Special Feature 1 基本のテキスト処理
Special Feature 2 Sambaサーバ構築

Software Design

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

2015年5月18日発行
毎月1回18日発行
通巻361号
（発刊295号）
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価 本体 1,300円 +税

2015 May 5

2号連続
増ページ&
特別付録
小冊子

Special Feature 1

UNIXのコツ シェル、sed、AWK、grep

テキスト処理ベーシックレッスン

手を動かしてデータを操ろう！

Special Feature 2

ファイル共有自由自在

[徹底入門] 最新・Sambaの教科書

特別付録

『3分間HTTP&メールプロトコル基礎講座』より

まるごと1章分収録！
Webのしくみを
基礎の基礎から
マスター

OSとネットワーク、
IT環境を支えるエンジニアの総合誌

Software Design

毎月18日発売

PDF電子版
Gihyo Digital
Publishingにて
販売開始

年間定期購読と電子版販売のご案内

1年購読（12回）

14,880円（税込み、送料無料）1冊あたり1,240円（6%割引）

PDF電子版の購入については

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

をご覧ください。
PDF電子版の年間定期購読も受け付けております。

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

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

Fujisan.co.jpからのお申し込み方法

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

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

2015
May
5
Software Design

第1特集

UNIXのコツ

シェル、sed、AWK、grep

テキスト処理ベーシックレッスン

手を動かしてデータを操ろう!

contents

2015
May
5
Software Design

contents

第2特集

[徹底入門] ファイル共有自由自在 最新・Sambaの教科書

特別付録

短期連載

Catch up trends in engineering

Inside View

アラカルト

Column

Development

OS/Network

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

ISBN978-4-7741-7256-9
A5判／272ページ
定価（本体2380円+税）

アプリビジネス成功への法務戦略

開発・リリース・運用に必要な法律知識

●GVA法律事務所　著
山本俊／本間由美子／橘大地
藤江大輔／野本遼平／飛岡依織

App StoreやGoogle Playといったアプリマーケットの発達により、ベンチャー企業のような独自のチャネルを持たない企業でも、全国・全世界に向けてアプリをリリースすることが容易に可能になりました。こうした中、アプリ内の仮想通貨管理、全世界のユーザを相手とする多数取引の契約整備、未成年者を含むユーザとの取引、ビジネスモデルの適法性判断といった観点からの法務戦略への対処の動きは遅れています。本書では、アプリビジネス運営で問題となる法規制について解説した上で、どのように法務リスクをヘッジすべきかをわかりやすく解説します。

ISBN978-4-7741-7295-8
B5変形判／352ページ
定価（本体2880円+税）

No.1スクール講師陣による

世界一受けたいiPhoneアプリ開発の授業

iOS 8 & Xcode 6 & Swift対応

●RainbowApps講師
桑村治良／我妻幸長／高橋良輔／七島偉之　著

iPhoneアプリ作りにチャレンジしてみたいけど、プログラミング経験がなくても大丈夫？本書は、そんな疑問にこたえます！

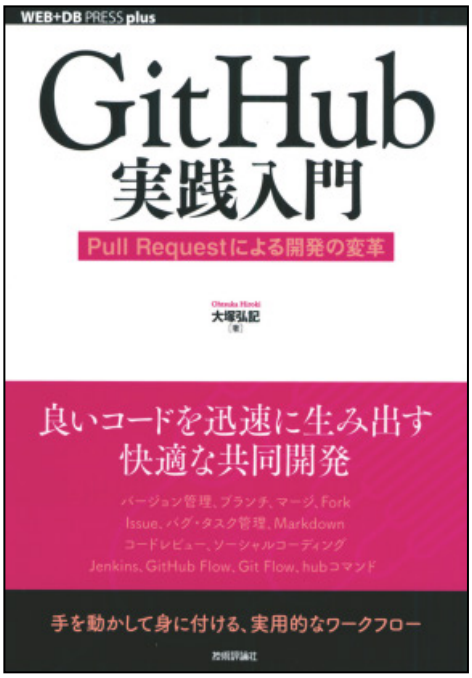
本邦初！アプリ開発の専門スクール「RainbowApps」講師陣の手によるやさしい学習書。
実際にアプリを作りながら、楽しくあきずにスキルアップ。
話題のプログラミング言語「Swift」に対応しています。

RainbowAppsは、数多くのアプリ開発未経験者に教えてきました。そのノウハウがぎっしりと詰め込まれています。

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

翔泳社主催
ITエンジニアに読んでほしい！技術書・ビジネス書大賞2015

技術書部門大賞受賞!!



GitHub 実践入門

Pull Requestによる開発の変革

大塚弘記　著

GitHubの実践的な使い方を、実際に手を動かす形で解説する書籍です。初学者の方にもわかりやすいよう、基本的なGitやGitHubの使い方から、「ソーシャルコーディング」の目玉機能であるPull Requestの送り方・受け方まで解説します。また、外部ツールとの連携、GitHub FlowやGit Flowなど、GitHubを中心とした開発手法についてもしっかり解説しているので、中・上級者の方にも参考になるはずです。

ISBN978-4-7741-6366-6　A5判／304ページ　定価（本体2580円+税）

ビジネス書部門ベスト3入賞!

ISBN978-4-7741-6522-6
四六判／216ページ
定価（本体1680円+税）

システムインテグレーション崩壊

これからSIerはどう生き残ればいいか?

斎藤昌義　著

国内の需要は先行き不透明。
案件の規模は縮小の一途。
単価が下落するばかり。
クラウドの登場で迫られるビジネスモデルの変革・・・

工数で見積もりする一方で、納期と完成の責任を負わされるシステムインテグレーションの限界がかつてないほど叫ばれる今、システムインテグレーターはこれからどのように変わっていくべきか？
日本IBMの営業を経て、数多くの企業にコンサルティングを行う著者が、豊富な図解とともに現状とあるべき姿を解説します。

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

技術評論社の本が電子版で読める！

電子版の最新リストは

Gihyo Digital Publishingの

サイトにて確認できます。

https://gihyo.jp/dp

GIHYO Digital Publishing

amazon.co.jp

Google play

楽天 Rakuten KC　kobo

法人などまとめてのご購入については
別途お問い合わせください。

■お問い合わせ
〒162-0846
新宿区市谷左内町21-13
株式会社技術評論社　クロスメディア事業部
TEL：03-3513-6180
メール：gdp@gihyo.co.jp

第77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。

HTTP/2

HTTPが16年ぶりに大幅アップデート

現在のインターネットにおける標準的な通信プロトコルであるHTTP/1.1が、RFC2616として正式に承認されたのは1999年のことです。それからおよそ16年を経た2015年2月、次世代の標準となる「HTTP/2」が標準化団体のIETF（Internet Engineering Task Force）によって承認されました。

HTTP/2の仕様化は2012年にスタートしました[注1]。その背景には、HTTP/1.1のままでは昨今のWebの急速な変化に対応しきれなくなったという事情があります。とくに、Webコンテンツの大容量化による通信量の増加や、Web体験の多様化、クラウドやモバイル端末の普及にともなう常時接続の一般化などは、1999年当時に想定されていたレベルをはるかに上回っています。そこで、通信の最適化・高速化を目指してGoogleが提唱した新技術の「SPDY」を皮切りに、次期HTTPの策定に向けた議論が一気に加速しました。

HTTP/2では、HTTP/1.1との互換性を確保しつつ、通信の高速化やネットワーク帯域の効率的な利用などを実現します。具体的には、おもに次のようなしくみが導入されました。

- 通信は完全に多重化され、1つの接続で並列処理ができる
- 通信内容は固定長のバイナリフレームにエンコードして扱われる
- サーバプッシュ型の通信がサポートされ、クライアントからのリクエストを待たずにレスポンスを送ることができる
- ヘッダを圧縮することでオーバーヘッドを削減する

ヘッダの圧縮には、「HPACK」と呼ばれる新しい圧縮方式が採用されました。HPACKはHTTP/2とは独立して規格化されています。HPACKではあらかじめ定義された番号付きのヘッダ情報に基づいて番号指定で短縮を行ったり、前回送信したヘッダとの差分圧縮を可能にすることなどによって、付加する情報の冗長化を防止します。HPACKによって通信のオーバーヘッドが縮小され、ページの送信の大幅な高速化につながるとのことです。

Web開発への影響は？

Webアプリケーションの開発者の立場で見た場合、HTTP/2が普及することで何が変わるのでしょうか。まず前提として、HTTP/2は当初からHTTP/1.1との互換性に配慮して設計されているため、既存のライブラリやツールがAPIを変更する必要はなく、それを利用する開発者もアプリケーションのコードを変更する必要がありません。つまり既存のWebサイトやWebアプリケーションはそのまま動作させられるということです。

その一方で、HTTP/2の能力を最大限に活かすためには、新しいしくみに対応したライブラリの採用やノウハウの蓄積が必要になります。たとえば通信が多重化できるとはいっても、実際にはHTTP/2の通信のしくみをよく理解していなければ正しい実装ができず、かえって通信効率を下げてしまう可能性があります。サーバプッシュ通信でも、新しいWeb体験につなげるためには試行錯誤が必要になるでしょう。

もっとも、HTTP/2の火付け役となったSPDYには現状で多くのWebブラウザが対応済みであり、すでに通信の多重化を利用できる環境が整っていました。サーバプッシュ通信についても、やはりSPDYでサポートされていたほか、独自の実装によってサポートしているWebサーバやフレームワークがありました。つまり、HTTP/2の目玉となっている新機能は、開発者にとって決して馴染みが薄いものではないということです。

なお、本稿執筆時点ではChromeやFirefox、Internet Explorer 11などのWebブラウザが正式にHTTP/2に対応しています。ただし、いずれもTLSで暗号化された接続でのみHTTP/2通信を利用できる実装になっています。HTTP/2では仕様上は暗号化を強制していませんが、Webブラウザのベンダーとしては独自のセキュリティポリシーに則って暗号化との併用を推奨しているのです。

今後、実際に多くのWebサイトやサービスにおいてHTTP/1.1からHTTP/2への切り替えが完了するまでには、まだしばらくの時間が必要でしょう。その間に、この新しい仕様を活用するための勘所をつかみ、次世代のWebにいち早く対応できるようにすることが重要です。SD

IETF HTTP Working Group
https://httpwg.github.io/

注1）当初の名称はHTTP/2.0でしたが、後にHTTP/2に改められました。

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。

DIGITAL GADGET

Volume 197

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

アートと3Dプリント

3Dプリンタの普及

3Dプリンタの話題が各所で聞かれるようになりました。一過性のブームとは違い、さまざまなところで多様に活用されつつあります。大きなビジネス展開にはまだまだ課題があるかもしれませんが、技術の発展とノウハウの蓄積とによって、確実に素晴らしい事例が出そろってきています。

読者の皆さんにも、実際に3Dプリンタを購入して楽しんでみたり、工房的なところで3Dプリントサービスを試してみたり、オンラインサービスで手軽にデータをアップロードして3Dプリントを試したことがある方も増えているのではないかと思います。

最近では3Dプリントの精度の向上とともに、フルカラーでの彩色、陶器、金属、貴金属といったさまざまな素材で制作が可能になってきています。もちろんそういった高度な3Dプリントが可能なプリンタは、家が建つほど高価ですが、プリントサービスを使えば、大きさや重さなどに応じた手頃な価格で3Dプリントが可能です。さらには、砂糖やチョコレート、パンケーキといった食べられる素材によるプリントも実験的に行われています。

3Dプリントで用いられる三次元ファイルフォーマットは、歴史的経緯もあり、三角形の集合で形状を表現するSTL（Standard Triangulated Language /STereoLithography）形式が多いのですが、これらは形状しか表現できません（一部、独自拡張し、色のデータを持たせるSTL表記方法もあり）。現在は時代のニーズに応え、業界団体がAMF（Additive Manufacturing File Format）という、材質や内部構造も表現できるファイルフォーマットを策定中です。AMFを活用すると1つのオブジェクト内に複数の材質を持つ物体を表現できるのです。

現在の3Dプリント技術の進化は驚くべきもので、実際に動くエンジンを成形したり、3Dプリンタで出力した外装をもった電気自動車や建築物、歯形、臓器模型、さらには宇宙ステーション内で特殊なサイズの道具をプリントアウトしたりと、ありとあらゆる分野に及んでいます。また三次元モデルを作れなくとも、物体を3Dスキャンしたり、モデルデータを購入したり、スマホで撮影して組み合わせて編集したりと、とても身近になってきていることは確かです。

» 複数の椅子デザインを融合した3Dプリント椅子（FormNation社のChairgenicsプロジェクト）

» 3Dプリントで装飾が施された鳩時計（Cuckoo Project）

※本記事で紹介しているものは国内未発表・未発売のものを含んでおります。

3Dプリンタがもたらす自由と造作

3Dプリントを活用した「作品」と呼べるアート的な自由な造作が増えてきました。古くは2005年頃、frontというグループが空間に描いたスケッチから3Dプリント家具を生み出すという「Sketch Furnitureプロジェクト」の頃から、3Dプリンタのアート的要素が期待されるようになりました。3Dプリンタのアート活用はいくつかの種類に分かれてとらえることができます。デジタルメディアを活用したアートに3Dプリンタ技術を取り入れたもの、彫刻などの従来のアートの文脈で3Dプリントの技術を取り入れた新しい表現、3Dプリントがなければできなかった新たなアート表現、といった3種類のアプローチがあります。

メディアアート系の表現

David Bowen氏の「Growth Modeling Device」は、植物が育つ様子を写真ではなく、3Dプリンタで出力した形状で記録していくというプロジェクトです。SHAPES iN PLAYの「infObjects」は、食器に音声波形を3Dプリントで再現。メディア変換を表現したプロジェクトです。

既存のアート作品の3Dプリント利用

Benjamin Dillenburger氏による「Digital Grotesqueプロジェクト」は、3Dプリントによる建築物の表現で、どれだけグロテスクなものが表現できるか?という挑戦でした。Gilles Azzaro氏による3Dプリンタ彫刻「Obama Voice Sculpture」は、オバマ大統領のスピーチの音声波形を彫刻のような物体に出力したものです。Fung Kwok Pan氏の「Fluid Vase」は、ミルクのような流体を一輪挿しとして表現した作品です。

3Dプリントならではの新しい造作

「Open Toysプロジェクト」では、茄子や人参、ズッキーニなどに、3Dプリンタで作られた部品を加えることで、野菜をアート的なオモチャにしてしまうプロジェクトです。茄子が潜水艦になったり、人参のレースカーを作ることができます。低温で融解する素材を使った、子供も安全に使える3Dプリンタも登場してきており、未来の子供達は、オモチャも自分たち自身で作り出したり、オモチャを作ることを楽しむ世代になってくるのかもしれません。

また、このオモチャのアプローチと同様に、安価に購入できる組み立て式の家具と3Dプリンタで作られた部品を組み合わせることによって、オリジナリティのあるアート的家具を作り出しています。これらの活動はIKEA Hackと呼ばれ、作品を紹介し合う専用のサイトも存在します。

http://www.ikeahackers.net

今後の3Dプリンタに期待

さまざまな環境が進化し、充実してきている3Dプリント技術ですが、今後はさらに周りを取り巻くサービスが充実してくることが予想されます。たとえば、より平易に三次元形状を作れるツールや、ネットワーク経由で三次元形状を共同制作できるしくみ、適切な三次元形状を作るためのチェックや修正、最適化のためのツール、三次元データの販売や共有などといった、業界全体のエコシステムが機能し始めていることが実感されます。

たとえば、最近3Dプリンタの世界的大手企業3D Systemsが3DモデルのクラウドサービスTeamPlatform（https://www.teamplatform.com/）を買収しました。TeamPlatformは、3Dモデルデータをオンラインドキュメントとして扱うことができる、3Dモデルデータのためのデータ蓄積プラットフォームです。特徴的なのは、単なるデータの置き場所ではないことです。TeamPlatformは次のような機能を有しています。

- 3Dモデルを作り始めてから完成するまでの作業とデータ受け渡しの流れを考えている
- 完成までの各作業をチェックボック

Sketch Furnitureプロジェクトで造作されたスケッチ家具

Growth Modeling Device
http://www.dwbowen.com/gmdmovie.html

infObjects
http://shapesinplay.com/en/projects/infobjects

Digital Grotesque
http://www.digital-grotesque.com/

Obama Voice Sculpture
http://www.gillesazzaro.com/pages/en/bio.html

Open Toys
http://www.thingiverse.com/thing:554850/

ス、ガントチャートで可視化できる
- 3Dモデルに対して三次元空間の場所を指定し、修正の指摘などのコメントがつけられる
- 3Dモデルデータの履歴管理が充実しており、過去のデータと比較することができる
- さまざまな形式の3DデータをWeb上で表示したり、変換したりできる
- WebブラウザⅠ上で動作するWebGLの機能で、形状の確認可能な3Dビューアーの提供
- 仕事のパートナーやクライアントを見つけてコミュニティを形成することができる
- API提供があり、他ツールや他サービスに組み込める

チームのメンバー数が無制限で500MBまでのプランが無料。すべての機能が無制限のプランが1人あたり25ドル／月で運営されています。

TeamPlatformのような3Dデータを作ったり扱ったりするしくみの進化とともに、3Dプリンタの技術も速いスピードで進化をし続けています。現在の一般的な3Dプリンタの100倍近くのスピードでプリントアウトできる革新的技術も研究されており、近い将来、紙のプリントのような手軽さで3D形状を扱える日も近いかもしれません。SD

ボンサイラボが発売予定の子供用3Dプリンタ「BS TOY」 http://www.bonsailab.asia

大量生産品のランプシェードを3Dプリンタで制作 http://www.ikeahackers.net/

GADGET 1

3Doodler 2.0

http://the3doodler.com/

3Dプリンタペン

3Doodlerは手に持って使うペンタイプの3Dプリンタです。先端から融けた樹脂が出てきて、立体的な形状を作り出すことができます。3Doodler 2.0は初期バージョンに比べて先端ノズルの細さが1/4に、ペンの重量が半分の50gに改良されました。樹脂の抽出速度も速くなったため、コツと熟練が必要だった最初のバージョンよりも扱いやすくなったそうです。周辺機器として、樹脂の抽出速度を足で操作するためのフットペダルも用意されています。本製品はクラウドファンディング、キックスターターの支援ユーザへ発送され、一般販売は未定です。

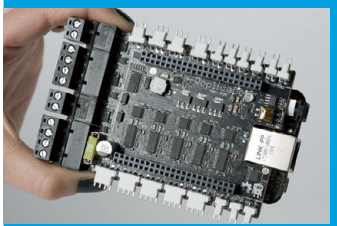
GADGET 2

BeagleBone Black（TIDEP0007）

http://www.ti.com/tool/tidep0007

3Dプリンタをパソコンなしでインターネット接続するデバイス

BeagleBone Black（TIDEP0007）はオープンソースの3Dプリンタコントローラのリファレンス実装です。3Dプリンタに接続することで、パソコンの介在なしで、インターネット対応デバイスとして動かすことができます。BeagleBone Blackは安価なシングルボードコンピュータで、UbuntuなどのLinuxディストリビューションやAndroid 4.2が動作するスペックを持ちます。このボードを3Dプリンタで活用しようというプロジェクトが進行しています（http://www.thing-printer.com/product/replicape/）



GADGET 3

OwnFone

https://www.kickstarter.com/projects/651968834/ownfone-make-your-own-mobile-phone

3Dプリンタ筐体のオリジナル携帯電話

OwnFoneは、シンプルな携帯電話の筐体そのものを3Dプリンタで出力し、オリジナルの携帯電話を作るためのサービスです。3Dプリンタの利用例としてよくとりあげられる、スマートフォンのオリジナルケースを作る時代はもう古いのかもしれません。通話ボタンなどはさまざまなタイプにカスタマイズすることができ、通常の10キーのある電話のみならず、あらかじめ設定した数ヵ所の電話番号にかけるだけの専用電話的なものも作ることができます。とくに子供用、高齢者用などに重宝すると思われます。イギリスの携帯電話会社より7,000円弱で販売の予定で、通話料はプリペイドでチャージするそうです。

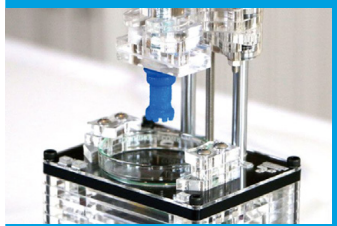
GADGET 4

iBox Nano

http://www.iboxprinters.com

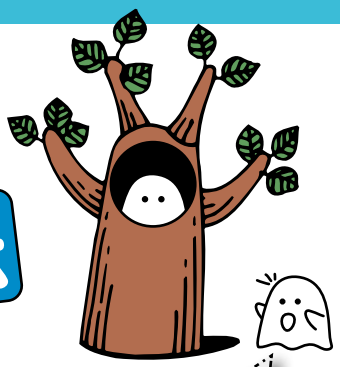
持ち運べる超小型3Dプリンタ

iBox Nanoは鞄に入れて持ち運べるほど小型の3Dプリンタです。現在、先行価格299.99ドルで販売を開始しています。大きさは85×110×235mmで、片手の上に載るくらい。重量は1.1Kg、バッテリーで約10時間駆動し、最大造形サイズは40×20×90mm。3Dプリンタ機器自身がWebサーバになっており、Webブラウザでアクセスして設定したり、データを送り込んだりします。巨大な造形ができる3Dプリンタもありますが、ほとんどの個人ユーザは時間やコストの理由で小さなものしか作らないそう。



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

結城浩の再発見の発想法 24



Wrapper

Wrapper——ラッパー

ラッパーとは

ラッパー（Wrapper）とは、**複雑で細かいたくさんのものを包んで、単純化するもの全般**を指します。英単語で“wrap”（ラップ）というのは「包む」という動詞で、“wrapper”は「包むもの」という名詞になります。

プログラミング技術ではときどきラッパーが登場しますが、分野によって目的は多少異なります。たとえば、HTMLやCSSで細かい構成要素をまとめて入れる大きな要素に対してコンテナ（container）やラッパー（wrapper）と名前を付けることがあります。あるいは、たくさんの関数（API）を提供しているライブラリをもとにして、もっと使いやすい少数の関数を提供するもののことをラッパーと呼ぶことがあります。具体的な表現方法は異なりますが、どちらも複雑で細かいたくさんのものを包むという働きは同じです。以下では、ライブラリに対するラッパーを使ってお話しします。

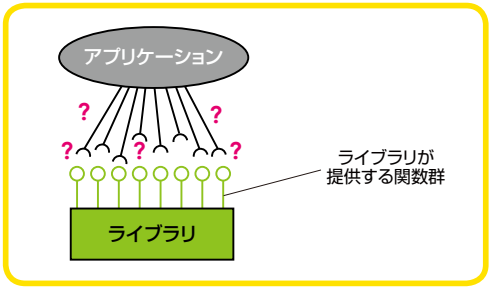
ラッパーの目的

そもそも、なぜラッパーが必要なのでしょうか。それは使いやすさを向上させるためです。たとえば、たくさんの関数を提供しているライブラリがあったとします。ファイルシステムの操作であれ、3Dモデルの変換処理であれ、プログラマは、そのライブラリの関数を呼び出して、アプリケーション（以下、アプリ）に必要な機能を実現します。

たくさんの関数が提供されているライブラリは、かゆいところまで手が届く細やかな処理ができるかもしれません。それはメリットです。しかし、作ろうとしているアプリにそれほど細やかな処理が不要なら、たくさんの関数が提供されていることは、逆にデメリットになってしまいます。なぜなら、アプリに必要な機能を実現するために、どの関数を呼べばいいのか、どんなパラメータを渡せばいいのかを調べる手間がプログラマにかかってしまうからです。複雑過ぎるライブラリをアプリが利用するイメージを**図1**に示します。

ラッパーが必要になる1つの理由は、この手間を軽減させるところにあります。すなわち、それほど細やかな機能を必要としないアプリでは、そのライブラリに「一皮」かぶせたラッパーを利用するのです。適切に設計されたラッパーが提供されれば、アプリを作るプログラマの手間は大いに軽減するでしょう（**図2**）。

▼図1　複雑過ぎるライブラリを利用するアプリ



良いラッパーと悪いラッパー

良いラッパーは良いライブラリ同様に、きちんと設計する必要があります。とくに、そのラッパーがどのような機能を提供するのかを明確にしなければなりません。ラッパーとして提供される関数だけでは機能が完結せず、低レベルのライブラリを直接呼び出さなければならないとしたら、ラッパーの意義は薄れてしまうからです。

ラッパーの必要性を意識するには、ライブラリの複雑さを意識する必要があります。そして、良いラッパーを作るためには、ライブラリがどのようなユースケースで用いられるかを把握する必要があるでしょう。

プログラミング技術は複雑さとの戦いです。油断すると、あっというまに関数の数も、パラメータの数も増えてしまいます。機能を落とさずに単純化する方法の1つ、それがラッパーと言えるでしょう。

日常生活とラッパー

日常生活の中で、ラッパーに類したものが必要になる局面はあるでしょうか。たくさんの情報がやりとりされるところにはラッパーを用意する余地があります。

たとえば、新規顧客に申込み書類を書いてもらう局面を考えてみましょう。顧客が選択するオプションがたくさんあると、顧客は記入するのがいやになりますし、時間もかかります。そんなときには、初心者向け、中級者向け、上級者向けのような「申込みパック」を用意しておき、それぞれの顧客に向けて適切なオプション設定がなされた申込み書類を用意しておけば、無駄な手間を省けます。この「申込みパック」は申込み書類に対してラッパーを提供していることになります。顧客が好むもの、顧客が選択するものを適切に集めてひとまとめにし、「これにします」と一言で済むようにしているわけですから。

小規模な会社や、個人が仕事を請け負うとき、「委細は要相談」というパターンになることがあります。業務の依頼主が現れたときに相談して仕事の進め方や料金を相談するという、一見理屈にあったパターンですが、意外に手間がかかったり、意思疎通に時間がかかったりする場合もあるでしょう。

相談することで細やかな対処ができるけれど煩雑になるということは、ラッパーを考える価値がありそうです。つまり、「私どもは普段、このようにしております」というパッケージを前もって作っておき、そこから外れるときに限って相談するという流れにするのです。料金プランや、得意分野、提供できるサービス一覧などを整理して提示することで、自分の進めたい方向に仕事を持っていき、意思疎通の時間を節約することもできるでしょう。

細かい制御ができることが良いとは限りません。細かい制御と煩雑さとの間のトレードオフを見極めることが大事なのです。適切なラッパーを作ることは単純化のために有効です。

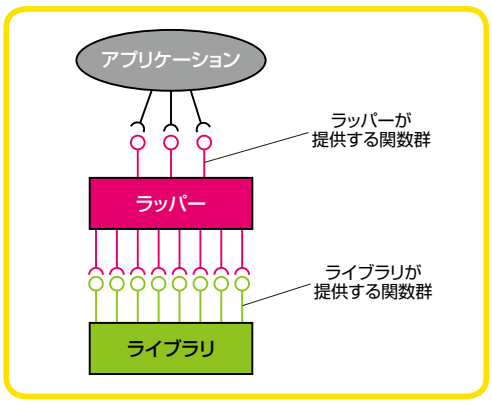
◆ ◆ ◆

あなたの周りを見回して、細かい制御はできるけれど複雑で使いにくいものはないでしょうか。それに対して、単純化したラッパーを作ることはできないでしょうか。

逆に、単純化しようとしているけれど、実際のユースケースからずれているために複雑なままになっているものはないでしょうか。

ぜひ、探してみてください。SD

▼図2　ラッパーを利用するアプリ



耽溺せよ電子工作

おとなラズパイリレー

第7回

「Raspberry PiでNarrative Clipモドキを作る（前編）」

村上 福之

おとなラズパイリレーは、Raspberry Piを文字どおり「リレー」し、好奇心旺盛なITエンジニアが電子工作をするという企画です。前編で構想を練り、後編で実装します。1年を通してどんなデバイスができあがるのか？……今回は、クレイジーワークスの村上総裁によるRaspberry Pi B+でNarrative Clipモドキを作る、の構想編です。

Writer 村上 福之（むらかみ ふくゆき） ㈱クレイジーワークス 総裁

締め切り直前と未開封のRaspberry Pi

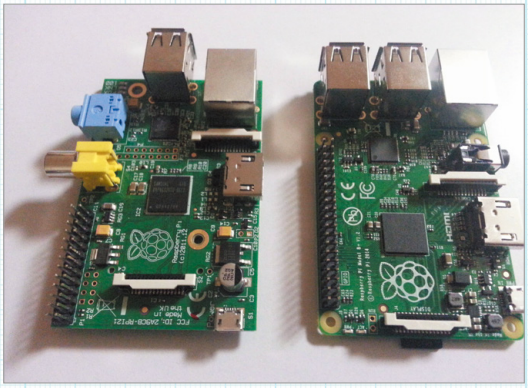
締め切りの前日なのに、まだ1文字も書いていない。

Raspberry Piについての記事を4ページほど書かないといけないのだが、まったく書いていない。おそらく多くのソフトウェアエンジニアと同じように、Raspberry Piを購入しても何をしていいのかわからないままほったらかして、押入れの肥やしになっていたのだ（**写真1**）。編集部から、Raspberry Pi B+が1ヵ月前に届けられてきたのに箱も開けていない……。

たぶん多くのエンジニアにとって、Raspberry Piでサクっとできることは、スマホでできてしまうからだろう。Rapsberyy Piで監視カメラを作るようなブログ記事は、この世にたくさんあるけれど、僕は「それってAndroidやiPhoneでできるやんけ！」と言ってしまうのだ。僕はもともと組込みエンジニアだったのだけど、才能がなさすぎてWeb系に転進した。だから、組込みLinuxのGPIOに何かを挿して動かして、ナンヤカンヤするという開発は、正直、若いころにした辛い思い出が多いし、DSPやCCDやさまざまなコントローラにつないでも、あまり感動しないのだ——そのうえ当時はコードを書くことがキライだった。

そのせいで、メーカー出身なのに、Makerブームに乗る気がしない自分がいる。開発がたいへんでコストがかかる割に、Web系ほどスピード感も利益率もない。Makerブームなんてクソくらえだと思っている自分がいるし、そんな自分が嫌いだ。

▼**写真1** 2台の手つかずのラズパイ……。どうしてこうなった。



Raspberry Piについて

いまさらRaspberry Piについて書くのも馬鹿馬鹿しい話だ。**Raspberry Piはイギリス発の教育用と言いながら、日本のソーシャル上ではオッサンしか出てこないコンピュータだ。**ソフトウェアはDebian GNU/Linux系のLinuxがちゃんと動作するうえに、GPIOがむき出しな

ので、いろいろなハードウェアに接続して、けっこう好きなことができる。また、最近はRaspberry Pi 2というスマホ並の性能を持つものも発表されて話題をさらった（次回紹介）。それに加えて、**Windows 10も対応表明しているので**、さらに面白いことになりそうで期待したい（しかし、Alpha/MIPS/PowerPC/ARM系のWindows RTなどなどの**非インテル系のWindowsは、今までいずれも悲運な最期を迎えてばかり**なので、今度こそは世に広まってほしいと思う）。

さらにチュートリアルもかなり充実している。「教育用コンピュータ」だけあって、子供向けも多い。一番衝撃的なのは、サンタクロース発見器[注1]だ。**これを作って、クリスマスイブの寝室のドアに置くと、赤外線センサーでサンタが来た瞬間に大音量が鳴るという、子供の夢を自らクラッシュさせるトラウマ確実のチュートリアルだ**。た、確かに子供向けだけど、そのあとの責任は取れるのかと、小一時間問い詰めたくなる。

意外と子供にはハードル高いし、ノマドに向かないRaspberry Pi？

子供向き入門コンピュータと聞くと、オッサン世代は、MSXやぴゅう太を思い出すだろう。僕も、もともとは『MSX・FAN』（徳間書店）の投稿少年だった。それらはテレビと電源をつなげばすぐに動いた。まさに家電感覚でスイッチポンのコンピュータだった。子供でもらくちんだ。親がコンピュータに無知でも子供だけで遊べて、自学自習できた。

一方で、**Raspberry Piは、親にも子供にもある程度知識がないと厳しい**。初期設定でなんやかんやで有線LANかHDMI接続のディスプレイが必要で、多くはそれに加えてUSBキーボードと、USBマウスが必要だ。無線LANとノートパソコン全盛の時代、親が普通の世界の人の場合、ルータなんてリビングの端っこに置かれていて、有線LANなんてなかったりするし、普通の家だとノートパソコンばっかりで、USBキーボードやUSBマウスがない家庭が増えてきたように思う。教育用コンピュータといいながら、コンピュータに詳しくない両親がいる子供が買って自由に遊ぶには、MSXよりはるかにハードルが高い。無線LANの設定も普通のLinuxと同じくコンソールで設定するので、たぶん教育用と言いながらかなり厳しい。

話がそれちゃうけど、僕と同じMSX世代の福野さんが、もっとらくちんなIchigoJamという教育用コンピュータ[注2]を作ったのだけど、その理由がすごくわかった気がする。Raspberry Piの思想は素晴らしいし、教育用としてすばらしいけど、ぼくらMSX世代が体験した世界とかなり違うし、ハードルがかなり高いのだ。

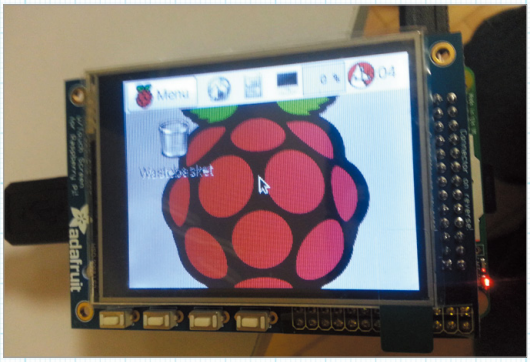
僕の仕事スタイルも、ノマドに近く、あちこち打ち合わせに行った途中のカフェで開発することも多い。そのため、有線やルータがないと開発できないRaspberry Piは、どうもめんどくさかった。一番たいへんなのは家の中。有線LANが届く場所にディスプレイを移動するのが難儀だった。教育用という割に手間がかかりすぎる。

わかんないので、過去にみんながやったことをやってみる

正直、締め切りが近いので（笑）、まとめサイトを縦覧して、過去に世間の方々が作ったものを調べてみる。——とりあえず、GPIOをつなげば何か動くのはわかる（**写真2**）。急いでいたので適当によくある周辺機器を購入してみた。やはり、もっとも多いのがカメラだ。純正のカメラがあって、これを使えば簡単に撮影ができるので、ほぼセッ

注1）http://www.raspberrypi.org/learning/santa-detector/
注2）http://ichigojam.net/

▼写真2　とりあえずいろいろ試してみた



▼写真3　Narrative Clip（http://getnarrative.com/）

ト商品に近い。次に多いのがメディアプレーヤ。前回の増井先生が使ったomxplayerを利用したもの。最後に、けっこうメジャーだったのがエミュレータを使ったゲーム機。Retro Piというイメージファイルが配布されている。これはファミリーコンピュータ、スーパーファミコン、Sony PlayStation、Game Boy Advance、Game Boy Color、Game Gear、MAME、Megadrive、Genesis、PC Engine、TurboGrafx、はてはIBM-PCやMacintoshまで対応している。CPUが遅いので実機並の速度が出るか心配だったが、スーパーファミコンを動作させたところ、普通と変わらないフレームレートが出た。法的な問題がクリアになればこれで十分楽しめると思う。

Narrative Clip モドキを作ってみる

今回、とくに何も思いつかなかったので、Narrative Clipモドキを作ってみる。Narrative Clipとはライフログカメラ（**写真3**）。30秒ごとに自動的に写真を撮影する。バッジやブローチのように身に付けて、1日の動きを自動的に画像で記録できる。非常に魅力的な製品だ。毎回、旅行や忙しいパーティに行くたびに、Narrative Clipが欲しいなーと思う。しかし、カメラの性能がヘッポコなうえに、明らかに3日で飽きそうな機能なのに、円安さまさまのおかげで価格が日本円で3万円近くする。ネットの情報でも、Narrative Clipの購入者の評判は素晴らしく、「すぐ飽きた」「ぶら下げているだけなのでマトモな写真がほぼ撮れない」「飽きたので売ります」という人が続出。3万円あればChromebookくらい買えてしまうのだ。

しかし、Raspberry Piを無料でもらえたので、今回はNarrative Clipもどきを作って、いかにすぐに飽きそうか検証してみる（決して、時間がないからではない）。

純正カメラが動かない

純正のカメラをとりあえず取り付けてみたがまったく動かない。ネットでググるとRaspberry Pi B+と純正カメラの相性がどうも悪いようだ。純正なのに動かない。動く物もあり個体差があるようだ。そこで、日経BPのムックについていた普通のRaspberry Pi Bにカメラを接続するとあっさり動作した。次は開発言語だ。Raspberry Piの開発言語はPythonだ。日本人には人気がいまいちな言語だが、非常に書きやすい。オッサン組込み経験者として、ターゲットマシンでコードを書いて、そのまま実行するという世界がなんだか気持ちが悪い。それをcronで一定時間で動作するように設定する。非常に感動したのが、組込みのプログラミングなんていつもいろいろ

悩みに悩んだり、ハードウェアの初期化コードだけで、半日掛かったりするけど、Rasberry Piはだいたい Python なので書いたらすぐ動くこと。**かなり頭が悪くても動く。Narrative Clip くらいだと30分もコードを書かない。**

Raspberry Piは普通に micro USB の電源で駆動する。今や Amazon で16,000mA のバッテリーが3,000円台で買えてしまう時代なので、数日もつのではないかと思う勢いだ。それを700円の USB の Wi-fi ドングルにつないで、手持ちの WiMax ルータでつなぐ。**結果、ものすごい重い Narrative Clip ができあがってしまった。**重量の9割がバッテリーだ(**写真4**)。本家 Narrative Clip のように持ち歩けないので、リュックに放りこんで、カメラ部分だけ外に出してみた。やはり、世間の Narrative Clip と同じような評判の画像がたくさん撮れた(**写真5**)。自動的に Twitter に投稿するように試してみた。カメラの性能もあるけど、絵は見られたものではない。やはり、カメラはちゃんと構えて、タイミングをきっちりしないと面白いものが撮れないということがわかった。

Raspberry Piでできることは、スマホと同じか

しかし、Android などでいろんなハードウェアをつなぐとなると、ADK 経由の接続で非常に開発が煩雑だ。また、開発環境も日に日に肥大化している。Android は肥大化しすぎてちょっと開発環境を用意するのに、1時間くらいはダウンロードと設定で時間がかかる(とくに肥大化しすぎた SDK のダウンロードはどうにかならんのか……)。Android や iOS でカメラアプリケーションを作るのは楽といえば楽だけど、Raspberry Piほどではない。何より、Raspberry Piは専用機なので、スマートフォンと違って、複雑な状態遷移がない、ホームボタンを押されたり、スリープに移ったりといったカオスな状態遷移を考える必要がないのだ。

Raspberry Piは教育用コンピュータとして、すばらしいと思う。SD

▼写真4　バッテリーがその重量のほとんどを占める村上総裁版の Narrative Clip

▼写真5　ツイートのテストページ（https://twitter.com/pirative0309）

軽酔対談

かまぷの部屋

第10献 ゲスト：多田 歩美さん

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

多田 歩美（ただ あゆみ）さん

本田技研工業㈱勤務。IT本部システム基盤部にてクラウド技術活用の推進をしている。昨年、ラスベガスで行われた「AWS re:Invent 2014」にて唯一の日本人登壇者として『HONDA』におけるクラウド上でのHigh Performance Computing（HPC）利用について発表。世界のクラウド技術者の注目を集める。くまもん好き。愛車はHONDA Shadow Slasher。
Twitter：@applebear_ayu

AWS re:Invent 2014登壇

今回は日本が誇る『HONDA』の中の人、多田歩美さんをお迎え致しました。

（鎌田）まずは自己紹介と、今されているお仕事のことを教えてください。

（多田）仕事は、簡単に言うとインフラ系の情シスです。本田技研工業に入社したのは2006年で、すぐに4ヵ月間、熊本にある工場で研修しました。バイク好きなので、そこでカブの組み立てを経験できたのはうれしかったです。その後、和光にある研究所のIT部門のCIS技術課という部署に配属となり、2013年まではそこにいました。今の仕事内容は、おもにCAE[注1]というHPC分野ですが、科学計算用のサーバなどの機種選定／管理やクラウド技術の推進をしています。

注1）Computer Aided Engineering（Aidedは支援という意味。ハイエンドなコンピュータ技術を活用し製造や工程設計の事前検討の支援を行うこと、またはそのシミュレーションツールなど）。

多田さんは理系なのでしょうか？大学時代は何を勉強されていたのですか？

理系です。早稲田の理工学部で電気電子情報工学科（当時の学科名）で、電子回路など幅広く勉強していました。研究では、興味があった中国語の、言語学習を最適にするためのアプリケーションを作ることに夢中になっていました。大学時代の研究室にもサーバはありましたが、もっぱらGUIのアプリケーション上で使っていました。CLIの黒い画面は正直怖かったのですが、まさか将来、仕事として触るようになるとは思ってもいませんでした。バイク好きだったというのと、企業風土に憧れて、3年の春にはすでに就職先の第一候補としてHONDAに行きたいと思うようになりました。

HONDAが第一志望だったのですね。黒い画面怖いですか（笑）。

私が触ったら爆発するかと思いましたもん（笑）。入社当時、サーバ管理の担当を任されたときに恐怖を感じていたら、先輩に「コンピュータに“使われる側”になりたいか？“使う側”になりたいか？」と言われました。負けず嫌いな私は、まんまとその言葉に乗せられました（笑）。

インフラに携わるメリットは何でしょうか？

サーバ管理部署に配属されたときは、アカウント管理とかネットワーク以外のインフラ全般を見ていました。初めは後ろ向きだったのですが、次に異動した際、社内リソースに何かがあっても基盤となるIT技術を理解していれば問題解決までの時間が短いと思ったのです。HONDAという恵まれた環境やネームブランドですが、私はこれを利用して世界を少しでも変えたいと思っています。

大学時代に中国語学習のアプリを作った話がありましたが、英語は得意ですか？

英語を本格的に勉強し始めたのは2013年からで、海外事業所とのやりとりが増えたことがきっかけです。自分の想いをちゃんと英語で伝えたり、相手のことを理解したかったのです。それで短期留学などをして勉強しました。

JAWSコミュニティイベントや勉強会によく参加されていますよね。




私はけっこう珍しいタイプだと思います。「AWS re:Invent 2013」の懇親会で知り合いができてから、積極的に勉強会に参加するようになりました。「情シスは要件出してればいいんでしょ」と思われていることも多く、そこに壁があると思っていたので、積極的に外に出るようにしています。実際、外に出てみていろんな方々と知り合うことができて、今は楽しくてしょうがないです。昨年には「AWS Samurai 2014」という賞をいただきました。E-JAWSとJAWSの両方の勉強会参加や、海外講演への登壇に挑戦した成果だと思います。

受賞おめでとうございます。ラスベガスで開催された2014年11月の「AWS re:Invent 2014」で、登壇されたそうですが、その経緯を教えてください。

きっかけは、私の勘違いから日本国内で行われる「AWS Summit」に登壇したことです。最初は、AWSの仕様改善の要望を伝えるために、シアトルで行われた会議に協力をしてくださったAWSの中の方から「(機会があったら)イベントでHPCについてお話ししてください」とお声掛けいただいたのです。そのタイミングが、偶然にも「AWS Summit Tokyo 2014」が開催される直前だったので、てっきり「(そこで)」と勘違いし、社内調整して進めてしまい、最終的に登壇することになりました。それを見た「AWS re:Invent 2014」にかかわる方から、グローバルコミュニティでも発表してほしいという依頼があり、「日本語で発表した物でよろしければ」とお返事しました。HPCでのクラウドの利用は未知の世界だったので、自分が不思議だったことをわかりやすく説明したかったのです。英語はもう……前日まで特訓でしたが。

すごいですね。1万人ぐらい集まるイベントで、しかも世界規模ですから。

「AWS re:Invent 2014」のあとは、持っているエネルギーすべて放出した気分で、放心状態でしたよ(笑)。

ところでそのHPCでのクラウドの利用って、どんなことをイメージすればいいのでしょうか。

たとえば、車やバイクのまわりを流れる空気抵抗を減らすために空気の流れを計算するのですが、この計算の精度を高めようとすると、高速かつ大規模なコンピュータのリソースが必要です。自前でサーバを用意して使うのと、クラウド上で拡張しながら使えるのでは使い勝手が大きく違います。スケールアウトできることが大きなメリットです。

お休みはどんなふうに過ごしているんですか？

基本的に動いていないとダメなタイプで、連休があると遠出します。熊本まで野球観戦とか。また、料理が趣味なので、時間があるとお肉と野菜を使った創作料理などをしています。そのためキッチンの広い部屋を借りました。だいたい自分が想定したとおりの味付けになります。The理系料理です。

料理ができる女性ってモテそうですね。どんな男性のタイプが好きですか？

理系男子好きです。彼らは理路整然と説明しますよね。その説明を聞くのが好きなんですよ♡　父は数学の教師だったのでその影響もあると思います。もう他界してしまいましたが、お父さんっ子でした。将棋が得意で、私の「歩美」という名前もそこから来ています。

多田さんのエネルギーはお父様譲りなんでしょうね。ありがとうございました。SD




はんだづけカフェなう

6LoWPANしてみよう(後編)

text:坪井 義浩　TSUBOI Yoshihiro　ytsuboi@gmail.com　@ytsuboi
協力:㈱スイッチサイエンス　http://www.switch-science.com/

IoTなプロトコル

今回のテーマの前編(2015年3月号の第53回)で、6LoWPANのアプリケーション層のプロトコルとして、MQTT(Message Queuing Telemetry Transport)とCoAP(Constrained Application Protocol)という2つの見慣れないプロトコルを記しました(図1)。この2つのプロトコルを簡単に紹介しましょう。MQTTやCoAPというのは、IoT(モノのインターネット)やM2M(Machine to Machine)での通信に使われるプロトコルです。

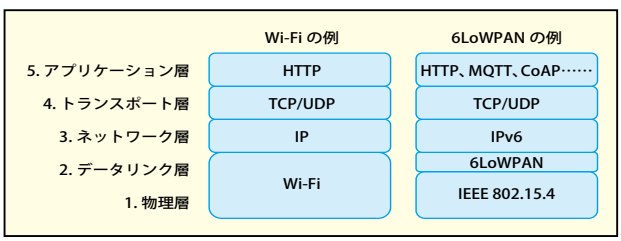
図1にはHTTPも記されていますが、マイコンでHTTPを扱った経験がある方は、HTTPヘッダ、とりわけレスポンスヘッダの大きさを疎ましく感じたことがあると思います。センサの値などの数byte程度のデータを通信するといった用途では、HTTPヘッダというのはあまりにも大き過ぎるのです。HTTPを使って通信しようとすると、限られているマイコンのメモリを大きく通信バッファに割り当てなければいけない、といったケースが多く見受けられます。

MQTTやCoAPは、HTTPと比較してはるかに軽量なプロトコルです。たとえばMQTTの固定ヘッダは2byteのみです。MQTTはTCPコネクションを張りっぱなしにして、このコネクションを使って双方向通信を行うプロトコルです。MQTTでは、こうしてTCPのハンドシェイクのコストを削減しています。また、MQTTにはQoS(Quality of Service)という概念があり、リトライをせず到達が保証されないQoS 0、重複する可能性はあるが少なくとも1回は届くQoS 1、正確に1回だけ到達するQoS 2といったレベルを選択できます。

MQTTのサーバは、MQTT Brokerと呼ばれます(図2)。Brokerはその名のとおり、メッセージの発行元(Publisher)と購読者(Subscriber)の仲介をします。オープンソースのMQTT Brokerの実装としては、Mosquitto[注1]が有名です。実は読者のみなさんもMQTTを使っていらっしゃるかもしれません。というのも、スマートフォンのFacebook Messengerアプリは、MQTTを使用しています。MQTTを使用することでレイテンシの改善や、省電力を実現しているということです。MQTTで受け取った情報の処理については、本誌2015年の3月号と4月号の短期連載、「Bluemixで試してみるIoT入門」で紹介されています。

CoAPは、UDPの上で動く、HTTPの簡易版とも言えるプロトコルです。CoAPのヘッダは4byteで、UDPですのでTCPのようにハンドシェイクがなく実装がシンプルです(図3)。

▼図1　6LoWPANの概念



注1)　http://mosquitto.org

CoAPはパケットごとにトランザクションIDを持ちますが、MQTTのようなQoSはありません。

技術基準適合証明と工事設計認証

少し技適の話もしましょう。「技適なしスマホ」といった文脈でニュースにもなっていますので、技適、技術基準適合証明のことはみなさんおおむねご存じでしょう。実は、スマホの技適というのは、あまり正しい表現ではありません。スマホなどの電波を利用する端末を日本で利用するには、電波法令で定めている技術基準に適合している必要があります。

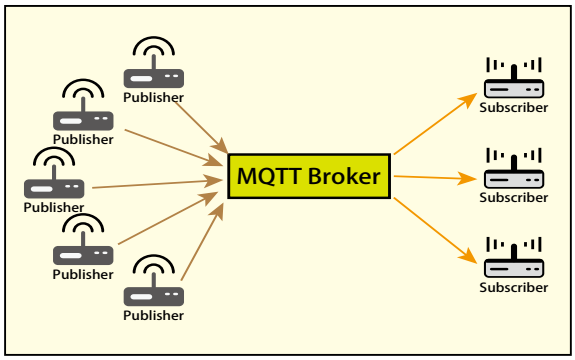
この基準を満たしていることを証明する方法には、技術基準適合証明と工事設計認証の2種類があります（図4）。

技術基準適合証明が機器1台ごとに証明するのに対し、工事設計認証は機種ごとに証明を行います。技術基準適合証明は、1台ごとに異なる証明番号が付与されるのに対し、工事設計認証は同一機種であれば何台でも同じ証明番号が付与されます。スマートフォンなど大量生産される無線機は、通常は工事設計認証を受けます。つまり、スマホが受けている認証は、厳密には技適ではなく、工事設計認証です。総務省の文章を見ると、技術基準適合証明および工事設計認証を併せ、「技適など」と表現されています。

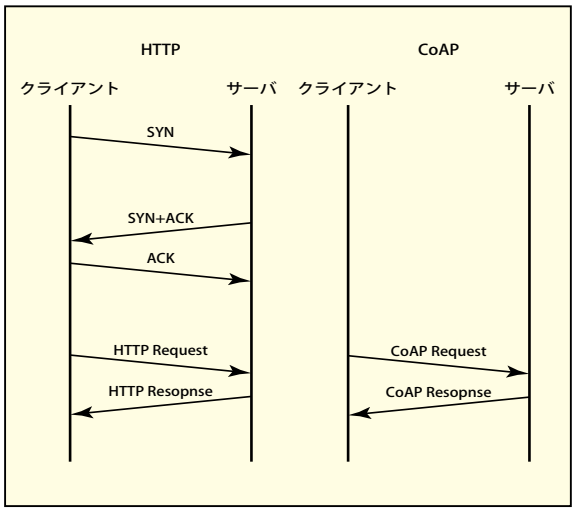
今回、技適などの話を記したのには理由があります。前編で記したように、nRF51822にはRAM容量が16kBのチップと32kBのチップの2種類があります。MQTTやCoAPのサンプルプログラムは、RAM容量32kBのnRF51822を搭載したnRF51-DK（メーカー型番PCA10028）用に提供されています。

今回、このnRF51-DKを入手したのですが、この基板は工事設計認証を受けてい

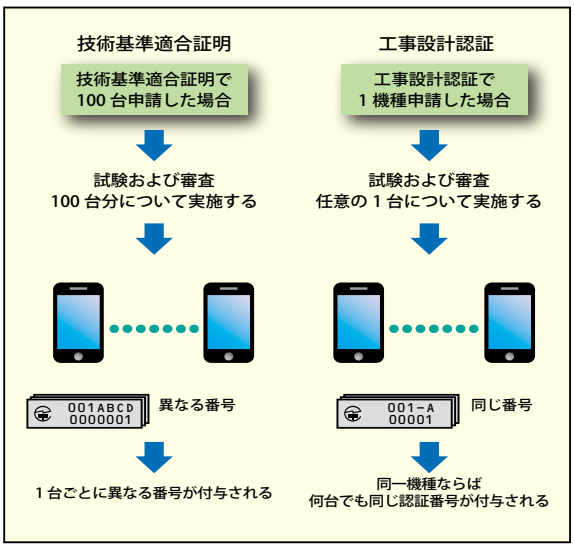
▼図2　MQTT



▼図3　HTTPとCoAPのシーケンス図



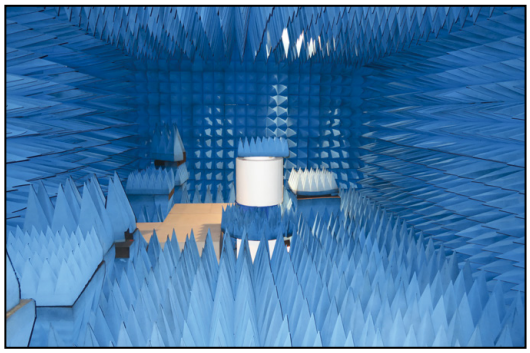
▼図4　技術基準適合証明と工事設計認証の違い



ません。つまり、電波法の基準に適合しているかどうかわかりませんので、このまま日本国内で使用すると、電波法に抵触する可能性が否めません。こういった事情で、電波暗室という、電波を部屋の外からの電磁波の影響を受けず、また、電磁波を部屋の外に漏らさない実験室を借用して実験を行うことにしました。

▼写真1　電波暗室の外観

▼写真2　電波暗室内部



▼写真3　携帯電話が圏外に

電波暗室

無線機器の性能を評価するには、外部の電波や、部屋の中で反射する電波がノイズになります。こういったノイズを排除するため、電波暗室は、電波吸収体という素材などを貼り付けて作ります。この電波吸収体はけっして安くない素材で、とあるメーカーが公開している「手作り電波暗室」という資料[注2]に掲載されている5m×2.4mの部屋の場合の概算で、約300万円という価格が記されています。このように、電波暗室を用意するにはとても高価な材料が必要ですし、それなりに大きな場所も必要ですので、個人で用意するのは現実的ではありません。

電波暗室は、電波を使用する機器を開発するメーカーや、工業試験場といった施設が所有しています。工業試験場の電波暗室を使わせていただこうと思い問い合わせをしてみましたが、2ヵ月先まで予約がいっぱいとのことでした。今回は、PHS電話機などを開発・販売している、㈱エイビットの電波暗室を借用し、実験を行わせてもらいました(**写真1**)。

電波暗室の扉は、外側からのみ開閉できる構造です。扉が閉まると、外部からの音が遮断されますし、電波吸収体が音をあまり反射しないために妙に静かで変な感じがしました(**写真2**)。当然ですが、筆者の携帯電話も圏外になりました(**写真3**)。

CoAP Client

今回は、Nordic IoT SDKのCoAP Clientというサンプル(`¥Nordic¥nrf51¥examples¥iot¥ipv6_coap_client¥boards¥pca10028¥arm`)をビルドして、nRF51-DKで動かしてみました。ビルドの手順自体は前回紹介したとおりです。ビルドが完了すると「`_build¥nrf51422_xxac_s1xx_iot.hex`」というファイルができあがりますので、これをnRFgo Studioを使っ

注2)　http://www.tssj.co.jp/pdf/absorber.pdf

て書き込みます。nRF51-DKにはJ-Link Lite相当のチップが搭載されていますので、nRF51-DKのUSBレセプタクルにケーブルを接続して書き込みました。

先ほど述べたように、CoAPはUDPを使うプロトコルです。CoAP Clientには、接続先のIPアドレスとポート番号を指定しなければなりません。サーバとなるPythonスクリプトを実行するLinuxマシンの上で、ifconfigを実行して、bt0のIPv6アドレスを確認しておきましょう。サーバのIPアドレスは、CoAP Clientのmain.cで、SERVER_IPV6_ADDRESSというマクロで指定するようになっていましたので、これを書き換えました。

サーバの準備

CoAPのサーバは、Pythonで書かれたサンプル実装がドキュメントに掲載されていますので、これを利用します。サーバとなるPythonスクリプトは、aiocoapというCoAPのプロトコルのライブラリを利用します。aiocoapはPython 3.4以降を要求しますので、注意してください。

今回の実験を行うにあたって、Raspberry Piを電波暗室に持ち込むのが面倒だった筆者は、Ubuntu 14.04.1のVMを作り、MacBookのVMware Fusionで実行する環境を作りました。6LoWPANをサポートするkernelは、make menuconfigで前回紹介したオプションを有効にしてビルドしました。これまでRaspberry Piに挿していたBT-Micro 4をMacBookに挿し、VMware Fusionの設定でBT-Micro 4をVMのUbuntuに接続しました。

このUbuntuでも、bt0インターフェースを起こす手順はRaspberry Piと同一です。UbuntuでCoAP Clientを動かすため、aiocoapを用意して、図5のような手順でサンプルのスクリプトを動かしました。

動かしてみる

nRF51-DKでCoAP Clientを実行したばかりで、BLEのアドバタイジングをしている段階では、nRF51-DKのLED1が点滅をしています。ここで、Linux側でbt0インターフェースを起こし、nRF51-DKとの間でのIPv6接続が確立できるとLED1は消え、LED2が点灯しました。Ubuntu側でCoAP Serverを起動すると、CoAPのメッセージがnRF51-DKから送られて来て、デバッグメッセージがターミナルに表示されました。nRF51-DKのButton 1や2を押すと、CoAPのメッセージが送られ、同様にデバッグメッセージが表示されます（写真4）。

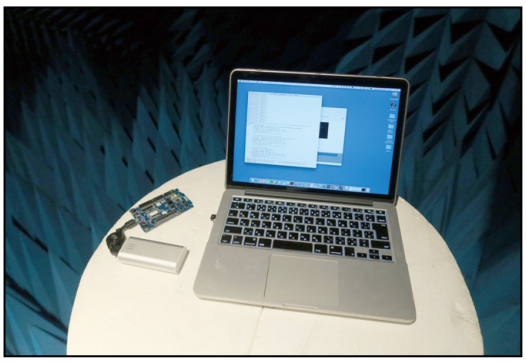
まとめ

3回に分けて、6LoWPAN over Bluetooth LEを使って、マイコンとサーバのEnd-to-EndのIPv6通信の話をしてきました。IoTの世界では、マイコンのメモリの大きさや、消費電力、接続回線の都合でMQTTやCoAPといったプロトコルが使われます。こんな小さなマイコンモジュールが単体でIPv6通信できることにテクノロジの進歩を感じています。SD

▼図5　サンプルスクリプトを動かす手順

```
git clone https://github.com/chrysn/aiocoap.git
cd aiocoap/
vi coap.py .
python3 coap.py
```

▼写真4　実験の様子（写真2の円筒形部分での作業）

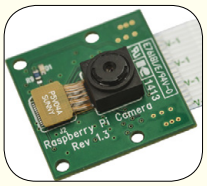


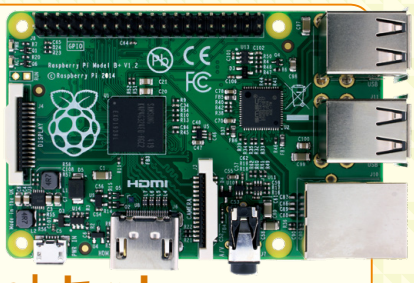
PRESENT 読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト http://sd.gihyo.jp/ の「読者アンケートと資料請求」からアクセスし、アンケートにご協力ください。ご希望のプレゼント番号をご入力いただいた方には抽選でプレゼントを差し上げます。締め切りは2015年5月17日です。プレゼントの発送まで日数がかかる場合がありますが、ご了承ください。

ご記入いただいた個人情報は、プレゼントの抽選および発送以外の目的で使用することはありません。アンケートのご回答について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。ご入力いただいた個人情報は、作業終了後に当社が責任を持って破棄いたします。なお掲載したプレゼントはモニター製品として提供になる場合があり、当選者には簡単なレポートをお願いしております（詳細は当選者に直接ご連絡いたします）。

01 1名





「Raspberry Pi B+」&「Camera module」セット

ARM搭載、名刺サイズのコンピュータボードです。4基のUSBポート、40ピンGPIOコネクタ、micro SDソケットを持ち、さまざまなデバイスと接続して自分だけのガジェットを作れます。今回は、Raspberry Piと接続できる、5Mピクセルセンサー搭載のカメラモジュールとセットでのご提供です（ロゴ入りのオリジナルバッグもお付けします）。

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

02 1名

テキストエディタ MIFES 10

テキストファイル、ソースコード（20種類のプログラム言語に対応）、HTML、CSVやXMLなどのデータファイル、バイナリファイルも編集できる多機能エディタ。「10」ではファイルの構造解析エンジン、ファイル比較機能、アウトライン機能などが強化されています。

提供元 メガソフト **URL** http://www.megasoft.co.jp

03 1名

USBメモリ Ultima U05（32GB）

USB2.0対応のFlashメモリです。コネクタ部を本体に収納できるスライド式筐体を採用し、端子部を傷や汚れから保護します。データのバックアップをするのに便利な機能を備えたWindows専用ソフトウェア「SP Widget」が内蔵されています。容量は32GBです。

提供元 シリコンパワー **URL** http://www.silicon-power.com

04 1名

プログラミング言語 C++［第4版］

ビャーネ・ストラウストラップ 著、柴田 望洋 訳／
B5変形判、1,360ページ／
ISBN = 978-4-7973-7595-4

C++の開発者ストラウストラップ自身による解説書。1,360ページの大ボリュームで、言語の基本からオブジェクト指向のための抽象化機能まで、すべてを網羅した1冊です。C++11に対応しています。

提供元 SBクリエイティブ **URL** http://www.sbcr.jp

05 2名

エバンジェリストの仕事術

西脇 資哲 著／
四六判、224ページ／
ISBN = 978-4-534-05257-5

日本マイクロソフトのエバンジェリストによるワークハック本。エバンジェリストという仕事の全貌を明かしながら、プレゼンのテクニックやスライドの作り方といった仕事術を紹介しています。

提供元 日本実業出版社 **URL** http://www.njg.co.jp

06 2名

CentOS 7 実践ガイド

古賀 政純 著／
B5変形判、320ページ／
ISBN = 978-4-8443-3753-9

CentOS 7の運用管理にかかわるシステム管理者を対象に、「7」において追加・拡張された機能に焦点を当てながら、運用ノウハウをまとめた1冊です。Hadoop 2.0のシステム構築も紹介しています。

提供元 インプレス **URL** http://www.impress.co.jp

07 2名

はじめよう！要件定義

羽生 章洋 著／
四六判、184ページ／
ISBN = 978-4-7741-7228-6

システムのUI・機能・データを明確にしていく「要件定義」は、顧客が納得のいくソフトウェアを実現するために必要不可欠。本書ではその要件定義の知識をポップなイラストとともに解説しています。

提供元 技術評論社 **URL** http://gihyo.jp

第1特集

シェル、sed、AWK、grep

テキスト処理ベーシックレッスン

手を動かしてデータを操ろう！

そもそもなんでテキスト処理なんて必要なんでしょうか。

シェルでコマンドを入力すれば、テキストで返事が返ってきます。走っているプログラムがはき出す状況報告(ログ)もテキストです。Unix/Linuxは、ユーザへの報告をほぼすべてテキストで提供します。これらのテキストを価値ある情報に変えるために、シェルを扱うエンジニアにはテキストを読む・加工するスキルが求められるのです。

本特集では、シェルを扱ううえで必ず役立つテキスト処理の基本を解説します。実例満載ですので、手を動かして自分のものにしてください。そして現場のエンジニアが実用として使っている、とっておきのテキスト処理も披露いただきました。

※リストや図中の➥は本来1行のものが折り返されていることを、↵はエンターキーの入力をそれぞれ表しています。

CONTENTS

Lesson 1 手を動かしてデータを操ろう！

これだけでも応用範囲は限りなく広い

テキスト処理で最初に習得すべきコマンド10選

シェルで自在にテキストデータを加工するテクニックの要は「コマンドをつなげる」ことですが、Lesson1では、まずテキスト処理で必須の10コマンドの使い方を説明します。単純な機能のコマンドでも、複数のコマンドを組み合わせると用途が広がります。本格的な連携テクニックはLesson2で扱いますが、本稿でもその一端を垣間見られるはずです。

Author 上田 隆一（うえだ りゅういち）　産業技術大学院大学／USP研究所／USP友の会
Twitter @ryuichiueda

はじめに

自称シェル芸おじさんこと産技大の上田でございます。今回の特集はシェル（およびシェル上でコマンド）を使ったテキスト処理ということで、筆者の顔面に白羽の矢が刺さりました。

Lesson1、2では「そもそもテキストとはなんだ」というところから、コマンドの紹介、コマンドを組み合わせるときの方法あれこれの話をして、最後に、「シェルでコマンドを組み合わせるプログラミングとはいったい何であるのか」に踏み込んで話をしてみたいと思います。初心者的な内容からベテランの間で論争が起きそうな話まで駆け足でするので慌ただしいのですが、ご自身のレベルに合った読み方をしていただければ幸いです。

準備運動

そもそもテキストとは

テキストとは、最近のUNIX環境を使っている場合はもっぱら「UTF-8のルールで記録された文字」であり、テキストファイルとは、「テキストをファイルに保存したもの」です。しかし、おそらくこう言っただけで通じる人は、全人類のおよそ数パーセント程度だと思われます。わかっている人には少々まわりくどいですが、ちょっとこの辺から話を始めたいと思います。

世間一般においては、自分が書いたそこそこの分量の文字情報を他人に送るときは、電子メールにPDFやWordやExcelのファイルを添付して送るのが一般的です。これらのファイルは、通常は「テキストファイル」とは呼ばれません。オフィス製品で読み書きされるファイルには、文字の大きさ、フォントの種類、文章を右寄せするとか左寄せするとか、そういう情報も含まれています。このような場合、基本的には各ファイルに対応付けられた専用のソフトウェアで読み書きすることになります。

一方、テキストファイルは、世間一般では「拡張子が『.txt』の怪訝（けげん）なサムシング」であり、「何で開いていいのかわからないし、読みにくい」ものです。不用意に「.txt」のファイルをメールに添付をすると、なぜか叱られます。

しかし、本誌読者ならご存じのように、テキストファイルは文字を保存するものとしては最も簡単で一般的なものです。フォントの情報など余計な情報は混入しておらず、例外を除き、ただただ文字（「~」や「@」やスペース、改行などの記号を含む）が並んでいます。

図1に筆者のMacのターミナル（Terminal.app）で表示したテキストファイル/etc/hostsを示します。cat /etc/hostsと打ち込むと、catというコマンドが/etc/hostsを開いて中に書いてある文字を表示してくれます。/etc/hostsには、先頭から「##<改行># Host ……」と文字が

詰めて書いてあるので、catはただ単にそれを出力しているにすぎません。自分が書いたプログラム中でテキストファイルを読み込んで処理したいときも、単純に1文字ずつ読み込んでいけば良いということになります。

テキストファイルでは、フォントや文字の大きさを指定できないので応用が利かないと考えるかもしれません。しかし、CSV(カンマ区切りデータ)やHTML、XML、JSONのように、何かしらのルールを決めてテキストで指定を書くことは可能です。しかもファイルの中身を見たければ、catで済んでしまいます。また、UNIX系OSの設定ファイルやログファイルの多くも、**図1**の/etc/hostsのようにテキストファイルで保存されます。また、端末で次のようにコマンドを打ったときの出力もテキストです。

```
$ date
2015年 3月 3日 火曜日 17時25分49秒 JST
```

ということで、いくらWordやExcel、そしてExcel方眼紙が便利といえども、プログラマの視点から見れば、テキストこそが自由に読み書きできる共通のフォーマットであると言えます。また、特定のアプリケーションの使用を強制されないということは、マウスの操作など手作業を要求されることなく、いくらでも自動処理できるということでもあります。

そういえばこの前、複数のExcelの表から別の表へ、プチプチとデータを移動する奴隷仕事をやりました。筆者がその仕事に入ったときにはすでにマウスを使ってプチプチやらなければならない状態に陥っており、目も時間も消耗してたいへん損をしました。おそらく世間のかなりの割合の人や企業がそんな腐った情報処理をしている状況ですので、少しでも本特集で救われる人がいたらと涙を流して祈るしだいです。いや、涙を流しても改善しないので、このようにメジャー誌の一番目立つところで吠えているわけです。わかっている人だけでなく、わかっていない人にもわからせないといけない根が深い問題ですので、長い長い道のりでしょう。みなさんも吠えてください。

シェル上でのテキスト処理は特別

さてLesson1、2では、いろんなテキスト処理の方法がある中でシェルを使う方法を扱います。シェルとは、端末(MacのTerminal.appのような黒い画面ですね)の裏にいて、操作者がキーボードで打った文字を端末から受け取り、それを解釈してコマンドなどを呼び出しているソフトウェアです。

テキスト処理には、どんなプログラミング言語を使っても良いのですが、その中でもシェルによるテキスト処理には特別な意味があります。まず、システム管理であろうが原稿書きであろうが、端末を使う場合、操作の起点はコマンドで、その出力の大半はテキストです。そして、その出力を加工しようとすれば、端末操作に慣れているなら最も手っ取り早いのはシェルの機能を利用し、パイプでコマンドをつなげる方法です。

たとえば、先ほどの**図1**の/etc/hostsから「localhost」と書いてある行だけを見たいとしましょう。このような場合、**図2**のようにcat /etc/hostsの出力を

▼図1　/etc/hostsをcatで表示

```
$ cat /etc/hosts
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1       localhost
255.255.255.255 broadcasthost
::1             localhost
```

▼図2　catの出力をgrepに渡して検索する

```
$ cat /etc/hosts | grep localhost
# localhost is used to configure the loopback interface
127.0.0.1       localhost
::1             localhost
```

grepという検索を行うコマンドにつなげると、ほしい出力が得られます。「|」の記号は「パイプ」というもので、あとから詳しく説明します。

このようにシェルは、コマンドとコマンドをつなげたり、後述するようにコマンドの出力をファイルに書き出したりという操作が簡単にできるようになっています。通常の言語は、おもにコマンドに相当する「プロセス」という単位の中で仕事をするのに対し、シェルはその外側、つまりプロセス同士を連携させる目的で利用されます。また、ファイル自体もプロセスの外のものですので、ファイルを操作するときもシェルで扱うことが自然です。

テキスト処理という意味では別に小難しいプロセスの話を持ち出す必要はないかもしれません。しかし、とくに端末を扱っているときはテキスト処理の起点となるのはコマンドです。そして、その出力から先を自然にさばくには、パイプやファイルの読み書きをシェルで一通りできることが必要となります。

環境

本題に入る前に、Lesson1、2で使う環境やシェルを決めておきましょう。UNIX系OSであればあまり大差はないと言えますが、やはり方言があるので決めておきます。

まずOSですが、Linuxを使うことにします。以後の端末操作については、Ubuntu 14.04上で行いました。これも**図3**のようにコマンドで確認できます。

これから使うコマンドは最初からインストールされているものがほとんどですが、たとえば後述のnkfなどは追加でインストールが必要です。その場合、**図4**のようにsudo apt-get install……でコマンドをインストールします。

次にシェルを選びますが、Linuxで標準のログインシェルになっているbashを使うことにします。bashについては2014年に大きな脆弱性が発見されて大騒ぎになりましたが、bashを今後どうするかは偉い人たちに任せて、我々はこれを使うこととします。適宜アップデートをしておけば良いでしょう。

最初に習得すべきコマンド10選

まずは初心者の方向けに、テキスト処理でよく使うコマンドを10個選んでみました。これだけでも、組み合わせることでさまざまな処理ができます。

cat（ファイルを表示・連結）

最初に**図1**でも出てきたcatを扱います。catはcatenate（連結する）という単語の頭の3字をとったものです。ただ、作ったときのもくろみとは裏腹に、**図1**のようにファイルを1つだけ表示するために使われてしまうコマンドでもあります。

せっかくですから2つ以上のファイルを連結してみましょう。**図5**のように、catの後ろにファイル名を並べるだけです。/etc/shellsはあとでも使いますが、使えるシェルのリストが書いてあるファイルです。

コマンドには、出力を少し変化させるために「オプション」というものを渡すことがあります。たとえば、**図5**のcatに-nというオプションを渡すと、**図6**のように行番号の付いた出力が得られます。

このようなオプションは、manというコマンドを使って調べられます。端末にman catと打つと、catの説明文が出てくるはずです。

▼図3　使用する環境をコマンドで確認

```
$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=14.04
DISTRIB_CODENAME=trusty
DISTRIB_DESCRIPTION="Ubuntu 14.04 LTS"
```

▼図4　コマンドのインストール（nkfをインストールする例）

```
$ sudo apt-get install nkf
[sudo] password for ueda: ←自分のパスワードを聞かれる
(...略...)
Processing triggers for man-db (2.6.7.1-1) ...
nkf (2.13-1) を設定しています ...
```

▼図5　catで2つのファイルを開く

```
$ cat /etc/timezone /etc/shells
Asia/Tokyo ←/etc/timezoneの中身
# /etc/shells: valid login shells ←以後、/etc/shellsの中身
/bin/sh
/bin/dash
/bin/bash
/bin/rbash
/usr/bin/tmux
/usr/bin/screen
```

▼図6　cat -nで行番号を表示

```
$ cat -n /etc/timezone /etc/shells
     1  Asia/Tokyo
     2  # /etc/shells: valid login shells
     3  /bin/sh
     4  /bin/dash
     5  /bin/bash
     6  /bin/rbash
     7  /usr/bin/tmux
     8  /usr/bin/screen
```

▼図7　for文でファイルごとに行番号を付ける

```
$ for f in /etc/timezone /etc/shells ; do cat -n $f ; done
     1  Asia/Tokyo
     1  # /etc/shells: valid login shells
     2  /bin/sh
     3  /bin/dash
     4  /bin/bash
     5  /bin/rbash
     6  /usr/bin/tmux
     7  /usr/bin/screen
```

▼図8　grepの基本的な使い方

```
↓図5～7で扱った/etc/shells内を検索する
$ grep 'bash' /etc/shells ←①
/bin/bash
/bin/rbash
$ cat /etc/shells | grep 'bash' ←②
/bin/bash
/bin/rbash
```

ところで、この行番号をファイルごとに付けたいときは、どのようにcatを使えば良いでしょうか？「そんなことあるかい？」と言われるとあまりケースが思い浮かびませんが、実際にテキスト処理をしようとすると、このようなイレギュラーな場合に幾度となく遭遇します。さて、正解ですが、**図7**のようにfor文で1つずつ処理を行います。

ここに出てきたforの書き方、その中で使っている変数の使い方についてはインターネットなどほかに説明を譲ります。最初からできなくてもかまいません。まずはforを使わずにエディタで手作業をやって、覚えるモチベーションを十分蓄えてから爆発させてください。ここで重要なのは、「細かい用途の違いに応じてコマンドの使い方を少しずつ変えるのはちょっとしたプログラミングであり、プログラマとして機転を利かせる訓練になる」ということです。なにせコマンドは単機能ですので、組み合わせないと本当にやりたいことができません。その代わり、組み合わせに慣れると大概のことはできます。

grep（ファイル内を検索）

次に紹介するのはgrepです。grep '<検索したい文字列>' [ファイル]というように使います。最初は**図8**-①のようにgrepにファイルパスを直接渡すのが良いかと思いますが、コマンド操作に慣れてきた多くの人は、**図8**-②の書き方をするようになります。「|」はコマンドの出力結果を別のコマンドに入力させるためのパイプを表す記号で、この例ではcatの出力結果をgrepに渡しています。grepはcatの出力結果の中からbashという文字列を検索します。

ところで**図8**では、bashという検索文字列でgrepを使ったところ、rbashと書かれている行も検索にマッチして出力されました。もう少し絞り込みたいときは検索文字列に工夫が必要です。**図8**と同じ例でbashの行だけを出力させたいときは、

```
$ cat /etc/shells | grep '/bash'
/bin/bash
```

と、検索文字列に「/」を足してやれば良いでしょう。また、「アルファベットの何か1文字＋ash」というシェルの名前を出力したければ、正規表現を使って、

```
$ cat /etc/shells | grep '/[a-z]ash'
/bin/dash
/bin/bash
```

と書くことになります。正規表現は、プログラミングの経験のない人には大きな壁になることがありますが、必要に迫られるような環境に自分を置いて少しずつ覚えていけば良いでしょう。覚えたら必ず良いことがあります。少なくともサーバをいじっていてログを読まなければならないときに、grepも正規表現も使えないとかなり遠回りをするハメになります。

grepについては由来等々を、『シェルプログラミング実用テクニック』[注1]に書きましたので、そちらもご参考に。また、grepには、あるディレクトリの下のファイルをすべて検索する-rオプションなど、さまざまなオプションがあります。これもmanやインターネットで調べると良いでしょう。

head、tail、sort（上からn行・下からn行を抽出、並び替え）

さて、さらにコマンドを紹介していくとともに、組み合わせの例も示していきます。端末を使ってサーバのログなどを見ていると、行数が多くて見たいところをピンポイントでなかなか見られないことがよく起こります。先ほどのgrepで件数を落とすというのが一番先に思いつく方法ですが、ここではデータを見たい順にソートして取り出すということを扱ってみます。

たとえば、ls -l /etc/と打つと/etc/ディレクトリ下のファイルやディレクトリの一覧が見られますが、このときの順番はファイルの名前順です。これをファイルの更新日順に表示したいときは、**図9**のようにls -ltと打ちます。

ただ、この出力もザーッと画面を流れていって読めません。このときは**図10-①**のようにheadを使うと良いでしょう。ファイルの下のほう（この場合、更新日の古いほう）はtailで出せますので、これも**図10-②**に示しておきます。

図10の場合、lsがソートをしていますが、もっと汎用的にsortコマンドを使うという方法があります。たとえば**図11**は、ls -lの出力の5列目（ファイルサイズ）でls -lの出力をソートして、その出力をさらにtailに通して下から3行（サイズ大きいほうから3個のファイル）を出力しています。sortのオプション-k5,5nは、5列目（-k5,5で指定）を数字の小さい順（-nで指定）でソートしろという意味です。

もう1つ、行数の大きいデータを端末で眺めたいときには、lessというコマンドを使う方法があります。これは説明をほかに譲りますが、頻繁に使う方法です。

sed（文字列の置換やその他の編集）

これまで見てきたコマンドはテキストを表示するだけですが、sedを使うと文字を加工できます。**図12**はsedを使い、/etc/shellsから、コマ

▼図9　ディレクトリの一覧をファイルの更新日順に表示

```
$ ls -lt /etc/
合計 760
-rw-r--r-- 1 root root     733  3月 17 11:12 mtab
drwxr-xr-x 2 root root    4096  5月 29  2014 default
drwxr-xr-x 6 root root    4096  5月 29  2014 apt
-rw-r----- 1 root shadow   549  5月 29  2014 gshadow
-rw-r--r-- 1 root root     658  5月 29  2014 group
(...略...)
```

▼図10　headでlsの出力の先頭だけを表示

```
↓①「head -n 3」で上3行を表示（-nを付けないと10行出力になる）
$ ls -lt /etc/ | head -n 3
合計 760
-rw-r--r-- 1 root root     733  3月 17 11:12 mtab
drwxr-xr-x 2 root root    4096  5月 29  2014 default
↓②「tail -n 2」で下2行を表示
$ ls -lt /etc/ | tail -n 2
-rw-r--r-- 1 root root     356  1月  2  2012 bindresvport.blacklist
-rw-r--r-- 1 root root    2570  8月  6  2010 locale.alias
```

注1）上田隆一（著），『シェルプログラミング実用テクニック』，技術評論社，2015年5月発売予定．

▼図11　sortでファイルサイズ順に並び替え

```
$ ls -l /etc/ | sort -k5,5n | tail -n 3
-rw-r--r-- 1 root root   17978  2月 14 22:13 ld.so.cache
-rw-r--r-- 1 root root   19558 12月 30  2013 services
-rw-r--r-- 1 root root   23922  5月 13  2013 mime.types
```

▼図12　sedで文字を加工

```
$ cat /etc/shells | grep -v '^#' | sed 's;.*/;;'
sh
dash
bash
rbash
tmux
screen
```

ンドの名前(たとえば/bin/shのshの部分)だけを取り出した例です。

ちょっとこの例のsedは難しいのですが、sed 's<delim><置換したい文字列の正規表現><delim><置換後の文字列><delim>'で入力された文字列を置換して出力します。<delim>というのは区切りの文字のことで、**図12**の例では「;」を使っています。置換の命令の先頭にある「s」ですが、これはsubstitution(置換)の頭文字です。ほかにはy、p、dなどを先頭に付けたり最後に付けたりすると、さまざまな変換ができます。

これまでの例では、どちらかというとシステムが出力するテキストを処理してきましたが、sedは人が書いた文章に対しても威力を発揮します。たとえば**図13**のように、表記揺れのひどい3つのテキストファイルを作って修正してみましょう。ファイルが1つならVimやEmacsなどのエディタで修正すれば良いのですが、この場合はファイルが複数あります。そして、**図13**のような1ファイル1行ではなく、もっと大量の文章が書いてあるファイルだと、手作業ではたいへんになります。

▼図13　表記が揺れまくっている文章を作成

```
↓「> a.txt」でコマンドの出力がa.txtに書き込まれる(後述)
$ echo 1箇所2カ所3ヶ所 > a.txt
$ echo 4～5個所 > b.txt
$ echo 6~7カ所 > c.txt
```

さて、修正しましょう。sedには-iというオプションがあり、これを使うと上書きでテキストを修正してくれます(**図14**)。-iの後ろに付けた.bakは、.bakという拡張子を付けて元のファイルをバックアップしろということです。また、引数で指定した置換の命令の後ろに「g(グローバル)」とありますが、これは1行で何回でも置換しろという意味になります。gがないと置換の回数は1行で1回になります。

ところで、**図14**ではsedで処理するファイルを「a.txt b.txt c.txt」といちいち書きましたが、これは**図15**のように略記できます。誌面の都合上、詳しくは説明できませんが、「ブレース

▼図14　sed -eを使って一気に修正

```
↓-eの後ろに条件を書いていく
$ sed -i.bak -e 's/[箇カ個]所/ヶ所/g' -e 's/~/～/g' a.txt b.txt c.txt
↓バックアップファイルができていることを確認
$ ls
a.txt  a.txt.bak  b.txt  b.txt.bak  c.txt  c.txt.bak
↓修正を確認
$ cat a.txt b.txt c.txt
1ヶ所2ヶ所3ヶ所
4～5ヶ所
6～7ヶ所
```

▼図15　ブレース展開、ワイルドカード展開

```
↓ブレース展開
$ sed -i.bak -e 's/[箇カ個]所/ヶ所/g' -e 's/~/～/g' {a,b,c}.txt
↓ワイルドカード展開
$ sed -i.bak -e 's/[箇カ個]所/ヶ所/g' -e 's/~/～/g' *.txt
$ sed -i.bak -e 's/[箇カ個]所/ヶ所/g' -e 's/~/～/g' [a-c].txt
```

▼図16　例題のテキスト

```
$ cat ips
192.168.1.1
192.168.1.11
192.168.1.5
192.168.1.1
192.168.1.5
192.168.1.7
```

▼図17　各IPアドレスの個数の集計

```
$ cat ips | sort | uniq -c
      2 192.168.1.1
      1 192.168.1.11
      2 192.168.1.5
      1 192.168.1.7
```

展開」「ワイルドカード展開」「ファイルグロブ」という言葉でインターネットで検索をかけると良いでしょう。

uniq、wc（数を数える）

今度はテキストから何かを集計するという問題を扱ってみます。ログファイルからsedを駆使してIPアドレスを抽出したあとという想定で、**図16**のようなファイルを例題にやってみます。

最初にIPアドレスがいくつあるかを数えてみましょう。**図17**のようにsortとuniq -cを使います。uniqは重複する行を除去するコマンドですが、-cというオプションを付けるとこのように同じ行が何行あるか数え、1列目に表示してくれます。uniqの前にsortするのは、uniqの重複チェックは同じ行が連続することを前提にしているからです。

次にIPアドレスが何種類あるか数えます（**図18**）。今度はuniqをオプションなしで使い、その出力をwc -lに入力します。wcは入力されたテキストのバイト数や文字数、行数、単語数等々を出力するコマンドで、-lは行数を出すオプションです。**図17**、**18**のように、機転を利かせて少しコマンドの使い方を変えるだけで、さまざまなものを数えられます。

ちょっとwcの小技を書いておくと、日本語環境の場合、wc -mで日本語の文字数を数えられます。字数制限のある文章を提出しろと言われたときに利用すると良いでしょう。**図19**に例を示します。

▼図18　IPアドレスが何種類あるのかを集計

```
↓まず重複を除去して……
$ cat ips | sort | uniq
192.168.1.1
192.168.1.11
192.168.1.5
192.168.1.7
↓wc -lで行数を求める
$ cat ips | sort | uniq | wc -l
4
```

▼図19　wc -mで文字数をカウント

```
$ echo あいうえお | wc -m
6 ←最後に改行が入って6文字になるので注意
↓入力から改行をとる方法
$ echo あいうえお | tr -d '\n' | wc -m
5
```

▼図20　列の入れ替え

```
$ cat ips | sort | uniq -c | awk '{print $2,$1}'
192.168.1.1 2
192.168.1.11 1
192.168.1.5 2
192.168.1.7 1
```

▼図21　数字を比較して行を抽出

```
$ cat ips | sort | uniq -c | awk '$1==2{print $2,$1}'
192.168.1.1 2
192.168.1.5 2
```

awk（ツブシを効かせる十徳ナイフ）

スペースやタブ区切りのデータを扱うときにはawkが便利です。たとえば、**図20**のようにすると1列目と2列目を入れ替えることができます。awkの引数に渡した{print $2,$1}はプログラムで、「毎行について2列目と1列目を出力しろ」というコードになっています。

また、awkはgrepよりややこしい検索にも使われます。**図20**を変化させて、個数が2個のIPアドレスを出力させます（**図21**）。検索条件に列を指定できること、検索条件で数値を比較できることがミソです。

awkについてはLesson3以降でじっくり取り組むそうですので、これくらいにしておきます。「コマンドを組み合わせてもやりたいことがで

▼図22　CSVを取り込んでスペース区切りにして保存

```
↓とりあえず読めるように変換
$ cat hoge.csv | nkf -wLux
あいう,えお,3.14
かきく,けこ,2.71$
↓最後に改行を入れてtrでカンマをスペースに
$ cat hoge.csv | nkf -wLux | awk '{print}' | tr ',' ' '
あいう えお 3.14
かきく けこ 2.71
$
↓hogeというファイルに保存しておく
$ cat hoge.csv | nkf -wLux | awk '{print}' | tr ',' ' ' > hoge
```

▼図23　スペース区切りのデータをExcel用のCSVに変換

```
↓変更を加えてカンマ区切りにする
$ cat hoge | sed 's/あい/愛/g' | tr ' ' ','
愛う,えお,3.14
かきく,けこ,2.71
↓Shift JISに変換して保存
$ cat hoge | sed 's/あい/愛/g' | tr ' ' ',' | nkf -sLwx > hoge2.csv
```

▼図24　Macでhoge2.csvをダウンロードしてExcelに表示

```
↓scpでLinux側のファイルを手元のMacにコピー
uedambp:~ ueda$ scp 192.168.1.5:~/hoge2.csv ./
ueda@192.168.1.5's password: ←パスワードを入力
hoge2.csv                     100%   34     0.0KB/s   00:00
↓Excelがインストールされていれば、openでExcelが立ち上がる
uedambp:~ ueda$ open hoge2.csv
uedambp:~ ueda$
```

きないぞ」というときは、awkにプログラムを書いて渡せるので、なんとかなる場合が大半です。PerlやRubyでも同じようなことができますので、慣れている場合はそちらを使っても良いでしょう。

nkf（WindowsやExcelと仲良くしてやる）

さて、これまではASCIIやUTF-8のテキストデータを扱ってきましたが、Linuxの外には別の形式のテキストもあります。このようなテキストを扱うときは、nkfというツールが使えます。

たとえば、ExcelでCSVを保存してLinuxやMacのターミナルでいじるとき、そのままcatすると次のように文字化けします。

```
$ cat hoge.csv
??????,????,2.71$
```

これはExcelで扱われる文字コードがShift JISだからです。こういうときは理屈抜きにnkf -wLuxと打っておけば大丈夫です。-w、-Lu、-xの意味は、それぞれ「UTF-8に変換」、「UNIXの改行コード(LF)に変換」、「半角カナを全角に変換しない」です。**図22**にCSVをスペース区切りのUTF-8のテキストにする方法を示します。もちろん、これで済むのはデータ中にカンマのない単純なCSVだけですが、ご了承を。awk '{print}'はデータの最後に改行を入れるための小技です。

さて、**図22**で保存したhogeに変更を加えて、CSVに戻してやりましょう(**図23**)。今度は-sLwxというオプションをnkfに渡します。-sはShift JISに変換、-LwはWindowsで使われる改行コード(CRLF)に変換しろという意味です。

図23で終わると味気ないので、筆者のMacにhoge2.csvを持ってきて開くという端末操作を**図24**に示しておきます。Windowsの場合は、ファイルを転送できるツールを見つけてインストールして試していただければ。SD

Lesson 2 手を動かしてデータを操ろう！

テキストを自在に加工するために

コマンドを自在に組み合わせるテクニック

コマンドが本当に威力を発揮するのは、コマンド同士を自由に連携させられるようになったときです。Lesson2では、標準入出力のしくみを整理したうえで、さまざまなコマンドの連携手段を紹介します。これらを使いこなせれば、テキストデータから必要な部分を抽出し、書き換え、好みの順に並び替えるといった加工が自在にできるようになります。

Author 上田 隆一(うえだ りゅういち)　産業技術大学院大学／USP研究所／USP友の会
Twitter @ryuichiueda

コマンド・ファイルの連結方法いろいろ

さて、Lesson2ではファイルやコマンドを自由に使いこなすために、シェルの操作について基本的なものから、変わったものまで雑多に紹介していきます。

リダイレクト

まずLesson1の図13、22でも使った「>」から。これはリダイレクト記号と呼ばれ、Lesson1の図22では、コマンドの出力をファイルに保存するときに使いました。普通、コマンドの出力は画面に出てきますが、この出る口の方向をファイルに変えるときにリダイレクトが使われます。リダイレクトにはほかにも種類がありますが、これを理解するにはコマンドの入出力について理解する必要があります。

コマンドは、今まで単に出力と言ってきた「標準出力」と、入力を受け付ける「標準入力」、エラーを出力するための「標準エラー出力」と、3個のデータの受け渡し口を持ちます。またこれらの口には、0(標準入力)、1(標準出力)、2(標準エラー出力)という番号(ファイル記述子)が割り振られています。これを知っていると、パイプでコマンドをつなぐということは、「左側のコマンドの標準出力(1番)と右側のコマンドの標準入力(0番)を接続する」ことだとわかります。

リダイレクトは、パイプと同様、この番号に対応する入出力(画面やファイル)を切り替える操作です。Lesson1の図13で次のような操作が出てきましたが、これはechoの出力をリダイレクト(=向きを変更)して、a.txtに貯めるというものでした。

```
$ echo 1箇所2カ所3ヶ所 > a.txt
↓実は「>」は「1>」の略記
$ echo 1箇所2カ所3ヶ所 1> a.txt
```

図1に、標準入力、標準エラー出力の操作例も示しておきます。

ファイル記述子を使うと、出力のつなぎ変えを自由自在に行えるようになります。次の例は、rev[注1]の出したエラーをパイプに通して別のrevで処理させたものです。

▼図1　標準入力と標準エラー出力

```
↓a.txtをrevの標準入力に向ける
$ rev < a.txt
所ヶ3所カ2所箇1
↓「<」は「0<」の略記
$ rev 0< a.txt
所ヶ3所カ2所箇1
↓標準エラー出力の例
$ rev -x < a.txt
rev: 無効なオプション -- 'x' ←エラーが出てくる
(...略...)
↓標準エラー出力は「2>」でファイルにリダイレクトできる
$ rev -x < a.txt 2> hoge
```

注1) 入力された文字列を逆さまにして出力するコマンド。

```
$ rev -x < a.txt 2>&1 | rev
'x' -- ンョシプオな効無 :ver
(...略...)
```

図2に、さらにややこしいリダイレクト操作の例を示します。lsで、存在するa.txtと存在しないz.txtを指定したものです。lsは標準出力にa.txtのファイル名、標準エラー出力に「z.txtがない」というエラーを出力します。**図2**のリダイレクトの例では、エラー出力をパイプに通し、元の標準出力をファイルに貯めるという操作をしています。リダイレクト操作は左から読んでいくと理解できます。まず、2>&1で、2番を今1番が指している先(パイプ)に向くようにします。そのあとで、> existで1番がファイルexistに向きます。

図3では、**図2**のリダイレクト操作の順番を入れ替えてみました。これだと2>&1のときに1番の向いている先はファイルexistですので、lsの出力はすべてファイルに入っていきます。

パイプからの入力を示す「-」

コマンドの多くはファイルを引数にとりますが、ファイルの代わりにパイプの出力を指定したいときは、「-」という表記がコマンド側に準備されていることがほとんどです。たとえば、Webサーバ(Apache)の吐くログは古いものは圧縮されて、新しいものは圧縮されていないのですが、それをくっつけて出力するときは、次のように打ちます[注2]。

```
$ zcat access.log-20150304.gz | cat - access.log
(ログが次々と表示される)
```

cat - access.logというように「-」がaccess.logの前にあるので、パイプを伝わってきた古いログが先に出力され、あとからaccess.logが出力されます。

この機能がないと面倒な場合があります。たとえば、zcatの出力とcatの出力を一緒にしてさらに別のコマンドにパイプで渡すときは、

```
$ ( zcat access.log-20150304.gz ; cat access.log ) | wc -l
16290
```

というように「( )」でコマンドをまとめるか、次に紹介するプロセス置換を使わなければなりません。ほかのコマンドでも「-」はたいてい使えますので、いろいろ試してみると良いでしょう。

プロセス置換(邪道)

普通、パイプを使っていると処理は一方通行ですが、シェルによっては抜け道が用意されています。たとえばbashだと<( )という記号を使うと、あるコマンドの出力を別のコマンドに渡すことができます。この機能は「プロセス置換」と呼ばれます。

使用例を**図4**に示します(ファイルa.txtとa.txt.bakは**Lesson1の図14**で出てきたもの)。diffは2つのファイルを引数にとってファイルの違いを表示するコマンドですが、**図4**ではファイル名を書く位置に<( )で囲まれたsedの処理(「所」という文字の

▼図2　複雑なリダイレクト操作

```
$ ls a.txt z.txt
ls: z.txt にアクセスできません(略) ←標準エラー出力
a.txt ←標準出力
↓2番を1番の向いているほうに向け、その後1番をファイルexistに向ける
$ ls a.txt z.txt 2>&1 > exist | rev
んせまりあはリトクレィデやルイ(略) ←エラーがパイプを通る
$ cat exist
a.txt ←標準出力はファイルへ
```

▼図3　リダイレクトの順序を変えると挙動が変わる

```
↓順序を変えると両方の出力がexistに入る
$ ls a.txt z.txt > exist 2>&1
$ cat exist
ls: z.txt にアクセスできません(略)
a.txt
↓両方の出力を1つにまとめたいならbashの場合、このように書ける
$ ls a.txt z.txt &> exist
$ cat exist
ls: z.txt にアクセスできません(略)
a.txt
```

注2) zcatは圧縮されたファイルの内容を表示するコマンド。

後ろに改行を入れる処理)が書かれています。

プロセス置換の乱用はわけのわからないコードを生む原因になってしまいますが、余計な中間ファイルを作らないで処理をしたいときなどに用いると便利です。

パイプからのデータを引数に

while read

コマンドを使いこなすようになると、パイプから通ってきたデータを別のコマンドに引数として渡したいことがちょくちょく出てきます。簡単に思いつく(しかし、ややこしい)方法はwhile read ……というイディオムを使う方法です。たとえばa.txt、b.txt、c.txtを、それぞれa.txt.bak、b.txt.bak、c.txt.bakとdiffする場合、慣れている人なら**図5**のようにコマンドを打ちます。ls ?.txtで「任意の1文字＋.txt」のファイルのリストができます。それをwhile read fで変数fに順番にセットして、while文の中でdiff $f $f.bakというように1行ずつdiffの引数に渡しています。

図5をちょっと補足すると、whileに渡すデータは、1回に渡す分ごとに改行が入っている必要があります。ls ?.txtの出力は横に並んで1行で出力されているように見えますが、これはlsが端末上に表示するときに小細工をしているからで、実際にコマンドにlsの出力が渡るときは、次のように改行が入って渡ります。

```
$ ls ?.txt | cat
a.txt
b.txt
c.txt
```

ですから、**図5**のwhile readは3行のデータを受け取っています。**図5**の下の例では、echo ?.txtでファイルのリストを作っていますが、echoの場合は表示上もデータ上も改行が入らないので、tr ' ' '\n'で改行を入れる必要があります。

xargs

さて、while readの例は何にでも使えるのですが、doやdoneなどいろいろと書かなければなりません。xargsを使うともっと簡単にできます。**図6**に、**図5**と同じ処理をxargsで行ったものを示します。-I@は、コマンドに引数を渡すときに、「@」と書いた位置にパイプから受け取ったデータを当てはめろという意味です。

while readとの使い分けに関して、xargsはあまりにも切れ味が鋭いので、mvやrmなど後戻りできない処理を慎重にやりたい場合は、while文の中で処理をechoで出すなどいろいろ確認してから実行するということをやる人がいます。ただ、処理が走り出してしまうとどちらを選んでも同じです。

xargsの使い方をもう少し示

▼図4　diffに2つのsedの出力を渡す

```
$ diff <(sed 's/所/&\n/g' a.txt) <(sed 's/所/&\n/g' a.txt.bak)
1,2c1,2
< 1ヶ所
< 2ヶ所
---
> 1箇所
> 2カ所
↓プロセス置換を使わないと、ゴミファイルが発生するなど面倒
$ sed 's/所/&\n/g' a.txt > aa
$ sed 's/所/&\n/g' a.txt.bak | diff aa -
(...出力は上の例と同じ...)
```

▼図5　while readの使い方

```
↓まずファイルのリストを作る
$ ls ?.txt
a.txt  b.txt  c.txt
↓そのままwhileに渡す
$ ls ?.txt | while read f ; do diff $f $f.bak ; done
1c1
< 1ヶ所2ヶ所3ヶ所
---
> 1箇所2カ所3ヶ所
1c1
(...略...)
↓echoでファイルのリストを作る場合は、改行を入れる必要あり
$ echo ?.txt | tr ' ' '\n' | while read f ; do diff $f $f.bak ; done
(...出力は上の例と同じ...)
```

▼図6　xargsの使い方

```
$ ls ?.txt | xargs -I@ diff @ @.bak
1c1
< 1ヶ所2ヶ所3ヶ所
---
(...略...)
```

▼図7　xargsの使い方2（適当な個数の引数を渡す）

```
↓普通にtouchに渡すと叱られる
$ touch file.{1..10000000}
-bash: /usr/bin/touch: 引数リストが長すぎます
$ echo file.{1..1000000} | xargs touch
$
$ ls -U ←lsにソートさせないで出力させるときは-Uを使う
file.935081
file.947716
file.268148
file.266114
(以下略。飽きたらCtrl+cで止めましょう)
↓指定した個数分ずつ引数を渡すには-nを使用
$ echo file.{1..1000000} | xargs -n 2 touch
(touchが50万回呼ばれるので当面終わらない)
```

します。-I@がなければ、xargsは受け取ったデータを適当な個数の引数に変換してコマンドに渡します。**図7**は、適当なディレクトリにtouchコマンドで空ファイルを100万個作る例です。file.{1..10000000}でfile.1 file.2……file.10000000という意味になりますが、これをそのままtouchに渡すと引数が多過ぎて引き取ってくれません。xargsを使うと、適当な長さで引数をtouchに渡し、適当な回数だけtouchを呼び出してくれます。

ちなみに、「適当な長さ」というのは、

```
$ getconf ARG_MAX
2097152
```

とやると調べられます。説明はほかに譲ります。

そして、某シェル芸勉強会で大ブーム（？）のxargsの使い方を最後に紹介しておきます。xargsの右に何もコマンドを指定しないと、echoしてくれます。このときに-nオプションを指定すると、指定した数ごとにechoされるので、**図8**のようにデータを整列させられます。

ただし、**図9**のようにechoのオプションを書いたデータを渡すという意地悪をすると意図しない出力になることがあるのでご注意を。

▼図8　xargsの使い方3（整列）

```
↓このようなechoの出力が……
$ echo {1..10}
1 2 3 4 5 6 7 8 9 10
↓3列になったり
$ echo {1..10} | xargs -n 3
1 2 3
4 5 6
7 8 9
10
↓4列になったりします
$ echo {1..10} | xargs -n 4
1 2 3 4
5 6 7 8
9 10
```

▼図9　-nや-eがechoのオプションになる例[注3]

```
$ cat hoge
-n 1
-e 2
$ cat hoge | xargs -n 2
12
```

▼図10　メタプログラミングの例

```
↓次のようにシェルスクリプトを作り……
$ ls ?.txt | awk '{print "diff",$1,$1".bak"}'
diff a.txt a.txt.bak
diff b.txt b.txt.bak
diff c.txt c.txt.bak
↓shに投げる
$ ls ?.txt | awk '{print "diff",$1,$1".bak"}' | sh
1c1
< 1ヶ所2ヶ所3ヶ所
---
> 1箇所2カ所3ヶ所
(...略...)
```

コードを作ってshにぶち込む

さて、while readとxargsは標準入力からのデータを引数に変換するときに使いましたが、標準入力からのデータをプログラムに変換するという荒業も存在します。簡単に例だけを**図10**に示しておきます。

データの流れを意識する

最後に上級者のために議論の火種を放り込ん

注3）@mutz0623さんのLT資料「xargsコマンドの細かい話」より
URL https://docs.google.com/file/d/0B7vwFIP4k87kZmRMVTlsU3ZtV3M/edit

でおきます。シェルの操作を駆け足で説明してきましたが、Lesson1の冒頭で述べたように、これはプログラミングの一種です。ただ、多くのプログラマが慣れ親しんでいる方法とはずいぶん異なります。果たしていったい我々は何をしているのでしょうか？　いつも筆者は「これはデータフロープログラミングだ」だとか「これはソフトウェアツールだ」という言い方をしているのですが、ここではまた別の説明をするために、いわゆるSICP[注4]の「2.2.3 Sequences as Conventional Interfaces」を持ち出します。

詳しくは本の内容がWebで公開されているので、それを読んでいただきたいのですが、ここで述べられていることをざっくり言うと、

- 違う処理のようでも、やっていることを分解すると似たような処理が出現する
- その似たような処理ひとつひとつを切り分け、電気信号を処理する素子のように一方通行で接続するとコードの見通しと再利用性が良くなる

ということです。SICPの2.2.3項にはコマンドもパイプも出てきませんが、言っていることはコマンドとパイプです。

コマンドをパイプでつないだ処理では、基本的に1行がリストの1つの要素とみなされます。そのリストがgrepのような「filter(ろ過器)」のコマンドや、awkで行う毎行処理のような「transducer(変換器)」、wcのような「accumulator(累算器)」を通って、何かしらの結果が出てきます。

これを通常のプログラミング言語(おもに1つのプロセスを使うために特化された既存の言語)で表現するのは結構難しい処理になります。たとえば

```
$ cat data
9
10 2
4 8 7
1
```

という数字が7個書いてあるデータがあるとします。5以上の数字と5未満の数字がそれぞれ何個あるか求めようとすると、コマンドならば、**図11**のような処理が考えられます。これは、「データの整列・分類・並び替え・数え上げ」という各プリミティブな処理を一方通行につなげた処理で、SICPの2.2.3項的な処理です。

一方、この処理はawkだけでも、**図12**のように書けます。ちょっと見にくいので**リスト1**にちゃんとプログラムを書いたものを示します。コードの詳細は読めなくてもかまいませんが、for文が出てきて、その中で集計処理が行われていることがわかると思います。

このfor文は、**図11**の最初のxargs -n 1(各数字を1行1個に並べ直す)が使えたならもっと簡略化できるのですが、これに相当するものを採用すると**リスト2**のようなコードになります。for文が2つ現れてちょっとやり過ぎのような気もします。少なくともデータの流れが最初のfor文で一度止まるので、「電気信号」の如く処理できているわけではありません。

また、**リスト2**の処理は、もっとたくさんのデータをさばこうとした場合、配列に中間データをいちいち保存することになりメモリが無駄です。**図11**の処理もソートでメモリを使うので、この例では大差ないのですが、やはり普通の言語を使うと**リスト1**のような書き方になります。「信号処理のような処理」を簡単に表現したいのならば、やはり標準入出力とパイプを使う方法

▼図11　5以上と5未満の数を集計

```
$ cat data | xargs -n 1 | awk '$1>=5{print "5以上"}$1<5{print "5未満"}' | sort | uniq -c
      4 5以上
      3 5未満
```

注4) H. Abelson and G. J. Sussman, "Structure and Interpretation of Computer Programs (Second edition)",MIT Press,1996.
URL https://mitpress.mit.edu/sicp/full-text/book/book.html

▼図12　5以上と5未満の数を集計（AWKワンライナー）

```
$ awk '{for(i=1;i<=NF;i++){if($i>=5){a+=1}else{b+=1}}}END{print "5以上",a;print "5未満",b}' data
5以上 4
5未満 3
```

▼リスト1　5以上と5未満の数を集計（AWKスクリプト）

```
#!/usr/bin/awk -f

{
  for(i=1;i<=NF;i++){
    if($i>=5) a+=1;
    else      b+=1;
  }
}
END{
  print "5以上", a;
  print "5未満", b;
}
```

が自然かなということになります。また、最近はCPUのコア数が増えているので、「信号処理の如く」がだんだん有利になっている点も見逃せません。このような視点でシェルでのテキスト処理を見ると、シェルでのコマンド操作という古風な方法が、実はいまだにユニークな存在であることを確認できるのではないでしょうか。

終わりに

Lesson1、2では、シェル（端末）でのテキスト処理について、はなはだ駆け足ですが、説明を行いました。世の中にはさまざまなデータがあり、本稿で扱ったような1行1レコードの単純なリストではない、木構造状のデータも多く存在します。そのようなデータについても、たとえば**図13**のfindの出力を見ればわかるように、1行1レコードのリストにすることは簡単です。「これ冗長で非効率じゃないか？」と思われるかもしれませんが、あとはコンピュータの馬鹿力とパイプの本質的な効率の良さに任せてしまえば、木構造を扱うプログラムも書かなくて済みます。

コマンドを使うテキスト処理をやろうとすると、これまで説明したように少々普通ではないプログラミングをしなければなりません。しかし、単に古いとか、コマンドが雑多であるとか、不統一であるとか、そういう理由で触らないよ

▼リスト2　5以上と5未満の数を集計（xargs -n 1相当の処理をもとのfor文から分離）

```
#!/usr/bin/awk -f

{
  n = 1;
  for(i=1;i<=NF;i++)
    num[n++] = $i;

  for(i=1;i<n;i++)
    if(num[i] >= 5) a++;
    else            b++;
}
END{
  print "5以上", a;
  print "5未満", b;
}
```

▼図13　findの出力は木構造でもあり、単純なリストでもある

```
$ sudo find /etc/ | head -n 5
/etc/
/etc/perl
/etc/perl/Net
/etc/perl/Net/libnet.cfg
/etc/perl/CPAN
```

うにしていたのであれば、ちょっと視点を変えて、使ってみてはいかがでしょうか。

◆　◆　◆

さて、5月上旬に『シェルプログラミング実用テクニック』が上梓予定です。これは本誌読者層のみなさんにコマンドを組み合わせてテキスト処理を行うための基礎力を養っていただこうと、筆者が書き上げたものです。

この本、各章で1つ処理の対象を決めてお題をいくつか提示し、ワンライナーやシェルスクリプトで解決するという構成になっています。処理の対象は、文章や設定ファイル、CSV、Web上のデータなど基本的なものをおさえました。が、一方で画像やExcel、Wordファイルの処理にも挑戦しています。雑多にやり散らかした感じが出てしまったのが反省点ですが、ぜひ本書で楽しみながらコマンド操作を覚えていただければ幸いです。SD

Case 1 手を動かしてデータを操ろう！

現場で使われるテキスト処理の実際

grepで検索！ ソースコードを効率的に読む方法

Author 中井 悦司(なかい えつじ)　レッドハット㈱
Twitter @enakai00

カーネルソースを読むという仕事

オープンソースにかかわるエンジニアの仕事というと、まず始めに思い浮かぶものは何でしょうか？　開発者として「ソースコードを書く」という仕事を思い浮かべる読者も多いかもしれませんが、筆者の場合は「ソースコードを読む」ことのほうが多いです。たとえば、Linuxカーネルのログメッセージについて、その意味を教えてほしいという問い合わせを受けて、おもむろにカーネルソースを読み始めるという感じです。

このときに役立つのは、言うまでもなくgrepコマンドです。特定のメッセージを含む行を多数のソースファイルからまとめて検索できます。具体例を用いて、実際に体験してみましょう。

grepコマンドによる探索例

業務の一部というわけではありませんが、実際の例として先日、筆者のブログに次のような内容の質問コメントが寄せられました。

> カーネルの起動メッセージにメモリ容量を示す**図1**のような行がありました。それぞれの数値は何を表しているのでしょうか？

▼図1　カーネルの起動メッセージ

```
Memory: 2041336k/2105344k available (2536k kernel code, 55360k reserved, 1751k data, 196k init)
```

このようなログメッセージの調査には、grepコマンドがうってつけです。まずは、Linuxカーネルのソースコードを GitHubからダウンロードしてきます。ここでは例として、バージョン3.10のソースコードをチェックアウトしておきます。

```
$ git clone https://github.com/torvalds/linux
$ cd linux
$ git checkout v3.10
```

Linuxカーネルには、多数のソースファイルがありますが、grepコマンドでは、-rオプションを用いるとディレクトリを再帰的にたどりながら、すべてのファイルをまとめて検索できます。質問にあったログメッセージの一部を検索キーワードにして、次のコマンドを実行してみます。

```
$ grep -Hr "Memory: .* available" ./
```

これは、カレントディレクトリ「.」を起点としてすべてのサブディレクトリを検索します。-Hオプションは、検索にマッチしたファイル名を表示するオプションです。実際にコマンドを実行すると、ディレクトリarch以下の複数のファイルがマッチします。これは、アーキテクチャ別のコードが収められたディレクトリですので、ここではx86アーキテクチャのディレクトリにある**図2**のファイルに注目します。

▼図2　カーネルソース内の該当のファイル

```
./arch/x86/mm/init_64.c:    printk(KERN_INFO "Memory: %luk/%luk available (%ldk kernel code, "
```

▼リスト1　arch/x86/mm/init_64.c

```
1085    printk(KERN_INFO "Memory: %luk/%luk available (%ldk kernel code, "
1086                     "%ldk absent, %ldk reserved, %ldk data, %ldk init) n",
1087            nr_free_pages() << (PAGE_SHIFT-10),
1088            max_pfn << (PAGE_SHIFT-10),
1089            codesize >> 10,
1090            absent_pages << (PAGE_SHIFT-10),
1091            reservedpages << (PAGE_SHIFT-10),
1092            datasize >> 10,
1093            initsize >> 10);
```

▼リスト2　arch/x86/mm/init_64.c

```
1077    codesize =  (unsigned long) &_etext - (unsigned long) &_text;
1078    datasize =  (unsigned long) &_edata - (unsigned long) &_etext;
```

エディタでファイルを開くと、該当部分は**リスト1**のようになっています。1085行目のprintkは、カーネル内部で利用されているカーネルメッセージの出力関数です。これで、メッセージの各項目に対応する変数(1,087～1,093行目)がわかりました。あとは、これら変数の定義部分を探しだして、それらの意味を調べていきます。

たとえば、codesizeとdatasizeについては、同じファイルのすぐ上の部分で**リスト2**のように定義されています。コンパイラ(リンカ)の知識がある人なら、「_text」「_etext」「_edata」などは、カーネルをコンパイルしたバイナリファイル内部のテキストセクションやデータセクションの開始・終了位置を表すことに気がつきます。結果、codesizeは、カーネル内部の実行コード領域のサイズ、datasizeは、静的データ領域のサイズを表すことがわかります。そのほかの調査結果は、ブログのコメント[注1]を参照してください。

grepコマンドの応用例

grepコマンドには、このほかにも覚えておくとよいオプションがあります(**表1**)。一度ビルド処理を行ったディレクトリ内は、ソースファイルとコンパイル済みのバイナリファイルが混じった状態になります。このときgrepコマンドは、バイナリファイルの中身まで検索してしまいます。このような際は、-Iオプションを指定して、バイナリファイルを検索対象外にします。

また、テスト用のコードが入ったtestsディレクトリがある場合、この中のファイルが大量にマッチすることがあります。testsディレクトリ内の

▼表1　grepコマンドの便利なオプション

オプション	説明
-r	ディレクトリを再帰的に検索する
-H	マッチしたファイル名を表示する
-I	バイナリファイルを検索対象外にする
-v	検索にマッチしない行を表示する
-E	拡張正規表現を使用する

注1)「/proc/meminfoを考える」 URL http://d.hatena.ne.jp/enakai00/20110906/1315315488

ファイルを除外したい場合は、次のようにもうひとつのgrepコマンドにパイプでつなぎます。

```
$ grep -Hr "keyword" ./ | grep -v tests
```

-vオプションは指定のキーワードを「含まない」行だけを表示するオプションです。これで、最初のgrepコマンドが出力するディレクトリ名に「tests」を含む行を除外しています。厳密には、ディレクトリ名以外の部分に「tests」が含まれる場合も除外されてしまいますが、ほとんどの場合は、これで用が足ります。

また、grepコマンドはその名前(Global - Regular Expression - Print)のとおり、正規表現(Regular Expression)で検索キーワードを指定します。このとき、-Eオプションを指定すると、「拡張正規表現」と呼ばれる、より複雑な正規表現が利用できます。たとえば、「foo」「bar」のどちらかを含む行を検索する場合は、次のようになります。

```
$ grep -Hr -E "(foo|bar)" ./
```

ソースコードにかかわる便利コマンド

Linuxのコマンドには、このほかにもソースコードを扱うのに便利なものがたくさんあります。たとえば、先ほどの**リスト1**では各行の先頭に行番号を付けていますが、これはcatコマンドの-nオプションを利用しています。次のように実行すると、各行の先頭に行番号を付けてファイルの内容を表示してくれます。

```
$ cat -n ./arch/x86/mm/init_64.c
```

また、運用業務にかかわっていると、数十行程度の短いスクリプトファイルを何本かまとめて作成することがあります。これらのスクリプトファイルをバックアップしたい場合、どうすればいいのでしょうか？　正式なバックアップは、tarコマンドで固めて取得するべきですが、作業ログ的に記録を残す場合、筆者は次のheadコマンドを実行します。

```
$ head -100 *.sh
```

これは、末尾が「.sh」で終わるファイルの先頭100行を表示するコマンドですが、複数のファイルがある場合、それぞれの出力の前にファイル名が付与されます。100行以内のファイルであれば、次のようにすべての内容がファイル名とともに画面に表示されます。

```
$ head -100 *.sh
==> script01.sh <==
#!/bin/bash
echo test01

==> script02.sh <==
#!/bin/bash
echo test02
```

この出力内容を手元のエディタにコピーすれば、バックアップは完了です。Windows PCからSSH端末ツールでログインしているときは、端末ツールの機能で、画面出力をログファイルに書き出しておいても良いでしょう。

◆ ◆ ◆

Linuxカーネルのソースコードを読むというと、前提知識が必要で難しいように思われるかもしれませんが、ここで紹介した例のようにログメッセージをキーワードにして、該当のメッセージ出力部分を調べるのは誰でも簡単にできることです。Linuxカーネルを勉強する手始めとして、ぜひ興味のあるメッセージを見つけて検索してみてください。もっと本格的にカーネルソースを読んでみたいという方は、筆者の書籍も参考にしてもらえると幸いです[注2]。SD

注2) 中井悦司(著),『プロのためのLinuxシステム・10年効く技術』—第4章　最後の砦！　カーネルソースを読む, 技術評論社, 2012.

Case 2

手を動かして データを操ろう！

現場で使われるテキスト処理の実際

コマンドを組み立て、Nginx・MySQLのログを読む

Author 吉川 竜太(よしかわ りょうた)　㈱ハートビーツ
Twitter @rrreeeyyy

はじめに

最近、ログ解析ツールやプロビジョニングツール、オーケストレーションツールが注目を集めています。これらのツールの内側では、本コラムで紹介する、区切り文字を用いたデータの分割・集計や、コマンドの出力結果を基にして次に実行するコマンドを組み立てるといったような基本的な技法が、言語は違えど多く用いられています。

コマンドの内容をひとつひとつしっかり理解して自分で組み立てることが、これらのツールを作ったり、中身を読み解いたりすることの手助けになると思います。もちろん、日々の作業の自動化・効率化にも寄与するので、しっかりと理解したうえで、使いこなせるようになるといいでしょう。

ここでは、Lesson1～4で紹介しているコマンドやawkなどが、MSP(Managed Service Provider)の現場ではどのように用いられているかの実例を、解説を含めながら紹介します。なお、本コラムで紹介するコマンドは、CentOS6.6にて動作確認を行っています。

Nginxのログ解析

まずは、基本であるWebサーバ(Nginx)のログを整形・集計してみます。ログ解析を始める前には、まずは解析対象のログをしっかり眺める必要があります。**図1**のようなNginxのアクセスログを例にして考えてみます。

最初に、拡張子「.js」「.ico」「.jpg」以外、つまりHTMLファイルやCSSファイルへのアクセス数を分単位で集計することを考えてみましょう。そのためにはまず、これらの拡張子が付いたファ

▼図1　Nginxのアクセスログ

```
# tail -n 10 /var/log/nginx/access.log
198.51.100.1 - - [11/Mar/2015:11:31:19 +0900] "GET / HTTP/1.1" 200 17638 "http://example.com/" "-" "-"
198.51.100.1 - - [11/Mar/2015:11:31:19 +0900] "GET / HTTP/1.1" 200 3881 "http://example.com/" "-" "-"
198.51.100.1 - - [11/Mar/2015:11:31:19 +0900] "GET /image.jpg HTTP/1.1" 200 1073 "http://example.com/"
"-" "-"
198.51.100.1 - - [11/Mar/2015:11:32:19 +0900] "GET / HTTP/1.1" 200 1205 "http://example.com/" "-" "-"
203.0.113.1 - - [11/Mar/2015:11:32:29 +0900] "GET /script.js HTTP/1.1" 200 1250 "http://example.com/"
"-" "-"
198.51.100.1 - - [11/Mar/2015:11:32:39 +0900] "GET / HTTP/1.1" 200 23416 "http://example.com/" "-" "-"
198.51.100.1 - - [11/Mar/2015:11:32:41 +0900] "GET /favicon.ico HTTP/1.1" 200 350 "http://example.
com/" "-" "-"
192.168.2.1 - - [11/Mar/2015:11:33:38 +0900] "GET / HTTP/1.1" 200 20552 "http://example.com/" "-" "-"
192.168.2.1 - - [11/Mar/2015:11:34:29 +0900] "GET / HTTP/1.1" 200 23416 "http://example.com/" "-" "-"
192.168.2.1 - - [11/Mar/2015:11:34:29 +0900] "GET /script.js HTTP/1.1" 200 1250 "http://example.com/"
"-" "-"
```

イルへのアクセスを除外する必要があります。

図1をよく見ると、アクセス対象のパスは両側がスペースで囲まれていることがわかります。スペースを区切り文字としてログを眺めてみると、ちょうど7列目にパスが来ています。そのため、実行するコマンドは**図2**のようになります。

awkのデフォルトでの区切り文字はスペース・タブであるため、このコマンドは、空白区切りの7列目が「.js」「.ico」「.jpg」で終わらない行を出力する、という意味となります。なお、awkについての詳細は、Lesson3、4などを確認してください。

次に、時刻ごとの集計について考えます。このときに必要なのは、ログの中の「11:32」「11:33」「11:34」といった部分になります。ログを見ると、これらの文字列は「:」で囲まれていることがわかります。そのため、区切り文字を「:」として考えるとコマンドを組み立てやすいでしょう。実行するコマンドとその結果は**図3**のようになります。このコマンドは、出力の各行を「:」で区切った結果の2、3列目を出力します。

最後に集計を行います。同一の行をカウントして出力するには、uniqコマンドの-cオプションを使用します(**図4**)。

また、欲しいデータに応じて、さらに出力をソートすることもできます。たとえば、前述のコマンドの出力結果から、もっともアクセスが多かった時間が知りたければ、コマンドの末尾にsort -k 1 -nを付けて実行することで実現できます(**図5**)。

本節で紹介したコマンドは、MSPの現場で、Webサーバの負荷増加が発生した場合などにアクセス数が増えていないかを調査する際に、実際に用いられています。

▼図2　空白区切りの7列目が「.js」「.ico」「.jpg」で終わらない行を出力

```
# tail -n 10 /var/log/nginx/access.log ➥
 | awk '$7 !~ /(\.js|\.ico|\.jpg)$/{print $0}'
```

▼図3　出力の各行を「:」で区切った結果の2、3列目を出力するようコマンドを追加

```
# tail -n 10 /var/log/nginx/access.log ➥
 | awk '$7 !~ /(\.js|\.ico|\.jpg)$/{print $0}' ➥
 | cut -d ':' -f 2,3
11:31
11:31
11:32
11:32
11:33
11:34
```

▼図4　同一の行をカウントして出力するようコマンドを追加

```
# tail -n 10 /var/log/nginx/access.log ➥
 | awk '$7 !~ /(\.js|\.ico|\.jpg)$/{print $0}' ➥
 | cut -d ':' -f 2,3 ➥
 | uniq -c
2 11:31
2 11:32
1 11:33
1 11:34
```

▼図5　出力をソートするようコマンドを追加

```
# tail -n 10 /var/log/nginx/access.log ➥
 | awk '$7 !~ /(\.js|\.ico|\.jpg)$/{print $0}' ➥
 | cut -d ':' -f 2,3 ➥
 | uniq -c ➥
 | sort -k 1 -n
```

MySQLのクエリをkill

MySQLで長時間実行されているSELECTを含むクエリを特定してkillする、といったオペレーションを考えてみます。

MySQLで実行されているクエリは、**図6**の1行目のようなコマンドで確認できます。ここでは、**図6**の出力を例にして考えてみます。すべての項目が「|」という文字区切りで出力されているため、とても解析しやすいですね。必要な項目である、クエリの実行時間とクエリの情報は、区切り文字を「|」とすると、6列目・8列目に来ていることがわかります。しかし、区切り文字である「|」を含まない項目が最初の列と最後の列にあるため、awkの列数のカウントが1つずれることに注意しましょう。

まずは行全体を出力してみて、出力されているクエリが条件にあっているかを必ず確認します(**図7**)。その後、killに必要なIDだけを出力します(**図8**)。

最後に、出力されたIDを元にして、killを実行するコマンドを組み立てます。killを実行する前に、

▼図6　MySQLで実行中のクエリ

```
# mysqladmin -u root processlist --verbose
+-----+------+-----------+----+---------+------+------------+----------------------+----------+
| Id  | User | Host      | db | Command | Time | State      | Info                 | Progress |
+-----+------+-----------+----+---------+------+------------+----------------------+----------+
| 984 | root | localhost | db | Query   | 23   | User sleep | SELECT SLEEP(60)     | 0.000    |
| 985 | root | localhost | db | Query   | 42   | User sleep | SELECT SLEEP(60)     | 0.000    |
| 986 | root | localhost | db | Query   | 48   | User sleep | SELECT SLEEP(60)     | 0.000    |
| 993 | root | localhost | db | Query   | 0    | init       | show full processlist | 0.000    |
+-----+------+-----------+----+---------+------+------------+----------------------+----------+
```

実際に実行されるコマンドをechoで確認してみましょう(**図9**)。xargsは標準入力からコマンドを生成し、実行できるコマンドです。-Iオプションは、行ごとに標準入力を受け取り、受け取った行を置きかえるための変数を指定するためのオプションです。

最後に、echoを外して実行することで、対象のクエリをkillすることができます(**図10**)。また、どのようなコマンドが実行されたかは、xargsの-tオプションで確認できます。

本節で説明したコマンドは、実際になんらかの理由でMySQLのSELECTクエリが長時間滞留してしまった場合に、障害対応としてクエリのkillを行う際に実行されることがあります。なお別解として、MySQLのクエリを用いて長時間実行されているSELECT句を出力したり、killすることもできます(**図11**)。詳細はここでは説明しませんが、ぜひ自分で読み解いてみてください。

◆ ◆ ◆

MSPの現場で実際に用いられているコマンドを例に取り、コマンドの組み立て方や、コマンドの出力結果を用いたオペレーションの実行を簡単に紹介しました。ここで紹介したことを覚えるのではなく、実際に自分で手を動かしてコマンドを組み立てることで、文字列のパースやパイプラインなどの、重要な概念を身につけることができるかと思います。ぜひ実際に手を動かしてみてください。SD

▼図7　長時間実行されているSELECT文を特定

```
# mysqladmin -u root processlist --verbose ↵
 | awk -F '|' '($7 > 30 && $9 ~ /SELECT/){print $0}'
```

▼図8　killに必要なIDだけを出力するように修正

```
# mysqladmin -u root processlist --verbose ↵
 | awk -F '|' '($7 > 30 && $9 ~ /SELECT/){print $2}'
```

▼図9　killを実行するコマンドを組み立て、ひとまずecho

```
# mysqladmin -u root processlist --verbose ↵
 | awk -F '|' '($7 > 30 && $9 ~ /SELECT/){print $2}' ↵
 | xargs -I%% echo "mysqladmin -u root kill %%"
mysqladmin -u root kill 985
mysqladmin -u root kill 986
```

▼図10　echoを外してkillを実行

```
# mysqladmin -u root processlist --verbose ↵
 | awk -F '|' '($7 > 30 && $9 ~ /SELECT/){print $2}' ↵
 | xargs -t -I%% mysqladmin -u root kill %%
```

▼図11　MySQLのクエリを用いて長時間実行されているSELECT句を出力し、killする

```
クエリIDの確認
# echo 'SELECT GROUP_CONCAT(ID) FROM PROCESSLIST WHERE TIME ↵
 > 30 AND INFO LIKE "SELECT %"' ↵
 | mysql -u root information_schema -N

クエリのkill
# mysqladmin -u root kill $(echo 'SELECT GROUP_CONCAT(ID) FROM PROCESSLIST WHERE TIME ↵
 > 30 AND INFO LIKE "SELECT %"' ↵
 | mysql -u root information_schema -N)
```

Case 3

手を動かしてデータを操ろう！

現場で使われるテキスト処理の実際

コマンドラインでJSONデータを作って利活用

Author 波田野 裕一（はたの ひろかず） 運用設計ラボ合同会社 JAWS-UG CLI専門支部
Mail operation@office.operation-lab.co.jp

はじめに

JSONは、これまではおもにアプリケーションエンジニアがWebアプリケーションなどで使うものと受け止められていました。しかし昨今、クラウドサービスの利用が進み、その設定情報にアクセスしたり、提供されているWeb APIを利用したりする中で、インフラエンジニアが直接JSONに触れる機会が急激に増えてきています。筆者も、AWS（Amazon Web Services）の各サービスをコマンドラインから操作する中でJSONとの付き合いが深まってきました。

本記事では、AWSのユーザ会の1つであるJAWS-UG CLIが開催するコマンドラインハンズオンでの事例をもとに、コマンドラインからJSONを扱う基本的な方法や便利なツールについて紹介します[注1]。

JSONデータを作成する

まず、コマンドラインからJSONのデータを作成してみましょう。ここではechoコマンドで作成する方法と、ヒアドキュメントで作成する方法の2つを紹介します。

echoコマンドで作成する

次のようなシンプルなJSONデータであれば、

```
{
  "key":"value"
}
```

次のようにechoコマンドで十分です。

```
$ echo '{"key":"value"}' > example-1.json
```

作成したexample-1.jsonは次のようなデータとなります。

```
{"key":"value"}
```

JSONデータを利用するコマンドラインツールはおおむね1行形式のJSONデータを正常に処理してくれるようです。必要がある場合は後述のjqコマンドで整形したうえで使用してください。

ヒアドキュメントで作成する

作成するJSONデータがある程度複雑な場合や、すでにフォーマットが決まっている場合はヒアドキュメントを利用しましょう。ヒアドキュメントとは、スクリプト本文に、改行や半角空白などを書いたとおりに埋め込むための方法で、シェルスクリプトでは、次のような書式になります。

```
コマンド << 終了文字列
ここに、改行や空白を含むテキストを記述する
終了文字列
```

注1）シェルスクリプトに組み込むことを考慮して、Bシェル系（ashやbash）環境を前提としています。ご了承ください。

▼リスト1　サンプルデータ(example-2.json)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::example-bucket/*"
    }
  ]
}
```

終了文字列は必ず行頭に記述し、余計な空白などを入れてはならないことに注意してください。ヒアドキュメントを使うと便利なケースとして、メールや設定ファイルなどの定型フォーマットにおいて、変動する部分に変数を埋め込んで使用する場合などが挙げられます。

ここでは、**図1**のようにヒアドキュメントを利用し、AWSのAPIで利用するポリシードキュメント(**リスト1**)をサンプルとして作成してみます。

▼図1　ヒアドキュメントでJSONを作成

```
$ S3_BUCKET_NAME='example-bucket'
$ FILE_POLICY_DOC='example-2.json'
$ cat << EOF > ${FILE_POLICY_DOC}
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::${S3_BUCKET_NAME}/*"
    }
  ]
}
EOF
```

JSONのデータのチェック

先ほど作成したJSONデータは、何のチェックもしていないので、このままでは正しいフォーマットになっているかどうか不明です。コマンドライン用の検証ツール(バリデータ)を利用して、必ずチェックするようにしましょう。

ここでは、npm(Node.js用パッケージマネージャ)のパッケージで提供されている**jsonlint**コマンドを使用します。誌面の都合でnpmの導入は割愛しますが、npm導入後は次のコマンド[注2]で簡単に導入できます。

```
$ sudo npm install -g jsonlint
```

jsonlintコマンドで検証する

先ほど作成したJSONデータを**jsonlint**コマンドで検証してみましょう。ここでは、パースされたJSON自体の表示を抑制するために次のように**-q**オプションを利用します。

```
$ jsonlint -q example-1.json
```

何も返ってこなければバリデートは成功しています。試しに、

```
{"key":value}
```

のような壊れたJSONデータを作成して、同じく**jsonlint**でチェックしてみましょう(この例では、値をダブルクォートで囲み忘れています)。すると、**図2**のようにエラーの原因を特定して表示してくれます。

▼図2　jsonlintコマンドのエラー出力

```
$ jsonlint -q broken.json
[Error: Parse error on line 1:
{"key":value}
-------^
Expecting 'STRING', 'NUMBER', 'NULL',
'TRUE', 'FALSE', '{', '[', got 'undefined']
```

注2) **-g**オプションで、システム全体共有のディレクトリにインストールします。

JSONデータを活用する

JSONデータの作成とチェックが完了したので、次はその活用方法を考えていきましょう。

2015年3月現在、コマンドラインでJSONデータを扱うツールとして最も著名なのはjqでしょう。公式サイト[注3]に、「jqはJSONデータのためのsedのようなものだ(jq is like sed for JSON data)」と書かれているように、jqコマンドは標準入力からJSONデータを読み込み、フィルタとして各種処理をします。

ここでは、jqコマンドの簡単な利用方法、活用方法について解説していきます。

jqコマンドのインストール

jqパッケージは、MacPorts(OS X)、HomeBrew(OS X)、yum(Linux)、ports(FreeBSD)のいずれでも提供されており、簡単に導入[注4]ができます。

jqでフォーマットを整形する

一番シンプルなjqコマンドの使い方として、先ほど作成した1行のJSONデータ(example-1.json)を整形して表示してみましょう。

```
$ cat example-1.json | jq .
{
  "key": "value"
}
```

jqコマンドでは、JSONデータのルートを「.」で表現し、「.」のみ指定された場合はJSONデータ全体が処理の対象となります。

jqに慣れるために、ちょっと読みづらいJSONデータがある場合はjqで整形してから使用する、というあたりから使い始めても良いのではないでしょうか。

jqでデータを抽出する

整形に慣れたら、次はjqコマンドで特定のキーの値を抽出してみましょう。

ここでは、example-1.jsonファイルのkeyの値をシェル変数に取り込んでみます。jqで通常どおりに値を取得するとダブルクォート付きの値になるため、シェル変数に取り込む場合は**図3**のように-rオプション(raw output)を指定します。これで、JSONデータから特定ノードの値を取り出すことができるようになりました。整形と抽出ができるようになっただけでも、コマンドラインからのJSONデータの活用の幅が大幅に広がるでしょう。

jqで条件に合致するデータを抽出する

本記事の最後として、jqコマンドを利用して条件に合致したレコードを取り出す例を紹介します。たとえば、AWSのコマンドラインツールでグルー

▼図3　jqコマンドで特定ノードの値を取り出す

```
$ VAR_TEST=`cat example-1.json ↵
 | jq -r .key`; echo ${VAR_TEST}
value
```

▼リスト2　サンプルデータ(example-3.json抜粋)

```
{
  "Group": {
    "GroupName": "admin",
    "GroupId": "AGPAXXXXXXXXXXXXXXXX"
  },
  "Users": [
    {
      "UserName": "taro",
      "UserId": "AIDAXXXXXXXXXXXXXXXX"
    },
    {
      "UserName": "jiro",
      "UserId": "AIDAXXXXXXXXXXXXXXXX"
    }
  ]
}
```

注3) URL http://stedolan.github.io/jq/
注4) Ubuntuでのインストールは Case5 で説明しています。

プとユーザのJSONデータを取得してみます。

```
$ IAM_GROUP_NAME='admin'
$ aws iam get-group --group-name ${IAM_GROUP_NAME} > example-3.json
```

そして、**リスト2**のようなJSONデータを取得したとします。このうち、taroに関する情報を取得したい場合は、**図4**のようにjqコマンドを利用することで簡単に取り出すことができます。

この例では、まずシェル変数IAM_USER_NAMEを--argオプションでjqの内部変数user_nameとして取り込んでいます。次に、rootノード(.)直下のUsersノード(複数の子ノードが配列で格納されているため、'Users[]'という形式で指定しています)から、'UserName'がuser_name(ここではtaro)と同じものをselect関数で抽出しています。

このように、jqで抽出を複数回繰り返すことによりかなり複雑な条件でもデータの抽出が可能になります。また、jqは今回利用したselect関数のような有用な機能が多数実装されているので、ぜひマニュアルを読んで使いこなしてみてください。

◆ ◆ ◆

以上、駆け足で紹介してきましたが、シェルの基本機能(echoコマンドやヒアドキュメント)を使うことでJSONデータを作成できること、jsonlintコマンドとjqコマンドを導入するだけでJSONデータの検証と利活用が簡単にできるようになることがご理解いただけたと思います。日常生活でJSONデータを駆使する、そんな新しい時代のコマンドライン活用方法を考えてみてはいかがでしょうか。SD

▼図4　taroに関する情報を取得

```
$ IAM_USER_NAME='taro'
$ cat example-3.json
 | jq --arg user_name ${IAM_USER_NAME}'.Users[]
 | select( .UserName == $user_name )'
{
  "UserName": "taro",
  "UserId": "AIDAXXXXXXXXXXXXXXXX"
}
```

Lesson 3 手を動かしてデータを操ろう！

シェルでのテキスト処理の幅を広げる
これだけは知っておきたいAWKの基礎

awkは、uniq, sort, grep, sedなどと同様に、テキストストリーム処理ツールとしての側面と、複雑な処理をこなすプログラミング言語としての2面がある強力なツールです。本章では、シンプルな例を中心にawkの基本を紹介します。

Author 中島 雅弘(なかじま まさひろ) (株)アーヴァイン・システムズ
國信 真吾(くにのぶ しんご) (株)アーヴァイン・システムズ
富永 浩之(とみなが ひろゆき) 香川大学
花川 直己(はなかわ なおき) 香川大学

いまさらAWK!?

クラウド環境や複数の技術を組み合わせて実現するWeb Baseシステムが普及し、システムの複雑化が進む今日、CLI(コマンドラインインターフェース)やスクリプト言語など、シンプルなツールを組み合わせて使うUnixの基本哲学に立ち返り、より迅速、安全に開発・運用・保守環境を維持できる技術者に注目が集まっています。こうした環境、ツールの代表であるAWKには、「プログラミングから実行、テストまでが俊敏に実施できる軽量性」、「システムの深淵まできめ細かく操作できること」に価値があります。

AWKの強力で機敏なテキストストリーム処理能力は、運用保守を担当するシステム管理者にとって強力な武器になることは間違いありません。そして、プログラミングを担当するソフトウェアエンジニアにとっては、サーバに対する作業、プログラムソースやデータの加工にテキスト処理ツールであるAWKを活用することで、作業の自動化や効率化ができる場面がたくさんあることでしょう。みなさんも、アジャイルで強力なAWKを見なおしてみませんか。

なお、AWKの名称が3名の開発者の頭文字から来ていると考えるとすべて大文字表記がふさわしそうですが、ここからはもろもろの事情によりすべて小文字でawkと表記することにします。

本稿で使用しているawkは、gawk 3.1.xと、gawk 4.x.xを対象としています。gawk 4になって追加された機能を使った一部の例は、gawk 3系では動作しませんので注意してください。また、掲載しているプログラムは、Linux(CentOS 5.5 64bit)、Mac OS X Yosemite 10.10.2での動作を確認しています。動作確認されたOS以外でも、一般的なUnix/Linux系の環境であれば、本稿の例題は動作すると思います。しかし、文字コードやテキストデータ形式(DOS形式、Unix形式、Mac形式)などの違いによって、期待の動作と異なる場合には、扱うプログラム・スクリプトや対象のデータのテキストタイプ、文字コードセットを確認してください。

awkの言語仕様

awkの特徴：他の言語とどこが違う?

C言語やJavaにはないawkの特徴として、awkは入力列(ストリーム)に対する処理に特化した構造が挙げられます。

C言語などでは、プログラム処理が中心で、プログラムからファイルの読み書きを行いますが、awkは入力列があることを前提として、その入力パターンに対応した処理(アクション)を実施します。

awkの使い方

gawkは次のような2とおりの実行方法があ

ります。

①gawk 'パターン-アクション'［入力ファイル名の並び］

②gawk -f プログラムファイル名［入力ファイル名の並び］

①は指定したパターン-アクションを入力ファイルに対して実行します[注1]。たとえば、

```
gawk '/インド/ {print "インドがあった"}' file1
```

は、file1の中に「インド」という文字列を含む行が見つかるたびに「インドがあった」と表示します。なお、パターン-アクションは、複数並べることもできます。

②の動作は基本的には①と変わりませんが、①のパターン-アクションが長い場合や繰り返し使いたい場合などに、ファイルにその内容をプログラムとして記述し実行します。

また、次のように複数の入力ファイルを指定することもできます。

```
gawk -f myprog file1 file2 -
```

これは、myprogというプログラムファイルをfile1、file2、標準入力の3つにそれぞれ作用させます。「-」だけのファイル名は標準入力を入力として指定したことになり、ファイル名を1つも記述しない場合にも標準入力からのデータが入力列として扱われます[注2]。

言語の構造

awkの処理は、与えられた入力列(一般にはファイル)1行ずつ[注3]に対して、プログラムによって指示されたパターンを含むかどうかを調査し、対象となる行がパターンを含む場合にはそれに伴うアクションを実行します。そして、このパターンとアクションの組の集まりが、awkのプログラムになります。awkのプログラミングを理解するには、これらパターンとアクションの記述をそれぞれ習得することが必要です。

図1はawkのプログラムの構造を表したものです。それぞれのパターンとアクションの組は、パターンまたはアクションのどちらかを省略することもできます。

それでは、パターンとアクションを用いた簡単なawkプログラムを書いてみましょう。

例：/abc/ { print }

次のようなファイル「diary.txt」があるとします。

> 今日は日曜日なので、近所のパチンコ屋
> パチンコABCホールで遊んだ。その後同じ
> ビルの中にある喫茶店abcで友達と暇を
> つぶした。

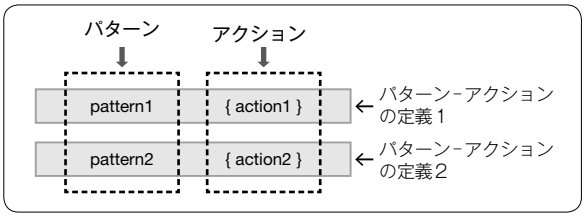
このdiary.txtに対して例として書いたスクリプトを適用すると、次のような実行結果になります。

```
$ gawk '/abc/ { print }' diary.txt ↵
ビルの中にある喫茶店abcで友達と暇を
```

このプログラムは、/abc/がパターン部分で、入力列中にabcという文字列を発見すると、アクションとして記述されている部分printによって、文字列abcを含む行を出力しています。

このパターンとアクションの組による単純な構造がawkプログラムの基本となります。そし

▼図1　パターンとアクション



注1）①のような短いプログラムを直接端末上に入力して実行する方式をワンライナーと呼びます。

注2）gawkには、このほかにもさまざまなオプションを指定した起動方法があります。詳しい起動方法についてはgawkのマニュアル($ man gawk)を参照してください。

注3）ここでは入力列を1行ずつ読み込むという表現をとっていますが、awkでは入力レコードの区切りを変更することが可能ですので、必要なら行以外の入力単位で処理をさせることもできます。

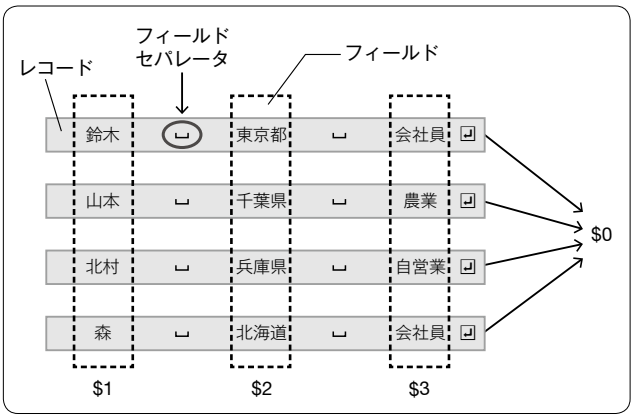
てawkも一般的なプログラム言語と同様に、繰り返し用いる処理をまとめてユーザ関数として定義したり、いくつかの文字列操作関数の機能も利用できます。これらの関数の呼び出しについては、後ほど説明することにしましょう。

入力列の構造：列と行を巧みに処理しよう

awkを使いこなすにはプログラムの構造ばかりでなく、データとして与えられる入力列の構造についても理解しておく必要があります。入力列については、プログラムで利用する側が勝手に記述できるケースはそれほど多くはないからです。

図2は、awkにおける入力列の取り扱い方を表したものです。awkでは、入力列をレコードと呼ばれる単位で分割し、さらにフィールドと呼ばれる単位に分割します。レコードごとにパターンのマッチングが行われ、アクションが実行されます。前出の例では、行全体に対してマッチングを行っていましたが、レコードに含まれるフィールドに対してマッチングを行うこともできます。それでは、レコード、フィールドについて詳しく説明していきましょう。

▼図2　awkにおける入力列の取り扱い



レコード

awkの入力列の処理単位をレコードと呼びます。入力列を区切り文字(レコードセパレータ)ごとにレコードに分割し、パターンの調査とアクションの実行を行います。一般的なawkの使用状態では、区切り文字が改行コードに設定されているため、1行が1レコードに対応します。しかし、レコードに複数の行を含めたいときなどには、その区切り文字を変更することもできます。

awkのプログラム中では、レコードは特殊な変数$0に格納されます。

フィールド

各レコードは、さらに特定の区切り文字(フィールドセパレータ)によってフィールド(欄、または列とも言います)に分けられます。フィールドの区切りは何も指定しなければ1文字のスペースまたはタブになりますが、必要に応じて変更することも可能です。

awkのプログラム中では、分割されたフィールドは、分割された順番にそれぞれ特殊な変数$1、$2、$3……に格納されます。

◆ ◆ ◆

このようにawkは、入力データをフィールドという論理的なかたまりを作って解釈していますので、利用者はデータを表として扱うなど、フィールドを有効に活用することが可能です。また、一般の文章データなどのように、フィールドを意識せずにデータを扱うことも容易です。

パターン：入力列の内容で処理を変える

awkプログラムの中のパターン部分は、それに伴うアクションを選択する式です。省略した場合には、すべてのレコードに対してアクションが処理されることになります。

例：`{ print $1, $3 }`

この例を次のようなデータを持つ「data.txt」に適用してみましょう。

```
Suzuki 90 85
Tanaka 80 95
Yamada 85 85
Tanaka 70 80
Sasaki 95 90
```

```
$ gawk '{ print $1, $3 }' data.txt ↵
Suzuki 85
Tanaka 95
Yamada 85
Tanaka 80
Sasaki 90
```

ここでは、すべてのレコードの第1フィールド($1)と第3フィールド($3)の内容を出力しています。

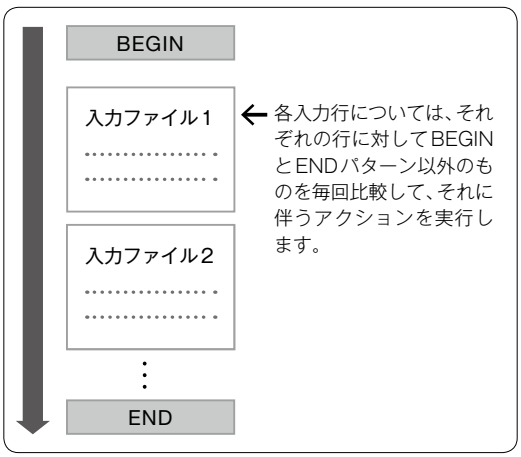
これは、いずれもすべてのレコードに対して照合が行われます。しかし、初期化処理や実行結果の出力など、一度だけ実行したい処理を記述したい場合には、次に説明する特殊なパターンを使用することで実現できます。

BEGINとEND

パターンBEGINは、awkスクリプトにおいてプログラムの初期化の働きをします。BEGINに伴うアクションは、入力列が読み込まれる前に処理されるので、変数を初期化したり、出力レポートのヘッダを印字したり、入力列の区切り文字などを設定することに用います。

パターンENDに伴うアクション部は、入力列がすべて読み終わった後で処理されます。そのため、集計結果の出力やレポートのフッタ出力、その他、後処理のために使用することになります(**図3**)。

▼図3　BEGINとENDの図



入力列として複数のファイルを指定した場合でも、BEGIN、ENDパターンはそれぞれ1度ずつしか実行されないことに注意してください。

例：`BEGIN { print "Name  ", "Math" } { print $1, $3 }`

```
$ gawk 'BEGIN { print "Name  ", "Math" } { ↵
print $1, $3 }' data.txt ↵
Name   Math
Suzuki 85
Tanaka 95
Yamada 85
Tanaka 80
Sasaki 90
```

BEGINFILEとENDFILE

gawk 4.0から利用可能になったパターンです。BEGINFILEは、ファイルからレコードを読み込む前に1度だけ実行されます。ENDFILEは、最後のレコードを読み込んでパターン-アクションを実行した後に1度だけ実行されます。

先に挙げたBEGINとENDパターンは、複数のファイルを指定した場合でも1度しか実行されませんでしたが、BEGINFILE、ENDFILEパターンは、それぞれのファイルごとに実行されます(**図4**)。

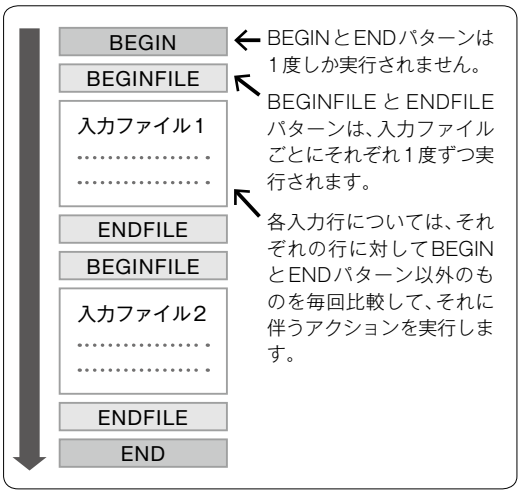
BEGINFILEパターン中では、特殊な変数FILENAMEに現在処理を行おうとしているファイル名が格納されます。BEGINFILEパターンの用途しては、

①指定ファイルが存在するかをチェックする
②指定ファイルの名前(とくに拡張子)によって処理を変更する

といったものが挙げられます。

BEGINやENDパターンを用いることでプログラムらしくなってきましたね。さらに、単純に文字列が含まれるかどうかの照合だけでなく、

▼図4　BEGINFILEとENDFILEの図



複雑な条件での照合を行うこともできます。

比較演算式と論理演算式

比較演算は、大小同値関係やマッチングについての演算です。そして、これらの演算をANDやORなどの論理演算と組み合わせて指定することができます。例として、1番目のフィールドが「Tanaka」で、2番目のフィールドが80未満のものをマッチさせる書き方は次のようになります。

```
$1 == "Tanaka" && $2 < 80
```

これをdata.txtに適用してみましょう。

```
$ gawk '$1 == "Tanaka" && $2 < 80 { print
$1, $2 }' data.txt
Tanaka 70
```

正規表現

強力な表現式である正規表現をパターンとして指定できます。たとえば、レコードが「S」からはじまるものをマッチさせるには次のように書きます。

```
/^S/
```

これもdata.txtに適用してみましょう。

```
$ gawk '/^S/ { print }' data.txt
Suzuki 90 85
Sasaki 95 90
```

パターンの範囲

```
pattern1, pattern2
```

というように、「,」で区切って2つのパターンを記述することで、pattern1が出現してからpattern2が出現するまでの範囲を選択の対象とすることができます。たとえば、「Tanaka」を含むレコードから「Sasaki」を含むレコードまでを選択するには、

```
/Tanaka/, /Sasaki/
```

と書きます。これをdata.txtに適用すると、

```
$ gawk '/Tanaka/, /Sasaki/ { print }' data.txt
Tanaka 80 95
Yamada 85 85
Tanaka 70 80
Sasaki 95 90
```

となります。なお、pattern1の後にpattern2が出現しない場合は、最後のレコードまでマッチします。

複合パターン

パターンを複数組み合わせることで、さらに複雑な条件の指定が可能になります。使用できる複合パターンは**表1**に示すものを用いることができ、さらに複合式を複数書き並べることも可能です。

ここまで、パターンについて紹介したものをまとめると**表2**のようになります。

アクション

アクションは、一般の言語でのプログラムそのものといえる部分です。パターンで選択されたアクションは、その部分に記述されている出力や計算などの処理を行います。awkで使用できる文を**表3**にまとめています。アクションが省略された場合には、パターンに合致したすべて

▼表1　複合パターン

パターン	説明
式1 論理演算子 式2	式1と式2を論理式(&& 論理積、|| 論理和)を使った演算によって評価する
!式	式が成立しないときに続くアクションが実行される
パターン1？パターン2：パターン3	パターン1を評価して真になる場合には、パターン2を評価した結果を採用し、パターン1が偽であるときには、パターン3を評価した結果によって続くアクションの実行を決定する

▼表2　awkで使用できるパターン

パターン	説明
空のパターン	すべての入力行に対してアクションを実行する
/正規表現式/	正規表現に合致する場合、アクションを実行する
式	式が真である場合、アクションを実行する
BEGINとEND	プログラム実行直後と終了直前にアクションを実行する
BEGINFILEとENDFILE	ファイルの読み込み前、ファイルの末尾まで読み込んだ後にアクションを実行する
パターン1, パターン2	パターン1が始まってパターン2が出現するまでアクションを実行する
複合パターン ・式1 論理演算子 式2 ・!式 ・パターン1？パターン2：パターン3	※表1を参照

のレコードを標準出力へ出力することになります。

フィールドの値を整形して出力する例は次のようになります。

```
$ gawk '{ printf("%8s : %3d,%3d n", $1, $2, $3); }' data.txt
  Suzuki :  90, 85
  Tanaka :  80, 95
  Yamada :  85, 85
  Tanaka :  70, 80
  Sasaki :  95, 90
```

また、特定の条件(第2フィールドが80より大きい)に合致するときのみ整形して出力する例は次のとおりです。

```
$ gawk '{ if ( $2 > 80 ) printf("%8s : %3d, %3d n", $1, $2, $3) }' data.txt
  Suzuki :  90, 85
  Yamada :  85, 85
  Sasaki :  95, 90
```

定数

awkには2つの種類の定数があります。1つは数としての定数であり、もう1つは文字列定数です。数値は、整数、小数点を含む表示、そして指数表示による値のどれでも可能です。

文字列は、連(つら)なった文字をダブルクォーテーション「"」で括ったものです。

①"This is a string"
②"日本語の文字列"

▼表3　アクションの中で使用できる文

アクション要素	書式
式	定数、変数、代入、関数呼びだしなど
出力	print 式の並び
	printf(書式, 式の並び)
制御	if (式) 文
	if (式) 文 else 文
	while (式) 文
	do 文 while (式)
	for (式; 式; 式) 文
	for (変数 in 配列) 文
	break
	continue
	next
	exit
	exit 式
	{ 文の並び }

▼表4　記号「\」でエスケープされた特殊文字

\a	alert文字。通常は[^G]([Ctrl]+[G])
\b	バックスペース[^H]([Ctrl]+[H])
\f	フォームフィード(改ページ)[^L]([Ctrl]+[L])
\n	改行
\r	復帰
\t	タブ[^I]([Ctrl]+[I])
\v	垂直タブ[^K]([Ctrl]+[K])
\num	8進コードで表現した文字(numは1桁から3桁の8進数)
\xnum	16進コードで表現した文字(numは1桁か2桁の16進数)
\c	文字cの文字列中で特別な意味を持つ意味を打ち消して、cそのものとして扱う(例：文字列中に"」を書き記すときは「\"」。「\」そのものは「\\」)

文字列や正規表現の中で、記号「\(バックスラッシュ)」によってエスケープされた特殊な意味を持つ文字があります(**表4**)。これらの文字には、printfなどで改行を表す文字としてよく使われる\nなども含まれています。また、空の文字列""はNULLと呼ぶことがあります。

変数、型

awkの変数は使用する前に宣言をする必要はありません。また、扱うデータにも変数にも型の指定をしません。実行時にawkが文脈に応じて適切な型として評価してくれるのです。

例：

```
{ line += 1 }
END { print "total line is", line }
```

上の例は、データを1行読み込むたびにlineという変数を1ずつ足しこみ、すべてのデータを読み込んだ後にその合計を表示するものです。lineという変数は、宣言せずに使用できているところに注目してください。

lineは、整数型もしくはそれと同様な振る舞いをする数値型の変数として扱われることを意図しています。また、数値型の変数の初期値がC言語の局所変数のように不定値ではなく、常に0に初期化されていることも重要です[注4]。

▼表5　特別な変数

変数	説明
$1、$2、……、$n、そして$0	$に続く数字は、現在の入力レコードにおいて何番目のフィールドであるかを示す。$0は現在の入力レコード全体を意味する
ARGC	コマンドラインの引数の数
ARGV	コマンドラインの引数の配列
FILENAME	現行の入力ファイル名が格納される
FNR	現在のファイルのレコード番号が格納される
FS	入力フィールドの区切り文字を定義しておく変数。区切り文字の指定には正規表現も可能
NF	現レコードのフィールド数が格納される
NR	読み込んだレコード数が格納される。複数ファイルを読み込んだ場合はそれらの合計が入る
OFMT	数値のprint文においての出力書式を定義する変数。定義しない場合、「%.6g」が既定値
OFS	出力フィールドの区切り文字を定義する変数。定義しない場合、スペースが既定値
ORS	出力レコードの区切り文字を定義する変数。定義しない場合、復帰改行が既定値
RS	入力レコードの区切り文字を定義する変数。定義しない場合、復帰改行が既定値
RSTART	組み込み関数match()で対応した最初の文字のインデックスが格納される
RLENGTH	組み込み関数match()で対応した文字列の長さが格納される
SUBSEP	添字区切り文字を定義する変数。定義しない場合、「\034」が既定値
IGNORECASE	英字の大文字小文字によって影響される正規表現の処理に作用する変数
ENVIRON	環境文字列を保持している配列
FPAT（gawk 4.0以降）	フィールドとして分割したい文字列を正規表現によって指定するときに使用する変数

配列、多次元配列

awkにもC言語などで使用できる、同じタイプの複数のデータをまとめて扱うために「配列」が用意されています。配列の添字には、C言語と同様にarr[1]、arr[2]というように数字に加えて、arr["apple"]やarr["1 2"]のように、文字列を用いることができます。

ただし、添字は内部的にはすべて文字列として扱われるため、arr[1]とarr["1"]は、同じデータを参照します。

```
$ gawk 'BEGIN { a["1"] = 10; a[1] = 20; print ↵
a["1"], a[1] } ↵
20 20
```

配列の配列といった多次元配列については、gawk 4.0以前では、次のような記述で多次元配列を表現していました。

```
arr[x,y]
```

gawk 4.0以降では、次のような記述で多次元配列を表現することができるようになりました。

```
arr[x][y]
```

なお、前者はあくまでもarr["x,y"]という一次元配列であり、後者のものとは別物であることに注意してください。

特別な変数

awkには、制御やデータの解析のためにいくつかの組み込み特別変数があります。ここでは、これらの特別な変数を**表5**で紹介することにしましょう。

文字列操作関数

次に、とても便利で頻繁に使う関数を紹介します。

注4） なお、文字列型とみるときの初期値は空列""になります。

● index(STRING, TARGET)

文字列STRING中に、文字列TARGETが存在する位置を返します。STRING中にTARGETが存在しなければ、0が戻り値となります。

```
例1：pos = index("123456789", "1")
→ posの値は1
例2：pos = index("123456789", "234")
→ posの値は2
例3：pos = index("123456789", "7890")
→ posの値は0
```

● length(STRING)

STRINGの長さを返します。全角文字、半角文字いずれも1文字とカウントされます。

```
例1：len = length("1234567890")
→ lenの値は10
例2：len = length("あいうえお")
→ lenの値は5
```

● match(STRING, REGEXP)

文字列STRING中に、正規表現REGEXPが現れた位置を返します。見つからなければ0が返ります。ここでの検索は、最左最長なものが選ばれ、組み込み特別変数RSTARTにその位置が、RLENGTHにその長さが設定されます。見つからなかった場合には、RSTART = 0、RLENGTH = −1となります。

```
例：match("ABCD1234EFGH56789", /[0-9]+/)
→ RSTARTの値は5
→ RLENGTHの値は4
```

● split(STRING, ARRAY, FIELD_SEPARATOR)

STRINGを区切り文字FIELD_SEPARATORに従って配列ARRAYに分割します。もし、FIELD_SEPARATORが指定されなければ、組み込み特別変数FSを代わりに指定したものとみなされます。そして、関数の戻り値は、分割された数になります。FIELD_SEPARATORには正規表現を使うことができます。

```
例：ハイフン「-」、水平タブ「\t」、コロン「:」を区切り
    文字として分割する
split("2015-04-01\t12:34:56", time, ↵
"[-\t:]+")
→ time[1] = 2015
→ time[2] = 04
→ time[3] = 01
→ time[4] = 12
→ time[5] = 34
→ time[6] = 56
```

● patsplit(STRING, ARRAY, REGULAR_EXPRESSION)

gawk 4.0から利用できる関数です。前出のsplit関数では、区切り文字を指定しましたが、patsplit関数では、STRINGからREGULAR_EXPRESSION(正規表現)に指定したパターンにマッチする文字列を切り出します。

```
例：ハイフン「-」、水平タブ「\t」、コロン「:」以外の連
    続する文字列で分割する。splitの例との違いは、
    正規表現にある「^」による否定
patsplit("2015-04-01\t12:34:56", time, ↵
"[^-\t:]+")
→ time[1] = 2015
→ time[2] = 04
→ time[3] = 01
→ time[4] = 12
→ time[5] = 34
→ time[6] = 56
```

● sprintf(FORMAT, EXPRESSION1,...)

書式指定文字列FORMATに従って書式付けられたEXPRESSION1, ...を文字列として返します。printf()関数が出力をせずに、文字列を返すものと考えることができます。C言語のsprintf()とは引数が異なっていることに注意してください。

```
例：str = sprintf("%sさんのテストの成績は↵
%3d点です。\n", "鈴木", 75)
```

この例を、print strで出力すると次のようになります。

```
鈴木さんのテストの成績は 75点です。↵
↵
```

● sub(REGEXP, REPLACEMENT_STRING, TARGET_VARIABLE)

文字列変数TARGET_VARIABLE中の最初の正規表現REGEXPをREPLACEMENT_STRINGに置換し、置換した数を返します。もし、文字列変数TARGET_VARIABLEが指定されなければ、現行レコードが対象となります。もちろん、文字列のマッチングは最左最長なものが対象になります。置き換える文字列の指定には、特殊文字「&」を含めることが許されていて、この文字はマッチした文字列を意味しています。sub()関数ははじめに発見された文字列を、ただ1度だけ置き換えますので、戻り値は1か0のどちらかになります。

たとえば、str = "ABC123ABC"だった場合、

```
res = sub(/ABC/, "abc", str)
```

とすると、str、resの値はそれぞれ次のようになります。

```
str = "abc123ABC"
res = 1
```

また、

```
sub(/ABC/, "&DEF", str)
```

とすると、strの値は"ABCDEF123ABC"となります。このようにsub関数を使うと、マッチした文字列の前後に文字を追加することができます。

● gsub(REGEXP, REPLACEMENT_STRING, TARGET_VARIABLE)

文字列変数TARGET_VARIABLE中のすべての正規表現REGEXPをREPLACEMENT_STRINGに置換し、置換した数を返します。もし、文字列変数TARGET_VARIABLEが指定されなければ、現行レコードが対象となります。文字列のマッチングは最左最長なものが対象です。置き換える文字列の指定には、特殊文字「&」を含めることが許されていて、この文字はマッチした文字列を意味しています。

たとえば、str = "ABC123ABC"だった場合、

```
res = gsub(/ABC/, "abc", str)
```

とすると、str、resの値は次のようになります。

```
str = "abc123abc"
res = 2
```

また、

```
gsub(/ABC/, "&DEF", str)
```

とすると、strの値は、"ABCDEF123ABCDEF"となります。

● substr(STRING, P, LENGTH)

STRINGのP文字目からLENGTH文字分の部分文字列を返します。LENGTHが指定されない場合には、文字列STRINGSの最後までとなります。

```
例：str = substr("あいうえお", 2, 3)
→ strの値は"いうえ"
```

まとめ

awkの基本的なしくみ、構造を理解していただけたでしょうか。インプットストリームの内容をパターンによって分類し、それに対する処理を記述することで、複雑なテキスト処理が可能になります。また、awkの言語構造や関数などは、C言語やJavaなど広く普及している言語との共通点がたくさんあります。紙幅の都合で解説できませんでしたが、awkにはプログラミングに必要な、算術演算子、制御構文、ユーザ定義関数などさまざまなしくみが用意してあります。一般的なプログラム言語の知識のある方でしたら、それらのしくみを使ったawkプログラムであっても、動作を容易に理解できることと思います。

次の章では、具体的なawkプログラム例を紹介します。ここまでの説明と他のプログラム言語の知識をベースに、それらのプログラムを楽しんでみてください。SD

Lesson 4 手を動かしてデータを操ろう！

習うより慣れよう！ サンプルをまねて AWKの実用性を実感

前章で基本を理解したところで、本章ではシンプルなワンライナーから段階的に、複雑な処理をするスクリプトまでを実例を挙げて、テキストストリーム処理ツールとプログラミング言語の2つのawkの側面を紹介します。awkの俊敏性と強力さの両方を実感してみてください。

Author 中島 雅弘(なかじま まさひろ) (株)アーヴァイン・システムズ
國信 真吾(くにのぶ しんご) (株)アーヴァイン・システムズ
富永 浩之(とみなが ひろゆき) 香川大学
花川 直己(はなかわ なおき) 香川大学

初歩的なスクリプト

awkの最大の特徴は、ここに挙げるような1行スクリプトだけでも数多くの処理がこなせることです。1行スクリプトを書く場合には改行による文の区切りが使えないので、文の連結「;」や文のブロック「{}」を省略できません。また、最大限簡潔になるように工夫します。1行スクリプトを理解し使いこなせるようになれば、本章の後半で紹介するような複雑なプログラムも容易に記述できるようになります。

行やフィールドの個数

前章で紹介したパターンとアクション、そして特別な変数などを使って簡単な1行スクリプトから実行してみましょう。

【スクリプト1】

```
{ f += NF }  END { print NR, f }
```

【スクリプト2】

```
{ c += length }  END { print c }
```

【スクリプト3】

```
{ print NF, length }
```

入力データに対して、スクリプト1は、全体の行数(NR)とフィールド数(NFを加算)を出力します。スクリプト2は、全体の文字数(lengthを加算。空白文字を含む)を出力します。スクリプト3は、各行ごとにフィールド数(NF)、文字数(length。空白文字を含む)を出力します。

上記スクリプトの実行例を示すために、読み込ませるデータ「jpn.txt」を次のようなテキストとして用意します。

```
はやくち ことば

かえる ぴょこ ぴょこ 3 ぴょこ ぴょこ
あわせて ぴょこ ぴょこ 6 ぴょこ ぴょこ
```

スクリプト1〜3は次のように実行されます。

スクリプト1の実行例

```
$ gawk '{f += NF} END {print NR, f}' jpn.txt ↵
4 14 ←4行 14フィールド
```

スクリプト2の実行例

```
$ gawk '{c += length} END {print c}' jpn.txt ↵
51 ←51文字
```

スクリプト3の実行例

```
$ gawk '{print NF, length}' jpn.txt ↵
2 8 ←1行目：2フィールド 8文字
0 0 ←2行目：0フィールド 0文字
6 21 ←3行目：6フィールド 21文字
6 22 ←4行目：6フィールド 22文字
```

ある文字列を含む行やフィールドの個数

次に、ある文字列を含む行やフィールドがいくつあるかを調べる書き方です。

【スクリプト1】

```
/EXP/  { c++ }  END  { print c }
```

【スクリプト2】

```
{ for ( i = 1; i <= NF; i++ ) if ( ➡
$i ~ /EXP/ ) c++ }  END { print c }
```

【スクリプト3】

```
{ c=0; for ( i = 1; i <= NF; i++ ) if ( ➡
$i ~ /EXP/ ) c++; print c }
```

入力データに対して、スクリプト1は、正規表現EXPにマッチする行数を出力します。スクリプト2は、正規表現EXPにマッチする全フィールド数を出力します。スクリプト3は、各行ごとに正規表現EXPにマッチするフィールド数を出力します。

先ほどのjpn.txtを前提に、「ぴょこ」という文字列についてスクリプト1〜3を適用すると、次のような結果が得られます。

スクリプト1の実行例

```
$ gawk '/ぴょこ/ {c++} END {print c}' jpn.txt ↵
2 ←2行
```

スクリプト2の実行例

```
$ gawk '{for(i=1;i<=NF;i++) if ($i ~ ➡
/ぴょこ/) c++} END {print c}' jpn.txt ↵
8 ←8フィールド
```

スクリプト3の実行例

```
$ gawk '{c=0; for(i=1;i<=NF;i++) if ($i ~ ➡
/ぴょこ/) c++; print c}' jpn.txt ↵
0 ←1行目：0フィールド
0 ←2行目：0フィールド
4 ←3行目：4フィールド
4 ←4行目：4フィールド
```

フィールドの出現回数

1行スクリプトからちょっとした処理を加えて、スクリプトファイル(awkの場合は拡張子に.awkとつけるとわかりやすいでしょう)にしてみます。スクリプトファイル(hoge.awk)を使って入力ファイル(input.txt)に対して実行させるには、前章で解説したように次のように書きます。

```
$ gawk -f hoge.awk input.txt ↵
```

では、簡単なスクリプトファイルを作ってみましょう。フィールドの出現回数をカウントするitem.awkです(**リスト1**)。

item.awkは、各フィールドの値となる文字列の出現回数を求めます。表示は、回数、文字列の順です。jpn.txtに対する実行結果は次のようになります。

```
$ gawk -f item.awk jpn.txt ↵
  8 ぴょこ
  1 かえる
  1 3
  1 あわせて
  1 ことば
  1 はやくち
  1 6
```

item.awkでは、各フィールドを添字とする配列に加算します。フィールドの大きさは一定していないので、先に出現回数を出力して見やすくしています。出現回数は、printf()の出力書式を使って、幅3で右詰めに表示されます。for文は配列要素で回しているので、awkによっては出力順が例と異なる場合があります。

テキストの大きさ

続いての例は、テキストの大きさを調べるwc.awkです(**リスト2**)。

wc.awkは、与えられたファイル中の、行数、単語数、文字数、バイト数(通常の全角文字は2文字と数える)を数えるスクリプトです。単語は、組み込み変数FSで区切られたものが1単語と数えられます。このスクリプトは、複数のファイルにまたがってカウントでき、すべてのファイルの合計も求めています。

入力データとして、jpn.txtに加えて次のようなdata.txtも用意します。

▼リスト1　item.awk

```
# フィールドの出現回数
{
  for ( i = 1; i <= NF; i++ )  list[$i]++
}

END  {
  for ( item in list )  printf("%3d  %s\n", list[item], item)
}
```

▼リスト2　wc.awk

```
# テキストの大きさ
{
  c[FILENAME] += length($0)
  w[FILENAME] += NF
  l[FILENAME]++
}

END {
  for (f in c) {
    printf("%d\t%d\t%d\t%s\n", l[f], w[f], c[f], f)
    tc += c[f]
    tw += w[f]
    tl += l[f]
  }
  printf("%d\t%d\t%d\ttotal\n", tl, tw, tc)
}
```

```
A 34    0.5
BB 7 12.4
C 0
D       777   1.23
```

実行結果は次のようになります。

```
$ gawk -f wc.awk jpn.txt data.txt ↵
4 14  51  jpn.txt
4 11  39  data.txt
8 25  90  total
```

出力された数値は左から順に、行数、単語数、文字数です。文字数は、全角・半角文字いずれも1文字としてカウントされます。

wc.awkでは、各入力ファイルごとに、配列を用意し、ファイル名を配列の添字としています。計算結果の出力と合計には、ENDパターンの処理(入力が終了した後)で行います。表示されるファイルの順序は連想配列の要素の取り出し方の実現方法によるので、不定と考えたほうが無難です。

Webサーバのログの解析

最後の例として、少し大きなプログラムを扱ってみましょう。

近年、企業だけでなく、個人でも気軽にサーバを運用するようになってきました。クラウドを利用したり、さまざまなWebサービスと連携することも当たり前になってきています。

しかし一方で、大量アクセスによるサービス停止や、クラッカーによる攻撃も問題になっています。効率的な運用を行い、セキュリティにも配慮するには、システムの動作を適切に設定し、ログも確認しておく必要があります。本節では、そういった作業へのawkの活用を紹介します。

Webサーバのログ

サーバには、多くのログが残されています。起動ログや操作ログはもちろんのこと、メールログやログイン失敗のログまで残されています。奇妙な操作や連続したログイン要求は、攻撃の前準備の可能性があります。それらがあまりに多い場合は、何らかの対策を行わなければなりません。そこで、ログを集計し、そういった危険を事前に察知できるようにしましょう。

一例としてWebサーバの1つであるApache HTTPD(以下、Apache)を題材にします。Apacheは、さまざまなログを出力します。アクセス

▼表1　ログの各フィールドの意味

フィールド	ログ中の文字列	説明
$1	127.0.0.1	クライアントのIPアドレス
$2	-	クライアントアイデンティティ
$3	-	HTTP認証のユーザID
$4	[22/Oct/2014:20:33:37 +0900]	リクエスト処理の終了日時
$5	"GET /hoge.htm HTTP/1.1"	クライアントのリクエスト内容
$6	404	HTTPのステータスコード
$7	360	クライアントへ送信したオブジェクトのサイズ
$8	"http://localhost/"	クライアントの参照元サイト
$9	"Mozilla/5.0 … Firefox/20.0"	クライアントのブラウザ情報

▼図1　Webサーバのログ(1行のみ抜粋)

```
127.0.0.1 - - [22/Oct/2014:20:33:37 +0900] "GET /hoge.htm HTTP/1.1" 404 360 "http://localhost/" ↵
"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:20.0) Gecko/20100101 Firefox/20.0"
```

▼リスト3　remove_error_access.awk

```
##====  事前処理
BEGIN {
    #----  フィールドパターンの指定
    FPAT="([^ ]+)|(\"[^\"]+\")|(\[[^\]]+[^\]]+\])"
    #----  出力ファイルの指定
    information = "information.tmp"   # 情報
    redirection = "redirection.tmp"   # リダイレクト
    client_error = "client_error.tmp" # クライアントエラー
    server_error = "server_error.tmp" # サーバエラー
    success = "success.tmp"           # 成功

    total = 0                         # アクセスの総計
}

#====  本体処理
{
    count($6, num)
}

##==== パターンマッチによる処理の分岐
##----  情報コードの処理
$6 ~ /1[0-9]+/ {
    print $0 > information
}

##----  リダイレクションコードの処理
$6 ~ /3[0-9]+/ {
    print $0 > redirection
}

##----  クライアントエラーコードの処理
$6 ~ /4[0-9]+/ {
    print $0 > client_error
}

##----  サーバエラーコードの処理
$6 ~ /5[0-9]+/ {
    print $0 > server_error
}

##----  成功コードの処理
$6 ~ /2[0-9]+/ {
    print $0 > success
}

##====  事後処理
END {
    for ( i in num ) {
        OFS=","
        print i "," num[i]
    }
    print "total," total
}

##====  ステータスコード別の集計
function count(code, array) {
    array[code]+=1
    total+=1
}
```

ログやエラーログ、SSLのログなどもあります。ここでは、アクセスログを対象として、ログ解析を行います。

さて、ログ解析を行うためには、まずログの形式を確認しなければなりません。Apacheでは、**図1**のようなデータ形式で保存されます。もちろん、サーバの設定によってはこれに限りません。

表1には、各フィールドの情報の意味をまとめました。この中から、有用なデータを持っていそうなものを集計していきます。

Webサーバのログの分割の概要

まずは、HTTPのステータス別にログを分解しましょう。これをすることによって、成功しているアクセス、失敗しているアクセスを分け、目的に応じた集計を行うことができます。そのためのawkスクリプトがremove_error_access.awkです(**リスト3**)。

なお、FPATはgawk 4系列からの機能です。FPATを使わない場合、各行の文字列が複雑になると、正しくフィールドに分割できないことがあります。

ログの分割のプログラムremove_error_access.awkの解説

事前処理(BEGIN部分)では、フィールドパターンの指定、出力ファイルの指定、計測変数の初期化を行います。Apacheのログでは、""や、[]で囲まれた文字列が1フィールド内に登場します。そこで、フィールドそのもののパターンを指定してやり、""や[]で囲まれているものを1フィールドとして認識させてやります。

本体処理では、$6、すなわち、HTTPのステータスコード別に集計する関数(後述)を呼び出しています。パターン-アクション部分では、コード別に出力を行っ

ています。事前処理で指定したファイルに、行そのものをリダイレクトで出力します。

事後処理として、ステータスコード別の集計結果を表示します。このときCSV形式になるよう、区切り文字を「,」に変更しています。

ステータスコード別に集計する関数count()を定義します。count()は、ステータスコードの文字列codeと、格納用の連想配列arrayを引数としています。count()では、受け取ったステータスコードに対応するarray[code]を1増分しています。また、総計を示すtotalも1増分しています。

このスクリプトをアクセスログデータ（例：/var/log/httpd/access_log）に適用すると、**図2**のように実行されます。内部で、ステータス別にしたファイルへログの内容を出力しています。また標準出力には、CSV形式でステータス別のアクセス数を出力しています。

プログラムを実行する前（**図2-①**）と後（**図2-②**）とで、ファイル数が変わったと思います。*.tmpのファイルが、remove_error_access.awkによってステータス別に分割されたログファイルです。

さて、これでステータス別にログを分割することができました。これらのログを利用して、2とおりの集計を行ってみましょう[注1]。

クライアント別の集計

まずは、クライアント別に集計を行ってみましょう。ここで注意すべきは、クライアントを格納しているフィールドはダブルクオート(")で括られており、内部にスペースが含まれている可能性があるという点です。

クライアント別の集計スクリプトの説明

スクリプトanalyze_client.awkはとても簡単

▼図2　ログの分割プログラムの実行結果

```
$ ls ↵
analyze_client.awk  analyze_hourly.awk  remove_error_access.awk
↑①

$ gawk -f remove_error_access.awk /etc/httpd/access_log ↵
200,147
206,1
301,1
302,1
304,5
400,4
401,8
404,168
405,9
total,344

$ ls ↵
analyze_client.awk  analyze_hourly.awk  client_error.tmp
redirection.tmp  remove_error_access.awk  success.tmp
↑②
```

▼リスト4　analyze_client.awk

```
##==== 事前処理
BEGIN {
    FPAT="([^ ]+)|(\"[^\"]+\")|(\[[^\[]+[^\]]+\])"
}

##==== 本体処理

{
    freq_of[$9]+=1
}

##==== 事後処理
END {
    OFS = ","
    for ( i in freq_of ) {
        if ( ! freq_of[i] ) {
            freq_of[i] = 0
        }
        print i, freq_of[i]
        total+= freq_of[i]
    }
    print "total " total
}
```

注1）余談ですが、クライアントを格納しているフィールドはその内部にさまざまな情報を持っています。ブラウザの種類やバージョン以外にも、カーネルの名前やOSのバージョンなども記載されています。興味のある人は、awkスクリプトを作って、どんなOSからのアクセスが多いかなども分析してみるのも楽しいですよ。

▼図3　クライアント別集計の実行結果

```
$ gawk -f analyze_client.awk success.tmp ↵
"Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0)",9
"GoogleBot/2.1",26
"Mozilla/5.0 (iPod; CPU iPhone OS 5_1_1 like Mac OS X) AppleWebKit/534.46 (KHTML, like Gecko) ➥
Version/5.1 Mobile/9B206 Safari/7534.48.3",1
"Mozilla/5.0 (compatible; DotBot/1.1; http://www.opensiteexplorer.org/dotbot, help@moz.com)",4
"Twitterbot/1.0",1
"Mozilla/5.0 (iPhone; CPU iPhone OS 7_1_1 like Mac OS X) AppleWebKit/537.51.2 (KHTML, like Gecko) ➥
Version/7.0 Mobile/11D201 Safari/9537.53",4
(...略...)
"Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)",31
"Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko",1
"Mozilla/5.0 (Windows NT 5.1; rv:9.0.1) Gecko/20100101 Firefox/9.0.1",1
total 148
```

です(**リスト4**)。事前処理として、フィールドパターンを指定しています。本体処理は、$9、つまりクライアントのブラウザ情報の連想配列の要素を1ずつ増分しています。事後処理として、CSV形式での出力を行います。

このスクリプトを先ほど分割して出力されたsuccess.tmpに適用した結果が**図3**のようになります。

時間別の集計

次に、時間別の集計を行います。時間別の集計も、クライアント別の集計と似ています。ただ、形式がほかのフィールドと少し異なる部分に注意が必要です。

時間別の集計のプログラムの説明

analyze_hourly.awkが処理用スクリプトです(**リスト5**)。事前処理と事後処理は、クライアント別の集計と同じです。

時間を格納しているフィールド$4ですが、形式は、「[dd/MM/yyyy:hh:mm:ss +0000]」という形です。しかし、集計に必要な時間を示す部分は「hh」部分です。すなわち、$4をパースしてhhの部分のみを抽出しなければなりません。さらに、括りを示す「[]」が必要ないですよね。

そこで本体処理では、まず「[]」をsubstr()関数を利用して削除しています。必要な文字は、フィールド先頭の「[」と、フィールド末尾の「]」を除いた部分ですので、2文字目から文字列の末尾2文字目までを取り出し、$4に再代入しています。

次に、時間の部分をsplit()関数を用いてパースし、time[]に格納します。time[]には、次のようにデータが格納されています。

▼リスト5　analyze_hourly.awk

```
##==== 事前処理
BEGIN {
    FPAT="([^ ]+)|(\"[^\"]+\")|(\[[^\[]+[^\]]+\])"
    total = 0
}

##====  本体処理
{
    $4 = substr($4, 2, length($4)-2)     # 大括弧始まりを除去
    split($4, time, ":") # $4を分割 1:日付, 2:時, 3:分, 4:秒
    hour = time[2]
    freq_of[hour]++
}

##==== 事後処理
END {
    for ( i in freq_of ) {
        if ( ! freq_of[i] ) {
            freq_of[i] = 0
        }
        print i "," freq_of[i]
        total += freq_of[i]
    }
    print "total," total
}
```

配列	格納データ形式
time[1]	dd/MM/yyyy
time[2]	hh
time[3]	mm
time[4]	ss +0000

この配列から必要な部分、今回はtime[2]を利用し、連想配列の参照に使っています。

また、出力ですが、連想配列の出力は昇順降順に統一することができません。そこで、Linuxのコマンドであるsortを使ってやることで、時間別に昇順で出力させています。なお、gawk 4系列では、END {の次の行でPROCINFO["sorted_in"] = "@ind_str_asc"と記述しておくと、パイプライン処理を使わず、スクリプトのみで整列させることができます。

このスクリプトをsuccess.tmpに適用した結果が**図4**です。

アクセスログという大量のデータを、これだけのスクリプトで処理をすることができました。また、ログのようにawkが得意とするテキストストリームであれば、他の言語と比較しても効率的な処理が期待できます。ログを眺める目的はさまざまだと思いますが、ぜひawkを使って、楽しいログ観察ライフを過ごしてください。

結びにかえて 書籍のご紹介

以上、awkの使い方を2章にわたって解説してきましたが、いかがでしたでしょうか。本稿は、書籍『AWK実践入門』(5月下旬発売予定)の内容から抜粋、編集したものになっています。

書籍では、CLIに馴染みのない技術者や、しばらく距離を置いていた技術者にとっても、躊躇(ちゅうちょ)なくCLIとスクリプト言語環境が使えるきっかけとなるような章が設けてあります。本特集で紙幅の都合上紹介できなかった正規表現は、きちんと学べるように単独の章に詳説し、強力な連想配列についても、ふんだんに例題を取り入れ解説しました。

また、当該書籍は、「awkをはじめて使う人から、プロのプログラマまで使っていただける」ことを目指し、次の目的をもって執筆しています。

- awkと正規表現のリファレンスとしての活用
- awkプログラミングをサポートするスクリプトライブラリ集
- awkを使った問題解決の事例集

そして、単なるスクリプトの域を越えてプログラミング言語としての可能性をお伝えするために、「ローリングハッシュによる文字列の部分一致」、「編集距離による文字列の類似度」、「マルコフ情報源によるランダム文字列の生成」、「JSONを使ったSNSデータ分析」など、アカデミックな内容から昨今の技術者が関心を寄せている事例まで取り上げました。

本稿を読んでawkに興味を持たれた方、スクリプト言語やアルゴリズムに興味のある方、システム運用保守の現場で小粋で機敏なツールが必要な方など、幅広い方々に本稿執筆陣によるこの書籍を手にとっていただければ幸いです。

SD

▼図4　時間別集計の実行結果

```
$ gawk -f analyze_hourly.awk success.tmp | sort↵
00,2
01,6
02,5
03,15
04,7
05,6
06,2
07,5
08,4
09,3
10,7
11,2
12,10
13,3
14,6
15,8
16,13
17,3
18,7
19,6
20,3
21,2
22,18
23,5
total,148
```

Case 4

手を動かしてデータを操ろう！

現場で使われるテキスト処理の実際

Postfix・Apacheのログを抽出して障害原因を特定

Author 荒井 健祐(あらい けんすけ)　㈱さくらインターネット
Mail kensuke.arai1987@gmail.com

はじめに

さくらインターネットで勤務している、荒井と申します。現在はおもに、弊社が提供するVPS、専用サーバの保守や運用、お客様からの技術的な問い合わせに対して、調査および回答を行う業務を担当しています。

今回はPostfixにおけるメールログや、Apacheのアクセスログなど、大量のテキストから障害原因や傾向を把握するのためのデータ抽出コマンドについて紹介します。

メールログの抽出

サーバ側でSMTP-AUTHを設定してセキュリティに注意していても、アカウントに対して安易なパスワードを設定し、第三者がそれらを悪用することでSPAMメールが大量に送信される、ということが運用上起こるかもしれません。そうした「メールアカウントの不正利用」が疑われる場合は、どのようにアプローチを試みるのが効率的でしょうか。筆者の場合は、接続元IPアドレスと認証アカウントを集計することで、どのアカウントが不正利用されているかを判断しています。**図1**はPostfixで認証が成功した際のメールログの一例です。

これを**図2**のawkコマンドを用いて見やすく集計します。

コマンドの詳細ですが、awk直後の「//」の間には、マッチングをさせたい行に含まれる文字列を指定します。今回の場合は、sasl_username(認証アカウント)に紐付く行を抽出します。そのマッチングされた行の7つ目と9つ目の要素(**図1**)を、さらに「=」を区切り文字としてsplitで分割します。分割された文字列は、一時的にbおよびc配列に格納されるので、各配列の2つ目の要素(接続元IPアドレス、認証アカウント部分)を取り出し、集計します。最後にsort -k 3でアカウント順にソートし、表示順序を整えます。

図2のような場合、複数国のIPアドレスから単一のアカウントへ認証をかけていること、認証回数もjpのIPアドレス以上に他国のIPアドレスの方が多いことから、不正利用の可能性が疑われます。ただし、複数人で単一のアカウントを共有している場合もあるため、一概にすべて不正利用と断定はできません。リソースの状況などを鑑みて、利用状況についてお客様へ確認するという判断もできるのではないでしょうか。

▼図1　Postfixで認証が成功した際のメールログ

```
Mar  2 13:20:04 server postfix/smtpd[29546]: 8C3252A2419: client=xxxx.ad.jp[xxx.xxx.xx.xx],
sasl_method=PLAIN, sasl_username=test1@example.com
```

client=xxxx.ad.jp[xxx.xxx.xx.xx]：接続元IPアドレス($7)
sasl_username=test1@example.com：認証アカウント($9)

▼図2　awkコマンドでメールログを集計

```
# awk '/sasl_username/ {split($7, b, /=/); split($9, c, /=/); print b[2],c[2]}'
 /var/log/maillog | sort | uniq -c | sort -k 3
          1 xxxx.ad.jp[xxx.xxx.xx.xx], test1@example.com
        471 xxxx.cn[xxx.xxx.xx.xx], test1@example.com
        681 xxxx.ru[xxx.xxx.xx.xx], test1@example.com
       1065 xxxx.de[xxx.xxx.xx.xx], test1@example.com
         34 xxxx.cn[xxx.xxx.xx.xx], test1@example.com
        732 xxxx.ru[xxx.xxx.xx.xx], test1@example.com
        354 xxxx.uk[xxx.xxx.xx.xx], test1@example.com
          6 xxxx.ad.jp[xxx.xxx.xx.xx], test2@example.com
         35 xxxx.ad.jp[xxx.xxx.xx.xx], test3@example.com
```

jpのIPアドレスから認証が1回であるのに対して、海外からのIPアドレス（複数）からtest1のアカウントに対して複数回認証がかけられている

▼図3　Webサーバのアクセスログの表示例

```
                    アクセス時刻（$4）
xx.xxx.xxx.xx - - [04/Mar/2015:09:58:44 +0900] "GET /index.html HTTP/1.1" 200 19 "-" "Mozilla/
5.0 (Windows NT 6.1; WOW64 rv:36.0) Gecko/20100101 Firefox/36.0"
          substr(2～14)
```

アクセスログの抽出

近頃、Webサーバに対してWordPressやShellshockなどの脆弱性調査を目的とした、海外からのアクセスが増加しています。ですので、そうした状況で使える、アクセスログの分析手法を紹介します。

Webサーバへのアクセスを見やすくするうえでもっとも効率的なのは、時間をフィルターにして集計する方法です。そこで、アクセスログから1時間ごとのアクセス数を集計してみましょう。**図3**はアクセスログの一例となります。ここに接続元IPアドレスというもう1つのフィルターを加えて表示させると、**図4**に示すワンライナーのスクリプトとなります。

まずfor文の繰り返しの範囲ですが、アクセスログの4番目の要素から、awkのsubstrを利用し、1時間ごとを指定します。指定された時間は${hour}として変数に格納されますので、その分grepをかけて対象時刻を限定し、接続元IPアドレスをカウントする、という流れになっています。

図4の実行結果を確認すると、xxx.xxx.xxx.251のIPアドレスからは定期的にほぼ同数のアクセ

▼図4　時間・接続元IPアドレスをフィルターにしてアクセスログを集計

```
# for hour in $(awk '{print substr($4,2,14)}'
 /var/log/httpd/access_log | sort | uniq);
 do echo -e "\n${hour}"; grep ${hour}
 /var/log/httpd/access_log
 | awk '{print $1}'
 | sort | uniq -c
 | sort -nr; done
01/Mar/2015:03
        45 xxx.xxx.xxx.251

01/Mar/2015:04
       228 xxx.xxx.xxx.251

01/Mar/2015:05
       229 xxx.xxx.xxx.251

01/Mar/2015:06
       229 xxx.xxx.xxx.251

01/Mar/2015:07
      6458 167.114.162.108
      2346 182.118.60.63
       228 xxx.xxx.xxx.251
       173 167.114.162.101
         1 94.102.53.195

01/Mar/2015:08
       228 xxx.xxx.xxx.251
```

ほかの時間帯に比べ、明らかにアクセス数が増加している。何をされているかの把握は必須！

▼図5　図4の結果に、さらに接続元IPアドレスと国別コードを紐づける

```
# for list in $(awk '{print $1}' /var/log/httpd/access_log | sort | uniq); ↵
 do printf "%-15s\t%6d\t%-3s %-s %-s\n" ${list} ↵
 $(grep ${list} /var/log/httpd/access_log | wc -l) ↵
 $(geoiplookup ${list} ↵
 | awk '/Country/ {print $4,$5,$6,$7}'); ↵
 done
xxx.xxx.xxx.251    4665  JP,   Japan
111.249.115.204    756  TW,   Taiwan
113.20.29.219      1  ID,   Indonesia
123.151.149.222    231  CN,   China
128.61.240.66      1  US,   United States
146.148.89.239     46  IP,   Address not found
167.114.162.101    273  US,   United States
167.114.162.105    1  US,   United States
167.114.162.108    6458  US,   United States
167.114.162.109    33  US,   United States
167.114.162.113    815  US,   United States
182.118.60.21      38  CN,   China
182.118.60.63      2346  CN,   China
xxx.xxx.xxx.233    /  JP,   Japan
```

下線のあるIPアドレスは**図4**実行時に記載のあったもの。アクセス数もほぼ一致しているため、海外から一時的にアクセスが増加していることがわかる

スを受信しているため、HTTPを利用した何らかのアプリケーションを利用している、もしくはクローラーなどで定期的にアクセスしていることが予想されます。しかし7時の段階では複数のIPアドレスから約9,000回のアクセスがあり、そのほかの時間と比較すると飛び抜けて多いため、この時間帯に何が行われたのかを調査する必要があります。コマンド中awk部分の「print $1」を「$7」へ変更し、「どのコンテンツに対してアクセスされているか」を確認すれば、スクリプトを使い回して簡単に調査できるはずです。

最後に接続元IPアドレスと国別コードを、**図5**のスクリプトで紐付けてみましょう。このスクリプトのポイントは`printf`と`geoiplookup`となります。前者はダブルクォートで括った中の記述フォーマットに合わせて、スペースに続く引数(今回はそれぞれのコマンドの実行結果)を整形し、出力させるコマンドになります。後者はGeoIPパッケージに含まれており、引数で指定されたIPアドレスの国別コードを検索して、結果を表示させます。検索先がローカルファイル(/usr/share/GeoIP/以下)となるため、適宜ファイルの更新が必要となる点に注意してください。

図5の結果と併せて表示結果を参照すると、「何時にどの程度の回数アクセスがあり、どのコンテンツにアクセスされ、それがどの国からなのか」を把握できます。こういった多角的な切り口があれば、脆弱性対応やリソース増強など、今後のサーバ運用方針を検討することができるようになるでしょう。

◆ ◆ ◆

ログなどの大量のテキストから、障害原因や傾向把握の調査をする際は、ログ全体の俯瞰(ふかん)から特定の条件でカテゴライズし、情報を見やすく抽出することが必要になります。

今回紹介した例はメールログ、アクセスログを題材としていますが、こうしたテキスト処理を即実行できるようにしておくことで、上述した調査や対策の検討にかかる時間を、飛躍的に短縮することができます。皆さんも独自のコマンドを考え、周囲へ展開し、より良い運用を目指していただければと思います。SD

Case 5 手を動かしてデータを操ろう！

現場で使われるテキスト処理の実際

構造化データを簡単に処理できる2つのコマンド

Author 水野 源(みずの はじめ)　㈱インフィニットループ
Twitter @mizuno_as

構造化されたテキストをもっと楽に処理しよう

UNIXの考え方の中に「すべてのデータはテキストとして保存せよ」「コマンドはフィルタとして振る舞え」というものがあります[注1]。パイプでコマンドを組み合わせることで、複雑なテキスト処理を組み立てられるのは、本特集を読むまでもなくみなさんよくご存じでしょう。UNIXライクなOSには、cut、tr、sed、sortといった、テキストを加工、整形するコマンドがたくさんあります。しかし、これらのコマンドは「文字列を操作する」ことに重点が置かれ、抽象度の高い構造を持ったデータを処理するのには向きません。たとえば最近ではWebサービスのAPIを叩き、結果をXMLやJSONで受け取ることも多いですが、これをcurlとsedで行うには、少々無理があります[注2]。そこで、知っておくと役に立つかもしれない、もう少しリッチなイマドキのフィルタコマンドを2つ紹介しましょう。

JSONをパースする「jq」

jqはコマンドラインから使える、軽量なJSONパーサです。Ubuntuではjqパッケージとして提供されていますので、次のように簡単にインストールできます。

```
$ sudo apt-get install jq
```

例として、livedoor天気情報「Weather Hacks」が提供しているREST API[注3]からJSONを取得し、jqを使ってパースしてみましょう。cityパラメータに取得したい地域のID番号を指定して、curlコマンドでAPI[注4]を叩きます(**図1**)。

jqは入力されたJSONに対し、引数で指定されたフィルタを適用して結果を出力します。まず「.」を指定してみましょう(**図2**)。「.」は何もしないフィルタで、入力されたJSONをそのまま出力します。ただし出力の際にJSONを整形するため、**図2**のようにAPIのレスポンスを人間が見やすい形に整える用途に利用できます。「.foo」のように、ドット

▼図1　道央(札幌、千歳、石狩など)の天気を取得する

```
$ WEATHER=$(curl -s "http://weather.livedoor.com/forecast/webservice/json/v1?city=016010"
 | sed -e 's/\\u\(....\)/\&#x\1;/g' -e 's/\\n//g' | nkf --numchar-input -w)
```

注1) Mike Gancarz(著), 芳尾 桂(訳)『UNIXという考え方―その設計思想と哲学』, オーム社, 2001.
注2) たとえばfreenodeにあるsedのIRCチャネルのトピックには「Do NOT try to parse markup (html, xml, etc) with sed!」という一文があります。
注3) URL http://weather.livedoor.com/weather_hacks/webservice
注4) サービス仕様にあるとおり、非ASCII文字はUnicodeエスケープシーケンスで表されているため、sedを使ってこれをUnicode数値文字参照に置換したうえで、nkfコマンドの--numchar-inputオプションでUTF-8に変換しています。またここでは後の解説で再利用しやすいよう、APIのレスポンスをいったん変数に格納し、変数をechoしてパイプでjqに渡しています。

▼図2 JSONを整形する

```
$ echo $WEATHER
{"pinpointLocations":[{"link":"http://weather.livedoor.com/area/forecast/0110000","name":"札幌市"},
(...略...)

$ echo $WEATHER | jq '.'
{
  "pinpointLocations": [
    {
      "link": "http://weather.livedoor.com/area/forecast/0110000",
      "name": "札幌市"
    },
(...略...)
```

に続けて文字列を指定すると、オブジェクトから指定されたキーを探し、その値を返します。またドットをつなげることで、オブジェクトの階層をたどれます。配列の要素を取り出す場合は「[0]」のように指定します。ブラケット内のインデックスを省略した場合は、配列内のすべての要素を返します。もちろん配列内のオブジェクトに対しても、キーを指定して要素を取り出すことができます(**図3**)。「Weather Hacks」ではforecastsという配列に今日から3日ぶんの天気や気温が含まれているので、**図4**のようにすれば3日ぶんの天気を取り出せます。

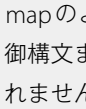
ほかにもjqは、文字列を結合したり数値を加減乗除する演算子、条件に合致するものを抽出するselectや、配列の全要素に処理を適用するmapのような関数、さらにはif-elseのような制御構文まで持っています。詳細はとても書ききれませんので、興味があったらマニュアルに目を通してみてください[注5]。

▼図3 キーを指定して値を抽出する

```
$ echo $WEATHER | jq '.title'
"道央 札幌 の天気"

$ echo $WEATHER | jq '.description.text'
" 千島近海に発達中の低気圧があって、北海道付近は
冬型の気圧配置となっています。この低気圧は次第に
東へ遠ざかり...(略)
```

▼図4 配列内に含まれるキーを指定して、3日ぶんの天気を取り出す

```
$ echo $WEATHER | jq '.forecasts[].telop'
"曇り"
"曇のち雪"
"曇り"
```



テキストをSQLで操作する「q」

テキストでデータを表す際、カンマやスペースでフィールドを区切った、2次元のテーブルにすることがよくあります。Excelで作成した表をCSVにエクスポートするのは一般的ですし、/etc/passwdはユーザ名やログインシェル、ホームディレクトリなどをコロンで区切って記録しています。こういったテキストのテーブルに対し、SQLライクな構文でデータの抽出を可能にするのがqです。Ubuntu 15.04以降では「python-q-text-as-data」というパッケージ[注6]で提供されています。Ubuntu 15.04以降の場合は、

```
$ sudo apt-get install python-q-text-as-data
```

でインストールできます。また、筆者のPPAでUbuntu 14.04向けのバックポートを提供していますので、Ubuntu 14.04の場合は、

```
$ sudo add-apt-repository ppa:mizuno-as/q-text-as-data
$ sudo apt-get update
$ sudo apt-get install python-q-text-as-data
```

注5) URL http://stedolan.github.io/jq/ オンラインでフィルタをテストすることもできます。
注6) Python 3.x向けのバイナリは「python3-q-text-as-data」です。

としてください。

qには、引数としてデータを抽出するためのSQLを指定します。「SELECT hoge FROM fuga」というありふれた形式ですので、RDBMSの利用経験がある人であれば、一目で理解できるでしょう。たとえば**表1**のような、メニュー名、価格、カロリーが書かれたCSVファイルがあるとします。この中からワンコインで食べられるメニューを抽出し、価格の安い順に並べてみましょう。もしもシェルスクリプトで書くとすれば、cutで価格のフィールドを抜き出し、testで大きさを比較、条件を満たした行だけをテンポラリファイルに書き出し、最後にsortする必要があるでしょう。しかしqを使えば**図5**のように、1行のSQLで期待した結果を得ることができます[注7]。

FROMに「-」を指定すれば、標準入力からデータを渡せます。たとえばpsコマンドの出力を集計し、ユーザごとに使用メモリ量(RSS、VSZ)の合計をする(**図6**)、なども簡単です。

◆ ◆ ◆

jqもqも非常に強力なコマンドですが、インストールされていない環境のほうが多い(むしろ使えるほうが珍しい)コマンドなのは間違いありません。やはりどんな環境にもある、cut、sort、tr、sedといったコマンドは避けて通ることはできないでしょう。しかし「こんな便利なコマンドもあるんだ」程度に覚えておくと、いつか役に立つ日がくるかもしれませんね。SD

▼表1　CSVファイル

	A	B	C
1	メニュー	価格	カロリー
2	牛めし	290	735
3	牛めし野菜セット	440	830
4	牛めし豚汁セット	520	1056
5	牛めしお新香セット	420	823
6	旨辛ネギたま牛めし	390	860
7	おろしポン酢牛めし	390	762
8	キムチ牛めし	390	781
9	プレミアム牛めし	380	737
10	麻婆カレー	430	908
11	オリジナルカレー	330	661
12	オリジナルカレギュウ	500	829
13	オリジナルハンバーグカレー	590	1074
14	キムカル丼	490	822
15	ビビン丼	450	767
16	ネギ塩豚カルビ丼	430	832
17	チキンガーリック定食	630	1029
18	牛焼肉定食	590	935
19	カルビ焼肉定食	630	943
20	豚バラ焼肉定食	550	898
21	スタミナ豚バラ生姜焼定食	590	933
22	肉野菜炒めセット	630	1093
23	デミたまハンバーグ定食	630	1064
24	鉄皿チキングリルセット	640	985
25	鉄皿デミたまハンバーグセット	640	1162
26	鉄皿うまトマハンバーグセット	640	1144

▼図5　CSVファイルから、ワンコインで食べられるメニューを価格順にソートして表示する

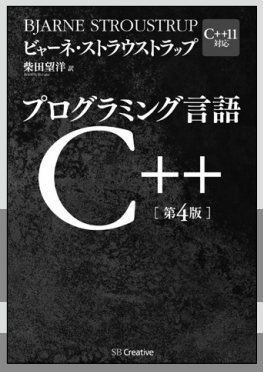
```
$ q -H -t "SELECT メニュー,価格 FROM menu.csv WHERE 価格 <= 500 ORDER BY 価格"
牛めし  290
オリジナルカレー  330
プレミアム牛めし  380
旨辛ネギたま牛めし  390
おろしポン酢牛めし  390
(...略...)
```

▼図6　psコマンドの結果を集計して、ユーザごとの使用メモリ量の合計を表示する

```
$ ps aux | q -H "SELECT USER,SUM(RSS),SUM(VSZ) FROM - GROUP BY USER"
postfix 2932 54868
root 36556 659664
syslog 1260 182112
www-data 54400 1028360
(...略...)
```

注7) -Hオプションはヘッダ行があることを、-tオプションはタブ区切りであることを表しています。

BOOK no.1 プログラミング言語C++［第4版］



ビャーネ・ストラウストラップ【著】、柴田 望洋【訳】
B5変形判、1,360ページ／定価＝8,800円＋税／発行＝SBクリエイティブ
ISBN＝978-4-7973-7595-4

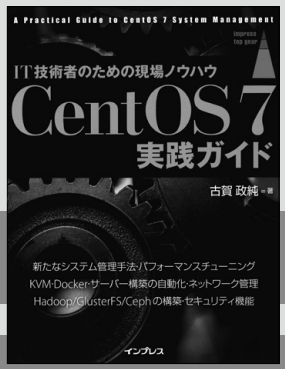
C++の開発者ビャーネ・ストラウストラップ自身による解説書（C++11対応）。1,360ページ・全44章と非常に厚い1冊となっているが、各章の独立性は強く、興味のある章から適宜読めるように作られている。すべての基本機能・すべての標準ライブラリから、オブジェクト指向・ジェネリックプログラミングのための抽象化機能まで、まさにC++のすべてを記した1冊と言える。制御構造とは何かといったプログラミングの基本については極力省かれており、やや上級者向けの解説書と言える。Visual Studioの無償化によってC++がより使いやすくなった今、マシンに近い言語を勉強したいといった人は、この本を片手に挑戦してみてはどうだろうか。

BOOK no.2 エバンジェリストの仕事術

西脇 資哲【著】
四六判、224ページ／定価＝1,500円＋税／発行＝日本実業出版社
ISBN＝978-4-534-05257-5

エバンジェリストと言えば抜きんでた才覚と技術を持ち合わせた憧れの職位だ。本書は、テクノロジの伝導師（エバンジェリスト）として、その仕事の内幕をかなりオープンに公開している。朝7時から始まる仕事のスケジュール管理から、何1つモノが置いていない机の上まで、さらには学生時代からどのように技術を習得してきたのか……あたかも技術者の自叙伝のようである。その中で多くのエンジニアに欠けている「伝える技術」の重要性をさまざまな観点から論じている。これはプレゼン技術向上の参考になるのではなかろうか。1つ気になるのは、縦書きの本ゆえか見出しで使われている不等号の向きが、わかりにくいことである。

BOOK no.3 CentOS 7実践ガイド



古賀 政純【著】
B5変形判、320ページ／定価＝3,000円＋税／発行＝インプレス
ISBN＝978-4-8443-3753-9

CentOS 7のシステム管理者向け、構築・運用・保守のノウハウをまとめた1冊。OSの機能ごとに、基礎知識から実践的な操作方法までを解説しているほか、コンテナ型仮想化の「Docker」、並列分散処理の「Hadoop」、分散ストレージ基盤の「GlusterFS・Ceph」という比較的新しいツールの使い方も扱っている。

「7」へのバージョンアップではサービス管理がSystemdに、ファイルシステムがxfsへ変更され、セキュリティ機能としてfirewalldが追加されるなど大きな仕様変更があったが、その変更点、ハマりどころについて各章で丁寧に述べられている。そのため、テスト環境での新機能の検証などにも適した本だと言える。

BOOK no.4 はじめよう! 要件定義 ～ビギナーからベテランまで

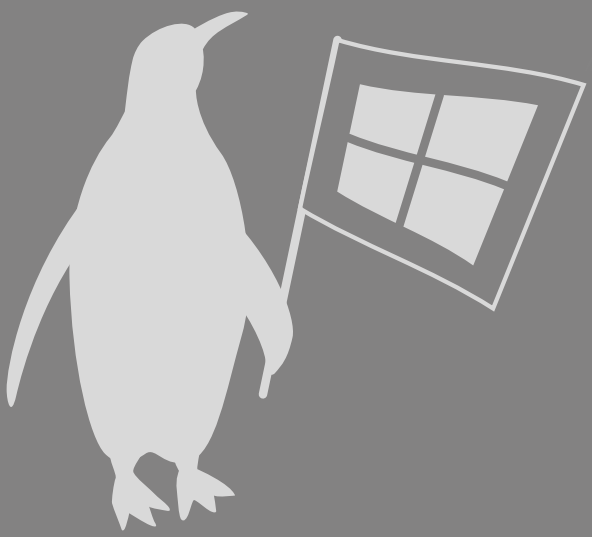
羽生 章洋【著】
四六判、184ページ／価格＝1,980円＋税／発行＝技術評論社
ISBN＝978-4-7741-7228-6

ソフトウェアが遍在化し、あらゆるものがデジタル化、Web化に向かう中で、顧客、そしてユーザが満足のいく形でITシステム／ソフトウェアを実現することの重要性が今後ますます増していくのは間違いない。本書はそのための重要なキーワード、「要件定義」をわかりやすく解説した書籍。書名のとおり、初心者にもわかりやすく、とっつきやすい内容であり、現場の若手エンジニアが要件定義についてポイントを押さえるために読むもよし、経験豊富なエンジニアがあらためておさらいとして読むもよし、という必要十分な解説になっている。同じテーマのほかの書籍と比べ、非常にコンパクトなので、息抜き程度に気軽に読めるのも良い。

第2特集

ファイル共有自由自在

［徹底入門］最新・Sambaの教科書



本誌、2013年2月号にSamba4.0.0の記事が掲載されました。それから約2年を経て現在のバージョンは4.2.0になっています。本特集では初心に返り、超定番のSambaサーバ構築テクニックをすみからすみまで徹底解説します。

第1章では、Windowsサーバ互換の機能を提供するオープンソースソフトウェアであるSambaの基本的な設定について解説します。

第2章では、Sambaのユーザ管理とファイル共有の基本的な設定について解説します。

第3章では、応用編としてActive Directoryへの認証連携について解説します。

ぜひ、皆さんのSamba環境構築にお役立てください。

Author たかはしもとのぶ　mail monyo@monyo.com　Twitter @dememonyo

第1章 Sambaのインストールと基本設定

Author たかはしもとのぶ mail monyo@monyo.com Twitter @damemonyo

本章では、Windowsサーバ互換の機能を提供するオープンソースソフトウェアであるSambaの基本的な設定について解説します。

Sambaとは

Sambaは、LinuxやFreeBSD、商用UNIXといった各種UNIX系プラットフォーム(以下Linuxと総称します)上で、ファイルサーバやドメインコントローラといったWindowsサーバ互換の機能やWindowsとLinuxとの連携機能を提供する、主要なオープンソースソフトウェアの1つです。

現在でも活発に開発が行われており、ほぼ月イチのペースでセキュリティやバグ修正版がリリースされているほか、次期バージョンの開発も平行して行われています。記事執筆時点の最新版は、3月4日にリリースされたSamba 4.2.0です。

Sambaの最新情報は、`https://samba.org/`や`https://wiki.samba.org/`から入手できます。

Sambaのおもな機能

最初に、Sambaが提供するおもな機能について簡単に紹介しておきましょう。

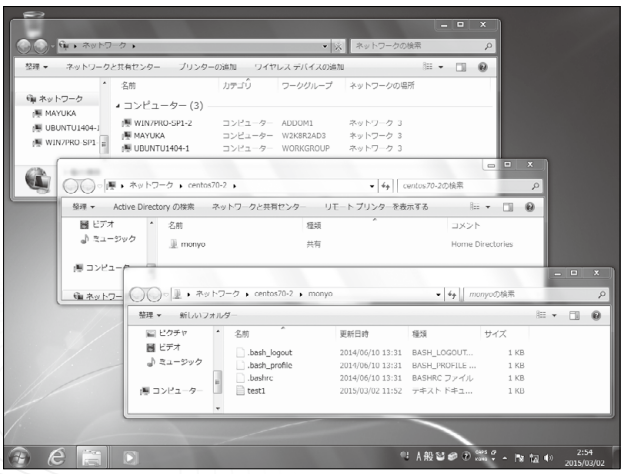
ファイルサーバ機能

Sambaを使用することで、Windowsのファイルサーバ機能を簡単に提供することができます。**図1**は、Sambaが動作しているLinuxサーバ(以下Sambaサーバ)にWindows 7クライアントからアクセスした際の画面イメージになります。このように一般のユーザが普通にアクセスしている限り、Windowsサーバとまったく見分けがつきません。

実際、数万円で販売されている廉価なネットワーク対応HDD(NAS)の多くにはSambaが内蔵されていますので、知らないうちにSambaを使っている方も多いかもしれません。

ファイルサーバの機能を活用することで、Linuxサーバとの間で気軽にファイル転送を行うこともできます。Linuxサーバとのファイルのやりとりには、WinSCPなどのツールやFTPを使うことも多いと思います

▼図1 Sambaサーバへのアクセス



が、Sambaを活用すればWindowsクライアントへのツールのインストールが不要であることに加え、Linuxサーバ上のファイルを直接編集することもできるので便利なことも多いでしょう。

Sambaサーバのファイルシステムが ACL(Access Control List)に対応している場合は、Windowsサーバ上のファイルと同様の操作で、各ファイルのプロパティから「セキュリティ」タブを選択すると表示される**図2**の画面からアクセス許可の設定を行うこともできます。

そのほか、少し複雑な設定が必要ですが、Windowsサーバの持つ分散ファイルシステム(DFS)やボリュームシャドウコピー機能といったエンタープライズ向けの機能を提供することができます。また、Windowsサーバと同様にプリンタサーバ機能を提供することもできます。

Active Directory連携機能

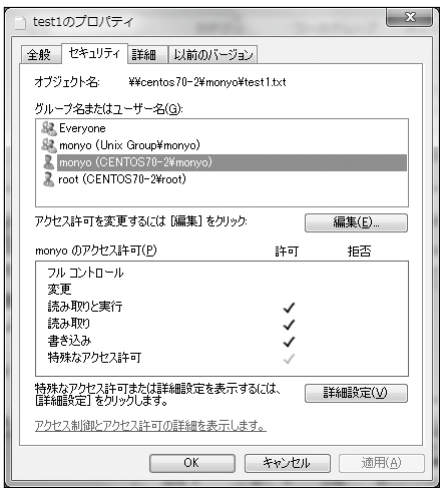
Sambaを構成することで、Linuxサーバを Active Directoryに「参加」させることができます。これにより、Sambaが提供するファイル共有にアクセスする際の認証を Active Directoryのユーザとパスワードで行うことが可能となります。さらにWinbindという機構を用いることで、**図3**のように Active Directoryのユーザやグループを自動的にLinuxサーバで使用することも可能になります。

PAM(Pluggable Authentication Module)を設定することで、sshなどのSamba以外のサービスの認証を Active Directoryで行うこともできます。

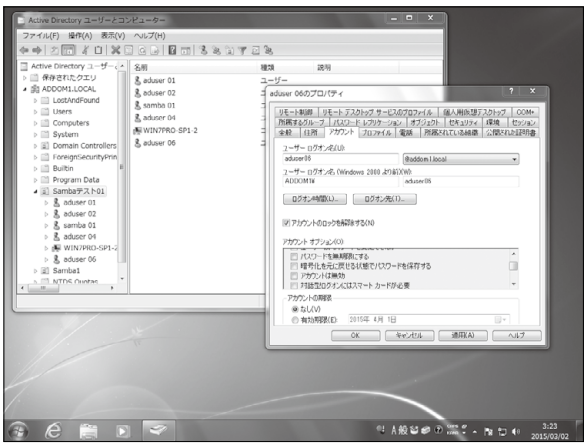
ドメインコントローラ機能

特殊な設定が必要ですが、Active Directoryのドメインコントローラとして機能することもできます。この機能は2012年12月にリリースされたSamba 4.0.0から提供されており、認証統合やグループポリシーを活用したクライアント管理を**図4**のようにWindowsの管理ツール(RSAT)から行うことが可能です。

▼図2 アクセス許可の設定画面



▼図4 RSATに含まれる「Active Directory ユーザーとコンピューター」による Active Directoryの管理



▼図3 Windowsドメインのユーザやグループの使用

```
# id W2K8R2AD1¥¥samba01
uid=10001(W2K8R2AD1¥samba01) gid=10000(W2K8R2AD1¥domain users) ➡
groups=10000(W2K8R2AD1¥domain users)
# getent passwd W2K8R2AD1¥¥samba01
W2K8R2AD1¥samba01:*:10001:10000:samba 01:/home/W2K8R2AD1/samba01:/bin/false
```

クライアント機能

SambaにはLinuxサーバからWindowsサーバ上のファイル共有にアクセスしてファイルのコピーを行うsmbclientコマンドやWindowsサーバのリモート管理を可能とするnetコマンドといった各種ユーティリティが付属しています。

smbclientによるファイルコピーの実行例を**図5**に示します。これらのユーティリティは自動で処理を実行することもできますので、業務システムでWindowsとLinuxとを連携させる際にも有用です。

ネットワーク機能

図6のように「ネットワーク」フォルダにLinuxサーバを表示させるブラウジング機能や、Microsoftネットワーク特有のWINSサーバやWINSクライアント機能といった機能も提供しています。

ここまでのまとめ

ここまでSambaの提供する機能について駆け足で解説しました。SambaはWindowsサーバ互換の機能を提供するため、各機能の詳細についてはWindowsの情報源も参照してください。

Sambaサーバのインストールと初期設定

ここからは、Red Hat Enterprise Linux(以下RHEL)のクローンとしてユーザが多いCentOSを例に、具体的なSambaのインストールとファイルサーバとしての設定について解説していきます。なおCentOSは、2014年7月にリリースされた7.0以降とそれより前のバージョンとで、設定が大きく異なっています。ここでは7.0以降を中心に両方の設定方法を解説し、併せてUbuntu 14.04 LTS(以下Ubuntu)での設定方法についても簡単に解説します。

設定方法はGUI、CUIなどいくつかの方法がありますが、サーバ用途でインストールする場合はGUIをインストールしないことも多いので、ここでは最小インストール状態でも設定可能な方法を中心に説明します。

▼図6　ブラウジング機能

▼図5　smbclientコマンドの実行例

```
# smbclient //madoka/monyo -U monyo
Enter monyo's password: ← パスワードを入力
Domain=[HOME] OS=[Unix] Server=[Samba 3.5.6]
smb: ¥> cd Archives
smb: ¥Archives¥> dir
  .                                   D        0  Sun Feb 15 11:40:47 2015
  ..                                  D        0  Mon Mar  2 03:37:18 2015
  pam_ldap.tgz                        A   163437  Fri Jan 14 08:02:01 2011
  Sharity-Light                       D        0  Sun Jun  8 13:00:47 2014
  Samba                               D        0  Wed Feb 11 11:41:20 2015
  rktools.exe                         A 12337752  Sun Jun  8 11:25:04 2014

                65535 blocks of size 33553920. 45962 blocks available
smb: ¥Archives¥> get rktools.exe
getting file ¥Archives¥rktools.exe of size 12337752 as rktools.exe
(3051.8 KiloBytes/sec) (average 3051.8 KiloBytes/sec)
smb: ¥Archives¥> quit
#
```

Sambaサーバのインストール

CentOSなど汎用のLinuxディストリビューションでは、例外なくSambaのパッケージが提供されています。以降では、パッケージを使用したSambaのインストールについて解説します。

▶パッケージからのSambaインストール

RHELやCentOSを含むRHEL互換ディストリビューション(以下RHEL系)やUbuntuでのSambaは複数のパッケージから構成されていますが、RHEL系、UbuntuともSambaサーバ本体の機能はsambaというパッケージによって提供されています。

sambaパッケージのインストール状態は次のようにして確認できます[注1]。

```
# rpm -q samba
```

sambaパッケージが未インストールの場合は、次のようにyumコマンドでインストールします[注2]。

```
# yum install samba
```

これらのコマンドによりsambaパッケージの動作に必要な各種パッケージも自動的にインストールされます。

注1) Ubuntuでは「dpkg -l samba」コマンドでインストール状況を確認できます。
注2) Ubuntuでは、「apt-get install samba」コマンドなどでsambaパッケージをインストールします。

▼図7 ファイアウォールの設定変更例(CentOS 7.0)

```
[root@centos70 ~]# firewall-cmd --add-service=samba
success
[root@centos70 ~]# firewall-cmd --add-service=samba --permanent
success
```

▼図8 ファイアウォールの設定変更例(CentOS 6.X)

```
[root@centos66 ~]# lokkit --service=samba
[root@centos66 ~]# service iptables restart
iptables: Setting chains to policy ACCEPT: filter          [  OK  ]
iptables: Flushing firewall rules:                         [  OK  ]
iptables: Unloading modules:                               [  OK  ]
iptables: Applying firewall rules:                         [  OK  ]
```

なお、次章ではsmbpasswdというコマンドを使用しますが、CentOS 7.0のsambaパッケージには含まれていないため、CentOS 7.0では次のようにしてsamba-clientパッケージもインストールしておいてください[注3]。

```
# yum install samba-client
```

▶ファイアウォール設定の変更

CentOSでは、セキュリティ強化のため各種セキュリティ設定がデフォルトで有効になっています。そのままではWindowsマシンからSambaサーバにアクセスできませんので設定変更が必須です。

まずはファイアウォールの設定を変更して、Sambaサーバへのアクセスに必要な「137/udp、138/udp、139/tcp、445/tcp」の4つのポートを開放します。CentOS 7.0以降ではfirewalldという新しいサービスがファイアウォールの設定を管理しており、**firewall-cmd**コマンドで設定を行います。設定例を**図7**に示します。

Sambaサーバへのアクセスに必要なポートはsambaという名称で定義済みです。**--add-service=samba**を指定することでポートが直ちに開放されますが、この設定は再起動すると元に戻ってしまいます。別途**--permanent**オプションを指定してコマンドを実行することで、この設定がファイルに保持され、再起動後も設定が維持されるようになります。

CentOS 6.Xでは**lokkit**コマンドなどで設定を行います。設定例を**図8**に示します[注4]。

注3) CentOS 6.X以前やUbuntuでは、sambaパッケージにsmbpasswdコマンドが含まれていますので、samba-client(Ubuntuではsmbclient)パッケージのインストールは不要です。Sambaの管理にsmbpasswdコマンドは必須ではないですが、利便性を考慮し、ここではインストールを前提とした解説を行います。
注4) 筆者が確認した限り、最小インストール構成ではlokkitコマンドの設定をもとに戻すには/etc/sysconfig/iptablesと/etc/sysconfig/iptables-configファイルを直接修正する必要がありました。

設定はiptablesサービスの再起動後に反映されます。

CentOS 5.X以前では、コマンドラインでlokkitコマンドを起動すると**図9**の画面が表示されますので、ここから「Customize」ボタンを押すと表示される**図10**の画面から設定を行います。

もちろん、上記以外の方法で設定を行ってもかまいません。またSambaを動作させるうえではファイアウォール機能自体を無効にしてもかまいませんが、セキュリティ上は可能な限り有効にしておくことを推奨します。

Ubuntuの場合、ファイアウォールはデフォルトが無効ですので、とくに設定を行う必要はありません。

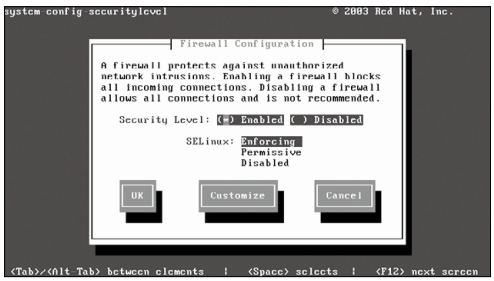
▶ SELinuxの無効化

引き続き、SELinuxの設定を変更します。SELinuxはセキュリティを高める機能としては非常に有用なのですが、反面熟練者でも適切に設定して運用するのが難しい機能です。ファイアウォールを適切に設定していれば外部からのアクセスについては必要最低限に絞ることができるため、初心者は、まずはSELinuxを無効化した状態で設定することをお勧めします。

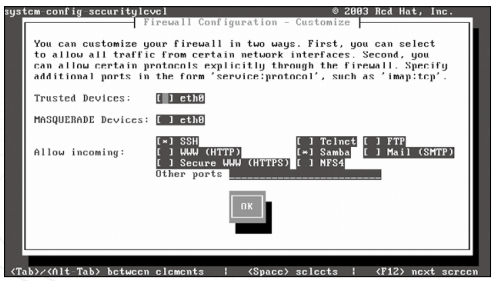
リスト1のように/etc/selinux/configファイル中のSELINUX行をdisabledにして再起動することでSELinuxが無効になります。

Ubuntuの場合、SELinuxはデフォルトで無効ですので、とくに設定を行う必要はありません。

▼図9　lokkitの起動画面（CentOS 5.X）



▼図10　ファイアウォールの設定変更例（CentOS 5.X以前）



▼リスト1　SELinuxを無効化する設定（CentOS 7.0）

```
...
#     disabled - No SELinux policy is loaded.
SELINUX=disabled ← この行を変更する
# SELINUXTYPE= can take one of these two values:
...
```

Sambaの起動と停止

Sambaはサービスとして動作しますので、サーバを再起動しなくても起動、停止を行うことができます。またサーバ起動の際に自動起動を行うかどうかも個別に制御できます。

RHEL系の場合、Sambaをインストールしただけではsambaは起動しません。またサーバ起動の際にも自動起動しません。CentOS 7.0でのSambaの起動／停止は**図11**のようにコマンドラインから`systemctl`コマンドを使用して行います。Sambaのサービス名は歴史的経緯で「smb」、「nmb」ですので注意してください。

サーバ起動時にSambaを自動起動させる場合は、**図12**のように設定します。

自動起動を止めたい場合は、enableの代わり

▼図11　コマンドラインからのSambaの起動と停止（CentOS 7.0以降）

```
[root@centos70 ~]# systemctl start smb
[root@centos70 ~]# systemctl start nmb

[root@centos70 ~]# systemctl stop  smb
[root@centos70 ~]# systemctl stop  nmb
```

▼図12　サーバ起動時にSambaを自動起動させる（CentOS 7.0以降）

```
[root@centos70 ~]# systemctl enable smb
[root@centos70 ~]# systemctl enable nmb
```

にdisableを指定してください。

CentOS 6.X以前でのSambaの起動、停止は、**図13**のように**service**コマンドで行います。

CentOS 6.X以前でサーバ起動時にSambaを自動起動させる場合は、**図14**のように**chkconfig**コマンドで設定します。

自動起動を止めたい場合は、onの代わりにoffを指定します。

Ubuntuの場合はインストールが完了した時点でSambaが自動起動し、サーバ起動時にも自動起動する設定が有効になります。手動で停止、起動する設定を**図15**に示します[注5]。

サーバ起動時のSambaの自動起動を無効にする場合は、/etc/init配下のsmbd.confおよびnmbd.confを、たとえばsmbd.conf.disableやnmbd.conf.disableのようにリネームします。

こうした設定により、Sambaを構成するnmbdとsmbdという名前のプロセスが起動、停止します[注6]。**ps**コマンドによるプロセス起動の確認例

注5) DebianやUbuntuでは、「/etc/init.d/samba [start|stop]」コマンドで起動、停止を行い、update-rc.dコマンドでシステム起動時の自動起動の設定を行います。

注6) 設定によってはwinbinddというプロセスやsambaというプロセスの起動が必要な場合もあります。

▼図15 コマンドラインからのSambaの起動と停止（Ubuntu）

```
root@ubuntu:~# initctl stop smbd
smbd stop/waiting
root@ubuntu:~# initctl stop nmbd
nmbd stop/waiting
root@ubuntu:~# initctl start nmbd
nmbd start/running  process 1083
root@ubuntu:~# initctl start smbd
smbd start/running  process 1088
```

▼図13 コマンドラインからのSambaの起動と停止（CentOS 6.X以前）

```
[root@centos 66 ~]# service smb stop
SMB サービスを停止中                                       [  OK  ]
NMB サービスを停止中                                       [  OK  ]
[root@centos 66 ~]# service smb start
SMB サービスを起動中                                       [  OK  ]
NMB サービスを起動中                                       [  OK  ]
```

▼図14 サーバ起動時にSambaを自動起動させる（CentOS 6.X以前）

```
[root@centos 66 ~]# chkconfig smb on        ← 自動起動の有効化
[root@centos 66 ~]# chkconfig --list smb    ← Sambaの起動状態の確認
smb             0:off   1:off   2:on    3:on    4:on    5:on    6:off
                                  ↑ on になっていることが確認できる
```

Column 「SELinux無効化」の是非について

インターネット上で、強固なセキュリティを提供するSELinuxを安易に無効化することへの是非がよく議論されていますが、それらの議論を意識しつつも、運用の容易性とセキュリティを天秤にかけた結果、ここではSELinuxを無効化することを推奨することにしました。

本文で書いたとおりSELinuxの運用は難易度が高く、筆者は熟練者であっても正しく運用することは難しいと考えています。またベンダの商用ミドルウェアを導入する際には無効化を求められることが多いため、社内の業務サーバでは真っ先に無効とされてしまうことが多いと考えていますので、SELinuxを有効にするケースは実態として少ないと考えているためです。

ただし、本稿のCentOS 7.0の設定例はSELinuxを有効にした環境で確認しています。またSELinuxを有効にした環境での注意点についても「注」などの形で補足することで、SELinuxを有効にした環境にも配慮しました。

を図16に示します。

Sambaの基本設定

Sambaの設定は、おもにsmb.confファイルで行います。このファイルのデフォルトのパスはRHEL系、Ubuntuとも/etc/samba/smb.confになります。

smb.confファイルの構造

smb.confファイルはリスト2のような構造をしています。

Sambaでは、smb.confファイルで設定可能なオプションのことを「パラメータ」と呼び、各パラメータの設定は「(パラメータの)値」と呼びます。値にはデフォルト値があり、明示的に設定されなかった場合は、デフォルト値が設定されたものとして扱われます。

▼図16　psコマンドによるSambaの起動確認例(CentOS 7.0)

```
[root@centos70 ~]# ps ax | grep mbd
 2511 ?        Ss     0:00 /usr/sbin/smbd
 2512 ?        S      0:00 /usr/sbin/smbd
 2573 ?        Ss     0:00 /usr/sbin/nmbd
 2578 pts/0    R+     0:00 grep --color=auto mbd
```

[]で囲まれた行から次の[]で囲まれた行までの間が1つの「セクション」となり、[]で囲まれた文字列(リスト2ではglobalやセクション名1など)がセクション名となります。セクション名は基本的に共有名に対応しますが、表1で説明

▼リスト2　smb.confファイルの構造

```
[global]
  パラメータ名 = 値
  パラメータ名 = 値
  …

[homes]
  パラメータ名 = 値
  …

[セクション名1]
  パラメータ名 = 値
  …

[セクション名2]
  …
```

▼表1　特殊なセクション

global	Samba全体の設定を記述する。特定の共有には関連付けられていない
homes	各ユーザのホームディレクトリを一括して共有する際の設定を記述する
printers	サーバで定義されているプリンタを一括して共有する際の設定を記述する

Column　Sambaの脆弱性対応

残念ながら、Sambaも脆弱性と無縁ではありません。本稿の執筆中にもSambaに関する脆弱性が報道されました。

・CVE-2015-0240: Unexpected code execution in smbd
https://www.samba.org/samba/security/CVE-2015-0240

Sambaの本家からは、セキュリティ対応は新規バージョンのリリースという形で行われます。実際、上記脆弱性に対応してSamba 4.2.0rc5、Samba 4.1.17、Samba 4.0.25、Samba 3.6.25といったバージョンがリリースされました。

しかしRHEL系やUbuntu系などのディストリビューションでは、基本となるSambaのバージョンは変えずに該当の脆弱性対策のみを取り込んだパッケージをリリースすることで脆弱性に対応することが多いようです。

通常即日～数日で脆弱性に対応した新しいパッケージのリリースがアナウンスされています。そのためパッケージ版のSambaを使用している場合は、最新版のパッケージを適用し続けることが、既知の脆弱性に対処する最善策だと考えてよいでしょう。

パッケージの更新は、新規インストールと同様に「yum update samba」や「apt-get upgrade samba」コマンドで行えます。

する3つのセクションだけは特殊な意味を持っています。

パラメータ名の大文字小文字の違いや、パラメータ名の前後およびパラメータ名中の空白は無視されます。たとえばnetbios nameというパラメータ名は、**リスト3**のどの書式で記述しても文法上は同義です。

「#」や「;」から始まる行はコメントとして扱われます。

RHEL系、Ubuntuともに、デフォルトのsmb.confは多くのコメントが含まれていて非常に長大になっていますが、実際に定義されているパラメータはわずかです。

定義済みのパラメータだけを抽出して表示する例を**図17**に示します。

globalセクションの基本設定

Samba全体の設定であるglobalセクションについて、実用上、最低限必要な設定について説明します。

▶日本語に関する設定

日本語ファイル名を正しく扱う上で、**リスト4**の設定が必要です。デフォルトのsmb.confにはこの設定がないので追加してください。

この設定はsmb.conf自身の文字コードも決定するため、[global]という行の直下に記述することを推奨します。

・dos charset = CP932

日本語の環境であることを指定します。

・unix charset = 文字コード

日本語ファイル名に使いたい文字コードに応じてUTF-8(デフォルト値)、EUCJP-MS、CP932のいずれかを指定します。

既存の環境との互換性などの理由で文字コードとして伝統的なEUCJP-MS(EUC)やCP932(シフトJIS)を使っている場合は、文字化けが発生しないように適切な設定を行ってください。

▼リスト3　パラメータ名の記述例

```
netbios name = sambasv
net BIOS name = sambasv
Net B I O S name= sambasv
```

▼リスト4　日本語を使用する設定

```
[global]
  dos charset = CP932
  unix charset = UTF-8(もしくはEUCJP-MSやCP932)
  ⋮
```

▼図17　smb.confの定義済みパラメータの表示例(CentOS 7.0)

```
[root@centos70 ~]# cat /etc/samba/smb.conf | egrep -v ^'[[:space:]]*[#;]' |grep -v '^$'
[global]
        workgroup = MYGROUP
        server string = Samba Server Version %v
        log file = /var/log/samba/log.%m
        max log size = 50
        security = user
        passdb backend = tdbsam
        load printers = yes
        cups options = raw
[homes]
        comment = Home Directories
        browseable = no
        writable = yes
[printers]
        comment = All Printers
        path = /var/spool/samba
        browseable = no
        guest ok = no
        writable = no
        printable = yes
```

とくに理由がなければデフォルト値のUTF-8のままでかまいません(その場合は指定不要です)。

▶ ネットワーク機能に関する設定

SambaではWindowsサーバと同様、**図6**のように「ネットワーク」上に自身のアイコンを表示させることができます。以降に関連する設定を紹介します。

・netbios name = コンピュータ名

デフォルトでは、サーバのホスト名がコンピュータ名としてそのまま「ネットワーク」上で表示されますが、事情があってホスト名とは異なる名前を設定したい場合は、このパラメータで任意のコンピュータ名を指定してください。

▶ Sambaを有効化するネットワークインターフェースに関する設定

Sambaのデフォルトでは、接続されているすべてのネットワークインターフェースでSambaが有効化されます。特定のインターフェースのみでSambaを有効化したいという場合は、次の設定を行います。

・interfaces = インターフェース(インターフェース名・IPアドレス)

インターフェースとしてはeth0やeth1といったインターフェース名やIPアドレスを指定します。なおSambaの動作に支障が出ないよう、必ず127.0.0.1をインターフェースに含めてください。

・bind interfaces only = yes

interfacesパラメータで指定したインターフェースのみでSambaを有効化します。

▶ ログ出力に関する設定

Sambaのログは/var/log/samba以下に出力されます。ログの詳細度のデフォルトは0で、本当に重大なログしか出力されません。詳細度を変更する場合は、次の設定を行います。

・log level = 詳細度(数値)

運用中は最大でも詳細度を3程度にしておくことをお勧めします。一時的にログの詳細度を変更したい場合は、**図18**のように`smbcontrol`コマンドで変更、確認することもできます[注7]。

smb.confの設定確認

`testparm`コマンドを使用することで、smb.confファイルの文法や、有効になっている設定を確認できます。

引数なしでコマンドを起動すると、**図19**のようにsmb.confファイルで有効になっている設定を表示するとともに、問題があるとエラーメッセージを表示します。

ここではbrowseaableという存在しないパラメータについてのエラーメッセージが表示されています。`testparm`コマンドは、これ以外にも簡単な文法ミスや矛盾もチェックしてくれます

注7) ログの詳細度の変更はプロセスごとに行います。Sambaは通常smbd、nmbd、winbinddというプロセスで構成されますので、必要な場合はsmbdという個所をnmbdやwinbinddに置き換えてコマンドを実行してください。

▼図18 ログの詳細度の動的な変更、確認

```
[root@centos70 ~]# smbcontrol smbd debug 1     ← smbdのログ詳細度を1に設定
[root@centos70 ~]# smbcontrol smbd debuglevel  ← smbdのログ詳細度の確認
PID 2451: all:1 tdb:1 printdrivers:1 lanman:1 smb:1 rpc_parse:1 rpc_srv:1 rpc_cli:1
passdb:1 sam:1 auth:1 winbind:1 vfs:1 idmap:1 quota:1 acls:1 locking:1 msdfs:1 dmapi:1
registry:1 scavenger:1 dns:1 ldb:1
[root@centos70 ~]# smbcontrol smbd debug 0     ← smbdのログ詳細度を0に設定
[root@centos70 ~]# smbcontrol smbd debuglevel  ← smbdのログ詳細度の確認
PID 2451: all:0 tdb:0 printdrivers:0 lanman:0 smb:0 rpc_parse:0 rpc_srv:0 rpc_cli:0
passdb:0 sam:0 auth:0 winbind:0 vfs:0 idmap:0 quota:0 acls:0 locking:0 msdfs:0 dmapi:0
registry:0 scavenger:0 dns:0 ldb:0
```

のでsmb.confファイルを修正したらtestparmコマンドで確認する癖をつけておきましょう。

testparmコマンドの書式と主なオプションを表2に示します。

書式：testparm [-s][-v][smb.confファイルのフルパス]

図19の実行例では、smb.confの内容を出力する前に一度ユーザからの入力待ちとなります。これを行いたくない場合は-sオプションを指定します。

また、デフォルトではデフォルト値のままのパラメータ行は表示されませんが、-vオプションを指定することですべてのパラメータ行を表示させることができます。多くのパラメータを変更した場合などは、意図したとおりに変更が行われているかどうかを確認するために、図20のようにして新旧両smb.confの差分を確認してみるのもよいでしょう。

まとめ

ここまででSambaのインストール、起動方法から、smb.confファイルの設定方法と、Samba全体の設定を制御するglobalセクションの設定について簡単に説明しました。

次章では、引き続きユーザの作成と最低限のファイル共有の設定、Windowsクライアントの接続といった実用的な最低限のファイル共有設定について説明します。SD

▼図19 testparmコマンドの実行例

```
[root@centos70 ~]# testparm
Load smb config files from /etc/samba/smb.conf
rlimit_max: increasing rlimit_max (1024) to minimum Windows limit (16384)
Unknown parameter encountered: "browseaable"   } 不正な（存在しない）
Ignoring unknown parameter "browseaable"       } パラメータに関する警告
Processing section "[homes]"
Processing section "[printers]"
Loaded services file OK.
Server role: ROLE_STANDALONE
Press enter to see a dump of your service definitions   ← Enterを入力

[global]
        workgroup = MYGROUP
        server string = Samba Server Version %v
        log file = /var/log/samba/log.%m
        max log size = 50
        idmap config * : backend = tdb
        cups options = raw

[homes]
        comment = Home Directories
        read only = No
        browseable = No
（以下略）
```

▼表2 testparmの主なオプション

オプション	説明
-s	smb.confの内容を表示する前に確認を求めない
-v	デフォルト値のパラメータもすべて表示する（デフォルトの動作は、デフォルト値以外の値を設定したパラメータのみを表示する）
smb.conf ファイルのフルパス	解析対象のsmb.conf ファイル。指定しない場合はデフォルトのパスのsmb.conf ファイルを解析する

▼図20 新旧smb.confの差分を確認する

```
$ testparm -s -v smb.conf.old > smb.conf-testparm.old.txt
$ testparm -s -v smb.conf.new > smb.conf-testparm.new.txt
$ diff -u  smb.conf-testparm.old.txt smb.conf-testparm.new.txt
```

第2章 Sambaのユーザ管理とファイル共有の基本設定

Author たかはしもとのぶ mail monyo@monyo.com Twitter @damemonyo

本章では、前章に引き続きWindowsサーバ互換の機能を提供するオープンソースソフトウェアであるSambaのユーザ管理とファイル共有の基本的な設定について解説します。

Sambaユーザの作成と管理

Sambaサーバにアクセスするには、何らかの方法で認証を行う必要があります。SambaはActive Directoryで認証を行うこともできますが、ここではSambaサーバ上で独自にユーザを作成してパスワードを設定する方法を解説します。

SambaユーザとLinuxユーザ

RHEL系、Ubuntuともに、サーバにログインする際にはSSHの公開鍵認証を使用している場合などをのぞき、通常ユーザ名とパスワードの入力が必要です。通常ユーザ情報は/etc/passwdファイルに、ハッシュ化されたパスワード情報は/etc/shadowファイルにそれぞれ格納されています。

これらの情報をそのまま活用できればよいのですが、Windowsユーザの属性をすべてサポートするには/etc/passwdファイルに格納された情報では不十分です。また、WindowsではLinuxとは異なるアルゴリズムを使ったNTLMハッシュという形式のパスワード情報を使用します。一例として「P@ssw0rd」という文字列をハッシュ化した際の文字列を次に示します。

・Linux(MD5ハッシュ)

```
$6$A792mjea$TklBcknsM19NAwjU0kFffkNrz
6J8.qyugha.JEolao/aDjcKFq9SQ.jjNe0C4Jn
2FFl3HhhpBj9phmTeU59F40
```

・Windows(NTLMハッシュ)

```
E19CCF75EE54E06B06A5907AF13CEF42
```

このようにハッシュ化した文字列が異なるため/etc/shadowファイルの情報も共有できません。このため、Sambaではサーバ上に存在するユーザ(Linuxユーザ)とは別にSambaユーザという独自のユーザが必要です[注1]。

ただし、認証されたSambaユーザがサーバ上のファイルにアクセスする際には、何らかのLinuxユーザの権限で行う必要があります。そのため、Sambaユーザには必ず対応するLinuxユーザが必要であり、認証成功後にはLinuxユーザとの対応付けが行われます。

▶Sambaの認証処理

ここまで説明したSambaユーザとLinuxユーザが、実際にWindowsクライアントからSambaサーバにアクセスする際にどのように動作するのかを**図1**に示しました。この図を例に少し動作を説明します。

①まずは、Windowsユーザ名と、ユーザが入力

注1) Windowsで平文パスワードによる認証を有効にすればこの限りではありませんが、セキュリティ上推奨されません。

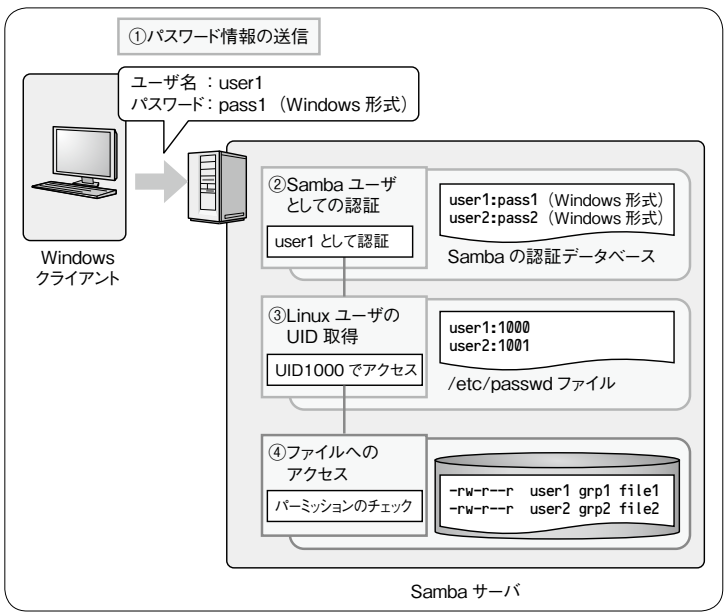
したパスワードをWindows形式でハッシュ化したパスワード文字列の情報がSambaサーバに送付されます。

② SambaはWindowsユーザと同じ名前のSambaユーザを検索し、見つかった場合はパスワード文字列を比較します。合致している場合はSambaユーザとしての認証を成功させます。

③続いてSambaユーザと同じ名前のLinuxユーザを/etc/passwdファイルなどから検索し、見つかった場合はユーザID(UID)情報を取得します。/etc/shadowに格納されているLinuxユーザのパスワード情報は参照されませんので注意してください。

④最終的にUID情報を使ってSambaサーバ上の各ファイルにアクセスします。ファイルに適切なパーミッションなどが付与されてない場合、アクセスは拒否されます。

▼図1 Sambaの認証処理



Sambaユーザの管理

Sambaユーザの作成や削除といった操作は、基本的にpdbeditコマンドで行います[注2]。**pdbedit**コマンドのおもな引数を**表1**に示します。

▶ Sambaユーザの作成

Sambaユーザを作成するには、「**pdbedit -a**」コマンドを使用します。

Sambaユーザを作成する際には、同名のLinuxユーザがすでに存在している必要がありますので、**useradd**コマンドなどを用いて事前に作成しておいてください。なお前述したようにLinuxユーザのパスワード情報は使いませんので、パスワードの設定は不要です。

▶ Sambaユーザの削除

Sambaユーザを削除するには、「**pdbedit -x**」コマンドを使用します。対応するLinuxユーザは削除されないので、不要な場合は別途削除してください。

Sambaユーザの作成、削除の実行例を**図2**に示します。

注2) 以前のSambaでは同じ目的でsmbpasswdコマンドが提供されていました。現在でもこのコマンドを使ってユーザの作成、削除を行うこともできます。

▼表1 pdbeditコマンドの主な引数

引数	説明
-a | --create <**Sambaユーザ名**>	Sambaユーザの追加
-x | --delete <**Sambaユーザ名**>	Sambaユーザの削除
-t | --password-from-stdin	パスワードのバッチ入力
-L | --list	Sambaユーザの一覧表示
-w | --smbpasswd-style	古いsmbpasswdファイル形式での一覧表示

▶ Sambaユーザの有効化、無効化

Sambaユーザを無効化することで、作成済みのSambaユーザの情報を保持したままログオンを禁止することができます。Sambaユーザの無効化、有効化は**smbpasswd**コマンドで行います[注3]。

Sambaユーザmonyoを無効化、有効化する際の実行例を次に示します。

```
# smbpasswd -d monyo
Disabled user monyo.
# smbpasswd -e monyo
Enabled user monyo.
```

▶ Sambaユーザのパスワード変更

Sambaユーザのパスワードを変更するには、**smbpasswd**コマンドを使用します[注4]。**passwd**コマンドと同様、rootはユーザ名を指定して任意のSambaユーザのパスワードを変更できます。一般ユーザは自分のパスワードだけを変更できます。一般ユーザによる実行例を次に示します[注5]。

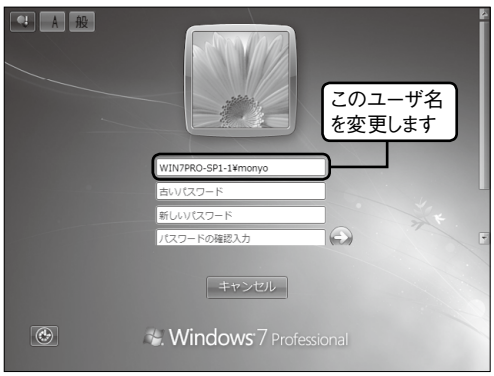
```
$ smbpasswd
Old SMB password:          ←現パスワードを入力
New SMB password:          ←新パスワードを入力
Retype new SMB password:   ←新パスワードを再入力
Password changed for user monyo
```

一般ユーザが自身のパスワードを変更する際は、デフォルトで5byte以上のパスワード入力が求められますので注意してください[注6]。

一般ユーザにLinuxのコマンドラインを使わせたくない場合、Windows 7までのクライアントであれば、少々面倒ですが、Alt+Ctrl+Delを押すと表示される画面から「パスワードの変更」を選択すると表示される**図3**の画面で表示されているユーザ名を「Sambaサーバのコンピュータ名¥ユーザ名」に変更のうえ、古いパスワード、新しいパスワードに適切なパスワードを指定することでパスワードの変更ができます。

注3) smbpasswdコマンドをインストールしていない場合、「pdbedit -c」コマンドで行うこともできますが、smbpasswdコマンドの方がより直感的です。
注4) smbpasswdコマンドをインストールしていない場合、「pdbedit -a」コマンドでユーザを再作成することでパスワードの再設定を行います。
注5) 「-s」オプションによりパスワード変更をバッチ処理で行うこともできます。
注6) この制限は「pdbedit -P "min password length"」コマンドで変更できます。

▼図3　Windows 7のパスワード変更画面



▼図2　Sambaユーザの作成と削除

```
# useradd -m monyo  ← UNIXユーザmonyoを作成（パスワードは設定不要）。ホームディレクトリも作成する 注7
# pdbedit -a monyo  ← Sambaユーザmonyoを作成
new password:  ← パスワードの入力
retype new password:  ← 再度パスワードの入力
Unix username:        monyo
NT username:
Account Flags:        [U          ]
（...中略...）
Last bad password   : 0
Bad password count  : 0
Logon hours         : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
# pdbedit -x monyo  ← Sambaユーザmonyoを削除
# userdel -r monyo  ← UNIXユーザmonyoを削除。ホームディレクトリも削除する
```

注7) CentOSの場合-mオプションを指定しなくてもホームディレクトリは作成されます。

▶ Sambaユーザの情報確認

「pdbedit -L」コマンドで作成済みのSambaユーザを一覧表示できます。実行例を**図4**に示します。

特定のSambaユーザの詳細情報を表示する場合は、次のように「pdbedit -v」コマンドに続いてSambaユーザ名を指定します[注8]。

```
# pdbedit -v monyo
```

Windowsクライアントからのアクセス

ここまでの設定で、ようやくWindowsクライアントからアクセスする準備が整いました。まずは各ユーザのホームディレクトリにアクセスしてみましょう。

ホームディレクトリを共有する

1章で解説しましたが、homesというセクションを設定することで、各ユーザのホームディレクトリを共有することができます。

RHEL系では、デフォルトでhomesセクションが定義されていますので、追加の設定は不要です。

Ubuntuの場合、デフォルトではhomesセクションが行頭の「;」でコメントアウトされていますので、**リスト1**のように該当部分のコメントを外したうえ、read only行の設定を修正してからsmbdを再起動します。

▶ ホームディレクトリのセキュリティ強化

デフォルトの設定では、「サーバ名¥ユーザ名」形式でホームディレクトリのパスを直接指定することにより、あるユーザのホームディレクトリにほかのユーザがアクセスできます。これを抑止したい場合は、homesセクションに次の設定を追加してください。

```
valid users = %S
```

RHEL系、Ubuntuともに、上記設定はsmb.conf中にコメントアウトされた形で記載されていますので、コメントを外しておくことをお勧めします。

注8）多くの属性情報が表示されますが、本記事での解説は割愛します。

▼図4 pdbeditコマンドによるユーザ情報の確認

```
# pdbedit -L
monyo:1000:
local1:1001:
```

▼リスト1 ホームディレクトリを有効にする設定

```
[homes]
   comment = Home Directories  ← コメントを外す
   browseable = no             ← コメントを外す

# By default, the home directories are exported read-only. Change the
# next parameter to 'no' if you want to be able to write to them.
   read only = no  ← コメントを外したうえで、yesをnoに変更する
```

Column SELinuxを有効にしている際の注意点

RHEL系でSELinuxを有効にしている場合、SELinuxの設定でホームディレクトリの共有が無効化されているため、次のようにしてホームディレクトリの共有を有効化します。

```
# setsebool -P samba_enable_home_dirs on
```

ホームディレクトリへのアクセス

設定が完了したら、さっそくWindowsクライアントからアクセスしてみましょう。確認用として、あらかじめホームディレクトリ直下に何かファイルを作成しておいてください。

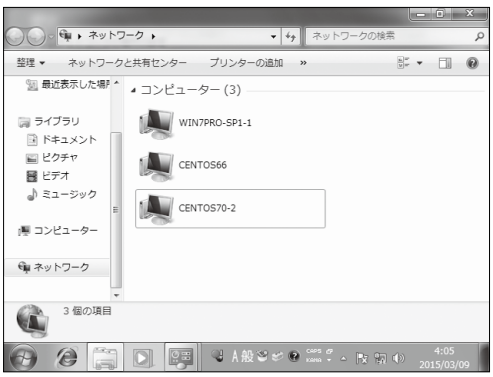
SambaはWindowsのネットワーク機能をほぼすべてサポートしているので、Windowsクライアントがサポートするさまざまな方法でのアクセスができます。あまり馴染みがないという方は、次のいずれかの方法でアクセスしてみてください。

・「ネットワーク」フォルダ経由のアクセス

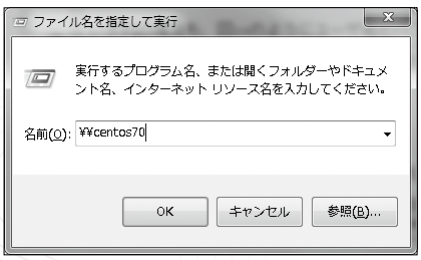
Windowsのスタートメニューから「ネットワーク」を選択すると、**図5**のようにファイルサーバとして機能しているコンピュータのアイコンが表示されます。

Sambaサーバのアイコンが表示されている場合は、そのアイコンをクリックしてください。

▼図5 「ネットワーク」フォルダ



▼図6 Sambaサーバ名の指定



ただし、適切な設定を行っていても、ネットワーク環境によってはアイコンが表示されない場合もあります。その際はもう1つの方法を使ってください。

・サーバ名(IPアドレス)を指定してのアクセス

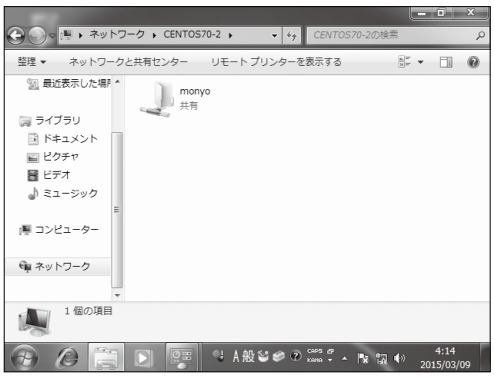
Windowsのスタートメニューの下部にある「プログラムとファイルの検索」欄、もしくは「ファイル名を指定して実行」メニューがある場合はクリックすると表示されるウインドウで**図6**のように「¥¥サーバ名」もしくは「¥¥IPアドレス」と入力します。

Windows 8の場合は、Win+Xキーなどで「ファイル名を指定して実行」メニューを呼び出すことで、同様の方法でのアクセスができます。

いずれの場合も、通常は**図7**のようにユーザ名とパスワードを確認する認証ダイアログが表

▼図7 認証ダイアログ

▼図8 共有一覧画面



示されますので、作成済みのSambaユーザのユーザ名とパスワードを入力してください。認証に成功すると、**図8**のように共有一覧の表示画面が表示され、ユーザ名(**図8**ではmonyo)のフォルダが表れます。

フォルダのアイコンをクリックすると、**図9**のようにSambaサーバであるLinux上のファイルやディレクトリ(フォルダ)を参照することができるはずです。Windowsの設定を変更して隠しファイルも表示する設定にすることで[注9]、.bashrcなどの設定ファイルも確認できます。

トラブルシューティング

ここまでの設定を適切に行っていれば、Sambaサーバへのアクセスは問題なくできるはずですが、設定を見直してもうまくいかない場合は、以降を参考にしてトラブルシューティングを行ってみてください。

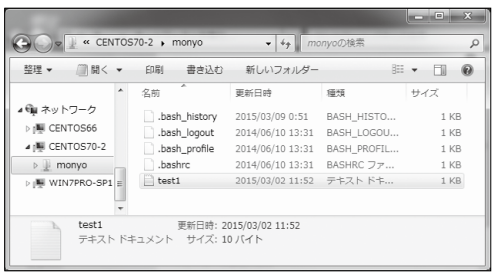
①ネットワークの接続性を確認する

WindowsクライアントからSambaサーバに対してpingコマンドを実行する、あるいはSamba以外のサービスにアクセスできるかを確認するといった方法で、Sambaサーバに対する通信が可能かどうかを確認してください。

もしくは、**図7**の画面が表示されるようであれば、少なくともSambaサーバに対する通信はできています。

注9) 「フォルダーオプション」の「表示」から「隠しファイル、隠しフォルダー、および隠しドライブを表示する」を選択します。

▼図9 ホームディレクトリの内部



②セキュリティ設定を確認する

アクセスできない原因のかなりのものが、ファイアウォールやSELinuxの設定といったセキュリティ関連の設定によるものです。一時的でかまいませんので、SELinuxやファイアウォールを無効にした状態でのアクセス可否を確認することをお勧めします。

同じくWindowsクライアントでパーソナルファイアウォールを動作させている場合は、一時的に無効にした状態でのアクセス可否を確認してみてください。

③Sambaの設定を確認する

Sambaの設定ミスを切り分けするため、簡単なsmb.confで設定を確認してみてください。**リスト2**のsmb.confをデフォルトのsmb.confと置き換えたうえで、Sambaを再起動してみてください。

上記の設定により、/tmpディレクトリがtmpという名称で共有され、Sambaユーザの設定にかかわらず誰でも読み取り専用でアクセスできるようになりますので[注10]、smb.confの設定ミスやパスワードの設定ミスに起因する問題を切り分けできます。

基本的なファイル共有の設定

前節でユーザのホームディレクトリの共有について解説しましたが、もちろんSambaではLinuxサーバ上の任意のファイルシステムを共有できます。

注10) SELinuxが有効な場合は、環境によっては動作しませんので、SELinuxは無効にした状態で確認するようにしてください。

▼リスト2 設定ミス切り分け用のsmb.conf

```
[global]
  map to guest = bad password
[tmp]
  path = /tmp
  guest ok = yes
```

以降では、グループ内のファイル共有を例に、ファイルサーバを構築するうえで最低限知っておいてほしいパラメータや設定について解説します。

ファイル共有の基本設定

▶基本的な設定

簡単なファイル共有の設定例を**リスト3**に示します。

セクション名(**リスト3**ではshare1)は共有名を示します。そのほか基本的なパラメータを説明します。

・path = /var/lib/samba/shares/share1

このパラメータの設定は、文字通り共有対象のパス名を示します。Sambaサーバ上に実在するパス名である必要があります。また、ファイルの書き込みを許可するためには、前述のパスのパーミッションを適切に設定する必要があります。動作確認という意味では、次のようにしてディレクトリを作成のうえ、誰でも書き込み可能にしておいてください。

```
# mkdir -p /var/lib/samba/shares/share1
# chmod 777 /var/lib/samba/shares/share1
```

▼リスト3　基本的なファイル共有の設定例

```
[share1]
  path = /var/lib/samba/shares/share1
  writeable = yes
```

・writeable = yes

作成した共有のデフォルトは読み取り専用です。書き込み可能とするためには、この設定が必要です。このパラメータはread onlyという反意の別名を持っているため、「read only = no」の設定を行っても等価です。

複数ユーザ間でファイルを共有する

リスト3の設定を行っただけでは、あるユーザが書き込んだファイルを別のユーザが更新できません。これは**図10**のようにLinux上で作成したファイルのパーミッションを確認すればわかりますが、パーミッション上ほかのユーザからの書き込みが許可されていないためです。

Column　SELinuxを有効にしている際の注意点

SELinuxを有効にしている環境では、次のようにpathパラメータで指定するパスに対して、samba_share_tというラベルを付与する必要があります。

```
# chcon -t samba_share_t /var/lib/samba/shares/share1
```

これにより、共有内に新規に作成されるファイルには同様のラベルが付与され、Sambaから参照可能となります。ただし、Linuxサーバ上で共有外のパスから(コピーではなく)移動されたファイルにはラベルが付与されないため、個別にラベルを付与する必要がある点に留意してください。SELinuxによってアクセスを拒否されたファイルは、Windowsからフォルダを参照しても表示されません。

また、/homeや/etcなどサーバ上の既存のディレクトリに対してこのコマンドを実行することは、既存のラベルが上書きされてしまい、システムの動作に影響を及ぼすことがあるため避けてください。SELinuxを有効にした状態で既存のディレクトリを適切に共有したい場合は、共有を読み取り専用(ro)にするか書き込み可能(rw)にするかに応じて、次のコマンドのいずれかを実行します。

```
# setsebool -P samba_export_all_ro on(読み取り専用)

# setsebool -P samba_export_all_rw on(読み書き可能)
```

これでは、ユーザmonyoしかファイルに書き込めません。あるユーザが書き込んだファイルを別のユーザが更新できるファイル共有を作成するには少しテクニックが必要です。設定方法はいくつかありますが、ここでは/var/lib/samba/shares/share2以下をshare2という名前で共有し、share2gグループに所属するユーザ間で互いに書き込み可能とする設定例を紹介します。

Sambaサーバ上で事前に**図11**の設定を行って共有するディレクトリを作成したうえで、smb.confで**リスト4**の設定を行います。

create maskとdirectorty maskパラメータにより、書き込まれたファイルのパーミッションが強制的に、グループに書き込み権がある664や775に設定されます。force groupパラメータによりファイルの所有グループが強制的にshare2gに設定されます。結果として、share2共有内のファイルの所有グループはすべてshare2gとなり、かつグループに書き込み権が設定されるため、複数ユーザ間で同じファイルを互いに書き込み可能な形で共有可能となっています。

必須ではありませんが、valid usersパラメータを設定にすることで、この共有にアクセス可能なユーザをshareg2グループ所属のユーザに限定するといったアクセス制御もできます。

上記設定を行った共有に対してWindowsクライアントからmonyoとlocal01というユーザでアクセスしてファイルを書き込んだ状態で、Linux上でパーミッションを確認した例を**図12**に示します。

ファイルのパーミッションが664、所有グループがshare2gになっていることが確認できます。これにより、ユーザlocal01がaaa.txtに書き込んだり、ユーザmonyoがbbb.txtに書き込んだりすることが可能となっています。

▼図10 Linux上でのファイルのパーミッション確認

```
[root@centos70-2 ~]# ls -l /var/lib/samba/shares/share1/
合計 24
-rwxr--r--. 1 monyo   monyo    9  3月  9 11:28 test1.txt
-rwxr--r--. 1 monyo   monyo    9  3月  9 11:28 test2.txt
```

▼図11 複数ユーザ間でファイルを書き込み共有する際の事前作業

```
# mkdir -p /var/lib/samba/shares/share2
# groupadd share2g
# chgrp share2g /var/lib/samba/shares/share2
# chmod g+w     /var/lib/samba/shares/share2
```

▼リスト4 複数ユーザ間でファイルを書き込み共有する設定例

```
[share2]
  path = /var/lib/samba/shares/share2
  writeable = yes
  create mask = 664
  directory mask = 775
  force group = share2g
  valid users = @share2g ← 共有へのアクセスをshare2gグループに限定する設定
```

▼図12 Linux上でのファイルのパーミッション確認

```
[root@centos70-2 ~]# ls -l /var/lib/samba/shares/share2/
合計 4
-rw-rw-r--. 1 monyo   share2g 26  3月 11 03:14 aaa.txt
-rw-rw-r--. 1 local01 share2g  6  3月 11 03:14 bbb.txt
```

まとめ

ここまでSambaのインストールから始まり、ユーザを作成して最低限の実用的なファイル共有の設定を行い、Windowsクライアントからアクセスするところまでを解説しました。

ここまで解説した設定をすべて含んだsmb.confファイルの例を**リスト5**に示します。任意の設定は斜体で示しました。また適宜コメントを入れています。

デフォルトのsmb.confファイルをもとに編集を行った場合は、これ以外にもいくつか設定が行われていますが、ここまで解説した動作に影響を与えるような設定はありませんので、そのまま残しておいてかまいません。

今回は誌面の関係もあり、ファイルサーバとしては、実用的な最低限の設定を紹介しましたが、Sambaではこれ以外にもACLによる詳細なアクセス制御や、ゴミ箱機能による削除済ファイルの復活、監査機能など多くの機能がサポートされています。

大半の機能はSambaのパラメータやVFSモジュールという拡張機能で実装されていますので、興味のある方は、ぜひSambaのマニュアルページやインターネット上の情報を探って設定してみてください。SD

▼リスト5　実用的なsmb.conf例

```
# Samba全体の設定
[global]
  ; 文字コード関連設定
  dos charset = CP932
  unix charset = UTF-8

  ; Sambaサーバのコンピュータ名をホスト名以外にする設定
  netbios name = SAMBASV

  ; Sambaが動作するインタフェースを制限する設定
  interfaces = lo0 eth0
  bind interfaces only = yes

  ; ログ出力の詳細度を制御する設定
  log level = 0

# ホームディレクトリをファイル共有する設定
[homes]
  comment = Home Directories
  browseable = no
  read only = no
  valid users = %S

# 簡単なファイル共有の設定例
[share1]
  path = /var/lib/samba/shares/share1
  writeable = yes

# 複数ユーザに対応したファイル共有の設定例
[share2]
  path = /var/lib/samba/shares/share2
  writeable = yes
  create mask = 664
  directory mask = 775
  force group = share2g
  valid users = @share2g
```

第3章 Active Directoryとの認証連携

Author たかはしもとのぶ mail monyo@monyo.com Twitter @damemonyo

本章では、応用編としてActive Directoryへの認証連携について解説します。

Active Directoryへの認証統合

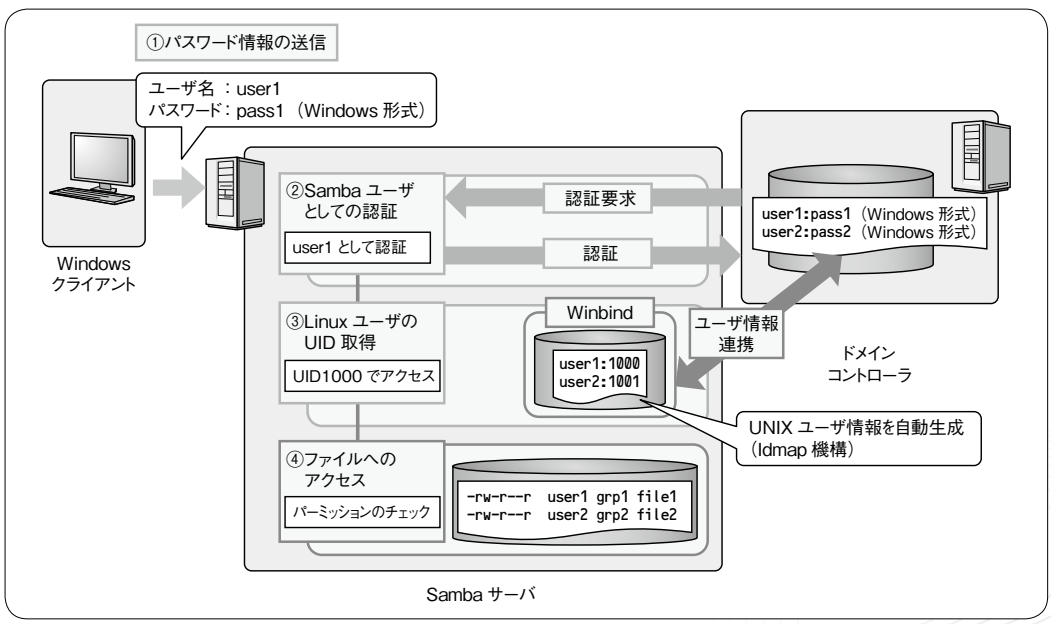
最近の企業ネットワークでは、Active Directory(以降、AD)によりWindowsサーバにアクセスする際の認証については一元管理されていることが多いでしょう。Winbind機構を使用することで、Sambaサーバの認証もADに統合することができます。

Winbind機構とは

Winbind機構とは、ADと連携して動作し、ADの認証情報をSambaやPAM(Pluggable Authentication Module)、NSS(ネームサービススイッチ機能)経由で動作する一般のプログラムから利用可能にするしくみです。これを実現するために、Sambaのデーモンであるsmbdやnmbdに加えてwinbinddというデーモンが起動されます。

2章の図1(P.77)に対比させて、Winbind機構が有効な環境で、SambaユーザとLinuxユーザが、実際にWindowsクライアントからSambaサーバにアクセスする際にどのように動作するのかを**図1**に示しました。

▼図1 Winbindの動作概念



①まずは、Windowsユーザ名と、ユーザが入力したパスワードをWindows形式でハッシュ化したパスワード文字列の情報がSambaサーバに送付されます。
②Sambaはこの情報をADのドメインコントローラ(DC)に転送します。DCは認証を行い、結果をSambaサーバに返却します。
③続いて、Winbind機構は自身の管理下にあるLinuxユーザを検索します。Winbind機構が生成済のLinuxユーザについてはその情報が返却されます。未生成の場合、Winbind機構がLinuxユーザを生成し、その情報が返却されます。
④最終的に生成されたLinuxユーザのUID情報を使って、Sambaサーバ上の各ファイルにアクセスします。ファイルに適切なパーミッションなどが付与されてない場合、アクセスは拒否されます。

Winbind機構の実体であるwinbinddが、Linuxユーザを自動生成するため、Sambaサーバ上では個別のLinuxユーザを管理する必要がなくなります。また、sshやftpといったほかのプログラムも、PAM、NSS経由でこの認証機能を用いることができます。

以降の節では、Winbind機構のインストールと設定について解説します。なお誌面の制限もあり、ADの用語や概念についての説明は割愛しています。

Winbind機構のインストールとADドメインへの参加

CentOSの場合、Winbindはsamba-winbindとsamba-winbind-clientsというパッケージから構成されています。samba-winbind-clientsには、後述する**wbinfo**コマンドなど、Winbind機構の管理に必須のコマンドが含まれていますので、次のように両方のパッケージをインストールします。

```
# yum install samba-winbind ↵
samba-winbind-clients
```

Ubuntuの場合は、winbindパッケージをインストールしてください。

以降では、centos70-2という名前のCentOS 7.0が動作しているSambaサーバを、IPアドレスが192.168.135.20のDCが存在するaddom1.localというFQDN名のADに参加させる場合を例に設定を行っていきます。

▶①静的IPアドレスの設定

ADに参加させるためにはDNSサーバとしてADのDCを指定する必要があります[注1]。またあらかじめ、AD参加後のFQDNが名前解決可能な状態に構成しておく必要があります。このため、Linuxサーバでは静的IPアドレスの設定が必要です。CentOS 7.0で静的IPアドレスを設定する手順を**図2**に示します。

▼図2　静的IPアドレスの設定手順

```
[root@centos70-2 ~]# nmcli d
DEVICE  TYPE      STATE      CONNECTION
ens33   ethernet  connected  Wired connection 1
lo      loopback  unmanaged  --
[root@centos70-2 ~]# nmcli c modify "Wired connection 1"  ipv4.addresses ↵
"192.168.135.26/24 192.168.135.2"  ← 192.168.135.26というIPアドレスを設定し、デフォルトゲートウェイを192.168.135.2に設定
[root@centos70-2 ~]# nmcli c modify "Wired connection 1"  ipv4.dns 192.168.135.20 ↵
ipv4.dns-search addom1.local  ← DNSサーバを192.168.135.20に、検索サフィックスをaddom1.localに設定
[root@centos70-2 ~]# nmcli c modify "Wired connection 1"  ipv4.method manual
                                                          ↑ 静的IPアドレスを指定
[root@centos70-2 ~]# nmcli c down "Wired connection 1"
[root@centos70-2 ~]# nmcli c up   "Wired connection 1"   ← 設定の反映のためにインターフェースを再起動する
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManager/
ActiveConnection/9)
```

注1）AD側で特殊な構成を取っている場合はこの限りではありません。

さらにAD参加後に自身の名前解決ができるよう、/etc/hostsファイルに次のような行を追加しておきます。

```
192.168.135.26 centos70-2.addom1.local centos70-2
```

▶②時刻の同期

ADの認証方式であるKerberos認証が機能するうえでは、DCとの時刻のずれが常時5分以内である必要があります。そのため、DCと常時時刻同期を行うように設定しておきましょう。NTPを用いてNTPサーバでもあるDCと時刻同期するのが一般的ですが、次のようにSambaの一部である**net**コマンドを使用して時刻を同期することもできます。

```
# net time set -S 192.168.135.20
```

▶③Sambaサーバの停止

ADへの参加を行う際には、Sambaサーバが停止している必要があります。1章で解説した手順に従ってSambaサーバを停止させます。

▶④smb.confの設定

smb.conf に**リスト1**のような設定を行います。realmパラメータの値は、必ず大文字で指定します。

CentOSの場合、次のコマンドを実行することでこの設定を行うこともできます。

```
# authconfig --smbworkgroup=ADDOM1 --smbrealm=ADDOM1.LOCAL --update
```

authconfigコマンドでsmb.confの設定を行うと思わぬ結果を招く可能性がありますので、意図したように変更されているか、必ず確認するようにしてください。

▼リスト1　smb.confの設定例

```
[global]
  workgroup = ADDOM1      ← 短いADドメイン名
  realm = ADDOM1.LOCAL    ← ADのFQDN名（大文字）
  security = ads
```

手作業で設定を行う場合、上記のパラメータはいずれもデフォルトのsmb.confでコメントアウトされた形で記述されていますので、該当個所のコメントを外して設定すればよいでしょう。Ubuntuの場合はworkgroupパラメータのみ設定されていますので、その近辺にまとめて**リスト1**の設定を行えばよいでしょう。

▶⑤ADへの参加

ここまでの準備が整ったら、次のように「**net ads join**」コマンドを実行してADドメインへの参加を行います。

```
[root@centos70-2 ~]# net ads join -U Administrator
Enter Administrator's password:
  ↑ パスワードを入力
Using short domain name -- ADDOM1
Joined 'CENTOS70-2' to dns domain 'ADDOM1.LOCAL'
```

「**net ads join**」コマンドの「**-U**」オプションでAdministratorなど、ADにコンピュータを追加する権限を持ったユーザを指定します。指定したユーザのパスワードを入力することでADドメインへの参加が成功します。

これにより、**図3**のようにComputersコンテナにコンピュータアカウントが生成されます[注2]。

引き続き、Winbind機構の設定を行っていきます。

▶⑥NSSの設定

/etc/nsswitch.confファイルのpasswd、group行について、**リスト2**のように「winbind」というキーワードを追加します。

CentOSの場合は、次のコマンドを実行する

注2）これ以外の場所に作成したい場合は、createcomputerオプションを指定して「net ads join」コマンドを実行します。

▼リスト2　/etc/nsswitch.confの修正例

```
passwd:         files winbind
group:          files winbind
※winbindというキーワードを追加
```

ことで上記設定を行うこともできます。

```
# authconfig --enablewinbind --update
```

▶⑦smb.confの設定

先ほどの**リスト1**の設定に加え、globalセクションに次のような設定を追加します。

```
idmap config * : range = 10000-19999
```

このパラメータは、Winbind機構が生成したLinuxユーザに割り当てるUIDとGIDの範囲を指定します。**authconfig**コマンドを使って**リスト1**の設定を行った場合、上記パラメータはすでに設定されています。UID、GIDの範囲を変更したい場合は、次のようにしてコマンドを実行してください。

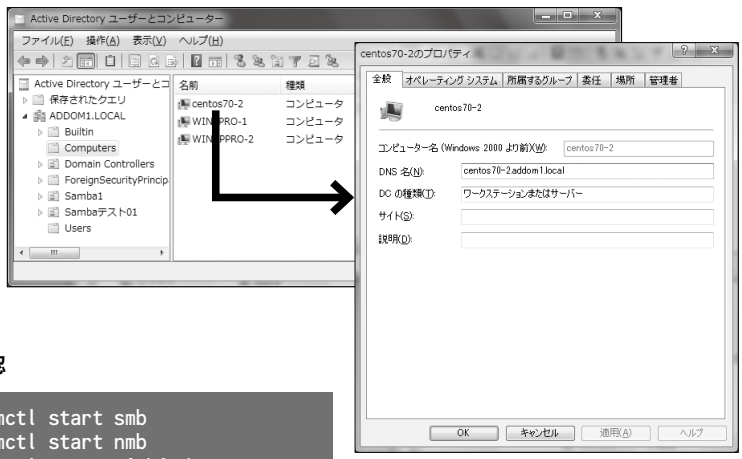
```
# authconfig --smbidmaprange=20000-29999 --update
```

▶⑧Sambaサーバ、Winbind機構の起動と動作確認

ここまで設定を行ったら、Sambaのプロセスを起動します。smbd、nmbdに続き、winbinddについても忘れずに起動してください。起動したら**wbinfo**コマンドなどを使って動作確認を行いましょう。CentOS 7.0での実行例を**図4**に示します。

UIDやGIDの値としては、パラメータで設定した範囲の値が先頭から順に払い出されます。

▼図3　Active Directoryユーザーとコンピューター上からの確認



▼図4　Winbind機構の動作確認

```
[root@centos70-2 ~]# systemctl start smb
[root@centos70-2 ~]# systemctl start nmb
[root@centos70-2 ~]# systemctl start winbind
[root@centos70-2 ~]# wbinfo -t ← Winbind機構とADとの通信が行われているか
checking the trust secret for domain ADDOM1 via RPC calls succeeded
[root@centos70-2 ~]# wbinfo -u ← ユーザの一覧を列挙する
CENTOS70-2¥monyo
CENTOS70-2¥root
ADDOM1¥administrator
ADDOM1¥guest
ADDOM1¥support_388945a0
(...中略...)
ADDOM1¥samba03
[root@centos70-2 ~]# id addom1¥¥administrator ← ユーザのUID、GID情報を表示する
uid=10001(ADDOM1¥administrator) gid=10000(ADDOM1¥domain users) ➔
groups=10000(ADDOM1¥domain users),10001(ADDOM1¥schema admins),10002(ADDOM1¥enterprise ➔
admins),10003(ADDOM1¥group policy creator owners),10004(ADDOM1¥domain admins)
[root@centos70-2 ~]# wbinfo -g ← グループの一覧を列挙する
ADDOM1¥helpservicesgroup
ADDOM1¥telnetclients
ADDOM1¥domain computers
(...中略...)
```

今回の例では10000から順に払い出されます。

▶⑨Windowsクライアントからの動作確認

最後にWindowsクライアントから実際にアクセスしてみましょう。まずはADDOM1.LOCALドメインに参加しているWindowsクライアントにADの一般ユーザとしてログオンします。ここではsamba01というユーザとしてログオンしたものとします。

ログオン後に「¥¥centos70-2」のように入力してSambaサーバにアクセスしてみると、サーバの/etc/passwdにはユーザsamba01の定義がないにもかかわらずアクセスが成功します。ここまでの設定を順に行っている場合は**図5**のようにホームディレクトリとshare1、share2という共有が参照できます。

share1共有に何かファイルを作成してからSambaサーバで参照すると、**図6**のようにWinbind機構が生成したユーザとしてアクセスしていることが確認できます。

Winbind環境下での注意点

2章までに解説した内容について、Winbindを有効にした環境特有の注意点を紹介します。

▼図5 Windowsクライアントからのアクセス

▼図6 Sambaサーバ上でshare1共有を参照したところ[注3]

```
[root@centos70-2 ~]# ls -l  /var/lib/samba/shares/share1/
合計 28
-rwxr--r--. 1 ADDOM1¥samba01 ADDOM1¥domain users  9  3月  9 11:28 テスト.txt
```

▶ADドメインのユーザ、グループによる設定

P.84の**リスト5**にある、

```
valid users = @share2g
```

の設定について、share2gの代わりにADDOM1¥Domain Usersグループを設定するケースを考えてみます。この場合、次のようにして指定を行う必要があります。

```
valid usrs = @"ADDOM1\Domain Users"
```

空白文字を含んでいる場合、両端を「"」で囲む必要があります。P.84の**リスト5**のforce groupや、その他のパラメータでも同様です。

▶ホームディレクトリの自動作成と共有

本記事の内容に沿って順番に設定を行った環境では、**図5**でホームディレクトリを示すアイコンをクリックしても、ユーザのホームディレクトリにアクセスできません。これは、ホームディレクトリのパスが/home/ADDOM1/samba01という実在しないパスになっているためです。

自動生成したユーザごとに手作業でホームディレクトリを作成してもよいのですが、それでは自動生成の意味がなくなってしまいます。

ホームディレクトリの自動作成を実施する際には、**リスト3**のようなスクリプトを作成し[注4]、**図7**のように、共有へのアクセス時にroot権限で自動実行される「**root preexec**」パラメータを設定するのがよいでしょう[注5]。スクリプトのパ

注3) 日本語ファイル名を適切に表示させるためには、dos charset/unix charsetパラメータの設定と仮想端末ソフトウェアの文字コード関連の設定を適切に行う必要があります。

注4) 後述する「template homedir」パラメータなどでホームディレクトリのパスを変更している場合、このスクリプトもそれに応じて変更する必要があります。

注5) この他PAMの機能を使用する方法もあり、以前のバージョンでは動作していたのですが、本記事執筆にあたり検証したところアクセス権の問題でどうやっても動作させることができなかったため、今回は紹介を見送りました。

スは任意ですが、ここでは/var/lib/samba/scriptsというパスにsmb_mkhomedirという名前で置いています。

UIDとGIDのカスタマイズ(Idmap機構)

Winbind機構により、ADドメインのユーザやグループ(以降、単にユーザと記述)に対応するLinuxユーザが自動生成され、その際にUIDやGID(以降、単にGIDと記述)も自動的に付与されます。

Linuxユーザが自動生成されるタイミングはSambaサーバごとにバラバラですので、あるユーザに対して付与されるUIDも一定しません[注6]。つまり、複数のSambaサーバが存在している環境では、同じADドメインのユーザに対して異なるUIDが割り当てられる可能性があります。

これは、NFSのようにUIDに依存しているプロトコルにとっては致命的な問題点です。

UIDの付与を管理するIdmap機構をデフォルトから変更することで、この問題を解決できます。Idmap機構はいくつかありますが、ここでは単一ドメイン環境前提ですが、設定が簡単なridという方式を紹介します。

▼リスト3　ホームディレクトリを作成するスクリプト例

```
#!/bin/sh

# ドメイン名のディレクトリがなければ作成
if ! [ -d /home/${2} ]; then
  mkdir -p /home/${2}
  chmod 755 /home/${2}
fi

# ホームディレクトリがなければ作成
if ! [ -d /home/${2}/${1} ]; then
  cp -pr /etc/skel /home/${2}/${1}
  chown -R ${2}¥¥${1} /home/${2}/${1}
  chmod 700 /home/${2}/${1}
fi
```

ridの動作原理

ridというIdmap機構は、ADドメインのユーザのSIDに含まれるRIDから計算された値を使用してUIDを付与します。SIDとはUIDやGIDのように各種オブジェクトを一意に識別する値で、たとえば、

```
S-1-5-21-1234995458-293493368-1744720997-513
```

注6) Linuxユーザを作成する時点で、使用可能なUIDの中から最も値の小さいものが付与されます。

▼図7　root preexecパラメータの設定(CentOS)

```
;   valid users = %S
;   valid users = MYDOMAIN¥%S
    root preexec = /var/lib/samba/scripts/smb_mkhomedir %U %D ← この行を追加

[printers]
    ⋮
```

Column　SELinux環境での注意点

CentOSのSELinux環境の場合、Sambaから起動するスクリプトは、必ず/var/lib/samba/scriptsというディレクトリ内に配置する必要があります。またこのディレクトリを作成した際には、必ず次のコマンドを実行して適切なラベルを付与してください。

```
# restorecon -R -v /var/lib/samba/scripts
```

ただし、この設定を行っても、スクリプトが/homeディレクトリ内に書き込むことができないため、リスト3のスクリプトは動作しません。その他の方法も試しましたが、筆者が試した限りではSELinuxを有効にした環境でホームディレクトリ自動作成の設定を手軽に行うのは困難そうです。

のような長い値となっています。この最後の「513」の部分が各オブジェクト(ユーザやグループなど)のインスタンスを一意に識別する値でRIDと呼ばれます。ridでは、この値に一定の数値を加算した数字をUIDとして使用します。

▶ ridの設定例

設定例を**リスト4**に示します。

実は「`idmap config * : ……`」という設定は、デフォルトのIdmap機構の設定を意味します。この設定はSambaが内部的に使用するユーザのUID付与のために必要ですので、削除してはいけません。

ADDOM1ドメインに対するIdmap機構の設定として、**リスト4**の下2行を追加します。ADDOM1という部分は実際のドメイン名にあわせて適宜変更してください。また「`range =`」に続く数字は付与するUIDの範囲を決めるものですので、/etc/passwdで定義済のユーザや、既存のrangeと重複しない範囲であれば自由に設定できます。

ここではADDOM1ドメインに対するrangeとして指定された最小値が20000のため、ridが0のユーザのUIDは20000、1なら20001、2なら20002といった具合に機械的に算出されたUIDが付与されます。実行例を**図8**に示します。

上記から逆算すると、ユーザADDOM1¥administratorのRIDは500であることがわかります。

このほか、よく使われるIdmap機構としてはActive Directoryに格納されたUNIX属性の値を参照するadなどがあります。興味のある方はコラムを参考に、ぜひ設定に挑戦してみてください。

▼リスト4 ridの設定例

```
[global]
  idmap config * : range = 10000-19999      ← デフォルトのIdmap設定(設定済)

  :

  idmap config ADDOM1:backend = rid         ← ADDOM1ドメインで使用するIdmap機構
  idmap config ADDOM1:range = 20000-29999   ← 割り当てるUID、GID値の範囲
```

▼図8 実行例

```
# id addom1¥¥administrator
uid=20500(ADDOM1¥administrator) gid=20513(ADDOM1¥domain users) groups=20513(ADDOM1¥domain ➥
users),20518(ADDOM1¥schema admins),20519(ADDOM1¥enterprise admins),20520(ADDOM1¥group ➥
policy creator owners),20512(ADDOM1¥domain admins),10007(BUILTIN¥users),➥
10006(BUILTIN¥administrators)
```

Column Winbindによる対応付け情報の削除

運用中にIdmap機構を変更した場合でも既存のIdmap機構のデータベースは消去されません。そのため、データベースに格納された古いIdmap機構で付与済のUIDは保持され[注7]、変更後に付与されるUIDから、新しいIdmap機構による付与が開始されます。

何らかの理由で古いIdmap機構で付与されたUID情報を削除したい場合は、「net cache flush」コマンドを使用します。

ただし、これを行うと、既存ユーザのUIDやGIDが変更になってしまいます。ファイルやディレクトリにパーミッションを付与している場合はそれらもすべて変更する必要があるので、UID情報の削除は慎重に行ってください。

注7) ridのようにUIDやGIDの情報を計算で生成するIdmap機構についても、付与済みのUIDやGIDについてはデータベースに格納されるため、Idmap機構の変更後も保持されます。

PAMによるSamba以外のプロダクトの認証統合

pam_winbindモジュールを使用することで、Winbind機構が提供する認証機能をPAM経由で使用することができます。これにより、ssh/telnet/ftpといったPAMに対応した一般的なプロダクトの認証を、Winbind機構が提供するADユーザの情報を使って行うことが可能となり、Sambaが動作するLinuxサーバに対する認証が、完全にActive Directoryに統合されます。

以降、sshによるログインをADユーザで行う設定を例に解説します。

▶ PAMの設定

まずはpam_winbindモジュールを有効にします。CentOSの場合は次のコマンドで有効化ができます。

```
# authconfig --enablewinbindauth --update
```

▶ シェルの変更

デフォルト設定の場合、Winbind機構により自動生成されるLinuxユーザのユーザ情報、グループ情報は**図9**のようになっています[注8]。

デフォルトでは、シェルとして/bin/falseという無効なものが指定され、ホームディレクトリは「/ドメイン名/ユーザ名」という形式になっています。

これらのデフォルト値を変更するために「template shell」や「template homedir」というパラメータが用意されています。たとえば、**リスト5**の設定をglobalセクションに行うことでシェルがbashに、ホームディレクトリのパスがドメイン名を含まないものに変更されます。

CentOSでは**図10**のようにコマンドで設定することもできます。

これにより、sshなどでログインする際にADユーザのユーザ名とパスワードで認証する仕様にできます。次のようにしてpam_mkhomedirモジュールを有効にしている場合は、ホームディレクトリの自動作成も行われます。

```
# authconfig --enablemkhomedir --update
```

図11にsshでログインした際の実行例を示します。

まとめ

本章ではActive Directoryとの認証連携につ

注8) Sambaのデフォルトでは、大規模環境におけるパフォーマンス上の理由で、単に「getent passwd」や「getent group」コマンドを実行した際にはWinbindが生成したユーザやグループが一覧表示されない設定になっています。表示させたい場合は「winbind enum users」および「winbind enum groups」パラメータをそれぞれyesにしてください。

▼リスト5　シェルとホームディレクトリの変更例

```
template shell = /bin/bash
template homedir = /home/%U
```

▼図9　Linuxユーザのユーザ情報

```
# getent passwd addom1¥¥administrator
ADDOM1¥administrator:*:10001:10000:Administrator:/home/ADDOM1/administrator:/bin/false
```

▼図10　CentOSでの設定

```
# authconfig --winbindtemplatehomedir=/home/%U --winbindtemplateshell=/bin/bash --update
```

▼図11　sshログインの認証とpam_mkhomedirモジュールによるホームディレクトリの自動作成

```
$ ssh centos70-2 -l 'ADDOM1\samba01'
ADDOM1\samba01@centos70-2's password:
Creating directory '/home/ADDOM1/samba01'.
Last login: Fri Mar 13 02:51:09 2015 from 192.168.135.16
```

いて解説しました。

小規模な環境であっても、Active Directoryが存在する環境では認証統合の要望は比較的強いと思いますので、ぜひ試してください。

本章で解説した設定を含んだsmb.confの例を**リスト6**に示します。適宜コメントを入れています。これらの設定がSambaサーバ設定の参考になれば幸いです。SD

▼リスト6　Active Directoryとの認証統合の設定例

```
# Samba全体の設定
[global]
  ; Active Directoryドメインへの参加設定
        workgroup = ADDOM1
        realm = ADDOM1.LOCAL
        security = ADS
  ; Winbind機構が自動生成するユーザのシェルとホームディレクトリ設定
        template shell = /bin/bash
        template homedir = /home/%D/%U
  ; デフォルトのIdmap機構の設定
        idmap config * : range = 10000-19999
  ; ADDOM1ドメインのIdmap機構をridにする設定
        idmap config ADDOM1:range = 20000-29999
        idmap config ADDOM1:backend = rid
```

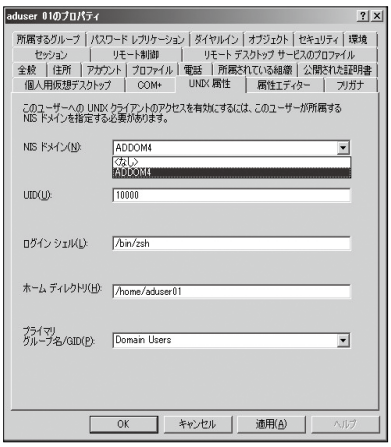
Column　Linuxユーザの情報をユーザごとに設定する

ユーザごとにシェルやホームディレクトリを指定したい場合は、あらかじめADに値を格納したうえで、**リスト7**の設定をglobalセクションに行います。ADへの値の格納には、**図12**の「UNIX属性」タブを使うのが簡便です。このタブはWindows Server 2008 R2以前に存在する「NISサーバ」機能をインストールし、「NISサーバ」サービスを起動することで使用できます[注9]。

残念ながらWindows Server 2012以降ではこの機能が廃止されたため、値を格納するには、属性エディタなどで各ユーザのuidNumber、gidNumber、loginShell、unixHomeDirectory属性を直接編集する必要があります。UIDやGIDの値もADに格納された値を参照させるには、adというIdmap機構を使用する必要があります。設定例を**リスト8**に示します。UNIX属性を**図12**のように指定したユーザの情報を参照した際の実行例を**図13**に示します。

注9）一度「UNIX属性」タブが使用可能になったら、「NISサーバ」サービスは停止してしまってかまいません。

▼図12　「UNIX属性」タブ



▼リスト7　ADに格納した値を参照する設定

```
winbind nss info = rfc2307
```

▼リスト8　adの設定例

```
[global]
  idmap config ADDOM1:backend = ad  ← ADDOM1ドメインで使用するIdmap機構
  idmap config ADDOM1:range = 20000-29999 ← 割り当てるUID、GID値の範囲
  idmap config ADDOM1:schema_mode = rfc2307
```

▼図13　Linuxユーザのユーザ情報

```
# getent passwd ADDOM1\\aduser01
ADDOM1\aduser01:*:10000:10000:aduser 01:/home/aduser01:/bin/zsh
```

短期集中連載

プログラマに優しい現実指向JVM言語

Kotlin入門

第2回 開発環境の構築

先月号の第1回では導入としてKotlinの概要や特徴について解説しました。今回はKotlinで実際に開発を始めるための環境を整える方法を紹介します。

Author 長澤 太郎(ながさわ たろう) Twitter @ngsw_taro Mail taro.nagasawa@gmail.com

開発環境を準備する

前回の記事でKotlinの魅力を知っていただけたでしょうか。それではコードを実際に書いて、遊んでみませんか。今回は、Kotlinを実行する開発環境として次の方法を紹介します。読者の皆さんの用途に合わせて利用してください。

・Webブラウザで利用できるエディタ
・シンプルなCLIコンパイラ
・IntelliJ IDEA

Webブラウザで利用できるエディタ

先に挙げた中でもっとも手軽で簡単な方法がWebブラウザ上で開発する方法です。JetBrainsが提供するKotlin Web Demoを利用します。Webブラウザから次のURLにアクセスしてください。

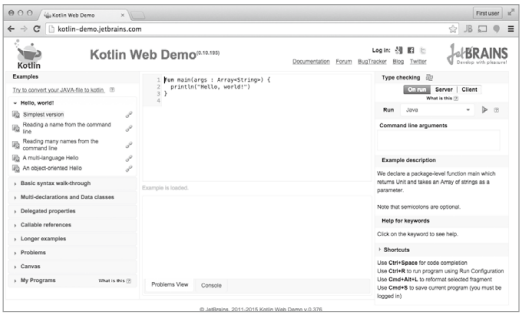
```
http://kotlin-demo.jetbrains.com/
```

アクセスすると**図1**のような画面が表示されます。

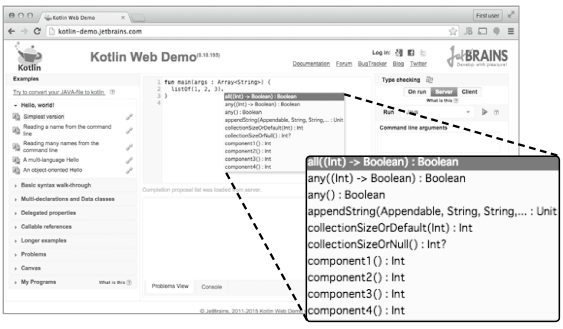
中央にコードを編集するエリア(=エディタ)があり、最初のアクセス時にはHelloWorldコードが入力された状態になっています。ここに入力されているコードを実行するには画面右側にある再生マークの実行ボタンをクリックします。コードが実行されると、その結果がコンソールに出力されます。

エディタはシンタックスハイライト機能はもちろんのこと、コードの補完機能まで備えています。コードの補完機能を有効にするために画面右側の[Type checking]のところで[Server]を選択します(**図2**)。これでコードの補完機能が有効になりました。試しにエディタに`listOf(1, 2, 3).`と入力します。最後の`.`を忘れないでください。ここで Ctrl キーと Space キーを同時に押してください。するとアクセス可能なメンバの一覧が表示されます。

▼図1 Kotlin Web Demo



▼図2 コードの補完



シンプルなCLIコンパイラ

手持ちのマシンの上でコマンドからコンパイルを実行する方法です。皆さんが現在使用されている任意のエディタでコーディングできます。

コンパイラを入手するには次のURLよりKotlin公式サイトへアクセスしてください。

```
http://kotlinlang.org/#get-kotlin
```

公式サイトへアクセスしたら[Download compiler]をクリックしてください。Githubのページが開くので「kotlin-compiler-0.10.195.zip[注1]」をクリックしてZIPファイルをダウンロードします。

ダウンロードしたZIPファイルを展開すると、kotlincファイル[注2]としてKotlinソースファイルをコンパイルするためのスクリプトが得られます。皆さんが使用されているPC環境に合わせて、ここへのパスを通しましょう。

コンパイラの準備は以上です。次にコンパイラを使用して、Kotlinソースファイルからclassファイルへコンパイルしてみましょう。

リスト1の内容を記述したファイルをHelloWorld.ktとして任意の場所に保存します。拡張子ktはKotlinのソースファイルのための拡張子です。

ターミナルを起動し、カレントディレクトリをHelloWorld.ktが保存されているディレクトリに変更します。そして次のようにコマンドを叩きます。

```
$ kotlinc HelloWorld.kt
$
```

行頭の$は便宜上、プロンプトを意味する記号とします。メッセージが出力されることなく次のプロンプトが表示されればコンパイル成功です。lsコマンドなどで同ディレクトリ内にclassファイルが作成されているのが確認できます。

コンパイルで得られたclassファイルを実行するにはjavaコマンドを使用します。ただし実行に際して、Kotlinランタイム(kotlin-runtime.jar)が必要です。また、実行対象のclassファイルは_DefaultPackage.classです[注3]。次のようにクラスパスを指定してjavaコマンドを実行してください[注4]。Hello, world!と画面に表示されれば成功です。

```
$java -cp $KOTLIN_HOME/lib/*:./ _
DefaultPackage
Hello, world!
$
```

▼リスト1　デフォルトパッケージのHelloWorld

```
fun main(args: Array<String>) {
  println("Hello, world!")
}
```

IntelliJ IDEA

最後に紹介するのはIDE(統合開発環境)であるIntelliJ IDEAを使う方法です。このIDEをダウンロードするには次のURLへアクセスしてください。

```
https://www.jetbrains.com/idea/download/
```

無償版であるCommunity Editionをダウンロードします。ダウンロードが完了したらお使いの環境に合わせてインストールしてください。

IntelliJ IDEAを起動すると**図3**のような画面が表示されます。

まずはKotlinプラグインを導入しましょう。画面右下の[Configure]→[Plugins]からプラグインの設定画面を開きます。

プラグイン設定画面を開いたら下部にある[Install JetBrains plugin...]ボタンをクリックします。インストール可能なプラグイン一覧が表示されるので、検索用テキストフィールドに「kotlin」

注1) 数字部は執筆時現在のバージョンを表しています。
注2) Windowsの場合は同ディレクトリ内のkotlinc.batファイルを使います。
注3) $KOTLIN_HOMEはZIPファイルを展開して得られたkotlincディレクトリのパスです。
注4) 今回はパッケージを定義しなかったのでこのような名前が付きました。

▼図3　IntelliJ IDEAの最初の起動

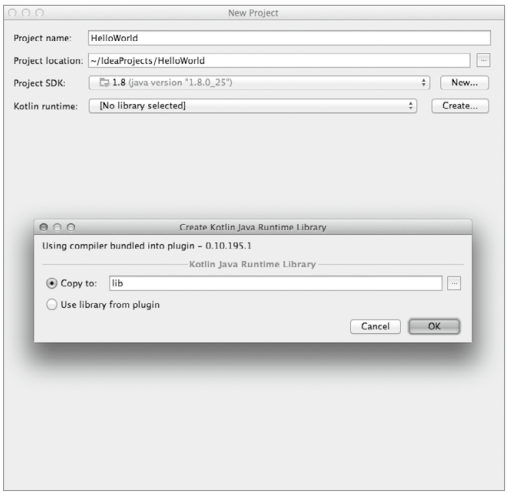
と入力してプラグインを絞り込みます。Kotlinプラグインを見つけたら右側の[Install plugin]ボタンを押してプラグインをインストールします。プラグインのインストールが完了するとIntelliJ IDEAの再起動を促されるので再起動します。そうすると、ふたたび**図3**のような画面が表示されます。

次はKotlinプロジェクトを作成して開発を始めましょう。[Create New Project]をクリックしてプロジェクト新規作成ウィザードを開きます。このウィザードの質問に適切に答えていき、プロジェクトを作成します。

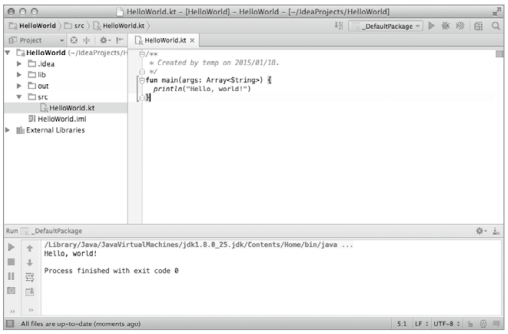
途中、Kotlinランタイムの設定をする**図4**のような画面があります。[Kotlin runtime]の[Create...]をクリックし、表示されたダイアログをそのまま[OK]をクリックするようにしてください。

プロジェクトを作成し、ウィザードが消えてエディタが表示されたら、Koltinソースファイルを追加します。srcディレクトリを選択した状態で、右クリックをし、Kotlinファイルを選んでソースファイルを作成します。ファイル名を聞かれるので「HelloWorld」としておきましょう。ソースファイルを作成するとすぐさまエディタ上に開かれるので**リスト1**の内容を記述します。コンパイル&実行するにはメニューバーから[Build]→[Run...]を選びます。すると実行対象を選択するダイアログが表示されるので[_DefaultPackage]を選んでクリックします。うまくいけばコンパイルが実

▼図4　Kotlinランタイムの設定



▼図5　実行結果



行され**図5**のように出力結果が表示されます。

まとめ

今回はKotlinの3つの開発環境とその導入方法、使い方を紹介しました。

Webブラウザ上で手軽に利用可能なKotlin Web Demoはちょっとした実験コードを試すにはうってつけです。基本的にはIntelliJ IDEAを使用してKotlinプログラミングを進めていくことになると思います。こだわりのエディタがある人には、シンプルなCLIコンパイラがおすすめです。

次回はKotlinの基本的な文法／機能を解説します。Kotlinの興味深い機能である、クロージャやNULL安全、拡張関数などを学習するための助走が狙いです。SD

バックナンバーのお知らせ　BACKNUMBER

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

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

2015年4月号

第1特集
トラブルシューティングの極意
達人に訊く問題解決のヒント

第2特集
［最新］DNSの教科書
ネットワークを支える本物のインフラを学ぶ

特別付録
・3分間ネットワーク基礎講座［特別篇］

定価（本体1,300円＋税）

2015年3月号

第1特集
カンファレンスネットワークの作り方

第2特集
いまからでも遅くない！
Hadoop超²入門

一般記事
・Cisco VIRLでネットワークシミュレーション［前編］
・Snappy Ubuntu Core

定価（本体1,220円＋税）

2015年2月号

第1特集
Linux systemd入門
あなたの知らない実践技

第2特集
そろそろ、やめませんか？
なぜ「運用でカバー」がダメなのか？

一般記事
・Intel DPDK技術詳解
・これはなんて読む？　UNIX用語読み方指南
・Googleベンチャーズが提唱するデザインスプリントとはほか

定価（本体1,220円＋税）

2015年1月号

第1特集
プログラマ・インフラエンジニア・文章書きの心得
Vim使い事始め

第2特集
SI崩壊を乗りきる3つの方法
ソフトウェア開発の未来

一般記事
・「ひみつのLinux通信」年末年始スペシャル
　ITエンジニア出世双六
・Jamesのセキュリティレッスン［最終回］

定価（本体1,220円＋税）

2014年12月号

第1特集
急速に普及するコンテナ型仮想環境
Dockerを導入する理由

第2特集
基礎の基礎から押さえる必須技術
やさしくわかるVPNの教科書

一般記事
・bashの脆弱性“Shellshock”その影響と対策
・SoftLayerを使ってみませんか？［最終回］
・Jamesのセキュリティレッスン［2］

定価（本体1,220円＋税）

2014年11月号

第1特集
Docker・Ansible・シェルスクリプト
無理なくはじめるInfrastructure as Code

第2特集
オンプレミスもクラウドも縦横無尽
サーバの目利きになる方法［後編］

一般記事
・8086時代から今を俯瞰する　CPU温故知新
・はてな謹製、サーバ管理ツール　Mackerel入門

定価（本体1,220円＋税）

Software Design　バックナンバー常備取り扱い店							
北海道	札幌市中央区	MARUZEN＆ジュンク堂書店　札幌店	011-223-1911	神奈川県	川崎市高津区	文教堂書店　溝の口本店	044-812-0063
	札幌市中央区	紀伊國屋書店　札幌本店	011-231-2131	静岡県	静岡市葵区	戸田書店　静岡本店	054-205-6111
東京都	豊島区	ジュンク堂書店　池袋本店	03-5956-6111	愛知県	名古屋市中区	三洋堂書店　上前津店	052-251-8334
	新宿区	紀伊國屋書店　新宿本店	03-3354-0131	大阪府	大阪市北区	ジュンク堂書店　大阪本店	06-4799-1090
	渋谷区	紀伊國屋書店　新宿南店	03-5361-3315	兵庫県	神戸市中央区	ジュンク堂書店　三宮店	078-392-1001
	千代田区	書泉ブックタワー	03-5296-0051	広島県	広島市南区	ジュンク堂書店　広島駅前店	082-568-3000
	千代田区	丸善　丸の内本店	03-5288-8881		広島市中区	丸善　広島店	082-504-6210
	中央区	八重洲ブックセンター本店	03-3281-1811	福岡県	福岡市中央区	ジュンク堂書店　福岡店	092-738-3322
	渋谷区	MARUZEN＆ジュンク堂書店　渋谷店	03-5456-2111				

※店舗によってバックナンバー取り扱い期間などが異なります。在庫などは各書店にご確認ください。

デジタル版のお知らせ　DIGITAL

デジタル版Software Design 好評発売中！紙版で在庫切れのバックナンバーも購入できます

本誌デジタル版は「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）と、「雑誌オンライン.com」（http://www.zasshi-online.com/）で購入できます。最新号、バックナンバーとも1冊のみの購入はもちろん、定期購読も対応。1年間定期購読なら5%強の割引になります。デジタル版はPCのほかにiPad／iPhoneにも対応し、購入するとどちらでも追加料金を払うことなく、読むことができます。

Erlangで学ぶ並行プログラミング

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

第2回 Erlangのプログラミングスタイル

この連載ではプログラミング言語Erlangとその並行プログラミングについて紹介していきます。今回は、Erlangの言語仕様に組み込まれた機能について、どのように並行プログラミングに役立つかを中心に紹介します。

並行プログラミング

並行プログラミングとは、複数の相互に関連するプログラムを同時に動かすことです[注1]。最近は複数のCPUコア[注2]を1つの機器の中で使えるのが一般的です。これらの機器のOSでは、プログラム(OSプロセス)ごとにCPUコアを割り当てます。複数のプログラムを違うCPUコアに割り当てることによって、同時に実行できます。多くのアプリケーションでは、同時に実行する個所をスレッドという部分に分け、それぞれにOSレベルでCPUコアを割り当てることによって、複数のCPUコアで処理を行い高速化を図っています。それぞれのスレッドはOSプロセスに属し、メモリやその他の資源を共有しています。

しかし、スレッドを使った並行プログラミングでは、「情報を共有すること」にまつわる問題がしばしば発生します。このような問題を相互排除問題[1]といいます。たとえば、同じメモリ領域に複数のスレッドが読み出しや書き込みを行う場合、時間的な一貫性を保つことは容易ではありません(**図1**)。具体的には、あるスレッドがその領域を読み出している間は書き込みを禁止し値が変わらないことを保証したり、一方、あるスレッドが書き込み作業を行っている間は他のスレッドは作業が終わるまで読み出せないようにしなければなりません(**図2**)。これらを正確かつ確実に行うのは大変困難です[注3]。

Erlangの考え方:共有を避ける「並行指向プログラミング」

Erlangでは並行プログラミングの安定動作を図るために、次に説明する「並行指向プログラミング」[2]という考え方を採用しています。これはスレッドを使ったものとはまったく違うアプローチです。

- プログラムを細かなプロセス[注4]に分割します。プロセスはErlangの関数として定義でき、他のプロセスとは一切情報を共有しません。

注1) 並行(concurrent)と似た用語に「並列」(parallel)がありますが、「並列動作」とは、1つの作業を部分に分けて複数の処理装置に割り振ることで速度向上を図ることが目的です。これに対し、「並行」な動作では、2つ以上の相互にやり取りが発生する作業を、複数の処理装置に分けるだけでなく、1つの処理装置を時分割して使うなどのやりくりをして処理します。このやりくりのことをスケジューリングといいます。

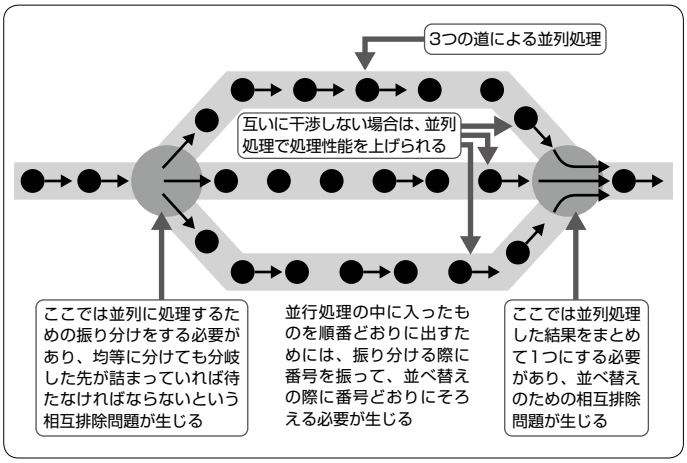
注2) CPUの多くは、「コア」と呼ばれるプログラムを実行するための部分(制御装置、レジスタ、演算装置など)と、コアがアクセスするキャッシュなどの記憶装置から成り立っています。最近のCPUは複数のコアを内蔵し、並行処理や並列処理で高速化を図るものが一般的になっています。

注3) ソースコードのバージョン管理システムのうち、Subversionなど書き込みにロックを使うものでは、同様の問題が発生します。

注4) 本連載では、以後「プロセス」はErlang言語でのプロセスを指します。OS上のプロセス(プログラムの実行単位)は「OSプロセス」と表記します。

・プロセス間で情報のやり取りをする必要がある場合は、メッセージとして明示的に送ります。送った相手のプロセスが受け取ったかどうかの確認はしません。
・各プロセスには、固有の識別子(Pid)がついていて、区別ができます。
・プロセスの間では、プロセス終了時に発生する「終了シグナル」、そして2つのプロセスの間で終了シグナルを伝えあうための「リンク」を使って、プロセスの死活監視を行うことができます。

▼図1　並行処理で起こる相互排除問題の例



この考え方を採用することによって、Erlangでは各プロセスを別々のBEAM(仮想マシン)で動作させることが可能になりました(**図3**)。暗黙の情報共有を避けることにより、複数のBEAM間であっても、通信さえできればプロセスがどのBEAMで動いていても同じように扱うことができます。

Erlangのサンプルコード

今回は実際のサンプルコード(**リスト1**)からErlangの動作を見ていくことにします[注5]。このサンプルコードはErlangでの基本的な逐次実行と関数の定義の仕方を示したものです。

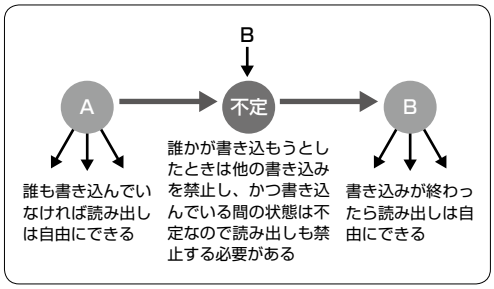
サンプルコードはrgb2huvというモジュールを定義し、その中にtohuv/1とtohuv/3という関数を定義するという簡単なものです。このコードの関数は、色のRGB値をHUV値に変換するという機能を持ちます。

リスト1の実行結果を示します(**図4**)。コンパイル、ロード、関数の実行、値のチェックの仕方をまとめています。

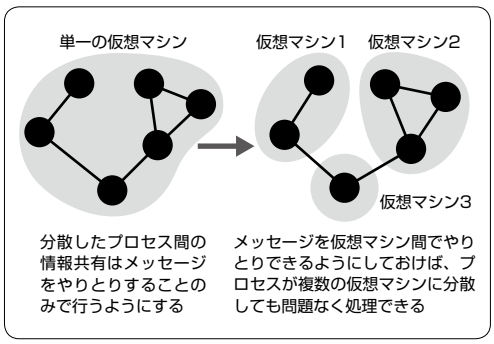
Erlangのモジュール、関数、式

Erlangでは、並行指向プログラミングを実現するために、関数型プログラミング[3]の考えを取り入れています。暗黙の情報共有をせず、

▼図2　共有資源の読み出し書き込みで起きる相互排除問題の例



▼図3　Erlangでの分散環境とプロセス配置



注5) 紙面に限りがあるため、本連載ではErlangの文法に関する詳細な説明は割愛します。連載第1回(本誌2015年4月号p.124〜129)に、Erlangの詳細を解説した各種参考書を紹介しています。

▼リスト1　rgb2huv.erl

```
%%% -*- coding: utf-8 -*-
%%% RGB値をHUV値に変換する関数を定義するモジュールです
%%% 参考URL: http://en.wikipedia.org/wiki/HSL_and_HSV

%%% ←行中のパーセントで始まる部分から行の終わりまではコメントです

%%% モジュール宣言（ファイル名はモジュール名と同じ(rgb2hiv.beam)にします）
-module(rgb2huv).

%%% モジュール内のどの関数を外部に見せるかを選択します
%%% exportの引数は関数のリストを取ります
%%% リストは大カッコでくくります
-export([tohuv/1,
         tohuv/3]).

%%% マクロ定義（モジュール内で関数の外でのみ有効）
-define(MAXVAL, 255.0).

%%% 型定義 huv()を3つの浮動小数点数を要素とするタプルとします
%%% タプルは中カッコでくくります
-type huv() :: {float(), float(), float()}.

%%% 関数 zerodiv/2 の定義です
%%% この関数はexportのリストに入っていないため外部から見えません
%%% 関数の型定義 zerodiv/2は整数の引数を2つ持ち浮動小数点数を返します
-spec zerodiv(integer(), integer()) -> float().
%%% 関数定義はここから始まります
% パターンマッチングのパターンが変わるごとにセミコロンを使います
% "_"はどんな引数にもマッチするという意味です
zerodiv(_, 0) -> 0.0; % ←ここには2番目の引数がゼロであればマッチします
% すべての変数は「大文字」で始まります
% 演算子「/」は浮動小数点数を扱います（整数同士の割り算は「div」剰余は「rem」）
zerodiv(X, Y) -> float(X) / float(Y). % ←マッチしない場合この式に来ます
%%% 関数定義は式の最後のピリオドで終わります

%%% 関数 diffdiv/3 の定義です
-spec diffdiv(integer(), integer(), integer()) -> float().
diffdiv(X1, X2, Y) -> zerodiv((X1 - X2), Y).

%%% 関数 tohuv/3 の定義です
%%% 0～255の間のRGBの整数値をそれぞれ引数に取り
%%% 返り値は{Hue, Saturation, Value}のタプルで返します
%%% (0.0 =< Hue < 360.0, 0.0 =< Saturation =< 1.0, 0.0 =< Value =< 1.0)
-spec tohuv(integer(), integer(), integer()) -> huv().
tohuv(R, G, B) ->
    % bandはビットAND演算子，borはビットOR演算子です
    % bslはビット左シフトを右側に指定したビット数分行います
    tohuv(((R band 16#ff) bsl 16) bor
          ((G band 16#ff) bsl 8) bor
           (B band 16#ff)).

%%% 関数 tohuv/1 の定義です
%%% tohuv/3と同様ですが引数はRGB値を8ビットごとにまとめたものを
%%% 24ビットの整数として与えます
%%% 例: HTMLのRGBコード"#88AA55"は"16#88AA55"と与えます
-spec tohuv(integer()) -> huv().
%%% 以下どの変数も一度しか代入されていません
tohuv(C) ->
    % 関数の中での逐次実行は式の最後にカンマを付けて続けます
    % 次の行はビットストリングのパターンマッチです
    <<R:8, G:8, B:8>> = <<C:24>>, % 24bitの整数を8bitごとに分割します
    Max = max(max(R, G), B), % erlang:max/2は大きな値を取るBIF
    Min = min(min(R, G), B), % erlang:min/2は小さな値を取るBIF
    D = Max - Min,
    H1 = 60.0 * (
        % 「case 条件式 of 値1 -> 式1; ... 値n -> 式n end」は
```

▼リスト1の続き

```
        % 条件式と各値の比較を行って一致すれば対応する式の値を返します
        % 最後の式にはセミコロンをつけず「end」で終わります
        case Max of
            R -> diffdiv(G, B, D) + 6.0;
            G -> diffdiv(B, R, D) + 2.0;
            B -> diffdiv(R, G, D) + 4.0
        end),
    Hue = case H1 >= 360.0 of
              true -> H1 - 360.0;
              false -> H1
          end,
    % 各関数は最後に評価した式の値を返します
    % マクロの参照は「?」を最初につけて行います
    {Hue, zerodiv(D, Max), Max / ?MAXVAL}.
```

▼図4　実行結果

```
ソースコードのあるディレクトリをカレントディレクトリにして "erl" コマンドを
起動してください（注意：コマンドの動作はすべてピリオドを打たないと完結し
ません）
Eshell V6.3.1  (abort with ^G)
 ↓コンパイルは「c(モジュール名)」で行います
1> c(rgb2huv).
{ok,rgb2huv} ←rgb2huv.beam ファイルが作成されました

2> l(rgb2huv).←モジュールのロードは「l(モジュール名)」で行います
{module,rgb2huv}  ←rgb2huv.beam ファイルがロードされました

3> rgb2huv:
 ↓モジュール名の後でタブ補完すると定義した関数の一覧が出ます
module_info/0  module_info/1  tohuv/1   tohuv/3

 ↓「16#」を先につけると16進数になります
4> rgb2huv:tohuv(16#ff00ff).
{300.0,1.0,1.0}

 ↓関数の実行結果を変数に代入します。式自体も値を持ちます
5> T = rgb2huv:tohuv(192,128,64).
{30.0,0.6666666666666666,0.7529411764705882}

 ↓変数の値を読み出します
6> T.
{30.0,0.6666666666666666,0.7529411764705882}0

 ↓タプルの要素の値を変数へのパターンマッチングで取り出します
7> {Hue, Saturation, Value} = T.
{30.0,0.6666666666666666,0.7529411764705882}

 ↓取り出した値の一部です
8> Hue.
30.0

9> {30.0, _, _} = T.
{30.0,0.6666666666666666,0.7529411764705882}
 ↑タプルへのパターンマッチングが成功した例です

10> {2.0, _, _} = T.
** exception error: no match of right hand
side value {30.0,0.6666666666666666,0.752
9411764705882} ←失敗するとマッチできないというエラーが出ます

11> q(). ←シェルの終了コマンドです
```

状態を保持したり入出力をする際もすべて関数を呼び出して行うというところにその特徴が表れています。

Erlangでは、関数が最小の実行単位です。関数はモジュールに属し、モジュール名、関数名、そしてアリティ（関数の引数の数）の組み合わせで示します。たとえばfooというモジュールに属するbarという2つの引数を持っている関数の場合は"foo:bar/2"と書き、"foo:bar(first, second)"のようにして呼びます。モジュール内部の関数は関数名だけで参照できます。また、BEAMの組み込み関数（BIF）の多くは"erlang"という名前のモジュールに属しており、関数名だけで参照できます。

モジュール名は単一階層しか持ちません。複数のモジュールの名前が重なった場合どれが使われるかは、モジュールに対応する.beamファイルをロードする順番で決まります[注6]。

関数は式の集まりです。Erlangの関数では、最後に評価された式の値が必ず呼び出し元に返ります。このためC言語のreturn文に相当するものはありません。呼び出す側は返った値を無視することもできるため、実用上困ることはありません。

モジュールは-module(モジュール名)宣言し、他のモジュールに見せる関数は-export([関数名/アリティ, ...])として宣言します。この

注6） Erlangシェルを呼ぶコマンドerlの-pa/-pzオプションで指定できます。

export宣言に入っていない関数は他のモジュールからは見えません。

Erlangの基本的なデータ型

基本的なデータ型をまとめて項(term)といいます。項には次の各データ型が含まれます。

- 整数(桁数制限なし、多倍長演算が可能、16進数の場合は"16#08ABCD"のように最初に"16#"をつける)
- 浮動小数点数(C言語のdouble相当)
- アトム(任意の名前で表現される識別子)[注7]
- タプル(項を要素とする固定長の配列)
- リスト(項を要素とする可変長の配列)
- ビットストリング(任意のビット長を持つパターン)
- バイナリ(ビットストリングの拡張、8ビットごとに区切られた任意のビット列)
- Fun(関数の実体、無名関数に相当)
- Pid(プロセスの識別子)
- リファレンス(erlang:make_ref/0で作られる参照子)
- レコード(要素の名前の付いた構造体、実体はタプルで実装される)
- マップ(バージョン17.xより使えるキーバリュー型テーブル)
- ポート(外部ファイルやプログラムとの入出力を示す識別子)

文字列は「各文字が要素となるリスト」で表現されます。一方、ネットワークなどを介して外部とやり取りするデータは、記憶領域の効率やデータの操作性の点から、バイナリとして扱われるのが一般的です。

既存のデータ型を組み合わせて複合データ型に名前をつけることもできます(-type)。また、各関数の引数と戻り値の型を宣言することで(-spec)、コンパイラや外部ツールDialyzerによる型チェックもできるようになります

Erlangの変数と単一代入原則

変数は大文字で始まる文字列で表されます。変数の有効範囲は関数の中だけで、使うのに宣言は必要ありません。また、同じ変数に代入できるのは1回だけです。値を変えたら、他の名前の変数に代入しなければなりません。

Erlangにはデータの参照を提供する言語要素はありません。変数にデータ構造を代入するときは、必ず独立した実体のコピーが作られます。実体への参照やポインタだけをコピーし、実体は共有したままになることはありません。一例として、代入の意味をJavaScriptと比較してみます。JavaScriptではオブジェクトを変数に代入した場合、参照渡しとなるため、新たな実体は作られません。その結果、代入した実体を操作すると、代入された側の値も変わってしまいます(**図5**)[4]。Erlangではそもそも一度代入した変数はその値を使うことしかできないため、そういうことは起こりません(**図6**)[4]。

このような言語仕様によって、Erlangでは代入する前の状態と代入した後の状態とを厳密に区別することができます。このような設計の利点は、何か操作をした後でも、操作前の状態が保存されているため、すぐに前の状態に戻すことができることです。デバッグの際も、関数を実行している間は変数の中身は変わらないため、変数の値の変化を追いかけていく必要がありません。

パターンマッチング

Erlangでは、関数の引数が一定の条件に合っているかどうかのパターンマッチングを行って、処理の場合分けをします。関数宣言はマッチングのパターンごとにセミコロン(;)で終わる節

注7) アトムは原則として小文字で始めます。小文字で始まらないアトムはクオート(')で囲うことが必要です。BEAM内で保持できるアトムの数には上限(既定値は1048576)があり、一度使ったアトムはBEAMの終了まで保持されるため、動的にアトムの名前を生成することは推奨しません。

▼図5 JavaScript(node.js)の例

```
// var a = {first: 1, second: 2}
// b = a // ポインタのコピーのみ
{ first: 1, second: 2 }
// a.second = 3
3
// b // 名前が違っても要素は共有
{ first: 1, second: 3 }
// b == { first: 1, second: 3 }
false // 何故?
```

に分かれ、最後の節はピリオド(.)で終わります。引数の内容によって処理を変えられるので、条件分岐と同様の役割をします。慣れてくると便利に使える文法要素の1つです[注8]。

case式

Erlangの関数内ではすべてが式として評価され値を持ちます。そのため、条件分岐も「……の場合であれば……という値を返す」という形で表現されます。代表的な条件分岐の構文としてcase式があります。基本的な形は

```
case 条件 of
    値1 -> 条件が値1の場合の返り値を示す式
                (をカンマでつないだもの);
    値2 -> 条件が値2の場合の返り値を示す式
                (をカンマでつないだもの);
    …… (略) ……
    値n -> 条件が値nの場合の返り値を示す式
                (をカンマでつないだもの)
          % セミコロンが最後にないのに注意
end
```

となります。それぞれの選択肢が複数の式をカンマでつないだものを含む場合は、最後の式の値が返されます[注9]。

ビットストリングとバイナリ

パターンマッチングと同様に強力なのが、ビットストリングとバイナリです。ビットストリングを使うことで、任意のビット数の整数を結合したり分けたりすることができます。また、パターンマッチングと組み合わせると、ビット列の特定の部分のパターンによって処理を変えたりすることもできます。これはネットワークのプロトコル解析などにとくに有用です。

▼図6 Erlangでの例

```
% A1 = {1,2,3}.
{1,2,3}
% B1 = A1.
{1,2,3} % B1はA1の実体のコピー
% A2 = setelement(3,A1,4).
{1,2,4}
% B1 =:= {1,2,3}. % 比較演算
true % 要素ごとの比較をしている
%%% A1, B1, A2 は個々に独立した実体
```

まとめ

今回は、プログラミング言語Erlangの言語仕様と、並行プログラミングを支援する機能について紹介しました。次回は今回紹介できなかった繰り返し処理、高階関数、Erlangのプロセス間通信、そしてメッセージパッシングを使った分散プログラミングについて紹介します。SD

参考文献

[1] 土井範久、「相互排除問題」、岩波書店、2011年、ISBN 978-4-0000-5618-2

[2] Joe Armstrong, "Concurrency Oriented Programming in Erlang",November, 9, 2002, http://ll2.ai.mit.edu/talks/armstrong.pdf

[3] 住井英二郎、「『関数型言語』に関するFAQ形式の一般的説明」(http://qiita.com/esumii/items/ec589d138e72e22ea97e)

[4] 力武健次、『Erlang/OTP並行プログラミングシステムに見る情報システム技術の課題』、第62回SEA関西プロセス分科会、2015年2月14日、スライド p.54/p.57(https://speakerdeck.com/jj1bdx/otpbing-xing-puroguramingusisutemunijian-ruqing-bao-sisutemuji-shu-falseke-ti-di-62hui-seaguan-xi-purosesufen-ke-hui)

注8) Erlangの代入とパターンマッチングは、イコール(=)の左右を同じ値にするという意味で、本質的に同じ動作です。

注9) case式のほかにif式やガードといった条件判定のための文法要素もありますが、詳細はここでは割愛します。

Android Wear アプリ開発入門

～より生活に密着するスマートデバイスの世界～

第3回 Android Wearアプリで音声入力機能を活用！

神原 健一（かんばら けんいち） Web http://blog.iplatform.org Twitter @korodroid
iplatform.orgにて情報発信するかたわら、「セカイフォン」などを開発。Droidconなどでのカンファレンス講演、MWC/CES/IFAでのプロダクト展示、執筆などの活動も実施。NTTソフトウェア株式会社テクニカルプロフェッショナル。現在はAndroid以外のモバイルOSにも取り組み、公私にわたってモバイルアプリの世界に没頭中。

はじめに

前号（4月号）では、Android Wear（以降「Wear」と表記）アプリ開発のうち、Wear実機での実行／通知機能について触れました。今回は、Wearアプリ上での「音声入力機能」をご紹介します[注1]。続いて、開発したWearアプリをGoogle Play上で配信するために必要な手順についても触れます。いずれもWearアプリ開発において重要なトピックですので、ぜひマスターしていただけると幸いです。

Wearの音声入力機能

スマホやタブレット（以降「Handheld」と表記）向けAndroidでは、今や当たり前のように使われている音声認識機能ですが、Wearでもサポートされています。たとえば、Wearが時間を表示している画面（**図1-A**[注2]）で、時計に向かって「OK Google」と話したのちに、「スカイツリーを探して」と言うと（**図1-B**）、Google検索した

注1） 開発環境構築やエミュレータ・実機での実行方法、通知機能の実装などは、これまでの連載をご参照ください。

注2） この画面はWatch Faceと呼ばれます。アプリ開発者が任意のWatch Faceを開発することもできます。

▼図1 時計表示画面での音声入力

▼表1 音声コマンド例

音声コマンド	実行される操作
[タイマー設定]+「5秒」	5秒後に鳴動するタイマーが設定される
[メールを送信]+「太郎、後で会おう」	太郎さんへ「後で会おう」というメールが送信される
[ナビを開始]+「レインボーブリッジまで連れて行って」	レインボーブリッジまでのナビゲーションが行われる
[歩数計を表示]	歩数計アプリが起動される

前半の[...]は音声で入力することも画面から選択することも可能です。後半の「...」は音声で入力します。

結果が画面に表示されます(**図1-C**[注3])。この指示は、音声コマンドと呼ばれています。Google検索以外にも**表1**に示すような音声コマンドがサポートされています。

このようにちょっとしたことであれば、Handheldをわざわざ取り出さなくても、Wearに向かって音声で指示するだけでやりたいことを行うことができます。WearはHandheld以上に画面が小さいということもあり、執筆時点で、OSの標準機能としてのソフトキーボードは提供されていません。そのためユーザの入力手段として、音声を適切に活用することは重要です。

アプリ内での音声入力

このように便利な音声入力機能ですが、我々が開発するアプリの中でも活用することができます。たとえば、前号で紹介した通知に対して、アクションとして音声入力機能を追加してみましょう(**図2**)。①音声の入力②入力結果の取得、の順に解説します。

▶①音声の入力

①を行うためには、**リスト1**のコードを記述します。

まず音声入力結果を渡すActivityを指定したIntentを生成します。次に、音声入力機能(RemoteInput)のインスタンスを生成します。その際、あとで音声結果取得に必要となる任意のキー(今回は、EXTRA_VOICE_REPLY)を指定しています。続いて、音声入力アクションの生成です。RemoteInputインスタンスをaddRemoteInput()メソッドにより設定します。音声入力アクションを通知マネージャに設定するために必要となる、ウェアラブル用拡張機能(WearableExtender)のインスタンスを生成します。そして、ウェアラブル用拡張機能のaddAction()メソッドを用いて、音声入力アクションを設定すれば完了です。「音声入力」ボタンを

注3) Google検索を行うには、Wear実機とHandheld実機のペアリングが行われ、かつHandheldがインターネットに接続されている必要があります。

▼図2　音声入力機能(自由入力)

▼リスト1　音声入力の処理実行

```
int notificationId = 1010;
final String EXTRA_VOICE_REPLY = "extra_voice_reply";

NotificationCompat.Builder builder = new NotificationCompat.
Builder(this).setSmallIcon(R.drawable.ic_launcher)
    .setContentTitle("音声入力").setContentText("音声で入力します。");
// 音声入力結果を渡すIntentの生成
Intent replyIntent = new Intent(this, MainActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,
    replyIntent, 0);
// 音声入力機能のインスタンス生成
RemoteInput remoteInput = new RemoteInput.Builder(
    EXTRA_VOICE_REPLY)
    .setLabel("あなたの趣味は何ですか？").build();
// 音声入力のアクション生成
Action replyAction = new Action.Builder(
    android.R.drawable.ic_btn_speak_now, "音声入力", pendingIntent)
    .addRemoteInput(remoteInput).build();
// ウェアラブル用拡張機能のインスタンス生成
NotificationCompat.WearableExtender wearableExtender =
    new NotificationCompat.WearableExtender();
// 音声アクションの設定
NotificationManagerCompat manager =
    NotificationManagerCompat.from(this);
manager.notify(notificationId, builder.extend(
    wearableExtender.addAction(replyAction)).build());
```

▼リスト2　音声入力の結果取得

```
// 音声結果が格納されたIntentを用いて処理実行
private CharSequence getMessageText(Intent intent) {
    // 音声入力結果を含むBundleの取得
    Bundle remoteInput = RemoteInput.getResultsFromIntent(intent);
    if (remoteInput != null) {
        // Bundle経由で文字列の取得
        return remoteInput.getCharSequence(EXTRA_VOICE_REPLY);
    }
    return null;
}
```

▼リスト3　音声入力機能（選択肢表示）

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(
    this).setSmallIcon(R.drawable.ic_launcher)
    .setContentTitle("音声入力").setContentText("音声で入力します。");

// 音声入力結果を渡すIntentの生成
Intent replyIntent = new Intent(this, MainActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, replyIntent, 0);

String[] replyChoices = getResources().getStringArray(R.array.reply_choices);   a

// 音声入力機能のインスタンス生成
RemoteInput remoteInput = new RemoteInput.Builder(EXTRA_VOICE_REPLY)
    .setLabel("あなたの趣味は何ですか？")

    .setChoices(replyChoices).build();   b

// 音声入力のアクション生成
Action replyAction = new Action.Builder(
    android.R.drawable.ic_btn_speak_now, "音声入力", pendingIntent)
    .addRemoteInput(remoteInput).build();
// ウェアラブル用拡張機能のインスタンス生成
NotificationCompat.WearableExtender wearableExtender =
    new NotificationCompat.WearableExtender();
// 音声アクションの設定
NotificationManagerCompat manager = NotificationManagerCompat.from(this);
manager.notify(notificationId, builder.extend(
    wearableExtender.addAction(replyAction)).build());
```

タップすると、音声入力を行うための画面が表示されます。

▶②入力結果の取得

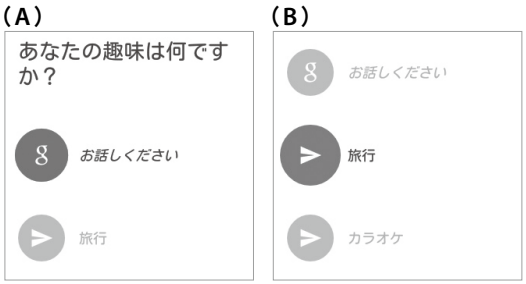
続いて、②について解説します。ユーザが入力した音声は、文字列として取得することができます。そのためには、**リスト2**のコードを記述します。

この処理は音声入力結果を受け取るActivity側で実装します。音声入力の完了後、送信されるIntentからRemoteInputクラスのgetResultsFromIntent()メソッドを用いて、結果が含まれるBundle型データを取得します。さらに、その中からキー（EXTRA_VOICE_REPLY）を指定して値を取得しています。このように音声入力結果を文字列として取得し、アプリ内の処理で利用することができます。

音声入力をユーザに行ってもらうときに、画面に入力候補を表示することも可能です（**図3-A**）。画面に収まらない候補は画面を下にスクロールすることで表示されます（**図3-B**）。具体的には、**リスト3**と**リスト4**のコードを記述します。

リスト3で、**リスト1**との差分は**a**と**b**です。まず**a**で、表示する選択肢をarrays.xml（**リスト4**）から取得し、String配列に格納しています。

▼図3　音声入力機能（選択肢表示）



▼リスト4　arrays.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="reply_choices">
        <item>旅行</item>
        <item>カラオケ</item>
        <item>食事</item>
    </string-array>
</resources>
```

続いて、**b**でRemoteInputのインスタンスにsetChoices()メソッドを用いて設定をしています。これにより、音声入力を実行した際に、これらの候補も画面に一緒に表示されます。候補内容は音声で入力することも、画面をタップして選択することも可能です。

音声を用いたアプリ起動

Wearアプリ（通知を除く）は、一般的にWear実機のランチャーからアプリを選択して起動します。その代わりに、「OK Google」のあとに「○○ 開始」という音声コマンドを用いてアプリを起動できるようにすることも可能です。そのためには、WearアプリのAndroidManifest.xmlを**リスト5**のとおり設定してください。起動に用いるアプリ名は、Activityのlabel属性として設定しています。このように設定しておけば、「ハロー 開始」としゃべることでアプリを起動することが可能となります。

Google Playでのアプリ公開

話題を変えて、Google Playでのアプリ配信方法について見ていきましょう。アプリ配信の話に入る前に、ユーザがWear実機にアプリをインストールする流れを解説します。本稿執筆時点では、Wear実機にGoogle Playは搭載されていません。そのため、ユーザがWear実機にアプリを直接インストールすることはできません[注4]。その代わり、Handheld実機からGoogle Playを経由して、Wear実機にアプリをインストールすることが可能です。

ただ、この方法を用いてWear向けアプリを配信するには、「アプリのパッケージング方法」を工夫する必要があります。具体的には**図4**に示すように、Handheldアプリの中に、Wear用モジュールを同梱させてパッケージングしたものをGoogle Playで公開するということを行います。こうしておけば、ユーザがGoolge PlayからHandheld実機にアプリをインストールすると、Wear実機にも自動的にWearアプリがインストールされます。パッケージングは一見難しそうですが、Android Studio（+ Gradle）なら簡単にできます。

パッケージングに関して、2つの重要なポイ

注4）開発者であれば、adbコマンドなどを用いてアプリをWear実機に直接インストールすることは可能です。詳細は前号をご参照ください。

▼リスト5　音声コマンドでアプリを起動できるようにするためのAndroidManifest.xml設定

```
〈activity
    android:name=".MainActivity"
    android:label="ハロー" 〉
    〈intent-filter〉
        〈action android:name="android.intent.action.MAIN" /〉
        〈category android:name="android.intent.category.LAUNCHER" /〉
    〈/intent-filter〉
〈/activity〉
```

ントを解説します。

▶[ポイント1] Handheld用のビルド定義ファイル(build.gradle)にて、Wear用モジュールを同梱する宣言

たとえば、連載第1回で開発した「HelloWear」アプリのように、Handheld用モジュールとWear用モジュールが各1つ含まれるプロジェクトを例に説明します。具体的には、Handheld用のビルド定義ファイル(**図5の(1)**)の内容を、**リスト6**に示すように**wearApp project(':モジュール名')**と定義しておくことで、Handheld用パッケージングを行うときにモジュール名に指定したWear用モジュール(今回は"wear")を同梱できます。正しく設定されているかを必ず確認してください。

▶[ポイント2] Handheld/Wear両方のbuild.gradleにて、リリース署名を付与する宣言

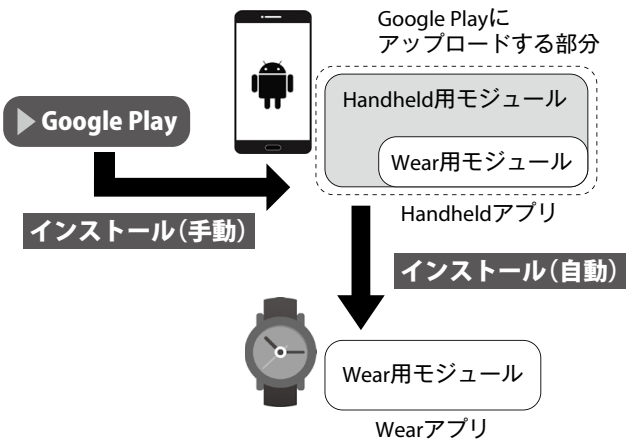
Google Playでアプリを配信するには、リリース署名の付与が必須となります。具体的には、Handheld/Wear両方のビルド定義ファイル(**図5の(2)**)で、**リスト7**に示すようにリリース署名に必要な情報の定義(signingConfigsのブロック)を行い、さらに、releaseビルドを行う際にリリース署名を実行するための宣言(signingConfigs.releaseConfig)を実施する必要があります。releaseConfigの定義内容は**表2**を参考に各自の環境に合わせて設定してください。

ちなみに、アプリ公開用Keystore(release.keystore)は、Android Studio(以降「Studio」と表記)上のメニューから[Build]→[Generate Signed APK]から生成可能です。さらに、ビルド時にリリース用ビルドが実行されるようにするための設定が必要です。Studio左下から「Build Variants」を選択し、同ウィンドウ内でHandheldとWearの両方のBuild Variantを「release」に設定してください(**図6**)。

▼リスト6　build.gradleの抜粋(1)

```
dependencies {
    (...略...)
    // wearという名前のモジュールを同梱する宣言
    wearApp project(':wear')
    (...略...)
}
```

▼図4　Wearアプリのインストールの流れ



ここまでで事前準備は完了です。Wear用アプリを内包したHandheld用アプリをビルドしてみましょう。Studioのツールバーの中央付近のリストボックスで「mobile」を選択し(**図7**)、緑色の実行ボタンを押してください。どのデバイスで実行するかを聞かれますので、Handheld実機を選択してください。設定に問題がなければ、Handheld実機でアプリが実行されるはずです。

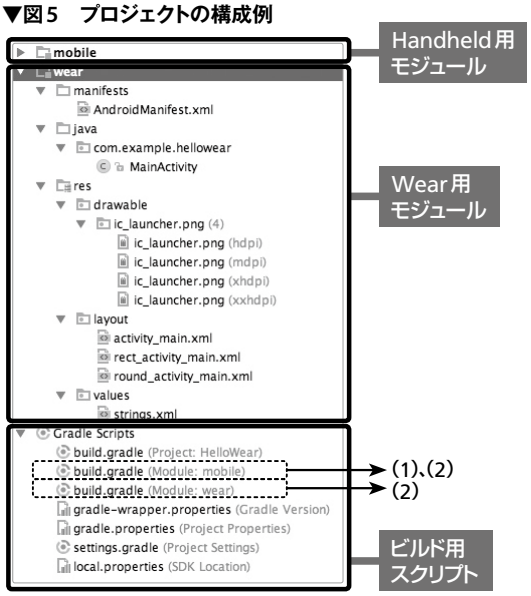
少し待つと、Wear実機にもBluetooth経由で自動的にWearアプリがインストールされます。Wear実機で、ホーム画面をタップ→「開始」→「Hello Wear」と選択することでアプリを起動できます(**図8**)。また、apkは、〈プロジェクトフォルダ〉/mobile/build/outputs/apkに「mobile-release.apk」として生成されているはずです。Google Playには、このファイル(Handheld用モジュール+Wear用モジュールをまとめたapk)を登録すればOKです。

おわりに

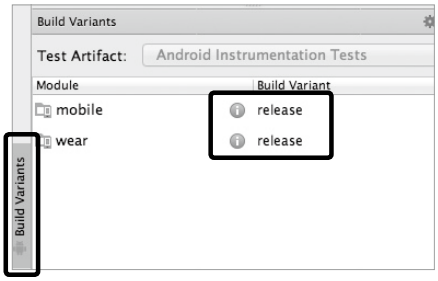
今回はWearアプリにおける音声入力に加え、Google Playでの公開方法について解説しました。音声入力を適切に活用することで、Wearアプリの操作性を向上させ、アプリとしての魅力を高めることができます。初回から読んでいただいている方には、Wearアプリ開発の世界を少しずつつかんでいただけているのではないかと思います。

次回もWearアプリを開発するうえで重要なトピックを取り上げます。お楽しみに！SD

▼図5　プロジェクトの構成例



▼図6　Build Variants設定

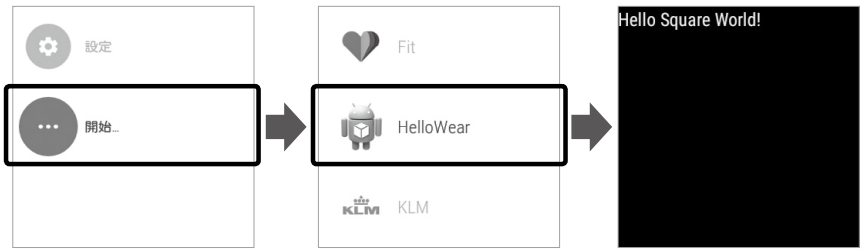


▼図7　ビルドモジュールの選択

▼リスト7　build.gradleの抜粋（2）

```
signingConfigs {
    releaseConfig {
        storeFile file("../release.keystore")
        storePassword "myPassword"
        keyAlias "myAlias"
        keyPassword "myPassword"
    }
}
buildTypes {
    release {
        (...略...)
        signingConfig signingConfigs.releaseConfig
    }
}
```

▶図8
Wearアプリの
起動



▼表2　releaseConfigの定義例

Key	Value例	内容
storeFile	file("../release.keystore")	アプリ公開用Keystoreのファイルパス
storePassword	"myPassword"	同Keystoreのパスワード
keyAlias	"myAlias"	同Keystoreの秘密鍵エイリアス
keyPassword	"myPassword"	同Keystoreの秘密鍵パスワード

Mackerelではじめるサーバ管理

Writer 坪内 佑樹(つぼうち ゆうき) ㈱はてな
Twitter @y_uuk1

第3回 運用しながら育てるサーバ監視のルール

Mackerelでは、サーバ監視においては外せない基本項目の監視機能が用意されているのはもちろん、ユーザの環境に合わせてさまざまな監視項目を追加できます。今回は、サーバ監視の基本、ルール設定のためのアイデア、アラートのしくみについて、筆者が現場の運用で培ってきたノウハウとともに紹介します。

前回はMackerelを使ったメトリックの可視化について紹介しました。第3回目となる今回はMackerelを使ってサーバをどのように監視したらいいのかについて紹介します。

サーバ監視のいろは

なぜ監視が必要か

前回紹介したメトリックの可視化(サーバの状態をグラフ化すること)によって、サーバに異常が起きていないかを目視で確認できるようになりました。一方で、サーバはハードウェアの故障やトラフィックのバーストなどにより、人間が寝ている間にも突然障害が発生することがあります。そういった突然の障害に備えて、人間がグラフを見ていない間でも異常に気づくための監視のしくみが必要です。

何を監視するのか

サーバの監視には外形監視、死活監視、リソース監視などさまざまな種類があります。

まず、外形監視はシステム全体に障害が起きていないかを監視します。たとえば、Webサイト自体が落ちていないかなど、システムを構築する個々のコンポーネントに依存しない監視を行います。

死活監視は個々のサーバやネットワーク機器が停止していないかを監視します。ApacheやMySQLなどのミドルウェアやアプリケーションのプロセスが落ちていないかを監視することもあります。

リソース監視はOSやミドルウェアのメトリックの値や変化率が閾値(いきち)を超えていないかを監視します。

そのほかにも、ログ監視やServerspec[注1]を用いたサーバ構成の監視など、監視の種類は多岐に渡ります。システムが正常稼働するために必要なものは何かを考え、適切に監視する必要があります。

どのように監視するのか

これらのような監視を実現するためには、どのようなしくみが必要でしょうか。

外形監視と死活監視の場合、監視対象のシステムの外からHTTPやICMPなどのネットワークプロトコルを用いて、定期的にアクセスする必要があります。さらに、レスポンスがないもしくはレスポンスが基準よりも遅い場合は、メールやチャットなどでサーバ管理者に異常を通知することも必要です。

リソース監視の場合は、SNMPなどを用い

注1) URL http://serverspec.org

てOSやミドルウェアのメトリックを定期的に収集し、収集したメトリックとあらかじめ設定した閾値を比較して、異常があれば外形監視と死活監視同様にサーバ管理者に通知します。

実際にこのようなサーバ監視を始めようとすると、自分で一からしくみを作るのはたいへんですので、NagiosやZabbixなどのOSSを用いて監視システムを構築することが多いでしょう。しかし、実際に監視システムのためのサーバを自分で構築し、運用するのは多くの苦労を伴います。とくにしっかりした監視を行う場合は、監視システム自体に障害が起きたときのために、監視システムの監視システムが必要なこともあるでしょう。そこで、MackerelのようなSaaS(Software as a Service)として提供されているサービスを利用すると、監視システムの構築と運用の手間を劇的に減らせます。

Mackerelにおける監視とアラート

Mackerelでは執筆時点(2015年3月)で死活監視の一部とリソース監視をサポートしています。Mackerelの監視機能を利用するには、監視ルールの設定、アラートの確認および通知の設定が必要となります。設定方法について、詳しくは本連載の第1回目(3月号)の記事またはMackerelのヘルプドキュメント[注2]を参照してください。

死活監視

Mackerelでは死活監視のうち「ホストの疎通確認」をサポートしています。**図1**のように「ホストの疎通監視」はデフォルトで設定されます。

第1回で説明したように、SaaSというサービスの性質上、MackerelはPush型アーキテクチャを採用しています。しかし、実際の監視対象サーバはプライベートネットワークに置かれていることが多いでしょう。したがって、Mackerelのサーバからユーザのサーバへ Pingを送ることは現実的ではありません。そこでMackerelでは、mackerel-agentが1分おきにメトリックを送信していることに着目し、メトリック送信を5分以上受信できなければ当該ホストとの疎通がなくなったと判断し、アラートを生成します。

リソース監視

mackerel-agentを用いたホストメトリックの収集と可視化については、4月号の第2回で紹介しました。Mackerelでは収集した各メトリックについて、閾値を設定できます。

すべてのメトリックを監視すべきかといえばそうではなく、監視すべきメトリックとそうでないメトリックを見分けることが重要です。一般に監視しておくと良いメトリックを次に挙げてみました(括弧内はMackerel上でのメトリック名を表しています)。

- CPU利用率(CPU %)
- メモリ使用率(Memory %)
- スワップ使用率(Swap %)
- ファイルシステム使用率(Filesystem %)

▼図1　ホストの疎通確認

注2) URL http://help-ja.mackerel.io

これらのメトリックはMackerelでは**図2**のように優先的に表示されるようになっています。とりあえず最低限の監視を設定しておきたいときは、これらのメトリックについて閾値を設定すると良いでしょう。

これらのOSが提供するメトリック以外に、ミドルウェアが提供するメトリックを監視することも有用です。OSのリソース消費が顕著になる前に、問題に気づけることもあります。ミドルウェアのメトリックを監視するためには、前回紹介したようにmackerel-agentのプラグインを用いて、メトリックを投稿する必要があります。詳しくは、公式プラグイン集のヘルプ[注3]を参照してください。

公式プラグインを用いたメトリックのうち、筆者が監視すると有用だと考えるメトリックの一部を**表1**にリストアップしてみます。

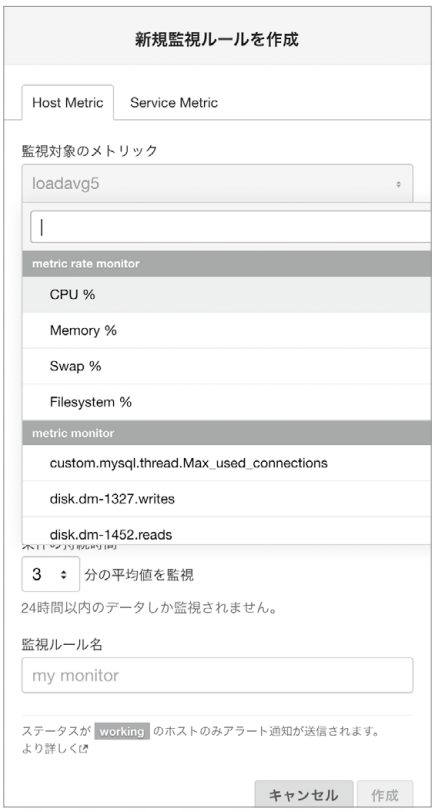
システム規模の成長に応じて変化するメトリックは閾値を設定しづらいですが、接続エラー数などのエラー系のメトリックは比較的閾値を設定しやすい項目です。即対応が必要な場合も多いため、監視しておいて損はないでしょう。

しかし、できればエラーが発生する前にシステムの異常の兆候を知りたいものです。MySQLの場合、接続上限数があらかじめ決まっていますので、接続上限数を確認しておき、接続上限数のうちたとえば90%の閾値を設定することにより、エラー発生前にアラートを発生させることができます。MySQLの接続上限数は次のようなクエリで確認できます。

```
# mysql> show global variables like
'max_connections';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| max_connections | 10000 |
+-----------------+-------+
```

ここで紹介した監視項目はほんの一部です。管理対象のサーバやミドルウェアの特性に合わせて、何を監視すべきかを考えることが求められます。

▼図2　基本メトリックの監視項目



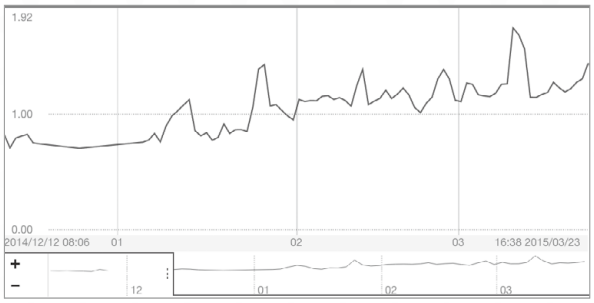
▼表1　お勧めの監視対象メトリック

MySQLの接続エラー数 (custom.mysql.connections.Aborted_connects)
MySQLのレプリケーション遅延 (custom.mysql.seconds_behind_master.Seconds_Behind_Master)
MySQLのテーブルロック数 (custom.mysql.table_locks.Table_locks_waited)
InnoDBの行ロック数 (custom.mysql.innodb_row_lock_waits.Innodb_row_lock_waits)
Redisの接続エラー数 (custom.redis.connections.rejected_connections)
Nginxのアイドル状態のコネクション数 (custom.nginx.queue.Waiting)
JVMのフルGC時間の割合 (custom.jvm.nettyserver.gc_time_percentage.FGCT)

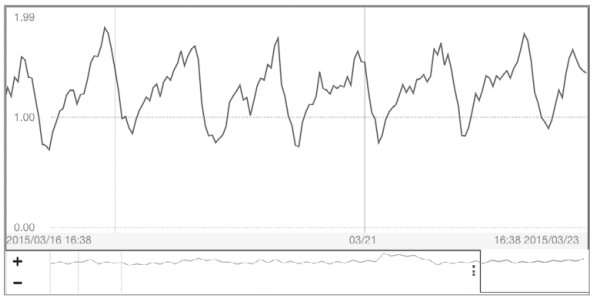
注3）URL http://help-ja.mackerel.io/entry/howto/mackerel-agent-plugins

▼図3　サービスメトリックの監視ルール設定

▼図4　直近数ヵ月のDBサーバのロードアベレージのグラフ



▼図5　直近1週間のDBサーバのロードアベレージのグラフ



サービスメトリックの監視

前回、Fluentdを用いたサービスメトリックの活用について紹介しました。Mackerelでは**図3**のようにサービスメトリックにも監視ルールを設定できます。

たとえば、Fluentdとリバースプロキシのアクセスログを組み合わせてレスポンスタイム情報を投稿することにより、レスポンスタイムの平均値、90パーセンタイル値、99パーセンタイル値などの項目を監視できます。レスポンスタイム情報の投稿方法については、公式サイトの記事「fluentdでサービスメトリックを投稿する」[注4]を参照してください。サービスメトリックによりあらゆるメトリックの監視が可能になり、クラウドサービスの金銭的コストの監視など、これまで監視していないまたは監視しづらかったメトリックを監視できます。サーバ特有のメトリック以外にも、どのような項目を監視できるかを考えるとおもしろいと思います。

監視ルール設定の考え方

閾値を設定するといっても、どのような基準で閾値を設定すれば良いのでしょうか。

筆者は、アラートは必ず実際に対応するものだけに限定すべきだと考えています。「このアラートは無視して良い」「あのアラートは対応しなければならない」などの暗黙の対応ルールができてしまうのはよくありません。したがって、本当にシステムの稼働に問題があると判断できる閾値を設定する必要があります。すでに稼働しているシステムであれば、ピーク時間帯のトラフィックに耐えられるように設計されているはずですので、ピーク時間における各メトリック値が閾値決定の参考になると思います。Mackerelでは執筆時点で最大2年分のメトリックを保存できますので、筆者の場合**図4**のように直近数ヵ月のシステム成長を確認しつつ、**図5**のように直近1週間のグラフでピーク時の値を確認して、閾値を決定することが多いです。

注4）URL http://help-ja.mackerel.io/entry/advanced/fluentd

とはいえ、とくに新規構築のシステムの場合、今後の負荷状況が予想しづらいため最初から適切な閾値を設定するのは難しいものです。筆者の場合、システムの障害を見逃すことがないように最初は厳しめの閾値を設定し、負荷状況を見つつ条件を緩くしていき、徐々に最適な閾値に収束させていくというようなステップを踏むこともあります。

このようなステップを踏む場合、監視ルールの設定が気軽に行えることが重要です。Mackerelはユーザインターフェースにこだわって開発していますので、すばやく閾値の変更ができるようになっています。

また、アラートには必ず対応するといっても、即時対応が必要な場合と後日対応で問題がない場合があるでしょう。Mackerelではほかのサーバ監視ツール同様Warning条件とCritical条件を個別に設定できます。即時対応が必要なレベルの閾値をCritical条件に設定し、後日対応で十分なレベルの閾値をWarning条件に設定するというような使い方ができます。

さらに、監視ルールの設定をサーバごとに変えたいという場合もあるでしょう。Mackerelでは、監視ルールをロール単位で設定できます。逆に、このロールだけルールから除外したいといった場合、**図6**のように「除外条件」を設定することもできます。CIサーバ(継続的インテグレーションを実行するサーバ)などの一時的に負荷がバーストするようなロールを列挙して条件を緩くすることや、「除外条件」によりそれらのロールを監視しないといったこともできます。

▼図6　除外条件

お勧めの監視設定フロー

最後に、筆者お勧めの監視設定フローを紹介します。

❶すべてのサーバに対してCPU使用率、メモリ使用率、スワップ使用率、ファイルシステム使用率の監視ルールを設定する

❷各種エラー系のメトリックの監視ルールを設定しておく

❸いったん様子を見て、特定のロールだけ頻繁にアラートが来るようなら除外条件で無視するか、別途当該ロール用の監視ルールを作成する

❹障害が発生したときに、特徴的な値の変化をしたメトリックについて監視ルールを作成する。同じ障害を早めに検知できるように、回帰的に監視ルールを育てる

いきなり最適な監視ルールを考えるのではなく、問題が起きたらルールを調整していき、システムの変化に合わせて監視を育てていくことが重要だと考えています。

まとめ

今回は、サーバ監視の基本の説明と、Mackerelにおける監視とアラートを紹介しました。サーバ監視は一定の決まりにしたがっていれば良いというわけではなく、自分たちの運用しているシステムや所属している組織の特性に合わせて、何を監視するのか、どのような基準でアラートを発生させれば良いかを常に考える必要があります。

次回は、Mackerelとさまざまな外部ツールとの連携方法について紹介する予定です。SD

書いて覚える Swift 入門

第 5 回 遺産の継承（その2）

Writer 小飼 弾（こがい だん）　twitter @dankogai

前回に引き続き、SwiftからCやObjective-Cの遺産を今回も活用します。前回はおもにC――厳密にはlibc ――の機能をSwiftから使うにはどうしたらよいかを見ていきましたが、今回はObjective-C、つまりフレームワークをどう活用していくかを見ていくことにしましょう。

import (Cocoa | UIKit)

Objective-CではCにおけるlibcに相当するのが、OS XではCocoa、iOSではUIKitです。ただしlibcよりできることははるかに多彩です。たとえば、あるURLにアクセスしてそのコンテンツを表示するというのは、Webすらなかった時代に作られたlibcでは簡単には書けませんが、CocoaやUIKitであればわずかこれだけです（**リスト1**）。

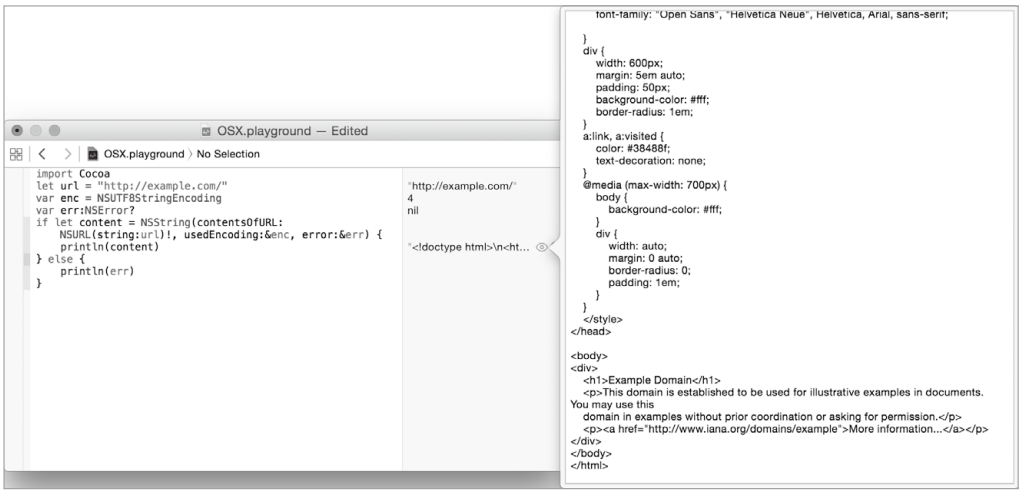
なんと、NSString()という文字列を初期化するAPIに適切なパラメータを渡すだけで、ソケットを初期化し、HTTPプロトコルでサーバにアクセスし、その内容をGETしてくれるわけです。あまりに楽なので、スクリプト言語でプログラミングしているような感覚です（**図1**）。

この場合URLのコンテンツはテキストです

▼リスト1　CocoaやUIKitのサンプル

```
import Cocoa // OS X の場合。iOS ならUIKit
let url = "http://example.com/"
var enc = NSUTF8StringEncoding
var err:NSError?
if let content = NSString(
  contentsOfURL: NSURL(string:url)!,
  usedEncoding:&enc,
  error:&err
) {
    println(content)
} else {
    println(err)
}
```

▼図1　リスト1の実行結果



が、JSONをパースする機能すら基本装備しています(**リスト2**)。

しかしパースしたJSONから特定のアイテムを抜き取りたいとなると、かなり面倒なことになります(**リスト3**)。

JSONをサポートする多くの言語でjson["ItemAttributes"]["Author"]と一度に書けるところを、まずlet item = json["ItemAttributes"] as? NSDictionaryでitemを取り出し、さらにlet author = item["Author"] as? NSStringとNSStringを取り出しという具合に、型が静的であるというSwiftの特徴がアダとなってしまっています。どうにかしてjson["ItemAttributes"]["Author"]と書く方法はないでしょうか？　さらに可能ならJavaScriptのようにjson.ItemAttributes.Authorと書けないのでしょうか？

▼リスト2　JSONのパース機能を備えている

```
import Cocoa
let url = "http://api.dan.co.jp/asin/4534045220.json"
var enc = NSUTF8StringEncoding
var err:NSError?
if let content = NSString(
  contentsOfURL: NSURL(string:url)!,
  usedEncoding:&enc,
  error:&err
  ) {
    if let json = NSJSONSerialization.JSONObjectWithData(
        content.dataUsingEncoding(enc)!,
        options: nil, error: &err) {
        println(json)
    } else {
        println(err)
    }
} else {
    println(err)
}
```

ラッパーのススメ

その試みがSwiftyJSON[注1]であり、拙作のSwift-JSON[注2]です。たとえばSwift-JSONならリスト3のコードは

```
let author = JSON(url:"http://api.dan.co.jp/asin/4534045220.json")["ItemAttributes"]["Author"].asString
```

と1行で済んでしまいます。さらにスキーマをclassとして実装すれば、**リスト4**のようにす

▼リスト3　JSONから特定のアイテムを抜き出す

```
import Cocoa
let url = "http://api.dan.co.jp/asin/4534045220.json"
var enc = NSUTF8StringEncoding
var err:NSError?
if let content = NSString(contentsOfURL: NSURL(string:url)!, usedEncoding:&enc, error:&err) {
    if let json:AnyObject = NSJSONSerialization.JSONObjectWithData(
        content.dataUsingEncoding(enc)!,
        options: nil, error: &err) {
            if let item = json["ItemAttributes"] as? NSDictionary {
                if let author = item["Author"] as? NSString {
                    println(author)
                }
            }
    } else {
        println(err)
    }
} else {
    println(err)
}
```

注1）https://github.com/SwiftyJSON/SwiftyJSON
注2）https://github.com/dankogai/swift-json

ら書けます。

SwiftyJSONやSwift-JSONはこれをどのように実現しているのでしょうか？　ソースコード全体を読んでいただければ一目瞭然なのですが、SwifyJSONは1,163行、Swift-JSONは432行で紙幅にとても収まりません(本原稿執筆現在で)。ここではSwift-JSONのキモだけ解説します。

Swift-JSONのインスタンス変数は、たった1つです。

```
public class JSON {
    private let _value:AnyObject
    // ....
}
```

これに対し、subscriptは2種類定義されています(**リスト5**、**リスト6**)。

▼リスト4　ラッパーの使用例

```
class ASIN : JSON {
    override init(_ obj:AnyObject){ super.init(obj) }
    override init(_ json:JSON)  { super.init(json) }
    var ItemAttributes: ASIN { return ASIN(self["ItemAttributes"]) }
    var Author:    String { return self["Author"].asString! }
}

let author = ASIN(url:"http://api.dan.co.jp/asin/4534045220.json").
ItemAttributes.Author
```

▼リスト5　subscript(idx:Int)

```
    public subscript(idx:Int) -> JSON {
        switch _value {
        case let err as NSError:
            return self
        case let ary as NSArray:
            if 0 <= idx && idx < ary.count {
                return JSON(ary[idx])
            }
            return JSON(NSError(
                domain:"JSONErrorDomain", code:404, userInfo:[
                    NSLocalizedDescriptionKey:
                    "[¥(idx)] is out of range"
                ]))
        default:
            return JSON(NSError(
                domain:"JSONErrorDomain", code:500, userInfo:[
                    NSLocalizedDescriptionKey: "not an array"
                ]))
            }
    }
```

つまり、json[0]のように添え字がIntであればインスタンス変数をNSArrayとみなし、json["name"]のように添え字がStringであればNSDictionaryとみなして、その要素から新たなJSONオブジェクトを生成しているわけです。そして要素が存在しない場合は、NSErrorからJSONオブジェクトを生成し、インスタンス変数がNSErrorの場合はそのまま自分自身を返すことで、HaskellのEitherが一度Nothingになればずっと Nothingであるように、最初に発生したエラーが引き継がれるというわけです。

このようなラッパーは同等の機能をフルスクラッチでSwiftで書くよりずっと簡単に書けますし、書くことによってSwiftとObjective-Cの連携がどのようになされているかを体得することもできます。読者の皆さんも、これぞというものがあったらぜひ書いて、GitHubなどで公開してみてください。

AnyObjectとAnyの違い

SwiftのAnyObjectは、Objective-Cにおけるidに相当します。id同様なんでも入りますが、適切に使うにはisで適切な型を判定したり、asで適切な型に変換したりしなければなりません。

また、CocoaやUIKitなど、Objective-C由来のフレームワークをimportしておく必要もあります(**図2**)。

```
import Cocoa
var ao:AnyObject
ao = "assign"
// ao += " any value"   // error
ao = (ao as String) + " any value"
ao = 40
// ao += 2              // error
ao = (ao as Int) + 2
```

ところがSwiftにはAnyObjectとは別にAnyという型も存在します。前述のAnyObjectをAnyに変えてもそのまま動いてしまいますし、import Cocoaをコメントアウトしてもそのまま動いてしまいます(**図3**)。

なぜ、Swiftには「なんでもありな型」が2つも存在するのでしょう？

sizeof()で双方の型を見てみると、面白いことがわかります。64bitプラットフォームではsizeof(AnyObject)は8なのに対し、sizeof(Any)は32。AnyObjectは1ワード、Anyは4ワードです。賢明な読者であれば、この時点で予想がつくでしょう。AnyObjectは参照、つまりclassであるのに対し、Anyは実値、つまりstructなのです。

さらに「禁断の組込み関数」、unsafeBitCastを使ってAnyがどうなっているのかを見てみましょう(**リスト7、図4**)。

なんのことはない。4ワードのうち頭から本来の値を詰め込んだうえで、最後の1ワードに「型ID」が入っているだけのです。Swiftの Structは、Intや Doubleが1ワード、関数が2ワード、StringやArrayやDictionaryが3ワードなので、Anyの中にすべて納まります。

これに対し、AnyObjectの正体は、Objective-Cで書けばid *、オブジェクトへのポインタで、型情報はAnyのように値そのものの一部ではなくその参照先に格納されています。

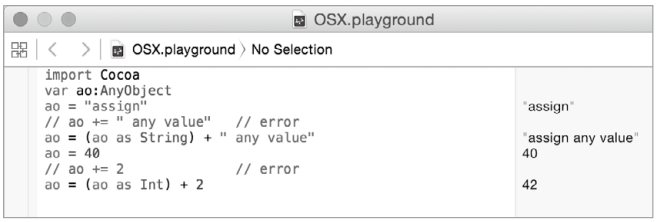
それではAnyはどこで使われているかというと、Xcodeの内部です。Xcodeは現在書かれているコードにあわせて振る舞いを変えますが、この振る舞いを受け取る関数は当然ありとあらゆる型を受け取れなければなりません。Swiftにはreflect()という関数がありますが、これがAnyを活用している関数の1つで、これを用いると内観(introspect)するためのコードを自作することもできます。

しかしそうでもない限り、Anyを使うケースはほとんどないでしょう。以前紹介したようにSwiftには総称関数とプロトコルがあるので、静的型の特長を活かすためにもAnyの使用は避けるべきです。

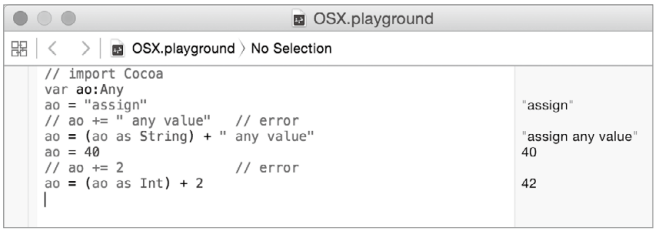
▼リスト6　subscript(key:String)

```
public subscript(key:String)->JSON {
     switch _value {
     case let err as NSError:
         return self
     case let dic as NSDictionary:
         if let val:AnyObject = dic[key] { return JSON(val) }
         return JSON(NSError(
             domain:"JSONErrorDomain", code:404, userInfo:[
                 NSLocalizedDescriptionKey:
                 "[¥"¥(key)¥"] not found"
             ]))
     default:
         return JSON(NSError(
             domain:"JSONErrorDomain", code:500, userInfo:[
                 NSLocalizedDescriptionKey: "not an object"
             ]))
        }
 }
```

▼図2　AnyObjectの実行例



▼図3　Anyの実行例



まとめると次のようになるでしょう。

・AnyObjectは、Objective-Cで書かれたフレームワークの連携においてのみ使う
・Anyは使わない
(複数の型を受け付けるコードには、総称関数とプロトコルを用いる)

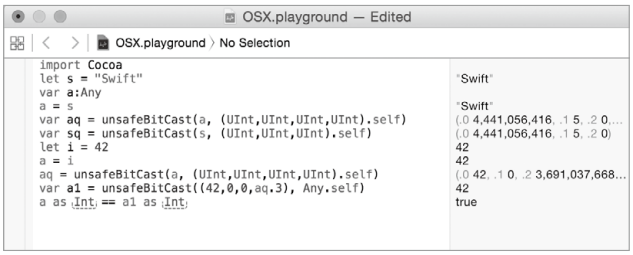
続きは次号

今回はSwiftからObjective-Cのフレームワークを用いる例としてSwift-JSONを紹介し、AnyObjectとAnyの違いを垣間見ました。次回はXcodeでCおよびObjective-Cのコードと Swiftのコードを同一のプロジェクトで連携する例を見ていくことにします。SD

▼リスト7　unsafeBitCastでAnyの動きを調べる

```
import Cocoa
let s = "Swift"
var a:Any
a = s
var aq = unsafeBitCast(a, (UInt,UInt,UInt,UInt).self)
var sq = unsafeBitCast(s, (UInt,UInt,UInt).self)
let i = 42
a = i
aq = unsafeBitCast(a, (UInt,UInt,UInt,UInt).self)
var a1 = unsafeBitCast((42,0,0,aq.3), Any.self)
a as Int == a1 as Int
```

▼図4　リスト7の実行例



Sphinxで始めるドキュメント作成術

川本 安武 KAWAMOTO Yasutake Twitter @togakushi

第2回 議事録を書こう(前編)——reSTの書き方、HTML変換の基本

議事録を題材にreSTを学ぶ意義

前回は誌面の都合もあり、インストール手順の紹介までででした。今回からは、実際の例を踏まえてreStructuredText(以下、reST)の文法やSphinxの基本的な使い方、ハマりどころを紹介していきます。

今回の記事で取り扱うドキュメントは「議事録」です。議事録は、議題にあがったテーマ(タイトル)、開催日時や場所、参加者といったドキュメントに付随する情報があり、内容が箇条書きになることが多いため、reSTの文法を学ぶ題材として非常に適しています。また、日常的に会議や打ち合わせなどで議事録を作成する機会も多く、Sphinxでドキュメントを作成するきっかけにできます。社内に提出する議事録のフォーマットが細かく決まっておらず、テキストで提出するような場合であれば、変換前のreSTをそのまま提出できるのではないのでしょうか。

Sphinxで変換できるフォーマットで一番使い勝手の良いHTMLを対象に、reSTの基本、Sphinxの基本を前後編の2回に分けて説明します。なお、本稿で扱うSphinxのバージョンは1.3です。

「プロジェクト」の作成

Sphinxでドキュメントを作成するためには、まず「プロジェクト」を作る必要があります。プロジェクトの作成はコマンドラインからの作業になるので、Windowsの場合はコマンドプロンプト、MacやLinuxの場合は仮想端末(ターミナルなど)を起動させます。

プロジェクトとは、いくつかの設定ファイルなどが配置された、ドキュメントを保存するための専用ディレクトリです。配置するファイルなどは、Sphinxに含まれる「sphinx-quickstart」というコマンドで作成します。

Sphinx1.3から非対話モードオプション(-q)が追加され、必要最低限の設定だけを指定してプロジェクトを作成できます(**図1**)。必須のオプションは、プロジェクト名を指定する-p、ドキュメントの製作者(Author name(s))を指定す

▼図1 sphinx-quickstart(非対話モード)の実行例

```
$ sphinx-quickstart -q -p project_name -a kawamoto -v 1.0 project_dir
Creating file project_dir/conf.py.
Creating file project_dir/index.rst.
Creating file project_dir/Makefile.
Creating file project_dir/make.bat.

Finished: An initial directory structure has been created.
  (... 以下略 ...)
```

る-a、プロジェクトのバージョンを指定する-vの3つです[注1]。

プロジェクトは引数で指定されたディレクトリに作成されます。指定されたディレクトリが存在しない場合は、ディレクトリが作成されます。省略されている場合はカレントディレクトリとなります。プロジェクトを作成するディレクトリは空である必要があります[注2]。

これでプロジェクトが完成します。

最初の「make html」

sphinx-quickstartの実行が完了すると、設定ファイルとひな形のindex.rstが生成されます（**図2**）。さっそくこのindex.rstをHTMLに変換してみましょう。

フォーマット変換にはmakeコマンドを利用します。利用している環境にmakeコマンドがなければ、yumやapt-getなどでインストールしてください。Windows環境の場合は、sphinx-quickstart実行時に生成されるバッチファイル（make.bat）を利用します。

フォーマット変換もコマンドラインからの作業になります。まずカレントディレクトリをプロジェクトのルートディレクトリ（Makefileやmake.batが存在するディレクトリ）に移動させます。ルートディレクトリで「`make html`」を実行すると、reSTがHTMLに変換されます（**図3、4**）。このときの画面には、進捗状況や警告などが表示されます。警告はreSTの書式に合っていない場合などに発生します。

Sphinxはフォーマット変換を行う際、最初にすべてのreSTファイルを読み込みます。読み込まれたreSTファイルは可能な限り変換が行われますが、警告となった部分はreSTに記述したテキストがそのまま出力されたり、該当個所が抜け落ちたりします。エラーとなった場合は変換処理が中断されます。

図4ではメニューなどが英語表記ですが、これは設定で日本語に変更できます（後述）。

注1）Sphinx1.3の非対話モードのオプションに日本語を指定するとエラーになる不具合があります。この問題は1.3.1で修正されています。

注2）Sphinx1.3.1でsphinx-quickstartが生成するファイル、ディレクトリが存在する場合のみプロジェクトの作成がエラーとなるように修正されました（それ以外のファイル、ディレクトリがあってもエラーにはなりません）。

▼図2　生成されるファイルとディレクトリ

```
project_dir/
    ├── _build/
    ├── _static/
    ├── _templates/
    ├── conf.py
    ├── index.rst
    ├── make.bat
    └── Makefile
```

▼図4　変換されたHTML（Firefoxで表示）

▼図3　make実行の様子

```
$ make html
sphinx-build -b html -d _build/doctrees   . _build/html
Running Sphinx v1.3
making output directory...
  (...中略...)
build succeeded.

Build finished. The HTML pages are in _build/html.
```

補足

実際のフォーマット変換は、Sphinxに含まれる「sphinx-build」というプログラムで行われます。sphinx-buildには、変換元のreSTファイルが保存されているディレクトリ、出力先のディレクトリを指定する必要があるのですが、これらのディレクトリはプロジェクト内で変わることはありません。出力先のディレクトリなどは、プロジェクト作成時に決まるのでsphinx-quickstart実行時にMakefile、make.batに埋め込まれます。利用者はsphinx-buildに必要なオプションを意識せずに「`make html`」と実行するだけで、フォーマット変換が行えます。

設定を変更する

sphinx-quickstartで設定したものはすべて「conf.py」というファイルに記述されています。内容を確認すると「**設定項目 = 設定値**」のように並んでいることがわかります。conf.pyは、拡張子からもわかるようにPythonで記述されたものです。マルチバイトを含む設定値は「**u'日本語'**」のようにクォートの前に「u」を付ける必要があります。「u」を指定していない場合、文字化けの原因となります。

設定できる内容は公式ドキュメント[注3]にまとまっていますので、そちらを参照してください。

先ほど変換した**図4**のHTMLはメニューなどが英語になっていましたので、試しにこれらを日本語になるよう設定を変更してみましょう。

メニューなどを日本語に変更するには、conf.pyの65行目にある「language」を設定します。デフォルトでは設定値が省略されており、「en」と同じになっています。languageを「ja」と変更することで、日本語に変更できます。

```
変更前
  language = None
変更後
  language = 'ja'
```

conf.pyの変更を反映させるために、再度`make html`を実行します。すると**図5**のようなHTMLに変換されます。

英語、日本語のほかに、ドイツ語やベトナム語など38言語に対応しています。設定できる値は公式ドキュメント[注4]を参照してください。

議事録を書こう

ここからは、実際に議事録をreSTで書いてSphinxでHTMLに変換していきます。サンプルの議事録は**リスト1**のものを使います。

リスト1で使用されているreSTの書式について、使用頻度の高いものから順に説明します。

reSTを保存するテキストファイルのエンコードは、一般的にUTF-8で記述します。そのほかのエンコードで記述する場合は、conf.pyの「source_encoding」を変更してください。

リスト1のreSTはSphinxで変換すると、**図6**のようなHTMLになります。

ドキュメントのスタート地点を作成する

Sphinxでは、どのドキュメントをプロジェク

注3）http://sphinx-doc.org/config.html
http://docs.sphinx-users.jp/config.html

注4）http://sphinx-doc.org/config.html#options-for-internationalization
http://docs.sphinx-users.jp/config.html#options-for-internationalization

▼図5　日本語に変更されたHTML（Firefoxで表示）

トの最初のページにするかを決める必要があります。このドキュメントを「マスタードキュメント」と呼びます。sphinx-quickstartのデフォルトの設定値は「index」となっており、この名前のひな形ファイル（index.rst）が生成されます。

プロジェクト内に複数のreSTがある場合は、マスタードキュメントからたどれる必要があります。マスタードキュメントと複数のreSTをつなぐ方法は次回で紹介します。

▼図6　sample.html

▼リスト1　sample.rst

```
.. _meeting-0630:                                          ⑥

==============================
Sphinxサイト ミーティング 6/30                              ①
==============================
:日時: 2000/06/30 10:00 - 12:00
:参加者:
  shimizukawa, tk0miya, usaturn, r_rudi


進捗状況について                                            ①
================
まず進捗状況の共有を行いました。
前回ミーティング :ref:`meeting-0513` からの進捗確認。        ⑦

* サイト概要: 未着手
* Sphinxの紹介: 大まかに完了。 *肉付けと見直しが必要*       ③   ②
* インストールページ: **完了**                               ③

.. 公式ドキュメント翻訳については省略

検討課題                                                    ①
---------
1. Sphinxの紹介で、以下の絵のような、Sphinxの全体像を
   表すイメージ図が必要

  .. figure:: sphinx-flow.png                               ④
     :width: 400

     入力から出力までの全体像、名称                          ②

2. sphinx-doc.org_ の紹介とリンクを追加しよう               ⑨

3. `進捗状況について`_ で確認したような進捗を自動的に       ⑤
   確認する方法

議事録に補足があればbitbucketの
`ここ <https://bitbucket.org/user/path>`__ でコメントを     ⑧
付けてください。

.. _sphinx-doc.org: http://sphinx-doc.org/                  ⑨
```

reSTのインデントと段落

reSTではインデントが重要な意味を持っています。連続する同じ高さのインデントは、1つのブロックとして扱われ、空行を1行以上入れると段落が分かれます。インデントの高さを変える場合は、次のインデントとの間に1行以上の空行を開ける必要があります。

段落と行の扱いは、変換先のフォーマットに依存します。HTMLの場合、段落はpタグで囲まれ、改行は無視され1行にまとめられます。

```
reSTの記述
  文章1
  文章2

  文章3
変換後のHTML
  <p>文章1 文章2</p>
  <p>文章3</p>
```

見出しを付ける(セクション)

ドキュメントには必ず見出しを付ける必要があり、最初の見出しはページのタイトルになります。見出しは、見出しにしたいテキストの前後の行、または後の行だけに同じ記号で線を引けば、その行が見出しとなります(**リスト1ー①**)。

```
見出しの記述例
  レベル1の見出し
  ==============

  ---------------
  レベル2の見出し
  ---------------

  レベル1の見出し
  ==============
```

見出しにするテキストは1行で書く必要があり、使用する記号はそのテキストの長さ以上の数が必要です。日本語などのマルチバイト文字では1文字あたり2個の記号が必要になります。見出しに使用する記号がテキストより短い場合はSphinxがフォーマット変換する際に警告を発します(警告されますがドキュメントは正しく変換され、見出しも正常に付きます)。

見出しに使用する記号は、「=」や「-」、「+」や「#」などの記号が使用できます。同じreSTファイル内で使用している見出しの付け方を変えると、自動的にレベルが1段下がります。一度使った記号は、同じreST内では常に同じ見出しレベルになります。reSTでは見出しとして使用できるすべてのパターンを許容しますが、変換するフォーマットによっては見出しレベルに上限があるので注意してください。

見出しに使用する記号がどのレベルの見出しになるかは、1つのファイル内で出現した順番で決まります。このため、ある記号がどのレベルになるかはファイルごとに決まります。プロジェクト内で使用する見出しの記号は統一しておいたほうが、混乱が少なくて済みます。

見出しに使用できる記号については公式ドキュメント[注5]を参照してください。

議事録を書く場合、会議名をレベル1に、アジェンダをレベル2に、さらに細かく分けるならレベル3以上を使用すると良いでしょう。

箇条書きをする(リスト)

reSTでは「*」、「-」、「+」の記号が箇条書きになります(**リスト1ー②**)。記号のあとにスペースを1つ挟んで文章を記述します。

```
箇条書きの記述例
  * レベル1-1
  * レベル1-2

    * レベル2

  * レベル1-3
```

番号付き箇条書きは、使用する番号の種類の後にピリオドを付けてスペースを挟みます。使用できる番号の種類は、アラビア数字(「1.」)、

注5) http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#id1
http://docutils.sphinx-users.jp/docutils/docs/ref/rst/restructuredtext.html#id2

ローマ数字（「I.」「i.」）、アルファベット（「A.」「a.」）の3種類です。

箇条書きを入れ子にするには、インデントの高さを変える必要があります（**図7**）。

入れ子になった箇条書きのインデントの高さは、親の箇条書きの文章の先頭と同じ高さに合わせる必要があります。

番号付き箇条書きの記号を「#.」にすると、直前に使用された同じ種類の番号が連番で自動的に振られます。最初の「#.」は「1.」と同じ意味を持ち、「5.」の後の「#.」は「6.」と同じ意味です。

記号は混ぜて使用することもできますが、同じレベルで異なる記号が使用されている個所で段落が別れてしまいます。

文字の修飾（インラインマークアップ）

文字を特定の記号で挟むと太字や斜体などの修飾ができます（**リスト1ー③**）。

- アスタリスク1つ：強調（斜体）
  記述例）＊テキスト＊
- アスタリスク2つ：強い強調（太字）
  記述例）＊＊テキスト＊＊
- バッククォート：コードサンプル（固定長）
  記述例）｀｀テキスト｀｀

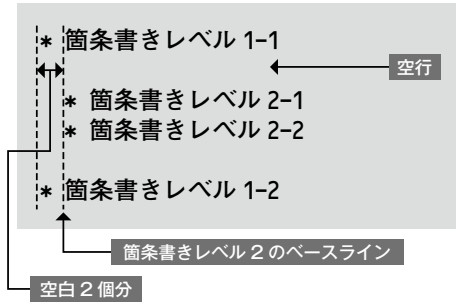
また、次のような制限があります。

- 入れ子にできない（固定長を太字にできない）
- 記号で挟まれた中のテキストの最初、最後にスペースを入れられない
- 周囲のテキストとは、テキスト以外の文字（スペース、カッコなど）で区切る必要がある

画像の差し込み（「figure」ディレクティブ）

ドキュメントに画像ファイルを差し込むには「figure」というディレクティブを使用します（**リスト1ー④**）。ディレクティブの詳細については、次回に説明します。今回は画像を差し込むには次のように書くとだけ覚えておいてください。

▼図7　箇条書きを入れ子にするときのルール



ベースラインが空白２個分なのは「＊」の場合。
「1.」の場合は空白３個。「10.」の場合は空白４個となる。
そのため、“空白２個”のように固定数で考えるとハマることがある。

figureディレクティブの記述例

```
.. figure:: 画像ファイル名
```

ドキュメントのリンク

同じファイル内の見出しには、次のようにしてリンクを張ることができます（**リスト1ー⑤**）。

```
｀見出し｀_
```

バッククォートで囲み、最後にアンダースコアを付け加えます。文字の修飾の制限と同じく、周囲のテキストと区切って記述する必要があります。該当する見出しが存在しない場合は、バッククォートとアンダースコアがそのまま表示されます（リンクにはなりますが、参照先がありません）。

また、ドキュメント間（異なるreSTファイル）へのリンクはラベルを使います。「ドット（.）2つ＋スペース＋アンダースコア（_）＋ラベル名＋コロン（:）」として定義します（**リスト1ー⑥**）。見出しの直前でラベルを定義すると、ラベル参照時に見出しの内容で表示されます。見出しが変更された場合は、`make html`を実行したときに置き換えられます。

ラベルの定義

```
.. _ラベル名:
```

ラベルの参照（2通り）
```
:ref:`ラベル名`  ←リスト1ー⑦
:ref:`表示するテキスト <ラベル名>`
```

ラベルを使用したリンクは同じドキュメント内でも使用できます。存在しないラベルを参照した場合、フォーマット変換時に警告になります。HTMLには指定したラベル名（または表示するテキスト）がそのまま表示されます。

使用するラベルは、プロジェクト内で重複があってはなりません。

ラベルを使用せず、ファイルパスを指定して別のreSTにリンクすることもできます。リンクに表示するテキストが指定されていない場合は、リンク先のreSTの最初の見出しが表示されます。

別のドキュメントへのリンク（2通り）
```
:doc:`リンク先reSTのファイルパス`
:doc:`表示するテキスト <リンク先reSTのファイルパス（拡張子なし）>`
```

ハイパーリンク

reST内に記述されているURLやメールアドレスは、HTMLでは自動的にリンクになります。表示させるテキストを変更する場合は、次のようにして指定します（**リスト1ー⑧**）。

表示するテキストを変更する
```
`ユーザ会 <http://sphinx-users.jp/>`__
```

「ユーザ会」がhttp://sphinx-users.jp/へのリンクとして表示されます。最後にあるアンダースコア（2つ）を忘れないようにしてください。

ハイパーリンクターゲットにもラベルを付けることができます（**リスト1ー⑨**）。ラベルの定義は次のように1行で記述します。

ラベルの定義
```
.. _ラベル名: ターゲットのURL
```
ラベルを使用したハイパーリンクの記述例
```
.. _Sphinx-users.jp: http://sphinx-users.jp/
```
ラベルの参照
```
Sphinx-users.jp_
```

参照は定義したラベル名の後にアンダースコアを付けるだけになります。ラベルの定義は同じファイルの中であれば記述する個所に決まりはありません。

◆ ◆ ◆

次回は、今回紹介しきれなかった書式の解説と、複数のreSTファイルを扱うプロジェクトの作り方について紹介します。SD

COLUMN

定義と参照

ラベルやハイパーリンクターゲットなど、別の場所で定義するものはアンダースコアが前に付きます。定義されたものを文中で参照する場合、アンダースコアが後に付きます。

参照で使用するテキストが英数字と一部の記号のみで構成されている場合、バックォートを省略できます。

COLUMN

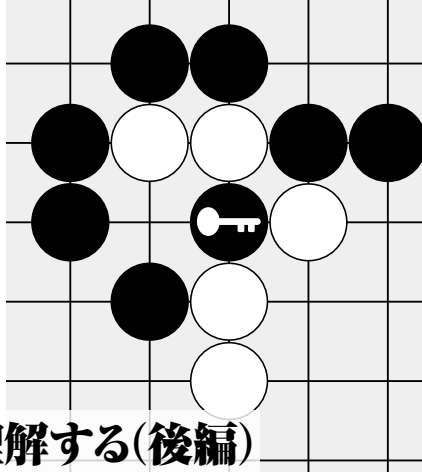
reStructuredText

reSTはロールとディレクティブを拡張可能なため、Sphinxはそのしくみを利用して拡張ロール、ディレクティブを提供しています（ロールとディレクティブについては次回以降で紹介します）。追加されている拡張の多くはSphinxがドキュメントを管理するためのものです。reSTのマークアップの基本は、docutilsのドキュメント[注A]も参考になります。

注A）http://docutils.sourceforge.net/rst.html
http://docutils.sphinx-users.jp/

セキュリティ実践の基本定石

すずきひろのぶ
suzuki.hironobu@gmail.com



みんなでもう一度見つめなおそう

【第二十回】GnuPGを通して暗号技術を理解する(後編)

前回は、Gnu Privacy Guard (GnuPG、GPG) プロジェクトやOpenPGP仕様がどのような経緯で作られ、インターネットやOSSの世界でどのような役割を果たしているのかを説明しました。今回は、OpenPGPの実装の1つ「gnupg」の使い方を説明するとともに、公開鍵の認証手続きに潜んでいる課題についても取り上げます。

gnupg

gnupg (gpg) はOpenPGP仕様に基づいた暗号ツールです。なお、ここでの具体的な説明は、Ubuntu 14.04 LTS上で提供し動作しているgpg 1.4.16をベースに行います(**図1**)。今回は最初の一歩を踏み出すための使い方と電子メールで使う範囲で説明します。

共通鍵暗号

まずは実際に使ってみましょう。ファイルfoo.txtを共通鍵暗号で暗号化してみます。"-c"オプションを指定すると共通鍵暗号で暗号化できます。

```
$ cat foo.txt ←ファイルの内容を表示
This is a text file for Software Design.
```

```
$ gpg -c foo.txt ←暗号化する
Enter passphrase: ←パスフレーズを入力
```

パスフレーズの入力は2度求められます。入力するとfoo.txt.gpgが作成されます。これはバイナリ形式のサイファーテキスト(暗号文)です(**図2の①**)。"-a"オプションを加えるとバイナリをアスキーアーマー(Ascii Armor)と呼ぶアスキーコードのテキストフォーマット(正確にはCRC24が付加されているradix-64)に変換してくれます(**図2の②**)。

なお、gpg2のデフォルトではピン入力画面が現れて入力を要求します。GNOME環境では**図3**のようなウィンドウが現れます。サーバのようなテキスト環境だとこのようなウィンドウ環境がインストールされていないので、**図4**のようなエラーが発生する場合があります。

この場合、アスキー文字だけの画面環境(curses)

◆**図1　gpgのバージョンなどの情報**

```
$ gpg --version
gpg (GnuPG) 1.4.16
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Home: ~/.gnupg
Supported algorithms:
Pubkey: RSA, RSA-E, RSA-S, ELG-E, DSA
Cipher: IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH,
        CAMELLIA128, CAMELLIA192, CAMELLIA256
Hash: MD5, SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
Compression: Uncompressed, ZIP, ZLIB, BZIP2
```

◆図2 gpgで暗号化したサイファーテキスト

```
$ ls -l foo.txt*
-rw-rw-r-- 1 hironobu hironobu  41 Feb 16 18:08 foo.txt
-rw-r--r-- 1 hironobu hironobu 238 Feb 16 18:10 foo.txt.asc ←②
-rw-r--r-- 1 hironobu hironobu 117 Feb 16 18:12 foo.txt.gpg ←①

$ cat foo.txt.asc ←アスキー化されたサイファーテキストの内容を表示
-----BEGIN PGP MESSAGE-----
Version: GnuPG v1

jA0EBwMCZtejYCT+izBg0mQB5Y+WJxAm/9G19XBXRrzzsLjib33h9nVb3uOwarHp
lhwdtZNO02E7BjsGvKgoSBniXKfq9EDKjsSBjrDHKZF48hh9bl2HVeQvdkbyiAll
yA8LCnODScU54oxtksKIFNWsfabF
=TdVU
-----END PGP MESSAGE-----
```

◆図3 パスフレーズのエントリー・ウィンドウ

◆図4 パスフレーズの入力でウィンドウ環境がない場合のエラー

```
/usr/bin/pinentry: line 22: xprop: command not found
Please install pinentry-gui
```

のインターフェースを使うことで解決できます。

```
$ export PINENTRY_BINARY=/usr/bin/pinentry-curses
```

公開鍵暗号

鍵ペアを生成する

gpgで公開鍵暗号を使うために、自分の公開鍵と秘密鍵のペアを生成します(**図5**)。鍵生成のためにシステムのエントロピーを消費します(コラム参照)。

これで公開鍵と秘密鍵ができました。デフォルトでは~/.gnupgの下に必要なファイルが作成されます(**図6**)。公開鍵はpubring.gpgに入っており、秘密鍵はsecring.gpgに入っています。trustdb.gpgは相手の公開鍵の信頼度が入っています。random_seedは内部での乱数生成に使用されます。

ちなみに、gpgで鍵を生成するときに入力する名前や電子メールのアドレスは、あくまでもユーザが

◉gpgにエントロピーを供給する方法

エントロピーとは乱雑さのことで、これは疑似乱数生成に必要な情報です。鍵を生成する際にシステムからのエントロピーの供給が少ないと、エントロピーの供給を待つので公開鍵と秘密鍵の生成に時間がかかります。システムのエントロピーは、ハードディスクへのアクセスやネットワークへのアクセスによる割り込みが供給源になっています。

“Not enough random bytes available. .... the OS a chance to collect more entropy! ”というメッセージが現れた場合、デスクトップ環境ならキーボードやマウスを操作したり、ブラウザでネットサーフィンをしたりしてみてください。

サーバなどでデスクトップ環境がない場合、ハードディスクへのアクセスでエントロピーを増やせます。**図A**のように大量のファイルにアクセスしてみてください。

利用しているハードウェアが乱数生成器(IntelのIvy Bridgeアーキテクチャの CPUなど)を持っており、ハードウェア乱数生成デバイス/dev/hwrngがある場合には、rngdコマンドも非常に有効です。

ネット上に、**rngd -f -r /dev/urandom**とするとgpgの鍵生成が早く終了すると書いているブログがありました。これは、/dev/randomからのエントロピーをシード(種)にして生成した疑似乱数系列を、再度フィードバックしているだけですので、タコが自分の足を食べるようなものです。確かに鍵の生成は速くなりますが、本来必要なエントロピーを十分確保しているわけではないことを理解してください。

◆図A 大量ファイルアクセスによりエントロピーを増やす一例

```
$ find /var/ /usr/share/ /lib -type f -print0 | xargs -0 sha256sum  > /dev/null
```

入力できる任意の文字列にすぎません。ですので、自分の名前をいっさい出さない偽名でもあっても生成できます(図7)。

◆図5　公開鍵と秘密鍵のペアを生成する手順

```
$ gpg --gen-key
gpg (GnuPG) 1.4.12; Copyright (C) 2012 Free Software Foundation, Inc.
(... 略 ...)
gpg: directory `/home/hironobu/.gnupg' created
gpg: new configuration file `/home/hironobu/.gnupg/gpg.conf' created
gpg: WARNING: options in `/home/hironobu/.gnupg/gpg.conf' are not yet active during this run
gpg: keyring `/home/hironobu/.gnupg/secring.gpg' created
gpg: keyring `/home/hironobu/.gnupg/pubring.gpg' created
Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1 ←RSAを選択
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)  2048 ←RSAの鍵長ビット数を入力
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)  0 ←鍵は無期限に使えるようにした
Key does not expire at all
Is this correct? (y/N) y ←確認

You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Hironobu SUZUKI ←自分の名前を入力
Email address: suzuki.hironobu@gmail.com ←自分のメールアドレスを入力
Comment: Author of Step by Step Security ←コメントを入力
You selected this USER-ID:
    "Hironobu SUZUKI (Author of Step by Step Security) <suzuki.hironobu@gmail.com>"
Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O ←確認してOKの0を入力
You need a Passphrase to protect your secret key.
Enter passphrase:  ←パスフレーズを入力

We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
(... 略 ...)

gpg: /home/hironobu/.gnupg/trustdb.gpg: trustdb created
gpg: key 8FE36F99 marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
pub   2048R/8FE36F99 2015-02-19
      Key fingerprint = 426B C482 DA1A 57E9 4CF9  F85A 8E5D 3758 8FE3 6F99
uid                  Hironobu SUZUKI (Author of Step by Step Security) <suzuki.hironobu@gmail.com>
sub   2048R/47A7E3D7 2015-02-19
```

ボブはアリスに公開鍵を渡す

前編で取り上げたボブとアリスの例で考えてみましょう。ボブは公開鍵と秘密鍵を生成したなら、自分の公開鍵を取り出しアリスに送ります。**図8**は公開鍵のIDを指定して取り出す方法です。"--armor"はアスキーコードで出力するためのオプションです。

ボブから電子メールや鍵ファイルのダウンロードなど何かの手段で公開鍵を受け取ったアリスは、まだgpgで自分の鍵などを作っていなくても、ボブの公開鍵を取り入れることができます(**図9**)。

◆図6　~/.gnupg下に作成されたファイル

```
$ cd ~/.gnupg
$ ls -al
total 40
drwx------  2 hironobu hironobu 4096 Feb 20 00:39 .
drwxr-xr-x 33 hironobu hironobu 4096 Feb 20 00:32 ..
-rw-------  1 hironobu hironobu 9188 Feb 20 00:32 gpg.conf
-rw-------  1 hironobu hironobu 1236 Feb 20 00:39 pubring.gpg
-rw-------  1 hironobu hironobu 1236 Feb 20 00:39 pubring.gpg~
-rw-------  1 hironobu hironobu  600 Feb 20 00:39 random_seed
-rw-------  1 hironobu hironobu 2613 Feb 20 00:39 secring.gpg
-rw-------  1 hironobu hironobu 1280 Feb 20 00:39 trustdb.gpg
```

アリスはボブに秘密の情報を送ろうとするが……

アリスはalicemessage.txtというファイルを作り、その内容をボブの公開鍵でサイファーテキストに変換するとします(**図10**)。オプション"-ea"は、"--encrypt"と"--armor"を一緒に指定した省略形もの、つまり暗号化し、出力をアスキーコード化するという指定になります。オプション"-r"は、"--recipient"の省略形で受信者の指定をします。ここでは名前ではなく鍵IDを直接指定しています。

すると「ボブの鍵としているものが、本当にボブの鍵であるかの保証がまだされていない」という警告メッセージが出てきます。

この時点では、この鍵がボブの鍵かどうかまだ保証はありません。途中でボブの鍵とすり替えた別の鍵、あるいは誰かが最初からボブのふりをして送ってきた鍵かもしれません。まだ確認がとれていない以上、この鍵は使うべきではありません。では、どうすべきなのでしょうか。

◆図7　偽名(Bob Smith)で生成した例(gpg --gen-keyを実行したときのメッセージ)

```
pub   2048R/707B7C34 2015-02-19
      Key fingerprint = 34A3 06E9 3AAE A6C8 D7F8  6DA2 C606 31E6 707B 7C34
uid                  Bob Smith ←偽名で生成された
sub   2048R/052FC614 2015-02-19
```

◆図8　ボブは生成した公開鍵を取り出す

```
$ gpg --export --armor 0x707B7C34
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1

mQENBFTmGB0BCADQfd9cZpdL81TcaWcKaRwTKmUkw+pimDCaubq6CO7tTO8r4f+m
(... 略 ...)
nZw4ZZi1FXIQyddbeiSMFr+EiC861+3abfgtI+qM/u0yc4saU4vDYkZdseKHac5N
JVatV1ymDjM=
=V3dE
-----END PGP PUBLIC KEY BLOCK-----
```

◆図9　アリスの環境で、初めてgpgを使ってボブの公開鍵を取り入れる

```
$ gpg --import publickey.asc
gpg: directory `/home/alice/.gnupg' created
gpg: new configuration file `/home/alice/.gnupg/gpg.conf' created
gpg: WARNING: options in `/home/alice/.gnupg/gpg.conf' are not yet active during this run
gpg: keyring `/home/alice/.gnupg/secring.gpg' created
gpg: keyring `/home/alice/.gnupg/pubring.gpg' created
gpg: /home/alice/.gnupg/trustdb.gpg: trustdb created
gpg: key 707B7C34: public key "Bob Smith" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

公開鍵の認証は実は考え方が難しい

ボブとアリスが長年にわたる付き合いをしていてお互いを深く知っているなら、ボブがなりすましかどうか、あらためて確認する必要もないかと思われます。また、お互いのことをあまり知らなくても、公的に確認できる写真つきの証明書(パスポートや運転免許証)で確認できるでしょう。お互いを確認し、主キー(Primary Key)のハッシュ値(Fingerprint)を確認しあうことで、公開鍵の認証とします。双方向で直接確認しあえば、この方法が最も保証する度合いが高いと言えます。

名刺などに載せておいて交換するのも1つの手かもしれませんが、お互いを確実に確認しあうよりも信頼性は低くなります。人が人を確認するというのは、人間の錯誤による誤りの可能性が必ず入ってきます。だまそうと巧みにしかけられれば、人は簡単にだまされてしまいます。お互いに知っている同士が直接会って確認しあう方法ですら、最初から準備万端でだまそうとすれば、だませるでしょう。

ボブとアリスは直接会わずに、権威のある認証局が2人をそれぞれ認証し、ボブもアリスもその認証局を信じる。この方法は基本的には、公的な証明書を使ってお互いを認証しあうのと同じぐらいに信頼できるでしょう。しかし、認証局自体が信頼できない場合、この方法も確実とは言えません。実際に、SSLの認証局自体が不正アクセスを受けて証明書の信頼性が保てないという事例や、認証局がだまされて不正な証明書を発行した事例もあります。

PGP(OpenPGP)はWeb of Trustと呼ぶ、簡単に言えば「友達の友達は友達である」という認証方法を採っているという説明があります。これは1992年にZimmermannが書いたpgpバージョン 2.0のマニュアルの中に出てきた考え方です。しかし、筆者は、このWeb of Trustが先述したほかの方法より、ずっと信頼性が低い、あるいは、逆にだまされやすい方法だと指摘したいと思います。

アリスとマルロイがお互いを認証したとしても、アリスはマルロイのことを本当に信頼して認証したのではなく、儀礼的に認証しただけかもしれません。なぜならば、お互いが会って認証する方法は技術的な認証ではなく、人間関係もかかわってくる極めてソーシャルなものだからです。ボブやアリスがどんな状況で認証したかという情報がないままマルロイを認証してしまっては、彼が何者なのかまったくわかりません。これはアリスだけではなく、キャサリンがマルロイを認証する場合も本質的には同じです。

筆者も昔は「Web of Trustの考え方も有効であるかもしれない」と考えていた時期がありました。しかし現在では、Web of Trustという方法に明確な根拠はないにもかかわらず、有効であると思われていることに危惧を覚えます。あくまでも、自分が自分の責任において相手を直接認証するのが基本であり、認証局を使うのは次善の策と考えてください。

ボブは電子署名付きデータを送る

ボブの公開鍵がすでにボブのものであると認証さ

◆図10　アリスはボブの公開鍵でファイルを暗号化する

```
$ cat alicemessage.txt ←ファイルの内容を表示
This is a secret message from Alice.
$ gpg -ea -r 0x707B7C34 alicemessage.txt ←暗号化する
gpg: 052FC614: There is no assurance this key belongs to the named user

pub  2048R/052FC614 2015-02-19 Bob Smith
 Primary key fingerprint: 34A3 06E9 3AAE A6C8 D7F8  6DA2 C606 31E6 707B 7C34
      Subkey fingerprint: 6DB6 4772 692E 8701 EB16  9A30 F25D C523 052F C614

It is NOT certain that the key belongs to the person named ←警告メッセージ
in the user ID.  If you *really* know what you are doing,
you may answer the next question with yes.

Use this key anyway? (y/N)
```

れているとします。その前提において、ボブがデータに電子署名を付ければ、アリスはデータが改ざんされているのか否かを確認することができます。

たぶん説明のしかたによる行き違いなのでしょうが、「電子署名があれば改ざんできない」ということで、「電子署名の付いたデータは常に100％有効に使えるものである」と思われる方がいます。

正確には、「データが改ざんされていることがわかる」あるいは「データが正規のものではないことがわかる」ので、「データを捨てることができる」ということを意味します。データが改ざんされてしまった場合、改ざん前の元のデータがない限り永久に正しいデータは失われてしまいます。その代わりだまされることはありません。

さて、ボブのデータ(bobtext.txt)にボブの署名を付けてアリスに送る準備をしましょう(**図11**)。オプション“-u”は“--local-user”の省略形で、ここではボブの鍵0x707B7C34を使うことを明示しています。もし明示しない場合、デフォルトのユーザ鍵が使われます。“-sa”は“--sign”と“--armor”の両方の省略形の組み合わせです。電子署名を付け、出力をアスキー化します。

本文をアスキー化し、電子署名部分を切り分けたいときは、“--clear-sign”を使います。これはメールへの添付やWebサイトに掲載するときなどに使われます。ただし注意してほしいのですが、インターネット上で日本語文字コードを扱っている場合、電子署名を行ったときの文字コードと、送付されたときの文字コードと、電子署名の検証を行ったときの文字コードが必ずしも一致しないことがよくあります。その点を考慮したうえでご利用ください。

さて、アリスはbobtext.txt.ascを受け取り、ボブによって電子署名がされているかどうかを確認します(**図12**)。

図12では、アリスはまだボブの公開鍵を信頼していないので、自分のキーサイン(公開鍵に対する署名)を付けていません。キーサインは相手の公開鍵に対して認証した、という意味合いを持ちます。アリスはすでに秘密鍵を生成して持っているとします。その状態で、ボブの公開鍵にアリスの電子署名を付けるには“--sign-key”を使います(**図13**)。

以降、ボブの電子署名の確認は**図14**のようになります。

◆図11　ボブはファイルに電子署名を付ける

```
$ cat bobtext.txt ←ファイルの内容を表示
I'm a boy.

$ gpg -u 0x707B7C34 -sa bobtext.txt ←ボブの鍵で電子署名を付ける
You need a passphrase to unlock the secret key for
user: "Bob Smith"
2048-bit RSA key, ID 707B7C34, created 2015-02-19

$ cat bobtext.txt.asc ←サイファーテキストの内容を表示
-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.4.12 (GNU/Linux)

owEBTAGz/pANAwACAcYGMeZwe3w0AawcYgtib2J0ZXh0LnR4dFTmP2ZJJ20gYSBi
(... 略 ...)
vUHiXxR6jyH3UMByYkcxwoPdrcEl6cDo0nv6Pgww78AJgnMmbbRnrPWpi3GHY446
0li/
=YWQA
-----END PGP MESSAGE-----
```

SHA1問題

ところで、gpg 1.4.12ではデフォルトのハッシュ関数がSHA1になっています。しかし、今日において大手ベンダはSSL証明書などのSHA1のサポートを終了していっている最中です。たとえば、Googleは2014年9月以降、Google

◆図12　アリスはボブの電子署名を検証する(初回)

```
$ gpg < bobtext.txt.asc
I'm a boy.
gpg: Signature made Fri 20 Feb 2015 04:54:14 AM JST using RSA key ID 707B7C34
gpg: Good signature from "Bob Smith" ←ボブの署名を確認できた
↓ただし、ボブの公開鍵が認証されていない旨の警告メッセージが出る
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 34A3 06E9 3AAE A6C8 D7F8  6DA2 C606 31E6 707B 7C34
```

Chromeに使われているSSLの証明書からSHA1は使っていません。

gpgでSHA1を回避したい場合、~/.gnupg/gpg.confに**リスト1**の記述を加えてください。これでSHA256がデフォルトで使われるようになります。

ボブが何者かはわからないけれど

アリスはボブのことを知らない(直接会ったことはない)とします。最初にボブから公開鍵が送られてきているので、その後の一連のボブから送られてくるデータが一貫してボブからのものであることは、電子署名を使えばわかります。途中で偽ボブからデータが送られてきても、正しい電子署名を付けられないので、見破ることができます。

これでアリスとボブは安全にやりとりができますが、前提条件があります。最初、ボブとアリスの関係は誰も知らず、かつボブがアリスに初めてコンタクトするタイミングを知らないという前提です。その前提があるならば、途中でボブのふりをしてアリスに偽データ送ることはできなくなります。

たとえば、匿名の相手と電子メールをやりとりしているとき、やりとりしている相手が同じ相手なのか、いつの間にか別人がなりすましているのかは、確認のとりようがありません。相手が別のメールアドレスに変更した場合はなおさらです。しかし、上記の方法を使うならば、匿名でも電子メールの相手が同一であることを確認することができます。

ただし、最初からボブとアリスの関係を知っていて監視していた場合は、中間者攻撃を行いアリスに偽ボブの公開鍵を渡すことができてしまうので、この方法は成り立ちません。

すべての人たちのための技術GPG

2回に渡りGPGとPGPの話題を取り上げました。使う使わないにかかわらず、このような技術の概要を理解し一度は試しておくというのは重要かと思います。このような情報セキュリティを支えるツールがフリーソフトウェアという形で提供されていることはさらに重要です。なぜならば、プライバシー保護や情報セキュリティのためのツールは、コンピュータやネットワークを使う人たちすべてに提供されなければならないからです。その意味においてGPGの存在はたいへん重要な意味を持っています。SD

◆図13　アリスはボブの公開鍵に自分の署名を付ける

```
$ gpg  --sign-key 0x707B7C34
pub  2048R/707B7C34  created: 2015-02-19  expires: never       usage: SC
                     trust: unknown       validity: unknown
sub  2048R/052FC614  created: 2015-02-19  expires: never       usage: E
[ unknown] (1). Bob Smith
 (... 略 ...)
Really sign? (y/N) y
```

◆図14　アリスはボブの電子署名を検証する(署名後)

```
$ gpg  --verify bobtext.txt.asc
gpg: Signature made Fri 20 Feb 2015 04:54:14 AM JST using RSA key ID 707B7C34
gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   1  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: depth: 1  valid:   1  signed:   0  trust: 1-, 0q, 0n, 0m, 0f, 0u
gpg: Good signature from "Bob Smith"
 ↑公開鍵が認証されていない旨の警告メッセージは出ない
```

◆リスト1　SHA256をデフォルトで使用する設定(~/.gnupg/gpg.conf)

```
personal-digest-preferences SHA256
cert-digest-algo SHA256
default-preference-list SHA512 SHA384 SHA256 SHA224 AES256 AES192 AES CAST5 ZLIB BZIP2 ZIP Uncompressed
```

目指せ！定時帰りのエンジニア！

Hinemosで学ぶジョブ管理超入門

㈱NTTデータ 基盤システム事業本部　眞野 将徳（まの まさのり）　manoms@nttdata.co.jp

運用自動化との格闘の日々もいよいよ大詰め。今回はジョブが時間どおりにきちんと起動・終了したかを判定・通知し、次の処理へつなげる「開始遅延」「終了遅延」について学びます。また、サーバ上でのジョブの同時実行数を制御する機能についても学び、よりトラブルに強いシステムを目指します。ラストでは藤井君に新たな展開も……。　イラスト（高野 涼香）

第8回　さらに高度にジョブを運用しよう

登場人物紹介

藤井　今年SI企業に入社した新人SE。運用自動化のために日々奮闘中。

上司　軽いノリで仕事を依頼してくるが、藤井君の成長を考えている。

定時先輩　藤井君の先輩社員。華麗に仕事をこなし定時に颯爽と帰っていく優秀な女性。

前回までのあらすじ

社内の「勤怠管理システムの運用」を上司から任された新人SEの藤井君。オープンソースソフトウェアの運用管理ツール「Hinemos」を使い、さまざまな困難に立ち向かいながら、運用自動化を進めてきました。そんなある日、藤井君はより良い運用のために必要なことに気がついたようです。

ジョブの遅延を判定する

藤井「すみません、定時先輩。勤怠管理システムの運用について相談があるんですが……」

定時「どうしたの？」

藤井「Hinemosに移行してから、勤怠管理システムで必要なバッチ処理が自動で行われるようになって、トラブルの発生もずいぶんと少なくなりましたよね」

定時「そうね、やっぱり手作業には限界があるから、ツールを使うと、定型的な作業をミスなく効率よく作業できるわね」

藤井「はい。もしバッチ処理が失敗しても、ジョブの異常終了を判定してメールで通知できるので、異常が発生したことに気づけるのも便利です」

定時「なら、ほかに何が気になるの？」

藤井「ちゃんとジョブが起動したり、終了したりしてくれる場合に通知されるのでいいのですが、もしジョブが起動されなかったり、終了しない場合には気づけないな、と思いまして」

上司「ほう、いいところに気づいたな。聞いた話だと、ほかのシステムの運用でバッチ処理が予定どおり終わらない、というのが問題になったことがあったらしいぞ」

藤井「やっぱりそういうことがあるんですね。そのときはどんな状況だったんですか」

上司「どうやら運用当初は問題なく処理できていたようなんだが、途中で実行する処理の増加や利用者の増加に伴って処理対象の量も増加していったらしい。さらに、そこではほかのジョブも同時平行で動いていて、そちらの動作状況の

影響も受けやすかったようだな」

藤井「それはたいへんですね。勤怠管理システムも社員数が増えたら当然処理する対象人数が増えるので、対策を打っておいたほうが良さそうです」

上司「とくに、処理に時間がかかっていることに気づけないことが一番の問題だったようだな。前の処理が終わらないせいで、想定以上に大量の処理が同時に動作することになり、最終的にリソースが枯渇してしまったらしい。異常終了のアラートが上がるまでそのことに気づかず、そのあとのリカバリがたいへんだったようだ。もっとも、そこはツールを使わずにcronでスクリプトを定期実行していたらしいがな」

藤井「手作業か……、以前の勤怠管理システムの運用と一緒ですね。でも今は運用管理ツールを使っているので何か対処ができると思います」

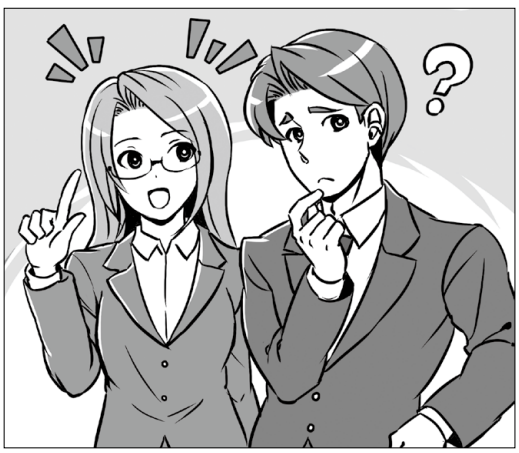
上司「よろしく頼んだぞ、定時さんもサポートしてあげてくれ」

開始遅延と終了遅延

藤井「とは言ったものの、本当にそんな機能あるのかな」

定時「運用管理ツールなら、ジョブが正常に動作しているか知る機能が用意されていることが多いわ。調べてみたら？」

藤井「うーん、そういえばまだジョブの設定ダイアログで使っていない設定があったような……。そうそうこれこれ、開始遅延だ。名前からしてもそれっぽい気がするし、マニュアルも調べてみよう」



開始遅延は、ジョブやジョブネットの起動が、想定よりも遅れた場合に通知やジョブの状態変更を行うことができる機能です。遅延の判定には、次の条件を設定できます。

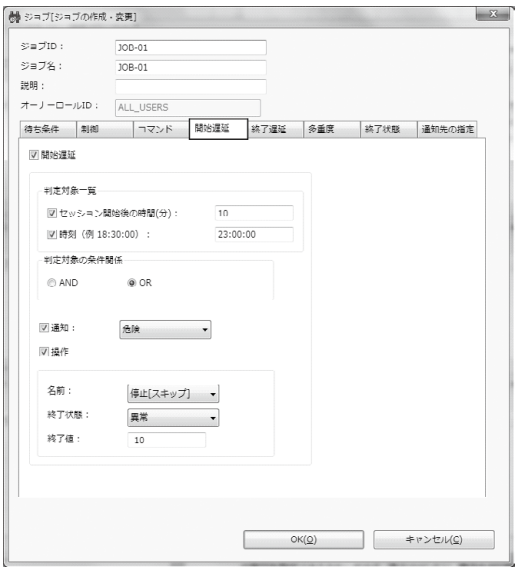
- 指定した時刻（例：23:00など）になっても特定のジョブが開始しない場合
- ジョブセッション開始後(スケジュールなどでジョブ全体が実行されてから)、○分後までにジョブが開始しない場合

遅延と判定された場合は、該当のジョブを「保留」状態や「スキップ」状態にできます（本連載2015年4月号「第7回」で説明）。

開始遅延はジョブやジョブネットに対して設定できます。ジョブに開始遅延を設定する場合は、ジョブ［ジョブの作成・変更］ダイアログの開始遅延タブで設定します（**図1**）。

開始遅延タブの開始遅延チェックボックスにチェックを入れると、該当のジョブの開始遅延判定が有効になります。判定対象一覧では、先述した時

▼図1　ジョブの開始が遅延していることを判定する



刻とジョブセッション開始後の時刻を指定できます。両方の条件を設定した場合には、条件同士の関係にANDかORを選択でき、両方の条件を満たした場合や片方の条件のみを満たした場合に、ジョブの開始が遅延していると判定できます。開始遅延と判定された場合には、通知によりユーザに開始遅延が発生したことを知らせるほか、ジョブの状態をスキップ状態や保留状態にして、そのあとのジョブの動作を変更できます。

藤井「遅延と判定された場合には通知をしてユーザに知らせることができるんですね。さっきの話だと、まず気づくことが重要ってことでしたので、これは助かりますね」

定時「そうね。それに、遅延が発生した場合のジョブの状態変化に『保留』を設定することによって、ユーザが『保留解除』操作を行うまで、そのジョブは実行されずに待機させることができるわ。予期しない事態でジョブに遅れが発生している場合、その原因を取り除くまでジョブを一時的に止めておいて、原因を解消してから再開できるわね」

藤井「確かに遅れの発生の原因を解消しないでジョブの実行を進めても、正常に処理が進まないかもしれないですもんね」

定時「そして『スキップ』を設定すれば、遅延が発生したらジョブを実行せずに終了させることができるわ」

藤井「開始遅延でスキップする場合には、終了値を設定できるんですね。後続のジョブの待ち条件にこの終了値を設定しておけば、開始遅延が発生した際に、本来行う処理の代わりに、遅延が発生した場合のリカバリ処理を実行できますね」

定時「そうね、Hinemosにはほかにも終了遅延というのもあるのよ。こちらも調べてみましょ」

開始遅延と同様に、次の条件でジョブの終了が遅延していることの判定もHinemosでは行うことができます。

・指定した時刻になっても特定のジョブが終了しない場合
・ジョブセッションが開始してから○分後までに特定のジョブが終了しない場合
・特定のジョブが開始してから○分後までにそのジョブが終了しない場合

終了遅延もジョブ［ジョブの作成・変更］ダイアログで設定します。ジョブ［ジョブの作成・変更］ダイアログの終了遅延タブに、開始遅延同様、遅延と判定する条件と判定時の動作を指定します。終了遅延と判定された際には、ジョブの実行状態を中断状態にするほか、コマンドタブで設定できる「停止コマンド」を実行できます。

藤井「なるほど、開始が遅延していることの判定だけじゃなくて、終了が遅延していることの判定もできるのか。さっきの話だと、ほかのシステムでは終了しなかったことが問題だったってことだし、今回は終了遅延を設定しますね」

定時「いいわね。Hinemosではジョブ起動時に実行されるコマンドである『起動コマンド』と、ユーザが手動で停止操作をする場合に実行されるコマンドである『停止コマンド』があるのよね。終了遅延の操作でも停止コマンドの実行を指定できるみたいだから、停止コマンドに、作業の切り戻しや必要なリカバリ処理ができるコマンドを指定しておけば、終了遅延が発生した場合、端末に駆けつけなくてもHinemosが自動で対処しておいてくれるわね」

多重実行制御でジョブの同時実行数をコントロール

藤井「よし、これでジョブの実行が遅れてた場合でも自動で対処できるようになって安心ですね」

定時「待って。たしかさっきの話だと複数のジョブが想定以上に同時実行されると、対象サーバで動作するジョブ同士が影響を受けやすいって話よね」

藤井「はい」

定時「ならサーバ上で実行されるジョブの数を制限しておいたほうが、より安心じゃないかしら」

▼図2　ジョブの同時実行数が上限に達した際に処理するための設定

▼図3　レポーティングオプションではジョブの運行状況をわかりやすく表示できます

セッションID	ジョブID	実行状態	終了状態	開始時刻	終了時刻	実行時間	0:00
20141002234000-000	JN_002 (日跨ぎ用ジョブネット)	終了	正常	23:40:00	1:20:03	01:40:03	
	JB_002_01 (バックアップ前処理)	終了	正常	23:40:00	23:50:01	00:10:01	
	JB_002_02 (バックアップ本処理)	終了	正常	23:50:01	0:50:02	01:00:01	
	JB_002_03 (バックアップ後処理)	終了	正常	0:50:02	1:20:03	00:30:01	
20141003000200-000	JN_003 (エラー用ジョブネット)	終了	異常	0:02:00	0:06:01	00:04:01	
	JB_003_01 (エラージョブ)	終了(終了遅延)	異常	0:02:00	0:06:01	00:04:01	
20141003000700-000	JB_001_02 (ファイルコピー用ジョブ)	終了	正常	0:07:00	0:07:03	00:00:03	
20141003001700-000	JB_001_02 (ファイルコピー用ジョブ)	終了	正常	0:17:00	0:17:03	00:00:03	
20141003002700-000	JB_001_02 (ファイルコピー用ジョブ)	終了	正常	0:27:00	0:27:03	00:00:03	
20141003003700-000	JB_001_02 (ファイルコピー用ジョブ)	終了	正常	0:37:00	0:37:03	00:00:03	
20141003004200-000	JB_001_01 (ディスク書き込み用ジョブ)	終了	正常	0:42:00	0:43:37	00:01:37	
20141003004700-000	JB_001_02 (ファイルコピー用ジョブ)	終了	正常	0:47:00	0:47:03	00:00:03	
20141003005700-000	JB_001_02 (ファイルコピー用ジョブ)	終了	正常	0:57:00	0:57:03	00:00:03	
20141003010200-000	JN_003 (エラー用ジョブネット)	終了	異常	1:02:00	1:06:01	00:04:01	
	JB_003_01 (エラージョブ)	終了(終了遅延)	異常	1:02:00	1:06:01	00:04:01	
20141003010700-000	JB_001_02 (ファイルコピー用ジョブ)	終了	正常	1:07:00	1:07:03	00:00:03	
20141003011700-000	JB_001_02 (ファイルコピー用ジョブ)	終了	正常	1:17:00	1:17:03	00:00:03	
20141003012700-000	JB_001_02 (ファイルコピー用ジョブ)	終了	正常	1:27:00	1:27:03	00:00:03	
20141003013700-000	JB_001_02 (ファイルコピー用ジョブ)	終了	正常	1:37:00	1:37:03	00:00:03	
20141003014200-000	JB_001_01 (ディスク書き込み用ジョブ)	終了	正常	1:42:00	1:43:36	00:01:36	
20141003014700-000	JB_001_02 (ファイルコピー用ジョブ)	終了	正常	1:47:00	1:47:04	00:00:04	
20141003015700-000	JB_001_02 (ファイルコピー用ジョブ)	終了	正常	1:57:00	1:57:05	00:00:05	
20141003020200-000	JN_003 (エラー用ジョブネット)	終了	異常	2:02:00	2:06:01	00:04:01	
	JB_003_01 (エラージョブ)	終了(終了遅延)	異常	2:02:00	2:06:01	00:04:01	
20141003020700-000	JB_001_02 (ファイルコピー用ジョブ)	終了	正常	2:07:00	2:07:04	00:00:04	
20141003021700-000	JB_001_02 (ファイルコピー用ジョブ)	終了	正常	2:17:00	2:17:03	00:00:03	

藤井「確かに、できるならやっておいたほうが良さそうですが……」

定時「もちろんできるわよ。Hinemosでは各ノードごとに同時に実行するジョブの数を制限できるの。上限数を越えて実行しようとした分は、ほかのジョブが終了するまで待機させたり、実行せずに終了できるのよ」

ジョブ管理ツールには、ジョブの同時実行数を制御する機能を有するものがあります。Hinemosにもその機能があり、ノードで実行されるジョブの数を制御できます。ジョブの同時実行数の制御は、ジョブ[ジョブの作成・変更]ダイアログの多重度タブで設定します(**図2**)。

「多重度が上限に達したときの挙動」の「通知」にチェックを入れると、ノードで同時に実行しているジョブ数が、設定した上限に達したときに通知によってユーザに知らせることができます。また、操作では「待機」と「終了」を選択できます。「待機」はほかのジョブが終了して同時実行している数が減るまで、待機します。「終了」は上限を越えたジョブについては実行せずにそのまま終了します。

なお、ノードごとのジョブの同時実行数の上限は、リポジトリ[ノードの作成・変更]ダイアログで設定できるノードプロパティの「ジョブ多重度」で設定します。デフォルトでは「0」が設定されており、「0」の場合は上限なしとなります。同時実行数の制限をしたい場合は「1」以上の値に設定します。

藤井「システムチェックなど、単一のジョブを繰り返し実行している場合なら、上限を越えた分は実行せずに終了させてしまうのもありだと思いますが、勤怠管理システムの場合は、どれもちゃんと実行してほしいものばかりなので『待機』を選ぼうと思います」

定時「そうね、それがいいんじゃないかしら。待機状態が続いて実行が遅れたら先に設定していた開始遅延、終了遅延と組み合わせることもできるからね」

藤井「はい、これで正常時の各サーバ間の連携や、スケジューリングしての定時実行、異常時の自動リカバリ処理と通知機能、それに異常となる前に遅延が発生していないか確認、とジョブの運用に必要な設定をすべて行うことができました。これで勤怠管理システムの運用自動化ができたといって良いんじゃないでしょうか」

定時「そうね、よくやったわ。そういえば、この前(第7回のコラム)話してたレポーティングオプション。これを使うとこんな風(**図3**)に実行したジョブを一覧にできるのよ」

藤井「すごい！便利ですね、これ」

ジョブ運用は続くよどこまでも

それからしばらくしたある日、藤井君は上司に呼ばれました。

上司「藤井君、ちょっといいかな」

藤井「はい、なんでしょう」

上司「ついに勤怠管理システムの運用自動化をやり遂げたようだね」

藤井「はい……、といってもツールを導入しただけでほとんどHinemosの機能を使っただけなんですけどね」

上司「何言ってるんだ。最初シェルスクリプトとcronの組み合わせで動いていたのを、属人性を排除して効率化するためにツールを導入したいと言ったのはほかでもない藤井君じゃないか。よくやり遂げてくれたね」

藤井「はい、ありがとうございます！」

上司「勤怠管理システムの運用で、ほかに何かすることはあるのかね」

藤井「ジョブ運用についてはもう、追加で設定したほうがいいところはないと思います。ただ、せっかくHinemosにはジョブ機能だけでなく、監視機能もついているので、サーバの死活監視やリソースの監視をすれば、もっとより良い運用になるのではないかと思っています」

上司「そうか、まだまだ勤怠管理システムに必要なことがあるんだな」

藤井「はい、でもそれがどうしたんですか」

上司「実は社内でクラウド環境でのシステム運用に人手が必要になってるようでな。藤井君は以前からクラウドに興味があると話していたことだし、そちらのプロジェクトに参画してみたらどうかと思っているんだ。そうすると、勤怠管理システムの運用にも手が回りにくくなるから、ほかの人に引継ぎが必要だと思ってな」

藤井「なるほど、勤怠管理システムはツールを導入したことによって引き継ぎも簡単になっています。ちょっと寂しいですけれど僕もクラウドには興味ありますし、できればそっちの仕事をやってみたいです。それにクラウド環境の運用でも、これまでに身につけてきたジョブ運用の知識と経験を活用できるかもしれません」

上司「よしわかった。勤怠管理システムの運用の後任者はおいおい決めることにして、まずは運用の引き継ぎができるように準備をしてくれ」

藤井「はい！」

今月の時短ポイント

開始遅延、終了遅延を活用することにより、ジョブの実行が想定よりも遅れていることに気づけ、自動で対処できます。実際にトラブルが発生してから対処するのではなく、トラブルの予兆を検知し対処することによって、その対応時間を短くし、システムの運用に与える影響も最小限に抑えることができます。

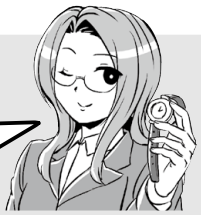
次回予告

急遽クラウド環境の運用チームへの参画が決まった藤井君。これまでの勤怠管理システムの運用を振り返って引き継ぎの準備を進めます。さらにクラウド環境でのジョブ運用についても調べてみるようです。藤井君の活躍はまだまだ続きそうです。

次回「これで引き継ぎもらくらく！　Hinemosによるジョブ運用のおさらい」SD　*To Be Continued...*

Column

ファイル作成を契機にジョブを起動する



これまでの連載では、おもにスケジュール実行したジョブについて扱ってきましたが、Hinemosでは、ジョブを実行する契機として、次の4つがあります。

- Hinemosクライアントで実行ボタンをクリックして手動で実行する
- あらかじめ決めておいたスケジュールの時刻が来たら自動で実行する
- 特定の監視結果が得られた場合に自動で実行する
- 特定のサーバに特定のファイルが作成されたら自動で実行する

本コラムでは、最後の「特定のサーバに特定のファイルが作成されたら自動で実行する方法」について紹介します。

この機能をHinemosではファイルチェックと呼んでいます。ファイルチェックを作成するためにはジョブパースペクティブのジョブ［実行契機］ビューで作成します。このビューの「ファイルチェック作成」ボタンをクリックすると、ジョブ［ファイルチェックの作成・変更］ダイアログが開くので、こちらで設定を行います（**図4**）。

基本的には第5回（2015年2月号）で紹介した、ジョブスケジュールを作成する手順とほとんど同じで、実行するジョブと実行するタイミングを設定すれば完了です。実行契機IDにはスケジュールとファイルチェックでユニークなIDを設定します。ジョブIDには、このファイルチェックで実行したいジョブやジョブネットを指定します。そして、ファイルチェック設定で具体的に、どのサーバの、どのファイルが、どのようになったときにジョブを実行するかを設定します。チェックするサーバは「スコープ」で設定します。またチェックするファイルはディレクトリとファイル名を入力します。ファイル名には正規表現が使えるので、たとえば末尾にその日の日付が入るファイル（masterdb_data.20150418など）をチェックできます。

そしてジョブを実行する条件についてはファイルの作成だけでなく削除や変更（タイムスタンプ変更とファイルサイズ変更の2種類）を指定でき、柔軟にジョブの実行条件を設定できます。

この機能を使うことによって、たとえばほかのシステムから送信されたマスタデータを、別のシステムに投入する、といったシステム間連携をシンプルに設定でき、複雑な処理を簡単に行うことができます。

▼図4　ファイルチェックの設定例

シェルスクリプトではじめるAWS入門

――AWS APIの活用と実践

第11回　AWS APIでのデジタル署名の全体像を明らかにする⑤

Writer　波田野 裕一（はたの ひろかず）／運用設計ラボ　operation@office.operation-lab.co.jp

トピック

前回のトピックでも紹介した国内最大級のAWSユーザイベント「JAWS DAYS 2015」に参加してきました。当日は過去最多の1,000人超の方々が実際に会場に足を運び、7つのトラックから興味のあるセッションを思い思いに聴講していました。JAWS DAYSの面白いところは、見本市のようにすべてのトラックが1つの大会場の中で同時並行して行われるので、興味しだいでつまみ喰いのように聞けるところでしょう。

今年のJAWS DAYSの大きな特徴の1つは、昨年10月に発表されたIntel Edisonの実機とAWSのサービスを組み合わせた可視化や機械学習のハンズオンが行われたことでしょう。定員の数倍にもなるキャンセル待ちが発生し、AWSユーザがいかにIoTに注目しているかが感じられただけでなく、当日は参加者のみなさんが熱心に聴講し手を動かしている光景がとても印象的でした。今後は、AWSコミュニティの中でもIoTを意識した活動が加速していくのではないでしょうか。

今回の流れ

第7回から、AWS APIを直接操作するために必要な次の3種類のディジタル署名について解説しています。前回の第10回までで、3つの署名方法のうちSignature Version 2、Signature Version 3についてそれぞれの概要と実例を解説しました。

今回から、最新かつ最も複雑な署名方法であるSignature Version 4（以下「v4」）の概要について解説をします。

Signature Version 4の作成手順

v4の署名付きリクエストデータの作成手順は**図1**のとおりです。

v4は、署名作成の過程でリクエストデータ本体を組み込むため、操作対象のサービスによって処理が大幅に異なります。たとえば、DynamoDBでは投入するJSONデータが署名対象に含まれ、S3では操作対象のオブジェクトのエンコードデータが必要になります。本記事では、比較的シンプルな操作であるIAMユーザの一覧を取得するリクエストを作成します。

最近、署名方法の公式ドキュメントの日本語

▼図1　リクエストデータ作成手順

版が公開されました[注1]。本記事では、このサンプルを例にコマンドラインでの署名方法を解説し、最後にopensslコマンドで実環境へ実際に投入したいと思います[注2]。

準備

最初に、サンプル用の日時情報と認証情報を変数に取り込みましょう。

日時情報の取得

v4では日付情報と時間情報が署名を行ううえで重要な意味を持っています。初回は、サンプルデータを作成するために次の情報を利用します。以降、黒地の行ではコマンドのサンプルを示します。

```
DATE='20110909'
TIME='233600'
```

実環境へのリクエストに対して署名を行う場合は、次のコマンドでそれぞれ取得してください。

```
DATE=`LC_ALL=C date -u +%Y%m%d`
TIME=`LC_ALL=C date -u +%H%M%S`
```

認証情報の取得

次に、AWS APIへの認証情報を変数に取り込みます。サンプルデータに対する認証情報は次になります。

```
aws_access_key_id=AKIDEXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/➥
K7MDENG+bPxRfiCYEXAMPLEKEY
```

準備は以上で完了です[注3]。これ以降はサンプルデータと実環境を問わず同じ手順で作業を進めていくことになります。

タスク1［正規リクエストを作成し、ハッシュ値を得る］

最初の作業として、正規リクエスト（Canonical Request）を作成し、そのハッシュ値を取得します。正規リクエストは次の要素から成り立ちます。

- HTTPリクエストメソッド（GET/POST/PUT/DELETE）
- 正規URI
- 正規リクエスト文字列
- 正規ヘッダ
- （空行）
- 署名ヘッダ
- ペイロードのハッシュ値

APIのバージョン、リクエスト内容の決定

正規リクエストを作成する前段階として、利用するAPIのバージョンとリクエスト内容を決定します。2015年3月現在、IAMの最新バージョンは'2010-05-08'です。

```
API_VERSION='2010-05-08'
```

リクエスト内容として、ListUserアクションをパラメータなしで指定します。

```
API_PARAM="Action=ListUsers"
```

最後にAPIバージョンとリクエスト内容を結合すると、APIに対するリクエスト情報が完成します。このときに、'&'区切りで昇順にソートされている必要があるので注意してください。

```
REQUEST_STRING="${API_PARAM}&➥
Version=${API_VERSION}" && echo ➥
$REQUEST_STRING
```

注1）URL http://docs.aws.amazon.com/general/latest/gr/sigv4-signed-request-examples.html

注2）opensslコマンドは1.x系列の最新版を使用してください。

注3）現在のAWS APIテストスイートでは、おおむねこの認証情報が使われているようです。実環境へのリクエストに対して署名を行う場合は、2015年3月号の第9回「3. 事前準備」を参照して実環境の認証情報を変数に格納してください。

結果は次のようになります(以降グレー地で入力結果を示します)。

```
Action=ListUsers&Version=2010-05-08
```

署名アルゴリズムの指定

次に、AWS APIへのリクエストに対する署名アルゴリズムを決めておきましょう。これは、各種ハッシュ値の取得や署名の際に同じものを指定する必要があります。ここでは、現時点で最も一般的なSHA-256を指定しておきます。

```
X_AMZ_ALGORITHM='AWS4-HMAC-SHA256'
```

以降、この手順でopensslコマンドを利用するときに、署名アルゴリズムとして"**-sha256**"オプションを指定することになります。

HTTPリクエストメソッド、正規URI、正規リクエスト文字列の決定

今回は、HTTPリクエストメソッドとしてPOSTを利用します。

```
HTTP_METHOD='POST'
```

IAMのAPIはクエリ形式で実装されているため、正規URI(Canonical URI)は"**/**"になります。REST形式のAPIの場合は操作対象のリソースを指定します。

```
CANONICAL_URI='/'
```

APIに対するリクエスト情報は、正規リクエスト文字列(Canonical QueryString)もしくは後に説明するペイロード(payload)のどちらかで指定します。

今回のPOSTメソッドの場合は、リクエスト情報はペイロードで指定するため、正規リクエスト文字列は空になります。

```
CANONICAL_QSTRING=""
```

以上の3つの情報を元に、最初の正規リクエストを作成します。

```
cat << EOF > canonical_form.tmp
${HTTP_METHOD}
${CANONICAL_URI}
${CANONICAL_QSTRING}
EOF

cat canonical_form.tmp
```

```
POST
/
(空行)
```

署名ヘッダの作成

次に署名ヘッダ(Signed Headers)を作成します。ここには、リクエストの署名に利用するヘッダ(まだ作成していないauthorizationヘッダ以外)すべての名称を記述します。なお、POSTによるリクエストで必要になるcontent-lengthヘッダは署名ヘッダには含みません。

必要となるヘッダは、操作対象のAPIやリクエスト内容によって異なりますが、今回のIAMでのListUsersアクションでは、下記の3つのヘッダを利用します。

- content-type
- host
- x-amz-date

次のように変数に格納してください。このときに、"**;**"区切りで昇順にソートされている必要があるので注意してください。

```
SIGNED_HEADERS='content-type;host;x-amz-date'
echo "SIGNED_HEADERS: ${SIGNED_HEADERS}"
```

```
SIGNED_HEADERS: content-type;host;x-amz-date
```

正規ヘッダの作成

署名ヘッダの記載順に各ヘッダを実際に記述

し、正規ヘッダ(CanonicalHeaders)を作成します。hostにはAPIのエンドポイントを指定します。IAMの場合はグローバルで1つしかない'iam.amazonaws.com'を指定します。

```
API_HOST='iam.amazonaws.com'
```

x-amz-dataには、冒頭で取得したDATEとTIMEを使用してISO8601基本形式で表現した日時情報を指定します。

```
X_AMZ_DATE="${DATE}T${TIME}Z"
```

content-typeには、'application/x-www-form-urlencoded; charset=utf-8'を指定します。各ヘッダの内容が確定したら、正規ヘッダをファイルに出力します。ヘッダが昇順にソートされている必要があるので注意してください。

```
cat << EOF > canonical_headers.tmp
content-type:application/x-www-form-urlencoded; charset=utf-8
host:${API_HOST}
x-amz-date:${X_AMZ_DATE}
EOF

cat canonical_headers.tmp
```

```
content-type:application/x-www-form-urlencoded; charset=utf-8
host:iam.amazonaws.com
x-amz-date:20110909T233600Z
```

ペイロードのハッシュ値取得

正規リクエストの6要素のうち最後の要素となるペイロードのハッシュ値を取得します。

```
echo "${REQUEST_STRING}" | tr -d '¥n' > payload.tmp
PAYLOAD_HASH=`
  openssl dgst -sha256 payload.tmp ¥
  | sed 's/^.*= //'
`
echo "PAYLOAD_HASH: ${PAYLOAD_HASH}"
rm payload.tmp
```

```
PAYLOAD_HASH: b6359072c78d70ebee1e81adcbab4f01bf2c23245fa365ef83fe8f1f955085e2
```

正規リクエスト文字列のところで説明しましたが、POSTメソッドの場合はペイロードにリクエスト情報が入るため、その内容によりハッシュ値も異なります(GETメソッドの場合はペイロードは空になるため、ペイロードのハッシュ値は常に同じです)。

正規リクエストの完成

6つの要素が出そろったので、これらを次のように結合して正規リクエストを完成させます。

```
cat canonical_headers.tmp >> canonical_form.tmp
echo "" >> canonical_form.tmp
echo "${SIGNED_HEADERS}" >> canonical_form.tmp
echo ${PAYLOAD_HASH} |  tr -d '¥n' >> canonical_form.tmp

cat canonical_form.tmp
```

```
POST
/

content-type:application/x-www-form-urlencoded; charset=utf-8
host:iam.amazonaws.com
x-amz-date:20110909T233600Z

content-type;host;x-amz-date
b6359072c78d70ebee1e81adcbab4f01bf2c23245fa365ef83fe8f1f955085e2
```

正規リクエストのハッシュ値取得

正規リクエストが完成したら、そのハッシュを取得します。

```
REQUEST_HASH=`
  openssl dgst -sha256 canonical_form.tmp ¥
  | sed 's/^.*= //'
`
echo "REQUEST_HASH: ${REQUEST_HASH}"
```

```
REQUEST_HASH: 3511de7e95d28ecd39e9513b642aee07e54f4941150d8df8bf94b328ef7e55e2
```

サンプル用の日時情報と認証情報を使用した場合、前述のREQUEST_HASHと同じハッシュ値が取得できるはずです。 AWS API側では、このハッシュを復号化してそのリクエストを解読することにより、適切なレスポンスを返します。正規リクエストのハッシュが取得できた時点で、正規リクエスト自体は不要になるのでcanonical_form.tmpは削除して問題ありません。

```
rm canonical_form.tmp
```

タスク2[署名文字列の作成]

正規リクエストのハッシュが取得できたら、次にそれを利用して署名文字列(String to Sign)を作成します。署名文字列は、メッセージ認証符号において必要な「秘密鍵」と「メッセージ」のうちの「メッセージ」となるもので、次の4項目で構成されています。

・署名アルゴリズム(定義済)
・日時情報(定義済)
・認証情報スコープ
・正規リクエストのハッシュ(定義済)

署名文字列に必要な4項目のうち3項目はすでに定義済なので、次は「認証情報スコープ」を作成していきます。

認証情報スコープの作成

認証情報スコープ(Credential Scope)には次の4項目が含まれます。

・リクエスト日付(定義済)
・リージョン
・サービス識別子
・スコープの終了文字列

未定義の3項目について定義していきます。今回、リージョンにはIAMのエンドポイントのある'us-east-1'を指定します。

```
AWS_REGION='us-east-1'
```

サービス識別子は、IAMの場合は'iam'を指定します。

```
AWS_SERVICE='iam'
```

スコープの終了文字列には、v4を意味する'aws4_request'を指定します。

```
SCOPE_TERM='aws4_request'
```

これらを次のように結合して、認証情報スコープを定義します。

```
CREDENT_SCOPE="${DATE}/${AWS_REGION}/${AWS_SERVICE}/${SCOPE_TERM}"
echo "CREDENT_SCOPE: ${CREDENT_SCOPE}"
```

```
CREDENT_SCOPE: 220110909/us-east-1/iam/aws4_request
```

署名文字列の作成

ここまでで必要な項目がすべてそろったので、これらを次のように結合して署名文字列を完成させます。

```
cat << EOF > s2s.tmp
${X_AMZ_ALGORITHM}
${X_AMZ_DATE}
${CREDENT_SCOPE}
EOF

echo ${REQUEST_HASH} | tr -d '¥n' >> s2s.tmp

cat s2s.tmp
```

```
AWS4-HMAC-SHA256
20110909T233600Z
20110909/us-east-1/iam/aws4_request
3511de7e95d28ecd39e9513b642aee07e54f4941150d8df8bf94b328ef7e55e2
```

タスク3[署名の計算]

メッセージ認証符号において必要な「秘密鍵」と「メッセージ」のうち、「メッセージ」は完成したので、次は「秘密鍵」を作成していきます。v4では、リクエストごとに、日付を含む複数の情報(認証情報スコープの4項目)とAWSのシークレットアクセスキーを利用して4重に署名を繰り返すことで比較的強度の高い「秘密鍵」を作成しています。

秘密鍵の作成①[AWSシークレットアクセスキーのHEX変換]

まず最初に、opensslで扱えるようにAWSシークレットアクセスキーを16進数形式に変換します。

```
K_SECRET=$(for x in $(echo "AWS4${aws_
secret_access_key}" | grep -o '.');do
printf "%2X" ¥"$x; done;)
echo "K_SECRET: ${K_SECRET}"
```

```
K_SECRET: 41575334774A616C725855746E4645
4D492F4B374D44454E472B625078526669435945
58414D504C454B4559
```

秘密鍵の作成②[日付情報に署名]

日付情報に対してAWSシークレットアクセスキーで署名し、最初の秘密鍵を作成します。

```
K_DATE=`
  echo "${DATE}" ¥
  | tr -d '¥n' ¥
  | openssl dgst -mac HMAC -macopt
"hexkey:${K_SECRET}" -sha256 ¥
  | sed 's/^.*= //'
`
echo "K_DATE: ${K_DATE}"
```

```
K_DATE a83ed188be5f4b074d7f66349f5077fbc
df797bf3471fb9d5f32730f936d41a5
```

秘密鍵の作成③[リージョン情報に署名]

最初の秘密鍵でリージョン情報に対して署名し、2番めの秘密鍵を作成します。

```
K_REGION=`
  echo "${AWS_REGION}" ¥
  | tr -d '¥n' ¥
  | openssl dgst -mac HMAC -macopt
"hexkey:${K_DATE}" -sha256 ¥
  | sed 's/^.*= //'
`
echo "K_REGION: ${K_REGION}"
```

```
K_REGION 957b5875f33834a85374b750011dc2d
6f0e1d6896eeb891d36a73c711961ad6e
```

秘密鍵の作成④[サービス識別子に署名]

2番めの秘密鍵でサービス識別子に対して署名し、3番めの秘密鍵を作成します。

```
K_SERVICE=`
  echo "${AWS_SERVICE}" ¥
  | tr -d '¥n' ¥
  | openssl dgst -mac HMAC -macopt
"hexkey:${K_REGION}" -sha256 ¥
  | sed 's/^.*= //'
`
echo "K_SERVICE: ${K_SERVICE}"
```

```
K_SERVICE 0116249d060bff83faa1a627e85a4c
6f83ce50d89c334765878dcf76e28bfc6e
```

秘密鍵の作成⑤最終「秘密鍵」の作成

3番めの秘密鍵でスコープの終了文字列に対して署名し、最終的な秘密鍵を作成します。

```
K_SIGNING=`
  echo "${SCOPE_TERM}" ¥
  | tr -d '¥n' ¥
  | openssl dgst -mac HMAC -macopt
"hexkey:${K_SERVICE}" -sha256 ¥
  | sed 's/^.*= //'
`
echo "K_SIGNING: ${K_SIGNING}"
```

```
K_SIGNING 98f1d889fec4f4421adc522bab0ce1
f82e6929c262ed15e5a94c90efd1e3b0e7
```

署名

ここまでで、メッセージ認証符号において必要な「秘密鍵」(K_SIGNING)と「メッセージ」(署名文字列)の両方の作成が完了しました・署名文字列に対して秘密鍵(K_SIGNING)で署名

することで、最終的な署名を取得できます。

```
SIGNATURE_HEX=`
  openssl dgst -mac HMAC -macopt
"hexkey:${K_SIGNING}" -hex -sha256 s2s.
tmp ¥
  | sed 's/^.* //'
`
echo "Signature: ${SIGNATURE_HEX}"
```

```
ced6826de92d2bdeed8f846f0bf508e8559e98e4
b0199114b84c54174deb456c
```

これが、今回のリクエストで必要となる署名の最終版となります。署名が完了した時点で、署名文字列は不要になるのでs2s.tmpは削除して問題ありません。

```
rm s2s.tmp
```

タスク4 [署名付きリクエストの完成]

署名が完成したら、最後に署名付きリクエストを作成します。POSTメソッドの場合は次が必要になります。

- リクエスト行
- 認証ヘッダ
- content-length
- 正規ヘッダ(定義済)
- (空行)
- ペイロード(定義済)

リクエスト行の組み立て

まず、HTTPリクエストの1行目(リクエスト行)を、以下のコマンドで組み立てます。

```
HTTP_REQUEST="${HTTP_METHOD}
https://${API_HOST}/${CANONICAL_
QSTRING} HTTP/1.1" && echo "HTTP
REQUEST: ${HTTP_REQUEST}"
```

```
HTTP REQUEST: POST https://iam.
amazonaws.com/ HTTP/1.1
```

認証ヘッダの組み立て

次に、認証ヘッダを次のコマンドで組み立てます。

```
AUTH_HEADER="${X_AMZ_ALGORITHM}
Credential=${aws_access_key_
id}/${CREDENT_SCOPE},
SignedHeaders=${SIGNED_HEADERS},
Signature=${SIGNATURE_HEX}"

echo "AUTH_HEADER: ${AUTH_HEADER}"
```

```
AUTH_HEADER: AWS4-HMAC-SHA256
Credential=AKIDEXAMPLE/20110909/us-
east-1/iam/aws4_request,
SignedHeaders=content-type;host;x-amz-
date, Signature=ced6826de92d2bdeed8f846f
0bf508e8559e98e4b0199114b84c54174deb456c
```

content-lengthの取得

POSTメソッドではペイロードのcontent-length(メッセージ本体のバイト数)の値が必要となるので、ペイロードの長さを調べて変数に格納しておきます。

```
CONTENT_LENGTH=`echo ${REQUEST_STRING}|
tr -d '¥n' | wc -c`
echo "CONTENT_LENGTH: ${CONTENT_LENGTH}"
```

```
CONTENT_LENGTH: 35
```

署名付きリクエストの完成

これで必要な項目がすべてそろったので、**図2**のように結合して署名付きリクエストを完成させます。**図3**が出力結果です。

図3に出力されたこのoutput.txtが、最終的な署名付きリクエストメッセージになります。サンプル用の日時情報と認証情報を使用した場合、手順どおりに進めることができていればまったく同じ内容になるはずです。正規ヘッダ

canonical_headers.tmpはもう不要なので、削除して問題ありません。

```
rm canonical_headers.tmp
```

実際にリクエストを作成してみよう

サンプルデータで動作確認ができたら、実際にAWS APIに投入できるリクエストを作成してみましょう。本記事の「準備」のステップに戻り、実際の日時情報と認証情報を指定します。続いて、タスク1～4を実行し、その結果得られた署名付きリクエストを**図4**のとおり**openssl**コマンドでAWS APIに投入します。

図5のようにIAMユーザの一覧が表示されればリクエストは成功です。

▼図2　コマンド例（署名付きリクエストのファイル結合）

```
cat << EOF > output.txt
${HTTP_REQUEST}
authorization:${AUTH_HEADER}
EOF

cat canonical_headers.tmp >> output.txt
echo "content-length:${CONTENT_LENGTH}" ➥
>> output.txt
echo "" >> output.txt
echo "${REQUEST_STRING}" >> output.txt


cat output.txt
```

▼図3　ファイルの結合結果

```
POST https://iam.amazonaws.com/ HTTP/1.1
authorization:AWS4-HMAC-SHA256 ➥
Credential=AKIDEXAMPLE/20110909/us-➥
east-1/iam/aws4_request, ➥
SignedHeaders=content-type;host;x-amz-➥
date, Signature=ced6826de92d2bdeed8f846f➥
0bf508e8559e98e4b0199114b84c54174deb456c
content-type:application/x-www-form-➥
urlencoded; charset=utf-8
host:iam.amazonaws.com
x-amz-date:20110909T233600Z
content-length:35

Action=ListUsers&Version=2010-05-08
```

次回は

次回は、今回解説したSignature Version 4の署名付きリクエストデータを作成するシェルスクリプトのサンプルを紹介する予定です。SD

▼図4　リクエストコマンドの例

```
openssl s_client -connect ${API_HOST}:443

   ……（略）……
    Timeout   : 300 (sec)
    Verify return code: 20 (unable to ➥
get local issuer certificate)
   入力待ち（ここに署名付きリクエストデータを貼り付け、
   2回Enterキーを押す）
```

▼図5　リクエストの成功例

```
HTTP/1.1 200 OK
x-amzn-RequestId: 8c950f6e-ce9e-11e4-➥
8522-0facd9acae23
Content-Type: text/xml
Content-Length: 3297
Date: Fri, 20 Mar 2015 01:15:08 GMT

<ListUsersResponse xmlns="https://iam.➥
amazonaws.com/doc/2010-05-08/">
  <ListUsersResult>
    <Users>
      <member>
        <UserId>AIDAXXXXXXXXXXXXXXXX</➥
UserId>
        <Path>/</Path>
        <UserName>admin</UserName>
        <Arn>arn:aws:iam::XXXXXXXXXXX:➥
user/admin</Arn>
        <CreateDate>2015-03-19:41:29Z</➥
CreateDate>
      </member>
    </Users>
    <IsTruncated>false</IsTruncated>
  </ListUsersResult>
  <ResponseMetadata>
    <RequestId>8c950f6e-fe9e-11e4-8522-➥
0facd9acae23</RequestId>
  </ResponseMetadata>
</ListUsersResponse>
```

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13回 標準コマンドから改めて見るEmacs

パッケージをインストールして、init.elにバリバリと設定を加えていき、Emacsをカスタマイズしていく本連載ですが、今回は趣旨を変えて標準コマンドを見ていきます。デフォルトではどんなコマンドがあるのかを把握し、プレフィックスキー・引数によるコマンドの成り立ちを知ることで、より深くEmacsを理解できます。

今こそ基本に立ち帰ろう

ども、るびきちです。本連載も先月号で1年になり、まさに1周したような感じです。最初の数回はEmacsを使ったことのない人を対象に、Emacsの基礎概念をゆっくりと紹介しました。そのあとはddskkによるOS非依存の日本語入力を紹介したことで、複数のOSを使う人でもEmacsは有用だと示しました。続けて、さまざまなカーソル移動、入力支援、検索・置換というテキストエディタとしてのEmacsの使い方を紹介しました。dired、eshellというOS非依存のファイラとシェルも紹介しました。Emacs上のワープロ・スーパー電子手帳であるorg-modeについても軽く触れました。

そして、先月号と先々月号の2回に分けて、Emacsの諸機能を統合するhelmという超強力なパッケージを紹介しました。helmを使いこなすことで、あなたのEmacsはかなり使いやすくなったことでしょう。

今回はあえて初心に帰り、キーに割り当てられた標準コマンドに焦点を当ててみます。今回が初めての読者はもちろんのこと、ベテランユーザも今一度、標準機能を見直してみてはいかがでしょうか。最近導入された機能は意外と知られていないものです。

キーに割り当てられたコマンドを見てみよう

Emacsはデフォルトの状態で無数のコマンドがキーに割り当てられています。<f1> bを実行すれば、キーに割り当てられたすべてのコマンドを見られます。しかし、それはカスタマイズされた状態での表示ですので、デフォルトの設定ではありません。

そこで、端末からemacs -Qで立ち上げたあとで同じコマンドを実行すればデフォルトの状態がわかります。あるいはeshellで**リスト1**を実行しても良いです。このコマンドは、デフォルトの状態のEmacsをバッチモードで起動し、<f1> b(describe-bindings)を実行、表示された内容を標準出力に出力して終了します。この出力は約1,000行にもおよびます。これだけ多くのコマンドが標準状態で割り当てられているのは驚きです。Emacsを使って20年になる筆者も、すべてのコマンドを使いこなせているわけではありません。

▼リスト1　eshellからデフォルトのキー割り当てを確認

```
emacs -batch -f describe-bindings -eval '(with-current-buffer "*Help*" (princ (buffer-string)))'
```

1ストロークのキー

Emacsで頻繁に使われるコマンドは1ストロークのキーに割り当てられています。

とくに Ctrl +英字は基本的なコマンドばかりなのでしっかりおさえておきましょう。とはいえ、すべてが使用頻度の高いコマンドという訳ではありません。C-tは文字の順序が入れ替わったタイプミスを修正してくれるのですが、日本語のローマ字入力では使えないので無理に使う必要はありません。C-qは制御文字や改行文字を入力するときに使うものですが、筆者はめったに使いません。C-zはEmacsのフレームをサスペンドしますが、Emacsひきこもり生活をしていればまったく使いません。

困ったらC-gやC-/を押すことは絶対に覚えてください。Emacsユーザはパニックになってはいけません。C-gでコマンドを取り消し、C-/でバッファへの変更を取り消します。たまにC-gで取り消されず、Emacsが固まってしまうこともありますが、それはEmacs本体かLispレベルのバグですのであきらめて再起動しましょう。

特殊キーの代わりに Ctrl を使うと指を動かす距離が短くなります。 Tab の代わりにC-i、 RET の代わりにC-m、 Esc の代わりにC-[が使えます。

Metaキー(Alt)+英字のコマンドには、だいたい Ctrl +英字の補助的なコマンドが多いです。作用対象が文字→単語、行→センテンスに変わったりします。おもしろいことに、M-pとM-nには何もキーが割り当てられていません。これらはメジャーモードが自由に割り当てられるようにわざと空けられています。名前を指定してコマンドを実行するM-xが重要なのは言うまでもないです。

Metaキー+記号文字には多くのコマンドが割り当てられています。たとえば、M-!でシェルコマンド実行、M-%で置換などがあります。とくにM-/のdabbrevはタイプミス撲滅・タイプ数節約のためには多用しておきたいです。中には使用頻度の低いコマンドも含まれていますが、そういうのは無理に覚える必要はありません。

プレフィクスキー

それでは、もっと深く見ていきましょう。Emacsにはプレフィクスキーが用意されており、2ストローク以上のキーにコマンドが割り当てられます。そのおかげで多くのコマンドをキーに割り当てられます。出力から抜き取り吟味した結果、**表1**のとおり、多くのプレフィクスキーが見つかりました。

各プレフィクスキー

それではおもなプレフィクスキーを見ていきましょう。

▼表1　プレフィクスキー

C-c	汎用プレフィクスキー
ESC	Metaキー、C-[と同じ
C-x	標準プレフィクスキー
C-x C-k	キーボードマクロ関係
C-x RET	エンコーディング関係
C-x ESC	C-x ESC ESC(repeat-complex-command)のみ
C-x 4	other-window関係
C-x 5	other-frame関係
C-x 6	2C(two-columns)関係
C-x 8	アクセント文字などを入力する
C-x a	abbrev関係
C-x a i	abbrev関係
C-x n	ナローイング関係
C-x r	レジスタ・矩形関係
C-x v	VC(バージョン管理)関係
C-h	ヘルプ
C-h 4	C-h 4 i(info-other-window)のみ
M-s	検索関係
M-s h	ハイライト関係
M-o	facemenu関係
M-g	カーソル移動関係
M-ESC	ESC ESC ESC(keyboard-escape-quit)のみ
<f1>	C-hと同じ
<f1> 4	C-h 4と同じ
<f2>	C-x 6と同じ

標準コマンドへの入口のC-x

C-xは常に使える標準コマンドのためのプレフィクスキーです。

C-x＋1文字にも重要なコマンドが勢ぞろいしています。たとえばC-x C-s(保存)やC-x C-f(ファイルオープン)やC-x b(バッファ切り替え)などは真っ先に覚える必要があります。

Emacsにはデフォルトで1,000ものコマンドが割り当てられているため、C-xから始まる2〜3ストロークのプレフィクスキーもあったりします。それらにはおおむね同じカテゴリのコマンドが割り当てられています。

たとえばC-x 4をプレフィクスキーとするコマンドは、other-window系列のコマンドであり、隣のウィンドウに表示するものばかりです。しかも普通のC-x系のコマンドから類推できるキーですので覚えやすいです。C-x 4 bはC-x bの、C-x 4 C-fはC-x C-fのother-window版です。ウィンドウが分割されていないときは分割します。

プレフィクスキーC-x 5はother-frame系列のコマンドで、別フレームで表示するものばかりです。C-x 4と同じくC-x 5 bはC-x b、C-x 5 C-fはC-x C-fに対応しています。

プレフィクスキーC-x nはナローイング(narrowing)関連のコマンド担当です。ナローイングとは、編集領域をバッファの一部に制限し、範囲外を一時的に見えなくする機能です。C-x n nでregionをナローイングし、C-x n wでナローイングを解除します。ナローイングのコマンドはほかにも用意されています。

プレフィクスキーC-x rはレジスタ(register)、矩形(rectangle)関連のコマンド担当です。それぞれ働きは異なりますが、どちらも頭文字がRですので共通のプレフィクスキーになっています。どちらも説明すると長くなるので割愛します。ほかに置き場所がないのか、なぜかブックマーク関連のコマンドもC-x rをプレフィクスキーにしています。

プレフィクスキーC-x vはバージョン管理システム関連のコマンド担当です。C-x v vでファイル登録・コミット、C-x v =でdiff、C-x v lでログを表示します。ほかにもたくさんのコマンドがありますが、通常使用ではこの3つがあれば事足ります。

プレフィクスキーC-x RET(C-x C-m)はバッファやファイルのエンコーディング(coding system)関連のコマンド担当です。バッファのエンコーディングを変更して保存するにはC-x C-m fで設定します。Emacsがエンコーディングを誤認識したときはC-x C-m rで明示的に指定してファイルを開き直します。この2つを知っていれば十分です。

モードコマンド・ユーザ割り当てのC-c

C-cは汎用的なプレフィクスキーです。C-cをプレフィクスキーとする割り当て方はキーバインドの規約で決まっています。

まず、C-c＋英文字はユーザが自由に割り当てられる領域になっています。elispプログラムではそれらにコマンドを割り当ててはいけないという厳密な決まりがあります。メジャーモードで使うのは論外ですが、実際には一部の外部パッケージによるマイナーモードでは使われていることもあります。よって、規約に違反しているマイナーモードがなければ、安心して好きなコマンドを割り当てられます。

C-c＋コントロール文字、数字、{、}、<、>、:、;はメジャーモードが使うように予約されています。ほとんどのメジャーモードはC-c＋C-英文字に割り当てれば間に合います。

プレフィクスキーのあとでもEmacsのキーの意味が継承されています。たとえばC-c C-pやC-c C-bで前の要素に、C-c C-nやC-c C-fで次の要素に移動するコマンドが割り当てられていることがしばしば見られます。

特定のメジャーモードに付随するマイナーモード(たとえばorg-modeに対するorg-capture-mode)は、org-modeの元のキーバインドを上書

▼リスト2　C-hを1文字後退に割り当てる

```
(global-set-key (kbd "C-h") 'delete-backward-char)
```

きするために、わざとC-c C-cなどにコマンドを割り当てていたりします。なぜなら、マイナーモードはメジャーモードより優先するからです。

C-cのあとにここで述べた以外の記号文字を置くコマンドは、マイナーモードのために用意されています。とはいえ、メジャーモードで使うことは禁じられていません。実際org-modeのような巨大メジャーモードではC-c 'などにコマンドが割り当てられています。

ヘルプの<f1>（C-h）

<f1>とC-hは、ヘルプのためのプレフィクスキーになっています。関数や変数の説明を見たりinfoを読んだりEmacsの変更履歴を見たりできます。ヘルプコマンドはとても多いので、これらのキーを2度押すことでヘルプメニューが出てきます。

コマンドを正確に覚えていれば<f1>は1回でかまいません。たとえば<f1> <f1>のあとにaを押すとaproposが起動しますが、<f1> aでも同じことができます。aproposとは、スペース区切りの正規表現にマッチするシンボル（コマンド、関数、変数、フェイス）を探すコマンドです。

実用の観点ではC-hはヘルプではなく1文字後退にすると良いです。delete-backward-charに割り当てるか（**リスト2**）、OSレベルでBackSpaceのキーコードを発行させると良いです。とくに後者はEmacsの外でもC-hで1文字後退してくれるようになってとても快適です。

カーソル移動補助のM-g

M-gはEmacs22から導入されたカーソル移動系プレフィクスキーです。M-g M-gで指定した行番号に移動（goto-line）します。Emacs24.3からはM-g C-iで指定した桁に移動（move-to-column）し、M-g cで指定した位置に移動（goto-char）するコマンドも追加されました。

外部プログラムとの橋渡しをするのもEmacsの重要な役割です。M-x compile、M-x grep、M-x executable-interpretはそれぞれ外部プログラムをEmacsのバッファで実行します。外部プログラム実行後に、M-g M-n（next-error）とM-g M-p（previous-error）を使うと、ウィンドウを選択することなく実行結果が指し示す行にジャンプします。M-g M-nは大昔からC-x `にも割り当てられています。また、これらのコマンドはelispによるバッファ内検索であるM-s o（occur）にも対応しています。

M-gにはこれくらいしかコマンドが割り当てられていないので、余っている部分にはユーザが好きなようにコマンドを割り当てれば良いです。

検索、そして新天地のM-s

M-sはEmacs23から導入された検索系プレフィクスキーです。比較的新しいので知名度はあまり高くないかもしれません。M-x occurはM-s oに割り当てられています。

M-s w（isearch-forward-word）で単語のisearchができます。旧来ではC-u C-s \bと正規表現を使う必要がありましたが、Emacs24.4からはシンボルisearchができるようになりました。とくにコーディングの際には現在位置のシンボルを検索することが多いです。従来のisearchの欠点は短い文字列をisearchしたとき、その文字列を含む長い文字列にもマッチする点です。そこでシンボルisearchを使うと、そのシンボルそのものに対してのみマッチします。たとえばsetqに対してシンボルisearchを使うと、setqにはマッチしますが、setq-localにはマッチしません。シンボルisearchはM-s .（isearch-forward-symbol-at-point）でできます。旧来ではC-u C-s _< C-wと操作する必要があり、と

ても面倒でした。

M-s hはバッファ自動色付け(hi-lock)機能のためのプレフィクスキーです。M-s h .(highlight-symbol-at-point)は現在のシンボルをすべて色付けします。M-s h l(highlight-lines-matching-regexp)は正規表現にマッチする行をすべて色付けします。M-s h u(unhighlight-regexp)で、色付けを解除します。

M-sも割り当てられているコマンドが少ないので、ユーザは安全にコマンドを割り当てられます。とくにM-s M-英字にはいっさい割り当てられていないのは、押しやすいキーを求めている人には絶好のチャンスです。

数引数・前置引数

数引数・前置引数は地味ながらも基本的なEmacsの機能ですが、まだ話していませんでした。数引数・前置引数とは一言で言えばコマンドの前に指定するC-uのことです。

数引数

一部のコマンドにC-uを前置したとき、そのコマンドの挙動が変わります。C-uが「数引数」と言われるのは、多くの編集コマンドや文字をその数だけ繰り返すからです。たとえばC-fの代わりにC-u 4 C-fを実行すると、4文字進みます。C-bやC-pやC-nに対しても同様です。負の数を指定した場合は逆の挙動をします。たとえばC-u - 3 C-fで3文字戻ります。

ただ、このように数字や負号を指定するためにCtrlを離すと操作しづらいですね。そこで、Ctrlを押しながらでも数を指定できるようになっています。C-u 4 C-fはC-u C-4 C-fと操作できます。

実はCtrlやMetaキー、あるいはその両方を押しながら数を指定したときは、直前のC-uを省略できます。C-u C-4 C-fはC-4 C-fと、C-u 4 M-fはM-4 M-fとなります。

数引数を取るコマンドで数を指定せずにC-uのみを指定した場合、4が指定されたことになります。よって、C-u 4 C-fなんて操作する必要はなく、C-u C-fでいいのです。「4」は絶妙な数で、C-fを3回押すのはともかく、4回押すのはだるいと感じる人間の心理を表しているようです。かといってC-4 C-fは左手人差指を上下する必要があり、打ちづらいです。そこでちょっと前に行きたいと思ったときに、サッとC-u C-fを打つのです。筆者は3回押すのも面倒ですのでC-3 C-fやC-3 C-bも多用しています。

C-uを何度も押すと4の累乗が指定されます。C-u C-u C-fで16文字進みますし、C-u C-u C-u C-fで64文字進みます。筆者は特定の文字のみの行を作成するのに、しばしばC-u C-u C-u =やC-u C-u C-u #を使っています。

数やハイフンをたくさん入力するにはC-u <繰り返す数> C-u <文字>と、操作します。Ctrlを押しっぱなしでC-5 C-u 5とすると、5が5つ入力できます。このようにC-uは数引数の終わりも意味します。

前置引数

C-uは数を指定するだけではなく、コマンドの挙動を変更する場合にも用いられます。たとえば、M-!はシェルコマンドを実行するコマンドですが、何も付けずに実行すると別ウィンドウに結果が表示されます。C-u M-!とした場合、実行結果がカーソル位置に貼り付けられます。

亜種であるM-|はregionを標準入力としてシェルコマンドを実行します。C-u M-|とした場合、regionの内容をシェルコマンドの出力に置き換えます。たとえば各行を逆順にする場合、M-x reverse-regionというコマンドを知らなくて、tacシェルコマンドを知っているならば、C-u M-| tacと実行できます。行のソートもEmacsのコマンドよりもC-u M-|でsortシェルコマンドを使うほうがより細かな指定ができます。

これらは挙動の変更程度の違いですが、C-SPC

の前にC-uを置くと、まるっきり別の挙動をします。カーソル位置をマークするのではなくて、過去のマークへジャンプするのです。

これらのケースではC-uに「数」という意味はなくなり、ただ置かれたかどうかがチェックされます。その場合は数引数ではなく「前置引数」と言われます。数引数は前置引数の特別な場合ともいえます。中にはC-uが押された数によって挙動が変化するコマンドも存在します。

このように、C-uを使うことで複数の機能を1つのコマンドにまとめられます。普段C-uを使わないコマンドに対してC-uを付けたときの挙動を追加すれば、限られたキーバインド資源を有効活用できます。それを行うパッケージがmykie.elで、筆者のサイトで紹介しています[注1]。

注1） URL http://rubikitch.com/2014/11/09/mykie

終わりに

いかがだったでしょうか？　パッケージ全盛時代でつい外部パッケージに目が行ってしまいがちな今日このごろですが、あえて標準コマンドに立ち帰ってみました。意外な発見がありましたでしょうか？

筆者は「日刊Emacs」以外にもEmacs病院兼メルマガのサービスを運営しています。Emacsに関すること関しないこと、わかる範囲でなんでもお答えします。「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。登録はこちら。

➡ http://www.mag2.com/m/0001373131.html SD

ShowNetが示すネットワークの近未来

第2回 ネットワーク設計方法と今年の取り組み

インターネット技術とビジネスが出会う国内最大のイベント「Interop Tokyo」。ほかでは類を見ないその最大の特徴である“ShowNet”は、会場全体に構築される最先端の技術を駆使したネットワーク環境です。この連載では2015年6月の開催に向けて動き始めたShowNetについて紹介していきます。2回目の今回は、次回の設計計画の一部をお伝えしながら、ShowNetで培ってきたネットワーク設計方法のノウハウを披露いたします。

Writer 中村 遼（なかむら りょう）
東京大学
Mail upa@haeena.net
Writer 渡邊 貴之（わたなべ たかゆき）
ジュニパーネットワークス（株）
Mail taka@interop-tokyo.net
URL http://www.interop.jp

Interop TokyoのShowNetは、インターネットの世界を模してつくられています。その中で、最先端の装置のお披露目や、インターオペラビリィ（相互運用性）の実験などを実施しています。それらがフォーカスされていてあまりご存じない方もいらっしゃるかと思いますが、実は出展社様のデモに使われるネットワークや来場者様にネットワーク接続を提供しており、サービスという面でもShowNetが運用されています。

そのため、ShowNetではデモなどの要望に応じたさまざまなサービスを提供しています。（**表1**）。そのため、通信キャリアやISP、構内のネットワークという具合にインターネットの世界を模していき、エンド・ツー・エンドでの接続性を担保するネットワークを構築します。加えてサービスとして運用するため冗長性や運用性を考慮した、柔軟なネットワーク設計が求められています。今回は、ネットワーク設計がどのようにされているのかと、今年の取り組みについて紹介します。

▼表1　ShowNetが提供するネットワークサービス

ShowNetが提供するネットワークサービス
●100Mbps/1Gpsイーサネット（100Base-TX/1000Base-T）
●10Gbpsイーサネット（10G Base-LR）
※上記接続の際のIPアドレスのオプション IPv4 Private IPv4 Global IPv6 Global IPv4/v6デュアルスタック

出展ブース間ネットワーク
●VLAN接続
●UTP、光ファイバケーブルによる接続

カスタムオーダー／出展ブースデモ支援サービス	
例：	●リアルタイムで攻撃への対策を行うデモンストレーション
	●認証の可視化と高性能を示すデモンストレーション
	●SNMPによるShowNet機器データ取得によるデモンストレーション

無線インターネットサービス
●802.11ac/n/aなどによる出展社、来場者向けのサービス

ShowNet設計のココロ

ShowNetでは、ネットワーク構成を、「エクスターナル」、「バックボーン」、「お客様収容」、「データセンタ」などのブロックに分けて設計をしていきます。それぞれのブロックでは現在運用されているもの、現状の問題点を提起するもの、これから活用されるべき通信・運用技術などを考慮して実際の構成に落とし込まれていきます。

たとえば、読者の皆さんに一番身近である「お客様収容」の部分では、数年前まではスイッチのスタッキング技術が安定しておらず、スイッ

チを束ねて仮想化し1台として利用することが難しく、挑戦的な取り組みでした。しかし肥大化するエッジスイッチに対して、ネットワーク運用者は、OSの管理、コンフィグの管理、スイッチの管理など管理コストの削減が急務となっています。そこで、スタッキング技術の啓蒙を行うと共に実際に運用構築を実施し、その検証結果を各メーカーの方々と共有してきました。その結果、徐々に各社のスタッキング技術は改善、進化し、実運用に耐えうる技術となりました。

このように各ブロックにおいてそれぞれ冗長性、運用性、技術的な挑戦を考慮しながら設計を実施し、ブロック間の接続でもさらに冗長性、運用性を考慮したうえでShowNet全体のネットワークを構成していきます。

今年のネットワーク設計と注目技術

今年のL2/L3では、とくにエクスターナルにおけるセキュリティ技術に焦点を当てています。近年ではDDoSが非常に猛威を振るっています。エクスターナルブロックではBGP flow specという技術を使用し、DDoSを軽減する取り組みを実施していきます。

通常、ルータのルーティングテーブルはIPのセグメント情報を元に作成されます。これまではBGPのNexthopをNullに向けた経路を広報することでDDoS対策としてきましたが、これでは正常なトラフィックもすべて遮断されてしまうという問題点がありました。そこで、Flow情報を広報することでソースアドレスやL4のポート情報なども対象に加え、より柔軟に特定のトラフィックを遮断できるようにします。

しかし単にFlow情報といっても、もともとルータにはそのような機能はありません。そのためBGPの拡張やルータの実装が必要になります。今年のShowNetでは、これらの拡張の実運用に挑戦し、より安全なインターネットの環境を目指していきます。

また、BGPでのRPKI[注1]にも取り組んでいきます。こちらは経路そのものの信頼性の向上を図っています。特定のネットワークアドレスをどの機関が保有しているのかを、AS(Autonomous System)と紐付けて証明書でやりとりをするものです。このRPKIによって自ASが広報しているアドレスの他人による乗っ取りを防止できます。

大規模イベントネットワークならではの構築ノウハウ

ShowNetは、ネットワーク疎通性の提供や最新技術の実証実験という側面に加えて、大規模イベントネットワーク特有の側面があります。こうした2つの性質を持つShowNetを構築するための数々のノウハウは、安定した接続性を提供しつつ多種多様な実験を内包するShowNetを“さまざまな背景を持つ多くのエンジニアが、2週間で構築し、そして壊す”というイベントネットワークとしての性質から生まれたものです。ここでは、そういったShowNet特有のノウハウから特徴的なものをいくつか紹介します。

短期決戦を戦い抜くための必需品“トポロジ図”

ShowNetでは、5月末から約2週間弱という短期間ですべての構築と実験を行います。その中でも、いわゆるバックボーンネットワークは、出展社様への接続性提供はもちろん、さまざまな実験に必要な、ShowNetの背骨として真っ先に構築されます。

図1に2013年のトポロジ図から、バックボーントポロジの一部を紹介します。短期決戦であるShowNetの構築現場では、“構築に必要な情報はこのトポロジ図を見ればすべてわかる”という状態にもっていくことがとても大切です。実際のネットワークの運用現場でもそうですが、

注1) Resource Public Key Infrastructure：IPアドレスやAS番号リソースに関して、正しい所有者が誰なのかを証明するための認証基盤。

必要な資料が複数の個所に散逸すると、各資料の更新への追従や、そもそも必要な情報を探すのに時間がかかってしまいます。構築期間の限られているShowNetにおいて、構築に携わるすべてのメンバーに必要な情報を齟齬なく1つの資料で共有するために生まれたのが、このトポロジ図です。

トポロジ図を用いた具体的な情報管理と設定

このトポロジ図に、たとえばルータ間のリンクのインターフェース情報とIPアドレス、ループバックアドレスなどの情報を埋め込むための工夫をShowNetでは、「装置ID」と「リンクID」と呼んでいます。

図1のトポロジ図の中で、丸にクロスした矢印アイコンがレイヤー3のルータを示しています。各ルータのすぐ近くに、点線で丸く囲われた「.X」という数字が装置IDです。この装置IDは、ShowNetバックボーンでOSPF[注2]に参加するすべての装置に割り当てられています。たとえば、図左上のasr9kの装置IDは3、右上のax86rの装置IDは4、になります。次に、各ルータ間のリンクの真ん中にある3桁の数字がリンクIDを示しています。asr9kとax86rの間のリンクIDは101、asr9kとax6708sの間のリンクIDは105です。ここから、各ルータのOSPFに必要な情報を取り出すことができます。

まず各装置のループバックアドレスを、「45.0.0.装置ID/32」というルールで生成します[注3]。たとえば、**図1**中のasr9kには45.0.0.3、ax86rには45.0.0.4というループバックアドレスを設定すればよいことがわかります。このルールのために、ShowNetでは毎年45.0.0.0/24のアドレス空間をループバックアドレス用のレンジとして割り当てています。

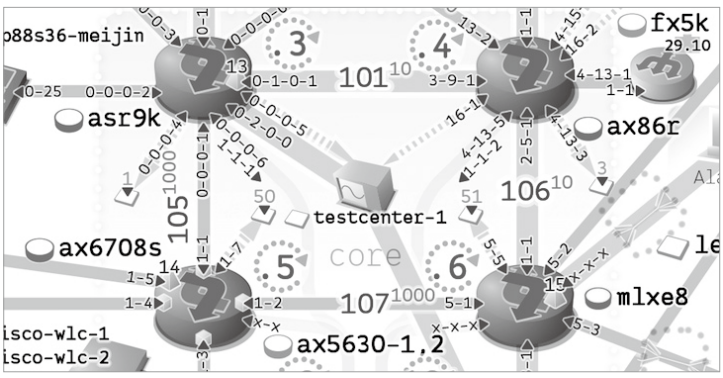
次に、リンクIDから各ルータ間のOSPF用リンクのIPアドレスを生成します。リンクIDXYZのリンクには、OSPF用として、VLAN ID「XYZ」、ネットワークアドレス「45.0.X.(YZ×4)/30」というルールでネットワークを作ります。たとえば、asr9kとax6708s間のリンクIDは105なので、VLAN ID 105、ネットワークアドレスは45.0.1.20(05×4)/30となります。リンクIDから生成されるネットワークアドレスは/30なので使えるIPアドレスは2つ、リンクIDが105の場合は45.0.1.20/30で45.0.1.21と45.0.1.22が使えます。どちらのアドレスをどちらの装置につけるか決定するために、ShowNetでは"装置IDが若番のほうに小さいアドレスをつける"というルールを決めています。

これによって、asr9kとax6708s間のOSPF用ネットワークは、VLAN ID 105でasr9k側が45.0.1.21/30、ax6708s側が45.0.1.22/30、

注2) Open Shortest Path First：組織内で使われる動的経路制御プロトコルの1つであり、ルータ間で経路を動的に学習するのに用いられる。

注3) ShowNetは45.0.0.0/15というアドレスブロックを持っています。

▼図1　2013年のトポロジ図からバックボーンの一部



※http://interop.jp/2013/shownet/point.html#topology から完全なトポロジ図をダウンロードできます

ということがトポロジ図から読み取れます。このようなルールでリンクIDを利用するため、リンクIDは100番(VLAN ID 100、45.0.1.0/30)から163番(VLAN ID 163、45.0.1.252/30)までの64個が使えます。ここ数年は、64個のリンクIDでは足りない場合もあるため、リンクID 200番台(45.0.2.0/30から45.0.2.252/30)と300番台までの128個を加え、最大192個のリンクIDを用途に応じて使い分けています。

また、トポロジ図上のリンクIDの右上にある数字は、OSPFのコストを表しています。OSPFのHello intervalやDead interval、認証といった設定情報は、毎年ShowNet全体で共通として事前に値を決めています。

以上のルールで、トポロジ図から各ルータ間のOSPF用ネットワークの情報を読み取ることができました。この装置IDとリンクIDのように、ほかに必要な情報もトポロジ図から読み取ることができます。たとえば物理接続に必要な情報ですが、まずリンクを示す線から物理線の速度がわかります。そして、装置に向かう三角形の色からファイバーがシングルモードかマルチモードか、また三角形から続いて表記してある数字からその装置のインターフェース番号がわかります。リンクID 105のリンクはasr9k側は10Gigabit Ethernetの0/0/0/1番ポート、ax6708s側は同じく10Gigabit Ethernetの1/1番ポートです。

ここまでわかれば、構築現場でこのトポロジ図だけをたよりに、装置に光ファイバーを接続し、VLANインターフェースを作り、IPアドレスを設定し、OSPFの設定をすることができます。

奥深きShowNetのトポロジ図

このようにShowNetでは、大規模イベントネットワーク構築という短期決戦を乗り越えるためにさまざまな工夫を行っています。装置IDとリンクIDのほかにも、たとえばトポロジ図からリンク・アグリゲーションの設定や装置間スタックの設定、果ては光タップの接続や仮想ルータなど、さまざまな情報を読み取ることができます。

このShowNetの工夫の結晶とも言えるトポロジ図は、長年ShowNetに携わってきた多くのエンジニアによって洗練されてきました。ShowNetのブログでその経緯を見ることができます[注4]。こちらの動画[注5]では、1994年から2013年までのトポロジの変遷を見ることができます。

2015年も、さまざまな組織から参加する多くのエンジニアと、20年という時間の中で培ってきたさまざまなノウハウによって、多くの実証実験やデモンストレーションと、安定したネットワーク疎通性の提供を目指しています。現在、本番に向けて鋭意準備中です。2015年6月は、実際に稼働するShowNetとともに、幕張メッセにて皆様のご来場をお待ちしております。

次回予告

次号はデータセンター、クラウド担当メンバーによる今年の取り組みについて紹介させていただく予定です。ぜひご期待ください。SD

注4) http://www.f2ff.jp/interop/2013/noc/-shownet-topology-map1.php

注5) https://www.youtube.com/watch? v=aAshcnSSOGQ

COLUMN

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第19回 ❖安定動作につながるディレクトリの知識（その3）

◉著者プロフィール

後藤 大地（ごとう だいち）

BSDコンサルティング（株） 取締役／（有）オングス 代表取締役／FreeBSD committer

エンタープライズシステムの設計、開発、運用、保守からIT系ニュースの執筆、IT系雑誌や書籍における執筆まで幅広く手がける。カーネルのカスタマイズから業務に特化したディストリビューションの構築、自動デプロイ、ネットワークの構築など、FreeBSD/Linuxを含め対応。BSDコンサルティングでは企業レベルで求められるFreeBSDの要求に柔軟に対応。

ファイルの削除にも性能の差

ファイルを活用したシステム構築を行う場合、ファイルの作成のみならず、ファイルの削除にかかる時間が性能を左右することがあります。たとえば何かしらのモニタリングシステムから上がってくるデータを秒ごとにファイルに落とし込む実装にしてあれば、1年間で31,536,000個ほどのファイルができることになります。対象となるモニタ機材が増えれば、1年間で作成されることになるファイルの数は数億個の桁になってきます。

ファイルシステムごとに削除の方法は異なります。UFS系のファイルシステムの場合、メタデータであるi-nodeの書き換え、または削除、ディレクトリのアップデート、データブロックの解放といった処理が行われます。メタデータの書き換えのみならず、使用していたデータブロックが解放されたことをすべてのデータブロック分書き込みます。このため、削除するファイルサイズが大きくなればなるほど、削除に時間がかかります。

ZFSの削除はUFSとは異なります。UFSの削除処理よりもZFSの削除処理のほうが複雑ですが、素のUFSよりも高速です。ZFSはUFSのようにデータブロックが解放されたことをすべてのデータブロックに対して実施するといったことはしませんので、削除するファイルのサイズが大きくなってもUFSのようには処理時間が増えません。

しかし、UFSはSoft updatesの機能を有効化すると性能が一気に変わります。削除特性はUFSのままですが、処理速度が一気に高速になります。削除するファイルのサイズが小さい場合、削除処理はZFSよりも高速になります。ただし、ファイルサイズが大きくなってくると、ZFSのほうが高速です。

大量のファイルを削除するベンチマーク

作成するファイルを1万個から90万個まで1万単位で増やしながら、削除にかかる時間がどのように変化するかを調査します。前回[注1]ベンチマークソフトを作りましたので、それをベースに書き換えたベンチマークソフト（bench1.c）を使います（p.160の**リスト1**[注2]）。

調査するファイルシステムはUFS2（Soft updates機能有効）、UFS2（Soft updates機能無効）、ZFSの3つとします。

ファイルサイズと削除時間を比較するベンチマーク

さらに今回は、削除するファイルのサイズが変わった場合、削除性能にどのような違いが現れるかを調べます。こちらも先ほどのベンチマークのソースコードをベースに、新しく書き換えたベンチマー

注1 本誌2015年4月号 第18回のリスト1。

注2 C言語がわからない、またはC言語が苦手だという方はコメントで処理を追ってみてください。

各コマンドのうしろについている括弧書きの数字は、manコマンドで見ることができるマニュアルに記載されている章番号を表しています。

クソフト（bench2.c）を使います（p.161の**リスト2**）。

大量のファイル削除

bench1の実行結果が**図1**になります。スケールをかえてUFS2（Soft updates機能有効）とZFSのグラフを見やすくしたのが**図2**です。

1万個から90万個までファイルを削除する時間を計測すると、UFS2（Soft updates機能無効）はリニアに削除時間が増える傾向を示しました。また**図2**から、UFS2（Soft updates機能有効）は65万個を超えたあたりでいっきに性能が悪化し、ふたたびリニアに処理時間が増えています。今回は詳しく調査していませんが、Soft updates関連の実装で何らかの閾値（しきいち）を超えたものと考えられます。

ZFSはSoft updatesの有効になったUFS2よりは遅いですが、UFS2とは逆にファイルの数が一定数を超えた段階で処理が高速になる瞬間があります。UFS2ではこれ以上ファイルが増えると一気に処理が低速になるので、さらにファイル数が増えるとUFS2とZFSの性能は逆転するものと考えられます[注3]。

ファイルサイズと削除時間

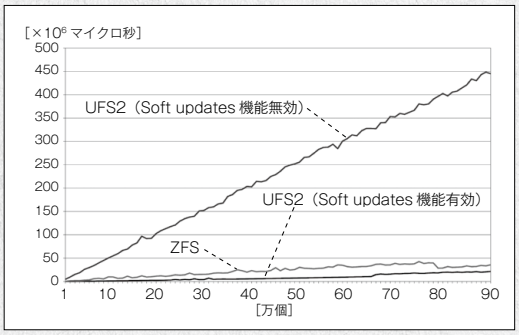
次に、bench2の実行結果を**図3**に示します。

UFS2ではファイルのサイズが大きくなると削除にかかる時間も増えています。Soft updatesの有無で性能は大きく違いますが、それでもZFSよりも高速になるのはファイルサイズが小さいときだけです（**図4**）。

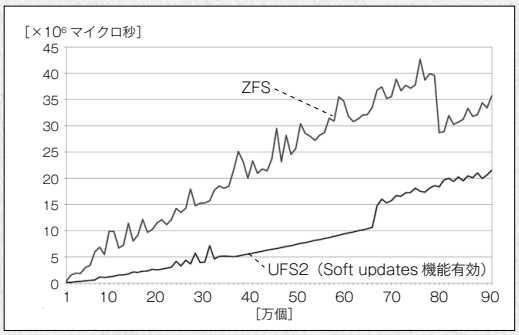
逆にファイルサイズが大きくなっても、そこまで性能に違いが現れないのがZFSです。ただし、ZFSはファイルサイズが16MBを超えると一気に性能が劣化しています。これはARC（ZFSのキャッシュ）を超えてディスクアクセスが頻発する

注3　本誌2015年4月号 第18回のベンチマーク結果も考慮したうえでの予想です。

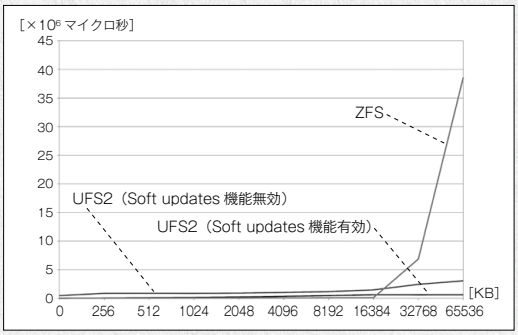
▼図1　bench1実行結果



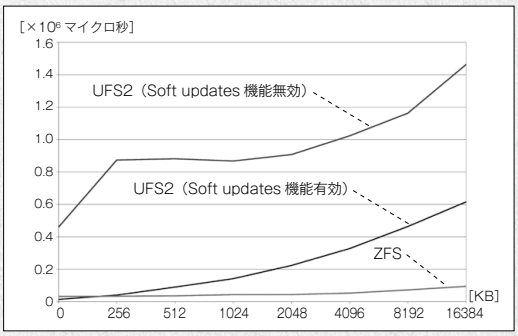
▼図2　bench1実行結果（UFS（Soft updates機能有効）とZFSのみ）



▼図3　bench2実行結果



▼図4　bench2実行結果（16MBまで）



ようになったためだとみられます。メモリ容量を増やすとこの上限はあがり、逆にメモリサイズを減らすと、もっと早い段階でこうした挙動を見せるものとみられます。

▼リスト1　ベンチマーク1：bench1.c

```
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/dirent.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <err.h>
#include <sysexits.h>

int
main(int argc, char *argv[])
{
    struct timeval tv1, tv2;
    char buf[MAXNAMLEN + 1];
    int i, j, fd;

    for (j = 10000; j <= 900000; j += 10000) {
        /* ファイルを作成するディレクトリを作成 */
        mkdir("test", 0755);
        /* 作成したディレクトリへ移動 */
        chdir("test");
        /* ファイルを作成 */
        for (i = 1; i <= j; i++) {
            /* ファイル名を用意 */
            snprintf(buf, MAXNAMLEN, "%010d", i);
            /* ファイルを作成 */
            fd = open(buf, O_CREAT, 0644);
            if (fd == -1)
                err(EX__BASE, "%s", buf);
            close(fd);
        }
        /* 作成したディレクトリから抜ける */
        chdir("..");
        /* マイクロ秒単位で現在時刻を取得 */
        gettimeofday(&tv1, NULL);
        /* ディレクトリごと作成したファイルを削除 */
        if (0 == fork())
            execl("/bin/rm", "/bin/rm", "-r", "test", NULL);
        else
            wait(NULL);
        /* マイクロ秒単位で現在時刻を取得 */
        gettimeofday(&tv2, NULL);
        /* 時刻の差分を出力（マイクロ秒単位） */
        printf("%s,%ld\n", buf,
            (tv2.tv_sec - tv1.tv_sec) * 1000 * 1000 +
            (tv2.tv_usec - tv1.tv_usec));
    }
}
```

まとめ

このようにファイルシステムごとに削除の特性が違いますし、同じファイルシステムでも機能の有無で性能に大きな差がでること、同じファイルシステムでもキャッシュなどの閾値を超えると一気に性能が劣化すること、などがわかります。

どういった使い方をすればどのような特性を示すか、UFSであればある程度推測はできますが、ZFSになってくると、これはもはや実際に使用する環境でデモ環境を構築して調査するほうがよいと思います。実機でも仮想環境でもよいのですが、実際に使用する環境で、実際の想定に近い状態で試してみることが大切です。

今回示したようなベンチマークの結果は、条件を変えると性能が突然劣化するポイントといったものが単一のベンチマークのときよりも早く現れることがあります。たとえば別の処理でカーネルが使用できるメモリ容量が少なくなっていれば、それだけキャッシュに回せるサイズも小さくなり、より早い段階でキャッシュを使い切った後の遅い動作が現れることになります。このあたりはオペレーティングシステムのバージョンが変わっただけでも出ますので注意が必要です。

また、最近のマシンは何十もスレッド(論理コア)を搭載していますので、これまで発現しなかったバグがより新しいマシンでは発現するといった現象も出始めていま

す。カーネルの内部は常にマルチプロセッサ対応を進めていますので、最新版になるほど性能が発揮できる反面、こうした新しい問題（デッドロックなど）が出ることがあります。

大切なのは「実際の実機」で「実際の状況」で試験することです。

ファイルの効率のよい活用は、システムを効率よく動作させる要です。みなさんも実際の環境で試してみてください。SD

column

大量にファイルがある場合の削除：ZFSデータセットを削除という方法

ディレクトリ構造が複雑で数十万から数億といった数のファイルが存在するようなディレクトリを削除しようとした場合、削除にかかる時間は数十分から数時間といったスケールになることがあります。

このような場合、削除対象をZFSのデータセットの上で展開し、ディレクトリを削除する代わりにデータセットを削除するという方法があります。数秒で処理が終わりますので、こちらのほうがよほど高速です。UFSであればファイルを展開する場所を1つのマウントポイントとしておき、削除する代わりにアンマウントumount(8)とフォーマットnewfs(8)するという方法もあります。

▼リスト2　ベンチマーク2：bench2.c

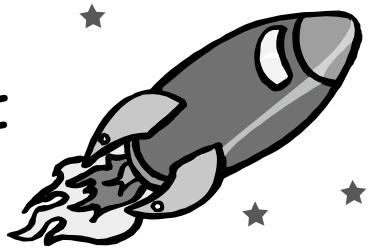
```
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/dirent.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <err.h>
#include <sysexits.h>

int
main(int argc, char *argv[])
{
    struct timeval tv1, tv2;
    const int size[] = {
            0, 256, 512, 1024, 2048, 4096, 8192,
            16384, 32768, 65536
    };
    const int len = sizeof(size) / sizeof(int);
    char buf[MAXNAMLEN + 1];
    FILE *fp;
    int i, j, k, l;

    for (j = 0; j < len; j++) {
        /* ファイルを作成するディレクトリを作成 */
        mkdir("test", 0755);
        /* 作成したディレクトリへ移動 */
        chdir("test");
        /* ファイルを作成 */
        for (i = 1; i <= 1000; i++) {
            /* ファイル名を用意 */
            snprintf(buf, MAXNAMLEN, "%010d", i);
            /* ファイルを作成 */
            fp = fopen(buf, "w");
            if (fp == NULL)
                err(EX__BASE, "%s", buf);
            /* 中身を書込 */
            for (k = 0, l = size[j] * 1024; k < l; k++)
                fputc('c', fp);
            fclose(fp);
        }
        /* 作成したディレクトリから抜ける */
        chdir("..");
        /* マイクロ秒単位で現在時刻を取得 */
        gettimeofday(&tv1, NULL);
        /* ディレクトリごと作成したファイルを削除 */
        if (0 == fork())
            execl("/bin/rm", "/bin/rm", "-r", "test", NULL);
        else
            wait(NULL);
        /* マイクロ秒単位で現在時刻を取得 */
        gettimeofday(&tv2, NULL);
        /* 時刻の差分を出力（マイクロ秒単位） */
        printf("%d,%ld\n", size[j],
            (tv2.tv_sec - tv1.tv_sec) * 1000 * 1000 +
            (tv2.tv_usec - tv1.tv_usec));
    }
}
```

26

Debian Developer　やまねひでき　henrich@debian.org

Debian 8開発の最新動向とそのほかのトピック

Debian Hot Topics



Debian 8は4月25日がリリースターゲット

リリースチームのNiels Thykier氏により「Debian 8"Jessie"を4月25日にリリースする」と案内が行われました。注意点として「testingからの自動削除ツールは動作し続けている」こと(つまり、RCバグがある場合はそのままでは削除されるので修正が必要)、RCバグの修正は4月18日までに実施する必要があること、期限を過ぎた場合は「pu(proposed updates)」タグでrelease.debian.org擬似パッケージに対しdiffを付けてバグ報告してStableリリースチームにコンタクトをとってほしいことが挙げられています。

「reproducible build」への取り組み

OSSの良いところとして、誰もがソースコードを確認できるという点は多くの人がうなずくところでしょう。これにより、バックドアなどの悪意あるコードが含められていないかという監査が、そうでない場合よりも容易になるという利点があります[注1]。

しかし、Debianなどの多くのディストリビューションでは、ユーザに対してソースコードからビルド済みの「バイナリパッケージ」を提供しています。ここで「本当に、提供されているバイナリは開示しているソースコードから生成されているのか？」という疑問が出てきます。実はソースコードをビルドしてみるとわかるのですが、ビルド環境によっては毎回違うバイナリが生成されてしまうことも稀(まれ)ではありません。

これに対して「同じソースコードであれば、必ず同じバイナリが生成できるようにしよう」という取り組み「reproducible build」が行われました。これを担保することで、アップロード権限を持つ開発者のシステムに侵入して悪意のある細工を施したバイナリをアップロードさせるような「標的型」攻撃のモチベーションを下げるなどが期待されています。また、地理的に独立および管理者権限が独立した複数のマシンを所有しているような場合には、必ず同一のバイナリが生成されることから、どのシステムに侵入されてバイナリに細工を施されているのかなどが比較によって容易に検査できるというメリットもあります。

「reproducible build」の実際の作業としては

①パッケージ生成プログラム「sbuild」のラッパー「srebuild」[注2]を作り、全パッケージが保存されているsnapshot.debian.orgから毎回同じビルド環境を生成できるようにする
②タイムスタンプ、ロケール、ホスト名などの変化についてはVMや「libfaketime」を使う

注1)「比較して容易になる」だけであって、監査そのものは容易であるわけでもなく、多くの人的なコストが必要になることは留意してください。OSSであれば安全、というのは短絡的な考えで幻想にすぎません(その逆もしかり、です)。

注2) URL https://wiki.debian.org/ReproducibleBuilds

ことで担保する
③2度ビルドを実施して生成されるバイナリが同一であることを確認する
④Jenkinsを使ってmainアーカイブについて随時処理を実施する

という形で行われました。

この結果、83.5%のソースパッケージについて、reproducible buildが可能であることが確認されました。費用対効果の点からは企業では採用されそうにない[注3]、学術的な研究からのアプローチですが、このような取り組みが実際に行われるのがコミュニティディストリビューションのおもしろいところですね。

cdn.debian.netの運用終了

近年はAWSのSA(ソリューションアーキテクト)として活躍しているDebian開発者の荒木靖宏氏によって作成／運用されていたcdn.debian.netの終了が、debian-develメーリングリストで宣言されました。

今後、cdn.debian.netへの接続は、別実装であるhttp.debian.netへリダイレクトされることになります[注4]。

cdn.debian.netもhttp.debian.netも特定のリポジトリを指定して利用するのが不便な人向けのサービスで、とくに地理的に移動が頻繁な人や、パッケージの初期設定で極端に遅いサーバに接続しないようにする場合に、非常に有効です(逆にそうでない人には、あまりメリットはありません[注5])。

最新Chrome/Chromiumが使えなくなる?

バージョン39以降のChrome/Chromiumに「SECCOMP_FILTER_FLAG_TSYNCをサポートするLinuxカーネル3.17以降(あるいは以前のバージョンに対してパッチを適用したもの)を利用する[注6]」という変更が元Canonical社、現Google社(かつDebian開発者)のKees Cook氏によって導入されました。

SECCOMP_FILTER_FLAG_TSYNCを含むseccomp2というセキュリティ機構がLinuxカーネルに導入されたのは3.5からで、「どのシステムコールを実行できる／できないを設定できる」機能をユーザランドのライブラリであるlibseccomp2パッケージと連携して実現します(そしてSECCOMP_FILTER_FLAG_TSYNCは最近入った拡張です)。もともとChrome/Chromiumのプロセスの一部をサンドボックス化したいという要求からLinuxカーネルに取り入れられた、という経緯があり、Chrome/Chromiumがこの機能を活用していこうとするのは、自然なことではあります。

当然、ユーザからはリリース間近のDebian 8 "Jessie"で対応の希望が出てきましたが、Linuxカーネルパッケージのメンテナである Ben Hutchings氏の答えは、「Sounds like another good reason to not use Google spyware.(Googleのスパイウェアを使わないほうがいい理由がまた増えたみたいだね)」と、にべもないものでした。

また、「Adobe Flash player for Linuxの最新版が使えるのはChromeだけ」という別のユーザからの投稿に対しては、別のカーネルパッケージのメンテナの1人Maximillian Attems氏が「使わないほうがいい理由が、もう1つ増えたね」とこれまたすげなく答える場面も(そしてどうやら両者とも、この時点でまだパッチの中身は見ていないようです)。

注3) ほかのディストリビューション、たとえばRHELでは採用しないだろうと思います。RHELにはいわゆるクローンであるCentOSがありますが、RHELとは完全にバイナリ互換性があるとは保証していないですし、保証できるようなしくみを入れてしまうとRHELの位置づけとメリットが揺らいでしまうのではないでしょうか(邪推ですかね?)。

注4) この変更を行った際、作業の余波でftp.jp.debian.orgの名前解決が一時期不安定になるという問題が起こりました。このような異常が起きた場合は、可能な限り公開メーリングリストでお知らせください。

注5) たとえば、筆者が接続テストを行った際には、期待される日本ではなく、韓国のミラーサーバに接続されました。

注6) オプションを無効にしてビルドと動作ができるのかどうかなどは、ちょっと確認できていません。

自由なソフトウェアではないChromeとFlash以外には大きなメリットが見いだせない変更を、リリース終盤のこの時期にLinuxカーネルという影響が大きいパッケージに導入するのは、メンテナにとってとても受け入れられない変更であろうことは容易に想像がつきます。そのため、このような方向での対応は自然で、タイミングが悪いとしか言いようがないですね（まぁそうは言っても、もうちょっと言い方があるだろう、とは筆者は感じましたが）。また、たとえパッケージメンテナがOKと言ってもリリースチームが拒否する可能性が高いです。

ただ、該当のissue[注7]については「seccomp-bpf[注8]をサポートしているにもかかわらずSECCOMP_FILTER_FLAG_TSYNCが有効になっていない場合は、この問題が発現する可能性があるのではないか」、「いや、TSYNCが無効になっているLinuxカーネルであっても問題なく動作するはずだ。Chromeのdevチャンネルのビルドでも問題なく動いている[注9]」などと情報が錯綜しています。実際、筆者の環境では、多少挙動が怪しい機能拡張もあるもののChromeは問題なく動作しており、検証のために導入したChrome beta（バージョン42）、Chrome dev（バージョン43）でも、Chromiumのバグレポートでクラッシュするという話が出ていた機能拡張のインストールも問題ありませんでした。

実際に自分の環境でセキュリティ機構がどのような状態になっていてChrome/Chromiumが動作しているのかについては、アドレスバーに「chrome://sandbox/」と打ち込むことで確認ができます（**図1**）。

より新しいカーネルが必要となった場合に、ユーザがとれる現実的な対応としては、今後リリースされるであろうtesting/unstableのLinuxカーネルパッケージを導入することでしょうか[注10]。あるいは、自身でカーネルをビルドしてみる、というのもありかもしれません。

Chromiumについては、また別の話で「可能であれば変更を戻すパッチを精査して当てるつもりである」とChromiumパッケージメンテナのMichael Gilbert氏が述べています。

技術委員会の新メンバーとプロジェクトリーダー選挙

systemdの問題の余波もあり退任した3名の技術委員会の穴を埋めるため、Sam Hartman氏、

注7）URL https://code.google.com/p/chromium/issues/detail?id=401655

注8）seccomp2の特徴である、BPF（Berkeley Packet Filter）をシステムコールに利用する機能のこと。Chrome/ChromiumのほかにはOpenSSHやvsftpdがサポートしています。

注9）ChromeはDebianでのstable/testing/unstableのように、stable/beta/dev（unstable）/canaryという形でリリースが分かれています。devチャンネルで動くということは、Debianで言えばunstableでは問題がないということで、stableで問題が起こるのはかなり奇妙です。おそらくはLinux上のChromeにおける普遍的な問題ではなく、ユーザの環境にかなり依存する事象ではないでしょうか。

注10）本誌2014年5月号の本連載にて、stableを使いながらtesting/unstableのパッケージを導入する方法を記載していますので、バックナンバーを見ることができる方は参考にしてみてください。

▼図1　chrome://sandbox/で現在の状態を確認する（左からstabel/beta/dev）

Tollef Fog Heen氏、Didier Raboud氏の3名が新たに指名されました。毎度こじれた問題ばかりを持ち込まれて解決と決定を要求される技術委員会はタフな仕事ではありますが、へこたれずに活動を続けてもらいたいものです。

そして本稿執筆時には、次のプロジェクトリーダーを決める選挙が行われています。今回はMehdi Dogguy 氏、Gergely Nagy 氏、Neil McGovern氏の3名が出馬しました。3名ともプロジェクト内でさまざまなチームや役割を歴任しており、誰が当選してもおかしくはないように思えます。

本誌が発売されるころには選挙結果が出ていますが、新たなプロジェクトリーダーには大きなビジョンを持つだけではなく、日々の時間を割いて細かな実務をこなせるかどうか、当選後の動きに期待したいと思います。

ディストリビューターの立場とフリーではないライセンス

DebianでPHP関係のパッケージをメンテナンスしているLior Kaplan氏のブログ[注11]によると、これまでDebianではnon-freeとなっていたPHPのJSON実装が、PHP 7[注12]では、喜ばしいことにfreeな実装に置き換えられたそうです。

Lior氏の次のコメントは、Debianのようなディストリビューションとしての立場をよく示していると筆者は思います。

For many the PHP JSON extension license might look like a storm in a teacup, but for many Linux distributions the bits of the free software licenses are very important.
(多くの人にとって、PHP JSONエクステンションのライセンスの件は単なる空騒ぎにすぎないように見えるかもしれないが、多くのLinuxディストリビューションにとってフリーソフトウェアライセンスはとても重要なことなんだ)

PHP 5でのJSONの何が問題だったのかというと、JSONライセンス[注13]を採用した実装であったということに尽きます。これは、一見単なるMITライセンスに見えるのですが、「The Software shall be used for Good, not Evil.」という一文が差し込まれています。これは解釈が非常に厄介で「何を持ってEvilというのか」という線引きは人によって千差万別であり[注14]、ディストリビューションベンダは、配布に際して無用な法的リスクを負うことになります。

Debianではそのような実装を別ライセンスの実装に置き換えたり、削除したりしています(結果、場合によっては性能の劣化や利用不可能な機能が出てきます)。

得てしてユーザ側は、「利便性が高ければそれでいいじゃないか」、「不便なのは嫌だ」という姿勢を取り、ディストリビューターを非難しがちです(面倒なのは誰しも嫌ですからね)。

ただ、Debianのようなコミュニティディストリビューションにおいては、Debianの理想とする「自由なOS」を実現することを主目的にして活動が行われていることを留意していただきたいな、と筆者は考えます。理想と現実のバランスを取りながら活動する中で「判断」が行われていることは、それなりに尊重していただきたいものです(そして、Debianの配布するパッケージで不十分な場合は、自身でソースコードを取得してカスタマイズしたバイナリを利用できるという自由がOSSの利用者にはある、ということも覚えておいてください)。SD

注11) URL https://liorkaplan.wordpress.com/2015/03/13/php7-replaces-non-free-json-extension/

注12) 内部機構の大改造を目標にして失敗したPHP 6はスキップされて、PHP 5の次は7になります。

注13) URL http://www.json.org/license.html

注14) 思考実験になりますが、酒蔵の宣伝サイトでPHP 5ベースのCMSを動かしていたら、アルコールを忌避するある種の宗教や信条にとっては「たいへん邪悪な行為」と映り、ライセンス違反とみなされるかもしれませんね。こんな例はいくつもあります。

Ubuntu 14.04で使用できるUSB無線LANアダプター7選

Ubuntu Japanese Team　あわしろいくや　ikuya@fruitsbasket.info

今回は5GHz帯に対応したUSB無線LANアダプターを7つ検証したので、そのレポートです。

昨今の無線LAN事情

住宅や企業が密集する場所では無線LANのSSIDがたくさん見つかる、という事態は誰しもが体験されていることだと思います。たくさんSSIDが見つかるということは、それだけ電波が出ているということであり、思ったよりも速度が出ない、ということもまたよくある話です。経験上、大部分は2.4GHz帯を使用しているものと思われます。

一方、5GHz帯を使用するIEEE 802.11acに対応したルータ／アクセスポイントが登場してからしばらく経ち、リーズナブルな価格で手に入るようになりました。そうであればルータ／アクセスポイントを買い換えたら速度の問題は一件落着といけばいいのですが、ノートPCや一部のデスクトップPCに搭載されている無線LAN機能が5GHz帯に対応しているかどうかはまた別の話です。最近発売されたPCであれば安価なものでも対応していますが、数年前までは高価なモデルでないと非対応ということがざらでした。そういった場合、USB無線LANアダプタを購入すればいいのですが、Linuxでは原則としてカーネルで対応している必要があり、非対応の場合はドライバ（カーネルモジュール）をビルドする必要があります。どれが認識してどれが認識しないのか、認識しない場合はどうすればいいのかは誰しもが試行錯誤しています。というわけで、今回適当に7つほど見繕って実際に動作するか確認してみました。**表1**が型番とlsusb[注1]の結果です。結論からいえば、すべて動作しました。

なお、今回検証したのはUbuntu 14.04/14.04.1のカーネル3.13です。Ubuntu 12.04.5でも同じカーネルを使用しているので、同じ結果が期待できます[注2]。14.04.2/14.10のカーネルは3.16であり、また違っ

表1　今回取り上げるUSB無線LANアダプタ

メーカー	型番	lsusbの結果
ロジテック／エレコム	LAN-W450ANU2E※1	0789:016b
ロジテック／エレコム	LAN-W300AN/U2※	0789:0170
NECアクセステクニカ／NECプラットフォームズ	AtermWL450NU-AG※3	0409:02f2
プラネックス	GW-900D※4	2019:ab30
プラネックス	GW-450S※5	2019:ab32
プラネックス	GW-450D※6	2019:ab31
ロジテック／エレコム	WDC-433SU2MBK※7	7392:b711

※1　http://www.logitec.co.jp/products/lan/lanw450anu2e/index.php
※2　http://www.logitec.co.jp/products/wlan/lanw300anu2/
※3　http://121ware.com/product/atermstation/product/warpstar/wl450nu-ag/
※4　http://www.planex.co.jp/products/gw-900d/
※5　http://www.planex.co.jp/products/gw-450s/
※6　http://www.planex.co.jp/products/gw-450d/
※7　http://www2.elecom.co.jp/products/WDC-433SU2MBK.html

注1） USB接続デバイスを表示するコマンドです。

注2） このあたりはUbuntuのHWE/HESというしくみに依存してます。本誌2014年6月号の第2特集で詳しく解説しているので、そちらをご参照ください。

た結果になる可能性があります。では、個別にみていきましょう。

LAN-W450ANU2E

とにかく簡単に済ませたい、という場合はこれ一択です。現在は流通在庫のみですので探すのに苦労するかもしれませんが[注3]、その苦労さえなんとかして購入できれば挿すだけで認識し、すぐに使えるようになるという手軽さです。価格も在庫処分価格か、安価で購入できる場合もあります。筆者は1,080円で購入しました。短いながらもUSB延長ケーブルが添付されているのもまたうれしいです[注4]。

チップはMediaTek/Ralink RT3573でした。同じものを使用している場合は、同じく挿しただけで認識するかもしれません。

LAN-W300AN/U2

こちらも挿すだけで認識し、使用できるようになります。ただ、LAN-W450ANU2Eよりもさらに入手性に難があるのではないかと思います。

チップはMediaTek/Ralink RT3572でした。すなわちLAN-W450ANU2Eと1番違いであり、同じドライバで動作しているものと思われます。

AtermWL450NU-AG

こちらは現在使用しているルータ（Aterm WR9500N）の添付品です。チップはMediaTek/Ralink RT3573ですが、残念ながらIDが登録されておらず、自動的には認識しません。IDが登録されているかどうかは、次のコマンドを実行すればわかります。何も表示されなければ、非対応ということです。

```
$ modinfo rt2800usb | grep 0409p02f2
```

とはいえ、実際に使用するカーネルモジュールとベンダーIDがわかっている必要があるので、なかなか難易度が高いです。WikiDevi[注5]に記載がある場合は簡単にわかりますので、一度参照してみるといいでしょう。

必要な情報がそろえば認識させる方法はわりと簡単で、まずは次のコマンドでカーネルモジュールを読み込みます。

```
$ sudo modprobe rt2800usb
```

続いて、IDを登録します。

```
$ sudo bash -c "echo 0409 02f2 > /sys/bus/usb/drivers/rt2800usb/new_id"
```

接続すれば使用できるようになります。これだと起動するたびにコマンドを実行する必要があります。自動的に接続できるようにするには、**図1**のコマンドを実行します。

動作するとはいえ高価ですし、新たに購入するべきかどうかはよく考える必要があるように思います。

GW-900D

USB 3.0対応というのが珍しいので購入してみましたが、自動的には認識しませんでした。ドライバ自体がカーネルにないので、自前で準備する必要があります。とはいえかなり簡単ですし、一度インストールしてしまえば安定して動作します。価格も手ごろでありお勧めではあるのですが、大きいので差

注3） もちろんインターネットでは在庫がある限り簡単に買えますが、店頭で買うのは厳しいかもしれません。

注4） 今どき100円ショップでも買えますが。

注5） https://wikidevi.com/wiki/Main_Page

図1　自動的に接続するようにする

```
$ sudo bash -c "echo rt2800usb > /etc/modules-load.d/WL450NU-AG.conf"
$ sudo bash -c "echo install rt2800usb /sbin/modprobe --ignore-install rt2800usb?; /bin/echo '0409 02f2' ?> /sys/bus/usb/drivers/rt2800usb/new_id > /etc/modprobe.d/WL450NU-AG.conf"
```

し込むUSBポートを考慮しなくてはいけないかもしれません。

ドライバは独自に改良したものをいくつかのWebサイトで配布しているようですが、今回はGitHubにあるgnab/rtl8812au[注6]を使用することにします。まずは右側のペインにある“Download ZIP”をクリックし、ソースコードをダウンロードしてください。以後、ホームフォルダの直下にあるDownloadsフォルダにダウンロードしたものとして解説します。ついでに、左にある“commits”の前の数字を記憶しておいてください。これはコミットの回数です。

ダウンロードが終わったら、端末を起動し、次のコマンドを実行してください。

```
$ cd ~/Downloads
$ unzip rtl8812au-master.zip
```

実行すると“rtl8812au-master”というフォルダが作成されているはずです。今回は“master”だと不都合ですので、これをバージョン相当に変更します。バージョンは何でもいいのですが、わかりやすくコミットの回数とします。今回は“30 commits”だったので、Revision 30、すなわちr30とします。

```
$ mv rtl8812au-master rtl8812au-r30
$ cd rtl8812au-r30
```

普通にUbuntuをインストールした場合は、コンパイラ(gcc)もカーネルヘッダもインストールされているはずですので、いきなりmakeを実行します。

```
$ make
```

コンパイルが無事に終わると、“8812au.ko”というファイルがフォルダの直下にできているはずです。これが今回使用するカーネルモジュールです。引き続きこれを読み込んでみましょう。

```
$ sudo insmod 8812au.ko
```

とくにエラーが表示されない場合は、読み込みができています。あとはGW-900DをUSBポートに接続すれば認識し、普通に使えるようになるでしょう。Network Managerから確認してみてください。

このままだと起動するたびにコマンドを実行する必要があり、またカーネルのアップデートがあった場合は再コンパイルが必要になります。とても面倒ですので、DKMS(Dynamic Kernel Module Support)というしくみを使うようにします。まずはdkmsパッケージをインストールします。

```
$ sudo apt-get install dkms
```

続いてソースコードを/usr/srcにコピーします。その前にコンパイルしたファイルを削除し、カーネルモジュールをアンロードします。

```
$ make clean
$ sudo rmmod 8812au
$ cd ../
$ sudo cp -r rtl8812au-r30 /usr/src
```

これで/usr/src/rtl8812au-r30/というフォルダが作成されました。この直下にdkms.confを作成します。

```
$ cd /usr/src/rtl8812au-r30/
$ sudo editor dkms.conf
```

内容は**図2**のとおりです。

気をつけるべきは、PACKAGE_VERSION変数でしょう。“30 commits”でない場合は、適宜修正してください。

あとはDKMSで使えるようにするため、登録します。次のコマンドを実行してください。「-m」でフォルダ名(パッケージ名)を、「-v」でバージョンを指定します。

```
$ sudo dkms add -m rtl8812au -v r30
$ sudo dkms build -m rtl8812au -v r30
$ sudo dkms install -m rtl8812au -v r30
```

ドライバがアップデートされた場合など、DKMSの登録を解除したい場合は、事前に次のコマンドを実行してください。

```
$ sudo dkms uninstall -m rtl8812au -v r30
$ sudo dkms remove rtl8812au/r30 --all
```

注6) https://github.com/gnab/rtl8812au

アップデートする場合は、このあとソースコードのダウンロードから行ってください。

GW-450S

搭載チップはRealtek RTL8811AUで、Realtek RTL8821AUを採用するGW-900Dとまったく同じ方法で使用できるようになります。小さいぶん速度が落ちるのは否めないので、速度よりも大きさを重視する場合はこちらを選択するといいでしょう。

GW-450D

これも自動では認識せず、ドライバのインストールが必要です。チップはMediaTek MT7610Uです。やはりドライバもいくつかのWebサイトで公開されていますが、今回はsanrath / MediaTek_mt7610u_STA_driver_Linux 64bit[注7]からダウンロードします。左の“ダウンロード”をクリックし、右に表示される“Download Repository”をクリックしてください。すると“sanrath-mediatek_mt7610u_sta_driver_linux-64bit-116843043b1b.zip”のようなファイル名でダウンロードされます。また、“コミット”をクリックしてコミットの回数を数えておいてください。これをバージョンとして使用します。今回は7とします。以降、ホームフォルダの直下にある“Downloads”フォルダにダウンロードしたものとして解説します。ダウンロードが完了したら、端末を起動して**図3**のコマンドを実行してください。

ひとまずファームウェア（というか、設定ファイル）をコピーします。

```
$ sudo mkdir -p /etc/Wireless/RT2870STA
$ sudo cp RT2870STA.dat /etc/Wireless/RT2870STA
```

ベンダーIDを登録するため、ソースコードを変更します。

```
$ editor common/rtusb_dev_id.c
```

38行目から、**リスト1**のようになっています。

これを**リスト2**のように変更します。冒頭の空白はタブです。

保存後、makeを実行します。

注7）https://bitbucket.org/sanrath/mediatek_mt7610u_sta_driver_linux-64bit/overview

図2　dkms.confの内容

```
PACKAGE_NAME="rtl8812au"
PACKAGE_VERSION="r30"
CLEAN="make clean"
BUILT_MODULE_LOCATION[0]="./"
BUILT_MODULE_NAME[0]="8812au"
MAKE[0]="cd ${dkms_tree}/${PACKAGE_NAME}-${PACKAGE_VERSION}; make KVER=${kernelver}"
DEST_MODULE_LOCATION[0]="/updates/dkms"
AUTOINSTALL="yes"
```

図3　ダウンロードしたファイルを解凍する

```
$ cd ~/Downloads
$ unzip sanrath-mediatek_mt7610u_sta_driver_linux-64bit-116843043b1b.zip
$ mv sanrath-mediatek_mt7610u_sta_driver_linux-64bit-116843043b1b mt7610u_sta_driver_linux-r7
$ cd mt7610u_sta_driver_linux-r7
```

リスト1　common/rtusb_dev_id.cの38行目からの内容

```
#ifdef MT76x0
        {USB_DEVICE(0x148F,0x7610)}, /* MT7610U */
        {USB_DEVICE(0x13B1,0x003E)}, /* MT7610U */
        {USB_DEVICE_AND_INTERFACE_INFO(0x0E8D, 0x7630, 0xff, 0x2, 0xff)}, /* MT7630U */
        {USB_DEVICE_AND_INTERFACE_INFO(0x0E8D, 0x7650, 0xff, 0x2, 0xff)}, /* MT7650U */
#endif
```

```
$ make
```

makeが終了すると、os/linux/mt7610u_sta.koができています。これが今回使用するカーネルモジュールです。次のコマンドで読み込んでみましょう。

```
$ sudo insmod os/linux/mt7610u_sta.ko
```

あとは実際に接続し、動作を確認してみます。うまく動作しない場合は、dmesgコマンドで確認してみてください。問題がなければ、DKMSを使用するようにします(**図4**)。dkmsパッケージをインストールしてない場合は、事前にインストールしてください。

dkms.confの内容は**リスト3**のとおりです。

DKMSに追加します(**図5**)。

ドライバをアップデートする場合は、事前にDKMSの設定を削除してください(**図6**)。

WDC-433SU2MBK

すでにお気づきかもしれませんが、GW-450Dと完全に同じ方法で動作するようになります。さらにカンのいい方はお察しのことと存じますが、実はお勧め順になっています。すでにお持ちの場合はさておき、新規に購入する場合はMT7610U搭載モデルよりもRT3572/RT3573搭載モデルのほうがお勧めです。けっこう長い時間使用しましたが、前者よりも後者のほうが圧倒的に安定していました[注8]。SD

注8) smbのアクセスだけがおかしかったので、環境依存の可能性はありますが。

リスト2　変更後

```
#ifdef MT76x0
        {USB_DEVICE(0x148F,0x7610)}, /* MT7610U */
        {USB_DEVICE(0x13B1,0x003E)}, /* MT7610U */
        {USB_DEVICE(0x2019,0xAB31)}, /* Planex GW-450D */        ←この行を追加
        {USB_DEVICE(0x7392,0xB711)}, /* Elecom WDC-433SU2MBK */  ←この行を追加
        {USB_DEVICE_AND_INTERFACE_INFO(0x0E8D, 0x7630, 0xff, 0x2, 0xff)}, /* MT7630U */
        {USB_DEVICE_AND_INTERFACE_INFO(0x0E8D, 0x7650, 0xff, 0x2, 0xff)}, /* MT7650U */
#endif
```

リスト3　dkms.confの内容

```
PACKAGE_NAME="mt7610u_sta_driver_linux"
PACKAGE_VERSION="r7"
CLEAN="make clean"
BUILT_MODULE_LOCATION[0]="os/linux/"
BUILT_MODULE_NAME[0]="mt7610u_sta"
MAKE[0]="cd ${dkms_tree}/${PACKAGE_NAME}-${PACKAGE_VERSION}; make LINUX_SRC=${kernel_source_dir} \
LINUX_SRC_MODULE=/lib/modules/${kernelver}/kernel/drivers/net/wireless/"
DEST_MODULE_LOCATION[0]="/updates/dkms"
AUTOINSTALL="yes"
```

図4　DKMSの設定の前処理

```
$ sudo rmmod mt7610u_sta
$ make clean
$ cd ../
$ sudo cp -r mt7610u_sta_driver_linux-r7 /usr/src
$ cd /usr/src/mt7610u_sta_driver_linux-r7
$ sudo editor dkms.conf
```

図5　DKMSへの追加

```
$ sudo dkms add -m mt7610u_sta_driver_linux -v r7
$ sudo dkms build -m mt7610u_sta_driver_linux -v r7
$ sudo dkms install -m mt7610u_sta_driver_linux -v r7
```

図6　DKMS設定の削除例

```
$ sudo dkms uninstall -m mt7610u_sta_driver_linux -v r7
$ sudo dkms remove mt7610u_sta_driver_linux/r7 --all
```

第38回

Linux 3.19の新機能
Intel MPX機能のカーネル側対応

Text：青田 直大　AOTA Naohiro

前号で「Linux 3.20の開発が続いています……」と書きましたが、その後、Linus氏がGoogle+で行った投票[注1]によってLinux 3.20はLinux 4.0と改名されることになりました。

Google+の投票で選んだことからわかるとおり、2.6.Xから3.Xになったときのように4.Xへの移行も「大きな機能の変化」によるものではありません。2.6.X時代は2.6.39までリリースされてから3.0へと移行しましたが、3.Xは3.19までで終わりを告げたことになります。この調子で今後もX.0からX.19までがリリースされ、(X+1).0に移行するという流れとなるのでしょうか。今から4年後のバージョンが楽しみです。

さて、今回からはLinux 3.19に入った機能を紹介していきます。今回取り上げるのは、CPU側でバッファオーバーフローを検知するIntel MPX機能のカーネル側対応部分です。

不正なメモリアクセスへの対策

昔から、バッファオーバーフロー(またはアンダーフロー)は大きな問題となっています。想定された範囲外へのメモリアクセスによって、プログラムが落ちるなどのバグが発生したり、場合によっては権限昇格を可能にするなどセキュリティ上の問題が発生することもあります。安全なコードにするために、メモリアクセスがバッファの範囲内であるかを確認しなければならない、とは長く言われていますが、人間のすることですからどうしてもバグが発生してしまうというのが現状です。

この問題への対策として、mudflapやAddress Sanitizerといったソフトウェアによるメモリ検査ツールが開発されています。これらのツールは、コンパイラの助けを借りて、すべてのメモリアクセスの前に範囲チェックのコードを挿入することで、その機能を実現しています。

AddressSanitizerの場合どうなるかを見てみましょう。**リスト1**のようなプログラムをAddress Sanitizerありとなしでコンパイル(「-fsanitize=address」を使って有効にします)し、ポインタをdereferenceしている関数foo()の部分がどのように変化するかを見てみましょう。

AddressSanitizerを有効にした場合の、元のコードに対応する部分は0x1f、0x25の行です(**図1**)。その前にはアドレスを検査するコードが、そして後ろにはエラー処理を行うコードがそれぞれ挿入されています。実行してみるとAddressSanitizerなしでは、とくにエラーもなく実行されていたプログラムが、Address

注1) https://plus.google.com/+LinusTorvalds/posts/jmtzzLiiejc

Sanitizerありの場合は**図2**のようにどこで不正なアクセスが起きたかといった情報が出力され、プログラムの実行が中断されています。

Intel MPX

Intel MPXは、このAddressSanitizerのような機能をハードウェアで実現するものです。ハードウェアを使うことでソフトウェアの実装よりも低いオーバーヘッドでアドレスの検査が行われると期待されます。

それでは、Intel MPXを試してみましょう。MPXは、2015年出荷予定のSkylakeアーキテクチャから搭載される機能であり、現在MPXが動くCPUは流通していません。

そこでIntel Software Development Emulator(SDE)を使用して、MPXをエミュレーションして実行してみます。また、MPX用のコードをコンパイラに挿入してもうらためには、MPXに対応したgcc、binutilsが必要となります。さらにMPXのライブラリも必要です。Intel SDEのサイト[注2]から、これらのバイナリをダウンロードできます。

先ほどと同じコードをMPX対応のgccでコンパイルし、SDEで実行します。コンパイルオプションは**図3**のようになります。実行してみる

注2) https://software.intel.com/en-us/articles/intel-software-development-emulator

▼リスト1　不正なアドレスにアクセスするプログラム

```
void foo(int *x)
{
  *x = 0x1234;
}

int *bar()
{
  static int a;
  return &a;
}

int main()
{
  int a[16];
  int b[16];
  foo(a);
  foo(a+16);
  return 0;
}
```

▼図1　AddressSanitizerによるプログラムの変化

• AddressSanitizerを無効にした場合

```
0000000000000000 <foo>:
   0:   c7 07 34 12 00 00       movl   $0x1234,(%rdi)
   6:   c3                      retq
```

• AddressSanitizerを有効にした場合

```
0000000000000000 <foo>:
   0:   48 89 f8                mov    %rdi,%rax       # アドレスの検査
   3:   48 c1 e8 03             shr    $0x3,%rax
   7:   0f b6 80 00 80 ff 7f    movzbl 0x7fff8000(%rax),%eax
   e:   84 c0                   test   %al,%al
  10:   74 0d                   je     1f <foo+0x1f>
  12:   48 89 fa                mov    %rdi,%rdx
  15:   83 e2 07                and    $0x7,%edx
  18:   83 c2 03                add    $0x3,%edx
  1b:   38 c2                   cmp    %al,%dl
  1d:   7d 07                   jge    26 <foo+0x26>
  1f:   c7 07 34 12 00 00       movl   $0x1234,(%rdi)  # 元のコード
  25:   c3                      retq
  26:   50                      push   %rax            # エラー処理
  27:   e8 00 00 00 00          callq  2c <_GLOBAL__sub_I_00099_0_foo+0x1c>
                        28: R_X86_64_PC32       __asan_report_store4-0x4
```

と（AddressSanitizerよりはわかりにくいですが）たしかに範囲外へのアクセスを検出しているようです。

boundsレジスタと境界チェック

では、MPX対応のgccでコンパイルしたとき

▼図2　AddressSanitizerの出力

```
$ gcc -O0 -fsanitize=address asan.c
$ ./a.out
=================================================================
==10558==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffe4d550ef0 at pc ↵
0x400889 bp 0x7ffe4d550e60 sp 0x7ffe4d550e50
WRITE of size 4 at 0x7ffe4d550ef0 thread T0
    #0 0x400888 in foo (/home/naota/src/sd/a.out+0x400888)
    #1 0x400934 in main (/home/naota/src/sd/a.out+0x400934)
    #2 0x7f01e64b7f9f in __libc_start_main (/lib64/libc.so.6+0x1ff9f)
    #3 0x400778 (/home/naota/src/sd/a.out+0x400778)

Address 0x7ffe4d550ef0 is located in stack of thread T0 at offset 96 in frame
    #0 0x4008a4 in main (/home/naota/src/sd/a.out+0x4008a4)

  This frame has 1 object(s):
    [32, 96) 'b' <== Memory access at offset 96 overflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism ↵
or swapcontext
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow ??:0 foo
Shadow bytes around the buggy address:
  0x100049aa2180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100049aa2190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100049aa21a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100049aa21b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100049aa21c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x100049aa21d0: 00 00 f1 f1 f1 f1 00 00 00 00 00 00 00 00[f3]f3
  0x100049aa21e0: f3 f3 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100049aa21f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100049aa2200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100049aa2210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100049aa2220: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Heap right redzone:      fb
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack partial redzone:   f4
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Contiguous container OOB:fc
  ASan internal:           fe
==10558==ABORTING
```

のアセンブラを見てみましょう。**図4**の4005e0、4005e6の行が元のコードに対応しています。その上の2行がアドレスの検査を行っている部分です。“bndcl”、“bndcu”という命令および“%bnd0”というレジスタが登場しています。また、よく見ると元のコードに対応した部分にも“bndretq”とbndのprefixが付いています。これらはMPXの拡張になります。

まず“%bnd0”を見てみましょう(**図5**)。これはboundsレジスタという128bitのレジスタで、上位64bitにアクセス可能なアドレスの上限を、下位64bitにアクセス可能なアドレスの下限を設定して使います。

boundsレジスタはBND0からBND3の4つが存在しています。これらboundsレジスタを使ってアクセスするアドレスの検査を行うのが“bndcl”および“bndcu”命令です。“bndcl”は指定したアドレス(ここでは“int *x”のアドレス)がboundsレジスタに設定された下限以上であるかを、“bndcu”は同じく指定したアドレス(intが4byteなので int *xのアドレス+3)が、上限以下であるかを確認する命令です。すなわち、ここではどこかで設定されたboundレジスタBND0のアクセス可能領域内に“int *x”のアドレスが入っているかどうかを確認しています。アクセス可能範囲でなかった場合には例外が発生し、(対応していなければ)SEGVでプロセスが終了することになります。

では、BND0はどこで設定されているのでしょうか。呼び出し元のmain()を見てみましょう。**図4**の4005e8の行 から4005e1の行がboundsレジスタを設定している部分です。いろいろと命令が実行されていますが、“bndmk”がBND1に値を設定している部分です。 ここでは“%rdi”のアドレスから“%rdi+%rax”の範囲をアクセス可能と設定しています。%rdiには“int a[16]”の先頭アドレスが設定され、%raxには“0x3f”(=4(intのサイズ)×16−1)が設定されています。

▼図3　MPXを使ったコードの実行

```
# バイナリの展開
$ cd ~/mpxtest
$ tar xf 2014-02-13-mpx-runtime-external-lin.tar.bz2
$ tar xf gcc_install_5.0.0-mpx-r214719.tar.gz
$ tar xf binutils-gdb_install_2.24.51.20140422.tar.gz
$ tar xf sde-external-7.15.0-2015-01-11-lin.tar.bz2
# 環境変数を設定
$ export MPX_BINUTILS=$HOME/mpxtest/binutils-gdb_install_2.24.51.20140422/bin
$ export MPX_RUNLIB=$HOME/mpxtest/2014-02-13-mpx-runtime-external-lin
$ export GCC=$HOME/mpxtest/gcc_install_5.0.0-mpx-r214719/bin/gcc
$ export SDE=$HOME/mpxtest/sde-external-7.15.0-2015-01-11-lin/sde
# コンパイル
$ $GCC -fcheck-pointer-bounds -mmpx -L$MPX_RUNLIB -B$MPX_BINUTILS -lmpx-runtime64 -Wl,➥
-rpath,$MPX_RUNLIB -O2 mpx.c
# 実行
$ CHKP_RT_PRINT_SUMMARY=yes CHKP_RT_MODE=stop $SDE -mpx-mode -- ./a.out
Bound violation detected,status 0x1 at 0x4005b9

MPX run time summary:
number of BRs: 1.
size of allocated L1: 2147483648B
total size of allocated L2 entries: 0B

Used environment variables:
CHKP_RT_MODE = stop
CHKP_RT_PRINT_SUMMARY = yes
```

すなわち、"int a[16]"全体をアクセス可能とBND1に設定している、ということになります。そして4005e1の行ではBND1で構成したアクセス範囲をBND0へとコピーしています。

同様にbar()のようにポインタを返す関数の場合も、4005c8の行のようにBND0を使ってポインタの範囲を返していることを見ることができます。

bound table

このようにMPX対応のコードではboundsレジスタを使って引数や戻り値のポインタの範囲が渡されていきます。では、boundsレジスタの数である4つ以上のポインタ引数がある場合はどうするのでしょうか。その場合に使われるのが、bound tableです。bound tableは指定したアドレスに対応するアドレス範囲は保存するデータ構造です。

bound tableは、**図6**のようにページテーブルと似た構造を持っています。boundtableの最上部であるbound directoryのアドレスはBNDCFGUレジスタ(userland用)またはBNDCFGSレジスタ(kernel用)に設定されています。64bit

▼図5　BND0の動作

▼図4　MPX版のコード

```
00000000004005c8 <bar>:
  4005c8:       66 0f 1a 05 70 05 20    bndmov 0x200570(%rip),%bnd0  # 600b40 <__chkp_bounds_of_a.2361>
  4005cf:       00
  4005d0:       b8 00 0c 60 00          mov    $0x600c00,%eax
  4005d5:       f2 c3                   bnd retq

00000000004005d7 <foo>:
  4005d7:       f3 0f 1a 07             bndcl  (%rdi),%bnd0
  4005db:       f2 0f 1a 47 03          bndcu  0x3(%rdi),%bnd0
  4005e0:       c7 07 34 12 00 00       movl   $0x1234,(%rdi)              # 元のコード
  4005e6:       f2 c3                   bnd retq

00000000004005e8 <main>:
  4005e8:       48 83 ec 58             sub    $0x58,%rsp                  # bound レジスタの設定
  4005ec:       48 8d 7c 24 10          lea    0x10(%rsp),%rdi
  4005f1:       b8 3f 00 00 00          mov    $0x3f,%eax
  4005f6:       f3 0f 1b 0c 07          bndmk  (%rdi,%rax,1),%bnd1
  4005fb:       66 0f 1a c1             bndmov %bnd1,%bnd0
  4005ff:       66 0f 1b 0c 24          bndmov %bnd1,(%rsp)
  400604:       f2 e8 cd ff ff ff       bnd callq 4005d7 <foo>             # foo();
  40060a:       66 0f 1a 04 24          bndmov (%rsp),%bnd0
  40060f:       48 8d 7c 24 50          lea    0x50(%rsp),%rdi
  400614:       f2 e8 bd ff ff ff       bnd callq 4005d7 <foo>             # foo(a+16)
  40061a:       b8 00 00 00 00          mov    $0x0,%eax
  40061f:       48 83 c4 58             add    $0x58,%rsp
  400623:       f2 c3                   bnd retq
```

環境の場合、bound directoryの各エントリは64bit(＝8byte)であり、上位61bitがそのエントリに対応するbound tableのアドレスを指し、最下位bitがそのdirectory entryが有効かどうかを示しています。directory entryのindexは28bitですのでbound directoryの大きさは$2^{28} \times 8 = 2^{31}$byte = 2GBとなります。bound tableの各エントリはアクセス可能なアドレス範囲の上限、下限、その範囲に対応するポインタの値、そして予約領域がそれぞれ8byteで合計32byteの大きさになっています。このindexは17bitで管理されるので、1つのbound tableの大きさは$2^{17} \times 32\text{byte} = 2^{22}\text{byte} = 4\text{MB}$となります。

プログラムは"bndldx"と"bndstx"を使うことで、bound tableにアクセスします(**リスト2**)。たとえば、保存はbaz2の4005e0で行われています(**図7**)。BND0に設定したアドレス範囲(＝aのアクセス可能範囲)、0x600d40(＝p[0]のアドレス)、%rax(＝p[0]の値＝aのアドレス)が登録されます。読み込み時も、これらの情報を"bndldx"の引数として使い、boundレジスタへの読み込みを行っています(4005adと4005f3の行)。

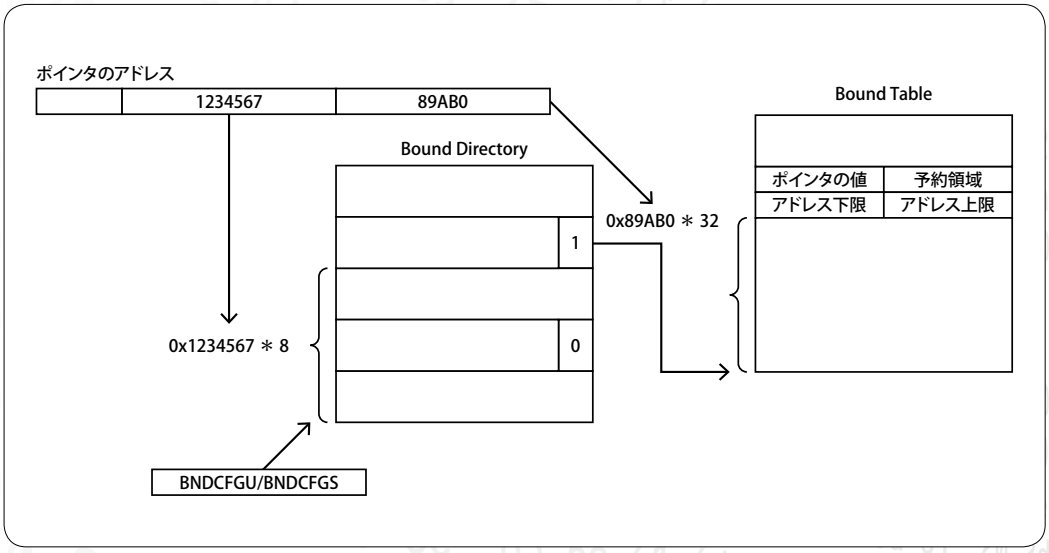
カーネルにおけるIntel MPXの対応

ここまでの話はどれもuserland内で完結する話でした。では、Linux 3.19のIntel MPX対応とはどのようなものでしょうか。カーネル側のMPX対応はおもに2点、例外発生時のシグナルハンドラへの情報通知と、カーネルによるbound tableの管理になります。

前述したように、アドレス範囲の検査に失敗すると例外が発生します。これまでのシグナルハンドラ通知では、どこで例外が発生したのかまではわかりますが、どのアドレス範囲から外れたのかを知ることができません。そこでこの通知情報を拡張し、アドレス範囲をuserlandからとれるようにしています。

もう1つの機能はbound tableの管理です。bound tableでは1MB分のアドレスに関する情報を管理するのに4MBのメモリを使用します。さらにbounddirectoryのサイズが2GBですので、たとえば1GB分の仮想アドレス空間を管理するbound tableを事前に作ろうとすると1GB × 4 + 2GB = 6GBのアドレス空間が必要となってしまいます。使うかどうかもわからないbound tableを持っているのは無駄ですのでオンデマンドに

▼図6　bound tableのしくみ



bound tableを作ることが望まれます。“bndstx”がbound tableに登録を行うとき、対応するbound directoryentryのvalid bitが立っていなければ、例外が発生します。カーネルはこの例外を拾い、新しいbound table用の領域を確保し、bound table entryの値を適切に設定します。

まとめ

今回はLinux 3.19で導入されたIntel MPX機能のカーネル対応について紹介しました。MPXのオーバヘッドがどのぐらいなのかはまだ分かりませんが、Skylakeが出てくるのが楽しみになりますね。SD

▼リスト2　bound tableが使われるコードの例（Cのソースコード）

```
int baz1(int *a, int *b, int *c, int *d, int *e)
{
  if (*a == *b && *c == *d && *d == *e)
    return 0;
  return 1;
}

int *p[16];
void baz2()
{
  static int a[16];
  p[0] = a;
}

void f()
{
  p[0][16] = 0x1234;
}
```

▼図7　bound tableが使われるコードの例（アセンブラ）

```
00000000004005a6 <f>:
  4005a6:       48 8b 05 93 07 20 00    mov    0x200793(%rip),%rax        # 600d40 <p>
  4005ad:       0f 1a 04 05 40 0d 60    bndldx 0x600d40(,%rax,1),%bnd0    # bound table から読み込み
  4005b4:       00
  4005b5:       f3 0f 1a 40 40          bndcl  0x40(%rax),%bnd0
  4005ba:       f2 0f 1a 40 43          bndcu  0x43(%rax),%bnd0
  4005bf:       c7 40 40 34 12 00 00    movl   $0x1234,0x40(%rax)
  4005c6:       f2 c3                   bnd retq

00000000004005c8 <baz2>:
  4005c8:       48 c7 05 6d 07 20 00    movq   $0x600cc0,0x20076d(%rip)    # 600d40 <p>
  4005cf:       c0 0c 60 00
  4005d3:       b8 c0 0c 60 00          mov    $0x600cc0,%eax
  4005d8:       66 0f 1a 05 50 06 20    bndmov 0x200650(%rip),%bnd0        # 600c30 <__chkp_bounds_f
_of_a.2372>
  4005df:       00
  4005e0:       0f 1b 04 05 40 0d 60    bndstx %bnd0,0x600d40(,%rax,1)     # bound table への保存
  4005e7:       00
  4005e8:       f2 c3                   bnd retq

00000000004005ea <baz1>:
  4005ea:       4d 89 c1                mov    %r8,%r9
  4005ed:       66 0f 1b 5c 24 e8       bndmov %bnd3,-0x18(%rsp)       # BND3退避
  4005f3:       42 0f 1a 1c 04          bndldx (%rsp,%r8,1),%bnd3      # (%r8)用のアクセス範囲をロード
⋮
  400621:       66 0f 1a 54 24 e8       bndmov -0x18(%rsp),%bnd2       # 退避した元のBND3をBND2にロード
  400627:       f3 0f 1a 11             bndcl  (%rcx),%bnd2
  40062b:       f2 0f 1a 51 03          bndcu  0x3(%rcx),%bnd2
  400630:       3b 11                   cmp    (%rcx),%edx
  400632:       f2 75 14                bnd jne 400649 <baz1+0x5f>
  400635:       f3 41 0f 1a 18          bndcl  (%r8),%bnd3
  40063a:       f2 41 0f 1a 58 03       bndcu  0x3(%r8),%bnd3
  400640:       41 3b 10                cmp    (%r8),%edx
⋮
```

May 2015

NO.43

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

各地の名産も密かな楽しみ!? プログラミング勉強会の旅

2014年12月に鳥取で行ったワークショップと、2015年2月の浜松での研究会の模様をお伝えします。

シェルスクリプトワークショップ

■シェルスクリプトワークショップ
【日時】2014年12月13日（土）～12月14日（日）
【場所】鳥取環境大学

jusは毎年1回ぐらいのペースでワークショップを開催しています。今回はUSP友の会との共催で、シェルスクリプトを勉強するワークショップを行いました。参加者は講師を含めて15人でした（**写真1**）。

■初心者講習会

初日の午前中は、オプションプログラムとしてシェルスクリプトの初心者講習会を行いました。講師はjus幹事の齊藤明紀さん（鳥取環境大学）です。前半は、起動形態、シェル変数、ワイルドカード、クォート、標準入出力、パイプラインなどの概念と主要な構文を紹介し、後半はいくつかの例題を挙げ、それを処理するシェルスクリプトを、頻出コマンドを紹介しながら解説しました。

■本編初日

初日の午後からがワークショップの本編です。前半は今泉光之さん（USP友の会）による「仕事で使えるシェルスクリプト」という講座でした。POSIXの話に始まり、シェル変数のオプションを使ったデバッグ、変数展開のあれこれ、さらにコマンドや正規表

写真1　シェルスクリプトワークショップの様子

現などを活用したプログラミングテクニックを、サンプルプログラムとともに解説しました。

後半は、同日に東京で行われていたシェル芸勉強会（USP友の会主催）のUstream配信をスクリーンに映し、斉藤博文さん（日本GNU AWKユーザー会）に解説していただきながら、勉強会で出題された問題の解答プログラムを考えました。解法は必ずしも1つとは限らないので、参加者からの回答の良し悪しを吟味するディスカッションも行いました。

夜は鹿野温泉山紫苑に移動しての合宿です。参加者一同でカニ鍋を囲みながら、シェルやUNIXの話で親睦を深めました。雪の中で入る露天風呂も気持ち良かったです。

■本編2日目

翌日は再び鳥取環境大学に舞台を移し、午前中は斉藤博文さんによる上級シェル芸講座を行いました。初日と同様に講師から問題が出され、各自でプログラムを考えたあとに、講師から解答例と関連テクニックの解説がなされるという構成で進行しました。問題もバラエティに富んでいて、ファイルの移動や

編集といったよく使いそうなものから、住所を縦書きにしてTrueTypeフォントに変換するというマニアックなものまでありました。

午後は再び今泉さんが講師を務め、「シェルスクリプトを極める」というセッションを行いました。ファイルディスクリプタを活用したプログラミング、evalを利用して変数を配列のように扱うテクニック、排他処理、exを用いたファイルの行操作などの技法が紹介されました。exの実体はviですので、viの編集操作コマンドがそのまま使えます。最後に、初日の宿題として出す予定だった問題や、これまでの課題を発展させた問題を参加者に課し、各自で解答プログラムを考えるうちに終了時間となりました。

久しぶりにシェルのプログラミングに触れ、あらためてその奥深さを知ることができました。また、参加者にも詳しい方が多く、随時コメントをくださったおかげで、参考になる知見も多く得られました。

jus研究会　浜松大会

■IPv6対応アプリの作り方
【講師】渡辺 露文（IPv6普及・高度化推進協議会）
【司会】法林 浩之（日本UNIXユーザ会）
【日時】2015年2月11日（水／祝）16:45～17:00
【場所】浜松市市民協働センター

2年ぶりに訪れた浜松での研究会は、IPv6普及・高度化推進協議会の渡辺さんを講師に迎え、IPv6対応アプリの作り方について話をしていただきました。参加者は35人でした。

■ここがヘンだよ！そのコード

はじめに、「ここがヘンだよ！そのコード」と題して、IPアドレスがソースコードにハードコーディングされているプログラムを例示し、「こういうイケてないコードを書かないようにしてほしい」という話がありました。書籍などに掲載されているサンプルコードにもこのような事例が散見されるそうです。

対応策としては、ソースコードにおいてはホスト名をFQDN（Fully Qualified Domain Name）で指定し、DNS（Domain Name System）を利用してIPアドレスを得ます。こうすればIPv4でもIPv6でも同一のプログラムが使えます。

■いまどきのIPv6を知ろう！

次に「いまどきのIPv6を知ろう！」と題して、IPv6の現状を解説されました。最近のOSはどれもデフォルトでIPv6が利用可能です。インターネット回線もIPv6対応が進んでいて、とくにフレッツ光ネクストが普及するとIPv6普及率も急激に増加する見込みです。IPv4とIPv6はアドレス体系も異なり、互換性がありません。IPv4アドレスはすでに枯渇しており、今後はCGN（Carrier Grade NAT）による接続が増えるため、セッション数や速度において品質の低下が予想されます。

さらに、2015年1月に発生したglibcの脆弱性についてもIPv6対応の観点からコメントがありました。問題となったgethostbyname関数はIPv6に登場に伴ってあまり利用されなくなっており、IPv6をサポートしているgetaddrinfo関数を使っていれば問題は起きないそうです。

■さあ、アプリケーションをIPv6に対応させよう！

最後に「さあ、アプリケーションをIPv6に対応させよう！」と題して、IPv6への対応方針などが紹介されました。IPv4/IPv6両方で動作するプログラムを1つのソースコードで実現するのが基本方針です。対応における要点は、IPv4/IPv6両対応のプログラミング言語と実行環境を使うこと、通信処理を両方に対応させること、データとしてIPアドレスを扱う部分を両対応にすることです。「けっして難しいことではなく、ちょっとした注意を払えば実現できるので、今後開発するアプリケーションはぜひIPv6に対応してほしい」と講演を締めくくりました。

今回の発表資料はWeb[注1]でも公開されています。参考にしてください。SD

注1）URL http://www.slideshare.net/v6app/osc2015-hamanako

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
小泉 勝志郎　Katsushiro Koizumi
Twitter @koi_zoom1

第41回 地元を盛り上げる萌えキャラ「渚の妖精ぎばさちゃん」

塩竈の離島、浦戸諸島も津波による大きな被害を受けた場所です。この離島を舞台に行われたハッカソンで、地域振興のために生まれたのはなんと萌えキャラ。しかもモチーフは海藻。この大胆なギャップ萌えキャラクターに、ラブアローシュート！

連載始まって以来の異色回？

いままで震災復興へのITでの取り組みを紹介してきた本連載ですが、今回は今までで一番の異色回！　萌えキャラ「渚の妖精ぎばさちゃん」についての紹介をします（**イラスト1**）。

♥渚の妖精ぎばさちゃん公式サイト
http://gibasachan.com

ぎばさちゃんはアカモクという海藻をモチーフにした萌えキャラです。「なぜ萌えキャラ？」と思う方も多いでしょう。実はこの連載でも二度ほど登場してるんです。

普段はTwitterを活躍の中心としているぎばさちゃん（TwitterID：gibasachan）。今日はTwitterを飛び出てSoftware Design誌上に遊びに来てもらいましょう！

では、ぎばさちゃん自己紹介お願いします！

★★★★★★

◆**イラスト1　ぎばさちゃん（等身大とSD）**

ぎばさちゃんってどんな娘？

はいっ！　わたし、赤杢（あかもく）ぎばさです！

髪の毛はアカモク、カバンにはヒトデ、萌える瞳は天使の炎！　人呼んで渚の妖精ぎばさちゃん！　海藻アカモクの販売促進キャラクターです！　毎日Twitterでアカモクのことや塩竈のことをつぶやいています！　最近では「飯テロ」も人気なんですよ。

身長は152cm、体重はヒミツです！　ヒントをあげるとアカモク43kg分です。体重わかります？

髪の毛はふだんは赤いんですが、感情のたかぶりによる体温上昇で髪が緑色になります！　アカモクに火を通した時と同じ！　これがッ！　銀波藻湯通現象（ギバサボイルドフェノメノン）ですッ！

そして、必殺技はサルガッソーホールド！　髪の毛のアカモクを利用して首肩肘膝のすべてを極める技！　これでCQC（近接格闘）が多い日も安心です。当然必殺なので必ずころ（ry

ぎばさちゃんが生まれるまで

……物騒なことを言い始めたので再度バトンをこちらに（笑）。

★★★★★★

ぎばさちゃんは以前のこの連載でも取り上げた「島ソン」で生まれました。「島ソン」は東日本大震災で大きな被害を受けた宮城県塩竈市の離島浦戸諸島を舞台としたハッカソン。島の課題解決がテーマでした。

この中でテーマとしてあがったものの中にアカモクがあり、Twitter BotやFacebookページと並んで進行したのがこのアカモクの萌えキャラプロジェクト「渚の妖精ぎばさちゃん」なのです！

第41回 地元を盛り上げる萌えキャラ「渚の妖精ぎばさちゃん」

しかし、そもそもなぜアカモクなのでしょうか？

◆写真1　アカモク

◆イラスト2　ぎばさちゃん涙姿

●アカモクとは？

アカモクは**写真1**にあるように枝分かれした房状の海藻です。これを湯通しして叩いて刻むとメカブのように強力な粘りが出ます。これをご飯にかけたり、味噌汁に入れたりして食します。磯の香りとヌルヌルネバネバとシャキシャキが共存する食感がおいしい逸品です。

写真1の上のように茹でる前は赤茶色……モノクロですね……んん、まあ赤茶色なのですが、茹でたあとは下のように鮮やかな(涙)緑色になります。ぎばさちゃんの髪の毛の設定はここから来ています。

今まではアカモクというと、ぎばさちゃんのサルガッソーホールドのごとく船のスクリューに絡みつくためかなり嫌がられる存在でもありました。しかし、近年栄養価の高さから注目が集まってきたことで塩竈でも「シーフーズあかま」さんを中心にアカモクを特産品として売り出そうという動きが出てきています。

ところで、この記事を読んでいる皆さんでアカモクをご存じの方はどのくらいいらっしゃいますか？塩竈でこれからブランド化していきたいと思われているアカモク。残念ながら知名度はいまだ高くありません。そこでぎばさちゃんを入り口にアカモクをいろいろな人に知ってもらおうと、ぎばさちゃんは活躍しているのです。

震災復興も時が流れることでフェーズが変わり、災害への対応の色合いよりも地域振興の色合いが強くなってきています。地域を振興するにはまず人に来てもらうこと。そして、来てもらいたくなるためにも、美味しいものを紹介すること。ぜひぎばさちゃんのTwitterを見てください。いつも何かおいしそうなものを食べているはずです。塩竈に、東北に、遊びに来てもらうため、日夜「飯テロ」に励んでいるのです！　おいしいものを食べるための屁理屈という説は却下します(笑)。

●コンセプトからイラストへ

キャラクターを作るにはまずコンセプトから。ぎばさちゃんの場合はこんな経緯です。

アカモクは秋田県では「ぎばさ」という名前で以前から食べられていました！

- 「つばさ」という女の子がいるなら「ぎばさちゃん」もありでしょ！
- 高杢（たかもく）という苗字があるなら、赤杢（あかもく）という苗字も良いよね！

ということで、名前は「赤杢ぎばさ」に。

- アカモクはネバネバした海藻なので、「粘る」と「ぎばさ」を合わせてさらに「ネバーギブアップ」とかけた決め台詞として「ねばぎば」！
- ネバネバを特徴にも出そうということで、「涙がネバネバ」というコンセプトも追加
- アカモクは火を通すと赤っぽい色から緑へ変わるので、髪の毛の色がテンションで変わるように！

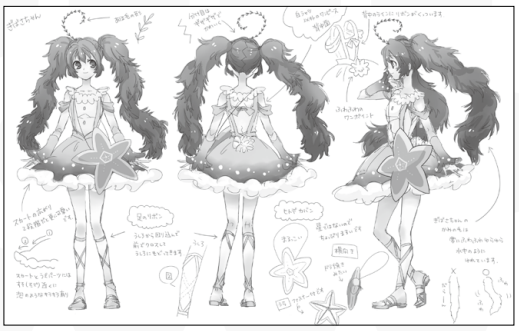
キャラクターの特徴もこうして決まりました！

しかし、萌えキャラなのにメンバーの誰も絵が描けません……。ここでは、クラウドワークスを使うことで、なんとコンペ形式で募集ができました。

涙がネバネバとか、こんなコンセプトで「本当に可愛くなるのかよ？(苦笑)」とコンセプトを作った人間も思うものでしたが、コンペには30件以上のデザインが寄せられ、上がってきたイラストはとても可愛らしく、絵描きさんの凄さを思い知るもので

◆写真2　ぎばさちゃん3D

◆イラスト3　ぎばさちゃん三面図



した！　キャラクターデザインには白飯ザリガニさんのものを採用。現在もTwitterを中心にぎばさちゃんは活躍中です。「ネバネバな涙」もご覧のとおり（**イラスト2**）可愛い感じに仕上がりました！
ちょっと昭和のアニメ風味ですね。

なぜ萌えキャラ？

なぜ“萌えキャラ”であって“ゆるキャラ”ではないのか？　最大の違いは「ファンによる拡散効果」です。Twitterで好きなキャラの絵を描いている方ってよく見かけますよね？　ああいった感じでの拡散が行われやすいのです。

また、基本的に人型なので、ゆるキャラと違い「ストーリーを作りやすい」のも大きなポイント。マンガにしてもらいやすいということです。

ゆるキャラは萌えキャラよりも対象は広く取れますが、「ふなっしー」や「くまモン」のように大ヒットするキャラは一部。萌えキャラはファンのつきやすさとファンによる拡散で、ゆるキャラよりも宣伝効果が高くできると考えています。実際にTwitter上ではファンの皆さんからぎばさんちゃんのイラストが寄せられています。「#ぎばさちゃん」でTwitterを検索してみてください！

●ぎばさちゃんのライセンス

ぎばさちゃんは積極的に二次創作（ファンによるイラストなどの創作）を推奨しています。アカモクの製品のパッケージと震災復興イベントには個人・法人を問わず無償で使用可能です。また、個人によるスマートフォンアプリの開発を促進したいため、かなりゆるいライセンス形態としています。詳細はサイトのガイドラインをご覧ください。

♥渚の妖精ぎばさちゃんガイドライン
http://gibasachan.com/guideline

●ぎばさちゃんには3Dモデルも

スマートフォンアプリ、とくにゲームアプリを作りやすくするために、ぎばさちゃんには3Dモデルデータもあります（**写真2**）。

♥ダウンロードリンク
https://github.com/koizoom1/gibasachan3D/

この3Dモデルデータはヒューマンアカデミー仙台校の授業の中で作成されました。制作を担当したのは仙台校ゲーム科グラフィック専攻の学生・萩原佳央瑠さん（2年）、小林悠さん（2年）、新沼伊緒奈さん（2年）ら3名。

この3Dモデルを作るために三面図も用意しました（**イラスト3**）。背面の情報もあるのでフィギュアなどの立体物を作るのに役立ちますし、イラストを描く際にも役に立ちます。

ところでこの3Dモデル、Blenderで読み込むとバグって、首から足が直接生えている仄暗い水の底から這い出たようなおぞましいクリーチャーになってしまいます。ちゃんとした形にするための手順が現在有志によって公開中です。しかし、このおぞましさが逆にクリエイターの創作魂を呼び起こし、H.P.ラヴクラフトのクトゥルフ神話になぞらえ「神話生物ぎばさ」（**イラスト4**）と呼ばれ、ニコニコ動画にはこの神話生

物が踊る動画まで上がっています！

♥【MMD】神話アイドルぎばさちゃん☆涙のルルイエライブ【ねばぎば！】
http://www.nicovideo.jp/watch/sm25341827

こういった3Dモデルのための設定が応用され、ファンの手による羊毛フェルト人形まで生まれています！（**写真3**）

ぎばさちゃんアプリも！

この本の読者の皆さんの多くがそうであるように、筆者もIT系の技術者です。その技術があるなら使わねば！ということで、ぎばさちゃんのiOSアプリを作成いたしました。その名も「渚の妖精ぎばさちゃん 塩竈クエスト」。ぎばさちゃんを主人公にしたスタンプラリーアプリです（**図1**）。アプリを起動すると、ぎばさちゃんが涙ながらに塩竈の行きたいスポットを教えてくれます（**図2**）。

●まちあるきで使いたい！

ぎばさちゃんが生まれることになった「島ソン」を開催したコミュニティ「Code for Shiogama」は現在隔月ペースで塩竈の“まちあるき”を行っています。この“まちあるき”で行ったスポットをチェックポイントとしてアプリに登録してあります。

そして、これからはこのアプリを使ってまちあるきを行うことで、ナビを簡易にし、さらに新しく見つけたスポットを足すというサイクルを回していきたいところです。

●オープンソースで公開します！

このアプリはオールSwiftで作っていて、リリース後、オープンソースで公開します。画像ファイルとチェックポイントのCSVを差し替えるだけで「各地域版」を作ることができる形にします。チェックポイントデータはクラウド上に置いたほうが更新も楽なのですが、今回は初学者が簡単に「各地域版」を作れるようにという配慮から、あえてCSVとしています。

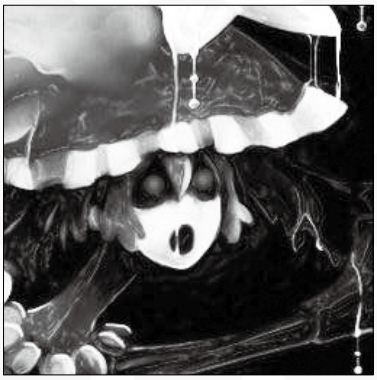
各地に存在する萌えキャラ。このアプリがそれぞれの地域の萌えキャラ版が出て各地域のまちあるきに使われ振興に役立つ。そんな未来になればと思っています。

♥以下のリンクで公開します（リリース後）
https://github.com/koizoom1/gibasachanStamp/

最後に

ぜひ、ぎばさちゃんのTwitterをフォローしてください。アカモクの情報はもちろんのこと、塩竈の情報、そして東北のおいしいところ情報も入ってきて、東北に、塩竈に、遊びに来たくなること請け合いです！　震災復興へねばぎば！ SD

◆イラスト4　神話生物ぎばさ



◆図1　ぎばさちゃん 塩竈クエスト

◆写真3　ぎばさちゃん羊毛フェルト人形

◆図2　地図にはチェックポイントがアカモクで表示

温故知新
IT むかしばなし

第43回

VRAMとbit演算

Software Design編集部

はじめに

筆者が大学生のとき、後輩に「なんでアセンブラでは、足したり引いたりする計算と、それを代入する命令がほとんどなのにいろんなことができてしまうのですか？」と聞かれたことがあります。たしかにアセンブラの命令を見てみると、単体で複雑なことができるものはほとんどありません。そんな中でも特殊なのがbit演算ですが、昔は非常に使用頻度が高い命令だったのです。今回はVRAM（Video RAM）とbit演算についてお話します。

パソコンでゲーム

1980年ごろに発売されていたパソコンには、8bit CPUである、Z80[注1]やMOS 6502[注2]、MC6809[注3]などが搭載され、画面の解像度は640×200（もしくは400）ドットと今から比べるととても粗いものでした。

今のようにインターネットに接続することもなかった時代、パソコンの楽しみといえばゲームをすることで、テンキーの[2][4][6][8]で画面に表示された自キャラクタを移動させ、スペースキーでミサイル発射したりするものが多く出回っていました。BASICで書かれたゲームも多くありましたが、速度重視のゲームではもっぱら機械語が多く使われており、雑誌に掲載された16進数のダンプリストを何人かで手分けして入力して、それをテープレコーダで記録するといったことが日常茶飯事だったのです。

VRAM

この8bit CPUが管理できるメモリ領域は64KBしかなく[注4]、すべてのプログラムやデータをこの中で処理していたのです（**図1**）。このメモリ領域にはVRAMと呼ばれる部分があり、そこにデータを書き込むと画面に文字やグラフィックが表示されるというしくみになっていました。

当時、一番多くのパソコンで使われていたZ80は、CPUのクロックサイクルが2.5MHzでした。現在のCPUは2.5GHz以上のものがほとんどですから、当時から比べると1,000倍以上の速度差（命令サイクルや高速化技術があるので実際はもっと速い）があります。

試しにグラフィック画面を消去するだけのプログラムを見てみましょう（**リスト1**）。

HLレジスタの示す場所にAレジスタを代入して0にするLD命令が7サイクル、HLレジスタのインクリメントに6サイクル、DJNZというカウンタ付きループ命令が13サイクルかかるので、1byteを0にするのに26サイクル[注5]かかります。1画面が640×200ドットとすると、640bit=80byteが200行分なので80byte×200で16KB。これがRGBの3画面分ですので16 KB×3画面で48KB。消去にかかる時間は48KB×26≒1.25Mサイクルで、2.5MHzのCPUでは0.5秒となります[注6]。これでは消して

注1） NECのPC-8001やPC-8801、シャープのMZ-80シリーズなどに搭載。
注2） Apple IIなどに搭載。
注3） FM-8/7などに搭載。
注4） プログラムの実行位置を示すPC（プログラムカウンタ）レジスタが16bitなので、その範囲内でした。
注5） 本当に最速にするならブロック転送命令のLDIRを使い、21サイクルで可能ですが、説明の都合上、ループを作っています。
注6） 昔のアセンブラプログラマは、どの命令が何サイクルかかるか計算しながら少しでも速く動くようにプログラムを組んでいたのです。

いるのが丸見えになってしまうので、書き終わるまで表示を消しておくことも多かったです[注7]。

このような速度でしたから、グラフィックのスクロールも遅く、多くの機種では文字の画面表示にはキャラクタVRAMと呼ばれる、書き込むと対応するASCIIコードの文字が画面に表示されるしくみと重ね合わせて使われていました。

ゲームキャラクタなどの中間色

このころのグラフィックは中間色がなく、1ドットにつきRGBの組み合わせの7色しか出せませんでした。そのため、ゲームキャラクタ(以降キャラクタ)などをデザインするときには、「緑黄緑黄緑黄緑黄」とドットを並べて黄緑色を作ったり、「桃黄桃黄桃黄桃黄」[注8]として肌色に近い色を作ったりしていました。横方向に1byte(=8ドット)のデータを2個並べ、下方向に16個並べて(これで16×16ドットのキャラクタができる)、先の中間色の並びを1ドットずらしたりして、なるべくきれいなものになるような工夫が必要でした。

キャラクタの移動も1byte(8ドット)単位で、動きも粗いものでした。キャラクタの重なりなどはbit演算のOR(論理和)を使ってしまうと、白っぽくなって見分けがつかなくなるので、XOR(排他的論理和)がよく使われていました。

リスト1ではAレジスタを0にするためにXOR Aを実行していますが、Aレジスタに何が入っていてもXOR演算をすると0になり、LD A,0は7サイクル、XOR Aは4サイクルと圧倒的に速かったため、その後のアセンブラでも同様のテクニックがよく使われています。

背景とキャラクタ

背景が真っ黒ならば、そこにキャラクタのデータを書き込めばよいのですが、バックに模様などがあるとそうはいきません。キャラクタを描画させるためにはキャラクタの存在する範囲が1になっているbit列を作るためにRGBの3プレーン分のデータをORしたものを作っておき(いわゆる抜型ですね)、それでキャラクタを描く前に、その範囲の背景データを保存しておき、背景データと型を反転したデータとAND(論理積)をとって背景のキャラクタを載せる部分を真っ黒にしておいてからキャラクタデータをORして合成することによって、やっと1つのキャラクタを表示できました。

その後、パレットという技術で各bitに中間色を割り当てられたり、スプライトと呼ばれるキャラクタの描画作業をハード的に重ね合わせてくれるものができたため、こういったbit演算はあまり使われなくなってしまいました。

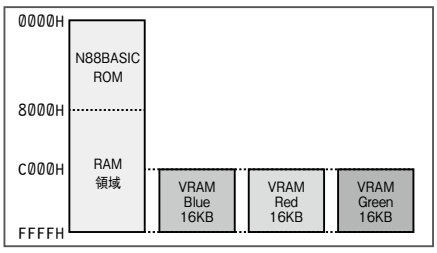
省メモリのためのbit演算

TrueかFalseかの値を記録するためだけなら、1bitで済みます。つまり1byteあれば8つの論理値を扱うことができます。フラグなどの状態をセット/リセットするために、ANDとOR、特定のbitを反転させるためにXOR、そのbitの状態を取り出すためにBITというbitテスト命令があったりと、今では考えられないほどメモリを大切にする時代でもありました。SD

注7) またDMA(Direct Memory Access)というメモリ読み出しとビデオ表示がかち合う方式だったため、画面の書き換えの隙間(垂直ブランキング期間)に収まるようタイミングを合わせることもあり、さらに遅くなっていました。

注8) 便宜的にマゼンタを桃と表記しています。

▼図1 PC-8801のメモリマップ(簡略)



▼リスト1 80byte分0を書き込むプログラム

```
      LD    B,80        Bレジスタ(カウンタ)に80をセット
      LD    HL,C000H    VRAMの開始をC000H(Hは16進数の表記)と仮定
      XOR   A           Aレジスタを0にする
L0:   LD    (HL),A      HLレジスタで示された場所を0にする
      INC   HL          HLレジスタを+1する
      DJNZ  L0          Bレジスタを-1し、0でなかったらL0にジャンプ
```

※Z80にはCPUが直接扱える8bitのA、B、H、Lなどのレジスタがあった。HとLレジスタはつなげて16bitとして使え、メモリ上のアドレスを指すために使われた。

開発のボトルネックはどこだ?

迷えるマネージャのための プロジェクト管理ツール再入門

第6回

SUUMOスマホサイトの開発裏話① 開発リードタイム短縮に向けアジャイルに取り組む

Software Design編集部

新築・中古のマンション、戸建ての情報や賃貸情報など、不動産情報や住宅情報を簡単に検索できる、リクルート住まいカンパニーが運営するサービスが「SUUMO」です。そのスマートフォン向けサイトでは、アジャイル開発の手法を採り入れています。その背景について、開発チームに所属する吉田拓真氏と山下芳生氏にお話を伺っていきます。

開発チームを悩ませた「不確実性」に対しての対応

――SUUMOのスマホサイトの開発チームでは、アジャイルを採り入れていると伺いました。そもそも、どういった背景からアジャイルに取り組むことになったのでしょうか。

吉田氏　不動産情報サイト市場は競争が激しく、常にスピード感を持ってWebサイトを改善し、使い勝手を高めることが求められています。しかし当時のSUUMOは、新機能の追加などを起案してからリリースするまでの期間、いわゆる開発リードタイムが非常に長かったのです。

▼図　SUUMOスマホ用サイト（http://smp.suumo.jp/）

そこで何が問題なのか話し合ったところ、「不確実性に対しての検証コストが大きい」「ステークホルダーが多い」「合意形成までの時間がかかる」といった問題が浮かび上がってきました。これらによって開発スピードが低下している。このような課題を可視化したのが始まりでした。

本来は、開発スピードを引き上げ、品質も高めれば、Webサイトに訪れるカスタマーも、物件情報を提供していただいているクライアントもハッピーになり、それが事業の成長につながるはずです。それに向けて何をすべきかを考えたとき、開発サイクルを短縮したり、もう少し小さいチームで開発を進めたりしたほうがいいのではないかと気づいたんです。そのときに初めて、アジャイル開発やスクラムといったキーワードが出てきました。

山下氏　とくに不確実性については、スマートフォンサイトは環境の変化も激しく、またPCサイトで培ってきたナレッジも展開しづらいため、何がカスタマーに良いのかを検証するために高速にトライ＆エラーを繰り返していく必要がありましたが、何が正解かわからないものに対しての検証や合意形成に時間を要していては競合サービスから取り残されていってしまうのではないかという危機感もありました。

吉田氏　高速にトライ＆エラーを繰り返すためにはスモールサイクルやスモールチームを実現する必要がありますが、実現するためのポイントになると考えたのは、「文化」「プロセス」「体制」、そして「基盤」です。文化は、常に挑戦する環境や風土の醸成、そしてカスタマーファーストの徹底です。プロセスは変化に柔軟に対応

できる開発手法の確立、そして高速なPDCAを回せる開発プロセスの構築を指しています。体制はスモールチームにつながるところで、目標に一丸となって向かっていける体制を目指しました。ただし、開発サイクルを短縮する中でも、今までどおりの品質は担保しなければなりません。それを支えるのが基盤の部分で、自動化のしくみを採り入れるなどで開発プロセスを効率的に回す。この4つを軸にアジャイル開発を進めようとしたんです。

情報共有基盤がなくチームごとにツールが乱立

――実際にスクラム開発を採り入れるのはたいへんだったと思います。その壁をどのように乗り越えたのでしょうか。

山下氏　まずは、スマートフォンサイトの一部の領域でスクラム開発を試すことから始めました。実はそのとき、どの程度の規模でどういったシステムなら適用できるのか、はっきりわからなかったのです。実際に試してみるとやはり効果があったので、次はもう少し大きな規模で検証するというように、見えていない部分を少しずつつぶしながら広げていきました。

――アジャイルに取り組む前、エンジニア間での情報共有はどのように行われていたのでしょうか。

山下氏　基本はメールですが、全体で統一されたツールはなく、乱立していた状態でした。一部でMantisが使われていたり、別のところはBacklogを使っていたりという状況です。それと、当時は縦割りの組織になっていて、開発チームと制作チームでそれぞれ異なるBacklogを使っていたこともありました。

吉田氏　驚いたのは、あるところでSubversionが使われていて、その内容を共有するために手動でBacklogにアップロードするといったことが行われていて、ツールを使うことで逆に効率が悪くなってしまう部分もありました。

山下氏　コミュニケーションのコストもすごくて、打ち合わせが週次で行われていて、1つのことを聞くのに1週間待たなきゃいけない。新しく開発に人が来たりメンバーが入れ替わったりしたときも、情報がまとまっていないので1から全部説明しなければならない。

吉田氏　そこでコミュニケーション基盤や開発基盤を標準化し、開発生産性を上げようということで「JIRA」や「Confluence」「Stash」「HipChat」といったアトラシアンのツールを導入することにしたんです。アトラシアンを選んだ理由として大きかったのは、強力なツール間連携によって、「企画」「開発」「運用」といったすべてのプロセスに適用可能だったことです。

▼写真　山下芳生氏（左）と吉田拓真氏（右）

開発リードタイムを短縮することを目的にアジャイルに取り組んだSUUMOの開発チームにおいて、コミュニケーション基盤、開発基盤として採用されたのがJIRAをはじめとするアトラシアン製品でした。次回は、これらのツールをどのように活用しているのかを伺っていきます。SD

リックソフトのWebサイトでは、各アトラシアン製品の体験版を提供しているほか、アトラシアン製品専用のコミュニティも運営しています。JIRAやConfluenceなどのアトラシアン製品に興味を持ったら、まずはアクセスしてみましょう。
https://www.ricksoft.jp/

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

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

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

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

Service

イントラリンクス合同会社、コラボレーションツール「Intralinks VIA」を提供開始

イントラリンクス合同会社は3月14日、社内外の関係者とのプロジェクトコラボレーションに利用できるクラウドベースのツール「Intralinks VIA」を提供開始した。

Intralinks VIAでは、ユーザが立ち上げたワークスペースに、ドキュメントをアップロードしたり、ほかのユーザを招待したりできる。招待されたユーザはアクセスが許可されたフォルダ、ワークスペースのみにアクセスできる。また、スマートフォンやタブレットからでもすばやくワークスペースにアクセスが可能。

本ツールの大きな特徴に「UNshare」機能がある。従来のコラボレーションツールでは“共有”することが重視されてきたが、今日の企業間のプロジェクトでは共有すると同時に情報を“コントロール”することが求められている。パートナー契約の終了、プロジェクト完了、人事異動などのイベントが起きれば、それまで認めていたアクセスを禁じ、ドキュメントがダウンロードされていればそれを取り戻す必要がある。「UNshare」により、ワークスペースで共有された機密文書から特定のユーザのアクセス権限を剥奪することで、すでにダウンロードされたファイルのアクセスを遮断できる。

CONTACT イントラリンクス合同会社
URL https://www.intralinks.com

Software

グレープシティ、「SPREAD for ASP.NET 8.0J」、「InputMan for ASP.NET 8.0J」を発売

グレープシティ㈱は、ASP.NETアプリ開発用、表計算グリッドコンポーネント「SPREAD for ASP.NET 8.0J」および入力支援コントロールセット「InputMan for ASP.NET 8.0J」を3月4日から販売している。

SPREAD for ASP.NETはWebアプリにおけるデータの一覧画面や集計機能を実装するExcelライクな表計算データグリッドコンポーネント。一方、InputMan for ASP.NETは細やかな書式や入力制御を行い、エンドユーザの快適で正確な入力を支援するコントロールセットとなっている。

今回の新バージョンでは、タブレット、クラウドといった運用環境への対応に注力したとのこと。対応ブラウザはInternet Explorerに加え、ChromeとSafari for iOSを新たにサポートしたうえ、タッチデバイスでの快適な操作性を強化している。運用サーバについては、Microsoft AzureやAmazon EC2といったクラウド仮想マシンを新たにサポートする。さらに、2製品の機能面での連携も強化されている。

1開発ライセンス価格はSPREADが172,800円、InputManが129,600円（ともに税込）。

CONTACT グレープシティ㈱
URL http://www.grapecity.com

Service

リンク、「ベアメタル型アプリプラットフォーム」でioMemory搭載モデルを提供開始

㈱リンクは、同社のサービス「ベアメタル型アプリプラットフォーム」において、3月25日より高速フラッシュストレージ「ioMemory」搭載モデルの提供を開始した。

同サービスは、ほかのユーザとリソースを共有しない物理サーバ型のクラウドサービス。そのため、パブリッククラウドと比べCPU性能やネットワーク性能が安定しており、パフォーマンスがより重要視されるような場合でも安心して運用できる。一方、近年のスマートフォンアプリ・ゲームなどは、CMやSNSとの連携によってアクセス数が短期間で膨大となり、物理サーバで運用している事業者であっても、恒常的にストレージI/Oがボトルネックになるケースが多い。

同サービスではそのような課題を解決するため、既存のioDrive2を搭載したモデルに加え、今回、サンディスク社が提供するFusion ioMemory SX300を搭載したモデルを提供開始することとなった。ioMemoryは、ioDrive2に比べて高いトランザクション性能を持ち、読み書き混合の処理に最適化されているため、負荷の高いデータベース用途で、より効果を発揮する。価格は、初期費用0円、月額129,800円（日額では5,200円）。

CONTACT ㈱リンク
URL http://www.link.co.jp

Hardware

センチュリー、ポータブル除菌消臭機「エアーサクセス」発売

㈱センチュリーは、ウィルスやカビ、菌、ニオイを除去するUSB駆動タイプのポータブル除菌消臭機「エアーサクセス」シリーズを発売した。使う場所を選ばないコンパクトなサイズ、フィルタレスによる手入れの簡単さといった“手軽さ”が売りの製品となっている。

本製品の特徴としては、次のようなものがある。

- 国際特許を取得したイオン発生技術「Maltiplex Ring Dischargerテクノロジ」を採用
- イオンと低濃度オゾンによる高い除菌消臭性能
- ファンを使わない静音設計

ラインナップは次の４商品（すべて税込）。

- 本体のみ（2,980円）
- 乾電池ユニット付属（3,480円）
- 車用電源プラグ付属（3,780円）
- AC電源ユニット付属（3,980円）

▲PCのUSBポートから給電

CONTACT ㈱センチュリー
URL http://www.century.co.jp

Report

アイ・オー・データ機器、「ポケドラCloud」編集部製品レビュー

石川県金沢市に本社を置く㈱アイ・オー・データ機器は、北陸新幹線の開業を記念し、コラボレーション商品を発売した。さらに同社の注力製品の販売促進を同時に展開している。その製品群の中から、編集部に届いた「ポケドラCloud」の製品レビューをさせていただいた。

本製品は、インターネットを利用してスマートフォンやタブレット、パソコンのデータを保存・再生できるパーソナルクラウドストレージ。今回はPC（Windows 8）からポケドラに画像を保存し、タブレット（Nexus 7）で閲覧してみる。

▲コンパクトな本体

①初期設定

LANケーブルで製品とルータを接続し、ACアダプタをつないで起動させる。本体の初期設定はとくに必要ない。スマートフォン・タブレットとの接続には専用アプリ「Remote Link Files」が用意されている。同製品に付属されているカードのQRコードを読み取ることで簡単にインストールと接続機器の登録ができた。

▲PCから画像ファイルを保存

②PCからデータを保存

同じネットワークのPCからは、エクスプローラーなどで「¥¥hls-<製品のMacアドレス>」を指定し、アクセスできる。ローカルでのファイル操作と同じように、画像ファイルをドラッグ＆ドロップで保存できた。

③タブレットから画像を閲覧

Wi-Fiに接続したタブレットでアプリ「Remote Link Files」を起動し、リストから①で設定した接続機器を選び、②で保存した画像を選択、無事にアクセスできた。

▲アプリを起動し

▲リストから機器を選ぶ

▲フォルダにアクセス

本製品はアプリを使ってスマートフォン・タブレットから簡単にでアップロードでき、外出先で撮った写真を家族と共有するなど、幅広い使い方ができる。

今回のレビューで使用したのは500GBモデル（16,500円）。ラインナップとしてはほかに、1.0TB（22,100円）と2.0TB（28,800円）がある。

▲閲覧成功!

CONTACT ㈱アイ・オー・データ機器
URL http://www.iodata.jp

Hardware

フルーク・ネットワークス、「ポケットサイズのネットワークテスター「LinkSprinter」シリーズを発売

米フルーク・ネットワークス社の日本法人である㈱TFFフルーク社は3月23日、ネットワークテスター「LinkSprinter」シリーズを発売した。

同シリーズは、軽量ポケットサイズ（大きさが110.7×40.6×32.5mm、バッテリーを含む重さが113～116g）の筐体が特徴で、次のような測定を行える。

①PoE（Power over Ethernet）
接続先ポートの電話機、監視カメラ、アクセスポイントに電力を確実に供給できるかをチェック

②スイッチへのリンク
CDP/LLDP/EDPのプロトコルを使い、接続スイッチの名前、モデル、スロット、ポート、VLANを表示

③DHCP接続
DHCPサーバが稼動状態にあり、応答するかどうかを確認する。IPアドレスをリクエストしてサブネット情報を取得し、デフォルトゲートウェイとDNSサーバを特定

④ゲートウェイ接続
Ping試験により、ゲートウェイ／ルータのアドレスおよび到達性を検証

⑤インターネット接続
クラウドへの接続性や内部ネットワークサービスへの到達性を確認して、DNSルックアップおよびアプリケーションポートの接続性を検証

また、測定時に何も操作する必要のないゼロタッチ機能により、専門知識のないユーザでも利用できる。

LinkSprinter 200と300はWi-Fiアクセスポイントを搭載し、スマートフォンとの連携稼動や「Link-Liveクラド・サービス」へのテストデータの自動保存が行える。

LinkSprinter 100（24,500円）、LinkSprinter 200（36,600円）、LinkSprinter 300（48,800円）の3製品（いずれも税別価格）がラインナップされている。


▲LinkSprinter 300

CONTACT ㈱TFFフルーク社
URL http://jp.flukenetworks.com

Hardware

アールエスコンポーネンツ、「Raspberry Pi 2 Model B」を発売

2月1日、名刺サイズのワンボードPC「Raspberry Pi」の新バージョン「Raspberry Pi 2 Model B」がアールエスコンポーネンツ㈱から発売された。

Raspberry Pi 2 Model Bは、シングルコアベースの前モデルから最大で6倍もの速度アップと、飛躍的な性能向上が実現された。新しいクアッドコアCortex-A7プロセッサに加え、1GBものRAMメモリを搭載。オペレーティングシステムカーネルは、最新のARM Cortex-A7テクノロジを活用できるようにアップグレードされ、新しいバージョン1.4のNOOBSソフトウェアを利用できる。なお、ハードウェアおよびソフトウェアの後方互換性は、Raspberry Pi 1 Model A+/B+との間で維持されているとのこと。


▲Raspberry Pi 2 Model B

また、発売の翌日にはMicrosoftがこのRaspberry Pi 2にWindows 10を無償提供すると発表したことで話題になった。これまでMicrosoftはインテルの開発ボード「Galileo」向けに開発環境を提供してきたが、新たにRaspberry Piも支援対象となりWindows 10を含む開発環境が無償で提供される。

▼おもな仕様

CPU	Broadcom BCM2836 900MHz ARM Cortex-A7 クアッドコアプロセッサ
GPU	VideoCore IV デュアルコア GPU
メモリ	1GB LPDDR2 SDRAM
ストレージ	MicroSD カードソケット
USB ポート	USB2.0×4
ビデオ	HD 1,080p ビデオ出力
オーディオ	ステレオ 3.5mm 4 極オーディオ
ネットワーク	10/100BaseT RJ45Ethernet ソケット
HDMI 出力	HDMI1.3/1.4 ビデオ、オーディオ
その他コネクタ	15 極 MPI CSI-2 コネクタ 15 極ディスプレイシリアルインターフェースコネクタ GPIO およびシリアルバス用の 40 ピンヘッダ
電源	+5V@2A、microUSB ソケット経由
寸法	86×56×20mm

本製品は下記の、アールエスコンポーネンツ㈱のオンラインストアなどから入手できる。

CONTACT RSオンライン
URL http://jp.rs-online.com

Service

エクセルソフト、「Network Performance Monitor v11.5」を販売開始

エクセルソフト㈱は、SolarWinds社のネットワーク監視ソフトウェアの最新版「Network Performance Monitor v11.5」を3月18日より販売している。

本製品は、ネットワーク上のルータなど、さまざまなデバイスのパフォーマンスに関する問題が発生する前に検知・診断・解析するWindows Server用ネットワーク監視ソフトウェア。直観的で使いやすい管理画面で、インストール後1時間以内で展開され、ネットワークの監視を開始できる。最新バージョンの「v11.5」では、ネットワーク内の帯域幅やメモリの利用量、ディスクの残りスペースなどのリソースを測定し、閾値（しきいち）を超えるキャパシティの消耗があった場合にはアラートによって通知する。また、デバイスのキャパシティやその消費量のピーク時と平均を計測し、リソースが枯渇するまでを予測、キャパシティが不足しそうな場合にもアラートする（たとえば、WAN内のディスクスペースが20日後に枯渇する場合など）。製品のラインナップと価格は右の表のとおり。

▼ラインナップ

製品名	価格（税抜）
SL-100	237,800円
SL-250	587,300円
SL-500	1,097,200円
SL-2000	2,264,900円
SL-X	4,297,800円

CONTACT エクセルソフト㈱
URL https://www.xlsoft.com

Report

「オペレーションカンファレンス 2015 Spring」開催

3月27日、「オペレーションカンファレンス 2015 Spring」が日本MSP協会の主催で開催された。

本カンファレンスは、インターネットインフラの運用やマネージドサービスプロバイダ（MSP）に対する考え、クラウドサービスの出現による情報基盤の設計・構築・運用の変化、運用サービス市場の今後についてなど、運用にかかわるさまざまな動向や各ステークホルダーの思惑思考などを共有し、議論することで健全な運用を目指すことを趣旨としている。セミナーでは運用を円滑に進めるための具体的な方法論やツールの紹介などの実践的な技術情報が示された。カンファレンスを締めくくるパネルディスカッションでは、MSPの将来についてユーザ企業側、運用企業側の立場からそれぞれ活発な意見が提示された。プライベート／パブリッククラウド混在環境での運用のあり方など課題は多く、その変化に対応しながらMSPの価値を向上させていくという。

▲パネルディスカッションの様子

CONTACT 日本MSP協会
URL http://mspj.jp

Hardware

ティントリジャパン、Microsoft Hyper-Vによる仮想化環境を新たにサポート

ティントリジャパン合同会社は3月5日、仮想化ならびにクラウド環境のスマートストレージ製品において、Microsoft Hyper-Vによる仮想化環境を新たにサポートしたことを発表した。

今回のサポート追加により、ティントリ社は主要な3つのハイパーバイザであるVMware vSphere、Red Hat Enterprise Virtualization、Microsoft Hyper-Vのすべてをサポートし、混在した環境において同一レベルのデータ管理・保護、仮想マシンの見える化を実現する。Hyper-Vをサポートすることによりティントリ社の製品は新たに、次のような特徴を持つことになる。

- NFS、SMB 3.0に対応し、複数のハイパーバイザによるさまざまなワークロードを単一のTintri VMstore上で稼働させることができる
- Hyper-V ManagerならびにSystem Center Virtual Machine Managerとネイティブに連携し、データ保護を仮想マシン単位で行える
- Hyper-V上で稼働するCitrix XenDesktopならびにXenApp VDIをサポートする

CONTACT ティントリジャパン合同会社
URL http://tintri.co.jp/

Hardware

メラノックステクノロジーズ、EDR InfiniBand製品のサンプル製品を出荷

メラノックステクノロジーズ社は、3月31日、日本国内で同社の100GB/s EDR InfiniBand製品のサンプル出荷の開始を発表した。

同社が今回サンプル出荷を行う製品は、SB7700、SB7790スイッチならびにConnectX-4アダプタ。2014年11月に発表した業界最先端のConnectX-4 100GB/s EDR InfiniBandアダプタは、片方向通信で100GB/s、双方向通信で195GB/sのInfiniBandスループット、610ナノ秒の低アプリケーションレイテンシ、毎秒1億4,950メッセージもの転送レートを実現する。

SB7700（マネジネント機能内蔵）、SB7790（外部マネジメント）は、Switch-IB（メラノックスが提供する第7世代のスイッチシリコン）を使用して開発されたEDR InfiniBandスイッチで、100GB/s EDR InfiniBand 36ポートを装備し、1Uという省スペース設計でありながら、7.2TB/sというノンブロッキングスイッチング性能、90ナノ秒のポート間レイテンシという驚異的なパフォーマンスを実現している。100GB/s InfiniBandによって、HPC、クラウド、機械学習などに最適な高レベルのパフォーマンスを実現できる。

CONTACT メラノックステクノロジーズ社
URL http://www.mellanox.com

Event

「U-22プログラミング・コンテスト2015」開催決定

4月1日、一般社団法人コンピュータソフトウェア協会から「U-22プログラミング・コンテスト2015」の詳細が発表された。

本コンテストは、1980年より経済産業省の主催により、優れた才能を持ったイノベイティブなIT人材の発掘と育成、単にプログラムのできる人材ではなく、アイデアに富んだソフトウェア開発に取り組む人材の発掘を目的として開催されてきた。2014年からは民間のIT企業から構成された「U-22プログラミング・コンテスト実行委員会」が主催し、一般社団法人コンピュータソフトウェア協会が運営事務局となっている。今年はサイボウズ㈱の代表取締役社長・青野慶久氏を実行委員長とし、「進め！　未知なる創造力　若きプログラマが進む新たな道！」というキャッチフレーズのもと開催される。

コンテストに応募できるのは、日本国内に居住する1993年4月2日以降に生まれた者。作品のジャンルは問わないが、未発表または2014年9月1日以降に発表したオリジナルのコンピュータプログラミング作品であることという条件がある。応募作品の審査に対しては「プロダクト」「テクノロジ」「アイデア」の3つの評価カテゴリを基に審査され、総合的に優れた作品、各評価カテゴリで優れた作品に各賞が与えられる。昨年のコンテストでは、Ruby向けのインタラクティブなグラフを簡単に製作できるソフトウェア「Nyaplot」、容易に並列処理を記述できるプログラミング言語「Copal」などが表彰された。審査委員としては、慶応義塾大学の夏野剛氏、まつもとひろゆき氏などが名前を連ね、次に挙げる賞が用意されている。

▲ポスター

- 経済産業大臣賞（副賞：10万円）
- 経済産業省商務情報政策局長賞（副賞：5万円）
- CSAJ会長賞（副賞：5万円）
- 各スポンサー企業賞（豪華商品）

受賞者に対しては、上記の各賞とともに提供される副賞のほか、希望によりスポンサー企業へのインターンシップへの参加権利などの特典が付加される。

コンテストの大まかな流れは次のとおり。

▼コンテストのスケジュール

応募受付期間	7月1日～8月17日
事前審査・1次審査	8月18日～9月中旬
1次審査結果通知	9月中旬
最終審査プレゼンテーション資料締切	9月25日
最終審査会（受賞者によるプレゼンテーション）	10月4日
情報化月間記念式典 （経済産業大臣賞・商務情報政策局長賞の表彰）	10月5日
入選作品展示（CEATEC JAPAN 2015）	10月7日～10日

詳しい募集要領は下記のアドレスから確認できる。

CONTACT U-22プログラミング・コンテスト2015
URL http://www.u22procon.com

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第16回 新人教育もてぇへんだぁ

本当は、学生時代から一〇A未踏プロジェクトの恐るべき新人が配属されて、戦々恐々のくつな先生にマジ励ましの言葉を！

to be Continued

皆さんのところにも新人さん、来ましたか？　今年はピチピチで元気なのが大漁ですよね！　新人がその後に死んだ魚の目にならないよう、ちゃんと見守ってあげてください。作者が新人のときは、秋葉原のベンチャー企業で、何を聞いても「man見ろ、man。なんでinfoとREADMEも見ないんだ」という返事だけが来て大変勉強になったのを覚えてます。なぜか入社1ヵ月でノースカロライナへ単身出張とかもしましたね。変わった会社でしたが、まわりに素晴らしい先輩がいたからこそ……オレ、全然素晴らしく育ってないな……。あー、えっと……新人を見守る立場の私達も、新人だったころの好奇心や向上心を忘れないよう日々精進していきたいものですね♪（最後棒読み）

Letters from Readers

賛否両論？　新MacBook発売!

3月10日に、AppleがノートPCの新製品を発表しましたね。今回はAirもProも付かない「MacBook」。1kgを切る極薄の本体にRetinaディスプレイ搭載と非常にワクワクする仕様ですが、ひとつだけのUSBポートや性能の面で少し頼りないCPUなど、ネットでは不安な声も見られます。背面のロゴが光らなくて失望した!という声もありましたが、これは重要なことなのでしょうか?

2015年3月号について、たくさんのお便りをありがとうございました！

第1特集　カンファレンスネットワークの作り方

「YAPC」「LL Diver」でインターネット接続環境を提供した、有志の技術者集団「CONBU」。そのメンバーの方々に、会場ネットワーク・無線LANのしくみ、構築の裏側、運用時のトラブル対策など、現場ならではのノウハウをお聞きしました。

大多数が集まる場所のネットワーク構築の裏方がよくわかり、勉強になりました。

兵庫県／コメットさん

大きな会場でのネットワーク設営について知る機会はあまりないので、良かった。

東京都／吉岡さん

苦労の一端とプロの一面が見られて参考になりました。

神奈川県／齋藤さん

podcastで活躍をお聞きしていたCONBUの実践ノウハウが詰め込まれていて、とても興味深く読みました。YAPCまで大規模なネットワークを扱うことはないと思いますが、今後業務にも生かせそうです。

埼玉県／犬棟梁さん

当たり前のように使えるWi-Fi環境も、エンジニアがそれぞれの会場の特性に合わせて試行錯誤しながら構築していることを知ると、ガラッと見方が変わりますね。そういった裏方の事情を知ることができてよかったという声が多く寄せられました。

第2特集　Hadoop超²入門

並列分散処理を実現するソフトウェア「Hadoop」。第1章ではHadoopの概要・基礎知識を、第2章ではHdoopのインストール方法からApache Hive・Tezを使った分散処理の実践を解説しました。

Hadoopの基本について勉強する機会が少なかったのでいいタイミングでした。

長崎県／romeosheartさん

グリッドコンピューティングと言われていた頃から分散処理嫌いで、気づいたらHadoopが全然わからなくなっていたので、概説の入門編助かりました。

神奈川県／吉田さん

Hadoopは以前使って、環境を使いこなすのが難しいなぁと感じてからウォッチしていませんでしたが、新しいしくみが増えてるんですね。記事を読んでまた試してみたくなりました。今回のTezのように、新しく追加された技術を記事にしてもらえると参考になるので、今後もお願いします。

千葉県／今井さん

8TB HDDの話題があったが、今後容量は拡大し高速性がより求められるようになるのは必然なので、こういった技術がないと対応できないと思う。

岩手県／隼さん

ビッグデータなどと併せて語られることが多いHadoopは、Tezなど実用的なフレームワークの登場によって盛り上がりを見せています。並列処理と聞くと敷居が高い技術というイメージですが、アンケートを見ると「なかなか手が出なかったがこれを機に」という方が多いようです。

短期集中連載　Bluemixでためしてみるlot入門【前編】

IBMが提供する「Bluemix」を使いながらIoT（モノのインターネット）について学ぶ前後編の連載。ブラウザさえあればという手軽さで、MQTTプロトコルを使ったIoTアプリを開発する手順を学びました。

自分も実践してみたいです。

神奈川県／眞　泰志さん

う〜ん、ちょっとやってみたくなりましたね。

徳島県／ききさん

試してみたくなります……IoTは電子工作ゴコロをくすぐりますね……。

神奈川県／くまーーーさん

IoTと聞くと少しバズワード気味ですが、センサーデバイスからデータを受け取り、サーバで処理を行って、クライアントにアクションを起こすという具体的なケースを学ぶことで印象が変わった方も多いのではないでしょうか。気軽さが売りのひとつであるBluemixには、食指が動いたという方も多いようです。

一般記事　Cisco VIRLでネットワークのシミュレーション【前編】

前後編に分けて、Cisco社製のネットワーク仮想化ソフト「Cisco VIRL」を紹介する短期連載です。前編では、ネットワーク機器全般の話題から、VIRLの概要・アーキテクチャの詳細までを扱いました。

メジャーなわりに公開資料が少ない印象なのでムズい。

神奈川県／まーく2さん

ネットワークのシミュレーションが15年前にあれば、当時の苦労はなかったのに。

奈良県／CPUは10個以上持っているさん

難しかったです。

神奈川県／あらさん

ソフトウェア、実行環境、コンピュータリソースと、仮想化はITのあらゆる分野に広がっています。ハードウェアを用意せずに本番を想定したネットワークの運用を試せるというのは、ネットワークの学習用途などにも期待できそうなソフトですね。

一般記事　Snappy Ubuntu Core

Ubuntu向けに開発された、コンテナ型仮想化環境を実現する「Snappy Ubuntu Core」。Dockerのホストとしても、また単体でも各種アプリの隔離環境を実現するものとして利用できます。記事ではコンテナ型仮想化の説明から入り、実際にSnappyを起動して基本コマンドの確認、アプリの開発などを体験しました。

現在のトレンドですね。

東京都／山下さん

普段、RedHat系ばかり触っているので、こういった機会にDebian系のOSに慣れるのもいいかなと思います。

愛知県／NGC2068さん

各種Linuxディストリビューションが、続々とコンテナ型仮想化技術を取り入れています。それらの多くは単にDockerをサポートするというだけではなく、自前でコンテナ技術を実装しています。ユーザ側としては、勉強するものがますます増えてしまいますね。

一般記事　「Fx0」が開発者にお勧めなワケ

Firefox OS搭載のスマートフォン「Fx0」。海外ではいくつかの端末がすでにリリースされていましたが、日本でもついに発売されました。細部までこだわったハードウェア、既存のWeb技術をソフトウェア開発に応用できるOSの環境など、ほかとは違う「Fx0」を紹介しました。

Fx0気になっています!

東京都／n0tsさん

「Fx0」なんて読むんだろう……。

愛知県／kmさん

Firefoxを手伝うのは卒業してしまったんですが……記事としてはおもしろく読みました。

千葉県／Tayuさん

興味を持っているという声が多く寄せられました。HTML5/CSS/JavaScriptというお馴染みの技術でスマホアプリを開発できるので、Webエンジニアにとってもすそ野の広いガジェットとなっています。ちなみに読み方は「エフエックスゼロ」です。

3月号のプレゼント当選者は、次の皆さまです

①テレビ用高音質スピーカー「Olasonic TW-D6TV」
群馬県　並木正直様

②モバイルバッテリーチャージャ「ZM-MB350-W」
東京都　安田幸彦様

③SoftLayerノベルティ手帳
大阪府　澤下夏実様
熊本県　鈴木浩様
埼玉県　小堀大介様

④詳解Swift
福岡県　石内博子様
石川県　荒田真一様

⑤Webエンジニアが知っておきたいインフラの基本
東京都　野崎啓太様
埼玉県　片野敬勇様

⑥Python言語によるプログラミングイントロダクション
和歌山県　兼行大将様
福岡県　田代海霞様

⑦MariaDB＆MySQL全機能バイブル
京都府　前場英二様
埼玉県　橿山公一様

※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（プレゼント応募後に住所が変更されている場合など、お届けできないことがあります）。2ヵ月以上ご連絡がない場合は、再抽選させていただくことがあります。

次号予告

Software Design
June 2015

2015年6月号
定価（本体1,220円+税）
192ページ

5月18日発売

［第1特集］新人さん歓迎特集

Git＆GitHubのABC
――開発現場のはじめの一歩

そろそろ新人さんも現場へ実戦投入!――の時期ではないでしょうか。ソフトウェア開発において、一番大事なのは、ソフトウェアにデグレ（手戻り）を起こさず品質を向上させること。そのために先輩たちは、GitやGitHubを活用して仕事をしています。本特集は初心者を対象に、ちょっと扱いが難しいGitとGitHubをやさしく解説します。

［第2特集］認証システムの定番

［入門］OpenLDAPの教科書
――ユーザ／ネットワーク管理の基礎を固める

※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

「RHELを極める・使いこなすヒント .SPECs（第12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。

■2015年4月号
●第1特集1-1　p.21　図1中のルータの右下のIPアドレス
［誤］192.168.2.2　［正］192.168.2.254
●第1特集2-2　p.46
［誤］㈱サーバワークス　［正］㈱サーバーワークス

SD Staff Room

●今年度の表紙のテーマは野生動物です。デザイナーさんから提示されたもののうち、虎のりりしい顔に惹かれ、これに決めました。そして春から新入社員の皆さんに向けての小冊子特別付録です。とにかくWebのしくみを押さえておこうというのが目的です。IT業界はとにかく勉強しつづけないとね!（本）

●連載にアカモクギバサなるキャラが……。えっ、三陸で一番旨い海藻は生マツモでしょ。日本は海藻の種類が豊富で季節毎に美味を提供してくれる。北陸で旨い海藻はツルモ。それと2月頃の岩場で採れる海藻をいしるで食べる海藻しゃぶしゃぶの旨いこと!　黒もずくも捨てがたい。（異論受け付けます海藻フリーク幕）

●毎日持ち歩ける高倍率ズーム付きコンデジが2万円くらいで欲しい!　変わりモノ好きな私はQX10に決めかけていたんですが、本誌3月号カンファレンスNW構築の記事を読んで一転。イベント会場での無線LAN接続の難しさを知り、WX350にしました。でも分離・合体は男のロマンだった……（キ）

●先日某国立大学の前を通ったら、入学式が行われていました。ついこの前、センター試験や二次試験をやっていたと思ったら、もう入学。内定が出てから約1年後にやっと入社する就職活動とはずいぶん違うと感じました。正直、1年前に内定をもらったって、入社の頃にはモチベーション下がっていそう。（よし）

●最近、ポテトサラダにハマっています。ジャガイモ（メークイン派）を茹でたら皮付きのまま潰して、たっぷりのマヨネーズと和える。ソーセージときゅうりを小さく切って混ぜ込んだら、黒胡椒をふって完成!　ビールによく合います。きゅうりはしっかりと水抜きをするのが今日のポイントですね。（な）

●近所に有名なお花見スポットがあり、初めて友人を招いてお花見をしました。場所取りの競争が激しく、朝5時でも一番乗りではなかったようです。前日、春の嵐に見舞われたので桜が残っているか心配でしたが、十分な量が残っていて綺麗でした。でも場所取り係は色々大変なので、来年は見物だけにします。（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

［FAX］
03-3513-6173
※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

［E-mail］
sd@gihyo.co.jp

Software Design
2015年5月号

発行日
2015年5月18日

●発行人
片岡　巌

●編集人
池本公平

●編集
金田富士男
菊池　猛
吉岡高弘
中田瑛人

●編集アシスタント
松本涼子

●広告
中島亮太
北川香織

●発行所
㈱技術評論社
編集部
TEL：03-3513-6170
販売促進部
TEL：03-3513-6150
広告企画部
TEL：03-3513-6165

●印刷
図書印刷㈱

特別付録

3分間 HTTP& メールプロトコル 基礎講座

特別編

網野衛二 著
EIJI AMINO

某所の某大学にて、情報処理を教える博士。専門はネットワーク。たった 1 人しかいないゼミ生であるネット君をこき使う。

インター博士のただ 1 人のゼミ生。ネットワークについてはまったくの素人。

Webのしくみを基礎の基礎からマスター

◎WWW、Hyper Text Markup Language、URIの基礎
◎HTTPの基本動作を完璧に押さえる
◎メッセージヘッダの種類と役割、Getメソッドを深く理解

技術評論社

第1回

World Wide Web

●インターネットとWWW

3分間HTTP&メールプロトコル基礎講座！！

どんどんぱふぱふ♪

さて、今回はインターネットの主役、Webとメールを中心にアプリケーションのプロトコルを説明していく講座だ。これで、ネット君をもうすこしましなネットワークエンジニアに育成するのが目的だ。

『3分間DNS基礎講座』（当社刊）に引き続き育成なんですね、よかった。

まぁ、そろそろそれぐらいできてもらわんと、教える側としても立場がない。さて、今回最初に話すのは、**World Wide Web**だ。略して**WWW**、もしくは単純に**Web**とも呼ぶな。

わーるどわいどうぇぶ。インターネットですね。

あーうん。確かにそう誤解している人もいるな。だが、「インターネット≠WWW」だ。インターネットとはネットワークそのものを指すからな。ともかく、まず君が想像しているWWWとはどんなものかを聞いておこう、ネット君？

え〜っと、あれですよね、「ホームページを見る」ことができる？　そうだ、「ホームページ閲覧サービス」ですよ。

まぁ、大きくは間違ってはいないが、曖昧すぎるな。ホームページを見ることができれば、それはWWWなのかね？　では、公開されているホームページをFTPでダウンロードしてきて閲覧しても、それはWWWなのかね？

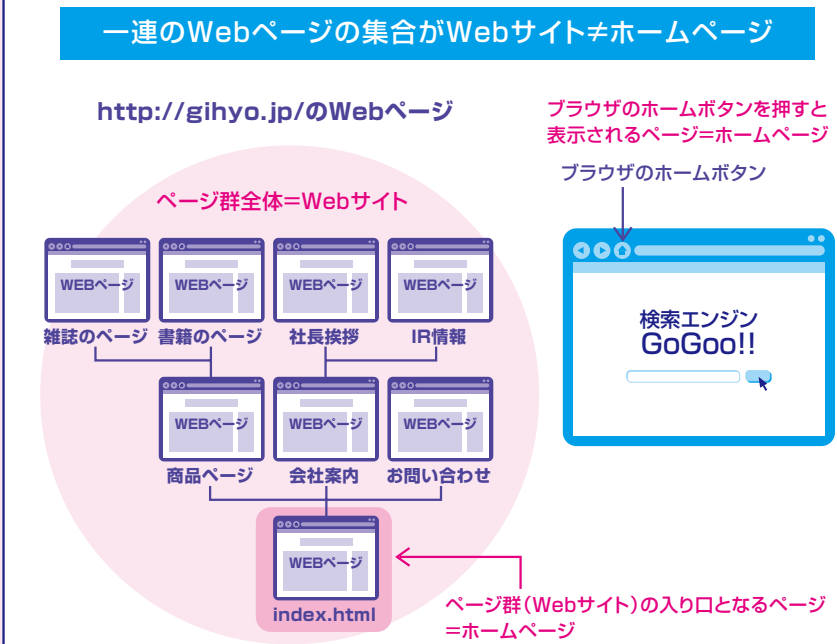
う、う〜ん。そういわれれば……FTPでダウンロードしたのを見るのは「ホームページ」じゃないですよね。

WWWとは何か、を説明する前に用語の説明をしたほうがよさそうだな。まず、「ホームページ」という曖昧な言葉を使うのはやめたまえ。「ホームページ」とは本来「ブラウザのホームボタンを押したときに表示されるページ」または「一連のWebページ群の入り口にあたるページ」のことを指す。

え？　そうなんですか？　じゃあ普通に使われる「ホームページ」を指す言葉はなんですか？

一般的には「Webサイト」と呼ぶな。ただ、ホームページという言葉は結構広範囲に使われているから、「Webページ」を指す場合もあるけどな（**図1-1**）。

図1-1　ホームページとWebサイト



ははぁ、WebサイトとWebページ、ですね。Webページがまとまって、1つのWebサイトを作る、と。

そういうことだ。さて話は戻って、WWWとは何か、という話だが。そうだな、**ハイパーリンクによる広域情報共有システム**というのが、私の解答だ。**(*1)**

ハイパーリンクによる広域情報共有システム？　ハイパーリンクってなんですか？　あと、広域情報システムってどういうことですか？

ハイパーリンクについては後で話すとして、WWWはおもにインターネットを使った「広い範囲で」、Webサイトに書かれた「情報を」、公開したり見たりすることにより「共有するシステム」、ということだな。

ふむふむ、確かにホームページ、じゃなかったWebサイトを見るのは、Webページに書かれた情報を共有していると言えますね。

そうだろう？　そして、このWWWを構成する3つの中核技術が、データの記述方法である**HTML**、データの位置情報である**URI**、データの転送方法**HTTP**だ。

HyperText Markup Language、Uniform Resource Identifier、Hyper Text Transfer Protocolですか？

そうだ。つまりWWWとは「HTML」によって書かれた文書を「URI」によって特定し「HTTP」でやりとりする「ハイパーリンク」によってつながれた「情報共有システム」、ということだな。

HTML、URI、HTTP。そしてハイパーリンクですね。これらによって情報の共有が行われるのがWWWだと。

●ハイパーリンクとハイパーテキスト

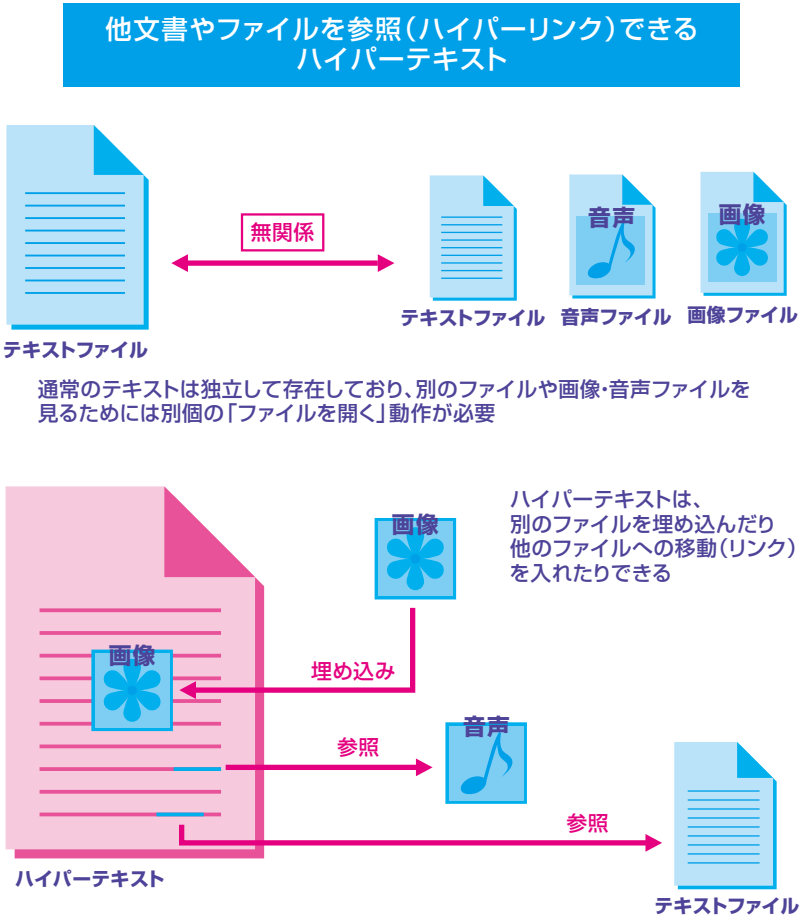
ではまず、「ハイパーリンク」とHTMLについて話そう。「ハイパーリンク」は「ハイパーテキスト」を**相互に結びつける**もので、まぁ、そうだな日本語に直すと「参照」なんだけどな。

(*1) ハイパーリンク [Hyperlink]

「参照」って言っちゃうと、なんかかっちょよさが薄れますね。んで、ハイパーテキストってなんですか？　それを相互に結びつける？

ハイパーテキストはハイパーリンクが可能な「ハイパー」な文書、だ。つまり通常の文書（テキスト）では不可能な、**他の文書や画像・動画などを「参照」し埋め込んだり連結したりできる**文書、ということだな（図1-2）。

図1-2　ハイパーリンク



あー、あれですか。ホームペー……じゃなくてWebページで使われてる「リンク」とか「ジャンプ」とか。あと画像を貼ったり？

そうだ。どれも普通のテキストでは不可能だ。これを行うことができる文書が「ハイパーテキスト」。WWWでは、**ハイパーテキスト記述言語であるHTML**で記述されたハイパーテキストが使われている。

HyperText Markup LanguageでHTML。ハイパーテキスト、マークアップ、言語。あれ？　マークアップってなんですか？

普通のテキストというのは、単なる文字の羅列（られつ）でしかない。タブや改行程度のことはできるがな。これに文章の「見栄え」や「構造」を付け加える命令である「タグ」を付けたものが、マークアップ言語だ**(図1-3)**。

ははぁ、タグ。え〜っと、あれですか？　Webページを作る時に使う＜ｐ＞とか＜h1＞とか？

そう、それだ。それにより単なる文章だけだったテキストに、「意味」を付けることができる。段落、見出し、強調などだな。つまりHTMLとは「ハイパーリンク」を持つ「ハイパーテキスト」で、「マークアップ」可能な文章を作成する言語、ってことだ。

なるほど。ハイパーテキストとマークアップ、ですね。で、WWWではこれを使っている、と。

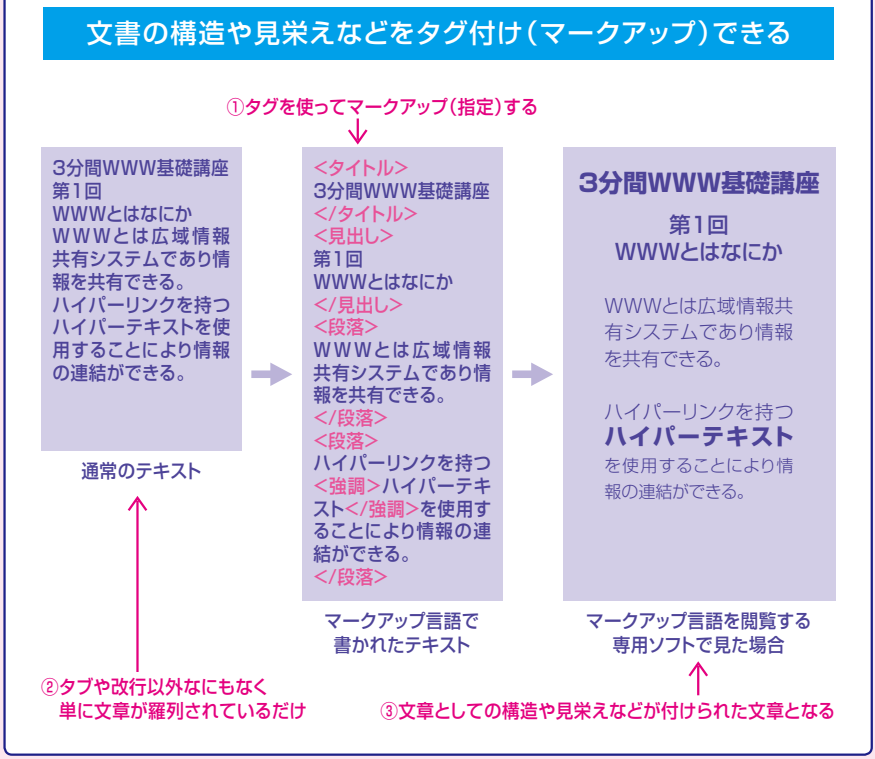
うむ。ハイパーリンクを持つハイパーテキストを使うことにより、「情報の連結」が可能になるわけだ。WWWを使っていてリンクができないことなんて考えられないだろう？　ハイパーテキストによるハイパーリンクは、WWWの中核とも言えるだろう。

確かにリンクがないと不便ですよね。そうかぁ、情報の連結かぁ。

情報を連結することにより、簡単に「情報の共有」が可能になるわけだな。さて、今回はここまでとしよう。

あいあい。3分間HTTP&メールプロトコル基礎講座でした〜♪

図1-3　マークアップ言語



ネット君の今日のポイント

- **WWWは広域情報共有システム。**
- **URI、HTML、HTTPの3つの基幹技術によりWWWは構成される。**
- **HTMLはハイパーリンクを持つハイパーテキストをマークアップで記述する言語。**
- **ハイパーリンクにより容易に情報の共有が可能になっている。**

〇月〇日　ネット君

第2回

URI

●URI

さてさて、WWWには3つの基幹技術があった。HTML、HTTP、URIだ。今回はこのURIについて話す。Uniform Resource Identifier、でURIだな。

ゆーあーるあい？　あの、博士、URL ってのはホームページ……じゃなくてWebページ関係で聞いたことがあるんですが、それとは違うんですか？

URL、Uniform Resource Locatorのことだな？

そうです、それです。IdentifierとLocatorしか違わないじゃないですか？関係あるんですか？

もちろんある。順番的に言えば、URLが最初にできた。そのあと、この概念を拡張して標準たるURIができた、というわけだ。

ん〜っと、最初にURLがあって。URLを拡張してURIができる。……標準？URIは標準なんですか？　URLは標準じゃない？

じゃないな。URLは正式には「非公式な名前」として扱おう、って話になっている。標準はあくまでもURIだ。さて、URIとは何かという話をしてなかったな。URIは**リソースを示す統一的な識別子**だ。

統一的な（Uniform）、リソース（Resource）、識別子（Identifier）。そのまんまじゃないですか、もうちょっと詳しく教えてください。

リソース、つまりモノだ。文書だったり、画像だったり、本だったり、電話番号だったりなんでもいい。その**リソースの書き方を定めたものがURI**だ。書き方は**スキームとリソース**からなる（**図2-1**）。

図2-1　URIとURL

「リソースの意味」と「リソースそのものの記述」？　これって何に使うんですか？

うむ、これを使って「リソースの場所」を示したり、「リソースの名前」を示したりする。
たとえば、http://www.3min.jp/index.htmlならば「httpで取得できるwww.3min.jpサーバにあるindex.html」という「場所」を意味するし、urn：ietf：rfc3989ならば、「IETFによって管理されているRFC3989文書」という「名前」を示す。これらを同じ形式で書こうって決めたのがURI ってことだな。

……博士、そのhttp://www.3min.jp/index.html ってURLですよね？　それを今ではURI って呼ぶってことですか？

そうだなぁ、もともとURLが先にあったということはさっき話したが、URLの書き方をもっと全般的に使おうと新しく定義しなおしたのがURI、だな。おかげで使える範囲がぐっと広がった。それで、かつてURLと呼ばれていた書き方もこの新しい定義の中に入ってしまっている(**図2-2**)。

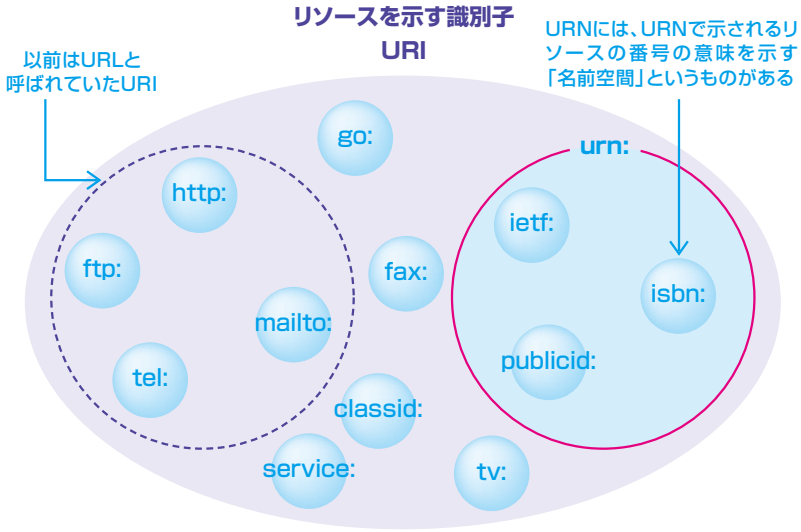
なるほど。URIに含まれちゃったので、使わないようにしようってことで「非公式」ってわけですね。でもそれだったら廃止しちゃえばいいのに。

たぶんURLという用語が定着してしまっているので、いまさら廃止ってわけにもいかないんだろう。ともかく、リソースを記述するものがURI、と覚えておきたまえ。

はい。それで、URIはWWWではどう使うんですか？

図2-2　URIとURL

URLは現在では非公式
URIのうちhttp:などがスキームのもの



●WWWで使われるURI

うむ、**WWWではリソースの場所を記述するためにURIを使用する。**主に**スキームとしてhttp：を使うURI**が使われるな。ほかにもftp：やmailto：、携帯電話のWebページではtel：などのスキームも使われているな。

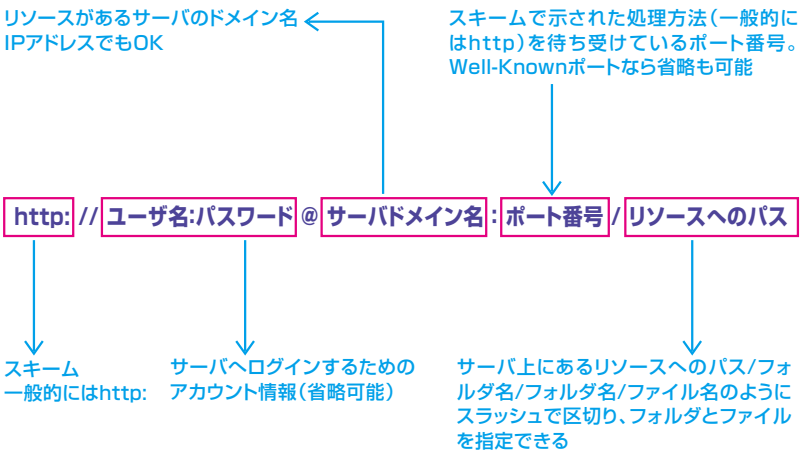
P9の図でhttp：やftp：はリソースの処理方法を示すってありましたけど。どういう意味です？

そうだな。http：やftp：がスキームの場合、**後ろのリソース部にはリソースのWWWでの場所を記述する。**つまりスキームと併せて、**その場所にあるリソースをスキームのプロトコルで処理する**という意味だ、と考えておけばいい。WWWでの一般的なURIの記述は図のとおりになる（**図2-3**）。

んんん？　ユーザ名：パスワード？　ポート番号？　そんなの記述しましたっけ？　Webサイトを見るときに、打ち込んだことないですよ？

図2-3　WWWでのURI

一般的にはhttp:がスキームで、サーバドメイン名、リソースへのパスなどから構成される



WWWは情報共有システムとして、「誰でも見られる」ことが前提となっている。そのため、本来なら「他のコンピュータの情報を取得する」ために必要な「ログイン」を**省略している**。その結果、ユーザ名とパスワードは通常書かなくてもよく、省略できる。**(*1)**

そうか、本来なら「ログイン」しなきゃダメなんだけど、それが省略されているってことなんですね。まぁ、会員制でもない限り、いちいちユーザ名とパスワードは必要ないですよね。あと、ポート番号は？

本来ならポート番号を付けるのが正式だ。だが、httpはWell-Knownポートとして80番を使用できるので、80番を宛先とする場合は省略できる。

なるほどなるほど。Well-Knownポートについては『3分間ネットワーク基礎講座』や『3分間DNS基礎講座』を読むといいですよね。

露骨な宣伝ありがとう。さて、ポート番号のうしろには、スラッシュのあとに要求したリソースの、そのサーバ上での場所を記述する。

でも博士、いちいち書かない場合もありますよね。技術評論社のサイトはhttp://gihyo.jp/ って書かれていることが多いし、博士のサイトだってhttp://www5e.biglobe.ne.jp/~aji/だから、リソースが書いてないですよ？

うむ。その場合は要求を受け取ったWebサーバのアプリケーションが、自動的にリソース名、通常はファイル名だが、それをくっつける。ここでのポイントは、最後にスラッシュがあるかないか、だ。

最後にスラッシュ？　たとえば「http://www5e.biglobe.ne.jp/~aji/」と「http://www5e.biglobe.ne.jp/~aji」みたいにですか？

うむ。その場合、動作が異なる。スラッシュがないとファイル名とみなしてしまうのだ(**図2-4**)。

(*1) 省略している　実際はログインを行わないわけではなく、「匿名ログイン」を行っている。詳しくは(『3分間HTTP&メールプロトコル基礎講座』(当社刊) P99参照)。

図2-4　リソースの最後のスラッシュ

最後にスラッシュがあるのとないのとでは動作が異なる

gihyo.jpサーバの場合 ……リソース名が省略されたら
index.htmlに対する要求とみなす

要求のURI	サーバが認識するURI	サーバの動作
http://gihyo.jp/	http://gihyo.jp/index.html	index.htmlを追加する
http://gihyo.jp	http://gihyo.jp/index.html	スラッシュとindex.htmlを追加する

サーバドメイン名の後ろのスラッシュがなくても、自動的にスラッシュを追加する

要求のURI	サーバが認識するURI	サーバの動作
http://gihyo.jp/tcpip/	http://gihyo.jp/tcpip/index.html	index.htmlを追加する
http://gihyo.jp/tcpip	http:/gihyo.jp/tcpip	tcpipファイルを探す

リソースのパスの最後にスラッシュがないと、ファイル名とみなし、そのファイルを探す。
もしそのファイルがない場合は、スラッシュをつける（P29参照）

へー、最後にスラッシュがないと、ファイル名とみなしてしまい、そのファイルがなければ最後にスラッシュをくっつけるんですね。

そうだ。このあたりの詳しい動作の違いは先で説明しよう（P28参照）。今回はここまでとしておこう。

はい。3分間HTTP&メールプロトコル基礎講座でした〜♪

ネット君の今日のポイント

- **URIはリソースを記述する書き方。**
- **WWWではhttp：などのスキームのURIが使われる。**
- **WWWのURIでサーバ名、リソースの場所を指定する。**

○月○日 ネット君

第3回

HTTPの動作

●HTTP

さて、WWWを構成する基幹技術のうち、HTMLとURIを説明した。今回からはHTTP、HyperText Transfer Protocolを説明する。

ハイパーテキストであるHTMLと、その場所を特定するURIですね。で、HTTPがそれの転送？

そう、HTTPは**ハイパーテキストの要求と転送を行うプロトコル**だ。最初に、HTTPのバージョンについて説明しておこう。HTTPのバージョンは「メジャーバージョン.マイナーバージョン」の書き方で記述され、0.9、1.0、1.1がある。まずWWWが開発された1989年に作られた最初のHTTP0.9があり、これを1996年に標準規格化したHTTP1.0がある。

へぇ、規格化までにずいぶん時間がたってるんですね。1.1はいつですか？

HTTP1.1はHTTP1.0を拡張したもので、HTTP1.0の1年後の1997年に規格化されている。**現在は1.1が主流**だ。ただし、WWWで使われるソフトは最低でも1.0に対応しなければいけない。WWWで使われるソフトというのは、要求を出すクライアントと、要求を受け取り応答を返すサーバソフトだな。

あれですね、**ブラウザ**ですね！！（*1）

そうだな。一般的にはブラウザだな。他にも、検索サイトが使う**ロボット**などもある。これらをまとめて、**ユーザーエージェント（UA）**と呼ぶ。一方のサーバソフトは**Webサーバアプリケーション**と呼ばれることが多い。その基本的な動作はこうなる（図3-1）（*2）（*3）。

えっと、まずURIのドメイン名をDNSで名前解決して、その後Webサーバアプリケーションへ「リクエスト」を送る。Webサーバアプリケーション側は「レスポンス」を返すわけですね。DNSについては前著『3分間DNS基礎講座』(当社刊)で詳しいですよね。

露骨な宣伝を何度も入れてくれてありがとう。ともかく、そのとおり。ポイントとしては、「ホームページを見る」とよく言うが、実際にはサーバにあるファイルを「見て」いるわけではない、というところだな。

え？　「見て」るんじゃないとすると、どうしてるんですか？

実際は「ダウンロードして、キャッシュに保存する」のだよ。それをUAで「開いて」「見て」いるのだ。単純に「見る」というとサーバ上にあるものを直接開いているというイメージがあるが、実際は「ダウンロードしている」ということだな。

キャッシュってあれですね、一時的に保存する場所のことですよね。そっか、そこに一時保存してから見ているんですね。

うむ。キャッシュについてはまた先で出てくるので覚えておいてくれ（P43参照）。そしてもう1つのポイントは、1つのWebページを閲覧するのには通常、複数のリクエストとレスポンスを行う、ということだな(**図3-2**)。

えっと、HTMLファイルを受け取ったらそこに書かれている内容を見て、画像などが必要だったらそれを別個に要求する？

そうだ。よく文字だけ表示されて、後で画像が出てくることがあるが、それはHTMLファイルの取得、HTMLファイルの解析、画像タグの発見、画像の要求と応答、という手順を踏んでいるからだ。

だから画像だけ後で遅れて出てくることがあるんですね、なるほど納得です。

(*1) ブラウザ [Browser]　正確にはWebブラウザ。画像ファイルの閲覧(画像ブラウザ)やファイルを見るためのブラウザ(ファイルブラウザ)もある

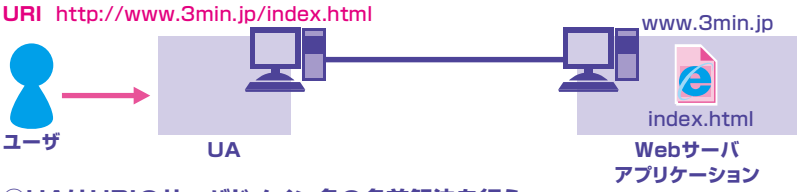
(*2) ロボット [Robot]　自動でWWWを巡回し、検索可能な情報を集めるソフトウェア。エージェント[Agent]とも呼ばれる。

(*3) ユーザーエージェント [User Agent]

図3-1　UAとWebサーバアプリケーション

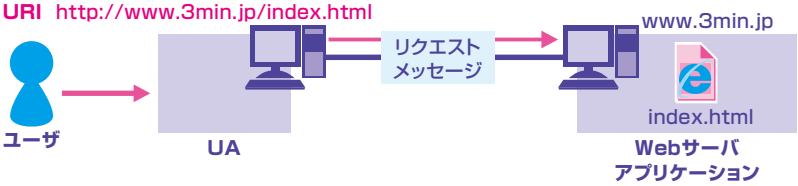
クライアントのUAからのリクエストに対し、
サーバのWebサーバアプリケーションがレスポンスを返す

①ユーザがUAに対し、要求するリソースのURIを入力する

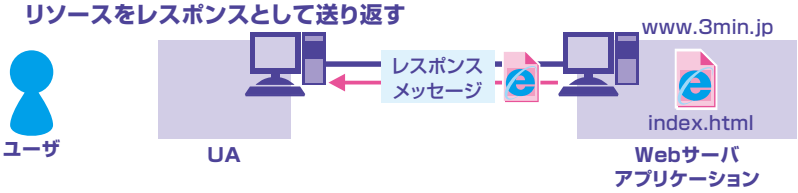


②UAはURIのサーバドメイン名の名前解決を行う

③UAはサーバあてにURIのリソースのリクエスト（要求）を送信する



④Webサーバアプリケーションはリクエストを受け取り、要求されたURIのリソースをレスポンスとして送り返す



⑤UAは受け取ったリソースをキャッシュとして保存し、その後画面に表示する

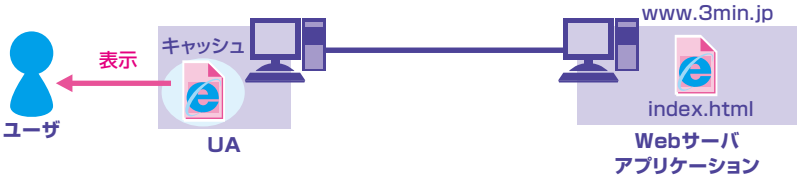
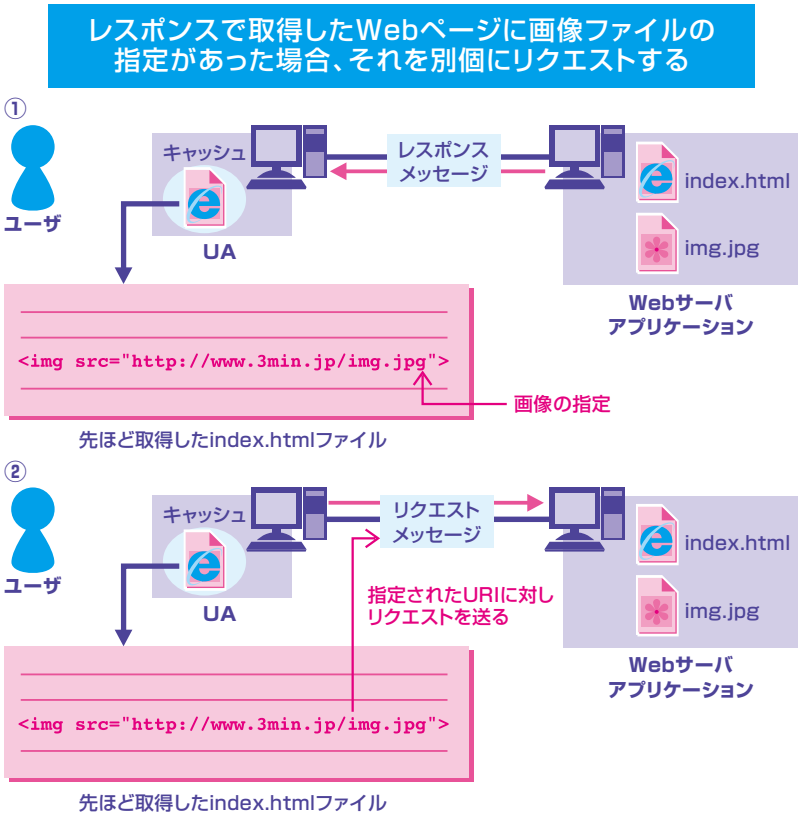


図3-2　Webページの閲覧



●HTTPメッセージ

さて、実際のリクエストとレスポンスの内容について説明していこう。まず覚えておいてほしいのが、HTTPは**文字ベースのリクエストとレスポンスを行う**ということだ。それと**改行が意味を持っている**ことも覚えておいてくれ。

文字ベースってどういう意味ですか？　あと、改行が意味を持つって当たり前のことのような気がしますけど？

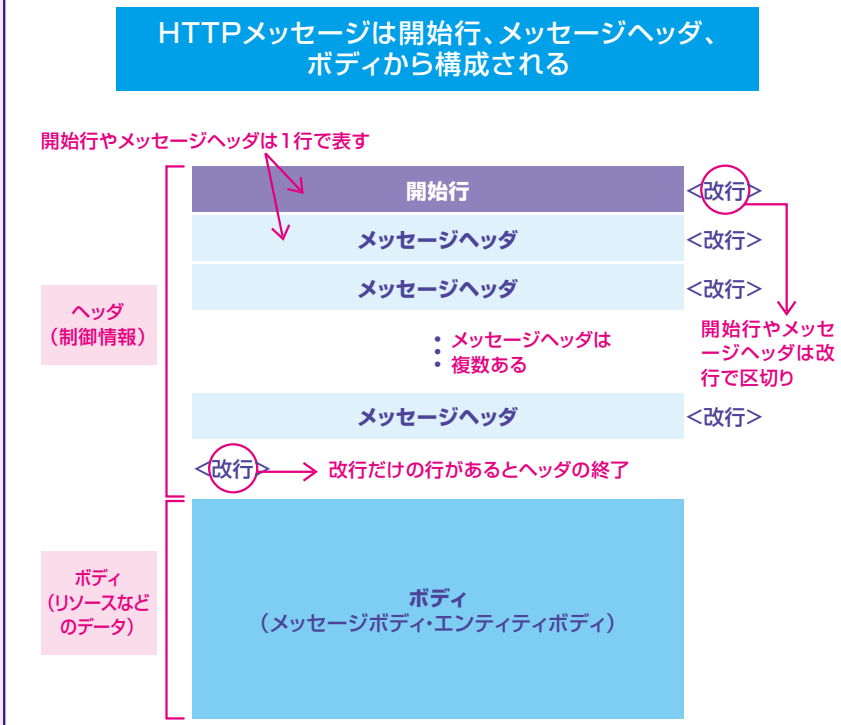
文字ベースというのは、リクエストやレスポンスが文字でのみ制御される、ということだ。DNSみたいに、ビットが0か1かで動作が決まる、などということがなく、すべて文字情報でやりとりされる。改行については、まぁ後でわかる（P18参照）。

そういえば、DNSではフラグがあって、そこのビットによってやりとりの内容が決まっていましたよね。詳しくは『3分間……。

それはもういい。ともかく、リクエストとレスポンスの制御情報は文字で書かれるということがわかればいい。さて、リクエストとレスポンス、まとめてHTTPメッセージと呼ぶが、この構成は「ヘッダ」＋「ボディ」の形をとる（**図3-3**）。

ヘッダとボディ。ヘッダはさらに、「開始行」と「メッセージヘッダ」があって、その後ろにボディ。……この間には改行コードが入る？

図3-3　HTTPメッセージ



そうだ。HTTPのヘッダでは、**改行コードにより区切りが行われる**。さっき改行が意味を持つといったのはこのことだ。

なるほど。改行があると、開始行の終わりだとわかるし、改行が2つ続くとヘッダの終了がわかるんですね。で、このメッセージヘッダは複数あるんですか？

うむ。開始行と複数のメッセージヘッダで「HTTPヘッダ」が構成される、ということだ。メッセージヘッダはHTTP1.1では最低1つは必ず必要だ。それ以上は必要に応じて付けることになる（P32参照）。

ヘッダがいつくかある、と。ちなみにボディには何が入るんですか？

うむ、レスポンスの場合はわかるだろう？　HTMLファイルや画像ファイルなど、リクエストによって要求されたファイルのデータが入ることになる。リクエストの場合は、「ボディなし」か、Webページを見るために必要なデータを入れることになる。

Webページを見るために必要なデータって、URIとかですか？

いや、URIは開始行にある。この場合の必要なデータというのは、例えば掲示板などに書き込んだり、オンラインショッピングで商品を選択したりした場合に、サーバに送るデータのことだな。こういう情報が必要ならボディに入れるし、必要ないならボディなしになる。では、今回はここまで。

はいな。3分間HTTP&メールプロトコル基礎講座でした～♪

ネット君の今日のポイント

- **HTTPではUAとWebサーバアプリケーションでデータのやりとりをする。**
- **HTTPでやりとりするHTTPメッセージにはリクエストとレスポンスがある。**
- **HTTPメッセージは開始行、メッセージヘッダ、ボディからなる。**

○月○日　ネット君

第4回

HTTPリクエスト

●メソッド

さて、HTTPでやりとりされるメッセージにはリクエストとレスポンスがあり、それの構造を理解できたと思う。

はい。開始行、メッセージヘッダ、ボディから構成されているんですよね。

今回はリクエストの開始行である、**リクエスト行**について説明しよう。リクエスト行には3つの情報が書かれている。**メソッド**、**リクエストURI**、**HTTPバージョン**だ。この3つは**空白文字で区切られる**。最後に改行があって、リクエスト行は終了となる。(*1)

URIとHTTPバージョンはわかりますけど、メソッドって何ですか？

そうだな、「リクエストの内容」とでも覚えてもらうといい。つまり、**リクエストで何を要求したか**ということだ。

ん〜、WWWなんですから、リクエストで要求するのは、Webページや画像なんじゃないですか？

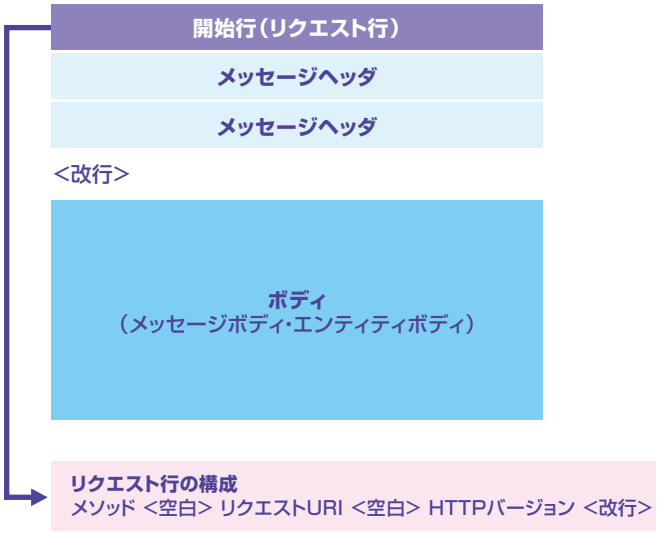
もちろんそれが多いが、それだけではない。メソッドには8種類とその拡張がいくつかあり、それぞれWebサーバに要求する項目が異なる。詳しくは先で説明するので（P43参照）、ここでは簡単に説明しておこう。まず**GET**。これは先ほどネット君が言ったとおり、**リソースの転送を要求**するものだ。これはHTTP0.9、つまり最初のHTTPから存在する基本のメソッドだな(**図4-1**)。

(*1) リクエスト行 [Request-Line]、メソッド [Method]

図4-1　リクエスト行とメソッド

リクエスト行は メソッド、リクエストURI、HTTPバージョンから構成される

HTTPリクエストメッセージ



代表的なメソッド

メソッド	対応バージョン	意味
GET	HTTP0.9～	リソースの取得の要求
HEAD	HTTP1.0～	リソースの取得の要求（ヘッダのみ）
POST	HTTP1.0～	データの送信と処理の要求
PUT	HTTP1.0～	ファイルの転送
DELETE	HTTP1.0～	ファイルの消去
OPTION	HTTP1.1～	メソッドやオプションの確認
TRACE	HTTP1.1～	経由するサーバのトレース
CONNECT	HTTP1.1～	中継サーバのトンネリング
PROPFIND	HTTP1.1（WebDav）～	ファイルのプロパティ取得

リソースの要求ですから、HTMLファイルや画像ファイルを要求して、送ってもらうっていうリクエストってことですよね。というか、これ以外のリクエストってのが思いつかないんですが？

その他の代表的なメソッドとしては、**POST**がある。POSTは掲示板などで**データを送信する**際に使用される。つまり、POSTする、「投稿する」ために使われるメソッドだな。**CGI**などにデータを送る時に使う。**(*2)**

そういえば、掲示板に書き込むことを「投稿」って言ったりしますね。で、CGI ってなんですか？

CGIは動的ページ、つまりこちらの入力に合わせてページを新しく作成するためのしくみだ。掲示板はこちらが入力した内容によって、表示されるページの内容が変わるだろう？　つまり、こちらの入力によってページの内容が新たに作られているわけだ。このような動的ページを作るしくみが、CGIと呼ばれている。

言われてみればそうですね。掲示板とかって、事前に用意してあったWebページを表示しているわけじゃないですもんね。

このように、「そのリクエストがどのような内容か」を決めるのがメソッド、ということだな(**図4-2**)。

●リクエストURI

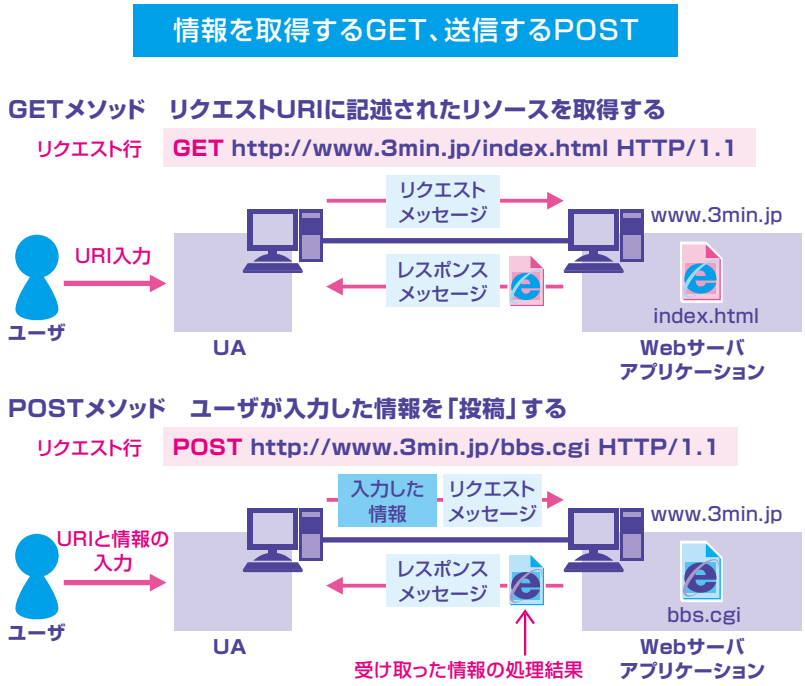
リクエスト行に、メソッドの次に書かれているのがリクエストURIだ。これはリクエストする**リソースを指定**するためのものだ。これの書き方だが、まず**相対URIを記述**するのが一般的だ。

URIは、前回出てきたリソースの識別子ですよね。相対URIの記述が一般的？　相対URI ？　一般的ってことは他の書き方もあるってことですか？

まぁまぁ、いくつも質問するな。まず、相対URIだが、これは**ホームディレクトリから見たリソースの場所**だ。ホームディレクトリとはWebサーバアプリケーションが指定した、そのWebサーバでの一番上のディレクトリ(フォルダ)のことだ。

(*2) CGI【Common Gateway Interface】　動的ページを作成するためのプログラムを呼び出すしくみ。

図4-2　GET、POSTメソッド



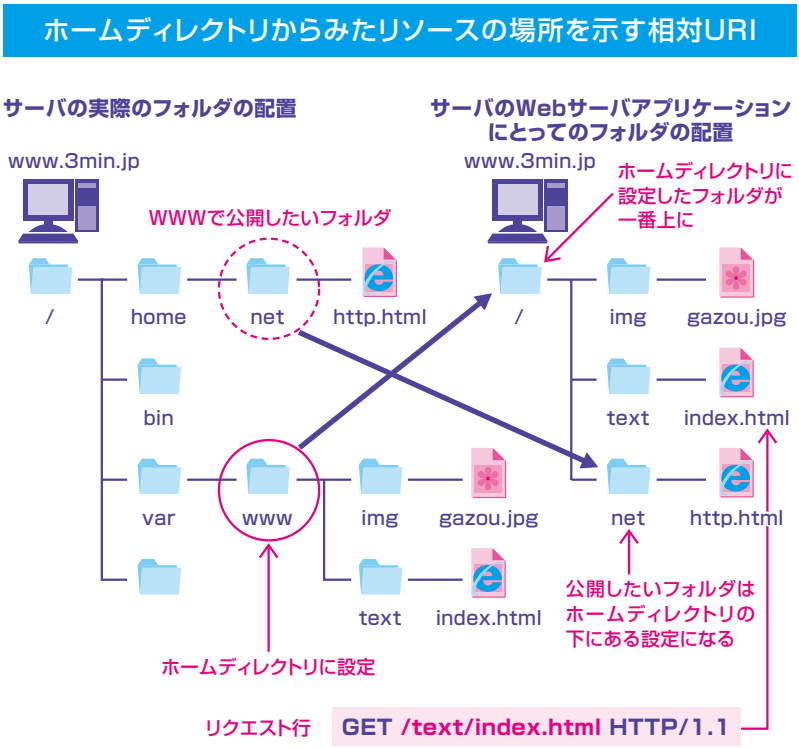
サーバで一番上のフォルダというと、WindowsならCドライブ、Linuxなら/（ルート）ですか？

いや、Webサーバアプリケーションにとっての一番上だ。なので、どこでもいい。すべてのリソースはこのホームディレクトリの下に配置されることになる。このホームディレクトリから見た場所を、リクエストURIに記述する。これが相対URIだ。相対URIは必ず**スラッシュから始まる**（図4-3）。

ははぁ、/text/index.htmlだとすると、ホームディレクトリの下にあるtextフォルダの中のindex.html って意味になるわけですね。

そういうことだ。これが、相対パスによって表記されるリクエストURIだ。これが通常の記述になるが、例外として、**絶対URIを記述する**場合がある。これは**プロキシサーバ経由でのリクエスト**を行う場合に使用する。

図4-3　相対URIによるリクエストURI



絶対URI？　プロキシサーバ？　なんですかそれ？

プロキシサーバについてはあとで詳しく説明するが（『3分間HTTP&メールプロトコル基礎講座』P78参照）、簡単に言えばHTTPメッセージを中継するサーバのことだ。プロキシサーバを使用する場合は、絶対URIを記述しなければならない。絶対URIは**http://から始まるURI**で、WWWでのリソースの場所を示す。

ふむふむ。相対URIが「ホームディレクトリからの場所」を指すのに対し、絶対URIは「WWW上で配置されている場所」を指すわけですね。プロキシサーバが中継する場合は絶対URIが必要、と。

そういうことだ。リクエストURIには他にも記述法があるが、かなり特殊なので省略する。さて、メソッド、リクエストURIの次はHTTPバージョンだな。これは**UAが使用できるHTTPのバージョンを示す**。

HTTPのバージョンってことは、0.9とか1.0、1.1を送るってことですか？

そうだな。記述方法は「HTTP/1.1」のように、HTTPスラッシュバージョンで記述する。クライアントが使用できるHTTPのバージョンを示すことで、使用する機能が決まるわけだ。

んん？　ってことは、HTTPのバージョンが違うと、使用できる機能が違うということですか？

そうだ。サーバ側は自身が使用できるバージョンと、要求のバーションを比較して、使用するバージョンを決定するわけだな。どちらも1.1なら1.1の機能を使うし、どちらかが1.0だったなら1.0の機能でやりとりする。

なるほど。

というわけで、リクエスト行については理解したかな？　これがリクエストでもっとも大事と言っていい部分なのでしっかり覚えておくように。ではまた次回。

いぇっさー。3分間HTTP&メールプロトコル基礎講座でした～♪

ネット君の今日のポイント

- **リクエスト行にはメソッド、リクエストURI、HTTPバージョンが書かれている。**
- **メソッドはリクエストの内容を決定している。**
- **リクエストURIはリソースの場所を指定している。**

第5回

HTTPレスポンス

●ステータスコード

さて、ネット君。HTTPメッセージは、リクエスト、レスポンスのどちらも、開始行、メッセージヘッダ、ボディから成り立っているという話はしたな（P17参照）。

はい。で、前回リクエストの開始行、リクエスト行の説明をしましたよね。リクエスト行はメソッド、リクエストURI、HTTPバージョンからなるって。

うむうむ。メソッドはリクエストの内容を、リクエストURIは要求するリソースを、HTTPバージョンはクライアントが使用しているHTTPのバージョンを示したわけだ。
で、今回はレスポンスの開始行、**ステータス行**の説明をしよう。**（*1）**

あれ？　リクエストの開始行の次だから、順番的に言えばメッセージヘッダの説明じゃないんですか？

メッセージヘッダはレスポンスにもあるから、後でまとめて、な。ともかく、ステータス行は**HTTPバージョン、ステータスコード、応答フレーズ**からなる。リクエスト行と同じく**空白文字で区切られる**。そして最後に改行があって、ステータス行は終了となる。

あー、空白で区切ることと、最後に改行があるのはリクエストと変わらないんですね。それで、最初がHTTPバージョン？

うむ。これは**サーバが使用できるHTTPバージョンを示す**。記述はリクエストと同じようにHTTPスラッシュバージョンで記述する。リクエストにもHTTPバージョンがあって、そちらはUAが使用しているHTTPのバージョンだったな？

（*1）ステータス行［States-Line］

そうでした。ってことは、UAが使ってるHTTPバージョンをリクエストで送り、サーバが使用しているHTTPのバージョンをレスポンスで送る？

そうだな。それにより、やりとりで使用されるHTTPのバージョンがわかるわけだ。そして、次がステータスコードと応答フレーズ。これは**レスポンスの意味**を説明するためのものだ。ステータスコードは3桁の数字で、どのようなレスポンスかということを示す値だ(**図5-1**)。

ふむふむ、ステータスコードの100の桁の番号で、大雑把なレスポンスの意味がわかるってことですか。

そういうことだ。そして、ステータスコードに対応したものが応答フレーズだ。これはステータスコードの意味を記述したものだ。

●代表的なステータスコード

さて、ステータスコードだが、40種類近くあり、さらに拡張も行われているのでさすがにすべて説明はできない。よって、ここでは代表的なステータスコードを説明していこう。まず、「200 OK」だ。これはもっとも一般的なステータスコードだろうな。「200 OK」は、**リクエストが正常であり、正しく受け入れられた**ことを示す。よって、レスポンスにはメソッドに応じたボディが付け加えられる。

つまり「OK」だから、「正常に応答した」って意味でいいんでしょうか？

そうだな。リクエストのリクエストURIのリソースに対し、メソッドで示した処理を正常に行った、ということだ。
次が「301 Moved Permanently」。
これは、リクエストURIで指定したリソースが「移動している」ことを示す。そして300番台のステータスコードだから、さらに次のリクエストを要求するステータスコードであるということがわかる。つまり「要求したリソースは移動したので、移動先のURIを要求してください」というレスポンスだ。

ははぁ、ずいぶんと親切なレスポンスなんですね。移動したから存在しないよ、って言うだけが普通かな、と思うのに、移動先までわざわざ教えてくれるなんて。

図5-1　ステータスコードと応答フレーズ

レスポンスの意味を説明する3桁の数値がステータスコード
それに対する説明文が応答フレーズ

HTTPレスポンスメッセージのステータス行

HTTPバージョン <空白> ステータスコード <空白> 応答フレーズ <改行>

ステータスコードの100の桁の意味

ステータスコード	意味	説明
1xx	肯定先行	正しいコマンドを受け付けて処理中である
2xx	肯定完了	正しいコマンドを受け付けて処理を完了した
3xx	肯定中間	正しいコマンドを受け付けて、次に別のコマンドを要求する
4xx	一時否定完了	誤ったコマンドを受け付けた。再送を望む
5xx	否定完了	サーバの状態により、コマンドの受け付けが不可能

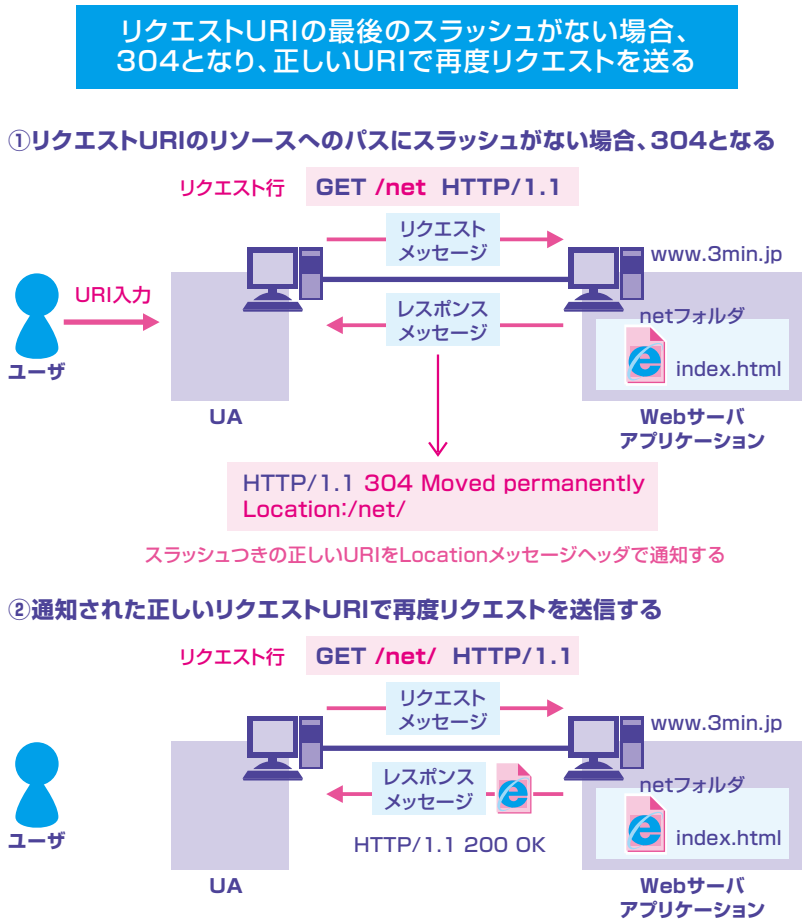
代表的なステータスコードと応答フレーズ

ステータスコード	応答フレーズ	説明
200	OK	正常にリクエストを受け付けた
202	Created	正常にリソースを作成した
205	Not Content	正常にリクエストを受け付けたが、応答するリソースなし
301	Movement Permanently	リソースは移動している
304	OK	リソースは更新されていない
400	Bad Request	開始行、メッセージヘッダの構文にミスがあり受け取れない
401	Unauthorized	リソースには認証が必要であり、認証されていない
403	Forbidden	コマンドは正常だが、サーバがその実行を拒否した
404	Not Found	リソースは存在しない
405	Method Not Allowed	そのメソッドの実行は許可されない
500	Internal Server Error	サーバ内部のエラー
501	Not Implemented	コマンドを実行する機能が存在しない
505	HTTP Version Not Supported	リクエストの実行に必要なHTTPバージョンに対応していない

いや、ネット君の言う通り、それが普通だ。通常は場所が変更されたリソースにリクエストが来たら「見つからない」と返す。では、この304 Moved Permanentlyは何に使われるかというと、**リクエストURIの最後のスラッシュがない場合**に使われる(図5-2)。

あ、前に説明してた「最後にスラッシュがない場合」の動作ですね（P12参照)。最後にスラッシュがないと、ファイル名とみなしてしまい、そのファイルがないと最後にスラッシュをくっつけるって言ってたけど、こういう動作をするんだ。

図5-2　304 Moved Permanently



そうだ。**最後のスラッシュがない場合、2回リクエストを送る**場合があるってことだ。正直言って、無駄でしかないとも言える。
次は、「401 Unauthorized」。認証失敗を示すエラーだ。

認証？　認証って言うと、ユーザ名とパスワードのアカウントを示して、「自分が正しいユーザであることを示す」んですよね。

そうだ、それによりページを見る権利があることを示すわけだな。WWWでは通常は認証は行わないが、会員制ページなどでは認証を行いたい場合がある。その場合に使用するのが401 Unauthorizedだ。
詳しくは「認証」のところで話そう（『3分間HTTP&メールプロトコル基礎講座』P97参照）。次は「404 Not Found」。これはちょくちょく見かけることがあるかもしれん。リクエストURIで指定したリソースが存在しないというステータスコードだ。

あー、もしかして「Webページが見つかりません」ってやつですか？

うむ。ブラウザによって表示が違う場合もあるが、ネット君が言っているそれだ。指定したリソースが存在しないというエラーだな。リクエストURIを間違えた、リソースが移動した・消されたなどで起きる。
そして、最後に説明するのが「500 Internal Server Error」だ。

サーバ内部（Internal）エラー？　それは見たことがないなぁ。500番台だから、サーバ側で実行できない？

このエラーは人によっては嫌というほど見るエラーなのだがな。ネット君の言う通り、リクエストは正常だが、Webサーバアプリケーション側の事情でエラーになった場合に送るステータスコードだ。一番よくある原因としては、CGIで使っているプログラムにミスがあって、プログラムが正常に動作しない場合だな**（図5-3）**。

CGIの動的ページを作るプログラムにミスがある、と。それって見る側からしたらどうしようもなくないですか？

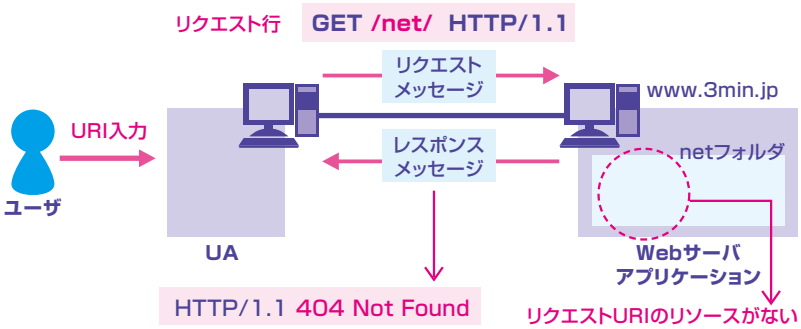
うむ、実はどうしようもない。そういう場合は、Webサイトを作った人に連絡するしかないな。さて、メジャーなステータスコードは説明した。今回はこれぐらいにしておこう。

あいあい。3分間HTTP&メールプロトコル基礎講座でした〜♪

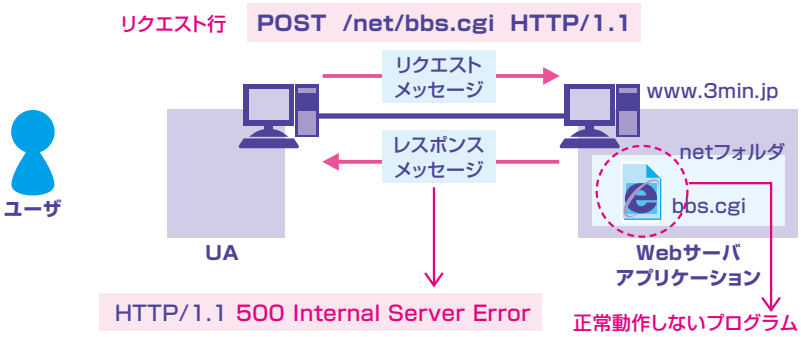
図5-3　400・500番台のステータスコード

ステータスコード400・500番台のレスポンスは
何らかのエラーであり、コードによって識別できる

404 Not Found



500 Internal Server Error



ネット君の今日のポイント

- ステータス行にはHTTPバージョン、ステータスコード、応答フレーズが書かれている。
- ステータスコードは100の桁でそのコードの意味がわかる。
- 応答フレーズはステータスコードの説明文。

第6回

メッセージヘッダの種類と役割

●メッセージヘッダの役割

さて、HTTPメッセージは、開始行、メッセージヘッダ、ボディから構成されているという話をした。それで、前々回、前回で開始行を説明したな。

はい。リクエストのリクエスト行、レスポンスのステータス行ですね。

うむ、そうだ。そしてその次に続くのが、メッセージヘッダだ。まずメッセージヘッダの役割だが、メッセージヘッダがない場合のことを考えてみよう。メッセージヘッダがない場合、開始行とボディだけになるな。ボディはやりとりするデータそのものだから、HTTPのやりとりの制御は開始行だけで受け持つことになる。

そうなりますね。で、開始行に書かれているものと言えば、リクエスト行だとメソッドとリクエストURIとHTTPバージョン。ステータス行だと、HTTPバージョンとステータスコード、応答フレーズ。

HTTPバージョンを抜くと、実質「要求するリソースとその扱い」をリクエストとして送り、「その結果」をレスポンスで送り返すことになる。さすがに制御情報としては少ないとは思わないかね？

う〜ん、少ないっていえばそうですけど、これだけでも大丈夫な気がしますけど？

まぁ、確かに最初に作られたHTTP0.9にはメッセージヘッダはなかったが、これだけでも動いていた。だが、HTTPで処理することが増えるにつれ、UAとサーバ間でもう少し情報をやりとりしたい、と考えたわけだ。そこで**HTTPのやりとりに情報を付加する役割でメッセージヘッダを付ける**ようになった、というわけだ。

ん〜、どんな情報を付けるようになったんですか？

たとえば、リソースの更新日、データ量、使用しているメディア、UAの種類、認証、中継などだな。確かにこれらの情報がなくても、純粋なリソースのやりとりには問題ないとはいえる。だが、HTTPが拡張され、さまざまなデータを運んだり、細かいやりとりの制御を行ったりするにはこれらのメッセージヘッダが必要になった、ということだ。

ははぁ。なんかかっちょよくいえば、WWWがインターネットの主役になったのは、HTTPが拡張されてさまざまなことができるようになったからですよね。その背景としてメッセージヘッダの導入があった、って感じですか？

なかなかうまいこと言うな。ともかく、メッセージヘッダを付けることにより、HTTPのやりとりで様々なことができるようになった、と覚えておくといい。どのようなことができて、どのようなメッセージヘッダが使われるか、という点についてはまた先で説明する（P37参照）。

●メッセージヘッダの種類

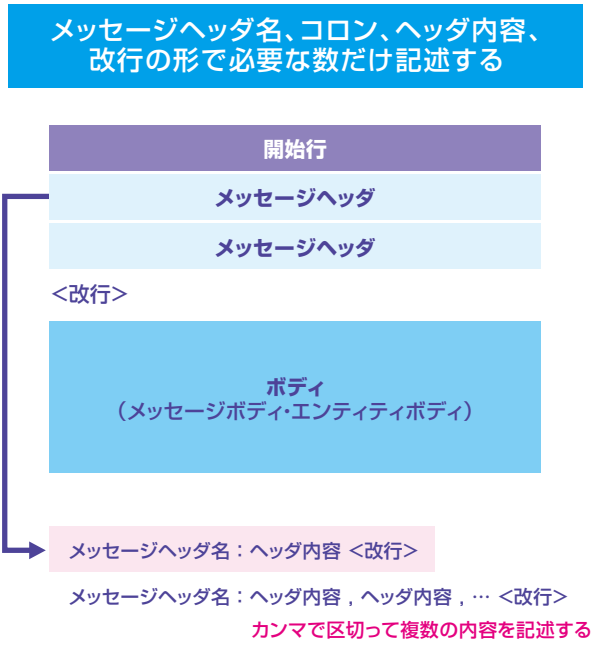
さて、では実際にメッセージヘッダの説明をしていく。まずメッセージヘッダの記述からだな。メッセージヘッダは**「ヘッダ名：内容 改行」**で1つのメッセージヘッダを構成する。改行がメッセージヘッダの区切りであることを忘れないように。このメッセージヘッダを必要な数だけ開始行の後ろに入れ、**最後にもう一度改行コードを入れる**。この改行がメッセージヘッダ群の終了を示し、その後ろがボディであることがわかる**（図6-1）**。

ふむふむ。メッセージヘッダは、必要な数だけ入れるんですね。開始行、メッセージヘッダ、メッセージヘッダ、……、ボディって形ですね。で、区切りに改行を入れる。

そういうことだ。さて、このメッセージヘッダはだいたい50種類弱ぐらい存在する。さすがにこれを全部説明はできないので、代表的なものだけ説明していくが、まずこのメッセージヘッダの分類について話そう。メッセージヘッダの分類には2種類ある。まず、**中継された場合の扱い**による分類だ。

中継？　そう言えば前にプロキシサーバが中継するとかなんとか話していましたよね。絶対URIと相対URIのところで（P23参照）。

図6-1　メッセージヘッダの記述



うむ、よく覚えていた。プロキシサーバは、HTTPのリクエストとレスポンスを中継するサーバだ。詳しくは先の回で説明するが（『3分間HTTP&メールプロトコル基礎講座』P78参照）、このプロキシサーバで中継される場合に、メッセージヘッダをどう扱うか、という話だ。これはメッセージヘッダによって異なり、**エンドツーエンド**か**ホップバイホップ**のどちらかになる**（*1）**（図6-2）。

ん〜っと、中継する場合も変わらないヘッダと、中継されたら変更されたり、なくなったりするヘッダがある、ってことですね。それで、もう1つの分類は？

もう1つは、**メッセージヘッダの内容**による分類だ。これには**一般、要求、応答、エンティティ**の4種類が存在する。まず「一般」メッセージヘッダだが、これはリクエスト・レスポンスの両方で使用されるメッセージヘッダだ。代表例としてはキープアライブで使用するConnectionなどがあるな。

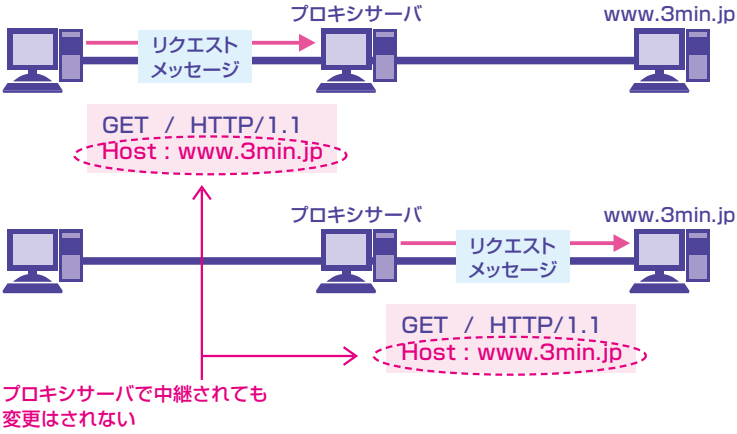
（*1）エンドツーエンドとホップバイホップ［End-to-End］［Hop-by-Hop］

図6-2　エンドツーエンドとホップバイホップ

中継されても変更されないヘッダがエンドツーエンド、変更されるヘッダがホップバイホップ

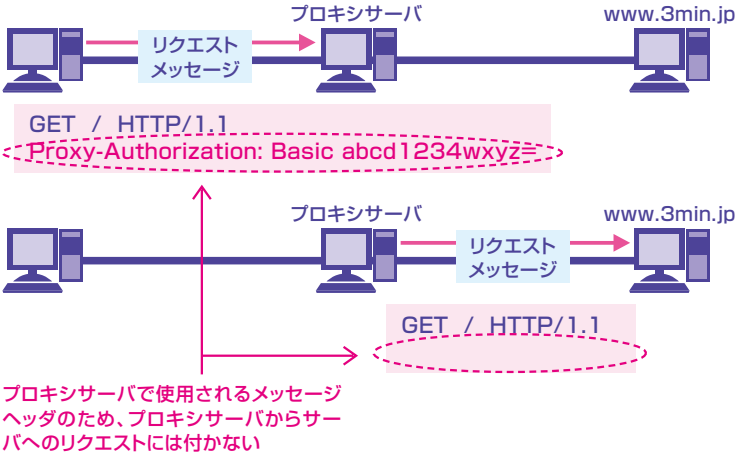
エンドツーエンドのメッセージヘッダは、中継サーバがあったとしても変更されない

Hostメッセージヘッダはエンドツーエンドのメッセージヘッダ



ホップバイホップのメッセージヘッダは、中継サーバで変更されたり、中継サーバだけで使用されたりする

Proxy-Authorizationメッセージヘッダはホップバイホップのメッセージヘッダ



ふむー。そうすると、「要求」メッセージヘッダはリクエストだけで使用し、「応答」メッセージヘッダはレスポンスだけで使用するということですか？

そのとおり。要求メッセージヘッダはUAの情報や、リクエストの内容の補助情報などを送る。一方の応答メッセージヘッダはサーバの情報、認証、リソースの補助情報などだ。「エンティティ」メッセージヘッダは、ボディにあるリソースの長さや使われているメディアなどだ(図6-3)。

うぇー、かなりいっぱいありますね。これ全部覚えなきゃだめなんですか？

いや、この中でもさらに代表的ないくつかのメッセージヘッダを説明するので、それを覚えてくれればいい。あと、下の表に挙げたメッセージヘッダはRFCで定義されているものだけだ。UAやWebサーバアプリケーションを作るベンダーが、独自にメッセージヘッダを定義して使用してもいい。ただし、もちろんUAやサーバの両方でその独自ヘッダを使用できる場合じゃないと使用しても意味がないがな。(*2)

図6-3　メッセージヘッダの種類

メッセージヘッダの内容によって、
一般、要求、応答、エンティティがある

一般	要求		応答	エンティティ
Cache-Control	Accept	If-Modified-Since	Accept-Ranges	Allow
Connection	Accpet-Charset	If-Range	Age	Content-Encoding
Data	Accept-Encoding	If-Unmodified-Sicne	ETag	Content-Language
Pragma	Accept-Language	Max-Forwards	Location	Content-Length
Trailer	Authorization	Proxy-Authorization	Proxy-Authenticate	Content-Location
Transfer-Encoding	Expect	Range	Retry-After	Content-MD5
Upgrade	From	Referer	Server	Content-Range
Via	Host	TE	Vary	Content-Type
Warning	If-Match	User-Agent	WWW-Authenticate	Expires
	IF-None-Match			Last-Modified

へぇ、そういう独自に作られたヘッダって存在するんですか？

もちろんある。たとえば、**ネットスケープ社**が定義したCookieヘッダなどがそうだ。これは独自メッセージヘッダなのに事実上の標準として利用されている。詳しくはまた後で話そう（『3分間HTTP&メールプロトコル基礎講座』当社刊 P69参照）。ともかく、メッセージヘッダの役割と種類は理解したかね？ **(*3)**

HTTPのやりとりに付加する情報で、これによりHTTPでいろいろなやりとりが可能になったんですよね。で、中継や内容によって分類されるよ、と。

よしよし。次回は代表的なメッセージヘッダについて説明する。ではまた次回。

了解っす。3分間HTTP&メールプロトコル基礎講座でした〜♪

(*2) RFC【Request For Comment】 IETF［the Internet Engineering Task Force］が発行する技術文書で事実上のインターネット標準となる。

(*3) ネットスケープ社 正確にはネットスケープコミュニケーションズ社［Netscape Communications Corporation］。ブラウザのNetscapeNavigatorなどを開発した。現在は買収されて存在しない。

ネット君の今日のポイント

- **リクエスト・レスポンスで開始行の後ろにメッセージヘッダを必要数入れる。**
- **メッセージヘッダはHTTPのやりとりに付加する情報。**
- **中継の際に変更されるホップバイホップと変更されないエンドツーエンドがある。**
- **その内容に応じて、一般、要求、応答、エンティティに分類される。**

○月○日 by ネット君

第7回

メッセージヘッダ

●代表的なメッセージヘッダ

前回、HTTPメッセージのメッセージヘッダについて説明したな。HTTPのやりとりに付加する情報として、メッセージヘッダを付ける、という話だった。

でした。開始行のうしろに付けるんですよね。開始行、メッセージヘッダ×n個、ボディって感じで。

そういうことだ。では今回は、代表的なメッセージヘッダを説明していこう。先の回で細かく説明するメッセージヘッダは、ここでは省いて説明する。
まず、**HTTP1.1のリクエストで唯一必須のメッセージヘッダ**である、**Host**だ。Hostは要求ヘッダのエンドツーエンドメッセージヘッダだ。
Hostはリクエストを送るサーバのドメイン名とポート番号を示す値だ。なお、ポート番号がWell-Knownの80番の場合はポート番号を省略できる。また、リクエストのリクエストURIが相対パスの場合、このHostに書かれているサーバのリソースと判断される。

ん、んんん？　サーバのドメイン名？　でも博士、HTTPリクエストって中継される場合を除けば、そのサーバに送られるんですよね？　たとえば、http://gihyo.jp/ならgihyo.jpに。リクエストを送るサーバにリクエストを送っているのに、どうしてさらにそのサーバ名をリクエストに含めるんですか？

それはもっともな疑問だ。これはHTTP1.1から拡張された**バーチャルホスト機能**のために必要なのだ。バーチャルホスト機能については先の回で説明する(『3分間HTTP&メールプロトコル基礎講座』P115参照)。逆に言えば、HTTP1.1でなくHTTP1.0を使うならばこのHostは必要ない。
さて次のメッセージヘッダだが、「User-Agent」。要求ヘッダのエンドツーエンドメッセージヘッダだ。これはUAの種類をサーバに通知するために使用する。

へー、たとえば、Internet Explorerだったり、FireFoxだったり、Opera、Chromeだったり、そういうUAの種類を伝えるってことですね。これ、何の役に立つんです？

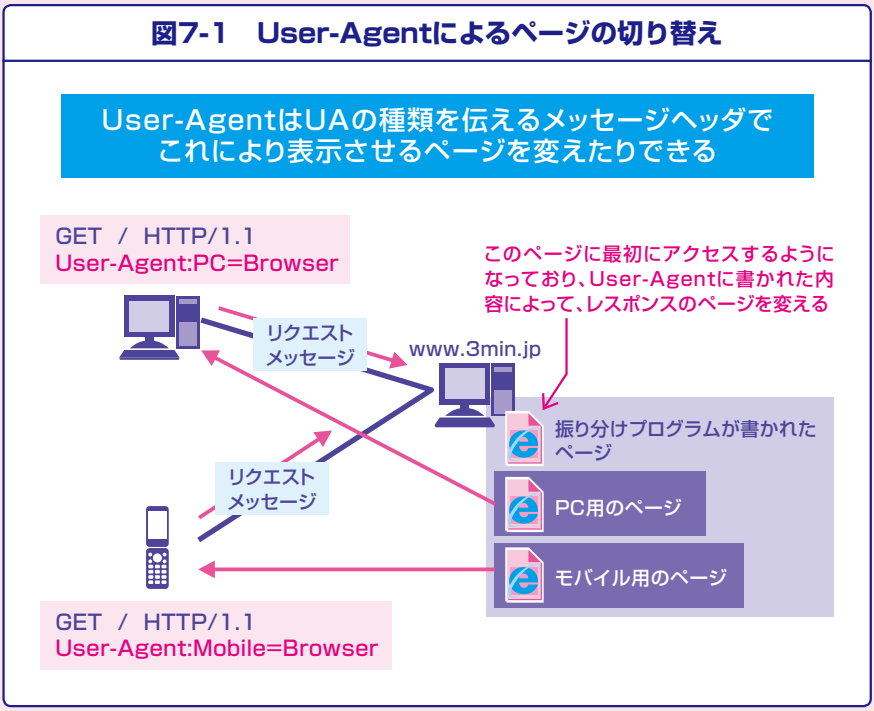
リクエストしてきたUAをサーバ側で判断し、表示させるページを変える時などに使用する。たとえば、UAの種類としてはネット君が例に挙げたパソコン用のブラウザもあるし、携帯電話などモバイル用のUAもある。パソコンでのアクセスと携帯電話でのアクセスでページを変えたい場合、とかだな(**図7-1**)。

あー、URIが同じWebサイトでも、パソコンで見た場合とケータイで見た場合でページが違うのって、User-Agentで判別してたんですか。知らなかった。

他にも携帯電話の場合、そのキャリアが使用するIPアドレスが決まっているから、送信元IPアドレスを見て切り替える方式もある。
さて、次のメッセージヘッダにいこう。「Referer」。要求ヘッダのエンドツーエンドメッセージヘッダだ。これはリクエストURIの参照元のページのURIだ。

参照元？　リクエストURIを参照した元……、え〜っと、リクエストURIは見たいページだから、それを参照した元のURI ？

図7-1　User-Agentによるページの切り替え



リクエストURIにハイパーリンクしたページのURIってことだな。つまり簡単に言えば「リンク元」だ。ちなみにホントはReferrerなのだが、なぜかRefererになっている。Refererによって、そのページへどこからハイパーリンクされてきたかという情報を集めることができる(**図7-2**)。

あれですか？　Webサイトの「アクセス解析ツール」とかで確認できる「リンク元」「アクセス元」ってやつですか。そうか、あのツールはこのRefererから情報を得てるんだ。

うむ、実はそうだ。ただしRefererはリンクされた場合にのみ付けられるので、直接URIを入力した場合や、ショートカット（ブックマーク）から選んだ場合はRefererは付けられない。
次は「Content-Length」。これはエンティティヘッダのエンドツーエンドメッセージヘッダだ。エンティティヘッダなので、リクエスト、レスポンスどちらでも使われる。**ボディのデータ長を示す**メッセージヘッダだな。

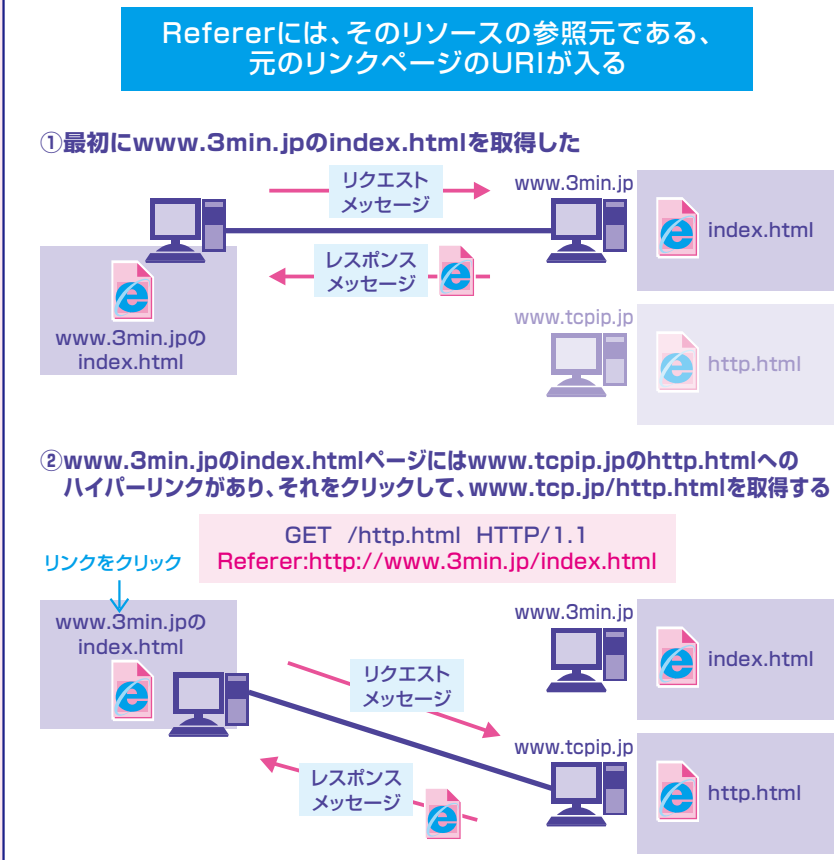
リクエストとレスポンスのどちらでも使う、ボディのデータ長。レスポンスはリクエストURIのリソースの長さですけど、リクエストの場合はどうなるんです？

リクエストのメソッドがPOSTの場合、「投稿」するデータがボディに入っている。その長さを示している。
次は2つまとめて説明しよう。要求ヘッダの「Accept」と応答ヘッダの「Content-type」だ。これはどちらもエンドツーエンドメッセージヘッダだ。この2つは**リソースのメディアタイプを示す**メッセージヘッダだ。Acceptは要求するリソースの、Content-typeは応答するリソースのメディアタイプを示す。

メディアタイプってなんですか？　媒体タイプ……なんの媒体ですか？

メディアタイプは**リソースの中身が何であるかを示す**ものだ。たとえば、テキストファイル、画像ファイル、音声ファイル、動画ファイルなどなど、リソースがどのタイプであるかを指定する。メディアタイプは「タイプ/サブタイプ」の形で記述される。
UAはAcceptで利用可能なメディアタイプをサーバに通知し、サーバはレスポンスに入れたリソースのメディアタイプをContent-typeで指定する、という形になる。

図7-2　Refererによるリンク元の確認



へー。じゃあ、たとえば、UAがAcceptで送ったメディアタイプと、サーバがレスポンスできるリソースのメディアタイプが違ったらどうするんですか？

その場合は、406 Not Acceptableというステータスコードで、利用できないことを通知する。あぁ、そうそう、**Content-typeはレスポンスでの必須ヘッダ**になる。ただし、ボディにリソースが入っていないならば、付けなくてもいい(**図7-3**)。

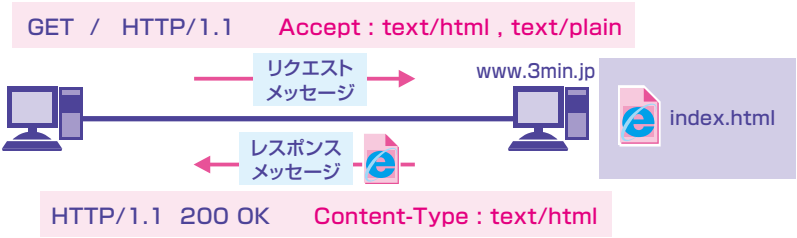
図7-3　メディアタイプ

使用できるリソースの種類をAccept、Content-Typeで指定する

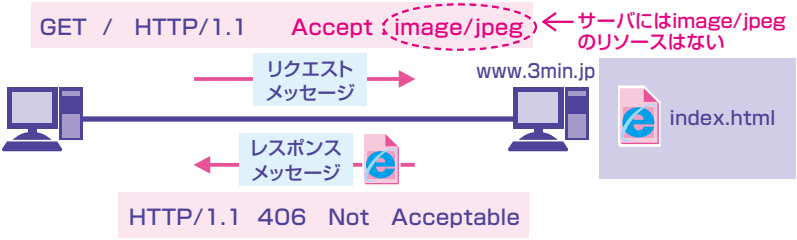
代表的なメディアタイプ

タイプ	タイプの意味	種類
*(アスタリスク)	すべて	*/*(すべてのメディアタイプ)
text	テキストファイル	text/html(HTMLファイル)、text/plain(テキストファイル)など
image	画像ファイル	image/jpeg(JPEGファイル)、image/png(PNGファイル)など
audio	音声ファイル	audio/midi(MIDIファイル)、audio/x-wav(WAVファイル)など
video	動画ファイル	video/mpeg(MPEGファイル)、video/x-msvideo(AVIファイル)
application	アプリケーションで使われるファイル	application/zip(ZIPファイル)、application/pdf(PDFファイル)、application/xhtml+xml(XHTMLファイル)など

UAは処理可能なメディアタイプをAcceptメッセージヘッダで通知し、使用するサーバはそれに応じてメディアタイプをContent-Typeに入れてレスポンスを返す



Acceptで通知されたメディアタイプがサーバに存在しない場合などでは、406　Not Acceptableを返す



ふむふむ、つまりリクエストには「Host」、レスポンスには「Content-type」が必須ってことでいいんでしょうか。

そうだな、それでいい。さて、ほかにもメッセージヘッダはあるが、今回はこれぐらいにしておこう。

はいなー。3分間HTTP&メールプロトコル基礎講座でした〜♪

ネット君の今日のポイント

- **Hostヘッダはリクエストに必須。**
- **Content-typeはレスポンスには必須で、応答するリソースのメディアタイプを示す。**

○月○日 ネット君

第8回

GETメソッド

●キャッシュと条件付きGET

ここまでのところで、HTTPメッセージの説明をしてきたわけだ。HTTPメッセージは、開始行、メッセージヘッダ、ボディから構成される。開始行、メッセージヘッダの中身と役割についてはすでに説明した。今回は、もう一度リクエストの開始行について説明する。

リクエストの開始行？　リクエスト行ですか？　えっと、メソッド、リクエストURI、HTTPバージョンでしたよね。

うむ。その中で、メソッドは簡単にしか説明していなかった。なので、もうちょっと細かく説明しよう。まず、基本となるGETだ。

GETは、リクエストURIで指定されたリソースの転送を要求するためのメソッドでしたよね。ぶっちゃければ、そのリソースを取得する？

その通り。リクエストURIで指定されたリソース、つまりファイルだな。HTMLファイル、画像ファイルなどのファイルをサーバから転送してもらう、つまり取得するというメソッドだ。まぁ、GETを単純に説明するとこれだけなんだが……。

なんだが？　含みを持たせますね、もちろんまだあるんですよね？

うむ。その前に、以前話したことを思い出してもらおう。Webページを「見る」とは、「ファイルをダウンロード」して見ることだ、という話をしたと思う（P14参照）。UAを持つ機器、つまりクライアントは、このファイルを保存しておくことができる。これを**キャッシュ**という。

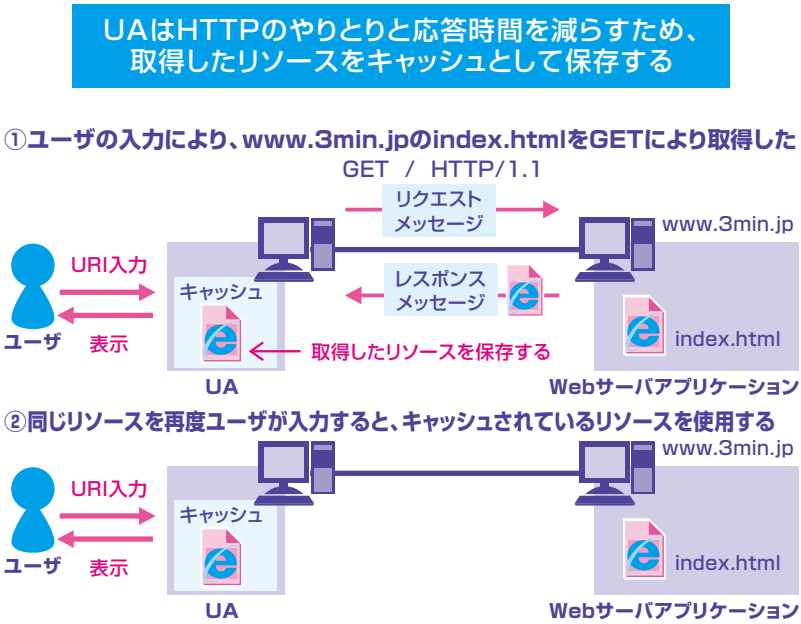
はいはい、HTTPの動作のところでそんな話が出てましたよね。んー、キャッシュとして「保存」しておくんですか？　見終わったら消してしまえばいいのに。

うむ。純粋に見るだけなら、確かに見終わったら消せばいい。だが、キャッシュとして保存しておくことにより、**同じページをもう一度閲覧する際にはキャッシュされているファイルを見る**ことができる。それにより、もう一度ファイルを取りに行く手間が省けるわけだな（**図8-1**）。

なるほど、一度キャッシュとして保存してあるページを、もう一度見る場合には、キャッシュから見る、と。確かにそうすれば同じページをまたサーバからGETする手間が省けますね。

ただし、キャッシュにも問題がある。それは「同じページを見る場合、サーバにあるページが更新されているいないにかかわらず、キャッシュのページを見てしまう場合がある」という点だ。つまり、サーバにあるページが更新され新しくなっていても、キャッシュに残っている古いページを見てしまうことがある、ということだな。

図8-1　キャッシュ



ははぁ、それは確かに困りますね。下手をするといつまでたっても古い情報の載っているページを見てしまって勘違いしたりするとか、ありえそうですよね。

うむ。そこでこれを解決する方法が用意されている。まず1つ目、**そのページのキャッシュの有無を制御するCache-Controlメッセージヘッダ**によって、クライアント側のキャッシュを制御する方法だ。これは特に頻繁に更新される、たとえばCGIを使った掲示板のような動的ページなどで使われる方法だ。

制御ってどんな制御ですか？

「このページはキャッシュしてはいけません」「このページはxxxxのxx時までキャッシュされます」、のようにキャッシュさせないか、キャッシュの有効期限を切ってしまうという制御だ。

なるほどなるほど。クライアント側で「キャッシュをどのように持つか」をサーバ側がCache-Controlを使って決めることができるんですね。

そしてもう1つの方法が**条件付きGET**だ。これには**If-Modified-Sinceメッセージヘッダ**を使う(図8-2)。

ははぁ、「もし今のキャッシュの更新日付よりも新しいならば」GETする、ってことで「条件付きGET」というわけですね。

そういうことだ。キャッシュを使うことにより転送量が減るという利点があるが、その欠点もちゃんと対策しているってことだな。

●部分的GET

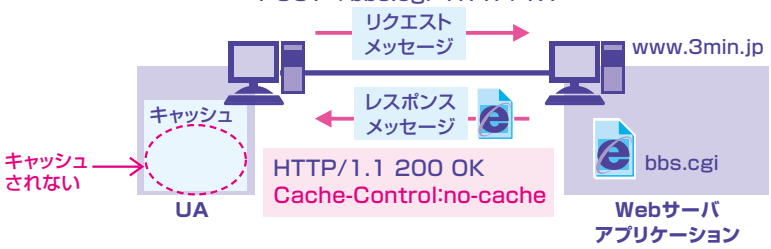
もう1つ、GETの中でも特徴的なGETを説明しよう。それは**部分的GET**と呼ばれるGETだ。
たとえば、大きなファイルなどをGETで取得するとして、ちゃんと最後まで取得できればよいが、途中でエラーが発生してしまったとする。そうするとHTTPの接続が切れてしまい、もう一度やり直しになる。

ん〜、とくに大きいファイルだと厳しいですね、やり直しって。

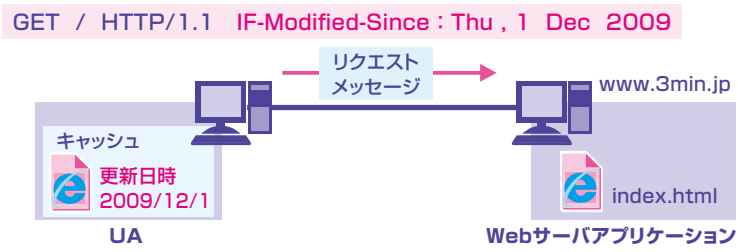
図8-2　Cache-ControlとIf-Modified-Since

Cache-Controlによりサーバ側からキャッシュをコントロールできる。
If-Modified-Sinceにより更新していた場合のみ取得する

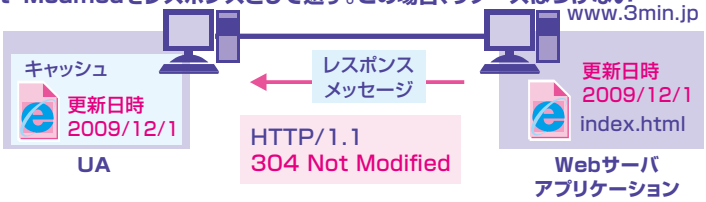
掲示板のような頻繁に更新されるリソースの場合、UAにキャッシュさせないようにするため、Cache-Controlで制御することができる



**条件付きGETを使う場合、リクエストにIf-Modified-Sinceメッセージヘッダを入れる
If-Modified-Sinceにはキャッシュされているリソースの更新日時を入れておく**



**サーバのリソースがIf-Modified-Since以降更新されていなければ、
304 Not Modifiedをレスポンスとして返す。この場合、リソースはつけない**

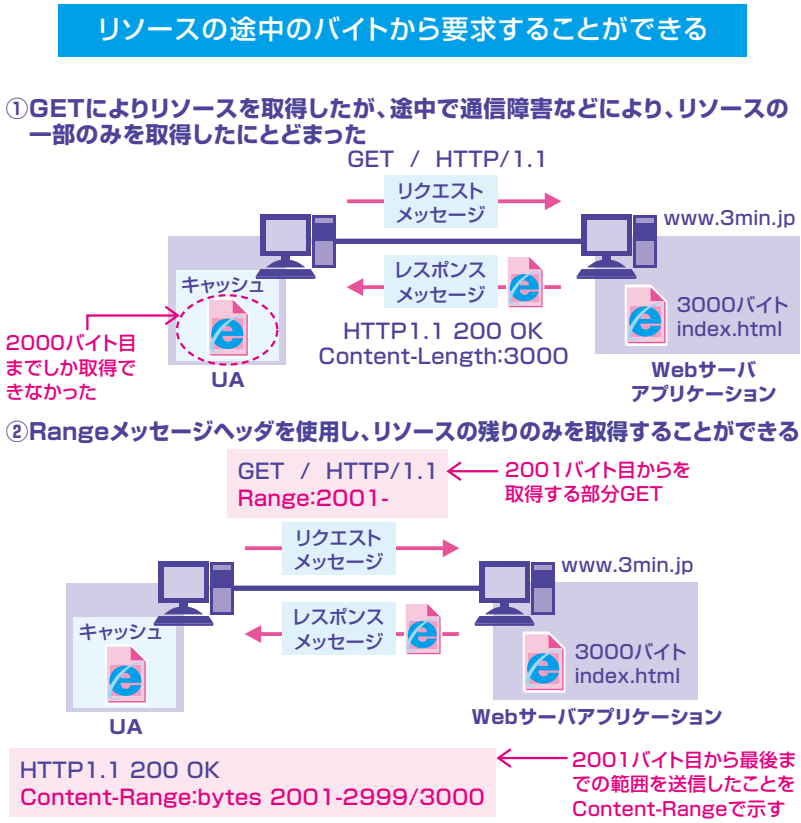


**サーバのリソースがIf-Modified-Since以降更新されていた場合、
200 OKとともにリソースをボディにいれて返す**

そういう場合に使うのが、部分的GETだ。ファイル全体のサイズはContents-Lengthでサーバ側から通知されているから、途中でエラーなどが発生した場合、不足分のサイズがわかる。そこで**Rangeメッセージヘッダを使い、途中からファイルを取得する**ことができる(**図8-3**)。

ははぁ、なるほど。たとえばデータが1000バイトだったとして、500バイト受信済みでエラーにより止まってしまったら、次は501バイトからって要求できるんですね。

図8-3　部分的GET



そういうことだ。UAでこの部分的GETができる場合、「レジューム機能」付きとか言われているな。レジューム、つまり「再開」ができるってことだ。他にも「ダウンロード支援ツール」と呼ばれるWWWでファイルのダウンロードを高速・簡単に使えるソフトウェアでは、この部分的GETをうまく使っているものもある。つまり、最初から全部をGETするのではなく、ファイルを分割して、複数の部分的GETを同時に行い、スピードアップを行う形だ。(*1)

んんっと、たとえば1000バイトのデータで、0バイト目から、250バイト目から、500バイト目から、750バイト目から、っていう4つの要求を同時に出すってことですか？

うむ、それだ。なかなかうまく考えているよな。そうすればかなりスピードアップになるからな。ただ、接続数が増えるからサーバ側には嫌われているけどな。

まぁ、確かにサーバ側から見れば1つのアクセスですむのに、4倍のアクセスがきてることになるから、嫌がられるのもしょうがない、かな？

うむ、そういうことだ。では今回はここまで。また次回。

はい。3分間HTTP&メールプロトコル基礎講座でした〜♪

(*1) レジューム [Resume]

ネット君の今日のポイント

- リソースの取得を行うメソッドがGET。
- クライアントのキャッシュを制御するためにCache-Controlを使う。
- 条件付きGETを使うことで、更新されたファイルを取得できる。
- 部分的GETを使うことで、必要な範囲のデータだけを取得できる。

○月○日 ネット君

第9回

HEAD、POST

○月○日 ○曜△△

●HEAD

さて、前回はもっとも基本的なメソッドであるGETの説明をした。「リクエストURIで指定されたリソースの転送を要求する」というメソッドだな。

はい、でした。それで、「条件付きGET」や「部分的GET」っていうテクニックもありましたよね。

んー、まぁ、テクニックというか、なんというか。
それはともかく、今回はその他の代表的なメソッドを説明していこう。まず、HEADだな。HEADはGETと同じ動作をするが、**HEADに対するレスポンスにはボディがない**のが特徴だ。

GETと同じってことは、「リクエストURIで指定されたリソースの転送を要求」と同じってことですよね。でも、レスポンスにボディがないってことは、転送するリソースがないってことだから、意味ないんじゃないですか？

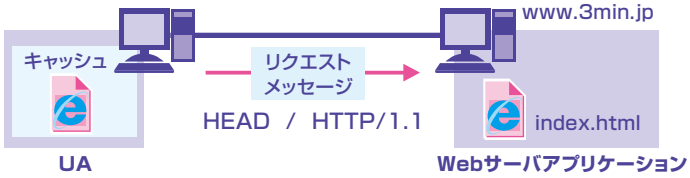
まぁ、確かにそう思えるのも無理はない。ただ、ボディがないだけで、ステータス行やメッセージヘッダはちゃんとレスポンスに含まれる。つまり、HEADは**ステータス行やメッセージヘッダの確認**を行うためのメソッドってことだな（**図9-1**）。
HEADを使うと、リクエストURIに指定されたリソースに対し、どのようなステータス行やメッセージヘッダが返ってくるかを確かめることができる。たとえばLast-Modifiedメッセージヘッダがある。これはリソースの最終更新日を含むエンティティヘッダだが、これがあると何がわかる？

そのリソース、つまりファイルの最終更新日がわかるんですよね？　ってことは、あー、そのファイルが更新されたかどうかを確認できる？

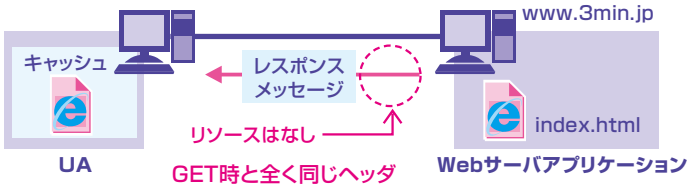
図9-1　HEADによる更新の確認

HEADではリソースが返ってこないこと以外はGETと
全く同じため、更新の確認や接続のテストなどに使用される

①HEADメソッドによりリソースを要求すると



②リソースがない以外、GETと全く同じ開始行、メッセージヘッダが返ってくる




```
HTTP/1.1  200  OK
Content-Length:2000
Content-type:text/html
Last-Modified: Thu, 1  Dec 2009
```

そうだ。クライアントが現時点で保存しているファイルよりも、新しいファイルがサーバにあるかどうか、レスポンスのLast-Modifiedメッセージヘッダを見れば確認できる。よって、これをHEADで取得する。確認だけしたいので、GETではなくてHEADを使うわけだな。

でも博士。前回の条件付きGETを使えばいいじゃないですか？　そうすれば更新されていれば新しいファイルを転送する、更新されていなければ転送されない。まるく収まりますよ？

確かにそうだ。だが、たとえば「更新の確認だけを行う」ツールがある。「巡回エージェント」と呼ばれるが、そういうツールはこのHEADを使うわけだ。実際に、更新があった場合にそれを取得するか否かはユーザの役割で、このツールは確認だけを行う。こういう時はHEADのほうが使いやすい。

あー、なるほど。そういうツールならば確かにHEADのほうがいいかもしれませんね。

●POST

さて、次のメソッドを説明しよう。POSTだ。以前は「投稿」を行うメソッドと説明したな（P22参照）。より正確に言えば、**ボディを送信しリクエストURIに書かれたリソースに処理させることを要求する**メソッドだ。

リソースに処理させることを要求するメソッド。受け取ったサーバはどうするんですか？

それはそのリソース次第だな。掲示板なら受け取ったデータを追加して、新たな掲示板の画面を生成するだろう。GETとの違いは**POSTはリソースの取得が目的ではない**というところだな。よって、レスポンスには必ずしもボディが必要とは限らない。

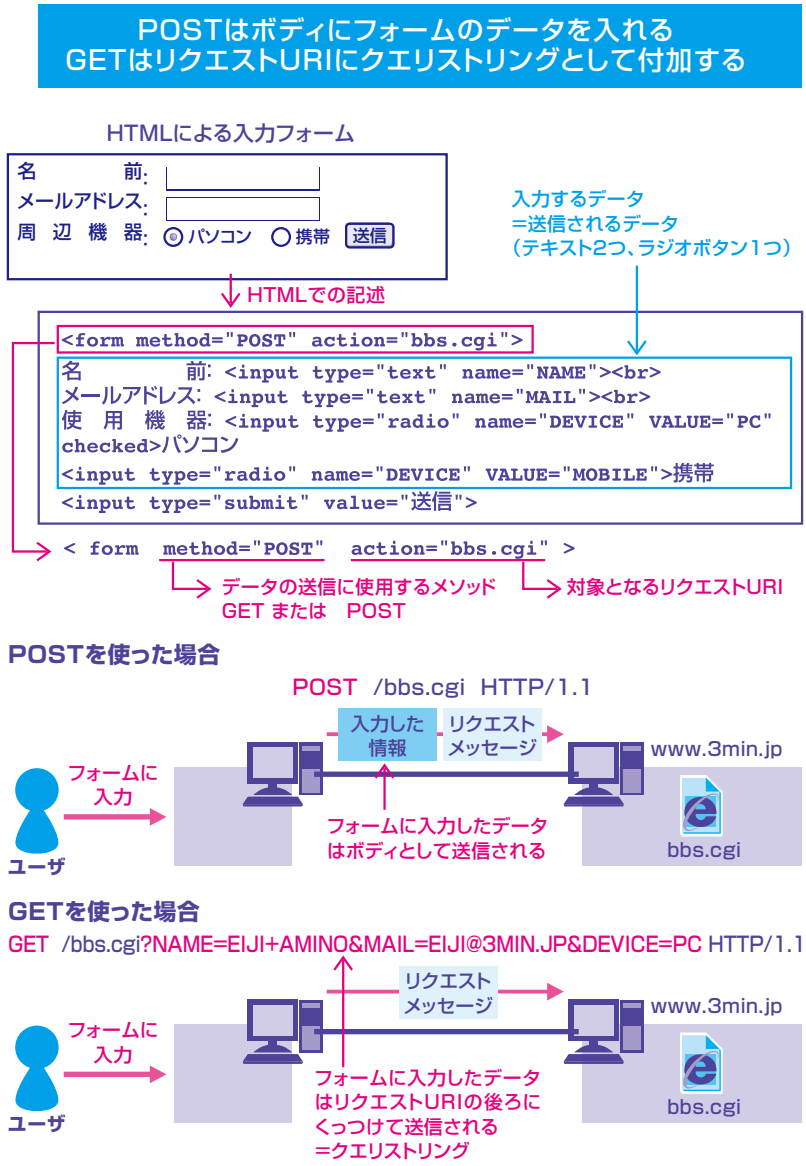
そうなんですか？　でも、掲示板とかだと送った「書き込みデータ」が反映された新しいファイルが返ってくるんじゃないですか？

まぁ、普通はリクエストURIに指定されたリソースが、受け取ったデータを処理した結果をボディに入れてレスポンスとして返す場合が多い。
あとは、そうだな。HTMLではGETとPOSTをどのように使い分けているか、ということを説明しておこう。HTMLで、ユーザが入力するための「ボタン」や「テキストフィールド」などの総称を「フォーム」と呼ぶ。このフォームの中に、ユーザが入力したデータを送信するURIを決定する項目と、その際にGETを使うかPOSTを使うか決める項目がある。

あれ？　データを送るならPOSTじゃないんですか？

いや、GETでもできる。その場合は、リクエストURIの中にデータを含めるのだ。**クエリストリング**と呼ばれる方法だ**（*1）**（図9-2）。
この方法だと、送信するデータがリクエストURIに含まれてしまうので、セキュリティ的にもいろいろと問題がある。たとえば、GETにクエリストリングを付けて送信し、その結果の画面を受け取ったとする。そしてそのページから他のページにリンクがあり、それをクリックした場合、Refererに「元ページのURI」としてクエリストリング付きのURIが残ってしまう。

図9-2　GETとPOSTの違い



ん〜、ダメじゃないですか。POSTならそういうことないですよね？　だったらPOST一択でいいような気がしますけど。

そう考えるのも無理はない。だが、違う側面から考えてみよう。クエリストリングにデータがある、ということは「そのクエリストリング（データ）付きのURI」を使えば誰でも同じデータを送信できる、ということになるよな。

んっと、そうなりますね。クエリストリング付きのURIがあれば、誰でも同じデータを送信できて、誰でも同じ結果を取得することができます。

そうだ。送ったデータによって、表示される画面が変更するページがあるとしよう。この変更されたページを、誰かほかの人にも伝えたいと思ったとする。この場合、そのページに同じデータを送ることが必要だ。そこで、GETによるクエリストリング付きURIを使う。このクエリストリング付きURIをその人に教えれば、その人は同じ変更されたページを取得できる。たとえば、検索エンジンの検索結果をほかの人に伝えたい、などだな（**図9-3**）。

あぁ、なるほど。確かにその方法なら、検索エンジンの検索結果を教えることができますね。

うむ、これをできるようにするため、ほとんどの検索エンジンはクエリストリングでデータを取得している。POSTでは、このようなことはできないからな。さて、だいたいWWWの、とくにHTTPの基本は説明し終わったわけだが、どうだ？

開始行とメッセージヘッダとボディですね！！

それだけだと、HTTPメッセージの説明だけみたいじゃないか。まぁ、確かにその説明を中心に行ってきたわけだが。HTTPは基本的なところはそれほど難しくはない。ネット君の言うとおり、開始行とメッセージヘッダの内容が中心で、それによりどのようにリソースをやりとりしているか、というだけだからな。

はい。これで僕もHTTPマスターですね、完璧です。

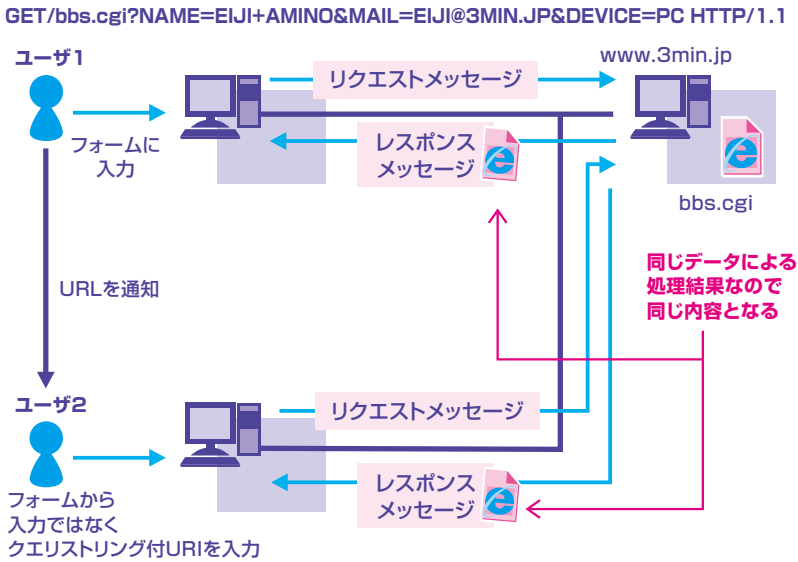
（*1）クエリストリング【Query-String】　リクエストURIのうしろにつなげて記入される投稿データ。Stringは文字列の意味。

……基本的な、と言ったはずだがな？ 『3分間HTTP&メールプロトコル基礎講座』（当社刊）のほうでは、これらのHTTPの基礎を踏まえたうえで、HTTPの応用的な話をしていこう。WWWは現在のインターネットの主役。できることも、考えなければいけないことも多くある。調子に乗るのは、もっとあとにしておけ。では今回はここまで。

了解です……。3分間HTTP&メールプロトコル基礎講座でした～♪

図9-3　クエリストリングを伝える

クエリストリング付きのURIを伝えることにより
同じ結果を入手することができる

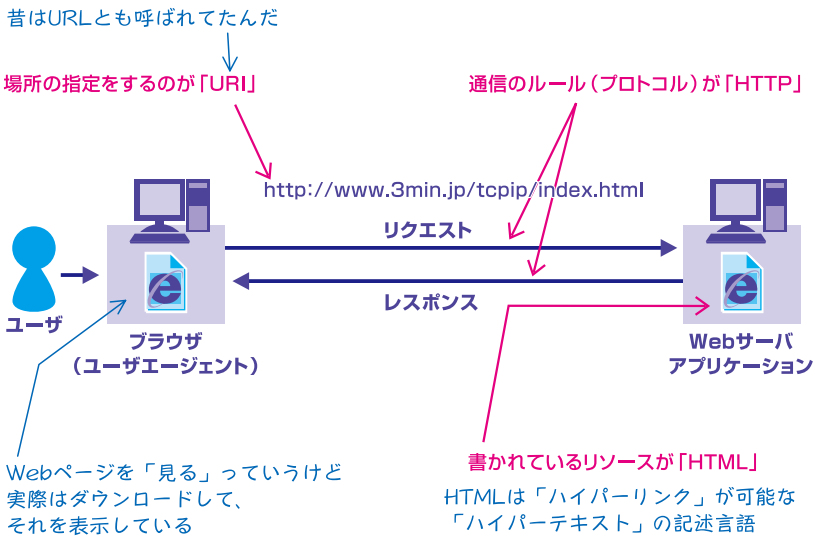


ネット君の今日のポイント

- HEADはレスポンスの開始行とメッセージヘッダのみを取得する。
- POSTはUAからデータを送信し、リクエストURIのリソースに処理させる。

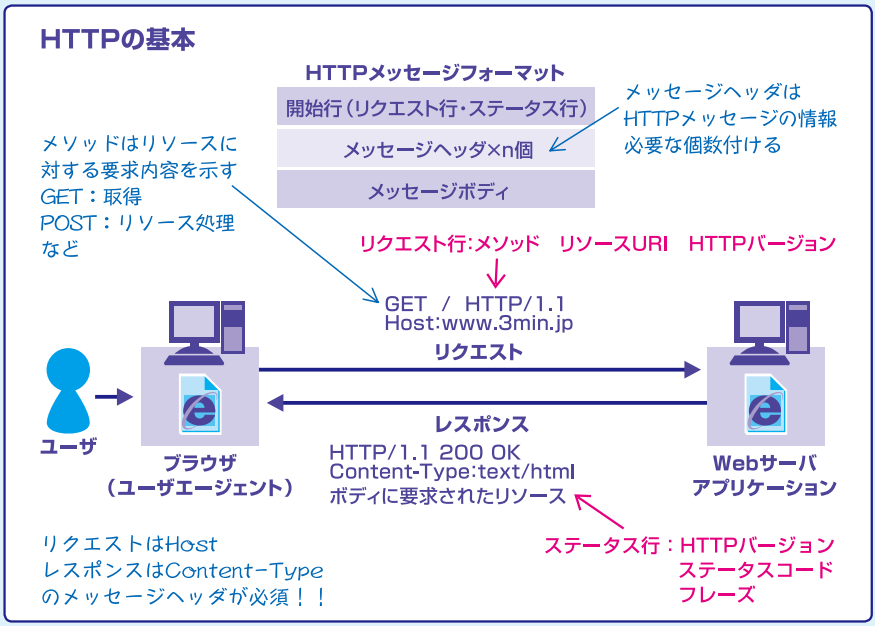
ネット君のまとめノート その①

WWW＝URIによって指定された場所にあるHTMLによって記述されたリソースをHTTPによって取得することによる「広域情報共有システム」



- **WWW**
  - ハイパーリンクによって結びつけられた広域情報共有システム
- **3つの基幹技術から成る**
  - HTML（HyperText Markup Language）
    - ハイパーリンクを持つハイパーテキストの記述言語
  - URI（Uniform Resource Identifier）
    - 統一リソース記述子。WWWでは主にリソースの場所を記述するのに使用する
  - HTTP（HyperText Transfer Protocol）
    - WWWでのリソース転送用プロトコル
    - 汎用性が高く、現在では多くのものを転送できる

ネット君のまとめノート その②



●HTTPメッセージ

－開始行

・リクエスト行…リクエストの内容。メソッドと要求リソースのURIがある

－メソッド…GET（取得要求）、POST（データ処理要求）など

・ステータス行…レスポンスの状態。レスポンスの状態コードがある

－メッセージヘッダ

・HTTPによる情報のやりとりに使用する付加情報

－メッセージボディ

・リクエストの場合は、GETならばなし、POSTなら処理に使うデータを入れる

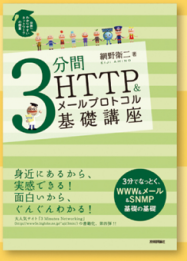
・レスポンスの場合は、要求されたリソースや処理結果のデータが入る

ネットワーク技術を身につけ極めよう！
[3 minutes networking seminar]

改訂新版 3 分間ネットワーク基礎講座
ISBN978-4-7741-4373-6
定価（本体 1780 円＋税）

改訂新版 3 分間ルーティング講座
ISBN978-4-7741-5737-5
定価（本体 1980 円＋税）

3 分間 DNS 基礎講座
ISBN978-4-7741-3863-3
定価（本体 2280 円＋税）



3 分間 HTTP＆メールプロトコル基礎講座
ISBN978-4-7741-4081-0
定価（本体 2280 円＋税）

SoftwareDesign 2015 年 5 月号 特別付録小冊子

著者／網野衛二（『3 分間 HTTP＆メールプロトコル基礎講座』より）
カバー＆レイアウトデザイン／トップスタジオ
発行所／株式会社技術評論社 〒162-0846 東京都新宿区市谷左内町 21-13

この個所は、雑誌発行時には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。