Software Design
[ソフトウェア デザイン]
OSとネットワーク、
IT環境を支える
エンジニアの総合誌
3
2017
2017年3月18日発行
毎月1回18日発行
通巻383号
（発刊317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

» ログ分析 » アジャイルなインフラ » メールのセキュリティ

マーケティング&
サービス向上に
役立つ

Special Feature _ 1

ログ&データ分析基盤入門

Fluentd
Embulk
Hadoop
Spark
Hivemall

実践!
DevOps

Special Feature _ 2

CIで インフラ開発を 効率化

Jenkins+Gerrit+Ansible+
Serverspecで基盤構築

どうなっている
セキュリティ?

Special Feature _ 3

なりすまし メール対策を していますか

Extra Feature _ 1

PostgreSQLのGPU拡張モジュール
PG-Strom入門(前編)

Extra Feature _ 2

その決済方法の内側を知る
Amazonログイン&ペイメントのしくみ

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。

3
2017
［目次］
Software Design

contents

Special Feature 1 第1特集

マーケティング＆
サービス向上に
役立つ

ログ&データ分析基盤入門

3
2017
［目次］
Software Design

Special Feature 2 第2特集

CIでインフラ開発を効率化 実践! DevOps

Special Feature 3 第3特集

どうなってる? なりすましメール対策

Extra Feature 一般記事

àla 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］
Anatolii / AdobeStock
［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。

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。

未来を味方にする技術

これからのビジネスを創るITの基礎の基礎

斎藤昌義　著

四六判／264ページ
定価（本体1580円+税）

ISBN978-4-7741-8647-4

数週間かかっていた仕事を、たった1日で終わらせる。「人間にしかできなかったこと」を機械に置き換え、「人間にはできなかったこと」を実現する
——あたらしい常識を次々と生み出す原動力となるITを使いこなすためには、どんなことを押さえればいいのか？
人工知能、IoT、FinTech、シェアリングエコノミー、bot、農業IT、マーケティングオートメーションといった最新のトレンドから、これからも変わらないITとの付き合い方までを図解とともに解説。
本文の図はPowerPointデータとしてダウンロード、ロイヤリティフリーで利用可能。

変革の軌跡

世界で戦える会社に変わる "アジャイル・DevOps" 導入の原則

A5判／160ページ
定価（本体1980円+税）

ISBN978-4-7741-8663-4

Gary Gruver・Tommy Mouser　著
吉羽龍太郎・原田騎郎　翻訳

「4500万ドルの開発費用の削減」「サポートする製品数が140%増加」「イノベーションのためのキャパシティが8倍に増加」・・・
これらは著者の2人が企業のソフトウェア開発に変革をもたらした結果である。現代において、ソフトウェアが生み出す価値はますます影響を増し、既存の企業のビジネスを1つのソフトウェアが脅かすことも少なくない。しかし、古くからソフトウェア開発に関わってきた企業でさえも、変化の速いマーケットの中で競争力を失っている。一部の企業では、ソフトウェア開発を効率化したいと考えはじめているが、エンタープライズレベルでうまくいっている例は少ないのが実情である。本書は、制限の多い組織全体にアジャイルとDevOpsを適用するこれまでにない組織変革の指南書である。

進化する銀行システム

24時間365日動かすメインフレームの設計思想

ISBN978-4-7741-8729-7

星野武史　著／花井志生　監修
A5判／256ページ　定価（本体2580円+税）

人々の生活や企業活動を支える銀行のオンラインシステム。地震などの大規模災害対策として、銀行の国際競争力を高める手段（振込の24時間化や休日・夜間の即時決済など）として、24時間365日止まらずに稼動し続けることが求められています。本書は、多くの銀行システムで採用されているメインフレームのしくみ、とくに信頼性、可用性、保守性を高める技術をわかりやすく解説します。銀行システムの歴史や特性、メインフレームやそのOS「z/OS」の特徴や機能を学びたい人、システムの障害・災害対策を検討したい人に役立つ1冊です。

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

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

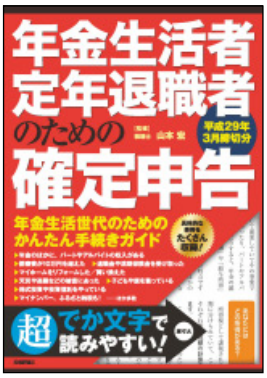
家族が働いていたり副業があるために確定申告が必要なサラリーマンや、主婦、パート、アルバイト、年金受給者はもちろん、個人事業主やフリーランサー、不動産オーナーまで、個別のケースごとに具体的な事例をたくさん掲載しています。自分の状況に近い事例を選んで、番号順にマネして書くだけで書類が記入できるので、忙しくて書類作成に時間をとれない方や、初めて確定申告を行う方などに、特におすすめしたい1冊です！
ふるさと納税をした方の申告方法や、確定申告の基礎知識もわかります。

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

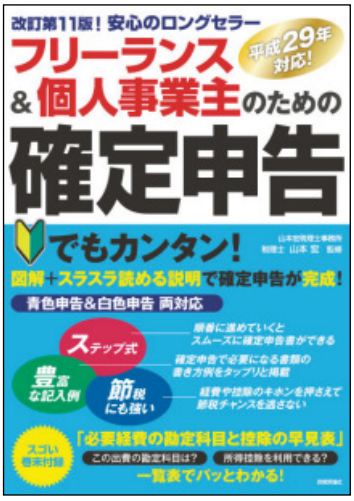
フリーランスや個人事業主の方が1年間の仕事の成果を書き入れる確定申告書。その申告書に何を書くのかによって、手元に残るお金の額は違ってきます。確定申告には、「こうしたらトクになる」というやり方がありますが、納税者に有利になる（納める税金が少なくなる）情報を税務署が教えてくれることはありません。節税にはとくに複雑な手続きは必要ないため、節税できるかどうかを分けるのは、ズバリ“知っているかどうか”です。確定申告を絶好の節税の機会にする方法を、元国税調査官の著者がお届けします！



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

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、ノ~\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 >> ノ~\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)

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5回 製品要求仕様書（PRD）の書き方 その2

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

PRDに含める内容

今回は、製品要求仕様書（PRD：Product Requirements Document）の内容についてもう少し詳しく説明していきましょう。

典型的なPRDの構成は次のようになっています。

- **概要**
- **背景**
- **製品原則**
- **スコープ**
- **対象ユーザ**
- **ユースケース**
- **市場分析**
- **競合分析**
- **機能要求**
- **そのほかの技術的要求**
  - －システム要求
  - －セキュリティ要件
  - －プライバシー要件
  - －パフォーマンス要件
- **リリーススケジュールおよびマイルストーン**
- **マーケティング計画**

まず、“概要”には開発する製品やPRDについての説明を書きます。特記するべきことがない場合はシンプルでかまいません。“背景”には、その製品を開発するに至った背景などを記述します。先行した製品や関連製品があり、それが今回開発する製品に関わる場合にはその内容を含めます。

次の“製品原則”から“ユースケース”までがPRDの中でもとくに重要な「何を作るのか」のコアの部分となります。製品原則は英語ではProduct Principlesと呼び、その製品はそもそも何なのかを簡潔にまとめる部分です。機能の取捨選択を行う場合やタスクの優先度を検討する場合などにぶれないために必要となります。この連載の第3回（本誌2017年1月号）でも書きましたが、スケジュールを死守するために、予定していた機能を諦めなければならないことや市場環境の変化に合わせて、予定していなかった機能の搭載を検討しなければならないこともあります。そのときの判断基準となる内容をここには記述します。書き方に悩むかもしれませんが、「何をするものか」が完結に記述されていれば十分です。たとえば、世界初の日本語ワードプロセッサである東芝のJW-10の開発では、製品コンセプトを次の3つにまとめています。

①誰でも手書きより速く入力できる
②持ち運びできる
③どこからでもアクセスできる

実際には最初の製品であるJW-10ではこのうちの①しか実現できませんでしたので、この「誰でも手書きより速く入力できる」ことが製品原則だったのでしょう。このJW-10の例でわかるように、製品原則はわかりやすく、覚えやすい必要があります。プロダクトチームの全員がしっかりと常に思い出せるものであることが大事です。

次の“スコープ”では、製品に含めるものと含めないものを記述します。ゴールとノンゴールとしても良いでしょう。“対象ユーザ”には対象とするユーザを、そして“ユースケース”にはその対象

ユーザがどのようにその製品を使用するかを記述します。開発する製品は全ユーザを対象にしているということもあるかもしれません。しかし、その場合でも誰がどのような状況でその製品を使うかは想定しておく必要があります。すべての人のための製品は結局誰のための製品にもなっていない、ということはよくあります。自分や家族が対象ユーザになり得るのであれば、本当に自分が使うものなのかを考えてみると良いでしょう。

“市場分析”や“競合分析”は任意記述でもかまいません。最初から用意されていなくても良いでしょう。市場や競合をただ単に分析するというよりも、製品の差別化を考える必要があります。

次の“機能要求”がPRDのメインです。製品に必要とされる機能を記述します。これを見れば、何を作ればよいかわかる内容である必要があります。そのための理由や背景はここに至るまでに書かれているはずですので、ここではシンプルに必要とされる機能の記述に徹します。ユーザが実際に操作する製品の場合はユーザインターフェースを含むユーザ体験(UX)フローが中心となることもあります。ワイヤーフレームなどを使い、しっかりとどのような体験を作り上げたいかを記述します。デザイナーがチームにいる場合には、デザイナーとともに作り上げていくのが良いでしょう。PRDはProduct Manager(PM)が中心となって書き上げますが、必ずしも他者の介在を否定するものではありません。プロダクトチームが常に参照すべき存在であるPRDは、その作成もチームで行うものであってかまいません。

製品をどのように実現するかはPRDの範疇ではありませんが、製品として技術的な部分での要求がある場合にはそれも記述します。たとえば、ある一定以上のパフォーマンスを実現することが製品として必須ならば、そのパフォーマンス要件を記述します。セキュリティやプライバシーでとくに守るべきことがあるならば、それなども記述します。

スケジュール管理は別途プロジェクト管理の一環としてしっかりと行う必要がありますが、製品として期待するスケジュールがあるならば、それも記述します。ここには社内テストやドッグフード[注1]の要否なども記載すると良いでしょう。そして最後に、“マーケティング計画”を立てます。

PRDはLiving Document

見ていただいたように、PRDはその内容が多岐にわたり、全部を記述するには時間がかかります。作り方としては、ユースケースまではいっきに書き上げてしまうことをお勧めします。この前半部分は、そもそもなぜこの開発を行おうとしているのかというプロジェクトの根幹に関わる部分であり、プロジェクトが開始されるに際してはすでに用意されているべきものです。プロジェクトの早い段階で、それをしっかりと文字化しておくことが大事です。

一方、市場分析や競合分析は市場が特殊だったり、競合の存在が製品を考えるに際して重要だったりしない限りは、当初から用意する必要はありません。最後まで省略したままで良いこともあります。

また、機能要求はプロダクトチームのメンバーとの議論で何度も変更が生じる可能性があります。一度決まった機能であっても、ユーザテストの結果を受けて、変更することもあるでしょう。本当はないほうが良いのですが、実装が始まった後に、仕様漏れが発覚し、機能の再検討をする可能性もあります。

そこで、PRDは一度完成したら、その後は更新しないという運用ではなく、永遠に更新し続けるような運用を目指すことをお勧めします。このようなドキュメントを英語ではLiving Documentと呼びます。常に、PRDを参照すれば、その時点でのその製品のもっとも正しい情報を得ることができる。そのような存在を目指して、PMを中心としてプロダクトチーム全員で育てていくことを推奨します。SD

注1) 開発中の製品や機能を社内で実際に使ってみること。

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99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。

MicroProfile

「MicroProfile」とは

「MicroProfile」は、エンタープライズJavaテクノロジーによってマイクロサービス・アーキテクチャを構築するための新しいプロファイルです。Red HatやIBM、Tomitribe、Payaraといったベンダーや、LJC（London Java Community）などによって共同で提唱され、2016年9月にバージョン1.0がリリースされました。また、同12月にEclipse Foundation傘下で新プロジェクト「Eclipse MicroProfile」もスタートしています。

マイクロサービス・アーキテクチャとは、システム全体がより小規模な独立したサービスの組み合わせによって構成されるようなアーキテクチャを指す用語です。その特徴を簡潔にまとめると次のようになります。

- 単一のシステムが複数の小さなサービスの組み合わせによって構成される
- 各サービスはそれぞれ最低限の管理システムを持つ独立したプロセスで実行され、軽量なしくみによって外部のサービスと通信する
- サービスごとに使用する言語やストレージ技術などが異なることもある

このアーキテクチャではシステムを構成する各サービスの独立性が高いため、サービス単位で開発やデプロイ、スケールすることができ、市場の変化への応答性が極めて高いという強みがあります。また、モジュールの境界が明確なことから問題発生時の影響範囲の特定がしやすい点や、複数のチームでの開発や運用が行いやすい点なども大きなメリットとして挙げられています。

MicroProfileが目指すもの

エンタープライズJavaの標準仕様であるJava EEでもマイクロサービスへの対応は進められていますが、市場の変化のスピードと比較してJCP（Java Community Process）による標準化のスピードは遅く、結果としてマイクロサービスのための現実的な選択肢になっていないという指摘もあります。

標準化されたJava EEは、互換性や安定性という観点からすれば開発者に大きなメリットを提供します。その反面、仕様の策定に時間を要するため、ペースの速いマイクロサービスの世界にはまだ完全に追随できているとは言えません。その一方で、Javaのエコシステムには Wildfly SwarmやWebSphere Libertyなどといった選択肢も登場しています。これらのプロジェクトではJava EEと非Java EEの両方のテクノロジーを活用することでマイクロサービスの実現を手助けする革新的な機能を提供しますが、標準ではないためにポータビリティや互換性に対するリスクを抱えています。

MicroProfileは、このような現状を打破するために提唱されました。MicroProfileが目指すのは、マイクロサービス・アーキテクチャの基盤となるJava EEベースの標準フレームワークを確立することです。そして、その標準に準拠したアプリケーションサーバの開発を促進することや、開発者に互換性のあるコンテナのエコシステムを提供することなどが目標として掲げられています。

MicroProfileがJCPによる標準化と異なるのは、市場ニーズに応じた素早いイノベーションを最優先事項に掲げている点です。そのために、まずは企業・団体の垣根を越えた中立的なコミュニティによる議論でプロファイルを策定し、その次の段階としてJCPやそのほかの標準化団体への提案を行うというプロセスが採用されています。標準化をゴールとすることで将来的なリスクを排除しながらも、標準化プロセスそのものがイノベーションの妨げにならないようにしようというわけです。

MicroProfileの最初のバージョンはJAX-RSやCDI、JSON-Pなどを含むJava EEテクノロジーをベースとして作られました。次のリリースに向けては、JWT PropagationやJCache、Persistence、Bean Validation、Web Socketsなどが主要な要件として検討されているとのことです。SD

MicroProfile.io
http://microprofile.io/
Eclipse MicroProfile
https://projects.eclipse.org/proposals/eclipse-microprofile

DIGITAL GADGET vol.219

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

新しい領域を切り開く家電

家電の操作は音声で？ 次に流行する家電とは？

スマートホームと呼ばれる、家電と住まいとが連携し、デジタル化、ネット化する「賢い（スマートな）」家への憧れは昔からありました。SF映画の中では、口に出すだけで希望の料理や飲み物が瞬時に調理されて出てくる家電も見かけますが、そこまでの便利さは、物理法則が変わらない限り無理でしょう。けれども「口に出すだけで」という部分に限っては、少しずつ実用レベルの技術が浸透し始めています。

数年前のCES（Consumer Electronics Show）から毎年スマートホームや家電のネット化は注目を浴び、さまざまな製品がリリースされてきましたが、そう簡単には一般に普及してきませんでした。今年のCES 2017では、Amazon Alexaなど、音声操作のためのプラットフォームの存在により、次の段階に進んだ印象があります。大げさな評価としては、Amazon Alexaは次世代のOS、プラットフォームとして君臨するのではないかという予想もあります。つまりハードウェアを利用したり、サービスを利用したりする手段としてOSやブラウザ、スマートフォンが存在していましたが、それらがAmazon Alexaで済むようになるのです。確かに、自社のサービスをAlexaで使えるように、Skill（後述）という手順を登録するサービスは増えつつありますし、パソコンメーカーであるLenovoがSmart Assistantという Amazon Alexa対応の製品を出してきたり、大手車メーカーのFordは、自社の車両にAmazon Alexaを搭載することを発表したりと、Amazonだけでなく、周辺の動きが活発なことが見て取れます。

また、家電製品全体の傾向としても、家電そのもの単体の製品だけではなく、その家電製品によって得られるライフスタイルや体験、サービスを売り、それらのメリットをアピールすることが多くなってきました。

過度なスマートホーム化にはまだまだ課題もあり、あまりにも自動化しすぎると手動で扱えなくなったり、リモコンをまとめすぎると、いざというときに困ったり、かえって不便になったりもします。またリモート監視や省エネ、電気代節約、電力消費などは、最初こそもてはやされても、わざわざ利用する頻度は下がっていくとも言われています。

一方、家庭内の出来事で、いままでわざわざ何かをしなければいけなかった事柄を、操作なしで情報提供してくれたり、操作を代行してくれたりするのは、とても便利で受け入れてもらいやすいと言われています。

CES 2017での製品発表は、一般的な家電製品はもとより、キッチンで使うもの、睡眠関連、赤ちゃんグッズ、ハイテクスポーツ用品、ペット関連、高齢者向け、子供向けのテクノロジーなど、多岐にわたり、かつ利用者層を絞ったもの、革新的なものが数多く登場しました。最近の技術革新のスピードが早まってきたこともあり、どこかで見たことがあるようなもの、従来の延長線上にあるものから、今までに見たことのなかったようなもの、想像しなかったようなものの数が増えてきました。Innovation Awardsを獲った製品のほか、いくつか革新的なデジタルガジェットを写真で紹介しましょう。

声による操作の時代

iPhoneのSiriやGoogleの音声検索の浸透と性能の向上で、何かを声に発して操作するという違和感も少しずつなくなってきました。もちろん人混みの中、電車の中などで声を出すにはまだまだ心理的障壁が高いかもしれませんが、Bluetoothヘッドセットをつけているビジネスマンや、イヤフォンに搭載されたマイクで通話している人も見かけるようになり、独り言を言っているような違和感は、以前ほどではないと思われます。

今回のCES 2017では、数ある展示物のうち、数百種類もの機器がAlexa対応をうたい、車内のコントロールから冷蔵庫まで、ありとあらゆる種

※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。

別の家電やサービスのAlexa対応が注目されました。

Amazon Alexaは、Amazonが音声アシスタントAmazon Echo用に提供しているクラウドベースの音声認識サービスを、Amazon以外のサードパーティに公開したもので、音声操作をさまざまなハードウェアやソフトウェアと連携させることができます。最初のうちは、音楽再生の音声操作やAmazonでオンライン商品を注文したり、単純な受け答えだったところから始まりましたが、最近では各社のサービスと連携を進め、サービスの幅が広がってきています。

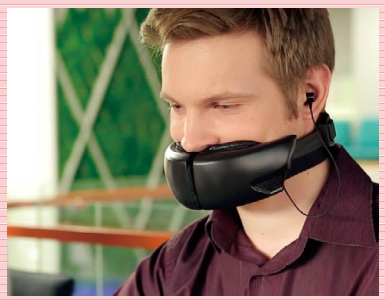
Skill（スキル）と呼ばれるAmazon Alexaのサービス登録は、最初の頃は数が少なかったのですが、現在では5千を超える規模に増えつつあります。さらにAmazonはAlexaの活用を進めるために、Alexa Fundという総額1億ドルの基金による支援も進めており、新興の機器ベンチャーも台頭してきています。Alexa Voice ServiceのAPIを追加するだけで、既存のハードウェアデバイスがAlexa対応になり、高品質な音声コントロールが可能になる様は、なんとも魅力的な環境でしょう。

ちなみにAlexaの受け答えは、対応機器を持たずとも、ブラウザで「https://echosim.io/」にアクセスすることで試すことができます（米国Amazon.comのアカウントが必要）。

これからの家電、これからの体験

CES 2017で注目された家電製品群は、デジタルガジェット的には次の分類ができると考えています。

- **今までなかった分野への家電の進出：** 洗濯物をたたむ装置、空気の状態を監視するエアトラッカーなど



ないしょ話用ヘッドセット「**Hushme**」
http://gethushme.com/

ワイヤレス給電「**Cota Tile**」
http://www.ossia.com/cota/

指輪型睡眠トラッカー「**Motiv Ring**」
http://mymotiv.com/

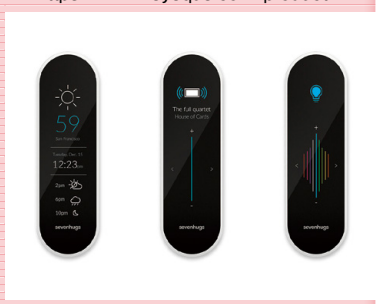
ポストイット専用プリンタ「**nemonic**」
http://www.mangoslab.com/

空間定規「**Cubit**」
http://letsplott.com/cubit.html

視覚検査ツール「**EyeQue Personal Vision Tracker**」
https://www.eyeque.com/product

360度ヘッドフォン「**OSSIC X**」
https://www.ossic.com/

ジェスチャー操作、プロジェクタ式TV「**Z4 Aurora**」
http://www.xgimi.com/Z4.html



向けた方向にある家電を操作する統合リモコン「**Smart Remote**」
https://remote.sevenhugs.com/

●**あって当然なのになかった分野:** バイク用ヘルメットに装着するブレーキランプなど

●**技術革新によって可能になったもの:** 「HoloLamp」などの立体投影装置、Hondaの倒れないバイクなど

●**既存の製品を革新するもの:** デジタルインスタントカメラ「Polaroid Pop」

●**ネット時代ならではのもの:** ネットにつながるトースター「Griffin Connected Toaster」、ネット対応ぬいぐるみ「Parihug」

●**人間の仕事を代替するもの:** 自動走行荷物運搬ロボット「TwinswHeel」

●**さまざまな出来事のパーソナル化:** ピンポイントでの天気予報「WEZR」

Amazon Alexaの台頭により、声による操作で、デジタル機器の操作も、人と人同士のコミュニケーションに近くなってきました。一方、「声で操作できるということは、人と同じくらいの能力なのだから、これくらいのことは難なくこなしてくれるだろう」という、実力以上の過度な期待感も高まります。

音声インターフェースは、場の空気の読めない人に対応するような感覚で指示をしなければいけないのでしょうか。それともその段階はもう乗り越えており、わざわざ細かい配慮なしに、人と同じような会話のやり取りで理解してもらえるのでしょうか。人と同じだと思えばイライラが募るかもしれませんが、こちらが合わせて話せば理解してもらえる相手として考えれば、もうすでに十分実用的なのかもしれません。

何かを操作し、その操作を覚えて慣れなければいけなかった時代から、人が本来したいことを日常の言葉で指示でき、慣れれば慣れるほど、そのコンテキスト(状況)を理解したうえで対応してくれるのが、音声インターフェースの理想形かもしれませんね。SD

Gadget 1

PowerUP FPV

https://www.poweruptoys.com/products/powerup-fpv

紙飛行機ドローン

PowerUP FPVは、カメラを搭載した紙飛行機ドローン。プロペラがいくつもついた一般的なドローンとは異なり、ベースとなるのは紙飛行機で、199.99ドルで販売中。コントロールはスマートフォンアプリで可能なほか、搭載されたカメラ映像は、スマホVR眼鏡で視聴することができます。VGAサイズ、30fpsの画質です。飛行時間は、紙飛行機の出来にもよりますが、だいたい10分ぐらいとのこと。スマホで操縦することも、オートパイロットで安定した飛行を選ぶこともできます。元々はクラウドファンディングKickstarterで5万ドルを募った商品です。

Gadget 2

Sleep Number 360 smart bed

https://www.sleepnumber.com/360

スマホ対応スマートベッド

Sleep Number 360 smart bedは、スマートフォンと連携して操作することのできるスマートベッドです。いびきの音を感知して頭部の位置を変えたり、体の位置に応じて、自動調整してくれます。また、ベッドの足の部分を温める機能もあり、就寝間にスマートフォンで設定しベッドを温めておくことができます。睡眠状況をBluetooth経由でスマートフォンと連携して記録していくこともできます。Sleep Number 360ではSleepIQと呼ばれる独自の睡眠品質指標をもとに、どれだけ熟睡できたかをスマートフォンアプリで確認できます。

Gadget 3

Mars by crazybaby

http://crazybaby.com/mars

宙に浮くスピーカー

crazybabyのMarsというスピーカーは、磁力で中に浮く部分が特徴的なスピーカーです。UFOのような円盤がクルクルと回りながら音を出します。360度方向に音が広がるスピーカーで、テレビ会議のスピーカーフォンとしても利用できます。もともとはクラウドファンディングIndiegogoから始まったプロジェクトで、予定の6倍の支援を受けて注目を集めました。スマートフォンとはBluetooth経由でステレオスピーカーとして連携します。バッテリー搭載で駆動し、IPX7クラスの水滴や一時的な水没に耐えうる防水性能を持ちます。

Gadget 4

Aipoly Vision

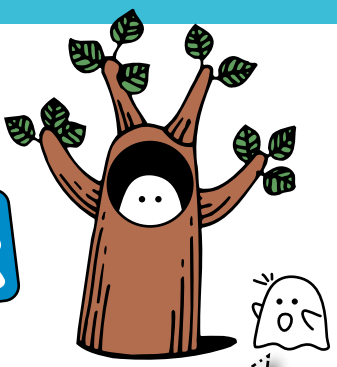
http://aipoly.com/

それが何かを教えてくれるアプリ

Aipoly Visionは、写っているものが何かを判別してくれるアプリです。文字で名前を示すとともに、音声で判別したものを読み上げてくれます。ハードウェアとしてのデジタルガジェットではありませんが、CES 2017で注目を集めたサービスの1つです。ときどき間違えた判別をする場合もあります。カメラを向けた物体の色を読みとる機能もあります。従来このような専用デバイスを作ろうとすると、とても複雑でおおごとになっていたと思われますが、スマートフォンアプリにすることで、手軽にいろいろな機能が使えるようになり、これらの傾向は益々増えていくことが予想されます。

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

結城浩の再発見の発想法 46



アドホック

アドホックとは

アドホック(ad hoc)とは、ラテン語で「これのための」という意味の単語です。一般には「行き当たりばったり」や「その場しのぎの」というニュアンスを持つこともあり、否定的な意味が強くなることもよくあります。しかし、技術的な文脈で使われるときには、「**特定の目的のための**」あるいは「**一時的な、その場の必要に応じた**」というニュアンスで使われ、否定的な意味はあまりありません。

アドホック・モード

「アドホック」という語が使われている例を挙げましょう。

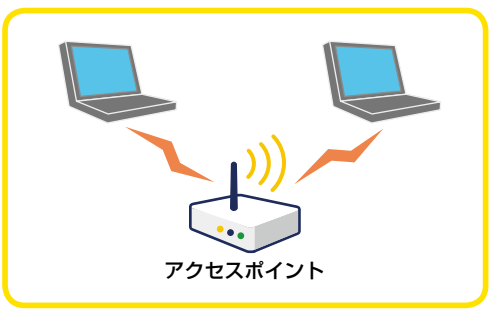
無線LANには「インフラストラクチャ・モード」と「アドホック・モード」という2つの通信モードがあります。

インフラストラクチャ・モード(**図1**)では、無線LANを使用する機器はアクセスポイントと呼ばれる無線機を介して通信します。これは、私たちが無線LANを使用するときに使う通常のモードです。

それに対してアドホック・モード(**図2**)では、無線LANを使用する機器は、アクセスポイントを介さず、直接通信を行います。アドホック・モードは、通信機器同士でファイル転送を行う場合などに使われます。また、アドホック・モードを使って携帯ゲーム機での複数人プレイを実現することもあります。

アドホック・モードでは、アクセスポイントを使いません。つまり、ネットワークを構築するときに前もってインフラ(設備)を必要としないと言えます。実際に通信する機器だけを持ち寄れば、それだけでネットワークが構築できますので、「必要に応じてそのつど構築できる」という意味で「アドホック」という単語を冠しているのでしょう。アドホック・モードには否定的な意味はありません。

▼図1　インフラストラクチャ・モード

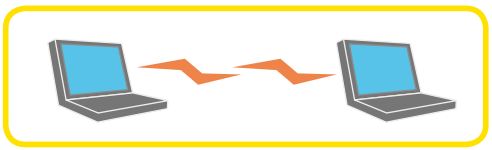


アドホック・レビュー

「アドホック」という語が使われている別の例です。

ソフトウェア開発において、ソースコードな

▼図2　アドホック・モード



どの成果物を他者に読んでもらい、誤りや改善ポイントの指摘などを受ける活動をレビューと言います。レビューは品質向上で大切な役割を担っています。

レビューにはいくつかの種類があります。たとえば「インスペクション」と呼ばれるレビューでは、参加する人の役割や、レビューの手順、指摘事項の記録などが定式化されています。インスペクションでは成果物の改善を組織的に行うことができ、その結果を組織全体で共有することができます。しかし、開催が大がかりになってしまいコストがかかる傾向もあります。

それに対して「アドホック・レビュー」というものもあります。アドホック・レビューは、開発者が同僚に「ちょっと、このコード見てくれない？」と声をかけて行うようなレビューです。開発者が「必要に応じてそのつど実施する」ので「アドホック」なのです。アドホック・レビューは前準備なしで気軽に実施でき、しかもピンポイントで必要が満たされるので、たいへん有用です。しかしその一方で、指摘事項が組織全体で共有されにくかったり、レビューの品質がばらついてしまったりという問題もあります。

アドホックの利害得失

無線LANのアドホック・モードであれ、アドホック・レビューであれ、「アドホック」であることはどんな意味を持つのでしょうか。

必要に応じて構築したり必要に応じて実施することのメリット、すなわちアドホックであることのメリットは明確です。それは「必要をすぐに満たすことができる」ことです。これは明らかですね。インフラを構築することなく、前準備を行うこともないのですから。

またアドホックであることは、「必要なことだけ行うので無駄がない」とも言えます。実際、無線LANのアドホック・モードでは電波の使用効率が良くなります。アドホック・レビューでは少数の人間だけが関わり、ピンポイントでレビューが実施されるので、その点では無駄がないと言えます。

一方、アドホックであることのデメリットはこの裏返しになります。長い期間同じ活動が続くならば、インフラ構築や前準備のためにコストをかけたほうがいいでしょう。また、アドホックな活動で部分的な必要だけを満たしていたら、その点での最適化はなされるかもしれませんが、全体では大きな無駄が発生する可能性もあります。

すなわち、解決したい問題や実施したい活動が時間的・空間的に**どれだけ局所性を持っているか**の判断が大切になるということです。

日常生活とアドホック

私たちの日常生活に溢れているたくさんの問題をアドホックに解決すべきか、考えましょう。

簡単な例を出しましょう。あなたが会社でAさんに書類を渡したところ、Aさんはその書類を紛失してしまいました。当然ながらあなたはAさんに「書類を紛失しないでくれ」と苦情を言うでしょう。これはいわば、アドホックな解決をはかろうとしているわけです。

もしもこのような書類紛失のトラブルが、あなたとAさんとのあいだだけで発生しているなら（局所性）、そして、一度で解決するなら（別の局所性）、アドホックな解決で十分です。わざわざ、組織全体の課題とする必要はありません。

しかしもしも、会社のあちこちで同様のトラブルが起きていたり、あるいはAさんとのあいだで頻繁にトラブルが起きるようなら、アドホックな解決ではなく、書類入れを整備するなどの組織的な解決が必要になるでしょう。

問題の局所性を理解することは、どのような解決方法をとるかに大きな影響を与えるのです。

あなたの周りを見回して、身近な問題を調べてみましょう。繰り返し起きる問題をアドホックに解決しようとしていないでしょうか。あるいは逆に、スピーディに解決すべき局所的な問題に、大がかりで時間がかかる解決法を取ろうとしていないでしょうか。

ぜひ、考えてみてください。SD

増井ラボノート

コロンブス日和

第17回　Scrapbox（2）

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

Scrapboxの記法

先月号では、シンプルで強力なWikiシステム「Scrapbox」の概要を紹介しましたが、今回はScrapboxの機能についてより詳しく紹介したいと思います。

Scrapboxでは、簡潔な**マークアップ記法**により、他のページへのリンクを張ったり、外部Webページへのリンクを張ったり、画像や動画を表示したり、文字修飾を行ったりすることができます。現在広く利用されているHTMLは「HyperText Markup Language」の略であり、テキストの装飾やリンクなどを指定するためにや<a>のような各種のマークアップ（タグ）が利用されています。

HTMLはあらゆるブラウザで利用できるので便利ですが、仕様が巨大で覚えるのが大変ですし、単に文字を太くするだけのために「」「」のような記述をしなければならないのは面倒です。このため最近は「Markdown」のような簡易マークアップ言語が人気があるようです。MarkdownはHTMLに比べると記述が簡潔ですし（**リスト1**）、GitHubなど多くのサービスで利用できるので便利です。

一方、Markdownで画像やリンクを表現するための言語仕様は、かなり複雑なため覚えるのが困難です。たとえば、**リスト1**の［**タイトル**］（URL）のような記法は私にはなかなか覚えられません。実際にマークアップはHTMLを併用して使われていることもあるようなので、Markdownの存在価値を私は疑問に感じています。

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

Scrapboxのマークアップ

Scrapboxでは極力簡単に目的を達成できるような簡単で強力なマークアップ記法を採用しています。基本的には、

・あらゆるマークアップは角カッコ・（［ ... ］）で表現する
・記述の制約をできるだけ緩くする

という方針に基づいています。

内部リンク

テキスト内の文字列を［と］で囲むと、囲んだ部分がその名前のページへのリンクになります。たとえばScrapboxの文中で［**増井**］と記述すると「増井」というページへのリンクになります。

外部リンク

外部のWebページにリンクを張りたいときは、

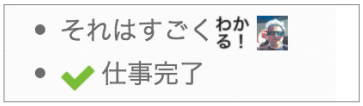
▼**リスト1**　Markdownの記述例

```
# Gyaim

* [MacRuby](http://www.macruby.org/)で作ったMac用のIMEです。
* 数百行のRubyで実用レベルのIMEを作れることを示すものです
* 単純な変換しかできませんがそれなりに使えます
* 見栄えはInterfaceBuilderで簡単に変更できます
* 変換アルゴリズムやIMEの動作はRubyで簡単に変更できます
* 自前のIMEをいろいろ作りましょう！
* 変換手法はSlimeと同じです
* 予測機能はありません

## インストール
```

▼図1 ［増井.icon］の表示結果

▼図2 ［増井.icon*10］の表示結果

▼図3 Scrapboxページへの適用結果



［URL タイトル］のように記述します。これはタイトルのようなHTMLに変換されます。［タイトル URL］のようにURLとタイトルを逆順に書いても同じ結果になるので、順番について気にする必要はありません。

画像や動画の埋め込み

テキスト中に［画像URL］と書くと、その場所に画像が表示されます。また［画像URL リンクURL］と書くとURLへのリンクを持つ画像が表示されます。この場合も、URLの順番は逆でも大丈夫です。画像URLの代わりにYouTubeやVimeoのURLを書くとページに動画が埋め込まれます。

文字修飾

［* ... ］のような記法で太字や斜体を表現できます。［* abc］と書くとabcのように太字になり、［/ abc］と書くと^_`のように斜体になり、［/* abc］と書くと*abc*のように太字の斜体になります。

アイコン記法

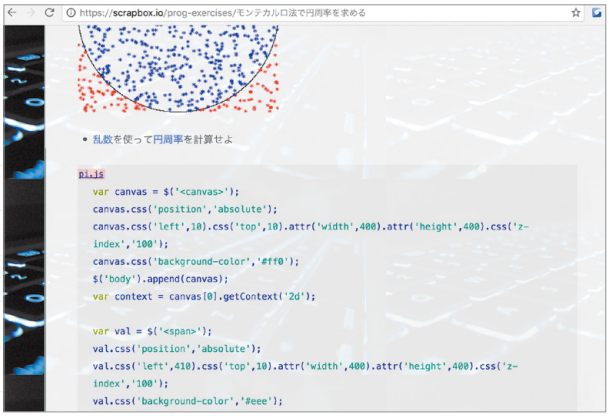
［増井.icon］と書くと、「増井」というページの代表画像が、アイコンのように行の中に表示されるようになっています（図1）。

アイコンをたくさん並べることもできます。たとえば［増井.icon*10］と記述すると、アイコンが10個並びます（図2）。

このような「アイコン記法」を利用すると、Scrapboxページを絵文字のように利用できて便利です（図3）。

Wikiページ上で複数の人が編集を行った場合、誰がどの部分を書いたのかわからなくなって困ることがありますが、このようなアイコンを常に書く習慣をつけておけば、そういう問題が起きにくくなります。

▼図4 コード記法



コード記法

ちょっとしたプログラムをWeb上で公開したいとき、Gistのようなサービスにプログラムの断片を貼ることがありますが、Scrapboxページの中にコードを記述できます（図4）。

コード記法したデータは「https://scrapbox.io/api/code/prog-exercises/モンテカルロ法で円周率を求める/pi.js」のようなURLでアクセスすることができます。

その他の便利な機能

アウトライン編集

Scrapboxのページでは、行頭に空白文字を入れてインデントすることによって行の階層構造を指定できますが、階層構造を利用してアウトライン編集を行うことができます。

コロンブス日和

行の古さの表示

Wikiページを長年利用していると、新しく記述を追加しても気付きにくいことがありますし、どの部分が新しいのかわからなくて困ることがあります。Scrapboxでは、行の左端に表示される灰色の矩形の大きさで行の古さがわかるようになっているので、記述の古さがすぐわかります。**図5**の例では、「藤沢店」あたりの記述が他の行よりも新しいことがわかります。

プレゼンテーションモード

スライドを使ったプレゼンテーションを行うときはPowerPointやKeynoteのような専用ソフトウェアが利用されるのが普通です。これらのソフトウェアは高機能で便利なのですが、ブラウザと連動して利用することはできないので困ることもあります。

Scrapboxには「プレゼンテーションモード」という機能があり、ページを使ってスライド表示できます(**図6**)。私は授業の資料をすべてScrapboxで作成しています。

入力補完

Scrapboxでたくさんのページを作っていると、ページ名で混乱することがあります。ページが存在するかどうかは検索すればわかるのですが、ハッシュタグやカッコの後でページ名を入力しようとするとき、すでに存在するページ名に対して曖昧検索を行って候補を表示するようになっているので、似て異なるページを作ってしまう可能性を減らしています。

ソート機能

Scrapboxのページは、編集時刻の順に表示したいことがあるでしょうし、アクセス時刻順に表示したいこともあるでしょう。Scrapboxではさまざまな指標でソートできるようになっています。

ピンどめ機能

どういう方法でソートした場合でも、古いページはリストの下の方に移動してしまうことになります。重要なページや最初に見るべきページについては常にトップに固定されるようにするための「ピン機能」を用意しています。ピンどめされたページは常にプロジェクトの先頭に表示されます。

各種の設定

ブログなどに利用するScrapboxのページは誰に対しても公開したいでしょうし、個人的なメモは誰に対しても秘密にしておきたいでしょう。Scrapboxでは、ページを公開するかどうかを設定可能であり、またページを編集可能なユーザを設定できるようになっているので、内容に応じて細かくアクセス制御を行うことができます。また、ページの見栄えも細かい設定が可能になっています。

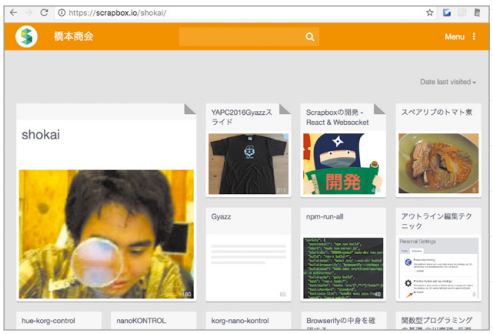
Scrapboxの実装

Scrapboxの編集機能や同時編集機能は、Atom、Gitの実装を参考にしてReactなどで

▼**図5** 行の古さを表示

▼**図6** テキストとプレゼンテーションモードでの表示

▼図7 「橋本商会」



すべて自前で実装しています。実装の詳細は、橋本翔氏のページ[注2]で解説されています（図7）。

Scrapboxの利用例

Scrapboxの利用例は先月号でもいくつか紹介しましたが、再度紹介しておきます。

個人メモとしての利用

個人用のちょっとしたメモは、パソコンのローカルファイルとして保存している人が多いと思いますが、ローカルにファイルを保存すると別のマシンやスマホから見ることができませんから、ちょっとした情報でもScrapboxのようなクラウド上に保存しておくのが良いと思います。私は電話帳も予定表もTODOリストもすべてScrapbox上で管理しているので、スマホでもパソコンでも同じように情報を管理できています。昔はデータを同期したり、共有ファイルとすることが多かったようですが、完全に一本化しておくのが一番便利だと感じています。

ブログとしての利用

各種のブログサービスが利用されていますが、Scrapboxを公開設定すれば、簡単なブログのように利用することもできます。

日本語入力辞書

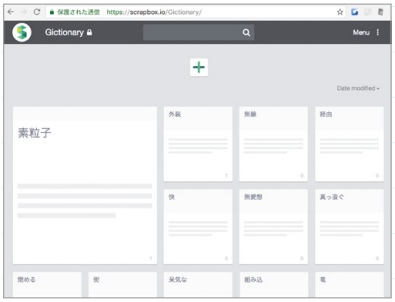
昨年5月号で「Gyaim」という日本語入力システムを紹介しましたが、私はGyaimで使われる辞書の管理にScrapboxを使っています（図8、図9）。

辞書は単純な形式ですが、間違いや追加があればWiki上で簡単に修正できるのが便利です。多くのユーザで共有して利用すれば、質の高い辞書に育てていくことも可能でしょう。

◆ ◆ ◆

2回にわたって汎用で使いやすいWiki「Scrapbox」を解説しました。現在のScrapboxはかなり複雑なシステムになってしまったので実装はコロンブス的とは言いがたいですが、ユーザから見ればその利用方法は、コロンブスの卵的だと感じられると思います。情報を階層的に関連するのをやめ、あらゆる情報をクラウドに置くことによって情報整理に関する精神の重荷はかなり軽減されますし、将来の情報共有にもとても有用だと思います。ぜひScrapboxをいろんな用途に活用ください。SD

▼図8 語のリスト



▼図9 「素粒子」のエントリ

注2）「橋本商会」橋本翔氏のブログ https://scrapbox.io/shokai/Scrapboxの開発%20-%20React%20&%20Websocketで作るリアルタイムWiki

第13回 広島から香川まで、学生さんと交流

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

はじめに

前回OSC福岡関係の話だけで2ページを使い切ってしまったので、今回は2016年11月27日に開催されたOSC広島を中心にレポートします。OSC福岡の翌週に開催と、かなりスケジュール的にも詰め詰めな開催でしたが、学生さんがたくさん参加してくれたことでたいへん盛り上がりました。

呉旧鎮守府を巡る

広島に入る前に、呉（くれ）まで足を伸ばしてみました。呉では、「大和ミュージアム」や「てつのくじら館（海上自衛隊呉史料館）」、「旧日本海軍の呉鎮守府関係の史跡」などを見ることができます。また、船に乗って海上自衛隊呉基地の港に係留されている海上自衛隊の護衛艦を見るツアーも出ているので、時間を確認して乗船してみるといいでしょう。

ランチは、呉海自カレーがお勧めです。呉基地所属の艦艇のレシピで作ったカレーが食べられます。お店によって出てくるカレーの種類が違うので、事前に調べておいた方がいいかもしれません。

また、「海軍さんの麦酒」という地ビールがあったので飲みたかったのですが、直営店は平日昼間はお休み。しかたがないので駅の売店で瓶入りを購入して、広島行きの電車の中で飲もうと思ったら、普通に混んでいて飲めませんでした。こんなことだったら、駅で電車待ちの間に飲めばよかった！

OSC広島前日企画 学生LT大会で大盛り上がり

さて、OSC広島には、山陰地方の島根、鳥取から学生さんが参加してくれたほか、四国からも香川大学の学生さんが30名以上、プログラミングサークルの旅行として参加してくれました。

せっかくたくさん学生さんが参加してくれるのに何もしないのはもったいないので、前日企画として学生LT大会を開催しました。発表者が全部で40人近くとなったので、発表時間は最短2分半、最長5分で、たっぷり2時間以上の大LT大会を開催しました。「ほー」と思うような内容もあり、たいへん盛り上がりました。

さらに終了後は前夜祭です。ここからは中国地方DB勉強会のみなさんも合流し、前夜祭として50人近くで盛り上がりました（**写真1**）。学校間での交流も深まったようで、開始前からよい雰囲気となりました。

OSC広島本番（**写真2**）ももちろん大盛り上がりでしたが、誌面の都合で割愛します（書くことが多過ぎます！）。

▼写真1　OSC広島前夜祭の様子。中央のピースサインは香川大学の花川直己さん

OSC広島が終わったあとは香川大学に

OSC広島終了後、今度は筆者が香川大学にお邪魔しました。OSC広島への参加のお礼として、香川大学で特別講義を行わせていただきました。

広島から岡山まで新幹線、マリンライナーに乗り換えて瀬戸大橋を渡って高松まで。うどん県に入ったわけですから、さっそく高松駅前の「めりけん屋」でうどんを食べます

▼写真2　OSC広島終了後の記念撮影。最後のライトニングトークまで大盛り上がりでした

▼写真3　めりけん屋の肉ぶっかけうどん。高松駅前なので、気軽に本場の讃岐うどんが楽しめる人気店です

（**写真3**）。さらにタクシーで香川大学に移動したら、学生さんが近所のうどん屋に案内してくれてさらにうどんです。おそるべし、うどん県。

夕方から、2時間ほど学生さん向けに講義を行いました。『オープンソースコミュニティによる技術交流の重要性』というタイトルで、OSCを10年以上続けてきたことによりわかったコミュニティのあり方や、技術を身につけていくうえでのコミュニティとの付き合い方などをお話しました。当初は1時間ぐらい話して、あとは質疑応答と考えていましたが、思った以上に熱が入ってしまって時間一杯まで話してしまいました。講義終了後は学生のみなさんと中華料理屋さんで食事会。この場でいろいろと質問してもらえたので結果オーライですかね。

香川に来たら金比羅さんにお参りを

香川には何度も来ていますが、前回来たときにいただいたお札をお返しに、ということで金比羅さんにお参りしてきました。金比羅さんは高松から南西に1時間ほど行ったところにあります。高松から琴平電鉄というローカル線が出ているので、ことこと揺られて向かいます。この路線、ものすごく揺れるので、乗り物酔いする人は注意してください。終点の琴電琴平駅から参道を抜けていくと、金比羅さんにお参りするための階段が現れます。登り始める前に、朝ご飯代わりに讃岐うどんをいただいておきます（2日間で3食目）。

大門まで365段の間、階段の両脇はお土産屋さんなどが立ち並んでいるので、なんとなく眺めながら登っていきましょう。実はこのエリアの階段はあまり休めるところがなく、傾斜もきつくてたいへんだったりします。大門をくぐると境内に入り、時折階段が出てくる程度に変わります。きつい傾斜の階段を休みなく延々と登らされることはなくなります。

785段を登りきると御本宮に到着です。お参りしたあと、古いお札をお返しし、新しい「会社繁栄」のお札をいただきます（**写真4**）。このお札もまた、次回お返しに来ないといけないですね。展望台からは香川の景色を遠くまで見ることができます。

▶写真4　金比羅さんの御札をいただきました。頑張って登ったかいがありました。「金」の文字が独特です

お役目を終えたので下山し、門前にある酒造所「西野金陵」さんに立ち寄ります（**写真5**）。いろいろと試飲させてもらい、最終的に美味しかった特別純米酒をお土産に購入。

その後、琴平温泉の日帰り入浴で汗を流し、JRに乗り換えて丸亀に向かいます。丸亀城に行くことを勧められたのですが、こちらも急坂を登らないといけない山城です。本丸から見える瀬戸内海の景色は格別でした。SD

▶写真5　西野金陵さん。以前は金比羅さんの門前で日本酒造りをしていたそうですが、今は元酒蔵が博物館と試飲販売になっています

ツボイのなんでもネットにつなげちまえ道場

第21回

「さくらのIoT Platform β」の通信モジュールを使ってみる（前編）

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

さくらのIoT Platform β

昨年の10月のことですが、さくらインターネットが「さくらのIoT Platform β」の発表を行いました。このサービスの特徴は、クラウドサービスだけではなく、セルラー通信と、そこに接続するためにIoTのエンドノードに組み込む通信モジュールまで**図1**のように一気通貫でサービス提供をしていることです。現在は、セルラー（LTE）でクラウドに接続をする「単体方式」と呼ばれるモジュールが提供されていますが、LoRa変調の通信モジュールをエンドノードに組み込み、ゲートウェイを通じて接続する「ゲートウェイ方式」の開発も行っているそうです。

さくらのIoT Platform βで提供されている「さくらの通信モジュール（LTE）β版」には、SIMカードが取り付けられています。このSIMカードの契約を使って、ソフトバンクモバイルのセルラー網を通じ、通信モジュールはさくらのIoT Platform βのサーバと通信をします。さくらのIoT Platform βからは、**図2**の連携サービスやさくらインターネットの既存サービスに接続できます。

この単体方式の通信モジュールは先述のようにLTE網を使ってネットワークに接続します。LTEの中でも、LTE カテゴリ1という規格の通信モジュールを使っている点が筆者としては印象的でした。LTE カテゴリ1は、IoT機器向けで低消費電力が特徴だとされているからです。

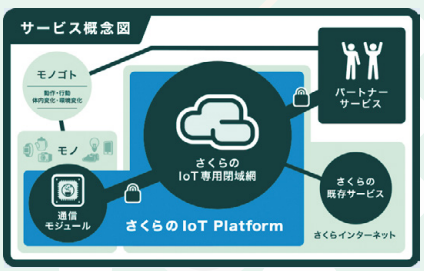
LTE カテゴリ1とは

ところで、LTE カテゴリ1とは、何のことでしょう。LTEはみなさんご存じだと思いますが、携帯電話の通信規格です。3GPP[注1]という標準化団体によって策定されています（**表1**）。

3GPPは、「リリース」という単位で標準化を行っていて、リリース8からLTEの規格です。リリース8は、ダウンリンクが最大150Mbpsのモバイルブロードバンド通信ができる規格で、このダウンリンクが最大150Mbpsの通信規格は、「端末カテゴリ4」として定義されています。リリース8では、ダウンリンクが最大10Mbpsの「端末カテゴリ1」も定義されています。「カテゴリ」は、「Cat.」と略して表記されることがよくあります。

なぜ高速なカテゴリ4と、スピードが出ない端末カテゴリ1が同時にリリース8として定義されたのでしょうか。これまで、セ

▼図1　サービス概念図



※さくらインターネットのWebサイト（https://iot.sakura.ad.jp/）より引用

▼図2　連携できるサービス

注1）http://www.3gpp.org

▼表1　LTEの規格とモデムの複雑さ（ノキアの資料などから作成）

	リリース8		リリース12	リリース13	
	Cat. 4	Cat. 1	Cat. 0	Cat. M1	NB-IoT
最大通信速度（下り）	150Mbps	10Mbps	1Mbps	1Mbps	26kbps
最大通信速度（上り）	50Mbps	5Mbps	1Mbps	1Mbps	62kbps
アンテナ数	2	2	1	1	1
モデムの複雑さ	100%	80%	40%	20%	15%以下

ルラー通信には高速な通信が求められてきましたが、IoT向けでは大容量のデータ転送よりも、低消費電力であることや端末（エンドノード）の単価を安くすることが求められるからでしょう。カテゴリ4とカテゴリ1を比較すると、カテゴリ4に対してカテゴリ1の複雑さは80%程度ということです。モデムの複雑さが低い方が、モデムの価格は低くなります。しかし、この複雑さが80%というのは十分だとは言えません。

より省電力なセルラー

そこで、リリース12ではカテゴリ0が用意されました。カテゴリ0は、送受信ともに通信速度が最大1Mbpsに抑えられた代わりに、アンテナ数が1つになり、また半二重通信をサポートしました。これによりモデムの複雑さは大きく減っています。さらに、リリース13では、カテゴリM1やNB-IoTといったカテゴリが追加され、とりわけ、eDRXという技術が規定されたことが消費電力に大きく影響を与えそうです。eDRX（extended DRX）は、従来（リリース8）から規定されていたDRX（Discontinuous Reception）という技術が拡張されたものです。DRXはその名のとおり電波の受信を間欠で行うもので、eDRXではこの間欠の時間を大幅に伸ばすことができるものです。この間隔は、DRXでは最大2.56秒であったところを、NB-IoTでは最大174分と、実に4,000倍近く伸びています。

なぜ受信しない時間を延ばすことが省電力につながるかというと、受信しない間は無線機をスリープ状態にできるためです。無線機の消費電力はIoTのエンドノード全体からすると相当に大きいため、スリープ状態にすることで消費する電力を大幅に減らすことができます。この無線機をスリープ状態にする時間が大幅に延びるということは、バッテリーでIoTのエンドノードを運用できる期間を大幅に延ばすことにつながります。

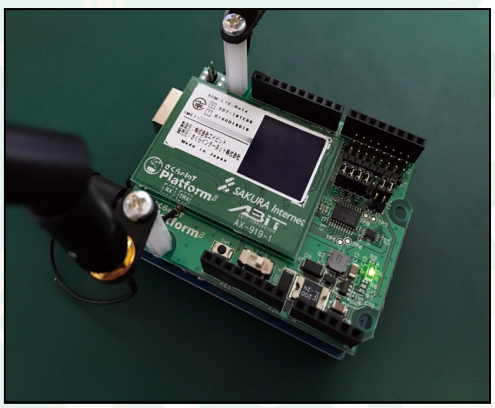
このeDRXですが、セルラーの基地局側もリリース13に対応することが必要です。また、eDRXはけっしてカテゴリM1やNB-IoTでのみサポートされる技術というわけではなく、カテゴリ1でも利用できそうです。昨年の後半から国内携帯電話オペレーター各社と、Altair Semiconductorらがテストを行っており、今年中には各社の基地局が対応していく模様です。

さくらの通信モジュール（LTE）β版

このように、本格的に低消費電力なLTE接続が使えるようになるには、まだ時間を要しそうです。とはいえ、「さくらの通信モジュール（LTE）β版」は、現状最も手軽に手に入るLTEカテゴリ1の通信モジュールです（**写真1**）。とりあえず使ってみましょう。今回、実験に使った基板は、さくらのIoT Platform βのページ[注2]

注2）　https://iot.sakura.ad.jp

▼写真1　さくらの通信モジュール（LTE）β版



にある販売の案内から、販売ページにアクセスして入手しました。入手した基板は、さくらの通信モジュール(LTE)-β版-と、Arduinoシールドボード-β版-の2つです(**写真2**)。

本連載ではずっとmbedを使っていますので、mbedでやりたいところです。しかし、筆者が少し試しただけではmbed用のサンプルプログラムでうまく動かすことができませんでした。そこで、今回はとりあえず「さくらの通信モジュール(β版)利用マニュアル」に記されていたArduino Unoを使って試用してみました。

試用の手順は簡単です。まず、コントロールパネルにアクセスをして、手元に届いたモジュールを登録します。モジュールの登録に必要な情報は、モジュールに貼られた黒い目隠しシールの下に印字されていました。コントロールパネルにアクセスするには、さくらインターネットの会員IDとパスワードが必要ですので、会員IDを持っていなければ登録も必要です。

モジュールの登録が済んだら、連携サービスとしてWebSocketを登録しましょう。あとはArduinoでサンプルプログラムを動かせば、このWebSocketのページでデータの受信を確認できます。

次はArduino側の手順です。まず、Arduino IDEを手元のパソコンに用意します。ダウンロードは、arduino.cc[注3]から行いましょう。Arduino IDEをインストールして起動したら、「スケッチ」→「ライブラリのインクルード」の「ライブラリを管理」をクリックします。そこで検索欄に「sakura」とタイプすると、通信モジュールを使うためのライブラリをインストールできます。

次に、Arduinoに書き込むプログラムを用意しましょう(Arduinoでは、プログラムは「スケッチ」と呼ばれます)。サンプルプログラムは、さくらインターネットのGitHubにあるリポジトリ[注4]で公開されています。今回は、Standard[注5]というサンプルプログラムを使いました。Arduino IDEでコンパイルを行うためには、このプログラムをコピーし、Arduino IDEを起動したときに出てくるウィンドウにペーストをします。

ここで、入手したモジュールとシールドボードを組み立て、Arduino Unoに取り付けましょう。シールドボードを取り付けたら、通電する前にいくつかの設定をシールドボードの上で行わなければなりません。シールドボードとArduinoは、I²CやSPI、UARTなど複数の方法で接続できます。この通信を行うピンの設定は、このシールドボード上で行うのです。詳細はシールドボードの取扱説明書にありますが、筆者は**写真3**のように設定を行いました。SPIの4つ

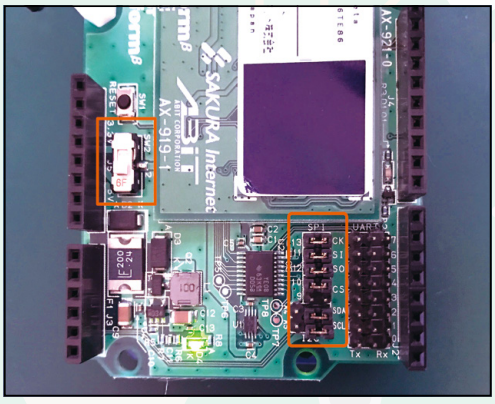
注3) https://www.arduino.cc/en/Main/Software

注4) https://github.com/sakura-internet/SakuraIOArduino

注5) https://github.com/sakura-internet/SakuraIOArduino/blob/master/examples/Standard/Standard.ino

▼写真2　通信モジュールとシールドボード

▼写真3　シールドボードの設定例



にジャンパピンを取り付け、I²CのSDAとSCLのほうにもジャンパピンを取り付けました。また、白いI/O電圧切り替えスイッチを、Arduino Unoは5Vですので、5Vのほうに設定します。これで、ハードウェアの設定は完了です。

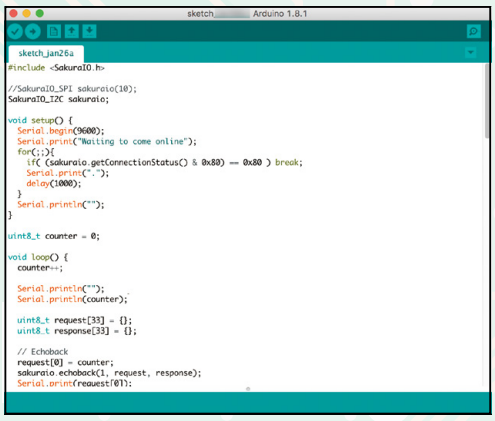
さて、ライブラリをインストールし、サンプルプログラムのペーストを終えたところで、Arduino Unoへのプログラムの書き込みを行いましょう。まず、Arduino UnoをパソコンにUSBケーブルで接続し[注6]、Arduino IDEの「ツール」→「シリアルポート」で、Arduino Unoが接続されたシリアルポートを選択します。プログラムを書き込むには、**図3**のArduino IDEのウィンドウにある、→のボタンをクリックします。すると、プログラムのコンパイルが始まり、コンパイル後にArduino Unoへの転送が開始されます。

転送を終えたら、先ほどのウィンドウの右上にある虫眼鏡のアイコンをクリックします。すると、Arduino Unoとのシリアルモニタ（**図4**）が立ち上がり、プログラムのデバッグメッセージが表示されます。「Waiting to come online」と表示されたあと、「.」が順に表示されます。初めての起動では、先に進むまでとても長い時間（筆者の手元では5分程度）を要しました。この間、とても不安な気持ちになりましたが、気長に待ちましょう。「故障かなと思ったら」というドキュメントにも記載がありましたが、初回起動時はファームウェアのアップデートが実行されていて、時間がかかるようです。

シリアルモニタでデータを送信していることが確認できたら、コントロールパネルで登録した連携サービスのWebSocketのページを参照してみましょう。少しタイムラグがありますが、モジュールから送信されたデータが、さくらのIoT Platform βで受信できていることが確認できます。

注6）このモジュールがどのくらいの電力を要求するのかわからないので少し不安でしたが、筆者の環境では、Arduino Unoに接続したUSBケーブルからの電力だけで動かすことができました。

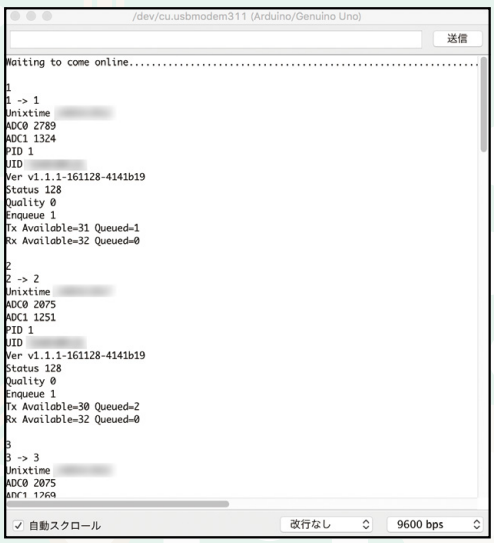
▼図3　Arduino IDEのウィンドウ



まとめ

いったん電源を切って、再度通電したところ、1分程度で起動と接続を終えるようになりました。取扱説明書を読むと、現在のところ、先ほど紹介したeDRXへの対応を始め、省電力制御機能はまだサポートされていないようです。接続に1分程度かかる現状では、IoTのエンドノードの開発からは遠いところにあるというのが率直な感想です。現状で、どの程度の電力が要求されているのか気になりますし、mbedでもぜひ動かしてみたいと思いますので、それらは次回やってみます。SD

▼図4　シリアルモニタ



読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」にアクセスし、アンケートにご協力ください（アンケートに回答するにはgihyo.jpへのお名前と住所のアカウント登録が必要となります）。ご希望のプレゼント番号を記入いただいた方の中から抽選でプレゼントを差し上げます。締め切りは**2017年3月16日**です。プレゼントの発送まで日数がかかる場合がありますが、ご容赦ください。

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

01

ディレクターズ 10周年記念レディースバッグ 1名

ホスティング事業のディレクターズが、2017年の創業10周年を記念して、オリジナルの女性向けPCバッグを作りました。本皮を使用した高級感のあるつくりで、中がふかふかの素材で作られたポケットには13インチのPCがすっぽり入ります。ディレクターズのロゴマークである矢印の形のキーホルダーには、USBメモリ（8GB）が付いています。今月は『ネイビー』をプレゼント。

提供元 ディレクターズ　http://www.directorz.ad.jp

02

Repro Tシャツ＆ステッカー

モバイルアプリ向けのアナリティクス・マーケティングツール「Repro（リプロ）」を提供するRepro㈱のノベルティTシャツとステッカーです。Tシャツのサイズはしサイズで、表面にはReproのロゴ、裏面には「すべて熟知」の文字がプリントされています。

提供元 Repro
https://repro.io

Repro 1名

03

pixivコミック 出張版2016-2017selection

ピクシブ㈱が運営するコミックサイト「pixivコミック」にて連載されている漫画をまとめた小冊子です。「ヲタクに恋は難しい」「腐女子のつづ井さん」など収録。2016年12月29～30日のコミックマーケット91にて配布されました。

提供元 ピクシブ
http://www.pixiv.net

2名

04

アルゴリズムクイックリファレンス 第2版

George T. Heineman ほか 著

40を超える主要アルゴリズムを、C、C++、Javaでの実装例を示しながら解説。第2版では新たに、マルチスレッドクイックソート、AVL平衡二分木といったアルゴリズムを、Pythonを使って説明します。

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

2名

05

あたらしい人工知能の教科書

多田 智史 著

人工知能関連の開発に携わるエンジニアを対象に、今後人工知能のコアとなる理論と技術を図解で解説した1冊です。機械学習、深層学習、IoTやビッグデータとの連携といった実務的な領域もカバー。

提供元 翔泳社
http://www.shoeisha.co.jp

2名

06

まつもとゆきひろ 言語のしくみ

まつもとゆきひろ 著

まつもとゆきひろ氏が新言語Streemを開発する取り組みを追った、月刊誌「日経Linux」の連載をまとめた1冊です。実際にRubyを作った著者だからこそ書ける、言語デザインの実際と考え方を学べます。

提供元 日経BP
http://corporate.nikkeibp.co.jp

2名

07

その「エンジニア採用」が不幸を生む

正道寺 雅信 著

エンジニア採用部門のコンサルタントとして数々の採用に携わった著者が、エンジニア採用がうまくいかない原因、エンジニアに活躍してもらうためのしくみづくりを教える1冊です。

提供元 技術評論社
http://gihyo.jp

2名

第1特集

マーケティング&サービス向上に役立つ

ログ&データ分析基盤入門

いまやWebサービス業界では、ログやデータの分析を行い、自社のビジネスに活かすのは当たり前……そんな状況になっています。けれど、
「マーケティングに活かすには、どんなふうにデータを分析すればいいの？」
「効果的なデータ分析を行うには、どんなシステム基盤が必要なの？」
「各技術や製品は何が違うの？　どんな場面で役に立つの？」
「全部、自前で構築するしかないの？　便利なクラウドサービスはないの？」
そんな疑問をお持ちの方もいるのではないでしょうか？　本特集では、その疑問にお答えします。自社のログ&データ分析基盤の構築にお役立てください。

第1章

データ分析基盤に求められる要件とは

Author 高橋 達（たかはし とおる） トレジャーデータ㈱

データ分析基盤の技術解説に入る前に、まずはマーケタの立場に立って、マーケティングのためのデータ分析とは何なのかを考えてみます。「どんな情報をもとにどんな分析をするのか」を整理し、データ分析に必要な技術要素やツールを見ていきます。トレンドの大枠をつかんでデータ分析基盤に求められる要件を理解しましょう。

広がるデータ分析基盤の世界

Amazon Redshift、Google BigQuery、Treasure Data Serviceなどデータ分析基盤を作るためのクラウドサービスが普及し、多くの企業でデータ分析基盤が必須のシステムとなっています。これまでは、「ユーザの行動を把握することでサービスの問題点を発見し改善する」ということが、データ分析基盤を構築する一般的な目的でした。この既存ユーザに対する取り組みを行うために、データ分析基盤はWebサーバのログやデータベースにある顧客情報といった自社で開発したシステムのデータ（1st party data）を集めることにフォーカスされていました。

一方で、デジタルマーケティングではデータ分析を活用して大きく2つの取り組みが行われてきました。

(1) 既存ユーザに対して、ヘビーユーザと非アクティブユーザといった分類を行い、ロイヤリティ化[注1]やリテンション[注2]を促す施策を行う

(2) 自社サービスに興味を持つ可能性がある非ユーザに対して、エンゲージメント[注3]を高める施策を行う

こうした目的のもとに、MarketoやSalesforce Marketing Cloudといったマーケティングオートメーションシステムを利用し、性別や年齢などの属性情報をもとにユーザをセグメント化（グループ分け）し、メール配信や広告配信といった施策に活かしていました。

しかし、マーケティングオートメーションシステムでは一般的に自社サービスの生ログを長期間保持するといったことが難しいという課題がありました。さらに、Google AnalyticsやAdobe AnalyticsといったWeb解析サービスや、Google AdsenseやFacebook Adsといった広告配信サービスなど、多種多様な外部サービスにある1st party dataを組み合わせることが非常に難しいという課題がありました。

また、自社サービスの顧客になり得る非ユーザに対して施策を実施するには、自社サービス以外のデータも集める必要があります。そのためには、さまざまなサイトに訪れたユーザのオーディエンスデータ（3rd party data）を持つPublic Data Management Platform（以下、Public DMP）と連携する必要もあります。

注1）ロイヤリティ化とは、既存ユーザに自社サービスのファンになってもらうことで、自発的にほかの人にサービスを勧めるといった効果を狙うこと。

注2）リテンションとは、既存ユーザがサービス退会することを防ぐ試みのこと。

注3）エンゲージメントとは、ユーザとのつながりを深めること。たとえば、テレビコマーシャルでブランドイメージを確立することで、ユーザに認知してもらえるようにすること。

さて、従来のデータ分析基盤を振り返ってみると、自社サービスの1st party dataを集めるためのしくみが簡単に構築できるという点については、よく述べられています。これに加えて、外部サービスに蓄えられた1st party dataやPublic DMPからの3rd party dataを収集するしくみを用意し、さらに、分析したデータをマーケティングオートメーションシステムなどのメール配信や広告配信に書き出すしくみを付け加えることにより、統合されたデータ分析基盤へと拡張させることができます（**図1**）。この新たなデータ分析基盤により、データ分析の結果をマーケティングの施策にスムーズに落とし込むことができ、データ自体の価値を高めることができるようになります。

そして、このようなデータ分析基盤のコンセプトとして、Customer Data Platform（以下、CDP）やPrivate Data Management Platform（以下、プライベートDMP）が現在、注目されています。

本記事ではデジタルマーケティングにおけるデータ分析基盤の役割を持つCDPやプライベートDMPについて紹介します。そして、それらのしくみを構築するうえでデータ分析基盤と同様の課題である収集・分析・自動化について、各々の課題を解決するためのオープンソースソフトウェアやサービスについて紹介を行います。

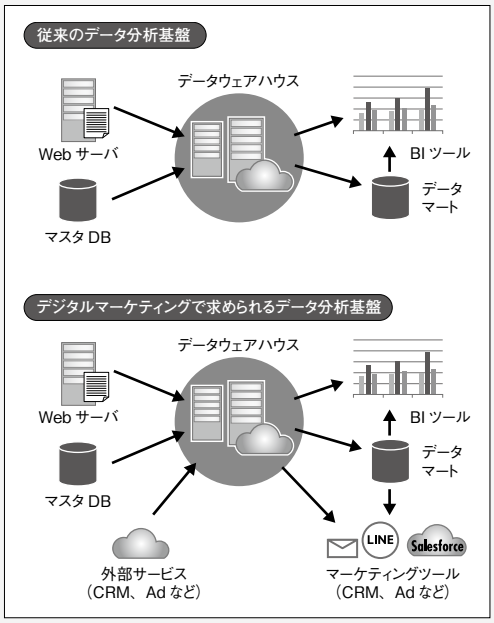
CDPとプライベートDMP

CDPとDMPとは

CDPは、David Raab氏によって2013年に新たに提唱された比較的新しいサービスコンセプトです[注4]。CDPとは、CRM（Customer Relationship Management）ツールや自社サービスのマスタデータに保存された顧客情報にあるようなユーザ名やユーザID、メールアドレス、電話番号といったユーザを識別するための1st party dataを用いて、未知のユーザおよび既知のユーザの両方に対して複数のコミュニケーションチャネルを利用してマーケティング施策を打つことを目的としたしくみです。たとえば、実店舗とECサイトを持つ小売業者のCDPがあるとします。スマホアプリやECサイトのログを集め、共通のユーザIDで統合しておきます。ECサイトで閲覧したけど買わなかった商品のクーポンを、スマホの位置情報を利用して、ユーザが実店舗近くに来たときにLINEでメッセージを送ることで購買を促すことができます。

一方で、プライベートDMPは、CDPと同様に自社サービスのログなどのデータに加え、自社サイト以外のCookie情報の3rd party dataを用いてユーザのセグメント化を行い、広告配信などのマーケティング施策を行うことをおもな目的とします。DMPでは、共通のIDとしておもにCookie IDが使われます。また、Cookie IDを用いて各々のユーザを特定するのではなく、同じ傾向を持つユーザを包括するセグメントを

▼図1　従来のデータ分析基盤と、デジタルマーケティングで求められるデータ分析基盤の比較



注4）URL http://customerexperiencematrix.blogspot.jp/2013/04/ive-discovered-new-class-of-system.html

作り、ターゲティング広告を行ったりします。たとえば、女性向けコスメECを展開していると仮定します。Webでターゲティング広告を出す場合には、50歳台でPCのECサイトにアクセスする男性にコスメの広告を送っても誰も興味を持たなさそうですが、20〜40歳台でファッションが趣味で頻繁にファンション系ECサイトに訪問する女性に広告を配信すれば効果があると考えられます。またすでに自社に訪れているユーザに対して何度も配信するのは非効率なため、1度自社サイトに訪問したCookie IDや広告IDには配信しないといったことができるようになります。これにより無駄な広告費の削減につなげることも可能です。

CDPとプライベートDMPは、主となるデータがユーザIDかCookie IDかというところに違いがあり、しくみとしては似ています。そのため、CDPとプライベートDMPを兼用したしくみも構築できるでしょう。基本的には併用できるものです。そして、CDPやDMPのように、さまざまなデータソースからよどみなくデータを集め、それらを活かすためのデータ分析とデータ連携をすることを、Live Data Managementと呼んでいます。

マーケティングに活用できるデータ分析基盤の機能とは

さて、CDPやプライベートDMPの役割を担うデータ分析基盤にはどのような機能が必要かをリストアップします。

- データの収集をより柔軟に行えて、多様なシステムと連携できること。また、それらがストリーミングやバッチに対応できること
- 集めたデータを統合し、またセグメント化するためのデータウェアハウスの機能を持つこと
- データウェアハウスで生成されたデータをほかのシステムにデータ連携できること
- 施策の前後の分析や施策検討のために、Business Intelligenceツール（以下、BIツール）が連携できること、またはその機能を持つこと
- 収集から施策までのデータ連携の処理フローを管理するためのワークフローオートメーションシステムを持つこと

とくにデータ収集とデータ連携の部分を柔軟に、かつさまざまなデータソースやサービスに対応できるしくみにしておくことが重要になります。次節ではそれぞれを実現するためのオープンソースソフトウェア（OSS）について紹介していきます。

データ分析基盤を支えるオープンソースソフトウェア

データウェアハウス

はじめに、データウェアハウスに関してはOSSを利用して、すべて自前で運用するケースは非常に少なくなってきています。それはデータ分析では、多くの計算リソースを使うために大量のノードが必要となり、その管理をすること自体の運用コストやソフトウェアのバージョンアップなどのメンテナンスコストが非常に高く、それだけで専任エンジニアが必要になってしまうためです[注5]。そのため、クラウドサービスのデータウェアハウスを使うことが一般的です。そこでデータウェアハウスの役割を持つクラウドサービスの分析エンジンの特徴を比べてみましょう（**表1**）。

これらを比較してみると、データ分析を処理する一般的なインターフェースとしてSQLが提供されていることが主となっており、データ分析にはSQLを介して、生データに触れることが必須となりつつあります。SQLを介することにより、さまざまなデータの結合やグルーピングを容易に行え、データのセグメント化をスムーズに行えます。しかし、各サービスのほ

注5）クラウドサービスが利用できない、データ分析自体がビジネスのコアである、といったケースでは自前で運用する必要があるでしょう。

▼表1　データウェアハウス比較

比較対象	Amazon Redshift	Amazon Elastic MapReduce	Google BigQuery	Treasure Data Service
レイテンシ	アドホック（数秒～数分）	バッチ（数分～数時間の処理）	アドホック	アドホック／バッチ
処理エンジン	SQL（PostgreSQL互換）	Hive/Presto/MapReduceなど	SQL（独自SQL/SQL 2011）	Hive/Presto
ストレージ	一体型	分離型	一体型	一体型
スケジューリング	外部ツール	外部ツール	外部ツール	一体型
ジョブの依存管理	外部ツール	外部ツール	外部ツール	一体型
データインポート	バッチ	バッチ	ストリーミング／バッチ	ストリーミング／バッチ
インスタンス管理	ユーザ管理	ユーザ管理	フルマネージド	フルマネージド
スキーマ	スキーマフル	スキーマレス	スキーマフル	スキーマレス

かの機能を見てみると、スケジューリングや依存関係を考慮したジョブの実行、データの可視化などは一般的に分析エンジン自体では備えておらず、別途用意する必要があります。

そのため、CDPやプライベートDMPのためのデータ分析基盤を作るためのデータウェアハウスを選定するときには、データウェアハウス単体のパフォーマンスを比較するだけでなく、サービスのエコシステムや他機能も含めて確認し、基盤全体としてスムーズに処理フローを実現できるかという視点で見てみることも重要となるでしょう。

データ収集

次に、データウェアハウスでデータ分析を行うためには、データウェアハウスにデータを集約する必要があります。しかし、データ収集を行うことは非常に手間と時間がかかる作業です。たとえば、ログデータはサーバの数だけ散在しているだけでなく、アプリケーションごとにログのフォーマットが異なるということも往々にしてあります。また、外部サービスと連携している場合やデータベースに保存されているデータのように、定期的にダンプされたCSVファイルを収集するケースもあります。こうした多種多様なデータを横断的に分析するためには、さまざまなデータソースに柔軟に対応できる収集ツールを使い、データウェアハウスに集計しやすい形で集める必要があります。また収集ツールは、データの発生頻度によってストリーミング処理とバッチ処理の2種類に分けて活用する必要があります。

ストリーミング処理

ストリーミング処理では、Webサーバのアクセスログやセンサーデータなどのリアルタイムに生成されるデータを対象とします。そのために適したツールとしてFluentdを利用することが、一般的になってきました。Fluentdとは、米トレジャーデータ社が中心となり、オープンソースとして開発を行っているログコレクタです。特徴として、スキーマレスであること、データ入出力加工がプラグイン機構であること、バッファリングの機構によって信頼性の高いログ転送のしくみになっていること、という点が挙げられます。

また、大規模サービスに導入されることも多くなっていますが、データ収集対象が数百～数千台のサーバといった規模になると、それぞれのサーバから直接データウェアハウスに書き込むと、データウェアハウスに過度な書き込み負荷がかかってしまうといった問題が新たに発生します。こうしたケースにおいては、データの中継サーバを用意し、そこでさらにFluentdで集約させて、転送することもあります。そのため、数百以上のFluentdサーバがある場合には、

一時的に保存データを格納することに特化したキューのシステムを利用することも有効です。Apache KafkaやAmazon Kinesisは、そうしたキューのシステムとして利用されます。Fluentdに関しては第2章で詳しく紹介されます。

バッチ処理

Fluentdはリアルタイムに生成されるデータに対しては有効なツールでした。しかし、RDBのマスタデータや外部サービスから定期的に生成される数GBのCSVファイルといったデータに対しては、Fluentdのようなストリーミングではカバーできない課題もあります。

たとえば、こうした巨大なファイルをインポートする際には、インポートして終わりではなく、このデータとほかのデータをIDで結合して正規化をしたり、インポートが完了したら即座に集計を実行したり、といった後続の処理があることが一般的です。こうしたバッチ処理のケースに対して、これまでユーザは自前のスクリプトやETL（Extract、Transform、Load）ツールを作って、データ収集を行っていましたが、エラーハンドリングやリトライのしくみ、インポート速度といったパフォーマンスの問題などの課題があり、スクリプトを作ること自体がたいへん労力のかかる作業となってしまっていました。

このような課題を解決するために、米トレジャーデータ社はEmbulkをオープンソースとして開発しました。並列データ転送フレームワークのEmbulkは、Fluentdと同じプラグイン機構を持っており、入力、出力、データ加工などをプラグインによって実現できます。また、プラグイン自体を開発することもでき、さまざまなユーザが開発したプラグインがオープンソースとして公開されており、実務で貯めたナレッジをもとにさまざまな課題を多くの人と共有することで、ノウハウが詰まったプラグインを継続的にメンテナンスできるようになります。Embulkのプラグインのリストは Webサイト[注6]で確認できます。Google AnalyticsやMarketo、Salesforceといった外部サービスのAPIを利用する場合には、基本的にはバッチ処理となるため、Embulkのような汎用的なしくみを利用することで、これらのAPIを使ったデータの取り込みが非常に容易になるだけでなく、データの他サービスとの連携もスムーズに行うことができるようになるでしょう。Embulkも第2章で改めて紹介します。

ワークフローオートメーション

ワークフローオートメーションとは、指定した時間に複数のジョブを実行し、各ジョブの進行を管理するためのシステムを指します。たとえば、ジョブのスケジュール実行、ジョブの依存関係の解決、エラー時の通知や自動リトライ、指定したジョブの再実行、過去の実行履歴の管理などといった機能があります。

なぜ、ワークフローオートメーションツールが必要となるのでしょうか。

大量のデータを扱う場合、1つのジョブに数時間かかることもよくあります。たとえば、データのロードに1時間、前処理に1時間、集計に1時間、結果のアウトプットに30分、といった具合です。ここで最後のアウトプットで予期せぬエラーが発生し、スクリプトの実行に失敗したとしましょう。最初からやりなおすとまた3時間以上かかってしまうので、最後のアウトプットだけやりなおしたいとなった場合、スクリプトを修正して、手動でリカバリ作業を行うことになります。しかし、スクリプトを実行した翌日になってエラーに気づくのもよくあることで、そうするとますます時間を使ってしまいます。

ワークフロー管理の役割の1つは、こうしたエラーのリカバリを簡単にすることにあります。最初からリカバリのことを想定し、どこでエラーが起きたらどうやりなおすのかを意識しながら

注6）URL http://www.embulk.org/plugins/

プロセスを記述します。大きなジョブを複数の小さな「タスク」に分割し、各タスクをリトライ可能となるように実装します。

もう1つ重要な役割が、タスクの並列実行です。データ分析エンジンで手軽に大量のデータにクエリを実行できるようになってはいますが、「データが大き過ぎるので24分割してください」などと言われる場合があります。多くのシステムは「巨大なタスクを1つ実行」するよりも「小さな多数のタスクを並列実行」するほうが効率良く動作するようになっているためです。経験的には、1つのタスクが数分から1時間程度で終わるように分割すると効率的なように思います。ただし、並列度があまりに高過ぎるのは問題で、どんなシステムであれ、同時に100万リクエストも発行しようものなら大量のエラーを受け取ることになるでしょう。

そこで必要になるのが「タスクキュー」のモデルで、それにより並列処理を適度に抑制することができます。たとえ100万タスクをキューに入れても、並列度を20に設定しておけば、一度に20以上のリクエストは実行されなくなります。利用するAPIやクエリ実行エンジンによって最適な並列度は異なるため、複数のキューを使い分けて並列度をコントロールするのが理想です。

最後に必要なのが、タスクの依存関係の解決です。タスクキューで多数のタスクを並列実行し、エラーのリトライを繰り返す中で、タスクの実行順序を保証するには、各タスクの依存関係を記述できなければなりません。

これらが、ワークフロー管理に求める必須要件で、これらに加えてジョブのスケジュール実行や、わかりやすいUIなどをパッケージにまとめたものが、ワークフローオートメーションシステムです。

最近のワークフローオートメーションシステムで話題のオープンソースソフトウェアを紹介します。

Luigi[注7]は、Spotify社で開発されているPython製のワークフローオートメーションシステムです。Hadoop以外にも、MySQLやPostgreSQL、Sparkなどさまざまなエンジンに標準対応しています。タスクをPythonのクラスとして定義する仕様で、利用するにはプログラミング能力が求められます。データベースやサーバがなくとも、単体のスクリプトとして実行できるため、手軽さは抜群です。ただし、それは欠点でもあり、完了したタスクの管理がユーザ任せになるなど、利用する側に負担を強いることになります。自分でコーディングするのが苦にならないPythonプログラマには扱いやすいですが、そうでなければ敷居が高そうです。

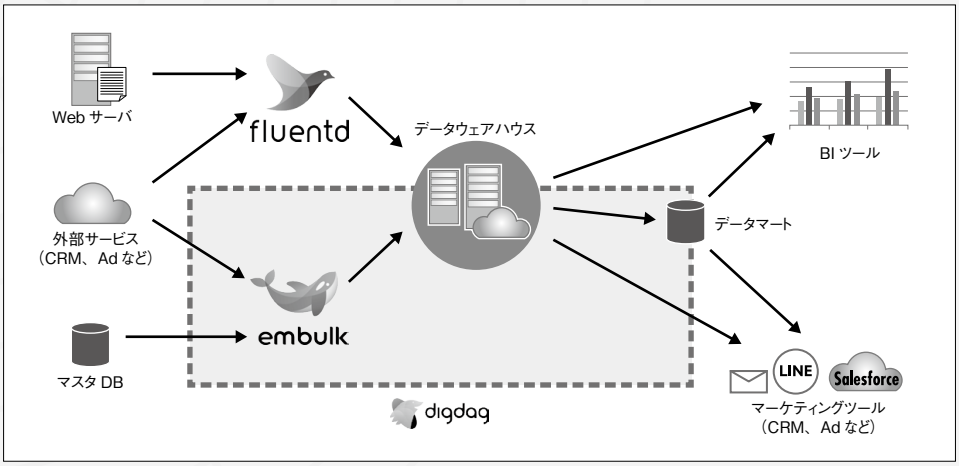
Airflow[注8]は、Airbnb社で開発されているPython製のワークフローオートメーションシステムです。Airflowはバッチ同士の関係を管理し、可視化することができます。また、実行した際のログや実行時間の推移など、さまざまなデータの閲覧をデフォルトの機能として使用できます。

Digdag[注9]は、米トレジャーデータ社が開発した、Java製のワークフローオートメーションシステムです。Digdagは上記の2つのツールとは違い、Pythonのコードではなく、YAML形式を用いてデータの依存関係を記述します。これにより、Pythonプログラマでなくとも、直感的にワークフローを構成することが可能です。また、Treasure Data ServiceやEmbulk、Amazon Elastic MapReduceなどのサービスに対してオペレータと呼ばれるプラグインによって操作することが可能になっています。また、サーバモードだけでなく、単体で起動ができるため、手軽に試すことが可能です。

このほかにも数多くのワークフローオートメーションシステムがありますが、Digdagを使ってcronの置き換えから始めてみるというのは

注7） URL https://luigi.readthedocs.io/en/latest/
注8） URL https://airflow.incubator.apache.org/
注9） URL https://www.digdag.io/

▼図2　ストリーミングとバッチ処理とワークフローの組み合わせ例



いかがでしょう（**図2**）。

Business Intelligenceツール

BIツールは、データから気づきを得たり、直感的に意味を理解したりするために必要です。データ分析という観点においては、レポーティング分析とアドホック分析の2つを行う必要があります。

1つ目のレポーティング分析とは、ユーザが分析を始めるための最初の入り口です。レポーティング分析を通して、チーム全体に共有するための基本KPI（Key Performance Indicator、詳細は後述）を用いた定型レポートを作り、同じ視点でデータを把握し、かつ簡単に情報を閲覧できるようにします。そして、この基本KPIを定期的に更新しながら、長期間に渡って、定点観測していくことにより、いつもと違う、という気づきを得ることができます。

2つ目のアドホック分析とは、レポーティングで得られた気づきから、その気づきをもとに、具体的な根拠を得るためにさまざまな条件や分析視点でデータを深掘りしていくことを目的とします。また、それだけでなく、ユーザが今までにない条件をもとに分析をするようなケースでも当てはまります。そのため、レポーティング分析を始める際の基本KPIを見つけ出すためにも利用されます。

最後に、これらレポーティング分析とアドホック分析を相互に補完できるようにすることで、レポーティング分析で得られた気づきから、違う視点で検証していくためには、別な条件でアドホックに分析をしていくことができるようになります。レポーティング分析によって発見された疑問をもとに、アドホック分析によって疑問に対するさらなる回答を得ることが可能になります。その一方で、アドホック分析で得られた気づきや、その気づきをもとにした知見をわかりやすい形で表現したレポートを、社内で共有することによって次の施策への一歩となります。そのためにはレポーティング分析の結果を保存、再利用できることが必要になります。

BIツールには、レポーティング分析向きのものと、アドホック分析向きのものがそれぞれあるため、それぞれについて紹介します。

レポーティング分析

レポーティング分析では、データを定点観測するために、ダッシュボードとしての機能を持ったBIツールを利用します。

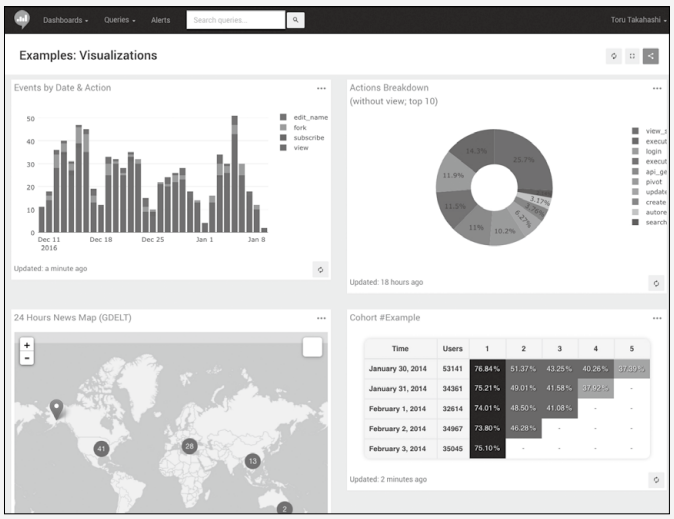
redash[注10]は、Python製のダッシュボードツー

注10）URL https://redash.io/

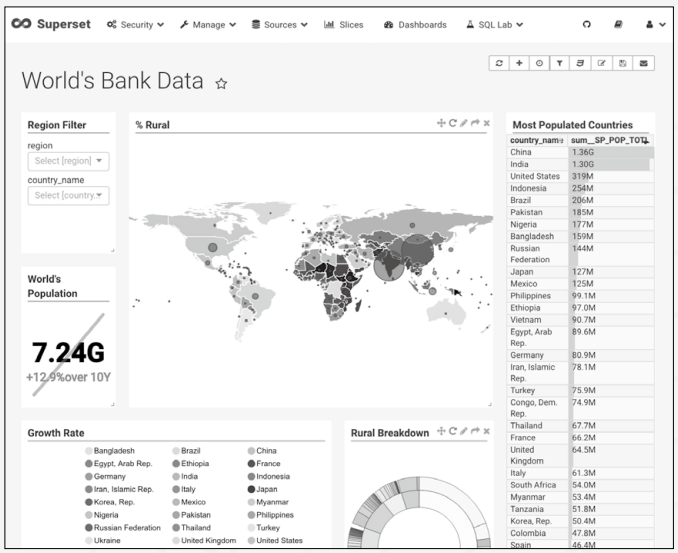
ルです（**図3**）。各種データソースに対してプラグインでクエリを実行できる点や、SQLの実行結果の可視化が簡単にできる点が、オープンソースソフトウェアのBIツールとしてエンジニアにとっては使い勝手が良いです。また、ツール自体にスケジューリング機能が付いていたり、SQLの共有なども可能です。

Superset[注11]は、Airbnbで開発されているPython製のダッシュボードツールです（**図4**）。SQLだけでなく、GUIの操作でグラフ作成も行える点がユーザの利用の敷居としては低いでしょう。その代わり、redashとは異なり、データソースがプラグインベースにはなっていないため、データソースの対応種類としてはredashよりも少ないのが特徴です。

▼図3　redashのダッシュボード



▼図4　Supersetのダッシュボード



アドホック分析

データ分析におけるアドホック分析では、レポーティング分析で得られた気づきをもとに具体的な根拠を得るために実施するケースと、ユーザが今までにない条件をもとに分析をするケースがあります。つまり、分析の当初はアドホック分析を行いながら、レポーティング分析に必要な基本KPIを見つけ出していくことが必要になります。

アドホック分析を得意とする商用のBIツールで代表的なものとしてはTableauが挙げられます。一方で、オープンソースソフトウェアでは、厳密にはBIツールではありませんが、JupyterとPandasを使ったPythonを用いたアドホック分析が普及しています。

Jupyterは、Webブラウザベースのインタラクティブシェルを提供するツールです。もともとは、iPythonという名前で、Python用のWebブラウザのインタラクティブシェルを提供していました。そこから現在は進化し、Jupyterという名前でPythonにとどまらずさまざまな言語に対応したWebブラウザベースのインタラクティブシェルとして公開されています。特徴

注11）URL http://airbnb.io/superset/index.html#

として、Notebook（.ipynb形式）というファイルに実行したシェルの内容や画像が保存されるようになっています（**図5**）。これをJupyter ServerやGitHubにアップロードすることで、分析内容をそのまま共有できるようになっており、分析結果の再現のしやすさもメリットの1つとなっています。

Pandasは、Python用の表や時系列データを扱うのに長けたデータ解析支援のためのライブラリです。Pandasを用いることでさまざまなデータベースからデータを取得するだけでなく、Jupyter上で簡単にデータ解析を行えるようになります。

レポーティング分析とアドホック分析によるKPI例

最後に、レポーティング分析とアドホック分析で算出する基本KPIと応用KPIの一例を紹介します。

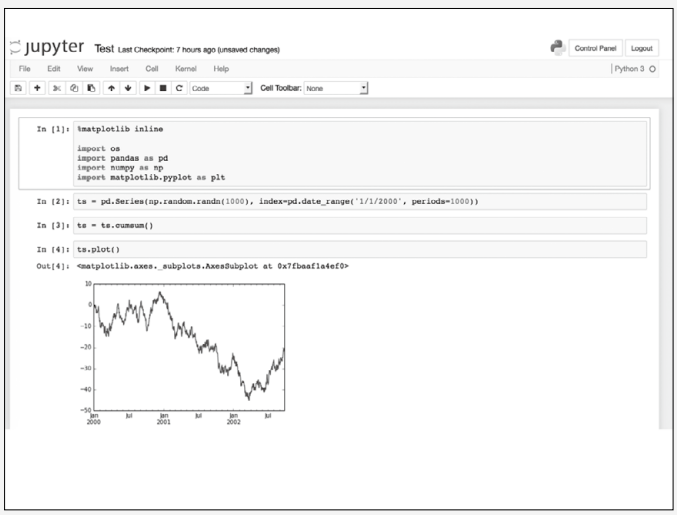
基本KPIでは、ユーザの動向を考えたときに最低限見るべき項目として**表2**の項目があります。これらは、非常にシンプルなもので当たり前のことかもしれません。しかし、この当たり前のデータを積み重ねることで意味のある新しい発見をもたらすことができます。そして、ここで明らかにしたデータをわかりやすい形で可視化し、社内に対して共有していくことが重要となります。そして、基本KPIを毎日収集し、定型レポートして観測できるようにすることで、長期間での変化がわかるようになり、何らかのイベントによる変化などを通して新たな知見を得られるようになります。さらに、基本KPIを通して得られた気づきからアドホック分析でさらに分析の深掘りをし、そこで得られた知見を定型レポート化し、新たな基本KPIとして定点観測するということが重要になります。

次に、基本KPIよりも複雑な応用KPIとして、退会分析について紹介します。ほとんどすべての会員制サービスには、ユーザの入会と退会という概念があります。そして退会における分析は、それを防止するという目的において非常に重要です。もし退会する可能性のあるユーザを事前に特定できれば、そのユーザに対して退会防止のための施策を打てるようになります。またこの退会分析に最低限必要なデータは、ユーザID、入会日時、退会日時の3つだけです。

退会分析は、おもにコホート分析と生存時間分析の2つの分析から実現することができます。1つ目のコホート分析は、同じ時期に入会したユーザが、時間を追うごとにどれくらい退会しているのかをテーブルで表示するものです。入会時期によってユーザをセグメント分けすることで、入会時期でその後の退会率に違いがあるのかどうかを見ていくことができます。また退会時期も、ある時期だけに異常にユーザが退会していれば、それが顕著に現れるようになっています。

2つ目の生存時間分析は、退会時期ではなく入会から退会までの継続期間を利用して、ユーザが将来離脱するであろう期間を統計的に補間しつつ推測する分

▼図5　Jupyter Notebookの例



▼表2　基本KPIで最低限見るべき項目

項目	説明
ページビュー（PV：Page View）	1日のアクセス回数を表す最も基本的な指標
ユニークユーザ数（UU：Unique User）	同日（または同月、同年）の1人のユーザの複数回のアクセスを1回と見なした指標
平均アクセス回数	PV/UUで計算される、ユーザ当たりの1日の平均アクセス回数
新規ユーザ数	サイトに初めてログインしたユーザを日別にカウントした指標
直近と最終訪問日までの期間の分布	ユーザ当たりの最初のログイン日と最後のログイン日の差の積み重ねによるユーザの継続期間の分布
最終訪問日の分布	各ユーザの最終訪問日ごとに集計
直帰率	外部ページから流入したものの、内部ページへ進まなかった（離脱した）ユーザの、アクティブユーザ数に対する割合
高頻度訪問ユーザの一覧（週n日以上）	直近1週間以内にn日間以上連続で訪問してくれたユーザの一覧
連続訪問ユーザー覧（直近n日連続）	直近n日間以上連続で訪問してくれるユーザの一覧

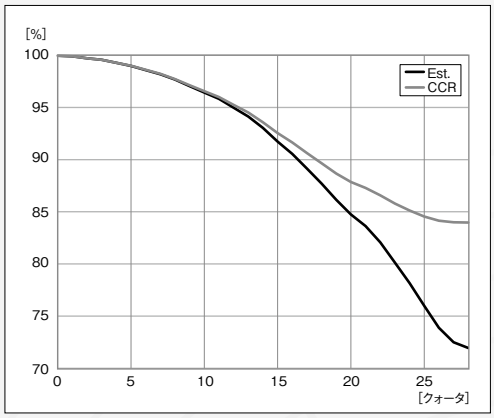
析です。注意点としては継続期間の集計は、短期間においてはそれなりに正確に集計することができても、長期の継続期間では、退会を観測できるユーザ数が（最初期に加入したユーザのみ）激減してしまうために、継続率が本来の値よりもはるかに大きく集計されることがあります。生存時間分析を用いて算出した継続率推移のチャートを**図6**に示します

図6ではクォータ（四半期）が経過するごとの累積継続率（CCR：Cumulative Continuation Rate）と推定値（Est.）による継続率を表しています。また、上述の誤差が累積継続率に含まれるために、生存時間分析によって推測値が算出されている図となっています。このようにユーザの退会分析を行うことに加えて、属性情報を付け加えることでより深い分析が行えるようになります。さらに特定のセグメントに異常値があれば、そのセグメントに対して退会防止のための施策を実施するといったことができるようになります。

ここで紹介した退会分析については、「トレジャーデータで実践「離脱分析」～コホート分析と生存時間分析～」[注12]のページでさらに詳しく紹介を行っています。

注12）URL http://blog-jp.treasuredata.com/entry/2016/10/26/170124

▼図6　退会分析の継続率推移チャート



まとめ

本章では、データ分析基盤をデジタルマーケティングに拡張させたCDPおよびプライベートDMPの紹介をしました。今、適切なユーザに適切なタイミングで適切な施策を打つために、このようなデータ分析基盤が求められています。しかしこれらは、しくみ自体はまったく新しいものではなく、従来のデータ分析基盤を拡張することで実現できるのです。CDPおよびプライベートDMPは、今後ますます、マーケティングだけでなくセールスにおいても重要な役割を持つしくみとなってくるでしょう。SD

第2章

効率的なログ収集を支えるFluentdとEmbulk

Author 吉野 哲仁（よしの てつひと） ヤフー㈱ ショッピングカンパニー テクニカルディレクター

第1章で述べられていたように、データ分析基盤には、大量で多種多様なログやデータを効率的に収集することが求められます。本章では、ログ収集ソフトウェアのデファクトスタンダードである「Fluentd」と、Fluentdのバッチ版の「Embulk」を取り上げます。

ログ収集方法の今昔

ログの収集は、以前からさまざまな方法で行われてきました。少し前までは、「ログを収集する」と言えば、**表1**のような方法を思い浮かべる方が多かったと思います。

以前はログの用途と言えば、アクセスログの集計や問い合わせ調査などに限られていました。ここ数年はログを可視化したり再利用したりするソフトウェアが増えてきて、収集したいログの種類や、連携したいシステムがどんどん多様化してきました。

Yahoo! JAPANも例外ではなく、前述の**表1**のやり方のほかに、独自技術のメッセージキューを使った連携も加わり、ログ収集のインターフェースは乱立状態でした（**図1**）。

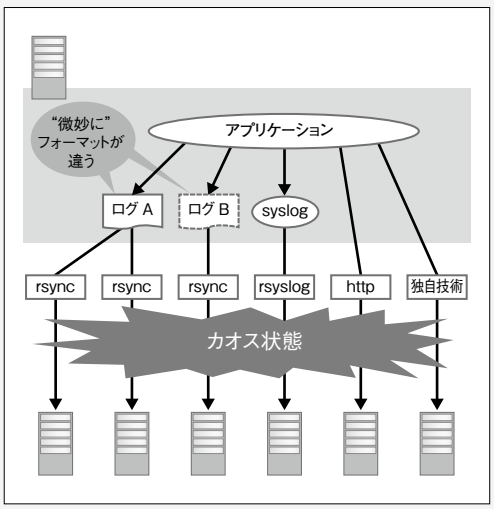
あるサーバから連携する際には“ファイル”、別のサーバでは“Web API”など、連携先を増やすたびにプロトコルの違う連携方法を個別に組み込まなければならず、さらに担当部署が違うサーバ間で連携をする場合は、連携する側もされる側もお互い気を遣いながら実装し、監視などの運用（本番で使う場合はこれが意外とたいへん）もバラバラにする必要がありました。

Yahoo! JAPANでは、かつては前述のような従来型の連携方法になっているサーバが数多く残っていました。しかし、Fluentdの登場以降は、新規に作るシステムやリプレイスのタイミングで、Fluentdに置き換えられるケースが多くなってきました。

▼表1 さまざまなログ収集方法

ログ収集の方法	欠点
ローカルディスクに落としたログファイルを定期的にrsyncして回収する	・二重取り込みリスク ・回収失敗時の考慮
syslog（rsyslog）をがんばって使う	・メッセージの転送量制限 ・本当に必要なログが埋もれてしまう
独自でWeb APIを作って呼び出し	・連携失敗時のリトライ ・APIの運用が面倒

▼図1 乱立するログ収集のインターフェース



Fluentd

Fluentdは、2011年に米トレジャーデータ社からリリースされたオープンソースのログ収集のミドルウェアです。シンプルで高い安定性、プラグイン機構などの特徴があり、Webサービスを中心に広く使われており、実質ログ収集のデファクトスタンダードとなっています。

Fluentdでできること

Fluentdは、いわゆる「ログコレクタ（ログ収集ツール）」に分類されるソフトウェアです。その名のとおり、用途はログを収集することですが、要は**テキストデータを効率良く転送・保管するソフトウェア**ですので、ログに限らずテキストデータであれば何でも扱えます（Base64エンコードすれば、画像などのバイナリデータも扱えます）。

なぜ注目されたのか

このようなソフトウェアがなぜ注目されるようになったかというと、キーワードはやはり**ビッグデータ**です。ビッグデータの元となる重要なデータソースは「ログ」です。しかし、ログの量は非常に膨大であり、かつ大量のサーバから集めるというのは、非常に手間のかかる作業でした。

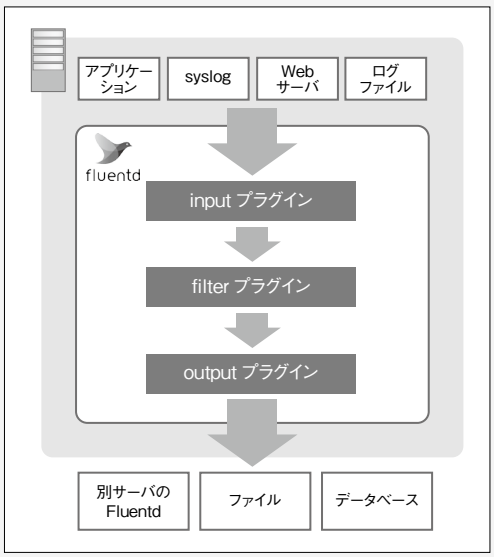
Fluentdは、それらの課題を解決してくれるソフトウェアとして、登場するやいなや一気にメジャーとなりました。

Fluentdの基本動作

Fluentdの動作は、大きく「input」「filter」「output」の3つのステップがあり（**図2**）、それぞれのステップごとに設定をしていくことになります。

つまり、シンプルに考えると、ログが「どこから来て（input）」「どのログが必要で（filter）」「どこに出力するのか（output）」を意識すれば良いわけです。

▼図2　Fluentdの3大処理ステップ



input

ログデータの入力ソースに関する処理を行います。たとえばhttpのプラグインであれば、設定されたポート番号でlistenするプロセスを立ち上げ、ログデータのリクエストを待ち受けます。

最もよく使われるパターンはtailプラグインを使ったinputでしょう。tailプラグインは、その名のとおりUNIXコマンドのtailのような動作をするプラグインで、ファイルの更新を監視し、更新イベントを拾って追加された行をinputとします。これを使うと、既存のシステムにはほぼ手を加えずにログを転送することができます。

filter

ログデータのフィルタを行います。このステップで不要なログを正規表現を用いて除いたり、タグを付けたりして振り分けしやすくできます。

output

ログデータの出力先に関する処理を行います。たとえばmysqlのプラグインであれば、「データベースに接続し、入力されたログデータをもとにSQLを作成し、insertを実行する」といったことを行います。

Fluentdを導入するには

Fluentdの導入は非常に簡単で、わずか3ステップで可能です。

①インストール

Fluentdは「ログの送信元サーバ」にインストールする必要があります（Fluentdでさらに中継する場合は、送信先にもインストールが必要です）。

LinuxやMacであれば、RPM・Deb・DMG・Ruby gemなどの方法で簡単にインストールできます。Windowsであれば、まずRubyとGitのインストールが必要で、それからGitリポジトリをcloneし、インストールします。

②設定およびプラグインのインストール

前述した、input（sourceディレクティブ）、filter（filterディレクティブ）、output（matchディレクティブ）の3つの設定をconfに記述します（**リスト1**）。filter処理が不要であれば、filterディレクティブは書く必要はありません。

また、必要に応じて、プラグインのインストールを行います。たとえばログをMySQLに保存したい場合は、mysqlプラグインをインストールします。プラグインのインストールは、fluent-gemコマンドで行うか、ソースコードをFluentdのpluginディレクトリの下に配置することでインストールできます。

▼リスト1　confファイルの3つの設定

```
<source>
  ログデータがどこから入力されるか
</source>

<filter>
  ログデータの内容をどうフィルタするか
  （任意設定のため省略可能）
</filter>

<match>
  ログデータをどこに出力するか
</match>
```

③起動

「fluentd -c confファイル」で起動します。これだけでログが流れるようになります。

Fluentdの特長

豊富なプラグイン

Fluentdを初めて使う人は、そのプラグインの多さに驚くと思います。よほど特殊なシステム構成になっていなければ、ほぼ要件にマッチするプラグインが見つかるはずです。

また、プラグインの実装も非常に容易ですので、微妙に機能がマッチしないプラグインがあったとしても、自分で簡単に拡張することができます。実装は基本的にRubyですが、Rubyを扱ったことのない人でも、基本的な動きが理解できていればそれほど難しくないはずです（筆者自身もRubyの経験は浅いですが、簡単に実装できました）。

この「豊富なプラグイン」と「プラグインの実装の容易さ」は、Fluentdの導入ハードルを大幅に下げ、Fluentdがここまで普及した要因になっていると言えます。

タグによるログの仕分け

ログのメッセージごとに「タグ」を付けて種類分けできます。さらに、タグにはツリー状に親子の階層を作ることができ、複雑なルーティングが可能となっています（**図3**）。

このタグのしくみは、ログの用途（連携先）が増えれば増えるほど威力を発揮します。たとえば**図3**のアクセスログは全社のアクセス統計

▼図3　ツリー状のタグの例

```
log
 ├─ apache          ← Webサーバログ
 │   ├─ access      ← アクセスログ
 │   └─ error       ← Webサーバエラーログ
 ├─ order           ← 注文ログ
 │   ├─ prod        ← 非個人情報
 │   └─ secure      ← 個人情報
 └─ monitor         ← 定期モニタリングログ
     └─ sysstat     ← sysstatログ
```

▼表2　便利なプラグイン

プラグインの名称	種類	機能
fluent-plugin-forest	output プラグイン	タグによるログ振り分けがより便利になるプラグイン。* を使いワイルドカード的にタグを記述することが可能になり、振り分け先の設定部分で同じ設定を何度も書く必要がなくなる
fluent-plugin-multiprocess	input プラグイン	前述したマルチプロセス化のプラグイン。複数コアを持つCPUでは検証の価値アリ
fluent-plugin-secure-forward	input プラグイン	SSL を利用してセキュアなログ転送を実現するためのプラグイン。平文での送信が許容できないサービスの場合は必須と言える
fluent-plugin-parser	filter および output プラグイン	ログの内容を正規表現でパースして、任意の内容に組み換えできるプラグイン。パースにはCPUを消費するので、使うサーバに気をつけること

システムへ、注文ログの非個人情報はサービスの統計システムへ、個人情報はセキュアな統計システムへなど、ログの送付先を個別に設定できるため、柔軟かつ簡単に仕分けできます。

また、filterプラグインであるfilter_grepを使うと、ログの内容を正規表現でフィルタし、マッチするログのみ処理するといった、さらに柔軟な仕分けも可能です（ver.0.12以降）。

豊富な言語モジュール

Fluentdでは、公式でもJava、Ruby、Python、PHP、Perl、Node.js、Scala、.NET用の言語モジュールが提供されています。これにより、独自のアプリケーションから直接Fluentdを使ってログを送信することができます。システムの制限でサーバのローカルストレージにログを落とすことができないシステムでとくに有効です。

JSON形式のログレコード

Fluentdのログの内容は、inputプラグインの段階で、内部的にJSON形式に変換されて扱われます。これによって、アプリケーションやプラグインの中での扱いが非常に容易になっています。開発者にとっても視覚的にわかりやすく、プラグインの開発のしやすさにもつながっています。

マルチプロセスでの動作

Fluentdはデフォルトではシングルプロセスで動作するため、複数コアを持つCPUでは、CPUリソースを十分に使うことができません。また、大量のログを扱う場面では、CPUがボトルネックになることが多いです。

マルチプロセス化のプラグインを使えば、「CPUのコア数の分だけ子プロセスを立ち上げ、別々のポートで待ち受けする」といったことが実現できます。

ただ、闇雲にプロセスを増やしても、今度はメモリやディスクI/Oがボトルネックになってしまうので、topコマンドやsysstatコマンドの出力結果をよく見ながらチューニングしましょう。

情報が豊富

Fluentdの公式ドキュメントは英語のみですが、Web検索をすると、日本語でもFluentdの解説記事や導入事例がたくさん出てきます。ほかのソフトウェアに比べても情報量が多く、今やデファクトスタンダードとなったFluentdの強みでもあります（ただし、非常に進化のスピードが速いソフトウェアですので、最新バージョンでは、既存の情報と動作や仕様が違っている可能性があります。最新バージョンの動きは自身でよく検証しましょう）。

便利なプラグイン

Fluentdのプラグインは非常にたくさんありますが、中でも多くのケースで使うであろうプラグインをいくつか**表2**に挙げます。

どんなシステムと連携ができるのか

基本的には、よほど特殊な用途のシステムでない限り、プラグインの実装さえできればどんなシステムとも連携できます。Amazon Web Servicesのようなクラウドサービスもプラグインが出ていますし、メジャーなDBもプラグインが用意されています。悩む前に、まずはFluentdの公式ページからプラグインを探してみることをお勧めします[注1]。

比較対象となるソフトウェア

Fluentdとよく比較されるログ収集のソフトウェアとしては、「Logstash」「Apache Flume」があります。

Logstash

Logstashは、Elasticsearchで有名なElastic社が公開しているソフトウェアです。こちらもプラグインが非常に豊富で、シンプルで導入しやすいソフトウェアです。Logstash + Kibana + Elasticsearchの組み合わせ（すべてElastic社製）で使われるケースが多いですが、Logstashの部分がFluentdに置き換えられることも多いです。

これは、Fluentdに比べて「ルーティング機能」「タグを使ったログ振り分け」「バッファ機能」などに差があるためで、プラグインである程度補完できるものの、比較をした結果、Fluentdが選択されることが社内でも多いです。

Apache Flume

Apache Flumeは、ASF（Apache Software Foundation）のトップレベルプロジェクトとして開発されているソフトウェアです。HDFSへの書き込みに親和性が高く、Cloudera社のHadoop関連のディストリビューション「CDH」に入っているということもあり、Yahoo! JAPANでもHadoop系のプロダクトとセットで使われるケースが多いです。

備えている機能が多く、冗長性も高いので安定している印象がありますが、その分設定がFluentdやLogstashに比べ、若干複雑です。また、プラグインのしくみを備えていますが、数はあまり多くないため、足りないものは自分で実装する必要があります（プラグインはJavaで実装）。

導入前に最新情報で比較検討を

ログ収集のソフトウェアはFluentd以外にも、商用を含めると最近は多くなってきました。Web上では比較記事を多く見ますが、差分だと思っていた機能が最新のバージョンでは対応されていたり、すでにプラグインができていて差分が埋まっていたりと、とても競争が激しい分野でもあります（たとえば、今までLogstashは「Windowsでも動作する」ということがメリットとして挙げられていましたが、FluentdもWindowsで動作するようになっています）。

また、プラグインの実装も、Ruby、JRuby、Javaなどで分かれていますので、現場のスキルも考慮した選択が必要です。

最新の情報を確認したうえで比較検討をしましょう。

導入にあたっての注意点

3つの“QoS”

Fluentdを使ううえで、QoS（Quality of Service、サービス品質）のレベルは必ず確認しておきましょう。QoSとは簡単に言うと、「ログの配信保証レベル」です。Fluentdのようなイベントログを送信するシステムでは、一般的に大きく3つのQoSレベルがあります（**表3**）。

表3の3つのQoSレベルの内、Fluentdがサポートしているのは「at most once」「at least once」の2つです。

注1）URL http://www.fluentd.org/plugins

▼表3　3つのQoS

QoSレベル	説明	リスク
at most once	ログは最高1回配信される。再送されることはない	障害時にログがロストする可能性あり
at least once	ログは最低でも1回配信される。送信失敗時は再送される	障害時にログが重複する可能性あり
exactly once	ログは確実に1回配信される（これが一番望ましい）	―

at most once

Fluentdのデフォルトの設定は「at most once」です。このレベルだと、障害時にログがロストするリスクがあります。このリスクを軽減するには、「at least once」を設定するか、送信側と受信側でログの突合を別途行うなどの工夫が必要です。

at least once

「at least once」はconfファイルに“require_ack_response”を記述することで設定できます（ver 0.12から）。require_ack_responseとはその名のとおり、送信先がACK（正常終了を知らせる応答）を返さないとログをバッファリングし、再送し続けるということです。

このレベルだと、ログが多重に送信される可能性があり、結果として同じログが重複して記録されるリスクがあります。このリスクを軽減するには、ログにユニーク番号を振って受信側で重複を省くなどの工夫が必要です。

“課金データ”は要注意

前述のとおり、Fluentdでサポートされているのは「at most once」と「at least once」の2つです。ログを“確実に”“1回だけ”送信されることが要求されるシステムにはあまり使うべきではありません。たとえば、ログ行数をもとにアクセス単位で課金をしているような場合など、ロストや重複が許されないデータは、Fluentdで扱うのはあまりお勧めできません。

特徴をきちんと理解して使う

ここまで述べてきたように、Fluentdは設定や使うプラグインによって動きが異なります。とくに「at most once」は気づかずにデフォルトのまま使っている方も多いかと思います。自分たちが採用している構成の特徴をきちんと理解したうえで、リスクを回避・軽減するための構成やプラグインを使っていきましょう。

設定の反映

設定変更の頻度はなるべく少なく

confファイルの設定反映やプラグインの修正反映には、Fluentdプロセスの再起動が必要です。本番運用開始後に、日常的に再起動がされるような状態はあまり望ましくありません。Bufferプラグインを使うなど、再起動時にログがロストしないような工夫をしましょう。

Yahoo! JAPANでの活用事例

全社のログ収集プラットフォーム

Yahoo! JAPANでは、全社のログ収集プラットフォームへの連携にFluentdを使用しています。当社は数万台規模のサーバを所有しており、それらのサーバから1秒間に数十万行という膨大な量のログが流れてきます。

大量のログをさばくために

その膨大なログをさばくためには、いかにログ収集サーバの負荷を分散するかがポイントになってきます。

ログ収集サーバでは、Fluentdをマルチプロセス化して対応しています。開発当初はマルチスレッドでの動作を検討していましたが、「メモリを大量に消費してしまう」「思ったより性能が出ない」「スレッドセーフを考慮しないと

いけない」などの問題があり、“シングルスレッド+マルチプロセス”という構成になりました。CPUコア数を考慮して、1サーバあたり12プロセスを立ち上げ、サーバリソースを最大限使える構成にしています（**図4**）。

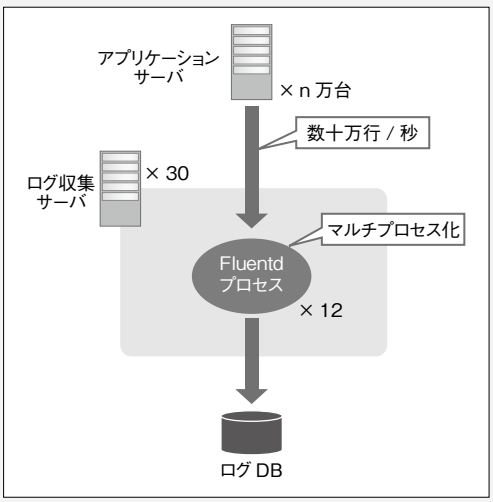
confの内容を配信型に

各クライアントサーバに入っているFluentdのconfファイルには、confファイルの内容を配信するAPIのURLが1行だけ記述されています。Yahoo!ショッピングやヤフオク!などのサービスのエンジニアは、内製した“設定内容編集ツール”（**図5**）で、任意の設定を入れることができます。

このツールでは、ホスト単位やサーバのグループ単位で設定を編集することができ、サーバにログインすることなく設定を反映させられます（**図6**）。

Yahoo! JAPANには、サービスのエンジニアのほかに、この全社のログ収集システムを運用管理する運用エンジニアがいます。実際に各サーバに反映されるconfファイルの内容には、そのログ収集システムの運用エンジニアが、全サーバに共通で入れたい設定の内容も反映されています。

▼図4　Fluentdのマルチプロセス化



管理者が入れたい設定というのは、「全サーバから共通で取得したいsysstat情報」「エージェントのHeartBeat形式」「1秒あたりのログ送信量」などがあります。

これらの設定を管理者が行うことで、たとえば「トラブルで特定のサーバからのログ送信量が増加」した際には、そのサーバのログ送信量を制限することで、他サービスへの影響を最小限にするなどのコントロールが可能になります。

ショッピングビーコンサーバ

Yahoo!ショッピングでは、商品ページにアクセスした際のユーザのアクセスログをすべて記録しています。このアクセスログは、ユーザから注文が行われた際に、「このユーザはどのページから来て」「どのページにランディングしたのか」といった情報を取得するために使用します。

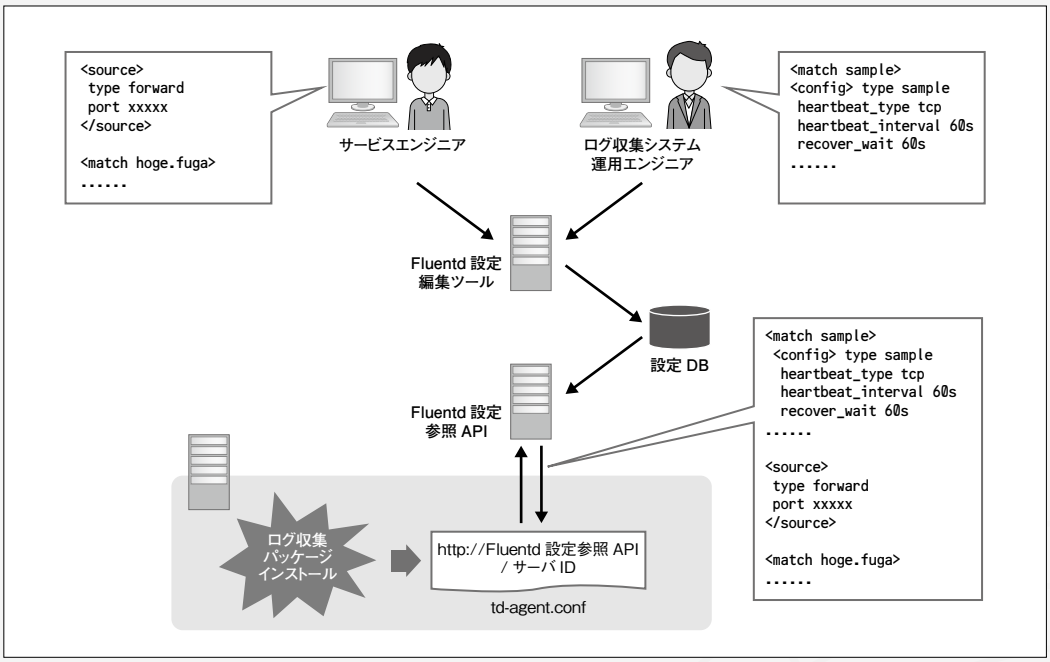
このアクセスログを記録するためのサーバを筆者たちは「ビーコンサーバ」と呼んでいます。ビーコンサーバでは、ページのHTMLに埋め込まれている1ピクセル×1ピクセルの画像を返すサーバで、画像のリクエストが来た際に取得で

▼図5　内製のFluentdのconfファイル編集ツール

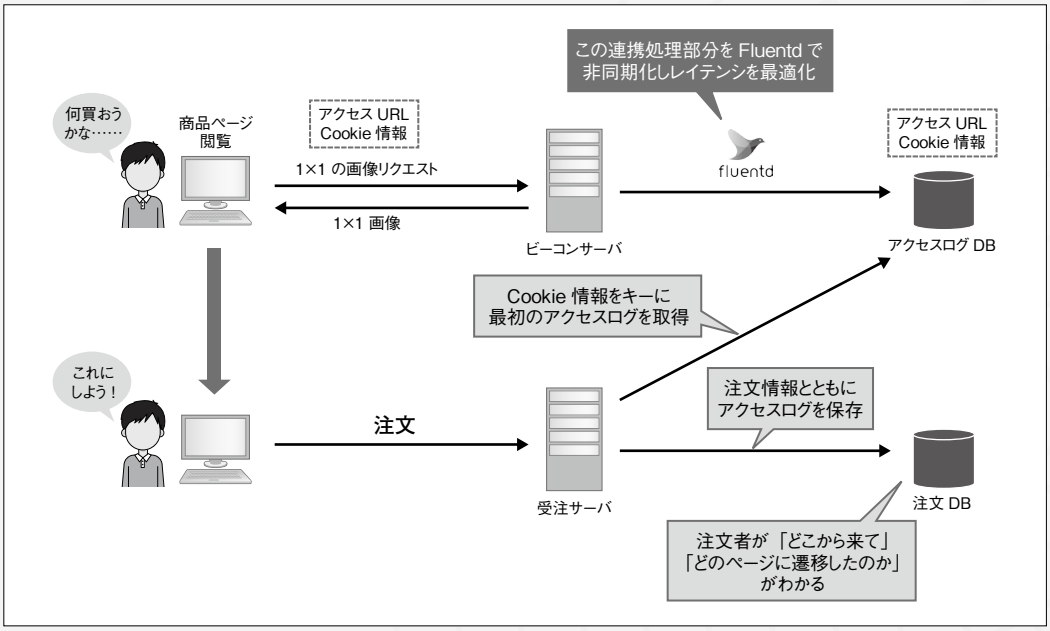
きる「アクセスURL」や「Cookie情報」をDBに保存する役割があります。これらのデータを連携する部分に、Fluentdを使用しています（**図7**）。

商品ページは、Yahoo!ショッピングの中で最もアクセスが多いページの1つです。そのため、ページの表示速度はとくに気をつけなければいけません。Fluentdでバックエンドへのデータ連携処理を切り離し、非同期化することで、

▼図6　confファイル内容の配信



▼図7　ショッピングビーコンサーバへのFluentd導入



ビーコンサーバのレイテンシに影響を与えることなくデータ連携ができます。

バックエンドへの連携部分をFluentdに置き換えて非同期化する方法は、Fluentdの事例ではよく見るパターンの1つです。ただし、「3つの“QoS”」の項でも述べたように、データを漏れなく確実に記録する必要がある場合は注意して導入しましょう。SD

COLUMN Fluentdのバッチ版 Embulk

Fluentdはリアルタイムなデータ収集で大活躍するミドルウェアですが、一方で一括でまとめてデータを投入したいというニーズは変わらず存在します。

- **Fluentdを導入したが、導入前の過去データも投入したい**
- **1日1回バッチ処理で、あるDBからあるDBにデータをコピーしたい**
- **あるサーバに置いてあるテキストファイルを、リモートにあるDBに取り込みたい**

Fluentdでの解決が難しいこれらのニーズに対応したものが「Embulk」です。EmbulkはFluentdの開発元である米トレジャーデータ社が開発した、いわば「バッチ版Fluentd」です。

Fluentdと同じく、プラグインによってinput、outputをカスタマイズしてデータ転送できます。プラグインはRuby、Javaで実装できるため、Fluentd同様に簡単に導入が可能です。

今までは、それぞれ専用の転送用バッチを作り、cronで動作させるなどをしてきた人も多いと思いますが、Embulkを使えば解決できます。

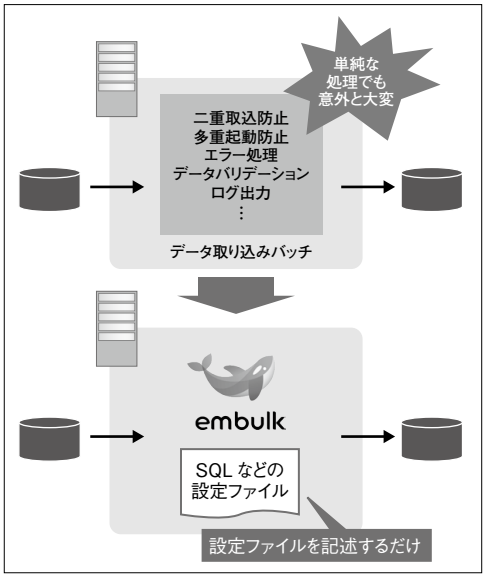
Yahoo! JAPANの導入事例を1つ紹介します。当社には、OracleやMySQLなどのDBの導入・管理・運用を一挙に引き受ける「全社DBチーム」があります。DBの運用管理において、「キャパシティプランニング」は重要な業務の1つです。ディスクやメモリなどのリソースの増加傾向を監視し、サーバ増設や移行の提案を行いますが、Oracle DBにおいては、オラクル社から提供されているツールではDB横断での管理が難しいなど、社内のニーズが満たせませんでした。そこで、お手製のバッチを作り、Oracleのメトリクス管理DBからデータを引き、お手製のツールを作って可視化していました。

やがてチーム体制が変わり、いつしかお手製のバッチやツールをメンテナンスできる人がいなくなってしまいました（これはどの現場でも非常にあるあるな展開だと思います）。

そこで、Embulk＋Elasticsearch＋Kibanaの組み合わせを導入しました。Oracleのメトリクス管理DBからSQLでデータを引き、Elasticsearchにデータ投入する個所はすべてEmbulkのプラグインで実現しています。SQLやDBの情報はすべて設定ファイルに書けばよく、コードの実装はほぼゼロになり、運用担当者の負担はグッと減りました（**図A**）。

単純なデータ転送のバッチでも、「二重起動防止」「エラー処理」「データのバリデーションチェック」「ログ出力」など、業務で使ううえで考慮しなければいけないことは意外とたくさんあります。今回のように、まとめてデータを転送・投入したい場面では、Embulkは強力なツールです。まだリリースされてから2年ほどしかたっていませんが、同じケースで悩んでいる人は一度試しに使ってみてはいかがでしょうか。

▼図A　Embulkでコード実装がほぼゼロに



第3章 ビッグデータの集計・加工を支えるHadoop

Author 小澤 祐也（おざわ ゆうや） クラスメソッド㈱

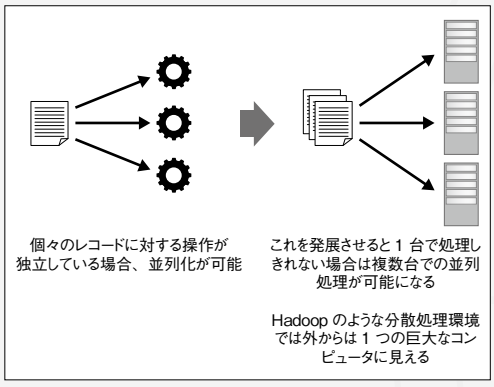
ビッグデータは蓄積しただけでは価値がありません。うまく活用して初めてその意味をなします。本章では1台のサーバでは処理しきれないような大量のデータを扱うためのソフトウェアである「Hadoop」と、「Spark」などの関連ソフトウェアについて解説します。

Hadoopで何ができる?

Hadoopと分散処理

Hadoopは分散処理を実現するためのオープンソースソフトフェアです。複数のコモディティなサーバを大量に並べることで、単一の環境では扱いきれないような大規模なデータに対する処理を実現します。コモディティなサーバを使うため、専用のハードウェアを用意する必要がありません。ただし、コモディティと言っても安価な低スペックマシンで動くということではなく、処理対象となるデータ量や性質から1台あたりのスペックと並列台数を適切に選択してやることで初めてその真価を発揮します。

▼図1 並列処理



分散処理では複数のサーバを用意して並列して同時に処理を実行します。これはCPUのコアに例えるとわかりやすいかと思います。CPUの1つのコアあたりの処理性能はクロック周波数で決定します。かつてはこのクロック周波数を向上させることでCPUの性能を向上させていましたが、現在ではコアの数を増やすことで同時に処理できる量を増やし、それによって処理性能を向上させています。普段はあまり意識することはないかもしれませんが、コンパイラが並列処理（**図1**）を行うように最適化してくれていたり、ライブラリの実装がマルチスレッドでの実行を行うようになっていたりします。

このようなしくみを考えると、並列処理が行えるようにうまく実装されていればCPUのコア数を増やせば増やすほど処理速度は向上するように思えます。しかし、実際には半導体素子の集積密度には限界があります。また、ビッグデータを扱う際にはTB（テラバイト）のオーダでは収まらない量のデータを扱うことも多いため、CPUだけでなくメモリやハードディスクの容量などといった制約があります。

そのため、ハードウェアのスケールアップではなく、スケールアウトによって複数台の環境全体で見たときのCPUのコア数やメモリ・HDDの容量を大幅に向上させるのが分散処理となります。

分散処理を行うと、物理的に別な環境に必要

なデータがあることや、複数のサーバをどのように連携させるかなど、1台の環境で処理していたときには気にする必要がなかった問題が発生します。

Hadoopはそれらの分散処理を行う際の難しい問題をフレームワーク側で実装してくれているため、利用者はそのしくみを利用して必要な処理の部分の実装に専念できます。

とはいえ、それらを実現するためにどのような動きをしているのかを知っておく必要がある場面も多いため、まずはその解説をしていきます。

Hadoopでの分散処理のしくみ

Hadoopでは、おもに大量のデータに対するバッチ処理を行います。前述のとおり、ビッグデータの分析では一度に扱うデータの量がTB、PB（ペタバイト）といったサイズになることもあり、1台の分析環境ではデータの保存や決められた時間内でのバッチ処理の完了が困難になります。

Hadoopで分散処理を実現するしくみとして、まず1つのデータ全体を1ヵ所に保存するのではなく、データを分割して複数のサーバに保存します。データに対する処理を行う際には、個々のサーバでは自身が処理する対象のデータのみを扱い、それ以外のサーバがどのように動いているかに後述のShuffleというフェーズを除き干渉しません。そのため、全体で協調しながら1つの処理を行うというよりは、1件1件に対して同じ処理を行っていくというイメージになります。

これは一見すると、複雑なことはできないと言っているようにも思えますが、この処理を組み合わせていくことにより複雑な処理でも実現可能になります。とはいえ、Hadoopが万能というわけではなくもちろん苦手な処理も存在します。

たとえば行列演算のように、データ全体が共有されている必要があるような処理を行う場合は、Hadoopを使うとサーバ間をまたいだデータのネットワーク転送を行う回数が多くなるため処理速度は低下してしまいます。

またサーバを大量に並べるということは、それだけ管理する対象が増えるということです。

これはたとえば簡略化した計算となりますが、1台あたりが99.99%の稼働率であったとしても、1,000台で分散処理を行っていれば全体を1つとして見た稼働率は0.9999の1,000乗＝0.9048となり、およそ10%の確率で最低でもどこか1台以上は故障していることになります。

Hadoopでは、このように故障などによって利用できない状態になっているサーバの切り離しや、処理途中で停止した際のリカバリなどもフレームワーク側で面倒を見てくれ、復帰やサーバの追加なども容易に行えるようになっています。

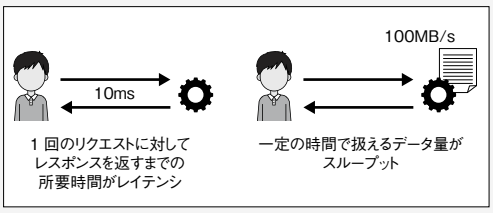
処理速度の単位・レイテンシとスループット

Hadoopは分散処理を利用することで処理速度を向上させていると書きましたが、ここで処理速度の定義について解説しておきます。処理速度といったときに一般に挙げられるものとして「レイテンシ」と「スループット」という2つの考え方があります（**図2**）。

レイテンシとは、1回の処理にかかる時間になります。これはWebサーバのレスポンスタイムのようにリクエストを投げてからどのくらいの時間で結果が返ってくるかというのを測定したものになります。OLTP（OnLine Transaction Processing）など、リアルタイムにレスポンスを期待するような処理ではこちらで処理速度を見ることが多くなります。

一方、スループットとは一定の時間の間にどれだけの量を処理できるかという単位になります。

▼図2　レイテンシとスループット



こちらはたとえばビッグデータに対して分析を行う際に、一定の時間でHDDからどれだけの量のデータを読み込めるかなどの指標になります。データ1件あたりに対しての処理内容が軽いものであっても、現実的な時間で読めるデータ量が少なければビッグデータの分析は行えません。

Hadoopでの処理速度は、スループットのほうを意識したものになります。扱うデータ量がどんなに少なくても、通常は最低でも数秒から数十秒の時間を要します。

一方で、扱うデータサイズが多くなった場合でもサーバ台数を増やしていけばリニアにスケールしていくといった具合です。これを実現するためにHadoopではデータが分散して置かれているという性質を利用し、読み込みや書き込みは各サーバで行うことでその部分も並列で実行しています。

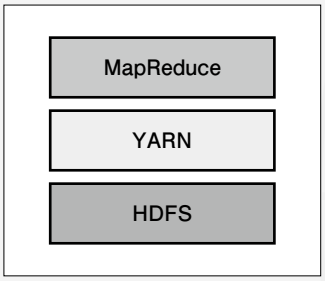
Hadoopを構成する基本要素

Hadoopとはどのようなものであるかを解説したところで、次にHadoopがどのようなしくみで成り立っているのかを見ていきます。Hadoopは次のような技術スタックで構成されます。

・HDFS
・YARN
・MapReduce

これらは**図3**のように、階層構造を持ったレイヤとして構成されます。順に解説していきます。

▼図3　Hadoopの技術スタック



HDFS

HDFS（Hadoop Distributed File System）はHadoopで利用するためのデータを格納するファイルシステムです。Hadoopで扱うデータはHDFSから読み込まれ、処理を行った結果はHDFSに書き込まれます。HDFSはLinuxライクな階層構造とパーミッションの設定を行うことができ、操作を行うコマンドもよく似ています。

入力として扱うデータはKafka[注1]やFluentdなどを利用し、ログデータなどをリアルタイムに流し込んだり、Embulk[注2]などでバッチ処理的にアップロードしたりできます。もちろんHDFSのAPIを利用して手動でアップロードしたり、独自のシステムを構築したりすることもできます。

HDFSの実態は、分散処理環境の個々のノードにファイルを分散しておくしくみになっており、ブロックサイズと呼ばれる単位で分割されたデータが複数個所に置かれます。このときに、元は同一ファイルのデータであっても分割されるため、処理の実行の際にあらためてデータを分割する必要がなくなります[注3]。

また、同じデータは複製されて複数個所に置かれるため、どこか1台が故障した際に、そこに置かれていたデータが消失することもありません。

HDFSではNameNodeとDataNodeと呼ばれるコンポーネントが登場します（**図4**）。NameNodeはHDFSでのマスタノードであり、全体のファイルツリーの管理や各DataNodeの生死、データの複製数が適切かの管理などを行っています。

注1）URL https://kafka.apache.org
注2）URL http://www.embulk.org/docs/
注3）実際にはブロックサイズ単位で機械的に分割されるため、その都合で分割されたレコードのみネットワーク転送が発生するといった状況がありえます。また、巨大なgzipファイルなどファイル全体を1ヵ所に集約する必要がある状況の場合、大量のネットワーク転送が発生してしまいます。そのため圧縮ファイルを扱う際は事前に分割しておく必要があります。

NameNodeにアクセスできなくなると、HDFS全体でどこにどのデータがあるかわからなくなります。古いバージョンのHadoopではSPOF（Single Point of Failure；単一障害点）でしたが、現在ではHA（High Availability；高可用性）構成をとることができます。

Hadoopの利用者側からはこれらのしくみは隠蔽されており、NameNodeとの間でやりとりするだけで、全体として1つの巨大なファイルシステムであるかのように見えるしくみになっています。

YARN

YARN（Yet Anathor Resource Negotiator）はHadoopでのコンピューティングの部分を支えるコンポーネントになります。

YARNは、

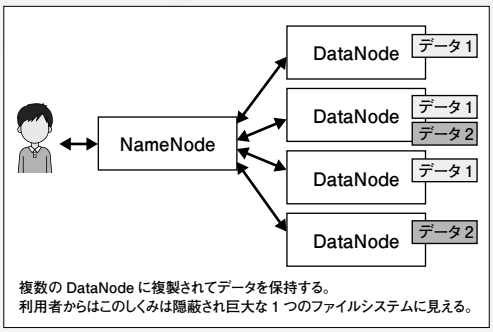
- ResourceManager
- NodeManager
- ApplicationMaster

という要素で成り立ちます（**図5**）。

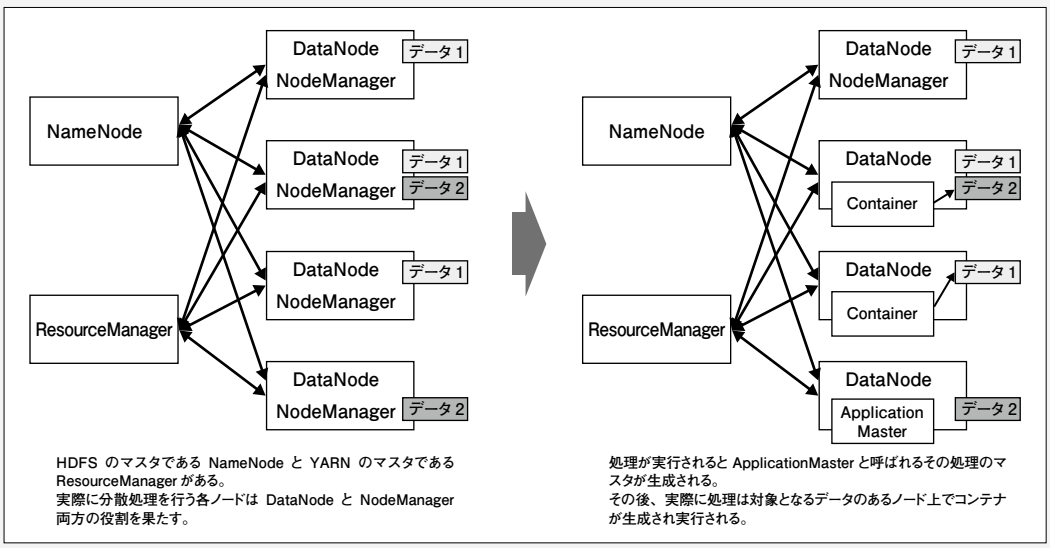
YARNでは、分散処理環境におけるクラスタ全体でのコンピューティングリソースの利用状況の制御や、各ジョブのスケジューリングなどを行っています。これはResourceManagerというコンポーネントによって実現されています。ResourceManagerは、利用者からジョブが投入された際にジョブの実行に関するスケジューリングを行います。

実行のタイミングで、ResourceManagerはまずApplicationMasterというそのジョブのマスタとなるコンテナを生成します。このコンテナは、NodeManagerの中からリソースに空きがあるものが選出され起動します。以降のジョブの実行の制御は、ResourceManagerではなくApplication Masterが行います。ApplicationMasterは必要に応じて、ResourceManagerにジョブの実行に必要なリソースの要求を行いながらジョブの実

▼図4　HDFSのデータ保存方法



▼図5　YARNの動き



行を行うことになります。

この際に利用者は、ジョブの処理内容を記載したファイルをJarでまとめて実行するのみとなり、この一連のフローはYARNによって隠蔽されます。そのため、ジョブやApplicationMasterが実際にはどのノードで動いているかなどを意識する必要はありませんし、ApplicationMasterを含め処理の途中で実行しているノードに障害発生した際のリカバリもYARNが行ってくれます。

ただし、ノードラベルという大まかに実行するノード群を指定する以上の制御は利用者側ではできないため、実行するノードを意識した処理の実装は行えません。

HadoopではYARNのスレーブであるNode ManagerとHDFSのスレーブであるDataNodeが分かれていません。クラスタ上にあるノードはNodeManagerでもありDataNodeでもあります。このしくみによって、ジョブを実行する際には可能な限り、必要なデータを持っているノードで処理を実行するデータローカリティというしくみを実現しています。データローカリティのしくみによってネットワークをまたいでのデータの転送によるオーバーヘッドを削減しています。

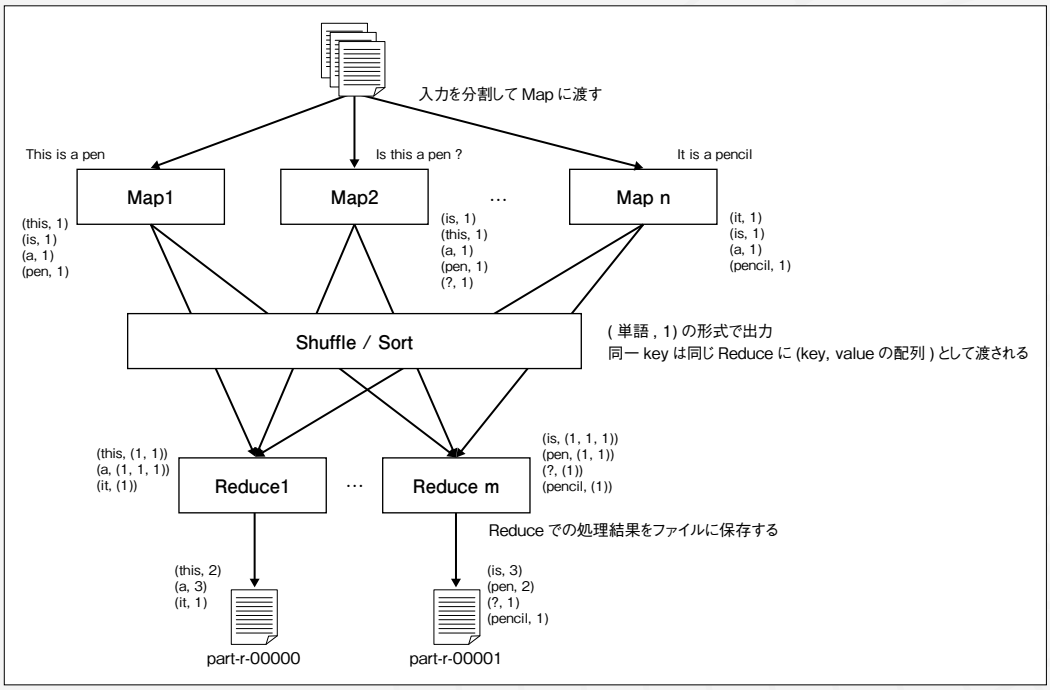
MapReduce

MapReduceは、実際にジョブの処理内容を記述するためのフレームワークとなります。利用者は、MapperとReducerという2つのクラスをそれぞれ継承したプログラムを実装するだけで、それらの制御は意識する必要がありません。

実際に利用者がMapperとReducerでどのような処理を実装するのかという例を**図6**を参考にWordCountというプログラムで解説します。

WordCountがどのようなプログラムかというと、データとして文章の集合が与えられ、その中に含まれる各単語の出現回数をカウントするというものとなります。WordCountはHadoopにおけるHello Worldのような位置付けのプログラムです。しかし、実際にはそれだけではと

▼図6　MapReduceの動き



どまらずユーザごとの訪問回数、ページごとのアクセス数など、実用的な処理でも似たような実装になることが多い実用的なものでもあります。

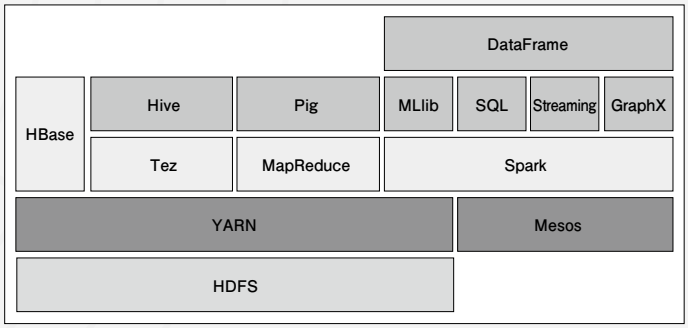
まずはMapperでの動きになります。Mapperでは各コンテナで自身の処理対象となるデータを1行ずつ読み込んでいきます。読み込んだデータは単語ごとに分割され、単語の数だけ（単語, 1）というkey-valueのペアを出力します。この際、1行に同じ単語が複数回出現する場合にはこのペアを複数個出力します[注4]。**図6**では、Map1、2、……Map nに相当する部分がMapperとなり、この場合n個のMap処理が並列で実行されます。

次に、ShuffleとSortというフェーズが入ります。このフェーズに関しては通常は利用者が実装する必要がない部分となります。Hadoopが内部的にこのフェーズを実行することで何をしているかというと、次のReducerの処理を行うために、すべてのMapperの出力で同一のkeyを持つデータを同一のReducerに送るための処理をしています。MapReduceではこのフェーズが唯一のネットワークを介した処理となります。

最後にReducerが実行されます。Reducerでは同一のkeyを持つすべてのデータに対するvalueのコレクションが渡されます。Mapperが先ほどのような出力の場合、（単語, (1, 1, 1, ……)）のような値が1レコードのデータとして渡されます。これに対してすべての値を足し合わせることで、全体での単語ごとの出現回数が計算できるというわけです。**図6**ではReduce1、……Reduce mに相当する部分がReducerとなり、この場合m個のReducer処理が並列で実行されます。

注4）ここでの処理を（単語, 出現回数）としても問題ありません。

▼図7　エコシステムを含めた技術スタック



以上がMapReduceの全体像となります。利用者はMapperとReducerを実装するのみという非常に単純な処理のみを行うフレームワークであることがわかったかと思います。しかし、実際には1回目のMapReduce処理の出力を2回目のMapReduce処理の入力にするなど多段構成にすることで、かなり複雑な処理も実装できるパワーを持っています。

Hadoopを支えるエコシステム

前節で説明したように、MapReduceは単純な処理となっており、多段に組み合わせていくことで複雑な処理を実現します。しかし、この際に組み合わせられるものはMapReduce処理のみであることから、どうやってそのフローに落とし込むかという点については従来のプログラミングと同じとはいかない場面も多くあります。

また、Hadoopはデータ分析によく利用されるということもあり、MapReduceを記述するためのJavaやプログラミングに関する知識がないデータ分析担当者などが利用したいシーンも多くあります。そのため、MapReduceを直接書かなくても利用できるようにさまざまなエコシステムと呼ばれるものが存在しています。ここではそのうちのいくつかを紹介します。

図3の技術スタックにHadoopのエコシステムを追加すると**図7**のようになります。実際にはHadoopのエコシステムはこれ以外にもたくさんありますが、ここで示しただけでもかなり

の数があることがわかるかと思います。

Hive

Hiveは、HiveQLというSQL風の言語を記述することでMapReduceの処理をラップしてくれるものとなります。古くからあり、現在でも非常に人気の高いエコシステムとなります。

Hiveの処理の流れとしては**図8**のように、

❶hiveコマンドまたはHiveServer2と呼ばれる、JDBC/ODBCから接続するためのクライアントからHiveQLを発行
❷HiveQLの構文解析と、メタストアにあるテーブル定義情報からMapReduceジョブを組み立て
❸Hadoop上でジョブを実行

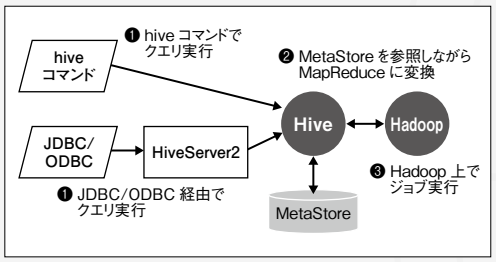
となります。

Hiveは一般的なRDBMSとは異なり、独自のフォーマットでデータを保持しているわけではなく、HDFS上にあるファイルに対してそのままテーブルの定義を行います。

そのしくみを実現するために、メタストアと呼ばれるテーブル定義に関する情報と、テーブルとHDFS上のデータの対応関係を定義したメタデータを持っているのが特徴になります。

メタストアは後述のSpark SQLなど多くのHive以降に登場したSQL風にHadoop上のデータを扱えるシステムからも利用でき、非常に汎用的かつ重要なコンポーネントの1つでもあります。

▼図8　Hiveの処理の流れ



Pig

Hiveとは別なアプローチでMapReduce処理をラップしたものに、Pigというものがあります。Hiveとの大きな違いはPig Latinという独自の言語を利用すること、テーブル定義が必要ないことの2点が挙げられます。

Pig Latinはデータに対する処理の手順を宣言的に記述していくもので、関数型のような記述形式になります。

また、データの読み込み時に区切り文字を指定したりするなど、事前の定義に基づいてデータを読み込むのではなく、読み込み時に利用者がデータ構造を指定することになります。

Pigは実行環境がHadoopに限定されておらず、開発環境でローカルモードで起動し、テストを行うことも可能な点もメリットとして挙げられます。

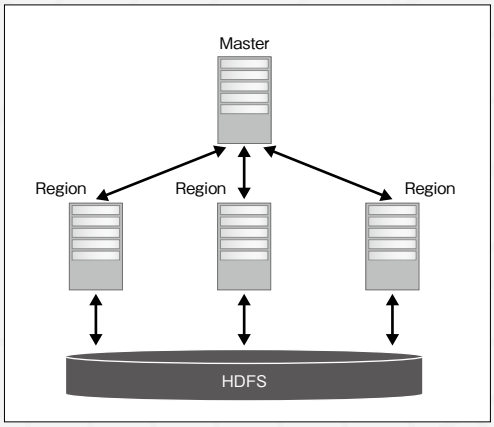
HBase

HBaseはHadoop上にあるデータを利用できるNoSQLとなります。

HBaseはマスタ／スレーブ構成になっており(**図9**)、リージョンサーバと呼ばれるスレーブノードが、実際に利用者がデータの取得などを行うコンポーネントになります。

マスタは各リージョンサーバへのリージョン

▼図9　HBaseの構成



（どのデータをどこに配置するか）の割り当てや障害時の復旧などを担当しますが、利用者がデータを取得する際にはリージョンサーバに直接アクセスするため負荷はそれほど高くはなりません。利用者はこのリージョンサーバのメモリ上にあるデータとHDFS上にあるデータに対して透過的にアクセス可能となっています。

HBaseはCAP定理でのCP型となっており、データの一貫性は保証されますが、ノード障害時などには復旧に時間を要します。

次世代MapReduce技術

先ほどは、MapReduceをラップし、より使いやすくしたライブラリについて紹介しました。次にHadoop上でMapReduceの代替となる技術として、TezとSparkを紹介します。

Tez

TezはMapReduceと同様に、Hadoop上で動くフレームワークとなります。YARNのApplication Masterが制御するアプリケーションは、必ずしもMapReduceである必要はありません。そのため、MapReduceの考え方を引き継ぎつつも、より効率的に処理を行えるようなしくみとして作られたのがTezとなります。

MapReduceではMapperとReducerの組み合わせのみであり、それを多段に組み立てることで複雑な処理も実現できると説明しました。このしくみは単純でわかりやすい一方で、多くのオーバーヘッドを伴う場合も多くあります。

たとえば多段に組み上げるときにMap→Reduce→Map→Reduceとなりますが、2つ目のMapが必要なく、入力をそのまま出力するだけという場合があります。MapReduceではしくみ上Mapのみというのはできますが、Reduceのみというのはできません。

また、MapReduceで機械学習のように繰り返し実行する必要のある処理を実装しようと思った際には、非常に多くのMapReduceが繰り返されることになります。その際、毎回HDFSから入力を読み込みHDFSに出力するという流れになり、その分のオーバーヘッドの繰り返し回数が多くなるほど全体での処理も遅くなってしまいます。

そこでMapやReduceのような処理を複数結合していき、DAG（Directed Acyclic Graph；有向非循環グラフ）という形式で処理のフローを定義するようにしたものがTezとなります。Tezを使うことでMap→Reduce→Reduceといった処理フローも実現可能になります。そのため、同じ結果を得るための処理をMapReduceで実装するよりもTezで実装したほうが多くの場合で高速化できます。

ただし、ApplicationMasterやNodeManagerのJVM起動などのコストはMapReduceと同様発生するため、ミリ秒のオーダで結果を返すなどトランザクション処理でそのまま利用できるほど高速化するというわけではないので、その点はご注意ください。MapReduceとTezの違いを示したのが**図10**になります。

このようにMapReduceを使うよりもメリットの多いように思えるTezですが、MapReduceやSpark（後述）と比較して知名度が高くないのはなぜでしょうか？　その理由として、実はTezを直接使って処理を実装するということはほとんどないことが挙げられます。

MapReduceを直接使うよりもエコシステム経由で利用する場面が増えてきたことから、Tezはそのまま使われるというよりは、HiveやPigといったエコシステムが実行時のエンジンとしてMapReduce以外にもTezを利用できるようになっており、利用者は意識することなくTezへの切り替えが可能になることを意識して作られています。そのため、Tezは既存のシステムとの親和性が高いとも言えます。

Spark

SparkもTezと同様MapReduceの代替として語られる場面の多いフレームワークです。

Sparkも処理のフローはDAGとして構成されます。また、可能な限りインメモリで処理を行うという特徴もあり、こちらもMapReduceと比較して高速に動作します。

Sparkは動作環境として、スタンドアローン(独自の分散処理)、Apache Mesos、そしてHadoopのYARNなどを選択できますが、ここでは**図7**のようなHadoop上でのMapReduceと同じレイヤのものとして解説します。

このようにSparkはHadoopのような分散処理環境そのものではなく、その上でアプリケーションを構築するためのフレームワークということになるため、“SparkとHadoopの違い”という意味では単純に比較できるものではありません。

SparkもTezと同様にHiveの実行エンジンとしての利用などもできますが、提供しているAPIが非常にシンプルであることやSpark自体が持っているライブラリの充実などもあり直接利用されることが多くなります。

SparkはRDD (Resilient Distributed Dataset)というイミュータブル(不変)で分散可能なコレクションに対する操作を行い最終的に結果を取得するという流れになります。

イミュータブルとはどういうことかというと、RDDの内容そのものは直接変更できず、操作を行ったあとの結果を新たな別のRDDとして生成するということになります。

また、分散可能とはデータ全体を1ヵ所で持っているわけではなくパーティションと呼ばれる単位に分割して、個々のノードでは自身の処理対象となるパーティションのみを保持するという形式になります。

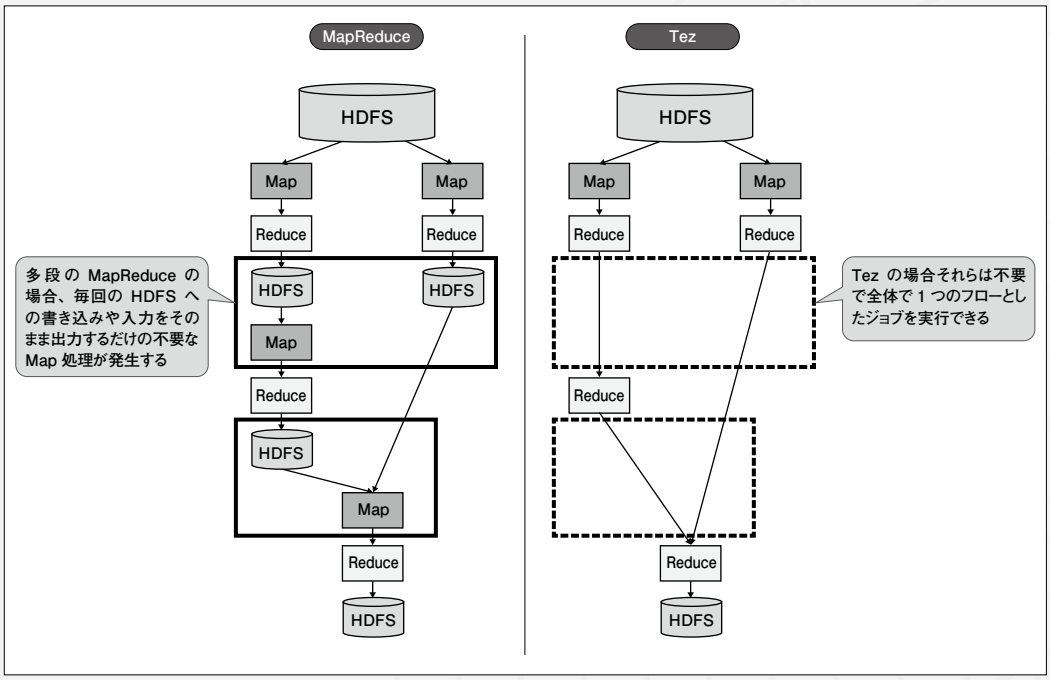
このSparkでの処理の流れを表したのが**図11**になります。

SparkのRDDに対する操作はまず大きくTransformとActionの2種類に分けられます。

Transformはmapやfilterといったもとの RDDから新たなRDDを得るための操作です。Transformはさらに狭い依存と広い依存の2つに分けられます。

狭い依存は、先ほども例に挙げたmapや

▼図10　MapReduceとTez



filterといった操作になります。これらは名前からもわかるかと思いますが、データの1レコードに対する操作のため、ほかのレコードの影響を受けずそこだけで完結します。

一方で、MapReduceのReducerに相当するreduceByKeyやjoinといった操作（広い依存）はほかのパーティションや異なるデータも必要になります。そのため、Sparkでは広い依存が発生する処理の単位を区切りとしてステージというものに分割します。狭い依存のみで構成されるDAGの一部分は個々のノードで処理されますが、広い依存が発生するタイミングでそれ以前の処理全体が完了するまでの待ちが発生するとともに、ネットワークをまたいだデータの転送が発生するわけです（**図12**）。

これらのTransform処理を終え、結果を取得するための操作はActionに該当します。ActionはRDDをScalaのコレクションに変換するcollectやデータ件数を取得するcountなどがあります。

SparkではこのActionに該当する処理が実行されるまでTransformは実行しないという遅延実行を行っています。Actionを実行するタイミングでそこに至るTransformからDAGを生成し、処理の実行を行うというわけです。実際の処理で生成されるDAGは**図12**のようになります。

Sparkを構成するライブラリ

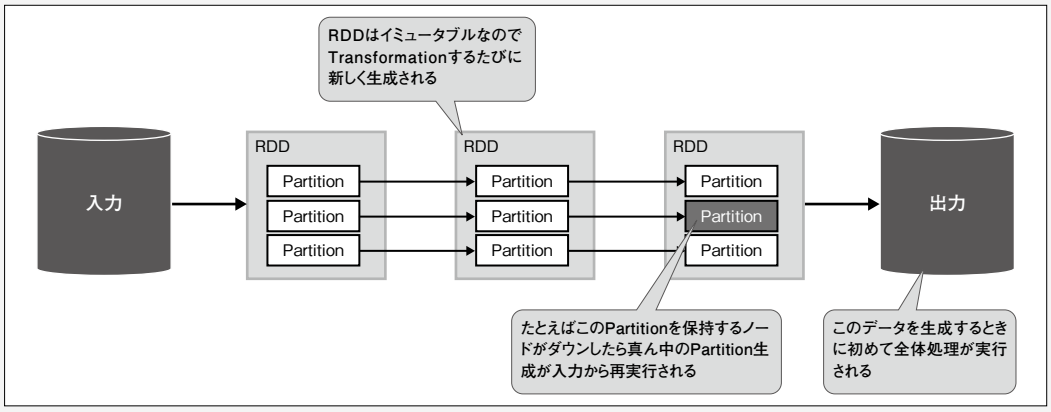
Sparkのもう1つの特徴として、標準で便利なライブラリがそろっていることが挙げられます。それらについて簡単に紹介します。

- MLlib
- SparkSQL、DataFrame、Dataset
- Spark Streaming
- GraphX

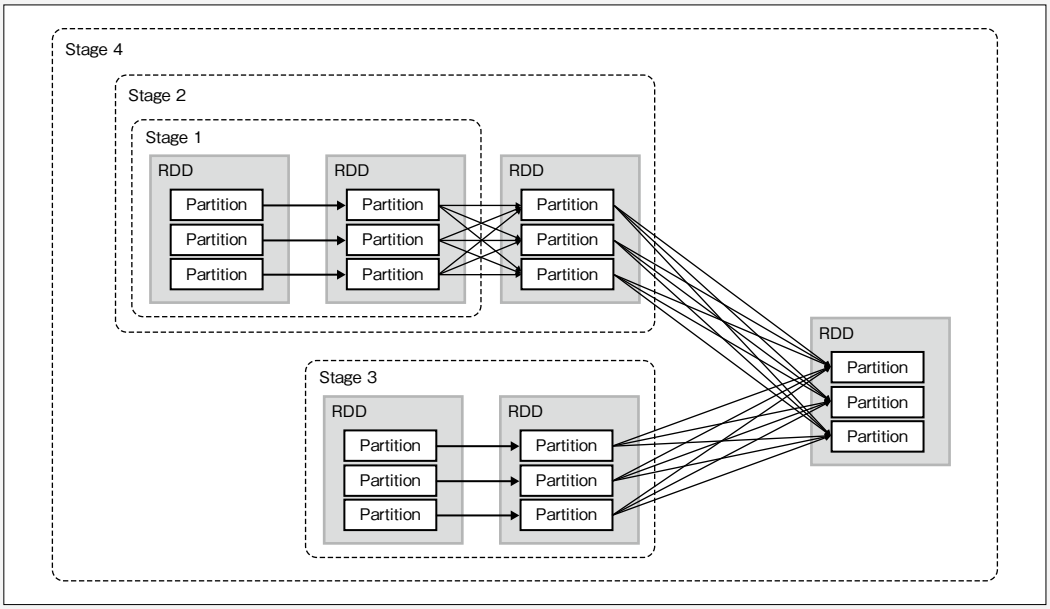
MLlibは機械学習のライブラリです。このライブラリの中には、さまざまな機械学習の実装が存在しており、現在でも非常に速いペースで次々に新たな実装が追加されています。Sparkが話題となった要因の1つにMapReduceと比較して繰り返し処理に強いという特徴があることから、機械学習との相性もよく、非常によく使われるライブラリとなっています。

SparkSQLはその名のとおりSQL風の言語で、データに対する操作を行うためのライブラリです。関連するものとしてDataFrame APIやDataset APIがあり、これらはデータフレームと呼ばれるSQLのテーブルやExcelの表のような形でデータを表すものになります。SparkSQLやデータフレームを利用することで、Hiveのメタストアを利用してのデータの取得やテーブル形式でのデータをそのまま扱うことができます。

▼図11　RDDの処理フロー



▼図12　SparkのDAG



データ分析では、SQLやデータフレームの形式でデータを扱えることの利便性は高く、またDAG生成の際に最適化も行うため、現在ではRDDを直接操作するのではなくDataFrame APIを利用する場面も多くなっています。

Spark Streamingはストリーム処理を扱うためのライブラリです。Spark Streamingでは、短い時間間隔で区切ったRDDを扱うようなイメージとなります。そのため、厳密には1件1件のデータをリアルタイムに扱うわけではない点には注意が必要ですが、バッチとストリーム両方に対して同じような方法で扱える点がその魅力となります。

GraphXはグラフ構造のデータに対する分析を扱うためのライブラリとなります。グラフデータとは、たとえばSNSの友達のようなノードとなる人と、エッジとなるその間の関連を表すものになります。ほかの3つに比べてGraphXの利用例はあまり多くありませんが、グラフデータの分析には独自の理論などがあるため、そういったものを扱いたい専門家にとって有用なライブラリです。

まとめ

本章ではHadoopとはどのようなものかについて解説しました。

ビッグデータやデータ分析とともに名前を聞くことも多くなっているHadoopですが、いざ使おうとするとどのように活用できるかわからないといった話も多く聞きます。

今回紹介できなかったものも含め、Hadoopにはさまざまなエコシステムが存在し、非常に多くの用途で利用できます。しかし、もちろんHadoopも万能ではないため、実際にはまずどのようにデータ活用をしたいのかという目的を明確にして、それに合わせた適切なツール選択が必要となります。

本章の内容が適切なツールの選択肢の1つにHadoopが加わるきっかけとなれば幸いです。SD

第4章

データ分析のためのクラウドサービス Treasure Data Service

Author 佐々木 海（ささき かい） トレジャーデータ㈱

現在では、データ分析基盤を一から作らずとも、クラウドサービスとして提供されています。その1つがTreasure Data Serviceです。サービスのベースとなっているのは、FluentdやHadoopですので、第2、3章で得た知識がさっそく役に立つはずです。すぐに始められるというクラウドサービスの利点を活かして、本章の内容を見つつ、さっそく試してみてはいかがでしょう。

トレジャーデータサービスとは

この章では具体的なデータ分析を行うクラウドサービスの1つであるTreasure Data Service（トレジャーデータサービス）について説明します。Treasure Data Serviceでは、第2、3章で紹介されたFluentdやHadoopを高可用で運用しているわけですが、どのようにそれを実現しているのか技術的な点から紹介します。

特徴

Treasure Data Serviceは米トレジャーデータ社が提供するデータ分析のためのクラウドプラットフォームです。米トレジャーデータ社は2011年にアメリカ、カリフォルニア州のマウンテンビューで創業されました。現在ではマウンテンビューと東京を中心として世界7ヵ国にメンバーがおり、世界中の多様なデータを集めています。

サービス開始以来、保存データ量、分析量ともに増加の一途をたどっており、2016年末時点で次のようなデータ量、クエリ数を管理するプラットフォームになっています。

- 保存されている総レコード数：1兆5000億レコード
- 秒間インポートレコード数：160万レコード
- 保存されている圧縮後総データ容量：50TB
- サービス開始以来の総実行クエリ数：1億1000万クエリ

Treasure Data Serviceはデータのインポート、エクスポート、分析、加工から機械学習、さらにはこれらを統合的に管理するワークフローエンジンまでを含んだデータ分析のためのフルマネージドサービスです。巨大なデータに対してこれらのサービスを安定して提供するためにTreasure Data Serviceはオープンソースソフトウェア（以下、OSS）を積極的に用いて開発されていることも大きな特徴です。

この章では、データ分析基盤の1つの選択肢としてTreasure Data Serviceがどのような課題を、技術的にどのように解決しているかという点に重きを置いて解説していきます。

アーキテクチャ

まずTreasure Data Serviceのアーキテクチャを概観します（**図1**）。Treasure Data Serviceはデータの統合、分析、連携までをワンストップで提供するフルマネージドサービスです。コンポーネントは大きく統合と分析の2つに分けられます[注1]。

注1）連携については広告配信、CRM、MAなどマーケティング施策を行うさまざまなアプリケーションと柔軟に接続することで、One to Oneマーケティングによる質の高いコミュニケーションを実現することができます。連携に関する技術的側面での説明はここでは省略します。

▼図1　Treasure Data Service概観

・統合（Unify）：データを集めて統合するための機能を担うコンポーネント
・分析（Analyze）：データを保管し、可視化・分析するコンポーネント

これらの機能がクラウド技術やOSSをベースに構築されています。それでは統合と分析のコンポーネントを具体的に見ていきます。

統合

統合のコンポーネントはさまざまな種類のデータをTreasure Data Serviceに入れて統合するためのレイヤで、データの入れ方は大きく分けて2通りあります。

・ストリーミングインポート
・バルクインポート

ストリーミングインポートはiOSやAndroidアプリなどリアルタイムにログを集めたい場合に使う方法で、ログが発生するとすぐにデータをTreasure Data Serviceに入れることができます。この機能を使いたい場合には、Treasure Data Serviceが提供するモバイルSDKやJavaScript SDK、あるいはFluentdを使うことになります。Fluentdは第2章で述べられたとおりプラグインによってインプットの方法を柔軟に変えられるので、さまざまなシステムからTreasure Data Serviceにストリーミングインポートでデータを入れることができます。

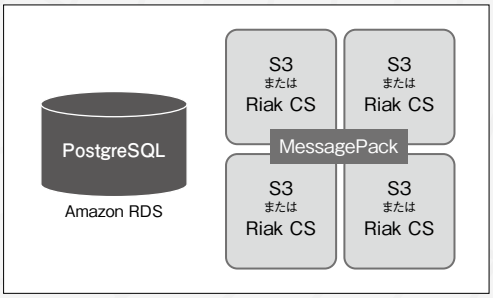
ストリーミングインポートが今まさに発生しているデータをリアルタイムに入れたい場合に向いているのに対して、すでにどこかに集められたデータをあらためてTreasure Data Serviceに入れ直す場合には、バルクインポートが向いています。バルクインポートは既存のデータをMessagePackベースの圧縮フォーマットにしてTreasure Data Serviceに入れます（MessagePackに関しては後述）。そのため、比較的大きなデータのインポートにも向いています。バルクインポートを利用するにはTreasure Data CLIを使うか、OSSとなっているEmbulkを使うことになります。具体的な方法については後述します。

分析

インポートされたデータはTreasure Data Serviceが独自に開発したRDBMSベースのPlazmaDBというストレージコンポーネントに格納されます。PlazmaDBは時刻ベースでのログデータの管理、分析を容易にしてくれる分散ストレージです。**図2**に概観を示します。

PlazmaDBはメタデータを保存するRDBMS（PostgreSQL）と実際のレコードデータを保存するオブジェクトストレージシステム（S3ま

▼図2　PlazmaDBの概観



▼表1　HiveとPrestoの特性

クエリエンジン	特性
Hive	Hadoop MapReduceベースのクエリエンジン。フォールトトレラントでHDFSベースで中間データを保存するので、巨大なデータセットに対しても安定してジョブを実行することが可能
Presto	Facebookが開発したオンメモリな分散クエリエンジン。中間データはメモリに乗せて処理させるため、リアルタイム性が必要なアドホックジョブの実行に向いている

たはRiak CS）を組み合わせた分散ストレージです。メタデータがRDBMSに保存されていることで、書き込み、更新操作をアトミックに行え、一貫した状態を保つことができます。

また、PlazmaDBに入っているデータは、独自のETL（Extract、Transform、Load）システムによりオブジェクトストレージに書き込まれた時点で、時刻ごとにパーティショニングされた列指向型のファイルフォーマットになっています。この列指向ファイルフォーマットはMessagePackをベースに作られており、圧縮と同時にPlazmaDBの大きな特徴である**Schema on Read**を可能としています。MessagePackはデータ型を持っているため、レコード自体に型情報が含まれています。そのため、あらかじめPlazmaDBに入れるデータは、各レコードの型情報やスキーマがわかっていなくてもとりあえず入れてしまい、読み出すときにデータ型のマッピングを決定してあげることが可能です。これがSchema on Readです。これは、データを利用したアプリケーションがまだ定まっていない場合や頻繁に仕様・要件が変更となることが想定される中でデータ分析を行うマーケタ、エンジニアにとっては、とても大事な機能です。

PlazmaDBに入れられたデータは、Treasure Data Service上で動くHive/Prestoでさらにデータの加工、分析を行えます。どちらもSQLベースの分散クエリエンジンで、若干の方言や独自のUDF（User Defined Function）はありますが、SQLに慣れ親しんだ方であれば簡単に使い始められます。なぜ2種類のクエリエンジンがあるのかというと、Treasure Data Serviceではユーザのワークロードに合わせてそれぞれのエンジンに合った使い方をしてもらっているためです（**表1**）。

つまり、リアルタイム性が重要視されるようなケース（例：BIツールのバックエンド）ではPrestoが、メモリに乗り切らないような巨大なデータを処理したい場合にはHiveが向いていると言えます。

Hiveでは各タスク、ジョブのステータスを管理し適切にリトライをすることでジョブの成功率を高めていますが、Presto自体にはその機能はありません。そもそも比較的短時間で終わるジョブにおいて、ジョブ内部のタスクの状態を管理して再計算などを行うよりも、ジョブをもう一度投げてしまったほうが実装もシンプルで安定するからなのかもしれません。

しかし、それではHiveとPrestoでユーザ体験が変わってしまうため、Treasure Data Serviceでは、サービスとしてPrestoを提供するために別段でキューシステムを開発しています。それが**PerfectQueue**です[注2]。PerfectQueueはRDBMSベースの分散キューで、次のことに重点を置いて開発されています。

・タスク実行時に失敗しても、ほかのキューがタスクを肩代わりして少なくとも一度は実行

注2）URL https://github.com/treasure-data/perfectqueue

▼表2　Treasure Data Serviceが利用しているおもなクラウドサービス

サービス	用途
Amazon EC2	HadoopやPrestoの各サーバインスタンス
Amazon RDS	PerfectQueue、PlazmaDBメタデータ
Amazon S3またはRiak CS on IDCFクラウド	ログデータの保存
Hosted Chef	各サーバインスタンスの構成管理

される
・クライアント側の障害によりクライアントが何度もジョブを投入しても、PerfectQueueは複数回実行を避ける
・複数キュー間でのFairスケジューリングを行う

このキューに対してユーザのジョブが投入されます。Treasure Data Serviceで稼働しているPerfectQueueアプリケーションは、エラータイプにより自動的にリトライすべきかどうかを判断してくれるので、もしPrestoでジョブが失敗しても一時的な障害だと判断されればリトライがかかるようになっています。この分散キューによりPrestoでのジョブの成功率はHiveと同程度になっています。

ここまででTreasure Data Serviceの概観を述べましたが、次項からこれらのコンポーネントがクラウドサービスにどのように支えられているかを具体的に述べたいと思います。

Treasure Data Serviceを支えるクラウドインフラ

Treasure Data Serviceは複数のクラウドに対応できるように作られており、Amazon Web ServicesやIDCフロンティアが提供するIDCFクラウド[注3]でも運用されています。Treasure Data Serviceを提供するうえで自分たちで開発、メンテナンスするコンポーネントは自分たちのコアビジネスに関わる部分に抑えて、それ以外は既存のクラウドサービス、PaaSを積極的に利用するようにしています。ここまで紹介したコンポーネントが利用しているおもなクラウドサービスを**表2**に記載しました。

大規模なデータを管理するプラットフォームを提供するうえで、クラウドサービスを利用するメリットは大きく2つあります。1つはインフラを任意のタイミングで廃棄できること。これはインスタンスの立ち上げや停止が簡単にできるということに加えて、ユーザのデータを既存の安定したストレージに集めることでその上で動くアプリケーションと分離できるためです。Treasure Data Serviceが動かしているHadoopやPrestoには永続化させなければならないデータは含まれていないため、マイグレーションやリリースのタイミングでいつでも廃棄、再作成ができるようになっています。

2つめは自動化がしやすいこと。HadoopやPrestoなどの分散システムは関わるインスタンスも多く、また外部コンポーネントとの依存も強くなりがちです。これらを手作業でインフラ管理から行うことはたいへんコストがかかります。Treasure Data Serviceではクラスタ作成、サービスディスカバリ、さらに各オペレーションがコードになっており、安全かつ簡単に作業を行えるようになっています。これらは各クラウドサービスがREST APIなど機械可読なインターフェースを提供しているからこそできることです。

利用しているオープンソースソフトウェア

Treasure Data Serviceでは積極的にOSSを活用し、またコミュニティへの参加も行っています。この項では、Treasure Data Serviceで利用しているOSSとその用途について簡単に紹介します。

MessagePack

MessagePack[注4]は米トレジャーデータ社の共

注3）URL https://www.idcf.jp/service/

注4）URL http://msgpack.org/

同創業者、古橋貞之によって開発されたバイナリシリアライゼーションフォーマットです。高速にシリアライズ、デシリアライズが可能で高い圧縮性能を兼ね備えています。現在50以上のプログラミング言語での実装があります。Treasure Data Serviceでは先述のとおり、お客様のログデータはMessagePackベースのフォーマットで格納することにより、Schema on Readと圧縮効率の良いストレージを実現しています。

Fluentd

Fluentdは第2章でも紹介されたように、簡単にログデータを集めるために作られたソフトウェアです。分散システムが多く動くTreasure Data Service内では、事象の分析、調査にシステムログは欠かせません。積極的にFluentdを使ってシステムログを集めています。各コンポーネントが生成したログデータは大きく分けて2通りの方法で収集されています。

・Fluentd → DataDog
・Fluentd → Treasure Data Service

1つめのDataDogはシステムモニタリングのためのPaaSで、Treasure Data Serviceではシステムログ、リソース使用のメトリック収集に利用しています。2つめはドッグフーディングも兼ねて自分たちのプラットフォーム、Treasure Data Serviceにデータを集めています。各ジョブの実行時間、ステータス、PlazmaDBのテーブルスキャンの効率などを定常的に集めて、これらに対してHive/Prestoクエリをかけて調査、分析を行っています。ここで得られた知見は開発や運用に継続的に活かされていきます。

Embulk

Fluentdが常に生成されるようなストリーム系のログデータを収集するのに向いているのに対して、比較的大きなデータを一度にインポートしたい場合にはEmbulkを用います。EmbulkもFluentdと同様OSSとなっており、プラグインを用いてデータソースと出力先を簡単に設定できます。現在オープンになっているものだけで約170のプラグインがあり[注5]、すでにここに必要なものが含まれていれば、新たな開発コストも少なくデータをインポートすることができます。

Embulkを利用してデータをTreasure Data Serviceにインポートするには、embulk-output-tdを出力用のプラグインに設定します。データをTreasure Data Serviceにインポートする際の詳細なやり方については後述します。

DigDag

DigDag[注6]は、米トレジャーデータ社が開発したデータのロード、クエリジョブの実行などのさまざまなタイプの定型的なタスクをパイプライン化し、管理するワークフローオートメーションシステムです。2016年にOSSとなりました。DigDagを利用することで複雑な依存関係のあるタスク群をシンプルに管理することができます。DigDagはTreasure Data Service内ではパッケージ化されたTreasure Work Flow（トレジャーワークフロー）として提供されています。

Hadoop/Presto

保存されているデータに対して分析を行う分散クエリエンジンとしては、先述のとおりHive on Hadoopか、Prestoを利用しています。データソースとしてPlazmaDBを利用するためのプラグインだけを開発して組み込んでいます。

Hivemall

Hadoop/Spark上で動く分散機械学習ライブラリです。HiveのUDFのコレクションとして実装

注5）URL http://www.embulk.org/plugins/
注6）URL https://www.digdag.io/

されており、SQLを使って機械学習のアルゴリズムを走らせることができます。Treasure Data Serviceで動くHiveには、Hivemallがデフォルトで組み込まれており[注7]、SQLが書ければ簡単に既存データ・セットに対して機械学習のアルゴリズムを試せます。昨年Apache Incubation Projectに選ばれ[注8]開発を加速させています。

Treasure Data Serviceの使い方

ここまでTreasure Data Serviceの全体像を紹介しましたが、具体的にどのようにデータを入れ、分析を進めるかを紹介していきます。まず必要なツールを紹介します。

ツール

Treasure Data CLI

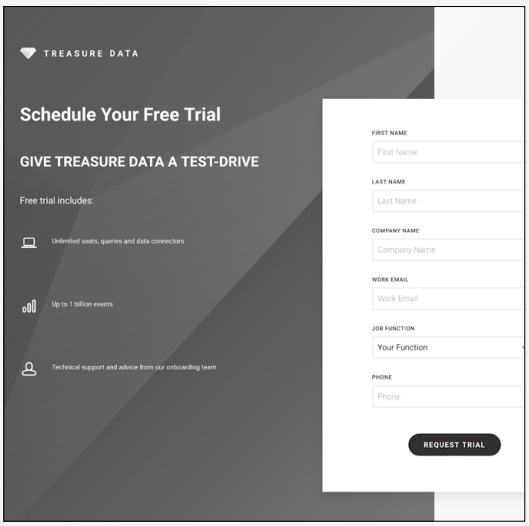
Treasure Data Serviceの各種リソースを操作するための基本的なコマンドラインツール[注9]になります。このツールを使ってデータのインポート、ジョブの実行、結果の取得などが行えます。macOSやWindowsなどさまざまプラットフォームに対応しています。詳細はインストールガイド[注10]をご覧いただくとして、ここではRubyGemsを使ったインストール方法を紹介します。

```
$ gem install td
$ td --version
0.15.2
```

Treasure Data CLIを使うには、Treasure Data Serviceのアカウントが必要です。まずは無料で利用可能なトライアルアカウントを作成します（**図3**）[注11]。

アカウントの作成が完了したらTreasure Data CLIで認証します。次のコマンドを入力します。

▼図3　トライアルアカウント登録画面




```
$ td -e https://api.treasuredata.com account -f
```

メールアドレスとパスワードを入力し認証が成功したら、API Keyが~/.td/td.confに書かれているはずです。

データをtdコマンドでTreasure Data Serviceにインポートするには、2通りの方法があります。

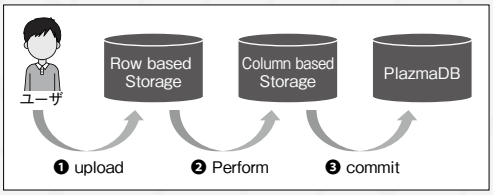
・バルクインポート
・Embulkインポート

バルクインポートは比較的大きなデータを安全に転送するためのしくみでimportサブコマンドに実装されています。これを利用するにはJava 6以上のランタイムが必要となります。バルクインポートでは安全にデータをインポートするためにいくつかの手順を踏む必要があります（**図4**）。

❶ upload：データを行ベースのフォーマットでTreasure Data Service上の一時ストレージにアップロードする
❷ perform：PlazmaDB上のフォーマットであるMessagePackベースの列指向フォーマットに変換する

注7）URL https://docs.treasuredata.com/articles/udfs#hivemall-generic-udfs
注8）URL http://incubator.apache.org/projects/hivemall.html
注9）URL https://docs.treasuredata.com/articles/command-line
注10）URL https://docs.treasuredata.com/articles/installing-the-cli
注11）URL https://www.treasuredata.com/request_trial/

▼図4　バルクインポートの手順



▼図5　サンプルCSVファイル

```
$ cat sample.csv
time,path,code,method
1412385195,/item/sports/2511,200,GET
1412385194,/category/electronics,200,GET
1412385192,/category/jewelry,200,GET
```

▼図6　import:listコマンドで、sessionが作成されたことを確認

```
$ td import:list
+------------+--------------------+-----------+--------+-------+--
| Name       | Table              | Status    | Frozen | JobID | ...
+------------+--------------------+-----------+--------+-------+--
| my_session | my_db.my_tbl       | Uploading |        |       | ...
+------------+--------------------+-----------+--------+-------+--
1 row in set
```

▼図7　prepareコマンドで*.msgpack.gz形式に変換

```
$ td import:prepare sample.csv \
    --format csv \
    --columns time,path,code,method \
    --time-column 'time' \
    -o ./uploadable
```

❸ commit：あらためてPlazmaDB上にアップロードし、メタデータをatomicに変更する

このように手順を踏むことで、データをTreasure Data Serviceに安全にアップロードすることができます。バルクインポートはさまざまなデータソース、ファイルフォーマットのアップロードに対応していますが[注12]、ここでは例として、**図5**のようなCSVファイルのインポートを試してみます。

バルクインポートの一連の状態はsessionの下に管理され、参照することができます。そのためバルクインポートを行うには、まずsessionを作成します。

```
$ td import:create my_session my_db ↵
my_tbl
Bulk Import session 'my_session' is ↵
created.
```

引数にはsession名、アップロード先のデータベース名、テーブル名を指定します[注13]。sessionが作成されたことはimport:listコマンドで確認できます（**図6**）。

Row based StorageへのアップロードはMessagePack.gzというMessagePackを圧縮した形式に変換する必要があります。これはprepareコマンドで行えます（**図7**）。

uploadableディレクトリの下に圧縮した*.msgpack.gz形式のデータができたはずです。これを先ほどのsessionに対してアップロードします。

```
$ td import:upload my_session ./uploadable/*
```

このアップロードの大きな特徴は、失敗した場合に何度でもリトライすることができる点です。なぜなら同じファイルのアップロードを重複排除させるしくみがあるためです。

アップロードされた*.msgpack.gzファイルはimport:showコマンドで確認できます（**図8**）。

次に、アップロードされたファイルを列指向フォーマットに変換させます。アトミックにファ

注12）URL https://docs.treasuredata.com/categories/bulk-import

注13）sessionを作成するには、以下のコマンドで事前にTreasure Data Service上にデータベースとテーブルを作成しておく必要があります。
【データベース作成】$ td db:create my_db
【テーブル作成】　$ td table:create my_db my_tbl

▼図8　import:showコマンドでファイルを確認

```
$ td import:show my_session
Name          : my_session
Database      : my_db
Table         : my_tbl
Status        : Uploading
Frozen        : false
JobID         :
Valid Records:
Error Records:
Valid Parts   :
Error Parts   :
Uploaded Parts :
  sample_csv_0_msgpack_gz
```

▼図10　td-agentのインストール

```
$ curl -L https://toolbelt.treasuredata➥
.com/sh/install-redhat-td-agent2.sh | sh
$ /etc/init.d/td-agent start
```

イル変換を実行させるため、sessionを凍結したあと変換します。

```
$ td import:freeze my_session
$ td import:perform my_session --wait
```

無事変換が完了したら、最後にcommitを行い、PlazmaDBから見えるようにします。使い終わったsessionは削除します。

```
$ td import:commit my_session --wait
$ td import:delete my_session
```

内部的にはこのようにしてデータを安全にインポートしているのですが、いくつもコマンドを打つのは面倒なので、pipelineモードがあります。**図9**のコマンドは先ほどの流れを一度にやってしまいます。比較的小さなデータであれば、pipelineモードを使うのが良いと思います。

もう1つEmbulkを使ったデータのインポートがあります。こちらはEmbulkのアウトプットプラグインとしてembulk-output-tdを利用します[注14]。Embulkにすでに慣れている方は、こちらのほうが親しみやすいかもしれません。Embulkの使い方の詳細はここでは割愛します。

▼図9　pipelineモードによるバルクインポート

```
$ td import:upload \
  --auto-create my_db.my_tbl \
  --auto-perform \
  --auto-commit \
  --column-header \
  --output uploadable \
  sample.csv
```

▼リスト1　td-agentの設定ファイルの例

```
# Treasure Data output
<match td.*.*>
  type tdlog
  endpoint api-import.treasuredata.com
  apikey <API Keyを記載します>
  auto_create_table
  buffer_type file
  buffer_path /var/log/td-agent/buffer/td
  use_ssl true
  num_threads 8
</match>
```

td-agent

td-agentはWebサービスなどに組み込んで、常に生成されるログをTreasure Data Serviceに送るためのシステムです。td-agentは、Fluentdの安定版を主要なプラットフォーム用にパッケージングしたもので、基本的な使い方はFluentdと同じです。たとえば、RPMパッケージを使ってインストールするには**図10**のようにします。

設定ファイルは/etc/td-agent/td-agent.confとなります。詳細な使い方はFluentdと同じで、Treasure Data Serviceにインポートするためにはfluent-plugin-tdを使います。設定ファイルに**リスト1**のように記載します。

Fluentdの詳細な使い方はドキュメント[注15]を参照ください。

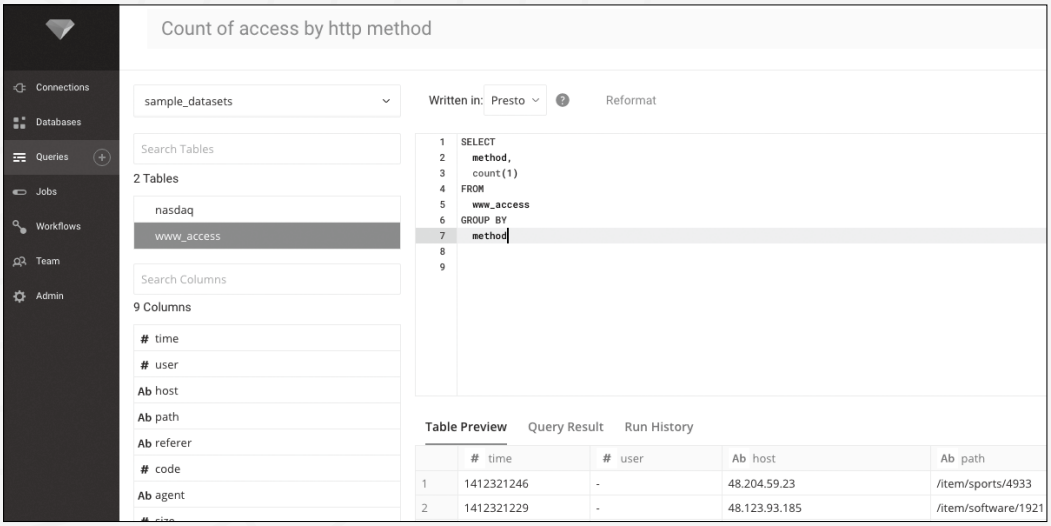
TD Console

Treasure Data ServiceをWebサービスとして利用する場合には、TD Consoleが便利です。TD Consoleはテーブルやジョブの参照、クエリエディタ、アカウントの各種統計情報を搭載

注14）URL https://github.com/treasure-data/embulk-output-td

注15）URL http://docs.fluentd.org/articles/quickstart

▼図11　TD Console（画面の一部）



したWebサービスです（**図11**）。利用にはTreasure Data ServiceのアカウントとWebブラウザが必要になります。アクセス先は「http://console.treasuredata.com」です。

利用できる機能の多くはTreasure Data CLIと重複していますが、普段CUIを利用しない方はTD Consoleが便利です。

Data Connector

Data Connectorは、Treasure Data Serviceが直接データソースからデータを取ってくるシステムです。ユーザは必要な認証情報などを入力するだけで、Treasure Data Serviceにデータを移行できます。たとえば、FTPを使ってデータをつなげる場合は**図12**のように認証情報を入れ、Connectorを作成します。

このConnectorを使ってデータをTreasure Data Serviceにいつでも移行させられます。現在70以上のサービス、プラットフォームに対してConnectorが用意されています。

機械学習（Apache Hivemall）

Hivemallは、SQLで記述できるスケーラビリティの高い機械学習ライブラリです。Treasure Data ServiceではデフォルトでHivemallがサポートされており、普段SQLを書くデータアナリスト、エンジニアが同じインターフェースで機械学習アルゴリズムを記述することができます。現在、次のようなアルゴリズムがサポートされています。

- **Feature Engineering**
  Scaling、TF-IDF、Feature Selection
- **Regression**
  Logistics Regression、Passive Aggressive、AROW Regression、
  Random Forest Regressor
- **Classification**
  Perceptron、AdaGrad、
  Confidence Weighted、
  Random Forest Classifier
- **Recommendation**
  Matrix Factorization、
  Factorization Machine

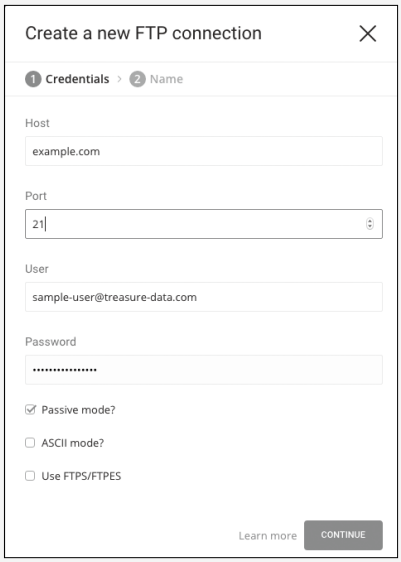
たとえば、二値分類のためLogistics Regressionを走らせる場合は、**図13**のようにします。

`training_data`は、`features`と`label`カラムを持つテーブルでこれを訓練データとしてモデルを学習させます。Hivemallのlogress UDFは出力としてモデルのパラメタを書き出すので、モデルのエクスポートをHiveテーブ

▼表3　Treasure Data Service、Amazon EMR、Google BigQueryの比較

機能	Treasure Data Service	Amazon EMR	Google BigQuery
データのバルクロード	バルクインポート／Embulk	Apache Sqoop、Embulkなど	bqコマンド、Web UI
データのストリーミング	td-agent、Fluentd	Fluentd	BigQueryクライアントライブラリ
クエリエンジン	Hive、Presto	Hive、Impalaなど	BigQuery
機械学習	Hivemall	Spark MLlibなど	Prediction API

▼図12　Data Connector（FTPでデータを取ってくる例）



▼図13　Logistics Regressionを実行させる場合の入力例

```
select
 cast(feature as int) as feature,
 avg(weight) as weight
from
 (select
     logress(
       addBias(features),
       label,
       "-total_steps ${total_steps}")
     as (feature,weight)
  from
     training_data
 ) t
group by feature;
```

ルへの書き出しとして行うことができます。

Hivemallは昨年Apache Incubatorプロジェクトにも選ばれOSSとしても積極的に開発が進められています。

他クラウドサービスとの比較

データ分析のためのクラウドサービスはほかにも多くあります。ここではおもに機能面で、Amazon Elastic MapReduce（EMR）、Google BigQueryと比較してみたいと思います（**表3**）。

Treasure Data Service、Google BigQueryはマネージドサービスで、基本的に自分でデータベースやインスタンスの管理をする必要はありません。対してAmazon EMRは素のHadoopのディストリビューションを使うため、自分で管理、運用を行う必要がありますが、柔軟にHadoopエコシステムのソフトウェアをインストールすることができます。

このほかにも、もちろん課金体系やユーザのワークロードに合わせたパフォーマンスの違いがありますので、ご利用の際はぜひ比べてみてください。

まとめ

クラウドを使ったデータ分析基盤の1つとしてTreasure Data Serviceを紹介しました。ここで記載しきれなかった情報も多くありますので、ぜひ公式ドキュメント[注16]を参照していただけたらと思います。クラウド技術の安定化、成長に伴い、データ分析のためのプラットフォームの形もかつてのオンプレミスでのシステムから大きく変わってきました。みなさんの要件、ニーズに合わせて最適な分析基盤を選定、構築するためのヒントになれば幸いです。SD

注16）URL http://docs.treasuredata.com/

BACKNUMBER バックナンバーのお知らせ

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

本誌バックナンバー（紙版）はお近くの書店に注文されるか、下記のバックナンバー常備取り扱い店にてお求めいただけます。また、「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）や、e-hon（http://www.e-hon.ne.jp）にて、Webから注文し、ご自宅やお近くの書店へお届けするサービスもございます。

2017年2月号

第1特集
なぜプログラマの役に立つのか
いまはじめるDocker

第2特集
Linuxファイルシステムの教科書
Ext3、Ext4、XFS、F2FS、Btrfsの特徴と進化

第3特集
エンジニアが採用できない会社と評価されないエンジニア（後編）

定価（本体1,220円+税）

2017年1月号

第1特集
bash書き初め
シェル30本ノック

第2特集
機械学習をどう学ぶべきか?
数学とコンピュータのつなぎ方

第3特集
エンジニアが採用できない会社と評価されないエンジニア（前編）

定価（本体1,220円+税）

2016年12月号

第1特集
NoSQLの教科書
MongoDB、Couchbase、Redis、MySQLでNoSQL!

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

特別企画
先人たちの知恵と足跡に学ぶ
温故知新 ITむかしばなしSP

定価（本体1,220円+税）

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）

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。

DIGITAL デジタル版のお知らせ

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%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。

第2特集

実践! DevOps

CIでインフラ開発を効率化

Jenkins＋Gerrit＋Ansible＋Serverspecで基盤構築

CI（Continuous Integration：継続的インテグレーション）とは、テストを自動化して高頻度のリリースを目指す、ソフトウェアにおける新しい開発手法のひとつです。本特集では、そのCIをインフラの領域にも広げ、運用と開発全体を効率化したチームのメンバーたちが、「インフラにおけるCI」について、基本から解説します。

第1章では実践に移る前に、「アジャイル」「スクラム」「CI」というキーワードについて、概要を押さえます。第2章ではスクラムの実践として、要件定義→計画→実装→テスト→評価→振り返りという、実際の開発の流れを追います。第3章ではCIの実践として、Jenkinsの設定紹介をメインに、AWS EC2＋Gerrit＋Ansible＋Serverspecを利用した環境を構築します。

実践前に基礎知識を押さえる

第1章 インフラをアジャイルに開発するためのしくみ

Author 梅森 直人（うめもり なおと） Twitter @unetto ㈱NTTデータ
Author 武田 勝輝（たけだ しょうき） ㈱NTTデータ

「短い期間を設けて少しずつ機能を実現しながら完成に近づける」というソフトウェアの開発手法を、インフラの領域にも適用することで、得られるメリットがあります。本章では、その実現における重要なキーワードである「アジャイル」「スクラム」「CI」について解説していきます。

はじめに

こんにちは。みなさんは「アジャイル開発」と聞いて何を連想されますか？　スクラム、Continuous Integration（以下CI）、スプリント、JIRA[注1]などなど、いろんなワードが思い浮かぶことでしょう。

本稿では、アジャイル開発やテスト自動化にまつわる基礎知識を整理していきます。具体的には、アジャイルとその一開発手法であるスクラムや、CIとは何かについて振り返り、これらをインフラ領域になぜ適用したのかを紹介します。

本稿が、アジャイル開発の基礎とその適用先を広げるきっかけとして、少しでも役に立てれば幸いです。

「インフラをアジャイルに開発する」とは

そもそもインフラとはどのような領域のことを指すのでしょうか。読者のみなさんが想像されるイメージは、それぞれ異なるかと思います。本稿では**図1**の点線で囲まれた、筆者たちがアジャイルの考え方で開発を行っていた範囲をインフラと呼ぶことにします。そして、インフラをアジャイルに開発するとは、「ソフトウェア領域におけるアジャイル開発のお作法に従って、ミドルウェアからサーバやネットワークまでを構築、管理すること」であると定義します。

仮想マシンや仮想ネットワークなどの生成自動化、各種ツールやライブラリ、ミドルウェアなどのインストール・設定自動化、設定確認テストの自動化を行うといったInfrastructure as Codeの考え方はこれまでにもありましたが、これをアプリケーション開発と同期しながら実行することが重要です。フルスタック領域でのシステム開発は、インフラ領域とアプリケーション領域が噛み合ってこそ意味があるものですから、お互いが同期化されるタイミングが計画されていなければなりません。

では、どうやって計画すれば良いのでしょうか。検討のベースラインとして適しているのがアジャイル開発の考え方であり、計画実行のう

▼図1　本稿で扱うインフラの範囲

注1）URL https://jira.atlassian.com/

えで適しているのがアジャイル開発の実現手法であるスクラムです。具体的な計画や実現プロセスに関しては第2章と第3章で紹介しますが、まずはアジャイル、スクラム、そしてCIとはそもそも何かについておさらいしてみましょう。

アジャイル開発とは

概要

ソフトウェア開発の現場では、開発の初期段階で要求・仕様を決定し、数ヵ月から数年の長い時間をかけて品質の高いソフトウェアを開発するというウォーターフォール型が主流でした。しかし近年のビジネス環境や市場環境が変化するスピードの加速に伴い、ウォーターフォール型では、顧客へソフトウェアがリリースされるときにはすでに、ビジネスのトレンドや顧客の要求が変化してしまうといった事態が起こるようになってきました。結果、開発したソフトウェアの価値が著しく低下するというリスクが発生するようになりました。

そこで、より迅速に、より顧客のニーズに合致した価値の高いソフトウェアを開発する手法が必要とされるようになってきました。そのような要望に対する1つの解となるのが、アジャイル開発です。

アジャイル開発の始まりは2001年2月。Kent Beck氏やJeff Sutherland氏を始めとした17名の著名なソフトウェアエンジニアたちが、より良いソフトウェア開発手法を検討するために集まり、そこでまとめられた「アジャイルソフトウェア開発宣言」[注2]が基となっています。そこには「顧客満足」を最も優先する、価値の高いソフトウェアを開発するための原則などが書かれています。

「アジャイル」とは「すばやい」「俊敏な」といった意味の言葉で、短い開発期間の中で価値が高いプロダクトを漸進的に開発していくのがアジャイル開発の特徴です。アジャイル開発では、数日から数週間の短い期間で、小さなスコープで要求定義から設計、実装、テスト、リリースを行い、これを複数回繰り返しながら開発を進めていきます。メリットとして、優先度の高い重要な機能から開発に着手できること、仕様の間違いや要求漏れに早い段階で気づけることなどが挙げられます。

アジャイル開発を実現する手法

アジャイル開発宣言では、顧客満足を最大化するためのソフトウェア開発の考え方や原則のみが記述されており、一方で具体的な開発手法は言及されていません。アジャイル開発を実現するための手法は世の中にたくさん出ていますが、その中で最も普及している手法の1つが「スクラム」です。

スクラムは、1986年に経営学者である野中郁次郎氏と竹内弘高氏が日本企業のベストプラクティスを研究し、論文としてまとめた「The New New Product Development Game」内で紹介された開発手法が基になっています。現在では、「スクラムガイド」[注3]としてまとめられ、30ヵ国語を超える言語に翻訳されたうえで、無料で公開されています。

また次章以降で詳細に説明しますが、スクラムはプロセス(開発工程)に関する手法であり、アジャイル開発を実現するための代表的なプラクティス(開発手法)としては、「エクストリーム・プログラミング(以下XP)」があります。XPは迅速に高品質・高価値なソフトウェア開発を実現するためのプラクティスの集合であり、その中にはたとえば、実装を行うより先にテストを作成する「テスト駆動開発」、2人一組になって1人がコードを書き、もう1人がリアルタイムでコードレビューを実施する「ペアプログラミング」など、計17のプラクティスが含まれてい

注2)「アジャイルソフトウェア開発宣言」
URL http://agilemanifesto.org/iso/ja/manifesto.html

注3)「スクラムガイド」URL http://www.scrumguides.org/

ます。

世の中の多くのアジャイル開発プロジェクトでは、プロセスとしてスクラム、プラクティスとしてXPが採用されています。

スクラムとは

概要

スクラムとは、複雑なプロダクトを開発・維持するためのプロセスフレームワークであり、可能な限り価値の高いプロダクトを、生産的かつ創造的に届けることを目的としています。スクラムでは、実際の経験と既知の事実に基づいて予測可能性の最適化とリスク管理を行います。このプロセスの実現は次の3つの柱に支えられます。

- **透明性**
  プロセスの用語を参加者全員で共有し、また、作業をする人とその作成物を受け取る人が「完成」の定義を共有する
- **検査**
  スクラムの成果物や進捗を頻繁に検査し、計画・予測からの変化を検知する
- **適応**
  検査の結果、プロセスの不備が許容値を超え、成果となるプロダクトが受け入れられないと検査人が判断した場合は、プロセスやその構成要素を調整する。調整はできるだけ早く行い、これ以上の逸脱を防がなければいけない

これら3つの柱を実現するために、プロセスフレームワークであるスクラムでは次の3つの要素を定義しています。

・ロール
・イベント
・成果物

スクラムのロール

スクラムでは、「プロダクトオーナー」「開発チーム」「スクラムマスター」の3つのロール(役割)で成り立つチームで開発を進めます。

プロダクトオーナーは、プロダクトの要求定義、受け入れ判断を行う人です。原則チームに1人です。

開発チームは、実際に手を動かし、プロダクトを開発します。3名から9名で構成され、プログラマ、テスター、設計者などの役割はなく、目標達成のため自由に行動できます。

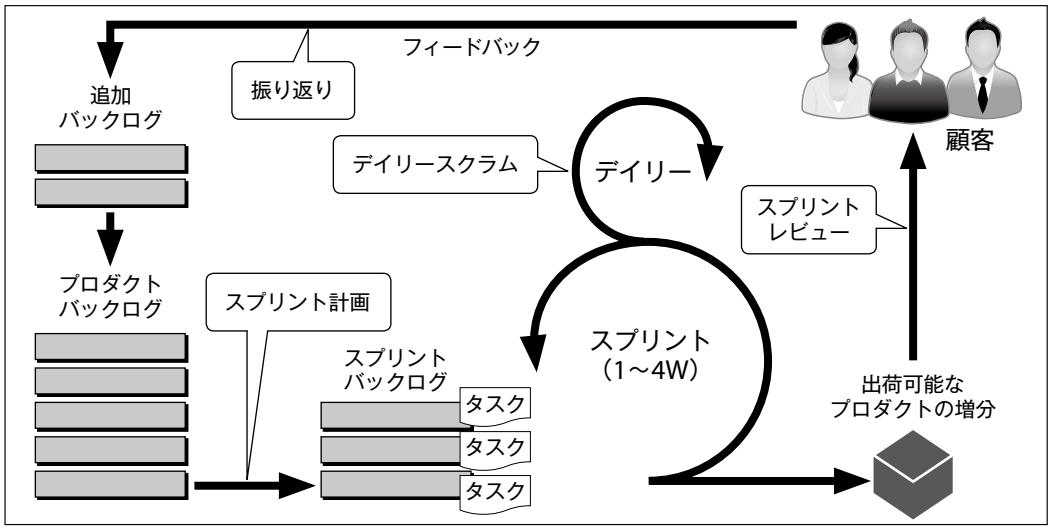
スクラムマスターは、スクラムチームが最高の生産性を保てるようにプロダクトオーナーと開発チームを支援します。

スクラムのイベントと成果物

スクラムチームは、「スプリント」と呼ばれる1週間から1ヵ月の固定された開発期間を繰り返していきます(**図2**)。スプリント中は外部からの新しい変更・要求を受け入れません。スプリントの長さは、開発チームが成果を出すまでにかかる最短期間、かつ開発するプロダクトの要求追加や変更を我慢できる最短期間が設定されるため、プロジェクトや開発チームによって変わります。

スプリント内では、固定された時間のイベントが明確に定義されています。まずスプリントの初めに「スプリント計画」を行います。スプリント計画は一般的に2部制で実施されます。前半ではプロダクトオーナーが、自身が作成した要求定義のリストである「プロダクトバックログ」の内容をスクラムチームへ説明します。そして開発チームは、各バックログの項目に対して見積もりを行い、当該スプリントでどこまで開発するかを計画します。そして後半では、計画された各バックログ項目を、具体的にどのように実現するかについて作業計画を立て、タスク一覧を「スプリントバックログ」として見える化します。

▼図2　スクラムによる開発の全体像



スプリント計画が終わりしだい、実際の開発が始まります。開発チームは毎日「デイリースクラム」と呼ばれるイベントを実施します。これは、決まった時間に決まった場所に集まり、15分間を目安に、問題の顕在化とその問題を改善するための再計画をします。日々、成果物や進捗がゴールに向かっているかを検査、適応させながら「出荷可能なプロダクトの増分」を開発していきます。

そしてスプリント期間が終わる前に、スプリントで完了した動くソフトウェアのデモを、プロダクトオーナーや顧客、そのほかのステークホルダーに対して実施します。これは、「スプリントレビュー」と呼ばれるイベントで、このソフトウェアのデモを通じて、プロダクトに対するフィードバックを引き出します。

スプリントレビューが終わったら、最後のイベントである「振り返り」を実施します。ここでは、開発チームがスプリントの問題・課題を洗い出し、それに対する改善計画を立案します。

スクラムのメリット

スクラムの大きなメリットとして、従来型のウォーターフォール型に比べて、無駄なプロダクトを作るリスクを減らせることが挙げられます。スクラムでは、動くソフトウェアをすばやく作ることに重きを置いています。動くソフトウェアをプロダクトオーナーやユーザにデモすることによって、適切なフィードバックを早い段階で得ることができるからです。

ユーザストーリー以外のタスクの扱い

開発を進めていると、対応が必要なものの、プロダクトバックログの実現とは関連が薄いような作業が発生します。その最たるものとして、たとえば「技術的負債」への対応が挙げられます。

技術的負債とは、自動化されていないテスト、解決されていないコーディング規約違反など、開発の中で先送りにされた課題のことです。こういった作業は、プロダクトバックログからブレークダウンされるものではありませんが、「透明性」の原則に従い、見える化される必要があります。したがって通常スクラムでは、こういったタスクもスプリントバックログとして列挙し、プロダクトバックログから分解されたタスクと同様、予測時間の見積もりと実績時間の管理を実施します。

スクラムの注意点

スクラムは、ユーザの要求などが不確定なプロダクトに対して最大限効果を発揮します。一方で、仕様が決まっているプロダクトに対しては、ウォーターフォール型で開発するほうが開発コストが少なくて済むということを忘れてはいけません。

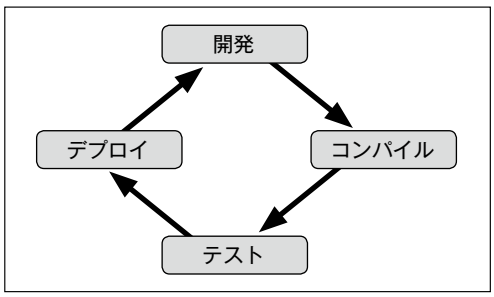
スクラムは、1つのプロダクトに対して少しずつ機能を追加していきます。このため、機能を追加していくごとに回帰テストが必要になります。スプリントが増えていくごとにこの回帰テストのボリュームは増えるので、自動テストが基本になります。自動テストのコード量が増えることを考えると、仕様が定まっているプロダクトに関してはウォーターフォールのほうが効率的であると言えます。

CIとは

概要

CIとは、コードの構築・テストを継続的に実行する習慣です。日本語では継続的インテグレーションとも呼ばれるXPのプラクティスの1つで、その特徴は**図3**のような循環型のビルドプロセスにあると言えるでしょう。短期間で何度もビルドを実行し、ソフトウェアを結合したときに発生する不具合を早期に検出し、改修を速やかに行うことで、ソフトウェア開発のQCD[注4]を向上させることが狙いです。

▼図3　CIのビルドプロセス



CIのメリットと注意点

次に、CIのメリットと盲点になりがちな注意点について**表1**に記します。

CIの一番のメリットは①で、問題の早期発見ができる点にあります。非CI開発では、テストまでに複数のコミットを経ることになるため、テスト時に問題の切り分けが難しくなります。CI開発ではコミットごとにテストを行うため、どこでバグを埋め込んだのかという原因解析とその解決が容易になります(**図4**)。

一方で、盲点となりがちなのが注意点の③です。たとえばテストを自動化していると、どうしても自動化が難しい、もしくはできないものが出てきます。この非自動化対象となった作業分のコストがかさむため、注意が必要です。

インフラにアジャイルを適用した理由

さて、そもそも筆者たちがなぜインフラにアジャイルを適用しようと思ったのか。最後に、その主な2つの理由について簡単に紹介します。

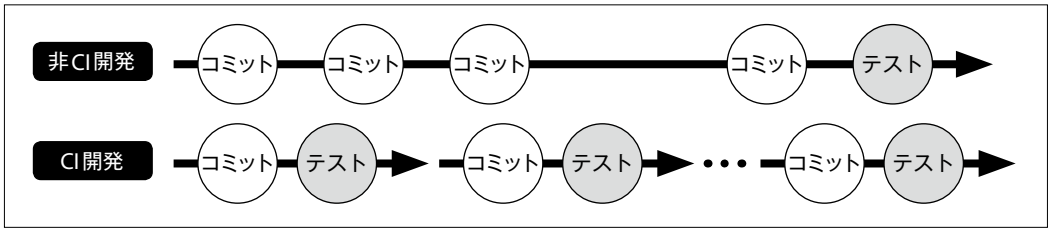
理由1：機能・非機能の要件が定まっていないプロジェクトで、プロトタイピングを通じた仮

▼表1　CIのメリットと落とし穴

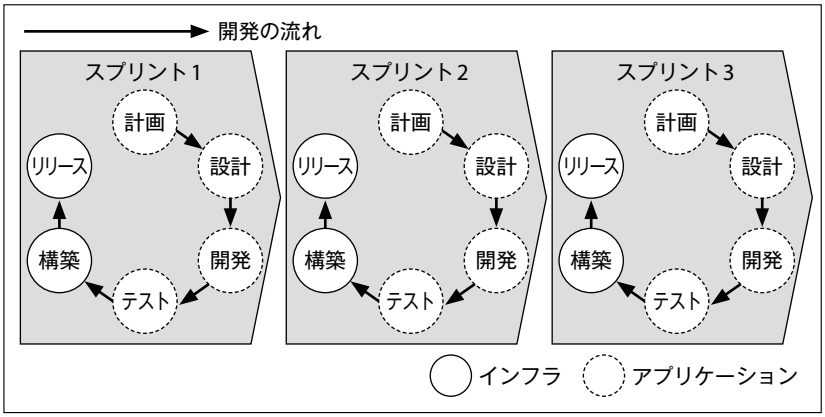
メリット	①ソフトウェア結合時の問題を早期に発見できる
	②開発・テストの流れがルーティン化されるので、作業の自動化がしやすい
	③ビルド実行履歴データが蓄積されるので、継続的なプロセス改善活動ができる
注意点	①しくみの導入コストがかかる(ツール選定および構築、コーディング規約、成果物品質基準の制定、ワークロードの検討・実施、テスト・開発環境数)
	②開発拠点間に時差が存在する場合、Jenkinsなどのバッチ処理実行時間帯が限られる
	③作業を自動化する領域と自動化しない領域の見極めが適応的になりがちで、自動化対象外となった作業のコストがかさむ

注4) Quality(品質)、Cost(費用)、Delivery(納期)の頭文字をとったもの。

▼図4　非CI開発とCI開発の違い



▼図5　スクラム開発のサイクル



説検証を繰り返し行う必要があったから

実現すべき機能と非機能を模索しながら仮説検証を進める、いわば試験研究のようなプロジェクトでしたので、開発開始時にゴールを定めることもできません。そのため、短期間で継続的にプロトタイピングを実施・評価し、進む方向を都度軌道修正できる開発手法を採る必要がありました。結果として、アジャイルの開発手法のうち導入が容易なスクラムを選択しました。

理由2：アプリケーション都合による構成変更がアジャイルに発生するため、確認作業も合わせて行わなければならないから

手元の環境でアジャイルにアプリケーションを開発し、手元の環境で動いているから大丈夫、というわけにはいきません。実際は、試験環境で動作確認が取れ、テストを終えて初めて完了となります(**図5**)。たとえば、パッケージのリビジョンがサーバ間で異なっていないか、期待どおりの通信経路上で電文が流れているかなど、アプリケーションを実環境へデプロイして動作させるには多くの確認作業が必要になります。この確認作業は、アプリケーション都合による構成変更が発生するたびに行う必要があります。確認作業を手動で毎回行うのは非効率ですので、自動化して実行させるようにしました。SD

アジャイルとウォーターフォールのハイブリッド？

「インフラは、アプリケーションに比べてスキルが属人的になりやすいからスクラムには向いていない」との有識者からのアドバイスを受け、プロジェクト開始当初は、アプリケーション開発はスクラム、インフラはウォーターフォールと別々の開発方法で進めましたが、うまくいきませんでした。考えてみれば当たり前なのですが、アプリケーションとインフラが密接しているにもかかわらずお互いの進め方が異なるため、調整するためのオーバーヘッドが発生し、スピード感が損なわれてしまっていたのです。

IoT基盤開発でのスクラム手法導入例

第2章 アジャイル開発実践！

Author 萩原 悠二（はぎわら ゆうじ）㈱NTTデータ
Author 梅森 直人（うめもり なおと）Twitter @unneto ㈱NTTデータ

本章では、筆者たちが実施している開発プロセスの概要と、各プロセスでどんなことを実施しているかについて紹介します。また、アプリケーション開発とインフラ開発を、どのような考え方でスクラムの枠組みの中で進めているかについて述べていきます。

開発プロセス概要――自分たちのスクラムを組んでみる

筆者たちはスクラムの手法を用いて、2週間単位のスプリントで、**図1**のようなスケジュールで開発を進めています。

スクラムはその拡張性の高さから会議体や進め方を柔軟にカスタマイズできることが特徴ですが、それゆえに基本をないがしろにしたカスタマイズは生産性や成果物の品質劣化を招くので危険です。

そこで筆者たちはスクラムの基本的な会議体や進め方は尊重してそのまま利用しつつ、チームの特性を考慮した会議体を設け、各々実施日を定めています。ここでは、筆者たちのインフラ領域でのスクラムを例として、各プロセスでどんなことを実施するのかについて見ていきましょう。

▼図1　スプリントスケジュール

月.1	火～金.1	月.2	火～水.2	木.2	金.2
デイリースクラム					休暇日
タスク ブレークダウン	開発	コードレビュー	開発	スプリント レビュー	
				レトロ スペクティブ	
開発		開発		スプリント計画 第1部	
				スプリント計画 第2部	
				スプリント計画 第3部	

まずは要件から――プロダクトバックログ定義と管理

スプリントを開始する以前に、達成すべきゴール（要件）を定める必要があります。スクラムでは、この達成すべき要件のことをプロダクトバックログと言いますが、筆者たちはこのプロダクトバックログを広く定義しています。ここでは、誰がどのような考え方に基づいてプロダクトバックログを定義し、管理していくのかについて、筆者たちが実際に活用しているツールとともに紹介していきます。

プロダクトバックログの定義

筆者たちのスクラムでは、**表1**のようにプロダクトバックログを大きく「ユーザストーリー（要件）」「リファクタリング」「バグ」「環境整備」の4つに分けて定義しています。

一般的にプロダクトバックログはプロダクト（成果物）に責任を持つプロダクトオーナーが作成するものですが、開発を進めるにつれて、開発者から「こういう機能があったほうが良いのではないか」という提案や「一度作ったはいいけど、コードが臭って

▼表1　プロダクトバックログの種類と定義

種類	定義
ユーザストーリー（要件）	達成すべき要件であり、開発者が開発できるように要求仕様と受入条件が定義されている
リファクタリング	アプリケーションもしくはインフラで改善すべき内容であり、改善方針と終了条件が定義されている
バグ	・アプリケーション、インフラで受入条件と異なる挙動を示している ・ユーザストーリー自身が破綻している（実現不可能）
環境整備	開発環境やテスト環境、コミュニケーション環境などのインフラ設計や設定、構築について、実施方針と終了条件が定義されている

きたのでリファクタリングしたい」「バグが見つかったけど、このスプリントでは改修する時間がないから次のスプリントで改修したい」といった要望が上がることがよくあります。さらに、「テスト環境にコードをデプロイしてほしい」「カーネルパラメータの設定を変更してほしい」といったインフラ側への要求も上がります。

これらを筆者たちは一元的にプロダクトバックログとして扱う運用にしており、まずは達成すべきリストとして積み上げます。そのうえで、適切なタイミングでスプリントに取り込むようにしています。アプリケーションもインフラも区別せず、あくまでも「何をしたいのか」に着目し、全体のバランスを見てプロダクトバックログに優先度付けを行います。

Redmineでプロダクトバックログを管理

作成したプロダクトバックログは、チーム全員が見える場所にあることが望ましく、何らかのコミュニケーション基盤を準備することが一般的です。数ある選択肢の中で、これまでの運用実績から筆者たちはRedmineを選択し、これにScrum Redmine Pluginを当ててカンバン方式でタスクを管理しています（**図2**）。

さあ計画しよう ――スプリント計画

前節までで、プロジェクトの要件をバックログという形で整理し、まとめました。

続いて、これらのバックログを実現するにあたり、スプリント期間中に開発チームがどのように動いていけば良いのかを計画します。

計画の重要性（なぜ：Why）

現在、企業のあらゆる業務や開発プロセスにおいて計画というものが重要視されているかと思います。スクラム開発も例に漏れず、計画は極めて重要です。

スクラム開発における計画の役割は次のようなものがあります。

1. 短期的な成果をプロダクトオーナーにコミットする
2. 限られた人員・時間で1.の成果を実現する

▼図2　Scrum Redmine Pluginによるカンバンの例

3.レトロスペクティブで次につなげる

⚠ 1.短期的な成果をプロダクトオーナーにコミットする

筆者たちのプロジェクトでは、外的要因が変動しやすい中で開発を進めるためにスクラム開発という方法を選択しており、中長期的な成果をコミットしにくい状況下にあります。しかしステークホルダーにとっては、今後何ができてくるのかが関心事としてあるため、2週間後、1ヵ月後といった短期的な成果の見込みをもとに方向性をすり合わせています。

⚠ 2.限られた人員・時間で1.の成果を実現する

スプリント期間が短く、メンバーも少ないスクラム開発では、計画を誤るとたちまちタスクの溢れや遅延を引き起こします。作業量を極力精緻に見積もることで、限られた人員で限られた期間内に確実に予定どおりのバックログを消化できるようにします。

⚠ 3.レトロスペクティブで次につなげる

スプリントは継続的に実施していくことですので、毎回のレトロスペクティブで振り返り、生産性や品質を向上していくことが重要です。計画がなければ振り返ることができなくなってしまいます。

立てるべき計画とは(何：What)

ここでいう計画とは、各スプリントで実現するバックログを、どのような段取りで進め、どのような成果物を出すかということを決めることをいいます。

筆者たちのプロジェクトでは計画時の成果物として、各バックログの受入条件、作業量見積もり(相対見積もり)、タスクチケットを作成しています。これらの成果物をもとに、開発チームが消化するバックログをプロダクトオーナーに対してコミットし、開発を実施します。

筆者たちが実践しているスクラム開発には次のような特徴があります。

- 2週間のスプリントで短期間に集中的に開発する
- 開発チームのメンバーが数名程度と小規模である
- スコープが確定しておらず、都度開発内容が変わり得る

上記のような特徴をふまえ、計画は次のような点が重要と思っています。

1. 詳細なガントチャートよりも、消化しなければならないタスクの見通しがよく、開発状況をデイリースクラムの場でチーム全員の認識合わせができること
2. 各メンバーの動き方が、○○機能専任というような縦割りではなく、タスクの優先順位の変化に応じて臨機応変にスイッチできるようにすること

筆者たちがスクラム開発を実践し始めた当初は、バックログの受入条件やタスクがあいまいなまま開発をスタートしてしまっていました。その結果、

- 成果物を作るための作業が自分でわかる高スキル者しか作業を進められず、ほかのメンバーは作業に着手できずに手が空いてしまった
- 開発対象のバックログの成果が、担当していたメンバーによってムラが出てしまった(あのバックログはこんな成果もあるけど、このバックログにはこれがない……など)

といった事態が発生してしまいました。

これに対し、筆者たちのプロジェクトでは、受入条件には内々に自明で暗黙の了解と思われていた成果物もきちんと明示する、またそれを作りこむための工程をメンバー全員が理解できる粒度でタスクチケット化するようにしました。

計画時点で、各タスクをメンバー全員が理解できるように詳細化しておくと、開発期間中に

▼表2　プロダクトバックログごとの主な受入条件

種類	定義
ユーザストーリー（要件）	・機能を満たす実装コードが完成していること（※注：バックログの内容に応じて詳細な実装内容、満たすべき機能を合わせて記載する） ・必要なミドルウェアの自動構築資材が完成していること ・対応する設計ドキュメントが執筆されていること ・ユニットテストコードが作成されていること ・シナリオテストコードが作成されていること ・デモ手順が記載されていること
リファクタリング	・実装コードが修正されていること ・対応する設計ドキュメントが執筆されていること
バグ	・実装コードが修正されていること ・自動構築資材が修正されていること ・設計ドキュメントが修正されていること ・回帰テストのための追加ユニットテストコードが作成されていること ・回帰テストのための追加シナリオテストコードが作成されていること
環境整備	・環境が構築できており利用可能なこと ・自動構築資材が作成されていること ・利用手順が記載されていること

タスクの分散がしやすくなります。また、やらなければならないタスクが明確になるので、バックログ間でムラが出にくくなります。こうすることで、メンバー間にスキル差がある中でも、各人が柔軟に活躍できるようになり、スクラム開発のメリットを得られるようになったと思っています。

計画の立て方（いかに：How）

計画は、開発メンバー全員が膝を突き合わせて1日に集中的に実施します。全員参加で計画を行うことで、ゴールや段取りを共有し、実作業として無理のない計画を立てるようにしています。

前述したとおり、計画では各バックログについて受入条件、作業量見積もり（相対見積もり）、タスクチケットを作成していくのですが、新しく取り組むバックログは1つとして同じものはないので、それぞれの進め方を考えながら上記を作成する必要があります。

▼図3　実現方式によって必要なタスクは異なる

とはいえ、各バックログを逐一まったくの別物として開発を進めるとメンバーが混乱しますし、また、それを確認するプロダクトオーナーにとっても妥当かどうかの判断がしにくくなります。そこで、筆者たちのプロジェクトでは、バックログの種類に応じて受入条件の雛形を用意しておき、これをもとに、個別バックログの事情に応じて実施するタスクをカスタムするようにしています（**表2**）。この方法を採ることで、当該バックログが完了することでどのような成果物ができるのか目線合わせを、プロダクトオーナー含めてプロジェクト全体でしやすくなりました。

また、具体的なタスクチケットを作成するにあたっては、どういうプログラムを実装し、またどういうミドルウェアを組み合わせることでバックログを実現するか、といった実現方式を検討しながら進めます。これは、実際にやらなければいけないタスクは実現方式によって変わるためです（**図3**）。

プロダクトオーナーとの合意

スクラム開発はスプリントという短い期間で確実に成果を出すという性質から、各スプリントでどの程度の成果をコミットするかがステークホルダーに大きな影響を持ちます。

したがって、ここで立てた計画をプロダクトオーナーと合意し、これをもって開発を開始します。

集中して開発

本節からは、いよいよ本格的にスプリントを開始します。前節の計画をもとにチケットを消化していくことになります。

開発対象と成果物

筆者たちのプロジェクトでは、インフラおよび必要なミドルウェアの開発を行っています。したがって、この工程ではこれらを設計・構築していくのですが、それぞれについて作成すべき成果物を定めています。

インフラ開発ではパッケージ群および自動構築用のAnsible Playbookを、ミドルウェア開発ではソースコードを、インフラとミドルウェアとで共通して、設計ドキュメント、シナリオテストコードおよびユニットテストコードを作成します。

とくにシナリオテストコードおよびユニットテストコードは、後述するテスト観点に基づく自動テストを行うもので、インフラ開発においてCIを実現するのにあたって重要です。スクラム開発では各スプリントで雪だるま式にシステムが拡大していくため、手動で回帰テストを行うのは不可能です。そのためCIを含む開発環境を整備し、常に自動テストし続けるしくみを導入しています。

開発環境の構成などについては第3章で詳しく説明しますが、開発した成果物はすべてGerrit(git)で管理しており、投稿されたパッチセットに対して自動的にテストおよびレビューのワークフローを回せるようにしています。

テスト観点

自動テストにあたり、筆者たちは2つの観点のテストを定め実施しています。

1.シナリオテスト

ユーザストーリーで定義した機能をシステム全体として実現できているかを確認するテストとしてシナリオテストを実施しています。

相対見積もりの難しさ

各バックログの計画を立てるときには、作業量を見積もり、自分たちのチームで限られた時間内に実施しきれるかを検討します。このときの作業量の見積もり方法として、見積もりポーカーを用いた相対見積もりを行っています。

相対見積もりは、あるバックログの作業量を1とし、そのほかのバックログの作業量をその何倍かと考えることで見積もる手法です。

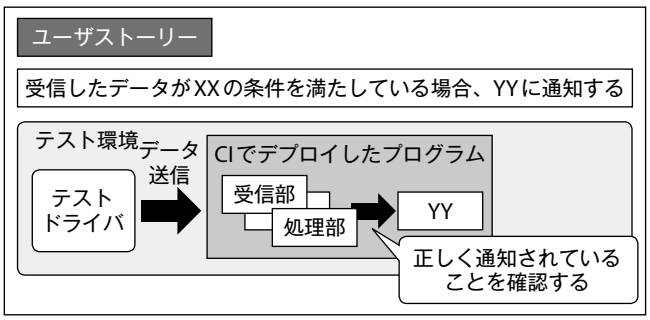
見積もりポーカーとは、1, 2, 3, 5, 8, 13, ……というフィボナッチ数列の書かれたカードのことをいいます。相対見積もりを実施するとき、各メンバーはバックログの作業量を見積もりポーカーにより提示します。ここで提示された値をすり合わせることで、チームとしての見積もり値を決定します。

相対見積もりと見積もりポーカーを使うと、大まかな作業見積もりがしやすい反面、見積もり精度を高めるのが難しく、見積もり値と実績値とのブレが大きくなりやすいです。

筆者たちのチームで実践したところ、とくに対象バックログの難易度によってメンバー間の見積もり値の差が大きくなり、見積もり精度に影響しました。各メンバーが明確に作業をイメージし、実施できると自信を持っているものはブレが小さく、高スキル者にとっては作業イメージができているが、他メンバーにとっては作業イメージが持てていないものはブレが大きくなる傾向があるようです。ブレが大きいときには、高スキル者が噛み砕いて説明することでチーム全体のスキル底上げと見積もり精度の向上を図っていますが、長丁場となることも珍しくなく、技術理解や見積もりの難しさを実感しています。

筆者たちのプロジェクトではIoTデバイスからの情報収集などを想定したシステムを開発しています。ですので、たとえばIoTデバイスからデータを受信したときに、定められた条件を満たしている場合にはあるソフトウェアに通知する、といった機能に対しては、IoTデバイスを模擬したテストドライバを用いてCIでデプロイしたシステムにデータを送信し、その結果として、通知が正しく行われる、といったことを確認します(**図4**)。

▼図4　シナリオテストの概要図



⚠ 2.ユニットテスト

実装したJavaプログラムのクラス単位や、構築した各ミドルウェアが、仕様どおり動作しているかを確認するテストとしてユニットテストを実施しています。Javaコードについては JUnitにより、ミドルウェアについてはServerspecによりテストコードを作成することで、自動テストできるようにしています。

成果物評価 ――スプリントレビュー

「スプリントレビュー」は、そのスプリントの成果物(動くコード)が、プロダクトオーナーの期待どおりの動きをしているかどうかを評価する場で、スプリントの一番の山場です。評価ポイントは、『スプリント計画 第1部』で定めたプロダクトバックログの受入条件をすべて満たしているかどうかで、インフラなのかアプリケーションなのかの区別は行わず、あくまでも総合的にどういう振る舞いを行うかで評価を行います。

カイゼン活動 ――レトロスペクティブ

「レトロスペクティブ」は「振り返り」とも呼ばれ、そのスプリントでチームが取った行動についてチームで振り返る場です。振り返りの方法はいろいろありますが、よく使われるのはKPT法(**図5**)でしょう。KPT法とは、そのスプリントでやってよかったことを継続的に実施しながら(Keep)、課題を明らかにして(Problem)、次

「完成」の定義

第1章のスクラムの概要説明で、透明性の原則について述べたとおり、スクラムではチーム内で共通認識を作ることが非常に大切です。共通認識の中でも、プロダクトに対して大きな影響を与えるものが「完成」の条件です。これは何をもってしてプロダクトバックログ項目や、各タスクを完成、完了とみなすかということを定義したものです。たとえば、あるメンバーはコーディングを実施し、動くソフトウェアを作成できれば実装タスクは完了だと考えていますが、他の開発者はコーディングだけでなく、単体テストも実施し、なおかつそのテストコードカバレッジが100%であることが実装タスクの完了だと考えるかもしれません。このような齟齬(そご)が顕在化しないまま開発が進んでしまい、終盤で認識に齟齬があることが判明すると、非常に短い期間で追加実装や修正が必要になったり、最悪の場合再実装になるなど、余計なコストが発生します。したがって、開発を開始する前に、何を完成の定義とするかを明確にし、チーム内に共通認識を醸成することが非常に重要となります。

▼図5　KPT法

Keep やって 良かったこと	Try 改善のために チャレンジ すること
Problem 課題と なっていること	

のスプリントを改善するために何にどうチャレンジするか(Try)をチームで振り返ります。

よくありがちなのが、成果について振り返り、行動についての振り返りを蔑ろにしてしまうケースや、Problemばかり抽出してTryが抽出できないといったケースです。レトロスペクティブのねらいは次のスプリントの改善につながる行動を抽出することにある、という点に留意します。

アレンジ部分の特徴とねらい

最後に、筆者たちのスクラム開発の運用でアレンジしている点について、3点に絞ってその特徴を紹介します。

特徴1：打ち合わせを集約している

開発メンバーが開発に集中できるよう、まとまった時間を確保することが狙いです。これなら「コーディングが捗っているのにあと10分で打ち合わせか……」と気持ちが沈む心配もありません。しかしその分、第2週目の木曜日は会議室にこもりっきりになるため非常にたいへんなのですが。

特徴2：タスクブレークダウン、コードレビューの時間帯を設定している

レトロスペクティブで、「タスクの粒度が開発者ごとに違っていて全体の進捗がよくわからないね」「ペアプロやピアレビューもいいけど、スプリントレビュー前に開発者間で成果物の意識合わせの時間帯がほしいよね」という開発者の声から生まれたイベント(会議体)です。

⚠ タスクブレークダウン

「タスクブレークダウン」は、開発者全員によってプロダクトバックログをタスクに分割する、という会議体です。開発者全員の思考や行動が完全均一化された金太郎飴のようなチームであれば不要な会議体でしょうが、現実はそうもいきません。また、プロダクトバックログの粒度を細かくするのも1つの手かもしれませんが、今度はプロダクトオーナーの負担が大きくなります。開発者には凸凹があって当然、プロダクトオーナーまで含めて稼働負荷を平準化する、それでも開発の計画段階くらいは足並みをそろえたい、という考え方に基づいて試行錯誤して生まれたのがこの会議体です。

⚠ コードレビュー

「コードレビュー」は、開発者全体での成果物に対する意識合わせの場で、実装に対するイメージギャップを埋めることが目的です。スプリントレビューの場で「え？　そういう実装になっていたんですか？」なんていう残念なコメントが発せられる惨劇を回避できます。

特徴3：休暇日が定常的に組み込まれている

スクラムに限らず、アジャイルは常に計画を修正しながら全力疾走するため、油断するとオーバーワーク気味になりがちです。

また、基本的にスプリントを回している最中は開発に集中しますので、成果物のアウトプットに関わる作業以外は行ってはいけません(このあたりはウォーターフォールでも同じですね)。

そのため、何をしても良い日をスケジュールに組み込んでおくことで、積極的に休暇を取ったり、そのスプリントで見つかった自分自身の弱点を補強したりできます。気持ちに余裕が生まれ、結果として生産性の向上につながります。ちなみに、「休暇日」を「バッファ」と読み替えると少々親近感が湧くのかもしれません。同じ意味合いのはずですが、「バッファ」では休みにくいと感じるのは筆者だけでしょうか。SD

AWS EC2をAnsibleとServerspecで管理

第3章 CI実践! JenkinsとGerritを動かしてみる

Author 荒関 翔（あらぜき しょう） ㈱エヌジェーケー

第1章ではアジャイル・CIの基礎知識を押さえ、第2章ではスクラムの流れを追いました。本章では、Jenkins、Gerrit(Git)、Ansible、Serverspec、AWS EC2を使い、実際にインフラをCIで開発するための基盤を構築します。

インフラCIのしくみ

インフラCIの定義

はじめに、筆者が本稿で述べる「インフラCI」の適応範囲について説明します。ここで定義するインフラCIとは、インフラ基盤の開発・テストを、駆動的・継続的に行うだけではありません。我々が開発している基盤の特徴として、OS、ミドルウェア、アプリケーションをそれぞれ組み合わせて開発しているため、正確には「インフラ＋ソフトウェアCI」という形でインフラCIのしくみを実現しています。そのため、本稿におけるインフラCIとは、インフラとソフトウェアを組み合わせた形のCIとして説明させていただきます。

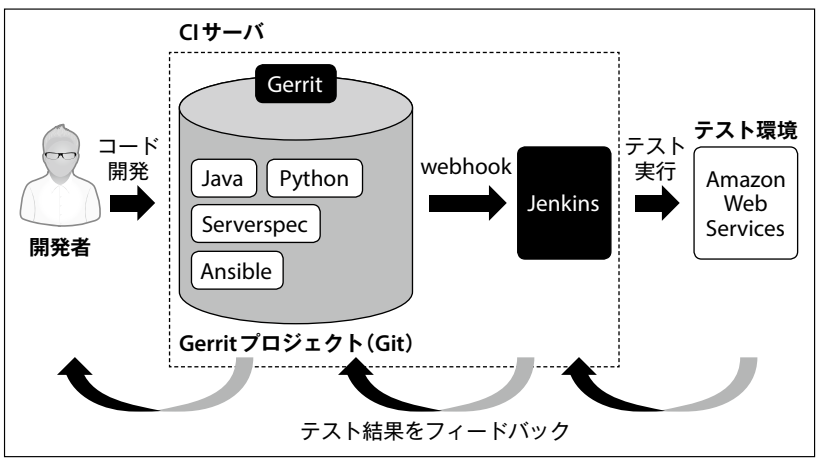
アーキテクチャ

アーキテクチャは**図1**のとおりで、CIサーバ上にCIツール「Jenkins」とコードレビューツール「Gerrit」が同居しており、開発者がコードをGerritへコミットすると、Jenkinsが自動的にビルドを行い、テスト環境にて単体テスト(Unit Testing、以下UT)やシナリオテストを自動的に行います。その後、テスト結果をGerritへ反映したり、開発者へ通知したりします。

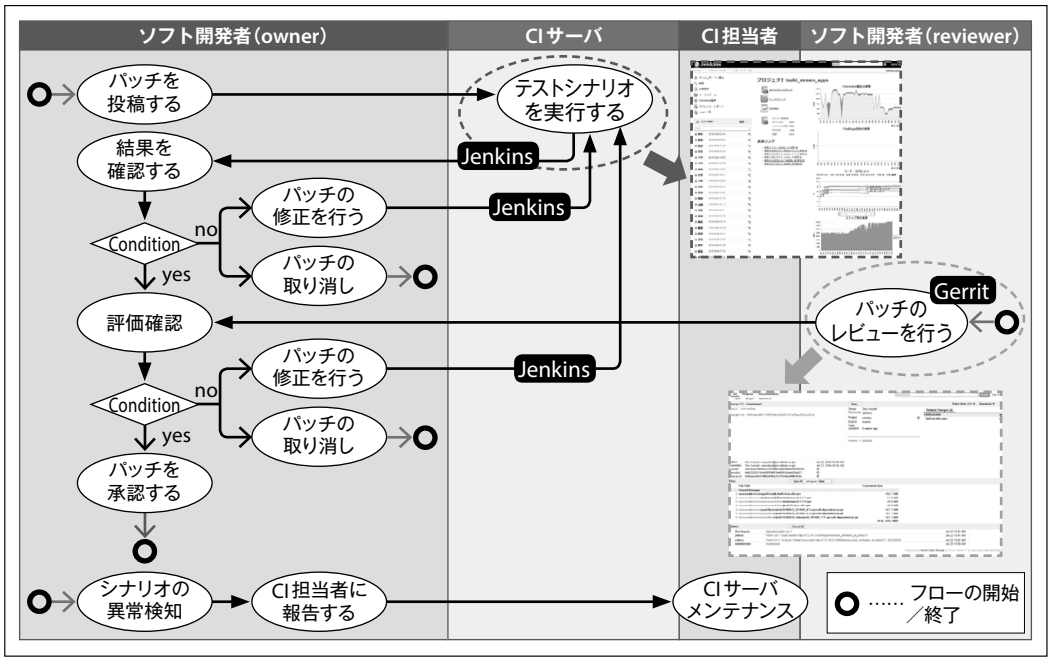
ワークフロー

具体的なワークフローは**図2**のとおりです。まず開発者がCIサーバへ、パッチ投稿という形でコードをコミットします。それをJenkinsが検知し、そのパッチに対するテストをテスト環境にて自動的に実施します。テスト終了後に結果が開発者へ通知され、JenkinsやGerritへアクセ

▼図1 インフラCIのアーキテクチャ図



▼図2 インフラCIのワークフロー



スすることでテスト結果の詳細を確認します。Jenkinsのテスト結果とレビュアの評価を受け、masterブランチへ反映するという流れになっています。

インフラCI環境構築

インフラCIを実現するための基盤について具体的に説明します。ベースとなるOSとしては、CentOS 7を用います。CI環境を構成する各パッケージは**表1**を参考にしてください。CIツールには、ソフトウェア開発におけるCIツールとして有名なJenkinsを使用しています。インフラ構築・開発には、Ansibleを使用しており、OS・ミドルレイヤのテストとしてServerspecを使用しています。またGerritと連携することで、「機械的なしかけ」と「人が確認するためのしかけ」を組み合わせ、両面からの品質担保ができるしくみを実現しています。

▼表1 インフラCIの環境を構成するパッケージ一覧

パッケージ名	バージョン	説明
jenkins	2.8-1.1	CIツール
ansible	2.0.1.0-2	構成管理ツール
serverspec	2.31.1	インフラテストツール
gerrit	2.12.2	コードレビューツール

Jenkins構築

Jenkinsの導入は、公式サイト[注1]でRPMを配布しているため容易にできます。パッケージをダウンロードしてrpmコマンドでインストールするだけです。デフォルトで8080ポートになっているため、インストールしたサーバの8080ポートにアクセスすればトップ画面が表示されます。

次に、プラグインのインストールについて説明します。Jenkinsのトップ画面より[Jenkinsの管理]→[プラグインの管理]と遷移していくと、[利用可能]タブからインストールできるプラグイン一覧を確認できます。Jenkinsには多種多様なプラグインが存在するため、本書ではインフラCIに必要なものについて紹介したいと思います。ここで紹介するプラグインはほんの一部で

注1) URL https://jenkins.io/index.html

▼表2　Jenkinsのプラグイン一覧

プラグイン名	バージョン	説明
Checkstyle Plugin	3.46	コーディング規約をチェックするための静的コード解析プラグイン。解析結果をグラフ・表形式でレポート
Cobertura Plugin	1.9.8	カバレッジレポートをグラフ・表形式でレポートするプラグイン
FindBugs Plugin	4.65	FindBugs(静的コード解析ツール)プラグイン。解析結果をグラフ・表形式でレポート
Gerrit Trigger	2.21.1	Gerritと連携するプラグイン。ジョブ結果をGerritへ反映するために利用
StepCounter Plugin	1.4.5	ステップ数をグラフ・表形式でレポート
Build Flow Plugin	0.20	DSL(Domain Specific Language)形式でジョブフローを定義するプラグイン

あるため、本稿はあくまで、読者のみなさんがインフラCI環境をスムーズに構築できるようになるための入り口になればと思います。インフラCIで利用しているプラグイン一覧は**表2**になります。

Gerrit構築

Gerritは公式サイト[注2]でWARファイルを提供しており、それを用いてインストールを行います。Gerritのインストールは少し複雑なため本稿では割愛させていただきますが、公式やWebサイトでさまざまな方が手順を紹介していますので、参考にしていただければと思います。ポイントとしては、次の4つになります。

- インストール前にGerrit用のユーザを作成しておくこと
- インストールはGerrit用ユーザで行うこと
- デフォルトがJenkinsのポートと同じため、競合しないようにしておくこと
- 事前にGerrit用のデータベースを選定しておき、インストールしておくこと

また、Gerritの設定例を**リスト1**へ載せておきますので、参考にしていただければと思います。

インストール後は、Gerrit用ユーザホームディ

▼リスト1　gerrit.config設定例

```
[gerrit]
        basePath = git
        canonicalWebUrl = http://172.16.0.9:18080/
[database]
        type = postgresql
        hostname = localhost
        database = reviewdb
        username = gerrit2
[index]
        type = LUCENE
[auth]
        type = DEVELOPMENT_BECOME_ANY_ACCOUNT
[receive]
        enableSignedPush = false
[sendemail]
        smtpServer = localhost
[container]
        user = gerrit2
        javaHome=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.91-0.b14.el7_2.x86_64/jre
[sshd]
        listenAddress = *:29418
[httpd]
        listenUrl = http://*:18080/
[cache]
        directory = cache
[mimetype "application/vnd.ms-excel"]
        safe = true
[mimetype "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"]
        safe = true
[mimetype "application/*"]
        safe = true
[hooks]
path=/home/gerrit2/hooks
```

注2) URL https://www.gerritcodereview.com

レクトリに./bin/gerrit.shというシェルがインストールされるので、引数にstartを与えて実行し、起動することを確認してください。次に、**リスト1**で設定したGerritのURLへアクセスし、トップ画面が表示されたらインストール完了です。

Gerritプロジェクトの作成

Gerritプロジェクトの作成はAdmin(管理者)権限を持つユーザでしか作成できないため、まず初めにadminユーザを作成しておきます。adminユーザは**リスト1**にある、

```
[auth]
        type = DEVELOPMENT_BECOME_ANY_ACCOUNT
```

という設定でないと作成できないため、Gerritのインストール後にすぐに作成しておくことをお勧めします。

次にプロジェクトの作成です。プロジェクトの作成はトップ画面より、[Projects]→[Create Project]で作成できます。本章では、Gerritのプロジェクトとして3つのプロジェクトを作成していきます。

①インフラコードを管理するプロジェクト(Ansible、Serverspec)
②アプリコードを管理するプロジェクト(Java、Python)
③シナリオテストを管理するプロジェクト(シェル、Python)

便宜上、リポジトリ名を各々①infra、②apps、③ciscriptsとしておきます。

プロジェクトの作成を終えたら、ユーザの設定などを行います。グループから先に作成しておくと管理を楽にできます。ユーザの設定は[Settings]から行えます。ユーザ作成でやっておかなければならないポイントとしては、開発者のSSH keyの登録です。登録の動作確認は、開発者端末で**図3**のとおりに接続できれば成功です。

▼図3 GerritプロジェクトへのSSH接続確認

```
$ ssh <user>@ciserver -p 29418
****    Welcome to Gerrit Code Review    ****
Hi hoge, you have successfully connected over SSH.
Unfortunately, interactive shells are disabled.
To clone a hosted Git repository, use:
git clone ssh://<user>@172.16.0.9:29418/REPOSITORY_NAME.git
```

Ansible構築

AnsibleはEPEL[注3]などでRPMを提供していますので、CIサーバにAnsibleパッケージをインストールします。

Serverspec構築

Serverspecは公式サイト[注4]でgemファイルを提供しているため、gemコマンドでインストールできます。

◆ ◆ ◆

以上でCI環境の事前準備は終了です。

テスト実装

実装方針

次に、どのようにテストを実装していくのかを解説します。

まずは、全体のジョブの流れを説明します。なお、ここでのテスト環境はAmazon Web Services(以下AWS)上で行っています。また、ここで利用するコードのすべてはGerritプロジェクト(Git)で管理されています。

始めに、デイリーで8:00からインスタンスの起動ジョブが実行され、Ansibleを実行し、OSのセットアップ、ミドルウェアの起動を行います。

次に、ServerspecによるインフラUT、LT(Link Testing：結合テスト)コンポーネント間

注3) URL https://dl.fedoraproject.org/pub/epel/7/x86_64/a/
注4) URL http://serverspec.org/

▼リスト2　AWS EC2インスタンスの起動スクリプト

```
# AWS上のtagに"test_*"が付与されているインスタンスIDを取得する。
IDS=`aws ec2 describe-instances \
    --filters "Name=tag-key,Values=Name" \
    --filters "Name=tag-value,Values=test_*" | \
    jq -r '.Reservations[].Instances[].InstanceId'`

# 取得したIDのインスタンスを順次起動する
for ID in $IDS;do
  aws ec2 start-instances --instance-ids $ID
done
```

が正しく動作しているかを確認するテストが実行されます。

以降は、インフラまたはアプリケーションのどちらかのコードがアップデートされるたびにテストが実行されます。インフラコード（Ansible、Serverspec）をアップデートした場合、そのコードのテストが行われ、テスト結果がGerritへ反映されます。ソフトウェアの場合は、アプリケーションのビルドを行い、デプロイ、シナリオテストと実行されます。

インフラUT実装

具体的にどういう形でジョブを定義しているのかを、AWS CLIのスクリプトなども交えて説明します。

⚠ インスタンス自動起動設定のスクリプト

AWS EC2インスタンスは、AWSが提供しているCLIを使用して起動しています。**リスト2**のようにスクリプトを作成することで起動できます。作成し終えたら、Gerritプロジェクトで作成したciscriptsリポジトリに保存しておきます。

⚠ インスタンス自動起動のジョブ定義

Jenkinsのジョブ定義を設定していきます。最初に[General]でプロジェクト名とその概要を書きます（**図4**）。重要なポイントは次の3点です。

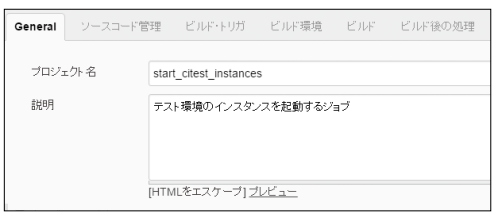
・[ソースコード管理]でGitを選択。ここのGitリポジトリにはciscriptsリポジトリを選択（**図5**）
・[ビルド・トリガ]で定期実行を設定し、毎日8:00に起動するように設定（**図6**）
・[ビルド]では、**リスト2**で作成したスクリプトを実行するように設定（**図7**）

⚠ インフラUT用ジョブ

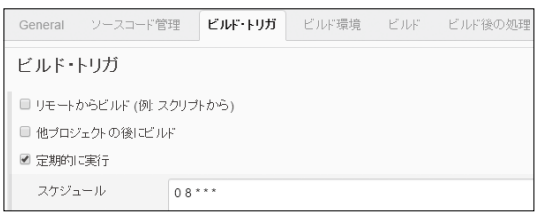
次に、インフラUT用のジョブ作成です。インフラUTコードはレビュー対象であるため、Gerrit連携のしかけを用いたジョブ作成を行います。さきほどと同様に[General]でプロジェク

▼図5　インスタンス自動起動ジョブ設定：[ソースコード管理]

▼図4　インスタンス自動起動ジョブ設定：[General]



▼図6　インスタンス自動起動ジョブ設定：[ビルド・トリガ]



▼図7　インスタンス自動起動ジョブ設定：[ビルド]

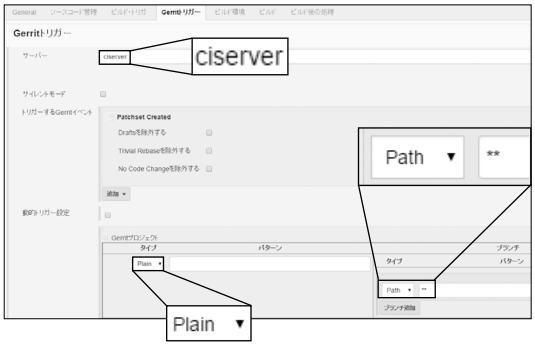
▼図8　インフラUTジョブ設定：[General]

▼図9　インフラUTジョブ設定：[ソースコード管理]

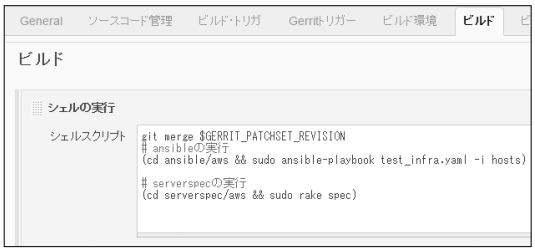
ト名とその概要を書きます(**図8**)。設定のポイントは次の3点です。

・[ソースコード管理]でGitを選択。ここでのGitリポジトリには、AnsibleコードとServerspecコードが含まれるinfraリポジトリを選択(**図9**)。次に[Repositories]画面内にある[高度な設定]を選択した際に設定する[Refspec]欄に「$GERRIT_REFSPEC」を設定する(ここはよく書き忘れてしまうことがあるため、注意)

▼図10　インフラUTジョブ設定：[Gerritトリガー]



▼図11　インフラUTジョブ設定：[ビルド]



・[Gerritトリガー]で環境に合わせて設定。タイプには「Plain」を選択し、パターンにはGerritのプロジェクト名を入力。ブランチ側も同様に設定。全ブランチに対して実行する場合はパターンの値を「**」と設定(**図10**)

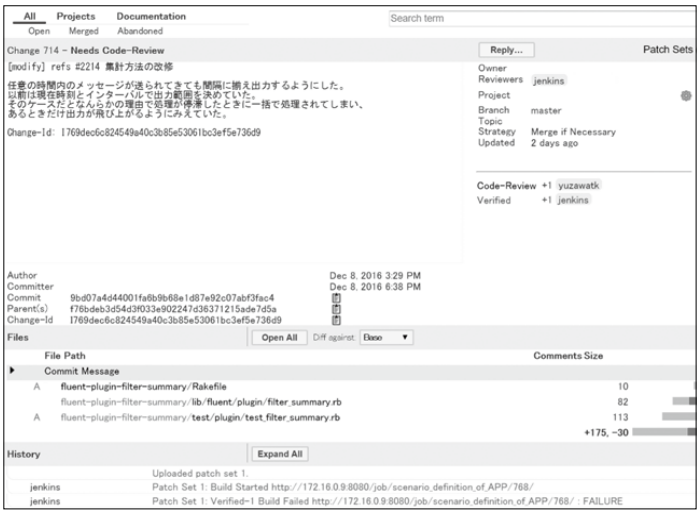
・[ビルド]では一番初めにマージの設定を行う(マージの設定を行わないと、パッチが当たっていないままテストが実行されてしまうため)。「$GERRIT_PATCHSET_REVISION」という環境変数はGerritトリガープラグインをインストールしたときに設定されるため、こちらでとくに設定を行う必要はない(**図11**)

動作確認は、Gerritパッチのクエリとトリガーから手動で実行することもできますので、テストパッチなどを投稿して、確認してみることをお勧めします。そしてJenkinsでのテストが流れ終わると、Gerritへ結果をフィードバックしてくれます(**図12**)。

◆ ◆ ◆

これら設定はパッチセットに対するテストまでですので、ここでテストされたコードは、ま

▼図12　Jenkinsでのテスト結果がGerritへフィードバックされたときの画面



だテスト環境へ実際に反映されたわけではありません。テスト環境へ反映させるためには、さらに別ジョブとして、masterブランチにmergeされたらテスト環境へ反映するというしかけを用意します。そうすることで、CIテストとレビューが完了したものが次々と反映されるようにできます。

今回はインフラの変更が少なかったため、反映はデイリー、またはアドホックに手動で行っていますが、更新頻度が高い開発では、Jenkinsのwebhookプラグインなどを利用することで、高頻度の反映が実現できます。

アプリケーションUT実装

アプリのジョブは3つで構成されており、本節ではそれぞれのジョブの特性について説明していきます。

▼図13　ジョブフローを定義するジョブの設定：[General]

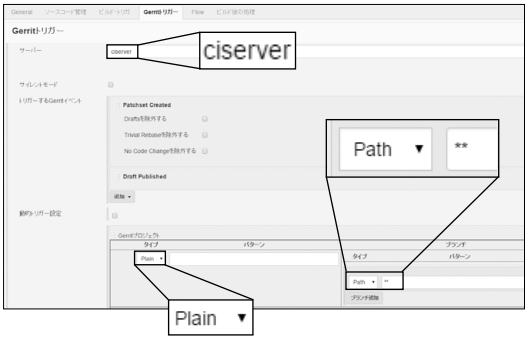
(1)ジョブフローを定義するジョブ

ジョブフローを定義するジョブは、インストールしたプラグイン「Build Flow Plugin」(表2参考)を活用して、フローをDSLで管理できるように設定します。プロジェクト名とその概要を[General]に書きます(図13)。テストフローを定義するポイントとしては、次の3点です。

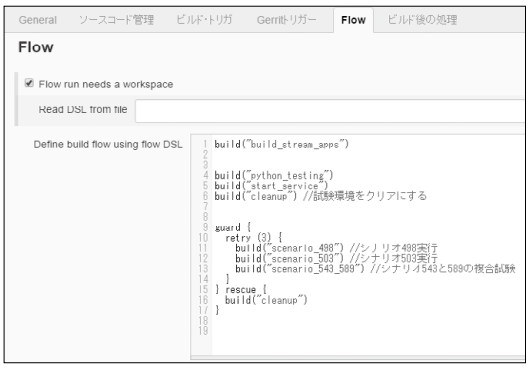
・[ソースコード管理]でGitを選択。ここで設定するGitリポジトリはJava、Pythonのコードを管理しているappsリポジトリを選択(図14)
・「インフラUT実装」節での説明と同様に、[Gerritトリガー]でフロー定義ジョブを設定(図15)
・実際のジョブの流れは[Flow]に定義(図16)。DSL形式で定義できるためコードベースで管

▼図14　ジョブフローを定義するジョブの設定：[ソースコード管理]

▼図15　ジョブフローを定義するジョブの設定：[Gerritトリガー]



▼図16　ジョブフローを定義するジョブの設定：[Flow]



理可能。またこのジョブ画面1つで管理できるため、既存のジョブフローを利用するよりも管理コストは大きく削減できる

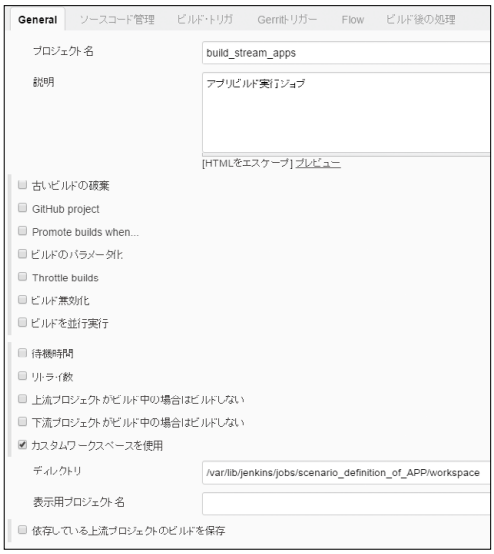
ここで定義したジョブの流れとしては、Gerritプロジェクトのappsリポジトリへのパッチセット投稿を契機に、アプリのビルドが開始されます。ビルドが成功すれば、インフラUTが実行され、テストシナリオが実行されていくしかけとなっています。後述する(2)(3)で作成するジョブ名をここで定義することで、ジョブフローの組み換えを容易に行うことを可能としています。

(2)ビルドジョブ

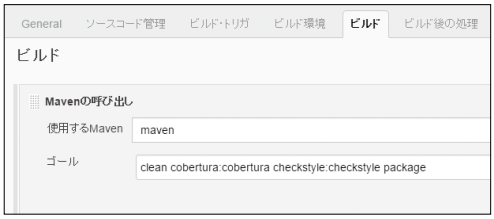
ビルドジョブはデフォルトで用意されているJenkinsのMavenプロジェクトを活用しています。設定のポイントは、次の2点です。

・カスタムワークスペースにシナリオジョブのワークスペースを選択(図17)

▼図17　ビルドジョブ設定：[General]



▼図18　ビルドジョブ設定：[ビルド]



・ビルド設定にMavenのビルド設定を選択します。[ゴール]欄にcoberturaやcheckstyleの設定をすることで、後にレポートをグラフ・表形式で参照できるようになる(図18)

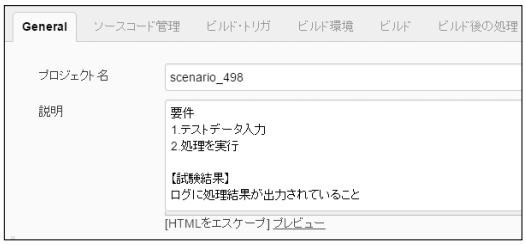
最後に、ビルド後の処理を設定します(図19)。

(3)テストシナリオジョブ

ジョブを作成する前に、テストシナリオを行うスクリプトの準備をしておきます。テストスクリプトには、実際にテストを実行するものだけでなく、環境をきれいにするcleanup用のスクリプト、ビルドしたアプリを環境へ配置するスクリプトなどを用意しておきます。図16で定義したフローの中で、cleanupジョブやtest_settingジョブが初めに呼ばれていますが、これらがそれにあたります。実際にテストするもの

▼図19　ビルドジョブ設定：[ビルド後の処理]

▼図20　テストシナリオジョブ設定：[General]



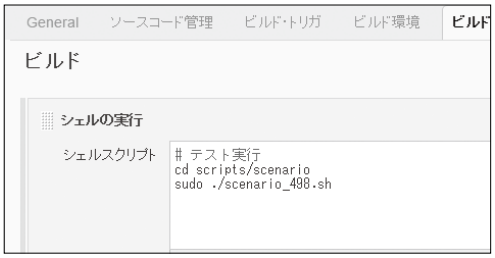
はsenario_XXXと書かれたジョブになります。

スクリプトの準備ができたら、ジョブを設定していきます。cleanupジョブ、test_settingジョブ、senario_XXXジョブはすべて作成のしかたが同じため、テストシナリオジョブの作成のしかたのみ紹介させていただきます。

[General]でプロジェクト名と概要を入力し(**図20**)、インフラUTのときと同様にテストシナリオであるシェルスクリプトなどはGit上で管理しておき(**図21**)、ビルド定義は簡潔にしています(**図22**)。SD

▼図21　テストシナリオジョブ設定：[ソースコード管理]

▼図22　テストシナリオジョブ設定：[ビルド]



Jenkinsのジョブにコマンドべた書きは厳禁！

Jenkinsのジョブ定義は[ビルド]設定の[シェルスクリプト]部分にべた書きできるため、当初はすべてのテストジョブをJenkins上に記載してジョブを作成していました。しかし、このやり方では管理コストが高くなってしまいました。テストジョブを修正するために、ひとつひとつジョブをGUI上で確認しなければならなくなってしまったのです。その結果テストジョブのせいで、何度も失敗したという苦い経験を味わいました。今回はテストケースもすべてGitで管理する方針を採り、管理コストの削減を図りました。

アルゴリズムクイックリファレンス第2版

George T. Heineman、Gary Pollice、Stanley Selkow 著／黒川 利明、黒川 洋 訳
A5判／440ページ
3,600円＋税
オライリー・ジャパン
ISBN＝978-4-87311-785-0

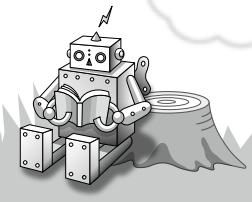
世界中で翻訳されているアルゴリズムの名著。改訂に際して、フォーチュン走査法、マージソート、マルチスレッドクイックソート、AVL平衡二分木、R木と四分木といったアルゴリズムを、Pythonを使って解説する内容が盛り込まれた。本書の特徴は、実践的なアルゴリズム本であるということ。それぞれのアルゴリズムの解説は、まず簡単な擬似コードで処理の流れを追ってから、どのような場面で効果を発揮するのかという「文脈」を説明し、C、C++、Java、Pythonで実装したコードを掲載したうえで、最後には振る舞いや性能について分析する、という流れをとっている。アルゴリズムの理論を押さえたい人というよりは、実践で使えるようになりたい人向けの1冊と言える。なお本書の実装コードはGitHubから入手できるとのこと。

あたらしい人工知能の教科書

多田 智史 著／石井 一夫 監修
A5判／352ページ
2,600円＋税
翔泳社
ISBN＝978-4-79814-560-0

流行りのディープラーニング関連の用語をいくつか拾うだけでも、順／逆伝播、活性化関数、バックプロパゲーション、勾配消失問題など聞きなれない言葉がバンバン出てくる。人工知能の話題を追いかけるにはこういった用語を知っておかないと先に進めないが、本書はその大きな助けになる。人工知能の歴史から解析学、統計学的な機械学習、処理能力のブレイクスルーとなった分散コンピューティング、そして近い将来深く関係するIoTと大規模データ、ロボットや脳研究といった幅広い話題を取り上げている。専門的な内容はWebや他の文献にゆずることが多いが、本書はそういった専門書を読むための下地として最適。深くなりすぎず、浅く流してしまいすぎず、うまいバランスでまとめられた解説が“あたらしい教科書”にふさわしい。

SD BOOK REVIEW



まつもとゆきひろ 言語のしくみ

まつもとゆきひろ 著
B5変型判／352ページ
2,800円＋税
日経BP社
ISBN＝978-4-8222-3917-6

プログラミング言語で別のプログラミング言語を作る。これが本書のテーマだ。『2週間でできる！ スクリプト言語の作り方』（千葉滋 著：当社刊）を担当したことがあるのだが、言語で言語を作るのは大学の情報処理学科でよく取り上げられる題材だ（コンパイラの授業など）。この本ではRubyに影響受けてJavaで“Stone”というスクリプト言語を作るというものだった。一方本書は、mrubyの改造をテーマに新しいプログラミング言語“Stream”を作りあげていく過程をみせていくものだ。そのつどなされる、まつもとさんの解説がとにかく的を射ていて贅沢。読み進めると至福の時間を味わえる。しかも新しい技術のトピックも平易に解説がなされているので、プログラマだけでなくすべてのIT技術者に勧めたい1冊と言えよう。

その「エンジニア採用」が不幸を生む

正道寺 雅信 著
四六判／272ページ
1,780円＋税
技術評論社
ISBN＝978-4-7741-8601-6

本書では、経営コンサルタントの著者が実際の案件をベースに、エンジニア採用における諸問題を洗い出している。この本の目的「エンジニアも会社も不幸になる要因の分析と解決策を検討すること」のとおり、前半では「スーパーエンジニアを、給与を抑えて採用しようとする」「エンジニア職の特性を理解しないままに、就活直前になってエンジニアを志す」など、採用する側／される側の両方の立場から、失敗の原因を指摘している。後半ではそれらをふまえたうえで、良いエンジニアを採用し、継続的に働いてもらえるための採用側の施策について考察している。本誌2017年1、2月号においても「エンジニアが採用できない会社と評価されないエンジニア」が連載されるなど、エンジニア採用のあり方に注目が集まっているのは間違いない。

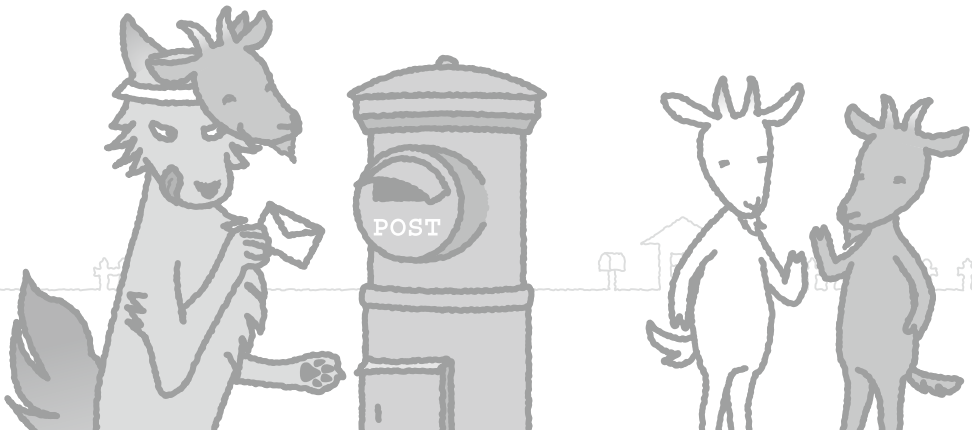
第3特集

どうなってる？ なりすましメール対策

DKIMとホワイトリストによる安心の可視化

またあやしいメールが届いていませんか？　ぱっと見であやしいとわかるものならすぐに捨てますが、最近の悪質なメールはメールアドレスや送信者名に有名企業や銀行などの名前(あるいは酷似した名称) を含んだもので送ってきます。見かけだけで判断するのはかなり難しく、現実に被害も発生しています。

本特集では、なりすましメールへの対策として考えられている、送信者の身元を検証するしくみと取り組みについて解説します。メールを安全で使いやすくするために、技術者として何ができるのかを考えるきっかけとしてください。



第1章 なりすましメールの動向と対策技術

SPF、DKIM、DMARC、S/MIMEのしくみを知ろう

Author 北川 直哉（きたがわ なおや）　東京農工大学 大学院工学研究院

本章では、なりすましメールがどのような手口を使って送信されているのかを例示し、現在のメールシステムでとりうる対策技術について説明します。

なりすましメールの動向

実在する金融機関や公的機関などを詐称し、受信者の個人情報（たとえばログインIDやパスワード）や機密情報を盗み出そうとする「なりすましメール」は重大な社会問題となっています。典型的ななりすましメールの例を**図1**に示します。この例では、銀行を装った攻撃者が個人顧客に対してパスワード変更を促す内容の文章とともに偽サイトへのリンクを付加し、フィッシング詐欺を試みています。このようななりすましメールによる被害は長年にわたって発生していますが、なりすましメールが悪用されるケースはフィッシング詐欺だけではありません。近年被害が急増している標的型攻撃の入り口として最も多く利用されているのもなりすましメール（同僚や上司、取引先の人物などを詐称）であり、なりすましメール対策は情報セキュリティ対策の最重要課題と言っても過言ではありません。

従来のなりすましメールには本文の日本語に違和感があるなどの特徴があり、受信者は多くのなりすましメールを容易に見破ることができていましたが、最近ではなりすましの手口が巧妙化しており、メール本文の内容から見破るのは極めて困難なものが多くなっています。

なりすましの手口

なりすましメールの多くは、ディスプレイネーム（**図1**-①）やメールアドレス（**図1**-②）を詐称して送信されます。

ディスプレイネームは個人名や企業名などの任意の文字列を記述可能であるため、フリーメール（GmailやYahoo!メールなど）を使用して送信されるなりすましメールでも頻繁に悪用されています。最近ではスマートフォンやタブレット端末でメールの閲覧をする機会が増えていますが、このような環境では送信者欄にディスプレイネームのみが表示されてメールアドレスは表示されないことが多く、受信者がメールアドレスの詐称を見破ることは困難です。

メールアドレスの詐称には、実在するドメイ

▼図1　なりすましメールの例

①　②
From:●●銀行<info@example.jp>

Subject: パスワード変更のお願い

○○様
お客様のアカウントの安全を保つためにパスワードの変更をお願いいたします。
下の URL をクリックして、当行の ID とパスワードでログインの上ご変更下さい。

https://login.example.jp

ン名のメールアドレスと同一のメールアドレスを使用する方法と、実在するドメイン名に類似した別のドメイン名のメールアドレスを使用する方法があります。前者は、電子メール配送の標準プロトコルであるSMTP(Simple Mail Transfer Protocol)通信では送信者のメールアドレスを容易に偽称可能であることを利用したものです。電子メール通信における送信者アドレスには、SMTP通信におけるMAIL FROM:コマンドの引数として与えられる「エンベロープFromアドレス」と、From:ヘッダ上に示される「ヘッダFromアドレス」が存在します。電子メールの世界では、これら2つのアドレスが同じでなければならないという規約はなく、送信者は容易に詐称したアドレスを用いて電子メールを送信することができてしまいます。後者は、「Cousin Domain Name(いとこドメイン名)」と呼ばれる本物のドメイン名に類似した文字列の別のドメイン名のメールアドレスを用いる方法(**図2上部**)や、昨今急増しているトップレベルドメインを利用し、あらかじめ取得した企業名や商品名のドメイン名のメールアドレスを用いる方法(**図2下部**)が存在します。

現在のところ、**図2**に示した例のようななりすましメールに対して有効と言える対策技術は提案されていませんが、実在する企業などのドメイン名のメールアドレスを詐称したなりすましメールや、配送途中での改ざんへの対策技術は複数存在します。これらの技術について、次節で解説します。

▼図2　メールアドレス詐称の例

●Cousin Domain Name を用いたなりすまし例

本来のメールアドレス	info@example.jp
なりすましの例	info@exampl**1**e.jp

※「l（エル）」が「1（いち）」になっている

●ドメイン所有者の正当性の判断が困難ななりすまし例

本来のメールアドレス	info@example.jp
本来のメールアドレス	info@example.tokyo
なりすましの例	info@example.asia

※自衛目的で多くのドメイン名を取得する企業が多いが、どのトップレベルドメインが意図した接続先の所有するドメイン名かわからない

なりすましメール対策技術

SPF

SPF(Sender Policy Framework)は、メール送信サーバの正当性を検証するしくみであり、RFC 7208で標準化されています。SPFは前節「なりすましの手口」中で述べた、エンベロープFromアドレスからの配送の正当性を検証するしくみです。送信側では、あらかじめ当該ドメインのDNS権威サーバでSPFレコードを公開しておきます。SPFレコードではレコードを公開しているドメインからメールを送信するSMTPサーバのIPアドレスのリストを宣言します。

▼図3　SPF検証の流れ

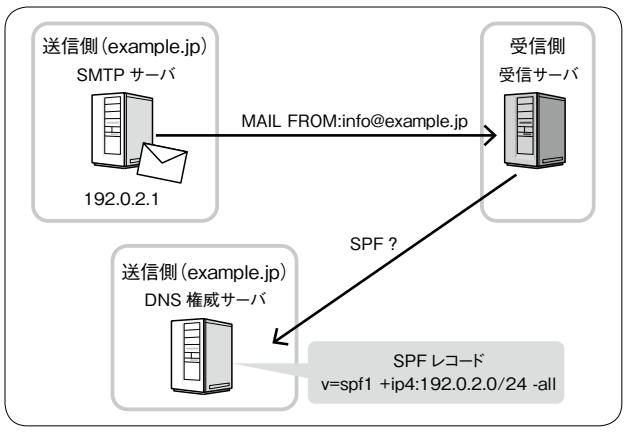
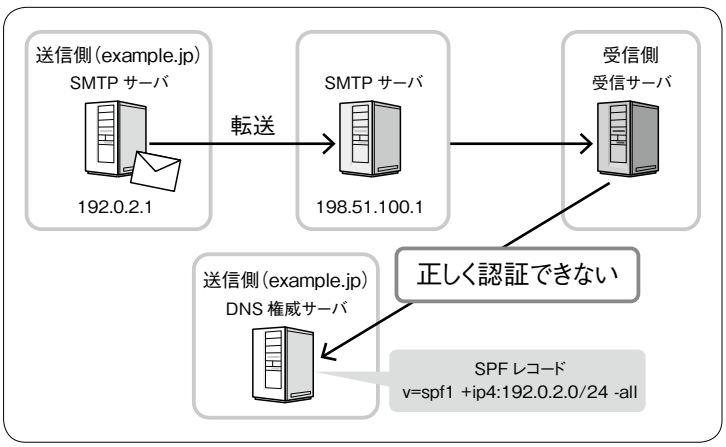


図3に、受信サーバがinfo@example.jpからのメールを受信した場合のSPFによる送信ドメイン認証の流れを示します。この例では、送信側はexample.jpからのメールは「192.0.2.0/24」のアドレスブロックに含まれるIPアドレスを持つサーバから送信する旨を宣言しています。受信側がこのメールを受信すると、example.jpのDNS権威サーバにSPFレコードを問い合わせて情報を

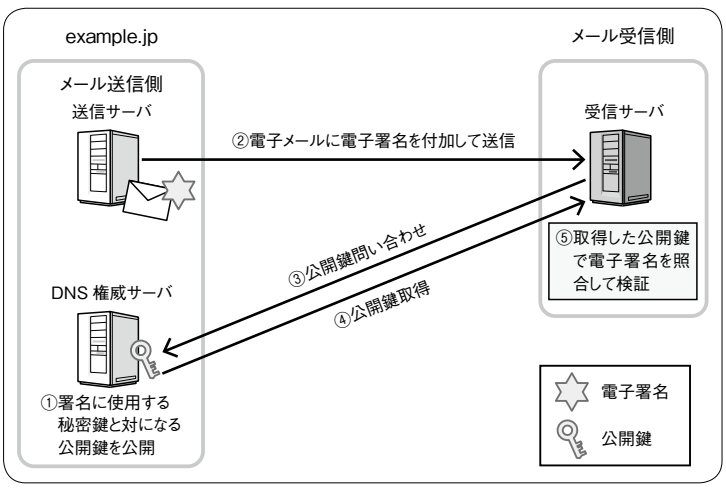
取得し、そのメールの送信サーバのIPアドレスと比較します。**図3**の例では、このメールを送信したSMTPサーバのIPアドレスは「192.0.2.1」であり、SPFレコードで宣言された「192.0.2.0/24」のアドレスブロックに含まれているため検証は成功します。

SPFはこのようにして正当な送信サーバの検証を行うしくみとして広く利用されていますが、**図4**に示すようにメールが転送された場合に、なりすましメールではない配送であるにもかかわらず検証に失敗してしまう弱点を持っています。**図4**の例では、IPアドレスが192.0.2.1のサーバから送信されたメールはIPアドレスが198.51.100.1のサーバに転送された後に受信者に送信されています。この場合、送信者のエンベロープFromアドレスは変化しませんが、送信サーバのIPアドレスが198.51.100.1となり、SPF検証は失敗してしまいます。

▼図4　転送によるSPF検証の失敗例



▼図5　DKIM検証の流れ



DKIM

DKIM(Domainkeys Identified Mail)は、電子署名を用いた送信ドメイン認証のしくみであり、STD 76で標準化されています。DKIMでは送信側で、メッセージのヘッダや本文を用いて生成した電子署名を電子メールに付加し、受信側でその電子署名を照合することによって送信ドメイン認証を行います。このためDKIMでは、中継するMTA(メール転送エージェント)で電子メールのデータが変更されない限り、転送メールであっても転送先で認証が可能です。

DKIMによる検証の流れを**図5**に示します。送信側はあらかじめ、署名に使用する秘密鍵と対になる公開鍵をDNS権威サーバで公開します(**図5-①**)。送信側がメールを送信する際には、メールの本文とヘッダを元に電子署名を付与して送信します(**図5-②**)。受信側のメールサーバはメールを受信すると、DKIM-Signatureの「d=タグ」で指定されたドメイン名のDNS権威サーバへ公開鍵を問い合わせます(**図5-③**)。なお、送信側がd=タグで指定するドメイン名はメール送信サーバと同一でなくてもよく、第三者ドメイン名を指定することができます。たとえば、

メール送信ドメイン名がexample.jpの場合でも、d=タグでは別のドメイン名であるexample.comを指定可能です。受信側は公開鍵を取得すると(**図5-④**)、その公開鍵を用いて電子署名を照合し、検証を実施します(**図5-⑤**)。

DMARC

DMARC(Domain-based Message Authentication, Reporting and Conformance)は、送信ドメイン認証(SPF・DKIM)を用いたレポーティングおよびポリシ制御のしくみであり、RFC 7489で標準化されています。DMARCは、前述したSPFやDKIMのように直接的な認証技術とは異なり、SPFやDKIMを利用して、検証に失敗したメールを受信側がどのように取り扱うかの方針を、送信ドメイン管理者側が宣言するためのしくみです。このため、DMARCは受信側のなりすましメール対策というよりも、送信側のドメイン管理者が自ドメインのブランド力確保を狙うための技術であると言えます。

DMARCには、このような送信ドメイン認証失敗時の制御ポリシに従ったメール配送制御の方法を宣言する機能に加えて、希望する送信側のドメイン管理者へ送信ドメイン認証のエラーレポートや、DMARC検証結果の集計レポートをXML形式で送付するレポーティング機能があります。

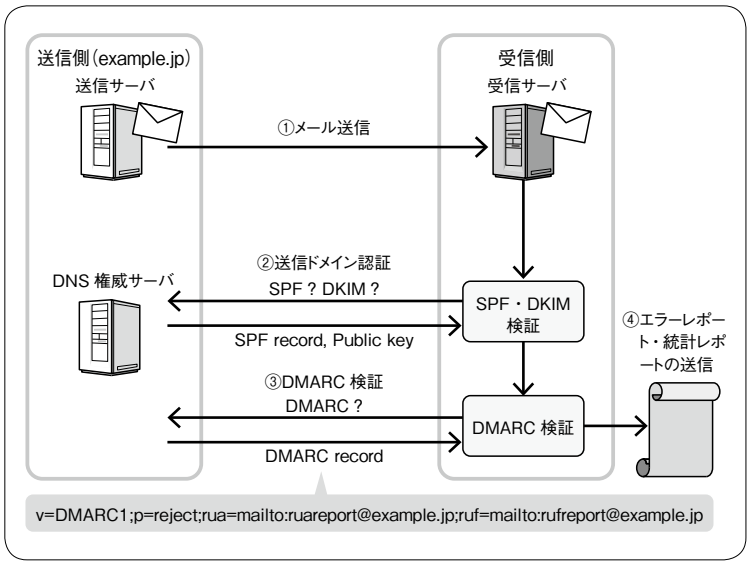
DMARCを利用するには、送信側と受信側でそれぞれ次のように対応する必要があります。送信側ではSPFおよびDKIMの両方、もしくはいずれか一方に対応する必要があります。これに加えて、送信側では自ドメインの先頭に「_dmarc.」を付加したドメイン名(**図6**の例では_dmarc.example.jp)のTXTレコードとして、DMARCレコードを公開します。DMARCレコード内の「p=タグ」で送信ドメイン認証(SPF・DKIM)に失敗したメールに対する受信側の処理ポリシを宣言することができます。宣言可能な処理ポリシは次の3通り存在します。

- none(処理方法を指定しない)
- quarantine(隔離する)
- reject(受信拒否する)

これらの処理ポリシはあくまで送信側が宣言するものであるため、受信側での実際の処理は運用方針によってさまざまですが、たとえば処理ポリシがquarantineの場合、受信側では迷惑メールボックスに隔離する方法が考えられます。また、処理ポリシがnoneの場合、受信者は通常どおり受信することになるためDMARC導入による受信拒否効果はありませんが、送信側がレポーティング機能を利用して統計情報やエラー情報を得るために設定していると考えられます。また、筆者らの調査[注1]によると、DMARCポリ

注1) Naoya Kitagawa et al., Design and Implementation of a DMARC Verification Result Notification System, Proceedings of the 13th APAN Research Workshop (APAN-RW2016), pp.8-14, Aug. 2016.

▼図6 DMARC検証



シを公開しているドメインのうち、およそ80%のドメインが処理ポリシをnoneとして公開しており、rejectやquarantineに設定した際に発生するFalse Positive(誤判定)を防ぐために慎重な設定としていることが多いのが現状です。

DMARC検証の流れを**図6**に示します。受信側のメールサーバでメールを受信すると、まず送信ドメイン認証(SPF・DKIM)を実施します。これらの送信ドメイン認証の両方に失敗した場合、送信側が公開しているDMARCポリシ(**図6**ではreject)が適用されます。また、送信ドメイン認証に失敗した場合にはエラーレポートが「rufタグ」で示されたメールアドレス(**図6**ではrufreport@example.jp)に対して送信され、送信ドメイン認証の集計レポートはDMARCレコード内の「ruaタグ」で示されたメールアドレス(**図6**ではruareport@example.jp)に対して送信されます。

S/MIME

S/MIME(Secure/Multipurpose Internet Mail Extensions)は、電子メールの暗号化による第三者の覗き見防止の役割と、電子メールへの電子署名の付加によるなりすましメール送信や内容の改ざんを防止する役割の2つの役割を持つしくみであり、RFC 5751で標準化されています。

暗号化では、送信者はあらかじめ受信者が公開している公開鍵を取得して、これを暗号鍵としてメール本文を暗号化して送信します。受信者はメールを受信すると、秘密鍵を用いて暗号化されたメール本文を復号します。これにより、本来の受信者以外はメール本文を閲覧することは不可能となり、配送中に第三者による覗き見を防ぐことができます。

電子署名の付加では、送信者はメール本文を元にハッシュ値を生成し、これを送信者の秘密鍵で暗号化したものを添付ファイルとしてメール本体とともに送信します。受信者はメールを受信すると、送信者の公開鍵で添付ファイルを復号し、これがメール本体から生成したハッシュ値と一致していれば、意図した送信者から送信されていることと、転送中に内容が改ざんされていないことを確認することができます。

まとめ

本章では、なりすましメール対策として利用されているしくみについて解説しましたが、いずれの手法も完全なものではなく、それぞれ弱点を持っています。

SPFでは転送メールを正しく認証できない点が弱点として挙げられますが、DKIMにもさまざまな弱点が存在します。たとえば、メーリングリストではメールの件名欄の先頭に、「[MLname:123]」のようにメーリングリスト名や通し番号が付加されることが多くありますが、この場合DKIMでは内容の改ざんと見なされるため認証に失敗してしまいます。さらに、DKIMでは先述のとおり第三者による署名が可能ですが、その第三者が正当な署名先であるかどうかを判断するのは極めて困難であるのが現状です。この問題に対して、DMARCではアラインメントと呼ばれる概念が存在し、第三者署名は禁止されています。ところが、DKIMに対応している非常に多くのドメインは第三者署名が許可されており、これがDMARCの普及を妨げる大きな原因となっています。

また、S/MIMEはSSL証明書を受信者のローカル環境に保存して使用するため、Gmailなどのほとんどのwebメールではs/MIMEに対応していないのが現状です。S/MIMEに対応していない環境では、受信者はメールに添付された「smime.p7s」という署名データを開いても署名を確認することができません。このように受信者の環境への依存が強いことがS/MIMEの普及を妨げる原因の1つとなっています。SD

第2章

なりすましメール防止のためのホワイトリスト活用と可視化などの取り組み

Author 大泰司 章（おおたいし あきら）
一般財団法人日本情報経済社会推進協会（JIPDEC）インターネットトラストセンター 企画室長
URL https://itc.jipdec.or.jp/

第1章でなりすましメールを見破るしくみを解説しましたが、認証技術をかいくぐってくるメールもたくさんあります。そこで本章では、送信ドメイン認証に加えて送信ドメインのホワイトリストを使うことで、メールの安全性を保証するしくみを紹介します。

ホワイトリストと可視化の必要性

第1章で述べられているように、SPFやDKIMといった送信ドメイン認証には、メールの送信元を詐称できなくするというメリットがあります。しかし、悪意を持った送信者（ここではスパマーと呼びます）も、メールを確実に届けて読んでもらうために、SPFやDKIMを使うことができます。それでは、送信ドメイン認証には意味がないということになるのでしょうか？

いえ、そうではありません。送信ドメインを詐称できなくなったことで、スパマーは堂々とスパマー自身のドメインを名乗ってきているともいえます。あとは、正しい送信者のドメインと、スパマー自身のドメインとを区別できればいいということになります。

スパマーは、受信者をだますために、正しいドメインにできるだけ似せたドメインを使ってきます（Cousin Domain Name）が、受信者は、送信者の正しいドメインを知っていれば、送信ドメインを目で確認することで、正しいメールを見分けることができます。

しかしながら、受信者がメール1通1通のドメインを確認するのは至難の業ですし、そもそも業務効率上望ましくないでしょう。

そこで、送信ドメインのホワイトリスト（信頼できる送信ドメインのリスト）を作り、それと自動的に照合し、かつ、その照合結果を受信者にわかりやすく表示することができれば、受信者にドメインの確認の負担をかけずにすみます。

ヤフー、ニフティでの「安心マーク」表示による可視化事例

ヤフーのYahoo!メールやニフティの@niftyメールでは、DKIMで検証を通ったメールについて、そのドメインをホワイトリストと自動的に照合し、一致したものには「安心マーク（**図1**）」と送信元を表示しています（**図2**、**図3**）。

ホワイトリストはISP（Internet Service Provider）が個別に持つのではなく、「サイバー法人台帳ROBINS[注1]」を使っています。ROBINSは、JIPDECが法人情報の基盤整備事業として整備しているもので、法人、団体、個人事業者の名称、法人番号、住所のほか、メール送信に使うドメイン、Webサイトのドメイン、SNSのアカウント、英字名称などのオフィシャルな情報を提供しています。

▼図1　安心マーク

このしくみを有効にするためには、送信者はあらかじめROBINSに送信ド

注1）https://robins.jipdec.or.jp/robins/

メインを登録しておく必要があります。Yahoo!メールや@niftyメールはAPIを使い、一定の間隔でROBINSから送信ドメインのホワイトリストを自動的に取得しています(**図4**)。

「安心マーク」が始まったきっかけは、2013年5月に施行されたネット選挙運動の解禁です。政党や候補者が電子メールによる選挙運動を行うにあたり、送信者側と受信者側の双方が安心して電子メールを利用できる環境を整備する必要性から、まず各政党からのメールマガジンに「安心マーク」をつけました。

2014年8月からは、銀行を装ったなりすましメールによって虚偽のWebサイトへ誘導され、暗証番号やパスワードなどを詐取されるフィッ

▼図2 Webメール表示イメージ(PC)

■ニフティ株式会社(@niftyメール)

＜メール一覧画面＞

＜メール一覧画面(安心マークマウスオーバー時)＞

■ヤフー株式会社(Yahoo!メール)

＜メール一覧画面＞

＜メール詳細画面＞

▼図3 Webメール表示イメージ(スマートフォン)

■ニフティ株式会社(@niftyメール)

■ヤフー株式会社(Yahoo!メール)

シング詐欺などの被害への対応として、銀行からのメールにも「安心マーク」をつけることが始まりました。現在では、常陽銀行、スルガ銀行、イオン銀行が対応しており、他行でもDKIMやDMARCの実装と並行して導入の検討が進んでいます。

2016年8月には、愛媛県上島町が地方自治体で初めて「安心マーク」を採用し、今後、官公庁など公的な部門でも普及が見込まれます。

「安心マーク」を導入することにより、なりすましメールの被害を減らせるというメリット以外にも、「受信者に安心してメールを開いていただけるということで開封率やクリック率が向上した」また、「受信者がメールの真偽についてコールセンターへの問い合わせをする際の対応工数が減った」、「送信側企業のイメージアップにつながった」というような効果も出ています。

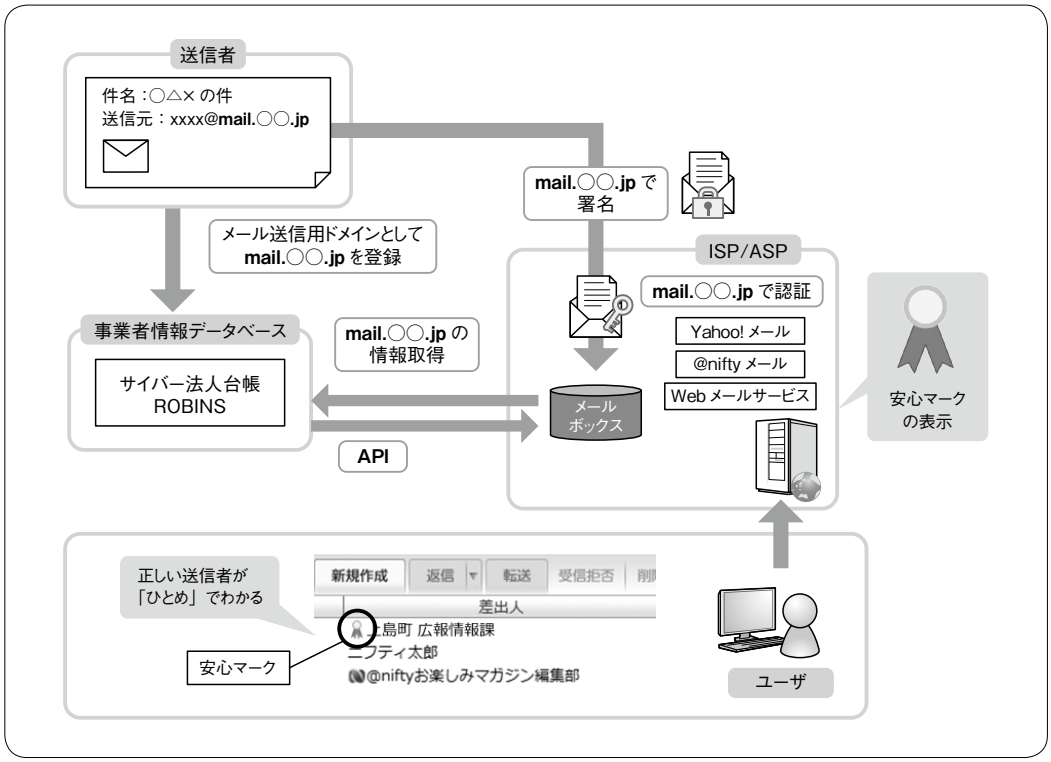
メールの安心・安全の実現に向けた今後の取り組み

2017年1月には、送信者、受信者、メールのソリューションを提供するベンダーなどの関係者が集まって「安心マーク推進フォーラム」を立ち上げました。ここでは、メールの安心・安全のために、さらに次のような取り組みを進めていく予定です。

■「安心マーク」登録の審査基準の策定と運用

これまでの「安心マーク」は、銀行や公的機関など、その実在も信頼性も疑いの余地がない組織でした。今後さらに他業界にも普及させてい

▼図4　安心マークのしくみ



※APIの例：https://robins.jipdec.or.jp/robins/api/dkim_sample?ref_id=XXXXXX
APIはJIPDECと契約することにより開示されます（今後、契約なしでも利用できるテスト用APIを公開する準備もしています）。

くためには、ドメインを登録する際の審査基準を策定して適用していく必要があります。また、適切ではないメールの送り方をした場合には、登録を取り消すといった運用も必要になると思われます。

DMARCへの対応

現在はDKIMのみの対応となっていますが、今後、受信側ISPがDMARCに対応していくことと歩調を合わせて、安心マーク自体もDMARCに対応していきます。

フィードバックループの実装とドメインレピュテーション

フィードバックループとは、受信側ISPなどが、メール受信者から迷惑なメールについて報告を受け、さらに送信者などにフィードバックするしくみです。スパマーにはフィードバックされません。こうしたフィードバックをもとに、送信ドメインを評価（ドメインレピュテーション）し、ホワイトリストに反映させることも検討します。

ディスプレイネーム詐称対策

ディスプレイネームの詐称に対しては、ホワイトリストに載っているドメイン以外のディスプレイネームは表示しない、あるいは色分けをする、といった実装を受信側で行うアイデアが出ているところです。

国際標準化への取り組み

上記のような取り組みは、受信側のメーラのユーザインターフェースによるところが大きく、共通化、標準化が難しい分野ではありますが、IETF（Internet Engineering Task Force）やM3AAWG（Messaging, Malware and Mobile Anti-Abuse Working Group）など国際標準化の枠組みの中で、日本発の取り組みとして情報発信をしていくべきだと考えています。

補足：S/MIMEへの対応について

受信者が、ISPが提供するWebメールやスマホアプリでメールを受信した場合は「安心マーク」を表示するといったことが可能ですが、Outlook、ThunderbirdなどのPCのメーラで受信した場合には、現状では、それができません。

一方で、こうしたPCのメーラは、標準でS/MIMEに対応をしていますので、送信者はS/MIMEの署名を行い、受信者はその表示を確認すればよいということになります。

ところが、実はメールアドレスさえあれば、誰でもS/MIME用の電子証明書は取得できてしまいますので、やはり類似のドメインにだまされるということは起こる可能性があります。

そこで、「安心マーク」同様にドメインのホワイトリストを利用して、信頼できる送信者からのメールであることを、受信者にわかりやすく表示をすることが有効だと考えられます。現在、そのためのアドオンを開発しているところです。Thunderbird版については、近々公開ができる見込みです。

送信ドメイン認証とS/MIMEのどちらがなりすましメール対策に有効かという議論もありますが、受信環境によって、どちらかが検証できたりできなかったりということもありますので、送信者は両者に対応することをお勧めします。JIPDECでは「エスマいぬ」と「ディーキぃぬ」というキャラクターを作って両者の普及活動をしているところです（図5）。SD

▼図5　なりすまし対策ステッカー

第3章

プログラマには何ができる？

Webメーラーへのセキュア機能実装例

Author 峰松 浩樹（みねまつ ひろき）　有限会社ランカードコム
URL https://lancard.com/　Mail mine@lancard.com

本章では、送信ドメイン認証の機能と送信ドメインのホワイトリストを使った安心マーク表示機能の実装について、Webメールクライアントを使って解説します。

はじめに

この記事では、GPL3ライセンスにて利用可能なWebメールクライアント「Roundcube」へDKIM署名検証機能を追加した方法を例示します。加えて、DKIM署名検証結果をホワイトリストを元にチェックし、送信元ドメインがなりすまされていないかを検証します。これらにより、なりすましメールについて安全・安心な確認機能の実装方法の参考としてください。

Roundcubeとは

jQueryを多用したUIにより、インタラクティブな操作を実現したオープンソースのWebメールクライアントです。GPL3ライセンスにて配布されており、世界中のインターネットプロバイダでもユーザ向けにサービス提供されています。簡単にインストールおよび設定が可能で、標準的なLAMP（Linux、Apache、MySQLもしくはPostgreSQL、PHP）環境で動作するサーバアプリケーションです。

HTML形式のメールも安全に取り扱うために、WebHTML無毒化機能[注1]を内蔵しています。また英語、日本語をはじめ70を越える多言語対応と、組織で使用できる共有アドレス帳にも対応しています。

動作環境としてはApache、Nginxそのほか PHP-5.3.7以降が動作可能なWebサーバと、MySQL、PostgreSQL、SQLiteなどのデータベースが必須です。またdovecotなど、IMAPv4 rev1をサポートするIMAPサーバと、メール送信のためのSMTP接続、またはsendmailコマンドなどによるPHPからのメール送信が可能であることが必要となります。PCRE、JSON、PDO、そして日本語でのメール送受信のために、PHPのmbstringモジュールまたはiconvモジュールが必要です。

Roundcubeのインストール

Roundcubeはバージョン1.0以降はLinuxなど各ディストリビューションにてrpmまたはdpkg形式にてパッケージが用意されていますが、公式サイトにてソースが配布されています[注2]。Version 1.3以降ではメール一覧の表示など大きな変更があり、またpluginもまだ非対応のものが多いため、Version 1.0.9、1.1.7、1.2.3が推奨です（2017年1月10日現在）。以下、ソースからの導入例です（/home/user/roundcube/以下に導入）。

注1）https://github.com/roundcube/roundcubemail/blob/master/program/lib/Roundcube/rcube_washtml.php

注2）https://roundcube.net/download/

```
$ tar xzf roundcubemail-1.1.7-complete.tar.gz
$ rsync -a roundcubemail-1.1.7/* /home/user/roundcube/
```

動作中に必要となるtempディレクトリとlogsディレクトリを作成し、権限を設定します。

```
$ mkdir -p /home/user/roundcube/{temp,logs}
$ chmod 1777 /home/user/roundcube/{temp,logs}
```

httpd側の設定を調整し、http://(install先)/installer/へアクセスすると**図1**のようにインストーラが起動しますので、表示された項目を埋めて生成されたconfig.inc.phpの内容をconfig/config.inc.phpとして保存します。

http://(install先)/へアクセスするとログイン画面が表示されるので、普段IMAPにてメール利用される際のID、パスワードでログインできましたら、安全のためにinstallerのディレクトリを削除またはリネームします。

```
$ rm -rf /home/user/roundcube/installer
```

OpenDKIMの導入について

DKIM(ディーキム)は受信したメールが正当な送信者から送信されたメールか、差出人のメールアドレスを含むメールヘッダおよび本文が改ざんされていないかを識別する技術です。差出人のメールアドレス(必須)を含むメールヘッダ、本文から生成されたハッシュ値に電子署名を行い、受信側でその電子署名とハッシュ値を検証することによって差出人のメールアドレスや内容が改ざんされていないかチェックします。

DKIM、SPF、DMARC(第1章にて解説)などの送信メールサーバの信頼性を付与するしくみを通過したメールには、受信側メールサーバにて「Authentication-Results:」ヘッダが付与される場合があります。この場合はすでにDKIMのための「DKIM-Signature:」ヘッダの検証ができています。受信側メールサーバにてAuthentication-Results:ヘッダが付与される設定でない場合に、メールサーバへOpenDKIMなどのフィルタを導入する方法と、MUA(Mail User Agent)側でDKIMの検証を行う方法があります。

Roundcubeにて利用するIMAPサーバから取得したメールに、「DKIM-Filter:」ヘッダが付与されている場合は、すでにフィルタが導入済みと考えられます。MUAにて都度DKIM-Signatureヘッダを検証するより、メールフィルタにて検証されているほうが速度的に有利であるため、以降ではメールサーバとしてPostfixが導入されている環境へ、フィルタとしてOpenDKIMを新規に導入する手順について簡単な解説を行います(CentOS 6.8を想定)。

受信メールの検証だけであれば、公開鍵の設定など必要ありませんが、DKIMの署名検証はメールサーバ運用上すでに必須と考えられますので、併せて設定するのが良いでしょう。

まず、OpenDKIMのインストールにあたりEPELリポジトリを追加します。

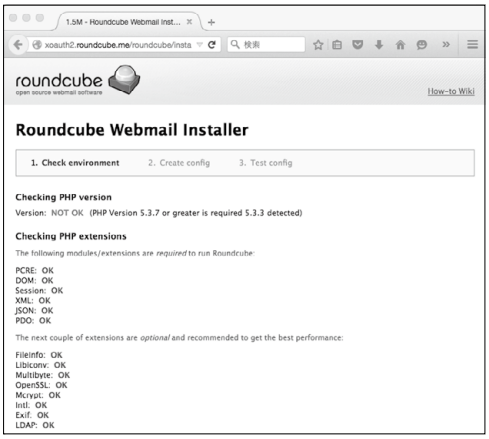
```
$ sudo yum install epel-release
```

OpenDKIMパッケージはEPELに含まれていますので、引き続きyumコマンドにてインストールします。

```
$ sudo yum install opendkim
```

DKIM署名用の秘密鍵と公開鍵をドメインご

▼図1 インストーラ画面



とに管理するためのディレクトリを作成します(次の例はexample.comドメインの場合)。

```
$ sudo mkdir /etc/opendkim/keys/example.com
```

署名用の秘密鍵、公開鍵を生成します。

```
$ sudo opendkim-genkey -D /etc/opendkim/keys/example.com -d example.com -s 20170118
```

-Dにて格納先のディレクトリ、-dにて対象となるドメイン名、-sはセレクタ名(生成日の日付などがよく用いられます)を指定します。

生成された各鍵ファイルが実行中のOpenDKIMから読み込み可能であるように権限を変更します(defaultではopendkim:opendkimとなります)。

```
$ sudo chown opendkim:opendkim /etc/opendkim/keys/example.com/20170118.private
$ sudo chown opendkim:opendkim /etc/opendkim/keys/example.com/20170118.txt
```

生成された公開鍵ファイル(拡張子が*.txtであるほう)の内容を、公開鍵レコードとしてDNSサーバの対象となるドメイン名のzoneファイルに設定します。

```
$ sudo cat /etc/opendkim/keys/example.com/20170118.txt
20170118._domainkey     IN      TXT     ( "v=DKIM1; k=rsa; "
  "p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC4nmPHIsZg ...略... lKTYSUEMK3znV8OLCK8SpxynyRb7t/48wIDAQAB" )  ; ----- DKIM key 20170118 for example.com
```

公開鍵レコードの書式は**リスト1**のようになっており、公開鍵ファイルの内容は書式に合わせた内容となっています。このテキストから'('と')'を取り除き、v=DKIMの行末尾とp=の行頭の'"'を取り除いてください。設定したレコードが反映されているか確認します。

```
$ dig 20170118._domainkey.example.com txt
 ...略...
;; ANSWER SECTION:
20170118._domainkey.example.com 300 IN TXT "v=DKIM1\; k=rsa\; p=MIGfMA0GCSqGSIb3DQEBAQUAA4 ...略... 8SpxynyRb7t/48wIDAQAB"
 ...略...
```

レコードが正しくひけていれば大丈夫です。

opendkimの設定ファイル/etc/opendkim.confを次のように調整します(変更が必要な行のみ)。

```
Mode      sv            検証だけなら v
Domain    example.com   署名したいメールドメイン名
Selector  20170118      セレクタ名
KeyFile /etc/opendkim/keys/example.com/20170118.private   秘密鍵ファイル名
```

/etc/opendkim/TrustedHostsのInternalHostsの行を#を取り除いて有効にし、信頼できる送信元IPアドレス群を次のようにファイルに列挙します。

```
127.0.0.1
::1
192.168.0.0/16
```

opendkimを自動起動に設定し、起動します。

```
$ sudo chkconfig opendkim on
$ sudo service opendkim start
```

opendkimによるDKIM署名のチェックがpostfixでのメール受信時に行われるよう、milter(mail filter)の設定としてopendkim側を、

```
smtpd_milters = inet:127.0.0.1:8891
```

のSocket設定に合わせて設定を追加します。

postfixを再起動して設定を有効化します。

```
$ sudo service postfix restart
```

dkimstatus_plus プラグインの追加

受信したメールにAuthentication-Results:

▼リスト1　公開鍵レコードの書式

```
セレクタ名._domainkey IN TXT "v=DKIM1; k=rsa; p=公開鍵の文字列"
```

ヘッダも付加されておらず、メールサーバへのOpenDKIMのインストール権限がない場合は、MUA(メールクライアント)のみにてDKIM-Signature:の検証をせざるを得ません。今回は、dkimstatus_plusプラグインによるDKIM署名検証機能を使用します。

以降では、Roundcubeへのdkimstatus_plusプラグインの導入手順について紹介します。

まず、Roundcubeがインストールされているディレクトリ配下にpluginsというフォルダがありますので、GitHubからcloneしたdkimstatus_plusプラグインをコピーします(以下は、/usr/share/roundcubemail以下にroundcubeがインストールされている場合)。

```
$ git clone https://github.com/minemaz/dkimstatus_plus.git
$ sudo rsync -a dkimstatus_plus /usr/share/roundcubemail/plugins/
```

このプラグインはPHP用のphp-opendkimモジュールを使用しますので、インストールします。

```
$ git clone https://github.com/minemaz/php-opendkim.git
$ cd php-opendkim
$ phpize && ./configure && make && make install
```

PHPへopendkim.soをmoduleとして読み込ませるため、次の内容をphp.iniへ追記します(CGI/FastCGI SAPIの場合は「.user.ini」への追記でも良いでしょう)。

```
extension=opendkim.so
```

Roundcubeの、config/config.inc.phpファイルへプラグイン名'dkimstatus_plus'を追加します(**リスト2**)。

このプラグインを使用すると、Roundcubeの受信メール一覧の送信者名の個所にDKIM署名の検証結果によってアイコンが付加され、送信元の検証ができているメールであるかをひと目で確認できます(**図2**)。また、アイコンの上にマウスカーソルを重ねることで、署名の詳細が表示されます(**図3**)。

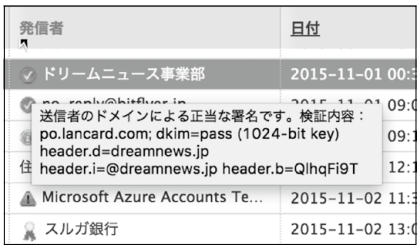
この実装でのアイコンの意

▼リスト2 プラグイン名'dkimstatus_plus'の追加(config/config.inc.php)

```
// ----------------------------------
// PLUGINS
// ----------------------------------
// List of active plugins (in plugins/ directory)
$config['plugins'] = array('jqueryui', 'dkimstatus_plus');
```

▼図2 プラグインによるアイコン表示

▼図3 正当な署名が付与されている場合の表示例



Column Author Domain Signing Practice(ADSP)について

OpenDKIMの設定について多くのインターネット上の記事では、DNSサーバへのDKIM-ADSP情報の設定について記載されている場合がありますが、ADSPについては「歴史的な」技術として、その役割はDMARCへ引き継がれています。また、OpenDKIMでも2.10以降のバージョンではADSPについてのコードが取り除かれています(http://lists.elandsys.com/archive/opendkim/users/2014/12/3348.html)。さらに、RFC 7489においても運用上の問題点について指摘されています(http://salt.iajapan.org/wpmu/anti_spam/admin/tech/rfc/rfc7489/page-54/)。

味は次のようになっています。

■正しいDKIM署名として検証済の場合

青いチェックマークのアイコンで表示。

■不正または検証できない署名

赤い三角の警告アイコンで表示。DNSにセレクタが登録されていない場合やメーリングリストなどでメール内容が変化した場合、ヘッダが壊れている場合などが該当します。

■Fromヘッダと異なる第三者ドメインによる署名

灰色にチェックマークのアイコンで表示。メール配送代行業者などにみられ、配送代行業者のDKIM署名が付与されている場合が該当します。配送代行業者が信頼できる先でない場合は「なりすまし」の可能性もあります。

■正しいDKIM署名であり、ホワイトリストにも登録されている

緑にチェックマークのアイコンで表示。DKIM署名のチェックを厳密に行いたい送信元ドメインについて、ホワイトリストとして列挙する設定がdkimstatus_plusプラグインにあり、配列形式で追記できます(**リスト3**)。ホワイトリストに記載があるのにDKIM署名が行われていない場合は「なりすまし？」警告表示が行われます。

ROBINSデータベースについて

ROBINSデータベースにて第三者による各種確認ができている場合には「安心マーク[注3]」を提示でき、DKIM署名の検証結果と合わせ、安全・安心なメール送信元ドメインとして取り扱うこともできます。今回導入した、dkimstatus_plusプラグインのROBINSデータベース連携機能を有効化すると、Roundcube上でもメール検証結果のリンクから企業情報のページを簡単に参照することができます。

なお、ROBINSのAPI仕様およびAPIキーについてはJIPDECまでお問い合わせください。

おわりに

DKIM署名検証によってドメインの正当なメール送信者であることが保証されても、ドメインそのものが安全であるかどうかは評価できません。詐欺サイトであってもDNSサーバの設定が可能であればDKIM署名を付与することは可能だからです。そのため、ROBINSのようなドメイン名運用者の信頼性を確認するしくみが必要となります。また、メール送信者各々がなりすまされていないかについては、S/MIMEのような送信者ごとの証明書を用いたしくみも利用が推奨されています。

これら電子署名の付与は徹底されないと、正しいメールと「なりすまし」の判別を逆に難しくしてしまうため、組織のメールサービス運営者はなりすましメールへの十分な対応が必要になっています。SD

注3) https://robins-cbr.jipdec.or.jp/usecase/anshinmark/index.html

▼リスト3　dkimstatus_plusプラグインのホワイトリスト設定例

```
<?php
// 'ドメイン名' => array('企業名')
$config['dkim_whitelist'] = array(
    'jipdec.or.jp' => array('一般財団法人日本情報経済社会推進協会'),
    'mail.yahoo.co.jp' => array('ヤフー株式会社'),
    'nifty.com' => array('ニフティ株式会社'),
    'lancard.com' => array('有限会社ランカードコム')
);
```

特別企画

身近な金融システム

Amazonログイン&ペイメントのしくみ

その決済方法の内側を知る

Author 塩田 修一(しおた しゅういち)
アマゾンジャパン合同会社

Amazonログイン&ペイメントは、Amazonでしか利用できないサービスだと思い込んでいませんか。実は総合オンラインストア「Amazon.co.jp」に登録されたアカウントでアマゾン以外のECサイトにログインができ、簡単に決済ができるサービスです。本稿では、そのしくみと特徴、そしてメリットを紹介します。

Amazonログイン&ペイメントとは何か

Amazonが日本で外部に提供している開発者向けツールとしては、アマゾンウェブサービスジャパン㈱が提供しているAWSがありますが、このAmazonログイン&ペイメント(以降、Amazonペイメント)は、アマゾンジャパン合同会社として外部向けに提供している唯一の開発者向けツールです。このAmazonペイメントは、日本におけるサービスであるAmazon.co.jp(以降、アマゾン)に登録されているアカウントで他のECサイトにログインし、すでに紐付いているクレジットカード情報を利用し、簡単に決済できます。つまり他のECサイトでもアマゾンのアカウントでショッピングができるサービスです。

Amazonペイメントを導入するメリット

アマゾンで取り扱いのない商品を販売しているECサイトにとって、アマゾンの顧客を新たに獲得できる可能性がある、というのが最初に挙げられます。また、アマゾンと自社サイトの両方で販売する場合はどうなるのか、という問い合わせを多く受けますが、実際に導入した結果、自社ECサイトの注文件数は増え、アマゾンでの注文件数は変化しませんでした。このようにアマゾン上で商品を販売していても導入するメリットがあることがわかりました。

そのほかAmazonペイメント導入メリットは次のとおりになります。

- アマゾンに登録されているアカウントのメールアドレスを含む購入者情報をサイト側に保有できる。そのため自社のマーケティングに活用できる
- クレジットカード情報をECサイト側で保持しないので、安心して購入できる。事業者側は管理が容易になる(PCI DSS 3.1にも準拠)
- カードの不正利用があった場合、アマゾン側で限度額以内で保証される
- アマゾンに販売された商品情報を連携する必要がない。決済情報だけ連携されるので自社のマーケティング情報の秘密が保持される
- Amazonペイメントの利用について問題が発生したときは、アマゾンの顧客サポートが対応する
- アマゾンに登録されている住所やクレジットカード情報を新たに入力する手間がなくウィ

ジェットから選択するだけなので、ユーザの入力手間が大幅に削減でき、新規顧客の獲得が期待できる

ECサイト事業者にとって、とくに大きなメリットは「すでにアマゾンで買い物経験がある、ネットショッピングに慣れたユーザの獲得が期待できる」ことです。しかし、導入にはいくつか条件があります。

Amazonペイメント導入の条件

Amazonペイメントを導入するには、Webサイト側のカスタマイズが必要です。

とくにAPIを利用してAmazonペイメントのサービスと通信しながら、処理する必要があるため、少々規模が大きいカスタマイズになります。さらに、すでに存在するショッピングサイトの仕様に合わせるとAmazonペイメントのメリットを享受できない場合があり、新たに購入フローを作成しなおして(そう判断された場合)、大きめの開発工数が発生する可能性があります。

こうした場合、解決策として、簡単に導入できるASPやパッケージのプラグインを提供しているソリューションプロバイダもあります(後述しますが、Amazonペイメントでは、このようなソリューションプロバイダを募集しています。パートナープログラムも提供しています)。

Amazonペイメントの技術要素

Amazonログイン&ペイメントは、Amazonログインと Amazonペイメントが統合されたサービスです。Amazonログインでは、「Amazonアカウントでログイン(**図1**)」か「Amazonアカウントでお支払い(**図2**)」というボタンを表示して、Amazonから提供しているログインポップアップ画面(**図3**)を表示してログイン処理を行います。

Amazonログイン処理が正常に処理できた場合は、次にAmazonペイメントのアドレス帳ウィジェットとお支払い方法ウィジェットを表示します(**図4**)。この2つのウィジェットは別々のページに表示しても良いですが、現在の購入フローの主流は、ワンページチェックアウトと呼ばれていますので、カートの次画面でこの2つのウィジェットを表示することを勧めています。

ショッピングサイトでのフロント側としては上記の部分とサーバ側でのオーソリ(認証)処理までです。以降はバックエンド側の処理になります。バックエンド側では注文データを確認して出荷や返品処理、顧客管理をします。このバックエンド側では出荷の前後で売上を確定する決済

▼図1　Amazonアカウントでログインボタン

▼図2　Amazonアカウントでお支払いボタン

▼図3　Amazon.co.jpへのサインイン(ログイン)ウィンドウ

▼図4　Amazonペイメントのアドレス帳ウィジェットとお支払い方法ウィジェットの表示

処理をします。クレジットカード決済では、よく実売上と呼ばれているものであり、Amazonペイメントでは売上請求処理と呼びます。こちらはAPIを呼び出して対応します。返品の場合はお客様に返金の可能性があります。この場合も同様にAPIを呼び出して対応することになります。

このように、Amazonペイメントでは、このフロント側でCSS/HTML、JavaScriptを利用してウィジェットを表示し、サーバ側でAPIを利用して取引情報を生成します。また、バックエンド側でAPIを利用して売上請求処理や返金処理などを実行します。

Amazonペイメントの API

Amazonペイメントでは、取引データの設定、生成から返金、レポート生成などを行うAPIが多数用意されています。このAPIはアマゾンから提供しているAmazonマーケットプレイスWebサービス(MWS)のプラットフォーム上で動作します。このMWSはアマゾン上での出品サービスで利用するためのサービスであり、アマゾン上で商品の出品、注文、支払い、レポートを処理するためのものです。

このMWSの中のAmazonペイメントAPIを利用してシステム開発をします。MWS全般についての説明は今回は省きますが、興味がありましたらMWSの情報は公開[注1]されていますので、参照してください。

Amazonペイメントで利用できるAPIは**表1**～**表4**のとおりになります。

ここで「ワンタイム支払い」と「定期(自動)支払

注1) URL http://services.amazon.co.jp/services/automation/for-developer.html

▼表1　ワンタイム支払い用API

API名	処理概要
SetOrderReferenceDetails	決済金額や注文説明などの注文の詳細をOrder Referenceにセットする
ConfirmOrderReference	必須情報がセットされたOrder Referenceを承認する
GetOrderReferenceDetails	Order Referenceオブジェクトの詳細と現在の状態を取得する(定期でも利用)
CancelOrderReference	以前に承認されたOrder Referenceをキャンセルする
Authorize	Order Referenceに保存されている支払方法に対して指定した金額を確保(オーソリ)する
GetAuthorizationDetails	オーソリオブジェクトの詳細と現在の状態を取得する(定期でも利用)
CloseAuthorization	オーソリオブジェクトを終了(クローズ)する(定期でも利用)
Capture	オーソリされた支払方法から金額を売上請求し事前に登録されている銀行口座に資金を移動する(定期でも利用)
GetCaptureDetails	売上請求オブジェクトの詳細と現在の状態を取得する(定期でも利用)
Refund	売上請求された資金を購入者に返金する(定期でも利用)
GetRefundDetails	返金オブジェクトの詳細と現在の状態を取得する(定期でも利用)

▼表2　定期(自動)支払い用API

API名	処理概要
SetBillingAgreementDetails	自動支払いの注文説明などの注文の詳細をBilling Agreementにセットする
ConfirmBillingAgreement	必須情報がセットされたBilling Agreementを承認する
GetBillingAgreementDetails	Billing Agreementオブジェクトの詳細と現在の状態を取得する
CreateOrderReferenceForId	指定されたオブジェクトからOrder Referenceを生成する。現在はBilling Agreementオブジェクトからだけ許可されている
ValidateBillingAgreement	Billing Agreementオブジェクトの状態と登録された支払方法を確認する
CloseBillingAgreement	Billing Agreementオブジェクトを終了し新しい取引データを生成できないようにする
AuthorizeOnBillingAgreement	Billing Agreementオブジェクトに保存されている支払方法に対して指定した金額分のOrder Referenceを生成する

い」が出てきましたので、この違いを説明します。2015年5月にリリースした時点では、この「ワンタイム支払い」だけ利用可能でした。これは単品購入の機能であり、1回限りの決済だけに利用できます。一方「定期(自動)支払い」は、2016年10月から利用開始になり、雑誌の継続購読や化粧品、健康食品などの定期購入などで利用できる機能を新しくリリースしました。

定期契約からオーソリまでの仕様とAPIは、「ワンタイム支払い」と「定期(自動)支払い」では異なりますが、売上請求処理、返金処理は仕様、APIとも両者は同じものを利用します。そのほかの相違点は**表5**のとおりになります。

Amazonペイメントを導入するための仕様、APIリファレンスが必要な場合は「Amazonログイン&ペイメント　インテグレーションガイド」[注2]、「Amazonログイン&ペイメント　定期購入機能インテグレーションガイド」[注3]、「Amazonペイメント　APIリファレンス」[注4]を参照してください。

Amazonペイメント オブジェクトの状態遷移

Amazonペイメントの仕様について各オブジェ

▼表3　共通API

API名	処理概要
GetServiceStatus	AmazonペイメントのAPIの利用状況を返す

▼表4　レポートAPI

API名	処理概要
GetReportRequestList	取得可能なトランザクションと決済レポートの一覧を取得する
GetReport	指定したレポートをダウンロードする
UpdateReportAcknowledgements	2回ダウンロードできないようにAcknowledgementsの値をTrueにセットして、すでにダウンロード済みであることを指定する

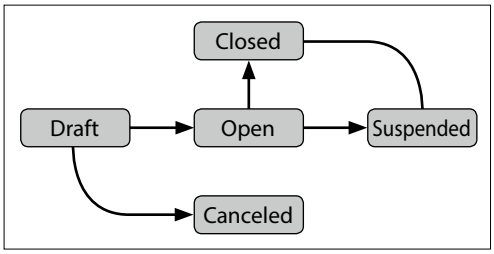
▼表5　ワンタイム支払いと定期(自動)支払いの違い

	ワンタイム支払い	定期(自動)支払い
利用オブジェクト	Order Reference(注文)	Billing Agreement(契約)
	オーソリ	Order Reference(注文)
	売上	オーソリ
	請求	売上請求
	返金	返金
基本となる単位	Order Reference(注文)	Billing Agreement(契約)
制限	1注文最大100万円まで	毎月5万円／契約まで
	注文は最大180日間有効	Billing Agreementの期限なし
	オーソリは最大30日間有効	オーソリは最大30日間有効
	増額は15%か8,400円のどちらか低い額まで可能	注文生成後のOrder Refernece の増額は不可
想定利用商品	単品購入商品	定期商品
	180日以内で出荷される商品	頒布会商品
		181日以上で出荷される商品

注2）URL https://images-na.ssl-images-amazon.com/images/G/09/AmazonPayments/LoginAndPayWithAmazonIntegrationGuide_JP.pdf
注3）URL https://images-na.ssl-images-amazon.com/images/G/09/AmazonPayments/LoginAndPayWithAmazonRecurringPaymentsIntegrationGuide_JP.pdf
注4）URL https://images-na.ssl-images-amazon.com/images/G/09/AmazonPayments/PayWithAmazonApiReferenceGuide_JP.pdf

クトの状態遷移に注意しながら導入開発する必要があります。先に各オブジェクトのしくみについて説明します。筆者はインテグレーションガイド、APIリファレンスを最初に読んだときは、まったく仕様を「理解できへん(関西弁)」でした。Order Referenceオブジェクト、オーソリオブジェクト、売上請求オブジェクト、返金オブジェクトが別々に存在すること、それぞれの状態がオブジェクト内で遷移することがわかりませんでした。通常であれば決済データというものがあり、その中で状態区分らしきものがあり、それが変更されるのが通常だと思ったのですが、Amazonペイメントでは、それぞれの処理が、それぞれのオブジェクトに分かれており、さらにそれぞれのオブジェクト内に状態を持っています。次の**図5**～**図9**と**表6**～**表10**が、APIリファレンスに記載されているそれぞれのオブジェクトの状態遷移になります。

▼図5　Billing Agreementオブジェクトの状態遷移図



とくに意識していただきたいのは、各オブジェクトのリレーションです。ワンタイム支払いの場合と定期(自動)支払いの場合で若干異なりますが、それぞれ**図10**、**図11**のとおりのリレーションになっています。

ワンタイム支払いの場合は、1つのOrder Referenceオブジェクトに対して複数のオーソリオブジェクトが生成でき、1つのオーソリオブジェクトからは1つの売上請求オブジェクトが生成でき、1つの売上請求オブジェクトからは複数の返金オブジェクトを生成できます。

定期(自動)支払いの場合は、1つのBilling Agreementオブジェクトに対して複数のOrder Referenceオブジェクトが生成でき、1つのOrder Referenceオブジェクトからは1つのオーソリオブジェクトが生成でき、以降はワンタイム支払いと同じリレーションになります。

ショッピングサイトとAmazonペイメントの状態を一致させるには

このオブジェクトの状態遷移ですが、ショッピングサイト側の決済状態とAmazonペイメントの状態を一致させる必要があります。多くの注文を受け付け、ショッピングサイト側で注文処理中の場合は、Amazonペイメント側にも多くオブジェクトが存在しており、それぞれの注

▼表6　Billing Agreementオブジェクトの各状態の説明

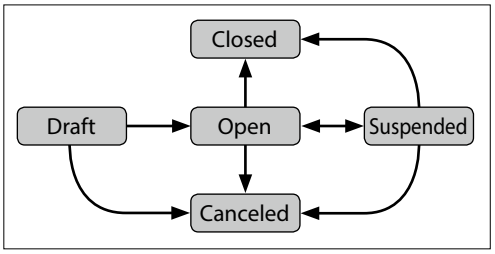
状態	説明
Draft	定期(自動)支払いでのフロント側でAmazonペイメントのウィジェットを表示した段階で、Amazonペイメントから提供しているJavaScript内でBilling Agreementオブジェクトが生成される最初の状態。必ずこの状態から開始され、確定した場合はOpen状態に遷移する
Open	ConfirmBillingAgreement APIの呼び出しに成功した場合にOpen状態になる。この状態になって、初めて以降の定期(自動)支払いの各オブジェクトを利用できる。また、この時点でAmazonペイメントのウィジェットから選択された配送先住所と支払い方法がこのオブジェクトにセットされる。重要なのは、この状態でないとオーソリ処理を要求できないこと
Suspended	AuthorizeOnBillingAgreement APIの呼び出しで失敗した場合にSuspended状態に遷移する。オーソリオブジェクトの状態遷移によって、このBilling Agreementオブジェクトも状態遷移する。オーソリが失敗する例としては、登録されたクレジットカードが有効でない場合
Canceled	Draft状態のBilling Agreementオブジェクトが3時間確認されなかった場合は、Amazonによって自動的にCanceled状態に遷移する
Closed	Open状態、または、Suspended状態から明示的にCloseBillingAgreement APIを呼び出してClosed状態になるか、Amazonペイメントのマイページより購入者が定期契約を解約した場合にこの状態に遷移する。この状態になれば、新しいオーソリを生成することができなくなる。ただし、すでに存在するオーソリオブジェクトから売上請求処理を行うことはできる

文ごとにGet系のAPIを定期的に実行するのは得策ではありません。そこで、Amazonペイメントでは「インスタント支払通知サービス(IPNサービス)」というものを別途用意しています。このIPNサービスを利用すれば、Amazonペイメントの各オブジェクトに対してAPIを実行した結果としてほかのオブジェクトの状態が変更された場合や、自動的に状態が遷移された場合に、指定されたURLへ状態が変更されたことをhttps通信で通知します。このIPNサービスを利用することで、cronなどで定期的にGet系のAPIを実行する必要がなくなります。

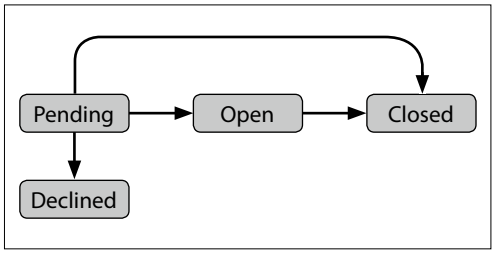
Amazonペイメントを楽に開発するためには

さて、ここまではAmazonペイメントの仕様

▼図6　Order Referenceオブジェクトの状態遷移図



▼図7　オーソリオブジェクトの状態遷移図



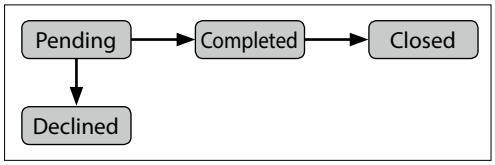
▼表7　Order Referenceオブジェクトの各状態の説明

状態	説明
Draft	ワンタイム支払いでのフロント側でAmazonペイメントのウィジェットを表示した段階で、Amazonペイメントから提供しているJavaScript内でOrder Referenceオブジェクトが生成される最初の状態。必ずこの状態から開始され、確定した場合はOpen状態に遷移する
Open	ConfirmOrderReference APIの呼び出しに成功した場合にOpen状態になる。この状態になって初めて以降のワンタイム支払いの各オブジェクトを利用できる。また、この時点でAmazonペイメントのウィジェットから選択された配送先住所と支払い方法がこのオブジェクトにセットされる。重要なのは、この状態でないとオーソリ処理を要求できないこと
Suspended	Authorize APIの呼び出しで失敗した場合にSuspended状態に遷移する。オーソリオブジェクトの状態遷移によって、このOrder Referenceオブジェクトも状態遷移する。オーソリが失敗する例としては、登録されたクレジットカードが有効でない場合
Canceled	明示的にCancelOrderReference APIを呼び出してCanceled状態になるか、Amazonによって自動的にCanceled状態に遷移する
Closed	明示的にClosedOrderReference APIを呼び出してClosed状態にするか、Amazonによって自動的にClosed状態に遷移する。また、定期(自動)支払いのAuthorizeOnBillingAgreement APIから一時的に生成されたOrder Referenceオブジェクトはすぐに Closed状態に遷移する

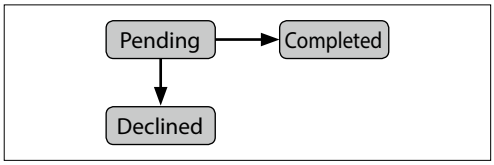
▼表8　オーソリオブジェクトの各状態の説明

状態	説明
Pending	オーソリには同期処理と非同期処理の2種類が存在する。この内の非同期処理の最初の状態がこのPending状態になる。同期処理の場合はこの状態にはならない
Open	非同期処理でオーソリに成功した場合は30秒後にこのOpen状態に遷移する。同期処理の場合は、最初にこの状態か次に説明するDeclined状態のどちらかになる。このOpen状態はオーソリに成功したことになる
Declined	オーソリに失敗した場合はAmazonによってDeclined状態に遷移させる。同時にこのオーソリオブジェクトを生成した元のBilling Agreement、または、Order Referenceオブジェクトは自動的にSuspended状態に遷移させる。このDeclined状態になるのは、登録されているクレジットカードに問題があった場合、タイムアウトの場合、Amazonがオーソリを拒否したなどの理由で発生する
Closed	明示的にCloseAuthorization API、CancelOrderReference APIの呼び出しでこの状態になるか、Amazonによって自動的にClosed状態に遷移する。また、オーソリオブジェクトが30日間Open状態であれば自動的にClosed状態になる

▼図8　売上請求オブジェクトの状態遷移図



▼図9　返金オブジェクトの状態遷移図



▼表9　売上請求オブジェクトの各状態の説明

状態	説明
Pending	オーソリオブジェクトをもとにCapture APIを呼び出すか、オーソリオブジェクト生成時にCapture Nowパラメータを設定して呼び出した場合にPending状態で売上請求オブジェクトが生成される。Amazonによって売上請求の処理が完了するまでこの状態を維持する
Completed	売上請求処理が成功した場合はこの状態に遷移する。同時にオーソリオブジェクトをClosed状態に遷移させる
Declined	売上請求処理に失敗した場合はこの状態に遷移する。この状態になる理由としてはAmazonが売上請求を拒否した場合などがある
Closed	売上請求オブジェクトに対して制限値までの返金を行った場合に、Amazonによって自動的にこの状態に遷移させる

▼表10　返金オブジェクトの各状態の説明

状態	説明
Pending	売上請求オブジェクトをもとにRefund APIを呼び出した場合にPending状態で返金オブジェクトが生成される。Amazonによって返金の処理が完了するまでこの状態を維持する
Completed	返金処理が成功した場合はこの状態に遷移する。返金が完了するまで通常は2、3時間を要する
Declined	返金処理に失敗した場合はこの状態に遷移する。この状態になる理由としてはAmazonが返金を拒否した場合

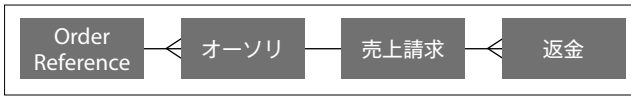
について概要を説明してきました。次からは、実際にシステムを楽に導入開発するための情報を説明します。

現在のショッピングサイトと同じ状態の開発環境があるかと思います。この開発環境でシステム開発を行い、多くのテストを実施するかと思います。これをAmazonペイメントではSANDBOX環境と呼んでおり、テスト用のクレジットカード、テスト用のアカウント、エラーになるクレジットカード情報を提供しています。このSANDBOX環境はAmazonペイメントへの申し込みと審査が完了した販売事業者、および、そのシステム導入開発を行っていただくソリューションプロバイダに無償で提供しています。

さらに、システム開発するうえで、PHP、Java、Ruby、Python、C#のプログラミング言語別のSDKを提供していますので、こちらを利用することで少し容易に開発できるかと思われます。これらのプログラミング言語以外でも、Perlなどのほかの言語で導入したことがあります。SDKの詳しい情報は、「SDKs and Samples[注5]」に表記されているそれぞれのプログラミング言語を確認ください。

注5）英語表記のみ
URL（https://payments.amazon.com/developer/documentation）

▼図10　ワンタイム支払い時の各オブジェクトのリレーション



▼図11　定期（自動）支払い時の各オブジェクトのリレーション

▼表11　日本国内におけるオープンソースへのAmazonペイメントの対応状況

オープンソースプロダクト	対応状況
Magento	世界で最も多く導入されている。ソリューションプロバイダが、Amazonペイメントを利用可能なエクステンションのローカライズを行っている
EC-CUBE	ソリューションプロバイダがEC-CUBEで動作するプラグインを提供している
WooCommerce	WordPress上で動作するショッピングサイト用プラグイン。導入実績が増加しており、2016年11月に日本語対応が完了

Amazonペイメントのモバイルアプリケーション対応

このように「ワンタイム支払い」からスタートし、2016年10月から「定期(自動)支払い」機能もリリースしたAmazonペイメントですが、2016年10月後半から新たにAmazonペイメントのMobile SDKを提供しました。こちらはスマートフォンのアプリケーション内でAmazonログインとAmazonペイメントを利用できるSDKを提供しています。このMobile SDKの特徴は、Amazonショッピングアプリケーションを導入しており、すでにログインしている場合は、このMobile SDKを導入しているアプリケーションで再度ログインが不要になる機能も搭載しています。このMobile SDKのインテグレーションガイドの日本語版は現在翻訳中ですが、2017年1月頃には公開できるよう準備しています。

また、海外のAmazonペイメントでは試験的にオムニチャネルでの利用を開始しています。お客様がモバイルデバイスをリアル店舗に持って入店すると店舗側でも来店を確認でき、商品を店舗で確認してAmazonペイメントで決済し、商品を配送するなどの新しい決済サービスを行っています。

オープンソースへの対応状況

ショッピングサイトの構築をスクラッチから開発することは少ないかと思われます。そこで構築するうえで最初に検討されるのがオープンソースであるかと思います。このショッピングサイト構築用のオープンソースについては、**表11**のとおりにAmazonペイメントはプラグイン形式ですでに対応しています。

Amazonペイメントの公式認定制度

このようにAPIを利用してシステムの導入開発を行っていただくうえで少なからず開発工数が発生します。そこで、ASPやパッケージにあらかじめ導入をしたサービス事業者やシステム開発会社を支援することを目的にAmazonログイン&ペイメントの「グローバルパートナープログラム」を提供しています。このパートナープログラムには「Premier Partners」、「Certified Partners」、「Certified Developers」の3つのプログラムがあります。認定基準は未公開ですが、興味がある方はぜひWebページ[注6]から問い合わせください。

おわりに

Amazonペイメントは、既存のショッピングサイトを少しカスタマイズする必要がありますが、ほかの決済システムと比べると自由度が高いカスタマイズができます。従来の決済システムの代替として、その機能を比較してみるのはどうでしょうか。新しいマーケティングサービスとして見れば、新たなビジネス施策を実現できる技術と言えます。本サービスの導入を検討してみてはどうでしょう。本誌読者の腕と経験ならば、それが可能に思います。SD

注6）URL https://payments.amazon.co.jp/global-partner-program

特別企画

PostgreSQLのGPU拡張モジュール

PG-Stromの構造と機能、そしてその威力とは 前編

計算・集計・解析系処理の高速化

Author 海外 浩平（かいがい こうへい）
PG-Strom Development Team
Twitter @kkaigai

はじめに

ビッグデータという言葉にはいくぶん手垢が付いてしまった感もありますが、日々のトランザクション、モバイルデバイスやセンサー機器などが生成するデータ量は年々加速度的な増加を続け、これらを蓄積し、処理するデータベース管理システムの重要性は年々高まっています。

増大し続けるデータを高速に処理するためさまざまな方法が提案されていますが、本稿では、GPUを使用することでPostgreSQLの計算能力を増強し、集計・解析系処理を高速化するというユニークなアプローチを採っているPG-Strom（ぴーじー・しゅとろーむ）を紹介します。

PG-Stromの背景——GPUの特徴を知る

PG-Stromとは、PostgreSQL v9.5以降[注1]に対応したオープンソースの拡張モジュールで、GPUを用いてJOINやGROUP BYなど集計・解析系のSQLワークロードを非同期・並列に実行します。GPUが計算処理の大部分を担うことでCPUの負荷を削減し、また、SQLクエリの内容を解析して自動的にGPU命令列を生成することで、利用者からは透過的にクエリ応答時間を短縮することが特徴です。

では、GPUを用いるというのはどういうことでしょうか？　Graphics Processing Unit（GPU）という名前が示すように、もともとこのデバイスは、三次元空間上のポリゴン頂点の計算など、**重い**画像データ処理をCPUの代わりに実行するために設計されたハードウェアです。

こういった処理は定型的な四則演算の組み合わせで表現される[注2]ため、ハードウェア実装に向いており、半導体集積度の向上に伴ってその並列度も飛躍的に向上してきました。

とくに2000年代以降は、座標計算専用の回路をハードウェアで実装するのではなく、CPUと同様にプログラムによって演算を制御するというプログラマブルシェーダの搭載が一般的になりました。これにより、画像処理にとどまらず汎用の数値計算処理にGPUを利用するGPGPU（General Purpose GPU）という新しい流れが生まれ、HPC（High Performance Computing）領域を中心に、開発環境やライブラリなど10年以上の技術的なノウハウが積み重ねられてきました。

ただ画像データと言っても、その中身は基本的に固定長の数値データの配列であり、突き詰

注1）現在、PostgreSQL v9.6への対応作業が進行中です。

注2）高校数学で出てくる行列・ベクトルの積と座標変換をものすごい勢いで繰り返しているに過ぎません。

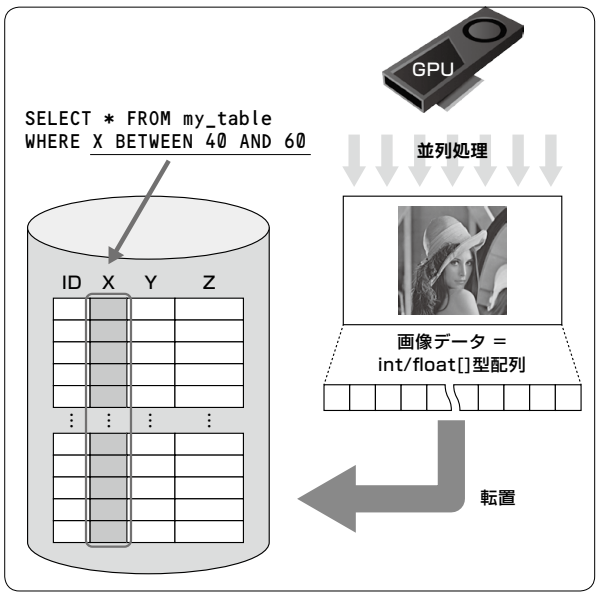
めれば、配列の各要素に同一の処理を繰り返すという処理は画像処理に限った話ではありません(**図1**)。

たとえば数MBの画像データの各ピクセルに何がしかの処理を施す、といった場合、これは画像データの実体である配列変数の各要素に"何がしか"を行うということを意味します。GPUの場合、後述するように非常に多数の演算コアが搭載されていますので、1回のサイクルで数千個の配列要素を処理することもできます。

数年前に筆者がこの話を聞いたときに考えたのは、条件句(WHERE句)付きテーブルスキャンとの類似性でした。数百万行を越えるレコードを持つテーブルは珍しくありませんが、条件句を伴う全件スキャンを実行する場合、これはレコード中の特定の列に対して同一の演算を繰り返すことと同等です。データ構造を転置して縦横を入れ替えたと考えれば、画像データを処理するのと本質的に変わりないことになります。

当時、すでに1,000コア以上の並列処理能力を持ったGPUがコンシューマ向けに数万円で販売されており、「もしかすると、これを使えばPostgreSQLの性能を安価に引き上げることができるかもしれない」という着想がPG-Strom開発のきっかけとなりました。

▼図1　基本的なアイデア



プロセッサの特徴を押さえる

PG-Stromの説明に入る前に、もう少しGPUの特徴について押さえておきます。

表1は現行世代のCPUとGPUについてスペックを比較したものですが、いくつか顕著な違いがあります。

CPUのコア数は比較的少ないものの各コアの動作クロックは高く、また豊富なキャッシュメモリを搭載しています。これは予測困難なメモリアクセスパターンや複雑な条件分岐を伴う処理に有効に作用します。

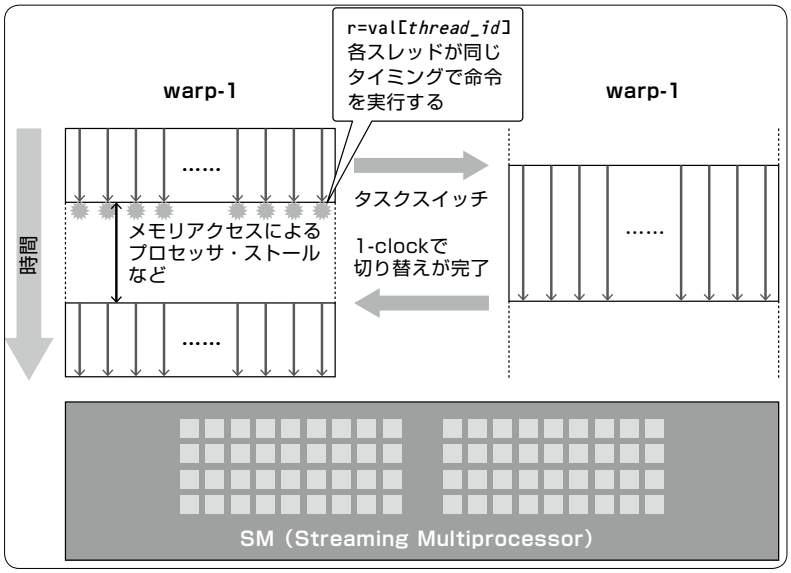
一方、GPUのフラッグシップモデルであるNVIDIA Tesla P100は、ワンチップに3584演算コアを搭載し、HBM2(High Bandwidth Memory)世代のメモリの採用によりCPUの10倍近い帯域を有しています。これは、処理対象のデータを高スループットで演算コアに送り込み、同一の演算処理を多数のデータに対して並列に実行することを意図した設計で、繰り返し計算を伴う処理に有効に作用します。

いわば、CPUは加減速性能に優れた乗用車のような、GPUは高速・大量輸送を行う鉄道のようなデバイスと考えることができますが、どちらも得意不得意があり、一概にコア数が多

▼表1　CPUとGPUの比較

	CPU	GPU
モデル	Intel Xeon E5-2699 v4	NVIDIA Tesla P100
アーキテクチャ	Broadwell	Pascal
発表	Q1-2016	Q2-2016
トランジスタ数	7.2billion	15billion
コア数/クロック	22 / 2.20GHz	**3584** / 1.328GHz
レジスタ容量	18KB	**14MB**
キャッシュサイズ	**55MB**	4096KB
メモリ容量	Max 1.5TB DDR4	16GB HBM2
メモリ帯域	76.8GB/s	732GB/s
TDP	145W	250W

▼図2　GPU上のタスクスケジューリング



いから優れているというわけではありません。

また、GPUではスレッドの振る舞いも独特です。数千個の演算コアが実装されているといっても、すべてのスレッドが個別のタイミングで動作するわけではありません。NVIDIA社のGPUの場合、32スレッドを束ねたwarpと呼ばれる単位でタスクスケジューリングが行われ、warp内の各スレッドはプログラムポインタを共有します。

これはどういうことかというと、隣接するスレッドが常に同じタイミングで同じ命令を実行しており、たとえば`r = val[threadIdx.x];`のように配列からレジスタにデータをロードする処理[注3]を行った場合、32個のスレッドが一斉にメモリアクセスを行います(**図2**)。

レジスタ参照に比べメモリアクセスは**重い**操作になりますが、こういった場合、GPUではほかのスケジューリングされていないwarpを実行ユニット上にスケジュールすることで、メモリアクセスのレイテンシを隠ぺいします[注4]。

CPUの場合、コンテキストスイッチはメモリアクセス以上に**重い**操作になるため、通常、コア数以上のスレッドを起動しても性能上のメリットは期待できません。しかしGPUの場合、プロセッサをスケジュール可能なスレッドで飽和させておくことで実効性能を高い水準に保つことができます。そのため、たとえば100万個の配列要素を処理するために100万個のスレッドを起動し、これらが最終的に物理的な演算コアにスケジュールされた結果、100万回のループ処理を行ったのと同一の結果が得られる……という考え方でプログラムを実装することが一般的です。

PG-Strom内部の処理も、このようなプロセッサの特徴をふまえて実装されています。CPU側がストレージ／バッファからデータブロックを読み出すと、数十万行程度の塊を単位としてどんどんGPUの処理キューに放り込んでいきます。そして、このブロックの処理完了を待つことなく、以降のデータブロックの読み出しを進めます。

データブロックには平均して数十万行のレコードが含まれているため、レコードに対して1個のスレッドを起動すれば、高々数千個のGPU演算コアを飽和させるには十分です。また、データブロックがGPU処理キューに詰まっていれば、GPUカーネルの実行完了後、直ちに次のGPUカーネルを起動することが可能となります(**図3**)。

注3) threadIdxは実行中のスレッドIDを意味するread-onlyレジスタです。たとえば512個のスレッドを実行している場合、val[0]～val[511]の内容を各スレッドが読み出すということになります。

注4) CPUにおいては、OSのスレッドよりもむしろ、プロセッサのHyper Threadingが似た概念かもしれません。

PG-Stromはどのように動作するか

PostgreSQLがクエリを処理する流れは、ざっくりと①～④のようになります(**図4**)。

①アプリケーションから受信したSQL文(テキスト)を、パーサ(Parser)が構文木(Parse Tree)と呼ばれる内部処理に適した構造に置き換える

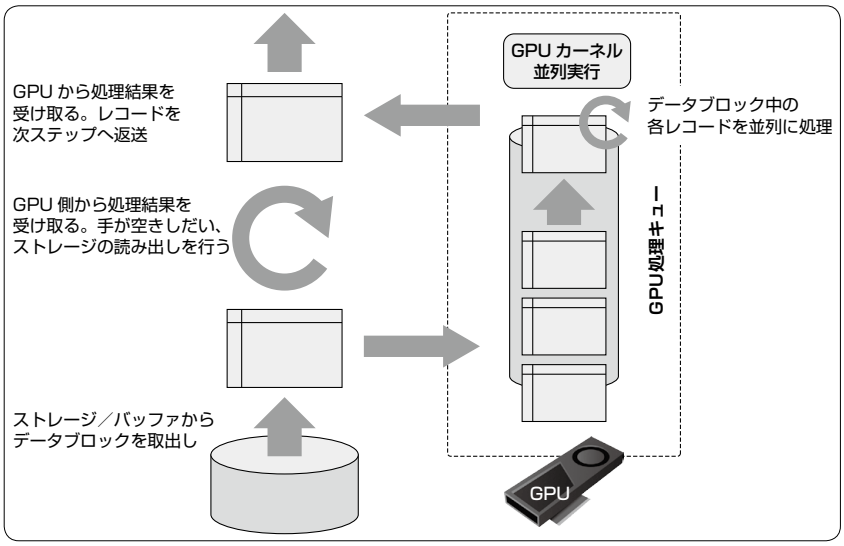
②構文木を解析し、オプティマイザ(Optimizer)が実行計画を作成する。アルゴリズムの特性や統計情報などからコスト値を算出し、最も推定コストが小さな実行計画を選択する

③クエリ実行計画に基づき、テーブルのスキャン、結合、集約などを実行。この過程でストレージマネージャを介してバッファやストレージからデータを読み出す

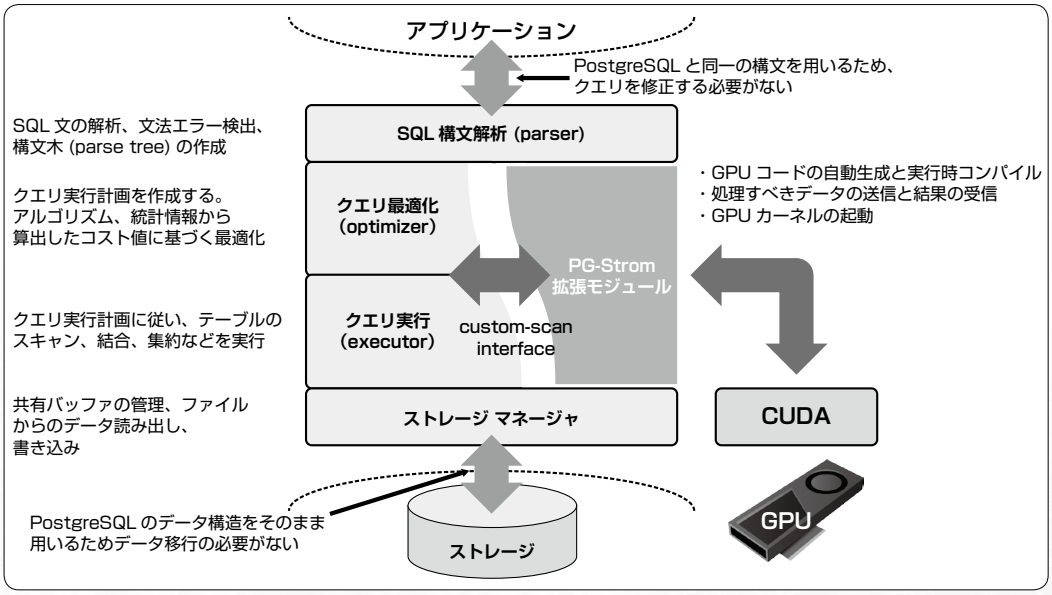
④実行結果をアプリケーションへと返却する

PG-StromはPostgreSQL向けの拡張モジュールですが、v9.5からサポートされたCustom-Scan Interfaceと呼ばれる一群のAPIを用いてオプティマイザ／エグゼキュータに介

▼図3　GPUを遊ばせないためのアーキテクチャ



▼図4　PG-Stromのアーキテクチャ



入します。

これは標準サポートの実行計画だけでなく、拡張モジュールが独自実装の実行計画を埋め込むための機能拡張です。たとえばテーブルAとテーブルBを結合する場合、PostgreSQLは標準でHash-Join、Merge-Join、Nest-Loopという3つのアルゴリズムを実装しており、結合条件やインデックスの有無、統計情報などからそれぞれの実行コストを見積もります。通常、この中から最もコストの小さな実行計画が選択されますが、PG-Stromはこれら3つに加えて「GPUを用いたJOIN」を実行計画の候補として追加します。最終的に最もコストの小さな実行計画が選ばれるのですが、たとえば結合対象のテーブルが非常に小さくGPUによる効果が見込めない場合には、PG-Stromの追加した実行計画は破棄されます。

図5の出力結果は、PG-Stromを有効にした状態でEXPLAIN文を実行したものです。

テーブル同士のJOINや、GROUP BYの前処理でGPUを用いた独自実装の実行計画が選択されているのがわかります。

実行計画が選択されると、エグゼキュータはAPIを介してPG-Stromを呼び出し、処理結果を返却するよう要求します。

APIの仕様はシンプルで、たとえば実行計画が表スキャンである場合は『どんな方法でも良いので表をスキャンし、次に条件句に合致するレコードを取り出す』です。したがって、標準のSeqScan(全件スキャン)であれば共有バッファから次のレコードを取り出し、条件句に合致するレコードを見つけた時点で1行を返却します。中には、Sortのように下位の実行計画の結果をすべて取り出したあとでないと最初の1行を返却できないような実行計画もあります。

PG-Stromの提供する実行計画[注5]では、次の流れで処理をします。

①WHERE句やJOIN結合条件、GROUP BY句などをもとにGPU側で実行するコードを生成し、実行時コンパイルによりGPU用命令バイナリを生成する
②スキャン対象の表や下位の実行ノードからデータを読み出し、数十万行程度の塊を単位としてGPU用命令バイナリとともにGPU処理キューに投入する
③ CUDAランタイムがこれをGPUへ転送し実行。一連の処理は非同期で行われる
④ GPUでの処理結果を受け取り、結果バッファから次の1行を取り出して上位の実行計画に返却

②～④のステップについては**図3**で説明したとおりです。

GPU用コードの自動生成について、もう少し掘り下げて説明することにしましょう。

注5) GpuScan、GpuJoin、GpuPreAggの3種類あります。

▼図5　GPU版実装を含むクエリ実行計画

```
postgres=# EXPLAIN SELECT cat,count(*),avg(ax) FROM t0 NATURAL JOIN t1 GROUP BY cat;
                                        QUERY PLAN
-----------------------------------------------------------------------------------------
 HashAggregate  (cost=3578311.16..3578311.48 rows=26 width=12)
   Group Key: t0.cat
   ->  Custom Scan (GpuPreAgg)  (cost=14139.24..2873626.84 rows=234 width=48)
         Reduction: Local + Global
         GPU Projection: cat, ax
         ->  Custom Scan (GpuJoin) on t0  (cost=10139.24..2838812.04 rows=98599882 width=12)
               GPU Projection: t0.cat, t1.ax
               Depth 1: GpuHashJoin, HashKeys: (t0.aid)
                        JoinQuals: (t0.aid = t1.aid)
                        Nrows (in/out: 98.60%), KDS-Hash (size: 13.47MB, nbatches: 1)
               ->  Seq Scan on t1  (cost=0.00..1935.00 rows=100000 width=12)
(11 rows)
```

GPUコードの自動生成

PG-StromがGPUコードを自動生成すると言っても、JOINやGROUP BYの処理ロジックをすべて動的に作り出しているわけではありません。

たとえば、最も単純なケースであるWHERE句を伴うスキャンの場合、WHERE句で評価する条件式こそクエリごとに異なりますが、GPUに転送されたデータブロックを読み出す、条件式が真であるレコードを結果バッファに転送する、といった処理はどんなスキャンであっても共通です。

このような共通部分はGpuScanやGpuJoinといった各ロジックごとに静的に持っており、PG-Stromはクエリごとに異なる部分だけを動的に生成し、実行時コンパイルでGPU用命令バイナリを生成します。

例で見てみましょう。**図6**は単純なWHERE句を伴う全件スキャンです。

GPU Filter: にGPU側で実行する条件式が示されています。多少表現は異なりますが、これはx BETWEEN y-20 AND y+20と等価な表現です。

▼図6　WHERE句を伴う全件スキャン

```
postgres=# SET pg_strom.debug_kernel_source = on;
SET
postgres=# EXPLAIN VERBOSE
           SELECT * FROM tbl WHERE x BETWEEN y - 20 AND y + 20;
                                QUERY PLAN
-------------------------------------------------------------------------------
 Custom Scan (GpuScan) on public.tbl  (cost=7669.74..54771.36 rows=444446 width=65)
   Output: id, cat, x, y, z, memo
   GPU Projection: tbl.id, tbl.cat, tbl.x, tbl.y, tbl.z, tbl.memo
   GPU Filter: ((tbl.x >= (tbl.y - '20'::double precision)) AND
                (tbl.x <= (tbl.y + '20'::double precision)))
   Extra: bulk-exec-support, slot-format
   Kernel Source: /opt/pgsql/base/pgsql_tmp/pgsql_tmp_strom_6592.2.gpu
(6 rows)
```

▼図7　WHERE句を伴う全件スキャン（続き）

```
STATIC_FUNCTION(cl_bool)
gpuscan_quals_eval(kern_context *kcxt,
                   kern_data_store *kds,
                   ItemPointerData *t_self,
                   HeapTupleHeaderData *htup)
{
  pg_float8_t KPARAM_0 = pg_float8_param(kcxt,0);
  pg_float8_t KPARAM_1 = pg_float8_param(kcxt,1);
  pg_float8_t KVAR_3;
  pg_float8_t KVAR_4;
  char *addr;

  assert(htup != NULL);
  EXTRACT_HEAP_TUPLE_BEGIN(addr, kds, htup);
  EXTRACT_HEAP_TUPLE_NEXT(addr);
  EXTRACT_HEAP_TUPLE_NEXT(addr);
  KVAR_3 = pg_float8_datum_ref(kcxt, addr, false);
  EXTRACT_HEAP_TUPLE_NEXT(addr);
  KVAR_4 = pg_float8_datum_ref(kcxt, addr, false);
  EXTRACT_HEAP_TUPLE_END();

  return EVAL((pgfn_float8ge(kcxt, KVAR_3, pgfn_float8mi(kcxt, KVAR_4, KPARAM_0)) &&
               pgfn_float8le(kcxt, KVAR_3, pgfn_float8pl(kcxt, KVAR_4, KPARAM_1))));
}
```

pg_strom.debug_kernel_source設定により、自動生成したGPUプログラムは一時ファイルに書き出されています。

自動生成されたgpuscan_quals_eval関数がWHERE句に相当し(**図7**)、各レコードごとにGPUの演算コア1個を割り当ててこの関数を実行します。

このクエリでは列xと列yを参照しますが、関数の前半ではレコードから該当するフィールドをKVAR_3変数とKVAR_4変数にロードしています。また、KPARAM_0とKPARAM_1は定数バッファを参照して値20.0をロードしています。定数を直接プログラム中に埋め込まないのは、たとえば次にx BETWEEN y-10 AND y+10といった条件で検索を行う際に、プログラムの再コンパイルを省略するためです。

続いて、これら変数にロードした値を計算式に与えています。GPU側の関数pgfn_float8geやpgfn_float8miというのは、それぞれ浮動小数点型の大小比較や減算を行う関数ですが、ここでは'<'や'-'などCUDA Cの演算子を使用できません。

GPU側での式評価は厳密にSQL側の挙動に従う必要があるため、たとえば加減算の際にも入力値のNULL検査を行ったり、演算結果のオーバーフローチェックを行う必要があります。そのため、PG-Stromでは単純な四則演算子であってもこれらのチェックを含むGPU側の関数を定義して使用しています。

当然、こういったチェックに伴う性能トレードオフは発生しますが、PG-Stromはこの辺のオーバーヘッドを省略して、GPUのフルスピードでロジックを実行するしくみも持っています。こちらは来月号で紹介します。

PG-Stromを動かしてみる

数年前にPG-Stromの開発を始めたころは、GPUを搭載したLinux環境といえばオンプレミスが当然でしたが、機械学習技術への注目が

▼図8　PG-Strom構築済みAMIとGPUインスタンスの選択

追い風となって、Amazon AWSやMicrosoft Azureといったクラウド環境でもGPUの利用が可能となってきています。

本稿では、AWSのGPUインスタンスと構築済みAMIを用いて手軽に環境構築を行う方法を紹介します。

AWSはGPU対応インスタンスの提供という点では最古参で、搭載するGPUの種類により3世代のインスタンスタイプが存在しているのですが、PG-Stromに対応しているのは最新のp2.*タイプだけです。

これは、NVIDIA社のGPUにはデバイスの対応する機能や命令セットを示すCompute Capability(CC)と呼ばれるバージョン番号が付与されており、PG-StromはCC3.5以上のGPUを要求するのですが、現時点では最新のp2.*タイプ[注6]だけがこの要件を満たすためです。

AWS EC2サービスでインスタンスを生成する際には、まずAMI(Amazon Machine Image)と呼ばれる構成テンプレートを選択し、それをもとにインスタンスを作成・実行します(**図8**)。AMIイメージは構築済みインスタンスから簡単に作成することが可能で、また面白いことに「コミュニティAMI」として公開することもできます。筆者も、p2.xlargeインスタンス向けにPG-StromやPostgreSQL、CUDAなどをインストール・初期設定を行ったAMIイメージを

注6) Kepler世代のTesla K80を搭載しており、Compute Capabilityは3.7です。

作成、公開しています。

コミュニティAMIの「PGStrom」で検索にヒットするはずですので、これを選択し、次画面で「GPUコンピューティング」分類に含まれるp2.xlargeタイプ[注7]を指定してインスタンスを作成すれば、PG-Strom動作環境の構築は完了です(**図9**)。

インスタンスが起動すると、ユーザ名「ec2-user」を用いてsshでログインできます。すでにPG-StromをインストールしたPostgreSQLサーバが起動しており、コンソールからpsqlコマンドを使用して接続できるはずです(**図10**)。

このデータベースにはすでにテスト用のテーブルが構築されており、PG-Stromがどのように振る舞うかを確認できます。

たとえば、**図11**のクエリはGPU側でJOINとGROUP BYを実行して結果を返します。EXPLAIN文の出力結果より、PG-Stromが独自に実装している実行計画であるGpuPreAggやGpuJoinを含んでいることがわかります。

これを実行すると、当然ですが、通常のSQL文と同様にテーブルを結合し、集約演算を実行した結果が返ってきます。

なお、起動直後はテスト用テーブルがキャッシュされておらず(数GB程度なのですが)、スキャンに非常に時間がかかります。性能測定におけるI/Oの影響を避けるため、あらかじめpg_prewarm()を使用して対象データをロードしておくことをお勧めします(**図12**)。

PG-Stromを無効化するには、パラメータpg_strom.enabledにoffを設定します。この状態で同じSQLを実行すると、素のPostgreSQLで実行させるのと同じ状態を再現できるわけです。

EXPLAIN構文を使って実行計画を確認すると、GpuPreAggやGpuJoinが消え、標準のHashJoinが選択されていることがわかります(**図13**)。

実行結果はPG-Stromを有効化している場合と同様ですが、実行に51秒を要しており、SQL処理をGPUにオフロードすることによって高速化できていることがわかります(**図14**)。

なお、勘の鋭い方は気づかれたかもしれませんが、0～100範囲の乱数値の平均を計算するために使用したavg()関数の結果が、PG-Stromの有無によって少し異なっています。

これは実数値(浮動小数点形式)の性質と計算順序に伴う計算誤差で、たとえばグループaaa

注7) 2017年1月現在、p2.*タイプはユーザの起動可能なインスタンス数が0であることに注意してください。EC2ダッシュボードから「制限緩和のリクエスト」を行う必要があります。

▼図9　GPUインスタンス(p2.xlarge)の選択

▼図10　SSHによるログイン

```
[ec2-user@ip-172-31-24-196 ~]$ psql postgres
psql (9.5.5)
Type "help" for help.

postgres=#
```

▼図11　GPU側でのJOINとGROUP BYの実行結果

```
postgres=# SELECT cat,count(*),avg(ax)
FROM t0 NATURAL JOIN t1 GROUP BY cat;
 cat |  count  |       avg
-----+---------+------------------
 nnn | 3845736 | 49.9122204644341
 ccc | 3842975 | 49.9253161146813
 ddd | 3841209 | 49.9346989307005
 aaa | 3848221 | 49.9265219996308
  :       :                :
 sss | 3846990 | 49.9213589517191
(26 rows)

Time: 11326.834 ms
```

▼図12　pg_prewarm()による対象データのロード

```
postgres=# SELECT pg_prewarm('t0'::regclass);
 pg_prewarm
------------
     833334
(1 row)
```

▼図13　PG-Stromの無効化

```
postgres=# SET pg_strom.enabled = off;
SET
postgres=# EXPLAIN SELECT cat,count(*),avg(ax) FROM t0 NATURAL JOIN t1 GROUP BY cat;
                                QUERY PLAN
---------------------------------------------------------------------------
 HashAggregate  (cost=3937016.94..3937017.26 rows=26 width=12)
   Group Key: t0.cat
   ->  Hash Join  (cost=3185.00..3197517.82 rows=98599882 width=12)
         Hash Cond: (t0.aid = t1.aid)
         ->  Seq Scan on t0  (cost=0.00..1833334.00 rows=100000000 width=8)
         ->  Hash  (cost=1935.00..1935.00 rows=100000 width=12)
               ->  Seq Scan on t1  (cost=0.00..1935.00 rows=100000 width=12)
(7 rows)
```

▼図14　PG-Stromの無効化による実行結果

```
postgres=# SELECT cat,count(*),avg(ax)
FROM t0 NATURAL JOIN t1 GROUP BY cat;
 cat |  count  |       avg
-----+---------+-----------------
 nnn | 3845736 | 49.9122204644368
 ccc | 3842975 |  49.925316114681
 ddd | 3841209 |  49.934698930695
 aaa | 3848221 | 49.9265219996321
  :       :               :
 sss | 3846990 | 49.9213589517182
(26 rows)

Time: 51031.084 ms
```

のavg()関数の結果は、PG-Strom ありの場合、49.9265219996308ですが、PG-Strom なしの場合49.9265219996321と、最後2桁の値が異なっています。

ご存じのとおり、平均値は、母集団の総和を要素数で割って求めます。総和を計算するには入力値を順に足し込んでやる必要があるのですが、実数(float8)の平均値を計算するavg()関数の内部カウンタもまた実数(float8)が使われています。そのため、一定範囲の実数を順番に加算していくと、件数の増加に従って「非常に大きな値」と「小さな値」との足し算が発生しますが、これは浮動小数点演算で桁落ちが発生する典型的なケースです。

PG-StromがGPUで平均値を計算するとき、まず近傍の1,024スレッドで小規模な集約演算を行い、次に全体で総和・要素数を計算します。そのため、件数の増加に従って内部カウンタの値が非常に大きくなる点は共通なのですが、GPU側では数十個の要素をあらかじめ足し合わせた値を加算していくため、結果として桁落ちの度合いが小さくなるのです。こういったところに、GPUでの実装が顔を覗かせているわけです。

後編にむけて

本稿では、PostgreSQLのオプティマイザ／エグゼキュータと連携して動作するPG-Stromのアーキテクチャ、透過的SQLアクセラレーションの中核であるGPUコードの自動生成、そして関連技術であるGPUの特徴と、AWSでの環境構築の方法について紹介しました。

次号の後編では、GPUでのJOIN処理をテーマとしてPG-Stromの動作を掘り下げて解説し、また、今後の展望についても紹介する予定です。SD

Column「PG-Stromの名前の由来について」

PG-Stromの"PG"はもちろんPostgreSQLが由来なのですが、"Strom"とは何ぞ？——とよく聞かれます。最初はPG-GPGPUというモジュール名を考えていたのですが、PとGばかりでダサいので却下。代わりに、GPUのプロセッサコアのことをStreaming Multiprocessorと呼んでいることに引っかけ、また筆者が当時ドイツに住んでいたことから、"Stream"を意味するドイツ語の"Strom"をモジュール名に選びました。ほかにも非同期にデータブロックを次々とGPUへ放り込んでいく様子があたかも川の流れのようである、というふうにも重ねていたりします。

書いて覚えるSwift入門

第23回 型は苦しい（かもしれない）が役に立つ

Author 小飼弾（こがい だん）　twitter @dankogai

なぜSwiftは静的な型を採用したのか?

前回紹介したとおり、Swiftの公式サイトのAbout Swift[注1]には、

・安全（Safe）
・高速（Fast）
・豊かな表現力（Expressive）

という3つの特長が挙げられています。ところでSwiftは静的型をしています。つまりSwiftの生みの親たちは静的型がこの3つの特長に有利だと判断したということですが、なぜ静的型を採用すると安全で高速で表現力が豊かになるのかきちんと型られた、もとい語られたことは意外に少ないように筆者には思われます。本記事では他の言語の例も交えつつ、Swiftの型に対する姿勢を見ていくことにしましょう。

varでも変えられないもの

では早速実例を見てみましょう。PlaygroundかREPLで、次のように入力してみてください。

```
var number = 20
number += 1
number *= 2
number += 0.195
```

最後のところでerror: binary operator '+=' cannot be applied to operands of type 'Int' and 'Double'というエラーが出ているはずです。整数に浮動小数点数は足してはダメだということです。

今度はRubyで同じことをしてみましょう。以下はirbで実行してみた例です。

```
irb(main):001:0> number = 20
=> 20
irb(main):002:0> number += 1
=> 21
irb(main):003:0> number *= 2
=> 42
irb(main):004:0> number += 0.195
=> 42.195
```

一見Rubyの方が便利に思えます。が、さらに続けてこう打ってみましょう。

```
irb(main):005:0> number = "#{number}km"
=> "42.195km"
irb(main):006:0> number *= 2
=> "42.195km42.195km"
```

いつの魔に数値だったnumberは文字列になってしまっています。

今度はSwiftに戻って次のようにしてみましょう。

```
var number:Double = 20
number += 1
number *= 2
number += 0.195
```

今度は42.195になったはずです。さらに続けて

```
number = "\(number)km"
```

としてみると、error: cannot assign value of type 'String' to type 'Double'で止まります。

注1）https://swift.org/about/

この観察から、Swiftの型の特徴が見えてきます。

・型は変数宣言の時点で決まる
・型を明示しなくともよい。しない場合は推論される
・変数に異なる型の値を代入することはできない

1つ注意が必要なのは、Rubyに型がないわけではないということ。Rubyの値は必ずクラスという名の型を持っています。変数にどんな型のどんな値でも代入できるだけで。一度宣言した変数に異なる型を代入できない言語を静的型言語(statically typed languages)、どんな型でも代入できる言語を動的型言語(dynamically typed languages)と呼びます。C、C++、C#、Objective-C、Javaなどは前者、JavaScript、PHP、Perl、Python、Rubyなどは後者に属します。いわゆるスクリプト言語はほとんど後者に属するのは興味深いところです。ではSwiftはスクリプト言語ではないかというと、以下が動く以上スクリプト言語であるという見方もできなくはありません。

```
$ cat > ./helloswift
#!/usr/bin/env swift
print("Hello, World!")
$ chmod +x ./helloswift
$ ./helloswift
Hello, World!
```

にもかかわらずSwiftが静的型を採用しているのは、型推論(type inference)を採用できたことが大きいでしょう。静的型言語も動的型言語もずいぶん昔からありましたが、型推論の導入は前世紀末で、実用的になったのは今世紀に入ってからといっても過言ではありません。なぜ実用的になったのが比較的最近かといえば、コンピュータの性能が上がったから。PlaygroundやREPLはユーザから見たらインタラクティブなので一見インタープリタに見えますが、実は都度コンパイルしているのです。

「やりたいことをやる」にはいちいち型を宣言するのはめんどくさい。しかし「やってほしくないことをやらない」ためには型でやれることを制約したほうがいい。せっかくコンピュータの性能が上がったのであれば、両者のいいとこ取りができた方がよいのは「宣言する」までもないことでしょう。

型ははめてなんぼ

それでは型があった方が本当に安全なのか。やはり実例で見てみましょう。ここでは標準入力に行番号をつけて出力するだけの簡単なお仕事をしてみます。コマンドであればcat -n、Perlのワンライナーであればperl -e 'print"$.: $_"while<>'ですが、Swiftではどうでしょう。

```
var count = 0
while let line = readLine() {
    count += 1
    print("\(count): \(line)")
}
```

簡単ですね。ではCは?

```
#include <stdio.h>
int main() {
    char line[64];
    int  count = 0;
    while (gets(line)) {
        printf("%d: %s\n", ++count, line);
    }
}
```

そんなに難しそうには見えませんね。#include <stdio.h>とかint main(){}とか余計なおまじないが入っていたり、char line[64]とかint countとか変数の型を明示している点を除けばSwiftとさほど違いはないように見えます。

本当に(?)実際にコンパイルして確かめて見ましょう。

```
$ cc -Wall cat-n.c
$ ruby -e 'puts "1"*42' | ./a.out
warning: this program uses gets(), which is
unsafe.
1: 111111111111111111111111111111111111111111
```

警告は出ていますが、一応動いているように見えます。しかし……、

```
$ ruby -e 'puts "1"*4096' | ./a.out
warning: this program uses gets(), which is
unsafe.
Segmentation fault
```

落ちちゃいました。何が問題だったのでしょう？　ソースを見てみると、lineはchar64個からなる配列として定義されています。なのに4,096文字もある行を喰わせたらバッファーオバーフローするのは当然ですね。文字列ではないのです。文字の配列です。じゃあ文字列を指定すればいい？　どこにあるんですかCの文字列なんて！

というわけでgets()ではなくfgets()を使って書き直して見ましょう。

```
#include <stdio.h>
#define CHARSPERLINE 64
int main() {
    char line[CHARSPERLINE];
    int  count = 0;
    while (fgets(line, CHARSPERLINE, stdin)) {
        printf("%d: %s\n", ++count, line);
    }
}
```

```
$ ruby -e 'puts "1"*4096' | ./a.out
1: 1111111111111111111111111111111111111111111
1111111111111111111
……（中略）……
65: 1111111111111111111111111111111111111111111
1111111111111111111
66: 1
```

確かにSegmentation faultは出なくなりましたが、1行のはずが66行に分かれてしまいました。明らかにCHARSPERLINEが64と小さいのが原因ですが、ここの適切な値はなんでしょうか？

4096？　65536？　そもそもそれってプログラマが頭を悩ませるべきことでしょうか？

気を取り直して、Swift版を試してみましょうか。

```
$ swiftc cat-n.swift
$ ruby -e 'puts "1"*4096' | ./cat-n
1: 1111…
……（中略）……
111111111111111111
```

今度はきちんと1行におさまりました。

Cの問題は一体なんだったのでしょう？　「文字列」を扱うプログラムなのに、文字列という型が不在で、「文字の配列」という不適切な型を使っていたところなのです。型というのは、それが扱いうる値をきちんと収納できてはじめて意味があるのに実はそうなっていない。それも文字列という、今や数値以上によく扱われる値に対してすら。

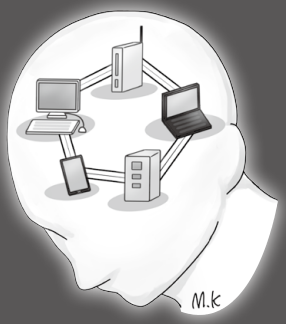
今となってはCという言語は静的型と動的型の悪いところ取り言語にどうしても見えてしまいます。Cで「やりたいことをやる」のは多少面倒だけど難しくはない。しかし「やれてはならないことをやれなく」するのは本当に難しい。難しいからこそHeartbleed[注2]やShellshock[注3]のような脆弱性があとをたたない。

そもそも型はなんのためにあるのか。安全で高速なプログラムを実現するためではないのですか？　余計なコードを実行せず、余計なデーターを浪費せず、余計な脆弱性を持ち込ませない。それができない型は面倒なおまじない以下ではないのですか？

もちろん型は安全性や高速性の特効薬ではありません。Swift自体、CやObjective-Cと簡単に相互運用できるようになっていますがこの点から考えれば諸刃の剣。動的にメモリーを確保してくれる静的なStringとて、過大なデータを食わせれば簡単にパンクしてしまいます。とはいえ、Cの置き換えを標榜する言語はSwiftに限らずRustもGoも「まっとうな」文字列型を持っているのは地獄に仏に筆者には見えます。

文字列という基本的な型一つ見ても、適切に設計された型の有用性は実感することができます。それではもっと複雑な型は？　たとえばJSONは？　次回はそれを見てみることにしましょう。SD

注2）https://en.wikipedia.org/wiki/Heartbleed
注3）https://en.wikipedia.org/wiki/Shellshock_(software_bug)



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10回 virshによる仮想マシンの作成とゲストOSのインストール

連載後半「仮想ネットワーク環境で使ってみよう〜現実的な使い方」の4回目です。

今回は、前回(2017年2月号 第9回)に続き、virshによる仮想マシンの作成を解説した後、前回のvirt-installコマンドや今回のvirshコマンドで作成・起動した仮想マシンの上に、ゲストOSをインストールします。

▼図1 virshによる仮想マシンの作成

```
[root@vhost1 newx1]# virsh dumpxml svm1>svm2.xml  dumpxmlによるxmlファイル作成
[root@vhost1 newx1]# uuidgen -t  UUID生成①
91673026-ba63-11e6-b1ce-00174265c4a9
[root@vhost1 newx1]# cat >>svm2.xml
91673026-ba63-11e6-b1ce-00174265c4a9
FreeBSD-10.3-RELEASE-amd64-dvd1.iso
[root@vhost1 newx1]# fdisk /dev/sda  fdiskによる/dev/sda6作成
警告: DOS互換モードは廃止予定です。このモード (コマンド 'c') を止めることを
      強く推奨します。 and change display units to
         sectors (command 'u').
コマンド (m でヘルプ): n
最初 シリンダ (7899-12161, default 7899):
Using default value 7899
Last シリンダ, +シリンダ数 or +size{K,M,G} (7899-12161, default 12161): +15G
コマンド (m でヘルプ): p
 ...略...
デバイス ブート      始点        終点     ブロック   Id  システム
/dev/sda1   *           1          66      524288   83  Linux
パーティション 1 は、シリンダ境界で終わっていません。
/dev/sda2              66        5288    41943040   83  Linux
/dev/sda3            5288        5940     5242880   82  Linux スワップ / Solaris
/dev/sda4            5940       12161    49972000+   5  拡張領域
/dev/sda5            5940        7898    15729421+  83  Linux
/dev/sda6            7899        9857    15735636   83  Linux
コマンド (m でヘルプ): w
パーティションテーブルは変更されました!
ioctl() を呼び出してパーティションテーブルを再読込みします。
警告: パーティションテーブルの再読込みがエラー 16 で失敗しました: デバイスもしくはリソースがビジー状態です。
カーネルはまだ古いテーブルを使っています。新しいテーブルは
次回リブート時か、partprobe(8)またはkpartx(8)を実行した後に
使えるようになるでしょう
ディスクを同期しています。
[root@vhost1 newx1]#
```

virshによる仮想マシンの作成

まずは前回に続き、virshによる仮想マシン作成の作成を解説します。

ドメインxmlファイルの作成

ドメインxmlファイルを一から作成するのは大変なので、既存の仮想マシンのドメインxmlファイルからコピーしたものを編集して作成します(**図1**)。

既存xmlファイルのコピー

ここではまず、前項のvirt-installで作成した既存稼働マシンのドメインxmlファイルをコピーします。

コピーは通常、xmlファイルを画面表示する「virsh dumpxml」コマンドの出力をファイルとしてxmlファイルに落とし込みますが、仮想マシン設定ファイル(/etc/libvirt/qemu内のドメインxmlファイル)自体をコピーしてくることも可能です。

編集

dumpxmlした後、そのドメイン用xmlファイルを新しい仮想マシン用のドメインxmlファイルに編集することが必要です。最低限、編集が必要な個所は**図2**のようなところです。

ドメイン名(仮想マシン名)、UUID、ハードディスク実デバイス名、ISO(CD/DVD)イメー

▼図2　dumpxmlからのxmlファイル編集

```
[root@vhost1 newx1]# vi svm2.xml  仮想マシン元ファイルの編集
[root@vhost1 newx1]# diff  /etc/libvirt/qemu/svm1.xml svm2.xml  元ファイルからの編集部分
 ...略...
<
< <domain type='kvm'>
<   <name>svm1</name>
<   <uuid>f496e029-d208-85e7-8a58-dd3614cdaeb2</uuid>
---
> <domain type='kvm' id='2'>
>   <name>svm2</name>
>   <uuid>91673026-ba63-11e6-b1ce-00174265c4a9</uuid>
33c26
<       <source dev='/dev/sda5'/>
---
>       <source dev='/dev/sda6'/>
34a28
>       <alias name='ide0-0-0'/>
39c33
<       <source file='/iso/windows-server-2008_r2_sp1_x64.iso'/>
---
>       <source file='/iso/FreeBSD-10.3-RELEASE-amd64-dvd1.iso'/>
 ...略...
75c83
<     <graphics type='vnc' port='5911' autoport='no' listen='0.0.0.0' keymap='ja'
passwd='Password1'>
---
>     <graphics type='vnc' port='5912' autoport='no' listen='0.0.0.0' keymap='ja'
passwd='Password2'>
79a88
>       <alias name='video0'/>
82a92
>       <alias name='balloon0'/>
86a97
>
[root@vhost1 newx1]#
```

ジ名、NICのMACアドレス、グラフィクスVNC情報(ポート、LISTEN IPアドレス、パスワード)などの個別の識別情報です。

ここでは、仮想マシン1と同様に完全仮想化で、仮想マシン名をsvm2、ハードディスクデバイスには/dev/sda6、VNCポートは仮想マシン1とは別の5912、ゲストOSとしてFreeBSD 10.3を設定しています。

なお、UUIDは、uuidgenコマンドで作成します(**図1の**①)。

また、MACアドレス先頭3オクテットのOUI(Organizationally Unique Identifier、管理組織識別子)は、qemu/kvmでは「52:54:00」です。VNCポート番号はKVMホストで使用されていないポートを指定します。

ドメインxmlファイルの定義登録と仮想マシンの作成

作成されたドメインxmlファイルを仮想マシン設定ファイルとするための定義登録処理は、「virsh define」(定義)コマンドで行い、仮想マシンの作成は「virsh start」(起動)コマンドで行います(**図3**)。

ドメインxmlファイルの設定が正しければ「virsh define」で/etc/libvirt/qemu内にドメインxmlファイルが作成されています。

そして、「virsh start」コマンドは正常に終了すると、ゲストOSインスールが自動的に開始します。

ゲストOSのインストール

仮想マシンを作成後、ゲストOSのインストールは自動的に開始されていて、CD/DVDからのインストール実行は利用者の操作待ちまで自動的に進みます。なお、virt-installコマンドやvirsh startコマンドの起動が正常ならば、プログラムは終了します。

ゲストOSのインストール操作は、連載前半で行っていたように、仮想マシンマネージャーの「開く」アイコンからグラフィカルコンソール(virt-viewer)のコンソールGUI画面で行うか、または、ホストシステムもしくは実LAN上の別システムからVNC(vncviewer)で行います。

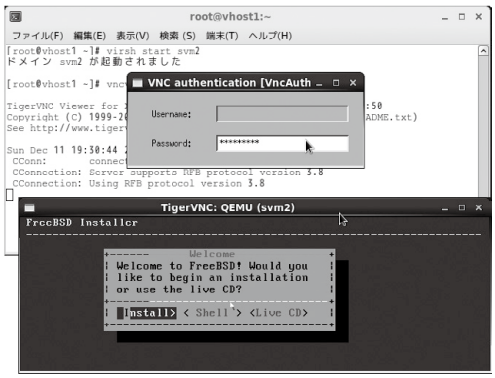
VNCによるゲストOSインストール操作

VNCからアクセスする際には、「KVMホストの」IPアドレスと、virt-installコマンドやドメインxmlファイルで指定したポート番号でアクセスし、virt-installコマンドやドメインxmlファイルで指定したパスワードでログインします(**図4**)。

▼図3　仮想マシンの定義登録と起動

```
[root@vhost1 newx1]# virsh define svm2.xml
ドメイン svm2 が svm2.xml から定義されました
[root@vhost1 newx1]# ls -al /etc/libvirt/qemu/
合計 24
drwx------. 4 root root 4096 12月  5 15:19 2016 .
drwx------. 5 root root 4096 12月  5 05:17 2016 ..
drwxr-xr-x  2 root root 4096 12月  4 12:15 2016 autostart
drwx------. 3 root root 4096 12月  3 12:37 2016 networks
-rw-------  1 root root 3146 12月  5 05:18 2016 svm1.xml
-rw-------  1 root root 3147 12月  5 15:19 2016 svm2.xml
[root@vhost1 newx1]#
[root@vhost1 newx1]# virsh start svm2
ドメイン svm2 が起動されました
[root@vhost1 newx1]#
```

▼図4　VNCによるゲストOSインストール



VNC操作はKVMホスト経由での理由

一般のVNCアクセスでは対象システム上のVNCサーバにアクセスすることになりますが、それは、「すでにインストール済み」で「システムが起動した後」であることです。つまり、今回のような「システムインストール」や「システム起動時」などでは「仮想マシンのシステムはインストールされていない」、あるいは「VNCサーバが起動していない」ので利用できません。

したがって、「KVMホストの」VNCサーバ経由で仮想マシン管理に直接入るわけです。

VNCアクセスのセキュリティ対策

実LAN上の別システムからVNCで行う場合には、発信IPアドレスや仮想マシンのVNCポート番号などによるセキュリティ制限やSSLトンネル使用によるVNCコミュニケーションのセキュリティ保護は必須の機能です(**リスト1**)。なお、詳細は以降の連載で解説します。

ゲストOSインストールでの注意ポイント

ゲストOSインストールの途中やその後などの操作で、以下のような注意すべきポイントがあります。

ゲストOSインストール時の再起動処理

CentOS 6のインストール直後にreboot(shutdown -r now)したとき、仮想マシンコンソールのVNC接続は切断なく、そのまま続行されています。

CD/DVD-ROM入れ替え

virshサブコマンド「qemu-monitor-command」を使用します[注1]。

ブートデバイスの設定

virt-installで作成される/etc/libvirt/仮想マシン名.xmlファイルにはcdromがbootデバイスとして登録されていません。

必要であれば、他の仮想マシンインストールのための設定や再インストール時のCD-ROM利用時にはCD-ROM設定を行います。そして、

注1) 参考:「Change_cdrom」
http://www.linux-kvm.org/page/Change_cdrom

▼リスト1　外部管理システムからのセキュアVNC接続設定の例

```
<iptables>

管理システム192.168.0.221からKVMホスト192.168.0.11のポート15912へは着信許可
-A INPUT -s  192.168.0.221 -i eth0 -p tcp -m tcp --dport 15912 -j ACCEPT

<stuunel.conf>

[vnc-vm2]
accept  = 15912
connect = localhost:5912
```

(再)インストール後、起動順序を「HD→CD-ROM」の順序に戻します。

設定変更は既存(登録済み)仮想マシンのXMLをvirsh editで次のように変更します。

```
<os>～</os>クローズ内
  <boot dev='cdrom'/> CD-ROMを最初
  <boot dev='hd'/>    HDを次に
```

ゲストOSインストール中に起こる問題と対策

virt-installコマンドまたはvirsh startコマンドから自動継続している、ゲストOSインストールで起こるトラブルシュートの例を以下に挙げます。

仮想コンソールのログインができない

「仮想マシンマネージャー」にある「仮想マシンのコンソール」で、仮想マシンのログイン画面でパスワードを入力しても何の反応もないときがあります。

対処策

このとき、「シャットダウン」がきかないので、仮想マシンを「強制的に電源OFF」で強制停止してから「仮想マシンの電源を入れる」で再起動すると、ログイン可能になります。

IRQエラー

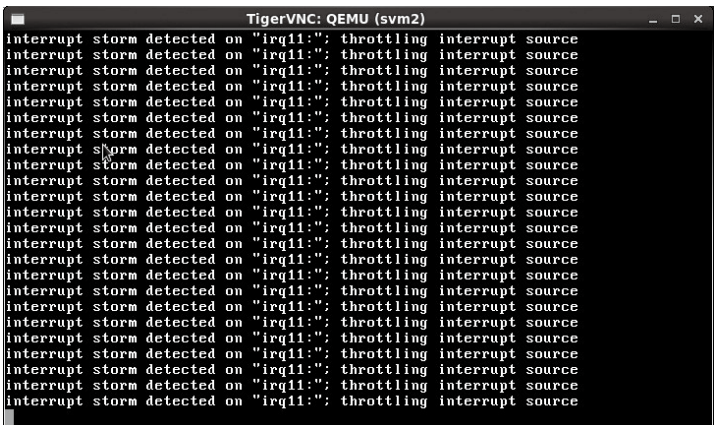
「virsh dumpxml」で作成したxmlファイルにより「virsh start ドメイン名」で起動したFreeBSDのインストールで**図5**のように、エラーメッセージが表示され続けることがあります。

対処策

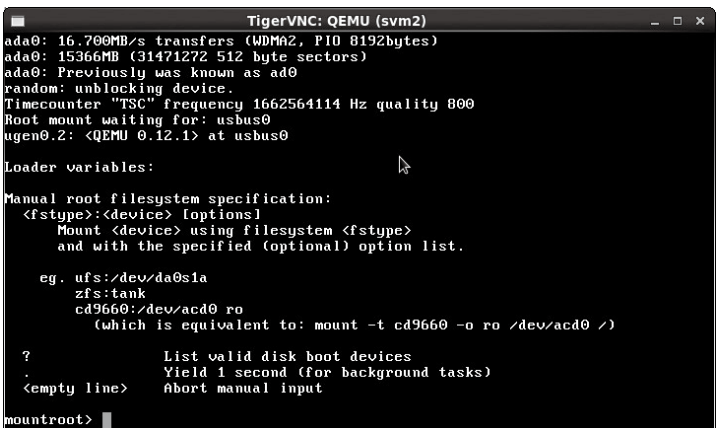
これは、「virsh dumpxml」で作成したxmlファイルに、USBタブレットが追加されることがあり、既定のPS2マウス(削除不可能)と競合して起きるIRQエラーです。

そこで、USBタブレットデバイスのエントリ(xmlファイル内の次の個所)を削除します。

▼図5　dumpxmlからの仮想マシンFreeBSDインストール時のIRQエラー



▼図6　仮想マシンFreeBSDインストール時の停止



▼リスト2　外部システムから仮想マシンへの接続許可

```
外部システム192.168.0.221から仮想マシン192.168.122.192のVNCポート5901への着信接続許可
-A INPUT -s  192.168.0.221 -d 192.168.122.192 -p tcp -m tcp --dport 5901 -j ACCEPT
```

```
<input type='tablet' bus='usb'>
  <alias name='input0'/>
</input>
```

ゲストOSインストール起動時停止

仮想マシン上のゲストOSでFreeBSDを使用してインストール途中で停止、再起動したとき、**図6**のように途中停止してしまうことがあります。

対処策

使用するディスク領域(ホストシステム上)を再割り当てするか、Linuxのmke2fsコマンドで次のように再作成します。

```
[root@vhost1 ~]# mke2fs /dev/sda6
```

UTC時刻になる

仮想マシンによっては、時刻表示がUTC時刻になってしまい日本時間とずれが生じてしまうことがあります。

対処策

virt-installインストールでのデフォルトのクロックは、Windows系ではlocaltimeであり、Linux/UNIX系ではUTCです。

Linux/UNIX系の仮想マシンでは、dumpxmlでコピーを作成しても同様にUTC時間です。そのため仮想マシン作成後、「virsh edit 仮想マシン名」サブコマンドで作成されたxmlファイルを開き、「clock offset」を次のようにUTCからlocaltimeに変更します。

```
修正前 <clock offset='utc'/>
修正後 <clock offset='localtime'>
```

VNCアクセスができない

「仮想マシンマネージャー(virt-viewer)」からの仮想マシン・ログインは可能ですが、ほかのマシンからのVNCアクセスができない、という現象です。

対処策

既存のiptablesフィルタではNAT接続で、かつ仮想マシン側のネットワークからの発信のみの許可設定です。

そこで、VNCアクセスするシステムから(使用する仮想マシンのIPアドレス宛)の着信を許可するフィルタを設定する必要があります。iptablesコマンドでINPUT/-j ACCEPTが必要です(**リスト2**)。

次回予告

次回は、今までに作成した仮想マシン、仮想ネットワークを使った「仮想マシンネットワークの利用」を、仮想マシン、仮想ネットワーク、ホストシステム、実LAN、そしてインターネットとの間のコミュニケーションの面からいろいろ見てみます。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

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13回 集計処理を分散させる意味って何ですか?

本連載では一貫して「集計はアプリケーションサーバ(以下、APサーバ)ではなくデータベースサーバ(以下、DBサーバ)ですべき」と主張してきました。一方で「DBサーバでの集計は負荷が高い」との意見も根強くあります。正しいのはどちらなのでしょう? 性能問題はハードウェアの観点で考えると解決策が見えてきます。

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。

SQLで集計をすると処理を分散できない?

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。

「生島さん、ご相談が!」といつものように大道君がやってきました。

「今日はどないしたん?」

「実は最近協力会社でこういうことを言う人が出てきまして……、直感的にオカシイとは思ったんですけど、本当のところ筋道立てて理屈を説明したらどうなるのか、を確認しておきたいんです」

▼リスト1 集計型/非集計型SQL

```
集計型SQL
SELECT SUM(billing), MAX(billing), AVG(billing)
FROM order
WHERE order_date = '2017/3/1';

非集計型SQL
SELECT billing
FROM order
WHERE order_date = '2017/3/1';
```

【主張1】
APサーバは簡単に台数を増やせる(スケールアウトしやすい)が、DBサーバはそれが難しい。そこで負荷のかかる集計処理を分散させたいのだが、SQLで処理をしていては分散できないので、AP側で集計させたほうが良い。

「集計処理を分散させたい、と言ったんかい? たとえば超単純化して言えば**リスト1**みたいな?」

リスト1はorderテーブルのbillingカラムを1日分集計するコードの例です。集計型SQLの場合は合計、最大、最小値の集計をDBで行い、非集計型SQLの場合はbillingカラムの生データをごっそりAPに転送してAP側で集計することになります。

「そうなんですよ」

「こういうコードだったら、集計処理をAP側に持っていく意味はゼロどころかマイナスやで」

「そのへんの理屈をお願いします!」

「まとめて処理する」のは本来SQLの得意分野

当連載の第1~3回あたりですでに触れた話題ですが、「同じ型のデータをまとめて処理する」

のは本来SQLの得意分野で、合計や平均といった単純な集計処理はその最たるものです。それをわざわざAP側に持っていく意味はまったくありません。しかし現実には前述のように「集計をDBでやるとDBの負荷が高くなる」という誤解をよく耳にしますので、ここで一度整理しておくとしましょう。

まずは**リスト1**の非集計型SQLのようなコードを走らせた場合に、DBサーバとAPサーバのどちらでどのような処理が行われるかをまとめたのが**図1**です。

図1ではorderテーブルに集計対象のデータが5件、それぞれbillingの値が100から500まで5つあるイメージでとらえください。すべてを合計すると1500になります。①に示すように、処理開始前、orderテーブルのデータはDBサーバのハードディスク(以下、HDD)上にあります。

「なぜ①のbillingカラム位置をそろえて書いてないんですか?」

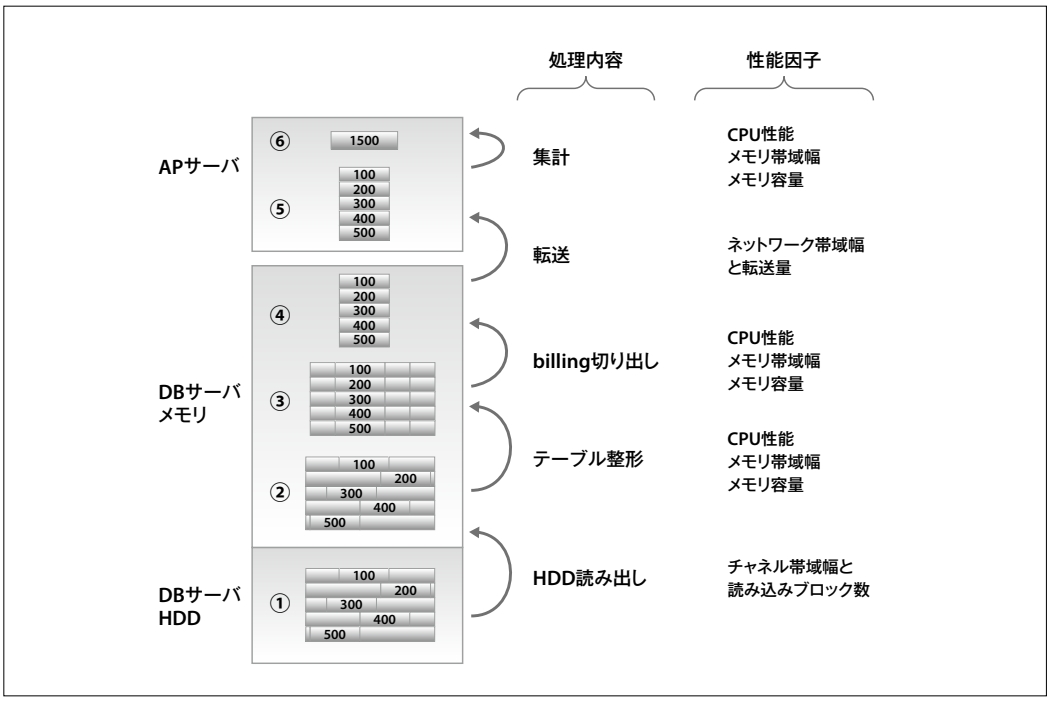
「実際にこういう形で保存されているからや」

データベースは通常、縦・横に整然とそろった「テーブル」形式でデータを扱いますが、HDD上の記録領域はテーブル単位で区切られているわけではなく、たとえばMySQL(InnoDB)の場合はデフォルトでは16KBのデータブロック単位でHDDの読み書きをします。100バイトのレコードを1つだけ読みたい場合でも、16KBをドカンと読んでその中から必要なものだけを残すわけです。HDD上のデータブロックには複数のレコードが書き込まれるため、たとえばbillingのデータは①のようにデータブロック上のあちこちに散らばって存在します。集計するためにはそれをかき集めなければなりません。

DBで集計したほうが低負荷になる理由とは

最初に行うのが①→②の部分、HDDからメモリへデータを読み出す処理です。ここはHDD上のデータをブロック単位でそのままドカンとメモリに読み出すだけなので、①と②は同じフォー

▼図1　非集計型SQLを使った場合の集計処理の流れ



マットで記載してあります。

次の処理はテーブル整形(②→③)で、HDDから読み出したままの生データ②を、扱いやすいようテーブルの形式③の形に整形します。②のデータを分割したりフォーマット変換をしたりしてメモリ上にテーブルを作るわけです。

このテーブルのうちの必要な部分、今回はbillingカラムの値を切り出し(③→④)、それをAPサーバに転送し(④→⑤)、APサーバ側で集計すると(⑤→⑥)、1500という集計結果が得られます。

「ああ、こういうことなんですね……それじゃ、集計型SQLを使うと⑥まで全部DB側でやって、最後の1500だけをAPに転送するわけですね？」

「そのとおりや。その場合、集計するのとしないのでは、DB側の負荷はどこが増えてどこが減ると思う？」

「集計をDBでやるわけだからそこは増えますよね。でも……転送が必要なデータ量は減るのと、転送のためにbillingカラムだけを切り出す処理は不要になりますよね」

「そのとおり！　すると問題はその損得が差し引きいくらよ？ってこと」

「やっぱりDBで集計したほうが得なんですよね？」

「結論はそうだけど、その根拠がいるんよ」

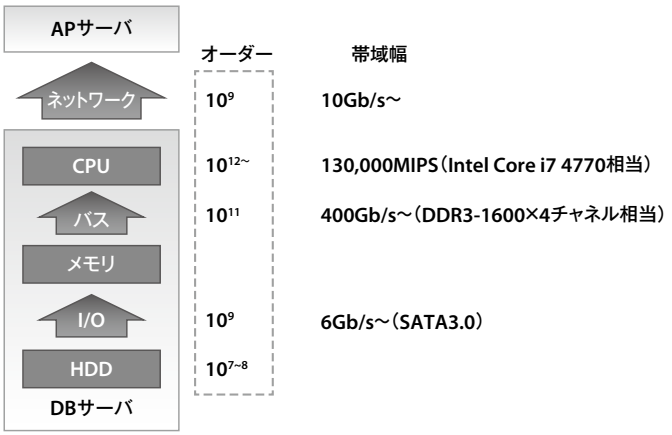
CPUとHDDの速度差は圧倒的

というわけで、次はサーバに使われるおもなインターフェース(以下、I/F)の帯域幅の目安をまとめた**図2**を見ましょう。たとえば低価格のPCやサーバのHDD接続によく使われるSATA3.0の規格は6Gb/s(ギガビット毎秒)、これをバイト単位に換算してざっくりと10のN乗のオーダーを示した数字を「オーダー」の欄に記載しておきました。HDDに使われるI/FにはSerial Attached SCSIやFibre Channel、InfiniBandなどもあり、ものによってはSATA3.0の10倍ぐらいにもなりますが、当記事で細かい数字を気にしても意味がないのでだいたいの目安として10の9乗(バイト/秒)のオーダーと思ってください。

一方、メモリ～CPU間の転送速度は10の11乗オーダーで、HDD I/Fとは2桁違います。これはDDR3-1600×4チャネル利用時のレートで、最近のDDR4規格ならその約2倍です。

さらにCPUそのものの演算速度を無理矢理換算すると10の12乗オーダー以上になります。演

▼図2　おもなインターフェースの帯域幅



算速度を帯域幅と呼ぶのも変ですが、ざっくり言ってメモリの転送速度の10倍以上あると思ってかまいません。だからこそCPU内にキャッシュメモリが必要になります。

こうしてみると、CPUとHDD I/Fではスピードが3桁以上違うわけです。これが、**リスト1**のような単純集計をDBで行っても負荷が増えない理由です。

「ああ、それにそもそも、集計の対象データってHDD上では散らばって記録されているから、余計に読み出しスピードは遅くなるわけですよね?」

「そのとおり!」

図1の①のイメージどおり、orderテーブルにはbilling以外のカラムもあるためHDDのデータブロック上にはbillingのデータはチラホラ散在しているだけです。つまり集計に必要なbillingデータだけをHDDから読み出す速度は10の9乗より2〜3桁以上遅いのが普通です。

さらに言えば、SATA3.0の6Gb/sというのはあくまでも「I/F」の上限性能であって、HDDのディスクそのものという物理媒体からの読み出し速度はこれより遅くなります。とくに不連続領域からの読み出しになると1〜2桁落ちることもあり、CPUとの差がさらに大きくなります。

もっとも、最近はメモリが豊富に使えるようになっているので、通常はバッファヒット率が90%を超えるように設定するため、その都度HDDまで読まずに済む場合が多くなりました。そのためHDDがボトルネックとなる性能問題は起きにくくはなっていますが、基本概念としてはこのとおりです。

ネットワーク転送のための余計な処理も増える

というわけで、帯域幅を考えれば、集計をAP側で行ってDBの負荷を減らせたとしても誤差の範囲でしかありません。しかもそれにより余計な処理も増えるため実際には減らないでしょう。

「余計というのは、APサーバへのデータ転送量と、billingの切り出しそのものですか……」

当然ですが、billingの明細すべてを転送するほうが集計結果だけを転送するよりデータ量は増えます。仮にデータ件数が100万件でbillingが4バイト整数なら少なくとも4MB必要です。当然1パケットでは送れませんので何回も通信プロトコルを処理する必要があります。さらに、集計をDB側でやるなら**図1**のbilling切り出し(③→④)の処理は本来必要ありません。これにかかるCPU負荷のほうが集計の負荷より高いのは明らかです。一方、もしbilling切り出しをせずにorderテーブルをまるごと転送するとさらにデータ量が増え、しかもネットワーク転送の帯域幅は10の9乗オーダーで、やはりメモリより格段に遅いためここがボトルネックになってしまいます。

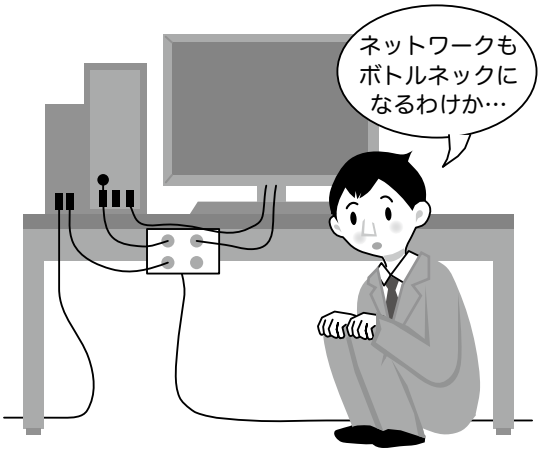
それでもAP側に明細を転送する意味があるとしたら?

「というわけで、単純な集計だったらもともとのCPU負荷自体がほとんどないから、DBでやらずにAPに飛ばす意味はないんよ」

「単純でなければ、意味があるんでしょうか?」

「**図1**の中で、ここから先はAPに飛ばしたほうがいい、というような重い処理がどこかにあると思う?」

「うーん……HDD読み出しはDB側でしかできないし、処理が重そうなところというと、あとはテーブル整形ですかね……?」



問題はそのテーブル整形時にソートやjoinがからむ場合。ソート処理はCPUとメモリへの負荷がたいへん重く、データの件数がn倍になると平均的な計算量は$n \times \log_2 n$倍に増えます。したがってインデックスのない項目にorder byを付けて大量のデータを取得するときは注意が必要です。

「その場合はSQLには、order byを付けずに③以降をAP側に飛ばしてソートをかけたほうがいいということですか？」

「そういうケースは考えられなくもない。joinやgroup byを使うときも暗黙のうちにソートが走ることがあるから、同じことが言えるよ。ただそれはそれで欠点があって……」

「それって巨大なテーブルを丸ごとネットワーク転送するようなものですよね？」

「それや。だから、損得は微妙なんよ。データ量とサーバのスペックによるから、どっちがいい、と一般論では言われへん。ただ少なくとも言えるのは、集計自体に負荷がかかるわけじゃないってこと。問題は集計のためのデータ整理のほうなんで、そこに触れずに集計の問題のように口にしている時点でオカシイわな」

負荷はピークではなく面積で考える

ということで、集計処理のデータが流れるI/Fの速度を考えれば、本日最初の「主張1」は間違いということはわかるはずなのですが、RDBについてよく知らない人が性能の議論をすると、このような非現実的な誤解をしやすいようです。

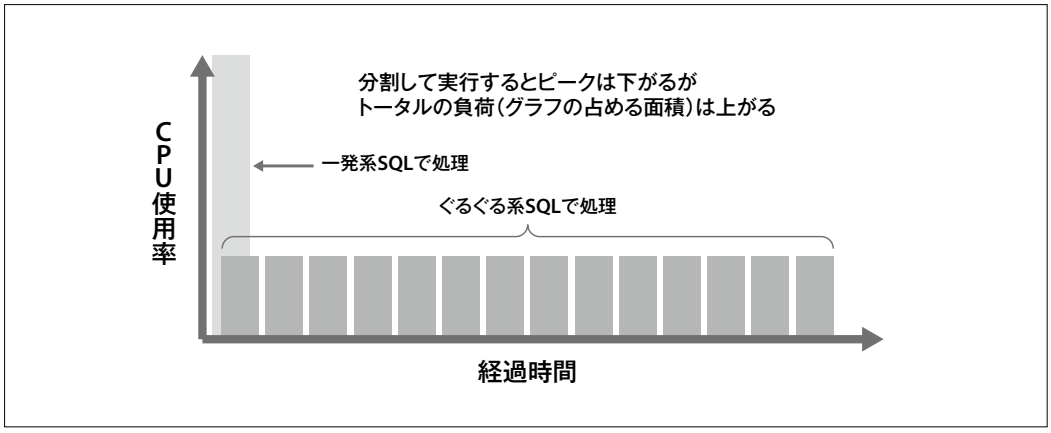
そんな「非現実的な誤解」で、もうひとつよくあるのが次の主張です。

【主張2】
複雑なjoinを使った一発系のSQLで集計処理をするとDBの負荷が高くなるので、joinを使わず、ぐるぐる系の処理をするほうが良い。

当連載の第4回で触れましたが、一発系というのはSQL文を一度投げてすべての処理を行わせるような方法、ぐるぐる系は同じ結果を得るために単純なSQLをデータ件数分繰り返し投げて行う方法です。

図3に示すように、同じ結果を得る場合でも一発系とぐるぐる系のSQLでは負荷のパターンが違います。一発系では瞬間的にピークが高くなりますがすぐに下がるのに対して、ぐるぐる系では低い負荷が長く続きます。瞬間的なピークが気になるかもしれませんが、**図3**で言えば一発系は「瞬間的」なので同じ処理がもう7、8回入ってもまだ余裕があります。一方、ぐるぐる系は長く続くためほかの処理と重なりやすく、同等の処理があと3本並行で走ると負荷が100%に張りついてしまいます。この場合、トータ

▼図3　一発系とぐるぐる系のCPU負荷イメージ



ルの負荷は「グラフに占める面積」で考えなければならないので、ピークの低さに惑わされないようにしましょう。メモリやネットワークの負荷についても同じことが言えます。

SQL解析のオーバーヘッドも無視できない

このようなことが起きる理由の1つは、SQL文の実行に要するオーバーヘッドです。**図4**に示すように、SQL文が実行されるまでの間にDBMSはシンタックスチェック、パースからコンパイルまでのさまざまな処理を行わなければならず、これが非常に重いのです。プリペアドステートメントを使うことによりある程度は解消できますが、基本的にはSQL文を投げる回数に応じてかかるので、その分だけ「ぐるぐる系」のほうが一発系よりもトータルの負荷が重くなるわけです。

低い階層の動作イメージを持つことが重要

「あ、なるほど……そうですね、わかります！　そうですよね……こういうイメージがわかると、なんなんでしょう、すごく納得感あります！」と大道君。

「結局、モノが動くにはみんな理屈があるんで、どこがどう動いてどれだけの性能出せるか、ちゃんと理屈をたどっていけばざっくりしたイメージが描けるはずなんよ」

「ざっくりしたイメージというのは**図3**のグラフや**図2**のオーダーみたいなところですよね」

「そうそう、ああいうイメージを持っておくと、根本的にオカシイ誤解はしないで済むわけや」

「1つ確認したいんですけど、SQLって本来はファイルの物理構造からデータ項目を切り離して、抽象化したテーブルという概念で集合的に扱えるようにしたもの、ですよね？」

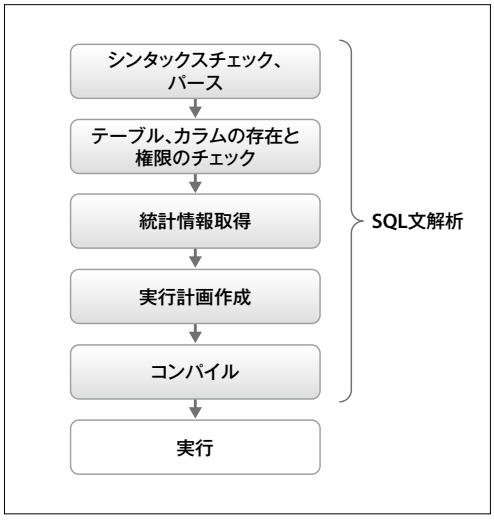
「そのとおり！」

「抽象化してあるから物理構造は忘れていられる、けれど実際には物理的なしくみで動いているんだから……性能問題を考えるときはそこまで考えないといけないんですね？」

「そういうこと！　今回データブロックとかI/Fの帯域幅とかの話が出てきたように、OSよりもハードウェアに近い、低い階層が動くイメージを持っておくことが、こういうときには重要なんよ。ざっくりしたものでかまわんけど、それは意識しといてや」

「了解です！」SD

▼図4　SQL文の解析に要するオーバーヘッド



思考をカタチにするエディタの使い方

るびきち流 Emacs超入門

Writer るびきち
twitter@rubikitch
http://rubikitch.com/

第34回 カーソルを分身させて編集! multiple-cursors

カーソルが忍者のように分身し、複数個所を同時に編集できるようになるパッケージ「multiple-cursors」を紹介します。コマンドの手順とハマりどころを確認したら、さっそく使ってみましょう。またphi-searchなどと組み合わせることで、さらに高度な操作も可能になります。

Emacs存亡の危機は回避された

どうも、るびきちです。2016年の年末は、まさかのEmacs存亡の危機[注1]が話題になっていましたね。筆者がソースコードを調査したところ、無事に回避されたことがわかりました。そして大晦日にはEmacs 25.2のpretestがリリースされました。本稿が世に出るころには25.2がリリースされているかもしれません。我々は安心してEmacsを使い続けられます。Emacs文化は不滅です！

カーソルを分身させる

まずはインストール

今回はカーソルを“分身”させるmultiple-cursors.elを取り上げることにします。矩形編集や置換よりもきめの細かい編集を複数行同時に行うことができる、超強力なパッケージです。筆者も日々愛用しています。

それでは、multiple-cursors.elのインストールをしましょう。本領を発揮させるためには、次のパッケージもあると良いです。

- phi-search：multiple-cursors専用isearch
- all-ext：正規表現にマッチした行を一覧編集
- helm-swoop：インクリメンタル絞り込み検索

どれもMELPAに登録されていますので、一緒にインストールしましょう（**リスト1**）。

メリット

multiple-cursors.elの大きなメリットは、連続した行、同じ単語・シンボルについて同じ編集操作が行えることです。

連続した行に対して同じ編集をするならば、ほかにも標準機能の矩形編集があります。とくにC-x r t（string-rectangle）は、連続した行に同じ文字列を挿入する便利なコマンドです。複数行の異なる列をregionにした場合は、左上から右下を対角線とする長方形の範囲を、同じ文字列で埋め尽くします。ちなみにEmacs 25.1からはC-x r tでの入力がリアルタイムに反映されるようになったため、multiple-cursors.elっぽい挙動になったと言えます。

けれども、multiple-cursors.elのM-x mc/edit-linesはその上をいきます。C-x r tででき

▼リスト1　multiple-cursors.elと関連パッケージをインストール

```
M-x package-install multiple-cursors
M-x package-install phi-search
M-x package-install all-ext
M-x package-install helm-swoop
```

注1）「Emacsは衰退しました」
URL http://qiita.com/itckw/items/ff079c7572d6a1acd349

ることは、桁数(N)回C-dを押すかC-u N C-dで削除し、新しい文字列を入力するだけです。

multiple-cursors.elでは、C-aやC-eで行頭や行末に移動できます。また、phi-search.elがインストールされているならばC-sで文字を指定して移動できます。そのため、行頭・行末や行間に対して同じ文字列を挿入することも、いとも簡単にやってのけます。キーボードマクロでも同様の操作を実現できますが、それには慣れが必要です。

何行にも渡る高度な編集を簡単にしてしまうのが、このmultiple-cursors.elです。リアルタイムに編集状態が反映されますので、正規表現置換が苦手な人でも簡単に扱えます。

デメリット

とはいえ、デメリットがないこともないです。

まず、multiple-cursors.elは巨大なパッケージでコマンドが多過ぎることです。数えてみると40個以上もありました。当然すべてを使いこなすことは難しいですし、キー割り当てにも困ってしまいます。

もうひとつの欠点は、分身カーソルが離れてしまうと、一覧できないことです。分身カーソルを見た目上近づけることはできますが、「一覧」とは言い難いです。

幸いこれら2つの欠点は、後述するように完全にカバーできます。

複数行を一度に編集する

M-x mc/edit-linesで十分

あまりにも多機能なmultiple-cursors.elですが、重要なコマンドはたった1つです。M-x mc/edit-linesは、regionの各行にカーソルを分身させます。これさえあればmultiple-cursors.elの恩恵を十分に受けられます。

M-x mc/mark-all-symbols-like-thisを使うと、バッファ内すべてのシンボルや単語に分身カーソルを置いてマークできます。この方法は確かにとても派手ですが、次の理由から筆者はお勧めしていません。

- 不必要な分身カーソルを確認して除去する必要がある
- 除去するのに別のコマンドが必要
- コマンドを使い分ける必要がある
- より簡便かつ強力な方法がある

具体例

それでは、分身カーソルを使ってみましょう。たとえば、

```
This is an orange.
That is a bread.
These are cakes.
```

を、

```
This is an [orange].
That is a [bread].
These are [cakes].
```

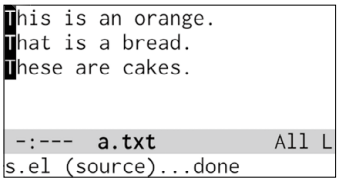
に変換してみます。

確かに、標準のキーボードマクロだけでも十分ですが、分身カーソルならば一度にリアルタイムにフィードバックされます。列はどこでも良いですので、1行目でC-SPC、3行目でM-x mc/edit-linesを実行します(**図1**)。すると、3つの分身カーソルが出ます。そこで、C-aを押せばそろって行頭へ移動します(C-eを押せば行末へ)。あとは、一番上の行のみを編集する感覚で操作します。

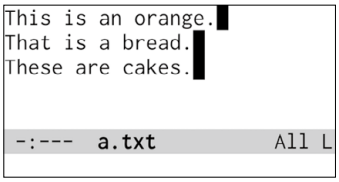
①C-eで行末へ移動(**図2**)
②C-b]で閉括弧を入力
③M-bで前の単語に移動
④[で開括弧を入力(**図3**)
⑤C-gで分身カーソルを終了する

実際に操作してみればわかりますが、分身カーソルやキーボードマクロ以外の方法ではとても面倒です。

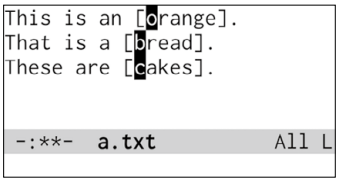
▼図1　M-x mc/edit-linesを行頭で実行



▼図2　C-eで同時に行末に移動



▼図3　一度に複数行を編集



プロンプトが出てきたら

multiple-cursors.elは、定番のコマンドについては複数行同時に実行すべきかそうでないかを自動判別します。

たとえば、C-x C-s(save-buffer)やC-v(scroll-up-command)は一度だけ実行するべきコマンドです。C-y(yank)やM-u(upcase-word)は複数行同時に実行すべきコマンドです。

知らないEmacsコマンドが入力されたとき、そのコマンドがどう振る舞うべきかは、multiple-cursors.elは知る由がありません。自作コマンドや他パッケージのコマンドについては、ときおり、「Do mykie:global-map:C-e:key for all cursors? (y or n)」といったプロンプトが出てきます。「現在実行したばかりのmykie:global-map:C-e:keyコマンドをすべての分身カーソルにおいて使いますか？　すべての分身カーソルに適用したければyと、そうでなければnと答えてください」という意味です。

質問の答えは設定ファイル「~/.emacs.d/.mc-lists.el」に即座に保存されます。もし間違ってしまった場合は、

①設定ファイルを開き、
②C-sで該当コマンドを検索
③C-a C-kで削除
④C-M-xを押してS式を評価

と操作してください。すると、再度分身カーソル上で該当コマンドを実行したときに同じ質問をされます。

all-extから使う

mc/*-like-thisはM-x allで十分

multiple-cursors.elにはM-x mc/*-like-this系列のコマンドが20個近くもあります。これらはシンボルや単語を同時にマークしたりマークを解除したりするコマンドですが、とくに使わなくてもかまいません。

実は前回(本誌2017年2月号)紹介した拙作all-ext.el[注2]は、multiple-cursors.elとの連携ができます。*All*バッファ上でC-c C-mを押せば、マッチ行全体に分身カーソルを出せます。

すなわち、前回紹介したM-x allで同じシンボルや単語を一覧しながら編集すれば良いのです。M-x allの中でC-c C-mを使えばマッチした行全体に分身カーソルを出せます。しかもこのとき、マッチした部分に各分身カーソルが移動した状態になりますので、M-x mc/*-like-this同等の編集が行えます。

それどころか、helmと組み合わせればMigemo対応絞り込み検索や除外指定も簡単にできます。M-x mc/unmark-*系のコマンドも、もはや覚える必要はありません。

せっかく多機能であっても、いざというときに使いこなせなくては意味がありません。すぐに使える機能と組み合わせて済むのであれば、それで十分ではありませんか。

注2) all-ext.elを使うには設定にて(require 'all-ext)が必要です。

helm-swoopで絞り込む

前回のおさらいですが、all-ext.elを使えばhelm-occurやhelm-swoopの結果を*All*バッファで編集できます。

ただし、multiple-cursors.elと連携して分身カーソルをマッチ個所に移動させる機能はhelm-swoop限定ですので、helm-swoop.elをインストールしてください。実はhelm-occurなどにも対応させようと試みましたが、どうしてもうまくいかず、helm-swoop限定にしました。

①M-x helm-swoopを実行する
②クエリを入力して行を絞り込む(**図4**)
③編集対象の行をマークする(任意)
④C-c C-aで*All*に移行する
⑤C-c C-mでマッチ個所に分身カーソルを配置し(**図5**)、自由に編集する(**図6**)

helm-swoop.elとall-ext.elとmultiple-cursors.elを組み合わせると、マッチ行に対して本当に何でもできてしまいます。ターゲットを絞って、好き放題に編集してください!

◆ ◆ ◆

今回は複数行同時編集のmultiple-cursors.elをお届けしました。helm-swoopやallと組み合わせると、高度な編集が楽々できてしまうことに驚くはずです。

次回は復活させたanything.elを取り上げる予定です。今はhelmという名前でforkされて活発に開発されていますが、anything.elはあえて長年寝かせておくことで、仕様をほぼ確定させました。昔ながらの仕様ですが、Emacs Lisp初心者でも簡単にオリジナルコマンドを使えることを念頭にしながら、今後は開発していきます。Emacs Lispにもやや踏み込みますが、その簡便性をお伝えしますね。

筆者はいろいろなことに興味を持っているため、サイトはサブドメイン単位で再構築しました。「日刊Emacs」改め「新生日刊Emacs」(http://emacs.rubikitch.com/)は日本語版Emacs辞典を目指し、毎日更新しています。やりたいことやパッケージから探せるようにしましたので、Emacsで何か実現したいことがあればぜひともご覧になってください。手元でgrep検索できるよう全文をGitHubに置いています。

▼図4 M-x helm-swoopでfooにターゲットを定める

```
(require 'all-ext)
(defun test ()
  (setq foo-a 10)
  (setq foo-b 20)
  (setq foo-c 30))

 -:---  test.el        All L3     (Emacs-Lisp)
[C-c C-e] Edit mode, [M-i] apply all buffers
test.el
3   (setq foo-a 10)
4   (setq foo-b 20)
5   (setq foo-c 30))

 *Helm Swoop* L1    [3 Candidate(s)]   C-c ?:Help TAB:Act RET/f
Swoop: foo
```

▼図5 C-c C-aで*All*に移行したあと、C-c C-m

```
(require 'all-ext)
(defun test ()
  (setq foo-a 10)
  (setq foo-b 20)
  (setq foo-c 30))

 -:---  test.el        All L1     (Emacs-Lisp)
From helm-occur
--------
      3:  (setq foo-a 10)
      4:  (setq foo-b 20)
      5:  (setq foo-c 30))

U:---  *All*          All L3     (All mc:3)
```

▼図6 barに置き換えると、そのまま反映される

```
(require 'all-ext)
(defun test ()
  (setq bar-a 10)
  (setq bar-b 20)
  (setq bar-c 30))

 -:**-  test.el        All L1     (Emacs-Lisp)
From helm-occur
--------
      3:  (setq bar-a 10)
      4:  (setq bar-b 20)
      5:  (setq bar-c 30))

U:**-  *All*          All L3     (All mc:3)
```

またEmacs病院兼メルマガのサービスを運営しています。Emacsに関すること関しないこと、わかる範囲でなんでもお答えします。「こんなパッケージ知らない?」「挙動がおかしいからなんとかしてよ!」はもちろんのこと、自作elispプログラムや文章の添削もします。集中力を上げるなどのライフハック・マインド系も得意としています。SD

登録はこちら➡http://www.mag2.com/m/0001373131.html

一歩進んだ使い方のためのイロハ

Vimの細道

第16回

mattn
twitter:@mattn_jp

QuickRunで開発を加速する（後編）

書いたプログラムをその都度実行できる便利なプラグインQuickRunを、2回に渡って取り上げます。後編ではQuickRunを自在にカスタマイズするための設定方法を解説します。最後にはQuickRun実行を拡張できるプラグイン「QuickRunEx」も紹介。

QuickRunを自在にカスタマイズする

前回（本誌2017年2月号）はQuickRunの基本的な使い方を説明しました。基本的な使い方だけでも十分に便利なのですが、実は簡単な設定を行うだけで、QuickRunの動作を変えたり、QuickRunが本来提供していないファイルタイプに対応できます。前回も説明しましたが、QuickRunはGitHub[注1]からインストールできます。

デフォルト動作の変更

QuickRunにデフォルトで用意された設定では、環境に合わせて実行できるコマンド、およびコンフィギュレーション（オプションなどの設定）が自動で選択されます。たとえばC言語のコンパイラであれば、Visual C++ Compiler、GNU C Compiler、Clangの中から実行可能なコンパイラが自動で判別され、そのコンパイラに合わせたコンフィギュレーションが選択されます。

リスト1はQuickRunで用意されているC言語コンパイラの設定です。コンパイラが実行可能かを判定して、フラグやファイル名などのコンフィギュレーションが自動で設定されます。

また環境に複数コンパイラがインストールされている場合でも、QuickRunの起動オプションを設定することで。特定のコンパイラを指定して実行できます。次は、Visual C++ Compilerを使って実行する方法です。

```
:QuickRun c/vc
```

JavaScriptではSpiderMonkey、V8、NodeJS、PhantomJS、Rhino、CScriptが、LispではSBCL（Steel Bank Common Lisp）、CCL（Clozure CL）、CLISPコマンドが選択されます。しかし、環境によってはコマンド名が標準と異なっていたり、QuickRunのデフォルト設定では提供されていない場合もあります。そういった場合にも、QuickRunは簡単にカスタマイズできる方法を提供しています。

新しいコマンドへの対応

前述のように、JavaScriptで起動できるコマンドは自動で選択されますが、執筆時点（2017年1月）ではES6（ECMA Script6）に対応していません。ES6の処

注1）URL https://github.com/thinca/vim-quickrun

▼リスト1　C言語のコンフィギュレーション

```
\ 'c': {
\   'type':
\     s:is_win && executable('cl') ? 'c/vc'  :
\     executable('gcc')            ? 'c/gcc' :
\     executable('clang')          ? 'c/clang' : '',
\ },
```

▼リスト2　JavaScriptの実行にBabelを使う設定

```
" QuickRun オブジェクトを用意する
let g:quickrun_config = get(g:, 'quickrun_config', {})

" javascript/babel を定義する
let g:quickrun_config['javascript/babel'] = {
\  'command' : 'babel',
\  'exec' : ['%c %o %s:p -o %s:p.babel', 'node %s:p.babel'],
\  'hook/sweep/files': '%S:p.babel',
\}
```

▼表1　execで指定できるシンボル

記号	意味
%%	%自身
%c	コマンド
%o	コマンドラインオプション
%s	ソースファイル
%a	スクリプトの引数

理系実装であるBabelを使いたい場合には**リスト2**の設定を行います。

commandには実行コマンド名のbabelを、execには実行する際のコマンド形式を指定します。hook/sweep/filesはテンポラリファイルを後片付けするためのおまじないです。hookについては後述します。

この設定を行ったうえで拡張子jsのファイルを開き、次を実行すると、ES6のファイルがbabelコマンドによりJavaScriptファイルへと変換され、その出力されたファイルがnodeコマンドにより実行されます。

```
:QuickRun javascript/babel
```

コマンドの指定方法

前述したexecの設定は、%で始まる記号を使って記述します。実行時にこれらの記号が置き換えられて実行されます。**表1**にexecで指定できるシンボルの一覧を示します。

%cと%sに渡されたファイル名などはエスケープされます。またそれらの大文字版、%Cと%Sについては、Vimのファイル名修飾(ファイル名の一部を取り出す機能)を使用して、ファイルのあるディレクトリや拡張子のないファイル名に変換できます。

```
java %S:.:r:gs?[/\\]?.?
```

この例では、ファイル名を相対パス(:.)に変換し、拡張子を除いて(:r)、\や/といったパスセパレータを.に置換して(gs?[/\\]?.?)、javaコマンドで実行できるクラス形式に変換しています。詳しくは、Vimのヘルプ:help filename-modifiersを参照してください。

新しいファイルタイプへの対応

QuickRunの設定はVimのファイルタイプにひもづけられています。先ほどのES6の例では、javascript/babelを設定しましたが、同様にVBS(Microsoft Visual Basic Scripting Edition)のファイルを実行する設定を作ってみます。

QuickRunは実行時にファイルタイプ名をキーに設定を調べ、見つかった場合にはその設定を使用します。見つからなかった場合には、そのファイルタイプ名をコマンド名として置き換えて実行します。Vimで拡張子vbsのファイルを開くとvbというファイルタイプ名が設定されますので、quickrun_configのキー名はvbとなります。

VBSファイルはcscriptというコマンドで実行できるのでcommandにcscriptを、execには%c /E:vbs /nologo %sを設定します(**リスト3**)。

ここで、/E:vbsというオプションで、スクリプトエンジンをVBSに強制している点に注目してください。QuickRunは編集中のバッファに変更がない場合には、execの%sをファイル名で置き換えるのですが、変更がある場合には変更

▼リスト3　VBSのファイルを実行する設定

```
let g:quickrun_config['vb'] = {
\  'command' : 'c:\\windows\\SysWOW64\\cscript',
\  'exec' : ['%c /E:vbs /nologo %s'],
\}
```

▼リスト4　リスト3に、出力エンコーディングの指定を追加

```
let g:quickrun_config['vb'] = {
\  'command' : 'c:\\windows\\SysWOW64\\cscript',
\  'exec' : ['%c /E:vbs /nologo %s'],
\  'hook/output_encode/enable' : 1,
\  'hook/output_encode/encoding' : 'cp932',
\}
```

中のバッファ内容をいったんテンポラリファイルに保存し、そのテンポラリファイル名を%sに指定します。

この動作はバッファが変更されたまま保存されていなくても内容を実行できるので、便利な機能なのですが、cscriptコマンドはファイル名の拡張子からスクリプトエンジンを決定しています。拡張子がjsであればScriptが実行され、拡張子がvbsであればVBSが実行されます。そこで、/E:vbsというオプションによってエンジンを指定しているのです。

ひとまずはこれで、VBSファイルが実行できるようになります。

フックによるエンコーディングの変更

「ひとまず」と書きましたが、1つ重大な問題が残っています。実は実行時に文字化けが発生します。

日本語ロケールのWindowsをお使いの場合は通常、VBSはShift_JISで記述し、出力もShift_JISで行います。しかし、Windowsユーザの中にはVimのencodingをCP932に設定している人もいれば、UTF-8に設定している人もいます。もしUTF-8に設定している場合には、実行結果が文字化けしてしまいます。

そこで、hook/output_encodeを使って出力エンコーディングの指定を行います(**リスト4**)。

ちなみに、ここではエンコーディング名をcp932と指定しましたが、実はcharという値にしてもかまいません。Vimが使っている文字コード変換ライブラリであるGNU iconvでは本来、IANA(Internet Assigned Numbers Authority)で定義されたエンコーディング名しか扱えないのですが、特別にcharという値が用意されています。これを指定すると現在のロケールに合わせたエンコーディング名が自動選択されます。

たとえば、Windowsでiconvコマンドが使えるならば、コマンドプロンプトで次を実行してみてください。UTF-8のファイルfoo.txtが、Shift_JISで出力されます[注2]。

```
> iconv -f utf-8 -t char foo.txt
```

カレントディレクトリの変更

スクリプトを実行する際に、そのスクリプトファイルが置かれている場所へ移動するにはhook/cdを使います(**リスト5**)。

これも、先ほどのファイル名修飾を使って実現します。:pは%Sをフルパスに、:hはファイル名を取り去ります。

評価テンプレート

たとえばPerlで、ある式を実行した際の結果をいち早く知りたいとしましょう。Perlで式の評価結果をダンプするのであればData::Dumperを使いますが、動作を確かめるためだけに、お決

注2) msysやCygwinのシェル環境ではUTF-8が出力エンコーディングになります。よってこのコマンドをmsysやCygwinで実行すると、逆に文字化けを起こします。

▼リスト5　リスト3に、カレントディレクトリの変更を追加

```
let g:quickrun_config['vb'] = {
\  'command' : 'c:\\windows\\SysWOW64\\cscript',
\  'exec' : ['%c /E:vbs /nologo %s'],
\  'hook/output_encode/enable' : 1,
\  'hook/output_encode/encoding' : 'cp932',
\  'hook/cd/directory': '%S:p:h',
\}
```

▼リスト6　Perlのコンフィギュレーション

```
\ 'perl': {
\   'hook/eval/template': join([
\     'use Data::Dumper',
\     '\$Data::Dumper::Terse = 1',
\     '\$Data::Dumper::Indent = 0',
\     'print Dumper eval{%s}'], ';')
\ }
```

まりのuse Data::Dumperやprint Dumperを書きたくありませんよね。

そこでQuickRunでは、hook/evalというフックを使用して、あらかじめこれらを含んだ内容で評価を行えます。**リスト6**は、QuickRunで設定されているPerlコンフィギュレーションです。これを見るとわかるとおり、ユーザが編集したバッファの内容は%sの部分に挿入されて実行されます。たとえば、ファイルタイプがPerlのバッファに「2 ** 3」とだけ書いて、

```
:QuickRun -eval/enable 1
```

を起動すると、このテンプレートが有効となり、「'8'」という結果が出力されます。

▼リスト7　_で一括設定

```
let g:quickrun_config['_'] = {
\  'hook/shebang/enable': 0,
\  'outputter/buffer/split' : ':botright 8sp',
\}
```

▼リスト8　PowerShellのコンフィギュレーション

```
\ 'ps1': {
\   'exec': '%c %o -File %s %a',
\   'command': 'powershell.exe',
\   'cmdopt': '-ExecutionPolicy RemoteSigned',
\   'tempfile': '%{tempname()}.ps1',
\   'hook/output_encode/encoding': '&termencoding',
\ },
```

shebangを無効に

スクリプト言語の多くはshebangをサポートしており、ファイルの先頭に、

```
#!/usr/bin/perl
```

といったshebang行が書かれています。OSはこのshebangを読み取り、そのコマンドに内容を受け渡します。QuickRunも、このshebangを読み取ってコマンドを実行するしくみになっています。これはhook/shebangというhookにより実装されています。

しかし環境によっては、存在しないパスのコマンドがshebang行に書かれていることもあります。とくにWindowsであれば、/usr/bin/perlや/usr/bin/envといったコマンドがCドライブ直下に存在することは、まずあり得ません。

そういった場合にはこのhook/shebangを無効にして実行することで、ファイルタイプに紐づけられたもともとのコマンドが有効となります。

```
:QuickRun -shebang/enable 0
```

すべてのファイルタイプで有効なオプション

先の例はコマンドラインでshebangを無効にしましたが、Windowsの場合、shebangを有効にしてほしいケースはほとんどありません。設定で、すべての言語で無効にしてしまいたいですよね。

QuickRunでは_というキーに設定を行うことで、すべてのファイルタイプで有効となるコンフィギュレーションを追加できます。**リスト7**では、次の設定がすべてのファイルタイプで有効になります。

・shebangを無効に設定する
・結果出力バッファを画面下に8行分で表示

テンポラリファイル名

VBSの設定にて、QuickRunは変更済みのバッファを実行する際に、テンポラリファイルを使用すると説明しました。cscriptコマンドにはスクリプトエンジンを指定するオプションがあったので問題を回避できましたが、そういったオプションがないコマンドでは、そのコマンドが扱えるファイル名を指定する必要があります。

たとえば、PowerShellはps1というファイル拡張子を実行しますが、ほかの拡張子では実行してくれません。そこで、tempfileアイテムで拡張子をps1に設定します。**リスト8**は、QuickRunのPowerShell用コンフィギュレーションです。

デフォルト設定の上書き

システムに有効なコマンドが複数あった場合、QuickRunは自動でコマンドを選択してしまいます。たとえば、前述のC言語向けのコンフィギュレーション(**リスト1**)であれば、c/vc、c/gcc、c/clangのうち、パス環境変数から先に見つかったコマンドが有効となります。:QuickRun c/vcによる手動指定も残した状態で、デフォルトの選択だけ変更する場合、次の設定を行います。

```
let g:quickrun_config['c'] = {
\  'type' : 'c/clang',
\}
```

これにより、clangがgccよりも優先され、もしgccを使いたい場合にも:QuickRun c/gccで実行できるようになります。

C言語のソースファイルを実行する場合、簡単なものならば標準ライブラリだけで動作するのですが、たとえばWindowsでwinsock.hをincludeする際には、ビルドに-lws2_32というリンクオプションを指定しないとエラーになります。また、C++でboostライブラリのヘッダをincludeする際には、必要なライブラリをリンクする必要があります。そういった場合にはQuickRunEx[注3]が便利です。

QuickRunExは、QuickRunのhook機能を利用して、本来QuickRunでは扱えない動的な拡張を行います。QuickRunExにはデフォルトでいくつか便利な拡張が入っており、**表2**の機能をサポートしています。

また、デフォルトの設定に機能を追加することもできます。**リスト9**のように、ファイルタイプをキーに設定します。

この場合、C言語のソースファイルに、

```
#include <foo/bar.h>
```

といったincludeディレクティブがあった場合は、-DFOOというコンパイルフラグを使い、-lfooをリンクオプションに追加します。

ユーザはソースコードを書いて<leader>-r[注4]をタイプするだけで、そのソースコードを実行できることになります。Makefileを書いてもいいのですが、ちょっとお試しでコードを書きたいときに重宝しています。

2回に渡ってQuickRunの基本的な操作と設定による拡張方法、さらに拡張するためのQuickRunプラグインQuickRunExを紹介しました。QuickRunはとても拡張性のあるVimプラグインですので、アイデアさえあればさらにいろいろな機能を付け足すことができます。ぜひおもしろいものを作ってみてください。SD

注3) URL https://github.com/mattn/vim-quickrunex

注4) 何も設定していない状態であれば<leader>-rは\rで動作します。

▼リスト9　QuickRunEXのデフォルト設定に機能を追加

```
let g:quickrunex_config = {
\ 'c': [['foo/', [' -DFOO', ' -lfoo']]]
\}
```

▼表2　QuickRunExがサポートする機能

対象	機能
libuv	ライブラリをリンク
libgc	ライブラリをリンク
libjannson	ライブラリをリンク
mruby	ライブラリをリンク
wininet	ライブラリをリンク
winsock	ライブラリをリンク
gtk2	pkg-configを使って動的リンク
boost	ライブラリをリンク
v8	ライブラリをリンク
libujson	ライブラリをリンク
fltk2/fltk3	ライブラリをリンク
qt4	ライブラリをリンク
go	パッケージをインストール

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第37回 ラッキーナンバー？

買い物の合計金額が6666円とか7777円になるとうれしくなるけど、おつりの半端感にモヤモヤするときは素数を数えるんだ！

to be Continued

自動車のナンバープレートの下4桁を有料で選択できるサービス「希望ナンバー制度」が始まったのは平成9年。数千円で好きな番号（希望が多いものは抽選）を得られるので結婚記念日や誕生日を付けたり、ギャンブラーのゲン担ぎに利用されているようです。最近「8008」のナンバーが多いなぁと思っていたら「エンジェルナンバー」という不思議なブームが来てるせいらしいです。ナンバーといえば、ITエンジニアとしてウェルノウンポート番号のナンバーを携帯電話で撮影して蒐集しようとしましたが、全然出会えなくて1週間で諦めたことがあります。全然Linuxと関係ない話になってますね。みんなの好きなポートは何番だい？（聞いてどうする）

Sphinxで始めるドキュメント作成術

渋川 よしき　Shibukawa Yoshiki　Twitter @shibu_jp

最終回　ゲームで学ぶドキュメント設計

ドキュメント設計

これまでの連載では、個々のSphinxの機能、本を書くといった特定のユースケースごとに紹介してきました。今回は最後ということで「なぜドキュメントを書くのか」「どう書くか」といったドキュメント設計について説明します。議事録やREADMEのテンプレートの話題があがることもありますが、プログラミング同様、パターンをたくさん覚えるよりも、遠回りでも「フィードバック」の方法を学ぶほうが応用が利きます。

ドキュメントの設計は抽象的な説明になりがちなので、イメージしやすいようにゲームのメタファで紹介していきます。

難易度のデザインは適切に

ドキュメントの読者はRPGの主人公です。50ゴールドと布の服だけ持ってボスの魔王を倒しにいきます。最初はスライムの集団にも苦労するレベル1です。進行度と敵の強さ、お店で売っている武器の価格になぜか正の相関があります。徐々にレベルが上がり、強い武器を手に入れて、強い敵を倒すというサイクルになっています。

「初期装備」は読者の持っている前提知識です。ドキュメントを読みながら敵を倒し（文書を理解し）てレベルを上げます。あなたが読者に伝えたかった一番難しい情報がボスです。RPGに例えるとドキュメントに必要な特性が見えてきますよね？　まずは「こんなゲームはやりたくない」というパターンを考えてみましょう。

- プレイヤーのレベルアップが追いつかないで、急速に強い敵が出てくる
- 延々と弱い敵ばかり出てきて退屈
- イベント進行に必要なアイテムが、イベント進行後じゃないと取得できない
- 何の説明もなく、脈絡もないボスが登場する

これらはどれもドキュメントにも通じるアンチパターンです。ドキュメントを書くときも、まず最終的にユーザに伝えたい知識を項目にばらし、適切に配置して目次を設計します。ビジネス書であれば目次とタイトル、表紙の帯の文言で売上の半分以上が決まるそうです。みなさんも本を買うときは、目次を眺めてから決める人は多いでしょう。数ヵ月かけて目次だけを決める著者もいます。このメタファはどちらかというと本1冊分書くような超長文ほどイメージしやすいですが、短い文章でも同じです。技術系ブログではたまに、ツール名だけ書かれていて、ゴールが書かれないまま作業手順だけ書かれているメモもあります。初期装備（ツールが何者かという前提知識）も倒すべきボス（解決すべき課題）もなく、ただ敵だけが出てきてもRPGの主人公に感情移入できません。短くても「プレイヤー成長のストーリー」を作ることが大切です。

Sphinxではセクションタイトルとtoctreeを駆使してストーリーを作っていきます。Sphinx

でも最初は章ごとぐらいの粒度でファイルを作り、セクションタイトルと説明すべき内容のメモだけを書いて全体を構成します。説明すべき内容や説明に必要な知識の前後関係、分量のバランスを整理します。執筆中に順序を入れ替えても、トップに表示される索引は最新の情報を反映したものになります。

マルチシナリオ／マルチエンディング

アクションRPGの場合、反射神経や類似ゲームのプレイ経験の有無などで、スタート時点で実力がすでに開いています。すべての人に同じ満足感を与えるのは難しいため、難易度が切り替えられるようになっていたり、不要ならチュートリアルもスキップできたりします。

現実のドキュメントは読者から見るとアクションRPGに近いでしょう。ドキュメントを作成するときは「難易度ふつう」のユーザのレベルをまず決めます。プログラミング初心者向けの丁寧なプログラミング言語の本もあれば、「変数」「関数」といった概念を知っている人向けのコンパクトな本もあります。ユーザのレベルごとに必要とするコンテンツが変わってきます。フォーカスを絞ることで文章の難易度が安定します。すでに十分に知っている人であれば、新バージョンの更新履歴のリファレンスだけあれば満足かもしれません。初めての人向けであればチュートリアルやハンズオンで触って学べるページを作っておいて、概要を伝えるのも良いでしょう。

場合によってはレベルだけではなく、必要とする分野も違うことがあります。ちょっとしたツールでも次のようないくつものロールがあります。

- 概要を把握したい人向けの紹介
- CI設定者向けのビルド／設定方法
- ユーザ向けの機能紹介
- メンテナ向けのコード解説

Pythonのドキュメントも**図1**のように「チュートリアル」「ライブラリリファレンス」「C拡張開発者用」と用途ごとに分類されています。Sphinxでも、トップページなどのわかりやすいところにリンク集を作成し、自分の体験したいものを自分でテーラーメイドできるドキュメントにします。C言語拡張開発ドキュメントの奥に初心者チュートリアルなどを配置することはありえないですよね？　情報はたくさん伝えればいいというものではありません。ユーザにとって不要な情報は隠すほうが良いこともあります。不要な情報をこれでもかと詰め込むのはマウンティング、あるいはマンスプレイニングと呼ばれる動物的な行為です。本連載を嗜(たしな)む紳士淑女には似合いません。

ゼルダの伝説効果

「金槌(かなづち)しかもっていないと、すべてのものが釘に見える」という格言があります。戒めの言葉ですが、実際にそれで解決できると全能感が感じられて気持ちは高ぶります。この金槌・釘効果がゲームのコアデザインとして使われているのがゼルダの伝説シリーズです。ダンジョンの中にはなぜか毎回道具が落ちています。その道具を使うとなぜか道がひらけてボスの部屋にたどり着けますし、ボスの弱点だったりもします。

得た情報がすぐに役立って即座に成果を感じ

▼図1　Python.orgのドキュメント例

られると、愛着もわいて知識の吸収は早くなります。概略などでイメージを頭の中に作り、それをもとにしてより複雑な抽象度の高いボスを攻略できると効果絶大でしょう。イメージを作る方法には大きく4つあります。

①図表で全体像を伝える
②コードサンプルを動かして体感してもらう
③本記事のようにメタファや例え話を使う
④Aという簡単な内容を説明して、そのAをもとにBという内容を攻略する

Sphinxでは図表としてはimageディレクティブによる挿絵、表、blockdiagシリーズ[注1]などが使えます。HTML出力ではスクリーンキャスト動画を入れるのも良いでしょう。コードサンプルはcode-blockディレクティブを使います。サンプルコードは断片だけではなく、なるべくそのまま動く状態が望ましいです。また結果もあるほうがいいでしょう。文字ばかりで抽象的な説明が多いなと思ったら使ってみてください。

④については少し注意が必要です。抽象化されて簡潔にされたテキストは分量そのものは少なくなりますが、理解する難易度は上がります。IT業界はオブジェクト指向が根付いているせいか、抽象的なレイヤから説明したがる人が高い割合でいます。残念ながらそのレイヤの説明で伝わるのは、その伝えたい情報をすでに学習している人だけでしょう。読む人が持っていない武器(知識)を使わせないように気をつけましょう。

注1) http://blockdiag.com/ja/blockdiag/

▼リスト1　バージョン情報のディレクティブの利用例

```
ミナデイン
==========

雷で敵単体に攻撃する呪文

.. versionadded:: 4

   勇者専用

.. versionchanged:: 5

   消費MPが減った

.. deprecated:: 8

   削除された
```

古い情報に気をつける

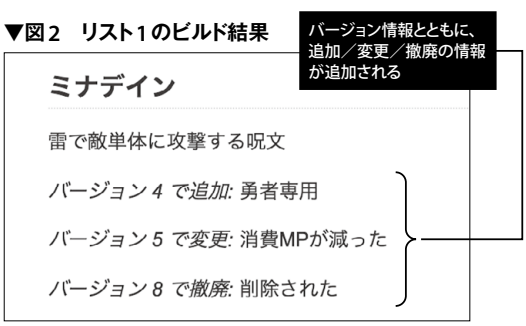
「あの洞窟に宝があります」と言われて行ってみたらすでになかった。ゲームであればイベントの伏線になるところですが、ドキュメントだと単なるストレスです。というかバグです。

ドキュメントの網羅性が低くドキュメント化されていないことはよく見る光景ですが、情報の鮮度が古くなることで「間違い」に化けることも多々あります。足りないドキュメントよりも間違ったドキュメントのほうが問題です。「最新JavaScriptフレームワーク紹介」のような文章であれば「2016年版」「ES2015向け」などと入れておくと良いでしょう。書かれた時点の日時や対象バージョンを明記することで、ドキュメントの消費期限もわかりますし、読む側にも問題解決の大切な前提条件を提供します。

Sphinxではバージョン情報を記述するディレクティブが使えます。使い方は**リスト1**のとおりです。ビルドすると**図2**のようにレンダリングされます。

チュートリアルの途中で間違いがあったら興味を持ってくれたユーザの多くはそこで諦めてしまいます。チュートリアル中のユーザはまだレベルが低く、システムの癖とかバージョンの差異を推測して問題を避ける術を知りません。

▼図2　リスト1のビルド結果



チュートリアルはくれぐれも古くならないように気をつけましょう。

また、本連載の第18回で説明したautodocを使うと最新のコードとずれのないドキュメントが書けます。

ローグライクなダンジョンデザイン

姫がさらわれる。故郷の町が破壊される……ファンタジーの物語の構造にはある程度の型があり、そのうえでディティールを作り込んでいきます。一方でローグライクと呼ばれるジャンルがあります。構成要素は部品化されダンジョンの配置だけがランダムに変わりますが、スタートとゴールなどの構造は固定されています。ダンジョン間のつながりはありません。少ない手間で長く楽しませることができるしくみです。

これと同じように、ページ間のつながりが基本的になく独立したドキュメントがあります。日記のようなコンテンツや本連載の第2回と第3回で取り上げた議事録が該当します。Q&A形式やパターン集も見かけます。議事録にはたいてい次のような項目があります。

- 議題(タイトル)
- 日時
- 参加者
- 決定事項、確認した事項
- 残課題、検討事項

Sphinxは読み物のようなドキュメント以外も作れます。ひな形となるファイルを作り、それを再利用するのが手法としては簡単です。

Q&Aやトピックごとの文章の場合は、最初はどんどん思いつくがままに書き出して、ある程度のコンテンツがそろったら分類ごとに整理したり、読みやすさを考えて順序を入れ替えたりします。それぞれのページが独立したドキュメントとなっていたら、入れ替えはtoctreeディレクティブ内での行の入れ替えだけです。システムに合わせてドキュメントをカスタマイズするときは本連載の第16回で紹介したように、includeディレクティブやcsv-tableディレクティブを使い、部品化したドキュメント片を読み手に合わせて組み合わせる方法も使えます。

今回の記事は「ゲームのメタファ」「ドキュメントに関する知識化」「Sphinxでの実現例」というパターンになっています。

世界地図と宝の地図

筆者がプレイしたRPGで、最初にゲーム内で世界地図を提供したのは確かドラゴンクエストIVでした。地図は今まで歩いてきた道のりを振り返るのに役立ちますし、これから進むべき未知の土地を知ることができます。また、宝の地図など、目的別の地図があると同じ世界が別の切り口で見えてきます。

ドキュメントの地図にあたる機能は目次と索引です。目次はドキュメントの内容を構造的に俯瞰(ふかん)するのに適しています。索引は詳しく知りたい内容があるときに、それに関連する内容がどこに書かれているかを示します。セクションのタイトルだけを見てもわからないような、内部で説明されている項目を探す手助けをします。目次は世界地図、索引は宝の地図です。

Sphinxの世界地図はtoctreeです。Sphinxではソースとなるre ST ファイルの親子関係を示すのにtoctreeディレクティブを使いましたよね。すべてのtoctreeをつなげると大きな1つの木構造になり、これがそのまま目次になるため、目次を作成する労力を別にかける必要はありません。ページ内部にミニ目次を表示する便利機能もあります。次のようにcontentsディレクティブを使うと、全体目次とは別に、ページ内部のみの目次を表示できます。

```
.. contents::
```

ドメイン機能で定義したプログラミング言語のクラスなどは索引の項目になりますが、本文

中に埋もれているキーワードを索引に登録するには明示的に指定する必要があります。索引に登録するには**リスト2**のようなindexディレクティブやロールを使用します。

インデックスは最大2階層あります。**表1**のsingle:は1階層、あるいは2階層の要素を作れます。pair:を使うと、2階層のエントリが2つ作成されます。**リスト2**では、セクションタイトルにジャンプする索引の項目ができます。

1つの項目にたくさんの索引を付けられます。本文そのものは時間軸順、難易度順などのなんらかの基準にしたがって1次元化されるでしょう。索引を付ける作業に馴染みがある人は少ないと思いますが、文章とは違う切り口で世界を表現できます。はじめは楽しくてたくさん付けたくなってしまいますが、本文の分量とバランスを取る必要があります。

サブクエスト

ちいさなメダル集めは古くからあるサブクエストとして有名ですが、近年では本編とは平行してチャレンジできるコンテンツがいくつも用意されることが多くなっています。いくつものサブクエストがつながり、裏ストーリーを形成しているものもあります。サブクエストは早くクリアしたい人には不要ですが、じっくり楽しみたい人、レベル上げをしたい人には最適です。

サブクエスト的ドキュメントの究極は、調べ物をしているうちにどんどん未知の領域に迷い込んでしまうWikipediaでしょう。それ以外にもパターン・ランゲージ的なドキュメントが該当します。これらの文章は、ドキュメントの読む順序が1方向に限定されません。ドキュメント中のキーワードから、その単語の定義や紹介のページにジャンプします。サブクエストをメインコンテンツにして、初心者向けに概略を得られるコンテンツをサブで用意しても良いでしょう。

ソフトウェアのパターン・ランゲージはWiki上で生み出されたのでWikiとの相性が良いように思われますが、Sphinxも負けていません。基本のtoctreeによる文書構造の縦糸に加えて、横糸のナビゲーションとなるクロスリファレンス機能が利用できます。Sphinxは**表2**のようなクロスリファレンスを作成する機能を豊富に持っています。ドキュメントへのリンクを作成するもの、リファレンスへのリンクを作成するものがあります。それ以外にもいくつかあります[注2]し、本連載の第17～19回で紹介したドメイン機能もこの延長線上にあります。

▼リスト2　indexディレクティブで索引を付ける

```
.. index::
   pair: イベントアイテム; 盗賊の鍵

ナジミの塔攻略
==============
```

注2）http://www.sphinx-doc.org/ja/stable/markup/inline.html

▼表1　indexディレクティブに指定できるオプション

書き方	結果
single: 薬草	薬草というインデックスができる
single: 乗り物; 船	乗り物／船というインデックスができる
pair: イベントアイテム; 盗賊の鍵	イベントアイテム／盗賊の鍵、盗賊の鍵／イベントアイテムという2つのインデックスができる

▼表2　Sphinxのクロスリファレンス機能

リンク先	リンク対象	リンク元のロール
ドキュメント	ファイル名	:doc:`ラベル <example>`
リファレンス	セクションタイトル、図表の名前	:ref:`ラベル <リファレンス>` :numref:`ラベル %s <リファレンス>`
用語	glossaryディレクティブ内の要素	:term:`ラベル <用語>`

glossaryディレクティブを使うと用語集を作成できます(**リスト3**)。用語集の項目は前述の索引にも追加されます。文書からはその用語へのリンクを張れます。用語へのクロスリファレンスをドキュメントから張るときは**リスト4**のようにロール機能を使って表現します。

Sphinxでは、**リスト5**のようにconf.pyにちょっとだけPythonコードを書くと、独自ディレクティブを簡単に作れます。(2)のパラメータはロール名です。このサンプルでは、(1)のディレクティブ名と一緒ですが、**リスト6**のように定義を作成したり、**リスト7**のように定義に対するリンクを張ったりできます。(3)のパラメータは、前の項目で紹介した索引を自動生成するテンプレートです。

まとめ

2年間の連載のまとめということで、Sphinxの機能ではなく、ドキュメントの品質を客観的に見るヒントを紹介しました。Sphinxをはじめとしたドキュメントツールにはさまざまな機能がありますが、どんなに機能を尽くしてもユーザに伝えたい情報が伝わらなければ意味がありません。ドキュメントを通じてユーザにどのような体験をしてほしいのかの指針が明確であれば、Sphinxはそれに答えて、過去の連載で紹介したようなツールを提供してくれます。

最後の最後に

さまざまなユースケースを通してSphinxの使い方を紹介してきた本連載も今回で一区切りです。本連載が始まったころ1.2.3だったSphinxのバージョンは、2年の月日を経て1.5.2になりました。Linux Kernelのドキュメンテーションツールに Sphinxが採用される[注3]など、Sphinxの利用事例も増えています。

本連載を読んでSphinxを使い始めてくれた方、興味を持ってくれた方、すでに使っている方、すべての方のドキュメントライフが幸せでありますように！SD

注3) https://www.kernel.org/doc/html/latest/index.html

▼リスト3　glossaryディレクティブで用語集を作成

```
.. glossary::

   HP
      ヒットポイント。ゼロになると死ぬ。宿屋や回復魔法で回復できる。

   MP
      マジックパワー。呪文を唱えると減っていく。宿屋で回復できる。

   EX(P)
      経験値。敵を倒すと増える。一定数貯まるとレベルが上がる。

   E
      装備している武器や防具に表示される記号。
```

▼リスト4　用語集へのリンク

```
アイテムを装備すると、道具一覧に :term:`E` の記号がつきます。
```

用語集の「E」の項目へのリンクになる

▼リスト5　独自ディレクティブ／ロールの定義

```
def setup(app):
    app.add_object_type('magic', 'magic',
                        objname=u'呪文',
                        indextemplate=u'pair: %s; 呪文')
```

(1) ディレクティブ名を指定
(2) ロール名を指定
(3) 索引を自動生成するための指定

▼リスト6　独自ディレクティブを使用

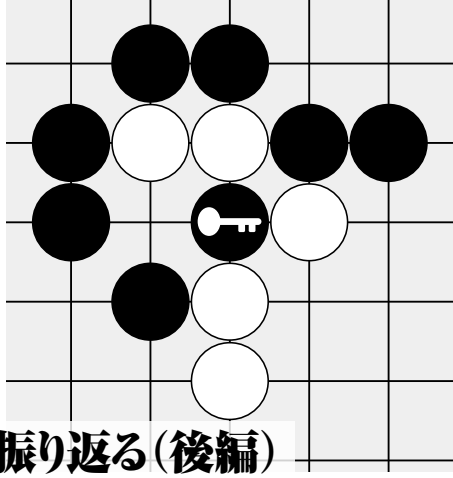
```
.. magic:: メガンテ

   自爆して敵にダメージをあたえます。
```

▼リスト7　独自ロールで定義にリンクを張る

```
作戦を「いろいろやろうぜ」にすると、キャラによっては :magic:`メガンテ` を唱えるので要注意。
```

セキュリティ実践の基本定石

すずきひろのぶ
suzuki.hironobu@gmail.com



みんなでもう一度見つめなおそう

【第四一回】2016年のセキュリティの状況を振り返る（後編）

前後編にわけて2016年のセキュリティの状況を振り返っています。前回は1～6月に起こった事件などを紹介しました。今回は7～12月の事件を振り返ります。

7月

佐賀県の教育ネットワークから個人情報流出

佐賀県が運用している教育ネットワークSEI-NETや、学校で運用している校務用サーバ、学習用サーバなどが広範囲に渡って侵入され、そこにあった個人情報を含むファイルが大量に盗まれたという事件です。17才少年と高校生7名が逮捕・補導されました。この問題を振り返ると、過去から現在に至るまでの情報セキュリティの諸問題がすべて含まれており、非常に古典的であり、また教訓的な事件に思えます。

この事件の舞台となったネットワークは大きく3つあります。

(1) 佐賀県内の学校をつなぐ教育ネットワークSEI-NET
(2) 校内のみで運用されている校務用サーバが置いてあるネットワーク
(3) 校内で使われている教育用サーバが置いてあるネットワーク

(2)のネットワークは、簡単に言えば職員室のネットワークです。高校の先生は家庭の状況や進学相談の状況など極めて個人的なことも把握しているのが仕事ですし、それを記録するのも仕事です。また個人の学校の成績や進路なども記録として残っています。これらの情報は個人情報の中でも極めてセンシティブな情報の1つであると言えます。

今回の事例に関しては、あちらこちらのWebメディアやブログにまとめが載っています。それを繰り返すのも無駄なので、筆者の視点からこの事件をまとめてみます。

なお、この事件を調べるにあたり地元である佐賀新聞のサイトにある一連の記事がとても役に立ちました。この事件の詳細を知りたくなった際は、まず佐賀新聞の記事を参考にすることをお勧めします。

■欺術（ぎじゅつ）を使う高校生

新聞では校務用サーバと書かれていますが、記事を読んでいくと、これは特別な管理をしているサーバでも何でもなく、先生たちがファイル共有に使っているファイルサーバ以上のものではないようです。同一のLANに入れば、そのままファイルをシェアできるか、簡単なパスワードでアクセスできるようで、事務系のLANではごく普通のものでしょう。

校務用LANに入れば容易にファイルにアクセスできるようですが、それでもLAN側にアクセスするためにはパスワードが必要です。

犯人の少年たちは、そのパスワードを古典的とも言える欺術、つまり人をだまして情報を取るという形で入手しています。記事に載っていた方法では、少年たちはタブレット端末上に、自分たちが作った偽の入力画面を表示して、操作がわからなくなったふりをして先生に接続の際のIDとパスワードを入

力させ、その情報を盗もうとしていました。

実際に高校生は少なくとも2014年あたりには、すでに成功してパスワードを入手していました。高校の先生に国家安全保障や軍事のレベルのセキュリティ知識を求めるわけではありませんが、生活の中で言えば「横断歩道を渡るときは左右を確認する」くらいのレベルの情報セキュリティは学んでほしいと感じます。

欺術で必要な情報を入手し、夜の校舎にそっと入り込み、校内のネットワークに接続して情報を盗む。1983年に公開された映画『ウォー・ゲーム』[注1]ですでにモチーフにされている古典的な手法です。やっていた少年らは、映画やマンガのスリルを味わっていただけなのかもしれませんが、親の学歴や年収まで書かれているファイルを盗まれる側の立場に立てばたまったものではありません。

■入札条件としての問題点

「学内のLANに接続しなければ大丈夫」。学校や県教育委員会はそれで十分だと考えていました。コンピュータやネットワークの知識がないと、そうなってしまうのも致し方がないとはいえ、その感覚でシステムが構築され運用されていれば、ターゲットとなった瞬間に耐えられないでしょう。

別に、セキュリティの専門家レベルの知識は必要ありません。ごく普通のネットワーク技術者が常識的にもっている範囲の知識で十分です。運用に関してもしかりです。

しかし、佐賀県教育委員会がシステムの入札条件を書く際に、そのようなセキュリティのことを考えた運用まで含めていたでしょうか。佐賀新聞の一連の記事を読む限り、考慮されている形跡はありません。どんなにエンジニア側がセキュリティに関して忠告を与えようとも、入札条件や発注仕様に入っていなければ、エンジニア側からできることはほとんどありません。

記者の質問に答えて、学校側は「システムは善意を前提として作られている」と答えていました。これは違います。システムの一面しか知らない素人の発想で発注しただけです。例えて言うなら、次のようなことをしているのです。

クルマの免許ももたず、運転もしたことのない人間が、知識もないのに自動車を選び、購入し、そして乗り回している

これでは事故を起こさないほうが不思議です。この件をきっかけに文部科学省が通達を出しました。しかし、「素人が入札仕様を作り、発注をかける」という本質的な問題が解決されない限り、情報システムのセキュリティに関連する似たような問題が繰り返されるでしょう。

■無視される警告と存在しなかったセキュリティ体制

佐賀新聞が2016年6月30日に「県教委、昨年6月に不正アクセス把握も通報せず　佐賀県教育情報システム不正接続事件」と報じています。

この事件が発覚する1年前の2015年6月14日、佐賀市の県立高校の教師がネットワーク内のファイルが開けなくなるトラブルがあり、保守業者が調査したところ、不正アクセスによりアクセス権限が変えられていることがわかりました。このときはパスワードなどを変更し、保守業者から校長への報告、県教委の教育情報課に事件の概要を報告しました。

しかし、県のほうからは警察に不正アクセスの通報をせず、また県の中でも校内LANを所管する教育総務課にも連絡せず、県教委内部で対応を協議することもなかったと新聞は報じています。

実際に問題が発生しているにもかかわらず、何が問題になっているのかが理解できないまま過小評価を行い、何も行動を起こしていません。「今までも何でもなかったのだから、これからも問題ない」と考える典型的な正常化バイアスです。担当者が情報セキュリティの基本的知識に欠けるという部分はあるでしょう。しかし本質的には、個人に頼ることなく、組織として情報セキュリティに対応する体制が

注1）ジョン・バダム監督、マシュー・ブロデリック主演のSF映画。主人公の高校生クラッカーが、いたずらで北アメリカ航空宇宙防衛司令部のコンピュータに不正アクセスしたことをきっかけに、実際に米国とソ連が戦争の危機に直面していくという物語。

存在していなかったというのが大きな問題です。

この事件を受けて、県教育委員会の中にCSIRTチームを構築するようです。しかし、これも同様な縦割りの問題があります。やるべきは県教委という形ではなく、県庁全体をカバーするCSIRTチームをまず作り、そのブランチとして県教委のCSIRTチームを存在させることです。県の行政の中の1セクションだけしかカバーせず、縦のつながり、横のつながりの欠いたCSIRTでは効率的ではなく、コストに対する十分な動きを期待できないでしょう。

8月末～9月初旬

広範囲へのDDoS攻撃

2016年8月末から9月初旬にかけて、日本国内の複数のサイトへDDoS攻撃が行われました。この中には技術評論社のサイトも含まれています。早いものは2016年8月22日あたりから始まり、9月3日ぐらいまで複数のサイトがDDoS攻撃の対象となり、アクセスができない状態になりました。

- DoS攻撃による断続的な接続障害についてのお詫び（技術評論社、2016年9月2日）
  http://gihyo.jp/news/info/2016/09/0901

- ヨドバシカメラ、ならびにグループ会社全てのインターネットサービスがつながりにくくなっていました（ヨドバシカメラ、2016年9月5日）
  http://www.yodobashi.com/ec/support/news/1609051211/

- 外部からのDoSトラフィックによるネームサーバ障害（さくらインターネット、2016年8月29日）
  http://support.sakura.ad.jp/mainte/mainteentry.php?id=20072

- DDoS攻撃と見られる大量のトラフィックによりスラドを含む国内の複数サイトがダウン（スラド、2016年9月5日）
  https://security.srad.jp/story/16/09/05/105231/

- 再度サーバを引っ越しました（家Tサーバー、2016年8月24日）
  http://ie-t.net/information/再度サーバを引っ越しました

上記のスラドのサイトに推移が詳しく書かれていますが、いずれもDNS amp攻撃のようです。DNS amp攻撃はオープンリゾルバ[注2]の問題と連動しており、この問題に対して最初に注意喚起が出てきたのは2006年[注3]ですから、もう10年越しの問題です。

DNS amp攻撃

DNS amp攻撃は、攻撃側がオープンリゾルバを経由してデータ量を増幅させ、被害者側に大量のデータを浴びせる手法です（**図1**）。攻撃側が出したデータの増幅量は、理論上では最大約49倍ですが、過去の実験結果を見ると実測で約40倍に増幅したという報告[注4]があります。

DNSサーバが麻痺したために、そこを使っていたサイトが巻き込まれて被害数が膨れたのは確かですが、DNSが独立していたサーバに直接攻撃が来ていたと思われるサイトも多くありました。しかも、そのサイトに関連性が見られないという、いまひとつ攻撃した背景がよくわからないものでした。

先ほどのスラドの報告によれば、約4.7万のユニークIPアドレスからDNSのクエリ・レスポンスがあったとのことです。正当なクエリ・レスポンスがある程度あったことを差し引いてもかなりの広範囲からの攻撃です。しかも、IPアドレスを変えるなどの対応をしても、それを追いかけてくるという執拗（しつよう）

注2）リゾルバ（resolver）とは、ホスト名からIPアドレスを引いたり、IPアドレスからホスト名を引いたりして名前解決などをする機能のこと。オープンリゾルバとは、どのクライアントからの名前解決などの問い合わせにも答えるようになっているDNSサーバのことを指す。
注3）DNSの再帰的な問合せを使ったDDoS攻撃に関する注意喚起 http://www.jpcert.or.jp/at/2006/at060004.txt
注4）DNSの再帰的な問い合わせを悪用したDDoS攻撃手法の検証について
https://www.npa.go.jp/cyberpolice/server/rd_env/pdf/20060711_DNS-DDoS.pdf

な攻撃をしています。

DNS amp攻撃にも複数の手法がありますが、相手にダメージをより多く与えるのは、DNSキャッシュへデータの「仕込み」をして、さらに仕込んだデータをターゲットに送りつける「攻撃」の2段階方式です。つまり、相手のIPアドレスが変わったことをチェックして、変わったらまた「仕込み」をして再度攻撃を行うということをしています。

技術評論社サイトも執拗な攻撃が行われており、再三に渡ってサービスを中断せざるを得ませんでした。

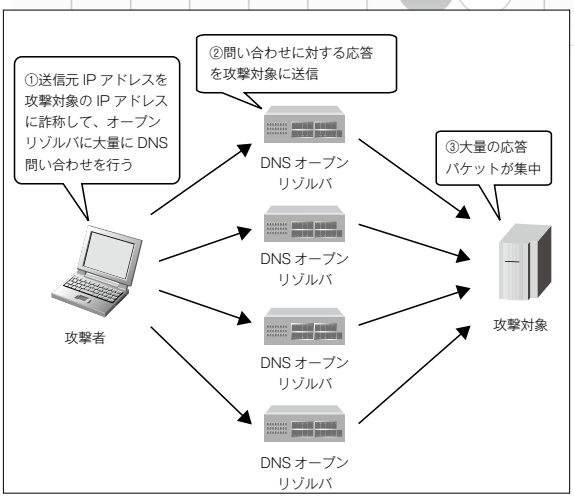
DDoS攻撃が生んだ波紋

DDoS攻撃は被害者の意向など関係なく、狙われてしまえば一方的に攻撃を受けます。ティーンエージャーの愉快犯であれ、経済的な損害を与えることを目的とした犯罪者であれ、やっていることや、その結果には違いはありません。このようなケースに対処するためには、ISP（Internet Service Provider）やデータセンターと協力しあいながら問題を解決しなければいけません。

ところが今回、DDoS攻撃の対象になったサイトが、契約していたサーバサービスを解除されてしまうということが発生しました。DDoS攻撃を受けていると、その会社のほかのサーバサービスに支障をきたすと判断し、DDoS攻撃のターゲットとなっている者との契約をサーバ会社側の判断で一方的に解除するというものです。これは何を意味するかというと、DDoS攻撃者は、労することなくそのサイトの息の根を止めることができる、ということです。

これによりDDoS攻撃の被害者は、さらに二次被害とも言える影響を受けてしまうのです。そもそも今回のケースでは、攻撃先の選択に関連性が見られません。たとえ攻撃の理由があったとしても、サイト側は威力業務妨害を受けた被害者です。その被害者が、このような理由で、サーバサービス会社側から一方的に契約が解除されるのは理不尽としか言えません。このサーバ会社は攻撃者にいいように使われています。

◆図1　DNS amp攻撃のイメージ



後日談ですが、そのサイトは別のサーバサービス会社と契約をしましたが、執拗な攻撃は続きました。しかし、新しく契約した会社は、攻撃中は通信を遮断するフィルタリングなどの処置を行い、契約を解除するようなことはしなかったようです。

9月末～

インターネット史上最大規模のDDoS攻撃

2016年後期で残った大きな話題は9月末から始まる一連のMiraiの話になります。しかし、これはすでに本連載の第38、39回[注5]で詳しく取り上げたので、そちらをご参照ください。

◆　◆　◆

2016年を振り返るとネットワークセキュリティの大きな問題が次々に発生しています。すでにお気づきでしょうが、規模は大きくなっていても、その根本的な原因の多くは昔から存在するものです。そして、そこから新しい問題を引き起こしています。やはり、セキュリティの基本定石を知ってこそ、そこから次々に起こる問題への応用ができるのではないか、と筆者は強く思うのでした。SD

注5）本誌2016年12月号、2017年1月号「IoT機器を使った過去最大規模のDDoS攻撃（前編・後編）」

SOURCES

レッドハット系ソフトウェア最新解説

第7回

Container Native Storage for OpenShift

DockerやOpenShiftなどのコンテナを使う環境において、アプリケーションと同様にコンテナ化されたストレージの概要や使い方について紹介します。

Author 小島 啓史(こじまひろふみ)
mail：hkojima@redhat.com
レッドハット㈱ テクニカルセールス本部 ソリューションアーキテクト

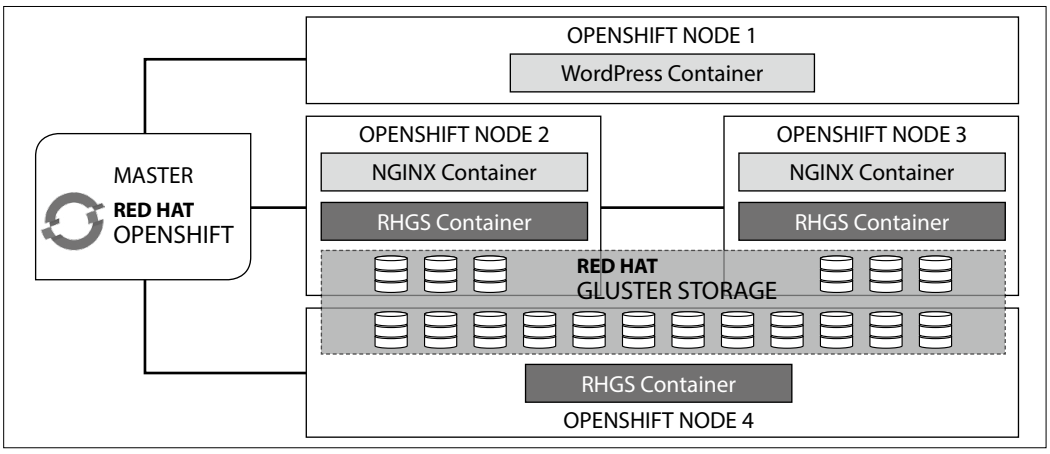
ストレージのコンテナ化

Docker、Kubernetes、OpenShiftなどのコンテナを使う環境の導入が進むにつれて、アプリケーションの状態などを保存するためのストレージが必要になってきます。このストレージには、NFSやクラウドストレージなどを活用できますが、最近ではストレージそのものもコンテナ化して、アプリケーションと同じホスト上で並列に動作させるというコンバージドなソリューションも出てきました。これにより、アプリケーション／コンテナ間ネットワーク／ストレージのライフサイクルを、Kubernetesなどの単一のコンテナオーケストレーションツールで管理できるようになります。

こうしたコンバージドなソリューションの1つとして、OpenShiftとSoftware-defined Storage(SDS)の Red Hat Gluster Storage(RHGS；Facebookで利用[注1]されているオープンソースのGlusterFS[注2]をベースとした、レッドハット提供のSDS製品)を組み合わせた構成があります。**図1**がその構成イメージです。OpenShift環

注1) URL http://blog.gluster.org/2016/01/scaling-glusterfs-facebook/
注2) URL https://www.gluster.org/

▼図1 OpenShift環境でのRHGS構成イメージ



▼図2　ポートの開放

```
RHGSコンテナを実行する全Nodeの/etc/sysconfig/iptablesに下記を追加
-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport 24007 -j ACCEPT
-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport 24008 -j ACCEPT
-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport 2222 -j ACCEPT
-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m multiport --dports 49152:49664 -j ACCEPT
# systemctl reload iptables
```

▼図3　RHGSコンテナ用のデプロイツールのインストール

```
# subscription-manager repos --enable=rh-gluster-3-for-rhel-7-server-rpms
# yum -y install cns-deploy heketi-client
```

▼図4　プロジェクトの作成と権限の追加

```
# oc login -u system:admin
# oc new-project storage-project; oc project storage-project
# oc adm policy add-scc-to-user privileged -z default
```

境の各Nodeでアプリケーションとストレージをコンテナとして動作させており、ストレージの冗長化やクラスタリングなどをRHGSの機能で実現しています。また、RHGSはKubernetesやOpenShiftとの連携強化を進めており、最新版のOpenShift(執筆時点ではv3.4)を利用すると、開発者がボリュームをオンデマンドに作成・利用することができるようになっています。

RHGSのセットアップ

RHGSとOpenShiftを組み合わせて利用する場合の、レッドハットサポート要件[注3]を満たすことを前提としてセットアップ作業を実施してみます。まず、RHGS(評価版を利用する場合は、当社Web[注4]からお問い合わせください)コンテナを実行することを想定した全Nodeの、GlusterFSサービスに関するTCPポートを開けて、iptablesサービスをリロードします(**図2**)。

デプロイツールのインストール

次にOpenShift環境の任意のMasterサーバで、RHGSサブスクリプションの利用を有効化し、RHGSコンテナ用のデプロイツールをインストールします(**図3**)。

図3でインストールするパッケージ名に含まれるHeketi[注5]は、GlusterFSボリュームのライフサイクルを管理するためのRESTfulなインターフェースを提供するオープンソースです。Hekетiを利用することで、Kubernetes、OpenShift、OpenStack Manilaのようなクラウドサービスの利用者が、GlusterFSボリュームを動的に作成・利用することが可能になります。cns-deployパッケージには、OpenShift環境でHekетiやGlusterFSを利用するためのテンプレート一式が含まれています。

プロジェクトの作成

続いてRHGSコンテナを実行するプロジェクトを作成し、作成したプロジェクト内で特権コンテナを実行するための権限を追加します(**図4**)。RHGSコンテナ内では、GlusterFSサービス関連のプロセスをroot権限で実行するように設定されているため、この権限追加が必要となります。

注3) URL http://red.ht/2klU9Uh
注4) URL https://engage.redhat.com/content/contact-japan-sales

注5) URL https://github.com/heketi/heketi

▼リスト1　GlusterFSクラスタのトポロジー例

```
### /usr/share/heketi/topology-sample.json を基に作成 ###
 ……(中略)……
{
  "node": {
    "hostnames": {
      "manage": [
        "node1.example.com"
      ],
      "storage": [
        "192.168.199.111"
      ]
    },
    "zone": 1
  },
  "devices": [
    "/dev/sdc"
  ]
},
 ……(中略)……
```

そして、GlusterFSクラスタのトポロジーを記載したjsonファイルを作成します(**リスト1**)。

/usr/share/heketi/topology-sample.jsonがサンプルファイルとして用意されていますので、このファイルを基に作成できます。このjsonファイルには、ホスト名、IPアドレス、ゾーン(ホストの所在地)、デバイス情報を記載します。なお、HeketiでGlusterFSボリュームを作成する場合のレプリカ数のデフォルト値は3なので、3台以上のノードとデバイスを記載しないと、後述のcns-deployコマンド実行時にエラーが発生します。ご注意ください。

これでRHGSコンテナをOpenShift環境にデプロイするための準備が整いました。cns-deployコマンドを利用して、RHGSコンテナをstorage-projectプロジェクト内にデプロイします(**図5**)。トポロジを記載したjsonファイルを指定することで、GlusterFSクラスタの作成から、作成したクラスタ内でボリュームを動的に作成・利用するためのHeketiサービスを実行するコンテナの起動までを自動的に実行してくれます。なお、作成されたGlusterFSクラスタの情報を確認するには、「oc get route」コマンドで確認できるルート情報を利用して、「heketi-cli topology info」コマンドを実行します。すると、クラスタに参加しているホストやデバイス、クラスタ内で作成されているボリュームの情報などを確認できます。

StorageClassの作成

最後に開発者が動的にGlusterFSのボリュームを作成する際に利用する、StorageClass[注6]を

注6) URL https://kubernetes.io/docs/user-guide/persistent-volumes/#storageclasses

▼図5　RHGSコンテナのデプロイとデプロイ結果の確認

```
# cns-deploy -n storage-project -g topology.json
 ……(中略)……
heketi is now running.
# oc get route -n storage-project
NAME     HOST/PORT                                  SERVICES
heketi   heketi-storage-project.cloudapps.com   heketi
# export HEKETI_CLI_SERVER=http://heketi-storage-project.cloudapps.com
# heketi-cli topology info
```

▼図6　StorageClassの作成

```
# oc login -u system:admin
# cat <<EOF > glusterfs-storageclass.yaml
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: gluster-storageclass
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://heketi-storage-project.cloudapps.com"
  restuser: "admin"
EOF
# oc create -f glusterfs-storageclass.yaml
```

▼図7　StorageClassを利用したPV/PVCの作成

```
# oc login -u testuser01; oc new-project test01
# oc get storageclass
NAME                TYPE
gluster-storageclass   kubernetes.io/glusterfs
# echo <<EOF > glusterfs-pvc1.yaml
 …… (中略) ……
metadata:
  name: pvc01
  annotations:
    volume.beta.kubernetes.io/storage-class: "gluster-storageclass"
 …… (中略) ……
EOF
# oc create -f glusterfs-pvc1.yaml
```

作成します(**図6**)。StorageClassはKubernetes v1.4から導入された機能であり、開発者による動的なPersistent Volume(PV)の作成を可能にしてくれるものです。StorageClassの作成はOpenShift環境の管理権限がないと実施できませんが、一度作成したStorageClassは開発者が自由に利用できるようになります。なお、StorageClassをあらかじめ複数作っておき、速いストレージ環境は「Gold」、遅いストレージ環境は「Bronze」などクラス分けをしておくことで、開発者が使い分けることもできます。

これでOpenShift環境を利用する開発者が、動的にボリュームを作成して利用する準備が整いましたので、OpenShift環境であらかじめ設定しておいたローカルユーザを利用して、PV/Persistent Volume Claim(PVC)を作成してみます。この場合、PVCを作成するためのYAMLファイルを用意して「`oc create`」コマンドで読み込ませるだけで、PVCに必要なPVも自動的に作成してくれます(**図7**)。このYAMLファイルでは、annotationによるStorageClass(この例ではgluster-storageclass)の指定が必要になります。作成されたPV/PVCやGlusterFSボリュームは、「`oc get`」や「`heketi-clitopology info`」コマンドで確認できます。また、このPVCを指定してアプリケーションにPVを接続し、データを永続化できるようになります。

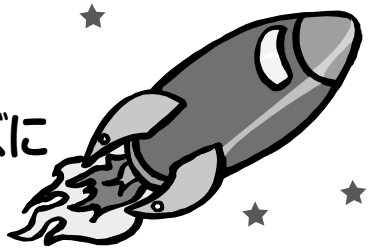
まとめ

今回はOpenShift環境でのGlusterFSボリュームの動的な作成・利用方法を紹介しました。OpenShiftとGlusterFSの統合が進み、コンバージドな環境を用意することが比較的容易になってきているので、ぜひとも利用してみてください。SD

44

Debian Developer　やまねひでき　henrich@debian.org

Debian 9の開発は最終フェーズに



Debian Hot Topics

年末年始休暇の関係で、Debianの開発周りは普段に比べてとても静かです(＝あまり取り上げるネタがないため、今回は縮小版でお送りします)。

Debian 8.7リリース

1月14日に、Debian 8のポイントリリースである8.7がリリースされました[注1]。特筆すべきことはなく、通常のセキュリティおよび重大な問題の修正が含まれています。

ポイントリリースはおおよそ3ヵ月に一度行われています。そのため、次の8.8リリースは通常ですと4月あたりになる予定ですが、Debian 9がリリースされる時期によっては若干前後してリリースされる可能性があります。リリース日程の調整はdebian-releaseメーリングリスト[注2]で行われていますので、そちらを注視してください。

Debian 9リリース作業は最終フェーズに

2月5日に、Debian 9開発が「full freeze」フェーズに入り、リリースのためのバグ修正作業も本格的になっています。「full freeze」期間は、testingに対しては次の対応となります。

- 基本的に「RC(Release Critical)バグ」の修正のみ対応
- すべての変更についてリリースチームの許可が必要
- パッケージの更新を行いたい場合は、リリースチームが手動でtestingへの移行のブロックを解く(unblockを実施する)ので、トラッキングのためDebian BTSでrelease.debian.org擬似パッケージ[注3]に対して登録をすること[注4]
- 変更点についてはdiffを添付し、レビューを受けること

このあと、「RCバグを0」にした時点でDebian 9はリリースとなります[注5]。「full freeze」に入った時点で、Debian 9はリリースにかなり近い形になっていますので、試してみたい方は、現時点で開発版のdebian-installer[注6]を使ってtestingのインストールを行ってみてください。

なお、本誌の2017年1月号で「merged-/usr」として取り上げていた変更は、残念ながらDebian 9リリース時にはすべて修正できる見込みが立たないためいったん撤回し、次のリリースで対応することになりました。

注1) URL https://www.debian.org/News/2017/20170114
注2) URL https://lists.debian.org/debian-release/

注3) Debian BTSはすべて「パッケージ」に対してのバグを取り扱うので、パッケージではないものを扱うときは「擬似パッケージ」として登録しています。
注4) リリースチームが処理対象をもれなく把握するため、reportbugコマンドを使って適切なオプションを設定することを推奨します。
注5) ほかのディストリビューションと異なり、Debianのリリース日は事前に定められていません。大きな問題を修正(あるいは重要度を下げるなど)して、RCバグを0にしてからリリースとなります。
注6) URL https://www.debian.org/devel/debian-installer/

piuparts autoreject

Debianのインフラには、パッケージがパッケージングポリシー[注7]に対して重大な違反をしている場合、このパッケージがリポジトリへアップロードされるのを拒否する「lintian autoreject」が以前から実装されていました。

そこに今度は「piuparts autoreject」が実装されました[注8]。「piuparts」はパッケージのインストール／アンインストール／アップグレードに問題がないかどうかをチェックするツールです。「piuparts.debian.org」のサーバで自動的にpiupartsを実行しているのですが、パッケージをunstableからtestingへ移行する際（testing migration）に、そのログを参照してエラーがあれば「重大な問題あり」と見なしてtestingへの移行をブロックするようになりました。

lintianもpiupartsも非常に有用なツールですが、その利用度合いは開発者によってマチマチなのが実情です。インフラ側でも利用することによって利用に漏れがなくなり、パッケージの品質をより高められるようになるでしょう。

さらに、Paul Geversさんは「Let autopkgtests be gating for testing migration in Buster: heads-up and brain-dump」というメール[注9]で、「パッケージの自動テストツール『autopkgtests』の結果を、同様にtestingへの移行のブロックに使えるようにしないか」という提案をしています。これは次のリリースサイクルでの導入になりますが、リグレッションの検出などを人手に頼らずできるようになり、フリーズ期間の短縮が見込めます。

15年ぶりの更新とその余波

net-toolsパッケージが更新され、「15年ぶりに開発を（forkして）再開したよ」という“一見”喜ばしいメッセージが寄せられました。しかし、続けて「コマンドの出力が変わっているから、これをパースして利用しているスクリプトを壊すかもしれないのでチェックしてね」との記述。net-toolsに含まれるコマンドには、netstat、ifconfig、ipmaddr、iptunnel、mii-tool、nameif、plipconfig、rarp、route、slattach、arpがありますが、これらの基本的なコマンドを使っているスクリプトとパッケージは多数存在しており、その動作が壊れる可能性があるとなると喜んでもいられません。

この変更を受けて、開発者メーリングリストでは「Can we kill net-tools, please?」と物騒なタイトルのスレッドが立ちました。内容としては、「net-toolsに含まれるコマンドの代替（iproute2など）がこの15年の停滞の間に出てきている。net-toolsとiproute2の両方をデフォルトでインストールするのは意味がないから、net-toolsは落とすようにしよう」というもので、妥当な主張に思えます。

これを受けてnet-toolsパッケージのPriorityはリポジトリサーバ側でimportantからoptionalに落とすよう上書きされました。また、多数のnet-toolsに依存するパッケージについても、依存関係を落としていくように変更が加えられ始めています。

筆者としては、「ギリギリのタイミングで地味に影響の大きそうな問題をぶち込んできたなぁ」という印象です。ほかのDebianパッケージには影響がなくとも、上記のコマンドを利用した自家製スクリプトが動かなくなったり間違った動作をするようになったり……ということがあるかもしれませんので、お手すきのときにtestingを利用して動作のチェックなどをしていただくのが良いでしょう。SD

注7）URL https://www.debian.org/doc/debian-policy/
このポリシーに従っているかをチェックするツールとして「lintian」があります。

注8）URL http://deb.li/3fKVI
筆者がいくつかパッケージのアップグレードエラーに遭遇して辟易し、メールにて「実装できないかな？」と質問したところ（URL http://deb.li/FT7I を参照）、Niels Thykierさんが「Todoリストにあるよ」とコメントし、サクッと実装してもらえました。何でも言ってみるものですね。

注9）URL http://deb.li/i4dge

LibreOffice 5.3の新機能

Ubuntu Japanese Team　あわしろいくや

今回は、2017年1月初頭にリリースされたLibreOffice 5.3の新機能や変更点について解説します。

LibreOffice 5.3概要

LibreOfficeは年に2回メジャーバージョンアップ版がリリースされますが、伝統的に年始めのリリースは大きな変更点が入ります。理由としてはGoogle Summer of Codeでの成果が反映されることが挙げられます。

LibreOfficeはマルチプラットフォームなアプリケーションで、プラットフォームごとの違いは可能な限り吸収されているのですが、例外もあります。とくにテキストレイアウトに関してはプラットフォームごとに大きな違いがありました。すなわちWindowsとmacOSとその他Unix系でプラットフォームごとのレイアウトエンジンに依存していたのです。

ということは、同じドキュメントを読み込んでも表示の違いがあり、またプラットフォーム特有のバグがあったということです。しかし、この5.3からは統一されました。同じフォントをインストールしたうえで同じドキュメントを読み込めば、同じレイアウトで表示されるようになったのです。もちろんプラットフォームごとにあったレイアウト関連のバグが消滅したということでもあります。もちろん新たなバグが生まれるということでもあるのですが、メンテナンスが容易になったことにより迅速に対応されることが期待できます。

昨年暮れにLibreOfficeの支持母体であるThe Document FoundationがMUFFINというコンセプトを発表しました。“My User Friendly & Flexible INterface”の略で、すなわちユーザの熟練度や使用環境に応じて適切なUIは異なるので、オプションで変更できるように複数のUIを提供しますということです。わかりやすく「ついにLibreOfficeでもリボンUIが!?」と受け止められたようですが、そんなにシンプルな話ではありません。そればかりか、この機能は5.3の段階では「実験的な機能」として提供されており、気軽に使えるようにすらなっていません。

ほかにも変更点は多数あるので、個別に見ていきましょう[注1]。

全般

テキストレイアウトエンジン

前述のとおり、テキストレイアウトエンジンがすべてのプラットフォームで統一されました。これにより、同じフォントがインストールされている限り、同じドキュメントならば別のプラットフォームで読み込んでも同一のレイアウトで表示されるようになりました。制限事項としては、既存のドキュメントではレイアウトの非互換（ズレ）が発生する可能性があります。また、この変更の一環として一部のフォントが使用できなくなりました。

注1）スクリーンショットは5.3.0.2（RC2）相当であり、リリース版とは異なります。

テキストレイアウトエンジンはドキュメントだけではなく、LibreOfficeの描画自体にも影響しています。

少なくとも5.3では、環境変数“SAL_NO_COMMON_LAYOUT”に何かを指定すると、以前のレイアウトエンジンのまま使用できます[注2]。どちらを使用しているか知りたい場合は、[ヘルプ]-[LibreOfficeについて]を表示し、[Layout Engine:]が何になっているかで確認できます。いずれ以前のレイアウトエンジンは削除されることになっているため、早めに対策するのがいいでしょう。

図1と**図2**を見比べてみてください。**図1**が新しいほうですが、**図2**と見比べてみると文字間が開いていて読みやすくなっています。

カラーパレット

カラーパレットも大きな変更がありました[注3]。[最近使用した色]が正しく動作するようになりました。さらにここからカスタムパレットが作成できるようになりました。これまでは[ツール]-[オプション]-[LibreOffice]-[色]でその役割を担っていましたが、この機能はなくなりました。さらにパレット(テンプレート)も増減や変更があります。

注2) ソースコードではこの環境変数がnullかどうかをチェックしていますで、“SAL_NO_COMMON_LAYOUT=yes”でも“SAL_NO_COMMON_LAYOUT=1”でもいいです。

注3) スクリーンショットを掲載しようと思いましたが、そういえばこの連載はモノクロですので違いがわからないので断念しました。

図1　[ヘルプ]-[LibreOffice]を新しいテキストレイアウトエンジンで開いたところ

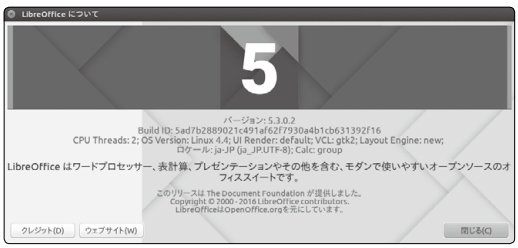
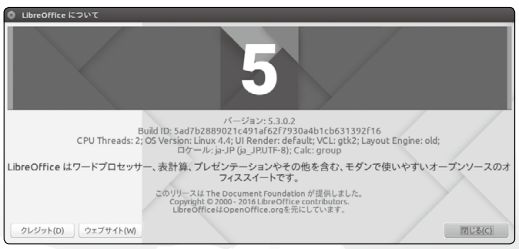


図2　[ヘルプ]-[LibreOffice]を古いテキストレイアウトエンジンで開いたところ



セーフモード

新たにセーフモードが追加されました。強制終了後や[ヘルプ]-[セーフモードで再起動]など、いくつかの方法でセーフモードにできます。ユーザ設定のリストア、拡張機能やハードウェアアクセラレーションの無効化、拡張機能のアンインストール、ユーザ設定の初期化などが行えます。

GUI

コンテキストメニュー(右クリックメニュー)にショートカットが表示されるようになりました。不要な場合は[ツール]-[オプション]-[LibreOffice]-[表示]-[コンテキストメニューのショートカット]を[表示しない]にすれば消すことができます。

領域／背景タブが刷新されました。これはスクリーンショットを見ていただくのが早いと思います(**図3**)。

サイドバーにも大きな変更があります。たとえばWriterではページの書式を設定できるようになりました。

線と矢

WriterとCalcでは図形描画ツールバーに[線と矢]が追加されました。これまではDrawとImpressにしかありませんでした。

図3　新しい背景タブ。右が見切れているのは開発版ゆえのご愛嬌

実験的な機能

ここで紹介する機能は、[ツール]-[オプション]-[LibreOffice]-[詳細]の[実験的な機能を有効にする]にチェックを入れ(**図4**)、再起動すると使えるようになります。

図4　[実験的な機能を有効にする]にチェックを入れる

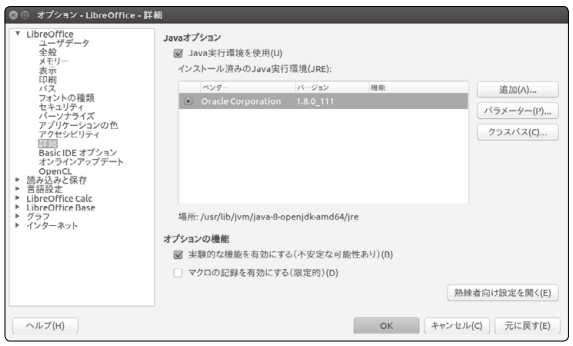


図5　Writerでノートブックバーを表示したところ

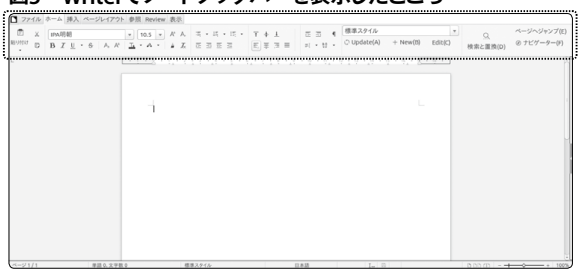


図6　Calcでノートブックバーを表示したところ

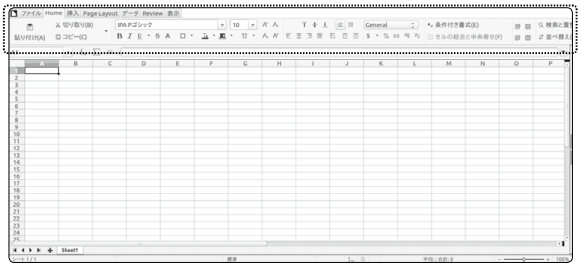
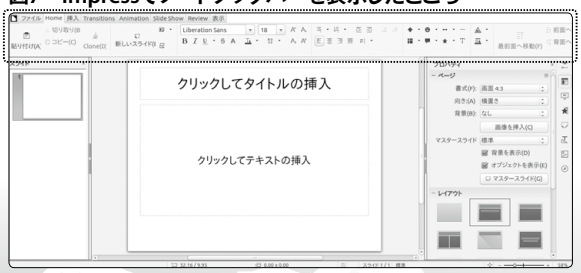


図7　Impressでノートブックバーを表示したところ



ツールバーの変更

[表示]-[Toolbar layout(ツールバーレイアウト)]でツールバーのレイアウトが変更できるようになりました。WriterとCalcでは[Default(既定)]、[Single toolbar(シングルツールバー)]、[サイドバー]、[Notebook bar(ノートブックバー)]の4つ、Impressでは[Default(既定)]、[Single toolbar(シングルツールバー)]、[Notebook bar(ノートブックバー)]の3つから選択できます。[ノートブックバー]を選択した場合、[表示]-[ノートブックバー]からWriterでは3つ、CalcとImpressでは2つからオプションが選択できます。

このうち[ノートブックバー]がリボンUIに似ていて一番注目されていますが(**図5〜7**)、[シングルツールバー]や[サイドバー]も活躍の機会はあるでしょう。とくにサイドバーは前述のとおり5.3でも強化されているので、いずれはここだけで作業が完結するようになることでしょう。

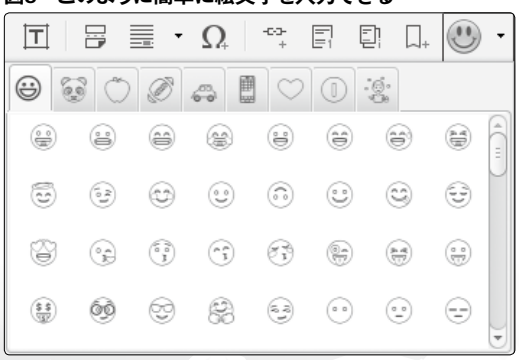
サイドバーの変更

Writerでは[変更の管理]がサイドバーから行えるようになりました。

その他

絵文字が入力できるようになりました(**図8**)。一番簡単な方法は、[ツール]-[カスタマイズ]を開き、[ツールバー]タブで使用しているツールバー(既定では「標準」)を選択し、[emoji(絵文字)]にチェックを入れることです。

図8　このように簡単に絵文字を入力できる



Writer

ページへジャンプ

［編集］-［ページへジャンプ］はこれまでナビゲーターを表示していましたが、5.3からは専用のダイアログが表示され、より簡単にページの移動ができるようになりました。

テーブルスタイル

これまで表のデザインを変更する場合は、［表］-［オートフォーマットのスタイル］で行っていました。ただし一度しか変更できないなど、制限が多かったのです。

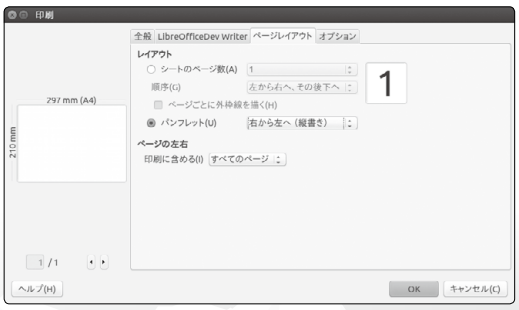
5.3からはテーブルスタイルとして実装されました。表のデザインもスタイルで管理するようになったということです。それにより何度も変更できるようになるなどの多大なメリットがあります。しかし、5.3.0の段階ではクラッシュバグがあります[注4]。このバグが修正されるまでは、テーブルスタイルを使用するのは避けたほうがいいでしょう。

パンフレット印刷

パンフレット印刷でページの方向を選択できるようになりました（**図9**）。今までは横書き（左とじ）のみでしたが、今後は縦書き（右とじ）でもパンフレット印刷できるようになりました。これは筆者の要望により追加されたものです。CTL（Complex Text Layout）向けには以前よりこのような機能があったので、CJK（中国語・日本語・韓国語）でも選択できるようにしただけであり、新機能として実装されたわけではありません。筆者としては同人小説などに便利ではないかと考えています。

注4） https://bugs.documentfoundation.org/show_bug.cgi?id=101648

図9　パンフレット印刷でページの方向を選択できるように



Calc

ワイルドカードと正規表現

Calcはこれまで、数式で正規表現が使えていました。しかしExcelではワイルドカードを使用できます。任意の1文字は“?”、任意の複数文字は“*”、ワイルドカードのエスケープは“~”で、これはMS-DOSのころから変わっていないので馴染みがあることでしょう。5.3からはこのExcelとの相互運用性を考慮し、同じくワイルドカードを使用するようになりました。

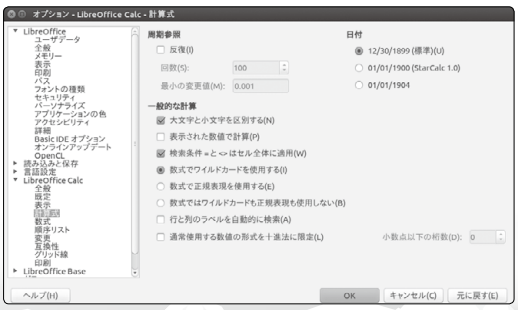
［ツール］-［オプション］-［LibreOffice Calc］-［計算式］の［数式でワイルドカードを使用する］、［数式で正規表現を使用する］、［数式ではワイルドカードも正規表現も使用しない］で挙動を変更できます（**図10**）。

Calcの熟練者には正規表現のほうが使いやすいと思われるので、必要に応じて設定を変更してください。

関数の変更点

5.3で新たに追加された関数はありませんが、EFFECTIVE関数がEFFECT関数に変更されました。これもExcelとの相互運用性の向上のためです。また、5.1より前のバージョンでは誤ってWEEKNUM関数をISOWEEKNUM関数として保存していたので、この暫定対応が行われました。

図10　ワイルドカードあるいは正規表現を使用するか、いずれも使用しないかを選択できる



分数の書式

［書式］-［セル］-［数値］タブの［分数］が強化され、Denominal places（直訳では分母の場所）が指定できるようになりました。

ピボットテーブル

ピボットテーブルで中央値（MEDIAN）関数が使えるようになりましたが、ODS形式のみ対応ですのでご注意ください。

その他

すでに入力済みのセルを結合する場合、挙動を指定するダイアログが表示されるようになりました。また、関数ウィザードで関数を検索できるようになりました（**図11**）。これは便利だと思います。

図11　［検索］欄が追加された

図12　モノクロなのでわかりにくいが、新しく追加されたVividとPencil

Impress

テンプレートを選択

Impress起動後［テンプレートを選択］が自動起動するようになりました。テンプレートマネージャーのうちImpressのテンプレートだけを起動できるようにしたものです。また、従来の［ファイル］-［ウィザード］-［プレゼンテーション］から起動できるテンプレートウィザードはソースコードから削除されました。

テンプレート2点追加

新たに［Vivid］と［Pencil］というテンプレートが追加されました（**図12**）。いずれも㈱アシスト[注5]が作成し、Creative Commons 0（いわゆるパブリックドメイン）で公開したものです。

フォトアルバム機能の強化

Impressには、実はフォトアルバム機能があり、［挿入］-［メディア］-［フォトアルバム］で起動できます。その名のとおり指定した画像をスライドにできます。これまでは画像をファイルに埋め込んでいましたが、ファイルサイズが大きくなって不都合なこともあるので、［画像をリンクする］というオプションが追加されました。

Draw

矢印の形状が追加されました。

Base

添付のFirebirdのバージョンを3.0にし、2.5以前で作成したファイルを扱えなくなりました。どうやらFirebirdというのはそういうデータベースのようですが、今のところはまだLibreOfficeで使用しているということはかなりのレアケースでしょうから、影響はあまり大きくないと思われます。SD

注5） アシストはLibreOfficeのサポートサービス事業を行っていましたが、昨年8月末で新規販売を停止し、事実上の撤退を公表しています。

第59回 メモリバリア問題を解決する membarrierシステムコール

Text：青田 直大　AOTA Naohiro

Linux 4.10-rc5が1月22日にリリースされています。この記事が出るころには、きっと今年最初のリリースとなる、Linux 4.10が出ていることでしょう。どんな機能が入っているのか楽しみです。

今回はLinux 4.3から、メモリバリアの問題を解決するmembarrierシステムコールを中心に紹介します。

マルチコア環境での落とし穴

マルチコアの環境が一般的になるにつれて、プログラム側も複数のCPUを活用するようなものが増えています。各CPU上でスレッドを動かして、効率よく処理を進めようというわけです。しかし、マルチコア・マルチスレッド環境では、シングルコア環境とは違ったプログラムの取扱いが必要です。

たとえば、**リスト1**のプログラムを見てみましょう。CPU0が関数`worker()`を動かし、CPU1が関数`client()`を実行しているとします。CPU0は、何か計算をしてその結果を`result`に書いてから`done`を1にして計算終了を通知します。CPU1は、`done`が0である間、busy loopをして計算の終了を待ちます。正常に動けば、loopを抜けたあとには`result == 42`が成り立つはずです。

しかし、このプログラムは3つの理由で期待どおりには動かないことがあります。なぜ期待どおりの動作が起こらないのか、どうすれば期待どおりの動作になるのかを見ていきましょう。

コンパイラによる入れ替え

1つ目の理由は、コンパイラの最適化による操作の入れ替えが起こり得ることです。コンパイラは書いたとおりの順番で、プログラムをコンパイルするとは限りません。

たとえば、`result = 42;`と`done = 1;`の順序は入れ替えられることがあります。結果として`done = 1;`が実行され、clientがbusy loopを抜け、`result == 42`を評価したあとに、`result = 42`の代入が行われることがあります。

したがって、コンパイラに命令の入れ替えを行わないように指示する必要があります。もちろん最適化を無効にしてしまえば、プログラム全体で入れ替えをなくすことができますが、それではほかの、問題のない部分まで最適化のメリットを無駄にしてしまいます。代わりに、命令を入れ替えたくない境界、ここでは`xxx();`

とコメントしている位置にbarrier()を入れて、その前後での命令の入れ替えを防ぎます。

メモリバリア

マルチコア環境では、コンパイラバリアを入れてもまだ期待どおりの動作をしないことがあります。それはCPUのキャッシュの構造上、あるCPUで行った操作がその順番どおりにほかのCPUには見えないことがあるためです。

たとえば、CPU0がプログラムに書いてあるとおりにresult = 42;してから、done = 1;しても、CPU1上ではdone = 1になってから、result = 42になることがあります。そこでメモリバリアが必要となります。

xxx()の位置にメモリバリアを入れることで、どのCPUにおいてもdone = 1よりも前にresult = 42の更新が通知されることが保証されます。また、yyy()の位置にもメモリバリアが必要です。これはresultの値をbusy loopを抜ける前に取得していた場合のキャッシュを無効にするためです。

これら2つのメモリバリアにより、doneが更新されたときには、最新のresultがclient()に見えることを保証します。

userspace RCU

こうしたメモリバリアを必要とするライブラリに、userspace RCUがあります。これはRCUという、Linuxカーネルで多く使われている同期メカニズムをuserlandで実装するものです。

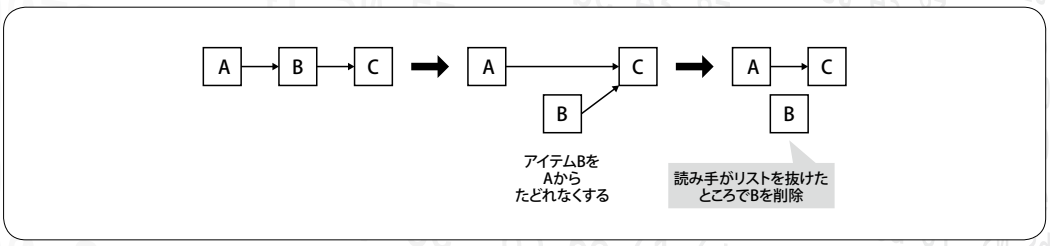
RCUの動作をリストからのアイテムの削除で見てみましょう。初期状態は**図1**のように、3つのアイテムがつながったリストになっています。ここから、アイテムBを削除します。まず、AからBのポインタを、AからCへとつなぎ替えます。この操作はatomicに行われるので、Aを見ているスレッド(また、このあとのスレッド)は問題なく、その「次の」アイテムCに進むことができます。一方、まだBからCへのポインタがつながっているので、Bを見ているスレッドも問題なく動作を続けることができます。

あとはのちほどアイテムBを解放すればいいわけです。しかし、そのタイミングが問題です。Bを見ていたスレッドがすべていなくなってから、削除する必要がありますが、それはどうやって知ればよいでしょうか。

▼リスト1　clientがworkerの実行待ちをするサンプルプログラム

```
#include <assert.h>

#define barrier() __asm__ __volatile__("": : :"memory")

int done = 0;
int result = 0;

void worker()
{
  // do something
  result = 42;
  // xxx();
  done = 1;
}

void client()
{
  int x;
  while(done == 0) continue;
  // yyy();
  assert(result == 42);
}
```

▼図1　RCUによるアイテム削除



RCUのキーとなるAPIは3つあります。読み手がデータにアクセスする区間を**rcu_read_lock()**と**rcu_read_unlock()**とで囲みます。この区間をRCUクリティカルセクションと呼びます。書き手は**synchronize_rcu()**を呼び出して、その時点でRCUクリティカルセクションにいるスレッドが、該当クリティカルセクションを抜けるのを待ち、アイテムの解放を行います。

Linuxカーネルでは、**rcu_read_lock()**から**rcu_read_unlock()**の期間で割り込みを禁止し、ほかのすべてのCPUが一度スケジューラを通るまで待ちます。RCUクリティカルセクションでは、割り込みが禁止されているので、スケジューラを通ったということは、**rcu_read_unlock()**を抜けた(もしくはRCUクリティカルセクションにいなかった)ことを示すというわけです。

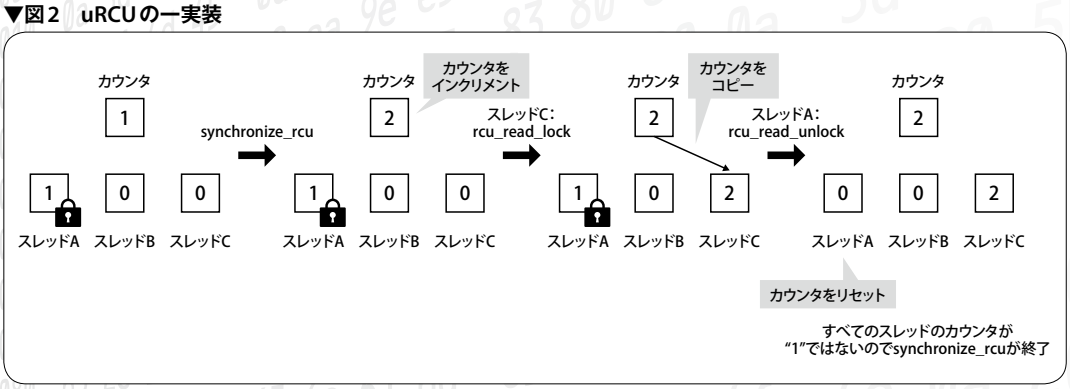
カーネル空間ですので、割り込みを禁止し、スケジューラにフックを入れるということができましたが、ユーザランドではそのようなことはできません。ではuserland RCUで、どのように**synchronize_rcu()**が**rcu_read_unlock()**を抜けるのを待つのか1つの実装例を見てみましょう(**リスト2**)。

この実装では、**synchronize_rcu()**ごとにインクリメントされるカウンタを持ちます。読み手は**rcu_read_lock()**で、スレッド固有変数にカウンタの値を読み込み、**rcu_read_unlock()**でその変数に0を入れてリセットします。**図2**のように、カウンタが"1"のときに**synchronize_rcu()**を呼ぶ動作を見てみましょう。このときスレッドAの変数は"1"でRCUクリティカルセクションにいます。スレッドBとCの変数は"0"でRCUクリティカルセクションの中にいません。**synchronize_rcu()**は、カウンタをインクリメントして、変数がもとの値を保持しているスレッドがいなくなるまで待機します。スレッドAが**rcu_read_unlock()**を呼べば、変数が0になりますし、スレッドCが新しく**rcu_read_lock()**を呼んでもカウンタの値が変わっているので、**synchronize_rcu()**を妨げません。

▼リスト2　ユーザランドでの実装例

```
void rcu_read_lock()
{
  thread[myid]->processing = cnt;
  smp_mb(); // メモリバリア
}

void rcu_read_unlock()
{
  smp_mb(); // メモリバリア
  processing = 0;
}

void synchronize_rcu()
{
  smp_mb(); // メモリバリア
  mutex_lock(&gp_lock);
  cnt++;
  barrier();
  for (i=0; i<n; i++)
    while (thread[i]->processing == (cnt-1))
    wait();
  mutex_unlock(&gp_lock);
  smp_mb();
}
```

▼図2　uRCUの一実装



ここでさきほどのメモリバリアの例を思い出しましょう。読み手がrcu_read_unlock()などで変数を読む後、そして書き手がsynchronize_rcu()で変数を更新する前には、メモリバリアが必要です。もし入っていなければ、さきほどと同様にRCUクリティカルセクション内の更新が、ほかのCPUに伝わっていない状況が起こりえます。

しかし、RCUは書き手に対して読み手が数多いケースにおいて、読み手のオーバヘッドを書き手に移すことでスケーラビリティを実現しています。そうすると、rcu_read_lock()にメモリバリアを入れるのは、好ましくはありません。特に書き手がいない場合には、そのメモリバリアは必要もないのに実行されることになります。

membarrier システムコール

こうした問題を「書き手から、読み手にメモリバリアを実行させる」という方式で解決するために作られたのが、membarrierシステムコールです。

このシステムコールが呼ばれると、(ちょうどカーネルでの実装のように)ほかのすべてのCPUが一度スケジューラに割り込まれるまでカーネル内で待機してから、ユーザ空間に戻ってきます。スケジューラに割り込まれることで、暗黙的にメモリバリアが行われます。

実際に、このシステムコールを使う様子を考えてみましょう。読み手ではコンパイルバリアを使い(冒頭の例のxxxの位置)、書き手ではmembarrierシステムコールを使います(yyyの位置)。client()はdone == 1を観測したあとにmembarrierシステムコールを実行します。

▼表1　membarrierシステムコールを追加したコミットでの比較

	読み込み回数	書き込み回数
メモリバリア	1,701,557,485	2,202,847
シグナル	9,830,061,167	6,700
システムコール	9,952,759,104	425
システムコール(check)	7,970,328,887	425

xxxの位置にコンパイラバリアがあることから、workerが動くCPU上ではresult = 42のあとにdone = 1が実行されています。したがって、ここでメモリバリアが起きれば、resultの更新がclientの動くCPU側で見えることが保証されます。その一方で、書き手がいないタイミングであれば、clientではコンパイラバリアしかないため、オーバーヘッドなく処理が進むようになります。

実はこうした「相手にメモリバリアを起こさせる」しくみはシグナルを使っても実装されています。その場合、ほかのスレッドにSIGUSR1など空いているシグナルを送って、そのシグナルハンドラでメモリバリアを実行します。

membarrierシステムコールを追加したコミットで、メモリバリア、シグナルによるメモリバリア、システムコールによるメモリバリアを比較しているので見てみましょう(**表1**)。「システムコール(check)」は、読み手側でmembarrierシステムコールの存在を確認する(そして、あればコンパイラバリア、なければメモリバリアを使う)実装です。

読み込み回数を見ると、メモリバリアに対して、その他の実装が大きく読み手のパフォーマンスを改善していることがわかります。シグナルとシステムコールの実装を比べてみましょう。チェックがなければ、システムコールの方がパフォーマンスが良くなっていますが、チェックを入れることで若干遅くなっています。また、書き込みのパフォーマンスはシグナルよりもシステムコールの方が遅くなっています。これはメモリバリアが実行されるまでにかかる時間が、シグナルよりもシステムコールの方が長くなるためです。

チェックを入れた場合には、シグナルよりも読み込みも書き込みも遅くはなりますが、システムコールを使う実装にはシグナルにはない大きな利点があります。シグナルを使う実装では前述したように、使っていないシグナルを活用します。ライブラリやほかのプロセスにコード

をinjectする場合には、どのシグナルを(将来でも)使わないのかを知ることはできません。こうしたケースでは、システムコールの方にアドバンテージがあります。

その他のLinux 4.3の変更：PIDs CGroup

最後にそのほかのLinux 4.3の新機能を紹介します。1つ目は、PIDs CGroupです。CGroupはプロセスをグループ分けし、各グループにさまざまなリソース制限をかけるための機能です。DockerやLXCなど、コンテナ環境でのリソース制限を実現するために使われています。PIDs CGroupはグループ内の(forkによる)プロセス数を制限します。

当然ながら、システムに作ることができるプロセスの数は有限です。このプロセス数の上限値は、/proc/sys/kernel/threads-maxから見ることができます。threads-maxは、上限までプロセスを作っても、プロセス情報の保持に使うメモリ量がシステムの全メモリの1/8を超えないように設定されます。したがって、メモリ制限にはひっかからずに、プロセスを大量に作ることができます。

プロセスを大量に生成し、プロセステーブルを使いきってしまえば、それ以上新しいプロセスを作ることができなくなります。既存のプロセスは動き続けることはできますが、新しいプロセスは起動できず、結果としてシステムが動きが大きく制限されることになります。こうした攻撃は、フォーク爆弾と呼ばれています。

もしコンテナの中のプログラムが「フォーク爆弾」を行えば、その影響はほかのコンテナにもおよびます。このような攻撃をされても、コンテナ内だけで被害がおさまるように、PIDs CGroupが実装されました。

サスペンド時のディスク同期を省略可能に

もう1つはサスペンド時のディスク同期に関する変更です。Androidなど携帯端末でLinuxが使われる中で、サスペンド時のsyncを省略したいケースが出てきました。Androidなどでは、ちょっとした処理、たとえば定期的な位置情報の確認やパケットの受信などを行うためにサスペンドから起きることがあります。処理が終わればまたサスペンドに入って電力を節約します。このときに、サスペンドに入るのに多くの時間がとられてしまえば、ほとんどの電力消費がサスペンドに入るためだけに使われる……といったことも起こりえます。こうなってはサスペンドをうまく活用できません。

サスペンドの時間を長くする要因の1つに、ディスク同期があります。Linuxでは、サスペンドの前にカーネルがsyncを呼び出して、変更中のデータをすべてディスクに書き出しています。syncしておくことで、サスペンドやそこからの復帰に失敗しても、そこまでの書き込みはディスクに反映されて失われることなく残ります。syncにかかる時間はタイミングとディスク環境に大きく依存します。とくに何もしていないタイミングであれば、書き込むものがなく、syncがすぐに終了します。しかし、もしもアプリケーションが大量の書き込みを遅いデバイス、たとえばSDカードなどに行った直後にサスペンドすれば、それだけsyncに時間をとられることになります。

サスペンドからの復帰に失敗した場合のことを考えれば、syncは大切な処理です。しかしながら、Androidのように比較的サスペンドが安定し、省電力が大切なシステムであれば、syncを省きサスペンドを有効活用したくなります。

そこでLinux 4.3から新たなコンパイルオプションが入り、syncを省略してサスペンドの時間を短縮できるようになりました。この場合、Androidなどサスペンドを行うシステム側が、適切なタイミングでsyncを行うことになります。

SD

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

小さなツールを使ったテキスト処理の第2回目。シェルの文字列演算、正規表現や文字クラスに注目します。

第11回 テキスト処理(その2)

Unixのツールは、1つ1つ独立したプログラムで、標準入出力を通して連結できることが特徴です。複数のツールを組み合わせたり、そこにシェルスクリプトによる制御を加えたりしていると、まるでオブジェクト指向でプログラムを作っているかのようです。Unixでは、オブジェクト指向という概念が生まれる前から、ソフトウェアモジュールの独立性や一貫したインターフェースがシンプルな形で実現されていました。

一歩進んだテキスト処理

連載第9回(本誌2017年1月号)で、機敏で小さなテキスト処理コマンドをいくつか確認しました。今回は、正規表現を中心に、少し高度なテキスト処理を目指します。シェークスピアーのハムレットの英語文[注1]を題材にしてみます。ファイル名はhmlt.txtにしました。

手始めに第9回でも登場したwcを使って、復習がてら対象データの状況を分析します。

行数、単語数、文字数を数える
```
$ wc hmlt.txt↵
    9086   32014  179670 hmlt.txt
```

3つ並んでいる数字は、左から行数、単語数、文字数です。これらを個別に表示させるには、-l(行数)、-w(単語数)、-m[注2](文字数)のオプションを指定します。

ここから先のコマンドは、^、$、[、]、(、)など、記号を用いて動作を制御します。$、{、}、(、)、;、|、␣(半角スペースの意味)などは、文脈によっては先にシェルが解釈して、それからコマンドに渡ります。シェルにいじられたくない文字は、バックスラッシュ(\)やシングルクォート(')を使ってエスケープしなければなりません。

tr - TRanslate characters

trは、単純な文字の置き換えや削除をします。置き換えや削除の対象を文字の集合で表せるのが大きな特徴です。さらに、**文字クラス**は、後の正規表現でも使う重要な文字集合の表現方法です。

trや以降のコマンドで使える文字クラスには**表1**があります。

trの置換は、第1引数に与えられた文字列を1文字ずつ第2引数に対応させて置換します。第2引数が第1引数より短ければ、対応先のない第1引数中の文字には、第2引数の最後の文字を割り当てます。'-'を用いて連続する文字コードの範囲を指定することができます。

注1) ネット上にある、英語のテキスト(たとえば、http://shakespeare.mit.edu/hamlet/full.html)をhtmlではなく、プレーンテキストで保存して使いましょう。

注2) 以前は文字数を数えるにあたって-cを使っていましたが、こちらはバイト数を数えるオプションとして使います。

trを活用する例を見ていきましょう。

小文字を大文字に変換する1
```
$ tr a-z A-Z < hmlt.txt
 ...略...
SPEAK LOUDLY FOR HIM.
TAKE UP THE BODIES: SUCH A SIGHT AS THIS
BECOMES THE FIELD, BUT HERE SHOWS MUCH AMISS.
GO, BID THE SOLDIERS SHOOT.

A DEAD MARCH. EXEUNT, BEARING OFF THE DEAD
BODIES; AFTER WHICH A PEAL OF ORDNANCE IS
SHOT OFF
```

次の書き方は、前の例と同じ動作になります。

小文字を大文字に変換する2
```
$ tr [:lower:] [:upper:] < hmlt.txt
```

trのオプション-cを指定すると、最初に指定された文字列の補集合を取ります。また、-sオプションを指定すれば、入力データ中で同じパターンが繰り返された場合1つにまとめます。\nは、C言語での表現と同様に改行を意味します。

アルファベット以外を改行に変換することで1行1単語の一覧を作る
```
$  tr -cs [:alpha:] "\n" < hmlt.txt | sort | uniq
 ...略...
your
yours
yourself
yourselves
youth
zone
```

▼表1　POSIX文字クラス(これらはロケールの設定によってどの文字とマッチするかが決まります)

クラス	意味
[:alnum:]	英数文字
[:alpha:]	アルファベット
[:cntrl:]	制御文字
[:digit:]	数字
[:graph:]	グラフィック文字
[:lower:]	小文字
[:print:]	印字可能な文字
[:punct:]	句読記号
[:space:]	空白文字
[:upper:]	大文字
[:xdigit:]	16進数

コントロール文字など、印字できない文字が入っているところを取り除きましょう。次の例では、改行も取り除かれます。

印字可能な文字だけにする
```
$ tr -cd [:print:] < hmlt.txt
```

シーザー暗号という、アルファベットを別のアルファベットに1対1で置き換える、簡単な暗号化をしてみます。A～MをN～Zに、N～ZをA～Mに置き換えています(小文字についても同じ)。第1引数と対応する第2引数の指定を確認してください。

シーザー暗号化して冒頭の8行だけ確認してみる
```
$ tr A-Za-z N-ZA-Mn-za-m < hmlt.txt | head -8
Unzyrg: Ragver Cynl



Gur Gentrql bs Unzyrg, Cevapr bs Qraznex
     Funxrfcrner ubzrcntr
     Unzyrg
     Ragver cynl
```

もう一度、同じ暗号化をすれば、もとの文を得られます。

冒頭8行の出力に同じシーザー暗号化で復号
```
$ tr A-Za-z N-ZA-Mn-za-m < hmlt.txt | head -8 | tr A-Za-z N-ZA-Mn-za-m
Hamlet: Entire Play



The Tragedy of Hamlet, Prince of Denmark
     Shakespeare homepage
     Hamlet
     Entire play
```

文字列の操作と正規表現

Unixでテキスト処理を自在に操るには、シェルの文字列演算子と正規表現の2つをきちんと扱えることが基礎技術です。ここからは、この2つの基本を見ていきましょう。

シェルの文字列演算

bashには、**表2**のような文字列を操作するしくみがあります。シェル変数と文字列演算${ : }演算です。

このしくみを使って、フルパスからベースネームを取り出す**basename**コマンド、フルパスからディレクトリ名を取り出す**dirname**コマンドと同じような働きをさせることができます。一連の操作を順に見ていきましょう。

```
変数の文字列操作の確認
以下'/etc/resolv.conf'を対象に試してみる
$ ls /etc/resolv.conf↵
/etc/resolv.conf

lsの結果を変数に入れる
$ i=$(ls /etc/resolv.conf) ; echo $i↵
/etc/resolv.conf

はじめに見つかった'.'までを取り除く
$ i=$(ls /etc/resolv.conf) ; echo ${i#*.}↵
conf

はじめに見つかった'/'までを取り除く
$ i=$(ls /etc/resolv.conf) ; echo ${i#*/}↵
etc/resolv.conf

フルパスからベースネームを取り出す
$ i=$(ls /etc/resolv.conf) ; echo ${i##*/}↵
resolv.conf

フルパスからディレクトリ名を取り出す
$ i=$(ls /etc/resolv.conf) ; echo ${i%/*}↵
/etc

次のように入れ子にする書き方はできない
$ i=$(ls /etc/resolv.conf) ; echo ➘
${${i##*/}%.*}↵
-bash: ${${i##*/}%.*}: bad substitution

フルパスから拡張子を取り除いたファイル名を取り出す
$ i=$(ls /etc/resolv.conf) ; i=${i##*/} ; ➘
echo ${i%.*}↵
resolv

こうすれば、拡張子を変更できる
$ i=$(ls /etc/resolv.conf) ; i=${i##*/} ; ➘
echo ${i%.*}.def↵
resolv.def

拡張子を取り出そうとして失敗。どこがおかしいかわかりますか？
$ i=$(ls /etc/resolv.conf) ; i=${i##*/} ; ➘
echo ${i##.*}↵
resolv.conf

フルパスから拡張子のみを取り出す
$ i=$(ls /etc/resolv.conf) ; i=${i##*/} ; ➘
echo ${i##*.}↵
conf
```

▼表2　変数内の文字列の操作

演算書式	意味
${変数＃パターン}	変数の値のはじめの部分とパターンが一致した場合、最も短く一致した部分を取り除き、残りを返す
${変数＃＃パターン}	変数の値のはじめの部分とパターンが一致した場合、最も長く一致した部分を取り除き、残りを返す
${変数％パターン}	変数の値の終わりの部分とパターンが一致した場合、最も長く一致した部分を取り除き、残りを返す
${変数/パターン/文字列}	変数の値で、パターンと最も長く一致した部分を文字列と置き換える。最初に一致した部分だけが置き換えられる。文字列がnullなら、一致した部分を削除
${変数//パターン/文字列}	変数の値で、パターンと最も長く一致した部分を文字列と置き換える。一致した部分は、すべて置き換えられる。文字列がnullなら、一致した部分を削除。変数に@か*を指定した場合、位置パラメータを順に処理して、展開結果はリストに入る

ほかにもbashには、CやRubyでおなじみのprintfによる書式付き出力もあります。これらbashの変数の文字列操作機能を使いこなせれば、いちいち外部コマンドを呼び出さなくてもさまざまな文字列の加工ができて、ファイル操作やテキスト処理能力が向上します。

正規表現入門

正規表現は、その成り立ちの経緯から、基本正規表現(Basic Regular Expression)、拡張正規表現(Extended Regular Expression)や、拡張正規表現のさらに拡張されたものなどがあります。各コマンドで、どのレベルの正規表現が使えるか異なりますので注意してください。

正規表現は、とても強力で表記要素もたくさんあります。拡張正規表現(ERE)は、扱えるメタ文字が基本正規表現(BRE)に対して増えていますが、完全な上位互換というわけではありません。場面によって使い分けが必要です。

表3に加えて、**表4**の文字クラス、前出のPOSIX文字クラスも扱えます。難しそうですが百聞は一見にしかず。この後も、実際にコマンドを使いながら見ていきましょう。

▼表3　正規表現のメタ文字

文字	意味
.	改行を除く任意の1文字
^	行頭
$	行末
*	直前の正規表現の0回以上の繰り返し
?	直前の正規表現の0回か1回の繰り返し（EREのみ）
+	直前の正規表現の1回以上の繰り返し（EREのみ）
()	括弧内の正規表現をグループ化する（EREのみ）
|	選択'|'の前の正規表現か後ろの正規表現のどちらかとマッチする（EREのみ）
\メタ文字	'\'に続くメタ文字をエスケープする（特殊文字としてではなくその文字そのものとしてあつかう）
\{n\}	直前の1文字のn回の繰り返し（EREでは括弧の前の\は不要）
\{n,\}	直前の1文字が最低n回繰り返し（EREでは括弧の前の\は不要）
\{n_1,n_2\}	直前の1文字がn_1回からn_2回のうちいずれか繰り返し（EREでは括弧の前の\は不要）
\(... \)	正規表現をグループ化する（EREでは括弧の前の\は不要）
\n	グループ化したn番目の正規表現を示す

▼表4　文字クラスの表現

表現	意味	例
[～]	文字クラスのうち任意の1文字	[Ade] → Aかdかeにマッチする
[^ ～]	文字クラスの補集合のうち任意の1文字（はじめの'['の直後に'^'を記述した場合のみ）	[^PA] → PとA以外の文字にマッチする
[○-◎]	'-'は連続する文字コードの範囲を示す	[b-d] → bかcかdにマッチ

grep fgrep egrep - Global Regular Expression Print

データの中に、文字列とマッチする行を抽出したり、マッチしない行を抽出したりと、とにかく便利に使えるのがgrepファミリーです。

基本正規表現（BRE）が使えるgrep。-Eオプションをつければ拡張正規表現（ERE）が使えます。egrepは、これと同じ意味です。Extendの意味ですね。正規表現を使わないなら、-Fオプションを付けます。Fixedの意味です。

STEP UP!

歴史的経緯は、grep、fgrep、egrepは異なるバイナリで配布されていました。正規表現を使わなければfgrepは高速です。egrepは強力で高速なパターンマッチアルゴリズムを使います。grepで片付くところでも、egrepを使っていると、「こいつわかっているな」と見てもらえるかも……。

現在は、grep、egrep、fgrepともmacOSでは同じバイナリです。CentOSでは、egrepとfgrepがgrepへのシンボリックリンク、Ubuntuでは、egrep、fgrepはシェルスクリプトで、内部でgrepを起動しています。

大文字で始まる行をすべて表示する
```
$ egrep '^[A-Z]' hmlt.txt
 ...略...
Take up the bodies: such a sight as this
Becomes the field, but here shows much amiss.
Go, bid the soldiers shoot.
A dead march. Exeunt, bearing off the dead
bodies; after which a peal of ordnance is
shot off
```

大文字で始まらない行をすべて表示する
```
$ egrep ^[^A-Z] hmlt.txt
 ...略...
[Aside]  And yet 'tis almost 'gainst my
conscience.
                    Look to the queen there,
ho!
                    He is justly served;
                    Let us haste to hear it,
```

次の例では、EREとBREの違いを見てみましょう。まずは、egrepを使って。

行がすべて大文字でできている行の行数（ERE）
```
$ egrep ^[A-Z]+$ hmlt.txt | wc -l
     780
```

上の正規表現をgrepで実行してみましょう。

結果は、780ではなくて0になりました。+がEREでしか使えないからです。そこで、grepではこう書きます。

```
行がすべて大文字でできている行の数(BRE)
$ grep ^[A-Z][A-Z]*$ hmlt.txt | wc -l
     780
```

grepに-vオプションを指定すると、マッチしない行を表示します。これを使って行がすべて大文字の行以外の行数を数えます。

```
行がすべて大文字の行以外の行数を数える
$ egrep -v ^[A-Z]+$ hmlt.txt  | wc -l
    8306
```

英単語は、qの後ろは必ずuが続きます。ハムレットの中には、qに続く文字がu以外のものはあるのでしょうか。

```
qの後ろにuが付かない単語はあるのか
$ egrep [Qq][^Uu] hmlt.txt
```

ありませんでした。:-) 続いて文字クラスを指定して、アルファベットか数字の後にumが続くところは、何行あるのか数えてみます。

```
アルファベットの後ろにumが付かない単語を含む行の数
$ egrep [[:alnum:]]um hmlt.txt | wc -l
      74
```

74行ありましたね。

必ず覚えておくべきgrepのオプション

grepには、いろいろと便利なオプションがあります。ここでは、必ず覚えておきたいオプションを紹介しておきます。

grepで文字列を検索する対象は、複数のファイルを横断できます。このときに便利な、「-H ファイル名も表示する(複数ファイル指定ではデフォルト表示)、-h ファイル名を表示しない」、「-n ファイル名と行番号を表示する」、「-L マッチしなかったファイル名を表示、-l マッチしたファイル名を表示」があります。

「-i 大文字小文字を区別しない」、文脈がわからないと困る場合に便利なオプションとして「-A *n* 続く*n*行表示」、「-B *n* 前の*n*行表示」、「-C *n* 見つかった行の前後*n*行」(後ろAfter、前Before、中央Center と覚えましょう)などは、活躍の場が多く必ず押さえておきたいです。

有名な「成すべきか、成さざるべきか、それが問題だ」のセリフを探してみます。大文字か小文字かわかりませんので-iを指定、何行目かも知りたいので-n、前後の文脈も知りたいので-Cも指定します。

```
有名なフレーズを探してみよう1
$ egrep -niC 2 'to be or not to be' hmlt.txtl
```

見つかりません。条件を緩めてみます。

```
有名なフレーズを探してみよう2
$ egrep -niC 2 'to be' hmlt.txt
 ...略...
8165-To let this canker of our nature come
8166-In further evil?
--
8282-dizzy the arithmetic of memory, and yet ➥
but yaw
8283-neither, in respect of his quick sail. ➥
But, in the
8284:verity of extolment, I take him to be a ➥
soul of
8285-great article; and his infusion of such ➥
dearth and
8286-rareness, as, to make true diction of ➥
him, his
```

たくさんマッチして、目的のセリフを確認するのは大変です。条件を変えてみましょう。

```
有名なフレーズを探してみよう3
$ egrep -niC 2 'to be or' hmlt.txt
```

これでは、見つかりませんでした。行が別れているのかもしれませんし、途中に‘,’があるのかもしれません。別のワードで探してみます。

```
有名なフレーズを探してみよう4
$ egrep -niC 2 'or not' hmlt.txt
 ...略...
3789-HAMLET
3790-
3791:To be, or not to be: that is the question:
3792-Whether 'tis nobler in the mind to suffer
3793-The slings and arrows of outrageous ➥
fortune,
--
5901-Deliberate pause: diseases desperate grown
```

```
5902-By desperate appliance are relieved,
5903:Or not at all.
5904-
5905-Enter ROSENCRANTZ
```

3791行目に確かにありました。やはりTo beの後ろに','がありました。でも、まだたくさんの個所にマッチしすぎていますね。もう少し絞ってみると、

有名なフレーズを探してみよう5

```
$ egrep -niC 2 'or not to be' hmlt.txt
3789-HAMLET
3790-
3791:To be, or not to be: that is the question:
3792-Whether 'tis nobler in the mind to suffer
3793-The slings and arrows of outrageous
fortune,
```

見事、目的の文脈が抜き出せました。

macOS限定のコマンドsayは、テキストを喋らせられるコマンドです。引数で与えた文字列を読み上げます。sayは、さまざまな国の言葉を喋る音声(ここでは声優と言っておきましょう)が登録されています(追加も可能です)。このコマンドを使って、どの声優がハムレット役に適しているのか、オーディションを開催してみましょう。

ハムレット役のオーディション macOS版

```
$ for actor in $(say -v ? | egrep en_ | egrep
 -o '^[A-Za-z]+'); do sleep 2; echo ${actor};
 say -v ${actor} ${actor}; sleep 1; (egrep -C
 4 -i "not to be" hmlt.txt| say -v ${actor})
 ; done
```

少々長いワンライナーですが、順番に読んでいけば大丈夫です。音声の一覧から、egrep en_で英語を、egrep -oで、名前部分(行頭からアルファベットの並びにマッチした個所)を抽出。名を名乗ってから、ハムレットの台詞を読み上げさせています。

Linuxにもespeakというコマンドがあります[注3]。オーディションを実施するスクリプトは、後に解説するawkを使っています。

ハムレット役のオーディション Linux版

```
$ for actor in $(espeak --voices | awk '$2 ~
/^en/{print $4}'); do sleep 2; echo ${actor};
 espeak -v ${actor} ${actor}; sleep 1; (egrep
 -C 4 -i "not to be" hmlt.txt| espeak -v
${actor}) ; done
```

最後は、ハムレットを離れて、日本の郵便番号にマッチするかどうかの正規表現を作ってみましょう(**図1**)。

BREでは、行頭、行末に郵便番号が現れるとマッチしません。EREの選択'|'を使えば、この問題を克服できます。

今回のまとめと次回について

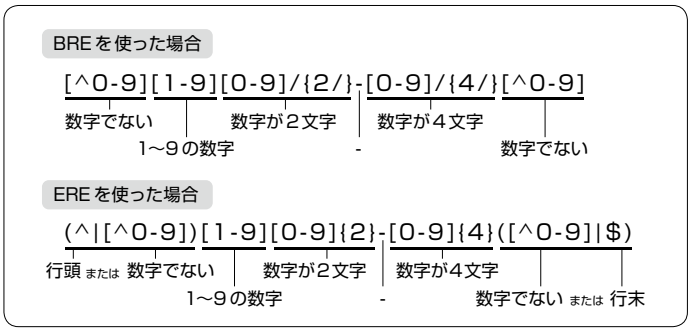
今回は、テキスト処理の2回目、とくに正規表現に重点を置いて解説しました。次回は、CUIゲーム&ジョークコマンド特集を予定しています。SD

注3) Debian系Ubuntuなどは、$ sudo apt install espeakで導入できるはずです。

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)】
basename(1), dirname(1), tr(1), grep(1), awk(1), say(1)(macOSのみ), espeak(1)(Linuxのみ)
【以下はinfoコマンドを使って確認】
printf

▼図1 BREを使った場合(上)とEREを使った場合(下)の日本の郵便番号



March 2017

NO.65

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

コミュニティとインターネットの議論で締めた2016年

今回は、11月に広島で行った研究会と、東京で行われたInternet Weekについて報告します。

jus研究会 広島大会

今年度としては6回目となる研究会を広島で開催しました。今回も地元のITコミュニティ運営者を迎え、運営に関する議題をいただいて討論するセッションを行いました。参加者は21人でした。

■ITコミュニティの運営を考える
【講師】西本 卓也（NVDA日本語チーム）、
火村 智彦（LT駆動開発／すごい広島）、
榎 真治（jus／LibreOffice日本語チーム）、
法林 浩之（jus）
【日時】2016年11月27日（日）15:15～16:00
【会場】サテライトキャンパスひろしま

■コミュニティ参加者や運営者を増やすには

まず榎さんから出た議題は「どうやって新しい人にリーチしてコミュニティに参加してもらうか」でした。これに対しては、「NVDAのようにソフトウェア自体が万人を対象とするものではない場合は、人を集めるのが難しい」、「自分に適したコミュニティがないから作って運営を始めたので、新しい人に入ってもらうことはあまり重要ではない」などのコメントが出ました。

火村さんからは「地域に新しいコミュニティ運営者をたくさん作るにはどうしたら良いか」というお題が出ました。これには、「1人ではコミュニティにならないので、フォロワーが付くようにすることが重要」という回答がありました。ちなみに、地方都市では複数のコミュニティがあっても、関係者の大半が重複していることが、少なからずあります。やはり広島でもどのコミュニティにも顔を出す層が一部いて、残りは各団体固有の参加者層だそうです。

■ほかにもいろいろな話題が

西本さんからのお題は「コミュニティはトレンドを追うべきか」でした。jusの場合、インターネットやオープンソースが流行したときに受け皿になる団体がなく、jusが引き受けたことでjus自身の寿命も伸びました。また、技術トレンドに追随しながらソフトウェアを進化させるにはある程度の開発者数が必要なので、その母集団となるユーザもそれなりに必要とのことです。

筆者からは「コミュニティの平和を保つには」というお題を出しました。これについては、「健全な運営を乱す人には毅然と対応すべきである」、「敵対するコミュニティのことは表立って言及しない」、「コミュニティの運営ルールをきちんとしておいたほうが良い」などのコメントがありました。

最後に参加者から「モチベーションが落ち込んだときにどのように立て直すか」という質問が出ました。これには、「モチベーションは思い出すものであり、落ちたときはやらない」、「イベントの場合は（参加者の集まりが悪い、準備が間に合わないなど）出来が悪くてモチベーションが下がっても、当日が来れば終わるので我慢できる」といった回答がありました。

このテーマによるセッションも通算10回を超え

るほど回数を重ねていますが、同じ質問でも地域が異なると別の見解を聞くことができて興味深いです。今後も継続して開催していきたいと思います。

Internet Week 2016

毎年恒例のInternet Week（JPNIC主催）が昨年11月末に開催されました。jusは今回も後援団体として参加し、プログラムの企画や告知などの協力を行いました。プログラムは4日間で約30本を実施し、約2,400人の参加者を集めました。ここでは最終日に行われたIP Meetingを中心に報告します。

■IP Meeting 2016 ～見抜く力を！～
【日時】2016年12月2日（金）9:30～17:30
【会場】ヒューリックホール＆ヒューリックカンファレンス 2Fホール

IP Meetingは、Internet Weekを総括するプログラムとして長く親しまれています。今回も午前の部は「Internet Today!」と題する各分野の現状報告、午後の部は2016年のホットトピックを紹介するセッションが用意されました。

■Internet Today!

はじめに、2016年のインターネット運用動向と新技術の標準化動向が紹介されました。運用動向のトピックとしては、Web通信のHTTPS化が進んでいることや、DDoSが目立つことなどが取り上げられました。

続いてインターネットの社会的動向として、ネットワーク中立性に関するセッションが設けられました。LINE MOBILEにおいて導入されたコミュニケーションフリー（一部サービスは課金対象外とする）のしくみを例に、特定コンテンツのみ課金しないことと中立性の関係についての議論が紹介されました。最後にセキュリティ分野の総括があり、2016年10月に発生した富山大学に対する標的型サイバー攻撃を例に、インシデント対応の改革の必要性を訴えました。

■IW2016セッション総括！

午後の部の前半は、Internet Week期間中に行われたプログラムのまとめが紹介されました。たとえばIPv6分野ではIPv6を理解したうえでのセキュリティやトラブルシューティングが必要になっていること、セキュリティ分野で初めてハンズオンを実施したこと、社会派分野では通信の秘密／表現の自由／検閲の禁止と事業者のあり方などの話題が取り上げられたことなどの話がありました。

■インターネットが作る、未来の暮らしを考える

最後のセッションは「インターネットが作る、未来の暮らしを考える ～これからを豊かにするための八つの視点～」と題するパネルディスカッションでした。モデレータをjus幹事の砂原秀樹さんが務め、社会、人材育成、IoT、災害復旧、AI、女性／子ども、アプリケーションの観点から登壇者が講演しました。

印象に残った話題としては、AIで作成されたものの知的財産権はどうなるのか、実践的なセキュリティ人材育成プログラムSecCapを5年以上に渡り継続実施し着実に成果が上がっていること、WELQ問題は現代の検索エンジンでも情報の真偽までは判別できないことを露呈してしまったこと、熊本地震の被災地にて衛星回線を利用したWi-Fiサービスを提供した話、などがありました。

■終わりに

Internet Weekは今回で20回目を迎えましたが、それを機にいくつかの変更を行いました。たとえば、会場を浅草橋にあるヒューリックホール＆ヒューリックカンファレンスに移動したことや、各日に1日通しプログラムを設けたこと、それらを4日間聴講できる通し券を発売したことなどです。これらの試みがすべて成功だったかどうかはなんとも言えませんが、より良いイベントを作るために試行錯誤を繰り返すことは必要であると感じました。次回もさらに良いイベントにすべく協力していきたいと思います。SD

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

第63回 福島発「エフスタ!! TOKYO」で人工知能を学ぶ

ITエンジニアのスキルアップを応援する福島県のコミュニティ「エフスタ!!」。今号は彼らが定期的に東京で開催している「エフスタ!! TOKYO」のレポートです。

我々、Hack For Japanではこれまでの復興支援に関わる活動から、東北地方で頑張るさまざまなITエンジニア達と出会い、その人達が活動するコミュニティともつながっています。本連載でも過去数回紹介している、福島県でひときわ元気のある「エフスタ!!注1」もそんなコミュニティの1つ。2016年12月10日に開催された「エフスタ!! TOKYO」は人工知能がテーマで、最近話題の人工知能に関する勉強会とともに、原発問題を抱える福島の様子が伝えられました。福島の未来を願う彼らの活動をHack For Japanスタッフの鎌田がお伝えします。

「エフスタ!!」とは

初めて「エフスタ!!」というコミュニティを目にする読者の方のために、簡単にこのコミュニティについて紹介しようと思います。そもそもは、IT業界で働く人たちが楽しんで仕事ができるように身の回りのIT業界を変えたい、夢と希望をもった技術者を育てたい、という理想があり、そのために教育に力を注ぎたいと考えたのが発端です。そこで「エフスタ!!」は、地元福島のITが変われば、面白くなれば、夢と希望をもった技術者が増え、やがて世界を変える技術者が誕生する、という考えのもと、それを実現するためのきっかけ作りの場として運営されています（**図1**）。東日本大震災から数年が経ち、この「エフスタ!!」の東京での勉強会にも徐々に人が集まるようになってきています。

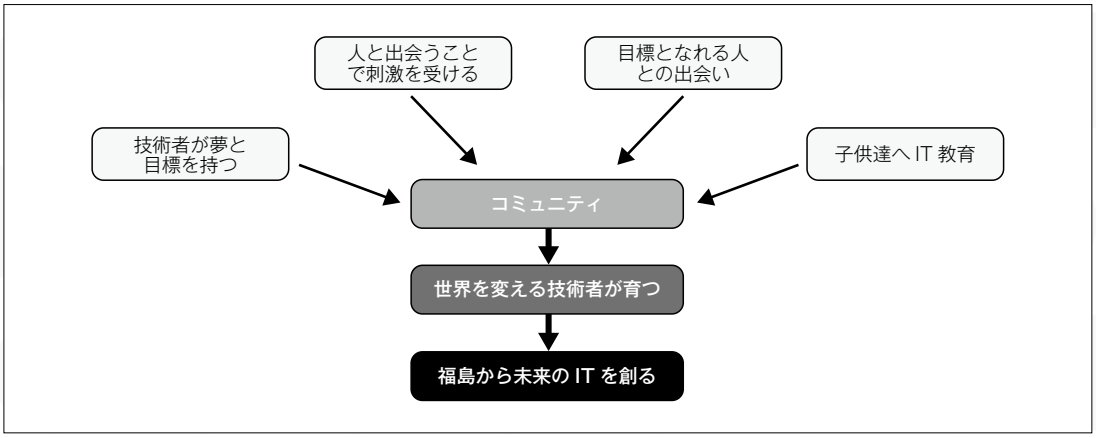
テーマ「人工知能って何？」

今回紹介する「エフスタ!! TOKYO」のテーマは「いまさら聞けない！ 人工知能×福島注2」です。世の中で注目を集めている人工知能に関して、実際に仕事で活用されている方が少ないという現実を受けて、そもそも人工知能って何？や、今どのように利

注1　http://efsta.com/

注2　https://efsta.connpass.com/event/45834/

◆**図1 「エフスタ!!」活動理念**



用され、今後どうなっていくのかを学び、さらに福島ではどのように活用できるかを参加者でディスカッションしました。

参加者には、エンジニアだけでなく人工知能に興味のあるさまざまな職種の方がおり、東京からはもちろん福島からもたくさん訪れていました。

いまさら聞けない人工知能

筆者は普段、Yahoo! JAPANの中で、先端技術を調査し、先端技術が会社に与える影響を経営層にインプットしたり、人工知能などの技術を一般の社員にも理解できるように教育を行っています。今回の「エフスタ!! TOKYO」では、人工知能がどういったものかわからない参加者の人たちに向けて、最初の講演で「いまさら聞けない人工知能」と題して筆者よりお話しましたので、その内容をお伝えします。

まず、人工知能の歴史的なところからおさらいしました。今の人工知能は3度目のブームと位置づけられ、過去2度のブームは人工知能の限界が垣間見えた結果、失望されて世間から注目を集めなくなった経緯と、そうしたことで研究者の数自体が少ない背景があります。注目を集めていなかった人工知能関連の研究から、従来限界とされていた精度を大きく上回るDeep Learningといった手法が生まれたため、世の中に大きな衝撃を与えました。しかし、そうした手法は発達した計算機リソースにより、従来の人工知能関連の理論が(特定用途向けとはいえ)現実的になったということであり、理論自体は非常に歴史あるものが現在適用されていると言えます。現在の人工知能ブームから研究者は増えていく傾向にありますが、今の人工知能関連の技術が世の中で適用可能な領域は限定的です。人工知能の適用可能な領域が広がるような新たな理論が提唱され、市場に適用されるにはまだ時間がかかると思われる点、また、大量のデータが必要なアプローチも多いため、十分な研究を行うための環境自体が、大量のデータが集まるインターネット企業などに偏ってしまうという課題があります。

そうした歴史的な背景や現在の状況を踏まえたうえで、人工知能関連の技術の中でも基本的な“機械学習の教師あり学習”を、男性と女性を人工知能が自動的に分類するという問題に照らして、予備知識がない方でもイメージが湧くように解説しました。

人間は男性と女性を自然と見分けることができていると思います。おそらくは無意識のうちに身長や骨格、声の高さや髪の長さといった特徴から性別を判断しているはずです。こうした判断を機械学習で実現する場合、このような特徴を「素性」と呼び、その重要度に応じて重み付けを行います。たとえば、「骨格」ががっしりしている人は男性らしさが強く現れていると言えるので、男性と判断するうえで重要なので重みを高くつけます。逆に「声の高さ」は女性らしさが強く現れると言えるので重みを逆に設定するような形です。

表1では3人の人物の「身長」「骨格」「声の高さ」「髪の長さ」それらの重みとの掛け合わせの和を「性別スコア」としています。この「性別スコア」が高ければ高いほど男性らしいと言え、逆に低ければ低いほど女性らしいという形で現れます。このスコアで男性と女性が分かれる境界線を引いて、機械が自動で男性か女性かを判定するのが人工知能関連技術の中の機械学習の簡単な考え方となります。また、実際には大量の男性のデータ、女性のデータが必要で、その中から特徴を見つけ出し、機械学習の素性として設定していく必要があります。近年ではDeep Learningなどの手法で、この素性を発見する部分を大量のデータから自動的に発見することもできるようになりつつあります。

さて、ここで1つ問題が出てきます。**表1**の「ひ

◆**表1　男女を分類する機械学習の考え方の説明**

名前	身長(cm)(重み：1)	骨格(g)(重み：3)	声の高さ(Hz)(重み：－3)	髪の長さ(cm)(重み：－2)	性別スコア
たろう氏	180cm×1	3,000g×3	300Hz×-3	10cm×-2	8260
ひかる氏	150cm×1	2,000g×3	400Hz×-3	60cm×-2	4830
はなこ氏	150cm×1	1,600g×3	800Hz×-3	80cm×-2	2390

かる氏」は身長が低く、髪の毛が長く、骨格も華奢な数字で女性らしさを感じる一方で、声は低く男性らしさも感じさせます。人間でもこの**表1**の数字だけで男性か女性かを判断するのは難しくないでしょうか？　そのため「性別スコア」が「たろう氏」や「はなこ氏」と比較しても、中間に位置しており、男性なのか女性なのかスコアで機械が自動的に判断するのも難しいと言えます。人工知能について理解するうえで、人間でも判断するのが難しいものは、基本的に人工知能でも判断させることは難しいことに加え、人間が判断を誤るように人工知能も誤判定を起こすものだというのを、前提条件として知っておく必要があります。

もう1つ、現在の人工知能がどういったものかを正しく理解してもらうためには「強い人工知能」と「弱い人工知能」という概念をお伝えしました。「強い人工知能」とは「ドラえもん」や「鉄腕アトム」、映画ターミネーターの「スカイネット」などのように、自ら思考する知能を持った人工知能を指します。もう一方の「弱い人工知能」とは、かなり限定的な機能に閉じた形で、人間が行うような処理を機械が代行できるというもので、知能があるわけではありません。現在、世の中を賑わせている人工知能とは後者の「弱い人工知能」と呼ばれるもので、たとえば、囲碁に強い人工知能は男性と女性を見分けるといったことはできません。人工知能に仕事が奪われるといった話から、今、世の中で人工知能と言うと前者の「強い人工知能」を思い浮かべる人が多いように思いますが、実態としては非常に限定的なものだということです。

このような説明から、今話題の人工知能と呼ばれているものができること、できないことの前提条件を持ってもらい、何だか得体の知れないままブームとなっている人工知能というキーワードに翻弄されないようにと、参加者の皆さんに伝えました。

近未来の人工知能のカタチ

続いての講演では、福島県出身で米国ミネソタ大学大学院で機械学習を専攻し、2006年に人工知能を使ったニュース推薦で起業されてからは、人工知能関連の業務を幅広く行うフルスタックAIアーキテクトとして活動されている遠藤太一郎さんに登壇いただきました。「近未来の人工知能のカタチ」と題して、近未来において現在の人工知能がどう発展し、社会にどう組み込まれていくかを今あるテクノロジーをベースに解説する講演です（**写真1**）。

今の人工知能の基本的な処理は、何かしらの「入力」を受けて、人間が「思考」し判断する部分を機械で自動化し、人間の「行動」につなげる、という流れであることが説明されました。とくに「思考」の部分が先ほどの「弱い人工知能」が担う“ある処理にだけ特化した人工知能”によって行われるといった、既存の人工知能技術の根底にある流れを解説していただきました。

とくにセンサーの進化とスマートフォンなどに搭載されるカメラやマイク性能の向上が、「入力」の部分として人工知能関連技術で必要とされる大量のデータを生み出し、それを一定の処理にだけ特化した人工知能によって、人間が短い時間で判断するようなところを機械が高速に年中無休で行うように置き換わっていくというお話でした。センサーやカメラ、マイクが人間の感覚器官である目や耳、口といった役割を果たすことで、自動運転技術やロボット技術の発展につながっており、IoTの時代になると今まで以上にそうした取り組みが増え、カンブリア大爆発のような形でいろいろな姿で人々の前にサービスが提供される未来を予測されています。

まとめとして、既存のある処理にだけ特化した人工知能の本質は「予測」とそれに伴う「最適化」で、基

◆**写真1　人工知能の処理の流れを解説する遠藤さん**

本的にデータありきだという点、また、センサーやカメラ、マイクとDeep Learningが組み合わさることで、人間が目や耳などを使って判断するようなところが人工知能に置き換わっていく近未来を予測されました。今の人工知能がデータありきな構造という点で、そうしたビックデータを生み出すIoTやVR、ARやMR、ロボットなどは、バラバラなように見えて実は人工知能と密接に関連しているということを伝えて講演を終えられました。

人工知能と福島！

2つの講演を終えて、会場の参加者も含んだ形でパネルディスカッションが行われました。今、巷を賑わせている“人間を超えた人工知能が生まれる点”を指すシンギュラリティは来るのか？といった質問や、人工知能に仕事を奪われるとどうしたら良いか？といった疑問に対して、遠藤さんと筆者が回答していくといったものでした。

しかし、ここまで記事をご覧いただいた読者の皆さんは、シンギュラリティが来るのは相当先のことや、仕事が奪われるようなところは人間の目や耳を使って判断できるようなところ、かつ大量のデータが存在し、自動化されてしまうようなところだというのは何となくおわかりいただけたかと思います。昨今のニュースなどで取り上げられる人工知能は過剰に危機感があおられていますが、正しい知識を持つことで不要な心配を抱かずに済むことになります。

エフスタ!! 恒例のLT大会

今回の「エフスタ!! TOKYO」はテーマが話題の人工知能に関する解説が中心となったため、福島県のお話自体は最後のLT大会に集中しました。その中でも紹介したいのは以前の記事でも紹介された「エフスタ!!」のスタッフでもある山中英治さんによる「1/10 Fukushimaをきいてみる[注3]」という福島の今を切り取って伝えるドキュメンタリー映画の紹介LTです。

震災から数年が経ち、福島県外に住んでいる人達に今、福島がどうなっているかという情報は伝わってこなくなりました。福島原発の除染や子育ての話、福島での食事の話などについて、福島に住む知人などが居なければ知ることもできません。このドキュメンタリーでは福島の現地に住む人の声を毎年1年かけて集め、1年単位で公開していくスタイルで、10年続けることを目標に監督の古波津陽さん、出演は福島県出身の女優である佐藤みゆきさん、撮影が柏崎佑介さんといった3人で2013年から始まった企画として紹介されました。山中さんもこの作品の紹介を「エフスタ!! TOKYO」が毎年続く限り、紹介し続けると表明して、LTを終えられました。

明るく福島の未来のために活動する「エフスタ!!」も長い時間をかけて、コミュニティとしても大きく育っています。勉強会という形を取りながらも、地域とITコミュニティを構築していく取り組みはほかのコミュニティにおいても、参考になるところは多いと思います。また、こうして福島を飛び出して活躍する人たちとともにこれから長い時間をかけて復興していく福島を、皆さんもどうぞ応援してください。SD

注3 http://fukushima-ask.info/

温故知新
IT むかしばなし

第63回

ハードディスク容量の壁

速水 祐(はやみ ゆう)　http://zob.club/　twitter @yyhayami

はじめに

コンピュータの記憶装置の代表であるハードディスク(以後HDD)は、過去にいくつかの容量の壁(上限)が存在し、そのたびにユーザの混乱を招いてきました。技術が進歩し、容量の大きなHDDが手に届く価格になったにもかかわらず、さまざまな問題により、すべての記録エリアが使えない状況が起こっていたのです。将来の技術の進歩を予測し、先のことを考えた規格でなかったことと、HDDの容量増加の速度が想定以上に速かったことがその原因でした。今回は、このハードディスクの容量の壁のお話をしましょう。

HDDの容量の壁の要因とその構造

HDDは、**図1**のような構造になっています。プラッタと呼ばれる円盤が1枚以上あり、データの記録にはプラッタの表裏を使います。磁気ヘッド(Head)も表裏にあり、データを読み書きする際にはプラッタの外周から内周に向かって移動します。

磁気ヘッドが移動することでアクセスする複数のトラックの位置を決め、プラッタの回転により、指定したディスクの最小単位のセクタ(Sector)にアクセスできるわけです。

別のプラッタ上の同じ半径のトラックをまとめてシリンダ(Cylinder)と呼びます。HDDの内部では、シリンダ(C)と磁気ヘッドナンバー(H)、およびセクタ(S)の3つのパラメータによって指定された、512バイトのサイズのセクタにアクセスするようになっています。

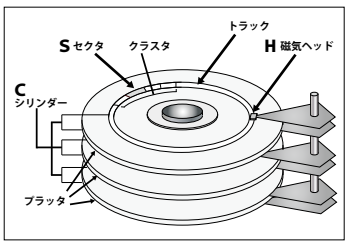
32MB/40MBの壁

1983年に登場したIBM-PC/XTに、10MBのHDDが搭載された機種が発売されました。次の年の1984年10月には、日本でもやはり10MBのHDDが載ったPC-9801F3が発売されました。当時は、そのころ主流だったフロッピーディスクに比べて、HDDの容量の大きさとアクセススピードには圧倒されましたが、その価格(当時758,000円)を見るととても手が出せない機種でした。その後、HDDを利用するIBM PCのユーザが増え、普及が進みましたが、日本では、そこまでの容量とアクセススピードは一般には必要とされず、それよりもフロッピーディスクの容量やスピードアップの方向に進んでいきました。

IBM-PC/XT は、ST-506というHDDに実装されたインターフェースをXTバスで直接アクセスするローレベルな方法を採用し、それをMS-DOS 2.0から利用する方式をとっていました。ここに最初の壁が発生しました。MS-DOS2.0は、セクタのカウントを16ビットで管理するため、32MB[注1]が容量の壁になっていました(**図2**)。1984年にIBM-ATが登場し、MS-DOSが3.0が出てきてもその状況は変わらず、1986年のIDE(後述)の登場

注1) セクタの基本サイズは512バイト。512 × 65,536 = 33,554,432バイト= 32,768KiB(1Kib = 1,024バイト)= 32MiB(1MiB = 1,024KiB)。本稿中では、2進接頭辞で容量の数値は表現しており、KB/MBは、KiBとMiBとして使っています。

▼図1　HDDの構造



を待つことになります。

PC-9801では価格の壁が立ちふさがり、HDDが普及し始めたのは1987年頃です。サードパーティのICM社や緑電子社などから普及価格[注2]の20MBと40MBのHDDが発売され転機となりました。PC-9801のインターフェースは、SCSI(Small Computer System Interface)の前身であるSASI(Shugart Associates System Interface)が使われており、容量の限界は40MBで、PC-98のBIOSも40MBまでしか使えない仕様になっていました。

当時のMS-DOSのバージョンでは、40MBの領域の確保も可能でしたが、SASI BIOSにおける40MB壁を越えることはできません。しかし、1988年にNECからSASIを進歩させたSCSIインターフェースボードであるPC-9801-55が発売され、それに追従してサードパーティからも安価なインターフェースボードつきの80MBのHDDも発売されることになります。SCSIは、連続的にセクタを管理する32ビット長のLBA(Logical Block Addressing)が使われ、BIOSもそれをサポートしており、当時としては容量の壁はもう将来には存在しないと思われました。

504MB/544MBの壁

1986年、IBM-PC/AT用のHDDでは、ST-506をインテリジェント化したIDE(Integrated Drive Electronics)が発表され、その後PCのHDDの主流となりました。PC-98シリーズでも、1992年に登場したPC-9821シリーズでIDEがサポートされ、インターフェースボードを必要とせず、安価なIDE HDDが広く使われるようになってきました。当時は、Windowsの黎明期で、ファイル数が増大し大きなディスクのエリアが必要になってきた時代です。そこで新たな壁が生まれます。

PC/ATのディスクBIOSのパラメータは、16ビットのレジスタに、シリンダ(C)の10ビットとセクタ(S)の6ビットをセットし、磁気ヘッドナンバー(H)は別の8ビットレジスタにセットするため255まで指定できます。しかし、IDEコントローラが扱える磁気ヘッドナンバーの最大値は16までですので、BIOSの限界と合わせると504MB[注3]が限界となりIDE HDDの容量の壁として大きな問題になりました。それを解決したのが1994年に登場したE-IDE(Enhanced IDE)で、C、H、Sのデータは扱わずに28ビット長のLBAを使用し、BIOSもこれをサポートするように修正され、高速化も実現していました。初期のPC-9821でも似た問題が生じ、BIOSの問題から544MB[注4]の壁が存在しました。

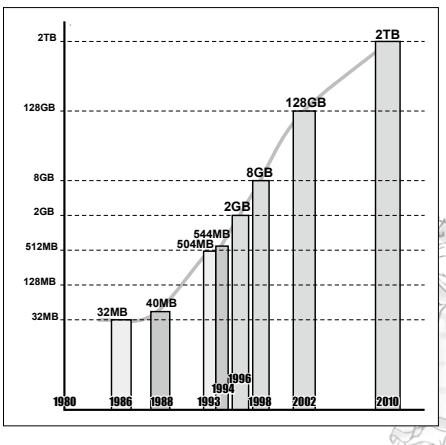
2GB/8GB/128GBの壁

E-IDEとLBAですべて解決したかに見えた壁は、そのあとも出現しました。1996年前後に発生したFAT16の問題から2GBの壁、98年にはBIOSの内部のパラメータの扱いから生じた8GBの壁がありました。21世紀になった2002年には、E-IDEの28ビット長のLBA[注5]の限界が表れて128GBの壁が発生しました。

最近の2TBの壁

そして2010年、遂にLBAの32ビットの壁にHDDの容量が届きます。1980年後半の最大80MBを使っていた時代では無限に思われた容量に達しました。それがMBRの管理の問題も含んだ2TBの高い壁であり、80年代からは26,000倍以上の容量になっていたのです。SD

▼図2　HDD容量と壁の変遷図



注2) とは言っても、40MBで188,000円程度。

注3) 512×1,024×63×16＝33,554, 432バイト＝504MiB。

注4) 1シリンダーあたり17セクタ固定とし、シリンダーの指定は16ビット長の値であるため。512×65,536×1×17＝33,554,432バイト＝544 MiB

注5) $512 \times 2^{28} = 128\text{GiB}$。

Software

グレープシティ、「SPREAD for ASP.NET 10.0J」発表

グレープシティ㈱は2月15日、ASP.NETアプリケーション開発用表計算グリッドコンポーネント「SPREAD for ASP.NET 10.0J」を発売した。

SPREAD for ASP.NETは、WebアプリにExcelと互換性のある表計算データグリッドを実装できるASP.NET用コンポーネント。データの一覧表示のほか、ソートやフィルタリング、列のドラッグ移動といったデータ操作や、チャート、条件付き書式、スパークライン、グループ化を使用したデータの可視化など、Excelで利用される頻度が高い機能が搭載される。これらの機能はExcelとの互換性を保つため、Excelファイルへのエクスポートやインポートにも対応している。また、数式の計算、行の追加や削除、ソート、フィルタリングといったExcel互換の基本操作をコールバックで処理し、ページ全体をポストバックせずに必要なデータだけを更新できる。

新バージョン10.0Jでは、チャートコントロールの提供、SPREADの持つ表計算機能をほかのコントロールに付与するエクステンダの追加など、ユーザエクスペリエンスを向上させる細やかな機能追加が行われたほか、iPhoneでの利用も可能になる。価格は、1開発ライセンスが172,800円（税込）。

CONTACT
グレープシティ㈱ URL http://www.grapecity.com

Service

アイレット、インフォテリア、協業とデータ分析基盤構築サービスを提供

アイレット㈱とインフォテリア㈱は1月23日、「データ分析基盤構築サービス」の協業開始について発表した。

今回両社は、アイレット㈱のAWS専業のクラウドインテグレータ「cloudpack」と、インフォテリア㈱が提供するデータ変換ツール「ASTERIA」を組み合わせた「データ分析基盤構築サービス」を開始する。これはデータウェアハウスとしてAmazonのRedShiftを使用し、ASTERIA WARP Coreから導入された莫大なデータを、BIツールであるAmazon QuickSightなどで視覚化するというもの。両社の優位ポイントが相互に組み合わされた、最小限の初期投資でビジネスに必要なデータ分析を推進できるソリューションである。

▲左から、インフォテリア㈱ASTERIA事業本部長 熊谷晋氏、同社代表取締役社長 平野洋一郎氏、アイレット㈱代表取締役 斉藤将平氏、同社執行役員 後藤和貴氏

CONTACT
アイレット㈱ URL http://www.iret.co.jp
インフォテリア㈱ URL http://www.infoteria.com

Topic

トレンドマイクロ、「2016年国内サイバー犯罪動向」を発表

トレンドマイクロ㈱は1月10日、2016年1月～11月までに日本国内で観測された脅威情報や統計データを元に分析した「2016年国内サイバー犯罪動向」を発表した。おもなトピックは次の3つ。

- **過去最大の国内ランサムウェア被害を確認**
  2016年1月～11月のランサムウェアの国内被害報告件数は2,690件に上り、前期比で約3.4倍に増加。ランサムウェアの国内検出台数も同期間で58,400台となり、前期比で約8.7倍に増加した
- **オンライン銀行詐欺ツールの国内検出台数が過去最大**
  2016年1月～11月のオンライン銀行詐欺ツールの国内検出台数は過去最大の98,000台に達し、前期比で約3.4倍に増加。情報窃取を狙う対象の金融機関は、都市銀行や地方銀行・信用金庫の利用者など幅広い
- **標的型サイバー攻撃の公表被害件数は低下、水面下で攻撃は継続中**
  標的型サイバー攻撃と思われる国内の公表被害例は、前期比で23件→7件と減少したものの、標的型サイバー攻撃の疑いのある不審な通信は、常に月10万件以上確認されている

CONTACT
トレンドマイクロ㈱ URL http://www.trendmicro.co.jp

Event

関西フロントエンドUG主催、「FRONTEND CONFERENCE 2017」が3月18日に開催決定

“Webで働くすべての人のために”がキャッチフレーズのイベント「FRONTEND CONFERENCE 2017」が3月18日に開催される。一般参加受付は2月中旬頃に発表予定（本稿執筆時）。会場はJR大阪駅から徒歩10分ほどにある新梅田研修センター。

2017年のFRONTEND CONFERENCEのテーマは「つながる」。デザイン、コーディング、エンジニアリングにおける技術は日々変化し、それらの中から必要なものを紡ぎ、1つのWebページに「つないでいく」というWeb制作の象徴として。そして、技術と技術、ヒトとヒトとのつながりをイベントを通して見つける、というメッセージが込められている。Web制作の開発現場で活躍する方々によるセッションやハンズオン、Web業界での働き方やキャリアについて意見交換をし合う座談会などが予定されている。また、学生作品コンテストもあり参加者を募集中（3月11日まで）。

●イベント情報

日時	2017年3月18日　10:00～18:00
場所	新梅田研修センター
主催	関西フロントエンドUG
対象者	Webに関心のあるすべての人
定員	300名
ハッシュタグ	#frontkansai
公式HP	http://kfug.jp/frontconf2017

CONTACT
FRONTEND CONFERENCE 2017（Facebookページ）
URL https://www.facebook.com/frontconf2017

Software

レッドハット、デジタルトランスフォーメーションを促進するAPI戦略を発表

レッドハット㈱は1月27日、2016年7月に買収した3SCALE社のAPI管理プラットフォーム「Red Hat 3scale API Management Platform」の国内販売を開始した。

本プラットフォームは、ネットワーク上に公開するAPIにセキュリティとユーザ管理機能を提供するもの。これによってユーザごとの課金も可能になり、マイクロサービスを推進し、デジタルトランスフォーメーションを実現するために必須のソリューションとなるという。米国からシニアディレクターのスティーブン・ウィルモット氏も来日し、技術面での解説も行われた。同氏によれば、システムの現代的なアジャイル型統合は、分散システムの連携とコンテナ（OpenShift）とAPIの3本柱で成り立っており、DevOpsによる自動化と安定運用を支えるものになっているという。またキャンベル社やオランダのスキポール空港のシステムで、すでに本プラットフォームが導入されている事例を紹介し、その優位性を示した。


▲レッドハット㈱プロダクト・ソリューション本部　本部長 岡下浩明氏

CONTACT
レッドハット㈱　URL https://www.redhat.com/ja/global/japan

Software

ビーブレイクシステムズ、ERPパッケージ「MA-EYES」が「駅すぱあとWebサービス」と連携

㈱ビーブレイクシステムズは、統合型基幹業務パッケージ「MA-EYES」と㈱ヴァル研究所の経路検索API「駅すぱあとWebサービス」との連携対応を行った。

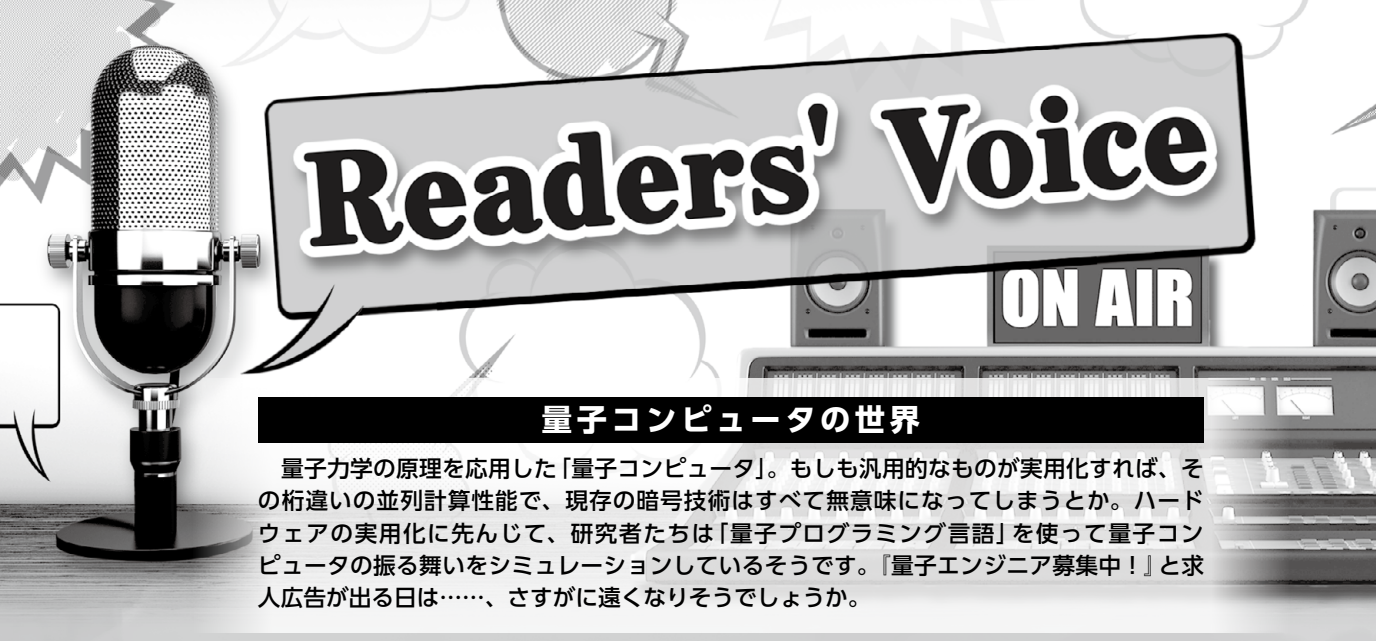
MA-EYESは、プロジェクト管理、作業実績・勤怠管理、購買・経費、グループウェアなど企業経営に必要な多くの機能を備えたERP（統合型基幹業務パッケージ）。

今回、同製品が「駅すぱあとWebサービス」と連携できるようになった。これにより公共交通乗換案内サービス「駅すぱあと」の経路検索機能や公共交通機関のデータをMA-EYESから利用できる。

従来は、申請者が旅費交通費精算時に経路や金額を調べ、その内容をMA-EYESの経費申請機能に入力して、その後、管理部門が内容をチェックしていた。

それが今回の連携により、「駅すぱあとWebサービス」利用時には、経路検索・選択した情報をMA-EYESの経費申請入力項目として自動で読み取りできるようになった。MA-EYESにて経費申請が承認されると、経費明細データは仕訳データに自動変換される。また、あらかじめ定期区間を設定することで、定期利用時の差額運賃を自動計算することもできる。

CONTACT
㈱ビーブレイクシステムズ
URL http://www.bbreak.co.jp/maeyes



量子コンピュータの世界

量子力学の原理を応用した「量子コンピュータ」。もしも汎用的なものが実用化すれば、その桁違いの並列計算性能で、現存の暗号技術はすべて無意味になってしまうとか。ハードウェアの実用化に先んじて、研究者たちは「量子プログラミング言語」を使って量子コンピュータの振る舞いをシミュレーションしているそうです。『量子エンジニア募集中！』と求人広告が出る日は……、さすがに遠くなりそうでしょうか。

2017年1月号について、たくさんの声が届きました。

第1特集 シェル30本ノック

シェルスクリプトに精通した「シェル芸勉強会」のメンバーが、シェルの練習問題をシーン別に合計30問出題し、解答を示しました。すべて解き終わるころには、複雑な作業でもワンライナーで記述できるようになっていることでしょう。

業務に活かせる。また、問題形式で解答も丁寧でわかりやすい。
天地丸さん／岐阜県

シェル芸すごいですね。
ゆめかけさん／神奈川県

cshとbashを昔から使ってる自分にはおもしろい特集でした。
Happyさん／宮崎県

シェルスクリプトを勉強する良い機会になりました。面倒臭がらずに覚えれば効率が上がりそうですね。
菅原さん／東京都

シェル30本ノックが、頭の体操とかオプションの再確認とかのきっかけになりました。
牧さん／大阪府

シェル30本ノック、いつもはただ読むだけでしたが、こういう問題形式だと頭や手を動かすので良いですね。
地引さん／茨城県

仕事や作業を何倍も楽にできるシェルを、ご自分の環境でぜひお試しください。ただ、一発でデータをすべて消してしまう危険なシェルもありますので、くれぐれもご注意を。

第2特集 機械学習をどう学ぶべきか？

人工知能、データサイエンスの分野でとくに注目される技術「機械学習」について、どのような数学的な手法が利用されているのか、どのように学べば良いかを、エンジニア向けに解説する特集でした。

機械学習には非常に興味があり、業務で実際に活かす方法を検討している。今後も同分野の記事を期待している。
鈴木さん／埼玉県

難しそうでなかなか手が出せなかった機械学習ですが、少し理解できたような気がします。
若杉達郎さん／石川県

機械学習を勉強するのにどこから手を付けるかヒントになったと思います。
かずさん／千葉県

あらためて、事象に対する数学的な観察、モデル化の重要性を痛感しました。
鈴木さん／熊本県

気になっていた機械学習について、基本的なところから勉強できてよかったと思う。
とろとろ。さん／東京都

昨年くらいから職場でも機械学習をよく耳にします。この特集で大枠は把握できました。関数解析、線形代数、確率統計を勉強しなおす所存です。
花子さん／大阪府

機械学習に興味を持っている読者の方は非常に多く、どこから手を付けるべきかを学べてよかったとの声が多く寄せられました。記事によると、やはり数学が要とのことでした。

第3特集　エンジニアが採用できない会社と評価されないエンジニア【前編】

ITエンジニアの「採用」「評価」をテーマにした小説テイストの特別企画。前編では、優秀なITエンジニアを採用できないという問題を抱えた企業の仕事を請け負った、人材コンサルティングの新米社員が主人公です。

転職して半年経ちましたが、新しい職

1月号のプレゼント当選者は、次の皆さまです

①ディレクターズ10周年記念レディースバッグ（黒）
nm様（東京都）

②Acronis True Image 2017
伊集院正敏様（東京都）、ももんが様（静岡県）、永作肇様（東京都）

③Qiitaノベルティ Tシャツ
長谷川陽平様（茨城県）

④『レガシーソフトウェア改善ガイド』
斉藤竜則様（東京都）、瑞慶山薫様（沖縄県）

⑤『実践力を身につけるPythonの教科書』
大久保宏様（広島県）、川上拓真様（愛知県）

⑥『ヤマハルーター実践ガイド』
前田尚希様（愛知県）、葉山明寛様（東京都）

⑦『はじめての深層学習プログラミング』
星翔太様（東京都）、中田憲吾様（千葉県）

※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。

場は人の入れ替わりが激しく、評価制度などもようやく作り始めたところで、すごくタイムリーな記事でした。続きが楽しみです！ binaさん／東京都

ちょうど転職活動をしている最中なので、続きが非常に気になります……！ ginさん／愛知県

プログラミングが必須の時代においては社長も人事もエンジニア思考が前提となるわけで、今後はすべてが適切に評価されることにつながっていくことになるのかもしれませんし、エンジニアはごまかすことができなくなりますね。 とんぴさん／沖縄県

物語形式の本記事でしたが、読者の声をみるとかなりリアリティがある内容のようです。実際に転職／採用活動中との読者の方からも、共感の声が寄せられました。

一般記事　Ejectコマンドで遊んでみませんか？

リムーバブルメディアをドライブから取り出すコマンド「Eject」で、車を走らせたり除夜の鐘を鳴らしたりする年末特別企画。「Ejectコマンドユーザー会」の主催者が、"非"実用的な使い方を教えます。

本職じゃないので、サーバールームでぜひやりたい。 トトロンさん／大阪府

物理スイッチとして光学ドライブを使う方法は、目から鱗でした。 小野さん／東京都

昔、似たようなことをやっていました。おもちゃのボーリングレーンで玉を打ってボーリングとか。 Romeosheartさん／長崎県

Ejectコマンドで工作など考えたことがありませんでした。光学ドライブで工作遊びなどたいへん楽しいですね。遊び心をくすぐるような記事にも期待します。 Qkobさん／富山県

「初めて知った」「同じことをやっていた」などいろいろな声が寄せられました。おもしろ記事ではありましたが、OSの知られざる機能の勉強になったのではないでしょうか。

短期連載　Jamesのセキュリティレッスン【8】

新しいファイル形式「pcap-ng」も使えるようになったパケットキャプチャのツール「Wireshark」の使い方を紹介する短期連載です。今回は、無線LANの暗号化通信を復号する術を解説しました。

機会があれば実験をしてみたいです。 dannaさん／東京都

毎回、パケット関連についての高度な内容で参考になる。 psiさん／東京都

Wiresharkにそのような機能があったとは知りませんでした。 sho-hさん／島根県

無線LANをWiresharkを使って復号する方法の勉強ができました。トラブルとかでも役に立ちますね。 猫魂さん／京都府

トラブル時、記事のように、パケットを収集してトラブルシューティング、とはなかなかいかないかもしれませんが、無線LANの通信について知識を持っておくことは無駄ではありません。

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
April 2017

2017年4月号
定価（本体1,220円+税）
184ページ

3月18日発売

［第1特集］新人歓迎企画【第1弾】目的から考えろ!

現場で動けるA君、できないB君のLinux入門（OS操作編）（仮）
Webサーバを構築できますか?

［第2特集］AWS LambdaとAzure Functionsで体験

はじめてのサーバーレスアーキテクチャ

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

休載のお知らせ

「Androidで広がるエンジニアの愉しみ」（第14回）は都合によりお休みさせていただきます。

お詫びと訂正

以下の記事に誤りがございました。読者のみなさま、および関係者の方々にご迷惑をおかけしたことをお詫び申し上げます。

■2017年1月号
●P.165　連載「Linuxカーネル観光ガイド」リスト1中10行目
［誤］present := (desc & 1) != 0　［正］present := (desc & 0x8000000000000000) != 0
●P.165　連載「Linuxカーネル観光ガイド」リスト1中14行目
［誤］return desc & pfnMask　［正］return desc & 0x7fffffffffffffff

SD Staff Room

●去年の代表的なテーマは機械学習でしょう。今年もその流れは変わらないと考えています。ITだけでなく物理・数学が必要な時代になってきています。ITエンジニアはとても大変ですが、本誌も皆さんの助けになるような特集を企画し、お役に立ちたいと考えております。（本）

●十数年ぶりにインフルエンザに罹患し、40度近い熱が出て会社を休んだ。思えば、年末に風邪で2日間寝込み、大晦日に自転車で車止めに激突して左腕打撲、初詣の帰りに転倒して左手指擦傷の後の駄目押しな出来事だった。これで今期の厄は全部使ったので、今後心配ないと思いたい今日このごろである。（幕）

●小誌のコーディングに関する人気記事を1冊の本にまとめました。『プロになるなら身につけたい プログラマのコーディング基礎力』です。小誌を定期購読されている方にはご不要ではありますが、興味のある方はお手にとってみてください。新人さんにお勧めする一冊としてもよろしいのではないかと。（キ）

●毎月末金曜日は15時に仕事を終えることを推奨する「プレミアムフライデー」が2月24日から始まるというのは本当でしょうか?　弊社はまったく実施する気配がありません。というか、会社が実施したとしても、当編集部は月末は繁忙期です。第2金曜日くらいに設定してくれれば良かったものを。（よし）

●TVでみかけてずっと気になっていた「手羽先餃子」を作りました。鶏の手羽先の骨を抜いて、空いた空間に餃子の餡を詰めて焼くという料理なのですが、欲張って餡をパンッパンッに詰めたので、焼いたときに中身が外へ吐き出される事態に……。皮と具をほぼ別々に食べるという虚しい形になりました。（な）

●ちびぬいで企画展参加や子どもたちの卒業を間近に控えバタバタな日々です。そんなときに限って7年前から使ってるガラケーが使用不能に。とりあえず修理に出してみたけれど直ってくるのかしら……電話が使えないお下がりiPhoneをやめてこの際スマホ1本にするか、まだまだガラケーでいくのか、悩ましい〜。（ま）

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［E-mail］
sd@gihyo.co.jp

［FAX］
03-3513-6179

※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

Software Design
2017年3月号

発行日
2017年2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。