を維持するには再度別のハッシュ関数を使用してバケットの数を増やして再割り当てする（再ハッシュする）必要があります。運用上は、バケットが全体数の半分程度埋まった時点で、再ハッシュしてバケット数を倍に拡張するのが良いようです[1]。

ハッシュと他の方法の比較

ハッシュテーブルは、十分なバケット数が用意できていれば、平均探索回数を少なく抑えられるため高速に扱うことができます。またハッシュ関数は同じ入力に対して同じ出力を返すことに特化した関数です。ハッシュテーブルをWebのキャッシュとして使う場合などは、キーが完全一致していれば良いため、その特徴を十分に活かすことができます。

一方、KVSであっても、キーの文字列としての順番を意識した部分検索が必要になる場合は、Trie（トライ木）[2]など文字列の特徴を反映したデータ構造を使ったほうが検索効率を上げられる場合があります。

分散ハッシュテーブルとコンシステントハッシュ

一般にデータベースをキーの値に応じて複数のノードに分割して管理することをシャーディング（sharding）といいます。分散ハッシュテーブル（DHT：Distributed Hash Table）はKVSに対するシャーディングの1つの方法で、複数の分散ノードが一体となってハッシュテーブルを構成します。

シャーディングではキーをどのノードに対応づけるかが重要なポイントになります[注2]。DHTでは、キーをハッシュした値に応じて、キーと値のペアであるバケットがどのノードに属するかを決定します。

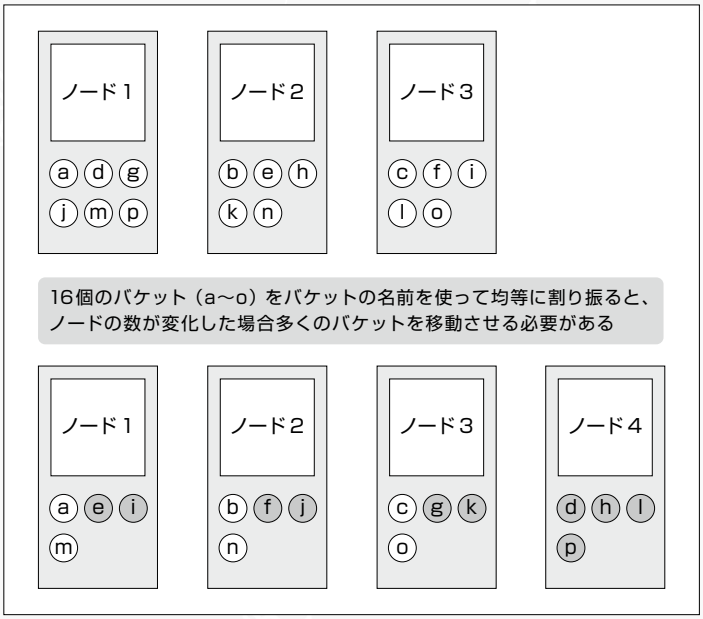
分散環境では、ノードがアクセスできなくなったときと復旧したときにどのように全体を再構成するかという問題があります。単純にノード数でバケットを分割してしまうと、ノード数が変わるとバケットをほぼすべてのノードで移動させなければならなくなるため、ノード間通信のオーバーヘッドが大きくなってしまいます（**図4**）。この問題を解決する手法の1つが「コンシステントハッシュ」です。

コンシステントハッシュでは、バケットは特定の「仮想ノード」に属します。この仮想ノードとバケットの対応はノードが増えても減っても変わることはありません。1つのノードには多数の仮想ノードが属しています。この方法であれば、ノード数が増えても仮想ノードとノードの対応付けを変えるだけでよく、（仮想ノード数／ノード数）分の仮想ノードを移動させるだけで済みます（**図5**）。

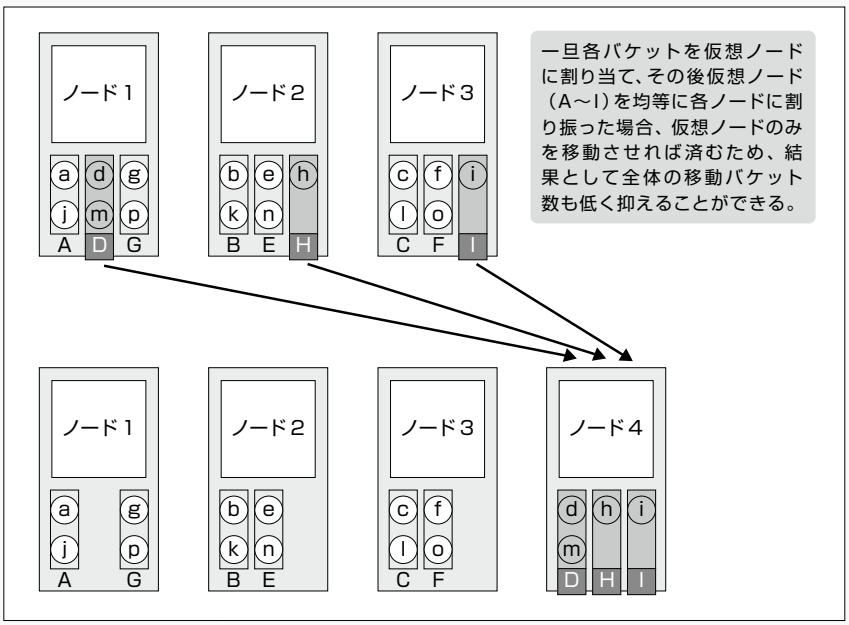
現実には、DHTのノードに障害が発生した場合はデータも失われるため、DHTの設計には各バケットの複製を複数ノードに分散して取ること、一時的な障害で落ちているノードへの書き込み内容は複製のための他ノードで記録しておいて復旧したときに本来のノードに戻すこと（ヒンテッド・ハンドオフ、hinted handoff）[3]などの数々の障害復旧手法が用いられています。

注2） 単純にキーの値を使って分割するという方法（例：1文字目の内容でノードを分ける）もありますが、このような固定的な方法では障害が発生してノードのデータが失われた際のデータ復旧が困難という問題があります。

▼図4　仮想ノードを使わずにノード間でバケットを分割した場合[8]



▼図5　仮想ノードにバケットを割り当てたうえでノード間で仮想ノードを移動させた場合[8]



「ドキュメント指向」とは

今まではKVSのキーと値の関係をとり持つハッシュテーブルについて説明してきました。ここからはKVSの値の中をどう扱うかについての手法である「ドキュメント指向」データベースについて紹介します。

「ドキュメント指向」の「ドキュメント」とは、一般的な「文書ファイル」という意味ではなく、厳格な文書構造を持たないデータの総称として使われています。筆者が調べた範囲では、それ以上のはっきりとした定義はないようです。あえて定義を設けるならば、何らかの構造を持つデータの総称と言い換えてもいいでしょう。具体的には後述のJSONのように、要素名とその要素に対応する値を列挙できるデータ構造を指しているようです。

従来のリレーショナルデータベース管理システム（RDBMS）では、テーブルは列（カラム）と行（ロウ）を持つ1つの表として扱っています。それぞれの列は行のフィールドとして型やデータ長に至るまで厳格に定義され、テーブル全体で統一した規則（スキーマ）を持っています。このようにしてテーブルの中にある情報の関係（リレーション）を定義しています。

一方、ドキュメント指向のデータベースでは、統一したスキーマを持つ必要がありません。ド

キュントの中身は、MongoDBならBSON[4]注3、CouchbaseならJSONと、JavaScriptのオブジェクトであればどのようなものでも受けいれるようになっています。たとえばMongoDBではRDBMSのテーブルにあたるものは「コレクション」として管理されますが、コレクションの中の各ドキュメントは一定の構造を持つ必要はありません。ドキュメントの中に別のドキュメントが入れ子になることも、またドキュメントの中にコレクションのキーが入ることも自由です。

◆ ◆ ◆

では、ドキュメント指向データベースは何を目指しているのでしょうか。代表的な実装である、MongoDBやCouchbase、そしてCouchDB[6]に共通しているのは、すべてのデータに識別用のID（RDBMSのテーブルにおける主キー（primary key）と同義）が付いているものの、検索についてはIDを基本的には使わずに注4、ドキュメントそのものに対する検索を行うことを前提にしていることです。具体的には、ドキュメント内の各項目に対して検索条件を設定し、それをデータベースに与えて検索結果を得る機能が用意されています。つまり、データベースとはドキュメントの並びであり、データベースの操作とはドキュメントに対する操作であるという前提を満たすべく、ドキュメント指向データベースは作られていると考えることができます。

検索に見るKVSでのデータの整合性の問題

ドキュメント指向データベースを含むKVSで情報検索を行う際に問題になるのは、データ構造を定義するスキーマが明示的に存在しないため、検索の完全性を目指すには全数検索を行うしかないということです。言い換えれば、データベースが持つ情報全体に対して検索を行うことが前提になっているといえます。この前提の実現にあたっては、昨今の全文検索エンジンの高速化を考えれば、十分に現実性はあるでしょう。JSONのようにある程度構造を持ったデータを扱うなら、その構造を利用してさらに効率の良い検索はできます。しかし、データが常に追加されたり変更されたりするデータベースにおいては、完全な検索を仮定するのは本質的に不可能です注5。そのような状況下では、検索結果についてもある程度の不確定性が伴うのは不可避だと言えるでしょう。

別の考え方をしてみます。本当に「漏れがなく過剰でもない完全な検索」は「毎回」必要でしょうか？　たとえばKVSをWebのデータキャッシュとして使うような場合は、そのキャッシュが完全である必要はそもそもありません。検索したデータがキャッシュに存在しなければ、元の情報をキャッシュに入れるだけのことです。また、ある条件を満たす検索結果をすべて用意する必要は必ずしもなく、そのうちのいくつかを得れば済むような場合もあります。このような検索を行う際は、全数検索をするのではなく、必要な数の検索結果を得たら検索処理を終了するように設計すべきでしょう注6。

おわりに

本記事ではハッシュテーブル、DHTとコンシステントハッシュ、そしてドキュメント指向データベースについて紹介しました。

注3）BSONではJavaScriptのオブジェクトであるJSONと表現の互換性を保ちつつ、バイナリデータによる効率的な表現に加え、オブジェクト中の各要素の型を明示したり、オブジェクトの大きさをオブジェクト自身の前に前置するなど、データの検索をより高速に行えるようにしています。

注4）実際にはIDをデータベースアクセス用URLの一部として明示的に使えたり、管理用のIDが予約されているなど、IDを使うことを前提としている機能もあります。また、一度IDを得たデータベースの内容については、そのIDを使って更新や消去といった操作ができます。

注5）トランザクション動作の可能な多くのRDBMSでは、データの検索を行うSQLのSELECT文の実行をトランザクションとして読み出しの整合性を維持したまま検索を行うことができますが、これとてデータベースの情報量が大きければ、トランザクションのために他の更新作業を停止した時間が長くなり過ぎて、ほかの作業に支障が出てしまいます。

注6）必要な個数だけ検索結果を得る際に何も特別な作業をしなかったときの動作はデータベースの実装に依存します。どんな結果が欲しいかについては検索条件に漏れなく記述しておかないと、予想外のデータが出てきてしまうことがあります。

KVSを含めたNoSQLでは、障害に対する可用性や速度を重視するあまり、データの整合性が緩く規定されていて、そのことがしばしば問題になります。DHTなどの分散環境でノード間の整合性が取れないのは日常茶飯事であり、修復が必要になることも珍しくありません。整合性の問題は多岐にわたりかつ解決困難な問題です。情報検索の問題を1つ取ってみても、データの整合性を実現することがいかに難しいかを考えざるを得ません。

しかし別の見方をすれば、データの整合性の制約を緩めれば処理を楽にすることも不可能ではないということになります。Amazon.comの研究者はこの問題について2007年に考察した論文を発表しており[7]、KVSを業務で使う際は必読と言えます[注7]。

ドキュメント指向データベースやその他のKVSを使う際は、そこにある情報が「完全なもの」かどうかに関する制約を緩めれば処理が楽になること、そしてその制約の緩さゆえに不都合が発生した場合どのような処理をするかを考えておくことが重要です。

データベースの技術は日新月歩の勢いで進んでいます。最近ではSILO[9]という、多数のCPUコアをノード内に抱えたメニーコア環境を前提とするトランザクションの高速化手法が提案されています[注8]。SILOのような技術が製品に実装されれば、NoSQLも、RDBMSも、その形を大きく変えてくるでしょう。そういう意味では、データベースをどう使うのかという根本的な問題を考えてシステム設計をすることがますます重要になるだろうと筆者は考えます。

SD

注7）Basho Technologies の Riak（URL http://docs.basho.com/）は、このamazon.comの論文に出てくるDynamoというKVS（Amazon DynamoDBとは名前は似ていますが無関係です）の考え方を実装したものです。アリエルネットワークの技術コラム「DHTからAmazon Dynamoまで」（2010年）にて、DHTとDynamoについての概要を網羅した日本語の解説があります。URL http://dev.ariel-networks.com/column/tech/amazon_dynamo/

注8）SILOについては、ノーチラス・テクノロジーの神林飛志氏が、その背景と技術的に重要な点について日本語で詳説しています。ぜひ一読をお勧めします。URL http://d.hatena.ne.jp/okachimachiorz/20161003/1475494676

参考文献

[1] “Five Myths about Hash Tables”, https://hughewilliams.com/2012/10/01/five-myths-about-hash-tables/
[2] https://en.wikipedia.org/wiki/Trie
[3] 「ヒンテッド・ハンドオフ：書き込みパス中のリペア」、DataStax Apache Cassandra™ 3.0（Linux）日本語ドキュメント、http://docs.datastax.com/ja/cassandra-jajp/3.0/cassandra/operations/opsRepairNodesHintedHandoff.html
[4] http://bsonspec.org/
[5] “Data Model”, Couchbase 4.5 Server, http://developer.couchbase.com/documentation/server/4.5/data-modeling/concepts-data-modeling-intro.html
[6] “Storing Documents”, CouchDB: The Definitive Guide（Draft）, http://guide.couchdb.org/draft/documents.html
[7] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: amazon's highly available key-value store. In Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles (SOSP '07). ACM, New York, NY, USA, 205-220. DOI=http://dx.doi.org/10.1145/1294261.1294281（論文の写しのPDFのURL: http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf）
[8] Tyler Hannan, Basho Technologies, "O'Reilly Webinar: Simplicity Scales - Big Data", http://www.slideshare.net/BashoTechnologies/oreilly-webinar-simplicity-scales-big-data
[9] Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden. 2013. Speedy transactions in multicore in-memory databases. In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13). ACM, New York, NY, USA, 18-32. DOI: http://dx.doi.org/10.1145/2517349.2522713（論文の写しのPDFのURL: http://people.csail.mit.edu/stephentu/papers/silo.pdf）

第2章

MongoDB使いにならないか？

多機能データストア MongoDB入門

Author 桑野 章弘（くわの あきひろ）
Twitter：@kuwa_tw　mail：akihiro.kuwano@gmail.com　Web：http://akuwano.hatenablog.jp/

NoSQLとしてよく話題にあがるMongoDB。本章では、MongoDBの特徴を押さえたうえでインストール、そして簡単なクエリの実例を学び、地図情報と連携して使うちょっとした応用例まで学びます。MongoDBの優れた機能をぜひ体験してみてください。

MongoDBとは何か？

こんにちは！　桑野と申します。目黒でカレーを探しながら働いているものです。

この章ではMongoDBというデータストアについて説明していきます。MongoDB[注1]は、MongoDB社[注2]が開発しているオープンソースのデータストアです。MongoDBという名前の由来は、「ばかでかい」という意味の英単語“huMONGOus”からきています。実際にインターネット上で名前だけは聞いたことがある、という人で、実際にどのようなデータストアなのかわからないという方も多いのではないでしょうか。たとえばSNSなどでは開発しやすい、機能が豊富という意見とともに、運用しづらい、とくに大規模に運用するとたいへんなデータストアという話が散見されます。

本記事ではMongoDBで実現されること、便利に使えることとは何か、という点を説明していければと考えています。

MongoDBの概要

MongoDBとはどんな特徴を持つデータストアなのか、その概要について最初に説明していきましょう。おっと、その前にテスト用のデータベースを作ってみましょう。

テスト環境の作成

MongoDBのインストール

公式サイトのドキュメント[注3]を確認してMongoDBの最新版をインストールしてみましょう。各プラットフォームへのインストール方法は割愛しますが一例としてUbuntuの場合のインストール方法を記載します（**図1**）。

テストデータのインポート

今回は**図2**のようにNASAが公開しているWebサーバのログのTSVファイルをインポー

注1）URL http://www.mongodb.org/
注2）URL http://www.mongodb.com/
注3）URL https://docs.mongodb.com/manual/administration/install-community/

▼図1　Ubuntu環境でのMongoDBのインストール

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927
$ echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.2 multiverse" | sudo ↵
tee /etc/apt/sources.list.d/mongodb-org-3.2.list
$ sudo apt-get update
$ sudo apt-get install -y mongodb-org
```

▼図2　TSVファイルのダウンロード

```
$ wget http://indeedeng.github.io/imhotep/files/nasa_19950630.22-19950728.12.tsv.gz
$ gzip -d nasa_19950630.22-19950728.12.tsv.gz
$ head -2 nasa_19950630.22-19950728.12.tsv
host    logname time    method  url     response        bytes   referer useragent
199.72.81.55    -       804571201       GET     /history/apollo/        200     6245
```

トしてテストデータとしてみましょう。**図3**ではこれをMongoDBへと取り込んでいます。テスト環境ができたら次の節でMongoDBの概要の説明を行います。

ドキュメント指向とは何か?

データストアと言えば、やはり日本で有名なのはMySQLを始めとしたRDBです。RDBは列構造でデータを保存するのですが、MongoDBはJSONを拡張したBSON（Binary JSON）形式でデータを保存します。配列や階層構造などのリッチなデータ構造に対応しています。BSONはJSONでは表現できないバイナリデータを扱うBinData型や、Date型などのデータ型にも対応しており、MongoDBはこのような構造的なデータが扱えるドキュメント指向データベースに分類されています。**リスト1**は、BSON形式のデータの例です。

スキーマレス

もう1つMongoDBのデータ構造には大きな特徴があります、それはスキーマレスです。RDBの場合CREATE TABLE文を用いて列名やデータ型などのスキーマ定義を行うことが必ず必要となります。テーブル構造はそのテーブル内では同じでなければならず、構造を変更する際にもALTER文を使用してすべての行の構造を変更する必要があるわけです。ですが、MongoDBのコレクション（RDBで言うテーブル）内の構造は各オブジェクトで同一である必要はありません。**リスト2**と**リスト3**のデータ構造が同一コレクション内に存在してもまったく問題ありません。

ここで考えなければならないのは、自由度が高いということは同一コレクションの異なるデータ構造のドキュメントのデータの違いをプ

▼リスト1　BSON形式データの例

```
{
  userId : 'akuwano',
  age : 39,
  favoritefood : [ 'hamburger', 'curry' ],
  logintime : ISODate("2016-10-16T13:04:38Z")
}
```

▼リスト2　データ構造の例

```
{
  userId : 'akuwano',
  sex : male,
  favoritefoods : [ 'curry', 'hamburger' ]
}
```

▼リスト3　データ構造の例2

```
{
  userId : 'hiroakis',
  favoritefoods : [ 'curry' ]
  job : {
    name : “ServerSideEngineer”,
    level : 10
  }
}
```

▼図3　MongoDBへのインポート

```
$ mongoimport --db nasa --collection wwwlog --type tsv --file nasa_19950630.22-19950728.12.
tsv --fields host,logname,time,method,url,response,bytes,referer,useragent
2016-10-13T17:55:28.268+0000    connected to: localhost
2016-10-13T17:55:31.267+0000    [##......................] nasa.wwwlog  15.5MB/142MB (10.9%)
  …… (中略) ……
2016-10-13T17:55:56.436+0000    [########################] nasa.wwwlog  142MB/142MB (100.0%)
2016-10-13T17:55:56.436+0000    imported 1891710 documents
```

ログラムで吸収する必要が出てくるということです。スキーマレスは柔軟であるという点でメリットでもありますが、単純に自由であるというふうに見てしまうと開発に苦労してしまう、バグの温床になってしまう可能性があります。ただし、データ構造が頻繁に変わる可能性のあるWebアプリケーション開発では、データ型の違いを吸収するしくみさえ作ってしまえば、スキーマレスであることは開発のスピードを上げるのに非常に大きなメリットともなりえます。

冗長化

最初にMongoDBのクラスタサーバ構成のオーバービューを**図4**に示します。

MongoDBには冗長化の機能が標準的に搭載されています、それがレプリカセットです。レプリカセットとは、複数のデータベース（mongod）プロセスで、データの同期とサーバの冗長化を行う機能です。MySQLなどのRDBでも、レプリケーションというデータ同期の機能がありますが、レプリカセットはそれを発展させデータ同期や負荷分散だけではなく、サーバの冗長性の担保つまり自動フェイルオーバーまでを行う機能です。レプリカセットの主な構成は**図5**を参照してください。

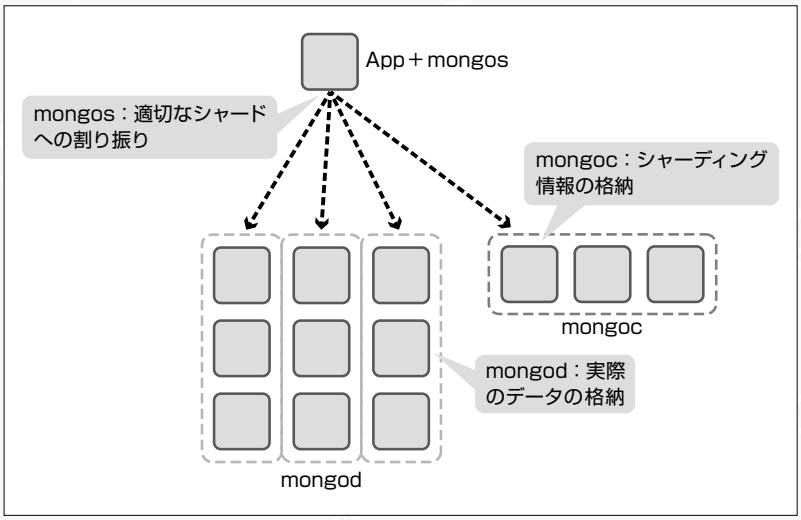
サーバの種別はPrimaryとSecondaryに分かれており、RDBのレプリケーションを行うようにサーバからの更新をPrimaryに、参照をSecondaryにと負荷分散することもできます。

プライマリサーバは各データ操作をOplogと呼ばれるデータベースに格納して、各SecondaryサーバはそのOplogを参照して、自身のサーバへ適用することでデータを同期します。

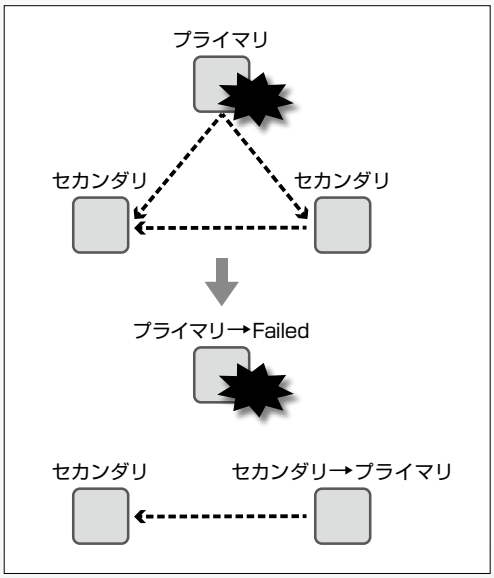
各レプリカセットのメンバは投票によりPrimaryの選出を行います。最低3つの投票権を持ちPrimary選出を行います。具体的には投票権を過半数取得したサーバがPrimaryとなります[注4]。例外としてArbiterプロセスと言われ

注4) レプリカセットはネットワーク分断時にレプリカセットがスプリットブレイン＝「ネットワーク分断などにより1つのクラスタが複数に分かれ、別のクラスタのように見える現象」を起こすのを防ぐためです。

▼図4　MongoDB構成のオーバービュー



▼図5　レプリカセットの概要



る投票権のみを持ち、データを持たない軽量なプロセスを使用することで起動サーバ数を削減することもできます。レプリカセットの具体的な設定方法については公式ドキュメント[注5]などを確認ください

負荷分散／スケールアウト

次に負荷分散について説明します。レプリカセットはデータとしては同一のデータを複数のデータベースで共有するためのものです。この場合、データの更新と参照を分散することでデータのスケールアウトを行うことができるといいました。

しかしレプリカセットは、データが同期されているだけで分散されているわけではないため、同一データベースの更新の負荷分散を行うことができません。シャーディングは複数データベースにおいて、データの受け持ち範囲をシャードに分割することで効率よく負荷分散、そしてスケールアウト、蓄積可能データ量を増やす、などを行うためのしくみとなります。

RDBでもシャーディングを行うことができますが、アプリケーション側での実装が必要だったり、データベース単体で行うことができない、できても手間がかかることが多いですが、MongoDBでは非常に手軽にシャーディングの設定を行えます。主な構成は**図6**を参照してください。

シャード構成を実現する際の登場人物は前述の**図4**にありますが、mongoc、mongos、mongodの3プロセスとなっており、mongocはどのシャードにどのデータを割り当てるかの情報が入っているサーバ、mongosはmongocに入っている情報をもとに各シャードへデータを割り振るためのルーティングを行うサーバ、そしてmongodは実際のデータが保存されているサーバとなります。

また、自動リシャーディングがデフォルトで有効になっており、データの偏りをできるだけなくすように動きます。運用中も常にデータの分散配置が行われ続けます。シャーディングの具体的な設定方法については公式ドキュメントなどを確認してください[注6]。

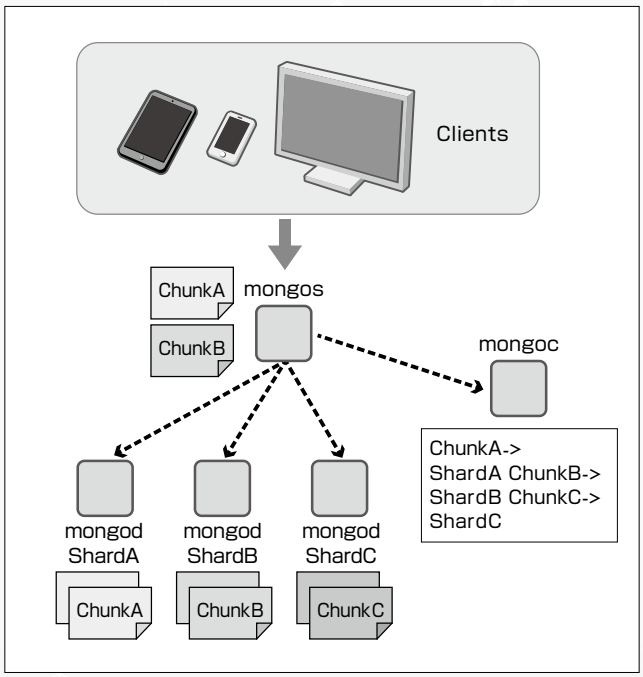
ストレージエンジン

MongoDBバージョン3系以降の特徴的な機能としてデータベースのストレージエンジンがプラガブルになり必要に応じて入れ換えることが可能になりました。今まで使われていたストレージエンジンに明示的な名前が付き、MMAPv1ストレージエンジンと呼ばれるようになりました。WiredTigerという、多重バージョン並行処理制御（MVCC）、ログ構造化マージツリー（LSMTree）

注5）https://docs.mongodb.com/manual/tutorial/deploy-replica-set/

注6）URL https://docs.mongodb.com/manual/tutorial/deploy-shard-cluster/

▼図6　シャーディングの概要



に対応した新しいNoSQL用のストレージエンジンも使えるようになりました。MongoDBバージョン3.2からはWiredTigerがデフォルトのストレージエンジンになっています。

MMAPv1ストレージエンジン

今までMongoDBではMMAPv1ストレージエンジンの特性により、次のような問題が言われてきました（言ってきましたw）これらの問題をどう避けて使うかというのがMongoDB使いにとっての命題だったわけです。

・マルチコアを使うことができない
・キャッシュメモリのインテリジェンスな管理ができない、結果サーバリソースを異常に使う
・データの大量アクセス時にディスクへのアクセスが増えてしまいスローダウンする
・コレクションレベルロッキングのため大量更新に弱い
・容量がドンドン増えてストレージを圧迫する

WiredTigerストレージエンジン

WiredTigerストレージエンジンを使うことによりこれらの問題が解決することになりました。

・マルチコアを適切に使うことができる
・自由な量のメモリをキャッシュとして設定し、インテリジェンスなキャッシュを持つことが可能
・ドキュメントレベルロッキング
・データ圧縮が可能

結果として、従来大量データへの一括アクセスなどには弱いと言われていたMongoDBですが、非常に汎用的な用途で使うことのできるデータストアになったわけです。**表1**に機能をまとめています。

MongoDBのクエリ

RDBはSQLを使ってアクセスしますが、MongoDBは、NoSQLと言うとおりJavaScriptライクなクエリでデータストアにアクセスします。それでは先ほど作ったデータベースにアクセスしてみましょう。

基本的なクエリ

図7のコマンド実行画面で基本的なクエリを示しています。それぞれ、ドキュメント挿入＝save、ドキュメント検索＝find、ドキュメント更新＝update、ドキュメント削除＝removeです。

JavaScriptを使ったクエリ

基本的なクエリの書き方を勉強してきましたが、もちろんこれだけではなくMongoDBで使えるクエリにはさまざまなものがあります。その究極とも言えるのがJavaScriptを使用したクエリです。MongoDBにはSpidermonkeyというJavaScriptエンジン[注7]が組み込まれています。そのためMongoDBではJavaScriptを駆

注7）以前はV8というエンジンでしたが、3.2から置き換わりました。

▼表1　MMAPv1とWiredTigerの機能比較

	MMAPv1	WiredTiger
ロック粒度	コレクションロッキング	ドキュメントロッキング
キャッシュコントロール	OS任せ（MMAPのしくみを利用）	ストレージエンジン側で設定可能
マルチコア	不可	可能
データ圧縮	不可	可能（Snappy、GZIP）
得意なアクセスパターン	ホットデータがあるもの	オールラウンドに使用可能
重視するリソース	メモリ	メモリ（CPUもうまく使えるためあると好ましい）

▼図7　MongoDBの基本クエリ

```
ドキュメント挿入
> db.wwwlog.save({ "host" : "192.168.0.1", "logname" : "-", "time" : 999999999, "method" : ➥
"GET", "url" : "/", "response" : 200, "bytes" : 1, "referer" : "", "useragent" : "" });
WriteResult({ "nInserted" : 1 })

ドキュメント検索
> db.wwwlog.find({ "host" : "192.168.0.1" } );
{ "_id" : ObjectId("57ffd0631f78a02c7cfc3b05"), "host" : "192.168.0.1", "logname" : "-", ➥
"time" : 999999999, "method" : "GET", "url" : "/", "response" : 200, "bytes" : 1, "referer" ➥
: "", "useragent" : "" }

ドキュメント更新
> db.wwwlog.update({ "host" : "192.168.0.1" } , { $set:{ "url":"/test"} })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.wwwlog.find({ "host" : "192.168.0.1" } );
{ "_id" : ObjectId("57ffd0631f78a02c7cfc3b05"), "host" : "192.168.0.1", "logname" : "-", ➥
"time" : 999999999, "method" : "GET", "url" : "/test", "response" : 200, "bytes" : 1, ➥
"referer" : "", "useragent" : "" }

ドキュメント削除
> db.wwwlog.remove({ "host" : "192.168.0.1" } );
WriteResult({ "nRemoved" : 1 })
> db.wwwlog.find({ "host" : "192.168.0.1" } );
```

使したクエリを使用することで、ほかのDBでは別途スクリプトを書かなければならないような処理というのをワンライナー的に表現できます。**図8**で実行しているコマンドは1万件のテストデータを作成できます。

もう1つの例として、**図9**で実行しているコマンドは、nameフィールドのユーザ名を1件ずつforEachで持ってきて表示するものです。

このようにSQLで表すと非常に難解になりそうなクエリもJavaScriptで記述することで簡潔に、そして柔軟に記述できるのはMongoDBならではの特徴です。スパゲティコードならぬスパゲティクエリになるようでは本末転倒ですが……。

Aggregation Framework

ログを入れたり、ユーザのアクティビティログを入れているようなデータベースの場合にデータを解析したい場合があるかと思います。そういった際に使用するのがAggregation Frameworkです。Aggregation Framework集計処理のための

▼図8　テストデータ作成

```
> for (var i = 1; i <= 10000; i++) { db.user.save({ id: i, name: "name"+i }); }
WriteResult({ "nInserted" : 1 })
> db.user.find()
{ "_id" : ObjectId("57ffd70fff9a982c8597578e"), "id" : 1, "name" : "name1" }
 …… (中略) ……
{ "_id" : ObjectId("57ffd712ff9a982c85977e9d"), "id" : 10000, "name" : "name10000" }
```

▼図9　JavaScriptによるクエリ例

```
> db.user.find().forEach( function(myDoc) { print( "user: " + myDoc.name ); } );
user: name1
 …… (中略) ……
user: name10000
```

フレームワークです。パイプラインとして定義された処理をつないでいくことで特別なアプリケーションを作成／使用せずに簡単に複雑な集計を行うことができます。

サンプルで作ったデータベースの解析をしてみましょう。「ステータスコードが200のデータ」の「各HTTPメソッドごとのデータ量の合計」を見たい場合は**図10**のようなクエリになります。

この場合、$match、$group、$sumが使われており、処理の流れとしては次のようになっています。

- $matchでresponse（レスポンスコード）が200のデータを抽出
- $groupで、集計処理を実行、HTTPのメソッドごとに$sumで、該当するbytesの値を合計して表示

ほかにもパイプラインとして提供されている機能はたくさんあり、MongoDBもバージョンアップごとに充実してきています。

それ以外にもMongoDBはHadoopとのコネクタを提供しているので、自分たちで立てたものや、クラウドサービスのHadoopからMongoDBのデータにアクセスすることもできます。もしもっと大規模に解析をしたい場合にはそういった構成を検討するのも良いかと思います。

座標処理

現在Pokémon GOが世間を席巻しています。みなさんも休日公園でポケモンと戯れていたりするのではないでしょうか？　いわゆる位置情報ゲームを始めとして、経路探索アプリなどは今日において座標計算というものは生活と切っても切れない関係になります。実はMongoDBでは座標系の計算を標準の機能として持っています。

pokemongodb.placesコレクションに緯度経度の情報を持ったオブジェクトを入れ、空間情報のインデックスを作成しましょう（**図11**）。

インデックスができました。それでは次は検索を行ってみます。目黒駅から徐々にアルコタワーアネックスへと歩いているという想定で検索を行います。先ほどインサートした座標と約100mまで近づいたら検索から出てくるように$maxDistanceで指定します。$maxDistanceの単位は、緯度経度の度になっているのでわかりづらいですが約100mで指定します。

まずは目黒駅です。

```
> db.places.find({ loc: {$near: [ 35.633848, 139.715733 ], $maxDistance: 0.0008987713795241903 }})
```

▼図10　Aggregation Framework

```
> db.wwwlog.aggregate(
...   { $match : { "response" : 200 } },
...   { $group  : { _id  : "$method", "totalbytes" : { "$sum" : "$bytes" } } }
... );
{ "_id" : "POST", "totalbytes" : 66446 }
{ "_id" : "HEAD", "totalbytes" : 0 }
{ "_id" : "GET", "totalbytes" : NumberLong("38692224996") }
```

▼図11　座標系のデータの使用例

```
> use pokemongodb
> db.places.save({'place':'アルコタワーアネックス', 'loc':[35.631460, 139.714035]});
> db.places.save({'place':'ハチ公前', 'loc':[35.659055, 139.700775]});
> db.places.save({'place':'秋葉原駅', 'loc':[35.698353, 139.773114]});
 # 二次元インデックスの作成
> db.places.ensureIndex({loc : "2d"});
```

何も出てきませんね、まだちょっと遠いようです。次は「とんかつとんき」です。並びますがおいしいとんかつ屋さんです。

```
> db.places.find({ loc: {$near: [ 35.633594, 139.714215 ], $maxDistance: 0.0008987713795241903 }})
```

こちらも出てきませんね、では「目黒雅叙園」を指定してみます。

```
> db.places.find({ loc: {$near: [ 35.631964, 139.713614 ], $maxDistance: 0.0008987713795241903 }})
{ "_id" : ObjectId("57ffe88d21a509a02b44d561"), "place" : "アルコタワーアネックス", "loc" : [ 35.63146, 139.714035 ] }
```

出てきました！ 「アルコタワーアネックス」から100m以内にあるという判定がされたわけです（**図12**）。このようにMongoDBでは座標の指定の機能は非常に便利に使えます。

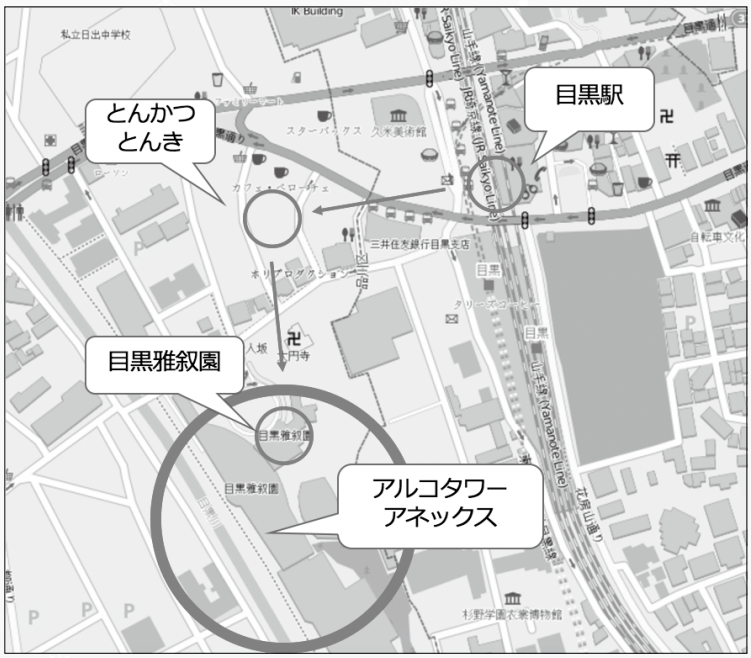
まとめ

ここまでMongoDBのさまざまな機能について説明してきました。ですが、今回紹介したのはMongoDBの機能のほんの一部でしかありません。これ以外にも「全文検索エンジン」「MapReduce」「分散ファイルシステム」などさまざまな機能が存在しています。データストアだけでこれだけいろいろな機能があるデータストアというのはなかなかないのではないでしょうか、そしてそれ以外にもWiredTigerストレージエンジンを搭載したことにより従来の弱点とも言える部分を克服しMongoDBは非常にさまざまなユースケースで使えるようになりました。ひと昔前のように運用で死ぬデータストア、ユースケースを限定しなければいけないデータストアというところから一皮むけてきたところではないでしょうか。ですが、じゃあ無条件にMongoDBを使えば良いのかというとそうではなく、データストアによって運用が楽になる、開発が楽になるというよりは、そのデータストアの理解をどんどん深めることでデータストアの運用が楽になるというのが基本的な原則です。これはMongoDBに限った話ではなく、RDBを含めたすべてのデータストアで共通の話となると考えています。

それを突き詰めることによって、これだけの恩恵を受けることができるデータストアであるMongoDBを、毛嫌いせずに1回向き合ってみるのも良いのではないかな、と筆者は考えつつ本章を終わりたいと思います。

SD

▼図12　歩いていった道程



NoSQLのダークホース

Couchbase Serverを試してみよう!

Author 仲川 樽八（なかがわ たるはち）（株）ゆめみ　Twitter @tarupachi

Couchbase Server（以降、Couchbase）というKVS（Key-Value Store）をご存じですか。日本では知名度が高くありませんが、ハイパフォーマンスと高可用性のため海外ではさまざまな企業で使われています。CouchbaseはEnterprise向け製品でありながら、非商用利用ならば無償で使用できます。さらにCommunityEditionならば無償での商用利用も可能なので比較的容易に導入検討ができます。本章ではインストールとその使い方を紹介します。

Couchbaseとは

概要

Couchbaseはいわゆる KVSの1つですが、Value部分をJSONで保存するドキュメント指向データベースです。その特徴を次に挙げます。

- 高スループット、低レイテンシでの応答
- 複数のCouchbaseサーバのノードを1つのクラスタとして管理
- ドキュメントの永続化、および生存期間（expiry）を設定した自動削除の両方が可能
- 冗長度を指定したデータの冗長化
- ドキュメントに対する強い一貫性（結果整合性ではない）
- KeyやValueに関する横断的操作（後述するViewやN1QLを利用）
- CASやロック機構を利用したドキュメントのアトミック操作
- 遠隔地のクラスタとのデータの双方向レプリケーション（XDCR：Cross Data Center Replication機能）

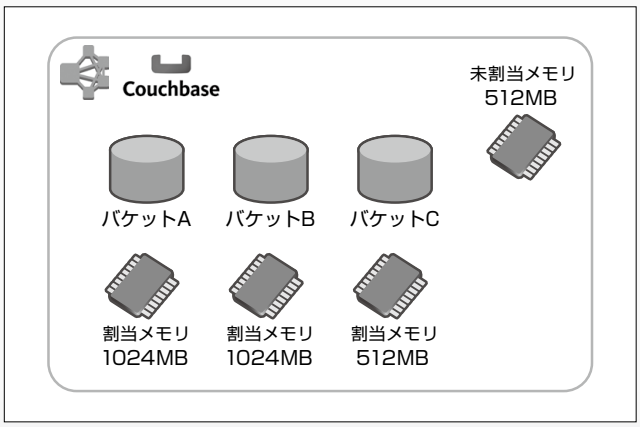
Couchbaseのクラスタ

Couchbaseは1台からでも導入ができますが、複数台のノードを利用してクラスタを組み、クラスタ全体を1つのCouchbaseリソースとして利用できます。

Couchbaseのバケット

Couchbaseクラスタ内には複数のバケットを作成できます（**図1**）。このバケットはRDBにおける論理DBに相当します（後述するN1QL利用時ではテーブルに近い概念）。

▼図1　バケットの概念（Couchbaseクラスタ内には複数のバケットを作成できる）



・バケットごとにあらかじめ利用メモリを割り当てるが、このメモリは管理画面から動的に変更可能
・バケットごとに冗長度の設定や接続パスワードの設定が可能
・この機能により、1つのクラスタを複数のアプリケーションで安全に共有できる

Couchbaseのドキュメント

Couchbaseはバケット内に複数のドキュメントを保存できます。ドキュメントはKeyとValueの組み合わせにより構成されますが、Value部分をJSONデータとして保持します。

ドキュメント間のトランザクションなどはサポートしませんが、単一のドキュメントに対するデータ操作では、ドキュメントのget時にメタ情報として同時に通知されるCAS（Compare And Swap）を次回保存時に指定することでアトミックな操作を実現します。冗長度を上げた場合、ドキュメントは複数のサーバにコピーされますが、フェイルオーバー機能（障害発生時などに特定のノードを切り離す機能）が行われるまで、同一のドキュメントに対する更新操作は常に同一のノードに対して行われます。そのためCouchbaseでは、ドキュメントは強い一貫性を保有しています。

4つのサービスとは

Couchbaseは次の4つのサービスで構成されます。サービスごとにリソースの使用状況に偏りが生じたり、分散困難な内容などが含まれるため、サービスを起動するノードについて個別で設定できます。

・Dataサービス
いわゆるKVSの本体。ドキュメントそのものの格納および利用一式を担う。後述するView機能の利用もこのDataサービスに含まれる
・Indexサービス
後述するN1QLのインデックスの作成、保存、利用を行うサービス
・Queryサービス
後述するN1QLのクエリを実行するサービス
・FullTextサービス
4.5.0ではβ機能として提供されているドキュメントの検索サービス（本稿では触れない）

Couchbaseを選ぶ理由

Couchbaseは一般的なRDBのようなトランザクション機構がありませんが、KVSとして4つの特徴があります。

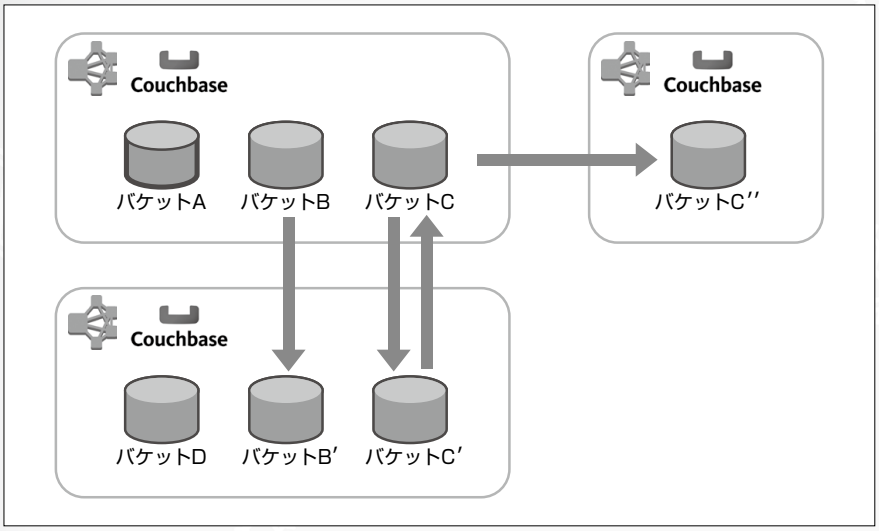
第1には高性能とスケーラビリティの実現です。具体的には次のようになります。

・ノード1台あたり100,000 read/write級のスループットが可能
・クラスタへのノードの追加によりread/writeのスケールアウトが可能
・1〜2ミリ秒単位での非常に低いレイテンシでの応答が可能
・クラスタ全体を1つのリソースと見なせるため、クラスタへのノードの追加により逼迫（ひっぱく）したリソースのスケールアウトが可能

第2は高可用性です。次のようになります。

・データが冗長化され、かつドキュメントに強い一貫性を持つため、クラスタ内の単体のノードが破損したケースでもデータの欠損なしに速やかに縮退運転に移行できる
・クラスタへのノードの追加、削除が無停止で可能
・ノードの入れ替えが無停止で可能なので、ソフトの更新も無停止でできる
・XDCR機能（Cross Data Center Replication）を利用することで広域なインフラ障害に対しても対応可能（図2）

▼図2　XDCR機能（バケット単位で他のクラスタ上のバケットへのレプリケーションができる）



第3としては横断的ドキュメント操作を提供するメソッドです。

・Viewを利用してドキュメント内の値を利用した検索ができる
・Elastic Searchにリアルタイムでデータ連携ができる（XDCR機能を利用）
・N1QL[注1]のようなSQLでのデータ検索ができる
・Full Text Serachが可能に

第4は、Webベースの管理コンソールです。各ノードの詳細なモニタリングや管理ができます。そしてCouchbaseのシステムに対するコマンドはREST APIとして提供されており、運用の自動化もできます。それらにより高いメンテナンス性を実現しています。

EnterpriseEditionとCommunityEdition

CouchbaseにはEnterpriseEditionとCommunityEditionの2つがあります。利用方法を**表1**にまとめました。

CommunityEditionの機能制限について

CommunityEditionの機能制限についてはWebサイト[注2]の情報が詳しいのですが、おも

注1）Couchbase4.x以降で、SQLによるデータ問い合わせ・更新機能（N1QL：ニッケルと読む）が、正式にサポートされました。N1QL上では複数のドキュメントのJOINなどもサポートされ、RDBのようなデータ操作ができますし、View機能では不可能なドキュメントの更新操作ができる強力な機能です。

注2）URL http://developer.couchbase.com/documentation/server/4.5/introduction/editions.htm

CouchbaseのViewとは

CouchbaseではドキュメントのKeyを利用したIndex作成ルールをJavaScriptで記載し、IndexおよびIndexに対応するデータの作成ができます。このIndexを利用した検索により、ドキュメントの抽出やドキュメントの集計結果を取得できます。簡単な例では、ドキュメントに更新時間を記載しておき、この更新時間を元にViewのIndexを作成します。これで更新時間順にドキュメントを整列して特定の範囲のものだけを取得するといった操作ができます。ViewのIndexの更新は非同期ですが、検索時に必ずIndexを更新して最新の状態にしたうえで検索を行うオプションなどもあります。

▼表1　各エディションによるライセンスなどの違い(※英語のみ)

	EnterpriseEdition		CommunityEdition
	年間ライセンス契約あり	無償利用	無償利用
ライセンス費用	利用ノード台数に比例	無償	無償
商用利用	可	2ノードまで可	可
非商用利用	可	可	可
サポート	24時間365日(※)	なし	なし
Hotfix	あり	なし	なし
マイナーバージョンアップ	あり	あり	多くのマイナーバージョンをスキップ
各言語用のSDK	最新版利用可能	最新版利用可能	最新版利用可能
機能制限	フル機能利用可能	フル機能利用可能	一部機能制限あり

な差異を次に挙げます。

・XDCR時のフィルタリング
・ラックゾーンアウェアネス機能
・差分データバックアップ機能
・セキュリティの機能
・N1QLの一部機能制限

この中でとくに重要なのはラックゾーンアウェアネス機能です。これがシステム構築時に必須かどうか、それが選定する際の判断基準になることが多いでしょう。

ラックゾーンアウェアネス機能とは、クラスタ内のノードをあらかじめ複数のグループに別れるようにタグ付けしておくことができる機能です。Couchbaseはデータを冗長化するとき、各ノードに書き込まれたデータを必ず他のグループに含まれるノードにコピーします。このグループ分けとして、サーバラックや、データセンター、Availability Zoneなどを利用することにより、それぞれの単位での障害が発生してもデータ欠損なくサービスを継続できます。

Couchbaseのインストール

AWS上にCouchbaseを構築する場合、Market placeにあるイメージから作成することもできます。しかし、本稿では公式サイトからパッケージをダウンロードしてインストールする方法を説明します。

インスタンスの推奨スペックはかなり高いのですが、開発環境では小規模でも問題ありません。ここではt2.small(2016年9月16日現在0.04$／時～100円／日)のインスタンスを利用することとします。管理コンソールを利用するために、今回は起動時にグローバルIPアドレスを利用可能な状態としておきます。

日本語版のページは古いので、本家サイト[注3]から最新版をダウンロードします(ここではCentOS6 64bit版)。ダウンロード先のURLがわかればwgetを使用してもかまいません。

```
$ wget http://packages.couchbase.com/
releases/4.5.0/couchbase-server-
enterprise-4.5.0-centos6.x86_64.rpm
$ sudo rpm -Uvh couchbase-server-
enterprise-4.5.0-centos6.x86_64.rpm
```

コマンドとしてはこれだけです。インストールはほぼ数秒で終わりますが、この時点でCouchbaseがサービスとして起動します。このとき、Couchbaseが利用するポートも表示されますので、利用するサーバ間ではそのポートを開けてください(今回はlocalhostで試験します)。

クラスタの初期化

最初に管理コンソールでクラスタの初期化が必要です。すでに初期化されたクラスタにノードを追加する場合もこちらから行うことができます。

注3) URL http://www.couchbase.com/nosql-databases/downloads

Couchbaseをインストールしたサーバの8091ポートで管理コンソールが起動しているのでブラウザでアクセスし、[Setup]を押下して次へ進んでください。初回はまだUser/Passwordを設定していないでそのまま進めます（**図3**）。

サービスはデフォルトでDataサービス、Indexサービス、Queryサービスを利用できます。

また、このノードではデフォルト状態だとRAMが足りなくなるので、DataRAM Quotaを1024MBに、Index RAM Quotaを256MBに変更します。

Couchbaseでは初回にサンプルバケットを作成します。サンプルバケットに投入するデータを選べますが、このバケットは最初に削除する予定のため、そのまま進めます。

メールアドレスなどを聞かれますが入力しなくてもかまいません。途中でパスワードを聞かれるので適宜設定します。このパスワードは次回以降に管理コンソールを使うときや、クラスタに他のノードを追加するとき、XDCR機能を利用するときなどに使用するので忘れてはいけません。

管理コンソールの利用開始

これまでの作業により、管理コンソールが使用可能になります。モニタリング可能な項目は膨大ですので説明は割愛します。各種のリソースがリアルタイムで更新されていく様子が圧巻なので、ぜひ一度試してください（**図4**）。

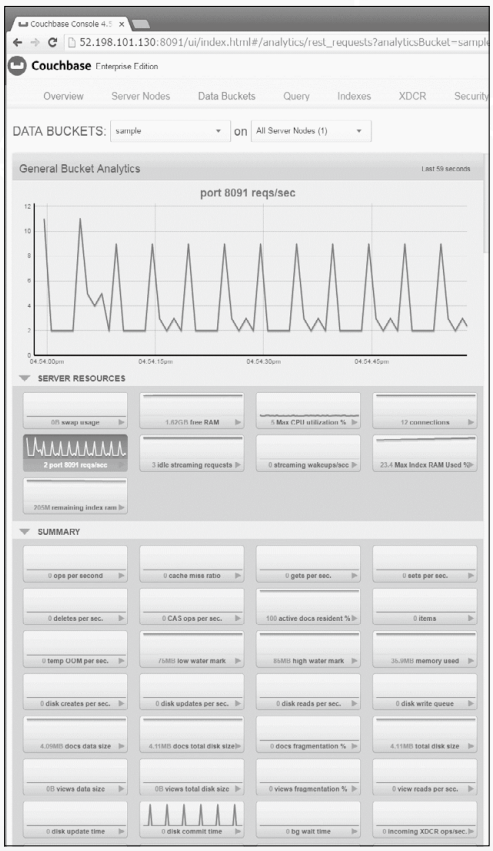
デフォルトバケットの削除

初期化時に作成されたデフォルトバケット[default]は特殊なバケットです。そのまま利用するのが難しいので削除します。

① [Data Buckets]タブを押下
② バケット名の横の[▶]を押下してバケットの詳細情報を展開
③ [Delete]を押下

▼図3　Webブラウザで「http://[Global IP]:8091/」で管理コンソールにアクセス（入力数値は例）

▼図4　管理コンソールのモニタリング画面（抜粋）

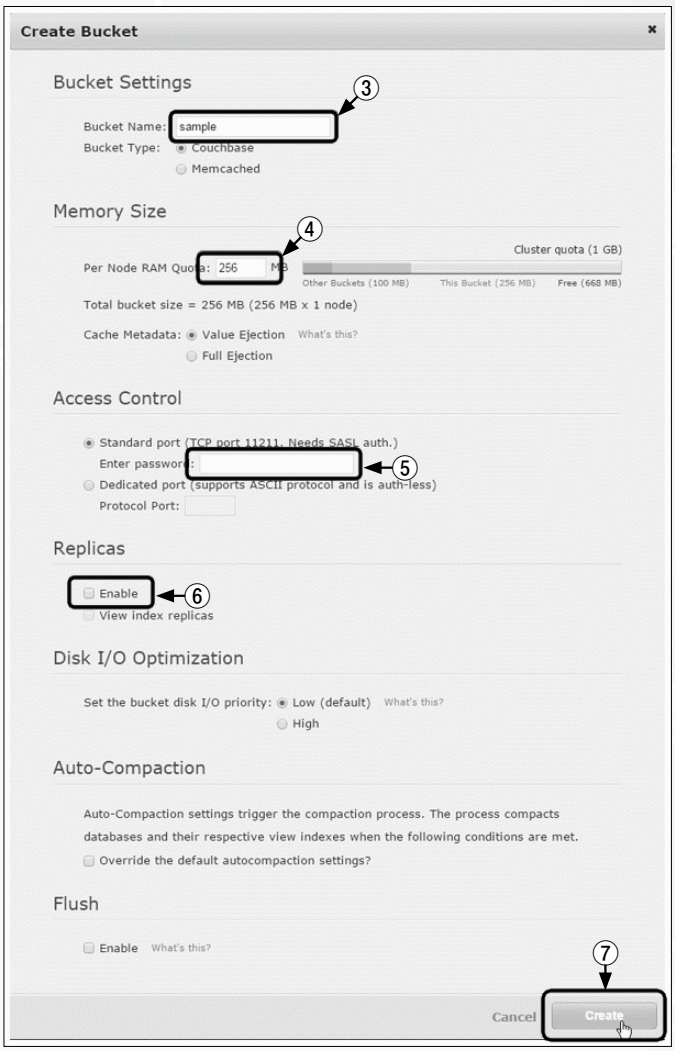


新規バケットの追加

次の手順でバケットを新規追加します（図5）。

① [Data Buckets] タブを押下
② [Create New Data Bucket] を押下
③ [Bucket Name] に sample と入力
④ [Per Node Ram Quota] に 256と入力
⑤ [Access Control] → [Enter password] にパスワードを入力（先ほど設定したものとは異なる新しいパスワードを設定）
⑥ [Replicas] → [Enable] のチェックを外す（ノードが1つのため）
⑦ [Create] ボタンを押下

▼図5　バケットの新規追加



KVSとしての利用

PHPスクリプトからの利用

PHPからCouchbaseを利用するための例を図6に示します。準備[注4]が必要です。php.iniにcouchbse.soを組込み、Apacheを再起動します。

```
sudo echo "extension=couchbase.so" > /etc/php.d/couchbase.ini
sudo /etc/init.d/httpd restart
```

注4）URL http://developer.couchbase.com/documentation/server/current/sdk/php/start-using-sdk.html

リスト1は、先ほど作成したバケットにドキュメントを格納するサンプルです。これはCouchbaseBucket::upsert()命令を利用して、元のドキュメントがなかった場合には新規追加、あった場合には上書きを実行しています。PHP上のarray()データ構造をそのままJSONとして

▼図6　PHPのCouchbaseへの導入

```
# Only needed during first-time setup:
wget http://packages.couchbase.com/releases/couchbase-release/couchbase-release-1.0-2-x86_64.rpm
sudo rpm -iv couchbase-release-1.0-2-x86_64.rpm
# Will install or upgrade existing packages
sudo yum install libcouchbase-devel gcc gcc-c++ php-devel
sudo pecl install couchbase
```

シリアライズして格納しているため、Couchbase上ではJSONであることを意識しないものとなっています。**リスト1**の実行結果は**図7**のようになります。

管理コンソールでもDocumentに追加されていることが確認されました（**図8**）。ここではexpiryとして30を指定していますので、30秒後にはこのドキュメントは削除されることも確認できます。

CouchbaseBucketに対する主要なMethodは**表2**のとおりです。例示した**リスト1**を改修しながらいろいろな挙動を試していただきたいと思います。

▼リスト1　cb_test.php

```
<?php
define("BUCKET_NAME",  "sample");

define("BUCKET_PASSWD",    "password");

    // Connect to Couchbase
    $cluster = new CouchbaseCluster("localhost");
$bucket = $cluster->openBucket(BUCKET_NAME, BUCKET_PASSWD);


// Store a document
echo "Upsert U:Taru8¥n";
$result = $bucket->upsert(
    'U:Taru8',
    [
        "doctype"     => "USER",
            "email"       => "taruhachi@***.**",
            "sex"         => "male",
            "birthday"    => "1975-08-08",
            "u_timestamp" => time()
        ],
    ['expiry'=>30]
    );
    print_r($result);

    echo "Get U:Taru8¥n";
    $result = $bucket->get("U:Taru8");
    print_r((array)$result->value);
?>
```

横断検索の利用

N1QLを利用した検索

Couchbaseはまさに NoSQL（Not Only SQL）で、KVS機能だけでなくSQLも利用できます。当初はcbqというコマンドラインツールを利用していましたが、最新版のCouchbase4.5では管理コンソールからクエリの実行ができますので気軽に試行できます[注5]。

N1QLにはかなりの癖がありますが、Couchbaseではチュートリアルサイト[注6]を準備しています。こちらのサイトから、実際にN1QLのクエリを発行しながらN1QLで何ができるのか体験してください。

N1QLを利用する際の注意

一方でN1QLはまだ発展途上の機能である部分もあるため、とくにデータ量が多い環境で

▼図7　スクリプトの実行結果[注7]

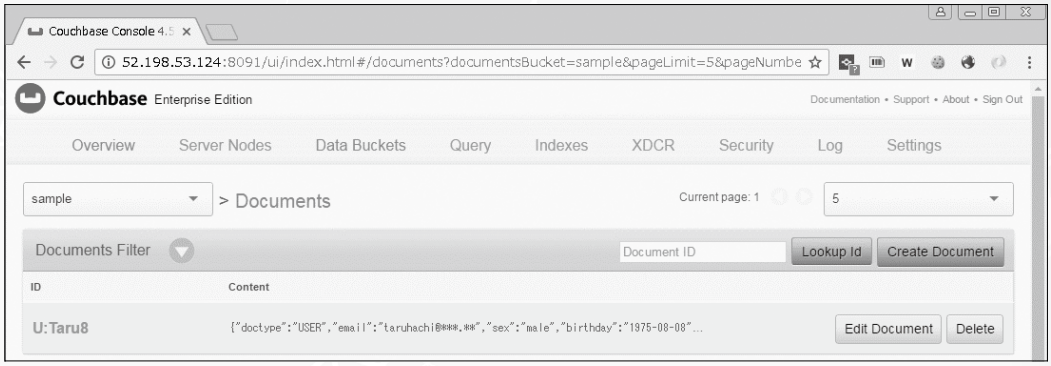
```
$ php cb_test.php

Upsert U:Taru8
CouchbaseMetaDoc Object
(
    [error] =>
    [value] =>
    [flags] =>
    [cas] => 2445gkpq8k
    [token] =>
)
Get U:Taru8
Array
(
    [doctype] => USER
    [email] => taruhachi@***.**
    [sex] => male
    [birthday] => 1975-08-08
    [u_timestamp] => 1474152241
)
```

注5）この機能を利用するためにはCouchbaseインストール後の最初に管理コンソールを立ち上げたときに、IndexサービスとQueryサービスを有効にする必要があります。

注6）URL http://query.pub.couchbase.com/tutorial/

注7）ドキュメントのキーおよびドキュメント双方にドキュメントのデータ区分がわかるようにしておくと管理コンソールの利用時、N1QL利用時などに便利。ここではドキュメントのキーのプレフィックスを"U:"とし、doctype="USER"をドキュメントに含めている。

▼図8　管理コンソールでの確認



▼表2　各種メソッドの説明(http://docs.couchbase.com/sdk-api/couchbase-php-client-2.2.2/)

Method	操作内容	特記事項
insert()	ドキュメントの新規挿入	元のドキュメントがすでに存在するときは例外が発生
upsert()	ドキュメントの挿入	元のドキュメントがすでに存在するときは上書き
replace()	ドキュメントの更新	元のドキュメントがないときは例外が発生
append()	ドキュメントの最後に追記	
prepend()	ドキュメントの先頭に挿入	
remove()	ドキュメント削除	元のドキュメントがないときは例外が発生
get()	ドキュメント取得	元のドキュメントがないときは例外が発生
getAndTouch()	ドキュメント取得と同時にドキュメント生存期間を更新	session保存期間の更新などに利用
getAndLook()	ドキュメント取得と同時にロック	
touch()	ドキュメント生存期間を更新	
counter()	アトミックカウンターを利用	
query()	View検索クエリまたはN1QLクエリの発行	それぞれ事前にViewやN1QLのIndexを作成しておく必要がある

の利用には十分な検証が必要です。

Couchbaseでの N1QLの使用にあたり、注意点を次にまとめます。

・Indexが作成されていない検索は非常に遅く、タイムアウトすることが多い
・Indexを追加すると追加されたIndexの数に応じて物理メモリを大量に要求する
・Indexの更新が非同期であり、取得した結果が最新のドキュメント内容とは一致しないことがある
・Index更新タイミングの制御が難しい
・(当然ながら)トランザクション操作を利用できない

最後に

足早にCouchbaseの紹介をしましたが、導入に関しては非常に簡単で比較的気軽に試すことができる製品です。今回はPHPのサンプルを紹介いたしましたが、Couchbaseには各種プログラミング言語用のSDKがすでにそろっていますので、まずは動作を試したうえで製品としての強力さを体験してください。SD

第4章

データの型や永続化機能が用途を広げる

高速なインメモリデータベースRedis

Author 大谷 祐司（おおたに ゆうじ） ㈱インテリジェンス

この章では、KVS（Key-Value Store）として人気の高いRedisについて紹介します。インメモリデータベースの高速性に加えて多彩なデータ型、データ保護や冗長化のしくみを備えているRedisは、Webアプリケーション開発現場で必須の技術となりつつあります。

Redis概要

Redisは、C言語で実装されているインメモリ（メモリ上にデータを保存する）データベースです。2009年に初回リリースされ、執筆時点（2016年10月）での最新安定版は3.2です。修正BSDライセンスのオープンソースソフトウェアで、GitHub上でソースコードを見ることができます[注1]。

開発状況

Redisは、イタリア人のエンジニアであるSalvatore Sanfilippo氏によって開発されました。現在も開発は活発で、継続的なバージョンアップが行われています。最近のものを挙げると、

●バージョン3.0（2015年4月）

- Redis Clusterの追加（複数のサーバでクラスタリングできるしくみ）
- WAITコマンドの追加（レプリケーション構成時にSLAVEへの書き込みが終わるまで、MASTERが待つ）
- LRUアルゴリズムの改善

●バージョン3.2（2016年5月）

- 文字列型がビットアレイとして利用可能になる
- GEO APIの追加（キーに対する緯度、経度、地名を扱える）
- Luaスクリプティングにデバッグ機能の追加
- Redis Clusterの機能改善

このように、バージョンアップごとにさまざまな機能が追加されています。

利用できる環境

推奨環境

RedisはPOSIX[注2]システム上で動作します。Redisの開発はLinux、Mac OS Xで行われており、本番環境ではLinuxで動かすことが推奨されています。

Windowsは公式にはサポートしていませんが、MS Open Techによって64bit Windows OS用にバイナリが配布されています[注3]。

AWSのElastiCache for Redis

2013年9月からAWS（Amazon Web Services）のキャッシュサーバ（ElastiCache）にRedisが追加されました。それまでの選択肢はmemcachedのみでしたが、現在はRedisとmemcachedの2つから選択できます。

注1）URL https://github.com/antirez/redis

注2）Portable Operating System Interfaceの略で、Unix系OSにおいて移植性の高いソフトウェアの開発を容易にするための規格。

注3）URL https://github.com/MSOpenTech/redis/releases

ElastiCache for RedisではデータスナップショットのS3へのエクスポート、データを保持したままのスケールアップ、Multi-AZで自動フェイルオーバーなど、運用時に役立つ機能を利用できます。

2016年10月14日にアップデートがあり、最新バージョンのRedis 3.2が利用できるようになりました。詳細についてはAmazon Web Servicesブログ[注4]で紹介されていますので、ぜひチェックしてみてください。

Redisの特徴

オープンソースのデータベースには多くの種類が存在しますが、Redisの特徴として次の点が挙げられます。

KVS（Key-Value Store）

データを保存する際、Keyに対応した値（Value）を格納します。データの格納や取り出し、削除などはすべてキーを指定して行います。RDBMSと異なり、スキーマ（テーブル定義）が存在しません。一般的なKVSは単純なキーと値をセットで保持しますが、Redisの値には後述する“データ型”が存在します。これによって単純なKVSではできなかったさまざまな操作が可能です。

高いパフォーマンス

Redisはインメモリで動作するため、ディスクにデータを保存するDBに比べると非常に高いパフォーマンスを発揮します。また**図1**のようにシングルプロセス／シングルスレッドで動作するために、プロセスやスレッド立ち上げにかかる時間を必要としません。ただしマルチコアのCPU環境で起動しても、使用するのは1つのコアだけになります。

それでは、マルチコアCPUの環境でパフォーマンスを高めたいときにはどうしたら良いでしょうか。Redisは単一サーバ上で、ポートを変えて複数の起動が可能です。複数のRedisを立ち上げて利用することで、複数コアを活用してパフォーマンスを上げることができます。ただし利用できるメモリも各Redisに割り当てなければいけないので注意が必要です。この方法は公式サイトでも紹介されています[注5]。

複雑な処理をまとめて行える機能

Lua言語で書かれたスクリプトを実行できる[注6]、トランザクションを利用して複数の命令を1コマンドで実行できる[注7]など、複雑な処理をまとめて行える機能を備えています。

データの保護

Redisには、データを保護するためにいくつかのしくみが備わっています。一般的なインメモリデータベースはすべてのデータをメモリ上に保管するため、サーバプロセスが終了すると

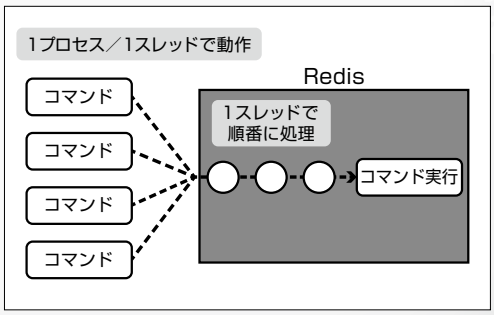
注4）URL https://aws.amazon.com/jp/blogs/news/amazon-elasticache-for-redis-update-sharded-clusters-engine-improvements-and-more/

注5）URL http://redis.io/topics/faq
*Redis is single threaded. How can I exploit multiple CPU / cores?

注6）Redis上でLuaというプログラム言語を実行することができます。これにより、複雑な操作を一括で実行することが可能です。EVALコマンドの引数としてLuaスクリプトを渡す方法、LuaスクリプトをRedisに登録する方法があります。工夫次第でさまざまな活用が可能ですので、ぜひ試してみてください。

注7）Redisでは複数の命令をまとめて実行することで、トランザクションを利用することができます。MULTIコマンドでトランザクションを開始し、その後発行されたコマンドのキューイングを行います。キューイングされたコマンドは、EXECコマンドがコールされた時点で実行されます。代わりにDISCARDをコールすると、トランザクションキューの内容がフラッシュされて、トランザクションは終了します。

▼図1　Redisのコマンド処理



データが消失してしまいます。しかしRedisはバックアップからデータを復旧することが可能です。

多彩なデータ型

一般的なKVSは単純なキーと値をセットで保持しますが、Redisは値をデータ型として保持することができます。文字列型（Strings）、リスト型（Lists）、セット型（Sets）、ハッシュ型（Hashes）、ソート済みセット型（Sorted Sets）という5つのデータ型を利用できます。

複数のデータ操作を一括して行える、データの集合から特定のデータを取り出すことができるなど、データ型を利用することで活用の幅が広がります。

多彩なコマンド

Redisには約200ものコマンドが存在しています。Redis本体の動作に関するコマンドや、データ型特有のコマンドなどさまざまなものがあります。スペースの都合から詳細な説明は割愛しますが、Redisの公式サイトでコマンドを確認できます[注8]。利用する際はぜひ目を通してみてください。

CLIを利用できる

Redisをインストールすると、CLI（コマンドラインツール）を利用できるようになります。動作状況の確認やデータ操作などができる便利なツールです。

柔軟なスケールアウト

RedisにはMaster/Slave形式のレプリケーションや複数台のクラスタなど、スケールアウトできるしくみが備わっています。Redis単体でも高速に動作しますが、複数台で負荷分散することでよりパフォーマンスの高いシステムを構築することができます（冗長化の節で詳しく説明します）。

Redisのデータ型について

「Redisの特徴」で取り上げたデータ型について、もう少し詳細に説明します。

文字列型（Strings）

シンプルなKVSとして、キーに対して値を格納し、キーを指定して値を得る、最も基本的なデータ型です（**図2**）。最大で1GBまで扱えます。画像ファイルなどのバイナリデータを扱うこともできますが、利用するクライアントライブラリによってはバイナリデータに対応していないものもあるので注意が必要です。

リスト型（Lists）

プログラム言語のリスト型と同じように、追加した順番に値が格納されるデータ型です（**図3**）。先頭や末尾に値を追加する、範囲を指定して値を取り出すなど、リスト操作のためのコマンドが存在します。また、LINSERTコマンドを使用すると、格納されている値を指定して前後に値を追加することが可能です。

セット型（Sets）

集合として値を保持するデータ型です（**図4**）。保持している値は順番を持たず、同じ値は重複

PUB/SUB

Redisでは標準でPUB/SUBを使用したメッセージングが可能です。PUB/SUBはパブリッシュ・サブスクライブの略で、メッセージの送信者（出版側）が特定の受信者（購読側）を想定せずにメッセージを送信できるモデルです。出版側と購読側の結合度が低いため、スケーラビリティがよく、動的なネットワーク構成に対応可能なのが特徴です。

注8）URL http://redis.io/commands

して保持できません。集合への値の追加や削除のほかに、複数の集合を使った演算用のコマンドが用意されています。

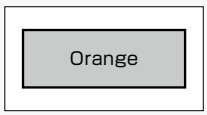
ハッシュ型（Hashes）

プログラム言語のハッシュテーブルと同じように、キーと値のペアを複数保持できるデータ型です（**図5**）。保持している値は順番を持たず、キーを指定することで値の操作が可能です。セット型は値を指定して操作を行いますが、ハッシュ型は値を指定して操作を行うことができません。

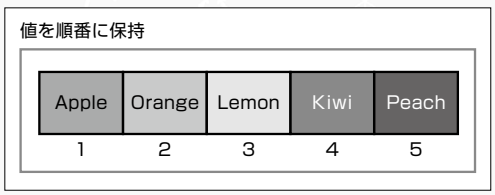
ソート済みセット型（Sorted Sets）

セット型と同じく値の集合を扱うデータ型ですが、それぞれの値に設定された「スコア」によって順序付けされているのが特徴です（**図6**）。順位を指定してデータを取得したり、スコアの値の範囲を指定してデータを取得することができます。注意点としては、複数の値が同一のスコアを持っている場合でも同一順位にはならず、辞書の順番に順位が高いとみなされます。

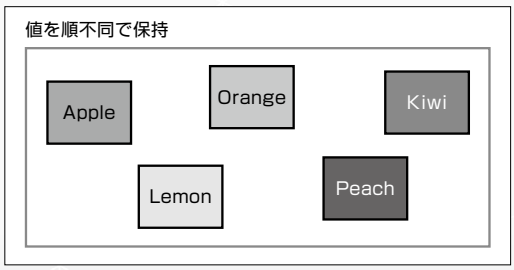
▼図2　文字列型



▼図3　リスト型



▼図4　セット型



活用方法

この節では、Redisの活用方法を紹介します。筆者はこれまで、次の用途でRedisを利用してきました。

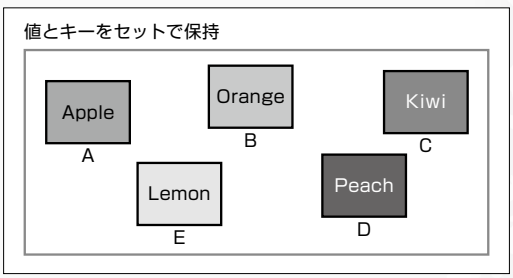
Webアプリケーションのセッションストア

ここ最近、Webアプリケーションのセッションストアに Redisを利用するケースをよく見るようになりました。セッションストアに求められる要件として次のようなものがありますが、これらの要件をRedisは満たすことができます。

- 一定の時間でデータを消失できる
- Webリクエストごとにデータアクセスするので、素早く動作する
- 冗長化された複数台のWebサーバで同じデータを共用できる

従来はセッションをWebサーバのファイルやDBに保存することが多かったのですが、最近はWebフレームワークでRedisの利用が増えてきました。Ruby on Railsや、PHPで人気のLaravel、Java/Scalaで人気のPlay Frameworkなど、多くのフレームワークでRedisに対応す

▼図5　ハッシュ型



▼図6　ソート済みセット型

るしくみが存在します。ぜひチェックしてみてください。

DBのキャッシュ

Webアプリケーションのレスポンスタイムを早くしたいとき、DBへのアクセスがボトルネックになるケースが多いのではないでしょうか。そんなときDBのデータをRedisにキャッシュすることで、パフォーマンスを改善することができます。筆者の場合にはマスタデータをあらかじめDBからRedisにコピーしておき、キーを指定して高速に取得できるようにしていました。

また、DBに発行した複雑なSQLの実行結果をRedisにキャッシュすることで、2回目以降のデータアクセスを高速に行うことができます。このようにDBとRedisを共存させることで、高速なWebアプリケーションを構築できます。

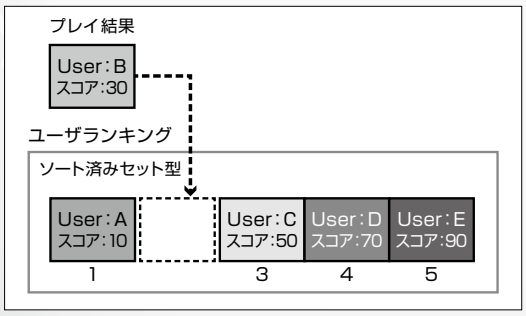
リアルタイムランキングでの活用

Redisの人気が高まったのは、ゲームの「リアルタイムランキング」で活用されたことが大きいように思います。筆者も以前ソーシャルゲームのランキングでRedisを活用していましたが、ユーザ数の多い環境においてもパフォーマンスの良いしくみが実現できました。簡単ですが以下に手順を記します（**図7**）。

ランキングの生成

ランキング用のソート済みセット型を作成し、ゲームプレイ結果の「ユーザID」「スコア」を登録します。この操作で、スコア順のユーザランキングが生成されます。

▼図7　ソート済みセット型を使ったリアルタイムランキング



ランキングデータの取得

ランキング用のソート済みセット型からは、次のようなデータを取得して利用できます。

- 登録されているユーザの数
- 指定したユーザのランクとスコア
- 順位の範囲を指定したランキング情報（例：1～100位の範囲）
- スコアを指定したランキング情報（例：100～500点）

スコアの増加／減少

ZINCRBYコマンドを利用することで、指定したユーザのスコアを増加／減少させることができます。スコアを増減させると、リアルタイムでランキングに反映されます。

◆ ◆ ◆

Redisには複数のデータ型やデータ永続化のしくみがあるので、データを一時的に保存する「キャッシュサーバ」を超えたさまざまな活用が可能です。読者の方はRedisをWebアプリケーションで利用することが多いと思います。DBとRedisを併用することで柔軟性の高いシステム構築ができます。

使ってみよう

この節ではRedisを利用する手順を説明します。

ソースからのインストール

公式サイト[注9]に記載されている方法で、MacやLinuxなどのPOSIX互換システムに最新安定版のRedisをインストールすることができます。

OSのパッケージ管理ツールにRedisが用意されていることも多いですが、Redisは頻繁にバージョンアップが行われています。バグ修正や機

注9）URL http://redis.io/download

能追加に追従するためにも、ソースから最新バージョンをインストールすることをお勧めします。

まずは公式サイトからRedisをダウンロードします。ダウンロードできたら、アーカイブを展開してディレクトリに移動します。そこでmakeコマンドを実行してRedisをビルドし、完了したらインストールを行います。一連のコマンドは次のようになります[注10]。

```
$ wget http://download.redis.io/releases/redis-3.2.4.tar.gz
$ tar xzf redis-3.2.4.tar.gz
$ cd redis-3.2.4
$ make
($ make test)
$ sudo make install
```

起動してみよう

Redisのインストールが完了したら、次のコマンドを実行することでRedisを起動できます。

```
$ redis-server
```

次のようなメッセージが表示されたら、起動完了です。

```
The server is now ready to accept connections on port 6379
```

Redisはデフォルトだと、ポート6379でクライアントからの接続を待ちます。デフォルトでは常駐プロセス（daemon）として動作するようになっていないため、立ち上げるとシェルが操作できなくなります[注11]。

常駐プロセスとしてバックグラウンドで起動するには、Redisを解凍したディレクトリに存在する設定ファイル（redis.conf）に次の修正を行ってください。

```
daemonize no
  ↓
daemonize yes
```

あとは起動時の引数で、設定ファイル（redis.conf）の位置を指定します。

```
$ redis-server ./redis.conf
```

Redisの起動はプロセスで確認できます。

```
$ ps -ax | grep redis-server
xxxx xx xx:xx.xx redis-server 127.0.0.1:6379
```
xは環境に応じて変化

このような表示が出力されればRedisは起動しています。

CLIから使ってみよう

Redisには標準でCLI（Redisと対話するためのコマンドラインツール）が用意されています。

```
$ redis-cli
```

上記のコマンドを実行すると、対話モードになります。

```
127.0.0.1:6379>
```

このような表示が出力されれば準備完了です。CLIからはRedisのコマンドを実行できます。

次のコマンドを実行して、値の登録と取り出しをしてみましょう。「test」というキーに対して、"value"という値を設定しています。

```
127.0.0.1:6379>set test "value"
127.0.0.1:6379>get test
"value"
```

Redisを停止するときは次のコマンドを実行します。

```
127.0.0.1:6379>shutdown
```

対話モードに入らずともCLIの引数にコマンドを入力することで、任意のコマンドを実行できます。たとえば上記のRedisの停止であれば、

```
$ redis-cli shutdown
```

と実行できます。

このほかにもたくさんのコマンドが用意されています。運用に役立つコマンドをいくつか**表1**

注10）wgetに入れるソースのパスはバージョンによって変更になります。最新バージョンについては公式サイトを参考にしてください。

注11）Ctrl＋cを入力するとRedisを終了し、操作できるようになります。

にまとめたので参考にしてください。

アプリケーションから使ってみよう

Redisの公式サイトには、各言語で利用できるクライアントライブラリが紹介されています[注12]。お勧めのライブラリもわかるようになっていますので、皆さんの使っている言語をチェックしてみてください。

ここでは、人気のプログラム言語であるPHPからRedisを操作する方法についてご紹介します。公式サイトでもお勧めされているphpredis[注13]を使ってみましょう。なおここでは、パッケージマネージャのyumが利用できるLinux環境を想定しています。

・phpredisをインストールできるように、epelをインストール

```
$ sudo yum install epel-release
```

・PHP、phpredisをインストール

```
$ sudo yum install php php-pecl-redis
```

リスト1はRedisに接続して値を保存し、それを取り出して出力するPHPプログラムのサンプルです。「redistest.php」というファイル名で作成し、次のコマンドで保存したプログラムを実行します。

```
$ php redistest.php
```

実行すると、Redisから取り出した値“test_value”が表示されます。

注12) URL http://redis.io/clients
注13) URL http://redis.io/clients#php

▼表1　運用に役立つCLIコマンド例

CLIコマンド	処理内容
CONFIG SET	confの設定値を変更することが可能。起動中に動的に設定を変更できるが、再起動すると設定は失われてしまうので要注意。設定可能な項目は、CONFIG GET *で表示されるもの
KEYS	正規表現で一致したキーを取得することができる。コマンドKEYS *で、登録済みのすべてのキーを取得可能
FLUSHALL	Redisに保存されているすべてのデータを削除する。Redisの初期化時などに有効
MONITOR	あるクライアントで実行すると、Redisに対して実行されたすべてのコマンドがダンプされる。実行状態を確認するのに役立つ

設定項目

この節ではRedisで設定可能な項目について、いくつかピックアップして紹介します。前述の「起動してみよう」でも少し説明しましたが、設定ファイルのサンプルとして、Redisインストール時に解凍したディレクトリに「redis.conf」が存在します。設定項目の説明が載っていますので目を通しておくことをお勧めします。

設定ファイルはRedis起動時の引数として指定します。未指定の場合には、デフォルトの設定値が使用されます。

```
redis.confを指定する例
$ redis-server ./redis.conf
```

データバックアップ

Redisには2種類のデータバックアップ方法が存在します。

RDBファイルでのバックアップ

RedisはRDBファイルというデータのスナッ

▼リスト1　redistest.php

```
<?php

$redis = new Redis();
$redis->connect("127.0.0.1",6379);

// 登録済みの値を削除
$redis->delete('test_key');

// 値をセットする
$redis->set('test_key', 'test_value');

// 値を取得する
$value = $redis->get('test_key');

// 表示
echo $value;
```

プショットを出力できます（**図8**）。RDBファイルを出力しておけば、そこから簡単にデータをリストアすることが可能です。

設定ファイルのsaveでは、RDBファイルを出力するタイミングを指定します。設定方法は独特で「X秒間にY回の変更があったらバックアップファイルを出力する」という指定を行います。この条件は複数指定可能です。次の設定だと「10分間（600秒）に100回の変更があったらファイルを出力する」という指定をしていることになります。

```
save 600 100
```

RDBファイルの出力時には、Redisは子プロセスを作成して動作を行います。よって親プロセスの動作が影響を受けないようになっています。RDBファイルを出力しない場合には、次の設定を行います。

```
save ""
```

RDBファイルを利用することで、データのリストアをスピーディに行うことができます。ただし、前回RDBファイルを出力した以降に書き込まれたデータについては、障害時に消失する可能性がある点に注意が必要です。

SAVEまたはBGSAVEコマンドを利用して、明示的にRDBファイルにバックアップすることも可能です。SAVEコマンド実行中はすべてのリクエストをブロックしますが、BGSAVEならリクエストを受け付けたままバックアップを実行できます。

AOFでのバックアップ

もう1つのデータバックアップ方法が、AOF（Append Only File）を使ったものです。設定ファイルに次のように記述することで有効になります。

```
appendonly yes
```

AOFは追記専用のログファイルで、Redisへ更新を行った内容が記載されています（**図9**）。AOFに書いた直後はディスクに書き込まれておらず、バッファキャッシュ上に存在します。ファイルに書き出す（fsyncを実行する）タイミングはalways、everysec、noの3つが選択できます。デフォルトはeverysecで、毎秒書き出しを行います。alwaysにすると毎回ログ書き出しを行うため、パフォーマンスが著しく低下します。noにするとfsyncでの書き出しは行わず、データの書き出しはOSに任せるため高速です。

AOFはファイルサイズが大きくなり、バックアップからのリストアにも時間がかかります。しかし任意の地点までの状態をリストアできるというメリットがあります。たとえば誤ったコマンドの実行で全データを消してしまった場合などでも、その命令を取り除いてAOFを読み込めば、データを復旧することが可能です。

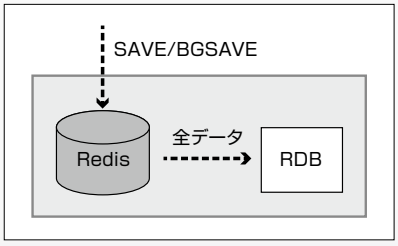
◆ ◆ ◆

このように、Redisにはデータ消失を防ぐ機能が備わっています。データの重要性や求められるパフォーマンスに応じて、最適なものを選択しましょう。

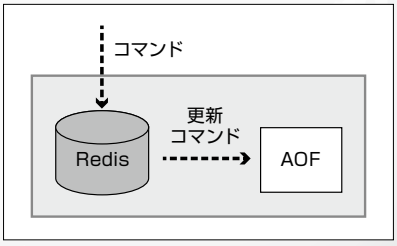
ログ出力

Redisには「実行ログ」「スローログ」という2種類のログが存在します。ここではログの用途と設定方法について解説します。

▼図8　RDBファイルによるバックアップ



▼図9　AOFによるバックアップ



実行ログ

loglevelでログの出力レベルを指定します。ログレベルにはdebug、verbose、notice、warningが存在し、デフォルトではnoticeに設定されています。用途に応じて切り替えるのが望ましいですが、若干ですがパフォーマンスに影響を与える点に注意してください。logfileでログの出力先ファイルを指定します。

スローログ

slowlog-log-slower-thanを指定することで、遅いコマンドをログとして保持することが可能です。slowlog-max-lenで指定した件数をメモリに保持します。ログを確認するためには、SLOWLOG GETコマンドを利用します（**図10**）。遅いコマンドを特定できるので、パフォーマンスチューニングに役立ちます。

メモリが上限に達したときの挙動

maxmemory-policyを指定することで、Redisの利用できるメモリが上限に達したときの挙動を次の6つから選択することができます。

- noeviction：エラーを返してこれ以上の登録は行わない。デフォルト設定
- volatile-lru：expireが設定されているキーからLRUアルゴリズム[注14]を使って削除
- allkeys-lru：すべてのキーからLRUアルゴリズムを使って削除
- volatile-random：expireが設定されているキーからランダムに削除
- allkeys-random：すべてのキーからランダムに削除
- volatile-ttl：expireの期限が近いものから削除

冗長化

この節ではRedisを冗長化する方法を紹介します。Redisには複数のサーバでデータを冗長化させるためのしくみがいくつか存在します。

レプリケーション

1台のMasterと複数台のSlaveでレプリケーションを構築することができます（**図11**）。データ参照のアクセスをSlaveに振り分けることで負荷を分散したり、集計やデータバックアップなど負荷の高いコマンドをSlaveで実行することでMasterの動作に影響を与えないようにする、などの用途で活用できます。

また、複数台のサーバでデータを保持するので、Masterに障害が起こってデータが消失したときでもSlaveからデータの復旧が可能です。レプリケーションは非同期で行われるため、MasterとSlaveでデータの差分が出ることがあります。Masterで登録したデータがSlaveから取得できないケースもありえる点には注意が必要です。Slaveはデフォルトで更新不可として、参照専用で動作します。

Redis Sentinel

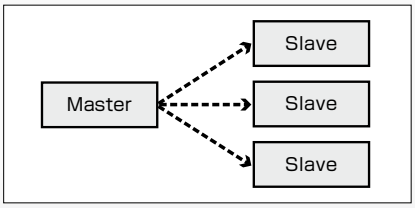
Redis 2.6からサポートされた、サーバの死活監視／通知および自動フェイルオーバー機能

注14）LRUとはLeast Recently Usedの略で、未使用の時間が最も長いものを抽出するアルゴリズムです。これによって、重要度の低いと思われるデータを操作することができます。

▼図10　スローログの確認

```
127.0.0.1:6379> slowlog get 10
1) 1) (integer) 3   自動採番されたID
   2) (integer) 1476082769   コマンド実行時のUnixタイムスタンプ
   3) (integer) 81   コマンド実行にかかった時間（マイクロ秒）
   4) 1) "slowlog"   以下、コマンドとそのパラメータ
      2) "get"
      3) "10"
```

▼図11　レプリケーション



を提供する管理サーバです。Sentinelが Master/Slaveサーバを監視していて、Master のダウンを検知したらフェイルオーバー（Slave のMaster昇格）処理を行います（**図12**）。

ただし、Sentinel自身はプロキシサーバ的な機能は持ちません。フェイルオーバー時のアプリケーションからのMaster接続切り替えは、別のしくみで行う必要があります。

Redis Cluster

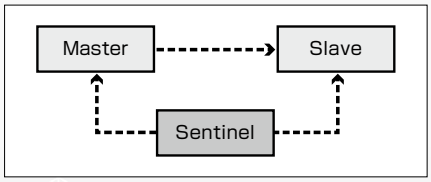
Redis 3.0で実装された、複数台のRedisサーバをクラスタリングするしくみです。Redis SentinelがMaster-Slaveをベースにしているのに対して、Redis Clusterは複数台のMaster（ノード）でデータを分散して保持します。それぞれのノードをMaster-Slave構成にすることができ、Masterの障害発生に自動でSlaveをMasterに昇格させるしくみを持っています（**図13**）。

どのノードがどのデータを持つのかは、キー値によって決定します。各ノードに0〜16383の範囲で数字（slot）が割り当てられており、キー値を計算してslotを割り出し、対象のサーバを決定します。ノードはお互いに監視していて、割り当てられているslot以外のキー値でリクエストを受けると、対象のノードに命令を転送するようにレスポンスを返します。

レプリケーションに比べて構成は複雑ですが、次の点でメリットがあります。

- データ書き込みの負荷分散ができる
- 自動でフェイルオーバーできる
- 各ノードがお互いを監視するので、監視専用のサーバが不要

▼図12　Redis Sentinel



注意点

Redis 3.0から実装されたRedis Clusterですが、クライアントライブラリによっては対応していないものも多くみられます。導入を検討する際に注意が必要です。また、リリースされて間もないので機能改善スピードも速いです。ネットなどの情報を参考にする場合には、対象バージョンに気をつけるようにしましょう。

◆ ◆ ◆

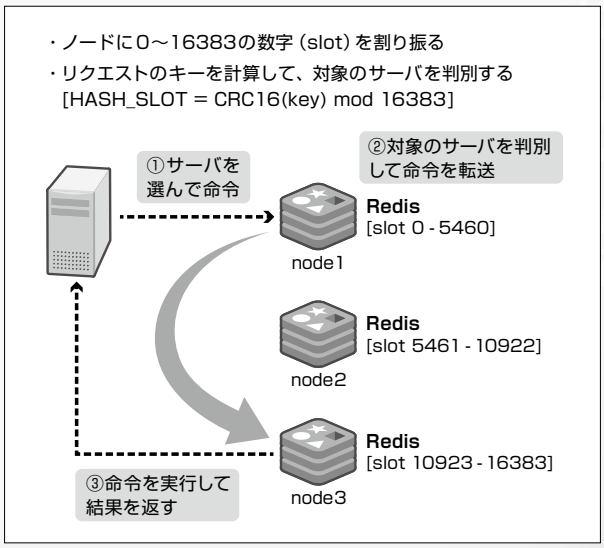
このように、Redisには冗長化を行うしくみが複数存在します。用途に応じた構成を検討してみてください。

さいごに

筆者は約5年前からRedisを利用していますが、Webアプリケーションを高速化する際の選択肢がとても広がりました。またRedis自身の開発のスピードが速いので、新機能を試すのがとても楽しみです。

今回の記事がシステム開発の選択肢を広げるきっかけになってもらえれば幸いです。SD

▼図13　Redis Cluster



第5章

RDBMSとNoSQLのいいとこ取り！

NoSQLとしても使えるMySQLとMySQL Cluster

Author 梶山 隆輔（かじやま りゅうすけ） 日本オラクル（株）

オープンソースのRDBMSであるMySQLは、近年のバージョンアップでNoSQLとしての機能が備わっています。シンプルな操作で高速にデータを読み書きできるNoSQLの利点を享受しながら、トランザクション管理などのRDBMSの恩恵を受けることもできます。

MySQLとNoSQLの関係

MySQLは「世界で最も普及しているオープンソースデータベース」として、大規模なWebアプリケーションやクラウドから組み込みデータベースまで幅広く利用されています。現在のMySQLはさらに進化を続け、RDBMSとNoSQLのハイブリッド型のデータストアとしての利用が可能になっています。

MySQL 5.6からKey-Value型APIのInnoDB memcachedプラグインが同梱され、Key-Value Store（KVS）として利用が可能です。

MySQL 5.7ではJSONデータ型が追加され、さらにプラグインとしてNoSQL APIであるX DevAPIが追加され、ドキュメントデータベースとして利用可能となります。

MySQL Serverとは別製品のMySQL Clusterは、共有ディスクを必要としない分散型の全ノードアクティブなRDBMSクラスタです。SQLと複数のNoSQL APIをサポートし、JSONデータ型も実装されているハイブリッド型データストアです。

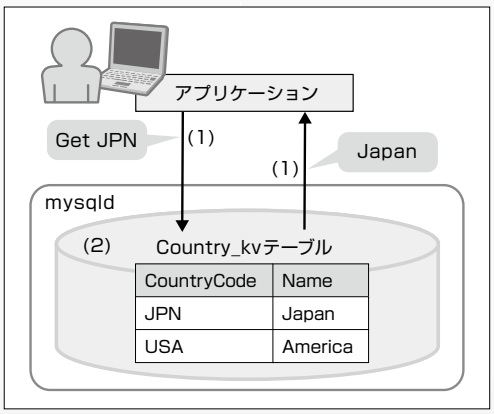
MySQLのKey-Value型API

MySQL 5.6では、プラグインとしてmemcachedベースのKey-Value型APIが利用可能となりました。InnoDB memcachedプラグイン（以下、memcachedプラグイン）と呼ばれ、アプリケーションからのアクセスはmemcachedプロトコルおよびクライアントライブラリを使用し（**図1の(1)**）、データはInnoDBストレージエンジンを利用したテーブルに格納されます（**図1の(2)**）。

通常のmemcachedは複数のサーバのメモリ上にデータを分散配置できます。一方、MySQLのmemcachedプラグインでは、データを分散して配置することなく1台のMySQL Server上に格納します。また、メモリ上だけではなくデータをディスク上にも格納できます。

memcachedそのものにはトランザクションの概念はありませんが、memcachedプラグインからのアクセスはInnoDBストレージエンジンのトランザクションの管理に含まれます。

▼図1 memcachedプラグインの利用イメージ図



なお、このmemcachedプラグインはLinux、Solaris、Mac OS Xで利用可能です。

アーキテクチャ概要

memcachedプラグインは、MySQL Serverプロセス（mysqld）内で、memcached APIを提供します。このAPIでは、クライアントプログラムから実行されるmemcachedのgetやsetなどのコマンドを、InnoDBストレージエンジンのAPIに変換しています（**図2の（1）**）。memcachedプラグインからのアクセスは、MySQL Serverのユーザ認証やパーサーによる構文解析、オプティマイザーによる実行計画の処理などを介さずに、直接ストレージエンジンにアクセスするため、オーバーヘッドが抑えられ、高いスループットが期待できます。

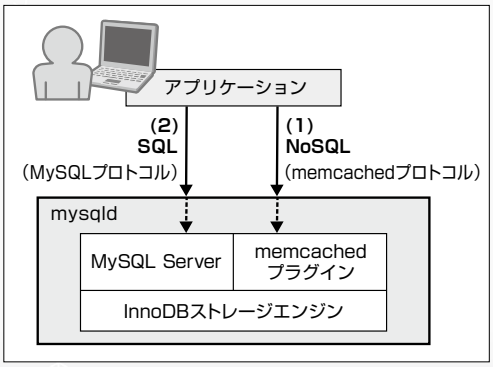
データはテーブルに格納されており、MySQLのSQL文でのアクセスも可能であり（**図2の（2）**）、Key-Value StoreとRDBMSとしての両立ができています。またキャッシュとデータの同期についてもアプリケーション側で実装する必要はなくなります。

memcachedプラグインを利用する

memcachedプラグインはデフォルトでは有効になっていないため、次の手順で設定／インストールを行う必要があります。

まず、innodb_memcached_config.sqlスクリプトで**表1**の3つの設定テーブルを作成します。

▼図2　memcachedプラグインのアーキテクチャ



その後、INSTALL PLUGINコマンドにてlib memcached.soをインストールします。memcachedプラグインは通常のmemcachedと同じTCPポートの11211をリッスンします。

ここまでの設定を終えた前提で、memcachedプラグインの利用例を示したのが**図3、4、5**です。

図3の（a）では、データを格納するためのテーブルを作成しています。**図3の（b）**では、containersテーブルに、キーのプレフィックスによって「どのテーブルのどの列にキーや値を格納するか」を定義しています[注1]。プレフィックスがない場合はcontainersテーブルの1レコード目に定義されたマッピング情報を使います。

図4の（c）では、setコマンドでデータを格納しています。格納先のテーブルは同じですが、キーのプレフィックスによって値の格納先となる列が異なっていることを確認してください（**図4の（d）**）。

図5の（e）は、（c）（d）で格納したデータをgetコマンドで取得する例です。

キャッシュポリシーの変更

表1のcache_policiesテーブルを利用すると、次のキャッシュポリシーをget、set、delete、およびflushの操作ごとに定義できます。

- innodb_only：InnoDBのテーブルへのデータ格納のみ
- cache-only：memcachedが管理するメモリへのデータ格納のみ
- caching：InnoDBのテーブルおよびmemcachedのメモリへのデータ格納

注1）InnoDB memcached Plugin Internals URL http://dev.mysql.com/doc/refman/5.7/en/innodb-memcached-internals.html#innodb-memcached-containers-table

▼表1　memcachedプラグインの設定テーブル

テーブル名	役割
containersテーブル	データ格納テーブルと列のマッピングを定義
cache_policiesテーブル	キャッシュポリシーの定義
config_optionsテーブル	memcached関連の構成オプションを定義

▼図3　データ格納用テーブルとcontainersテーブルの準備

```
(a) テーブル作成
mysql> CREATE TABLE test.country_kv (
  CountryCode VARCHAR(3) NOT NULL,
  Name VARCHAR(256),
  Continent VARCHAR(256),
  col_flags INT(11),    ←incrやrependコマンドの処理対象を指定するフラグ
  col_cas BIGINT(20),   ←casコマンドのcas ID
  col_exp INT(11),      ←値の有効期限
  PRIMARY KEY (CountryCode)
);

(b) containersテーブルにマッピングを定義する
キーのプレフィックスがcountryの場合、testデータベースのcountry_kvを使用 CountryCode列がキー、Name列を値とする
mysql> INSERT INTO innodb_memcache.containers VALUES ("country", "test", "country_kv", ↵
"CountryCode", "Name", "col_flags", "col_cas", "col_exp", "PRIMARY");

キーのプレフィックスがcountry_ncの場合、Name列とCountry列のデータを|(バーティカルバー)でつないだものを値とする
列の区切りはconfig_optionsテーブルのseparator列で指定
mysql> INSERT INTO innodb_memcache.containers VALUES ("country_nc", "test", "country_kv", ↵
"CountryCode", "Name|Continent", "col_flags", "col_cas", "col_exp", "PRIMARY");
```

▼図4　データの格納

```
memcachedのTCP/IPポート11211に接続
$ telnet 127.0.0.1 11211

(c) memcachedのsetコマンドにて値を格納
キーのプレフィックスがcountryの場合
set @@country.AAA 0 0 16   ←@@から.(ピリオド)までがキーのプレフィックス。@@country.AAAのあとはフラグ、cas ID、レコード長
Dummy Country 01   ←格納する値を指定
STORED

キーのプレフィックスがcountry_ncの場合
set @@country_nc.JPN 0 0 10
Japan|Asia   ←Name列とCountry列のデータを|(バーティカルバー)でつないだものを値とする
STORED

(d) 格納されたデータをSQL文にて確認
mysql> SELECT CountryCode, Name, Continent ↵
FROM test.country_kv;
+-------------+-----------------+-----------+
| CountryCode | Name            | Continent |
+-------------+-----------------+-----------+
| AAA         | Dummy Country 01 | NULL      |
| JPN         | Japan           | Asia      |
+-------------+-----------------+-----------+
```

▼図5　データの取得

```
(e) memcachedのgetコマンドで値を取得
キーのプレフィックスがcountryの場合
get @@country.JPN
VALUE @@country.JPN 0 5
Japan
END

キーのプレフィックスがcountry_ncの場合
get @@country_nc.JPN
VALUE @@country_nc.JPN 0 10
Japan|Asia
END
```

KVSとRDBMSとのレプリケーション

MySQL Server の innodb_api_enable_binlog オプションを有効にすると、memcachedプラグインから書き込んだ処理もバイナリログに書き込まれ、MySQLのレプリケーションによってほかのMySQL Serverに展開できます。

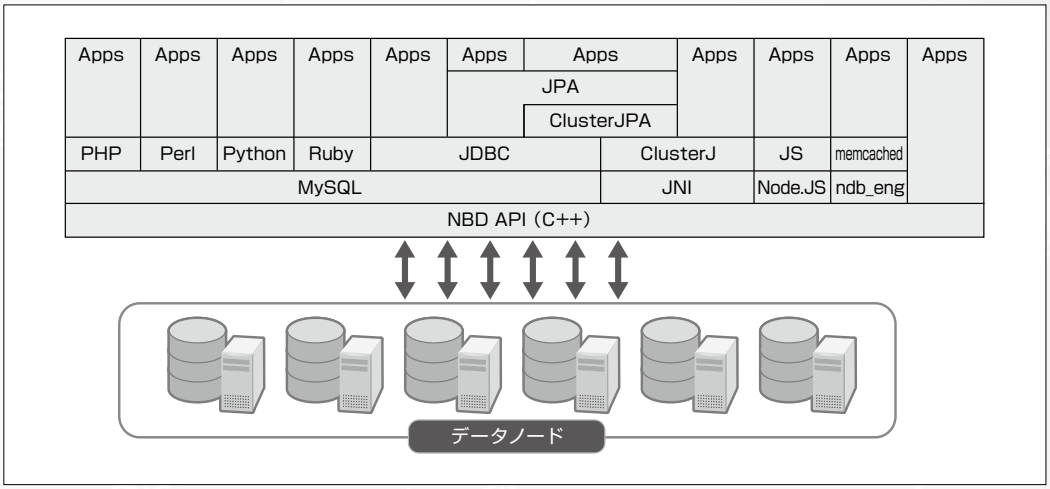
MySQLのmemcachedプラグインによって、同じデータに対してSQLとKey-Valueによるトランザクショナルな処理が可能となるばかりか、Key-Valueデータストア的な使い方のMySQLからRDBMS的な使い方のMySQLへのレプリケーションも可能となります。

トランザクション対応KVSとしてのMySQL Cluster

MySQL Clusterは高い可用性と性能拡張性を持った、全ノードアクティブなRDBMSクラスタです。さらに各種のNoSQL APIを持っており、SQLとNoSQLでのアクセスが可能です(**図6**)。

SQLを利用するアプリケーションは、MySQL Cluster構成内でSQLノードと呼ばれる

▼図6　MySQL Clusterのアーキテクチャ



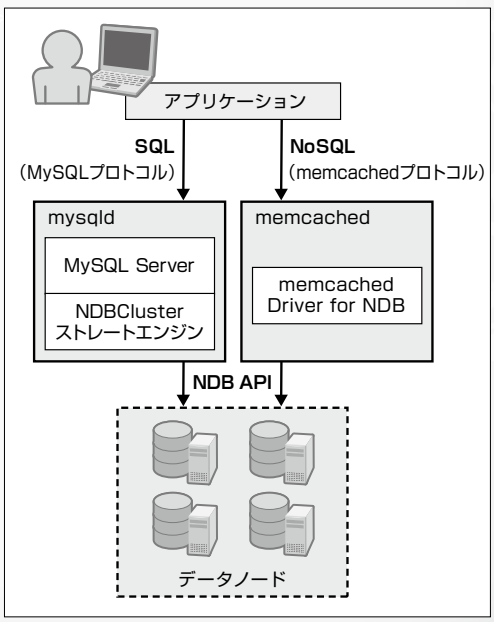
MySQL Serverプロセス（mysqld）に対して接続してSQLを発行し、ndbclusterストレージエンジンがデータノードとやりとりをしてデータの参照更新を行います。

MySQL Clusterではデータは各テーブルの主キー（またはユニークキー）のハッシュパーティショニングによって分割され、データノード間で変更点を同期的に複製することで冗長性を持たせています。パッチ適用やテーブル定義の変更、ノード追加などのメンテナンス中も無停止で利用できます。

MySQL Clusterの開発当初はC++のAPIであるNDB APIを持っているだけでしたが、MySQL Serverと組み合わせることでSQL対応のデータストアとなりました。NDB APIはデータノードに直接アクセスするためオーバーヘッドが小さく、リアルタイム性を求められる通信システムなどで利用されています。MySQL Cluster 7.4では、データノードを増加させることでリニアにスループットが向上しており、データノード32台構成で秒間2億件の参照処理を実現しています[注2]。

このNDB APIを、JNI（Java Native Interface）でラップしたJava用のAPI「ClusterJ」や、JPA（Java Persistence API）に準拠させた「ClusterJPA」も用意しています。MySQL ClusterのmemcachedAPIは、MySQL Serverのmemcachedプラグインとは異なったアプローチを採っています。アプリケーションからはMySQL Cluster用のドライバを追加したmemcachedに接続しリクエストを送ることで、データノードのデータを読み書きできます（**図7**）。キー

▼図7　MySQL Clusterのmemcached APIのアーキテクチャ



注2）MySQL Clusterベンチマーク　URL http://www-jp.mysql.com/why-mysql/benchmarks/mysql-cluster/

のプレフィックスによって利用するテーブルや列を制御すること、キャッシュのポリシーを設定することも可能です。Node.js用のAPIも用意されており、Node.jsアプリケーションからSQLをいっさい書かずにデータノードのデータを読み書きできるようになっています。

MySQL Clusterは高い可用性と性能拡張性を持つRDBMSクラスタとしての性格だけではなく、トランザクション対応のNoSQLデータストアとしての性格も兼ね備えています。いずれのAPIからのアクセスも、データノードでトランザクショナルに処理されます。また外部キーによる制約も適用されます。さらに変更内容をSQLノードに取り込んでバイナリログに記録し、ほかのMySQL ClusterやMySQL Serverにレプリケーションできます。

MySQLのJSON対応

MySQL 5.7では、JSONデータ型および、JSONデータ型に格納されたJSONドキュメント内の値の取得や変更などを行うためのJSON関数が追加されました。

MySQLのJSONデータ型とJSON関数

MySQLのJSONデータ型は、JSONドキュメントをバイナリ形式にて格納し、文字コードはutf8mb4として扱います。データ全体を走査することなくキーや配列の内部インデックスを参照させることで、検索性能の向上を図っています。JSONデータ型の利用例を**図8**に示します。

MySQL 5.7では、JSON文字列およびJSONデータ型を処理するための関数や演算子が複数用意されています。また、JSONドキュメント内の階層（パス／Path）を表現する方法が用意されています（**図9**）。

生成列と、JSONデータ型に対するインデックス

MySQL 5.7では生成列（Generated Column）という機能が実装されました[注3]。JSON関数を使ってJSONドキュメントから抽出した値を生成列に格納し、その生成列にインデックスを作

注3）生成列は既存の列に対して演算を行った値を格納するしくみ。デフォルトではVIRTUAL生成列となっており、該当のレコードを参照するたびに演算が行われその値を返す（実際にストレージには格納されない）。STORED生成列にすると、データの挿入または更新時に演算が行われて、その値をストレージに格納する。

▼図8　JSONデータ型の列を持つテーブルの作成とデータの格納

```
JSONデータ型の列menuを持つテーブルを作成
mysql> CREATE TABLE pz (menu JSON);

データを追加
mysql> INSERT INTO pz(menu) VALUES ('{
    '> "Name":"Plain Pizza",
    '> "price":300
    '> }');

配列を含むJSONドキュメントを追加
mysql> INSERT INTO pz(menu) VALUES ('{
    '> "Name":"Cheesy Pizza",
    '> "price":400,
    '> "toppings":"More Cheese",
    '> "additionals":[{"Name":"B Cheese","price":100}]
    '> }');

JSONドキュメントではない文字列を追加しようとするとエラーになる
mysql> INSERT INTO pz(menu) VALUES ('some text');
ERROR 3140 (22032): Invalid JSON text: "Invalid value." ➥
at position 0 in value for column 'pz.menu'.
```

複数行に渡るJSONドキュメントの場合、改行を入れてもとくに問題ない。JSONドキュメント全体をシングルクォーテーションで囲む。JSONドキュメント内の文字列はダブルクォーテーションで囲む

▼図9　JSON関数およびJSON演算子の利用例

```
JSONを展開するJSON_EXTRACT関数の利用例
mysql> SELECT JSON_EXTRACT(menu, ➥
"$.Name") FROM pz;
+-----------------------------+
| JSON_EXTRACT(menu, "$.Name") |
+-----------------------------+
| "Plain Pizza"               |
| "Cheesy Pizza"              |
| "Classic Pizza"             |
+-----------------------------+

JSON_EXTRACT関数と同様の動作をするJSON演算子 -> の利用例
mysql> SELECT menu->"$.Name" FROM pz;
+-----------------+
| menu->"$.Name"  |
+-----------------+
| "Plain Pizza"   |
| "Cheesy Pizza"  |
| "Classic Pizza" |
+-----------------+
```

成することで、JSONデータ型へのインデックスを実現できます（**図10**）。

◆ ◆ ◆

JSONドキュメント内の値とテーブル内の値のJOINや、JSONドキュメントとテーブルを1つのトランザクションで更新することなどは、異なるデータストア間では実現しにくいものです。JSONデータ型を使うことで、これらの処理をMySQL内で実施できることは大きなメリットとなります。

MySQLのドキュメントデータベース機能

MySQL Document Store

MySQL Document Storeは、NoSQLの1つであるドキュメントデータベースとしてMySQLを利用するための方法です[注4]。リレーショナルデータベースであるMySQLをこれまでどおり運用しながら、アプリケーション開発者から要望の多いより柔軟な開発を実現することを目指しています。

注4）MySQL as a Document Store URL http://dev.mysql.com/doc/refman/5.7/en/document-store.html

▼図10　生成列の作成とインデックスの追加

```
生成列の作成
mysql> ALTER TABLE pz ADD COLUMN pz_name VARCHAR(32)
    -> GENERATED ALWAYS AS
    -> (JSON_UNQUOTE(JSON_EXTRACT(menu, '$.Name')))
    -> VIRTUAL;

作成した生成列に対してインデックスを作成
mysql> ALTER TABLE pz ADD INDEX(pz_name);

EXPLAINで実行計画を確認
mysql> EXPLAIN SELECT * FROM pz WHERE pz_name = "Cheesy Pizza"¥G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: pz
   partitions: NULL
         type: ref
possible_keys: pz_name
          key: pz_name
      key_len: 35
(..略..)
```

JSON内のName要素を展開し、その値をVIRTUAL生成列とする

keyの項目でインデックスpz_nameが利用できていることが確認できる

MySQL 5.7.12からMySQL X Plugin（以下、Xプラグイン）が利用可能となっています。MySQL 5.7のGA（製品版）リリース以降に追加された新機能のため、MySQL本体には含めずにプラグインとして提供することで、本機能を必要としない方には影響が出ないしくみとしています。インストールは次の方法で行います。

```
mysql> INSTALL PLUGIN mysqlx SONAME ➥
'mysqlx.so';
```

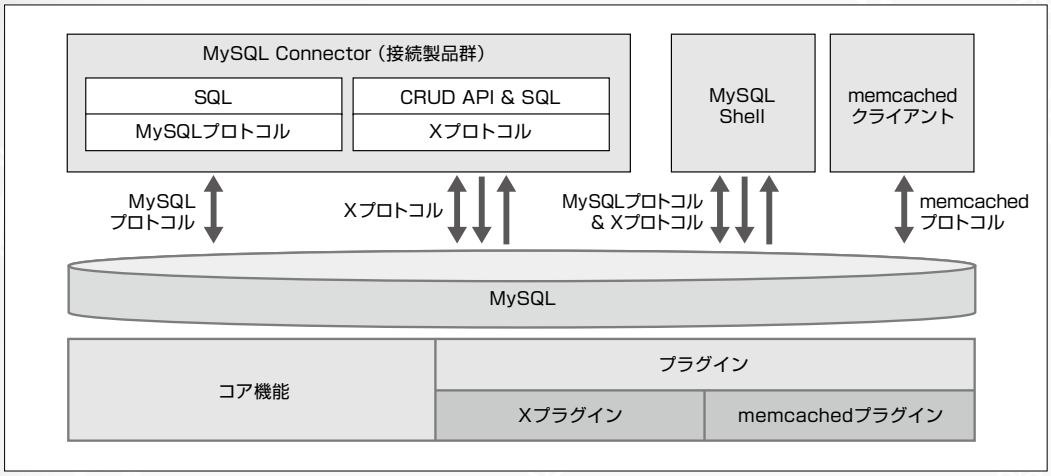
このXプラグインは、MySQLのクライアント／サーバ間の新しい通信プロトコルMySQL X Protocol（以下、Xプロトコル）を実装しています。従来のSQL文に加えて、CRUD（Create、Read、Update、Delete）操作でのデータの読み書きをサポートしています（**図11**）。そして、Xプロトコルをアプリケーションから利用するための新しいAPI群「MySQL X DevAPI」が用意されています。MySQL X DevAPIを試す環境としては、コマンドラインクライアントのMySQL X Shell（コマンド名「mysqlsh」）があります。

2016年10月現在、GA（製品版）はMySQL同梱のXプラグインのみです。MySQL X Shell、Node.js用／Java用／.Net用のMySQL X DevAPI対応Connector、Visual Studioへのプラグインは、RC（リリース候補版）またはDMR（開発途上版）となっています。

MySQL X DevAPI

MySQL X DevAPIは、JSONドキュメントとテーブルへのアクセスを一体化するもので、CRUD操作は「流れるようなインターフェース（Fluent Interface）」と呼ばれるスタイルとなっています。

▼図11　Xプラグインとmemcachedプラグインの利用イメージ図



従来、テーブルに対してはSQL文でアクセスしていましたが、MySQL X DevAPIを使うと、**表2**のようなCRUD操作でアクセスできるようになります。テーブル全体に対してはTableオブジェクトとして、JSONドキュメントに対してはCollectionオブジェクト（以下、コレクション）として、CRUD操作ができるようになります。とくにJSONドキュメント内のデータへのアクセスは、従来のSQL文でのアクセスと比較してシンプルなります。

コレクションはcreateCollection()関数で作成できます。作成すると、**図12**のような定義のテーブルが作成されます。

図13、14は、従来のMySQLクライアントでSQLを用いてJSONデータ型へアクセスした様子と、MySQL X ShellのJavaScriptモードでMySQL X DevAPIを用いてCRUD操作を試した様子です。後者ではSQL文を使わずにデータにアクセスできている点、またJSONドキュメントのデータ抽出がシンプルになっている点を確認してください。

MySQL X Shellで、JavaScrptおよびPythonからMySQL X DevAPIを利用する場合のチュートリアルも用意されています[注5]。ぜひお試しください。

▼表2　テーブルとJSONドキュメントに対してCRUD操作を行う関数

処理	CRUD操作	テーブル	JSONドキュメント
作成	Create	Table.insert()	Collection.add()
参照	Read	Table.select()	Collection.find()
更新	Update	Table.update()	Collection.modify()
削除	Delete	Table.delete()	Collection.remove()

▼図12　createCollection()関数で作成したコレクションのテーブル定義

```
mysql> SHOW CREATE TABLE tbl_x¥G
*************************** 1. row ***************************
       Table: tbl_x ←テーブル名はcreateCollection()関数の引数で指定した名称
Create Table: CREATE TABLE `tbl_x` (
  `doc` json DEFAULT NULL, ←JSONデータ型のdoc列
  `_id` varchar(32) GENERATED ALWAYS AS (json_unquote(json_extract(`doc`,'$._id'))) ➡
  STORED NOT NULL,PRIMARY KEY (`_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
                ↑JSONドキュメント内に格納される自動生成されたidを抽出したSTORED生成列
```

注5）X DevAPI User Guide 11.5 Table CRUD Functions
URL https://dev.mysql.com/doc/x-devapi-userguide/en/crud-ebnf-table-crud-functions.html
X DevAPI User Guide 11.3 Collection CRUD Functions
URL https://dev.mysql.com/doc/x-devapi-userguide/en/crud-ebnf-collection-crud-functions.html

▼図13 「JSONドキュメントへのSQLアクセス」と「コレクションへのCRUD操作」の比較(データの追加)

```
MySQLクライアントでのSQL
mysql> INSERT INTO tbl_sql(doc) VALUES('{"id": 1, "name": "Mike", "Team": "Products"}');
Query OK, 1 row affected (0.06 sec)

MySQL X Shellでのテーブルに対するCRUD操作
mysql-js> db.tbl_sql.insert(['doc']).values('{"id": 2, "name": "Joe", "Team": "Sales", ➥
"Title": "VP"}').values('{"id": 3, "name": "Tomas", "Team": "Development"}').execute();
Query OK, 2 items affected (0.06 sec)

MySQL X Shellでのコレクションに対するCRUD操作
mysql-js> db.tbl_x.add({"id": 1, "name": "Mike", "Team": "Products"}).execute();
Query OK, 1 item affected (0.06 sec)
```

新通信プロトコル MySQL X Protocol

Xプラグインによって、今後の機能拡張の土台となる新しいXプロトコルが利用可能となります。Xプロトコルは非同期APIをサポートし、並列処理や複数のリクエストをまとめて送信するパイプライン処理が可能です。アクセス先などのルーティング情報、シャーディング構成のデータ分割のキーの情報、また参照と更新の分散などのサポートを準備中です[注7]。

▼図14 「JSONドキュメントへのSQLアクセス」と「コレクションへのCRUD操作」の比較(絞り込み検索)[注6]

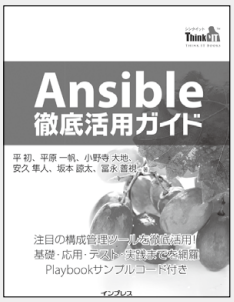
```
MySQLクライアントでのSQL
mysql> SELECT doc FROM tbl_sql WHERE JSON_UNQUOTE➥
(JSON_EXTRACT(doc,'$.name')) LIKE 'Mike';
+-----------------------------------------+
| doc                                     |
+-----------------------------------------+
| {"id": 1, "Team": "Products", "name": "Mike"} |
+-----------------------------------------+
1 row in set (0.00 sec)

MySQL X Shellでのテーブルに対するCRUD操作
mysql-js> db.tbl_sql.select(['doc']).where("JSON_UNQUOTE➥
(JSON_EXTRACT(doc,'$.name')) like 'Mike'");
+-----------------------------------------+
| doc                                     |
+-----------------------------------------+
| {"id": 1, "Team": "Products", "name": "Mike"} |
+-----------------------------------------+
1 row in set (0.01 sec)

MySQL X Shellでのコレクションに対するCRUD操作
mysql-js> db.tbl_x.find('name like "Mike"');
[
    {
        "Team": "Products",
        "_id": "52d0afae9286e611cd3b81294d063805",
        "id": 1,
        "name": "Mike"
    }
]
1 document in set (0.00 sec)
```

まとめ

MySQLはRDBMSだけではなく、Key-Valueデータストア、そしてドキュメントストアを組み合わせたハイブリッド型のトランザクション対応データストアとしての利用の可能性が大きく広がっています。JSONデータ型はMySQL Cluster 7.5にも導入され、全ノードアクティブなデータベースクラスタとトランザクション対応Key-Valueデータストアからさらに機能強化されます。MySQLおよびMySQL ClusterをRDBMSとしてお使いの方も、トランザクション対応のNoSQL製品をお探しの方も、ハイブリッド型のデータストアとしての利用をぜひご検討ください。SD

注6) URL http://dev.mysql.com/doc/refman/5.7/en/mysql-shell-tutorial-javascript.html
URL http://dev.mysql.com/doc/refman/5.7/en/mysql-shell-tutorial-python.html

注7) Xプロトコルの実装の詳細は次の情報を参照。MySQL Internals Manual Chapter 15 X Protocol
URL https://dev.mysql.com/doc/internals/en/x-protocol.html



Ansible徹底活用ガイド

坂本 諒太、安久 隼人、小野寺 大地、平原 一帆、平 初、冨永 善視 著
B5変形判／110ページ
1,800円＋税
インプレス
ISBN＝978-4-8443-8166-2

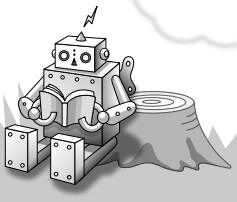
Infrastructure as Codeを実現するための構成管理ツールとして、Chef、Puppet、そしてAnsibleがよく挙げられる。いずれのツールも、パッケージのインストールやサービスの起動といった操作を仕様書のように記述しておき、コマンド1つで、複数サーバに対してそれら操作を実行できる。Ansibleは、RubyベースのDSLを使用するChef、Puppetと違い、YAMLという単純な表現形式で操作を記述する。またほかの2ツールと違ってサーバ・クライアント構成を取る必要はなく、この特徴も加えてハードルが低いツールとして人気がある。本書はそんなAnsibleに関するWeb記事の再編集本で、110ページと気軽に読める1冊。概要、始め方、使い方、良い記述の仕方と、入門にはぴったりの内容だ。

確かな力が身につくPHP「超」入門

松浦 健一郎、司 ゆき 著
B5変形判／344ページ
2,480円＋税
SBクリエイティブ
ISBN＝978-4-7973-8872-5

「確かな力が身につくシリーズ」のPHP版。マスコットキャラクターがキーウィでイラスト図を多用しているので親しみやすい。本文は9つのChapterに分かれ、Chapter1はPHPの概念、Chapter2はXAMPPを使用した環境設定で、PHP 7.0.8をベースに解説を行っている。環境設定はWindows 10を基準としているが、macOS関連もNoteとして若干の記載がある。解説の最初にエラーや文字コードにも触れているが、初心者にはハードルが高いかもしれない。Chapter3からは変数や関数、制御構造などを一通り試し、Chapter6以降では、MySQLを使ってショッピングサイトの構築まで行っているので難易度は高くなっている。全体的に駆け足での解説になっているので、PHP、Apache、MySQLなどの入門書の併読をお勧めする。

SD BOOK REVIEW



インフラエンジニアの教科書2

佐野 裕 著
A5判／256ページ
2,130円＋税
シーアンドアール研究所
ISBN＝978-4-86354-186-3

「ITにおける広範囲の業務を担当するインフラエンジニアは、どんな知識を習得すれば良いのか」を教えてくれる、まさに教科書と言える本書。プロトコル、OS、セキュリティ、サーバサイド開発言語といった各領域での重要な概念、用語を説明しながら、筆者が実務の中で培ってきたさまざまな実用情報をコラムとして載せている。新人インフラエンジニア向けだった前作の「インフラエンジニアの教科書」に比べ、本書は実務経験を数年積んだインフラエンジニアがステップアップできるような内容となっている。目立つものでは、「インターネットの運用と発展をつかさどる組織や団体」「RFCの読み方と作られ方」という、技術からはやや離れるが、知識として持っておけばエンジニアとしての視野がより広がるような章が設けられている。

ポートとソケットがわかればインターネットがわかる

小川 晃通 著
A5判／272ページ
2,280円＋税
技術評論社
ISBN＝978-4-7741-8570-5

本誌2016年10月号はおかげさまで大変好評だった。Amazonでは追加注文をたくさんいただいた。実は「Webのしくみ」特集は本書の抜粋だといっても過言ではない。もともと「ポートとソケット」のアイデアは本誌2014年5月号の新人向け特集だが、このテーマはずっと読者の皆さんに受け入れられているのだ。とてもありがたいことだ。皆、技術がどうなっているのか知りたいのだ。専門家は魔術のようにその秘密を隠す。あたかも既得権益を守る悪代官のようだ。だが本書はその秘密のベールを『小悪魔女子大生のサーバエンジニア日記』のaicoさんの力を借りてさらっと剥がしてしまう。知ることの力は偉大だ。いろんなことが爆発的に進むようになるからだ。本書でガッチリ未来をつかんでほしい。読んだ人ならばできるはずだ。

第2特集　HTML Java Ruby MySQL

文字コード攻略マニュアル

HTML・Java・Ruby・MySQLのハマりどころ

文字列の変換・数え上げ、Webページの作成、クローリング……。プログラミングと文字列は切っても切れない関係です。そして文字列を扱う以上、文字コードを避けては通れません。

本特集では文字コードの扱いで失敗しないために、文字コードの基本と主要プログラミング言語＋DBMSでの扱いを押さえましょう。Part1では文字列の歴史をひも解きながら、各文字コードの成り立ちと特徴を学びます。Part2からはHTML・Java・Ruby・MySQLをピックアップして、各処理系でどんな文字コードをサポートしているのかを解説し、文字化けやエラーを発生させないためのTipsを紹介します。

contents

PART 1

ゼロからはじめる文字コード

符号化のしくみと、ASCIIからUTF-8への系譜

コンピュータで扱うデジタルな情報は0と1からできている。そう知ってはいても、「Hello World☕!」という目の前のテキストがコンピュータでどのように扱われているか、ピンとこない方も多いのではないでしょうか。Part1ではコンピュータが文字をどのように扱うのか、その歴史としくみに触れながら各種文字コードを見ていきます。

Author 田所 駿佑(たどころ しゅんすけ) 株式会社ビズリーチ スタンバイ事業部 Twitter @todokr

0と1で文字をどう表現するか

「文字」と言われてイメージするものは何でしょうか？ 紙に書いたアルファベットやひらがな、人によっては古代エジプトのヒエログリフなどを思い起こすかもしれません。いずれも情報を記録・交換するための視覚的な表現[注1]であるということは共通していそうです。文字の形から「これはアルファベットの『A』」や「これはひらがなの『あ』」、「ヒエログリフの『葦の穂』」と私たちは判断しているわけです。

このような視覚的な表現をコンピュータで扱うにはどうしたら良いのでしょうか？ コンピュータでの情報の最小単位は0と1、すなわちビット(bit)です。1ビットは0もしくは1という2種類の情報を表現できます。たとえば、アンケートの解答の「はい」を1、「いいえ」を0にするといったルールを決めておけば、その結果を1ビットで表現できます。「はい」もしくは「いいえ」そのものではなく、ビットに置き換えることによって、情報がコンピュータで扱えるようになるわけです。

文字を表現する際も同様です。すなわち文字そのものではなく、「文字を表す符号」を0と1からなる表現によって扱っています。

文字を符号で表すとはどういうことでしょうか？ 簡単な例として、ジャンケン専用の架空の文字コードを考えてみましょう。ジャンケンの手はグー・チョキ・パーの3種類です。2ビットの情報量は$2^2 = 4$ですので、2ビットあればすべての手を表現できることがわかります。ここでは次のように定めてみました。

文字	符号
✊	00
✌	01
✋	10

「グーはビット列で00と表現する」などというルールが共有されていれば、00というビット列を見て「これは✊だ」と正しく解釈できます。このように文字をビット列で表すことを「符号化」といいます。符号化されていれば、視覚的な表現を使わずともビットの情報だけで文字のやりとりができるわけです。この「どの文字をどのようなビット列で表現するか」のルールがすなわち「文字コード」です。表現と解釈のルールが一致しない場合、情報を正しく扱うことができません。これがいわゆる「文字化け」です。

符号とフォント

ある文字を指し示す符号が文字コードですが、その文字がどのように視覚的に表現されるかを決定するのが「フォント」です。フォントを変更すれば文字の見た目が変化するということは、広く知られていると思います。

注意したいのが、“文字コードは視覚的な表現についての責務を持たない”という原則です。書道の表現を借りるなら、ある文字の一画がトメかハネか、はたまたハライなのかを指示する情報を、文字コードは持ちません。

フォントによる視覚的表現の差が顕著なのが

注1）もちろん点字など視覚的な表現に依らない文字もありますが、形から意味を読み取るという点では同じです。

絵文字です。Unicode Emojiについての規格書、Unicode Technical Report #51[注2]の「2 Design Guidelines」を見てみましょう(**図1**)。U+1F36D LOLLIPOPのそれぞれは、例示図形の「渦巻き模様を持つ」「持ち手がある」などコアな表現を守りつつも、向きなどが異なることがわかります。

▼図1　絵文字の表現例(出典：http://unicode.org/faq/emoji-examples.png)

文字コードの種類と歴史

ここまで見てきたように、文字コードとフォントが互いに手を取り合うことによって、コンピュータは文字を扱っていることがわかりました。ここからは各種文字コードとその歴史について見ていきましょう。

ASCII

現在一般的に使われる文字コードの祖先にあたるのが、1963年に誕生した「ASCII」です。当初はコンピュータではなく、テレックスと呼ばれるキーボード付きFAXのような通信機器で使われていました。American Standard Code for Information Interchangeの名のとおり、はじめから異なる端末間での情報交換を目的として作られた文字コードです。

ASCIIは7ビットで符号化された1バイトコードです。現在、たいていのコンピュータでは8ビットを1バイトとして扱いますが、そのような環境では最上位ビットを常に0にして使用します。扱える文字の種類は最大で$2^7 = 128$種類、コードポイント(個々の文字に割り当てられた数値)の範囲は0x00～0x7Fです。

表1はASCIIのコード表です。行が上位3ビット、列が下位4ビットを表しており、0x41は「A」、0x5Cは「\」となります。英数字や記号など、英語圏での情報交換に必要な文字が収録されていることがわかります。

図の網掛けの範囲に注目してください。0x00のNULから0x1FのUS、そして0x7Fの計33字は「制御文字」と呼ばれる特殊な文字です。現代においては大半が有名無実化してしまいましたが、0x0Aの「LF(改行)」や0x0Dの「CR(復帰)」など、プログラミングにおいて重要な意味を持つ制御文字もあります。目に見えない制御文字に対して、英数字や記号などを「図形文字」と呼びます。0x20の「SP(スペース)」も図形文字に含まれます。

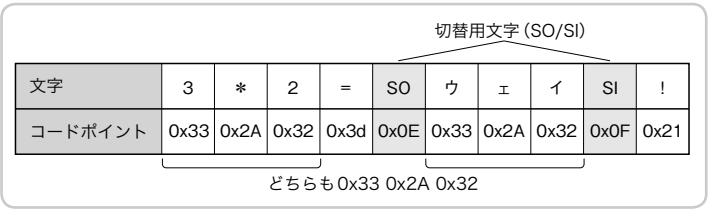
ASCIIの国際規格版にあたるのが、ISO(International Organization for Standardization)によって規格化された「ISO/IEC 646」です。アメリカ以外の国でも使用できるよう、各国の事情に合わせて一部の文字や記号の入れ替えが可能と定められましたが、アクセント表記を多用するヨーロッパなどの非英語圏では制約が大きいため、あまり使われていません。

▼表1　ASCIIコード表

	上位3ビット								
下位4ビット		0	1	2	3	4	5	6	7
	0	NUL	DLE	SP	0	@	P	`	p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	"	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	BEL	ETB	'	7	G	W	g	w
	8	BS	CAN	(	8	H	X	h	x
	9	HT	EM	)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[	k	{
	C	FF	FS	,	<	L	\	l	|
	D	CR	GS	-	=	M	]	m	}
	E	SO	RS	.	>	N	^	n	~
	F	SI	US	/	?	O	_	o	DEL

注2) URL http://unicode.org/reports/tr51/index.html

▼図2　SOとSIによる文字集合の切り替え



JIS X 0201

ISO/IEC 646に基づき、英数字に加えてカタカナを扱えるよう拡張した日本版が、1969年に誕生した「JIS X 0201」です。その名のとおり、JIS（日本工業規格）として規格化されています。誕生した当時はJIS C 6220と呼ばれていましたが、部門Cの「電子機器及び電気機械」から部門Xの「情報処理」に移され、現在の名前になりました。正式名称は「7ビット及び8ビットの情報交換用符号化文字集合」と言います。

英数字においては、ASCIIとの違いは2字です。0x5Cの「\（バックスラッシュ）」が「¥（円記号）」に、0x7Eの「~（チルダ）」が「‾（オーバーライン）」に変更されています。カタカナ集合にはカタカナだけでなく、濁点や半濁点、句読点や鉤括弧も収録されています。

7ビット版は制御文字である0x0Eの「SO（SHIFT OUT）」と0x0Fの「SI（SHIFT IN）」を使って英数字とカタカナを切り替えます。たとえば、通常時では0x33 0x2A 0x32は「3*2」ですが、SOからSIの間に現れた場合には「ｳｪｲ」になります（**図2**）。このように“状態”を持つ文字コードを「ステートフルエンコーディング」と呼びます。

8ビット版は$2^8 = 256$のコードポイントを持ち、英数字とカタカナを一緒に収められるため、SOやSIによる切り替えを必要としません。このように状態を持たない文字コードは「ステートレスエンコーディング」と呼ばれます。データの欠損などで切り替え用の文字が失われた場合でも、文字化けしないのが特徴です。

JIS X 0208

英数字や各種記号、カタカナに加え、ひらがなや漢字を扱えるようにした2バイト文字コードが、1978年に誕生した「JIS X 0208」です。正式名称を「7ビット及び8ビットの2バイト情報交換用符号化漢字集合」といいます。最大で94行×94列＝8,836種類の文字が扱えますが、2012年の第5次規格では6,879文字が収録されています。

漢字については第一水準[注3]および第二水準のものを収録しています。英数字や各種記号も収録していますが、あらゆる文字が2バイトで表現されるため、ASCIIのそれとは互換性がありません[注4]。そのため単独ではあまり使われず、ほかの文字コードと組み合わせて使われることが一般的です。

そのほか、JISで規格化されている文字コードには、補助漢字を扱うための「JIS X 0212」や「JIS X 0208」を拡張し、第三・第四水準漢字などを追加で収録した「JIS X 0213」などがあります。

多種多様な文字を混在させるには

ここまで見てきた文字コードだけでは、「// UserIDを取得する」のように日本語の文字と半角英数字とを同時に使えません。ひらがなや漢字、英数字を混在させるにはどうしたら良いのでしょうか？　大きく分けて2つのアプローチがあります。

(1) あらゆる文字を1つの符号化文字集合に収める
(2) 各種の符号化文字集合を組み合わせて符号化する

注3）JIS X 0208制定にあたり、利用頻度などを鑑みて定められたのが「漢字水準」です。常用漢字を中心に収録した第一水準、部首や旧字体、難しい人名用漢字を収録した第二水準のほか、第三・第四水準や補助漢字などの区分があります。
注4）俗にいう「全角文字」として扱われています。

それぞれのアプローチで代表的なものを見ていきましょう。

Unicode

(1)のアプローチとして最も有名なのが、GoogleやAppleを中心とした非営利団体であるUnicodeコンソーシアムによって策定された「Unicode」です。最初のバージョンであるUnicode1.0は1991年に誕生しました。

Unicodeは世界中の文字を1つ(= Uni)の符号化文字集合に収めようとする規格で、変わったところでは錬金術記号や、未解読であるファイストスの円盤文字なども収録されています。

当初は2バイト(= 65,536)の範囲に世界中の文字を収める計画でしたが、各国からさまざまな文字を追加する要求があったことなどから、その計画は早々に破綻(はたん)してしまいました。現在ではUnicodeといえば4バイト(=約43億)として符号化された文字集合を指します。前者の2バイト文字集合をUCS-2、後者の4バイト文字集合をUCS-4と呼びます。なおUCS-4では、UCS-2にあたる0～65,535までの領域をBMP(Basic Multilingual Plane:基本多言語面)と呼びます。

2010年に公開されたUnicode6.0からは絵文字も収録しています。もともとは各通信キャリア間での「ケータイ絵文字」の相互運用性向上が収録の目的でしたが、近年ではEmoji Modifiersと呼ばれる文字を使って肌の色を変化させるしくみや、ZERO WIDTH JOINERと呼ばれる制御文字を使った合字など、ケータイ絵文字ではできなかった多種多様な表現が可能になっています。

ISO/IEC 2022

(2)の、各種符号化文字集合を切り替えて使うアプローチとしては、「ISO/IEC 2022」があります。これは文字コードそのものではなく、複数の文字コードを切り替えて使うための規格として定められています。ある符号化文字集合がISO/IEC 2022に準拠していれば、準拠しているもの同士は互換性を持つために、この方式で切り替えができます(**図3**)。

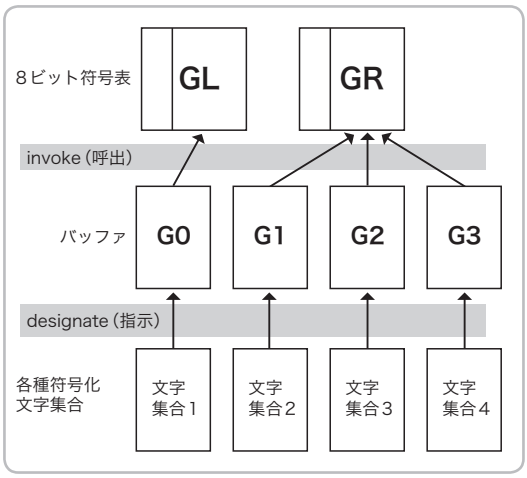
このようなしくみを用いた文字コードは「文字符号化方式」と呼び、符号化文字集合と1対多の関係になります。たとえば、EUC-JPはASCIIとJIS X 0208、JIS X 0201、およびJIS X 0212から成り立っている文字符号化方式です。Unicodeも、1つの文字集合に対してUTF-16やUTF-8など多数の文字符号化方式があります。

EUC-JP

EUCはExtended Unix Codeの略で、アメリカの通信会社AT&Tによって1980年代に策定された文字符号化方式です。名前のとおりUnixで広く使われました。その日本語版が「EUC-JP」です。EUC-JPではASCIIをGL[注5]領域に固定、JIS X 0208(JIS第一／第二水準漢字)、JIS X 0201(半角カナ)、およびJIS X 0212(JIS補助漢字)を切り替えながら文字を表現します(**図4**)。

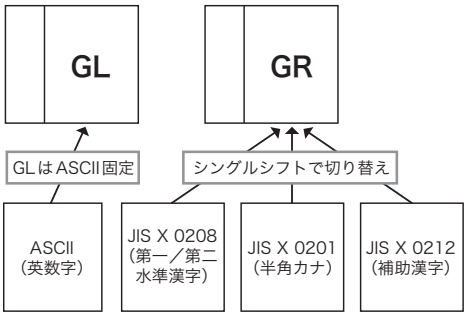
切り替えの際には「シングルシフト」と呼ばれる、後続の1文字だけを切り替え対象とする制

▼図3　ISO/IEC 2022のしくみ



注5) 2つの7ビット符号表を左右に並べると8ビット符号表になりますが、そのうちの左側の図形(Glyph)文字領域をGL領域、右側の図形文字領域をGR領域といいます。

▼図4　EUC-JPの構造



御文字を用います。次の切り替え用制御文字が来るまでモードが固定される「ロッキングシフト」は使いません。そのためステートレスエンコーディングと言えます。

このしくみの長所としては、文字列中の1バイトを見ればそれが1バイト文字なのか2バイト文字の一部なのかが判別できるという点が挙げられます。ASCIIがGL領域に固定されているため、0x7F未満のコードポイントはすべてASCII、それ以上なら2バイト文字、といった具合です（**図5**）。そのため、ソースコード内の文字の一部が意図せずに制御文字として解釈されることがありません。

短所としては、各国版のEUCを混在させられないという点があります。切り替え対象の文字集合は各国版によって異なりますが、どの国のEUCであるかという情報を自身で持たないためです。

Shift_JIS

Windowsでお馴染みなのが「Shift_JIS」です。SJISやMS_Kanjiと呼ばれることもあります。名前に「JIS」と入っていますが、JISではなく株式会社アスキーを中心とした企業体によって考案されました。現在ではJISでも「シフト符号化表現」という名称で規格化されています。JIS X 0201の8ビット符号化方式との互換性があったため、かつては日本語環境でのデファクトスタンダードとも言える存在でしたが、現在ではUTF-8などに取って代わられています。

扱える文字集合はJIS X 0201（半角英数字＋半角カナ）とJIS X 0208（第一・第二水準漢字）です。JIS X 0201をベースに、半角英数字＋半角カナが使っていない範囲にJIS X 0208漢字の1バイト目をずらして（＝シフト）配置した構造になっています（**図6**）。

漢字や日本語の2バイト目として0x7F未満のコードも使われるため、環境によっては意図しない動作を引き起こす可能性があります。たとえば「能」は0x945Cというコードで表されます。この5Cが制御文字である「\（バックスラッシュ：0x5C）」にあたり、ダブルクォートがエスケープされてしまうなどの困った問題が起きることがあります。

ほかにも、あるバイトを見ても1バイト文字なのか2バイト文字の一部なのかがわからない、拡張性が乏しいなどの短所があります。

UTF-16

「UTF-16」はUnicodeの文字符号化方式の1つで、1文字を2バイトで表現します。2バイトではBMPの範囲内にある65,536種類の文字しか扱えないため、範囲外の文字は2バイト文字を2つ組み合わせて表現します。この特別な組み合わせが「サロゲートペア」と呼ばれるものです（**図7**）。

1つの文字でUCS-4を表現できるUTF-32

▼図5　EUC-JPの符号化例

シ		ヤ		乱		Q
0xA5	0xB7	0xA5	0xE3	0xCD	0xF0	0x51

0xA1以上は2バイト文字の1バイト　／　0x7F以下はASCII

▼図6　Shift_JISの符号化例

シ		ヤ		乱		Q
0x83	0x56	0x83	0x83	0x97	0x90	0x51

漢字やひらがなにASCIIと同じ範囲のコードが使われる

と比較すると、サロゲートペアのために扱いがやや複雑になるという短所があります。しかし、頻繁に使われるBMPの文字に関してはUTF-32の半分のバイト数で表現できることから、メモリやストレージの容量を抑えられるという長所があり、プログラミング言語の内部コードとして広く使われています。

UTF-8

Unicodeの文字符号化方式のうち、もっとも広く知られているのが「UTF-8」でしょう。UTF-8は1文字が1〜4バイトで表現される可変長の符号化方式で、1バイト文字についてはASCIIそのものであることが特徴です。ASCIIで書かれたファイルなどをそのままUTF-8としても扱えることから、ASCIIの上位互換として急速に普及しました。また、2〜4バイト文字ではASCIIの範囲である0x00〜0x7Fを使わないため、EUC-JPなどと同じく、文字の一部が意図せずに制御文字として解釈されることがありません(**図8**)。また、2〜4バイト文字の先頭バイトは特定の値であるため、文字の区切りを簡単に見つけられます。

(われわれ日本語ユーザにとっての)短所としては、ひらがなや漢字が3バイトで表現されるため、日本語中心のテキストなどではUTF-16よりも容量が大きくなってしまう点が挙げられます。

Microsoftが拡張した文字コード

これまで見てきた各種文字コード以外に知っておきたいのが、各ベンダによって拡張された文字コードです。AppleやAdobe、富士通などさまざまなベンダが拡張した文字コードがありますが、ここではMicrosoftによって拡張された文字コードである「CP932」と「CP51932」について紹介します。

CP932

もともとMicrosoftは日本語環境での文字コードとしてShift_JISを採用していましたが、IBMやNECといったOEMメーカーに対してJIS X 0208部分の拡張を許していました。そのため各メーカーが、空き領域に独自に符号を割り当てました。これがいわゆる「機種依存文字」と呼ばれるものです。

1993年、MicrosoftはWindows 3.1Jの開発に際し、各メーカーの機種依存文字を整理して再配置し、Shift_JISを拡張しました。これが「CP932」と呼ばれる文字コードです。Shift_JISには含まれない「NEC特殊文字」や「IBM拡張文字」、「NEC選定IBM拡張文字」などを収録しています。

・NEC特殊文字
① ㊤ Ⅳ ㎞ ㍾ ㈱ 髙 Σ
・IBM拡張文字
ⅳ ¦ ㈱ 彁 或 彈 羽 髙

CP51932

CP932のEUC-JPバージョンが「CP51932」です。Internet Explorer 4.0以降や秀丸エディタなどで採用されていますが、CP51932ではなく「日本語(EUC)」と表記されることが多いようです。EUC-JPにはないNEC特殊文字とNEC選定IBM拡張文字が追加されているほか、JIS X 0212補助漢字が削除されています。SD

▼図7 サロゲートペアのしくみ

吉	吉	
0x5409	0xD842	0xDFB7

「吉」はBMP外の文字なので2バイト(16ビット)×2で表現する

▼図8 UTF-8の符号化例

シ			ャ			乱			Q
0xE3	0x82	0xB7	0xE3	0x83	0xA3	0xE4	0xB9	0xB1	0x51

2〜4バイト文字ではASCIIの範囲内のコードを使わない

PART 2

HTMLと文字コード

仕様を理解し、文字を正しく表示する

HTMLで文字コードを扱う際にはいくつかの注意点があります。適切に扱うためには、HTMLの仕様を理解することが大切です。基本を押さえたあとは、CJK統合漢字を扱う方法、指定した文字コードに存在しない文字を表現する方法など、実践的な技術も身につけましょう。

Author 田所 駿佑(たどころ しゅんすけ) 株式会社ビズリーチ スタンバイ事業部 Twitter @todokr

仕様から探る

この章ではHTMLの仕様という観点から文字コードについて見ていきます。

HTMLの仕様は、Web技術の標準化を行う非営利団体である「W3C(World Wide Web Consortium)」や、Apple・Mozilla・Operaを中心としたコミュニティである「WHATWG(Web Hypertext Application Technology Working Group)」によって策定されています。なお、この章の内容はW3Cの「HTML 5.1 Proposed Recommendation」[注1]に準拠します。

文字コードを宣言する

HTMLはHyper Text Markup Languageの略で、インターネット上の文章やWebアプリケーションを表現するためのマークアップ言語です。HTML自体はプレーンテキスト形式で記述されます。仕様では、HTML内で使える文字はUnicodeに収録されているものに限られています[注2]。しかし、これはUnicodeの符号化方式を使わないといけないという意味ではなく、ファイルの文字コードにはShift_JISやEUC-JPなどを使うこともできます。

プレーンテキスト自体はどんな文字コードで符号化されたかを伝えるしくみを持たないので、なんらかの形で文字コードを明示する必要があります。HTML内で明示するには、meta要素のcharset属性、もしくはhttp-equiv属性を使ってリスト1のように指定します。この文字コード宣言はファイルの先頭から1,024バイト以内に記述しないといけません。また、宣言には標準化団体IANA(Internet Assigned Numbers Authority)によって登録された文字コード名[注3]を正確に記載する必要があります。

宣言がないなど、文字コードについて判断する情報が与えられない場合、たいていのブラウザは文字コードの自動判別を試みますが、バイトの並びによっては幾通りもの解釈ができる場合があります。たとえば、EUC-JPで符号化さ

▼リスト1 HTMLでの文字コード宣言

```
<head>
...(略)...
  <!-- HTML 5での形式 -->
  <meta charset="utf-8">
  <!-- HTML 4以下と互換性のある形式-->
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
...(略)...
</head>
```

注1) URL https://www.w3.org/TR/2016/PR-html51-20160915/
注2) 現時点(2016年10月)での最新版であるUnicode 9.0には128,172種類の文字が収録されており、使える文字が限られているといってもまず困ることはないと思います。
注3) 「Character Sets」 URL http://www.iana.org/assignments/character-sets/character-sets.xhtml

れた「文字コード」という文字列は、Shift_JISで符号化された「ﾊｸｻ妹ｳ。ｼ･ﾉ」という文字列であるとも解釈できます。どちらが正しい解釈なのかを判断する材料がこれ以上ないなら、どちらに解釈されてもおかしくないわけです。

文字コードが誤って解釈されてしまった場合は、人が読めないだけでなく、検索エンジンがコンテンツを発見できない事態にもつながります。文字化けを起こしている状態で検索エンジンにインデックスされてしまったら、検索の結果が望みどおりになるはずがありません。

また、セキュリティの問題も無視できません。かつてのInternet Explorerは、文字コード宣言がないHTMLや誤った文字コード名が記載されている場合に、そのHTMLをUTF-7[注A]と解釈していました(**図1**)。その挙動を悪用したXSS(クロスサイト・スクリプティング)攻撃[注4]が引き起こされる可能性などがあります。

▼図1　UTF-7として解釈されるとXSSを引き起こす可能性のある文字列

```
+ADw-script+AD4-alert(+ACI-Hello+ACI-)+ADw-/script+AD4-
```

↓ UTF-7として解釈

```
<script>alert("Hello")</script>
```

なぜUTF-8なのか

W3Cのドキュメントでは、よほど特別な理由がない限りはUTF-8を使用するべき[注5]だとされていますが、なぜでしょうか。そこにはHTMLがインターネットでやりとりされるデータであることに由来する、いくつかの理由があります。順番に見ていきましょう。

まず挙げられるのは、UTF-8があらゆる文字を扱えるUnicode系のエンコードである点です。世界中から閲覧される多言語サイトでも、一種類の文字コードですべてのページを扱うことができれば、HTMLを出力するサーバサイドのロジックもシンプルになります。

圧倒的な普及率による文字化けのしにくさも特筆すべき点です。2012年時点では、Googleにインデックスされているwebページの60%がUTF-8を使用していました[注6]。サブセットであるASCIIも含めると、その普及率は80%ほどになります。

また、英数字が中心であればUTF-16などに比べて容量を削減できるというメリットもあります。2バイト固定長の文字コードと比較すると、ASCIIにあたる文字だけで書かれたHTMLは、単純計算でサイズが半分になります。HTMLはネットワーク越しにやりとりされるという点を考えると、ページが軽く済むというのは大きな利点です。

HTMLでのハマりどころ

宣言と実際の文字コードが異なる

慣れないうちはついやってしまいがちなミスです。charset属性で宣言したとしても、ファイル自体の文字コードが自動的に変化するわけではありません。宣言と同じ文字コードを指定して保存する必要があります。ページ全体が文字化けしてしまった際には、宣言と実際の文字コードが同一かどうかをまず確認してみると良いでしょう。

注A) UTF-7は、かつてUnicodeの規格として存在した7ビットの文字コードです。E-mailなどでの利用を想定し、Base64をもとにした変換方法とシフト文字によって、Unicodeの文字すべてを7ビットで符号化するのが特徴です。

注4) Internet Explorerの挙動を悪用したXSSについてはこの記事に詳しいです。「本当は怖い文字コードの話第1回　UTF-7によるクロスサイトスクリプティング攻撃[前編]」 URL http://gihyo.jp/admin/serial/01/charcode/0001

注5) URL https://www.w3.org/International/getting-started/characters#choosing

注6) URL https://googleblog.blogspot.jp/2012/02/unicode-over-60-percent-of-web.html

BOM付きUTF-8

HTMLファイルを保存する際、文字コードとして普通のUTF-8のほかに、「BOM付き」のUTF-8を選択できるエディタがあります。本来のBOM(Byte Order Mark)は2バイト文字であるUTF-16のバイト並び順を指定するためのもので、目に見える文字としては出力されません(**表1**)。原理的にUTF-8には必要のないものですが、Internet Exploler10/11以外のモダンブラウザはこのバイト列を、「文字コードはUTF-8である」と判断するための材料として使います。

▼表1　UTF-8とUTF-16のBOM

文字コード	BOM
UTF-8	0xEF 0xBB 0xBF
UTF-16(ビッグエンディアン)	0xFE 0xFF
UTF-16(リトルエンディアン)	0xFF 0xFE

※UTF-16のバイトの並び順において、上位8ビットが先頭に来るものをビッグエンディアン、下位8ビットが先頭に来るものをリトルエンディアンと呼びます。

BOM付きUTF-8の場合、HTMLの文字コード宣言を省略することもできますが、ファイルがどの文字コードでエンコーディングされているかがプログラマやデザイナーにとってわかりやすいよう、UTF-8と明示的に宣言することがドキュメントでは勧められています。

また、BOM付きUTF-8はシステムによってはエラーを起こしたり、画面の上部に謎の余白ができることがあるので注意が必要です。

ブラウザの言語によって漢字の字体が変わる

中国、日本、韓国、台湾で字体が微妙に異なるものの、Unicode収録に際して1つに統合された漢字が存在します。この統合された漢字をCJK(China、Japan、Korea)統合漢字と呼びますが、ブラウザによっては言語設定に応じて字体を描画し分ける場合がありますので注意が必要です。たとえば中国語を表記する際は、lang属性でzhと明

Column　Yahoo! Japanと<!-- 京 -->

筆者がHTMLに触れ始めた2000年代後半ごろ、Yahoo! JapanのHTMLには上のほうに<!-- 京 -->というコメントが入っていました(**図A**)。この謎のコメントは文字化け防止のおまじないで、Yahoo! JapanのサイトがEUC-JPを採用していたことに関係があります。

EUC-JPにおいて「京」は0xB5FEと符号化されます。この2バイト目にあたる0xFEはShift_JISやUTF-8の2バイト目には出現しないバイトのため、ブラウザの文字コード判別機能が文字コードを特定しやすくなる、という効果があったようです。いわば、UTF-8におけるBOMのEUC-JP版のような働きをしていたわけです。初めて見るコーダーやプログラマにとっては意味が伝わりづらいですし、検索エンジンによってはコメントの内容までインデックスされる可能性があるため、今ではこのような手法を用いるのはお勧めできません。

なお、現在のYahoo! Japanのページにはこのような記述はありません。また、UTF-8を採用しています。

▼図A　2007年当時のYahoo! Japanトップページのhtml(Wayback Machineより)

```
<html>
▼<head>
    <script type="text/javascript" src="/static/js/
  ▶<script type="text/javascript">…</script>
    <link type="text/css" rel="stylesheet" href="/s
    <meta http-equiv="Content-Type" content="text/h
    <!--京-->
    <title>Yahoo! JAPAN</title>
    <meta name="description" content="日本最大級のポー
    ュニティ、ショッピング、など80以上のサービスを展開。あな
    きます。">
  ▶<style type="text/css" media="all">…</style>
```

示的に一緒に指定すると良いでしょう(**図2**)。

指定した文字コードに存在しない文字を表現するには

サーバサイドの都合などでUTF-8が使えないケースもあるでしょう。しかし、たとえばShift-JISをcharset属性に指定していると、JIS X 0213に含まれる第3水準漢字などが扱えません。文字コードにない文字をHTML内で表現するにはどうしたら良いのでしょうか。

画像を使う

1つ目が、文字の代わりにSVGなどの画像ファイルを使う方法です。<img>要素やCSSのbackground-imageプロパティで文字を画像として表現します。変わったところでは、js-emoji[注7]などの絵文字を扱うためのJavaScriptライブラリが、この手法で絵文字表示のフォールバックを提供しています。

文字参照を使う

2つ目はUnicodeスカラー値を直接表記する方法です。𩸽のように10進数で表記する「10進数文字参照」と、𩸽のように16進数で表記する「16進数文字参照」があります。ある漢字のUnicodeスカラー値を調べる際は、Unicodeコンソーシアムの「Unihan Database Lookup」[注8]というツールを使うと便利です(**図3**)。

また、数値ではなく名前を使った「名前文字参照」[注9]もあります。「<」がHTML要素の一部と認識されないように使う<や、「©」の表記に使う©などは見る機会も多いかと思います。

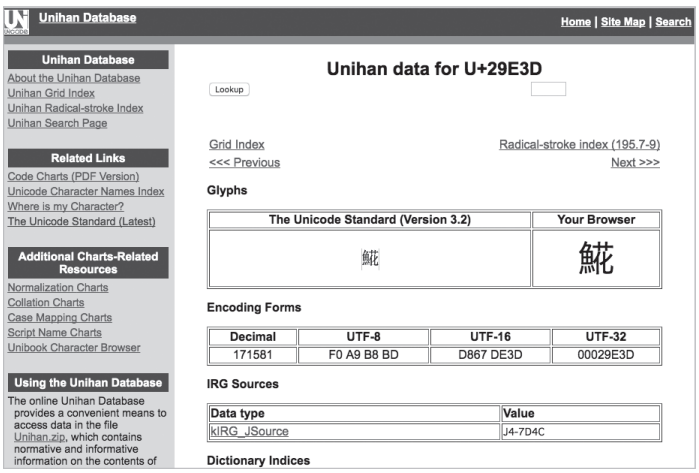
フォントを作成する

最後に挙げるのは独自の符号化文字集合やフォントを作成してディストリビュートする方法です。この方法はいわゆる文字を表現するためではなく、FontAwesome[注10]に代表されるように、アイコンなどを表現するために使われることが多いようです。Webフォントにすればユーザはフォントファイルをダウンロード・インストールする手間がなくなるため、文字集合とフォント作成の手間さえクリアできれば、それなりに現実的な方法ではあります。SD

▼図2　CJK統合漢字とlang属性

lang属性の値	表記
ja	直角
zh（中国語）	直角
zh-tw（中国語-台湾）	直角

▼図3　Unihan Database Lookupで「鰹」を検索したところ



注7）URL https://github.com/iamcal/js-emoji
注8）URL https://www.w3.org/TR/html5/syntax.html#character-references
注9）HTML 4では「実体参照」と呼ばれていました。
注10）URL http://fontawesome.io/

PART 3

Javaと文字コード

char型の落とし穴と文字化け予防策

本章で取り上げるのはJavaにおける文字コード。Javaがどのような文字コードをサポートしているのか、処理系内部ではどのように文字列を扱っているのかを理解したあとは、文字数カウントやファイルの読み書きなど、文字コードにまつわるハマりどころとその回避方法を見ていきましょう。

Author 田所 駿佑(たどころ しゅんすけ) 株式会社ビズリーチ スタンバイ事業部 Twitter @todokr

Javaがサポートする文字コード

Javaがサポートしている文字コードはJDKのドキュメント[注1]に記載されています。記載されているものの内、日本語表現に使われる文字コードを**表1**にまとめてみました。歴史的な経緯により、データの入出力を行うためのAPIであるjava.io、同じく入出力を行うjava.nio、および言語の基本機能を提供するjava.langで文字コードの名称が異なりますが、java.nio APIで使われる名称はIANAに登録されているそれに一致します。

また、Java 7で追加されたjava.nio.charset.StandardCharsetsクラスには、文字コード名を表す文字列が定数として用意されています。文字コード名を文字列で指定する際、たとえば「_(アンダースコア)」を「-(ハイフン)」と間違えるなどのミスが起こりがちですが、これらの定数を使えば安心です。StandardCharsets.UTF_8のように使用します。また、定数になっている文字コードは、"Javaプラットフォームのあらゆる実装で使用できることが保証"されています。

処理系内での文字列の扱い

Javaでは世界中の国でプログラムを扱えるよう、内部処理用の文字コードとしてUnicodeを採用しています。採用しているバージョンはjava.lang.Characterクラスのドキュメントに記載されており、Java 8ではUnicode 6.2.0となっています。

文字列を表すStringクラスやchar配列、およびStringBufferクラスはUTF-16で符号化されます(**図1**)。文字を構成する最小の要素がchar型ですが、「char = 文字」ではないので注意が必要です。詳細は後述します。

Javaのソースコードをコンパイルするコマンドがjavacですが、このjavacの実行時にソースコードの文字コードからUnicodeへの文字コード変換が行われます。この際javacコマンドは、ソースコードをOSのデフォルト文字コードで符号化されているものとして、Unicodeに変換を行います。

▼表1 Java 8でサポートされている代表的な日本語文字コード

java.nio API	java.io、java.lang API	説明
EUC-JP	EUC-JP	JISX 0201, 0208 and 0212, EUC encoding Japanese
ISO-2022-JP	ISO2022JP	JIS X 0201, 0208, in ISO 2022 form, Japanese
Shift_JIS	SJIS	Shift-JIS, Japanese
windows-31j	MS932	Windows Japanese
x-euc-jp-linux	EUC_JP_LINUX	JISX 0201, 0208, EUC encoding Japanese
x-eucJP-Open	EUC_JP_Solaris	JISX 0201, 0208, 0212, EUC encoding Japanese

注1)「Supported Encodings」https://docs.oracle.com/javase/8/docs/technotes/guides/intl/encoding.doc.html

もちろん、デフォルトの文字コードがUTF-8であるMacでShift_JISを使うなど、ソースコードにOSのデフォルト文字コード以外を使用することもできます。使用している文字コードにない文字をソースコード内に使用したいときには、Unicodeスカラー値で表記します。たとえばEUC-JPには収録されていない「𨑨(「長」の中国語簡体字)」は、この字を表すUnicodeスカラー値であるU+957FをUnicodeエスケープシーケンスである\uに続けて、\u957Fと表記します(**リスト1**)。

また、デフォルトの文字コード以外を使用したソースコードをjavacコマンドでコンパイルする際には、-encodingオプションで文字コードを指定します。

```
$ javac -encoding Shift-JIS EncodingTest.java
```

Eclipseではワークスペース単位、プロジェクト単位、コンテンツ・タイプ単位でソースコードに使用する文字コードが設定できます。Mavenでビルドする際には、各種プラグインのconfigurationで文字コードを指定します。詳細は各種ドキュメントなどを確認してください。

Javaでのハマりどころ

サロゲートペアと結合文字列

Javaのchar型に対する一番素朴な理解は「1つのcharが1文字にあたる」ですが、これは正確ではありません。1つのcharでは表せない文字があるからです。

▼図1　文字列をUTF-16で符号化

あ	な	た	と	J	a	v	a
0x3042	0x306A	0x305F	0x3068	0x004A	0x0061	0x0076	0x0061

英数字も16ビットで符号化される

▼リスト1　Unicodeエスケープシーケンスの使用例

```
System.out.println("「𨑨」の中国語簡体字は「¥u957F」です。");
// => 「𨑨」の中国語簡体字は「长」です。
```

先に述べたように、Javaは文字をUTF-16で処理しています。その16ビットにあたるのがcharです。16ビットで表現できるのは65,536種類の文字、すなわちUnicodeの第一面であるBMPの範囲内に存在する文字しか表現できません。BMPの範囲外にある文字は、16ビットのcharを2つ組み合わせたサロゲートペアで表現する必要があります[注2]。Unicodeには3.1からサロゲートペアが導入され、Javaでは1.5から対応しています。Java 1.4と1.5のドキュメントでString#charAtメソッドの説明を比較すると、前者ではcharが「文字」とされているのに対し、後者では「char値」に変わっていることが確認できます。

またUnicodeには結合文字列と呼ばれるしくみがあり、これが文字の扱いをさらにややこしくしています。身近なところでは、Macで作成したフォルダ名の初期値である「名称未設定フォルダ」の「ダ」が、「タ(U+30BF)」と「゙(U+3099)」の結合文字列です(**図2**)。サロゲートペアがcharのペアで1つのコードポイントを表現していたのに対し、こちらは2つのcharがそれぞれ1つのコードポイントを表現しています。

サロゲートペアや結合文字列の存在によって問題になりやすいのが、文字数カウントといったchar単位での処理です。たとえば、サロゲートペアで表現する「鰹(U+29E49 とびうお)」と、

注2) BMP、サロゲートペアについてはPart1を参照。

▼図2　結合文字列のしくみ

名	称	未	設	定	フ	ォ	ル	タ	゙
0x540D	0x79F0	0x672A	0x8A2D	0x5B9A	0x30D5	0x30a9	0x30eb	0x30BF	0x3099

0x30BFと0x3099で「ダ」を表す

「か」と「゙」による結合文字列「が」を含んだ「鰄が釣れた」というテキストの文字数をカウントするとします。

一番シンプルな方法はString.length()でカウントする方法です(**リスト2の(1)**)。この方法はcharの個数をカウントしているだけですので、サロゲートペアである「鰄」と結合文字列である「が」がそれぞれ2文字とカウントされるため、結果は7文字になります。

Unicodeコードポイントの個数をカウントするString.codePointCount()を使うと、「鰄」は1文字になりますが、「が」はあくまでも結合文字列ですので依然として2文字のままです。その結果、6文字となります(**リスト2の(2)**)。

結合文字列を1文字とカウントするには、java.text.Normalizerを使って文字列を正規化したうえでカウントする方法があります(**リスト2の(3)**)。この例では「か」と「゙」による結合文字列が合成済みの「が」になるので、(2)の方法と組み合わせてカウントすると5文字になり

▼リスト2　さまざまな文字数カウントの方法

```
String s = "鰄か\u3099釣れた"; // 見た目上は「鰄が釣れた」の5文字

// (1) String.length()でカウントする方法
// シンプルだがサロゲートペアや結合文字列は2字としてカウントされる
System.out.println(s.length()); // => 7

// (2) String.codePointCount()でコードポイントの数をカウントする方法
// 「鰄」は1字とカウントできるようになったが、結合文字列は2字のまま
System.out.println(s.codePointCount(0, s.length())); // => 6

//(3) java.text.Normalizerで結合文字列をUnicode正規化したうえでカウントする方法
// 「か\u3099」を「が」に正規化したうえでコードポイントをカウントする
String normalized = Normalizer.normalize(s, Normalizer.Form.NFC);
System.out.println(normalized.codePointCount(0, normalized.length())); // => 5

// (4) java.text.BreakIteratorを使う方法
// 文字分割用のイテレータを使い、適切な位置で文字を区切ってカウントする
BreakIterator bi = BreakIterator.getCharacterInstance(Locale.JAPANESE);
bi.setText(s);
int count = 0;
while (bi.next() != BreakIterator.DONE) {
    count++;
}
System.out.println(count); // => 5
```

▼リスト3　FileReaderとInputStreamReader

```
int ch;
File file = new File("./hello.txt");

// (1) FileReaderはファイルがデフォルト文字コードで符号化されているものとして読み込むため、
// ファイルの文字コードと実行環境のデフォルト文字コードが異なると文字化けする
FileReader fr = new FileReader(file);
while ((ch = fr.read()) != -1) {
    System.out.println((char)ch);
}
fr.close();

// (2) InputStreamReaderで文字コードを明示的に指定し、
// デフォルト文字コードに依存しないようにする
InputStream is = new FileInputStream(file);
InputStreamReader isr = new InputStreamReader(is, "Shift_JIS");
while ((ch = isr.read()) != -1) {
    System.out.println((char)ch);
}
isr.close();
```

ます。

またjava.text.BreakIteratorを使い、自然な文字の区切り位置でカウントする方法もあります(**リスト2の(4)**)。

サロゲートペアや結合文字列を1文字としてカウントするのかという点に関しては、システムの要件や仕様によって判断が分かれるところですので、どの手法が最善かは結論づけられませんが、“人が素朴に認識している文字数とUnicodeでの文字数の概念には隔たりがある”ということは、認識しておくと良いでしょう。

Unicodeへの変換に際した文字化け

先に述べたように、Javaは内部コードとしてUnicodeを使うため、ファイルの読み書きやネットワーク経由でのやりとりの際にデータの文字コードとUnicodeとの間で変換が行われます。この変換の際に文字化けが起きることがあります。

Javaにはファイルを読み書きする簡易なAPIとしてjava.io.FileReader/FileWriterクラスが用意されていますが、読み書きする際の文字コードは実行環境のデフォルト文字コードに依存します(**リスト3の(1)**)。そのため、Windowsでは正しく読み書きできていた処理をLinux上で動かすとファイルが文字化けする、などの不具合が起きる場合があります。java.io.InputStreamReader/InputStreamWriterクラスを使い、文字コードを明示的に宣言するとこのような問題を防ぐことができます(**リスト3の(2)**)。

Unicodeに変換されるシーンとしてはほかにも、

- リクエスト時に送信された文字列(クエリパラメータ、リクエストボディなど)の文字コードからUnicodeへの変換
- レスポンス時のUnicodeからレスポンス用の文字コードへの変換
- データベースへのアクセス時のデータベースの文字コードとUnicode間の文字コード変換

などがあります。これらの境界で適切に変換が行われないと文字化けにつながります。

文字化けが起きた際には、どの境界で起きたのかをまず把握し、変換が行われる処理や各種ミドルウェア、フレームワークの設定を見直してみましょう。SD

PART 4

Rubyと文字コード

プログラム中での異なるエンコーディングの扱い方

Rubyプログラムの中では、同時に異なる文字コードの文字列オブジェクトが存在できます。本PARTでは、スクリプトそのもののエンコーディング、ファイル入出力時のエンコーディング、エンコーディングの変換、エンコーディングが混在するときの処理などを解説します。また、不正なバイト列が混入した場合の処理も紹介します。

Author とみたまさひろ 日本Rubyの会 Twitter @tmtms

はじめに

Rubyでは文字エンコーディングのことを単にエンコーディングと呼びます。Rubyの文字列オブジェクトのエンコーディングは1つに決まっているわけではありません。プログラム実行時にどれか1つに決まるというわけでもありません。Rubyの中で生成される文字列オブジェクトごとにエンコーディングを持っています。つまり実行中の1つのRubyプログラムの中で、異なる文字コードの文字列オブジェクトが同時に存在し得るということです。

スクリプトエンコーディング

Rubyプログラム中に直接文字列を記述したら、そのエンコーディングは何になるのでしょうか。文字列オブジェクトのエンコーディングは、encodingメソッドで知ることができます。試してみましょう。

```
% cat test.rb
p "abc".encoding

% ruby test.rb
#<Encoding:UTF-8>
```

これでエンコーディングがUTF-8であることがわかりました。

プログラムファイル(スクリプト)を解釈するときに使用されるエンコーディングを、スクリプトエンコーディングと言います。文字列だけではなく、正規表現、シンボル、変数名などもスクリプトエンコーディングによって解釈されます。

先ほどの例ではString#encodingを使用して調べましたが、スクリプトエンコーディングは簡単に__ENCODING__で取得できます。スクリプトエンコーディングのデフォルトはUTF-8です。現在ではスクリプトエンコーディングがUTF-8であることで困ることはほとんどないと思いますが、変更することもできます。

スクリプトの先頭に「# coding:エンコーディング名」を記述することで、そのファイル全体のスクリプトエンコーディングを指定できます。次の例ではエンコーディングにCP932(Windows-31J)[注1]を指定した例です。

```
% cat cp932.rb
# coding: cp932
p __ENCODING__

% ruby cp932.rb
#<Encoding:Windows-31J>
```

エンコーディングが異なる複数のスクリプトを、1つのプログラムで使用することもできます。その場合は、異なるエンコーディングの文字列が存在することになります。次の例は、スクリプトエンコーディングがそれぞれUTF-8とCP932のスクリプトを1つのプログラムが読み込んだ場合、各スクリプト中で定義された文字列がそれぞれのエンコーディングになっていることを示しています。

注1) 詳しくはPart1の「Microsoftが拡張した文字コード」の節を参照してください。

```
% cat utf-8.rb
# coding: utf-8
$str_utf8 = "あいうえお"

% cat cp932.rb
# coding: cp932
$str_cp932 = "あいうえお"

% ruby -r./utf-8 -r./cp932 -e 'p $str_utf8.encoding, $str_cp932.encoding'
#<Encoding:UTF-8>
#<Encoding:Windows-31J>
```

ASCII互換エンコーディング

Rubyが持つエンコーディングの一覧はEncoding.listで見ることができます。Ruby 2.3.1では101個のエンコーディングがあります。

ですが、このすべてのエンコーディングをスクリプトエンコーディングに指定できるわけではありません。指定できるのは、ASCII互換エンコーディングのみです。ASCII互換エンコーディングとは、00から7Fの範囲の文字がASCIIと同じものです。ASCII互換エンコーディングはEncoding#ascii_compatible?で調べることができます。Ruby 2.3.1では88個あります。

ファイル入出力

通常、テキストファイルはファイル自身にエンコーディング情報を持っていないため、プログラム側が読み書き時にファイルのエンコーディングを意識する必要があります。

RubyではIOオブジェクトが外部エンコーディングと内部エンコーディングを持っていて、それに従って読み書きが行われます。外部エンコーディングはファイルの内容が何のエンコーディングで書かれているかを示し、内部エンコーディングは読み込んだ文字列をプログラム中で何のエンコーディングとして扱うかを示します。

読み込み時のエンコーディング

ファイルから文字列を読み込むときの外部エンコーディングは、ファイルのオープン時にオープンモード文字列で指定します。"r:cp932"が外部エンコーディングの指定部分です。

```
f = File.open("cp932.txt", "r:cp932")
s = f.gets  #=> CP932文字列
```

さらに続けて内部エンコーディングを指定し、読み込んだ文字列を何のエンコーディングとして扱いたいかを指定することもできます。これを指定するとファイルを読み込む際にエンコーディングが変換された文字列が返ります。次の例では、"r:cp932:utf-8"の指定により、ファイルからの文字列読み込み時にCP932からUTF-8にエンコーディングを変換しています。

```
f = File.open("cp932.txt", "r:cp932:utf-8")
s = f.gets  #=> UTF-8文字列
```

オープン後にIO#set_encodingでエンコーディングを設定することもできます。

```
f = File.open("cp932.txt", "r")
f.set_encoding("cp932")
s = f.gets  #=> CP932文字列
f.set_encoding("cp932", "utf-8")
s = f.gets  #=> UTF-8文字列
```

外部エンコーディングを指定しない場合はEncoding.default_externalに従います。

Encoding.default_externalはシステムのロケールによって決定します。Linuxの場合は、LC_ALL、LC_CTYPE、LANG環境変数の値によって決定します。

```
% LC_ALL=ja_JP.UTF-8 ruby -e 'p Encoding.default_external'
#<Encoding:UTF-8>
% LC_ALL=ja_JP.EUC-JP ruby -e 'p Encoding.default_external'
#<Encoding:EUC-JP>
% LC_ALL=C ruby -e 'p Encoding.default_external'
#<Encoding:US-ASCII>
```

▼表1　外部・内部エンコーディング、Encoding.default_internalの値と、読み書きの変換

外部エンコーディング	内部エンコーディング	default_internal	読み込み	書き出し
未指定	未指定	nil	default_externalとして読み込み、変換しない	変換しない
未指定	未指定	指定	default_externalからdefault_internalに変換	default_externalに変換
指定	未指定	nil	外部エンコーディングとして読み込み、変換しない	外部エンコーディングに変換
指定	未指定	指定	外部エンコーディングからdefault_internalに変換	外部エンコーディングに変換
指定	指定	nil	外部エンコーディングから内部エンコーディングに変換	外部エンコーディングに変換
指定	指定	指定	外部エンコーディングから内部エンコーディングに変換	外部エンコーディングに変換

環境変数の値によって読み込むファイルのエンコーディングが変わってしまうことが望ましくない場合は、プログラムの最初のほうでEncoding.default_externalを設定しておくのがよいでしょう。

内部エンコーディングを指定していない場合はEncoding.default_internalの値が使用されます。この値のデフォルト値はnilですので、値を設定しなければ変換は行われません。

書き出し時のエンコーディング

文字列をファイルに書き出す際には、その文字列のエンコーディングから外部エンコーディングに変換されて書き出されます。文字列は自身のエンコーディングを知っているため、内部エンコーディングは書き出し時には影響しません。

外部エンコーディングを指定していない場合は、文字列オブジェクトのエンコーディングのまま変換されずに書き出されます。ただし、Encoding.default_internalが設定されている場合は、外部エンコーディングを指定していなくてもEncoding.default_externalに変換されて書き出されます。Encoding.default_internalの値は設定されているかどうかだけ参照されて、変換には使用されません。

外部・内部エンコーディング、Encoding.default_internalの値と、読み書きの変換をまとめると**表1**のようになります(Encoding.default_externalは常に値が設定されています)。

バイナリデータ

ファイルがテキストデータではなくバイナリデータの場合、読み書き時にエンコーディングが変換されてしまっては困ります。

Rubyではバイナリデータも文字列オブジェクトとして扱います。バイナリデータを扱うためのエンコーディングはASCII-8BITです。BINARYという別名も持っています。

I/Oの外部エンコーディングとしてASCII-8BITを指定すると、Encoding.default_internalの設定にかかわらず常にASCII-8BITエンコーディング文字列として読み込まれます。書き出し時には文字列のエンコーディングにかかわらず無変換でI/Oに書き出されます。

また、メソッドによってもエンコーディングの影響を受けないものがあります。たとえば、IO#getsは改行まで読み込むというメソッドですので外部エンコーディングや内部エンコーディングで指定したエンコーディング文字列を返しますが、IO#read(size)メソッドは、指定したバイトサイズのデータを読み込むためエンコーディングの指定にかかわらずASCII-8BITエンコーディング文字列を返します。なお、サイズを指定しないIO#readメソッドはエンコーディングの影響を受けます。

▼リスト1　UTF-8の「日本語」をCP932に変換

```
"日本語".encode("cp932")
#=> "\x{93FA}\x{967B}\x{8CEA}" (CP932で「日本語」)
```

▼リスト2　変換先のエンコーディングにない文字が含まれている場合と、元の文字列に不正なバイト列が含まれている場合

```
"♡".encode("cp932")
#=> `encode': U+2661 from UTF-8 to Windows-31J (Encoding::UndefinedConversionError)
"\xFF".encode("CP932")
#=> encode': "\xFF" on UTF-8 (Encoding::InvalidByteSequenceError)
```

▼リスト3　:undef、:invalidによるString#encodeのエラー制御の例

```
"いろは\xFF日本語♡".encode("cp932", undef: :replace, invalid: :replace)
#=> "\x{82A2}\x{82EB}\x{82CD}?\x{93FA}\x{967B}\x{8CEA}?" (CP932で「いろは?日本語?」)
```

▼リスト4　異なるエンコーディングの文字列同士の処理の例

```
utf8 = "いろは".encode("utf-8")
cp932 = "いろは".encode("cp932")
utf8 == cp932
#=> false
utf8 + cp932
#=> incompatible character encodings: UTF-8 and Windows-31J (Encoding::CompatibilityError)
utf8.include? cp932
#=> incompatible character encodings: UTF-8 and Windows-31J (Encoding::CompatibilityError)
```

変換

String#encodeメソッドは、文字列を別のエンコーディングに変換した新しい文字列オブジェクトを返します。**リスト1**はUTF-8の「日本語」をCP932に変換した例です。

文字列のバイト列をASCII-8BITとして取り出すことはよくあるため、短いString#bというメソッドも用意されています。

エンコーディングによって文字集合が異なるため、元の文字列にあった文字が変換先のエンコーディングに存在しない場合もあります。また、元の文字列に文字としては不正なバイト列が含まれている場合もあります。そのような文字を含む場合はエラーが発生します(**リスト2**)。

String#encodeメソッド時に:undef、:invalidオプションを渡すことでこれらのエラーを制御できます(**リスト3**)。

String#force_encodingメソッドは、エンコーディングを変換するのではなく、文字列のバイト列はそのままでエンコーディング情報だけを変更します。これはencodeとは異なり、新しい文字列オブジェクトを返すのではなく文字列オブジェクトを変更する破壊的メソッドです。

異なるエンコーディング同士の処理

今まで説明してきたように、Rubyは1つのプログラム中で複数のエンコーディングの文字列オブジェクトを持つことができます。異なるエンコーディングの文字列同士の処理がどうなるか見てみましょう(**リスト4**)。

エンコーディングが異なる場合は同じ文字列であっても==での単純な比較はfalseになります。

異なるエンコーディングの文字列結合や文字列検査などの処理はEncoding::CompatibilityError例外が発生します。

複数の文字列に対する処理を行う場合は、事前にエンコーディングを合わせておかないと思わぬエラーの原因になることもあるので注意しましょう。

正規表現も文字列と同じようにエンコーディングを持ちます。文字列と正規表現の比較の場合もエンコーディングを合わせておかないとエラーになるので注意しましょう。

なお、ASCII互換のエンコーディングで、ASCII文字だけで構成されている文字列や正規表現は、エンコーディングが異なってもエラーになりません。

不正なバイト列

文字列中に文字として不正なバイト列が混入していると、正規表現との比較でエラーになることがあります。不正なバイト列が混入するのは**図1**のような場合です。

文字列がエンコーディングとして正しいバイト列で構成されているかはString#valid_encoding?メソッドを使用して調べられます。

```
"あいうえお".valid_encoding?      #=> true
"あいう\xFFえお".valid_encoding? #=> false
```

また、String#scrubメソッドを使用して文字列中の不正なバイト列を置換することができます。

```
"あいう\xFFえお".scrub        #=> "あいう◆えお"
"あいう\xFFえお".scrub("?")   #=> "あいう?えお"
```

信頼できないファイルなどから読み込んだ文字列を安全に処理するには、これらのメソッドを使用したほうが良いでしょう。SD

▼図1　文字列として不正なバイト列が混入する例

・スクリプト中のリテラル文字列に直接文字コードを指定した場合

```
str = "あいうえお\xFFかきくけこ"
# 0xFF は UTF-8 には存在しないバイト
str =~ /./
#=> invalid byte sequence in UTF-8 (ArgumentError)
```

・内部エンコーディングを指定せずにファイルから不正なバイト列を読み込んだ場合

```
str = File.open("/dev/urandom").gets
# 変換処理が行われないのでここではエラーにならない
str =~ /./
#=> invalid byte sequence in UTF-8 (ArgumentError)
```

・encodeやforce_encodingで誤ったエンコーディングを指定した場合

```
str = "あいうえお".force_encoding("cp932")
# UTF-8文字列のバイト列をそのままにCP932エンコーディング化
str =~ /./
#=> invalid byte sequence in Windows-31J (ArgumentError)
```

PART 5

MySQLと文字コード

charsetでの文字集合の指定方法とエンコーディングの対応

本稿では、MySQLで文字コードを扱う際に指定するcharsetについて、エンコーディングとの対応と、クライアント／サーバ接続時の問題について解説します。また、MySQL 5.6での文字処理時のエラー対応方法についても紹介します。

Author とみたまさひろ 日本MySQLユーザ会 Twitter @tmtms

charset

MySQLの文字コードはcharacter setまたはcharsetという単語で表されます。MySQLの構文としてもcharacter setとcharsetは同等に扱われます。以降、本稿ではcharsetを使用します。日本語が使用できるcharsetはutf8、utf8mb4、cp932、sjis、eucjpms、ujisの6つです。

それぞれ、どのような文字集合とエンコーディングに対応しているのかを**表1**に示します。

今から新しくMySQLを開始するのであれば、utf8mb4を使用すべきです。Unicodeは現在コンピュータ上で扱うことのできる世界中のすべての文字を含んでいますが、Windows-31JやJISは日本語と、日本でよく使用されることのある一部のラテン文字しか含んでいません。

またutf8 charsetはUnicodeのうちU+10000～U+1FFFFFの文字を含んでいません。この範囲には一部の漢字も含まれますし、多くの絵文字も含まれます。

漢字のほうは「丈」「丄」「堅」などあまり使用されることのない文字ですが、今や「🍆」「🍺」「😈」「💕」などの絵文字は普通に入力され得る文字です。これらの文字に対応できない方が影響が大きいかもしれません。

utf8mb4の名前は、Unicodeの U+10000～U+1FFFFFの文字が、UTF-8で表現すると1文字が4byteになることに由来しています（mb4 = multibyte 4）。

使用できるcharsetの一覧はSHOW CHARSETクエリで表示できます。MySQL 5.7.16では41個のcharsetがあります。

▼表1 MySQLのcharsetの文字集合とエンコーディングの対応

charset	文字集合	エンコーディング
utf8mb4	Unicode	UTF-8
utf8	Unicode（U+0000～U+FFFFの範囲のみ）	UTF-8
cp932	Windows-31J	CP932
sjis	JIS	SHIFT_JIS
eucjpms	Windows-31J	CP51932
ujis	JIS	EUC-JP

charsetの指定

MySQLでは、サーバ、データベース、テーブル、カラムごとにそれぞれ異なるcharsetを指定できます。

指定できるといっても、異なるcharsetを指定してもトラブルの元になるだけです。実際に使用するcharsetはできるだけ1つに統一しておくべきです。

MySQLのデフォルトのcharsetはlatin1（ISO 8859-1）です。何も指定しなければlatin1が使われるのですが、一見日本語も使用できるような挙動に見えるため、あとあとトラブルになったりすることもあります。できれば最初に設定しておくのがよいでしょう。

サーバcharset

サーバcharsetは、新規にデータベースを作成するときのデフォルトのcharsetです。実はこれさえ指定しておけばだいたい問題ありませ

ん。このサーバが扱うすべてのデータベース／テーブル／カラムがただ1つのcharsetだけを使用するのであれば、これを指定しておくだけで十分です。

/etc/my.cnfなどの設定ファイルにcharacter-set-serverを指定します[注1]。

```
[mysqld]
character-set-server = utf8mb4
```

データベースcharset

データベースcharsetは、このデータベース配下に作成するテーブルのデフォルトのcharsetです。CREATE DABASEでデータベースを作成するときにCHARSETを指定します。何も指定しなければサーバcharsetが使用されます。

```
mysql> CREATE DATABASE dbname CHARSET
utf8mb4;
```

データベースcharsetを変更するにはALTER DATABASEを使用します。

```
mysql> ALTER DATABASE dbname CHARSET utf8mb4;
```

なお、データベースcharsetを変更しても、すでに作成されているテーブルには影響しません。

テーブル／カラムのcharset

テーブルのcharsetは、CREATE TABLEでテーブルを作成するときにCHARSETを指定します(**図1**)。何も指定しなければデータベースcharsetが使用されます。

カラムのcharsetは、何も指定しなければテーブルのcharsetと同じになります。普通はしないと思いますが、カラムの定義にCHARSETを指定すれば、テーブルのcharsetと異なるcharsetのカラムを作成することもできます(**図2**)。

テーブルのcharsetを変更するには、ALTER TABLEを使用します。テーブルのcharsetを変更してもカラムのcharsetや格納されているデータには影響しません。

```
mysql> ALTER TABLE tblname CHARSET cp932;
```

カラムのcharsetを変更するのもALTER TABLEです。カラムのcharsetを変更すると格納されている文字列も新しいcharsetのデータに変換されます(**図3**)。

クライアント／サーバ接続のcharset

MySQLのcharsetまわりで問題が発生しやすいのが、クライアント／サーバ接続のcharsetです。接続のcharsetはクライアントから指定します。特別な事情がなければテーブル／カラムに設定しているcharsetと同じものを指定すべきです。

接続のcharsetはクライアントごとに設定方法が異なりますが、MySQLに付属のコマンドであればたいていはdefault-character-setというパラメータで指定できます[注2]。my.cnfに次

▼図1　CREATE TABLEでテーブル作成時にCHARSETを指定する

```
mysql> CREATE TABLE tblname (col VARCHAR(10)) CHARSET utf8mb4;
```

▼図2　カラム定義にCHARSETを指定し、テーブルと異なるcharsetを指定する例

```
mysql> CREATE TABLE tblname (col1 VARCHAR(10), col2 VARCHAR(10) CHARSET cp932) CHARSET utf8mb4;
```

注1）mysqldがどの設定ファイルを読み込むかは環境によって異なります。設定ファイルのファイル名はmysqld --help -vコマンドの出力の「Default options are read from the following files in the given order:」の行で知ることができます。

注2）すべてのクライアントコマンドがdefault-character-setパラメータを解釈できるわけではないため、単にdefault-character-setを指定しただけだとコマンドによってはエラーになることがあります。loose-をパラメータ名の先頭につけることで、そのパラメータを解釈できない場合はエラーの代わりに無視するようになります。

▼図3　ALTER TABLEでカラムのcharsetを変更する

```
mysql> SELECT col2,HEX(col2) FROM tblname;
+-----------+--------------+
| col2      | hex(col2)    |
+-----------+--------------+
| いろは    | 82A282EB82CD |
+-----------+--------------+
mysql> ALTER TABLE tblname MODIFY col2 VARCHAR(10) CHARSET utf8mb4;
mysql> SELECT col2,HEX(col2) FROM tblname;
+-----------+--------------------+
| col2      | hex(col2)          |
+-----------+--------------------+
| いろは    | E38184E3828DE381AF |
+-----------+--------------------+
```

▼図4　COLLATEでテーブル／カラムにcollationを設定する

```
mysql> CREATE TABLE tblname (col VARCHAR(10)) CHARSET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

のように設定しておけばよいでしょう。

```
[client]
loose-default-character-set = utf8mb4
```

MySQL付属ではない独自に作成したアプリケーションで接続のcharsetを指定する方法は、そのアプリケーションしだいです。charsetを指定する方法がなく、デフォルトのlatin1のまま使用するアプリケーションもあるかもしれません。

そのようなアプリケーションでは、サーバ側でskip-character-set-client-handshakeパラメータを設定することで解決できる場合があります。このパラメータを指定すると、クライアントが指定したcharsetを無視して、character-set-serverパラメータの値を接続のcharsetとして扱います。ただし、これはすべてのクライアントに影響するため、設定する場合は注意が必要です。

collation

collationは照合順序／照合規則と訳されます。文字をソートしたときの順序や、文字を比較したときに等しいとみなすかどうかの規則を定めたものです。charsetごとにcollationがあり、テーブル／カラムでcharsetを指定するということは、そのcharsetのデフォルトのcollationを指定しているのと同じ意味になります。

テーブル／カラムにcollationを設定するには、COLLATEで指定します(**図4**)。

MySQL 5.7.16には、全部で222個のcollationがあります。そのうちutf8mb4のcollationは26個です。さらにその中で日本語で使用されることのあるutf8mb4_general_ci、utf8mb4_bin、utf8mb4_unicode_ci、utf8mb4_unicode_520_ciについて説明します。

utf8mb4_general_ci

utf8mb4charsetのデフォルトのcollationです。末尾のciはcase insensitiveの意味で大文字小文字を区別しないということを示しています。比較時に英字の大文字小文字は区別されません。全角文字も大文字小文字は区別されません。「Å」「Ô」なども「A」「O」と同様に扱われます。U+10000～U+1FFFFFの文字はすべて区別されません。「丈」「土」「  」「  」などは同じ文字として扱われます。「寿司ビール問題」として有名です。

utf8mb4_bin

すべての文字が区別されるcollationです。文字列型カラムにBINARY属性を指定したときに、このcollationが使用されます。

utf8mb4_unicode_ci

UCA(Unicode Collation Algorithm)4.0.0準拠のcollationです。

ciとついているように大文字小文字を区別しません。utf8mb4_general_ciとの主な違いは、かな文字のひらがな／カタカナ／全角／半角／濁音／半濁音を区別しないことです。つまり「は」「ば」「ぱ」「ハ」「パ」「バ」「ﾊ」を区別しません(「ハハ＝パパ問題」)。

utf8mb4_unicode_520_ci

UCA 5.2.0準拠のcollationです。utf8mb4_unicode_ciとの主な違いは、U+10000～U+1FFFFFの文字が区別されることです。つまり「寿司ビール問題」は発生しません。なお「ハハ＝パパ問題」は発生します。

utf8mb4の、各collationでの文字の扱いを**表2**にまとめます。用途に応じて適切なcollationを使用しましょう。

sql_mode

MySQLの最新版は5.7ですが、5.6を使っている人もまだ多いと思います。MySQL 5.6では、文字処理周りでおかしなことが発生しても、デフォルトではエラーにならないことに注意が必要です。

- カラム長を超過した文字列をカラムに入れてもエラーにならず、はみ出した分が削られる
- 不正な文字が入っているとそれ以降の文字がすべて削除される
- 文字変換に失敗した文字が「?」となって格納される

これらはあまり望ましい挙動とは言えず、MySQL 5.7からはエラーになるようになりました。

MySQL 5.6でもエラーになるようにするには、my.cnfでsql-modeパラメータにSTRICT_ALL_TABLESを設定します[注3](**リスト1**)。

すべてのクライアントではなく、特定のクライアントだけエラーにするには、クライアントがサーバに接続した後に**リスト2**のクエリを発行します。SD

▼表2　utf8mb4の各collationでの文字の扱いの違い

collation	A：a	A：A(全角)	🚗：📄	は：ぱ：ば
general_ci	＝	≠	＝	≠
bin	≠	≠	≠	≠
unicode_ci	＝	＝	＝	＝
unicode_520_ci	＝	＝	≠	＝

▼リスト1　MySQL 5.6でもエラーになるようにSTRICT_ALL_TABLESを設定

```
[mysqld]
sql-mode = NO_ENGINE_SUBSTITUTION,STRICT_ALL_TABLES
```

▼リスト2　特定のクライアントだけエラーにする

```
set sql_mode='NO_ENGINE_SUBSTITUTION,STRICT_ALL_TABLES';
```

注3) sql_modeのデフォルトはNO_ENGINE_SUBSTITUTIONなため、それに追加する形で設定しています。

【年末特別企画】

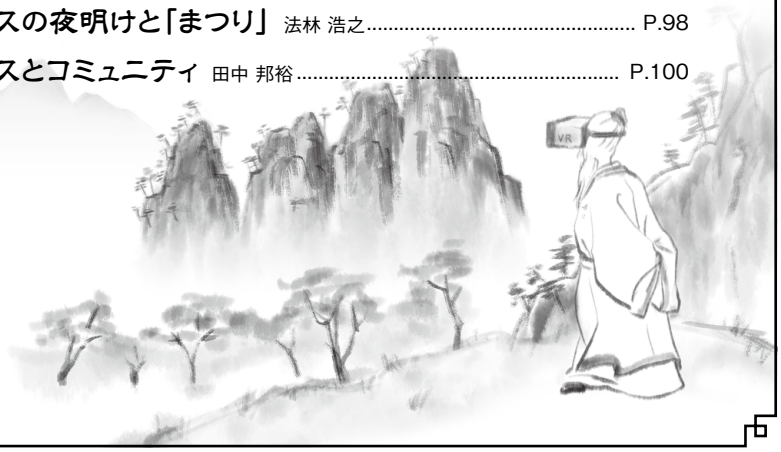
先人たちの知恵と足跡に学ぶ

スペシャル

温故知新 IT むかしばなし

今回は年末特別企画として、いつもは連載の「IT むかしばなし」を特別編でお送りします。日本国内でパソコンという言葉が使われ始めた1980年代、マシン環境は昨今のような高速なスペックではなく、CPUは周波数だけとっても千分の一程度。本体メモリは百分の一程度でした。

そんな時代を過ごされた9人の方に、それぞれの経験を披露していただきました。懐かしく思う方も、想像がつかない方もいらっしゃると思いますが、むかしばなしをお楽しみください。



第1話

パソコンの揺籃（ようらん）期に進化を続けたPC-9800シリーズ

小高 輝真（こだか てるまさ） kodaka@webtech.co.jp twitter @koda256

はじめに

今やすっかりコモディティ化して、国内・国外ともにPCメーカーは淘汰の波にさらされていますが、20～30年前は、まさにパソコン揺籃期ともいうべき時代でした。

その当時、日本のパソコン市場でトップシェアを占めていたNECのPC-9800シリーズ（**写真1**）には、PC-9801VMに代表されるメインストリーム機以外に、多分に実験的要素の強い、ある意味で未来を先取りしたような機種も開発されていました。

本稿では、PC-9800シリーズのうち、突然変異的機種について、振り返ってみます。

ハイレゾモード

1985年に発売されたPC-98XAは、CADなどでの利用を想定したハイレゾモード（画面解像度1,120×750ドット）専用機です。当時のアプリケーションソフト（以下、ソフト）は直接ハードウェアを制御してグラフィックスを描画していたため、ハイレゾモードはノーマルモード（画面解像度640×400ドット）と互換性がなく、PC-98XAでは大多数のPC-9801向けソフトが使用できず不評でした。そのため、後継機種にあたるPC-98XL・XL²・RLとPC-H98では、ハイレゾモードとノーマルモードをシステム起動時に切り替えられるようになりました。

ラップトップ、ハンディ、電子ブックリーダーの先駆け

1986年11月に発売されたPC-98LTは、PC-9800シリーズ初のラップトップ型[注1]です。CPUはV50（V30[注2]に周辺回路を集積したLSI）、重量は3.8kg。当時の技術的限界から、グラフィックスは画面解像度640×400モノクロ2値のVRAMのみ搭載だったため、PC-9800シリーズのノーマルモード用ソフトはそのままでは動作しませんでした。

PC-98LTアーキテクチャの製品は1990年にも発売されます。PC-98HA（Handy98）です（**写真2**）。本体重量は1.1kg、RAMドライブ（フロッピーディスク版SSDのようなもの）やPCMCIA[注3]1.0カードスロットも装備し、パソコンというより、PDAの市場を狙った製品でした。ノーマルモードとの互換性がなかったのは、やはり当時の半導体技術の限界のためです。

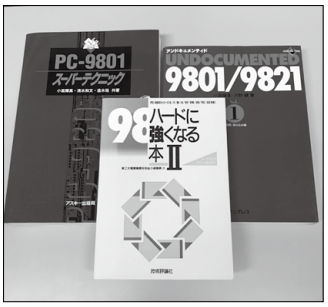
1993年に発売されたデジタルブックプレーヤーDB-P1も、実は

注1） 今のノートパソコンに近いが、ディスプレイと本体が一体型のポータブルパソコンのような意味。膝の上で使えるということで命名された。

注2） V30、V50は、NECによるIntel 8086の上位互換CPU。

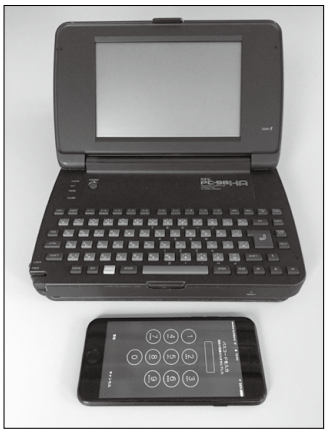
注3） 携帯型パソコン向けのカード型デバイスの規格。メモリやモデムなどを内蔵するためのもの。

▼写真1　筆者が執筆したPC-9800関連の書籍



『98ハードに強くなる本II』（技術評論社 1988）
『PC-9801スーパーテクニック』（アスキー 1992）
『UNDOCUMENTED 9801/9821』（インプレス 1994）

▼写真2　PC-98HAとサイズ比較用に置いたiPhone 7



パソコンの揺籃期に進化を続けたPC-9800シリーズ

PC-98LTアーキテクチャです。画面解像度は320×400ドットと横方向が半分になっています。本体重量は340g。今から20年以上も前に、電子ブックリーダーを発売していたというのは驚きですが、時代の先を行き過ぎていたようです。

タブレットPC

DB-P1と同じく1993年に発売されたPC-9801P(98PEN)は、今のタブレットPCをだいぶ分厚くしたような形状です。ワコム製のペンタブレットユニットを内蔵し、プリインストールOSとして、Pen DOS 2.0、Windows for Pens 1.1、Go社PenPoint 2.0と3種類も用意する意欲的な製品でした。

非公開のPen BIOS(『UNDOCUMENTED 9801/9821』に記載あり)では、ペンの座標位置取得は当然のこと、ペンの傾きまで取得できる機能を持っていたことに驚きました。

日本版MCA/EISAバス

1990年に、PC-H98(HYPER 98)シリーズが発表されました。ハイレゾ・ノーマル両用機で、NESAバスという32bit対応の高速・高機能バス(拡張スロット)を備えた高級機です(**写真3**)。NESA(New Extend Standard Architecture)バスは、PC/AT機におけるMCA(Micro Channel Architecture)バスあるいはEISA(Extended Industry Standard Architecture)バスのような存在ですが、EISAと異なり従来の16bit拡張ボード(Cバス)用コネクタとNESAバスのコネクタは独立していました。

PC-H98は高機能ゆえの高価格が市場で受け入れられず、1992年発売のPC-H98 model105を最後に、シリーズ終了となりました。

NESAバス用の専用コネクタは、1993年発売のMate Aシリーズのローカルバスコネクタに流用されています。

そのMate Aシリーズのローカルバスも、同時期にMate Xシリーズで採用されたPCI(Peripheral Component Interconnect)バスに取って代わられました。PCビジネスでは、価格と互換性の要素が非常に大きいという証左のひとつでしょう。

マルチメディアの先端を行く

PC-98GS(1991年発売)は、PC-9800シリーズで初めてVGA解像度(640×480 1677万色中256色)をサポートした機種です。

当時、非常に高価だったCD-ROMドライブをはじめ、FM(Frequency Modulation)音源+SSG(Software-controlled Sound Generator)音源、画面合成機能など、マルチメディアPCとして考えられる機能を詰め込んだようなスペックを持っていました。Windows 3.0AとMacromedia(2005年にAdobeに買収)Authorware Starというソフトをプリインストールしていて、マルチメディアタイトルのオーサリング機として企画されました。

大ざっぱな言い方をすると、PC-98GSを低価格化したものが初代PC-9821(98MULTi。1992年発売)で、これはその後のPC-9821シリーズの原型と言えます。

▼写真3 PC-H98とNESAバス技術説明書

Intel製PC-98専用CPU

切り口を変えて、特殊なCPUを搭載した機種について触れてみます。

i386SL(i386SXをベースにISAバス用周辺回路などを集積した省電力CPU)をベースに、PC-9800用としてIntelとNECが共同開発したのがi386SL(98)です。搭載機種がPC-9801NS/T(1992年発売)のみで終わってしまったのは、共同開発がスムーズではなく、細かなバグが多かったからだという噂を聞いたことがあります。

翌1993年には、i486SX(J)を搭載したPC-9801NS/Rが発売されました。i486SX(J)は、i486SXの外部データバス16bit版です。i486SX(J)はPC-9801NL/R・Pにも搭載されましたが、他メーカーでは採用されておらず、PC-9800シリーズ専用品だったようです。

Intel CPUではありませんが、1990年に発売されたPC-98Do+は、V30をハードワイヤード化したV33Aを搭載していました。V33AはV30に比べて高速でしたが、主力機種がi386/80286に移行する時期だったため、他の機種で採用されることはありませんでした。SD

第2話

富士通 FM-7とCPU動作周波数
～搭載CPU 68B09(2MHz)はどこまで速いか～

速水 祐(はやみ ゆう)　http://zob.club/　twitter @yyhayami

はじめに

最近、富士通が中国のパソコン大手レノボとパソコン事業統合に向けて調整中であることが報じられました[注1]。レノボといえば、すでにNECとパソコン事業を統合しており、1980年代当初のマイコンのライバル2社が同じメーカーに統合されるとは、当時には夢にも思えないことでした。

今回は、富士通FM-7と、そこに搭載されたCPU 68B09(2MHz)と、そのライバルのNEC PC-8801のCPU Z80A(4MHz)のスピードの違いについてのお話をしましょう。

FM-7とは

1982年、8bitマイコンは、グラフィックが緻密になり画面表示の高機能化が図られたのに対して、それを描く8bit CPUのスピードは従来のままで、ユーザは遅い描画速度に我慢を強いられていました。

1982年当時の8bitマイコンのグラフィック描画は、CPUが直接に表示用メモリ(以下VRAM)にアクセスすることで描いていました。

横640×縦200ドットごとに8色を表示するためのVRAMは48KB必要なので、64KBのメモリエリアしかない8bit CPUでは半分以上のエリアを占有してしまうことになります。FM-8ではサブCPUで、メインCPUのメモリエリアとは別エリアでVRAMのエリアを確保して描画もサブCPUが行うような構成でした[注2]。したがって、サブCPUを高速化すれば、描画がそのまま高速化されることになります。

1982年11月、FM-8[注3]の機能はそのままに、メインCPUもサブCPUも高速な68B09を搭載したFM-7が登場しました。本体価格が126,000円と、FM-8の218,000円に対して10万円近く廉価で、メインCPU、サブCPUともに動作クロック周波数が2MHzと圧倒的な高速化を実現していました。

1982年当時のマイコンは、2月に量産出荷されたNEC PC-8801が、Z80A(4MHz)搭載で228,000円、4月登場の日立ベーシックマスタレベル3 MarkIIは、初代レベル3に対してキーボードの配色が変わっただけでCPUは6809(1MHz)のままで198,000円、そして5月に発売されたシャープMZ-2000は、オプションのグラフィック機能を加えると258,000円(本体218,000＋グラフィックボード32,000円＋グラフィックメモリ32KB 8,000円)でしたので、FM-7はホビーユーザに歓喜の声で迎えられました。

また、PSG[注4]音源も標準で搭載され、ゲームの効果音を奏でる機能追加も好評でした。しかし外見的には、重厚さは陰を潜め、やや小柄で軽い筐体となり、キーボードも軽く安価なものに替えられていました。

FM-7でもFM-8のキーを離した状態を検知できないという問題[注5]はそのまま残り、アクションゲームを遊ぶには適したマシンにはなりませんでした。しかし、当時流行り始めたテキスト文字列を打ち込んでゲームを進めるアドベンチャーゲームにおいては、最適なマシンとして認められ大人気を博しました。

CPUのクロック周波数と動作周波数

FM-7の付属のマニュアルには、

注1) 2016年10月7日現在。

注2) 2016年10月号の本連載参照。

注3) FM-8はメインCPUがMBL68A09(1.2288MHz)、サブCPUがMBL6809(1.008MHz)。

注4) PSG(プログラマブル・サウンド・ジェネレータ)。GI社製のチップで3声同時に出力可能で、ノイズ音発生の機能もあり、爆発音などを出力できる。

注5) 2016年10月号の本連載参照。

富士通 FM-7とCPU動作周波数 ～搭載CPU 68B09(2MHz)はどこまで速いか～

クロック周波数8MHz、CPUの動作周波数2MHzと記述されています。内部を調べてみると16.128MHzの水晶発振器があります。FM-7ではこのクロックを1/2して8.064MHzを68B09に供給しています。68B09は、内部にクロックジェネレータを内蔵していて、外部からのクロック(EXTAL)を1/4にして2.016MHz(周期約0.5μS)のEパルス(信号)と1/4周期先行したQパルスを作成して、周辺チップ、メモリ、バスの同期信号としています(**図1**)。これらのパルスの周波数を動作クロック周波数と呼んでいるのです。

CPUの基本となる動作は、

❶フェッチ(Fetch):読み込み
❷デコード(Decode):解析、解読
❸実行(Execute)
❹ライトバック(Write-back)

の4つの段階で成り立ちます。6809では最初のマシンサイクル[注6]で❶を処理し、次のマシンサイクルで❷❸を同クロック内で実行します。❸の実行は命令コードに続くデータの読み取り処理も含みます。

メモリからの読み取りは、Eパルスの立ち下がり[注7]でデータをリードします。フェッチサイクル❶で命令コード$86を読み取り、次の実行サイクル❷❸で即値[注8]の$05を読み取り、CPU内部でAレジスタ[注9]に代入しています。即値をレジスタに代入する命令は2マシンサイクルで処理可能であり、2M Hzの68B09では、1μ秒で実行できることになります。

注6) CPUから見た1動作が実行できる最低の時間単位。
注7) 信号が1から0になるタイミング。
注8) 機械語内に直接書かれた数値。
注9) CPUが直接計算に使用する専用メモリ。

それに対してZ80は、インテル社80系CPUが外部に各種パルスを作成するクロックジェネレータを使うことが一般的でしたので、それにならい供給する外部クロック周波数とクロックが一致します。

Z80は、最初のマシンサイクルM1でフェッチ処理を行い、ここで動作クロック4周期を使い3周期(T3)の立ち上がりでコードを読み取ります。メモリの読み取りは半周期分フェッチのほうが短くなっており、ここは68B09の2MHzと同じアクセスタイム以下のメモリが必要になることになります。最初のマシンサイクルM1のフェッチ処理で命令を読み込み、次のM2サイクルは動作クロック3周期を要して3周期(T3)の立ち下がりで即値データを読み込み、内部処理でAレジスタにデータを代入しています。

ウェイトがないとすると、即値をレジスタに代入する命令は、68B09が2マシンサイクルで1μ秒、Z80Aが7動作クロックで1.75μ秒かかることがわかります。

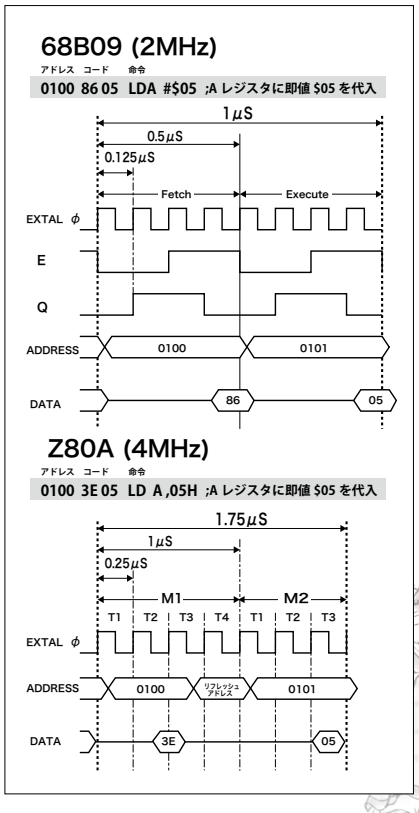
Z80は、データをCPU内部に一時的に格納するレジスタ数が多く、レジスタ間のデータ転送の命令ではM1マシンサイクルのみで処理が完了するため4動作クロック(1μ秒)になり、処理スピードが一気に上がります。それに対して6809では豊富なアドレッシングモード[注10]を駆使したプログラムを組めば、それぞれ高速なプログラム作成につながっていったのです。

注10) 即値のほかに、値が入っている場所を示す方法。

CPUのその後

その後Z80は、1985年10月に発売されたシャープMZ-2500でZ80B 6MHzが搭載され、1986年に発売されたPC-8801FHでは2倍の8MHzクロックのZ80Hで動くようになり、高速化が進んでいきました。しかし、6809の方は1984年に発売された日立S-1でも68B09 2MHzのままであり、遂に2MHzより高い周波数のマシンは現れませんでした。SD

▼図1 68B09とZ80Aのタイミングチャート



第3話

初期のインターネットダイヤルアップ接続とユーザ認証

伊勢 幸一(いせ こういち)　twitter @ibucho

はじめに

「ピーィーヒョロロローヒョロロー、ガァーッガァーッ、ガッ……」

これは何の音なのかおわかりになるでしょうか？　筆者と同世代でIT業界にいた方なら日に何度かの頻度で聞いていたと思います。そうです、この音はTelebit社のTrailblazerというモデムがダイヤルアップしたときのネゴシエーション音です注1。

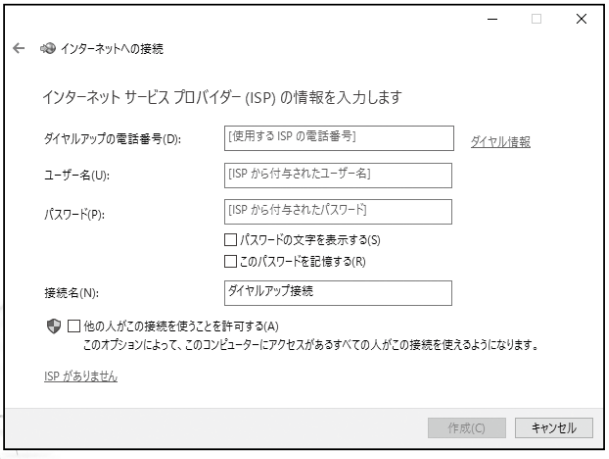
インターネットがまだ一般的でなかったころ、筆者たちはUUCP(Unix to Unix CoPy)というUNIXのシリアル回線を使った通信プログラムによって一日に何度かリモートサイトと送受信し、バケツリレー方式で各サイトへメールやNetNewsを受配信していました。そしてこのモデムを介したアナログ回線ダイヤルアップによる通信接続は、その後のインターネット接続に対しても利用されることになります。

注1）音はメーカーや通信規格、変調速度によって微妙に変わります。

電話回線でつなぐダイヤルアップ

現在、一般的にスマートフォンやタブレット、またラップトップPCであっても電源を入れるだけであらかじめ設定されている携帯通信網や無線LANなどと自動的に接続され、みなさんはさまざまなオンラインサービスを利用していることでしょう。しかし、今から十数年前までは個人がインターネットを利用する前に「インターネットに接続する」という手続きが必要でした。すでにアナログ電話回線とモデムを使うことはほとんどないと思いますが、いまだにMS Windowsのネットワーク接続コントロールパネルでは「ダイヤルアップ」という項目が残っています(**図1**)。

▼図1　Windows10のダイヤルアップ画面



ターミナルサーバ

そしてインターネット接続サービスを提供するISP側にはユーザからのダイヤルアップを受信し、インターネット接続を提供するターミナルサーバという機器が利用されていました。ターミナルサーバとは、もともとLAN上に接続されたUNIXホストマシンに対し、複数のVT端末を接続し、何人かで1台のUNIXマシンを共有するための機器です。このターミナルサーバはLAN接続用のEthernetインターフェースと複数のシリアルポートを持っていました。このVT端末を接続するシリアルポートに

初期のインターネットダイヤルアップ接続とユーザ認証

受信用モデムをつなぎ、リモートからのダイヤルアップを受信して通信路を確立するのです。当時のターミナルサーバにはXylogics社のANNEXシリーズやEMULEXのP4000シリーズなどがありました。

あるユーザがある電話番号にダイヤルアップしてインターネット接続をすると、ほかのユーザは同じ電話番号にダイヤルアップしても通話中で接続できません。そこで当時のISPは複数の電話回線を契約し、同時に複数のユーザに対して接続サービスを提供できるようにしていました[注2]。

ユーザ情報が異なっていたターミナルサーバ

通常ターミナルサーバのシリアルポートは4ポートから8ポートしかなく、一度に提供する接続サービス数を増やすには複数のターミナルサーバを用意する必要があります。ところがこのターミナルサーバによる接続サービスにはユーザ認証の運用に問題がありました。

ISPでは契約したユーザからのダイヤルアップだけに接続サービスを提供する必要があります。そのためターミナルサーバ内に接続を許可するユーザ名とパスワードを設定します。これらユーザ名・パスワードはターミナルサーバ間で共有することができないので、1つの接続拠点内にあるターミナルサーバのすべてに契約ユーザの情報を書き込む必要があります。しかし、その拠点が東京と横浜というように離れた場所にあると、各拠点間でユーザパスワード情報を即時共有することが難しく、拠点ごとに別々のユーザパスワード情報を管理していました。したがって、東京のユーザは東京の拠点の電話番号に接続し、また横浜のユーザは横浜の拠点に接続する必要があります。

RADIUSの登場

そのような接続先拠点の制限を解決したのが、今やユーザ認証プログラムとしてデファクトスタンダード化したRADIUSです。RADIUSはもともとISP向けに特化した数多くのシリアルポートを擁するターミナルサーバ(PortMasterシリーズ)を開発販売していたLivingstone社が実装したプログラムです。Livingstone社は、その後買収されてしまったため、現在その社名と製品を見ることはできませんが、RADIUSの仕様を提案しているRFC 2058、2138あたりに彼らの痕跡が残っています。

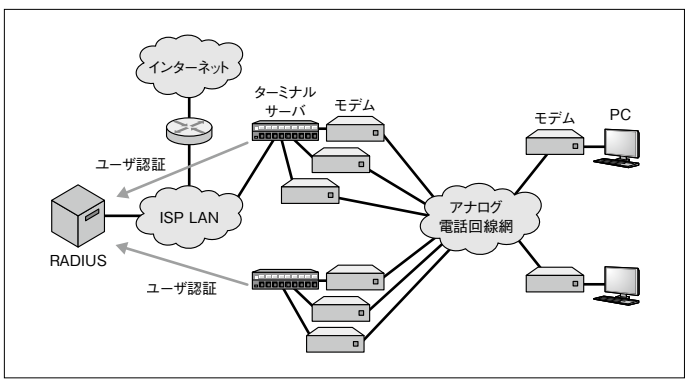
RADIUSをサポートしているターミナルサーバはダイヤルアップしてきたユーザ情報がローカルにない場合、LAN上にあるRADIUSサーバにその認証を移譲し、RADIUSサーバ側でユーザ認証を実行します(**図2**)。

したがって、ターミナルサーバとRADIUSサーバとの間にTCP/IP接続が確立されている限り、ターミナルサーバが地理的にどこにあっても同じユーザ情報を共有できることになります。ユーザからみると、ISPが提供するどの電話番号にダイヤルアップしても同じユーザ名とパスワードで認証されるので、利用する場所に依存せず、最も近い拠点の電話番号を利用できるようになりました。

今、多くの人には「インターネットに接続する」という行為も意識もないと思いますが、それらはいまだにコンピュータ内のプログラムが代行しています。同じように今、我々が行っているさまざまな作業もやがてプログラムで自動化されていくことでしょう。そのとき、自分にどんな作業が残っているのかを考えると楽しくもあり、怖くもあります。SD

注2) 個人でISPを運営しようとし、マンションに多くの回線を引き込んだ結果、公安に踏み込まれたという事例もありました。複数の回線を契約する個人宅はノミ屋か何らかの活動団体の連絡係と疑われるようです。

▼図2 RADIUS認証トポロジー



第4話

汎用機のLISP
～大文字でタイプライタで会話していたあのころ～

五味 弘（ごみ ひろし）　沖電気工業㈱　mail gomi@gomi.info

はじめに

1980年ごろ、パソコンではBASICやアセンブラが主流で、LISPは汎用機[注1]やミニコン[注2]上でしか動いていませんでした。このころのLISPはタイプライタで入力し、計算結果がアルファベット大文字で紙に印字されるものでした。タイプライタをガシャガシャと打つ音と、プリンタがジジジと印字する音で、本当に動いているんだなという実感があったLISPでした。今のようにカシャカシャと遠慮したキーボードを打つ音と、無言で画面に表示される結果を1人で見るものではなく、まわりからも作業をしているのが一目瞭然でした。本稿では、この時代のLISPとそれを取り巻く汎用機やミニコンのプログラミングについてお話ししましょう。

汎用機のLISPは小さかった

1970年代後半から1980年代前半にかけて、LISPは汎用機やミニコンで動作していました。当時大学生だった筆者は、学内のミニコン（OKITAC System50/40）で動作していたLISP50[注3]や、近くの大学の大型計算機（FACOM 230-75）で動作していたUtiLisp[注4]でLISPの初体験をしました。LISP50とUtiLispの開発者2人のおかげでLISPの初体験ができたのです。**写真1**は当時のLISPのマニュアル、**写真2**はミニコンです。

筆者が最初に触れたLISP50は、今から思えばメモリやディスクなどの環境は極少で極貧でした。LISP50のプログラムは10MB[注5]のディスクパック2個に入っていました。一方、リスト用のメモリは8kセル[注6]しかありませんでした。つまり、リストの要素が8,192個しか使えません。LISPはこの8,192個のリストの中にプログラムとデータを格納しなくてはいけません。これは非常に厳しいです。10年ほど前に筆者が（スマホでなく）携帯電話上に作ったLISPと比べても2桁以上少ないものです。図体は携帯電話の何倍になるかわからないほどですが。

もちろんこれは作者の太田氏が

注1） mainframe。基幹業務を処理する大型のコンピュータのことでメインフレームとも呼ばれる。広い場所を占有し、冷房が完備された部屋に設置された。

注2） mini computer。汎用機よりも小型なコンピュータであるが、それでも入出力機器を合わせれば、一室を占有する設備になり、ミニと呼ぶには勇気が要った。

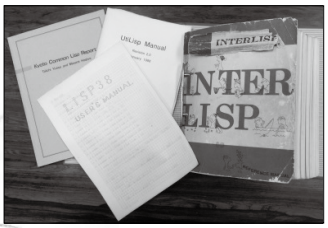
注3） LISP50は三重大学教授の太田義勝氏によって開発されたLISPで、OKITAC System50/40で動作していた。LISP50はFORTRANで記述されていて、FACOM 230-38上で開発したLISP38を移植したもの。

注4） UtiLispは東京大学教授の近山隆氏によって開発されたLISPで、FACOMなどの汎用機上で動作していた。UtiLispはアセンブラで記述されていて、多くの大学で使われていた。

注5） 単位は間違っていません。G（ギガ）でなくM（メガ）です。

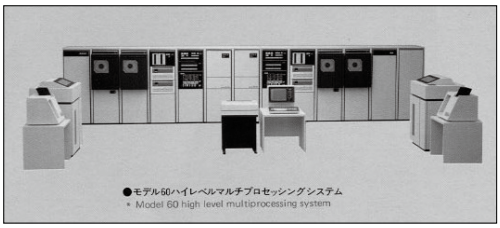
注6） LISPでリストを構成するための最小単位で、データ部と次のアドレス部がペアになったもの。たとえば16bitマシンであれば、1セルは32bitになる。

▼写真1　当時のLISPマニュアル



LISP38やUtiLisp、INTERLISP、KCLのマニュアル。なおLISP38のマニュアルは手書きだった

▼写真2　ミニコンOKITAC System50/60



ミニコンという名前にも関わらず、一室を占有していた

汎用機のLISP〜大文字でタイプライタで会話していたあのころ〜

悪いのではなく、仮想記憶もない時代のミニコンではしかたがないことでした。でもこの8kセルの中でどのようにプログラムを作るかが楽しかったのも事実です。実際にこのLISP50の8kセルだけを使い、プログラムの正当性証明システムを構築していた先輩もいました。

LISPの入出力はすごいものだった

最初に触れたLISP50は、今ふうの入出力でなく、タイプライタでLISPプログラムを打ち込み、結果が紙に印字されるというものでした。今振り返れば、何というレトロ感！　何という昭和感！だったのでしょうか。LISP50の次に触れたUtiLispは、すでにディスプレイにキャラクタとして表示されるようになり、このレトロ感がないのは残念でした。

しかし、ディスプレイとタイプライタで大きく違うことがあります。ディスプレイでは、表示された文字は誰に見られることなく、恥ずかしい間違いを闇夜に葬り去ってくれますから、Lisperとしてのプライドも保てます。しかしタイプライタでは印字結果が物理的な紙に残ります。その紙を燃やしてしまわない限り、黒歴史として残ります(ゴミ箱の中かもしれませんが)。

タイプライタのキーボードとディスプレイのキーボードも天と地ほどに違います。打鍵の重さと、それ以上に打鍵音が違います。今のキーボードのように軟弱なものではありません。まさに硬派でした。実際に硬過ぎて指が疲れましたが。でもこの重さに慣れた筆者はUNIXのキャラクタ端末を使うときも、VT-100系[注7]を選んでいて、今でもHHK[注8]です。慣れって恐ろしいものです。

革命的だったプログラミング環境

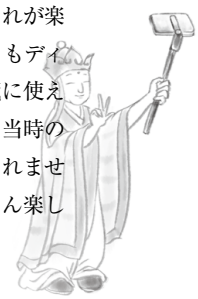
その当時の汎用機やミニコンでは、プログラミングと言えば、紙カードと紙テープでした。紙カードにプログラムをパンチャー(カードパンチマシン[注9])で打鍵して穴を開け、そのカードを1箱に詰め込んでいました。またプログラムをコンパイルしてリンクして実行するためのジョブコントロール文の紙テープを探して、紙テープリーダーに挿入しました。そして、コンソールタイプライタから実行させるのですが、そのまま素直に行くとは限りません。紙カードがジャムる[注10]のです。気を取り直して再び、実行！……でもエラーの文字がラインプリンタに勢いよく印字されます。もう一度、振り出しに戻ることになります。

この一連の喜劇と悲劇を横目に、涼しい顔で眺めていました。紙テープリーダーの近くに、専用のタイプライタがありましたので、筆者はこのタイプライタでLISPプログラムを打ち込んで、その場でデバッグをして、バグも何もなかったかのように、涼しい顔で再計算をさせていました。

LISPには構造化エディタがもれなく内包されていました。構造化エディタとはLISPのプログラムやデータ構造(これをS式(シンボリック式)と呼んでいます)を意識して、会話的に編集できるものでした。近くでコーディング用紙にプログラムを鉛筆と赤鉛筆で書き、それをパンチャーしていた人たちと比べると革命的なものでした。しかしその後、構造化エディタはEmacsなどの高性能なテキストエディタにとって代わられるものになり、今のLISPには内包されなくなりました。

おわりに

今回は1980年代前半の汎用機のLISPを見てきました。いかがだったでしょうか。LISPプログラムは昔も今も同じ思想で作ることができますが、その環境は天と地の差がありました。このような環境で今、流行の人工知能プログラムの研究も行われていました。このころは機械学習ではなく、機械をうまく使えるように人間が学習しているときでした。でもそれが楽しい時代でした。今はメモリもディスクもプロセッサも無尽蔵に使える環境になっていますが、当時の楽しみは少し減ったかもしれません。でも今のほうがもちろん楽しく遊べます！ SD



注7) DECのキャラクタ端末の1つで多く売れた端末であり、キーボードのタッチが深く重いものであった。当時、廉価版のVT-101はどこでも見かけた。

注8) Happy Hacking Keyboard。東京大学名誉教授の和田英一氏とPFUが開発したキーボードで、UNIX英字キーボード準拠で最小限のキーしか配置されておらず、一部で絶大な人気がある。なお東大の和田研はUtiLispの開発元の研究室。

注9) Card punch machine。紙カードに穴を開けるマシン。キーボードで打鍵すると紙カードに穴が開いた。この作業をする人をパンチャーと呼んでいた。それからカードパンチマシンのこともパンチャーと呼んでいた。

注10) 機関銃などで空になった薬莢が詰まること。これからプリンタの紙が詰まることや、紙カードリーダーに紙カードが詰まることを意味する。

第5話

IDEのさきがけとなったTurbo PascalとTurbo C

大野 元久（おおの もとひさ） twitter @mohno

はじめに

C言語は古くから使われ続けている言語の1つで、とくに日本では昭和の終わりから平成にかけては大ブームが起こりました。今なおCやC++言語を使う開発者が多いのは、その時代から受け継がれた知識や資産があるためです。

1987年にBorland International社（当時、以下Borland）が発売したTurbo Cは、そのブームの中心にいました。

Turbo Pascal

もともとBorlandは、Turbo Pascalという製品を販売していました。今でこそコンパイラや開発環境は、高機能なものを無料で入手できますが、当時は「ちゃんとしたもの」が高額で販売されているか、機能が制限されたものが個人レベルで無料～安価で提供されていた程度でした。プログラムを編集するためのエディタすら、別個に販売されていた時代です。

その中で、Turbo Pascalは安価（最初のバージョンは$49.99～$69.99[注1]）にもかかわらず、コンパイラとエディタを一体化させた開発環境を持っていました。コンパイラの性能は抜群で、高価なものに引けを取らないどころか、スピードも効率も圧倒的でした。

また、コピープロテクトのない製品があり、「書籍のように扱う」というユーザ本位のライセンスも特徴的でした。

Turbo Pascalは人気を博しましたが、残念なのは、それが「Pascal」だったことでしょう。Pascalはもっぱら「教育用の言語」と評されており、プログラミング学習や趣味では使われるものの、業務用プログラム開発の主流に食い込むことはできませんでした（後年、Pascalを使うDelphiという画期的な製品が登場しますが、本稿では割愛します）。

C言語の隆盛

C言語は、早くから注目されていました。『プログラミング言語C』、いわゆるK&R本の訳書が登場したのは1981年のことです。C言語はもともとUNIXを記述するために開発された言語で、それくらい実行速度が速く「高級言語なのに低レベルの処理ができる」といった特徴があります。

C言語への人気が高まる中、数多くのC言語製品が登場しましたが、性能の良いものは高価であり、比較的安いものは制約があったり性能が悪かったり、クセ（あるいは不具合）がある、というものでした。「Turbo PascalのようなCコンパイラがあればいいのに」という願望は常にありました。

Turbo Cの登場

そして1987年、ついにTurbo C 1.0が登場しました。Turbo Cが「コンパクト」だったのは$99.95という価格だけで、コンパイルの速度も作成されるプログラムの性能も高額な製品に匹敵するものです。エディタやコンパイラ、リンカという開発に必要な一通りの機能を単独のツールとして統合開発環境（Integrated Development Environment；IDE）と呼ぶようになったのはTurbo C 1.0（および同時に発売されたTurbo Pascal 4.0）が最初です。

Turbo Cは、Microsoft C（当時）と並ぶ人気を博し、Borlandとしても Turbo Pascalを上回る大ヒット商品に成長しました。

注1）Turbo Pascal発売の1983年の為替レートは1ドル約240円、Turbo C発売の1987年では1ドル約140円。

IDEのさきがけとなったTurbo PascalとTurbo C

日本における状況

もともとTurbo Pascalは、さまざまなマシンに搭載されていたCP/MというOS向けに作られていたものです。キーボードや画面に対する入出力は汎用性の高い方法が使われており、そのままPC-9800シリーズなどの日本のパソコンで使うことができました。

ところが、Turbo Cの統合開発環境はIBM-PCに特化して作られていました。開発機能は格段に向上しましたが、Turbo Pascal(バージョン3.0以前)のように、そのまま日本のパソコンで使うということはできませんでした。

そんな中、月刊『The BASIC』誌(技術評論社)にSIM(星野操氏作)というツールが掲載されました。これはIBM-PC向けに開発されたソフトを、PC-9800シリーズで使うためのツールです。Turbo Cも実行ファイルを少し書き換えるだけで使えるようになるため、筆者を含め、日本語化を待ちきれずにSIMを使ってTurbo Cに触り始めた、という人も多いでしょう。

日本語版が発売されたのはTurbo C 1.5からでしたが、この日本語化は、当時Borlandの正規代理店だった2つの会社によって独立に行われました。Borlandとしては2社に競わせて、より売上の高いほうに今後の日本市場を託す意味もあったのでしょう。結果としてマイクロソフトウェア・アソシエイツ(当時)が優位に立ち、Borlandの日本法人(ボーランドジャパン)設立のパートナーとなったのです(1989年4月)。

ライブラリ

最初のバージョンでは、IBM-PC固有の機能を使っていたのは統合開発環境だけでしたが、次のバージョン(Turbo C 1.5)では、IBM-PCの画面(コンソールやグラフィック)を直接操作する機能がライブラリに含まれました。当時、この手の機能は製品自身ではなく、サードパーティから別売りされるのが一般的でした。Turbo Cは、それを製品の一部として取り入れたのです。追加の出費なしで、こうした機能が利用できるのも魅力になっていました。

また、BorlandはTurbo Cのライブラリのソースコードも販売しました。今では考えにくいかもしれませんが、当時、ほとんどのC言語製品はライブラリのソースコードは他社にノウハウを盗まれないようにするため非公開なのが当たり前でした。「どんな処理をしているかわからないものは使えない」と、標準ライブラリを使わず独自に機能を実装することもあったくらいです。

Turbo Cは、ライブラリのソースコードを公開することで、開発者が自由に中身を確認し、改変できるようにしたのです。

実際、ソースコードを見られることで、Turbo Cのライブラリ関数が速い理由もすぐにわかりました。Turbo CにはC言語のプログラム中にアセンブリ言語を記述できる機能がありましたが、ほとんどの関数はそうしたアセンブリ言語を使って記述されていたのです。

▼図1 日本語版Turbo C 2.0の画面(同製品のカタログより)

ANSI C

Turbo C、あるいはC言語そのものが幅広く受け入れられた理由の1つにANSI Cという規格があります。それまでのC言語製品は、それぞれ文法や関数の機能に少なからず違いがありました。これでは開発者が安心できないと、業界を挙げて規格化が進み、1989年にANSI Cが制定されました。

規格化により、製品の違いに悩まされることも少なくなり、C言語の普及がいっそう進みました。Turbo Cは規格化が進む段階からANSI Cへの対応に積極的であり、安心できる選択肢になりました。

おわりに

かつて、休日にプログラミングする人たちのことを「サンデープログラマ」と呼んだ時代がありました。Turbo PascalやTurbo Cは、そうした人たちの味方であるとともに、仕事でも使える製品として成長していきました。

開発ツールがソフトウェア市場の一端を担っていた時代に、先進的な機能を提供し続けることができたのは誇らしく、利用者の心にTurbo PascalやTurbo Cが刻まれているのであれば、当時の関係者としてうれしいことです。SD

第6話

VZエディタ開発秘話

兵藤 嘉彦(ひょうどう よしひこ) twitter @c_mos

VZエディタとは

VZエディタは、ビレッジセンターが1991年頃から発売していたMS-DOS用のテキストエディタです。当時のエディタとしては画期的だったメモリ常駐モードを備え、ESCキーだけで呼び出せたり、DOSの簡易シェル機能のような、バックスクロール(消えてしまった画面を記録して呼び出す)が扱えたり、マクロ機能を有しており、いろいろな機能追加ができました。当時は数々の賞をとっている画期的なエディタでした。

はじめに

よく「VZの○×機能は良かったですね!」と言われますが、実はよく覚えていなかったりします(笑)。ひと仕事終えると、苦労や辛いことはすっかり忘れちゃうんですよね。そういう性格だからこそ、懲りずに何十年もプログラマやっていられるのかな?

VZのWindows版を開発している夢をよく見ます。目が覚めたときは、せつないです……。未だにバージョンアップを続けているWZ EditorやMIFESのニュースを見かけると、「よく続くなぁ……」と感心する一方、ちょっと悔しいです(笑)。

ただ、プログラマはIDEでコーディングするのが主流になって、もはやエディタをメインツールとして使うのはライターの方くらいでしょう。ちなみに現在、筆者はEclipseがメインで、C++のみVisual Studioの無償版を使っています。先日、Markdownの編集用にVisual Studio Codeをちょっとかじりました。この原稿は秀丸で書いてます。

VZの誕生

さて、VZを作った頃の話を少ししましょう。筆者が初めて買ったパソコンは、PC-9801U2でした。プログラムを開発するためには、当然、まずエディタから作ります(笑)。当時、最初に使っていたエディタは、大学のゼミのPC-9801で使っていたシンプルなスクリーンエディタでした。UNIX系やWordStar[注1]といった正統派のエディタを使った経験はなく、タッチタイピングもできなかったので、矢印キーでカーソルを移動し、左手側のCtrlキーで編集、というスタイルです。これはまわりの玄人衆からは評判が良くなく、渋々WordStarふうのキーアサイン(ダイヤモンドカーソル[注2]や2ストロークコマンド[注3]との導入など)にしたわけです。今から考えると、後年のMacに始まる標準エディタの編集スタイルには、むしろ筆者のほうが近かったなぁと思います。

そして最初にできたのが、偶然にも(笑)、9,801byteのEZエディタです。それを最近の本連載を執筆されている速水祐氏のマイコンクラブ[注4]でデモする機会があり、そこのメンバーから指摘されたさまざまな意見を取り入れて完成に漕ぎ着け、某雑誌社から発売の運びとなりました。サイズは倍の20kbyteになっていました。その後、雑誌社の経営が思わしくなくなり、印税の支払いも滞るようになったため、ビレッジセンターからVZエディタとして新しく発売することになりました。実は、ビ

注1) テキストベースのワープロ。最初はCP/M上で動作していたが、MS-DOS版も発売された。

注2) Ctrlキーとその近くのE(上)、S(左)、D(右)、X(下)を同時に押すことでカーソル移動を行う方法。菱形に並んだキー配列からダイヤモンドカーソルと呼ばれた。後にWindows3.1がCtrl+Xにカット機能を割り振ったために使われなくなった。

注3) Ctrlキーと Qキーの同時押しのあと、特定のキーを押して行う操作。

注4) http://zob.club/

レッジセンターの中村満社長と引き合わせてくださったのも、速水祐氏です。氏と出会わなければ、VZエディタ(**図1、2**)が世に出ることは、決してなかったでしょう。

VZの機能紹介

ここでVZの筆者お気に入りの機能を紹介します。まずは「テキストスタック」。当時、クリップボードやUndo/Redoという概念は、まだ知りませんでした。テキストスタックには複数の行またはブロックを格納できるため、1行削除を連打し、カーソルを移動して、今度は1行挿入を連打することで、範囲指定することなく、簡単に複数行の移動ができました。

次に「ヒストリー機能」。検索文字列やコマンドライン入力で↑キーを押すと、過去に入力した文字列を順次表示します。さらに、str↑と打てば、「str」で始まる文字列だけを表示します。後者はとても便利な機能だと思いますが、Windows上のDOS窓(コマンドプロンプト)では未だに使えません。

逆に、これは失敗したなぁという機能もあります。次ページ／前ページコマンドが、3モードのトグル[注5]になっていた点です。タイトル行サーチは、「*** / ;**」などの記号で始まり、直前が空行の行にジャンプする機能です。プログラム上の関数の頭出しを意図しています。しかし、これをいちいち切り替えるのはやはり面倒でした。モードというのはUI的に素性が良くなかったようです。

おわりに

過去の自分に戻って、人生をやり直す……というお話があります。今の記憶を持ったまま30年前に戻ったら……、もう一度アセンブラでエディタを書く根性なんて、もちろんありません(笑)。といって記憶もリセットしたら、また同じ人生を辿るだけでしょう。

大学に入ってマイコン(H68/TR)と出会ってからの約10年間、起きている間はひたすらプログラム書いたり、考えたりしていました。まったく新しいRPGを作る、という野望もありました。そのためのエディタでしたが、結局そこで力尽きてしまいました。まぁ自分にはツール作りが向いていたのかもしれません。

先日、新入社員のプログラミング研修をお手伝いする機会がありました。プログラムが苦手な人の間違い方には、興味深い共通点があります。プログラマ向きの性格って何なんでしょう。おそらく、コード、ツール、棲む空間が、もっと使い勝手がよくならないか、快適にならないか、考えずにはいられない性格です。与えられたモノを漫然と頑固に使い続けている人のいかに多いことか。常に工夫し続ける、それがプログラマだと思います。

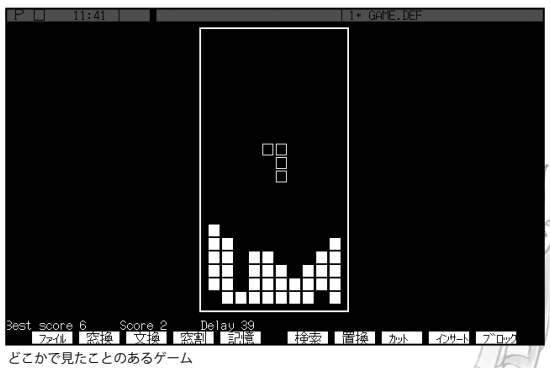
最後にネタを1つ。NHK BSで新世紀エヴァンゲリオンのテレビシリーズが始まりました。第7話のJAというロボットが起動する場面で、コンソールをよーく見てください。VZのおまけのメモリ表示ツール「VMAP.COM」の画面にそっくりです(笑)。VZが人々の記憶から忘れられても、筆者の生きた証はここに刻まれています……。SD

注5) 通常の全画面／半画面送り、タイトル行サーチ、文字列サーチの3モードです。

▼図1　VZエディタの画面

モードメニューを表示したところ

▼図2　VZエディタのマクロ実行画面



どこかで見たことのあるゲーム

第7話

あこがれのグラフィックスソフト

古籏 一浩(ふるはた かずひろ) Twitter @openspc

はじめに

30年前、筆者が所有していたマシンはSHARPのMZ-700でした。ところが、このマシンはグラフィックス機能がありません。文字しか表示できないのです。プロッタプリンタ注1を使えば紙の上では絵は描けますが、ほかのマシンのようにディスプレイ上に、自由にグラフィックを描くことは夢でした。

ベクターとピクセル

開発者のみなさんも、図やイラストを描く場面はあるかと思います。表計算ソフトとして使われるExcelでもいろいろな図形が描けます。Excelを簡易的なグラフィックスソフトとして使う人もいるでしょう。

グラフィックスソフトは大きく分けてピクセル形式のものとベクター形式注2のものがあります。1980年代前半のコンピュータの多くは320 × 200／640 × 400pixelサイズの画面解像度で、同時に可能な発色数は8色(黒、青、赤、紫、緑、水色、黄、白)でした。

当時はお絵かきソフトがほとんどなく、BASICなどのプログラムを使ってグラフィックを表示していました。この場合、座標をもとにして描画するのでベクター形式になります。ゲームではデゼニランド注3など多くがベクター形式を採用していました。ベクター形式は座標データ+αだけでデータ量が少なくて済むので当時のメモリが少ないマシンにとっては好都合だったのです。

8bitから16bitへ

1980年代中期になると、記録メディアとしてフロッピーディスク(FDD)が一般的になりました。大容量のデータを扱えるようになり、サイズの大きいピクセル形式の画像データを、そのまま画面に表示できるようになりました。スキャナで取り込んだ画像を修正しFDDに記録しておきそれを直接転送することで、高速に画面表示することが可能になりました。これにより表現の幅が格段に広がりました。ゲームでは、日本ファルコムが販売していたアステカが、データをディスクから転送し表示しており、瞬間画面表示と謳(うた)っていました。後に同社の知名度を向上させたXanadu(ザナドゥ)でも同様の手法が使われていました。タイトル画面でキャラクタがエフェクトとともに格好良く表示されていくのは今でも記憶に残っています。

多くの画像データが扱えるようになると、必要になるのがお絵かきソフトです。それもベクター形式ではなく1pixel(ドット)ごと自由に点を描けるピクセル形式のお絵かきソフトです。1980年代中期になるとパソコンも8bitから16bitマシンに移行していきました。PC-9801、X68000、FM TOWNSなど8bitマシンとは比較にならないパワーを持っていました。そんな中、登場したお絵かきソフトがZ's STAFF(ジーズスタッフ)でした。X68000版はZ's STAFF PRO-68Kという名前で販売されていました。当時、最強とまで言われたZ's STAFFですが、残念ながら金欠で買えませんでした。高額なX68000を買って貧乏になり過ぎた、というのもあります。X68000は65,536色同時

注1) プロッタとも呼ばれる。ヘッドの代わりにペンを使って線を描くように印刷するプリンタ。

注2) 最近のベクター形式は、滑らかな線で結べるスケーラブルなデータ形式を指しますが、本稿では座標を指定して線や図形、塗りつぶしを行うものをベクター形式と呼んでいます。

注3) ハドソン社による某ネズミ系テーマパークをもじったゲーム。

発色可能でしたが、富士通のFM77AV40はさらに多くの色(26万色中64,000同時発色)を表現できます。実はFM77AV40も購入したのですが、いまだに未開封のまま家にあります。

MacPaint

もう1つ、1980年代中期で欠かせないお絵かきソフトがあります。それはMacintosh用のMacPaint(マックペイント)です(**写真1**)。MacPaintは白黒のお絵かきソフトです。当時のマシンは8色以上表現できるのが一般的でしたが、Macintoshは白黒でした。当たり前ですが、白黒よりも多色のほうが表現力は上に決まっています。決まっているはずなんですが、MacPaintを使うとあら不思議。白黒なのに意外なほどの表現力。おまけにすごく描きやすい。これはマウスの影響も大きかったのかもしれませんが、すらすらと絵が描けることは驚きでした。

そのあと、会社にMacintosh IIが導入されPixelPaintというお絵かきソフトがインストールされていました。時代としてはZ's STAFFと同時期ですが、使ってみてあまりの凄さにビックリ。フルカラーでお絵かきができ、斜めグラデーションなども可能でした。なげなわツールで領域をくり抜いて移動なんてのも楽々。おまけに高速でスムーズ。日本と海外の差を感じた瞬間でした。

花子とIllustrator

1980年代中期のベクター形式のソフトとして、ジャストシステムが販売していた花子があります。筆者は仕事で回路図などを描くときなどに使っていました。けっこう描きやすいなあと思っていましたが、Macintoshが導入され、MacDraw(**写真2**)を使った瞬間に花子で何かを描くという選択肢がなくなりました。描きやすさがあまりに違いました。しかし、MacDrawは精度が足らずPostScriptプリンタ(レーザープリンタ)で印刷すると線がはみ出したり、文字が期待する位置に表示されなかったりと、いくつか問題がありました。

そこでPostScriptプリンタをフル活用できるグラフィックスソフトであるAdobe Illustrator 1.0(英語版)を導入しました。最初はAdobe独特の職人っぽい操作方法に慣れず四苦八苦しました。慣れてくると、けっこう使いやすいソフトです。筆者は今でもIllustratorを愛用しています。

そしてIllustratorのあとに、ライバルソフトとなるAldus FreeHandが登場します。Illustratorは作図には便利でしたが、イラストを描くのには向いていませんでした。FreeHandのほうがイラストを作るのには向いていたような感じがありました。そんなFreeHandもさまざまな事情により今ではなくなってしまいました。

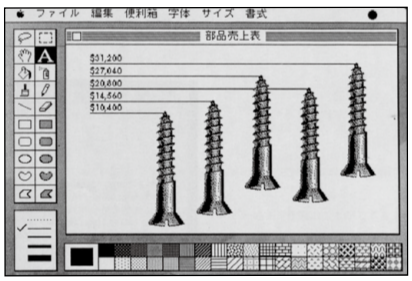
終わりに

1990年代に入ると有名なPhotoshopが登場します。初期バージョンはたしかPhotoshop 1.0.7だったと思いますがレイヤー機能などはなく本当に写真のレタッチ程度しかできませんでした。また、当時のマシンでPhotoshopを扱うのは荷が重くハードディスクから、ゆっくりと1ラインずつ画像が転送されてくるのを眺めるほどでした。

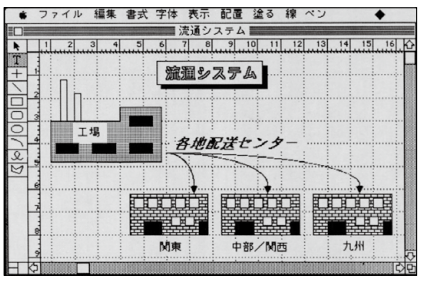
30年前と比べてグラフィックスソフトは格段に進歩しました。しかし、いくらすごいグラフィックスソフトを持っていても、自分に絵心と画力がないので宝の持ち腐れです。筆者は、以前から絵が上手に描けないのはグラフィックスソフトが悪いのだと思っていましたが、結局悪かったのは地道に絵の練習をしない自分のほうでした。やはり地道な努力は大事なのです。

SD

▼写真1　MacPaint



▼写真2　MacDraw



第8話

オープンソースの夜明けと「まつり」

日本UNIXユーザ会 http://www.jus.or.jp/
法林 浩之（ほうりん ひろゆき）　hourin@suplex.gr.jp

はじめに

今ではオープンソースの概念は広く知られていますが、昔からITの世界にいる人達にとっては後天的に得た言葉です。本稿では、オープンソースが日本にやってきたころのことを紹介します。

オープンソースの来日

オープンソースという言葉が作られたのは1998年ぐらいですが、筆者にとっての初見は、1998年11月に行われたThe Perl Conference Japan[注1]（オライリー・ジャパン主催）におけるティム・オライリー氏の講演でした。証拠はありませんが、オープンソースの概念が日本国内で紹介されたのは、これが最初かもしれません。にもかかわらず、筆者をはじめ聴衆の多くは、おそらくその考え方をすぐに理解できたと思われます。というのは、もともとインターネットの世界では、ソフトウェアのソースコードを公開することは普通に行われていたからです。つまりオープンソースとは、技術者達の間で暗黙の内に共有されていた考え方に後から名前がついたものであり、それゆえに広く受け入れられたのではないかと考えられます。

オープンソースの来日と前後して、1990年代に入ると各種ソフトウェアのユーザ会が設立されるようになりました。ちょうど日本国内におけるインターネットの普及が進んだ時期であり、メーリングリストや掲示板などネットを介して情報交換ができるようになったことが、ユーザ会の発展を促したのです。

jusを取り巻く状況と、コードネーム「まつり」

jus（日本UNIXユーザ会：Japan UNIX Society）は、1986年から10年間に渡り、UNIX機器およびソフトウェアの展示会・UNIX Fairを開催してきました。しかし、安価なWindowsマシンの登場によるUNIXのシェア低下とともに、UNIX Fairは終了し、後継イベントとして1997年に開催したNetwork Users'も惨敗に終わってしまいました。

そのころからjusでは、商業的な展示会とは異なる、エンジニアたちが持ち寄った技術を見せ合うお祭りのようなイベントを模索していて、それを「まつり」と仮称していました。また、当時はFreeBSDやLinuxなどのPC-UNIXが広まりつつある時期で、秋葉原でPC-UNIXマシンを販売していた「ぷらっとホーム」と一緒にイベントを企画していました。そこへオープンソースがやってきたことで、一気に開催の機運が高まったのです。

まつり開催へ

「まつり」の開催にあたり基本線と考えていたのは、仕事でも趣味でも来場できるように、

- 金曜・土曜の2日間開催する
- 日本最大の電気街である秋葉原で開催する
- 各種ユーザ会に集まってもらって展示やセミナーをしてもらう

ことでした。とくに会場に関しては、IT業界だけでなく一般の方々

▼写真1　第1回まつりのステージ。スクリーン右で司会をしているのが筆者

注1）http://www.oreilly.co.jp/pcjp98/

オープンソースの夜明けと「まつり」

にも見てもらうことを狙って、秋葉原の中央通りに面した廣瀬本社ビルのイベントホールをメイン会場として利用しました。

イベント名称は開催直前まで悩みましたが、オープンソースという言葉の目新しさを買って「オープンソースまつり '99 in 秋葉原」[注2]としました(**写真1、2**)。日本Linux協会(jla)も主催に加わり、jus、ぷらっとホーム、jlaの三者共催で、1999年11月12、13日(金、土曜)にまつりは決行されました。14件のセミナー、20件の展示、21件のステージセッションを行い、2日間で7,300人を動員したのです。

2回しかできなかった理由

オープンソースまつりは、我々の世界を知ってもらうという意味では成功しましたが、財政的には非常に苦しいものでした。今でこそオープンソースやコミュニティ主催のイベントにも、多額の協賛金が出るようになりましたが、この時代はまだオープンソースを事業とする企業はごくわずかで、この種のイベントへの協賛はほとんど得られませんでした。また、大規模なイベントをボランティアベースで動かすノウハウがなく、事務的な部分をイベント業者に委託していたため、多額の運営委託費を支払っていました。その結果、初回も2回目(2001年2月)[注3](**写真3、4、5**)も1千万円近い赤字を計上し、開催を重ねられる状況ではなくなってしまったのです。

まつりの遺伝子

こうしてまつりは2回で終了しましたが、イベントとしてはエンジニアの心をつかむものだったので、自分たちもあれをやりたいという動きが各地で出てきました。たとえば大阪では、2002年から「関西オープンソース+フリーウェア」が開催され、現在も関西オープンフォーラム[注4]として継続しています。

2004年からはオープンソースカンファレンス[注5]が始まり、こちらは全国展開して今も健在です。さらにコミュニティ主導のイベントは、オープンソース以外の分野にも波及し、協賛してくれる企業も増えたので、今では年間を通して数え切れないほどのイベントが開催されるようになりました。

伝えたいこと

今ではオープンソースの誕生後にITの世界に入った人も多く、そういう人々にはオープンソースは当たり前の概念と思われるかもしれませんが、そうではないのです。先人たちがさまざまな苦労や痛い思いをしながら普及に尽力し、今日の地位を勝ち取ってきたのです。オープンソースまつりは、その一端を担ったにすぎませんが、日本のオープンソースの歴史には欠かせない出来事の1つであり、今にして思えばこのころがオープンソースの夜明けだったと思います。かつて、こういう行事があったことを知っておいてもらえるとうれしいです。SD

▼写真5　オープンソースまつり2001のパンフレット

▼写真2　第1回まつりのメイン会場

▼写真3　第2回まつりのメイン会場

▼写真4　第2回まつりのステージ

注2) http://www.jus.or.jp/matsuri/index.html

注3) http://www.jus.or.jp/matsuri2001/index.html

注4) https://k-of.jp/

注5) http://www.ospn.jp/

最終話

オープンソースとコミュニティ

田中 邦裕（たなか くにひろ）　さくらインターネット㈱　twitter @kunihirotanaka

学内のインターネット

筆者は1993年に中学を卒業して、ロボコンに出るべく舞鶴工業高等専門学校（舞鶴高専）へ入学したのですが、ロボコンの傍（かたわ）らCADシステムのパソコンやネットワークにどっぷりはまりこみ、2年生だった1994年ころには学内LAN上に自分のサーバを構築して遊んでいました。

そのころは、ちょうどAT&T JensやIIJなどがダイヤルアップ[注1]によるインターネット接続サービスを始める時期で、舞鶴高専はまだインターネットに常時接続されておらず、UUCPで1日1回だけメールのやりとりができるという環境でした。

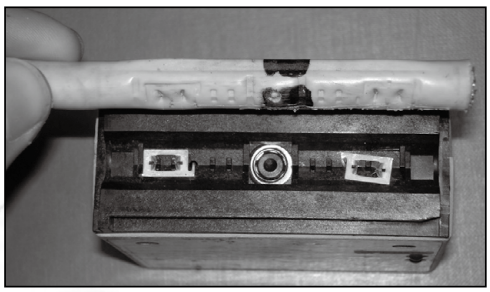
なお学内のLANは、通称イエローケーブルと呼ばれる同軸ケーブルを使った10MbpsのEthernetで、ケーブルに「バンパイアタップ」と呼ばれる針を差し込んでノードを増設していました（**写真1**）。ちなみに、バンパイアタップの差し込みに失敗すると同軸ケーブルの中心線を切断してしまって通信ができなくなり、とてつもなく怒られるので、非常に慎重にやっていたというのは舞鶴高専時代の語り草です。

そこにつないでいたサーバは、当初Sun-3という古いワークステーションで、ビックリするくらい大きなカセットをドライブに差し込んで、OSインストールからネットワーク設定、サーバ構築を行っていました。

その後、DECのAlphaというサーバが研究室に入り、今度はそれをおもちゃにしていたのですが、コンパイラを始め、多くのソフトウェアが非常に高価で、何百万円もするものもザラだったので、学生の身分で遊ぶにはなかなかたいへんだったと記憶しています。

注1）電話回線とモデムでネットワークに接続する方法。

▼写真1　バンパイアタップとケーブル



By No machine-readable author provided. Alistair1978 assumed (based on copyright claims). - No machine-readable source provided. Own work assumed (based on copyright claims)., CC BY-SA 2.5, https://commons.wikimedia.org/w/index.php?curid=426331

オープンソースとの出会い

商用ワークステーションやサーバ、そしてソフトウェアが非常に高かったこともあり、FreeBSDやCERN httpdなどのオープンソースを使って自分のパソコンをサーバにすることにしました。これにより安価で高速な環境と、たくさんの機能が得られるようになりました。

とはいえ、当時のオープンソースはまだ枯れたものではなく、かつFreeBSDはUNIXライセンスの問題もあって、技術的にも法的にも不安定な状況にありました。またFreeBSD 1.Xでは、米国の暗号化規制もあって、/etc/passwdのパスワードが暗号化されていないという、今から思うととんでもない仕様になっていました。

このような状況だったので、常に新しいバージョンのオープンソースを追い求めるという状況にあったのですが、そのダウンロードが課題になっていました。

学内LANは、未だにインターネットに常時接続されていなかったので、LinuxやFreeBSDのパッ

チを取得する際には、archieという公開FTPサーバのファイル検索ができるシステムと、ftpmailというメール経由でのファイル取得ができるシステムを使い、メールを使ってファイルを取得していました。

そのため、「archieで検索をメールで送信して、返信があるまで2日」「ftpmailで送信してuuencodeされたファイルを受け取るまでに2日」「コマンドを間違えていたらやり直し」といった感じで、1週間くらいかけてパッチを取得していたことを思い出します。まるで郵便のようなレスポンスでデータのやりとりをしていたので、1996年に学内のネットワークがインターネットへ常時接続されたときの感動は忘れられません。

自分のパソコンから、自由に世界中のサーバに接続できるというのは本当に衝撃で、意味もわからずCERNやNASAなどのホームページへアクセスしていたのは良い思い出ですし、自由にオープンソースやパッチをダウンロードできるようになったのは本当に感動しました。

こう思うと、オープンソースの発展とインターネットの発展というのは、重なり合う部分があるなぁとひしひしと感じます。

コミュニティ活動

そのころ、CERN httpdよりも良いWebサーバソフトウェアがあるということでApacheに手を出したのですが、当時はバージョンがまだ1.0になったばかりでした。a patchが語源ともされるApacheはまだまだバグが多く、筆者もしょっちゅうバグレポートや修正パッチを送っていました。

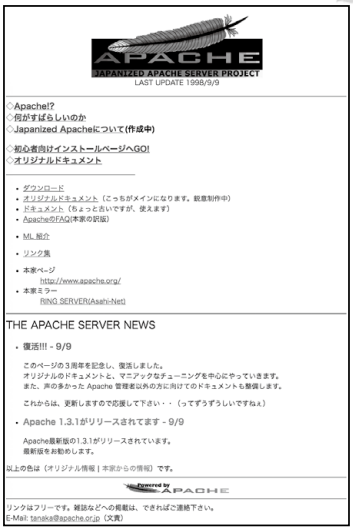
余談ですが、オープンソースの価値は無料であることだけではなく、自分でソースを見ながらバグ潰しができるという利便性や、バグレポートやパッチを送ってソースへ取り込まれたときの感動を得られるなど、ITエンジニアの成長過程において重要な価値を持っていると思います。

また、筆者は当時東大におられた安東孝二氏などと「日本Apacheユーザ会」というコミュニティを立ち上げ、Apacheの日本語ドキュメントの翻訳や、MLの開設、apache.jpによる情報提供などを行っていました(**図1**)。そのため、このときに出会った方々とのつながりは今でも深く、オープンソースに加えて、コミュニティ活動の大切さを知ることになりました。そして、筆者たちが翻訳していたドキュメントは公式のドキュメントとなり、MLは今でも情報交換の場として活用してもらっています。

なお、このころは、今につながるような多くのオープンソースがほかにもたくさん産まれていた時代でした。Apacheにモジュールとして組み込めるスクリプト言語、PHP/FI(今のPHPの原型)も勢力を伸ばし始めていて、筆者も日本語関係のバグ報告をしたりパッチを送ったりしていました。

また、「ようこそ私のホームページへ」のようなサイトが流行っていたころでもあり、PHP/FIを使って画像ではないインラインのアクセスカウンタを作ったり、掲示板やチャットなどを作ったりしていたころでもあります。

▼図1　1998年当時の日本Apacheユーザ会のサイト



なお、学校に居候していた筆者のサーバを利用する人も増え、その影響でトラフィックも増え、学外にサーバを移設する際に、その維持のためにお金をいただくようになって、さくらインターネットを創業したのが、まさにこのころの話です。

終わりに

今回紹介したのは、1994～1996年のことですが、ちょうど日本のインターネットが成長を始めた時期であり、LinuxやFreeBSD、Apache、PHPといった、今でも使われる各種オープンソースが勃興(ぼっこう)しつつあった時期でもあります。

また、オープンソース系のコミュニティが活発になり始めた時期でもあり、今では以前にも増してその文化が根付いたことはたいへんうれしいことですし、今後もオープンソースとコミュニティがインターネットを引っ張って行ってくれることだろうと思います。SD

特別企画

［次世代言語］Elixirの実力を知る 後編

Phoenixで高機能Webアプリ開発

Author 大原 常徳（おおはら つねのり）
株式会社ドリコム
Twitter @ohrdev
Mail ohr486@gmail.com

Rubyのような書き味で、簡単に並行処理が実現できる関数型プログラミング言語「Elixir」の入門記事。前編では、Elixirの概要、環境作成、簡単なコーディングを紹介しました。後編では、プロセスによる並列処理、mixを使ったプロジェクト作成、PhoenixによるWebアプリ作成について解説します。

Elixirにおけるプロセスと Phoenixによるアプリ開発

Elixirでアプリを作ろう

本記事では、次に挙げる執筆時点（2016年10月）の最新バージョンを前提としています注1。

- Elixir 1.3.4、Erlang/OTP 19.1
- Phoenix 1.2.1

注1）Elixir、Erlang/OTPのインストールについては、前編（本誌2016年11月号）を参照してください。

Elixirではアプリをどう作るか

手続き型言語では、プログラムを組む際に、実行する処理（手続き）を中心に考えてプログラミングを行います。一方Elixirでは、プロセスを中心に考えてプログラミングを行います。

簡単な例として、ランダムな時間sleepして引数を返す処理を挙げて説明します（**リスト1**）。関数do_something（①）を100回繰り返すプログラムを考えてみましょう。

▼リスト1　プロセスの利用例

```
defmodule ProcessSample do
  # ランダムな時間sleepして引数を返す――①
  def do_something(arg) do
    sleep_time = round(:rand.uniform * 1000)
    :timer.sleep(sleep_time)
    IO.puts " sleep:#{inspect sleep_time}msec, arg:#{inspect arg}"
    arg
  end

  # 直列に実行――②
  def serial_exec do
    1..100
    |> Enum.map(&do_something(&1)) # fn(arg) -> do_something(arg) endは&(do_something(&1))と書けます
  end

  # 並列に実行――③
  def parallel_exec do
    1..100
    # do_somethingを実行するプロセスを作成
    |> Enum.map(&Task.async(ProcessSample, :do_something, [&1]))――③-1
    |> Enum.map(&Task.await(&1)) # ③-1の結果を取得――③-2
  end
end
```

素直に処理を記述するのであれば、②のようにEnum.mapに関数do_somethingを引数として渡して、1〜100の引数に対してそれぞれ適用させます[注2]。実行結果は(**図1**)となります。このように直列に処理が実行される場合、**図2**のように繰り返しの数が多ければ多いほど、処理時間は長くなります。

次にこの処理を、プロセスを使って並列に実行してみましょう。Elixirは、非同期に処理を実行する手段として、Taskモジュール[注3]を提供しています。

Taskモジュールによって、Task.async(モジュール, 関数, 引数リスト)(③-1)で引数として与えられたモジュール.関数(引数)を実行するプロセスを作成し、Task.await(my_task)(③-2)でその結果を取得できます。実行結果は、**図3**となります。並列に実行した場合

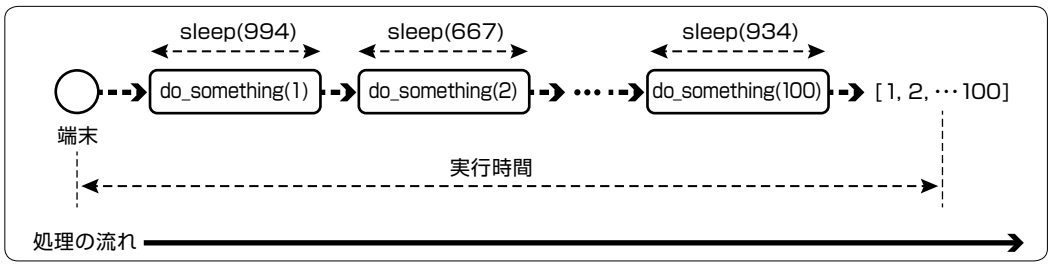
注2) Elixirでは関数を値として扱えます。関数を引数として渡す場合は、キーワードfn・endを使い、無名関数「fn 引数 -> 実行したい処理 end」として表現します。また、&演算子を使って「&実行したい処理(&1, …)」と省略もできます。&1は第一引数、&2は第二引数……、となります。よって2つの引数の和を求める関数は「fn(a, b) -> a + b end」または&(&1 + &2)と表現できます。

注3) URL http://elixir-lang.org/docs/stable/elixir/Task.html

▼図1　直列実行の結果(リスト1、②)

```
$ iex process_sample.ex
iex(1)> ProcessSample.serial_exec
 sleep:600msec, arg:1
 sleep:86msec, arg:2
 sleep:624msec, arg:3
 ...(中略)...
 sleep:228msec, arg:98
 sleep:729msec, arg:99
 sleep:934msec, arg:100
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
 43, 44, 45, 46, 47, 48, 49, 50, ...]
iex(2)>
```

▼図2　直列実行のイメージ



▼図3　並列実行の結果(リスト1、③)

```
$ iex process_sample.ex
iex(1)> ProcessSample.parallel_exec
 sleep:0msec, arg:35
 sleep:2msec, arg:54
 sleep:9msec, arg:17
 ...(中略)...
 sleep:972msec, arg:61
 sleep:973msec, arg:60
 sleep:973msec, arg:8
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
 43, 44, 45, 46, 47, 48, 49, 50, ...]
iex(2)>
```

は、**図4**のように繰り返しの数によらず、一定時間で結果が返ってきます。

辞書アプリの作成

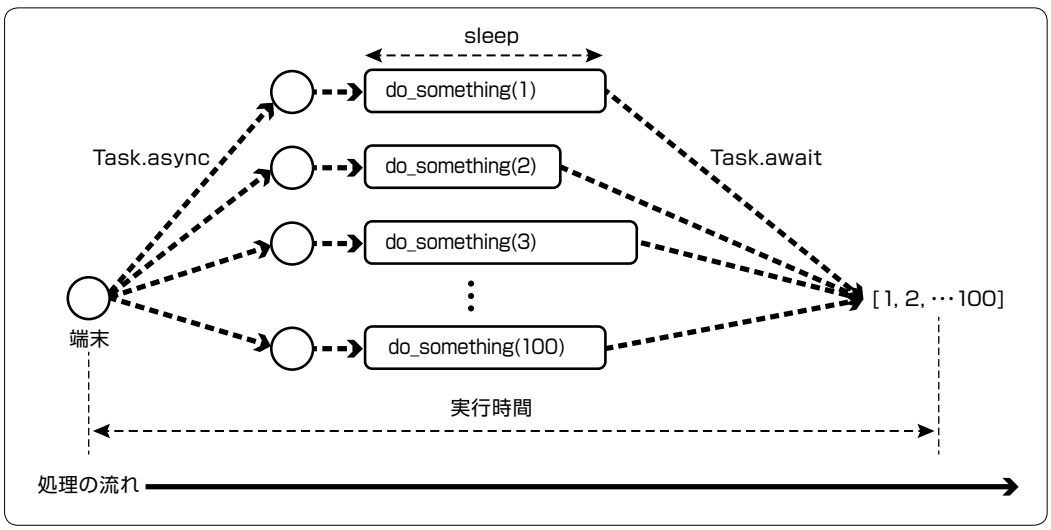
今回作成するのは辞書アプリです。アプリの仕様として、次の操作ができるものとします。

(1) get(キー)で引数のキーに対応する値を取得し、対応する値がなければnilを返す
(2) put(キー, 値)で引数のキーに対応する値を設定する
(3) cleanupで設定された辞書をクリアする

Rubyによる辞書アプリ

ここではオブジェクト指向言語とElixirを比べてみます。Rubyによる実装は**リスト2**のとおりです。オブジェクト指向によるプログラミングの場合は、まず辞書を取り扱うオブジェクトを作成します(**図5♣**)。作成したオブジェクトに対してget、put、cleanupのメソッドを呼び出して、オブジェクト内部にある辞書(ハッシュ@my_dict)への操作を行います(**図6**)。

▼図4　並列実行のイメージ



▼リスト2　Rubyによる辞書アプリの実装(ruby_dict.rb)

```
class RubyDict
  def initialize
    @my_dict = {}
  end

  def get(key)
    @my_dict[key]
  end

  def put(key, val)
    @my_dict[key] = val
  end

  def cleanup
    # @my_dictを{}に設定する事で初期化
    @my_dict = {}
  end
end
```

▼図5　リスト2(ruby_dict.rb)の実行結果

```
$ irb
irb(main):001:0> load 'ruby_dict.rb'
=> true
辞書オブジェクトを作成――♣
irb(main):002:0> d = RubyDict.new
=> #<RubyDict:0x007f847c83a8d0 @my_dict={}>
irb(main):003:0> d.get(:a)
=> nil
irb(main):004:0> d.put(:a, 1)
=> 1
irb(main):005:0> d.get(:a)
=> 1
@my_dictを{}に設定することで初期化
irb(main):006:0> d.cleanup
=> {}
irb(main):007:0> d.get(:a)
=> nil
irb(main):008:0>
```

Elixirによる辞書アプリ

一方、Elixirでは辞書アプリを、プロセスを中心に考えて実装します(**リスト3**)。Elixirは状態を管理するプロセスを扱う手段として、Agentモジュールを提供しています[注4]。

まず初期状態として、空のハッシュを持つAgentプロセスAgent.start_linkを作成します(**図7★**)。そのプロセスに対してget、updateのメッセージをプロセスに対して送信すると、辞書プロセスが内部のハッシュを更新し、その結果をメッセージとして送信元プロセスに返却します(**図8**)。

mixを使ってプロジェクトを管理

mixはElixirのビルドツールです。mixはプロジェクトの作成、ソースのコンパイル、アプリケーションのテストといったさまざまなプロジェクトの管理機能(タスク)を提供します。mixの標準タスクは、mix helpを実行することで確認できます(**図9**)。

本節では、前節で作成した辞書アプリ(ex_dict.ex)をmixプロジェクトとして作成していきます。まず手始めに、mix newコマンドでプロジェクトを作成しましょう(**図10**)。

▼リスト3　Elixirによる辞書アプリの実装(ex_dict.ex)

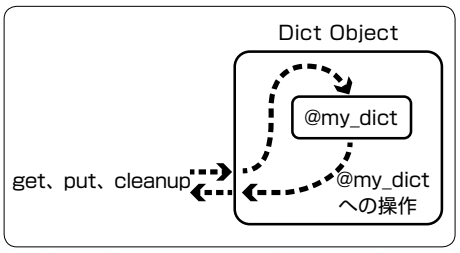
```
defmodule ExDict do
  # Agentを起動
  def start_link do
    Agent.start_link(fn -> %{} end, name: __MODULE__)
  end

  # Agentを停止
  def stop do
    Agent.stop(__MODULE__)
  end

  def get(key) do
    Agent.get(__MODULE__, &Map.get(&1, key))
  end

  def put(key, val) do
    Agent.update(__MODULE__, &Map.put(&1, key, val))
  end

  def cleanup do
    # Agentプロセスの状態を%{}に設定する事で初期化
    Agent.update(__MODULE__, fn _ -> %{} end)
  end
end
```

▼図7　リスト3の実行結果

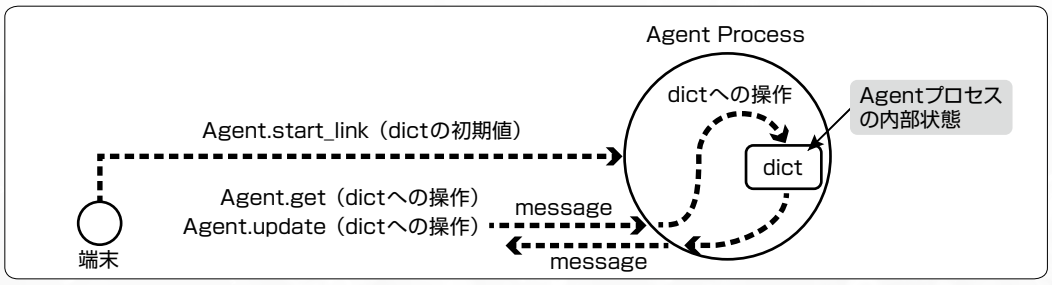
```
$ iex ex_dict.ex
iex(1)> ExDict.start_link ← 辞書プロセスを作成――★
{:ok, #PID<0.86.0>}
iex(2)> ExDict.get(:a)
nil
iex(3)> ExDict.put(:a, 1)
:ok
iex(4)> ExDict.get(:a)
1
Agentプロセスの状態を%{}に更新することで初期化
iex(5)> ExDict.cleanup
:ok
iex(6)> ExDict.get(:a)
nil
iex(7)>
```

注4) URL http://elixir-lang.org/docs/stable/elixir/Agent.html

▼図6　オブジェクトベースのプログラム



▼図8　プロセスベースのプログラム



ディレクトリ構成は(**図11**)となります。また、-S mixオプションを付けることで、プロジェクト内でiexを実行できます(**図12**)。

OTPの上にアプリを建てる

それでは、ビヘイビアを使ってOTPに則った辞書アプリをプロジェクトに組み込んでいきましょう[注5]。使用するビヘイビアは、アプリケーションの起動・停止の振る舞いを提供するApplicationビヘイビアと、プロセスの監視と再起動を行うSupervisorビヘイビアです。

アプリケーションビヘイビア

アプリケーションの起動のために、Applicationビヘイビアを組み込んでいきましょう。ビヘイビアを使うには、

・useビヘイビアの宣言

・ビヘイビアが要求するコールバックの実装

を行う必要があります。Applicationビヘイビアは、コールバックとしてstart(type,args)の実装を要求します[注6]。

今回はMixSampleモジュールに対してアプリケーションビヘイビアを組み込んでいきます。lib/mix_sample.ex(**リスト4**)でuse Applicationを宣言し(❶)、start(type,args)を実装します(❷)。なお、後述するSupervisorの起動(❸)をこのstartの中で行いますが、今は暫定として自身のプロセスIDを返却する{:ok, self}を有効にし、後ろのSup.start_linkをコメントアウトしてください。

また、mix.exsのapplicationに、mod:オプションを追加することで(**リスト5**♠)、プロジェクトの起動時に、指定したモジュール、引数で

注5) OTP、ビヘイビアの詳細については、前編(本誌2016年11月号)を参照してください。

注6) URL http://elixir-lang.org/docs/stable/elixir/Application.html#c:start/2

▼図9　mix helpの実行結果

```
$ mix help
mix                 # Runs the default task (current: "mix run")
mix app.start       # Starts all registered apps
mix app.tree        # Prints the application tree
...(中略)...
mix test            # Runs a project's tests
mix xref            # Performs cross reference checks
iex -S mix          # Starts IEx and runs the default task
```

▼図10　mix newの実行結果

```
$ mix new mix_sample
* creating README.md
...(中略)...
Your Mix project was created successfully.
You can use "mix" to compile it, test it, and more:

    cd mix_sample
    mix test

Run "mix help" for more commands.
```

▼図11　mixプロジェクトのディレクトリ構成

```
mix_sample
├── README.md
├── config ← アプリ固有の設定ファイル
│   └── config.exs
├── lib ← プロジェクトのソースコード
│   └── mix_sample.ex
├── mix-files.txt
├── mix.exs ← プロジェクトの設定ファイル
└── test ← テストコード
    ├── mix_sample_test.exs
    └── test_helper.exs
```

▼図12　iex -S mixの実行結果

```
$ iex -S mix
Compiling 1 file (.ex)
Generated mix_sample app
Interactive Elixir (1.3.4) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)>
```

アプリケーションを起動できます。再度、iex -S mixでプロジェクトを起動します(**図13**)。

スーパバイザビヘイビア

次にスーパバイザを作成しましょう。スーパバイザとは、特定のプロセスを監視し、そのプロセスがクラッシュしたら再起動を行うしくみです[注7]。

lib/sup.exとしてスーパバイザモジュールを作成します(**リスト6**)。use Supervisorを宣言し(①)、コールバックとしてinitを(②)、起動のためにstart_linkを(③)実装しています。スーパバイザでは、クラッシュした場合の再起動の設定をstrategyオプションで指定し(④)[注8]、監視対象のプロセスの起動方法(モジュール、関数、引数)を指定します(⑤)。

このスーパバイザの起動(Sup.start_link)を、前節のMixSample.startの中で実行することで(**リスト4**、❸)、アプリ起動時にスーパバイザが起動し、監視対象のプロセスが立ち上がります。start関数のSup.start_linkを有効にし、その前の{:ok, self}をコメントアウトしてください。

辞書の組み込み

それでは準備が整ったので辞書をアプリに組み込んでいきましょう。

前節で作成した辞書アプリ(**リスト3**、ex_dict.ex)をlib/ex_dict.exとして配置し、supervisorの監視対象(ワーカー)としましょう(**リスト6**、④)。なお、Supervisorの監視対象としてモジュールを組み込んだ場合、起動時に呼び出される関数はデフォルトでstart_linkとなります。

スーパバイザによるプロセスの再起動

もし辞書プロセス(ExDict)が何らかの理由でクラッシュした場合はどうなるでしょう？

当然、辞書への問い合わせと登録はできません。しかしご心配なく。スーパバイザによって辞書プロセスは監視されていますので、辞書プ

▼リスト4　lib/mix_sample.ex

```
defmodule MixSample do
  use Application # ビヘイビアの宣言――❶

  # callbackの実装――❷
  def start(_type, _args) do
    # 起動時に何も実行しない時は以下のように記述
    #{:ok, self} # selfは自身のプロセスID

    # 起動時にスーパバイザを起動する時は以下のように記述
    Sup.start_link # スーパバイザの起動――❸
  end
end
```

▼リスト5　mix.exs

```
defmodule MixSample.Mixfile do
  use Mix.Project

  def project do
    [app: :mix_sample,
     version: "0.1.0",
     elixir: "~> 1.3",
     build_embedded: Mix.env == :prod,
     start_permanent: Mix.env == :prod,
     deps: deps()]
  end

  def application do
    [
# プロジェクト起動時にMixSampleを引数[]で起動――♠
      mod: {MixSample, []},
      applications: [:logger]
    ]
  end

  defp deps do
    []
  end
end
```

▼図13　アプリケーションの起動

```
$ iex -S mix
Compiling 1 file (.ex)
Generated mix_sample app
Interactive Elixir (1.3.4) - press Ctrl+C to exit (type h() ENTER for help)
```

注7) URL http://elixir-lang.org/docs/stable/elixir/Supervisor.html#c:init/1

注8) URL http://elixir-lang.org/docs/stable/elixir/Supervisor.html#module-strategies

▼リスト6　lib/sup.ex

```
defmodule Sup do
  use Supervisor # ビヘイビアの宣言――①

  # スーパバイザの起動――③
  def start_link do
    Supervisor.start_link(__MODULE__, [], name: Sup)
  end

  # スーパバイザの起動設定――②
  def init([]) do
    # スーパバイザによる監視対象――④
    children = [worker(ExDict, [])] # ExDict.start_linkでワーカーが起動される
    supervise(children, strategy: :one_for_one) # 再起動戦略――⑤
  end
end
```

ロセスがクラッシュした場合はスーパバイザが再起動してくれます[注9]。

それでは、ExDict.stopで、辞書プロセスを明示的に停止してみましょう(**図14**)。やりました、スーパバイザが辞書プロセスを再起動するので、辞書への問い合わせが続行できていますね。

Phoenixとは

PhoenixはElixirで実装されたWebアプリケーションフレームワークです。この節ではPhoenixによるミニブログアプリ作成を通して、Phoenixの使い方を紹介します。Phoenixのアプリケーションもmixプロジェクトの規約に則っていますので、基本的な操作はmixプロジェクトと同様に行えます。

Phoenixの特徴

PhoenixはRailsの影響を強く受けており、「高い生産性とパフォーマンス」「チャネル[注10]を使ったWebSocketベースのリアルタイム通信を簡単に行える」といった特徴があります。また、さまざまなジェネレータがmixタスクとして提供されているので、高速にアプリケーション作成が行えます。

Phoenixでアプリを作ろう

本節ではPhoenixを使ったWebアプリを作成します。まずはPhoenixまわりの環境を整えましょう。

Phoenixのインストール

Elixirはパッケージ管理をhex[注11]で行います。Phoenixもhexによって管理・インストールするので、まずはhexをインストールしましょう。hexはmixタスクからインストールできます(**図15**)。

▼図14　プロセスの再起動

```
$ iex -S mix
iex(1)> ExDict.put(:a,1)
:ok
iex(2)> ExDict.get(:a)
1
iex(3)> ExDict.stop ← 辞書プロセスを停止
:ok
iex(4)> ExDict.get(:a)   ← スーパバイザが辞書プロセスを再起動、ただしプロセス内の辞書の状態は初期値になる
nil
iex(5)> ExDict.put(:a,2)
:ok
iex(6)> ExDict.get(:a)
2
iex(7)>
```

注9）Elixirではプロセスが壊れたとき、プロセスがリカバリを試みるのではなく、一度クラッシュさせてそれを監視するプロセスで再起動させるというのが、良いアプローチとされています(Let it crash)。

注10）URL http://www.phoenixframework.org/docs/channels

注11）URL https://hex.pm/

次に、phoenixのパッケージをmixタスクでローカルにインストールしてください(**図16**)。インストールされるとmix helpコマンドでphoenixのタスクが表示されるようになります。

Node.jsとMySQLを準備

なぜNode.jsをインストールするのか？と疑問に思う方がいるかもしれません。Phoenixは Node.jsで書かれたビルドツールであるbrunch.io[注12]を使って、JavaScriptコードのビルド・変換・ミニファイや、CSS・画像ファイルといったアセットの処理、ページのリロードといった処理を行うので、Node.jsが必要となります。また本アプリでは、データはMySQLに保存するものとします。

お使いの環境に合わせ、Node.jsとMySQLをローカルにインストールしてください。

注12) URL http://brunch.io/

ブログアプリケーションを作る

それではPhoenixでミニブログアプリを作成していきましょう。作成するブログは**表1**のテーブルを持ち、ブログの記事データに対してCRUDの操作を行えるものとします。

phoenixプロジェクトの作成

PhoenixはデフォルトのデータベースとしてPostgreSQLを採用しています。今回はデータベースにMySQLを使うので、--database mysqlオプションを付けてPhoenixプロジェクトを作成します(**図17**)。途中、「Fetch and in

▼表1　アプリのデータを保存するblogsテーブル

カラム名	データタイプ	説明	備考
id	bigint	内部ID	phoenixが自動生成
title	varchar	blogタイトル	ユーザが定義
body	varchar	blog本文	ユーザが定義
inserted_at	datetime	レコード作成日時	phoenixが自動生成
updated_at	datetime	レコード更新日時	phoenixが自動生成

▼図15　hexのインストール

```
$ mix local.hex ← hexをローカルにインストール
Are you sure you want to install archive "https://repo.hex.pm/installs/1.3.0/hex-0.13.2.ez"? [Yn] Y
* creating /Users/ohara_tsunenori/.mix/archives/hex-0.13.2
$ mix hex ← hexの情報を表示
Hex v0.13.2
Hex is a package manager for the Erlang ecosystem.
...(後略)...
```

▼図16　phoenixのインストール

```
ローカルにインストール
$ mix archive.install https://github.com/phoenixframework/archives/raw/master/phoenix_new.ez
Are you sure you want to install archive "https://github.com/phoenixframework/archives/raw/マ
master/phoenix_new.ez"? [Yn] Y
* creating /Users/ohara_tsunenori/.mix/archives/phoenix_new
$ mix help | grep phoenix ← phoenixパッケージの確認
mix local.phoenix     # Updates Phoenix locally
mix phoenix.new       # Creates a new Phoenix v1.2.1 application
```

▼図17　phoenixプロジェクトの作成

```
$ mix phoenix.new blog_sample --database mysql
* creating blog_sample/config/config.exs
...(中略)...
Fetch and install dependencies? [Yn] Y
* running mix deps.get
* running npm install && node node_modules/brunch/bin/brunch build
...(後略)...
```

▼リスト7　DBの設定（blog_sample/config/dev.exs）

```
# Configure your database
config :blog_sample, BlogSample.Repo,
  adapter: Ecto.Adapters.MySQL,
  username: "root", # インストールしたMySQLのユーザー名を設定
  password: "", # インストールしたMySQLのパスワードを設定
  database: "blog_sample_dev",
  hostname: "localhost",
  pool_size: 10
```

stall dependencies? [Yn]」と関連パッケージとnpmモジュールのインストールを行うか聞かれるので、Yを入力してインストールしてください。

blog_sample/config/dev.exsにMySQLの設定項目（**リスト7**）があるので、前節でインストールしたMySQLのユーザ名とパスワードを設定してください。これでPhoenixからMySQLに接続できるようになるので、次のようにmixタスクでアプリが使用するデータベースを作成します。

```
$ mix ecto.create
The database for BlogSample.Repo has been ↵
created
```

準備が整いました。それではアプリを起動して（**図18**）、ブラウザで「http://localhost:4000」にアクセスしてみましょう。不死鳥の画面が確認できます（**図19**）。

本稿で使用したPhoenixのアプリは、本誌サポートページ[注13]からダウンロードできます。そちらを実行する際は、**図20**を実行してください。

注13）URL http://gihyo.jp/magazine/SD/archive/2016/201612/support

ブログページの作成

それではブログのページを作っていきましょう。PhoenixのHTMLジェネレータを使って**図21**のように作成します。関連ファイルの作成メッセージのあとに、ルーティングの指定とマイグレーションの実行を行う旨のメッセージが表示されていますので、それぞれ設定・実行しましょう（**リスト8**、**図22**）。

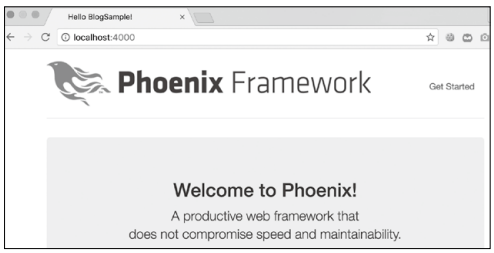
以上です。たったこれだけの操作でミニブログができました。

CRUD操作

それでは動作確認をしてみましょう。

`mix phoenix.server`でアプリを起動して、ブラウザで「http://localhost:4000/blogs」を開いてみてください（**図23**）。［New blog］リンクからブログ記事の投稿してみましょう（**図24**）。右の［Show］［Edit］［Delete］ボタンから、投稿したブログ記事の参照、編集、削除の操作ができ

▼図19　アプリの起動を確認



▼図18　アプリの起動

```
$ mix phoenix.server
[info] Running BlogSample.Endpoint with Cowboy using http://localhost:4000
17 Oct 05:39:48 - info: compiled 6 files into 2 files, copied 3 in 1.1 sec
[info] GET /
[debug] Processing by BlogSample.PageController.index/2
  Parameters: %{}
  Pipelines: [:browser]
[info] Sent 200 in 47ms
```

▼図20　サンプルコード実行のためのコマンド

```
$ mix deps.get && npm install && node node_modules/brunch/bin/brunch build
```

ますので、試してみてください。

いかがでしょう。Rails同様、簡単にWebアプリが作成できることを実感できたのではないでしょうか？

まとめ

本記事では、Elixirのプロセスをもとにしたアプリ作成とmixによるプロジェクト管理、Phoenixによるミニブログ作成についての説明を行いました。Elixirについてさらに詳しく知りたい方はぜひ『プログラミングElixir』[注14]を、Phoenixについては『Programming Phoenix』[注15]を読んでみてください。

前編・後編と2回に分けElixirについて紹介させていただきました。日本でもElixir/Phoenixの採用事例は増えつつあり、来年2017年4月には「ElixirConfJapan」が開催されるなど、盛り上がりを見せています。しかし残念ながら、日本語のElixirに関する情報はまだまだ少ないのが実情です、本記事がElixirの利用にあたり読者のみなさんの一助になれば幸いです。SD

▼リスト8　ルーティングの追加（router.ex）

```
defmodule BlogSample.Router do
  use BlogSample.Web, :router
  ...(中略)...
  scope "/", BlogSample do
    pipe_through :browser # Use the default browser stack

    get "/", PageController, :index
    resources "/blogs", BlogController # blogsのルーティングを追加
  end
  ...(中略)...
end
```

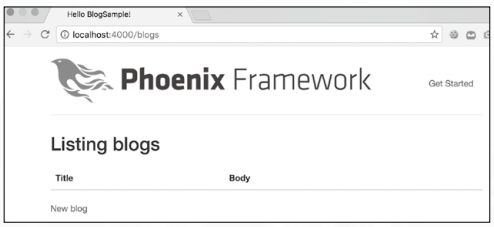
▼図21　ブログページの作成

```
$ mix phoenix.gen.html Blog blogs title:string body:string
...(中略)...
Add the resource to your browser scope in web/router.ex:
    resources "/blogs", BlogController ← ルーティング追加の設定
Remember to update your repository by running migrations:
    $ mix ecto.migrate ← マイグレーションの実行方法
```

▼図22　マイグレーション実行

```
$ mix ecto.migrate
05:53:53.535 [info]  == Running BlogSample.Repo.Migrations.CreateBlog.change/0 forward
05:53:53.536 [info]  create table blogs
05:53:53.547 [info]  == Migrated in 0.0s
```

▼図23　ブログ記事一覧

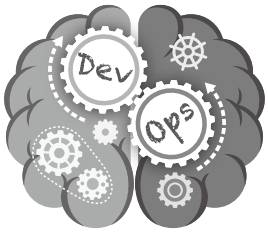


▼図24　ブログ記事作成

注14）Dave Thomas 著／笹田耕一・鳥井 雪 共訳，オーム社，2016年，ISBN=978-4-274-21915-3　URL http://shop.ohmsha.co.jp/shopdetail/000000004675

注15）Chris McCord, Bruce Tate, José Valim　著，Pragmatic Bookshelf,2016年，ISBN=978-1-68050-145-2　URL https://pragprog.com/book/phoenix/programming-phoenix

アプリエンジニアのための［インフラ］入門



Author 出川 幾夫（でがわ いくお） レバレジーズ株式会社 teratail開発チーム Twitter @ikuwow

最終回 インフラ設計入門

連載最終回で取り上げるのはインフラの設計、とくに実サービスへの展開にも耐えうるための分散・冗長化です。Webサーバ、ロードバランサ、DBサーバの設計について、それぞれ気を付けるべきことを解説していきます。

インフラの分散・冗長化

Webシステムを動かすサーバやそのほかインフラの構成を設計する際は、負荷の大きさや通信の流れなど、さまざまなことを考慮しながら設計する必要があります。アクセスが増えるなどの影響で処理量が大きくなるにつれて、負荷の分散や冗長化を進めていくのが一般的です。

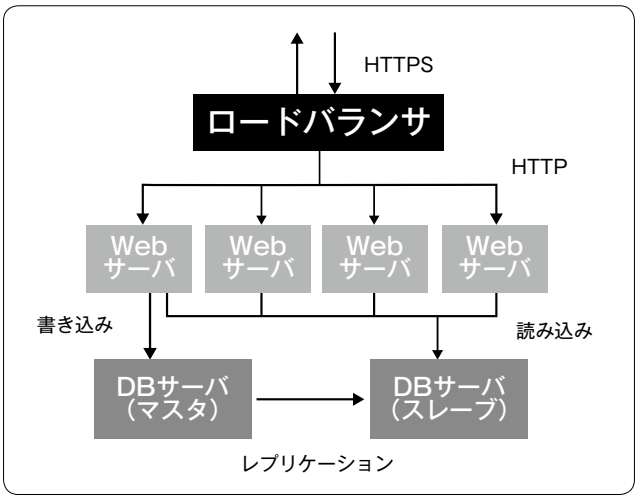
今回は典型的なWebアプリケーションのインフラ構成を1つ例として取り上げ、構成の内容や設計にあたって考えるべきことについて説明します。

典型的なWebアプリケーションのインフラ構成

今回挙げる例は**図1**のような構成になります。ユーザからのリクエストをHTTPS（TCP443番ポート）で待ち受けるシステムを想定しています。ユーザからのリクエストはまずロードバランサへ送られます。そこから複数のWebアプリケーションサーバ（Webサーバ）へリクエストが分散されます。

それぞれのWebサーバは、リクエストに応じてレスポンスを作成するための処理を行います。その過程で、データベースサーバ（DBサーバ）へ読み取りや書き込みの処理を行います。DBサーバにはマスタとスレーブという2つの

▼図1 典型的なWebアプリのインフラ構成の例



役割のサーバがあり、マスタからスレーブにデータが同期(レプリケーション)されています。Webサーバはデータの読み込み処理はスレーブに送りますが、書き込み処理はマスタに送るようになっています。

このようなインフラ構成の基本的な考え方は冗長化と呼ばれます。Webのシステムは24時間365日稼働していることが前提になっている場合が多く、数分、場合によっては数秒でもサービスが停止するのは大きな問題となります。このため、同様のサーバを複数設けて単一障害点を少なくし、サービスの停止やパフォーマンスの低下、データ損失などのリスクを小さくする必要があります。

Webサーバを冗長化

Webサーバは、リクエストに応じてビジネスロジック[注1]を実行してレスポンスを作成する処理を行う、インフラの中でも中核となる部分です。NginxやApacheなどのミドルウェア、PHPやRuby、Javaなどのサーバサイド言語によって処理が行われます。

同一サーバを複数配置

サービスのトラフィックが増えるにつれてまずはじめに負荷が上がるのがこのWebサーバですので、**図1**では複数台の構成にしています。負荷に応じてサーバの数を変化させるオートスケールのしくみを作ることもあります。これら複数のWebサーバは基本的に同一のものにする必要があります。サーバのスペックや、OS、ミドルウェアの種類、バージョン、アプリケーションのコードなど、すべてを同一にするのが基本です。そのため、サーバの設定が再現可能ですぐに作りなおせるようなしくみを作っておく必要があります。

注1) データの保持やリクエストのやりとりなどを除いた、アプリケーション内部の固有の処理のこと。

スムーズなデプロイ

非常に重要なのがアプリケーションのデプロイの方法です。デプロイは毎週、毎日と頻繁に行うものですので、それらが複数のサーバにスムーズに配布され、動作可能な状態になるしくみを作る必要があります。当然、一定以上の自動化は必須です。複数のサーバに同時に変更を反映するために、あらかじめデプロイが完了したサーバを構築しておいて、古いコードのあるサーバとすげ替えて瞬時に切り替えを完了するように構成することもあります。

インフラを使い捨てる

データベースやキャッシュなど、永続的に保持すべきデータをWebサーバにまったく置かず、ビジネスロジックの処理だけに集中させるのが理想です。またデプロイなどのたびにサーバなどのインフラを使い捨て、更新のたびに作りなおす考え方をイミュータブルインフラストラクチャと言います。これが実現できると、再現性が常にテストされ、同一の環境を構築して検証を行うことなどが非常に楽になります。構成管理の際に冪等性(べきとうせい)を考慮する必要がないのも大きな利点です。

ロードバランサを設置

Webサーバを冗長化したら、ユーザからのリクエストをそれらに振り分ける必要があります。これを行うのがロードバランサです。ミドルウェアではLVSやPoundなどが有名ですが、先述したApacheやNginxをロードバランサとして扱うこともできます。

ロードバランサの役割

ロードバランサはL4スイッチとL7スイッチの2つに分類できます。L4、L7とはOSI参照モデルの階層のことで、L4はトランスポート層、L7はアプリケーション層を表します。

L4スイッチはトランスポート層プロトコル(HTTP通信の場合はTCP)を終端し、ユーザ

⇔ロードバランサ、ロードバランサ⇔Webサーバでそれぞれ別のコネクションを張ります。この場合、HTTP通信で行うというのは共通です。

L7スイッチではアプリケーション層プロトコル(今回はHTTP(S)通信)を終端するので、HTTPのヘッダの内容を修正でき、通信内容を見て柔軟な振り分けができます。またレスポンスをロードバランサに返すか、ユーザに直接返すかなどの違いもあります。

ロードバランサには単にリクエストを振り分けるだけでなく、付随するさまざまな処理を行えます。とくにバランシングに付随する機能として、振り分け先のWebサーバが正しく動作しているかを確認するヘルスチェックを行う機能も併せ持っていることが多いです。これにより、正しく動作していないサーバを検出したときに、そのサーバへは振り分けないという制御ができます。

図1では、HTTPSのSSL暗号化通信を終端し、HTTPリクエストに変換する機能も持たせています。こうすることで、ロードバランサ以降のサーバなどを閉じたネットワークに置けば、以降の通信にHTTPSを利用する必要がなくなり、証明書を配布したりする必要性がなくなります。

リバースプロキシとは

ロードバランサとWebサーバの間にリバースプロキシというものを設けることもあります。リバースプロキシはユーザからの通信のインターフェースとして、HTTPの通信を最適化するためのさまざまな役割を担います(**図2**)。

ロードバランサのようにHTTPS通信を終端するほか、

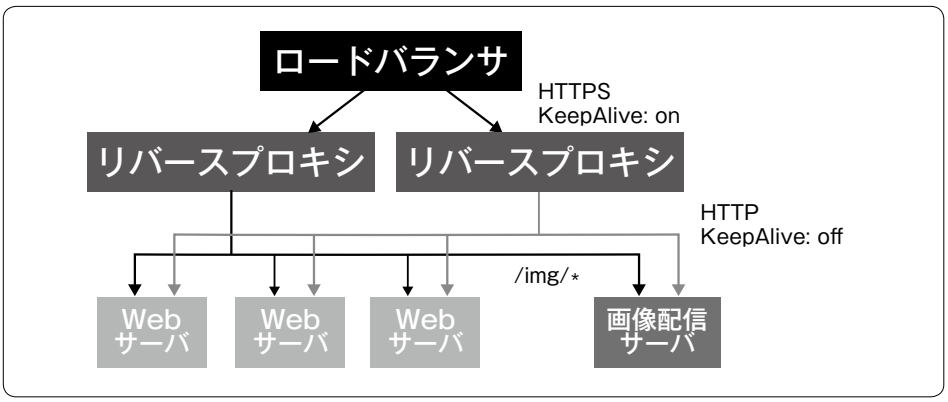
- /hogehoge/ikuwow/1のようなURLを/hogehoge?user=ikuwow&page=1に書き換える
- KeepAlive: onの通信を終端してWebサーバにはKeepAlive: offで通信を行う
- コンテンツのGZIP圧縮

など、HTTP領域で制御できることなら、かなり柔軟な操作ができます。

Webアプリケーションのビジネスロジックを実行する部分はCPUやメモリなどの多くのリソースを消費する一方、これらの単純な変換は1リクエスト当たりのリソース使用量はそこまで大きくありません。このため、たとえばリバースプロキシではキャッシュした静的リソースをWebサーバを利用せずに返すなど、役割分担をさせることでリソース仕様の効率化とパフォーマンス改善が図れます。

リバースプロキシのうしろには複数台のアプリケーション・サーバを置くことが多いので、実質的にロードバランサとしての機能も担うことが多いです。Amazon Web Services(AWS)やGoogle Cloud Platform(GCP)のロードバラ

▼図2　リバースプロキシを用意した例



ンサ機能を見ても、さまざまな機能が使えるようになっており、2つの呼び名や役割の境目はあいまいになっていると感じます。

ロードバランサやリバースプロキシは、その実装や製品によってできることがさまざまです。また2つを明確に分けて考えることもあれば、同一として扱う場合も多くややこしい部分です。構築の際は何が必要かをきちんと把握してから、それを実現できる適切なツールを選ぶようにしましょう。

DBサーバを冗長化

DBサーバの構成は、マスタとスレーブという2種類を設けることが一般的です。スレーブはマスタのデータと同期して常に同じデータを保持するようになっており、このしくみをレプリケーションといいます。

DBサーバは使い捨てにしない

この構成の冗長化には、負荷の分散に加えてバックアップの目的もあります。DBサーバはWebサーバやロードバランサと違い、ユーザの利用するデータを永続的に保持する部分ですので、基本的に使い捨てをしない構成にします。

マスタ・スレーブ構成にする理由

Webサーバのように、すべてが同一の構成となっておらず非対称なのは、書き込み処理(CREATE、UPDATE、DELETE)はデータの不整合を起こす可能性があるからです。もし同じデータに対して複数の更新処理が同時に行われてしまうとこれらは衝突を起こし、正しい状態が保てなくなります。この場合はサービスの負荷が上がるに連れて、マスタは1台のままスレーブを複数台に増やしていくことが多いです。このため書き込み処理はマスタのみに集中させ、整合性を壊す恐れのない読み取り処理(SELECT)をスレーブに送ります。

一般的なWebアプリケーションでは、書き込み処理よりも読み取り処理のほうがずっとリクエスト数が多いので、ある程度の負荷まではこの構成でうまく負荷の分散ができます。

書き込み処理の負荷を分散するためには、機能ごとにDBを分割してマスタ1つに対してスレーブが複数の組をいくつか作る設計にしたり、シャーディングやクラスタリングを行う必要があります。また書き込み処理に限らず、全体の負荷を下げるためにキャッシュのためのサーバを用意するのも非常に効果的です。

そのほかの冗長化構成

図2に示した冗長化は一般的なもので、負荷によってはさらに冗長化をする必要があります。たとえば、ここではロードバランサが1つですので、これが故障するとサービスが停止する単一障害点となっています。

これを解消するなら、複数のロードバランサを用意し、ルータを使ってネットワークレベルで分散させるなどの方法があります。単にこれを行うと今度はルータが単一障害点となり、これが問題となるなら、VRRPプロトコルを利用した冗長化をするなどの対策を行うことになります。

データストレージの冗長化や、サーバ間のネットワークの冗長化など、サービスの高い信頼性が求められるほど使うリソースを増やしていくことになります。

冗長化構成を設計する際に考えること

このような冗長化構成を実際のサービスで作る際には、どのように考えたら良いでしょうか。今回示した例はほんの一例で、システムによって必要なものは大きく変わってきます。

見積もり

まず、トラフィックや負荷がどれぐらいになるかを見積もる必要があります。Webサーバであればリクエスト数やLoad Average、使用メモリ量などを見積もることが多いです。既存の構成がある場合は、そこから予測を行うと良

いでしょう。新しくサービスを立ち上げる際などは厳密に見積もりを行うことは難しいので、AWSやGCPなどのクラウドを利用して簡単にスケールできるような、変化に強いしくみを作るほうに力を注ぐのが得策です。最悪、たとえばロードバランシングをせずに1台のサーバにアプリケーションもDBも置くこともできなくはありませんが、限界がきた場合の変更が容易ではないので、Webとデータベースを分けるなど、役割の分担だけはしておいたほうが良いでしょう。

構成管理ツール

冗長化のためには同一のサーバを複数用意する必要があるので、ChefやAnsibleなどのプロビジョニングツールを使った構成管理は必須です。Dockerを利用してコンテナ単位で管理するのも良いでしょう。AWSやGCPなどでは、Terraformなどの構成全体のオーケストレーションツールなども非常に役に立ちます。これらの導入の際には、実験的に手動で部分部分の構築を行ってから、それをあらためてオーケストレーションツールのテンプレートに書き直して全体を再構築すると、作業がスムーズです。

クラウド特有の考え方

AWSやGCPなどのクラウドサービスには、これまで挙げてきたこと以外に、特有の考え方や設計のコツがあります。

多くのクラウドサービスにはゾーンという概念があります。これはデータセンターとほぼ同義の言葉です。たとえばAWSの場合、複数のアベイラビリティーゾーンにリソースを分散配置できる機能が、多くのサービスで利用できます。クラウドサービスは、高度に仮想化と冗長化がされているとはいえ、サービスが利用できない時間や障害がまったくないわけではありません。たとえば、**図1**のWebサーバなどを2つのアベイラビリティーゾーンに分けて配置することによって、より高い信頼性が得られます。

クラウド内で使えるサービスを必要に応じて利用するのも、サービスの冗長化と管理の簡略化に効果的です。AWSにおけるS3やGCPにおけるCloud Storageは非常に高い耐久性があり、ストレージが必要であればぜひとも使っておきたいところです。HTTPのリクエストを受け付けることもできるので、画像の配信や、ユーザがダウンロードする大容量のファイルの配置やバックアップに利用しても良いでしょう。使用した容量や通信量に応じて課金されるため、ストレージの残り容量や通信のキャパシティを気にする必要は基本的にありません。

まとめ

今回はWebアプリケーションのインフラ構成の全体について、かなり広い分野の内容をいれながら、冗長化のベストプラクティスや設計の際の考え方などを説明しました。

全体の概念がわかっていると、インフラの分野の技術やツールの理解の速さが違います。もちろん、Backend as a Service(BaaS)やPlatform as a Service(PaaS)で個々の機能やツールは異なりますが、基本的な考えは同じです。また最近は、複数のPaaSやBaaSを利用して構成するサーバレスアーキテクチャや、コンテナという単位で実行環境を小さく取り回すDockerなど、Webのインフラ管理には今までになかったものが次々と登場しており、変化のあるおもしろい分野になっています。

仕事や趣味で開発をしているWebシステムのインフラ構成について、現在の状態を整理したり、今後どうあるべきかを考えてみると、インフラをより良くする良い取っ掛かりになると思います。

◆ ◆ ◆

今回でこの『アプリエンジニアのための[インフラ]入門』の連載は最終回となります。今まで読んでくださりありがとうございました。SD

バックナンバーのお知らせ　BACKNUMBER

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

本誌バックナンバー（紙版）はお近くの書店に注文されるか、下記のバックナンバー常備取り扱い店にてお求めいただけます。また、「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）や、e-hon（http://www.e-hon.ne.jp）にて、Webから注文し、ご自宅やお近くの書店へお届けするサービスもございます。

2016年11月号

第1特集
クラウドコンピューティングのしくみ
AWS・Azure・SoftLayer・Heroku・さくらのクラウド

第2特集
レガシーコード改善実践録
サイボウズ流バグゼロまでの道のり

一般記事
・［次世代言語］Elixirの実力を知る【前編】
・Jamesのセキュリティレッスン

定価（本体1,220円+税）

2016年10月号

第1特集
Webサーバはなぜ動くのか?
HTTP、CGI、サーブレット、Node.js、Railsを一挙解説

第2特集
いますぐ始める本格派データベース
新しいPostgreSQLの教科書

一般記事
・CHIRIMENシングルボードコンピュータ入門
Webプログラミングで WoT サイネージ制作

定価（本体1,220円+税）

2016年9月号

第1特集
知りたい情報集まっていますか?
ログ出力のベストプラクティス

第2特集
良いPHP、悪いPHP
——すぐ効くWeb開発入門

一般記事
・「良いプログラム」のための「良いコメント」
コードを読みやすくするための6つの書き方

定価（本体1,220円+税）

2016年8月号

第1特集
GitHubさいしょの一歩
はじめてのPull Requestから、チーム導入へ

第2特集
案外知らなかった
YumとAPTのしくみと活用

一般記事
・Ruby on Railsへの導入でわかったRRRSpec
による分散テストの効果

定価（本体1,220円+税）

2016年7月号

第1特集
試して実感!
プログラマが知っておくべきTCP/IP

第2特集
手を動かして学ぼう正規表現入門
記事とWebツールでトレーニング

新連載
・アプリエンジニアのための［インフラ］入門

定価（本体1,220円+税）

2016年6月号

第1特集
速く堅実に使いこなすための
bash再入門

第2特集
RDBの学び方
MySQLを武器にSQLを始めよう!

巻頭特集
・Android Wearアプリ開発入門［特別編］
・フリーで始めるサーバのセキュリティチェック［後編］

定価（本体1,220円+税）

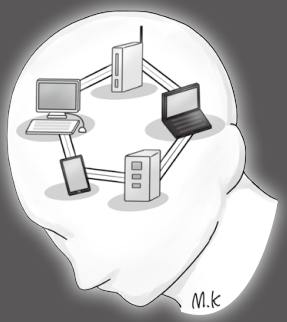
Software Design　バックナンバー常備取り扱い店							
北海道	札幌市中央区	MARUZEN & ジュンク堂書店　札幌店	011-223-1911	神奈川県	川崎市高津区	文教堂書店　溝の口本店	044-812-0063
	札幌市中央区	紀伊國屋書店　札幌本店	011-231-2131	静岡県	静岡市葵区	戸田書店　静岡本店	054-205-6111
東京都	豊島区	ジュンク堂書店　池袋本店	03-5956-6111	愛知県	名古屋市中区	三洋堂書店　上前津店	052-251-8334
	新宿区	紀伊國屋書店　新宿本店	03-3354-0131	大阪府	大阪市北区	ジュンク堂書店　大阪本店	06-4799-1090
	千代田区	書泉ブックタワー	03-5296-0051	兵庫県	神戸市中央区	ジュンク堂書店　三宮店	078-392-1001
	千代田区	丸善　丸の内本店	03-5288-8881	広島県	広島市南区	ジュンク堂書店　広島駅前店	082-568-3000
	中央区	八重洲ブックセンター本店	03-3281-1811		広島市中区	丸善　広島店	082-504-6210
	渋谷区	MARUZEN & ジュンク堂書店　渋谷店	03-5456-2111	福岡県	福岡市中央区	ジュンク堂書店　福岡店	092-738-3322

※店舗によってバックナンバー取り扱い期間などが異なります。在庫などは各書店にご確認ください。

デジタル版のお知らせ　DIGITAL

デジタル版Software Design 好評発売中！紙版で在庫切れのバックナンバーも購入できます

本誌デジタル版は「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）、「雑誌オンライン.com」（http://www.zasshi-online.com/）、「Gihyo Digital Publishing」（https://gihyo.jp/dp）で購入できます。最新号、バックナンバーとも1冊のみの購入はもちろん、定期購読も対応。1年間定期購読なら5%強の割引になります。デジタル版はPCのほかにiPad／iPhoneにも対応し、購入するとどちらでも追加料金を払うことなく、読むことができます。



仮想化の知識、再点検しませんか?
使って考える仮想化技術

本連載は「仮想化を使う中で問題の解決を行いつつ、残される課題を整理する」ことをテーマに、小さな仮想化環境の構築・運用からはじめてそのしくみを学び、現実的なネットワーク環境への実践、そして問題点・課題を考えます。仮想化環境を扱うエンジニアに必要な知識を身につけてください。

Author
笠野 英松(Mat Kasano)
有限会社 ネットワーク・メンター
URL http://www.network-mentor.com/indexj.html

第7回 ホストシステムと仮想環境の構築

後半スタート:仮想ネットワーク環境で使ってみよう

本連載の前半では「まずは使ってみよう」ということで、仮想化インフラKVMを使ってホストシステムや仮想環境の構築から、仮想マシン/ゲストシステムの利用までシンプルな環境で進めてきました。

後半では「現実的な使い方を考えよう」ということで、複数仮想マシン/ゲストシステムの仮想ネットワークと物理ホストを含む実ネットワークから構成される、ネットワーク環境での仮想化について考えてみます。

具体的には、ホストシステムや仮想環境の構築から仮想マシン/ゲストシステムの利用までを、ホスト-ゲスト間と仮想ネットワーク内、そして、ネットワーク間(仮想ネットワークと実ネットワークの間、そしてインターネット接続)という実際の仮想環境の中で作業を進め、問題点をクリアしながら課題を整理します。

また現実の環境では、インストールや運用管理などの作業を手作業で行うことは考えにくく、さまざまな自動化の手だてを考えなければなりません。

なお、本連載後半ではOSとしてCentOS 6.8を使用します。

今回行う作業

まず、仮想マシン構築のためのインフラを作

▼表1 CentOS(6.8)インストール項目

※インストールDVD #1/#2必要(CentOS-6.8-x86_64-bin-DVD1/2.iso)

画面/処理	設定値
インストール言語	⇒Japanese(日本語)
キーボード言語	⇒日本語
ストレージデバイスタイプ選択	⇒基本ストレージデバイス
ホスト名	⇒vhost1.example.com
ネットワーク設定	⇒アドレス「192.168.0.11」/ネットマスク「24」/ゲートウェイ「192.168.0.100」/DNS「192.168.0.100」
タイムゾーンの選択-都市	⇒アジア/東京、
UTC	⇒チェックオフ
root パスワード	⇒設定
パティション(領域)の作成方法	⇒カスタムレイアウト
パティション作成	⇒swap、3+2G=5120M/boot(ext4)、512/(ext4)40GB=40960M
ブートローダー	⇒ /dev/sda
ソフトウェアセットの選択	⇒Virtual Host
ソフトウェア選択のカスタマイズ方法	⇒今すぐカスタマイズ(記事末の表2参照)
ユーザー追加	⇒ユーザー名:user1、フルネーム:Test User No.1、パスワードおよび確認
日付と時刻/日時	⇒ネットワーク上で日付と時刻を同期化します
kdump	(デフォルト:オン)

※以降、表3へ続く

成します。①ホストシステムと仮想環境の構築、そして、②仮想ネットワーク環境とセキュリティ保護の設定です。

今回は前者の①ホストシステムと仮想環境の構築を行います。次回が②仮想ネットワーク環境とセキュリティ保護の設定の予定です。

ホストシステムの構築

仮想環境をのせる物理ホストシステムの構築です。今回からは実際の場でのシステム構築なので、この物理ホストにのせる仮想環境および、仮想ネットワークが物理ホストの実ネットワークとどのように関係し、どのように運用するかまで考えておかなければなりません。

OSのインストール項目

OSのインストール項目は**表1**のようなものですが、連載前半のようなGUI／対話型で手作業していては、現実の場では大変です。

ソフトウェアの選択構築

システム設計をするうえでは仮想環境のほかにも、システム機能として通常のサーバ環境を

▼リスト1　CentOS自動インストールのためのキックスタートファイル
※このks.cfgのサンプルファイルは、本誌Webサイトのサポートページ（http://gihyo.jp/magazine/SD/archive/2016/201612/support）からダウンロードできます。

```
# ks.cfg for Virtualization
autostep
#version=DEVEL
# Firewall configuration(★)
firewall --disabled
install
cdrom
# Root & 1 user(★) password
rootpw  --iscrypted $6$j3M33cInttK7hREq$9GDMWvR8QZD4.LnKUkxhrQXWMniHgYv/36.fMwpm/ (...略...)
user --name=user1 --password=password1
(...略...)
# Use graphical install(★)
graphical
# Disable Firstboot(★)
firstboot --disabled
(...略...)
# SELinux configuration(★)
selinux --disabled
# Installation logging level(★)
logging --level=info
# Network information
network --onboot yes --device eth0 --bootproto static --ip 192.168.0.11 --netmask ➥
255.255.255.0 --gateway 192.168.0.100 --noipv6 --nameserver 192.168.0.100 --hostname ➥
vhost1.example.com
(...略...)
# Clear the Master Boot Record(★)
zerombr
# Partition clearing information(★)
clearpart --all --initlabel
# Disk partitioning information(★)
part /boot --fstype=ext4 --size=512
part / --fstype=ext4 --size=40960
part swap --size=5120
#(★)repo --name="CentOS"  --baseurl=cdrom:sr0 --cost=100
# reboot after completion of install with eject CDROM(★)
reboot --eject
%packages
(...略...)
%end
```

▼図1 「openssl passwd」によるハッシュパスワードの作成

```
 1. ApacheタイプのMD5ハッシュ・アルゴリズムによるパスワード生成
[root@vhost1 ~]# openssl passwd -apr1
Password:
Verifying - Password:
$apr1$cii9S7Ho$x6h0nQ8R0IAgIkIIDtvIZ0

 2. MD5ハッシュ・アルゴリズムによるパスワード生成
[root@vhost1 ~]# openssl passwd -1
Password:
Verifying - Password:
$1$b4d7B8Nd$NL5MliHuIbz70lxHBOcic1

 3. Unixパスワード・アルゴリズムによるパスワード生成:8文字制限
[root@vhost1 ~]# openssl passwd
Password:
Verifying - Password:
Q5kzGfX49Tfc6
```

加え、また、付加的なサーバ追加のための開発環境も入れておきます。

記事末の**表2**は、CentOSインストール時のソフトウェアのカスタマイズリストです。

自動インストール

インストールは自動化します。

CentOSやRed Hat Enterprise Linuxでは自動インストール手段として「キックスタートインストール」があるので、これを使います。

キックスタートインストールはキックスタートファイル「ks.cfg」という、インストール中に入力が必要な項目についての回答を含んだ自動インストールのための手順ファイルを利用します。このファイルは複数のホストに共通で使用できます[注1]。

ks.cfgの作成

ks.cfgは、CentOSインストール後に/rootディレクトリ内に生成される「anaconda-ks.cfg」をもとに編集・作成することができますが、一部コメント化や追加がされているので、基本的なフォーマットルールを理解したうえで編集・作成するほうがよいでしょう。

今回使用したks.cfgは前ページの**リスト1**のようなものです。注意ポイント(anaconda-ks.cfgからの変更点)は**リスト1**の「★」印の行で、おもに次のような点です。

- 無効化設定(firewall、firstboot、selinux)
- 追加設定(ユーザ/平文パスワード、ログ、MBR初期クリア、ディスク初期クリア、終了時CDイジェクト)
- コメント化(repo)

このうち、ユーザのパスワードは平文にしていますが、**図1**のように「openssl passwd」を使ってできる、ハッシュパスワード文字列を値として「--iscrypted」を指定すればセキュリティを強くできます。

ks.cfgのインストールメディアへの組み込み

ks.cfgは、たとえば、DVDのトップディレクトリに入れ、そのパスを含むks.cfgによる自動インストールの設定は「isolinux」ディレクトリの中の「isolinux.cfg」に記載します(**リスト2**)。

なお、CentOSイメージをDVDだけではなく、ハードディスクやNFS、URL(ほかのHTTP/

注1) 参照:「Red Hat Enterprise Linux 6インストールガイド」第32章 キックスタートインストール

FTPサーバ)に置くこともできます。

DVD isoイメージの作成

isoファイルをCentOSで作成し(**図2**)、適当なシステムでDVD(#1)化します。

ホストシステムの自動構築

作成したDVD#1を使用してホストシステムを起動メニューAuto installで自動インストールします。なお、インストール途中(1267/1271完了時)メッセージ表示なくDVD#1がイジェクトされるので、DVD#2と入れ替えます。

インストール後の設定

インストール後、ホスト環境の設定調整を行います(**表3**)。

仮想環境の構築

仮想環境の主なものは「仮想マシンマネージャー」の「接続の詳細」にあるようにネットワークとストレージです。

ネットワークなどは次回に説明するとして、今回は仮想環境の基本のストレージを説明しま

▼リスト2　isolinux/isolinux.cfg追加行(ks.cfgによる自動インストールエントリの追加)

ファイル：isolinux/isolinux.cfgの最後に追加

```
label autoinstall
  menu label ^Auto install
  menu default
  kernel vmlinuz
  append ks=cdrom:/ks.cfg initrd=initrd.img
```

▼図2　ks.cfg組み込みISOイメージの作成手順

❶ CentOS 6.8 DVD#1のマウント

❷ CentOS 6.8 DVD用作業ディレクトリの作成

```
mkdir /root/CentOS_6.8U_Final
```

❸ CentOS 6.8 DVD#1オリジナルのコピー

```
cp -pfR /media/CentOS_6.8_Final/* /media/
CentOS_6.8_Final/.??* /root/CentOS_6.8U_Final/
```

❹ オリジナルDVDをイジェクト

```
eject cdrom
```

❺ CentOS 6.8 DVD用作業ディレクトリへ移動

```
cd /root/CentOS_6.8U_Final
```

❻ ks.cfg と isolinux.cfg の編集

```
vi ks.cfg
```
(リスト1参照)

```
vi isolinux/isolinux.cfg
```
(リスト2参照)

❼ ISO生成ディレクトリの作成

```
mkdir /data
```

❽ ISOイメージを/data/CentOS_6.8U_final.isoとして作成

```
mkisofs -input-charset utf-8 -v -l -R -J
 -joliet-long -o /data/CentOS_6.8U_Final.iso
 -b isolinux/isolinux.bin -c isolinux/
boot.cat -no-emul-boot -boot-load-size 4
 -boot-info-table .
```

❾ 確認

```
ls -al /data
```

▼表3　インストール後のホスト環境の設定調整

概要	設定内容
サービス停止	service NetworkManager stop
	service ip6tables stop
	service bluetooth stop
サービスのランレベル設定の変更(無効化)	chkconfig --level 35 ip6tables off
	chkconfig --level 35 bluetooth off
SELinux(セキュアOS機能)無効化	setenforce 0
	vi /etc/selinux/config："SELINUX=disabled"(SELinux無効化)
ネットワーク設定ファイルの修正と再起動	vi /etc/hosts(192.168.0.11 vhost1.example.com　vhost1)
	vi /etc/sysconfig/network-scripts/ifcfg-eth0(NIC起動設定)
	vi /etc/resolv.conf(レゾルバ設定)
	vi /etc/sysconfig/network(ネットワーク利用有無、ホストFQDN名、ゲートウェイIPアドレス)
	service network restart
su 使用可能ユーザの制限設定	vi /etc/pam.d/su－"auth　required　pam_wheel.so use_uid"(非コメント化)
	wheel グループにユーザ追加
	vigr－"wheel:x:10:root,ユーザ"
	vigr -s－"wheel:::root,ユーザ"

▼図3

▼図4

▼図5

▼図6　ストレージプール作成後のファイルシステムの状況

```
[root@vhost1 ~]# ls -al / | grep vm_storages マウントディレクトリ
drwxr-xr-x    3 root root  4096 10月 10 20:30 2016 vm_storages
[root@vhost1 ~]# ls -al /vm_storages ディレクトリの内容：未使用なので空
合計 24
drwxr-xr-x   3 root root  4096 10月 10 20:30 2016 .
dr-xr-xr-x. 28 root root  4096 10月 10 20:36 2016 ..
drwx------   2 root root 16384 10月 10 20:30 2016 lost+found
[root@vhost1 ~]# more /etc/mtab | grep vm_storages マウント情報
/dev/sda7 /vm_storages ext2 rw 0 0
```

す。連載前半ではデフォルトのディスクイメージファイルで構築しましたが、後半はさまざまなストレージを使用して構築します。

仮想マシンのストレージとしては、物理ディスクやファイルシステムのほかに、KVM/libvirtであらかじめ確保する、ストレージプールと呼ばれるストレージ領域内に割り当てることもできます。

ストレージプールには、ディスクボリューム、ディスクパーティション、ディスクディレクトリ、LVMボリューム、iSCSIボリューム、NFS、などさまざまな形態があります。

ストレージプールの作成

ここではストレージプールとして、ディスクパーティション(事前フォーマット済みブロックデバイス)を作成してみます。

あらかじめ、fdiskなどでパーティション(たとえば、/dev/sda7)を作成し、フォーマット(mkfs /dev/sda7)しておきます。

そのうえで、仮想マシンマネージャーの[編集]→[接続の詳細]→[ストレージ]でストレージプールを作成します。その手順が**図3、4、5**です。

作成した後のファイルシステムの状況は、**図6**のようになっています。ルートディレクトリにディレクトリが作成されていて/dev/sda7がここにマウントされています。

次回予告

現実のネットワーク環境に対応するためには仮想ネットワーク環境を整備し、ネットワーク接続のセキュリティ設定を確実にしなければなりません。また、運用管理も「すべて」管理者が行うのではなく、利用者と「分担」することも必要になります。そのための、しくみやセキュリティも考えておかねばなりません。次回はそうしたことがらについて考えてみます。SD

連載各回では、読者皆さんからの簡単な「ひとくち質問(QA)コーナー」や「何かこんなこともしてほしい要望(トライリクエスト)コーナー」を設けて「双方向連載」にできればと思っていますので、質問や要望をお寄せください。

宛先：sd@gihyo.co.jp
件名に、[仮想化連載]とつけてください

▼表2　ソフトウェアカスタマイズリスト（一般的なサーバ環境での仮想化設定を例として）

選択ソフトウェアセット	パッケージグループ（- 追加パッケージ）
Webサービス	PHPサポート　←追加選択のみ
	Webサーバー　←追加選択のみ
アプリケーション	インターネットブラウザ　←追加選択のみ
サーバー	CIFSファイルサーバー　←追加選択のみ
	FTPサーバー　←追加選択のみ
	サーバープラットフォーム　←デフォルト
	システム管理ツール　←追加選択のみ
	ディレクトリサーバー　←追加選択のみ
	ネットワークインフラストラクチャサーバー　←追加選択+カスタマイズ
	- bind - DNS（Domain Name System）サーバーBIND（Berkeley Internet Name Domain）はDNSプロトコル　★追加選択
	- bind-chroot - A chroot runtime environment for the ISC BIND DNS server, named(8)　★追加選択
	電子メールサーバー　←追加選択+カスタマイズ
	- sendmail - 広く普及しているメール転送エージェント（MTA）　★追加選択
	- sendmail-cf - Sendmailを再設定するために必要なファイル　★追加選択
システム管理	SNMPサポート　←追加選択のみ
	システム管理　←追加選択+カスタマイズ
	- watchdog - Software and/or Hardware watchdog daemon　★追加選択
デスクトップ	X Window System　←追加選択のみ
	グラフィカル管理ツール　←追加選択のみ
	デスクトップ　←追加選択+カスタマイズ
	- tigervnc-server - A TigerVNC server　★追加選択
	リモートデスクトップ接続クライアント　←追加選択+カスタマイズ
	- tigervnc - A TigerVNC remote display system　★追加選択
	レガシーX Winodwsシステムの互換性　←追加選択のみ
	汎用デスクトップ（GNOMEデスクトップ）　←追加選択のみ
データベース	MySQLデータベースサーバー　←追加選択のみ
ベースシステム	Javaプラットフォーム　←デフォルト
	Perlのサポート　←デフォルト
	コンソールインターネットツール　←デフォルト+カスタマイズ
	管理者用コンソールインターネットアクセスツール
	- ftp - 標準的なUNIX FTP（File Transfer Protocol）クライアント　★追加選択
	ディレクトリ接続クライアント　←デフォルト
	デバッグツール　←デフォルト+カスタマイズ
	- glibc-utils - GNU Cライブラリからの開発ユーティリティ　★追加選択
	ネットワーキングツール　←追加選択+カスタマイズ
	- arpwatch - ネットワーク上のIPアドレスを追跡するためのネットワーク監視ツール　★追加選択
	- nmap - ネットワーク調査ツールおよびセキュリティスキャナ　★追加選択
	- stunnel - SSL暗号化ソケットラッパ　★追加選択
	ネットワークファイルシステムクライアント　←デフォルト
	ハードウェア監視ユーティリティ　←デフォルト
	パフォーマンスツール　←デフォルト
	ベース　←デフォルト
	レガシーUNIXの互換性　←追加選択+カスタマイズ
	レガシーUNIX環境使用／移行のための互換性プログラム
	- dump - ファイルシステムのバックアップと復元のためのプログラム　★追加選択
	- telnet - telnetリモートログインプロトコル用のクライアントプログラム　★追加選択
	- telnet-server - telnetリモートログインプロトコル用のサーバープログラム　★追加選択
	互換性ライブラリ　←追加選択
	大規模システムのパフォーマンス　←デフォルト
仮想化	仮想化　←「ソフトウェアセット」で「Virtual Host」選択時デフォルト+カスタマイズ
	- qemu-kvm-tools - KVM debugging and diagnostic tools　★追加選択
	仮想化クライアント　←「ソフトウェアセット」で「Virtual Host」選択時デフォルト
	仮想化ツール　←追加選択+カスタマイズ
	- libguestfs-tools - System administration tools for virtual machines　★追加選択
	- virt-v2v - Convert a virtual machine to run on KVM　★追加選択
	仮想化プラットフォーム　←「ソフトウェアセット」で「Virtual Host」選択時デフォルト
言語	日本語のサポート（インストール言語=日本語、の場合デフォルト）を含む、各国言語のサポート選択
開発	サーバープラットフォーム開発　←追加選択のみ
	開発ツール　←追加選択+カスタマイズ
	- compat-gcc-34 - Compatibility GNU Compiler Collection　★追加選択
	- compat-gcc-34-c++ - C++ support for compatibility compiler　★追加選択

RDB性能トラブルバスターズ奮闘記

原案 生島 勘富(いくしま さだよし) info@g1sys.co.jp ㈱ジーワンシステム
構成・文 開米 瑞浩(かいまい みずひろ) イラスト フクモトミホ

第10回 「スケールアウトしにくいからJOIN禁止」はあまりにも短絡的

性能改善をするときには、どこがボトルネックになっているのかを正確に見極めないと、効果的な対策が行えないどころか、かえって事態を悪化させる場合があります。今回、大道君が遭遇した「ぐるぐる系SQLでのバッファプールの割り当て増強」や「JOIN禁止」という対策がまさにそんな悪手だったようです。

登場人物紹介

生島氏
DBコンサルタント。性能トラブルの応援のため大道君の会社に来た。

大道君
浪速システムズの新米エンジニア。素直さとヤル気が取り柄。

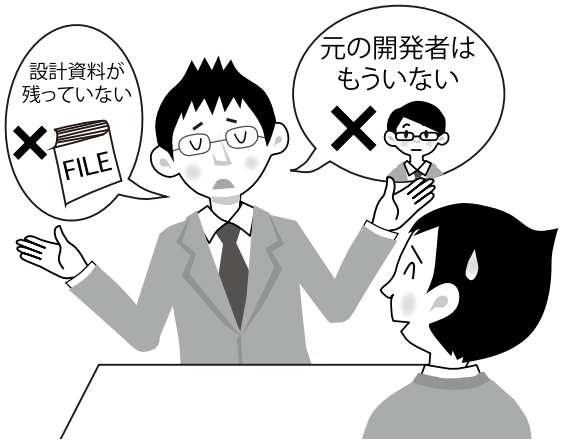
五代氏
大道君の上司。プロジェクトリーダーでもある。

開発元がギブアップしたシステムの改修依頼

大阪を中心に20年間システム開発に携わったあと、現在は東京で仕事をしている「SQLの伝道師」ことジーワンシステムの生島です。

ある日のこと、技術アドバイザーとしてかかわっている取引先の浪速システムズから、いつものようにヘルプの依頼がやってきました。

「また、性能改善依頼です。もともと他社で開発されたシステムなんですが、開発元がバンザイしてしまって、ウチに持ち込まれた案件なんです」と、浪速システムズの大道君。



「そらー、危ない予感でいっぱいやな。元の開発者はもうおらん、設計資料もろくにない、とかいう話とちゃうの？」

「どうしてわかるんですか!?」

「わかるわ！」

この業界ではよくあるパターンとはいえ、予想どおりでも全然うれしくありません。状況を整理するとざっと次のようになりました。

①ECサイトのシステム
②稼動当初は問題なかったが、データ件数が増えるにつれて遅くなっていった
③全体的に遅いが、とくに商品一覧や受注一覧などの一覧系の画面は遅い
④DBサーバのチューニングはいろいろとやってみたが、万策尽きた

「万策尽きた、って具体的に何やったんや？データベース(以下、DB)の設定いじってみた程度じゃないの？」

「物理メモリの増設、バッファプールの割り当て増強、CPUもグレードアップした、だそうです」

「それだけか、まあ、ありがちやけど……SQL

に手を入れようとするとアプリケーション(以下、アプリ)側のコード改修が必要になって、話が大ごとになるからそこまでやらないケースが多いんよね」

「あまり期待できないですか、これでは」

「大道君だったら、原因として何を疑う？」

「一覧系の画面が遅いということですから……真っ先に『ぐるぐる系』を疑いますね」

「うん、それが一番怪しい。でもそれが原因なら解決にはSQLの改良、つまりアプリのアルゴリズム変更が必要で、バッファプールいじっても意味ないんよ」

『ぐるぐる系』というのは当連載初期に何度か触れた、何重にもSQL文のループをまわすような処理のことを言います。たとえばcustomerテーブルにid=1から100までの全100件のデータがあるとして、`SELECT * FROM customer WHERE id=?;`というSQLの?を1から100まで変えて100回SQL文を発行するのが『ぐるぐる系』SQLです。それに対して、同じ結果を得るのに`SELECT * FROM customer ORDER BY id;`として1回のSQLで全データを取得する方式を本連載では一発系SQLと呼びます。

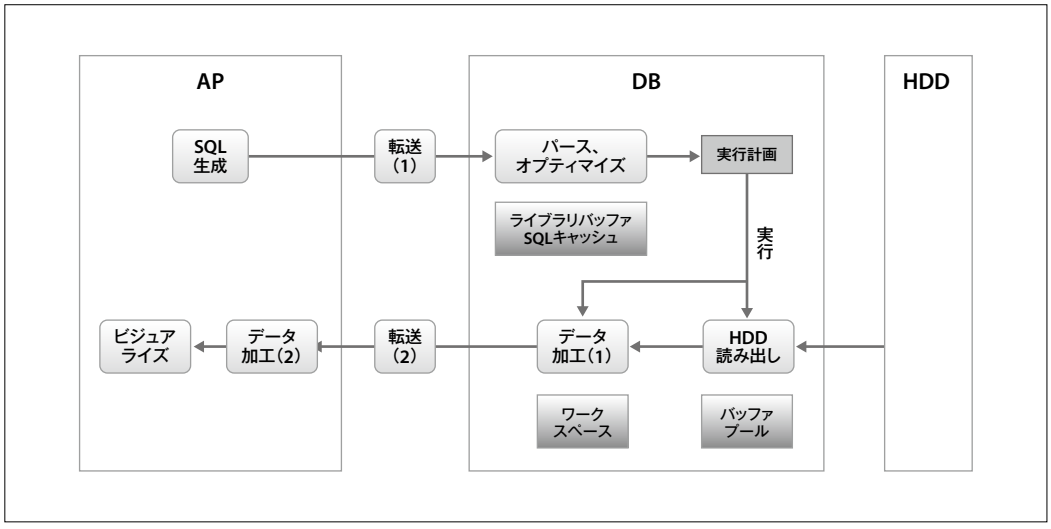
バッファプールが「ぐるぐる系」に影響しない理由とは?

ここでDBアクセスの性能にかかわる要因を少し整理してみましょう。**図1**はSELECT系のSQL文をアプリケーションサーバ(AP)からDBサーバ(DB)に投げて結果を得る場面のイメージです。

APでSQL文を生成してDBに転送(1)し、DBでパース、オプティマイズをかけて実行計画を生成、それを実行してハードディスク(以下、HDD)読み込みとデータ加工(1)を行って結果をAPに転送(2)し、AP側でさらにデータ加工(2)をして画面に表示(ビジュアライズ)するという流れです。

この流れの中でメモリを使うポイントは大きく3つあり、パース、オプティマイズにかかわるのがライブラリバッファとSQLキャッシュ、HDD読み出しにかかわるのがバッファプール、データ加工(1)にかかわるのがワークスペースです。これらの具体的な名前はDB製品によって違い、概念も微妙に違いますので**図1**はイメージを持ってもらうための目安と考えてください。バッファプールというのはMySQLでの用語で、おもな役割はテーブルのデータをキャッシュし

▼図1　DB負荷への影響要素イメージ(1)



てHDD読み出し回数を減らすことです。

次に「ぐるぐる系」がこれらの流れのどこにどんな影響を与えるかを大まかに整理したものが**表1**です。「設計ガイドライン」は設計上望ましい方針、「ぐるぐる系の悪影響」はそれに対してぐるぐる系SQLがどの程度の影響を与えるかの目安を小～中～大で示し、さらにその操作で使用するメモリを示しています。

「さて、さっき言った『ぐるぐる系の遅さはバッファプールじゃ解決しない』というのはどういうことか、大道君だったらどう見る？」

「え、そうですね、こういうことでしょうか」

❶ぐるぐる系は多くのポイントで悪影響「大」になっている

❷しかし、バッファプールを増やして効果があるのはHDD読み出しだけ。ここへのぐるぐる系の影響は「小～中」なので、「大」を放置して「中」以下のところに手をつけても、たいした効果は期待できない

バッファプールのイメージ
手元にデータの複製を置いておけば、
HDDまで取りに行かずに済む

「そういうこっちゃ。じゃあ、HDD読み出しに対するぐるぐる系の悪影響が『中』以下なのはなぜ？」

「それは……」

「バッファプールがどんな動作をするのか考えればわかるよ」

「……あ、こういうことですか？」

❶バッファプールはHDDのデータをブロック単位で1度だけ読みにいってそれをキャッシュしておくことで、2度目以降のHDD読み込みを不要にするためのもの

❷ぐるぐる系SQLを使っても、バッファプールが働いていればHDD読み出しは増えない(**図2**)

「ビンゴ！　結局、一発系でもぐるぐる系でも、使うデータが同じならHDDを読み出す回数は同じなわけや。バッファプールはHDD読み出しが無駄に多いときに、つまりHDDの同じブロックを何度も読み出してるようなときにその回数を減らすために役に立つもの。だいたい「ぐるぐる」される先のほとんどはマスタ系のデータで、参照頻度が高いのでずっとキャッシュされていることが多いからね。もともとのバッファプールがよっぽど少ないときを除いて、ぐるぐる系SQLに対しては効果はないんよ」

▼表1　DB負荷への影響要素イメージ(2)

影響項目	設計ガイドライン	ぐるぐる系の悪影響	使用するメモリ
SQL生成、転送(1)、パース、オプティマイズ	回数を減らす	中～大	ライブラリバッファ、SQLキャッシュ
HDD読み出し	回数を減らす	小～中	バッファプール
データ加工(1)	大規模なソート、ジョインを最小限にする	小	ワークスペース
	回数を減らす。コネクションをつないでいる間の待ち時間を減らす	大	ワークスペース
転送(2)	データ量、回数を減らす	中～大	ワークスペース
データ加工(2)	集合操作系の処理を避ける(必要な場合は極力DB側の「データ加工(1)」に移す)	中～大	
ビジュアライズ	なし	なし	

もっとも、完全に増えないのであれば影響は「なし」なのですが、一発系で複雑な結合条件・抽出条件を使用する場合には、ぐるぐる系で同じ結果を得ようとするとHDD読み出しが増えてしまうことはありえます。そこで、影響度なしではなく「中以下」としましたが、ほかの項目に比べて相対的に小さいことは確かです。

「メモリを増やせば効果がある、ってものじゃあないんですね」

「足りていないなら増やせば効くけど、足りているところに足しても意味はないよね。でも、昔はメモリのチューニングだけで効果が出る場合がよくあったんよ」

「昔は？　なんでですか？」

DBMSソフトウェアのインストール直後は、メモリの少ないノートPCでも動くような設定になっている場合があり、それを変更せずに使っているケースを10年ぐらい前はよく見かけました。最近ではそこまで極端な例は少なくなりましたが、サーバにメモリを増設したときに、DBのパラメータを変えるのを忘れてそれを使えないままだったという例には、今年もある大手企業のプロジェクトで遭遇しています。

そんな場合には、DBパラメータのチューニングだけで効果が出る場合もありえます。しかし、そもそも下手なSQLを使っているのが原因の場合はそちらに手を打つべきです。

「というわけで、実際のところどうなんや？　ぐるぐる系なのか、これ？」

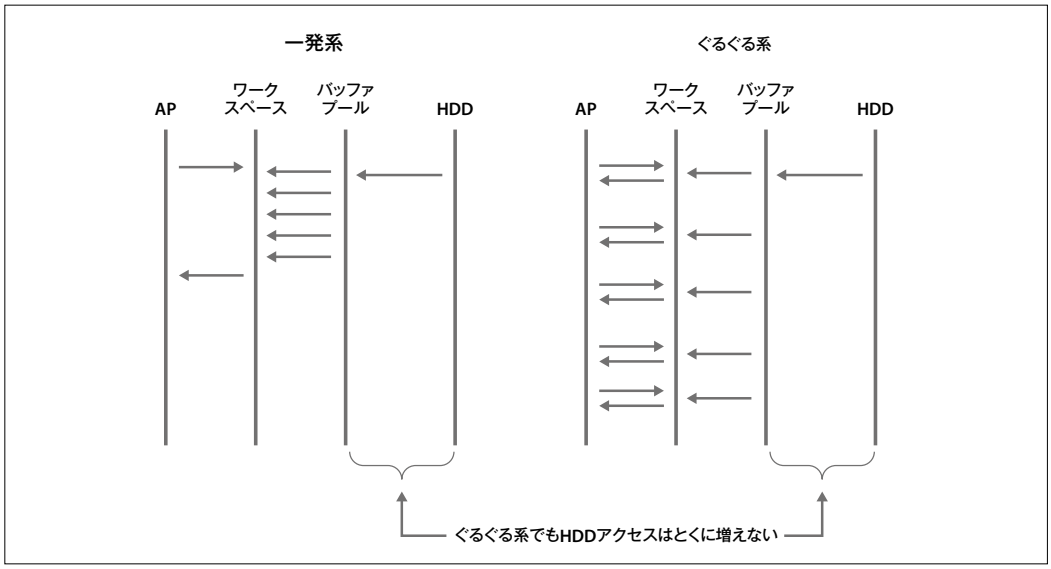
「調べてみます」と言って大道君は調査に入りましたが、その結果は驚くべきものでした。

スケールアウトしにくいからJOINを禁止する？

「JOINを全然使ってないやて……？」

「そうなんです。ぐるぐる系ばっかりですね」

▼図2　バッファプールの動作イメージ



「なんやそら。ド素人かいな!!　メモリもCPUも効くわけないやろ」
と嘆いてはみたものの、実際のところこういうケースはときどきあります。会社によってはJOIN操作を禁止している例があるのです。

「JOIN禁止？　ぐるぐるを推奨するんですか？　なぜ……？」

「いちおう理屈が立つと言えなくもない理由のときもあるけど、実のところしょうもない理由のときもあるよ。しょうもない言うのは、要するにJOINするとSQLが複雑になって扱いに困るからやめろ、という奴で、まあ『私は勉強する気ありません！』宣言みたいなもんやな」

「一緒に仕事したくないタイプですね。もうひとつは？」

「JOINを使っていると負荷分散が必要になったときにスケールアウトしにくいからやらない、という方針をとっていた会社があったね」

「え……それは理にかなってるんですか？」

高負荷に対応する手法には、おおまかにスケールアップとスケールアウトがあります。スケールアップはサーバの性能を上げる方法ですが、高性能なサーバは高価ですし、そもそも無限に速いCPUが手に入るわけでもありません。

スケールアウトはサーバの数を増やす方法で、相互に独立した複数の処理が同時に発生するような用途では、それぞれを別なサーバに分担させるという「安価なマシンの並列動作」により高負荷に対応できます。しかしそのためには「並列動作させる複数の処理が相互に独立して」いなければなりません。データベースの処理はそれが成り立たないことがあり、スケールアウトさせるうえでボトルネックになりやすいのです。

DBがボトルネックになる理由とは?

「DBがボトルネックになりやすい理由はわかる？」

「複数の処理が相互に独立していないということですか……ああ、つまりトランザクションってそのためにあるんですよね？」

「そう。口座残高や商品在庫、商品マスタみたいな、全ユーザが共通して参照／更新する単一のオブジェクトに更新をかける処理は並列動作できない。だからテーブルやレコードにロックをかける必要がある。この部分はスケールアウトできないわけや」

「それはわかりましたが……だからといってJOIN禁止にするのが合理的な方法なんでしょうか？」

たとえばECサイトにログインして注文履歴を参照する処理を素直に考えると、注文履歴が入っている売上明細テーブルと商品マスタテーブルをJOINするのが最も合理的です。これを負荷分散させようとしても、商品マスタは「全ユーザが共通して参照する単一のオブジェクト」ですので分散させられません。すると**図3**のようにサーバをまたいでJOINをすることになり、性能低下を招きます。

「あ、だからJOIN禁止、と……」

「そういうことやね」

「でも商品マスタなんてあまり更新されるデータでもないですし……各サーバにコピーを持たせるってわけにはいかないんでしょうか？」

「実はこんな方法があるんや」

マスタ系データをコピーする方法

図4のようにサーバをマスタ（1つ）＋トランザクション系（複数、以下トラン系と表記）に分け、マスタ系のデータはマスタサーバに入れ、そのコピーをトラン系サーバに作ります。

参照の処理はトラン系サーバ内で完結するのでJOIN操作に支障はありません。マスタ系のデータを更新するときはマスタサーバに更新をかけます。その更新が自動的にほかのトラン系サーバにも反映される、というしかけを作れば全体の一貫性が保たれます。これを実装するためには、DBリンクを使ったマテリアライズドビュー、トリガーでコピー、DBリンクがないMySQLの場合はレプリケーション、などのし

くみが使えます。

「ユーザ同士がお互いの売上明細（＝注文情報）を知る必要はないから、トランAとトランBに全然別のトランザクションデータが入っていても問題ない」

「あ、つまりAとBの明細を同期させる必要はない、と……そこは『相互に独立した複数の処理』になるわけですね」

「そう。マスタ系を同期させる負荷はどうしても発生するけど……」

「マスタ系は……データ量も更新頻度も少ないので、たいしたことはない？」

「そのとおり！」

「なるほど……おもしろいです！」

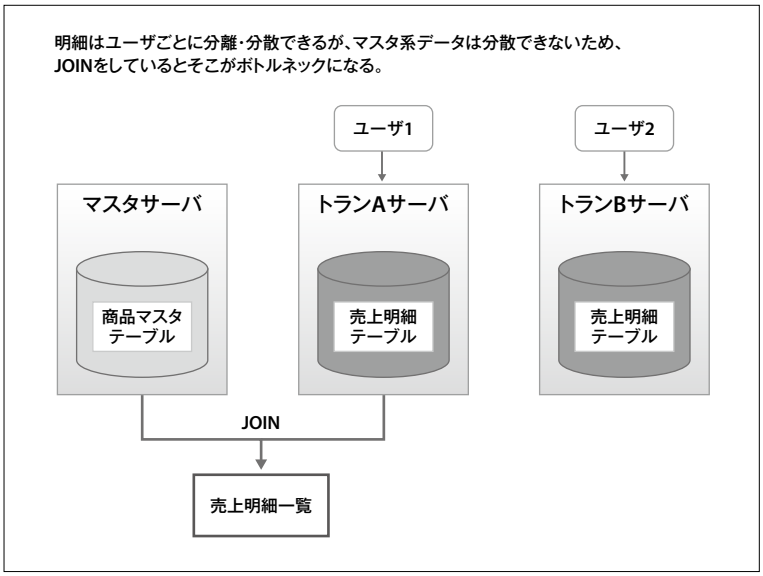
JOIN禁止はかえって負荷を増やす

JOINには確かに負荷がかかります。COBOL時代にはとくにそうだったので、そのイメージを引きずっている人もいるようです。しかしだからといってJOIN禁止というのはあまりにも短絡的な考え方です。

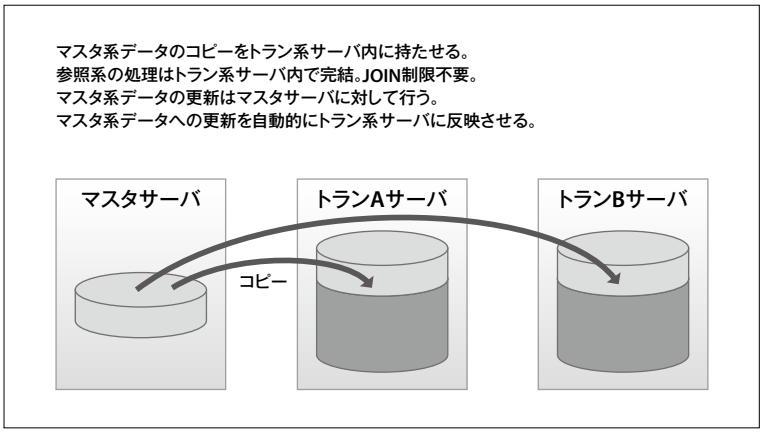
当連載でこれまで書いてきたように、RDBは集合操作が効率良くできるように特化したシステムであり、JOINはその集合操作の核となる機能です。本質的にJOINが必要なのに、RDBにそれをやらせずにAPサーバに移すという「ぐるぐる系」の設計をすると、RDBへのSQL1回あたりの負荷は減りますが、それ以外の部分で負荷が増えて差し引くとかえって有害になる場合が多いですし、開発効率の面でも悪影響を招きます。

性能というのはシステム全体でトータルに考えるべきもので、どこでどんなボトルネックが発生しているのかを見極めて適切な手を打たなければなりません。それをきちんと考えているならば、「スケールアウトさせるためにJOIN禁止」などという短絡的な方針は出てこないはずです。もし設計ガイドラインとしてそんな方針を決めているプロジェクトがあるなら、あらためてその意図を精査することをお勧めします。SD

▼図3　サーバをまたぐJOINは性能低下を招く



▼図4　マスタ系データをトラン系にコピーして負荷分散



一歩進んだ使い方のためのイロハ

Vimの細道

第13回

Vimの標準ファイラ「Netrw」（基本編）

mattn
twitter:@mattn_jp

多機能・便利だけどあまり知られていないのが、Vimの標準ファイラ「Netrw」。その活用方法を基本編・応用編の全2回で追います。基本編では、基本的な使い方とキーマップ、ソート、マーク機能によるファイルへの一括操作、プロトコルを越えたファイルの編集方法などを紹介します。

Vimのファイラ「Netrw」

みなさんはテキストエディタを使う際に、どのようにファイル名を指定しているでしょうか——シェルからエディタを起動する際のコマンド引数で指定して開く、起動したエディタのファイル選択画面から指定して開く、はたまたプロジェクト内でgit管理されたファイルだけを抽出して開く——いろいろな方法があると思います。

Vimには「Netrw」というファイラが付属しており、ディレクトリをブラウジングしながらファイルを選択できます。NetrwはDr.Chip氏が開発しており、古くからVimにバンドルされているプラグインです。Vimとともに成長してきました。特徴は次のとおりです。

- マルチプラットフォーム
- コピーや削除といったファイル操作
- 豊富なビュースタイル
- 拡張可能なアクション
- リモートファイルの編集

実はこのNetrw、機能がたくさんあって便利なのですが、残念なことにそれほど多くの人には使われていない現状があります。使っていたとしても、すべての機能を使いこなしているユーザはほとんどいないのではないでしょうか。

Netrwの機能は多く、とても1回では紹介し切れません。そこで今月号と次月号の2回に渡って、この隠れた機能がたくさんあるNetrwの便利な機能を紹介したいと思います。今回はNetrwの操作方法を、次回はNetrwのカスタマイズ方法を紹介します。

Netrwの使い方

まずは基本的なNetrwの使い方を紹介します。

ファイルの開き方

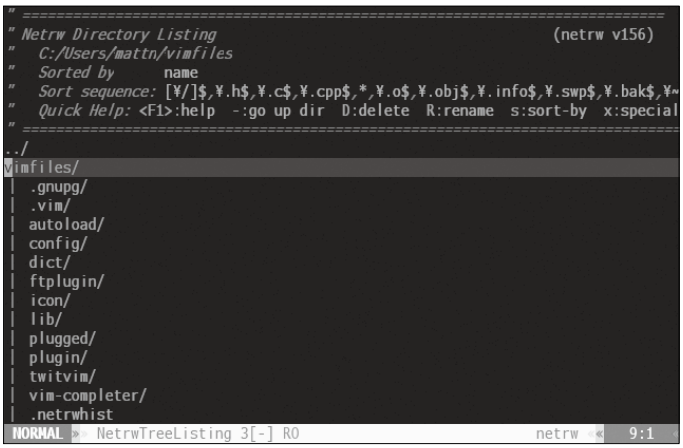
Vimは本来ファイルを編集するためのテキストエディタですが、ディレクトリ名を指定して開くことでファイルブラウザが開きます(**図1**)。

ファイルブラウザではj/kでカーソル移動を行えます。目的のファイルの上で Enter を押すと対象のファイルのバッファが開きます。これが一番簡単なNetrwの使い方になります。

次に、ファイルブラウザで使えるキーマッピングを示します(**表1**)。

Ctrl -hから隠し項目リストを入力すると、ファイルブラウザに表示される一覧から特定のファイル／ディレクトリを非表示にできます。ここでは正規表現でパターンを入力します。次の例は拡張子が「.jpg」と「.png」のファイルを表示しないパターンです。

▼図1　Netrwのファイルブラウザ画面




```
\.\(jpg\|png\)$
```

隠し項目リストのデフォルト値はg:netrw_list_hideで設定できます。

qfで表示されるファイルの情報はファイル名、サイズ、更新時刻です。Vim7.4.1976以前ではサイズが2GB以上のファイルサイズを正しく表示できませんでしたが、このバージョンにて64bitの数値を扱えるようになったため、きちんと表示できるようになりました。

始めはキーが覚えられないかもしれませんが[F1]でヘルプを表示できるので基本的な操作はそちらからすぐに参照できます。[F1]がOSの機能などに割り当てられている場合は、:help netrwで参照してください。

▼表1　基本操作のキーマッピング

キー	動作
[Enter]	ディレクトリを開く／ファイルを開く
[Delete]またはD	ファイル／ディレクトリを削除
[Ctrl]-h	隠し項目リストの編集
[Ctrl]-l	一覧の再表示
-	1つ上のディレクトリに移動
qf	ファイルの情報を表示
p	ファイルをプレビュー
d	ディレクトリの作成
x	関連付けられたプログラムでファイルを実行
X	カーソル配下のファイル名をsystem()で実行
%	移動したいパスを入力
[F1]	ヘルプを表示

編集ウィンドウ設定

ファイルブラウザで[Enter]を押すと編集用ウィンドウでファイルが開きます。ファイルを変更していないのであれば、同じウィンドウでファイルが開きます。ファイルが変更されている場合は、プロンプトで保存、または変更を破棄するかを聞かれます。

この編集用ウィンドウは:NetrwCコマンドを使ってカレントウィンドウに設定できます。また:NetrwC 3のようにウィンドウIDを指定することでも変更できます。今いるウィンドウのIDは:echo winnr()で確認できます。ファイルブラウザでCをタイプすることで現在の編集用ウィンドウIDを確認することもできます。

ウィンドウの操作

Vimはスクリーン上に複数のウィンドウを開

▼表2　ウィンドウ関連のキーマッピング

キー	動作
o	カーソル配下のアイテムを縦分割ウィンドウで開く
t	カーソル配下のアイテムの新しいタブで開く
v	カーソル配下のアイテムを横分割ウィンドウで開く

くことができます。ファイルブラウザで**表2**のキーをタイプすると別のウィンドウが開き、そこで対象のファイルを編集できます。これらのキーは、押すたびに新しいウィンドウが開かれます。

ファイルブラウザの操作

Netrwが表示するファイルの一覧では、**表3**のようにソート順や表示／非表示の切り替え、カレントディレクトリの移動などが行えます。

デフォルトのビューは、次の4つの中からg:netrw_liststyleで設定できます。

・0：ファイル名のみ
・1：ファイル名とサイズ、タイムスタンプを表示
・2：マルチカラム
・3：ツリー

ソートのサフィックス(S)には、正規表現のパターンをカンマセパレートで入力します。デフォルト値はg:netrw_sort_sequenceで設定できます。

ブックマーク

Netrwでは、**表4**のようにブックマーク・閲覧履歴に関する操作も行えます。Netrwの閲覧履歴およびブックマークは、設定ファイルにおいてruntimepathで指定される最初のパスに、それぞれ「.netrwhist」「.netrwbook」というファイル名で保存されます。置き場所を変更したい場合はg:netrw_homeという変数で指定します。次の例は、g:netrw_homeにホームディレクトリを指定しています。

```
let g:netrw_home = expand('~/')
```

マーク操作

Netrwの操作の肝はマーク操作です(**表5**)。ファイルブラウザで対象のファイルをマークして操作を一括実行できます。mfをタイプするとカーソル配下のファイルの色が変わるので、マークされたことがわかります。

quickfix/location-listからNetrwへのマークに変換する機能(qF、qL)については、たとえばあるパターンで:grepした結果見つかったファイルを、一括で削除するといったユースケースが考えられます。

mdは2つ以上のファイルをマークした状態で実行すると、1つめに指定したファイルと残りのファイルとの差分をdiff形式で表示します。

ファイルのコピー(mc)や移動(mm)を行うには、まずターゲットを指定する必要があります。mtでディレクトリをターゲットとして設定し、次にmfでファイルをマーク、最後にmcでコピーを実行します。ただしこのコマンドには、Windowsで動かないという問題が残っており、これについては今後vim-jpで修正していきたいと思います。

mvを実行すると、Netrwはマークしたファイルをいったん Vimで開き、:で始まるコマンド

▼表3　ソート、表示に関するキーマップ

キー	動作
a	隠しモード、表示モードの切り替え
c	Vimのカレントディレクトリを閲覧ディレクトリに移動
gh	ドットファイルを表示／非表示
gn	カーソル配下のディレクトリをツリーのトップにする
i	ビューを切り替える(簡易／詳細／ワイド／ツリー)
r	ソート順を逆に切り替え
R	ファイルをリネーム
s	ソートを切り替え(名前／時刻／ファイルサイズ)
S	名前ソートのサフィックスを指定

▼表4　ブックマークに関するキーマップ

キー	動作
gb	1つ前のブックマークディレクトリに移動
qb	ブックマークディレクトリと履歴を一覧表示
mb	ディレクトリをブックマーク
u	最近表示したディレクトリへ移動
U	次に表示したディレクトリへ移動

を適用したあとファイルを保存します。ですので、たとえばmxをタイプしたあと、

```
s/foo/bar/g
```

を入力すると、ファイル内のfooがbarへ一括置換が行われます。ちょうど、sedの「-i」フラグと同じ動作になります。

Netrwはネットワークを超えられる

Netrwの特徴的な機能として、各種ネットワークプロトコルを扱えることが挙げられます。次に、Netrwがサポートするネットワークプロトコルを示します。

- WebDAV
- FTP
- HTTP
- RCP
- rsync
- SCP
- SFTP

HTTPについては、HTMLの読み込みができますが、書き込みはできません。そのほかのプロトコルでは書き込みもできます。FTPやBASIC認証を使う場合は「.netrc」ファイルを読み込むことで認証を行うことができます。編集はVimのコマンドから:eで指定できます。また、コマンドラインから引数で指定することもできます。

たとえば、SSHポートが空いているサーバの~/.vimrcを直接編集するのであれば、次のように実行します。

```
$ vim scp://server/~/.vimrc
```

この場合、パスワードやパスフレーズの入力を避けるために、ssh-agentを起動しておく必要があります。そのほか、プロトコル別に設定方法が異なるので、詳細は:help netrw-xferを参照してください。

▼表5　マーク操作に関するキーマップ

キー	動作
qF	quickfixにあるファイルをNetrw上でマーク
qL	locaiton-listにあるファイルをNetrw上でマーク
mc	マークしたファイルをターゲットディレクトリにコピー
md	マークしたファイルでdiffを実行
me	マークしたすべてのファイルを編集
mf	ファイルをマーク
mF	ファイルのマークを解除
mr	正規表現でマッチしたファイルをマーク
mg	マークされたファイルでvimgrepを実行
mh	マークしたファイルのサフィックスを隠しリストに設定
mm	マークしたファイルをターゲットディレクトリに移動
mp	マークしたファイルを印刷
mt	カレントディレクトリをターゲットに設定
mT	マークしたファイルでctagsを実行
mu	すべてのマークを解除
mv	マークしたファイルにVimのコマンドを適用
mx	マークしたファイルにそれぞれシェルコマンドを適用
mX	マークしたファイルに一括でシェルコマンドを適用
mz	マークしたファイルを圧縮／解凍

おわりに

今回はNetrwの基本動作とキーマッピングについて説明しました。巷にはVimで扱えるファイラがほかにもたくさんあり、どうしても目新しいほうに目が行きがちです。ですが、実際にはNetrwだけで事足りてしまうことの方が多かったりもします。

Netrwは使ってみると意外にも多機能で、それだけでも実はいろいろなことができるようになっています。マーク機能を使えば一括コピーや一括移動もできるので、Vimの中で全部やってしまいたい人にはとても便利だと思います。

次回はNetrwのカスタマイズ方法と、活用方法などをご紹介します。SD

書いて覚える Swift 入門

第21回 “hello again”を待ちながら

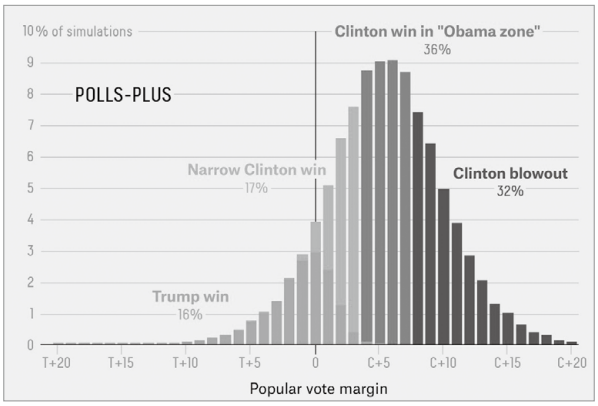
Writer 小飼 弾（こがい だん）　twitter @dankogai

連載作家殺しのアップル

前号から早1ヵ月。macOS Sierraは無事リリースされましたが、月刊誌連載の執筆者としては困ったことに、締切日もなお「微妙な時期」は続いています。Apple Pay[注1]日本スタートは10月25日。新型 MacBook Pro が発表されるであろうhello again[注2]イベントは10月27日。「パチンコガンダム駅」からもう4年、やっと日本国内でも純正マップで公共交通機関検索できるようになるのも、Swift PlaygroundがiCloudでMacとiPadを行き来できるようになるのも、間に合いません（涙）。たった数日なのに！

10月下旬現在の筆者と本記事を手にしている読者の皆さんと、Appleの新製品以上に隔てているものがあります。読者の皆さんは、すでにObamaの次の米国大統領が誰かを知っているのです。なんてうらやましい！

▼図1　ヒラリーの勝利は確実?



Bad hombre

全3回の公開討論も終わり、あとは投票日11月8日（現地時間。日本時間では翌日）を待つだけとなった執筆現在（10月下旬）。今日、最も正確な選挙予測をするFiveThirtyEight[注3]のNate Silver[注4]によると、状況は**図1**のグラフのとおり、Hillary Clintonの勝利は決まったも同然に思われます。

にもかかわらず、普段はノンポリなギークたちも、コードそっちのけで今回の選挙について語らずにはいられないようです。Perlの父、Larry Wallですらこの調子（**図2**）。

もう一方の候補、Ronald Trumpの「反人気」の高さは過去に例を見ません。普段は二大政党の党大会双方に機材を貸し出すAppleも今回は取りやめていますし、それどころかトウの共和党自体、自ら選んだ候補への推薦を下げているような状態の中、異色を放っているのがピーター・シール（Peter Thiel[注5]）。

注1）http://www.apple.com/jp/apple-pay/getting-started/
注2）http://www.apple.com/apple-events/october-2016/
注3）FiveThirtyEight（http://fivethirtyeight.com/）
注4）https://twitter.com/NateSilver538/status/789625104657514496
注5）http://www.politico.com/story/2016/06/apple-wont-aid-gop-convention-over-trump-224513

PayPalの共同創始者で『ゼロ・トゥ・ワン[注6]』の著者でもあるビリオネアは、Trumpの選挙に125万ドル寄付[注7]することを表明しました。

しかしこのニュース以上に物議を醸し出したのは、Y CombinatorとFacebookの対応でしょう。双方ともTrumpへの不支持を表明する一方、Thielとの関係を維持[注8]することを表明したのです。

Ruby on Railsの作者であるデビッド・ハイネマイヤ・ハンソン(David Heinemeier Hansson、@dhh)のツイートに、『ハッカーと画家[注9]』の著者でもあるポール・グレアム(Paul Graham、@paulg)は「もし共和党を支持している会社がヒラリーのサポーターを解雇したらどう感じるか」(**図3**)と答えています[注10]。

しかしDHHが指摘するとおり、ベンチャーキャピタル(VC)のパートナーというのは被雇用者ではありません。労働基準法に相当する各国の法が被雇用者支持政党による差別を禁止しているのは、それが生活基盤を人質にとるのに相当する行為だというのが理由ですが、VCのパートナーというのは雇用側というまさに正反対の立場であり、Trumpを支持することはTrump的な判断を投資先に下すのではないかという懸念をむしろ増すものだと判断せざるをえないでしょう。結局@dhhと@paulgのやりとりは、@paulgが@dhhをブロックすることで終わるのですが、この事件はIT業界における多様性とはいったい何なのか、筆者にも再考を迫られるものでした。

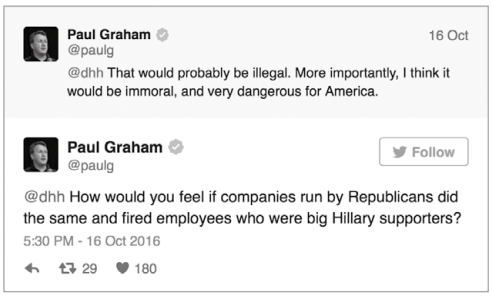
最悪の脆弱性とは

ちょうどその頃起こったのが、脆弱性を突かれて乗っ取られたセットトップボックスやウェブカメラによるDDoS[注11]。「IoTによるDDoS」がこれまでの脆弱性と異なるのは、機器のアップデートで収束を図るという、パソコンやスマフォの常識が通用しないこと。機器によっては、ファームウェアのパスワードを変更することすら不可能とあっては、物理的にプラグを抜くしか解決法はありません。

そう。「物理的に外す」。これこそが「モノ」の欠陥を修復するたった1つの冴えたやり方だったのです。むしろ「論理的に交換」できることがパソコンやスマフォでは新常識として確立されていたにもかかわらず、旧来から存在していたというだけで旧来の常識のまま進めてしまったことこそが、IoTにおける真の脆弱性なのかもしれません。

▼図2 “More than one way”but“no Trump way”

▼図3 DHHとポール・グレアムのTwitterでのやりとり



注6) http://amzn.to/2f79YzJ
注7) http://www.nytimes.com/2016/10/16/technology/peter-thiel-donald-j-trump.html
注8) https://theintercept.com/2016/10/19/when-the-genius-men-of-silicon-valley-suddenly-dont-seem-so-smart/
注9) http://amzn.to/2f7aePh
注10) https://twitter.com/paulg/status/787571526891663360
注11) http://krebsonsecurity.com/2016/10/hacked-cameras-dvrs-powered-todays-massive-internet-outage/

しかし、「物理であれ論理であれ、脆弱な部品は交換する」というのは実はIoTどころかインターネット以前には確立された常識でもあるのです。Galaxy Note 7を挙げるまでもなく、スマフォでさえ論理的に直せなければ物理的に交換するしかないのですし、家電にもクルマにもリコールは制度として確立しています。

そして交換するのであれば、脆弱性が健在する前に交換してしまえればなおよし。

その点から考えると、「脆弱性」があろうがあるまいが、定期的な交換を制度化した選挙制度というものは、「期限付き保証はあっても使わなくなるか壊れるまで放置」という「アプライアンスの常識」よりよほど先進的にも感じます。もっとも今回の米国大統領選挙をめぐる混乱は、交換品自体の品質保証体制の脆弱性を突かれた結果とも言えるのですが。

トランプ候補と言えば、GitHubにTrump Script[注12]が公開されています。これを実行すると(もちろんMacでは動かない！)、**図4**のようになります。

Keep Walking

この2つの事件を通してあらためて思い起こしたのが、今年のノーベル生理学医学賞[注13]。数多の研究者が「新しいものが、どう形作られるのか」を追ってるのを横目に、大隅良典先生は「古いものはどう捨てられるのか」を追い続けた結果、単独受賞にたどり着きました。しかしオートファジーにほかの研究者が遅まきながら目を向け始めたのも、オートファジーという現象自体より、オートファジーというメカニズムが故障するとヤバイということに気がついてからだったというのも、人というのは痛い目に合わないと学ばないものなのかと自戒しています。去年左腕が壊れなければ、今年の私が1日15km歩くようになっていたか。

◆　◆　◆

"If it ain't broke, don't fix it"、「壊れてもいないのに直すな」というの本誌の読者もよくご存じの格言ですが、「壊れているのに気づいてない場合」に対して脆弱でもあります。

互換性を損ねてでも直し続けるSwiftという言語はその点においてもモダンなのですが、Xcode 8.0現在、新たに壊れたものが8.1以降で直るのを待っているという、歯がゆい状況が「史上最低」の米国大統領選挙とともに終わっていることを祈りつつ、今月はこれにて失礼します。SD

▼図4　TrumpScriptの実行結果

```
Traceback (most recent call last):
  File "./bin/../src/trumpscript/main.py", line 30, in <module>
    main()
  File "./bin/../src/trumpscript/main.py", line 24, in main
    Utils.verify_system(args.Wall)
  File "/Users/dankogai/github/TrumpScript/src/trumpscript/utils.py", line 34, in verify_system
    Utils.boycott_apple()
  File "/Users/dankogai/github/TrumpScript/src/trumpscript/utils.py", line 72, in boycott_apple
    raise Utils.SystemException('boycott');
  File "/Users/dankogai/github/TrumpScript/src/trumpscript/utils.py", line 21, in __init__
    raise Exception(random.choice(ERROR_CODES[msg_code]))
Exception: Mac? 'Boycott all Apple products  until such time as Apple gives cellphone info to ➥
authorities regarding radical Islamic terrorist couple from Cal'
```

注12) https://github.com/samshadwell/TrumpScript
注13) http://www.nobelprize.org/nobel_prizes/medicine/laureates/2016/press.html

Sphinxで始めるドキュメント作成術

山田 剛　Yamada Go　Twitter @usaturn

第21回　PDFを出力しよう

LaTeXを利用したPDF出力

Sphinxでよく利用される出力形式の1つがPDFです。PDFはポータブルで印刷に向いているなどの特徴があることから、業務でもよく利用されるフォーマットです。

Sphinxでは外部の拡張やツールを利用する方法を含め、PDFに変換する方法がいくつも存在します(**表1**)。本稿では、生成されたPDFの見栄えが良いLaTeXを使ったPDF出力を紹介します。この方法では、LaTeXのテクニックを使って出力がカスタマイズできます。

Sphinx環境の確認／TeX環境の構築

本稿では次のWindows環境で検証／確認をしています。UNIX系OSを利用されている方も同様の手順で進めることができます。

- OS：Windows 10
- Python：3.5.2
- Sphinx：1.4.8

簡単に最新のTeX環境を構築できることからTeXのディストリビューション「TeX Live」のインストールを推奨します。最新版のTeX Live 2016をインストールしましょう。

TeX Liveのインストールは付属のインストーラを実行します。詳細な手順はSphinx-Users.jpで紹介されていますので、ここでは説明を省きます[注1][注2]。

また、Windows環境ではTeX Liveに加えてmakeコマンドをインストールします。筆者はGow(Gnu On Windows)[注3]というUNIXコマンド集を利用しています。

そのほか、出力されたPDFの確認用としてAdobe Acrobat ReaderをAdobe公式サイト[注4]よりインストールしてください。

PDFの出力手順

PDFを出力するには次の手順を実行します。

①Sphinxプロジェクトの作成
②日本語ドキュメント向けのLaTeX出力設定の追加

▼表1　SphinxからPDFを生成する方法

方法	概要
LaTeX出力	Sphinx標準のPDF出力方式。組版エンジンであるLaTeXを介してPDFを生成する。出力結果の品質が非常に高い
rst2pdf拡張	Sphinx拡張であるrst2pdfを用いてPDFを生成する。出力はシンプルだが、依存関係は少なく済む
HTMLをPDFに変換する	SphinxでHTML(singlehtml)をブラウザなどを介してPDFに変換する。変換の際に操作を必要とするが、依存関係の少なさと一定の出力品質が担保されている

注1）http://sphinx-users.jp/cookbook/pdf/latex-install-tl.html
注2）Windows以外のOSへのインストールは、次のTeX Wikiから「TeXをインストールしよう」→「TeX入手法」→「TeXをインストールする方法」を参照。https://texwiki.texjp.org/
注3）https://github.com/bmatzelle/gow/releases
注4）https://acrobat.adobe.com/jp/ja/acrobat/pdf-reader.html

③reSTファイルの編集
④makeコマンドの実行

■Sphinxプロジェクトの作成

はじめにsphinx-quickstartコマンドでSphinxプロジェクトを作成します。

```
> sphinx-quickstart -q -p project_name ➥
-a test -l ja -v 1.0 project_dir
```

初期化の際、-lオプションで日本語(ja)を指定するのがポイントです。既存のプロジェクトを利用する場合はconf.py内のlanguageを'ja'に設定してください。

■日本語ドキュメント向けLaTeX出力設定

続けて、日本語ドキュメント向けのLaTeX出力の設定を行います。conf.pyに次の設定を追加します。

```
日本語向けLaTeX出力設定
latex_docclass = {
    'howto': 'jsarticle',
    'manual': 'jsbook'
}
```

latex_docclassを設定するとLaTeXのドキュメントクラスを差し替えることができます。ここでは日本語環境でよく使われるjsarticleやjsbookを利用するように設定しています。

■reSTファイルの編集

次にreSTファイルを編集します。ここではindex.rst(**リスト1**)を変更し、chapter1.rst(**リスト2**)を追加します。

■makeコマンドの実行

文章を書き終えたら、いよいよPDFファイルを出力してみましょう。PDFの生成にはmake latexpdfjaコマンドを実行します。

```
> make latexpdfja
```

処理の様子がターミナルに大量に流れ、次のメッセージが出てプロンプトが返ってきたら完了です。

```
Build finished; the PDF files are in ➥
_build/latex.
```

メッセージどおり_build/latexディレクトリ以下にPDFファイルが生成されていることを確認してください。

```
project_dir¥_build¥latex¥project_name.pdf
```

どうでしょう。想像していたよりも簡単にPDFが出力できたのではないでしょうか。

LaTeX経由で出力したPDFの特徴

LaTeX経由で出力したPDFには次の特徴が

▼リスト1　index.rst

```
=====================================
project_name のドキュメントへようこそ！
=====================================

見出し１の手前に表示されるコンテンツ

- この章はマスタードキュメント(index.rst)➥
に書かれている内容です。
- toctree の直前に記述されています。


.. toctree::
   :maxdepth: 2

   chapter1

見出し２
========

２章の本文

- この章はマスタードキュメント(index.rst)➥
に書かれている内容です。
- toctree の直後に記述されています。
```

▼リスト2　chapter1.rst

```
========
見出し１
========

１章の本文

- この章は toctree 配下のドキュメント➥
(chapter1.rst)に書かれている内容です
```

あります。

- 表紙が自動的に生成される
- 目次が自動的に生成され、章番号が割り振られる[注5]
- rsSTファイルが1つのPDFに統合される

一方でLaTeX出力特有の動作として、ドキュメントを書く際に意識しなくてはならないところもあります。

1つめは、index.rstの最初のセクションタイトルがPDFに出力されないことです。通常、最初のセクションタイトルはドキュメントの見出しとなる部分ですが、LaTeX出力ではドキュメントのタイトルはconf.pyのlatex_documentsで設定するため、LaTeXソースコードに変換する際に読み飛ばされます。先ほど出力したPDFでも、目次には最初のセクションタイトル「project_name のドキュメントへようこそ！」は含まれず、次に登場するセクションタイトルである「見出し1」が先頭に表示されます(**図1**)。

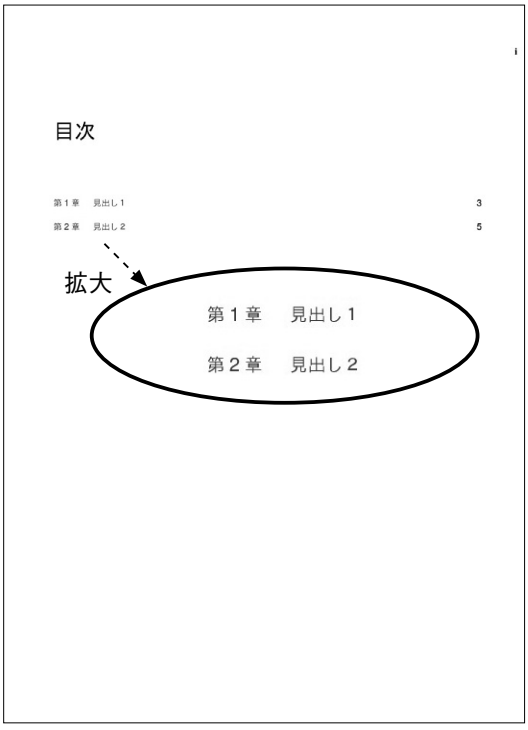
もう1つの注意点は、最初のセクションタイトルに続くコンテンツが、独立したページに配置されることです。先ほど触れたとおり、2番目に登場するセクションタイトルである「見出し1」が最初の章とみなされるため、最初のセクションタイトルから2番目のセクションタイトルまでにあるコンテンツは第1章の手前にあると解釈されます。そのため、これらのコンテンツは独立したページに配置されます(**図2**)。

こうしたLaTeX出力特有の動作を回避するには、次の点に注意してドキュメントを作成すると良いでしょう。

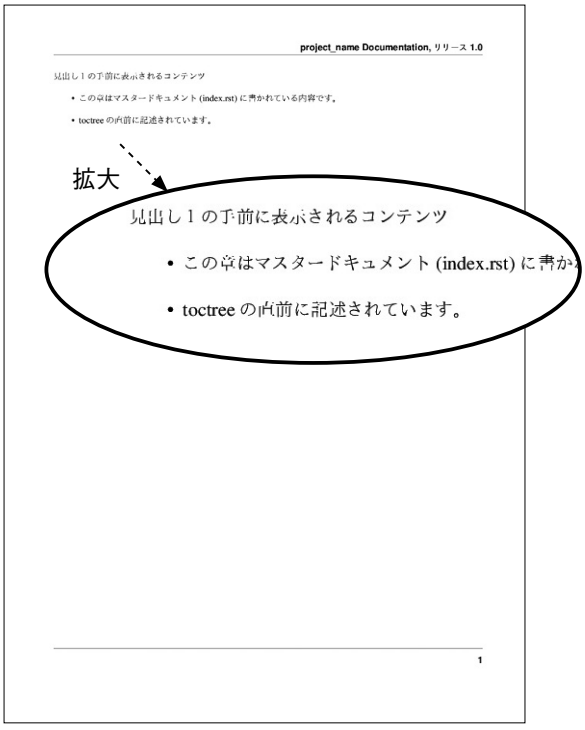
- 2階層以上のセクションを持った文章構造にする
- 先頭のセクションには本文を持たせない

注5) LaTeXによって自動的に割り振られるものです。HTMLで出力したときと異なる章番号が割り振られることがあります。

▼図1　出力された目次



▼図2　第1章の前に配置されたコンテンツ



具体的には**リスト3、4**のようなマークアップが良いでしょう。

PDFの形式をカスタマイズしよう

LaTeX出力にはさまざまなオプションが提供されています。また、LaTeXのマクロを記述することで見た目を大幅に変更できます。

ここでは例を挙げながら、いくつかのカスタマイズ方法を紹介します。

白紙のページを削除する

LaTeX出力の標準の設定では、紙の書籍を想定したスタイルが選択されています。そのため、目次や章扉が必ず右側のページから始まるよう空白ページが挿入されることがあります(**図3**)。

このスタイルは見開きで綴じられるような印刷物向けの設定であるため、PDFを直接閲覧するような用途には向いていません。オンラインでPDFを参照する場合は、conf.pyに**リスト5**の設定を追加します。この設定を行うことで、空白ページが詰められたPDFが生成されます。

簡素なPDF出力に変更する

独立した表紙／目次を持たない、シンプルなPDFを出力したいというケースも考えられます。その場合はconf.pyのlatex_documentsの設定を変更します。latex_documentsの5番目の引数はスタイルを表します。初期設定では書籍を想定したmanualが指定されていますが、これをレポート用のスタイルであるhowtoに変更します(**リスト6**)。なお、howtoスタイルは空白ページなどを持たないため、先

▼リスト3　LaTeXに変換しやすいドキュメント構成(1)

```
==================
ドキュメントタイトル
==================

  ※ここには本文を書かない

見出し１
=========

テキストテキストテキスト...

見出し２
=========

テキストテキストテキスト...
```

▼リスト4　LaTeXに変換しやすいドキュメント構成(2)

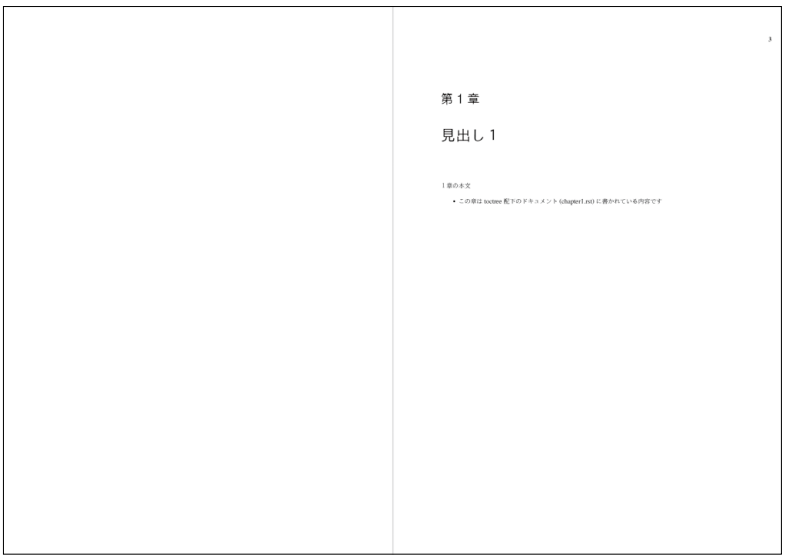
```
==================
ドキュメントタイトル
==================

  ※ここには本文を書かない

.. toctree::

   chapter1
   chapter2
```

※index.rstには toctreeのみ配置

▼リスト5　非印刷用PDFの出力オプション

```
非印刷用PDFの出力オプション
latex_elements = {
    'extraclassoptions': 'oneside,openany',
}
```

▼図3　書籍を想定したスタイルの見開き



ほど設定した非印刷用PDFのオプション(latex_elements)は削除してかまいません。

howtoスタイルでPDFを出力すると**図4**のような見た目に変わります。

段落で字下げする

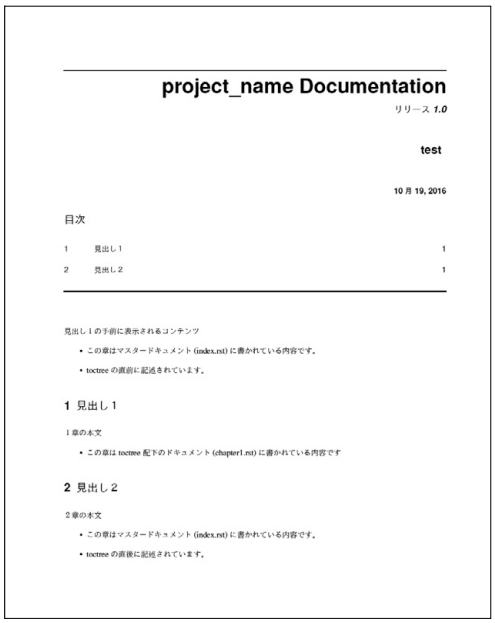
標準では段落ごとの字下げが無効になっています。次の設定により字下げを有効にします。

字下げの設定

```
latex_elements = {
    'preamble': r'¥setlength¥parindent{1zw}'
}
```

ここで設定したものはLaTeXソースコードのプリアンブル部という個所に記述されます。プリアンブル部にLaTeXマクロを記述することでPDFの見た目を変更できます。設定にはLaTeXの知識が必要ですが、ページ構成や見た目など、さまざまなカスタマイズが可能です。

▼図4　howtoスタイル



▼リスト6　出力形式をレポートに変更する

```
latex_documents = [
    (master_doc, 'project.tex',
     'project¥¥_name Documentation', 'test',
     'howto'),  ←manual から howto に変更
]
```

その他のカスタマイズ

ここで紹介した以外にもSphinxでは数多くの設定が提供されています[注6]。これらを組み合わせて読みやすいPDFを生成すると良いでしょう。

フォントの埋め込み

フォントを埋め込まないPDFでは、異なる環境で閲覧する際に、フォントが見つからず意図しない表示になることがあります。TeX Live 2016ではフォントの埋め込みが行われるよう初期設定されていますが、念のためフォントが埋め込まれていることを確認してみましょう。Adobe Acrobat ReaderでPDFファイルを開き、「ファイル」→「プロパティ」→「フォント」と選択していくと、確認できます(**図5**)。

TeXのフォント設定は欧文と日本語でそれぞれ違いますので、分けて説明をします。

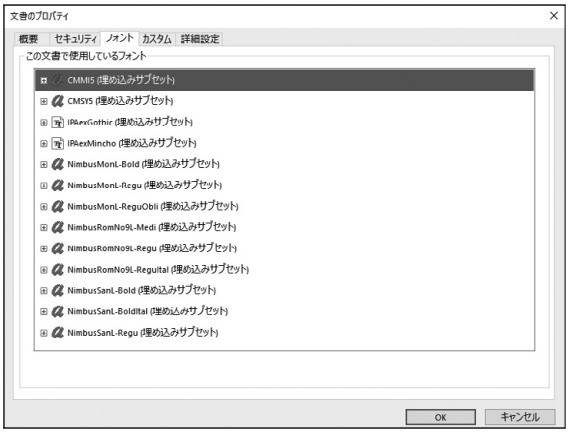
欧文フォントを埋め込もう

TeX Live 2016をインストールしている場合は、標準で「times」という欧文フォントのパッケージが使用されます。

設定を変更するにはconf.pyのlatex_elements

注6) Sphinxリファレンス LaTeX出力のオプション
http://www.sphinx-doc.org/ja/stable/config.html#options-for-latex-output

▼図5　フォントの確認



にfontpkgを追加します。次の例ではpalatinoというパッケージを指定しています。

```
欧文フォントの指定
latex_elements = {
     'fontpkg': r'¥usepackage{palatino}',
}
```

指定できるフォントのパッケージ名は、Windows 10にTeX Live 2016をインストールした場合、「C:¥texlive¥2016¥texmf-dist¥tex¥latex¥psnfss」に配置されるパッケージファイルから拡張子を抜いた文字列となります。標準で利用できる文字列は次のとおりです。

「avant」「bookman」「chancery」「charter」
「courier」「helvet」「mathpazo」「mathpple」
「mathptm」「mathptmx」「newcent」
「palatino」「pifont」「times」「utopia」

それぞれのフォントについての説明は省きますが、欧文フォントの変更は手間をかけずにドキュメントのイメージを変えられますので、一度お試しください。

日本語フォントを埋め込もう

欧文フォントと同様にTeX Live 2016をインストールしている場合は、標準で「IPAex/IPAフォント」が埋め込まれます。

古いバージョンのTeX Liveや、それ以外の環境でフォントが埋め込まれていない場合は日本語フォントパッケージの設定が別途必要となります。日本語フォントの設定はconf.pyの書き換えではなく、TeXのコマンドを実行します。

▼表2　日本語フォントパッケージ

フォント名	備考
noEmbed	フォントを埋め込まない
hiragino、hiragino-elcapitan、hiragino-elcapitan-pron、hiragino-pron	ヒラギノフォント
ipa、ipaex	IPA/IPAexフォント
ms	MSフォント
yu-win、yu-win10、yu-osx	游書体フォント
kozuka	小塚フォント
morisawa、morisawa-pr6n	モリサワフォント

```
①IPAexフォントを設定するコマンド
> kanji-config-updmap-sys ipaex
②フォントが設定されたことを確認する
> kanji-config-updmap-sys status
```

②を実行した結果、次のように表示されていれば成功です。

```
CURRENT family : ipaex
Standby family : ipa
Standby family : ms
Standby family : yu-win
```

kanji-config-updmap-sysコマンドの引数として利用できるフォントパッケージの文字列は「C:¥texlive¥2016¥texmf-dist¥fonts¥map¥dvipdfmx¥jfontmaps」配下のディレクトリに対応しています。

パッケージ名(**表2**)に対応したフォントがインストールされていないと、コマンドを実行しても有効にならないことに注意してください。

そのほか、初期設定にないフォントを埋め込みたい場合や、より詳しくフォントの埋め込みについて知りたい場合は、TeX Wiki[注7]を参照してください。

まとめ&次回予告

Sphinxは「TeX Live」のインストールと簡単な設定のみで、実用的なPDFファイルを出力できます。Sphinxが標準で提供していない設定に関しても、LaTeXの設定により変更可能です。

本稿は、Sphinx関連以外ではWeb、書籍のTeX関連情報を参考にしました。とくにTeX Wiki、『LaTeX2 ε 美文書作成入門』[注8]は非常に勉強になりました。TeXコミュニティの方々に感謝いたします。

次回は「Sphinxで本を書こう！」です。SD

注7）次のURLから「一覧」→「TeXとフォント」を選択。
https://texwiki.texjp.org/
注8）奥村晴彦、黒木裕介 著、技術評論社、2013年。

COLUMN

Sphinx-1.5のLaTeX事情

Author 小宮 健

本連載執筆陣の一人、小宮です。現在、Sphinxプロジェクトでは次期メジャーリリースであるSphinx-1.5の開発作業が大詰めを迎えています。Sphinx-1.5は半年ぶりのメジャーバージョンアップで、数多くの機能が追加されています。

ここでは、今回の記事で紹介したLaTeX出力周りの変更点について紹介したいと思います。Sphinx-1.5の開発では新たにTeX使いがメンテナに加わったこともあり、LaTeXビルダがとても便利になっています。

■日本語向けの設定がシンプルに

Sphinx-1.5では言語ごとに適切な初期設定を持つようになります。その結果、conf.pyで言語を日本語(`language = 'ja'`)に設定している場合、今回紹介したlatex_docclassの設定が不要となります。また、PDFを生成する際のコマンドも`make latexpdfja`ではなく`make latexpdf`の利用を推奨するようになりました[注A]。

この変更により、日本語PDFを生成するために設定しなくてはならない項目が減り、また、言語によらず、常に`make latexpdf`コマンドでPDFが生成できるようになっています。

■画像のサイズ指定にpxが利用可能に

これまでのLaTeX出力では、画像のサイズを指定する際は、LaTeXの単位系であるcmやemなどを使う必要がありました。一方、Webでは一般にpxを使うため、HTML出力向けにpxを使って画像のサイズを指定してしまうとLaTeX出力には反映されませんでした。

Sphinx-1.5では画像の単位系としてpxをサポートしたので、HTML出力とLaTeX出力の双方で画像のサイズ指定にpxを使えます。なお、LaTeX出力のデフォルトは96dpiです[注B]。96pxの画像は1インチ(2.54cm)として扱われます。

■TeXエンジンが切り替え可能に

Sphinx-1.5では、latex_engineオプションの指定により、PDFの生成に利用するTeXエンジンを切り替えられるようになります。通常、日本語向けにはpTeXが利用されるようになっていますが、これをLuaTeXやXeTeXを利用するように設定できます。

■カスタマイズ性の向上

Sphinxで使用しているTeXマクロを見直し、カスタマイズしやすくなります。各種インラインテキストやnoteやwarningなどの警告ブロックなど、数多くの要素に専用のスタイルが提供され、利用されるようになります。これにより、利用者は**リストA**のようにマクロを上書きすることで見た目をカスタマイズできます。

また、Sphinxプロジェクト内に_templates/latex.tex_tファイルを配置すると、生成されるTeXのソースコードそのものをカスタマイズできるようになりました。

■Sphinx-1.5のリリース予定

Sphinx-1.5では、ここで紹介した内容のほかにも、クロスリファレンス機能(numref機能)の強化やHTML5/EPUB3への対応強化などの大きな機能追加など、バージョン1.4から100件を超える変更が行われています。2016年10月現在、アルファ版として1.5a2がリリースされており、正式版は11月末ごろにリリースされる予定です。便利になったバージョン1.5を楽しみにお待ちください。

注A) これまでどおり`make latexpdfja`も利用できます。

注B) この設定は、latex_elements["pxunit"]オプションで変更できます。

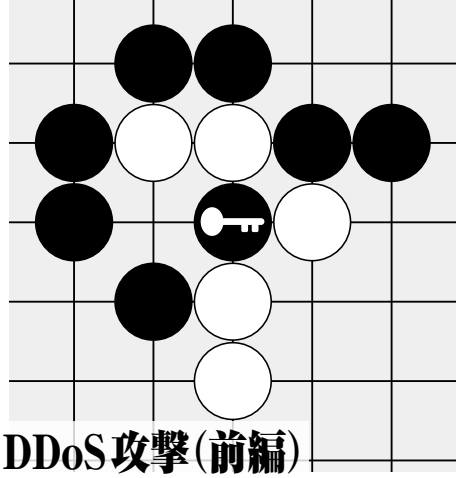
▼リストA　参照(:ref:など)を太字にする設定

```
latex_elements = {
    'preamble': '¥¥renewcommand{¥¥sphinxcrossref}[1]{¥¥textbf{#1}}'
}
```

セキュリティ実践の基本定石

すずきひろのぶ
suzuki.hironobu@gmail.com

みんなでもう一度見つめなおそう



【第三八回】IoT機器を使った過去最大規模のDDoS攻撃（前編）

IoTのセキュリティに関しては、本連載でも過去に何度も取り上げてきました[注1]が、ついに懸念していた大規模なIoTデバイスの乗っ取りが発生し、それがDDoS攻撃に使われるという事例が現れました。その攻撃のトラフィック量は1Tbpsを越えたという報告もなされています。今号と次号の2回に分けてその詳細について取り上げます。

記録的なDDoS攻撃

筆者がこの事件を最初に目にしたのは、世界で3本の指に入る巨大ホスティング企業OVH社の創業者兼CTOであるOctave Klaba氏（@olesovhcom）が、2016年9月22日につぶやいたツイート（**図1**）でした。

図1に記載されているのは、2016年9月20日1時40分47秒にOVH社のサーバで記録された攻撃トラフィックですが、パケット数では9,300万パケット/秒、転送量では799Gbpsという今まで見たことのない数字でした。

これまで本連載[注2]で取り上げた、NTPのmonlist機能を使ったDDoS攻撃では400Gbpsでした。今回はそれの約2倍の量で攻撃されるという状況になっています。

続くツイートによれば、インターネットにつながれた約145,000台のカメラで構成されたボットネットからの攻撃で、IPアドレスあたり1～30Mbpsが計測されているので、理論上は1.5Tbpsを上回るトラフィックを発生させることが可能であると説明しています。

インターネット接続の監視カメラ／ビデオレコーダー

インターネットに接続しているIoTデバイスを見つけてしまうSHODANのサービスについては、こ

◆**図1　Octave Klaba氏の2016年9月22日のツイート**

Octave Klaba / Oles
@olesovhcom
フォローする

Last days, we got lot of huge DDoS. Here, the list of "bigger that 100Gbps" only. You can see the
simultaneous DDoS are close to 1Tbps !

翻訳を表示

```
log /home/vac/logs/vac.log-last | egrep "pps\|.....
bps" | awk '{print $1,$2,$3,$6}' | sed "s/ /|/g" | cut -f
1,2,3,7,8,10,11 -d '|' | sed "s/.....bps/Gbps/" | sed
"s/.....pps/Mpps/" | cut -f 2,3,4,5,6,7 -d ":" | sort | g
rep "gone" | sed "s/gone|//"
Sep|18|10:49:12|tcp_ack|20Mpps|232Gbps
Sep|18|10:58:32|tcp_ack|15Mpps|173Gbps
Sep|18|11:17:02|tcp_ack|19Mpps|224Gbps
Sep|18|11:44:17|tcp_ack|19Mpps|227Gbps
Sep|18|19:05:47|tcp_ack|66Mpps|735Gbps
Sep|18|20:49:27|tcp_ack|81Mpps|360Gbps
Sep|18|22:43:32|tcp_ack|11Mpps|136Gbps
Sep|18|22:44:17|tcp_ack|38Mpps|442Gbps
Sep|19|10:13:57|tcp_ack|10Mpps|117Gbps
Sep|19|11:53:57|tcp_ack|13Mpps|159Gbps
Sep|19|11:54:42|tcp_ack|52Mpps|607Gbps
Sep|19|22:51:57|tcp_ack|10Mpps|115Gbps
Sep|20|01:40:02|tcp_ack|22Mpps|191Gbps
Sep|20|01:40:47|tcp_ack|93Mpps|799Gbps
Sep|20|01:50:07|tcp_ack|14Mpps|124Gbps
Sep|20|01:50:32|tcp_ack|72Mpps|615Gbps
Sep|20|03:12:12|tcp_ack|49Mpps|419Gbps
Sep|20|11:57:07|tcp_ack|15Mpps|178Gbps
Sep|20|11:58:02|tcp_ack|60Mpps|698Gbps
Sep|20|12:31:12|tcp_ack|17Mpps|201Gbps
Sep|20|12:32:22|tcp_ack|50Mpps|587Gbps
Sep|20|12:47:02|tcp_ack|18Mpps|210Gbps
Sep|20|12:48:17|tcp_ack|49Mpps|572Gbps
Sep|21|05:09:42|tcp_ack|32Mpps|144Gbps
Sep|21|20:21:37|tcp_ack|22Mpps|122Gbps
Sep|22|00:50:57|tcp_ack|16Mpps|191Gbps
You have new mail in /var/mail/root
```

（https://twitter.com/olesovhcom/status/778830571677978624）

注1）第16回「IoTのセキュリティについて考える」（本誌2015年1月号）
第27回「同じ轍を踏まないために。IoT時代に向けてできること」（本誌2015年12月号）
第34回「電力施設から家電クーラーまでセキュリティを考える時代」（本誌2016年8月号）
注2）第10回「根深くはびこるDDoS攻撃の脅威」（本誌2014年4月号）

れまでも本連載でたびたび取り上げてきました。その際に、ネットワーク接続の監視カメラが見つけられ、そして、なんらかの方法で認証を回避されて乗っ取られてしまう危険性についても指摘してきました。

現実に、監視カメラ／ビデオレコーダーが乗っ取られ、それがボットネットを構成する端末となりDDoSに使われているという報告が、すでにSUCURIBLOG（セキュリティ企業Sucuri社が運営するブログ）で説明されています[注3]。これによれば、監視カメラ／ビデオレコーダーが乗っ取られWebサーバに対して毎秒35,000～50,000回アクセスをする攻撃のノードとして使われたということです。

Digital Video Recorder

監視カメラ／ビデオレコーダーについて少し詳しく説明します。防犯カメラを店内に配置し、それをビデオレコーダーに記録する装置がDVR（Digital Video Recorder）です。おもにアナログカメラ（CCDカメラ）からの入力をデジタル化し、それをデジタルで記録する機材で、今日では防犯カメラのキットとして一般家庭、事務所、駐車場、店舗、倉庫などに幅広く導入されています。

インターネット上でマニュアルが公開されているDVRの機種を参考に、これはという特徴を抜き出してみると次のようになります。

- **操作などのためにアクセスするにはパスワードが必要**
- **ネットワーク接続が可能である**
- **監視ソフトウェアを別途導入することで、複数のDVRをネットワーク経由で集中的に管理できる**
- **ソフトウェアのライセンスから中身はGNU/Linuxとわかる**

取り扱い説明書を見るとわかりますが、管理者のアカウントとパスワードが機種によってすべて同じです。あと別途購入する中央監視ソフトを使えば、最大16台までDVRをネットワーク経由で接続し管理可能となっています。

日本の国内代理店が用意している日本語マニュアルでは、LANでのセットアップについてのみ説明しています。インターネット側からのアクセスに関しては言及していません。

ただし、メーカーのサイトにある海外向けマニュアル（オリジナル）では、DVRをインターネット側からアクセスできるように地域的／地理的に広範囲に設置し、それを中央監視ソフトで切り替えて監視できるという、低コストで便利に使えるシステムの使い方を説明していました。

もちろんVPN（Virtual Private Network）などいっさいなしで、直接インターネット側からDVRにアクセスできることを前提としています。

この機種のネットワーク設定マニュアルを見てみると、とても興味深いことがわかりました。

- **内部に固有コードを持っており、それをホスト名にしてDDNS（ダイナミックDNS）に自動登録することが可能**
- **ルータの設定でポートフォワードを行いインターネット側からDVRへ直接アクセスできるように設定できる**

マニュアルには、「DVRはHTTPサーバとして動作するので、セキュリティのためにデフォルトの80ポートから別なポート番号に変更するように」と説明書きはありますが、世の常として管理者はデフォルトのパスワードは変更せず、ポート番号も80のまま運用しているのが多数でしょう。よくありがちなのは、業者が最低限の設定だけして設置し、そのままユーザが使うというようなモデルだと思います。

攻撃元の統計

さて、インターネット接続された監視カメラ／ビデオレコーダーから成るボットネットの攻撃ノードはどこの国にあるのか、というのをSUCURIBLOGは先ほど紹介したサイト[注3]で分析しています。

注3）Large CCTV Botnet Leveraged in DDoS Attacks
https://blog.sucuri.net/2016/06/large-cctv-botnet-leveraged-ddos-attacks.html

- 台湾：24%
- 米国：12%
- インドネシア：9%
- メキシコ：8%
- マレーシア：6%
- イスラエル：5%
- イタリア：5%

（以下略）

先ほど説明したDDNSのサーバの設置場所が台湾です。つまり台湾のベンダのDVR装置だというのがだいたい想像つきますし、また、ご当地ですからそれだけ多くの設置数となっているのでしょう。

台湾の出荷先として東南アジアが入っており、また、防犯のニーズの高い地域が入っているという感じでしょうか。

興味深いことに、国別リストの上位には「香港」「日本」「中国（本土）」というのはありませんでした。日本は国内ベンダが充実しているので、台湾の機種を入れるというのは少ないのかもしれません。また、先ほど紹介したように、オリジナルのマニュアルではインターネット側からのアクセスを想定しているのに、日本国内向けのマニュアルではあくまでもローカルなネットワークで利用する範囲で説明していたのも一因かもしれません。

IoTをボット化するシステム

145,000台ものIoT機器からなるボットネットは、脅威以外の何者でもありません。これだけの数のIoT機器を1台1台人間が手間暇かけて集めたわけではないのはわかります。ここまで大量のIoT機器を乗っ取るためには自動化するしかありません。どのような手法やマルウェアだったのか調べようと思ったところ、なんと、そのシステムのコードがGitHubで公開されていました[注4]。作者の名前は“Anna-senpai”、このボットネットのシステムのコード名は“Mirai”です。名前からわかるように明らかに日本のアニメに影響を受けています。

GitHubで公開された文章を読んでみると、Anna-senpaiは、これまでDDoSのプロフェッショナルとして仕事をしてきたこと、これを機にDDoSから身を引くことを表明しています。それらの経緯や、コードの分析からわかった攻撃方法の詳細については、次回に詳しく紹介します。

なぜ無防備なIPカメラが出てくるのか考えてみる

監視用IPカメラやDVRといった機材は建屋内外の配線や設置などがありますから、自分で取り付ける方はごく少数で、専門の設置業者に発注して付けてもらう形になるはずです。

そして、外部からの攻撃や乗っ取りなどの脅威への対応は設置業者の作業範囲ではないですから、ルータもカメラもデフォルト設定のまま設置していったとしても設置業者の責任は問えません。家電がそうであるように、機材を置くべき場所に設置し、そしてマニュアルのデフォルトどおりに設定すれば終了です。

ベンダのマニュアルには、パスワードが変更できることや、ポート番号は安全のために変更することがきちんと記載されています。しかし、すでに設置され動いており、使える状態にあるのに、セットアップのための説明書を読んで設定を変更するユーザはほとんどいないでしょう。下手にポート番号を変更すると、NAT（Network Address Translation）をかけているようなルータなどの設定内容と齟齬（そご）が出るので、たぶん、そのままにするでしょう。

ベンダは最低限、一律なパスワードなどは避けて、デフォルトでハードウェアひとつひとつに異なるパスワードを設定すべきでしょう。実際に日本国内で作られている家庭やSOHO向けのルータは、デフォルトで個々に違う値となっています。しかし、海外でこれまでに作られたものは、「ユーザの責任でセットアップをすること」というスタンスで、ベンダは責任をユーザに預けています。しかし、こ

注4）https://github.com/jgamblin/Mirai-Source-Code/

こまで見たように、結果的には責任は誰も持たず、たらい回しにされているだけです。

このような状況が、今回のDDoS攻撃が効果的に実現できた背景へとつながったのではないかと筆者は考えています。

US-CERTからの注意喚起

2016年10月14日に米国政府のCSIRTであるUS-CERTから、ボットネット“Mirai”のソースコードの公開のタイミングで、「Miraiやほかのボットネットが引き起こす深刻なDDoS脅威について」というタイトルの注意喚起が発行されました[注5]。

今回の“Mirai”が引き起こした1Tbpsを越えるトラフィック量でのDDoS攻撃は、これまでで最大というだけではなく、政府レベルの組織であっても、まともに攻撃を受けるとネットワークが長期間に渡り麻痺するレベルの大きな脅威であることは間違いないようです。

Anna-senpaiは何を目的にしていたのか

“Mirai”の最初の攻撃は、サイバー空間においては兵器クラスの破壊力を持っていたと言っても過言ではないでしょう。また、“Mirai”は、これまで言われてきたIoTセキュリティの問題を具現化したものです。新しい技術は導入していないけれども、技術をうまくまとめることに関してはたいへん良くできています。コード難読化などを見ると、この手の技術に非常に熟練しています（このあたりのコードの特徴については、次回に詳しく解説します）。

しかし、冷静に考えてみると、歴史的とも言えるDDoS攻撃の先は何だったかというと、Brian Krebs氏というITセキュリティを専門とするジャーナリストのブログ（**図2**）[注6]や、DDoS対応には世界でもトップクラスの実績を持つ巨大ホスティング企業のサイトでした。

銀行でもなく、政府機関でもなく、巨大イベントでもなく、ましてや軍でもなく、選んだ先はある意味、Anna-senpaiと同じ分野とも言えるプロフェッショナルです。

そして、その被害者の立場にいる人が触媒になり、IoTセキュリティの問題点と、今回の巨大DDoS攻撃の脅威をリアルタイムで詳しく情報を流してくれました。それが波紋のように広がりIT系メディアが取り上げ、米国政府のCSIRTから注意喚起がなされ、そして、今、筆者がこのことを原稿に書いています。

これまで本連載でもIoTのセキュリティやDDoS攻撃の脅威、ボットネットについて取り上げてきました。しかし、どんなに言葉をかさねようと、今回の“Mirai”のような具体的な事象と、その手法をソースごと公開するインパクトには負けます。

そう考えるとAnna-senpaiは、このDDoS業界から足を洗うにあたり、最後の警告を残していったのかもしれません。SD

◆図2　Brian Krebs氏のブログへのサイバー攻撃を伝えるBusiness Insiderの報道

BUSINESS INSIDER　ENTERPRISE

Akamai kicked journalist Brian Krebs' site off its servers after he was hit by a 'record' cyberattack

Paul Szoldra　Sep. 22, 2016, 7:05 PM　58,523　1

FACEBOOK　LINKEDIN　TWITTER　EMAIL　PRINT

The cloud-hosting giant Akamai Technologies has dumped the website run by journalist Brian Krebs from its servers after the site came under a "record" cyberattack.

"It's looking likely that KrebsOnSecurity will be offline for a while," Krebs tweeted Thursday. "Akamai's kicking me off their network tonight."

Since Tuesday, Krebs' site has been under sustained distributed denial-of-service, or DDoS, a crude method of flooding a website with traffic to deny legitimate users from being able to access it. The assault has flooded Krebs' site with more than 620 gigabits per second of traffic — nearly double what Akamai has seen in the past.

Brian Krebs. (brian Krebs/@briankrebs

To put it more plainly: It's the digital equivalent of jamming a bunch of gunk into a drain pipe. Eventually, water won't be able to pass through.

Websites targeted by this type of attack typically go down for a short period and then come back online. And for hosts, the attacks mean shifting resources to different servers to mitigate the damage.

"I can't really fault Akamai for their decision," Krebs added. "I likely cost them a ton of money today."

（http://www.businessinsider.com/akamai-brian-krebs-ddos-attack-2016-9）

注5）Heightened DDoS Threat Posed by Mirai and Other Botnets　https://www.us-cert.gov/ncas/alerts/TA16-288A
注6）冒頭で紹介したOVH社へのDDoS攻撃に先立って、Brian Krebs氏のブログにも、同様の攻撃がありました。

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第37回 ❖FreeBSD 11.0登場

◉著者プロフィール
後藤 大地（ごとう だいち）
BSDコンサルティング（株） 取締役／（有）オングス 代表取締役／FreeBSD committer
エンタープライズシステムの設計、開発、運用、保守からIT系ニュースの執筆、IT系雑誌や書籍における執筆まで幅広く手がける。カーネルのカスタマイズから業務に特化したディストリビューションの構築、自動デプロイ、ネットワークの構築など、FreeBSD/Linuxを含め対応。BSDコンサルティングでは企業レベルで求められるFreeBSDの要求に柔軟に対応。

FreeBSD 11.0-RELEASEの特徴

FreeBSDリリースエンジニアリングチームは2016年10月10日（協定世界時）、FreeBSDの最新版となる「FreeBSD 11.0-RELEASE」を公開しました。amd64版、i386版、powerpc版、powerpc64版、sparc64版、armv6版、aarch64版が用意されたほか、Amazon EC2、Google Compute Engine、HashiCorp/Atlas Vagrantで利用できます。

FreeBSD開発版に取り込まれた新機能や新しいドライバは、随時FreeBSD 10系にもバックポートされていきましたので、FreeBSD 10.3-RELEASEとFreeBSD 11.0-RELEASEの間にはそれほど大きな違いはありません。ユーザから見た場合にはとくに違いに気がつかないかもしれません。

10系にバックポートされていない機能もいくつかあります。中でもとくに注目されるのは、zfsd(8)デーモンが導入されたあたりです。これはiXsystemsとSpectra Logicのスポンサーシップのもとで取り組まれた成果物で、ホットスペアと自動交換を実施するための機能です。

SolarisのZFSにはホットスペア機能が実装されているそうですが、これまでFreeBSD ZFSではこの機能は使えませんでした。ホットスペア機能というのは、簡単に言うとディスクが壊れたら自動的に控えておいたディスクに入れ替える機能です。この機能が使えると、ストレージの管理がさらに簡単になります。

FreeBSD 11からはzfsd(8)が動作するようになりましたので、故障してもあまり手を入れたくない、システムの再起動ができないといった場合には、あらかじめこのホットスペア機能を仕込んでおいて、壊れた場合に自動的にディスク入れ替えが実施されるようにしておくといったことができます。

FreeBSD Updateでアップグレード

FreeBSD Updateで、この最新版へのアップグレードが可能です。詳しい理由は後述しますが、今回はちょっとアップデートの方法が異なります。OpenSSLのセキュリティ脆弱性が絶妙なタイミングで発表されてしまったために、イレギュラーなアップデートになっています。

まず、次のようにfreebsd-update(8)コマンドを実行してカーネルとユーザランドをアップグレードします。

```
# : > /usr/bin/bspatch
# freebsd-update upgrade -r 11.0-RELEASE
# freebsd-update install
# shutdown -r now
... (略) ...
# freebsd-update install# newaliases
```

freebsd-update(8)を実行する前に、記述にあるように/usr/bin/bspatchの中身を空にすることを忘れないでください。: >は記述間違いでも顔文字でもなく、そのあとに記載されているファイルの中身

各コマンドのうしろについている括弧書きの数字は、manコマンドで見ることができるマニュアルに記載されている章番号を表しています。

を空にするためのリダイレクト指定です。

コマンドにまとめると、このように簡単なものになりますが、マシンによってはファイルのダウンロードからシステムへの適用まで長い長い時間がかかります。最近のマシンだとけっこうサクッと終わりますが、IoT系のデバイスなどだとかなり時間がかかります。

また、作業中にインタラクティブ形式で/etc/の下の設定ファイルのいくつかを編集するように求められると思いますので、それも指示に従って編集してください。

FreeBSDはサーバとして運用されていることが多いと思いますので、ssh(1)経由でログインしてこうしたアップグレード作業をすることが多いと思いますが、アップグレードに長い時間がかかるため、途中でssh(1)の接続が切れることがあります。そうやって何度もやりなおしをすることになると、かなりのストレスになります。

ssh(1)が途中で切れてお話にならないという場合、次の設定を~/.ssh/configファイルに追加して接続が切れないようにしてみてください。

```
Host *
    ServerAliveInterval     120
```

または、次のような設定を~/.ssh/configに追加して、2つのターミナルからログインして作業を行うという方法もお勧めです。

```
Host *
    ControlMaster auto
    ControlPath ~/.ssh/sshmux-%r@%h:%p
    ControlPersist 10
```

この設定を行った場合、2つめのターミナルから同じホストにログインしたセッションには、1つめのターミナルからログインしたときに確立したセッションが使われるようになります。最初にログインしたほうのターミナルではfreebsd-update(8)を行い、2つめのターミナルではtop(1)を実行し続けるといった操作をします。こうするとモニタリングもできますし、ssh(1)接続も切れにくくなりますのでお勧めです。

カーネルとユーザランドをアップグレードしてシステムを再起動したあとは、次のようにしてパッケージをすべてFreeBSD 11.0向けのバイナリに入れ替えます。バイナリが入れ替わりますので、すべてのアップグレードが終了したらもう一度システムを再起動しておきます。

```
# pkg-static install -fy pkg
# pkg upgrade -y
# freebsd-update install
# shutdown -r now
...（略）...
```

10系の段階でビルドしたバイナリを使い続ける必要があるとか、バイナリを再構築しないで使い続ける必要があるといった場合には、次のように10系互換ライブラリをインストールしてみてください。

```
# pkg install -y compat10x-amd64-10.3
```

FreeBSD 11.0-RELEASEへのアップグレードが完了していれば、**図1**のようにuname(1)やfreebsd-version(1)でバージョンを確認できます。

とくにパッチレベルが1以上になっていることを確認してください。FreeBSD 11.0-RELEASEのリリースは「FreeBSD 11.0-RELEASE」ではなく

▼図1　FreeBSD 11のバージョンを確認

```
# uname -a
FreeBSD virt.ongs.co.jp 11.0-RELEASE-p1 FreeBSD 11.0-RELEASE-p1 #0 r306420: Thu Sep 29
01:43:23 UTC 2016     root@releng2.nyi.freebsd.org:/usr/obj/usr/src/sys/GENERIC  amd64
% freebsd-version -uk
11.0-RELEASE-p1
11.0-RELEASE-p1
#
```

チャーリー・ルートからの手紙

「FreeBSD 11.0-RELEASE-p1」となっています。

パッチレベルがリリースの対象となることは、普通ないことです。今回がイレギュラーなのですが、今回はこのバージョンが対象となってしまったので、確実にこのパッチレベル以降になっていることを確認してください。

なぜパッチレベルがリリースの対象に?

パッチレベル(-p?)はセキュリティ脆弱性を修正したり、不具合を修正した場合などに提供されるものです。

たとえばFreeBSD 10.3であれば、FreeBSD 10.3-RELEASEがリリース時のバージョンであり、以降セキュリティ脆弱性が修正されるごとに、「FreeBSD 10.3-RELEASE-p1」「FreeBSD 10.3-RELEASE-p2」といったようにパッチレベルが増えたバージョンが公開されていきます。

メジャーアップグレードバージョンの最初のリリース、つまり今回のような「FreeBSD 11.0-RELEASE」のリリースには、当初のリリーススケジュールから大幅に遅延して公開される傾向がありましたが、今回はそれに比べるとかなりスケジュールどおりにリリース作業が進みました。

2016年9月の中頃にはイメージファイルやパッケージのビルドも完了し、ミラーサーバへの配布も完了、あとはリリースアナウンスを実施するだけといった状況になりました。アナウンス前でしたが、すでにfreebsd-update(8)でアップグレードが可能でしたので、この段階でFreeBSD 11.0-RELEASEへアップグレードした方も少なからずいたようです。

しかしリリース直前になって、OpenSSLのセキュリティ脆弱性が発表されます。FreeBSDはベースシステムにOpenSSLを取り込んでいますので、このままリリースすると、セキュリティ脆弱性を抱えたままのOpenSSLを含んだ状態でリリースを出すということになります。

リリースエンジニアリングチームは、それは避けるべきと判断してリリースを延期しました。OpenSSLのセキュリティ脆弱性を修正し、ビルドをやりなおし、パッケージも新しくビルドしなおしました。このため、リリースの対象となったのは「FreeBSD 11.0-RELEASE」ではなく「FreeBSD 11.0-RELEASE-p1」だったのです。アップグレードの方法がいつもとちょっと違うのは、このためです。

リリースアナウンスの前にFreeBSD 11.0-RELEASEへアップグレードしてしまったという場合には、次のように作業してFreeBSD 11.0-RELEASE-p1へアップグレードする必要があります。

```
# : > /usr/bin/bspatch
# freebsd-update fetch
# freebsd-update install
# shutdown -r now
```

パッケージも再構築されましたので、次のようにアップデートしておきましょう。

```
# pkg update
# pkg upgrade -y
```

OpenSSLのセキュリティ脆弱性であるHeart Bleedが発見されて以来、OpenSSLにはセキュリティ脆弱性の発見が続いています。こういったソフトウェアをベースシステムにマージした状態にしておくのは避けるべきではないか、という考えが出てくるのも当然です。

実際、OpenBSDはすでにLibreSSLを採用していますし、FreeBSDベースまたはFreeBSDから派生したディストリビューションであるTrueOS(旧:PC-BSD)、HardenedBSDおよびDragonFly BSDはすでにLibreSSLをデフォルトのライブラリとして採用し、OpenSSLをデフォルトでは利用しないように変更しています。

FreeBSDは、こうした変更にはかなり慎重なほうですので、いますぐにOpenSSLからLibreSSLへの変更が実施されるということはなさそうです。しかし、利用するSSLライブラリの実装系を入れ替えできるようにする取り組みが進められているなど、パッケージも含めて対応は進められています。

今後のリリースで、OpenSSLがベースから外れるときが来る可能性もあります。

最低5年間セキュリティサポートモデルへ変更

FreeBSD 11.0-RELEASEからはセキュリティモデルが、これまでのリリースバージョンベースでの1年間または2年間といったものから、ブランチベースで最低5年間というものに変更となります。これには、長期セキュリティサポートを明言するとともに、セキュリティサポートチームのリソースをより効率化する狙いがあります。

従来のセキュリティサポートモデルでは、リリースバージョンごとにデフォルトで1年間、拡張で2年間のセキュリティサポートが提供されてきました。つまりこれは、FreeBSDセキュリティチームがサポートすべきバージョンが、常に数個存在するという状況を作り出してきました。

FreeBSD 11.0-RELEASE以降の新しいセキュリティサポートモデルは、ブランチベースで最低5年間はセキュリティサポートを提供するといったものに変わります。サポートするバージョンはブランチでの最新版です。つまり、FreeBSD 11.1-RELEASEがリリースされてからは、FreeBSD 11.0-RELEASEに対してはセキュリティアップデートは提供されなくなります。常にそのブランチで最新のリリース版がセキュリティアップデートの対象となります。

新しいモデルでは、新バージョンがリリースされた場合、3ヵ月間の移行期間を設けるとされています。こうやって常に最新のブランチ版に更新し続けることで、セキュリティアップデートを適応していくというモデルに変わります。バイナリアップデートが主流になりつつあって、マイナーバージョン間のアップグレード負荷が軽くなっている現状では、これはかなり現実的なセキュリティサポートモデルです。

FreeBSD 11.1-RELEASEは、カーネルとユーザランドのアップデート作業はさらに細かく、かつ簡単に実施できるようになる見通しです。だいたい5年間をサポート期間と考えて、サーバやサービスの構築を検討すると良いのではないかと思います。

カーネルとベースシステムのパッケージ化は11.1から

FreeBSD 11.0を目処にカーネルとユーザランドをパッケージで管理する取り組みが進められてきました。

この取り組みは「pkgbase」と呼ばれていますが、最終的に解決すべき課題がいくつかあり、11.0のリリースには間に合わないということで、11.0-RELEASEへの採用は見送られました。11.1-RELEASEで導入される見通しです。

カーネルとユーザランドをパッケージ化するというのは、言葉のとおりpkg(8)で管理できる対象にするということです。これまでカーネルとユーザランドのアップグレードにはfreebsd-update(8)を使ってきたわけですが、これがpkg(8)で実施できるようになります。これまでよりもアップグレード作業が簡単になります。

この取り組みはFreeBSDベースのディストリビューションであるTrueOSではすでに実現されています。TrueOSはFreeBSD-CURRENTをベースにカーネルもユーザランドもパッケージも、すべてpkg(8)経由でローリングリリース的にアップデートするといったしくみへ移行しました。UNIX系のディストリビューションとしては、かなり先進的な取り組みへ移行したと言えます。

このため、もしかするとFreeBSD 11.0から11.1へのアップグレード作業も、ちょっとイレギュラーな操作になる可能性があります。そういった最新の情報は、Gihyo.jpの連載「BSD界隈四方山話」[注1]に掲載していきますので、ぜひそちらをご覧ください。SD

注1 http://gihyo.jp/admin/serial/01/bsd-yomoyama

41

Debian Developer　やまねひでき　henrich@debian.org

GNOME、Perlほか、パッケージ取り込みの近況

Debian Hot Topics

GNOME 3.22リリース

9月21日に、GNOME 3.22がリリースされました。Debianでは、GNOME 3.22のβ版であるバージョン3.21.xのパッケージが、3.22のリリースを待たずに続々とunstableに投入されており、3.22がリリースされた直後にはほぼ出そろったかたちとなりました。現状、testing/unstableともに3.22の投入がほぼ完了した状態です[注1]。Debian 9“Stretch”ではこのGNOME 3.22が使われます。

この記事の執筆時点では、設定ユーティリティである「gnome-control-center」や、Windowsのエクスプローラー、MacのFinderであるところの「ファイル」アプリケーションなどの一部のアプリケーションでは、残念ながら日本語訳が間に合っていないところがあり、英語のままになっていますが、日常の使用には問題はないでしょう。今後の翻訳作業に協力したいという方は、「https://l10n.gnome.org/」を参照してください。

GNOME 3.22の変更点ですが、ドラマティックに変わったところはなく、既存のGNOME 3の細かな使い勝手を磨き上げています。

たとえば、「ファイル」アプリケーションでは、ファイル名を連番で付け直すことが簡単になりました。複数ファイルを選択した状態で右クリックし「Rename」を選択する（**図1**）と、「どのようにファイル名を付け直すのか」の確認画面が出てきます。**図2**は、[001,002,003]という指定で連番を付けようとしている様子です。画面右のリネーム後には、ファイル名の先頭に00xという数字が入っていることが確認できます。

図3は、ファイル名の「Disc」という文字列を

▼図1　複数ファイルを選んで右クリックし「Rename」を選択

▼図2　[001,002,003]という指定で、ファイル名に連番を付けられる

注1）URL https://www.0d.be/debian/debian-gnome-3.22-status.html

「でぃすく」に置き換えようとしているところです。ターミナルからシェルでサッと作業できる事柄ではありますが、GUIで変更後の状態を確認しつつ実行できるのは良いですね。

また、圧縮機能はアーカイブ形式が、zip、tar.xz、7zの3種類から選べるようになっています(**図4**)。

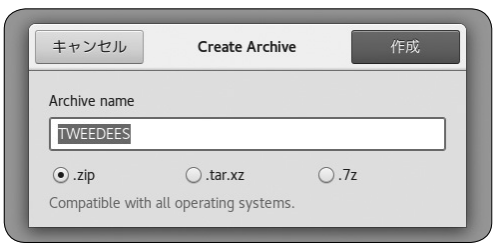
Perl 5.24投入

最新のPerl 5.24が、Debianにやってきました[注2]。これがunstableに投入されたことで、パッケージングのコアツール周りに依存関係の問題が生じ、2日間ほど各種パッケージのビルドができない状態が続きました[注3]。しかし、5.24対応のための数百パッケージの再ビルドが無事に終わり、現在は元どおりになっています。Perl 5.24では内部コードが改善され、パフォーマンスが良くなっているそうです。

他方、ビルド自体は通っているものの互換性が破壊されていて動かなくなっているモジュールも、多少ながら出てくるでしょう。Debian 9 "Stretch"のリリースまでには完全に安定した状態になることを期待しましょう。

▼図3　ファイル名の「Disc」を「でぃすく」に一括置換

▼図4　圧縮形式を選択できる



ほかのディストロからの取り込み

Debianでは、ほかのディストリビューションのプロダクトでも、有用な場合は取り入れることがあります。今回は現在作業が進行しているopenSUSE由来の「snapper」と「openQA」を例にして、その活動を取り上げてみます。

snapper

snapper[注4]はファイルシステムのスナップショットを取りやすくするツールです。たとえば、何かの作業前にスナップショットを取っておき、「やっぱり、なかったことにしたい！」という場合には復元する、などという作業が可能です。VirtualBoxなどの仮想マシンではお馴染みの機能ですね。

Btrfsはスナップショットがファイルシステムの機能としてありますし、LVM(Logical Volume Manager)もシンプロビジョニングという機能を使うことでスナップショットを取ることができます。

ただ、BtrfsもLVMも、スナップショットを取ろうとするとサブコマンドやオプションの指定が結構複雑で面倒です(**表1**)。これをいい感じに楽にしてくれるのが、snapperというわけです。現在、snapperを使えるファイルシステム、ディストリビューションとしては**表2**のようなものがあります。

snapperの導入と利用は、対応しているファイルシステムであるLVMシンプロビジョニングやBtrfsを使っていれば非常に簡単です。パッ

注2) URL https://bugs.debian.org/830200
注3) 名前どおり「unstable(不安定版)」の宿命ですね。
注4) URL http://snapper.io/

ケージのインストール後、標準の設定ファイルをcreate-configコマンドで作成し、設定ファイルを編集します。そして、snapper自体の実行を管理者権限なしでもできるように、実行可能なユーザを追加します。

```
パッケージのインストール
$ sudo apt install snapper
設定ファイルの生成
$ sudo snapper create-config /
設定ファイルの編集
$ sudo vi /etc/snapper/configs/root
```

スナップショットを取るには「snapper create」だけですのでお手軽です[注5]。スナップショットが取れたかどうかは「snapper ls」で確認できます。

スナップショット間で変更されたファイルが何かを確認するには、「snapper status 5..6」のように実行します(ここではスナップショット番号5と6の間の差分を確認することになります)。すると**図5**のように表示されます。

先頭が「+」となっているのは追加されたファイル、「-」となっているのは削除されたファイル、「c」となっているのは変更されたファイル(changeのc)です。ここでは毎度出力されてほしくない/var/log以下のファイルも出力されてしまっていますが、/etc/snapper/filters/に「/var/log/*」と書かれたlog.txtファイルを置くことでsnapperの出力から除くことができます[注6]。/etc/snapper/filters/にはほかにもファイルがあるので、参考にしてください。

具体的な変更内容は、(バイナリファイルでなければ)「snapper diff」で確認できます。

```
$ snapper diff 105..106 /etc/systemd/journald.conf
```

必要に応じて「snapper undochange」で変更

注5) さらに、初期設定が済むと、パッケージの追加と削除を実行する前後で、自動でスナップショットを取るようになっています。

注6) あくまでもsnapperの出力から見えなくなるだけで、スナップショット自体は取ることになります。スナップショット自体を取らないようにするには、LVMシンプロビジョニングでは別々のパーティションとして分割する、Btrfsではサブボリュームを分ける必要があります。

▼図5　スナップショット間の差分表示

```
+..... /usr/games/LS
+..... /usr/games/sl
+..... /usr/games/sl-h
+..... /usr/share/doc/sl
+..... /usr/share/doc/sl/README
(..中略..)
c..... /var/lib/dpkg/status
c..... /var/lib/dpkg/status-old
c..... /var/log/apt/history.log
c..... /var/log/apt/term.log
c..... /var/log/dpkg.log
c..... /var/log/kern.log
c..... /var/log/messages
c..... /var/log/snapper.log
c..... /var/log/syslog
c..... /var/tmp/snapper-apt
```

▼表1　標準コマンドとsnapperの比較

ファイルシステム／ツール	スナップショット取得の実行例
Btrfs標準コマンド ※1	# btrfs subvolume snapshot / /snapshot01
LVMシンプロビジョニング ※2	# lvcreate -s --thinpool vg001/pool origin_volume --name snapshot01
snapper	# snapper create

※1　Btrfs標準コマンドの場合は、どのサブボリュームに対してどこにどの名前でスナップショット(=サブボリューム)を作るのかを常に意識して指定する必要がある。ここでは/(ルートディレクトリのサブボリューム)に対して/snapshot01というサブボリュームを作っている。
※2　LVMシンプロビジョニングの場合も、ボリュームグループ名などを含めさまざまなオプション指定が必要。

▼表2　snapperが使えるファイルシステムと代表的ディストリビューションなど

ファイルシステム	ディストリビューション
LVMシンプロビジョニング	Red Hat Enterprise Linux(※3)、CentOS(ともに7から)
Btrfs	SUSE Linux Enterprise Server、Oracle Linux、openSUSE(デフォルト)など
EXT4(一応)	実験的サポート。カーネル側の設定とe2fsprogsユーティリティのパッチが別途必要で、実質的には不可

※3　RHEL 7でのLVMシンプロビジョニングによるsnapperの利用については、Red Hatの森若和雄さんの記事が参考になります。URL https://oss.sios.com/redhat-ch/blog/rhel7snapper

前に戻すなどしましょう[注7]。

```
$ sudo snapper undochange 105..106 /etc/
systemd/journald.conf
```

snapperパッケージは、Debianではアップデートが滞っていたので、筆者がパッケージメンテナを引き継ぎました。Debianパッケージにはbash completionも追加したので、コマンド補完によって使いやすくなっています[注8]。

ただ、SUSEのウリである管理ツール「YaST」[注9]との統合などの機能は、Debianには同等のものがないために、さすがに取り入れられませんでした。このあたりはSUSEに1日の長がありますね。

また、このツールを触っていくうちに、現状のdebian-installer(d-i)でのパーティショニング問題や、GRUB2でのサポート問題も見えてきました。このあたりは、Debian 10"Buster"をにらんで提案をしていこうかと思っています。

この問題について、「なぜにDebian 9での解決を目指さないの？」というと、次のような理由になります。SUSEでは「snapper rollbackコマンドで過去に取得したスナップショットを選んで、そこからロールバックして再起動」という技ができるのですが、これを実現するために結構な数のパッチがブートローダーであるGRUB2に当てられています。これをDebianに入れるのはなかなか骨になります。

また、d-iではパーティションのファイルシステムとしてBtrfsが選べますが、作成したBtrfsにサブボリュームを作成してサブボリュームごとにディレクトリを割り当てることができません(これができるとスナップショット対象が絞り込めてスナップショットの取得サイズを小さくできますし、全体をロールバックして起動するときに運用しているDBのロールバックによるリグレッションも起きなくなります)。

すでにフリーズの足音が聞こえている中で、インストーラに対して具体的なコードを伴わない大きな機能拡張の提案は実現が難しいのです[注10]。ただ、debian-installerチームががんばってくれれば、Debian 9に間に合うかもしれません。

openQA

openSUSEで積極的に新しいパッケージを投入するのに役立っているのが「openQA」というシステムです。こちらは仮想マシンでOSを起動し、OpenCVを使った画像認識をベースに、インストーラと主要GUIアプリケーションのテスト実施とリグレッションが起きていないかを検出するというツールです。openSUSEでは各ビルドの動作確認に利用しています。テストについては動作のビデオやスクリーンショットも取得できるという、中々の優れものです。

Debianでは、openQAを構成するフレームワークの機能の1つ「os-autoinst」については筆者が無事にパッケージ投入を完了しました。

しかし、openQAが依存しているライブラリについて、Selenium部分のバイナリ生成がエラーになってばかりで、まだパッケージ投入の目処がついていません[注11]。

同等のテストが行えるようになれば、今後のDebianの品質向上に役立つと思うので、もう少し作業を続けてみたいと思います。

◆ ◆ ◆

ほかのディストリビューションからの取り込み活動は、いかがでしたでしょうか。ほかにも「○○○には、こんな有用なツールがあるよ！」という情報があれば、ぜひお寄せください。

SD

注7) ここでは、書き戻すファイルの権限が、一般ユーザでは書き込みできない設定になっているので「sudo」を付けて実行しています。

注8) upstreamへのPull Requestもしておいたので、この記事が読まれるころには取り入れられているかもしれません。

注9) openSUSEやSUSE Linux Enterprise Serverで採用されている設定／インストール作業を一元的に行えるツール。

注10) 一応バグレポートという形で提案自体はしておきましたので、気になる方は追いかけてみてください。
URL http://bugs.debian.org/840248

注11) 今後の進捗については次のバグを参照してください。
URL https://bugs.debian.org/839569
URL https://bugs.debian.org/840253

Ubuntu 16.10 とそのフレーバーの変更点

Ubuntu Japanese Team　あわしろいくや

今回は10月13日にリリースされたUbuntu 16.10とそのフレーバーの変更点をお知らせします。

次のLTSに向けた小さな第一歩

思い返せば、Ubuntu 13.10のころから新しいディスプレイサーバであるMirに変更するのかという話題が出ては消えています。最初はMirをX.Orgで動作させたXMirから開始するという話だったのが、13.10でのスキップが決定し、当然LTSである14.04でもスキップし、14.10以降はUnity 8とMirをセットにしてデフォルトにするという話が出ては消え、16.10でも本年5月の段階でデフォルトにはしないという決定が発表されました。しかし、同時にテクニカルプレビューとしてUnity 8のセッションをデフォルトでインストールすることも発表されました。ようやくUnity 8/Mirを手軽に試せる準備が整ったということになります。

そのほかにも変更点はありますが、どれも次のLTSに向けた小さな第一歩という感じであり、16.04と比較してあまり大きな変更点はありません。例外と言えるのはUbuntu GNOMEとUbuntu MATEです。では、それぞれ細かく見ていきましょう。

16.10概要

リリース日とコードネーム

Ubuntu 16.10は、前述のとおり10月13日にリリースされました。コードネームはYakkety Yakで、「かしましいヤク」という意味です。ヤクは野牛だそうです。名前負けというのはおかしいですが、かしましいというよりもおとなしいリリースとなりました。

共通の変更点

GNOME関連パッケージが中核となるものは3.20に、それ以外のアプリケーションは3.22にアップデートされました。過去にはこれほど積極的に最新版のパッケージを採用するということはありませんでした。今回はより積極的な開発が行われたということです。

16.04ではファイル（Nautilus）のバージョンは3.14でしたが、16.10では3.20にアップデートしました。サイドバーにあった「ネットワーク」と「コンピューター」と「サーバーへ接続」がなくなり「他の場所」に統合されたなど、操作に大きな変更点があります。

initデーモンはずいぶん前にUpstartからsystemdに変更されましたが、グラフィカルセッションの起動には依然としてUpstartが使われてきました。16.10ではついにその部分もsystemdに変更されましたが、Unityのインジケーターなどはまだ Upstartが使われています。前者は`systemctl --user status`コマンドで、後者は`pstree`コマンドで確認すると簡単にわかります。

ネットワークの設定を一括して管理するしくみとしてnetplanが採用されました。パッケージ名は

nplanです。これは/etc/netplan/*.yamlにネットワーク関連の設定を保存し、バックエンドであるNetwork Managerとsystemd-networkdを使用して設定の反映をしようとするものです。現在はデスクトップ版UbuntuとUbuntu TouchではNetwork Managerで、それ以外は/etc/network/interfacesファイルで管理しており、そのあたりを統一すべく用意されました。しかし、現段階のnetplanは開発初期バージョンといったところで、実用レベルに達しているとはいい難い状況です。

カーネルのバージョンは4.8です。16.04の4.4からは4回のバージョンアップを経ており、過去のUbuntuはだいたい2〜3回のバージョンアップにとどまっていることを考えると、ジャンプアップと言っていいでしょう。ただしカーネルモジュールが対応していない場合はビルドに失敗しますので、ハードウェアを増設している場合などはご注意ください。

LibreOfficeは5.2にアップデートされました。リリース時点では5.2.2になっています。LibreOffice 5.2に関しては2016年9月号の本連載第77回で詳しく取り上げているので、そちらをご覧ください。

アップグレード方法

16.04をインストールした場合、デフォルトでは16.10へのアップグレード通知は表示されません。「ソフトウェアとアップデート」を起動し、「Ubuntuの新バージョンの通知」を「すべての新バージョン」に変更し、「閉じる」をクリックしてください(**図1**)。

Ubuntuの変更点

Unity 8のセッション

ログイン時にセッションを選択することにより、Unity 8が手軽に試せるようになりました(**図2**)。プロプライエタリなドライバを使用している場合や仮想マシンを使用している場合[注1]はUnity 8、正確にはMirが使用できずにログイン画面に戻ってしまいますので、注意が必要です。

ログインに成功すると「スコープ」が起動します(**図3**)。現状、ここから起動できるアプリケーションが3つとシステム設定があります。端末は起動時にパスワードの入力が求められるのですが、正しいパスワードを入力しても文字が打てるようになりません。一度フォーカスをほかのウィンドウ(というか事実上スコープしかありませんが)に変更し、もう一度端末に戻ってくると文字が打てるようになります。

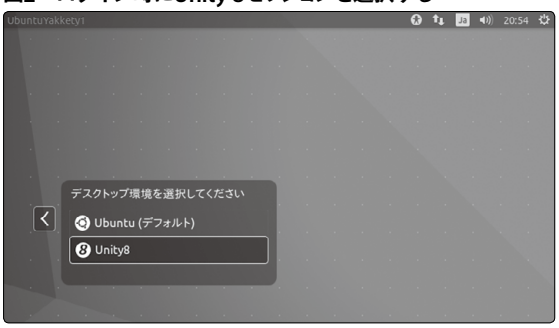
Webブラウザもありますが、そもそも日本語が打てないので検索もおぼつかないのです。

Print Screenキーを押せばスクリーンショットを撮影することはできますが、ファイルマネージャーも画像ビューアもなく、確認できません。また、まれにフリーズしてしまうこともありました。

筆者が普段使用している、検証用のPCに接続しているディスプレイの解像度はフルHDなのですが、スクリーンショットを撮影して記事に掲載するにはいささか解像度が大きく過ぎます。しかし、ディスプレイの設定はないので、結局1,366×768ドットの液晶ディスプレイがあるノートPCにインストール

図1　ソフトウェアとアップデート

図2　ログイン時にUnity 8セッションを選択する



注1) 少なくともVirtualBoxではログインできません。VMware WorkstationはMirに対応しているはずですが、同様にログインできませんでした。

し、スクリーンショットを撮影しました。

このとおり、実用に耐えるとはいい難いのが現状です。プレビューにしても、もう少しなんとかならなかったものかと思いますが、リリース直前のかなりギリギリのタイミングだったのでどうにもできなかったのでしょう。Unity 8のセッションが選択できるようになったのは10月に入ってからですので、もし16.04や15.10のときのように本誌での紹介とリリースのタイミングを合わせていた場合、紹介できませんでした。今回は1ヵ月遅れにしたので、このように紹介できたということです。

おそらく次のバージョンでは、少なくともGTK+ 3のアプリケーションは普通に動くようになっていると思われるので、それに期待しましょう。もちろんUnity 8/Mirをデフォルトに変更するということがあり得るかもしれませんが……。

Kubuntuの変更点

現在KDEはKDE Software Compilationとして3つに分割してリリースを行っています。Plasmaが5.7.5、Frameworksが5.26.0、Applicationsが16.04.3というバージョンを採用していますが、最新版はそれぞれ5.8.0、5.27.0、16.08.1ですので、少し前のバージョンです。

気になる変更点としては、ドキュメントビューアであるOkularがデフォルトでインストールされなくなりました。理由はよくわかりませんが、Frameworks 5.xへのポーティングが完了していないのが原因であると推測できます。PDFやPostScriptファイルはもちろん、EPUBにも対応しているので便利であり、今後の対応が待たれます。

もちろんデフォルトでインストールされなくなっただけであり、パッケージが削除されたわけではないので、必要な場合は“okular”と“okular-extra-backends”パッケージをインストールしてください。

Xubuntuの変更点

Xubuntuは16.10にアップグレードしてもあまり変更点はありません。追加あるいは削除されたパッケージもありません。すぐに気づくのは壁紙が明るい青になったことくらいです。サポート期間を考えると16.04にとどまるのがベストです。

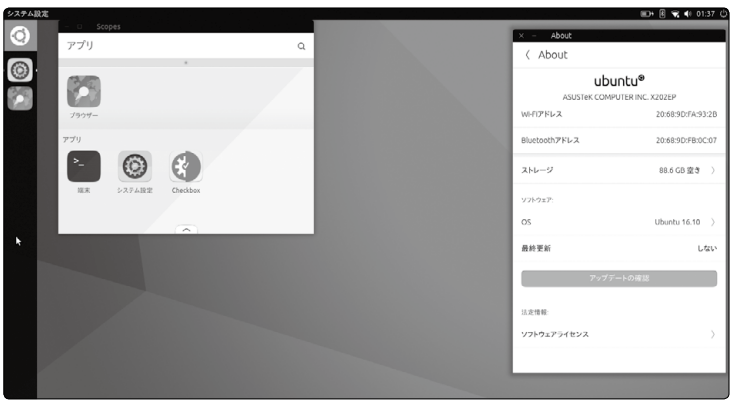
現在XfceはGTK+ 3へのポーティング作業を実施しており、揃いしだい4.14をリリースすると公表しています。すでに完了した作業ももちろんあり、その代表格はxfce4-terminalですが、16.10ではGTK+ 2でビルドしたままです。

Xfce 4.14はいつリリースされるのかは未定です。18.04に間に合うといいなというくらいのスパンで考えておくとよいのではないでしょうか。

Lubuntuの変更点

Lubuntuはついに独自のソフトウェアセンターの

図3　左側にある「アプリ」を表示しているのがスコープ



採用をやめ、ソフトウェア（gnome-software）に変更しました。大きな変更はそのくらいです。

LubuntuはいつLXDEの採用をやめ、LXQtに移行するのかに注目されていました。開発期間中にLXQtに切り替えたインストールイメージを配布するという話は出てきましたが、結局見送られ、現在に至るまでリリースされていません。

もちろん現在のLubuntuにLXQtをインストールする方法もあるのですが（**図4**）、素直にインストールイメージがリリースされるまで待つのがいいのではないかと思います。ほかのLinuxディストリビューションではすでにLXQtを採用しているものがあるので、試したい場合はそちらをインストールするのがよいのではないかと思います。

Ubuntu GNOMEの変更点

前述のとおりGNOMEの中核となるパッケージは3.20に、そうではないアプリケーションはおおむね3.22にアップデートしているため、Ubuntu GNOMEに関しては16.10へのアップグレードに値するリリースといえます。

今回から初回ログイン時に初期セットアップ画面が表示されるようになりました。入力の項目では必ず「日本語（Mozc）」を選択してください（**図5**）。

一部のアプリケーションでは、Ctrl + ? キーあるいはCtrl + F1 キーを押すとショートカットキー一覧が表示されるようになりました（**図6**）。これはファイル（Nautilus）での例ですが、メニューから消滅したようなことでもショートカットであれば実現できます。

Ubuntu MATEの変更点

MATEデスクトップ環境は、9月にリリースされたばかりの1.16にアップデートされています（**図7**）。

図4　LubuntuにLXQtをインストールした例

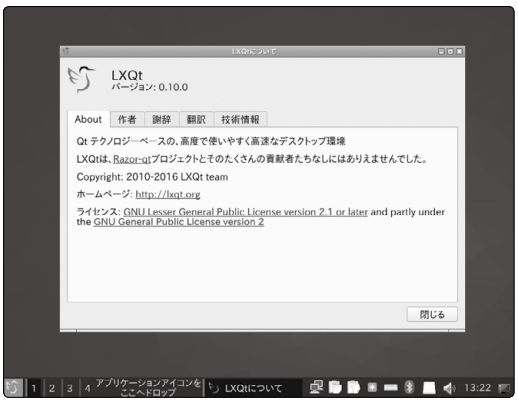
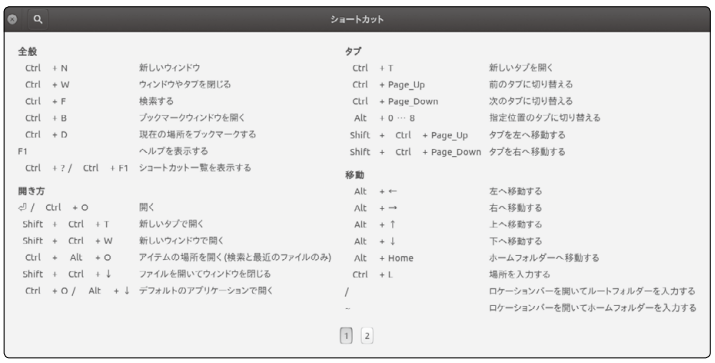


図5　「日本語（Mozc）」を選択する

図6　ファイル（Nautilus）のショートカット1ページ目



このバージョンでGTK+ 3へのポーティングが完了しました。もちろんUbuntu MATEでも積極的にGTK+ 3でビルドを行っています。

MATEはGNOMEから派生したこともあり、もともと軽量志向ではないのですが、GTK+ 3へのポーティングで、ますますメモリを消費するようになり、Raspberry Pi 2/3を含めてメモリ1GBしかないような環境では採用の見送りを含めて検討する必要があるかもしれません。軽く計測してみたところ、16.04では起動直後のメモリ消費量は387MBでしたが、16.10では529MBでした[注2]。

16.04では1.12だったのでかなり大幅なバージョンアップとなりますが、基本的には機能の増減はあまりなく、内部的な変更が多いので違いに気づかないかもしれません。

なお、開発期間中にプロジェクトリーダーのMartin WimpressがCanonicalの社員になったというトピックもありました。

注2）もちろんこれはあくまで参考値であることにご注意ください。

図7　最速でMATE 1.16を採用したUbuntu MATE 16.10

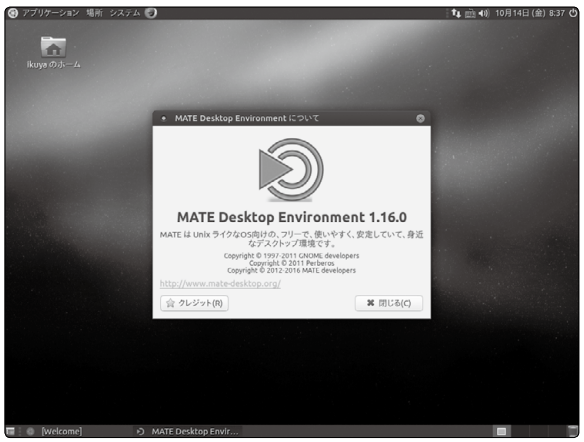
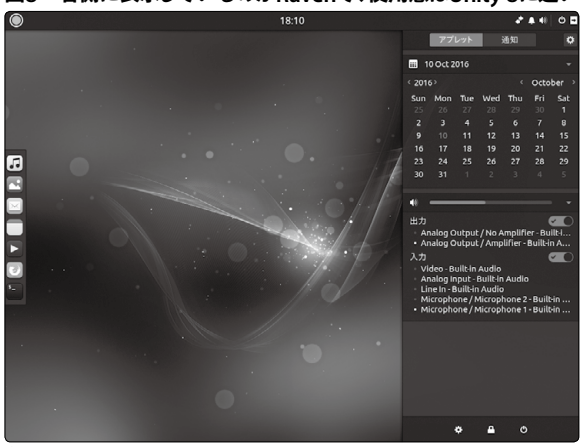


図8　右側に表示しているのがRavenで、使用感はUnity 8に近い



その他：Budgie

16.10からBudgieというパッケージが加わりました。「Budgieデスクトップ環境」と称していますが、実体はGNOMEのデスクトップシェルであるGNOME ShellをBudgieに置き換えたものです。

左上の○アイコンはアプリケーションメニューで、カテゴリごとに並ぶという旧来の方法を提供しています。上右端の矢印アイコンをクリックすると、Ravenというメニューが表示されます（**図8**）。ここでカレンダーや通知やテーマのカスタマイズを行うことができます。

もちろんUbuntu GNOMEにBudgieをインストールしてもいいのですが、Budgie Remix[注3]という非公式フレーバーが公開されているので、こちらをインストールするのが簡単です。

Budgie Remixも16.04ベースと16.10ベースがありますが、今回使用したのは後者です。

Budgie Remixをインストールするとデフォルトで IBusが起動していますが、右上にIBusのアイコンは表示されていません。これはBudgieがアイコンを表示するしくみ（Appindicator）をサポートしていないからです[注4]。IBusを使いたい場合は、**ibus-setup**コマンドを実行して「次の入力メソッド」を「<Control>space」に、「プロパティーパネルを表示する」を「常に表示する」にし、［入力メソッド］タブで「日本語-Mozc」を追加し、「閉じる」をクリックしてください。これでCtrl＋スペースキーを押すとMozcに切り替えることができます。

近い将来Budgie Remixがオフィシャルフレーバーになるようなことがあるかもしれません。SD

注3）https://budgie-remix.org/

注4）https://github.com/solus-project/budgie-desktop/issues/286

技術評論社

現場で使える 実践テクニック みんなのGO言語

注目のプログラミング言語Goを習得するメリットはいくつかあります。シンプルな言語設計のため学習しやすく、整理されたコーディング規約によりチーム開発で運用しやすいこと。マルチプラットフォームに対応し、さまざまな環境へのツールをつくるときに有用であること。インフラ部門のスループットの重い作業の処理速度を並列実行により改善できること、などが挙げられます。CやLL言語（Ruby/Perl/Pythonなど）を使っているのであれば、Go言語を利用しそのメリットを享受できるでしょう。本書で紹介するTipsや利用方法を参考にすれば、Go言語を適材適所で利用するための勘所をつかむことができます。

松木雅幸、mattn、藤原俊一郎、
中島大一、牧大輔、鈴木健太　著
B5判／144ページ
定価（本体1,980円+税）
ISBN 978-4-7741-8392-3

大好評発売中！

こんな方におすすめ

- Go言語を使ってみたい方
- Go言語に携わることなった方
- インフラエンジニア

技術評論社

Xcodeではじめる［Xcode 8 & Swift 3 対応］簡単 iPhoneアプリ開発

本書は、「iPhoneアプリを開発してみたい!」と思う人が、最初に手に取っていただくことを想定した解説書です。難しいことは気にせず、開発ツールである「Xcode」をとにかく説明文のとおりに操作すればアプリを作ることができます。画面上の操作も、1つひとつのステップを掲載しました。プログラミング言語「Swift」のことをまったく知らなくても、iOSアプリを作れます。
iOSアプリの開発は、Swiftのプログラムとストーリーボードでのグラフィカルな設定の組み合わせで成り立っています。本書では、とくにストーリーボードでの作業に重点をおき、極力プログラムを書く量を少なくしています。

正健太朗　著
B5変形判／256ページ
定価（本体2,680円+税）
ISBN 978-4-7741-8422-7

大好評発売中！

こんな方におすすめ

- iPhoneアプリ開発に興味がある人
- お手軽にiPhoneアプリを作ってみたい人

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

bashは、高機能で全部理解するには多くの時間が必要です。まずは、シェルスクリプトを読んでなんとなくわかる、簡単なスクリプトは書ける、といったレベルを目指してシェルスクリプトの基本を探検します。

第 8 回 シェルスクリプトへの入り口

シェルスクリプトを書くにあたり、POSIX準拠についてどうすべきか迷うことがあります。プロジェクトとして要求があれば要求に従うべきです。でも要求がないときはどうしたらよいでしょうか。

> POSIX：Portable Operating System Interface。OSが百花繚乱の時代だった1988年ごろから、互換性を維持するために、Unixを中心としたOS共通のインターフェース(APIやshell、そのほかのコマンド)としてIEEE(アイ・トリプルイーと読みます)によって定められた標準規格です。現在まで改訂が続けられ、維持されています。この範囲内のインターフェースを使っている限り、今後新しいOSが出てきても長年互換性が維持されることを期待できます。

筆者のお勧めは、規格よりも「広く使われている技術を重視する」です。いつ効果を発揮するかわからない、互換や規格の曖昧さの補完、プラットフォームやミドルウェアのバグ回避、性能との両立などのために初心者がむやみなハッスル[注1]をするのは蟷螂(とうろう)の斧(おの)です。情報産業界では、みんなが使っているものが標準＝デファクトスタンダードという考え方が重要な価値基準です。突拍子もない新技術は来年にはなくなってしまっているかもしれませんが、たとえばこれだけ多くの人が使っているbashは、これからも長く使い続けられることでしょう。

bashスクリプトに挑戦

シェルには、ボーンシェル(bourne SHell) sh[注2]の系統と、BSD(Berkeley Software Destribution)を源流とするcshの系統の2つのメジャーな系譜があります。シェルが違えばスクリプトの記法も異なります。現在最も使われているのは、shを大幅に拡張したボーン・アゲイン・シェル(Bourne Again SHell)でしょう。今回はこのbashにフォーカスをあてます。

シェルスクリプトは、ワンライナー(1行スクリプト)に代表される対話的スクリプトと、プログラムをファイルに記述して実行するスタイルがあります。ほかの言語を知っていれば、シェルをよく知らなくても、なんとなくはスクリプトの意味を理解できるでしょう。しかし、外部のコマンドの起動や連携、シェル特有の動作や記法があり、シンプルに書けることも、逆に遠回しでわかりにくい記述になることもあります。

シェル変数

シェルには変数があります。シェルの内部で使う変数をシェル変数と呼びます。シェル変数は、PATHやPWD、動作オプションなどシェル自

注1) ハッスルが必要なときには、巨人の肩に乗りましょう。『すべてのUNIXで20年動くプログラムはどう書くべきか』(松浦智之著 C&R研究所)など、良き先輩が苦労を知恵として記録してくれています。

注2) POSIX準拠。

身の動作を決定する情報や、ユーザが任意で定義できる変数があります。通常のシェル変数は、子プロセスに継承されません。環境変数という特殊なシェル変数は、子プロセスに継承されます[注3]。

シェル変数の定義と参照

シェル変数は次のように定義[注4]します。スペースもしくはタブで区切って、複数の変数を1行に書くこともできます。読み出すときは、変数名の先頭に$を付けます。

```
変数の定義と参照
$ a="abc" b=10 c=↵
$ echo $a↵
abc
$ echo $b↵
10
$ echo $c↵
cは定義されているが、値は空
```

どこまでが変数名か混乱するときは、変数名を{ }でくくります。$*name*という表記は、${*name*}を省略した記法です。例を見てみましょう(**図1**)。

スペース／タブは、コマンド、引数や構文上の区切りとしてあつかわれます。文字列は、クォート('か")でくくれば、文字列中にスペースがあっても一連の文字列として代入できます。

注3) 子プロセスを生成(execve)するときに親プロセスの環境変数が引き継がれます。

注4) bashはスクリプト言語ですので、データの型は実行時都合よく解釈されます。しかし明確に変数の型を指定したいときは、**declare**を使って宣言します。

▼図1　変数と文字列の連結

```
$ serifu="Oishii"↵   serifuという変数に文字列「Oishii」を設定
$ echo $serifu_pyon↵   serifu_pyonという変数を参照という解釈になり、うまくいかない
$ echo ${serifu}_pyon↵   正しくは{}を使う
Oishii_pyon
```

▼図3　算術展開構文

```
$ expr 10 + \( 2  - 10 \) \* 4↵   数字や演算子の前後のスペースや\によるクォートは必須
-22
$ echo $((10 +(2 - 10) * 4))↵   こちらのほうがずっと簡潔・柔軟に書ける
-22
```

環境変数とシェル変数

bashでは、シェル変数を**export**で環境変数にして、子プロセスと情報を共有できます。

```
export e=10
```

すでに定義されている変数に対しても、後からexportできます。

```
e=10
export e
```

環境変数は**printenv**コマンドで確認できます。引数なしで実行すると、設定されているすべての環境変数が表示されます。同様に**set**コマンドを引数なしで実行すると、シェル変数や定義されている関数がすべて表示されます。

図2のように、環境変数は子プロセスに継承することができます。一方、子プロセスで値を変更しても、親プロセスの変数は変更されないことに注目してください。

変数と数

「文字列を連結したい」、「1加えたい」などのさまざまな演算もできます。整数の演算をするには、外部コマンドの**expr**が使えますが、使い勝手の優れた算術展開構文$((...))があります(**図3**)。算術展開構文の中では、変数の参照

▼図2　変数のプロセス間でのスコープを確認する

```
$ a=9↵
$ export e=10↵
$ echo $a $e↵   内容を確認
9 10
$ printenv a↵   結果は何も出力されない
$ printenv e↵   10と出力される
10

$ bash↵   子プロセスを起動してみる
$ echo $a $e↵   aとeの内容を確認。eしか定義されていない
10
$ e=8↵   eの値を変更してみる
$ printenv e↵   確認
8
$ exit↵   親プロセスにもどる

$ printenv e↵   eの内容を確認
10
```

に$を付けなくても(付けても)かまいません。

制御構造

bashには、フロー制御のためのしくみ(if/else、for、while、until、case、select)があります。今回ははじめの4つを紹介します。プロセスの終了ステータスがフロー制御にかかわってきます。直前に実行したコマンドの終了ステータスは、"$?"で参照できます。成功が0、失敗が0以外です。

forループ

forループの基本形は次のとおりです。

```
for 変数 [in list]
do
  処理
  ...
done
```

それぞれ、for ... 、do 処理、doneは、1文(1行)として記述します。inで指定されたリスト(スペース/タブで区切られた要素の列)を1つずつ"変数"に代入して、処理を繰り返します。in *list* を省略すると、コマンドライン引数のリスト(変数$@)です。

次の例は、古くから使われている記法です。seqコマンドで、3、4、5、6、7、8という数字を生成してループする例です。バッククォート(`)は、くくられたコマンドの出力を展開します。bashには、$(...)というコマンド置換機能があり、より明快でネストなどがしやすく強力な記述ができます。次の2つの例は、同じ動作をします。

```
for i in `seq 3 8`
do
    echo $i
done
```

```
for i in $(seq 3 8)
    do echo $i
done
```

ワンライナーで書くには次のように、文をセミコロン(;)で区切ればOKです。

```
$ for ((i=3; i<9; i++)) ; do echo $i ; done↵
```

do～doneは、代わりに{～}と記述できます。さらに((...))構文を使えば、C言語っぽくなりますね。算術展開と同様で、((...))の中では$参照する必要がないのも便利です。

```
for ((i=3; i<9; i++))
{
    echo $i
}
```

forを使ってコマンド引数を取得する

引数は、$1、$2、$3、...、$*n*(位置パラメータ)に入っています。変数$@には、これらすべてが入っています。$0は、実行しているコマンド自身です。$#はコマンド引数の数です。

arg1.sh
```
#!/bin/bash
n=$#
for (( i=0; i<n; i++ )) ; do
    echo $i $1
    shift
done
```

```
$ ./arg1.sh a b c↵
0 a
1 b
2 c
```

STEP UP! コマンド引数の取得でshiftを使わない方法

evalを使って位置パラメータを操作します。evalは、文字列を評価して実行します。

arg2.sh
```
#!/bin/bash
n=$#
for (( i=0; i<n; i++ )) ; do
    eval echo '$i ' \$$((i+1))
done
```

もちろん、n番目の引数であると表示しないなら、for構文の基本形でin以降を省略して、次のようにもっと単純にできます。

```
for i ; do echo $i ; done
```

whileループ

whileの構文は、次のように書きます。条件

を満たしている間、do ... doneを繰り返します。

```
while 条件
do
  処理
  ...
done
```

次の例は、bashの内部コマンドreadでinput.txtの内容を1行ずつ表示するスクリプトです。while文のブロック全体にリダイレクト処理する場合には、doneの直後に書くところにも注目してください。

```
whileを使ってcatコマンドのように動作させる
while read line
do
  echo $line
done < input.txt
```

readコマンド

readは、標準入力や指定したファイルディスクリプタからの入力を読み取り、その値をIFS環境変数(デフォルトはスペース、タブ、改行)で区切られた語に分割してシェル変数に収納します。readの引数に複数の変数を指定できますが、読み取った語より変数の数が少ないと、最後の変数に余りもまとめて代入されます。引数を省略すると、REPLYという変数に結果が収納されます。readを使うと、対話的なスクリプトを記述できます。

```
$ read a b↵
A B C↵
$ echo $a↵
A
$ echo $b↵
B C
```

```
$ read↵
A B C↵
$ echo $REPLY↵
A B C
```

untilループ

untilは、条件を満たしていない間ループを繰り返すという部分がwhileと逆です。

```
until 条件
do
  処理
  ...
done
```

$(...)のコマンド置換、パイプ処理も使ってちょっとこった例を見てみましょう。**図4**の例は、untilを使って特定のコマンドが実行されていない間は、そのプロセスを待ちます。同様の処理をwhileで書けば、特定のプロセスが実行されている間は待つというような記述ができます。このしくみで前日のバックアップスクリプトの処理が終わらないうちに、新たなバックアップが2重に実行されないようにすることもできます。

$()の中に注目してください。はじめに、psコマンドで実行中のプロセス一覧を取得しています。この出力をパイプラインでegrepコマンドに引き渡し、一覧中にemacsが含まれている行を抽出します。さらに次のegrepコマンドに抽出結果を渡し、-vオプションによってgrepが含まれていない行だけ取り出し、2つめのegrepemacsのプロセスを一覧から取り除いています。出力は無用なので、/dev/nullに捨てます。最終的に1行もなければ、$()の処理は失敗し、1行でも見つかれば(おそらく)emacsプロセスが実行されているということで成功です。

if、test、[

プログラミング言語を1つでも知っている人なら馴染みのあるifを見てみましょう。[...]の部分は省略可能です。if <条件>、then 処理、elif 処理、else 処理、fiは1文(1行)として書き分けます。条件は、コマンドの終了ステータスで判定します[注5]。真偽値だけを返すコマンドtrueとfalseもあります。

注5) C言語などでの真値＝0以外と逆であることに注意してください。

▼図4 特定のプロセス(ここではemacs)が存在していない間は待つ

```
until $(ps aux | egrep emacs | egrep -v grep
> /dev/null )
do
  echo 'waiting for emacs.'
  sleep 3
done
echo 'Yep, move on next!'
```

```
if <条件>
then 処理
  ...
[elif <条件>
then 処理
  ...]
[else 処理
  ...]
fi
```

testコマンドは、整数、文字列の比較、ファイルの属性・存在などのチェックをし、結果を終了ステータスとします。多くのオプションがあるので、manで詳しく調べてください。

testの例
```
$ i=10 j=10
$ if test $i -eq $j ; then echo "yes"; fi
yes
```

前例と同じ処理をしますが、次のように[ ]を使って記述すると少しプログラム言語らしくなります。

```
$ if [ $i -eq $j ]; then echo "yes"; fi
```

なんと[はコマンド注6です。/bin/[もしくは、/usr/bin/[というコマンドがあるはずです。

testと[はほぼ同じように動作します。違いは、[コマンドは、最後の引数に"]"が必要なことです注7。[はコマンド、]は引数ですので、スペースでほかの引数と区切られていなければなりません。

条件式には、コマンド以外に、数値を演算・比較しやすい((...))構文が使えます(図5)。

注6) 後述しますがbashでは内部コマンド。

注7) 構文的、審美的に括弧が閉じていないとおかしいですから。

▼図5　((...))構文を使った条件式

```
$ if ((i>j)) ; then echo "iが大きい" ; else
"それほどでもない" ; fi
```

▼リスト1　kazuate.sh

```
 1 #!/bin/bash
 2 echo '[[数あてゲーム]]'
 3 thenumber=$(($RANDOM % 100))
 4 numtry=1
 5
 6 read -p '私が思っている数(0から99まで)を当ててく
ださい。: ' ans
 7 until (( ans == thenumber ))
 8 do
 9     if (( ans > thenumber )) ; then
10         echo "大きすぎます。"
11     else
12         echo "小さすぎます。"
13     fi
14     echo -n "$((++numtry))回目 "
15     read -p 'いくつ? (0-99): ' ans
16 done
17 echo "${numtry}回で当たりました。"
```

STEP UP! ヌルコマンド

nottrue.sh
```
if false ; then
  何もしない
else
  echo "false"
fi
```

を実行すると、次のようなエラーが生じて正しく処理できません。

```
./nottrue.sh: 3: ./nottrue.sh: Syntax error:
"else" unexpected
```

thenに続く処理ブロックに、なにもコマンドを実行しないのは、構文上認められません。これを避けるために、条件文の真偽を反転させるという方法もありますが、開発の途中で変更はできるだけ避けておきたいこともあるでしょう。そのようなときには、thenのブロックに、何もしないコマンドを書くようにします。何もしないコマンドには、true、falseも候補ですが、ヌルコマンド:があります。このコマンドは、何も処理せず常にステータス0を返します。無限ループを記述する場合も、このヌルコマンドは便利に使えます。

nottrue.sh 改
```
if false ; then
  :
else
  echo "false"
fi
```

whileを使った無限ループの例
```
while :
do
    sleep 1
    echo "無限ループ中。停止はControl-C"
done
```

数あてゲーム

ここまで出てきたしくみを使って、簡単なゲームを作ってみましょう(**リスト1**)。

3行目で、組み込みの環境変数RANDOMから乱数を取り出し、100で割った余りを求めています。6、15行目のようにreadに-pオプションを付けると、プロンプトメッセージを表示することができます。14行目のechoは、-nオプションで改行処理を抑止しています。また、++の算術演算子を使っています。シングルクォート(')による文字列のエスケープでは、すべての文字がエスケープされますが、ダブルクォート(")による文字列のくくりは、変数の参照などシェルが解釈すべき文字列はエスケープされません。次が実行例です。

```
$ ./kazuate.sh
[[数あてゲーム]]
私が思っている数(0から99まで)を当ててください。: 50
小さすぎます。
2回目 いくつ? (0-99): 75
小さすぎます。
3回目 いくつ? (0-99): 85
3回で当たりました。
```

終了ステータスと&&、||

ワンライナーなどで便利に使えるのが、**&&**と**||**です。コマンドの終了ステータスを、C言語での論理演算子と同じように、短絡評価します(**表1**、**図6**)。

▼表1　&&、||の評価と終了ステータス

	コマンド1	コマンド2	終了ステータス
コマンド1 || コマンド2	失敗	実行する	コマンド2の結果
コマンド1 || コマンド2	成功	実行しない	成功
コマンド1 && コマンド2	失敗	実行しない	失敗
コマンド1 && コマンド2	成功	実行する	コマンド2の結果

▼図6　&&と||を使ったワンライナーの例(/binに[コマンドがあるかを調べる)

```
コマンド[は、/binにある、/usr/binにある？
$ [ -e /bin/[ ] && echo "binにあります。" || echo "binには、ありません。/usr/binにあるかもしれません。"

OS X / macOSでの結果例
binにあります。

UbuntuなどLinux系での例
binには、ありません。/usr/binにあるかもしれません。
```

STEP UP! コンテキストスイッチ

プロセスはコンテキスト――プログラムの実行中の状態・リソース(CPU内部のレジスタや、メモリ、オープンしているファイルやデバイスなど)を含むすべて――を持つ動作／再開可能なプログラムです。プロセスを起動するには通常、fork→execという処理、つまり親プロセスのコンテキストは保存して、子プロセスに制御を移します。実行しているプロセスを切り替える処理をコンテキストスイッチといいますが、コンピュータにとってコンテキストを保存したり、復帰するコストは高いのです。

test、[、true、false、:、echoなどは、/bin、/usr/binにある実行可能なプログラムですので、forやifで何かを判定するたびにコンテキストスイッチが生じますが、bashはこのコストをなくせるよう内部コマンドとして実装しています。

今回のまとめと次回について

今回は、シェルスクリプトを書くにあたってのbashの基本的なしくみを紹介しました。次回は、テキスト処理に入門しましょう。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)】
bash(1), printenv(1), execve(2), environ(7), expr(1), seq(1), egrep(1), emacs(1), test(1), [(1), true(1), false(1), fork(2), exec(3)
【以下はbashのhelpコマンドを使って確認】
set, export, declare, shift, eval, for, while, until, if, read, test, [, true, false, :

第57回

仮想マシンのライブマイグレーションを支えるuserfaultfd

Text：青田 直大　AOTA Naohiro

先月の予想どおりにLinux 4.8が10月2日にリリースされました。今回のLinux4.8はリリース直前にVM_BUG_ON()でカーネルを殺してしまうバグが入ってしまったことで話題となりました[注1]。その後、Linux 4.9の新機能パッチも出揃って、10月15日にはLinux 4.9-rc1がリリースされています。だいたい2ヵ月ほどでLinux 4.9のリリースとなるので、それが年内最後のリリースとなることでしょう。

今回はLinux 4.3の機能から、より効率的により柔軟にページフォルトをユーザ空間で処理することを可能にするuserfaultfdについて紹介します。

仮想マシンのライブマイグレーション

仮想マシン関連技術の1つに、マイグレーションがあります。これはある仮想マシンを、ある物理マシンから別の物理マシンへと移動することを言います。物理マシンの負荷をバランスしたり、ホストのメンテナンスを行うためにマイグレーションが使われます。

マイグレーションにはいくつかの方法があります。一番安全な方法として、仮想マシンを一度シャットダウンして別の物理マシンで起動しなおすという手段が考えられます。この方法は安全で簡単ではありますが、すべてのサービスを止めて起動しなおすということで、長い間、仮想マシンが使えない期間(ダウンタイム)ができてしまいます。Webサーバなどが動く仮想マシンではこれは好ましくないでしょう。

ダウンタイムを小さくするため、仮想マシンを動かしたままマイグレーションすることが求められます。この場合、シャットダウンしていたときとは違ってメモリの内容を転送する必要があります。とはいえ、マシンが起動されている限りメモリの内容はどんどん更新されていくので単純なコピーではうまくいきません。そこで書き込みのあったページ(dirty page)を追跡する機能を使います。

図1をご覧ください。仮想マシンを動かしたままコピーを行い、コピー中に更新されたページのみを再度コピーするということを繰り返し、更新されたページが十分に少なくなったところで仮想マシンを一時停止して、それ以上の更新を止めて最後の転送を行います。転送先では受け取ったメモリイメージをもとに、仮想マシンを復帰させてマイグレーションが完了します。

注1) http://gihyo.jp/admin/clip/01/linux_dt/201610/06

この方法では最後の転送から転送先での仮想マシンの復帰までがダウンタイムとなり、マシンを停止させてから転送するメモリの量を減らすことで小さいダウンタイムを実現しています。

図1の方法は、仮想マシンの実行を開始する前にメモリを転送するという意味で、プレコピーと呼ばれています。これとは逆に仮想マシンの実行を開始してからメモリのコピーを行うポストコピーという方法があります(**図2**)。

ポストコピーではまずマイグレーション先での仮想マシンの実行に必要な最低限の情報を転送し、仮想マシンの実行を開始してから、メモリの残りの部分を転送していきます。メモリがすべて揃わないまま実行を開始してしまうわけですので、まだ転送が終わっていない領域へのアクセスも起こりえます。したがって、ポストコピーでは、未転送領域へのアクセスをキャッチして、オンデマンドにメモリを転送することも必要になります。

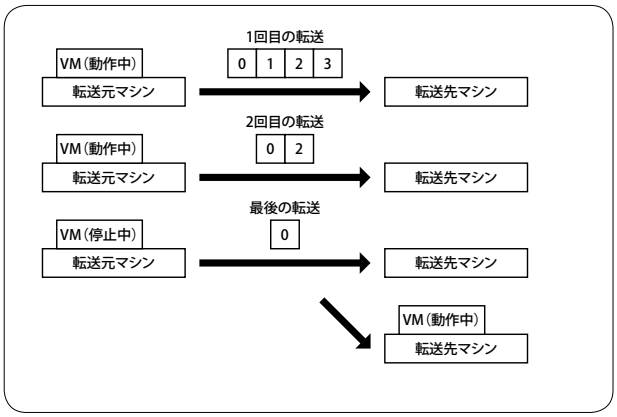
mprotectによるページアクセスのトラップ

転送されていないページへのアクセスをどのように捕まえるかが、ポストコピーの実装上のポイントといえます。では、その部分はどのように実装したらいいのでしょうか？　従来の方法として、mprotectとシグナルハンドラを使う方法があります(**図3**)。

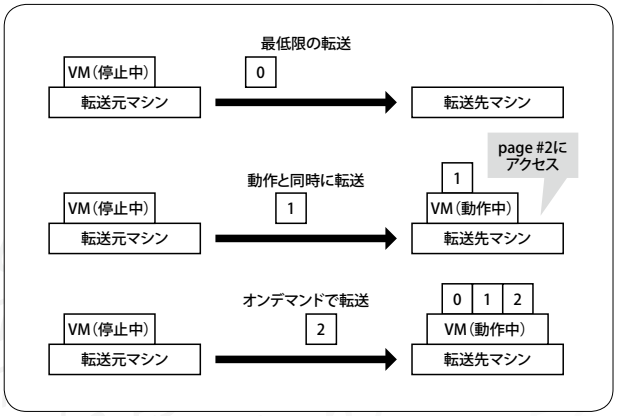
リスト1がその方法を使ったプログラムです。`main()`から見ていきましょう。まず、対象となるメモリ領域を8GB分確保します。`mprotect()`の対象となるアドレスはページ境界にアライメントされてなければいけないので、`posix_memalign()`を使います。次にこの領域全体を`mprotect()`を使ってアクセス禁止にします。これで、この領域に読み書きを行うとカーネルがSIGSEGVシグナルを発行するようになります。

あとは`sigaction()`を使って、SIGSEGVのハンドラとなる関数`hanlder()`を導入します。`handler()`では、`si->si_addr`からアクセスされたアドレスを取得し、そのページをふたたび`mproctect()`を使って読み書き可能にします。

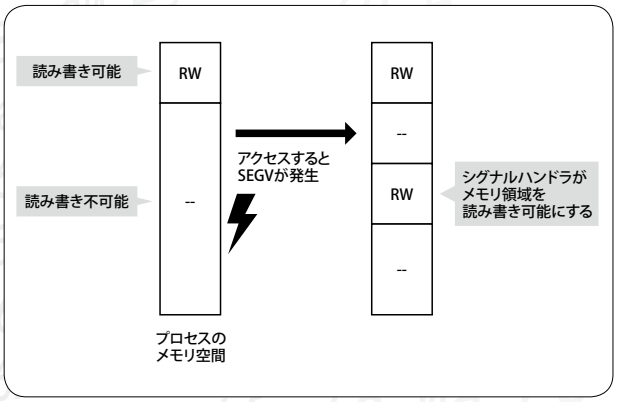
▼図1　仮想マシンを動かしたままマイグレーションする（プレコピー）



▼図2　仮想マシンを動かしたままマイグレーションする（ポストコピー）



▼図3　mprotectとシグナルハンドラ



▼リスト1　mprotectとシグナルハンドラを使うページアクセスのトラップ

```
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <sys/mman.h>
#include <time.h>
#include <unistd.h>

#define PAGE_SIZE 4096

const size_t SIZE = (size_t)8 << 30;
char *buf;
unsigned long cnt = 0;

static void handler(int sig, siginfo_t *si,
 void *unused)
{
  char *addr = si->si_addr;
  if (buf <= addr && addr <= buf+SIZE) {
    if (mprotect(addr, PAGE_SIZE, PROT_READ|
PROT_WRITE)) {
      perror("mprotect");
      char cmd[1024];
      sprintf(cmd, "cat /proc/%d/maps", getpid());
      system(cmd);
      exit(1);
    }
    *(unsigned long*)addr = cnt++;
  } else {
    exit(1);
  }
}

int main(int argc, char *argv[])
{
  posix_memalign((void**)&buf, PAGE_SIZE, SIZE);
  if (!buf) {
    perror("memalign");
    return 1;
  }

  if (mprotect(buf, SIZE, PROT_NONE)) {
    perror("mprotect");
    return 1;
  }
  struct sigaction sa;
  sa.sa_flags = SA_SIGINFO;
  sigemptyset(&sa.sa_mask);
  sa.sa_sigaction = handler;
  if (sigaction(SIGSEGV, &sa, NULL) == -1) {
    perror("sigaction");
    return 1;
  }

  char cmd[1024];
  sprintf(cmd, "cat /proc/%d/maps | wc -l",
getpid());

  int i;
  volatile unsigned long x;
  for (i=0; i<100000; ++i) {
    if (i%1000 == 0) {
      printf("%d #vma: ", i); fflush(stdout);
      system(cmd);
    }
    x = *(unsigned long*)(buf+i*2*PAGE_SIZE);
    if (i%1000 == 0) {
      printf("read %lu\n", x);
    }
  }

  free(buf);

  return 0;
}
```

さらに、書き込みが行われたことを確認するためにハンドラが呼ばれるたびにインクリメントされる数字を書いておきます。

main()に戻って読み込み側を見ていきましょう。ここでは対象のメモリ領域を1ページずつ飛ばしながら読み込んでいきます。ここでは1,000ページに一度読み込んだ値を表示しています。ハンドラ側がうまく動いていれば、1,000回目のreadで"1000"などと表示されるはずです。

では、このプログラムを動かしてみると……図4のように32,000ページ読み込んだところでメモリ不足でシグナルハンドラ内のmprotect()が失敗してしまいました！　これがmprotect()による実装の問題点です。プロセスの、どの領域がどのような保護状態にあるのかはカーネル内でstruct vm_area_structによって管理されています。そして、あるプロセスのstruct vm_area_structがどのような状態にあるのかは/proc/<PID>/mapsを見るとわかります。

図5の1行が1つのstruct vm_area_structに対応しています。プログラム中ではcat /proc/<PID>/maps | wc -lの結果を"#vma:"のあとに表示しています。1つページにアクセスして1つのページへのアクセス許可を変更するたびに、図3のようにメモリ領域が分割されるのでstruct vm_area_structが2つずつ増えていくことになります。

結果として、32,000ページ読んだところで、struct vm_area_structの数が64,000以上の数になっています。実際、リストのプロセス終

▼図4　mprotectを使ったページアクセスのトラップの出力

```
$ ./mprotect
0 #vma: 22
read 0
1000 #vma: 2020
read 1000
2000 #vma: 4020
read 2000
...
30000 #vma: 60020
read 30000
31000 #vma: 62020
read 31000
32000 #vma: 64020
read 32000
mprotect: Cannot allocate memory
./mprotect > /dev/null  0.10s user 3.44s system 101% cpu 3.485 total
```

▼図5　プロセス終了時の/proc/PID/mapsの出力

```
00400000-00401000 r-xp 00000000 08:03 16331100                           /home/naota/mprotect
00600000-00601000 r--p 00000000 08:03 16331100                           /home/naota/mprotect
00601000-00602000 rw-p 00001000 08:03 16331100                           /home/naota/mprotect
01aa4000-01ac6000 rw-p 00000000 00:00 0                                  [heap]
7fae5006f000-7fae50071000 rw-p 00000000 00:00 0
7fae50071000-7fae50072000 ---p 00000000 00:00 0
7fae50072000-7fae50073000 rw-p 00000000 00:00 0
7fae50073000-7fae50074000 ---p 00000000 00:00 0
(..中略..)
7fae60054000-7fae60055000 rw-p 00000000 00:00 0
7fae60055000-7fae60056000 ---p 00000000 00:00 0
7fae60056000-7fb050070000 ---p 00000000 00:00 0
7fb050070000-7fb050071000 rw-p 00000000 00:00 0
(..後略..)
```

了時のmapsファイルを見るとメモリ領域が読み書き可能な“rw-p”領域と、読み書きができない“---p”領域に1ページずつ分割されていることがわかります。これがこのプログラムがメモリ不足で終了してしまう原因です。

userfaultfdによるページアクセスのトラップ

Linux 4.3の新機能userfaultfdは、mprotectの方法における**struct vm_area_struct**が増大する問題を発生させずに、ページアクセスをトラップするために使うことができる機能です。

さっそく、先ほどと同じプログラムをuserfaultfdを使って実装してみましょう(**リスト2**)。

まずuserfaultfdのシステムコールを実行し、userfaultfdのファイルデスクリプタを取得します。このデスクリプタにioctl、read、pollで操作を行っていきます。

次に**ioctl(UFFDIO_API)**を使って、カーネル側のAPIのバージョンが適合しているかを確認します。そして**ioctl(UFFDIO_REGISTER)**を使って、対象とするメモリ領域を登録します。今後、この領域へのアクセスはuserfaultfdの読み込み側でハンドルされるまでブロックされます。

userfaultfdの読み込み側は**if ((pid=fork())==0){}**の中にあります。userfaultfdもファイルデスクリプタですので、このようにファイルデスクリプタを引き継ぐ子プロセスから処理を行うことができます。上記のように設定したuserfaultfdはreadすることで**struct uffd_msg msg**の情報を読み込むことができます。この構造体の**event**には、どのようなイベントが起きたのか(現状、**UFFD_EVENT_PAGEFAULT=0x12**のみ)、フォルトが起きたアドレス、そし

▼リスト2　userfaultfdを使うページアクセスのトラップ

```
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <sys/mman.h>
#include <time.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/syscall.h>
#include <linux/userfaultfd.h>
#include <string.h>

#define PAGE_SIZE 4096

const size_t SIZE = (size_t)8 << 30;
char *buf;
unsigned long cnt = 0;

int main(int argc, char *argv[])
{
  posix_memalign((void**)&buf, PAGE_SIZE, SIZE);

  char *zeropage;
  posix_memalign((void**)&zeropage, PAGE_SIZE, PAGE_SIZE);
  memset(zeropage, 0, PAGE_SIZE);

  int fd;
  fd = syscall(__NR_userfaultfd, 0);
  struct uffdio_api api;
  memset(&api, 0, sizeof(api));
  api.api = UFFD_API;
  if (ioctl(fd, UFFDIO_API, &api) < 0) {
    perror("ioctl(UFFDIO_API)");
    return 1;
  }

  struct uffdio_register reg;
  memset(&reg, 0, sizeof(reg));
  reg.range.start = (__u64)buf;
  reg.range.len = SIZE;
  reg.mode = UFFDIO_REGISTER_MODE_MISSING;
  if (ioctl(fd, UFFDIO_REGISTER, &reg) < 0) {
    perror("ioctl(UFFDIO_REGISTER)");
    return 1;
  }

  pid_t pid;
  if ((pid=fork())==0) {
    struct uffd_msg msg;
    while(read(fd, &msg, sizeof(msg)) == sizeof(msg)) {
      fprintf(stderr, "Event: 0x%x Address: 0x%llx Flags: 0x%llx\n",
      msg.event, msg.arg.pagefault.address, msg.arg.pagefault.flags);

      *(unsigned long*)zeropage = cnt++;

      struct uffdio_copy copy;
      copy.dst = (unsigned long) msg.arg.pagefault.address;
      copy.src = (unsigned long) zeropage;
      copy.len = PAGE_SIZE;
      copy.mode = 0;
      copy.copy = 0;
      if (ioctl(fd, UFFDIO_COPY, &copy) < 0) {
```

（次ページに続く）

（前ページから続き）

```
        return 1;
      }
    }
    return 0;
  }

 (..中略..) mprotectと同様の/proc/<PID>/mapsの行数表示コード

  kill(pid, SIGKILL);

  free(buf);
  free(zeropage);

  return 0;
}
```

▼図6　userfaultfdを使ったページアクセスのトラップの出力

```
0 #vma: 22
Event: 0x12 Address: 0x7fa1e646c000 Flags: 0x0
read 0
1000 #vma: 22
Event: 0x12 Address: 0x7fa1e646e000 Flags: 0x0
read 1000
(..中略..)
99000 #vma: 22
Event: 0x12 Address: 0x7fa2171aa000 Flags: 0x0
read 99000
./userfault 2> /dev/null  0.19s user 1.15s system 66% cpu 2.022 total
```

てイベントのフラグが入っています。読み込んだイベントに対する処理は**ioctl(UFFDIO_COPY)**を使って行います。**struct uffdio_copy**にコピー元とコピー先(フォルトしたアドレス)、サイズなどを指定してioctlを実行すると、フォルトした領域にコピー元のメモリの内容がコピーされます。コピーが完了したところで、フォルトで止まっていたプロセスの動きが再開されます。

では、userfaultfdを使ったプログラムを動かしてみましょう。今度はメモリ不足のエラーで終了せず、最後までメモリアクセスが行われています(**図6**)。

実際に**struct vm_area_struct**の数も22のまま変わっていないことがわかります。また、mprotectのほうでは、33,000ページ未満を読むまでに3.485秒かかっているのに対して、userfaultfdでは100,000ページ読んでも2.022秒とパフォーマンスも高いことがわかります。これは**struct vm_area_struct**のallocateをしないこと、mmapのセマフォをとらないことなどに起因しています。

このようにuserfaultfdはこれまでmprotectとシグナルハンドラで行っていた処理をより高いパフォーマンスで実現しています。また、ファイルデスクリプタに対して処理することから、UNIXドメインソケットによってほかのプロセスに転送できるなど、シグナルハンドラにおけるプロセスに閉じている制約も解決しています。

◆　◆　◆

今回はよく使われる例として仮想マシンのポストコピーを取り上げましたが、いろいろと応用できそうなおもしろい機能です。SD

December 2016

NO.62

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

みんなプログラミングでつながれ!　LLoT開催!

今回は、今年で14回目を迎えたLL（Lightweight Language）イベントについて報告します。今回はイベント名称を「Lightweight Language of Things」（通称：LLoT）とし、昼の部を日本電子専門学校、夜の部をwatford新宿店にて行いました。参加者は、昼の部が118人、夜の部が53人でした。

■Lightweight Language of Things
【日時】2016年8月27日（土）10:00～20:00
【会場】日本電子専門学校、watford新宿店

昼の部

■Language Update

- 出演：高橋 徹（Java）、hnw（PHP）、石垣 憲一（Perl）、清原 弘貴（Python）、佐藤 鉄平（JavaScript）、高橋 征義（Ruby）
- 司会：りゅうちてつや

各言語の近況を伝えるセッションで、1人ずつ発表する形式で実施しました。常連の言語が並ぶ中、初出場だったのがJavaです。過去のLLイベントではあまり取り上げてこなかったのですが、Java 8で関数型言語の考え方が採り入れられるなどの大きな変化があり、LLプログラマに状況を伝えたほうが良いだろうということで参加していただきました。

■キーボードにこだわろう

- 出演：奥野 幹也、松本 秀樹（㈱PFU）、持田 智彦（東プレ㈱）、小林 充（東プレ㈱）
- 司会：前田 薫

プログラミングをするうえで欠かせない道具であるキーボードについて語り合うセッションです（**写真1**）。奥野さんはErgoDox、松本さんはHappy Hacking Keyboard、東プレのお2人はRealforce、司会の前田さんもKinesisを担当し、単なる製品紹介にとどまらず、キーボードの構造や動作のしくみなど濃い話題が展開されました。

■Dynamic Typing再考

- 出演：まつもとゆきひろ、梅澤 真史、佐藤 鉄平
- 司会：高橋 征義

LLイベントに登場する言語は動的型付け（Dynamic Typing）を行うものが大半ですが、最近は静的型付け言語が再び注目されています。そこで本セッションでは、動的型付け言語であるRuby、JavaScript、Smalltalkの有識者を招き、型について議論しました。JavaScriptおよび類似の言語への型宣言導入の動向、まつもとさんがRubyに型宣言を採り

写真1　「キーボードにこだわろう」セッションの様子

入れない理由、梅澤さんによるSmalltalkの爆笑デモなど、内容の濃いセッションでした。

■Kotlin vs Swift
～新言語はモバイル開発をどう変えるか？～

- 出演：長澤 太郎（エムスリー㈱）、岸川 克己（Realm）
- 司会：えんどう やすゆき

新進のモバイル開発言語として注目されているKotlinとSwiftに焦点を当てたセッションです。長澤さんがKotlin、岸川さんがSwiftの担当で、それぞれの言語仕様や最近の状況、開発者への利点、開発時の注意点などについて話していただきました。

■物販＆見本誌・機材展示

今年もIT系出版社による書籍の即売と見本誌展示を行いました。参加した出版社は、アスキードワンゴ、㈱オーム社、㈱技術評論社、SBクリエイティブ㈱、㈱オライリー・ジャパン、㈱達人出版会、㈱マイナビ出版、USP出版でした。

また、今回は書籍だけでなく、「キーボードにこだわろう」セッションに関連してPFUと東プレによるキーボードの展示も行われました。とくにPFUからは漆塗りのHappy Hacking Keyboard（**写真2**）が出展され、注目を集めていました。

夜の部（LLoT Night）

■フロントエンドだめ自慢

- 出演：大竹 智也、河村 奨

写真2　漆塗りのHappy Hacking Keyboard

- 司会：小山 哲志

JavaScriptフレームワークについて、良いところだけでなくダメな部分も一緒に語り合おうというセッションです。大竹さんがReact.js、河村さんはRiot.jsを担当しました。React.jsはCSSをどうするか問題、Riot.jsはコンパイラが正規表現でしかないなどの問題が提起されました。

■帰ってきたデモ自慢

- 演題と出演者
  - 「5分で出来るIoT」呉屋 寛裕（ニフティ㈱）
  - 「プロジェクト°D ツワモノどもが夢の跡」海老原 寛之（㈱サイタスマネジメント）
  - 「Googleカレンダーで図書館の貸出予約状況を管理するLiblendrsv（Ruby）」堀田ほつた（東海大学）
- 司会：法林 浩之

2005年に行われたLLDN（LL Day and Night）の夜の部で実施した「デモ自慢」のリメーク版で、今回はライトニングトークの要素も採り入れて制限時間5分でデモをしてもらいました。5分の発表の中にデモを入れるのが難しいのか応募者が3人しかいませんでしたが、それぞれ果敢にデモに挑戦し、参加者の拍手喝采を浴びていました。

■終わりに

ここ数年のLLイベントは、LLと周辺技術に焦点を当てたセッションが多かったのですが、今年は久々に言語仕様やプログラミングを正面から扱ったものが多く、内容は充実していました。ただ、昨今のイベント供給過剰の状況下で、このイベントを選んで来てもらうことはなかなか難しく、思うように参加者を集められなかったのが残念です。もっと多くの人に見てもらうための工夫が必要と感じました。

発表資料、写真、映像などはWebサイト[注1]に置いてあります。こちらもぜひご参照ください。SD

注1）URL http://ll.jus.or.jp/2016/

Hack For Japan
エンジニアだからこそできる復興への一歩

● Hack For Japanスタッフ
鎌田 篤慎　KAMATA Shigenori
Twitter @4niruddha
及川 卓也　OIKAWA Takuya
Twitter @takoratta

第60回 減災ソフトウェア開発に関わる一日会議2016

今回は2016年10月1日に開催された「減災ソフトウェア開発に関わる一日会議2016」にて、Hack For Japanメンバーでもあり、防災、減災に取り組む団体IT DARTにて活動する及川による熊本地震で実施した活動内容の紹介と、本イベントの内容を同じくHack For Japanメンバーの鎌田がレポートします。

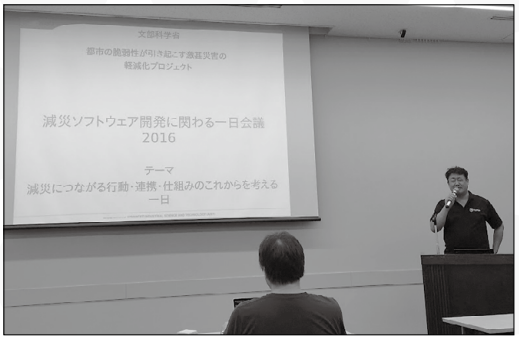
減災ソフトウェア開発に関わる一日会議とは

「減災ソフトウェア開発に関わる一日会議」は2013年から毎年開催されているもので、東日本大震災以降、毎年多くの災害が発生する災害大国日本における災害時の対応に必要な情報技術や、それによって連携できる支援活動について、参加者間で情報交換やディスカッションを行う場となっています。これまでもそうした活動に携わってきた我々、Hack For JapanやIT DARTといった団体の関係者も数多く集まり、直近に発生した災害などで得た知見などを共有してきました。

今回の会議では、近く発生した熊本・大分での地震や、東北地方を襲った豪雨による被害状況、情報ボランティアの活動実績、そこで生じた課題を共有し、何ができて、何ができないのかを知ることで、平時に準備しておくべきノウハウを整理することを目的として開催されています。また、本イベントでは防災や減災、被災地域で活動されてきた多くの人たちの課題や学びを共有し、参加者がそれぞれディスカッションしたい議題を提示、話し合うアンカンファレンスを通して、今後の防災や減災に向けてどういったアクションを取るべきか、次の行動につなげるための議論が行われました（**写真1**）。

それでは、本イベントで発表されていた内容を中心に紹介していきたいと思います。

◆ **写真1　「減災ソフトウェア開発に関わる一日会議」開会の様子**



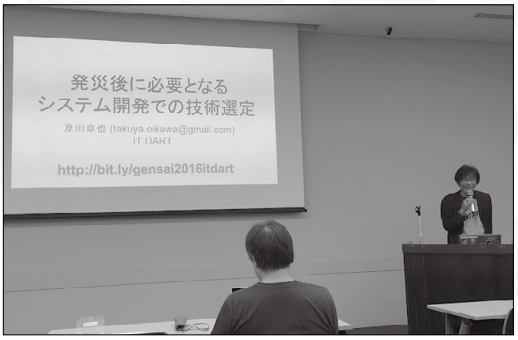
発災後に必要となるシステム開発での技術選定

筆者（及川）からは発災後に必要となるシステム開発での技術選定について説明しました（**写真2**）。発災後でのITの役割は、Webサイトやアプリケーションなどの開発に留まりません。むしろ、それ以外での対応がメインであることも多いです。それは、我々が東日本大震災以降でのHack For Japan活動でも学んだことです。ですが、ここでの話はそれを踏まえたうえで、「開発」が必要になるニーズがあった場合の技術選定について考察したものです。

ターゲットユーザとシステム内容

まず、発災後に必要となるシステムのターゲット

◆ **写真2　筆者（及川）による発表の様子**



ユーザと内容は次のように分類できます。

ターゲットユーザ

- 他支援団体および個人
- 自治体関係者
- 一般市民（おもに、被災地以外の市民）

内容

- 情報発信
- 支援作業へのIT支援
- 他IT支援で使われる汎用技術

IT DARTの熊本地震での支援プロジェクトをこの分類に従って整理すると、**表1**のようになります[注1]。

共通のシステム要件

このようにユーザや内容は異なりますが、これらのシステムには次の3つの共通の要件があります。

1. 利用者が使うプラットフォームで利用できる必要がある
2. 迅速に提供する必要がある
3. 提供後にも仕様変更の可能性が高い

まず、プラットフォームについては、実際のユーザを考えるとWindowsかスマートフォンやタブレットとなります。開発ではMacを使うことが多いかもしれませんが、一般ユーザはWindowsを利用していることがほとんどです。これらのプラットフォームでのシステムは大別すると、ネイティブアプリケーションかWebとなります。

注1　各プロジェクトは連載第56回（本誌2016年8月号）で紹介しています。

ネイティブアプリケーションは普段使いしているアプリケーションと同じ操作性の提供という点では優位です。もちろん、プラットフォームでのUXデザインガイドラインなどに準拠するのは前提となります。一方、Webは操作性はネイティブアプリケーションには劣る場合があります。とくに、ファイル操作などローカルシステムとの連携が必要となる場合はネイティブアプリケーションに分があります。

一方、ネイティブアプリケーションは迅速性に関しては課題があります。スマートフォンやタブレットはApp StoreやPlay Storeなどのプラットフォームでのストアを経由してインストールや更新する必要があり、それにはAppleやGoogleの審査が入ります。開発者がシステムの提供タイミングをコントロールできないのです。ストアを経由しないアプリケーションを使うように設定を変更することもできますが、セキュリティを考えるとお勧めできません。Windowsはストアを経由せずにアプリケーションを提供することができますが、逆にインストールと更新を手作業で行わなければならず、そのための手間はかかります。WebはURLをブラウザで開いてもらうだけですので、システムの提供も更新も最短に行えます。

このようにネイティブアプリケーションとWebはそれぞれ一長一短ありますが、Windowsにはネイティブアプリケーションを、スマートフォンやタブレットにはWebを使うのが一般的には勧められます。

Webサイト制作

情報支援として常にニーズがあるのがWebサイト制作になります。システムダウンしてしまった自

◆表1　IT DARTの熊本地震での支援プロジェクト分類表

プロジェクト	ターゲットユーザ	内容
自治体HPレスキュープロジェクト※1	1）自治体関係者	情報発信
	2）一般市民	
Excel Geo※2	他支援団体および個人	他IT支援で使われる汎用技術
マークシートを用いた災害ボランティア登録システム（検討のみ）	他支援団体および個人	支援作業へのIT支援
物資管理帳システム	他支援団体および個人	支援作業へのIT支援
り災証明発行システム（未使用）	自治体関係者	支援作業へのIT支援

※1　http://klgmonitor.itdart.org/　　※2　http://excelgeo.itdart.org/

治体サイトから重要な情報だけを載せ替えた暫定サイトの制作や支援団体の情報発信用のサイトなど、今までの災害でも提供してきました。

このようなWebサイト制作に対する手段としては、Jimdo[注2]、Wix[注3]、Strikingly[注4]、Weebly[注5]などに代表される簡易Webサイト制作サービスを使う方法やTumblrを使う方法、WordPressを使う方法、スクラッチからすべて作る方法があります。

それぞれの制作の難易度、ホスティングの手配の要否、ユーザによる更新の可否などをまとめると**表2**のようになります。

Web技術

単純な情報発信以上の機能を持つWebサービスを開発する場合は、一般的にはIaaSよりもPaaSのほうが迅速にサービスを立ち上げられます。IaaSの場合は事前にコンテナを用意しておくなど、発災後にスクラッチから作業をしなくて良いように準備をしておくことが望まれます。

また、認証や通知などの機能が必要な場合は、Google FirebaseなどのBaaSを活用することも勧められます。BaaSだけでなく、ありもののサービスやライブラリなどを最大限活用することで、本当に必要な機能の開発に集中することができます。

さらに、使う技術の移植性も考慮する必要があります。実際の例なのですが、簡単な処理だからとシェルプログラミングで組みました。プログラム完成後、レンタルサーバを引っ越す必要があったのですが、このシェルプログラムが引っ越し先で動作しなかったのです。調べてみると、LinuxとFreeBSDの違いが原因でした。これなど、最初からPerlやPython、Rubyなどのスクリプト言語で組んでいれば起こらなかった問題です。

及川からは、以上のような解説とともに、アクセス過多への対応とセキュリティの重要性を強調して、発災後のシステム開発での技術選定のセッションを終了しました。

情報共有の重要性と災害時のシステム

京都大学防災研究所教授でIT DARTのメンバーでもある畑山満則氏からは、被災地での情報支援システムの導入の困難さからの学びを紹介していただきました。

被災した現地では、情報支援システムを含むIT環境は、徹底的にサポートしなければ定着しないという問題を指摘されていました。これは情報共有を行うシステムを導入したとしても、現地の人たちがパソコンを含む、そうしたIT環境の利活用を身につける学習コストが高すぎて、定着しないというものです。IT環境に不慣れな利用者側は、システムの作りに合わせて融通を利かせてデータ入力をしたりすることがなかなかできません。たとえば、数字ひとつ取っても半角、全角、英数字、漢数字、手書き文字など、これらを利用者に統一的に入力してもらうことが困難であり、そうした状況をシステム面で吸収していこうとすると、非常に工数のかかるシステム開発に発展していきます。

この課題に対して、畑山氏の取ったアプローチは、被災地の人たちに必要な支援物資を紙の伝票に記入して、その写真を読み取って仕分けするシステムです。利用者のバラバラな入力をOCRで読み取らず、畑山氏の身近な大学生に協力を依頼し、人力で写真の伝票を読み取り、データベースに登録して

注2　http://jp.jimdo.com/
注3　http://ja.wix.com/
注4　https://www.strikingly.com/?locale=ja
注5　https://www.weebly.com/jp

◆**表2　Webサイト制作にかかるコスト**

サービス	難易度	ホスティング	ユーザによる更新
簡易Webサイト制作サービス	低	込み	可（優しい）
Tumblr	中	込み	可（やや難しい）
WordPress	中～高	別途用意	可（難しい）
すべて自作	高	別途用意（GitHub Pagesを使えば無料）	不可（技術がわかるユーザの場合を除く）

いく作業に切り替えました。このことで必要な支援物資を被災地の人の手間をかけずに正しく仕分けて郵送できるようになりました。

そうした経験から導かれた畑山氏の学びは、災害発生時にその時点の状況に合わせて最適な技術を選択し、要件を満たしていくほうが、被災状況にあったシステムを素早く提供できるのではないか？というものです。この視点から「災害時に支援するならソフトウェアは使い捨てにすべき」という提言がなされました。これは中越地震のときには世の中になかったクラウド環境などの誕生が、旧来のシステムを流用することのミスマッチさを実感されていたからこその提言のように感じられました。

熊本地震で起きた課題

熊本市消防局 情報司令課の池田光隆氏からは、先日に発生した観測史上最大の地震となった熊本地震の被災直後の状況、その後の救助活動で発生した課題などが共有されました（**写真3**）。

震災時、消防センターそばにいた池田氏は前例のない規模の揺れに慌てて消防センターに戻るも、その後に続いた熊本地震の本震から、はじめて前震という言葉を知ったとのことでした。そして、落ち着いた現在でも余震は続いているそうです。そうした中で熊本地震の影響で発生した状況は、今後想定されている首都圏直下や南海トラフを視野に入れた防災、減災の観点でも、非常に示唆に富んだ報告となりました。

まず、報告されている避難所に18万人が生活しているという情報は、車中泊の被災者の数が含まれておらず、家屋の倒壊などの二次災害を恐れ、車中泊を選択した被災者の数を含めると、およそ、その倍の数の避難民が潜在的に予想されているとのことで、この車中泊が駐車スペースを埋め尽くしてしまう影響で、各地から駆けつける救助隊を受け入れるスペースが減り、速やかな支援が困難になったそうです。これは駐車スペースがある地方都市であれば、まだ問題として発展する可能性は低いですが、首都圏などの駐車スペースが限られているような都市では、避難所も限られ、非常に危機的な状況を生む可能性を感じさせました。

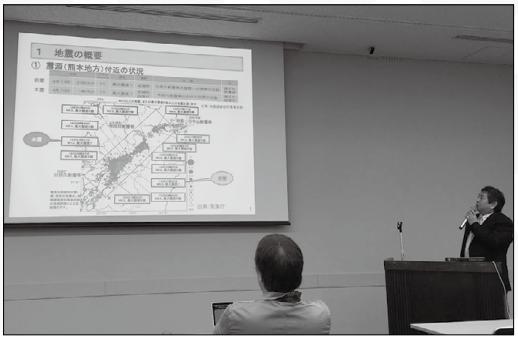
また、SNSの普及から、Twitterなどの投稿がない、あるいは知人の投稿内容を見て、消防に通報する人がこれまでにない規模で発生し、同じ人に対する119番通報が発生するなど、通報の輻輳（ふくそう）が非常に多い割に、位置情報が含まれていない内容から、どこに救助に行けば良いのかわからないケースも多く発生したとのことでした。しかし、後から振り返ると危機的な状況ほど、通報の輻輳が発生していたという学びもあったそうです。

ほかにも例のない災害を他県の消防にも情報共有するために救命対応に追われる中でも、報道への対応を真摯（しんし）に行うも、意図しない誤った解釈の取り上げ方をされ、それを見た視聴者の新たな対応が発生したり、現場の士気を下げるなど、メディアとの関わり方が現場に与える影響も報告されました。

将来の減災、防災に向けて

こうしたいくつかの報告を受けて、将来に起きるであろう新たな災害に向けて、参加者はどういった取り組みで減災、防災の効果を上げていくかという軸から、アンカンファレンスを行い、未来に向けての取り組みを参加者それぞれが発表し、「減災ソフトウェア開発に関わる一日会議2016」は終了しました。読者の皆さまにも減災、防災のヒントが伝われば幸いです。SD

◆写真3　池田氏による熊本地震の被害状況の共有の様子



うまくいく チーム開発のツール戦略

第5回 継続的インテグレーション（CI）ツールで安定した本番リリースをしてみよう

Author リックソフト㈱ 持田 秀敏（もちだ ひでとし）

はじめに

前回は品質を保ちながらコードを改善していくコツや技術的負債の賢い返済方法について説明しました。今回は、修正したコードが本番環境で動かない状況を避けるにはどうしたらよいかをお話しします。

修正したコードが本番環境で動かない状況を避けるには、「継続的インテグレーション（Continuous Integration、以下CI）」ツールによる自動化が有効な手段の1つです。今や開発現場になくてはならない存在となったCIですが、みなさんの環境では導入されているでしょうか。そもそも何のためにCIが必要なのか、まずはおさらいしたいと思います。

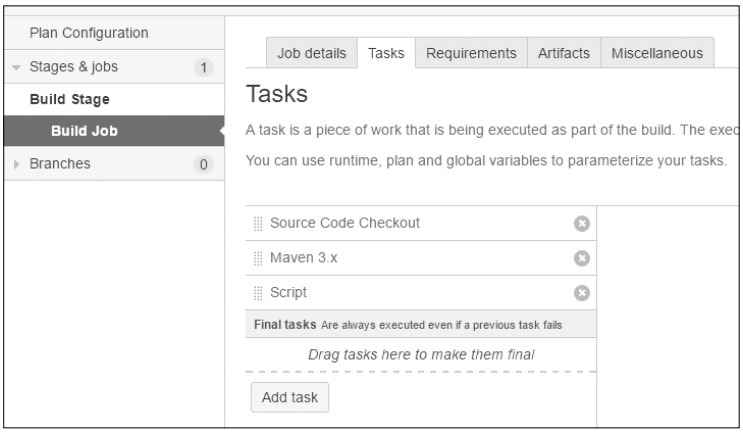
CIとは、ソフトウェア開発過程において、ビルドやテストを頻繁に繰り返し行うことにより問題を早期に発見する開発手法です。ソフトウェアの不具合は早い段階で発見するほうが、あとの工程で発見されるより対策にかかるコストが抑えられるため、頻繁にビルドを行うのが好ましく、ビルド結果が正しいことを検証するためにテストを行います。

しかし、ビルドやテストを頻繁に毎回手動で行うのは手間がかかります。そこで、特定のタイミングで自動的にビルドやテストを実行し、完了後にその結果を通知する「CIツール」が広く利用されています。CIツールによる自動化のメリットは多く、単純作業から開発者を解放し、設計や実装などの知的な作業に集中できるようになります。

従来の開発では、「私の環境ではビルドできるが他人の環境だとビルドできない」「ビルドはアノ人にお願いしないとできない」「ビルド手順を間違えてうまくソフトウェアが動かない」といった場面がたびたびありました。作業を自動化して暗黙知の作業や属人性を排除することで、作業のミスがなくなり、誰でも実行可能なソフトウェアを作成できるようになります。

本稿では、数あるCIツールの中から、Atlassian社が提供するCIツール「Bamboo（バンブー）」をベースにCIの方法を説明します。

▼図1 ビルド用のプロジェクト



Bambooについて

Bambooでは、ビルド用のプロジェクトとデプロイ用のプロジェクトを作成できるようになっています。

ビルド用のプロジェクトでは、プランという単位でビルドのシナリオを定義します。ビルドのシナリオには、「ステージ」という概念と「ジョブ」という概念があります。ジョブでは、一連のビルドタスクを構成していきます。たとえば、ソースコードをチェックアウトして、ビルドツールでビルドを行い、テストを実施する、といったタスクを組みます。

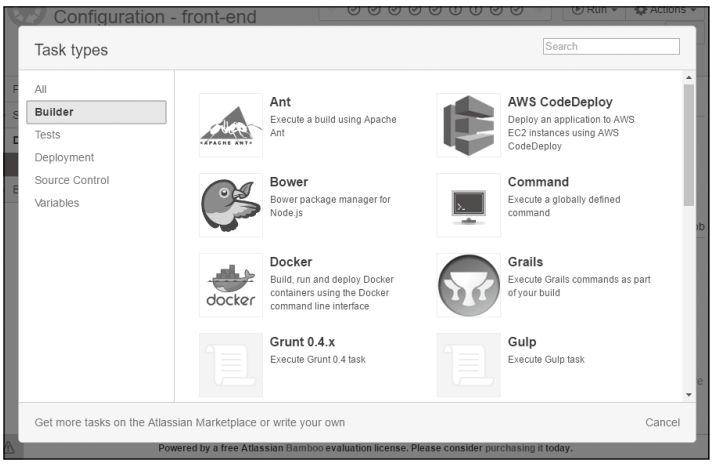
図1は、「Build Stage」というステージに「Build Job」というジョブがあり、そのジョブが「Source Code Checkout」「Maven 3.X」「Script」という3つのタスクを持っている構造を示しています。これらのタスクは、あらかじめ用意されているものを利用できます。そのためスクリプトをゴリゴリ書くことは必要最低限で済みます(**図2**)。

デプロイ用のプロジェクトでは、どのビルドの成果物をどの環境にデプロイするか、タスク、実行タイミングをどうするか、などをデプロイ対象の環境ごとに定義します(**図3**)。デプロイもビルドと同様に、あらかじめ用意されているタスクから選択してデプロイを構成します。

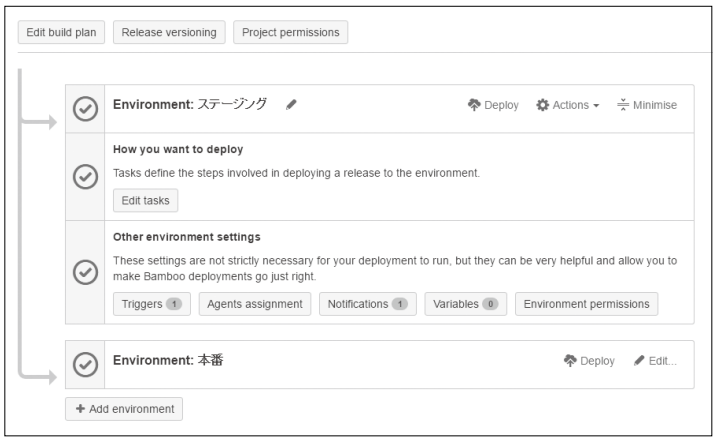
実行タイミングは、ビルドが成功したら即時にデプロイするとか、1日1回決まった時間にデプロイする、などの設定ができます。もちろん手動でデプロイすることもできます。手動でのデプロイは数クリックで簡単に実行できます。

ほかにもBambooは次のような機能を持っています。

▼図2　Bambooにあらかじめ用意されているタスク



▼図3　デプロイ用のプロジェクト



ブランチの自動検出

GitやMercurial、Subversionのリポジトリを使用している場合に限りますが、リポジトリでブランチが作成された際に、Bambooが自動でブランチを検出し、プランを自動で作成します。そのため、開発者はブランチを作成し、作成したブランチにコードをコミットするだけで、Bambooが自動的にブランチを検出してビルドやテストを行ってくれます。

通知設定

ビルドやデプロイの実行時に特定の条件で通知ができるようになっています。ビルド完了時に必ず通知する設定は言うまでもなく、最新のビルドが成功から失敗に変化した場合といった条件で通知を設定することもできます。連続でビルドに失敗した場合という条件も指定できるので、たとえば5回ビルドに失敗した場合はプロジェクトの管理者に通知してエスカレーションするといった使い方も考えられます。

通知はEメール、Atlassianが提供しているチャットアプリであるHipChat、その他にもXMPPプロトコルを使用してIMアプリへ送ることも可能なので、ビルドやデプロイの状況をリアルタイムに把握できます。

リモートエージェント

ほかのサーバにビルドを分散させるリモートエージェント機能を利用できます。ビルドやテストは多くの場合、メモリやCPUを大量に必要とするので、複数サーバによる分散ビルドを行うことで負荷を分散させることができます。リモートエージェント機能は負荷分散以外にも有効で、Bambooがインストールされているos以外のプラットフォームでビルドやテストを行いたい場合にも利用できます。

Bambooを使った開発の流れ

ここまで、Bambooの基本的な機能を説明してきました。Bambooを利用することで柔軟にCI環境を構築できることがおわかりいただけたと思います。次は、Atlassianが提供している課題管理システムであるJIRA、Gitリポジトリ管理ツールのBitbucketと連携した場合の開発の流れを紹介します。Bamboo単体の導入でももちろん効果がありますが、JIRAやBitbucketと組み合わせることで、相乗効果が生まれます。

1. ブランチの作成

まずは、対応する課題をJIRAから決めます。対応する課題を決めたら、課題の開発セクション(画面は掲載できませんでしたが課題画面右下)にある「ブランチを作成」をクリックすると、Bitbucket Serverのブランチ作成ページが表示されるので、ブランチタイプや分岐元などを選択してブランチを作成します。

このときに、Bambooでブランチの自動検出を設定している場合は、ブランチのプランが自動的に作成されます。

2. チェックアウト～コミット

先ほど作成したブランチをGitクライアントでローカル環境にチェックアウトして、ソースコードを修正していきます。ここで使用するGitクライアントとしては、Atlassianから無料で提供されている「Source Tree」が、Bitbucket Serverとの親和性が高いのでおススメです。なお、ソースのチェックアウトから次に説明するプルリクエストまでの詳しい流れについては、本連載の第2回「Bitbucket Server + SourceTreeで快適Git環境！」をご参照ください。

意味のある変更ができたらコミット、プッシュしていきます。Bambooのビルドプランの実行トリガーを、リポジトリの変更が発生したときに実行するようにしておけば、プッシュしたあとにビルドやテストが実行されます。

ビルドとテストが頻繁に繰り返し行われるので、早い段階で問題を発見でき、あとあと発生する問題をぐっと抑えられます。ビルドの実行結果は、通知やJIRAの課題画面、Bitbucketのブランチ一覧画面で確認できます。

ビルドやテストが失敗した場合は、Bambooの画面で詳細を確認しましょう。Bambooの詳細画面では、ビルド結果のサマリやテスト結果、コードの変更、ビルド成果物、ログ、ビルドに含まれるJIRAの課題一覧などを確認できます。

Atlassianのコードカバレッジツールである

Cloverを利用している場合は、コードカバレッジのレポートも詳細画面に表示できます。

3. プルリクエスト～マージ

実装を終えたら、次はレビューです。JIRAからプルリクエストを作成してレビューを依頼します。承認を得たらマージです。

ここで万が一ビルドが通っていないソースがマージされると、ほかの開発者に迷惑がかかります。Bitbucketでは、ビルドが成功していないブランチをマージできないように設定できるので安心です。

4. デプロイ

マージを終えたらデプロイです。ビルドが終わったときに自動でデプロイを実行させることもできますが、ここでは手動でデプロイをしてみます。

Bambooから対象のデプロイプロジェクトを開き、デプロイしたい環境のデプロイアイコンをクリックします。デプロイ画面が表示されるので、デプロイしたいビルド結果を選択して、「Start deployment」をクリックします。すると、デプロイが実行され、実行結果が通知されます(**図4**)。デプロイした結果はJIRAの課題画面にも表示されるので、実装がデプロイ済みかどうかがわかります(**図5**)。

▼図4 デプロイ画面

▼図5 デプロイした結果

おわりに

今回はAtlassian社のBambooを紹介しました。CIツールには、ほかにもJenkins(ジェンキンス)、Travis CI(トラヴィス)などがありますが、Bambooの良いところは、世界中で利用される課題管理／バックログ／バグトラッキングツールの「JIRA」、Gitリポジトリの「Bitbucket Server」、ユーザ管理ツールの「Crowd」などとのツールチェインによって開発プロセスを連携できる点です。Crowdによってアカウント管理が統合され、JIRAという課題管理システムからブランチ作成、ビルド／デプロイなどの実行やステータスの自動切り替えができます。

開発システム基盤が分かれていることで、開発者が手動でステータス変更をしなければならないなど、情報の連携に手間がかかっていましたが、Atlassian製品によるツールチェインを利用すると、その手間から開発者を解放することができます。GitリポジトリであるBitbucket ServerとCIツールである「Bamboo(バンブー)」を組み合わせたツールチェインはぜひ試していただきたいです。

Bitbucket Serverについては、本連載の第2回(2016年6月号)「Bitbucket Server＋SourceTreeで快適Git環境!」もお読みください。SD

Hardware

NECプラットフォームズ、Wi-Fiホームルータ「Aterm WG2600HP2」発売

NECプラットフォームズ㈱は10月13日、IEEE 802.11ac規格のWi-Fiホームルータ「Aterm WG2600HP2」を発売した。価格はオープン。

Aterm WG2600HP2は、Atermシリーズの主要機能をすべて搭載した最上位モデル。特徴は次のとおり。

・業界最速の実効スループット

4本のアンテナを利用する4ストリームへの対応により、5GHz帯で最大1,733Mbpsの高速通信が可能となる。また、NECの先端技術である「極技」を活用し、無線用CPUドライバの最適化かつ独自チューニングなどを行って、業界最速の実効スループットとなる約1,428Mbpsの高速通信を実現した。


▲Aterm WG2600HP2

・全方位の電波送信

内蔵アンテナから球体に近い形で全方位に電波を送信できるため、ユーザはアンテナを調整することなく、家中どこにいても高速かつ安定した通信が行える。

・同時使用でも快適な高速通信

Wi-Fi端末に向けて集中して電波を送信する「ビームフォーミング機能」、各Wi-Fi端末の電波強度などにより、適切な周波数帯に自動的に移動させる「バンドステアリング機能」に対応することで、快適な高速通信が可能。

・Wi-Fiの見える化

ルータに接続中のWi-Fi端末の一覧、機器ごとの接続周波数帯や電波強度といった状態表示をスマフォやタブレットなどで確認できる「見えて安心ネット」機能を搭載。また、不正アクセスしたWi-Fi端末を検出し、それを不正な端末として設定すれば、以降は接続拒否できる。

CONTACT
NECプラットフォームズ㈱
URL https://www.necplatforms.co.jp

Service

KDDI・ソラコム、IoT特化型回線サービス「KDDI IoTコネクト Air」をリリース

10月19日、KDDI㈱は、㈱ソラコムとの共同開発によるIoTに特化した回線サービス「KDDI IoTコネクト Air」をリリースした。

KDDI IoTコネクト Airは、KDDIの携帯電話通信網を利用したIoT向けの回線サービス。KDDIから提供されるSIMカード（4G LTE）をIoTデバイスに挿して利用する。大きな特徴は次の2点。

・申し込み～通信の管理はすべてWebから

Webから本サービスに申し込むとSIMカードが送られてくる。そのSIMカードを使ってネットワーク接続設定を行うことで、すぐに利用を開始できる。さらに、通信の利用開始／休止／中断／再開／解約、利用状況の確認、通信速度クラスの変更などもすべてWebから可能で、複数のSIMを一括・集中管理できる。

・IoTに特化した料金体系

SIMカード1枚につき、契約事務手数料1,500円、SMS基本料含め1日10円の基本料金、1MBあたり0.2円～のデータ通信量と、シンプルで安価な設定となっている。本サービスの提供は2016年12月以降を予定しており、今後は回線サービスだけではなく、IoTのための上位のアプリケーション・クラウドサービスまでもセットで提供していくとのこと。

今回発表されたKDDI IoTコネクト Airは、ソラコムの携帯通信コアネットワーク「SORACOM vConnec Core」と、KDDIのネットワーク回線を活用し、2社が共同開発したものとなっている。SORACOM vConnec Coreは、ソラコムが提供するSORACOM AirやSORACOM Beamといった各種サービスの中核となるテクノロジ。Amazon Web Services上に、パケット交換・顧客管理・課金・帯域制限・API/Web・クラウド連携といった機能を実装した、いわば「バーチャルキャリア」であり、ソラコムのコアコンピタンスとなっている。KDDIから「できるだけ早くサービスを立ち上げ、IoT業界に参入したい」という相談を受け、本コア技術を提供するに至ったという。

CONTACT
KDDI㈱ URL http://www.kddi.com
㈱ソラコム URL https://soracom.jp

Topic

アカマイ・テクノロジーズ、2016年第2四半期「インターネットの現状」レポートを発表

アカマイ・テクノロジーズ合同会社は10月3日、2016年第2四半期の「インターネットの現状レポート」を発表した。

インターネットの現状レポートでは、世界中に存在するアカマイのエッジサーバから収集した攻撃トラフィック、ブロードバンド普及状況、モバイル接続、インターネットおよびその利用状況に関連したトピックに関するデータと、そこから分析した長期的な傾向などが四半期ごとにレポートされる。報告のハイライトは次のとおり。

- 世界における平均接続速度は、2016年第1四半期から2.3%減の6.1Mbpsになり、前年比14%増
- 第2四半期の世界における平均ピーク接続速度は、3.7%増の36.0Mbpsになり、前年比2.5%増
- 世界における10Mbpsのブロードバンドの普及率は、四半期比0.7%増、15Mbpsおよび25Mbpsのブロードバンド普及率は、それぞれ0.8%と2.1%減
- アカマイのエッジサーバに接続しているユニークアドレス数（IPv4）は、2016年の第1四半期より1%減で8億をわずかに上回る

CONTACT
アカマイ・テクノロジーズ合同会社
URL https://www.akamai.com/jp/ja

Software

アクロニス、バックアップソフトウェア「Acronis True Image 2017」を発表

アクロニス・ジャパン㈱は9月14日、Windows PCやMac対応のバックアップソフトウェアの新バージョン「Acronis True Image 2017」を発表した。

Acronis True Imageは、ローカルやクラウドに高速にイメージバックアップを行えるソフトウェア。最新バージョンでは次の機能強化が図られた。

- モバイルデバイスのデータをWi-Fi経由でWindows PCに自動バックアップ
- Windows PCやAcronis Cloudへの、モバイルデバイスのデータバックアップ機能を、デバイスの台数に関係なく利用可能
- デバイスの場所に依存することなく、遠隔地からでもバックアップの設定や管理が可能
- Facebookの投稿や写真などのコンテンツのバックアップに対応

オンラインから入手できる製品ラインナップは、ライセンス版（税別4,980円～）と、クラウドストレージが付いたサブスクリプション版（税別3,980円～／年）。

CONTACT
アクロニス・ジャパン㈱
URL http://www.acronis.com/ja-jp

Software

セキュアソフト、セキュリティソフトウェア「SecureSoft mamoret」発表

㈱セキュアソフトは10月18日、ランサムウェアなどのマルウェア侵入・拡散防止ソフトウェア「SecureSoft mamoret」を発表した。

SecureSoft mamoretは1台のパソコン上にインターネットブラウジング専用環境を生成し、一般業務を行うローカル環境と完全に分離することで、ランサムウェアなどのマルウェアからパソコンを守るソリューション。マルウェアをブラウジング専用環境内に封じ込め、さらにWebメールの添付ファイルをその環境内で実行させることで、ローカル環境の安全が保たれる。管理サーバは不要で、パソコンにソフトをインストールするだけで利用できる。対象ユーザとしては、ウィルスチェックソフトのみに頼り、標的型攻撃などの未知脅威対策製品を導入できていない企業、インターネットアクセス環境分離を検討している企業、社内規定上インターネットアクセスを制限されている企業などを想定している。

本ソフトウェアは12月19日より提供開始で、価格は1ユーザの年間使用料が税別9,000円を予定。対応OSはWindows 7/8.1/10。対応するWebブラウザは、現在Internet Explorer 8/9/10/11だが、今後ほかのブラウザにも対応予定とのこと。

CONTACT
㈱セキュアソフト URL https://www.securesoft.co.jp

Report

日本初のプロダクトマネージャー向けカンファレンス、「Japan Product Manager Conference 2016」開催

○日本初の開催でも高い注目度

プロダクトマネージャーカンファレンス実行委員会が主催する「Japan Product Manager Conference 2016」が、10月24、25日の2日間にわたって開催された。プロダクトマネージャー（以下、PM）という職種は、まだ日本のIT業界の職種としては認知度が高くはないと思われているが、そうは思えないほどの盛況ぶりだった。ここではいくつかの講演を紹介する。

○登壇者：Google 徳生氏

グーグル㈱ 製品開発本部長の徳生裕人氏からは、「世界を変えるプロダクトマネージャーになるために」というテーマで講演が行われた。YouTubeのアジア最高責任者としてプロダクトに携わっていた氏は、PMのあるべき姿を模索する良い方法として「デキるPMから知識をぬすむ」ことを推奨。プロダクトスケジュールを考える際には、「なぜこの時期に、これをやらなければいけないのかという意思を明確にする」ことで、プロダクトメンバーの意識やモチベーションが違ってくるとのこと。そして“必要であれば何でもやる”のがPMではあるが、PMがすべてのアイデアを出す必要はなく、多くのアイデアをつなぎ合わせる能力が求められること。外の力を利用してさらに加速する、ときには回り道もするスイングバイ方式を身につけてほしい、などといったアドバイスが語られた。

○登壇者：東京大学 馬田氏

東京大学 本郷テックガレージ ディレクターの馬田隆明氏の講演は「ゼロから始めるPM生活」。スタートアップやプロダクトの成功に必要な「アイデア×プロダクト×実行×チーム×運」の5つの項目について、Y Combinatorの知見を引用しながら解説した。

急成長するためのアイデアは、「悪いように見えて、実は良いアイデア」が望ましいとのこと。「困難な課題」や「小さな市場の独占」を狙うこともポイントとして挙げられた。また、小さな成長を重ねることが重要であり、CEOやPMはその実行のために行動の基準を設定し、KPIの進捗を追跡すること。良いPMは、ほとんどのことに「No」と言う必要があるため、メンバーに多少嫌われる傾向にあるとも指摘。精神的にも肉体的にもタフである必要があることなどが述べられた。

○登壇者：DevJam社 David Hussman氏

DevJam社のDavid Hussman氏は、「プロセスからプロダクトへ」というテーマで講演。同氏の会社でのプロダクトの進め方を3つの開発モデルで紹介した。基本はスクラムだが、プロセスではなくプロダクトの管理を文脈にするようにアジャイルマニュフェストを更新し続けているとのこと。会場にいる人たちに向けて次のようなメッセージが贈られた。「過去10年では、デリバリのしくみを改善することに注力し、実現できた。これから先はそのしくみの上で、良いプロダクトを生み出すしくみ作りがどうやったらできるか、それに挑戦してほしい。」

○登壇者：Niantic, Inc. 河合氏 ×Increments 及川氏

Pokemon Goで注目を浴びるNiantic社のプロダクトマネージャー 河合敬一氏と本カンファレンスの実行委員でもあるIncrements㈱のQiitaプロダクトマネージャー 及川卓也氏による講演は「グローバル企業におけるプロダクトマネージメント」。来場者やTwitterからの質問に答える形で行われた。

PMがうまく仕事をするためのポイントについて河合氏は、会社とプロダクトの軸（ガイドライン、プロダクトビジョン）がしっかりあることが大切だとした。何かを決定しなければならないとき、この軸が常に依りどころとなる。

また、プロダクトの意思決定としてトップダウンがいいのか、ボトムアップがいいのか、という質問には、エンジニアのオーナーシップを高めてあげることはPMにとって重要な役割だとした。最終的にはトップダウンでの決定が必要になるかもしれないが、エンジニアたちがプロダクトに大きくかかわっている意識を持ってもらうことはプロダクトの質を高めることにつながる。

PM自身の評価は、チーム内でリスペクトされているか、きちんとモノを出しているか、で判断される。きついことを言っても、価値のあるプロダクトをきちんと出していれば、自然とリスペクトされる。そして、チーム内の雰囲気作りには、お酒とお肉も重要だとも語った。

◆ ◆ ◆

ここで紹介した以外の講演も多数行われた。興味のある方はhttp://pmconf.jp/をご覧いただきたい。

講演が終わった後には登壇者へ直接質問ができるコーナーが設けられたり、2日目の夕方からはアンカンファレンスの時間があるなど、聴講者自身も積極的に参加でき、プロダクトマネジメントにかかわる知識が深められるイベントだった。

CONTACT
Japan Product Manager Conference
URL https://www.facebook.com/pmconf.jp/

Service

ブロケードコミュニケーションズシステムズ、ネットワーク自動化に関する技術戦略発表

ブロケードコミュニケーションズシステムズ㈱は10月14日、ネットワーク自動化プラットフォーム「Brocade Workflow Composer」の戦略的な展開、および日本での導入事例を発表した。

本製品は、同社が買収したスタートアップ企業Stack Storm社がオープンソースとして公開している「Stack Storm」をもとにしており、サブスクリプションモデルで提供される。この戦略は仮想ルータであるVyattaと同じく、ユーザコミュニティとのフィードバックを重視し、囲い込みではなくオープンな企業姿勢を示すもの。

Brocade Workflow Composerは、ますます複雑化するネットワーク環境において迅速な自動化を推進する製品である。その特徴は、クロスドメイン、すなわちすべてのアプリケーション、ストレージチケットシステムをワークフロー中心にまとめ、発生するイベントごとにまとめて自動化ができること。すでに日本でも、富士通㈱の仮想デスクトップサービス「V-DaaS」で実運用されている。当日の発表会では、同社の髙野徳巳氏が「Brocade Workflow Composer」の導入経緯、システムの運用状況などを発表した。

CONTACT
ブロケードコミュニケーションズシステムズ㈱
URL http://www.brocade.com/ja.html

Report

「QCon TOKYO 2016」開催、～ITエンジニアの思考と技術を深める～

㈱豆蔵は10月24日、「ITが変革するビジネス・組織・社会」を基本テーマに「QCon TOKYO 2016」を開催した。

今回は技術者向けだけでなく、経営層などに向けたセッションも用意され、より幅広い層へのITテクノロジの発信を目指すイベントとなった。講演テーマは、AIやフィンテックなど注目を集めているホットな話題が中心。基調講演は3本あり、最初はHakka Labs社のPete Soderling氏による『エンジニアリングの物語』。これは、IT企業がクリエイティブであるためには、ストーリーとして自社の企業文化を語ることが大事であるというもの。続いて国立情報学研究所教授佐藤一郎氏が『ポスト・ムーアの法則時代のコンピューティング』を発表。CPUの性能がムーアの法則に適合しない時代になって、技術者はますますソフトウェア工学に基づいた仕事をせねばならないという内容。最後は、損害保険ジャパン日本興亜㈱のCTO浦川伸一氏による『日本の情報システムの未来革新に向けて』。5つの保険会社が1つに合併することでシステム統合をいかに進めてきたか、その複雑極まる舞台裏を明かした。いずれのセッションも啓発されるものが多く、ビアバッシュに至るまでエンジニアとして刺激的な1日を得ることができるイベントであった。

CONTACT
Qcon Tokyo 2016 URL http://www.qcontokyo.com

Service

さくらインターネット、「KUSANAGI for さくらのVPS」を提供開始

さくらインターネット㈱は10月31日、プライム・ストラテジー㈱が開発する超高速WordPress仮想マシン「KUSANAGI」を「さくらのVPS」で実行できる「KUSANAGI for さくらのVPS」を提供開始した。

KUSANAGIは、オープンソースのブログ／CMSプラットフォームであるWordPressを高速に動作させるための仮想マシンおよびそのイメージ。「WordPressの実行時間3ミリ秒台」「秒間リクエスト処理1,000」をページキャッシュ非使用で実現する。本製品は、Microsoft Azure、Amazon Web Servicesといったパブリッククラウドのほか、Dockerにも対応している。

さくらインターネットでは2015年5月より、同社のクラウドサービス「さくらのクラウド」でのKUSANAGIの提供を行っていたが、今回新たに、さくらのVPSでの同製品の提供を開始する。さくらのVPSのコントロールパネルから、「CentOS 7（64bit）」上にKUSANAGIがインストールされたサーバを作成することで利用を開始できる。追加料金不要で導入できるため、企業のWebサイトはもちろん、個人のブログにおいても、より快適にWordPress環境を利用できる。

CONTACT
さくらインターネット㈱
URL https://www.sakura.ad.jp

ひみつのLinux通信

作)くつなりょうすけ　@ryosuke927

地獄篇

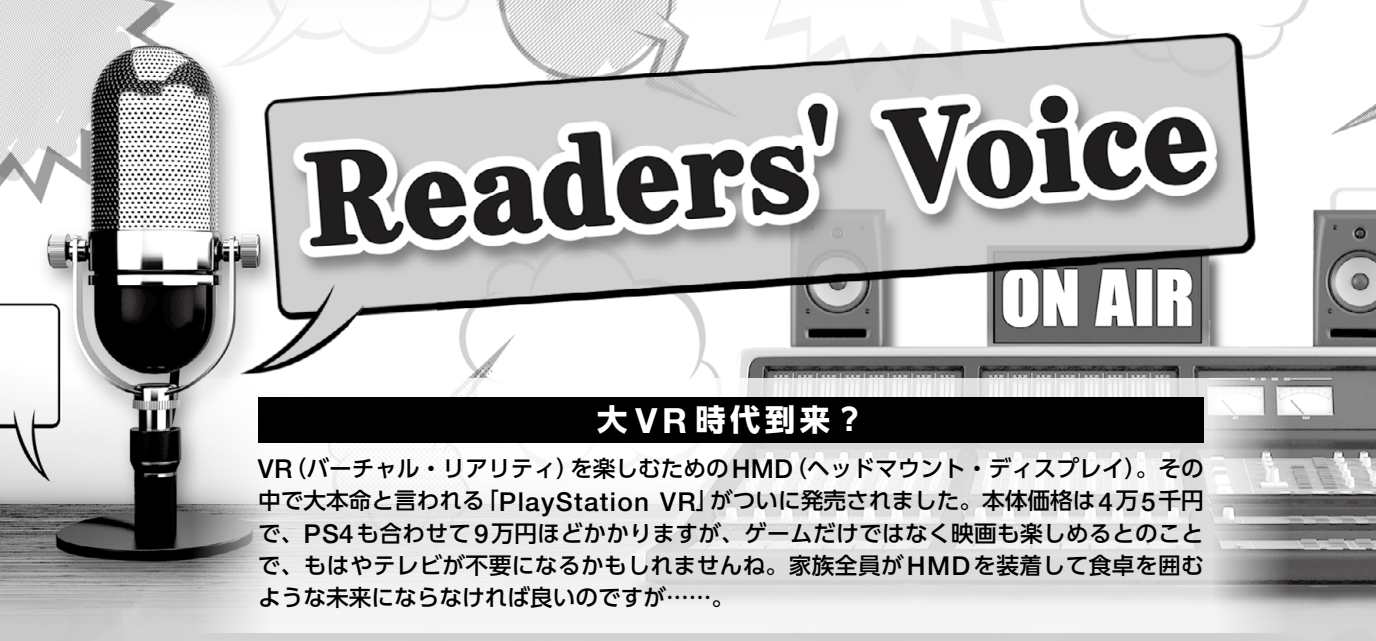
揚揚はあざなえる縄のごとし。チャーシューすれば甘する自称プロ・ジロリアンのくつな先生のマンガが読めちゃったりなんかしちゃったりする(広川風に！)のは本誌だけ。

恒例年末年始特番

第34回 天国と地獄

天国篇

どんなに寂しくても、OSCに行けば宮原さんに逢えるから大丈夫



大VR時代到来？

VR（バーチャル・リアリティ）を楽しむためのHMD（ヘッドマウント・ディスプレイ）。その中で大本命と言われる「PlayStation VR」がついに発売されました。本体価格は4万5千円で、PS4も合わせて9万円ほどかかりますが、ゲームだけではなく映画も楽しめるとのことで、もはやテレビが不要になるかもしれませんね。家族全員がHMDを装着して食卓を囲むような未来にならなければ良いのですが……。

2016年10月号について、たくさんの声が届きました。

第1特集 Webサーバはなぜ動くのか？

Webサーバについての疑問、「どのようにクライアントとデータをやりとりしているのか」「どうやってプログラムを実行しているのか」「いかにしてDBと接続しているのか」の疑問に答えました。さらに具体例として、Node.jsとRuby on Railsのしくみも解説。

基本的なこともあらためてまとまっていると、自分の技術知識の整理にとても良いです。 **のりぃさん／埼玉県**

Node.jsの動作が最近気になっていたのでちょうど良かった。
澤田さん／千葉県

Node.jsのイベントループモデルがわかりやすかった。 **本の海さん／奈良県**

Node.jsをはじめとするサーバサイドのJavaScriptの何が良いのか、理解できました。 **tack41さん／愛知県**

WebサーバやCGIの動作のしくみのところは、個人的には初めて調べたときの20年ちょい前を思い出しました。
福田さん／神奈川県

わからないで開発している人、けっこういるんじゃないかなぁ……。
tekitoizmさん／東京都

クラウドやWebはほぼ毎日利用しています。その入り口のWebサーバは、水や空気のようにあって当たり前の存在になっています。詳しいしくみを知ることでこれからもっと活用していきたいです。 **牧さん／大阪府**

Webを利用するとき、またWebアプリを開発するときでさえ、Webサーバの詳しいしくみは意識したことがなかった人が多いようです。今回は、若い人に人気があるNode.jsとRuby on Railsも取り上げたことで、興味を示した読者も多いようでした。

第2特集 新しいPostgreSQLの教科書

2016年に生誕20周年を迎えたDBMS「PostgreSQL」を特集しました。PostgreSQLの歴史、インストールと使い方、特徴、Oracleからの移行Tips、さらにはコミュニティ情報まで、盛りだくさんの内容でした。

子供が情報システム部に異動、そしてPostgreSQLを使うことになり、タイムリーな内容でした。
お花ところ助さん／大阪府

今読んでいるミック氏のデータベース本でPostgreSQLを使っているため、タイムリーだった。
トミオくんさん／福岡県

自分の知識の確認と新人に読ませるのにとても良いと思った。
なおきさん／千葉県

データベースといえばMySQLとばかり思っていた人は……自分だけ？　データベースを学ぶなら知っておいたほうが良い内容が満載でした！
南雲さん／埼玉県

Oracleの移行を検討しており、注意点や高可用性の構成例が参考になりました。 **橿山さん／埼玉県**

DBサーバについては選択肢があまりないので、PostgreSQLの動向は常に見ておきたいのですが、こうやってまとめて特集してもらえるととても助かります。また使おうかな。
romeosheartさん／長崎県

以前のMySQLの特集と比べながら読

10月号のプレゼント当選者は、次の皆さまです

①OWL-KB109CBRU2-BK
村橋究理基様(北海道)

②世界の盾 JF-PEACE55
(黒)りょうじ様(東京都)、(白)高松直斗様(愛知県)

③LinuxCon Japan 2016 ノベルティグッズ
髙橋輝大様(東京都)

④『Amazon Web Services活用入門』
ぴょうへい様(大阪府)、土屋建様(神奈川県)

⑤『脆弱性診断スタートガイド』
今井英敏様(千葉県)、永作肇様(東京都)

⑥『パーフェクトJava EE』
矢野誠様(東京都)、yasu様(広島県)

⑦『独習Python入門』
小林采豊様(神奈川県)、ゆんど様(長野県)

※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。

んでいます。MySQLとPostgreSQLはどちらか一方しか選べないのが不便です。同じデータに対してどちらからでもアクセスできるとか、どちらかが上位互換とかになってもらえれば、違いを気にせずに済ませられると思います。 中村さん／大阪府

オープンソースのDBで、MySQLとシェアを二分するPostgreSQL。注目度はかなり高いようです。コメントをみると、Oracle Databaseからの以降を考えている人も多いようですね。

一般記事　CHIRIMENシングルボードコンピュータ入門

CHIRIMENは、HTML/CSS/JavaScriptというWebの技術で制御できるシングルボードコンピュータ。記事では環境のセットアップから、距離センサとミニモニターを使ったWoTサイネージの開発までを扱いました。

リアルタイムの動画エンコード／デコードができそうだなと思いました。 A758さん／山口県

Webの技術でIoTができるって、夢があると思った。 tomato360さん／東京都

HTML/JavaScript/CSSで開発可能という点が目新しく、ぜひ試してみたいと思った。 若山さん／千葉県

WoTサイネージの記事をたいへん興味深く読みました。CHIRIMEN買います。 niwaさん／神奈川県

Webの技術が使える今までにないIoTデバイスということで、「興味が湧いた」「買いたい」という声が多く寄せられました。いろいろな用途に応用できそうですね。

短期連載　乱数を使いこなす【3】

シミュレーションやセキュリティに欠かせない乱数について、作り方／使い方の両面を全3回で追います。最終回では、OSやCPU内部でどのように乱数が作られるのか、それらをどう利用するのかを解説しました。

古典的な合同法などの乱数を過去に勉強しただけでしたので、最近の乱数についての記述がたいへん役に立ちました。参考文献もありがたかったです。新しい乱数生成機も興味深かったです。 Qkobさん／富山県

普段何気なく使っている乱数も、背景にはこんな技術があるのかと感心した。 かっぱてっくさん／熊本県

同号のLinuxカーネル観光ガイドでも乱数について取り上げられており、2つの共通点などを探しながら学べた。 渡邊さん／東京都

ユニークキー生成など、乱数使用頻度は高いのでためになった。 とんぴさん／沖縄県

エントロピーと仮想化環境についての記述が参考になった。 psiさん／東京都

難易度が高いながらも人気の高かった本連載。乱数は、数学や物理学が絡む奥が深い領域で、すぐには応用できない知識かもしれませんが、押さえておいて損はありません。

コメントを掲載させていただいた読者の方には、1,000円分のQUOカードをお送りしております。コメントは、本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」からアクセスできるアンケートにてご投稿ください。みなさまのご意見・ご感想をお待ちしています。

次号予告

Software Design
January 2017

2017年1月号
定価（本体1,220円+税）
192ページ

12月17日発売

［第1特集］新春bash書き初め
シェル30本ノック
基本からちょっとした応用までシェル力を鍛える

［第2特集］心機一転・乾坤一擲
転職で成功するには何をすべきか?

［第3特集］中井先生直伝
機械学習の勉強方法

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

お詫びと訂正

以下の記事に誤りがございました。読者のみなさま、および関係者の方々にご迷惑をおかけしたことをお詫び申し上げます。

■2016年12月号
●P.22　第1特集「クラウドコンピューティングのしくみ」第1章　左段　注7）
［正］http://www.vmware.com/jp/products/esxi-and-esx.html
［誤］http://www.vmware.com/jp/products/esxiandesx.html
●P.26　第1特集「クラウドコンピューティングのしくみ」第1章　右段　注17）
［正］https://www.ibm.com/marketplace/cloud/bare-metal-server/jp/ja-jp
［誤］https://www.ibm.com/marketplace/cloud/baremetalserver/jp/jajp
●P.26　第1特集「クラウドコンピューティングのしくみ」第1章　右段　注19）
［正］https://cloud.google.com/container-engine/　［誤］https://cloud.google.com/containerengine/
●P.33　第1特集「クラウドコンピューティングのしくみ」第2章　右段　図10 キャプション
［正］CloudTrailによるログ管理　［誤］CloudTrailによる構成管理（その1）

休載のお知らせ

「Androidoで広がるエンジニアの楽しみ」（第12回）、「るびきち流Emacs超入門」（第32回）、「SOURCES～レッドハット系ソフトウェア最新解説」（第5回）は都合によりお休みさせていただきます。

SD Staff Room

●今年は「猫」様に表紙参加いただいたおかげで、5月以降盛り返して非常によかった。猫が地球をまわしているのだ。渡る世間は猫ばかり。これからずっと猫が表紙で良いかもしれない。昨年の今ごろデザイナーさんとの打ち合わせで天啓があり、そこで猫に決めたのだった。でも、本当は犬のほうが好きなんだけどね。（本）

●今回の「IT昔話SP」はいかがでしたか?　業界長い方は、昔のことをいろいろ思い出されたのではないでしょうか。若い方にはチンプンカンプン（←死語?）だったかも。今現在のIT業界も、先達の苦労の蓄積があったからで、あと数十年もすると今現在の苦労話が「昔話」になっていることでしょう。（ボンコツ幕）

●TOTOのショールームに行ってきました。トイレだけじゃなく、浴室やキッチンといった商品も扱っていて、私は浴室にも興味津々。サウナみたいに浴室にベンチがあるモデルを熱心に見ていたら、「それ、ご高齢者向けですけどね」と担当者。ふむ、老後の楽しみに浴室にこだわるってのもありだな。（キ）

●十数年同じ美容師さんに髪を切ってもらっています。当初はスタイリストという役職だった美容師さんも、数年でトップスタイリストになり、今では店長に。頼むのは毎回同じカットでも彼女の出世のたびに価格がアップ。予想外の値上げに毎度ビックリですが、きっと私の髪型もレベルアップしているはず!（よし）

●姉の結婚式に行ってきました。和式だったので神前式ですね。新郎は紋付羽織袴、新婦は白無垢、宮司と巫女が式を執り行い、始終厳かな雰囲気で進行しました。ただすごく小さな神社だったので、すぐ外で賽銭箱に小銭が投げ入れられるチャリンという音が鳴りっぱなし。金運が上がりそうな一日でした。（な）

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
2016年12月号

発行日
2016年12月18日

●発行人
片岡　巌

●編集人
池本公平

●編集
金田富士男
菊池　猛
吉岡高弘
中田瑛人

●編集アシスタント
根岸真理子

●広告
中島亮太
北川香織

●発行所
㈱技術評論社
編集部
TEL：03-3513-6170
販売促進部
TEL：03-3513-6150
広告企画部
TEL：03-3513-6165

●印刷
図書印刷㈱

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

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