Special Feature 1 | HTTP/2超入門
Special Feature 2 | Firewallのしくみ

Software Design

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

2015 November 11

2015年11月18日発行
毎月1回18日発行
通巻367号
（発刊301号）
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価 本体 1,220円 +税

[Special Feature 1]

すいすいわかる HTTP/2

そろそろ
押さえておきたい
Web最新技術

HTTP/1.1から
↓
変わること、
変わらないこと

[Special Feature 2]

攻撃を最前線で防ぐ
ファイアウォールの教科書
iptablesからfirewalld、Web Application Firewallまで完全理解

[Extra Feature]

SMB実装をめぐる冒険 File System for Windowsの作り方

[New Serialization]

新連載！ 増井ラボノート「コロンブス日和」

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
November
11
Software Design

第1特集

そろそろ
押さえておきたい
Web最新技術

すいすいわかる HTTP/2

HTTP/1.1から
↓
変わること、変わらないこと

contents

2015
November
11
Software Design

contents

第2特集

攻撃を最前線で防ぐ
ファイアウォールの教科書 059

短期連載

Catch up trend

アラカルト

Column

Development

OS/Network

［広告索引］
システムワークス
http://www.systemworks.co.jp/
前付
GMOインターネット
https://www.conoha.jp/
裏表紙
日本コンピューティングシステム
http://www.jcsn.co.jp/
裏表紙の裏

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

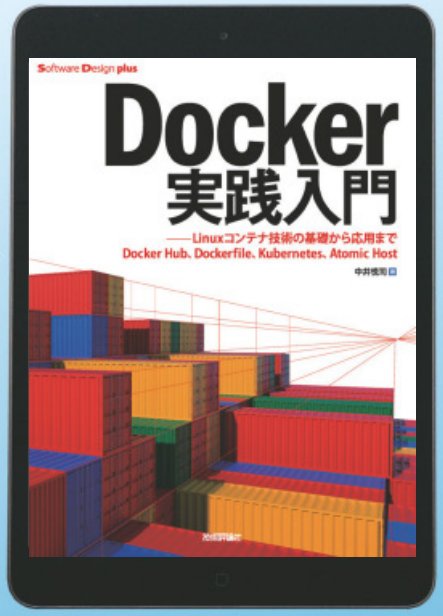
この個所は、雑誌発行時には広告が掲載されていました。編集の都合上、
総集編では収録致しません。

GIHYO Digital Publishing

技術評論社の本が電子版で読める!

https://gihyo.jp/dp

イチオシの1冊!



Docker実践入門
——Linuxコンテナ技術の基礎から応用まで

中井悦司　著

2,680円　PDF　EPUB

Linuxのコンテナ技術の1つであるDockerは，迅速なWebサービスの展開に必要不可欠なものであり，多くのIT企業が注目している重要なものである。本書では，そのしくみを明らかにし，DockerをGitHubと連携したデプロイ方法を基礎から解説する。Dockerfileの書き方や管理ツールであるkubernetesとの連携方法，レッドハット社のAtomicHostでの使い方など，最新かつ定番的な情報を盛り込んだ実践的な入門書である。

https://gihyo.jp/dp/ebook/2015/978-4-7741-7693-2

あわせて読みたい

たのしいインフラの歩き方

EPUB　PDF

PHPはどのように動くのか
~PHPコアから読み解く仕組みと定石

EPUB　PDF

あなたの知らない
超絶技巧プログラミングの世界

EPUB　PDF

データサイエンティスト養成読本
機械学習入門編

EPUB　PDF

他の電子書店でも
好評発売中!

amazon kindle　honto　楽天 kobo　ヨドバシカメラ www.yodobashi.com

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

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。

あなたの知らない超絶技巧プログラミングの世界

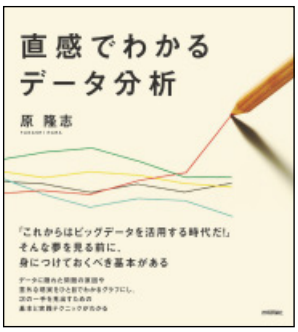
●遠藤侑介　著
B5変形判／272ページ
定価（本体2680円+税）

ISBN978-4-7741-7643-7

―役に立たない
　プログラムには価値がない？
―プログラミングは
　仕事以外でやりたくない？

本書には、アスキーアート化、自己生成、縛りのあるプログラミングなど、実用性を追求するだけでは出会えないテクニックが満載。プログラミングが好きな方はもちろん、プログラミングが苦手な方でも楽しめる遊びをIOCCC入賞常連の著者が紹介します。ちょっぴり不思議なプログラミングの世界をのぞいてみませんか。

直感でわかるデータ分析



●原隆志　著
B5変形判／144ページ
定価（本体1780円+税）
ISBN978-4-7741-7651-2

「これからはビッグデータを活用する時代だ！」
そんな声がよく聞かれるようになりましたが、現実的にはそうそう大量のデータを扱う機会などは無く、そもそも分析の基礎知識がなければ宝の持ち腐れになるだけです。
本書は、データ分析の基本にして強力な武器であるグラフを使い、データに隠れた問題の原因や真実をひと目でわかるようにし、次の一手を見出す方法を解説。分析にExcelを活用するためのポイントや、実践的なケーススタディも豊富に掲載した、これからデータ分析を学ぶ方のためのいちばんやさしい入門書です。

ISBN978-4-7741-7642-0

PHPはどのように動くのか

PHPコアから読み解く仕組みと定石

●蒋池東龍　著
A5判／248ページ
定価（本体2280円+税）

同じようなスクリプトなのに、なぜパフォーマンスが違うのか？オブジェクト指向だと、なぜ遅いのか？PHP7は、なぜ速くなったか？
最も人気のあるWeb用プログラム言語であるPHPの知られざる内部構造を解説した、日本初の書。「メモリを節約したり、処理を軽くしたりするスクリプトを書くには」「パフォーマンスの高いExtensionを作るには」「Zend Engineをハックするには」といった、ほかにはない話題が満載です。

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

Pocket Reference

小さくても、中身充実！

「あれ何だったっけ？」
「こんなことできないかな？」
というときに、すぐに調べられる
機能引きリファレンス。
軽くてハンディなボディに
密度の濃い内容がギューッと凝縮！
関連項目への参照ページもあって、
検索性もバツグン！

鶴長鎮一／馬場俊彰 著
四六判／400ページ
定価（本体2780円+税）
ISBN978-4-7741-7633-8

沓名亮典 著
四六判／608ページ
定価（本体2380円+税）
ISBN978-4-7741-7404-4

石田つばさ 著
四六判／448ページ
定価（本体2180円+税）
ISBN978-4-7741-4836-6

岡本隆史／武田健太郎／相良幸範 著
四六判／272ページ
定価（本体2480円+税）
ISBN978-4-7741-5184-7

土井毅／高江賢／飯島聡／高尾哲朗 著　山田祥寛 監修
四六判／488ページ
定価（本体2580円+税）
ISBN978-4-7741-4948-6

高橋晶／安藤敏彦／一戸優介／楠田真矢／道化師／湯朝剛介 著
四六判／544ページ
定価（本体2980円+税）
ISBN978-4-7741-7408-2

朝井淳 著
四六判／640ページ
定価（本体1980円+税）
ISBN978-4-7741-3835-0

高江賢 著　山田祥寛 監修
四六判／536ページ
定価（本体2580円+税）
ISBN978-4-7741-4592-1

山田祥寛 著
四六判／416ページ
定価（本体2680円+税）
ISBN978-4-7741-7078-7

大垣靖男 著
四六判／648ページ
定価（本体2580円+税）
ISBN978-4-7741-7229-3

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

第83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。

WebAssembly

Webの進化を促す新バイナリフォーマット

2015年6月、GoogleやMicrosoft、Mozillaといった Web基盤技術の開発にかかわる主要な企業・団体のエンジニアが、共同で「WebAssembly」と呼ばれる新しいフォーマットの開発に着手することを発表しました。これはWebブラウザ上で実行されるWebアプリケーションのためのオープンなバイナリフォーマットであり、各種ブラウザによって共通でサポートされる標準フォーマットになることを目指しています。

現在の一般的なWebアプリケーションはJavaScriptで記述され、ブラウザ上で構文解析されて動作します。それに対してWebAssemblyの場合は、任意の言語で記述されたコードを事前にバイナリコードにコンパイルしておき、それをブラウザ上のエンジンで直接実行します。これによって、Webアプリケーションにおいてもハードウェアの能力を活用したネイティブの処理速度を実現できるようになります。当初のターゲット言語はC/C++とされていますが、将来的にはそのほかのメジャーな言語もサポートしていくとのことです。

WebAssemblyプロジェクトには、冒頭に挙げた3社のほかにも、Webレンダリングエンジン「WebKit」やゲームエンジン「Unity」、クロスプラットフォームなコンパイラ基盤「LLVM」などの開発担当者が参加の意思を表明しています。W3Cにはコミュニティグループ「WebAssembly Community Group」が設立されており、開発中のソースコードはGitHub上にオープンソースで公開されています。

WebAssembly誕生の経緯

ここ十数年のWebの進化は、JavaScriptを共通の基盤言語とすることで成り立ってきました。そしてさらなる進化を促すために、JavaScriptの言語的な限界を突破するさまざまな試みが行われてきました。その一例が、TypeScriptやCoffeeScriptといった、JavaScriptをベースとする新しいプログラミング言語の登場です。これらのプロジェクトでは、JavaScriptに足りない有用な機能を備えた言語を用意することで、生産性や安全性の向上を目指しました。

別の試みとしては、Mozillaが中心となって開発を進めてきた「asm.js」があります。asm.jsはJavaScriptのサブセット言語であり、事前コンパイルによって最適化されたコードを出力することで、ネイティブに近い高速な処理を実現するというものです。開発から実行までをJavaScriptのままで行えるという点が大きな特徴ですが、すでにこれ以上の大幅な性能改善は難しいレベルに到達しているという問題がありました。

WebAssemblyもこれらの試みの延長線上にあるものと言えますが、バイナリフォーマットというJavaScriptよりも一段低いレイヤにおいて、特定のブラウザに依存しない共通機能として実現しようとしている点が大きく異なります。WebAssemblyプロジェクトがバイナリフォーマットの採用に踏み切った理由としては、JavaScriptをチューニングするasm.jsのような方式よりも圧倒的に高速であることや、ファイルの圧縮率が高く転送速度の向上が見込めることなどが挙げられています。

その一方で、WebAssemblyはJavaScriptに取って替わるものではなく、あくまでもJavaScriptエンジンという基盤の上に成り立つものだという点も強調されています。WebAssemblyはJavaScriptに不足する部分を補完する位置づけのもとに設計されており、おもにパフォーマンスが要求される場面などで威力を発揮するとのことです。

WebAssemblyプロジェクトの最初の目標としては、asm.jsと同等の機能を備えたプロトタイプの開発が挙げられており、すでに開発中のC++用コンパイラなどが公開されています。また、WebAssemblyをサポートしていないブラウザのために、WebAssemblyコードをJavaScriptに変換するツールの開発も行われています。

まだスタートしたばかりのプロジェクトですが、競合する主要ブラウザの開発陣が一致団結し、ノウハウを持ち寄って取り組んでいるだけに、今後の成果に期待が寄せられています。SD

WebAssembly Community Group
https://www.w3.org/community/webassembly/

DIGITAL GADGET vol.203

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

コンピュータグラフィックスの祭典SIGGRAPH 2015［前編］
映画の都ロサンゼルス。研究と展示編

CG技術と、その応用

コンピュータグラフィックスとインタラクティブ技術に関する世界最大の学会・展示会である、第42回SIGGRAPH 2015が8月9日から13日までの5日間、米国ロサンゼルスで開催されました。今年は70ヵ国から14,800人の参加者があり、143社の展示が行われました。

42回目となる今年のテーマは、“Xroads of Discovery”です。「X」は「Cross」の省略で、Crossroads of Discoveryとなり、直訳すると「発見の交差」です。アート業界とテクノロジー業界、または異なった道のりを進む人達が交差することによって、新しい発見や新しい作品が生まれることを歓迎するといったイメージです。

今年はスターウォーズの特殊効果制作などで知られるILM（Industrial Light and Magic）が40周年の年で、記念セッションも開催されました。ILMはこれまでに、15本のアカデミー賞、28本のアカデミー技術賞を獲得しています。40年前は映画の特殊効果のための道具やツールなどまったくなく、すべてILMのメンバーが工夫して作り上げてきました。CG黎明期の映像から現在の最新技術を駆使した映像までがスクリーンに映し出され、CGやVFXの40年の歴史を振り返る映像、思い出深い作品の数々に、若手・ベテランを問わず、セッションの参加者からは歓声があがっていました。

基調講演は、MIT Media Labの所長である伊藤穰一氏が登壇しました。元Media Lab所長のニコラス・ネグロポンテ氏の言葉「Bio is the new digital」を借り、インターネットの登場でさまざまな事柄が大きく変化し進歩したように、バイオの力がこれからいろいろな事柄を変化させていくという現在の技術の変化について触れた講演でした。

また、今年のSIGGRAPHアワードのうち、Coons Awardを受賞したのはHenry Fuchs氏でした。同氏が考え出したBSP Treeという三次元空間を分割して計算するためのツリー構造として把握するアルゴリズムや、バーチャルリアリティの黎明期を支えた長年の貢献が讃えられました。

今年のSIGGRAPHの傾向として、企業勢と大学との協同研究の成果が例年以上に目立っており、3Dプリンターや3Dスキャナを活用したり、応用したりする技術が多く取り上げられました。また、クラウドコンピューティングの浸透や、コンピューティングパワーが安価に活用できるようになったことから、機械学習を活用した研究や、Amazon Merchant Talkなどのクラウドソーシングを活用した研究が多くみられました。VR、HMDなどでも安価な機材が浸透し、大規模予算の作品もいくつか登場するなど、ブームの兆しが感じられます。

先進的なアイデアとヒントの固まり。CG論文の数々

今年のSIGGRAPH論文は118本。カテゴリもCGレンダリングからアニメーション、3Dプリンター活用、画像処理、動画処理まで多岐にわたっています。その中からデジタルガジェット的視点でいくつか紹介しましょう。

SIGGRAPH会場となったロサンゼルス・コンベンション・センター

アートギャラリーと先進技術展示コーナーの入り口付近。巨大な骸骨風プロジェクションマッピングは、BARTKRESA designによる「諸行無常」という作品

※本記事で紹介しているものは国内未発表・未発売のものを含んでいます。

Augmented Airbrush for Computer Aided Painting
http://dl.acm.org/citation.cfm?id=2699649

完全デジタルのエアーブラシ。傾きや位置を検知するデジタルブラシにより、テクニックがまったくなくとも素材に近い絵画を描くことができる。デジタルブラシはあくまで人間の作業を補助する役目であり、道具を操作するのは人であるが、習熟が必要ないのがポイント。

Computational Design of Twisty Joints and Puzzles
http://www.cs.columbia.edu/cg/twisty/

どんな形状も、3×3×3のオブジェクトを回転して遊ぶルービックキューブ風のパズルにしてしまうアルゴリズム。簡単そうに見えるが、回転した際に相互にぶつからないようにあらゆるパターンで衝突検知計算したり、分解してしまわないように回転軸を考慮したりと工夫されている。データ生成した3Dパズルは、3Dプリンターで出力できる。

Reduced-Order Shape Optimization Using Offset Surfaces
https://www.graphics.rwth-aachen.de/publication/249/musialski-2015-souos-preprint.pdf

立体形状を、なんでも「起き上がり小法師」にしてしまうアルゴリズム。底面の適度な滑らかさと、素材と空洞を考慮することで重心をコントロールし、起き上がり小法師化する。

SecondSkin:Sketch-Based Construction of Layered 3D Models
http://www.dgp.toronto.edu/~depaolic/projects/secondSkin/SecondSkin.pdf

すでに形状が決定しているフィギュアや人体模型にぴったりと合う、鎧や衣服のデザインを平易に実現できるようにする「第二の皮膚」というプロジェクト。

Foldabilizing Furniture
http://honghuali.github.io/projects/foldem/

どんな形状の家具でも、折りたたみ可能な形状にしてしまうアルゴリズム。家具の省スペース化を目的としたもの。ヒンジなどの折りたたみ構造や折り畳んだ際のぶつかりを検知し、競合状態を排除して、実現可能な折りたたみ形状を導き出す。試作には3Dプリンターを活用。

Computational Interlocking Furniture Assembly
http://www.ntu.edu.sg/home/cwfu/papers/interlockfurniture/

ある家具形状を、組み立てパズル化し、実用的な形状として安定させるためのアルゴリズム。組み立て家具を組み立てる際、悩まずに的確にできるようにという効果も狙っているもよう。接着剤なしで、丈夫な構造を組み立てるのも目的の1つ。

Single-View Hair Modeling Using A Hairstyle Database
http://www.hao-li.com/publications/papers/siggraph2015SVHMUAHD.pdf

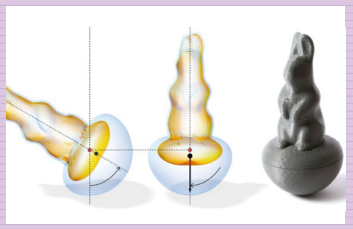
1枚の写真と、数本の手作業によるストローク指定で、髪の毛の三次元モデルを生成してしまうアルゴリズム。仮想空間の中の3Dキャラクタを作成する際、特殊な3Dスキャナなどなしに、1枚の写真から簡易的な髪の3Dモデルを作れる。大量の髪形状のデータベースから、類似性のある形状を再利用して活用している。

Elastic Textures for Additive Fabrication
http://vcg.isti.cnr.it/Publications/2015/PZMPCZ15/

3Dプリンターでプリント可能な幾何学パターンで、形状の変化の度合いを想定し、適切な変形をコントロールする技術。従来も複数の3Dプリント素材による形状変化のコントロール技術はあったが、一種類の素材で実現したところがポイント。家庭用の安価な3Dプリンターでも再現できるので、衣服やオモチャへの応用など楽しみ方や可能性が広がる。


Augmented Airbrush for Computer Aided Painting


Computational Design of Twisty Joints and Puzzles


Reduced-Order Shape Optimization Using Offset Surfaces


SecondSkin:Sketch-Based Construction of Layered 3D Models


Foldabilizing Furniture


Computational Interlocking Furniture Assembly

OmniAD:Data-Driven Omni-Directional Aerodynamics

http://www.disneyresearch.com/publication/omniad/

どんな形状でも空飛ぶ凧にしてしまう、機械学習によるプロジェクト。三次元剛体の空気特性をモデル化するもの。従来は大規模な風洞実験などを行わなければならなかったが、このアルゴリズムでは、物体が落下する際の映像から空気特性を抽出し、シミュレーションすることができる。

これからのコンピュータグラフィックスの進化

最近のSIGGRAPH論文の傾向として、論文のみならず、サンプルソースや実際に試すことのできる実行ファイルを公開する事例も増えてきました。また、難解な英語論文を読まずとも、YouTubeでわかりやすいデモ映像を観られるものも多くなりました。論文で取り上げられた最新技術が、今日明日すぐに実際の仕事で使えるわけではありません。実用化までは少し時間がかかりますが、どのような傾向や、どのような可能性が広がっているのかを把握するには、心躍る内容の数々がそろっています。11月2日から神戸で開催されるSIGGRAPH ASIA 2015にも期待が寄せられています。SD

Single-View Hair Modeling Using A Hairstyle Database

Elastic Textures for Additive Fabrication

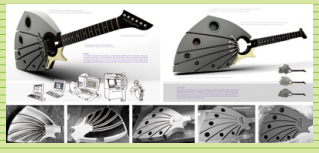
Gadget 1

The Other Way Around: From Virtual to Physical

http://s2015.siggraph.org/attendees/art-gallery/events/other-way-around-virtual-physical

3Dプリンターを活用した斬新なデザインの木製ギター

アートギャラリー「ハイブリッドクラフト展」から。Amit Zoran氏、Seppo Valjakka氏による作品。3Dプリントでギターの試作をし、そのままでは良い音がしないので、実際のギターと同じように木材で制作してみたもの。伝統的な手法で作られる楽器の制作で、従来の手順に縛られず、テクノロジーを活用した新しい発想で形あるものを作ろうというプロジェクト。



Gadget 2

Bicycle Frame Domestic Fabrication

http://s2015.siggraph.org/attendees/art-gallery/events/bicycle-frame-domestic-fabrication

鉄パイプ部品を3Dプリント素材に置き換えた自転車

同じくアートギャラリー「ハイブリッドクラフト展」から。Atar Brosh氏による作品。自転車の鉄製フレーム部品を、3Dプリントした部品に置き換えることによる、ハイブリッドな作品例。十分な強度を持つ樹脂でできた部品に置き換えることで、輸送コストの軽減や、独自の自転車組み立てキットを提供できようになったもの。

Gadget 3

Dandelion Diptych

http://s2015.siggraph.org/attendees/art-gallery/events/dandelion-diptych

LED埋め込みのデジタル掛け軸

同じくアートギャラリー「ハイブリッドクラフト展」から。Jie Qi氏、Nicole Teeny氏とRebecca Kleinberger氏による作品。馴染みのある絵画の質感と、デジタルのペイントの意味合いをもった点滅などの光を制御したLEDライトが組み合わさった作品。電子回路そのものも、絵画の一部として構成されているところが特徴。

Gadget 4

3D Printing and Jewelry Making

http://s2015.siggraph.org/attendees/art-gallery/events/3d-printing-and-jewelry-making

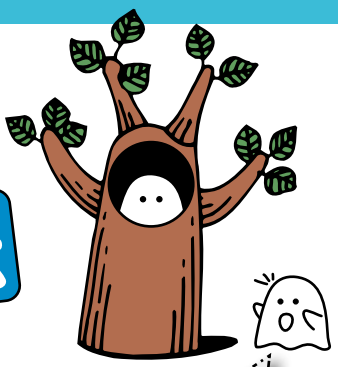
パズル式の3Dプリント指輪

同じくアートギャラリー「ハイブリッドクラフト展」から。Yael Friedman氏による作品。3Dプリントで製作した緻密な部品がパズルになっており、そのパズルを組み立てると実用的な指輪になるというアート作品。3Dプリンターも、樹脂だけでなく、チタンや貴金属などさまざまな金属素材がプリントできるようになったことで、大きな可能性が広がっている。

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

30

結城浩の再発見の発想法



Pattern Match

Pattern Match ——パターンマッチ

パターンマッチとは

パターンマッチ(Pattern Match)とは、**文字列中のパターンを見つけたり、それを置換したりする**機能です。たとえば、テキストファイルの中にcatという文字列が何百個も出てくるとき、それをすべてdogに書き換えたいとします。テキストエディタを使ったり、簡単なプログラムを書いたりして置換を行うことが多いでしょう。これは、catという**パターン**を使ってテキストファイル中の文字列をdogに置換したことになります。パターンマッチによって作業効率は上がります。

しかし、パターンマッチがいくら便利でも「catをdogに置換」と単純に考えられない場合があります(**図1**)。cat(猫)という文字列はcatalog(カタログ)の始めにも登場するので、単純な置換ではcatalogがdogalogになってしまうからです。

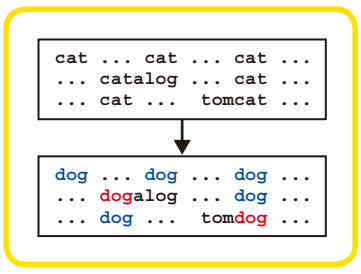
だからといって「catの後がスペースになっているときに限りdogに置換」というのも安易です。確かにこれならcatalogはdogalogになりませんが、今度はThis is a cat.に出てくるcatが置換されなくなってしまいます。また、tomcat」もtomdog」に置換されてしまいます。どのようなパターンなのか、きちんと表現しなければなりません。

多くのプログラミング言語では、よく出てくるパターンを記述するために**正規表現**が使われます。たとえば、正規表現で\bと書けば「単語の境界」を表すパターンになります。「catという単語」を表すパターンを正規表現で書きたかったら、catを\bではさんで\bcat\bのように書けばいいことになります(**図2**)。そうすれば、catにはマッチし、catalogやtomcatにはマッチしないパターンとなります。

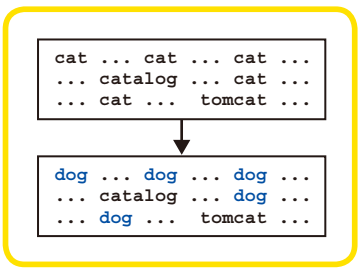
パターンを表す正規表現

ところで、考えてみますと、\bcat\bというパターンではcat(単数形)にはマッチしますが、cats(複数形)にはマッチしませんね。「sがあってもなくてもいい」を表現するため、正規表現ではs?のように書きます。つまり、\bcats?\bと書けば「catという単

▼図1　catをdogに置換すると……



▼図2　\bcat\bで単語を指定



語」と「catsという単語」のどちらにもマッチするパターンになるのです。

正規表現では多様なパターンを記述できます。dog|catのように縦棒(|)を使うと「dogとcatのいずれか」にマッチするパターンになります。a+のようにプラス(+)を使うと、a、aa、aaa、……のように「aの1回以上の繰り返し」にマッチするパターンとなります。おもしろいことに、0回以上の繰り返しを表すアスタリスク(*)も別に用意されています。a*は「空文字列」にもマッチするパターンとなります。

パターンマッチが有用な理由

テキストエディタであれ、プログラミング言語であれ、パターンマッチの機能はとても有用なものです。パターンマッチが有用な大きな理由は、それが**大量のデータ処理**に向いているからです。もともと、プログラムを使う大きな目的は、人間にはうまく処理できないほど大量のデータを処理することにあります。人間はひとつひとつを判断せず、コンピュータにまかせられる。それがプログラムの利点です。

パターンマッチは、プログラムを使うその目的にぴったりと合致しています。パターンマッチを使うと、大量のデータの中から、関心のあるものを見つけ出すことができるからです。たとえば、テキスト中のcatは、その「関心のあるもの」に相当します。

文書としてのテキストファイルに限りません。Webサーバが出力するログや、開発中にプリントアウトされるデバッグ情報など、とにかく人間が処理できないほど大量なデータから「関心のあるもの」を見つけたいとき、パターンマッチは強い味方となるのです。

言語内言語

パターンマッチを記述する正規表現は、「sがあってもなくてもよい」という条件や、「catまたはdog」という論理和や、「aの繰り返し」という反復を記述できます。つまりプログラムの中に書かれたパターンは、プログラムの中に、小さな別のプログラムが埋め込まれているようなものです。

プログラマはプログラミング言語を使ってプログラムを書きますが、その途中でパターンマッチが必要になったときには正規表現を使います。自分の「関心があるもの」をプログラム中に記述するのです。

日常生活でのパターンマッチ

パターンマッチでは、繰り返しを発見し、その繰り返しの表現手段を提供することが大事になります。この「**繰り返しの発見**があったら、そのために**表現手段の提供**を行う」という発想は、あちこちに現れます。

- プログラミングではパターンマッチをよく使う(繰り返しの発見)。だから、パターンをプログラミング言語で表現できるようにしよう(表現手段の提供)
- パターンマッチでは繰り返しをよく使う(繰り返しの発見)。だから、正規表現では繰り返しを表現できるようにしよう(表現手段の提供)
- 同じ繰り返しでも「0回または1回」「0回以上」「1回以上」はそれぞれよく使う(繰り返しの発見)。だから、これらを区別して表現できるようにしよう(表現手段の提供)

私たちは日常生活の中で「先日も同じようなことをやった」と感じることがよくあります。これはまさに「繰り返しの発見」です。このような発見は、私たちが心の中でパターンマッチの一種を行っているからでしょう。その繰り返しをうまく表現する手段があれば、効率化ができるかもしれませんね。

◆ ◆ ◆

あなたのまわりを見回して「同じようなことをやっている」と感じることはありますか。そのパターンをうまく表現する手段はあるでしょうか。ぜひ、考えてみてください。SD

増井ラボノート 新連載！

コロンブス日和

第1回 GyaTV

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

Author 増井 俊之（ますい としゆき） 慶應義塾大学

GyaTV——Webページを自動的に表示する

楽するためには作業の絶対量を減らさなければなりません。ソフトウェアを作成する場合は、良い言語やエディタを使って効率を上げるのは基本でしょう。似た操作を繰り返す場合は、次の操作を予測することによって作業量を減らせることもあります。

しかしいくら作業を減らす工夫をしても**何もしない**ことにはかないません。何も作業せずにソフトウェアは作成できませんが、情報を眺めたり収集したりという程度であればまったく手を動かさなくてよい可能性があります。テレビのニュースをつけっぱなしにしておくと、なんとなく世間の新しい情報について知ることができます。記憶したい情報をずっとどこかに表示しておくようにすれば、知らないうちに覚えてしまうかもしれません。

GyaTV（ギャツビー）は、登録しておいたURLをスクリーンセーバのようにランダムに表示させるWebサービスです。原理は簡単で、私が運営しているGyazzというWikiのページに登録したURLのリストから、定期的にランダムにURLを1つ選択してそのページを表示するというものです。Web上のコンテンツの中身を吟味して選択したり観賞したりしているといくら時間があっても足りませんが、自動的に表示されたものを眺めるだけなら楽です。

GyaTVの原理はとても単純ですが、さまざまな応用ができて便利に利用できます。

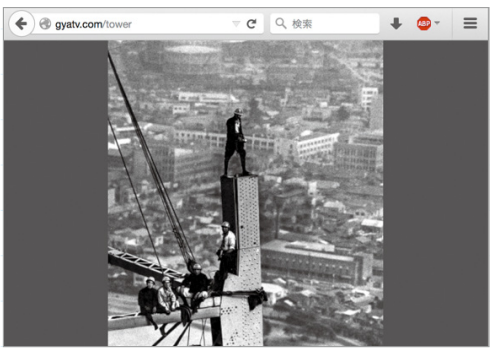
デジタルフォトフレーム

自分のデジカメ写真をランダムに液晶画面に表示する「デジタルフォトフレーム」という製品が流行ったことがあります。こういう製品は最近あまり話題になっていないようですが、デジカメ写真をスクリーンセーバのように表示するのは気持ちが良いものです。

手持ちのパソコンやタブレットコンピュータでGyaTVを動かしておけば、デジタルフォトフレームとして使うことができます。きれいなイラストや写真のまとめがよくWebに投稿されていますが、私は「東京タワー」（**図1**）や「スミソニアン博物館」（**図2**）の写真をGyaTVで表示して楽しんでいます。

GyaTV.com/xxxというURLでスライドショー表示を行うためには、Gyazz.com/GyaTV/xxxというWikiページにURLリストを登録します。

▼**図1** 東京タワーのスライドショー（毎日新聞のサイトの写真をランダム表示）



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

たとえば**図3**のようにhttp://Gyazz.com/GyaTV/towerにURLのリストを登録しておけば、http://GyaTV.com/towerでスライドショーを楽しむことができます。

Wikipediaのランダム表示

あまり知られていないかもしれませんが、Wikipediaページの左側には「おまかせ表示」というリンク(**図4**)があり、これをクリックするとWikipediaのランダムなページに飛ぶようになっています。

Wikipediaの情報は膨大ですので、ランダムにページを表示したとき、自分が知ってる内容が出ることはほとんどありません。クリックするたびに新しい情報に触れることができるのは楽しいものです。手動でクリックしても楽しいのですが、GyaTVにおまかせページを表示するURLを登録しておけば、一定間隔で新しいページが表示されることになるのでなかなか新鮮です(**図5**)。

単語帳の表示

「門前の小僧、習わぬ経を読む」と言われるように、何度も同じものを見聞きしていると知らない間に覚えてしまうことがありますが、この性質を勉強に利用できます。英単語とその意味、用例、画像を並べたWebページを用意しておき、これをランダムに表示するようにしておけば、ときどき画面をチラ見しているうちに単語とその雰囲気を覚えてしまうでしょう。私はこのようなWikiページをたくさん用意しており、これをGyaTVで表示することによって単語力を上げようとしています(**図6**)。

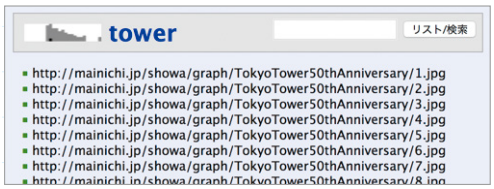
画像を検索すると単語の用例が直感的に理解できることもあります。「convocation」という単語を普通の辞書で調べると「会合」のような意味が出てきますが、Google画像検索を行うと**図7**のような写真ばかり出てきます。どうやらこの単語は「卒業式」で使われるらしいということがわかります。こういう画像をずっと眺めていると「convocation」という単語の雰囲気を理解できます。

ニュースの表示

昔は街中にニュースを表示する電光掲示板が

▼ **図2** スミソニアン博物館のコレクション

▼ **図3** URLのリスト



▼ **図4** Wikipediaの[おまかせ表示]ボタン(https://ja.wikipedia.org/wiki/特別:おまかせ表示)

▼ **図5** ランダムに表示される謎Wikipediaページ

よくあったものです。現在でも新幹線の車両の端ではニュースが表示されていますし(**写真1**)、ニュースを知るために家でテレビをつけっぱなしにしている人も多いでしょう。

ニュースのような情報ソースは、真剣に見るのではなく、なんとなく聞こえてくるのが好まれるということでしょう。新しいニュース情報をWikiページに書き込んでおけば、GyaTVでいつも最新のニュース情報を表示させておくことができて便利です。

表示順序の制御

GyaTVでは、登録したURLがランダムに表示されますが、"http://GyaTV.com/tower?play=seq"のようにオプションを指定するとリストした順番のとおりにページを表示できます。旅の写真を順番に表示したい場合や、スライド的に情報を伝えたいような場合はページの順番が大事になるので、このオプションを付けて表示するのが良いでしょう。

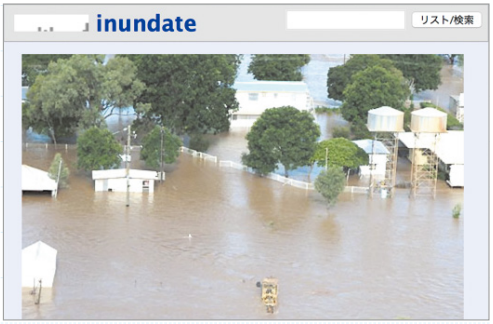
情報共有や情報通知

GyaTVページはWeb上にあってどこからでも見えますから、自分のきれいな写真を自動再生するGyaTVページを作って他人に自慢できます。写真を並べて旅日記を作っておけば、自分で楽しめるうえに他人に自慢もできるのでうれしいでしょう。

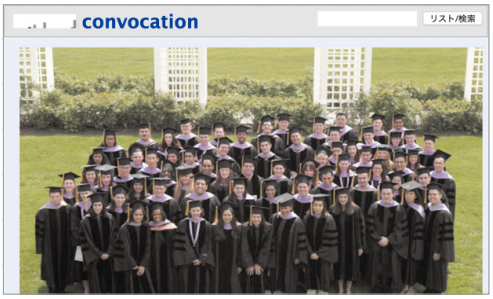
コンピュータを使うのが苦手な老人などにネット経由で情報を伝えたい場合、そのような人の家のiPadなどで常にGyaTVのページを表示するよ

▼**写真1** 新幹線の電光掲示板

▼**図6** 「inundate(洪水)」で表示される映像



▼**図7** 「convocation」という単語の画像検索



うにしておき、情報を伝えたい人がGyazzの内容を書き換えるようにしておけば、常にその人に対するメッセージを表示し続けることができます。

表示時間の指定

GyaTVでは標準では15秒ごとにページが更新されますが、URLの後に数字を書いておけば表示する秒数を指定できます。読むのに時間がかかるページや、長く表示しておきたいページは長時間表示し、急いでたくさんのページを表示したい場合は小さい数字を指定すれば良いでしょう。また、きちんと秒数指定しておけば、YouTubeなどで音楽や動画を順番に表示させることもできます。

置くだけ再生

AndroidスマホやタブレットにはNFCリーダーが登載されており、SuicaのようなRFIDカード／タグを読み出すことができます。Androidの場合、端末をタグに近付けることによってアプリケーションを起動することもでき

るので、タグを置いた場所に置くだけでGya TVのページを開くことができます。この機能を利用すると、タブレットやスマホを特定の場所に置くだけでスライドショー／予定表／時計などを表示できるので便利です(**写真2**)。

自動表示で気づくこと

GyaTVを使っていると次のようなことに気づきます。

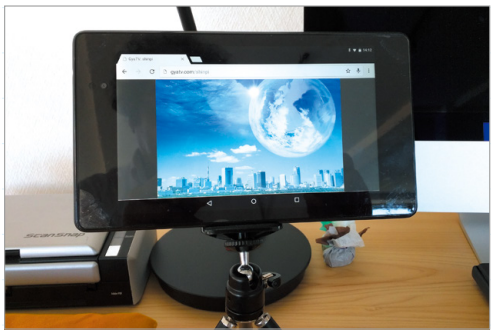
時間感覚のあやしさ

15秒ごとにランダム表示を行っている画面をずっと見ていると、なかなか画面が更新されないことにイライラするものです。一方、仕事をしているときに別のマシンでGyaTVを動かしているときは更新の遅さが気にならないどころか、更新が速すぎると感じることもあります。どうやら人間は、自分が注目しているものとしていないものに関して時間感覚がかなり違うのだろうと思います。小さいと思っていた親戚の子供に久しぶりに会ったら成長ぶりに驚くことがよくあります。あまり興味のないWebサービスが開始したと思ったらいつの間にか終了していることもあります。自分が深くかかわっていないものは進行速度が速く感じられる気がします。つまり、仕事に極度に集中すれば時間の進行が遅く感じられ、離れたところに置いたものは時間の進行が速く感じるのでしょう。このような時間感覚を活用できれば、大事なことには時間をかけつつ、どうでも良いことはすぐに終了させることができるのかもしれません。

乱数感覚

ランダム表示を行うとき、普通の乱数を利用するとランダム感が得られないことがあります。10枚の写真をランダムに表示させると、10回に1回は同じ写真が連続して表示されることになります。これを

▼**写真2**　置くだけ再生しているところ



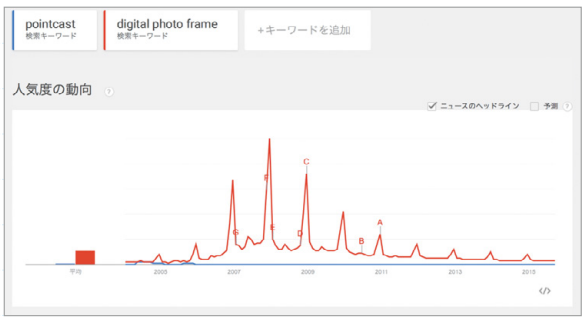
避けるためにはrand()のような乱数関数を利用するかわりに、履歴を考慮したニセ乱数関数を使うのが良いと考えられます。実際、音楽プレーヤのシャッフル再生機能では、本当の乱数の代わりにニセ乱数が利用されているようです。

◆ ◆ ◆

インターネットの黎明期、ニュースをスクリーンセーバのように配信するPointcastというサービスが注目されたことがありましたが、結局あまり流行しませんでした。また、たくさんの写真を自動的に表示してくれるデジタルフォトフレームも最近あまり流行していないようです(**図8**)。

これらのシステムがあまり流行らなかった理由はいろいろあるのでしょうが、ネット上の情報を自動表示すること自体は有益なはずです。ぜひGyaTVを使っていろいろ楽して楽しんでみてください。SD

▼**図8**　GoogleTrendによる情報



かまぷの部屋

第16献 ゲスト：角田 千佳さん

㈲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

角田 千佳（つのだ ちか）さん

㈱エニタイムズ　代表取締役。慶応義塾大学法学部政治学科卒業後、㈱野村證券に入社。その後、㈱サイバーエージェントに転職し、事業立案、ITなどを学び、独立を決意。2013年、「豊富な幸せの尺度を持った社会の実現」と目標を掲げ、生活密着型シェアリングエコノミーサービス「Any+Times（エニタイムズ）」を創業する。最近では、TVや雑誌、Webメディアなど、さまざまなメディアに取り上げられている。趣味はヨガ・ダンス。

京都にて。

ご出身はどちらでしょうか？　子供時代は何に興味を持っていましたか？

出身は東京都杉並区です。幼稚園から高校まで、近所にある国学院久我山へ通学していました。子供のころはスポーツと本が好きでした。実家には司馬遼太郎作品をはじめ、歴史ものの本がたくさんあり、それを読んでいるうちに、漠然と「世の中を変えたい」と思っていました（笑）。

大学では政治学を学ばれたとのことですが。

大学のゼミでは、オーストラリアの先住民「アボリジニ」とその地域について、研究していました。本格的に地域・社会貢献を考え始めたのは緒方貞子さんの本を読んだ小学生のころからです。一生を通して、『地域活性』に携わりたいという想いがあります。政治、社会学、地域研究への関心は、どちらかというと父の影響です。母は理系でエンジニアだったんですよ。

お母様の世代で女性エンジニアは珍しいですよね。

そうなんです。新卒で銀行のオンラインシステム開発の部署に就いたということなのですが、母が女性で初のエンジニアで、アセンブラを使って開発していたそうです。

アセンブラで開発されていたとは！　IT関連の仕事をされているのはお母さんの影響でしょうか。

実はITは最近まで苦手だったんです。新卒で證券会社に就職したのですが、マシン操作といえば、Excelが使えれば問題ないレベルで、社内システム以外のインターネットやメールも使えない環境でした。その後、サイバーエージェントに転職したのですが、大学時代にインターンをしていたときにできた縁がきっかけでした。

エニタイムズの起業に至る経緯を教えてください。

実家を出て、単身者が多い地域で1人暮らしを始めたときのことです。近所に引っ越しの挨拶をしようと回ったのですが、ドアを開けてくれない方も多かったのです。地元での近所づきあいに慣れていたので、それには驚きました。開けてくれたところでも、挨拶すると訝（いぶか）しがられる始末です（笑）。そのあと、困ったことがあったときに便利屋さんを頼んだことがあったのですが、そういう




ことは近所づき合いでお願いできたほうが便利だと気づいたんです。そして、これをマッチングするサービスが創れないかと閃(ひらめ)きました。

エニタイムズを一言で言うと、どんなサービスでしょうか？

「ちょっとした用事のマッチングで地域活性化」でしょうか。エニタイムズのサービスは、依頼者も提供者も対等であるところがポイントです。また、私自身ももちろん利用者であり、仕事提供者です。ご近所同士が助け合えるようなちょっとしたお仕事を、ITのプラットフォームを利用して活性化させていきます。

サーバは自社で運営されているのですか？

現状のサーバはAWS上でRuby on Railsで稼動しています。PC版のサイトはSEOに力を入れていますが、ユーザはiOSやAndroidなどスマホ上でモバイルアプリを使う方がほとんどです。

PCで使っている方が多いのかと思っていました。

ただ、モバイルアプリがまだ使いにくいので、現在、抜本的に見直しを図っているところなんです。Go言語も検討していますが、エンジニア人口が少ないという点が悩みです。SwiftやUnityなど、モバイルに強いエンジニアがこれからほしいと思っています。

エニタイムズを運営していてうれしかったことはありますか？

私もよくエニタイムズの家事などのサービスを利用するんですが、そのときに来てくれた専業主婦の方が「家では家事をやっても旦那はおろか子供も感謝してくれない。片付けをすれば、逆に怒られることもある。

ここで依頼された方には感謝してもらえて、さらにお金ももらえるので本当に楽しいし、社会とつながる感覚もうれしいです」と言っていたんです。

会社のWebサイトに顔出てますよね、お客さん気づかないんですかね(笑)。

わからないです(笑)。このサービスを開始して初めて知ることは多く、さらにデータ分析をすると思わぬ発見があります。実際のところユーザは男性が多く女性が少ないんですね。当初は女性が多いと考えていました。

子育て世帯にはうれしいサービスだと思うのですが、そういった利用は多いですか？

はい。実際のところはベビーシッターの利用よりも、子供の面倒は自分たちがみるから、家事をお願いしますという利用がほとんです。欧米のように、ベビーシッターの利用もこれからは増えていくと見込んでいます。私も子供ができたら積極的に利用する予定です。

将来の自分が利用することも考えているのですね。クレーム対応は多いのですか？

サービスの目的をよく理解して利用したいただく方が多いので、想定していたより少ないです。

現在、プライベートで大事にされていることは？

今は仕事のことしか考えられません。土日も旅行中もサービスのことを考えていて、友人には呆れられています(笑)。家族に関しては、弟の奥さんも社員なので、一緒にいることも多く、実の妹のようでとてもうれしいです。私自身は家族になってもいいな、という人ができたら結婚したいですね。

住みやすい社会になれば、少子高齢化にも歯止めがかかるかもしれませんね。角田さんのサービスが広がりますように期待しています！　今日はどうもありがとうございました。




ツボイのなんでもネットにつなげちまえ道場

第五回

マイコンとデバイスの通信プロトコル

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

はじめに

前回まで、LEDを光らせるといった、とても単純な出力について説明をしてきました。実際にマイコンに各種のセンサを接続するときには、さまざまなプロトコルが使われます。本誌読者にとってプロトコルというと、TCP/IPやHTTPといったプロトコルでしょう。マイコンでも、UART、I²CやSPIといった定番プロトコルがあります。

UART

UART（Universal Asynchronous Receiver Transmitter）は、ここで紹介するプロトコルのうち、最も身近なプロトコルでしょう。ルータやインテリジェントスイッチなど、ネットワーク機器を設定するときに使う「シリアル通信」です。このシリアル通信注1というのは、たいてい、このUARTのインターフェース規格の1つであるRS-232Cを指しています。インターフェース規格というのは、使用するコネクタやピン配置、電圧などを定めているものです。

UARTの電気信号は、**図1**のようになっています。シリアル通信ですので、1bitずつ順に送ります。UARTは、Asynchronous（非同期）という単語が名前に含まれているように、非同期な通信方式です。非同期ですので各bitの状態を表す時間の間隔は、UARTで通信するデバイス双方で一致していなければなりません。このため、UARTで通信を行うときには、9,600bpsといった具合に通信速度を指定します。9,600bpsの場合、1秒間に9,600bitを送りますので、1bitあたり1÷9,600≒0.0001042秒、つまり104.2マイクロ秒ごとに電圧を見て0か1を決定します。ほかにUARTの通信のオプションには、データビット数、パリティ、ストップビット長があります。ネットワーク機器のコンソールに接続するとき、「8N1」といった具合に指定します。これは、データビット長が8ビット、パリティビットなし、ストップビット長が1ビットであることを指します。**図1**の例は、この「8N1」のケースです。

UARTには、TXという信号を送る線と、RXという信号を受け取る線の2本の信号線が使われます。このほかにも、UARTでは、RTSやCTSといった信号線を使う場合があります。送信側は送信をしたいときにRTSをHIGHにし、受信側が受け取れる状態になるとCTSをHIGHにします。送信側はCTSがHIGHになったことを確認してから、データを送信し始めます。こうした制御方法は、フロー制御と呼ばれます。

注1）RS-232Cを指して「シリアル通信」だとか「シリアルポート」と呼ぶのは本来の意味では正しくありません。シリアル通信は、信号線の上を一度に1bitずつ、順にデータを送ることを指します。昔はシリアル通信を行う場合、RS-232Cを使うケースがほとんどだったかもしれませんが、今ではUSB（ユニバーサルシリアルバス）やSATA（シリアルATA）、SAS（Serial Attached SCSI）など、さまざまなシリアル通信規格が使われています。

▼図1　UARTの信号例

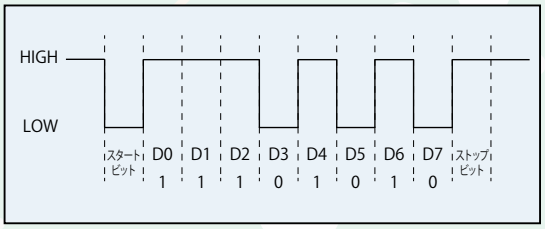


図2のように、UARTは、一方のTXを他方のRXに、一方のRXを他方のTXにと、互い違いに接続をします。もちろん、通信する相手と基準とする電位を一致させなければならないので、GNDどうしを接続する必要があります。

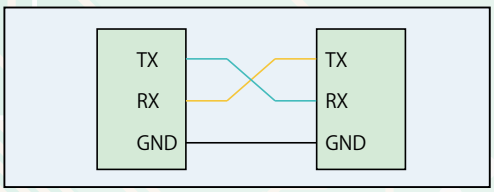
UARTはとても頻繁に使われているプロトコルで、USB-UARTアダプタ(**写真1**)などをパソコンに接続し、ターミナルソフトを使えば簡単にパソコンに喋らせることができます。Linux搭載の組み込み機器のコンソールのみならず、マイコンの開発を行うときにデバッグメッセージを簡易的に表示するために使われたりしています。

SPI

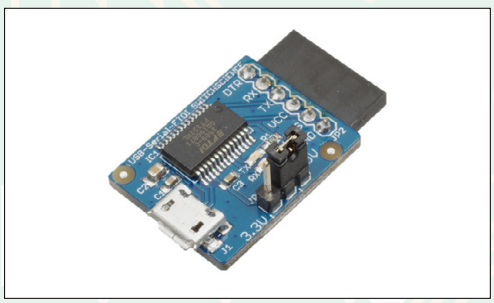
SPI(Serial Peripheral Interface)もシリアル通信ですが、非同期通信のUARTとは異なり、同期通信を行うプロトコルです。少ないピン数で接続できるバスとして、モトローラ(現在は、NXPセミコンダクターズに買収されたフリースケール・セミコンダクタ)が開発した規格です。同期ですので、SPIの通信線にはSCLK(Serial Clock)と呼ばれる同期用のクロック信号があります。UARTのTXやRXと同様、SPIにはMOSI(Master-Out Slave-In)とMISO(Master-In Slave-Out)という2本の信号線があります。UARTとは違い、SPIでは、SCLKのクロックを供給するマスタと、その通信相手のスレーブという役割が明確です。SPIは、1つのバスで複数のデバイスと通信ができるのですが、通信相手を指定するためにSS(Slave Select)という信号線を使います。SSは、通信相手ごとに通信線を1本ずつ用意する必要があり、マスタは通信したい相手と接続されているSSをLOWにすることで、相手を指定します。**図3**中で、SSのあとに「#」が付いているのは、LOWのときにActiveである(アクティブ・ロー)であることを示すための印です。

SPIの電気信号は、**図4**のようになっています。SPIは全二重で、MOSIとMISOでデータの送受信が同時に行われます。SPIの通信はマスタが主導して行いますので、スレーブからマスタがデータを読み込むためには、マスタがSCLKを送らなければなりません。マスタがSCLKを出せば、MISOを通じてスレーブから

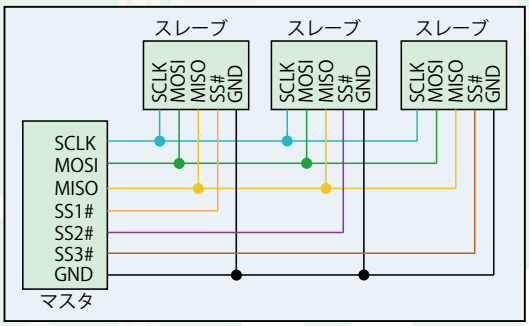
▼図2　UARTの接続例



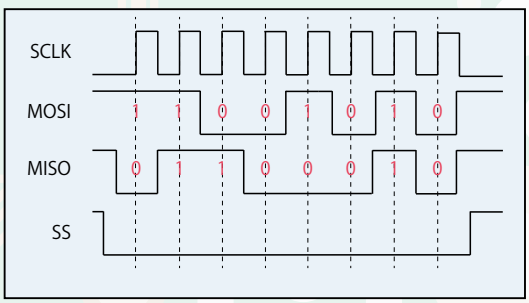
▼写真1　FTDI USBシリアル変換アダプタ



▼図3　SPIの接続例



▼図4　SPIの信号例



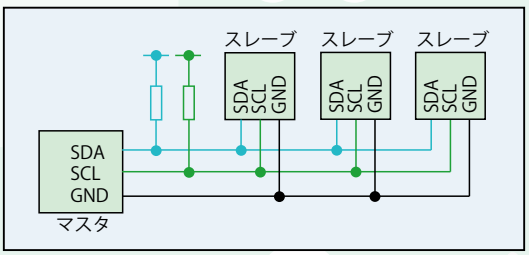
マスタにデータが送信されます。UARTを使えば、スレーブは任意のタイミングでマスタにデータを送ることができるのですが、SPIではこのような方法が採れないのです。こういった、スレーブがマスタに伝えたい情報があるということを知らせたい場合、SPIとは別に「転送したい情報がある」ことを知らせる信号線を用意し、その信号線をHIGHやLOWに状態変化させて知らせるといった実装を見かけます。

SPIのデータの送受信は、クロック信号(SCLK)の立ち上がり(LOWからHIGHへの切り替わり)や立ち下がり(HIGHからLOWへの切り替わり)に合わせて行われます。このクロック信号の極性(CPOL)2種と、MOSIやMISOの信号とSCLKとの位置関係(CPHL)2種を組み合わせると、4通りになります(**図5**)。

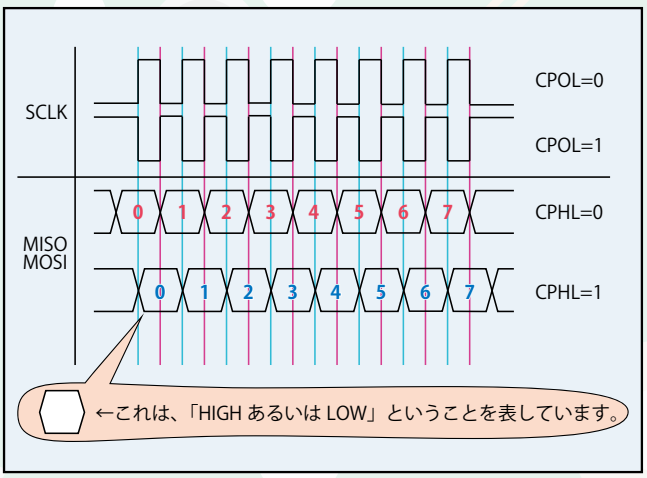
このクロックと信号の位置関係は、位相と呼ばれます。この組み合わせ4種類は、モードと呼ばれています。SPIの設定は、SCLKの周波数と、一度に転送するデータのビット長、それからモードです。SCLKの周波数は、UARTと比べて相当に高速で、場合によっては数十MHzでSPIは使われます。

SPIは各種センサやディスプレイ、あるいはフラッシュメモリをマイコンに使うために多く使われています。ディスプレイなどスレーブからマスタに転送する情報がない場合、MOSIだけをつなぐケースもあります。SPIはUARTと違い、複数のデバイスを1つのデバイスに接続できますが、SSをデバイス分用意しなければならないため、複数のデバイスを接続するときにはマイコンの入出力ピンを使ってしまいます。また、先述のように、マスタ主導で通信を行うため、少しコツが必要です。

▼図6　I²Cの接続例



▼図5　CPOLとCPHL



I²C

I²C(Inter-Integrated Circuit、アイ・スクエアド・シーと読みます)は、フィリップス(現在は、NXPセミコンダクターズ)が開発した規格です。SPIと同じく同期シリアル通信で、SDA(シリアルデータ)とSCL(シリアルクロック)の2本の信号線を使います。I²CはSPIと同様に、1つのバスに複数のデバイスを接続して通信できます。SPIはSSという信号を使って通信相手を選択しますが、I²Cはそれぞれのデバイスが持つアドレスを指定して通信をします。ですからデバイスを複数接続しても、追加の信号線を必要としません(**図6**)。

図6には、I²Cの信号線と電源の間を接続する抵抗が2つ書かれています。これはプルアップ抵抗というもので、信号線に何もデータが流れていないときに、信号線をHIGHに保つために取り付けられています。ですので、I²Cバスに接続されているデバイスが情報を送るときには、信号線をLOWにします。このような方法は、オープンドレインと呼ばれています。1つの信号線に複数のデバイ

スが接続されている場合、デバイスAがHIGHを出力しているときにデバイスBがLOWを出力すると、デバイスAからデバイスBに向かって大きな電流が流れてデバイスが破損してしまいます。このようなことを避けるため、複数のデバイスを接続するI²Cバスのデバイスは LOWしか出力せず、LOWを出力しているときにデバイスに流れ込む電流はプルアップ抵抗で制限しています。

ところで、LOWを出力していないとき、デバイスのピンはどうなっているのでしょう。前回、マイコンの出力段にはpチャネルとnチャネルのFETが入っているという話をしました(図7)。pチャネルのFETをONにするとHIGHが出力され、nチャネルのFETをONにするとLOWが出力されます。pチャネルとnチャネルのFETがともにOFFの状態、つまりHIGHもLOWも出力していないときは、「ハイインピーダンス」と呼ばれる、ピンが電源にもGNDにもつながっていない浮いた状態になります(図8)。

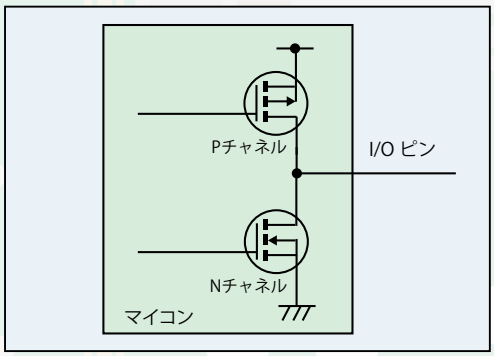
I²CもSPIと同様に、常にマスタが転送を始めます。マスタは、スタートコンディションという信号を出して転送の開始を宣言したあと、スレーブのアドレス(7bit)と転送方向を指定する1bitを送って通信を開始します。受け取ったスレーブは、1bitのACKを返して、送られたデータが有効かどうか返信をします。こうしてI²Cの通信は開始されるのですが、I²Cのプロトコルを説明するには誌面が足りませんので、ここでは省略したいと思います。興味を持った方は、NXPの「I²Cバス仕様およびユーザーマニュアル注2」を参照してください。I²Cは機能が豊富なためか、規格を正しくサポートしていないデバイスが多数存在します。こうしたデバイスを使うときには、いろいろと工夫が必要ですので注意が必要です注3。

I²Cにはいくつか規格があり、それぞれSCLのクロック周波数の上限が異なります。最も古いスタンダードモードでは100kHz、次に出たファストモードで400kHzです。1MHz、3.4MHz、5MHzといったモードもありますが、SPIと比べるとやはり低速な通信です。しかしI²Cは、シンプルで製造コストを抑えることができることもあり、パソコンやサーバ、携帯電話などさまざまな機器の内部で使われています。温度センサやRTC(リアルタイムクロック)などは代表的な用途です。

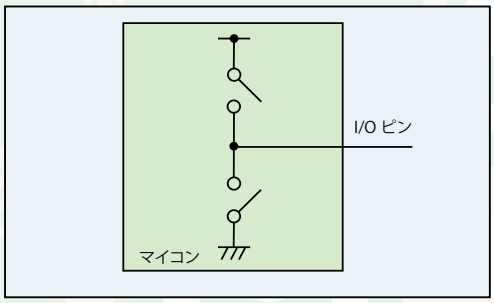
まとめ

今回は、マイコンに周辺デバイスを接続するときに使う代表的なプロトコルをざっと紹介しました。実際には、使用したいプロトコルからデバイスを選択するのではなく、使用したいデバイスを選び、そのデバイスがサポートしているプロトコルを使うことがほとんどでしょう。次回からは、実際にこれらのプロトコルを使って、マイコンとデバイスを接続していきたいと思います。SD

▼図7　マイコンの出力段



▼図8　ハイインピーダンス



注2)　http://www.nxp.com/documents/user_manual/UM10204_JA.pdf

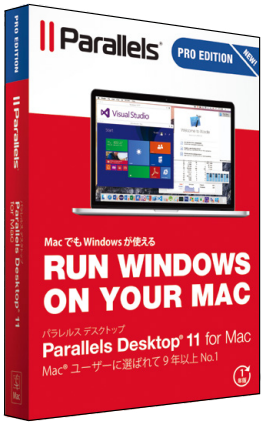
注3)　たとえば、Raspberry Piに搭載されているマイコンのBCM2835は、I²Cの実装にバグがあることが有名です。

読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」にアクセスし、アンケートにご協力ください（アンケートに回答するにはgihyo.jpへのお名前と住所のアカウント登録が必要となります）。ご希望のプレゼント番号を入力いただいた方の中から抽選でプレゼントを差し上げます。締め切りは**2015年11月17日**です。プレゼントの発送まで日数がかかる場合がありますが、ご了承ください。

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

01 Parallels Desktop 11 for Mac Pro Edition（1年版）



2名

Mac OS X用仮想化ソフトの定番がWindows 10対応で新しくなりました。動作環境として、また実行する仮想マシンとして、OS X El Capitanにも対応しています。「Pro Edition」は、Vagrantなどの開発ツールに対応し、仮想マシンに割り当てられる仮想RAMが64GB、仮想CPUが16CPUとなっています。1年を越えて使用する場合は別途、年額の利用料が必要です。

提供元 パラレルス　http://www.parallels.com/jp

02 東芝1号機ものがたりII & 12digit premium calculator

2名

東芝が開発した国産第1号／世界初の電気製品を、当時の時代背景や開発秘話を織り混ぜながら紹介した本（非売品）。税込／税抜計算が簡単にでき、税率設定機能も付いたSHARP製の高性能計算機。2つをセットにしてプレゼントします。

提供元 東芝　http://www.toshiba.co.jp
シャープ　http://www.sharp.co.jp

03 GitHub Tシャツ

1名

今年、新しく日本法人も発足したGitHub社。エンジニアに人気のリポジトリサービス「GitHub」を提供しています。今回は、GitHubのマスコット「Octocat（海賊バージョン）」が描かれたTシャツ（Mサイズ）をプレゼントします。

提供元 ギットハブ・ジャパン　http://github.co.jp

04 入社1年目からの「Web技術」がわかる本

濱 勝巳 著

ネットワーク、サーバ、ソフトウェアと技術分野が多岐にわたるWebアプリ開発について、「全体像と基本知識を習得したい」「しくみについて理解したい」という人に向けて、幅広い情報を提供した1冊です。

提供元 翔泳社
http://www.shoeisha.co.jp

2名

05 初めてのSpark

Holden Karau ほか 著

分散処理基盤「Apache Spark」について、RDD（抽象データセット）を使ったプログラミング、キー／値ペアの処理など基礎的な説明から、クラスタ上での本格的な利用まで解説した、Sparkの入門書です。

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

2名

06 たのしいインフラの歩き方

齊藤 雄介 著

技術ブログで有名な外道父こと齊藤氏によるインフラの本。ネットワークの知識、最新のクラウド活用法から、オフィス移転やコスト削減などに対処するための考え方など、実践的な知識を詰め込んだ1冊です。

提供元 技術評論社
http://gihyo.jp

2名

07 データサイエンティスト養成読本 機械学習入門編

比戸 将平 ほか 著

機械学習を利用するのに必要なアルゴリズム、プログラミングの方法、ビジネスでの活用例をまとめています。機械学習のトップランナー10人によるムック本です。

提供元 技術評論社
http://gihyo.jp

2名

第1特集

そろそろ押さえておきたいWeb最新技術

すいすいわかるHTTP/2

HTTP/1.1から変わること・変わらないこと

Webサービスはどんどんリッチに、ますます便利になり、そのぶん通信データは増大しています。そんな状況でも快適なパフォーマンスを実現するために、HTTP/2は策定されました。すいすい動くWebサービスを作れるようになるために、本特集でHTTP/2の要点をすいすい理解しましょう。

CONTENTS

第1特集

第1章

HTTP/1.1のしくみと限界、そしてHTTP/2へ

Author 川田 寛(かわだ ひろし)／Twitter @_furoshiki　東京Webパフォーマンス
Author 後藤 ゆき(ごとう ゆき)／Twitter @flano_yuki　http2study

Webサービスのコンテンツは日々リッチになっていますが、サービス提供側は「表示が遅い」といったストレスをユーザに与えない工夫をする必要があります。HTTP/1.1ではさまざまな制約のもと、アプリケーションレベルで対策する必要がありましたが、今回登場したHTTP/2は、プロトコルレベルで対策を行います。

インターネットはなぜ遅い?

インターネットは遅い。非効率なことの集まりです。これは、インターネットを支える技術が生まれた経緯、普及した理由を考えれば必然のことと言えるでしょう。

IPのメリット／デメリット

インターネットの基礎を作っているのは、IP(インターネット・プロトコル)と呼ばれる技術です。IPでもっとも評価される点は、あらゆる通信のしくみ／端末を柔軟に取り込めるようにしたことです。これらの技術なくしてインターネットはここまで大きく発展していなかったと言っても過言ではありません。IPでは、どんな通信技術が使われていても、どんな端末がつながっていても、そのネットワークはほかのネットワークとゲートウェイを通じて相互に接続できます。障害が発生しても、迂回するためのしくみを組み込めます。この柔軟さが、ネットワークをここまで大きくスケールさせ、インターネットを世界で相互接続させる巨大なネットワークへと変えました。

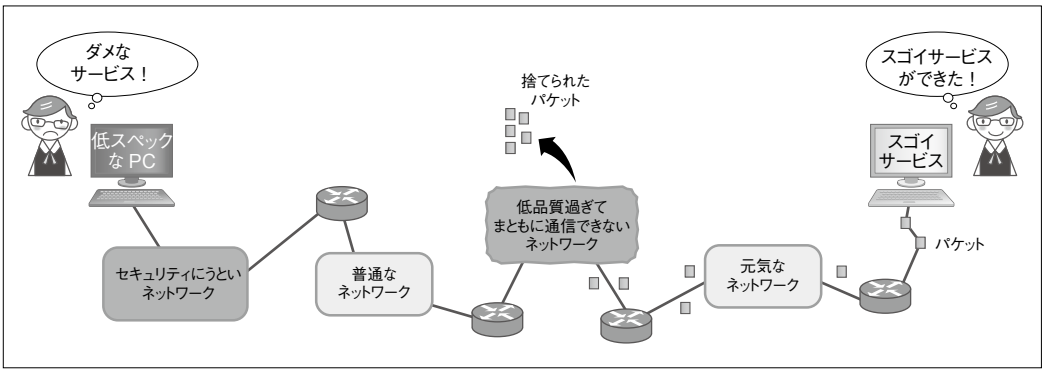
しかし、IPにもデメリットはあります。利用者がFacebookやTwitterのような第三者によって運営されるサービスへ接続するとき、問題のあるネットワークを経由すると、それがサービス自体の品質へダイレクトに影響を与えてしまいます。また、サービス提供者側がどんなに優れた通信技術をネットワーク内に取り込んでいたとしても、パブリッククラウドを使えば足回り(アクセス回線)の帯域の不足に悩まされます。さらに、カフェなどで無料Wi-Fiスポットやモバイルルータを使ってアクセスすれば、セキュリティに悩まされることになるでしょう。

ひと昔前まで、このような大規模なネットワークはNTTのような企業が1社ですべてを賄っていました。品質を阻害するさまざまな問題に対して、1社だけで対処できました。しかしインターネットは、何千、何万という企業がネットワークの運用に関わっているため、どこか1社がまとめて品質を担保できるものではありません。Webサービス提供者は、どこの誰が作ったのかもわからないようなネットワークを使わざるを得ないわけです。一部の人がどれだけがんばり、どれだけ優れたネットワークを作ったとしても、通信の経路全体でみれば品質が悪くなるようなことが起こったりします(**図1**)。

上位レイヤで解決を図る

こうした中、サービス提供者側でもカバーできるようなネットワークのレイヤ、つまりTCPより上のレイヤでカバーしていこうという動きが出てきました。セキュリティ面では

▼図1　一部が全体に影響を及ぼしてしまうIPのしくみ



SSL/TLSが代表的です。このプロトコルは、IPが想定していないような機密性、真正性、完全性などの要求をカバーしてセキュリティを高めてくれます。近年では、Googleがすべてのwebサイトでtlsを適用するよう推進しています。IPの問題点を上位レイヤで塗り替えようとしているのです。

一方でパフォーマンスについても、上位レイヤでカバーしようという動きが生じています。それはセキュリティのSSL/TLSほどしっかりしたものではありません。HTTP/1.1時代に発明された方法はどれも泥臭いものですが、ほかに選択肢があったわけでもないので結果として広く活用されることになります。

HTTP/1.1のしくみと問題点

そもそも、HTTP/1.1がどのようなしくみで動作しているのか、要点を絞ってお話しておきましょう。

大量のリソースを必要とする現在のWebサイト

ブラウザはHTMLドキュメントを読み込み、Webページを表示し、完成させるまでのあいだに大量のリソースを読み込みます。リソースとは、HTMLドキュメント、画像、CSS/JavaScriptなど、URIによって一意に識別できるファイルと考えると良いでしょう。

ハイパーリンクやブックマーク、アドレスバーといったさまざまな個所からHTMLドキュメントのURLが指定されると、ブラウザはWebページの読み込みを開始します。HTMLドキュメントの内容を逐次読み込み、リソースのURLを判別し、ブラウザ内のキャッシュやサーバからリソースを取得してWebページを完成させていきます。こうしてWebページを完成させるまでに必要となるリソースの数は、日々増加しています。3桁台はもはや当たり前で、某有名ECサイトですと、ナビゲーションが完了するまでに実に800以上のリソースを取得していたりします(**図2**)。

リソースをサーバから取得する場合、サーバとブラウザの間をTCPによって接続します。TCP接続の初期フェーズは、パフォーマンスが良くありません。TLSの鍵交換や、スリーウェイハンドシェイクといったRTT(Round Trip Time：パケットの往復時間)に強く依存する手続きが生じたり、任意ではありますが、スロースタートと呼ばれる通信が最適化されるまでにそこそこの時間を要するようなアルゴリズムが使われたりしています。

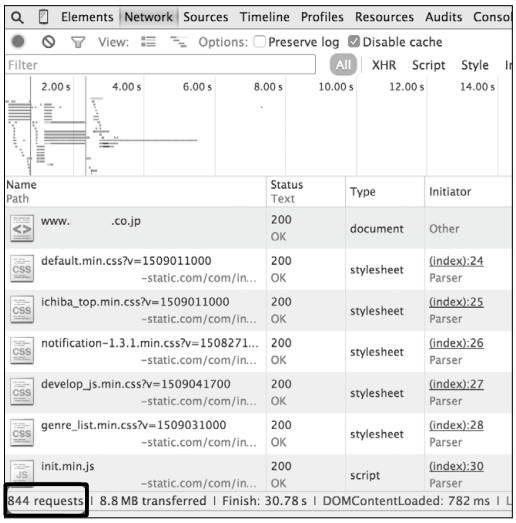
1つの接続で同時に取得できるリソースは1つだけ

これを改善しようと、多くのWebサイトでは「KeepAlive」と呼ばれるフラグをONにします。これにより、1つの接続で複数のリソース

を取得できるようにして、TCPを最適に扱おうと試みます。1つの接続を使い回し続ける、という表現が良いかもしれません。

ただ、このKeepAliveで扱われる接続は、同時に1つのリソースしか読み込みが行えません。複数のリソースを取得する場合には、1つずつ順番に読み込んでいくことになります。極端な話をすれば、応答が遅いリソースがあった場合、ネットワークの帯域が空いているにもかかわらず使われない、といったことがあるわけです。データサイズの大きなリソースの読み込みが求められると、並列にしたほうが効率的なことがよくあります。

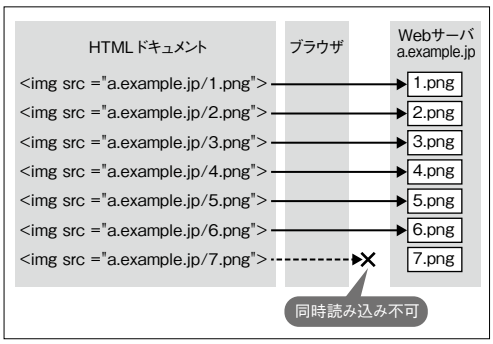
▼図2　1つのWebサイトを表示するのに844のリソースが必要な例も



複数の接続で並列的に取得、でも……

そこでHTTP/1.1では、「1つのオリジン（コラム『オリジン』を参照）に対する最大接続数は2つまで」というルールを作りました。しかし、このルールは実態として守られず、6つ接続するのがデファクトスタンダードになっています。おかげで、1つのオリジンに対して同時に6つのリソースを読み込めるようになりました。しかし、それでも7つめのリソースからは同時に読み込みができず、待ち状態が発生してしまいます（**図3**）。

▼図3　6つの接続により6つまでなら同時にリソースを読み込める



COLUMN　オリジン

オリジンとはuri-schemeとuri-hostとuri-portの3つで構成される情報です。たとえば、次のURLはどれもuri-scheme＝http、uri-host＝example.com、uri-port＝80であり、同一のオリジンです。

```
http://example.com/
http://example.com:80/
http://example.com/path/file
```

構成要素のうち1つでも異なれば、別のオリジンになります。たとえば、次のURLはどれも異なるオリジンです。

```
http://example.com/
http://example.com:8080/
http://www.example.com/
https://example.com:80/
```

HTTP/1.1下での試行錯誤

90年代の末に決まった先述のルールは、今でも現役としてWebを支えています。しかし時代は変わり、ネットワーク環境も変わり、ビジネス環境も変わり、リソース読み込みに対する要求も大きく変わりました。こうした中、今あるルールの中でなんとか対策しようという、とてもとても泥臭いアイデアが発明されていくことになりました。そのいくつかの方法について、簡単にはなりますが紹介しておきましょう。

リソースの結合

繰り返しますが、HTTP/1.1では同一オリジンに対して同時に送信できるHTTPリクエストの数が制限されており、7つめのHTTPリクエストは、先に送信したHTTPリクエストのレスポンスが返ってくるまで待ち状態になってしまいます。また、HTTPリクエストを送信するたびにTCPのスリーウェイハンドシェイクを行うためオーバーヘッドがあります。ですので、HTTP/1.1では、HTTPリクエストの数を減らすことが求められています。

CSSスプライト

複数の画像を1つに結合し、ブラウザ側でCSSによって分割することでHTTPリクエストの数を減らすことができます。これは「CSSスプライト」と呼ばれ、広く使われています。

図4の例では、使用する画像を1枚の画像ファイルに結合しておき、個別の画像を表示する際はCSSのbackground-positionプロパティを使用して、使いたい部分を抜き出しています。これにより、本当なら4回あったはずのHTTPリクエストが1回に節約されています。

しかし、CSSスプライトでは複数のページで同じ画像を使いまわすような場合に、ページによっては使用しない部分があってもダウンロードせざるを得なくなってしまいます。

▼図4　CSSスプライト

通信オーバーヘッドこそ小さくはなりますが、データサイズで見ればけっして効率的にはならないことが多い対策です。

JavaScript／CSSを結合

結合できるのは画像ファイルに限った話ではありません。jQueryの登場で、大量のプラグインを読み込む羽目になったJavaScriptやCSSにも、同様の対策が求められています。JavaScriptやCSSの読み込みは、パフォーマンスが悪いとブロッキングという現象が生じるため、Webページ読み込みのUX(User Experience)を著しく低下させます。JavaScriptやCSSは、複数のファイルを1つにまとめて(concatして)もコンパイルはできるので、TCPのオーバーヘッドやコネクションの空き待ちによるブロッキングを緩和します。近年は、GruntやGulpといったNode.jsベースの開発ツールや、Ruby on RailsのSprocketsのようなJavaScriptをまとめるためのツールがそろい、ほとんどの環境でビルドプロセスに組み込めるようになりました。運用では、もはやそのことを意識しなくても良いのがあたりまえです。しかし、本来のWebの使われ方としては想定されていない、泥臭い対策であることに変わりはありません。

インライン化

HTTPリクエストのオーバーヘッドが問題となり、Webページの表示までに多くの時間を要するケースでは、対策としてインライン化を採用することがあります。インライン化とは、画像、CSS、JavaScriptをHTMLドキュ

メントに埋め込むことでHTTPリクエストをなくす、という対策です。

HTTPリクエストをなくしたいわけ

インライン化の詳細を説明する前に、HTTPリクエストのオーバーヘッドについて考えてみましょう。

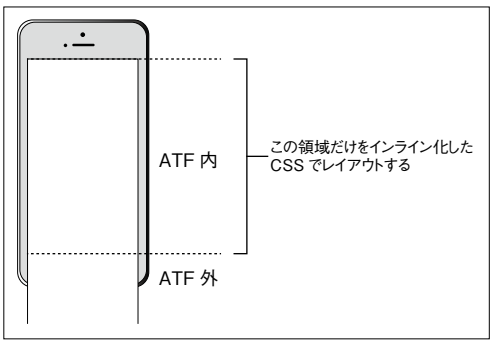
たとえば、サーバからリソースを取得するまでにオーバーヘッドとして3秒が必要だったとします。ほとんどの場合、Webページの描画を開始するまでに、CSSやJavaScriptといった描画をブロックするようなリソースを読み込まなくてはいけないのですが、前項で説明した方法でいくら結合しようと必ず1つは読み込まなくてはいません。

JavaScriptファイルはHTMLドキュメントの末端に配置できることがほとんどです。しかし、CSSファイルの場合は、スタイルの当たっていない要素に対して、あとでCSSが適用されることで画面がフラッシュを起こす「FOUC(Flash Of Unstyled Content)」という現象を引き起こします。この技術的制約により、CSSは必ずと言っていいほどHTMLドキュメントの序盤で読み込まれます。そうなると、このユーザは最低でも6秒間も待たされることになります。

ATFをすばやく表示させるために

Webサービスにおいて、パフォーマンスはコンバージョン[注1]にダイレクトに響いてきます。とくにモバイルにおけるATF(Above the fold)[注2]のスピードの速さは重要で、ここだけはCSSやJavaScriptのようなブロックを引き起こすリソースをサーバから読み込むことなく、Webページのレイアウトを完成させることが求められます(**図5**)。Googleもこのやり方を推奨しており、かつてWeb標準もこのしくみを正式に採用しようと試みたことがあります。最近ではATF内をレイアウトするためのCSSプロパティだけを抜き取るGrunt-Critical-CSSといったツールが、ちょっとした注目を集めました。

▼図5 ATFをすばやく表示させる



インライン化の方法と欠点

具体的な方法についても触れておきましょう。CSSやJavaScriptはstyleタグ／scriptタグを用いることでHTMLドキュメント内に直接記述できます。これは多くの場合、圧縮済みの状態で挿入されます。画像についてはdata:URLスキームを指定し、続けて画像データをBase64化した文字列を記述することで画像データをHTMLドキュメントに埋め込むことができます(**リスト1**)。

インライン化の欠点としては、キャッシュがうまく活用できなくなることです。HTMLドキュメント内に直接データを埋め込むため、これらのリソースをキャッシュすることはできず、毎回読み込むことになります。また、画像はBase64化するためにデータサイズが増えてしまいます。これらのデメリットをふまえたうえで対策すべきかを考えなくてはいけません。

ドメインシャーディング

ドメインシャーディングとは、リソースを提供するオリジンを複数にすることで、ブラ

注1) 商品購入や会員登録など、サービス提供者が想定する成果。
注2) Webページ読み込み時に最初にスクリーンに表示される領域。

▼リスト1　インライン化の例

```
<!DOCTYPE html>
<html>
 CSSのインライン化
  <head>
    <style type="text/css">
      img { box-shadow: 0px 0px 20px -5px rgba(0, 0, 0, 0.8);}
    </style>
  </head>

 スクリプトのインライン化
  <script type="javascript/txt">
    alert("this is inlined");
  </script>

 画像のインライン化
  <img src="
AxA4AAAAAAAAAAAAA////AA==">

</html>
```

ウザが一度に送信するHTTPリクエストの数を増やす手法です。たとえばホスト名を変更すれば、ドメインが別のものとして扱われるため、6つ以上のHTTPリクエストを同時に処理できるようになります。

▼図6　ドメインシャーディングにより、7つ以上のリソースも同時読み込み可能

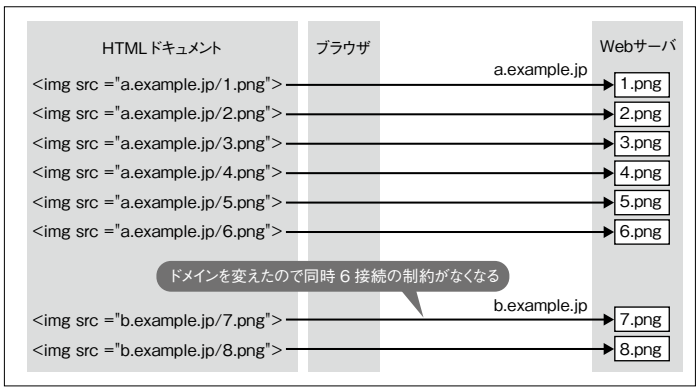


図6のように、a.example.comとb.example.comから別々にリソースを取得する場合、a.example.comに6つまでのHTTPリクエストを、b.example.comに7つめ以降のHTTPリクエストを処理させることで、同時接続数の制約を回避しています。1つの物理サーバに複数のドメイン名を割り当てることができるので、a.example.comとb.example.comは同一のサーバでも問題ありません。

クライアントからすると、同時に取得できるリソースは増えますが、その分名前解決を行う回数が増加します。Webサイトを管理する側としては、複数のオリジンを扱うので煩雑さが増加してしまいます。

HTTP/2の登場、その背景

繰り返しますが、サービス提供者側はインターネットの通信品質に踏み込んで制御できません。ネットワークがさまざまな人々の管理下で運営される以上、ここまで紹介した数々の泥臭い対策は、永遠と議論が繰り返され、泥臭いままの姿で進化し続けることになります。しかし、インターネットをビジネスの主戦場とするGoogleは、これを大きな課題ととらえていたようです。

SPDYからHTTP/2.0へ

Googleが他社と違ったのは、彼らがブラウザの開発にまで手を広げたことです。これはつまり、HTTPそのものを書き換えることができることを意味します。彼らは本章で取り上げたような泥臭い対策を、アプリケーションのレベルよりも下位であるHTTPのレベルに移すことで、パフォーマンス対策を合理化することにしました。2009年、Googleが同社のブログにて発表したプロトコル「SPDY」がこの流れの始まりです。

HTTPの改定／仕様策定を行っていたIETF（Internet Engineering Task Force：インターネット技術の仕様を策定する組織）のHTTPbisワーキンググループでは2012年8月ごろ、HTTP/1.1のセマンティクスを維持したままパフォーマンスとセキュリティを向上することを目的としたHTTP/2の議論が開始されました。

当初は「HTTP/2.0」と呼ばれ模索が始められたこのプロトコルは、さまざまな可能性をふまえたうえでいくつかの考えが提案されました。その中で、すでに実装が存在し、2009年から運用された実績を持つSPDYが、HTTP/2.0の開始点として選ばれました。2012年11月ごろ公開されたHTTP/2.0の最初の草案（draft-00）は、SPDY/3の完全なコピーです。

標準化と、現在のブラウザ／ミドルウェアの対応状況

そのあと、多くの議論が重ねられ、draftも17版まで公開されました。2015年5月、ついにRFC 7540としてHTTP/2が仕様策定されました。執筆時点では、すでに大手Webサービス（Google、Twitterなど）や主要ブラウザ（Google Chrome version 42以降、Mozilla Firefox version 38以降、Safari 9以降、Microsoft Edgeなど）はHTTP/2に対応しています。

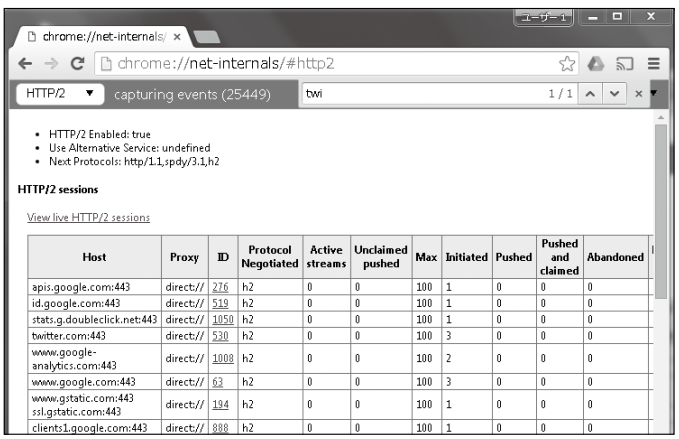
ChromeではURLバーに「chrome://net-internals/#http2」と入力することで現在HTTP/2で接続しているサーバを確認できます（**図7**）。別途新しくタブを開いてFacebookやTwitterにアクセスしてみてください。そのときHTTP/2でアクセスしていれば、その情報が表に追加されるのが確認できるはずです。Protocol Negotiatedが「h2」と表示されているものがHTTP/2で通信しているホストになります。表中のIDのリンクをクリックすることで通信の中でどのようなメッセージが交換されているかも確認できます。

FirefoxではDeveroperToolsのNetworkタブより、各HTTPリクエストの詳細画面を確認すれば「Version:HTTP/2.0」と確認できます（**図8**）。

HTTP/2のライブラリはC言語、Java、Node.js、Haskell、Rubyと、さまざまな言語で実装されています。実装リストについては、GitHub上のhttp2-specリポジトリのWikiページ[注3]より確認できます。

既存のミドルウェアの対応状況は、Apache Traffic Serverが対応しているほか、Apache 2のモジュールとしてmod_h2が、Nginx1.9.5で、HTTP/2のモジュールがそれぞれ組み込まれており、

▼図7　Chromeで、HTTP/2接続しているサーバを確認



注3）URL https://github.com/http2/http2-spec/wiki/Implementations

COLUMN HTTP/2関連そのほかのツール

既存のミドルウェア／ツールがHTTP/2に対応していますが、新しく便利なツールもいくつかあります。簡単にではありますが、以下に紹介します。

- **h2i**[注A]
  HTTP/2のインタラクティブコンソールデバッガツール。サーバからの返信を確認しながらPINGフレーム、SETTINGSフレーム、HEADERSフレームなどを手入力で送信することができる
- **h2load**[注B]
  第3章で紹介する「nghttp2」に含まれるHTTP/2で負荷を生成するツール。自身のサーバに対して負荷テストをする際に非常に有用
- **h2spec**[注C]
  サーバの実装テストツール。サーバがHTTP/2の仕様どおりに実装されているかを確認できる。自身でサーバを実装する際に、確認するのに有用
- **HTTP/2 and SPDY indicator**[注D]
  ブラウザのプラグイン／拡張。現在アクセスしているページにおいて、HTTP/2もしくはSPDYで通信している場合に、稲妻マークが表示される

注A）URL https://github.com/bradfitz/http2/tree/master/h2i
注B）URL https://github.com/tatsuhiro-t/nghttp2
注C）URL https://github.com/summerwind/h2spec
注D）URL https://addons.mozilla.org/ja/firefox/addon/spdy-indicator/
URL https://chrome.google.com/webstore/detail/http2-and-spdy-indicator/mpbpobfflnpcgagjijhmgnchggcjblin?hl=ja

COLUMN HTTP/2と日本コミュニティ

HTTP/2のRFCであるRFC 7540の一番最後に謝辞が書かれています。その中で日本コミュニティについて述べられています。

> The Japanese HTTP/2 community provided invaluable contributions, including a number of implementations as well as numerous technical and editorial contributions.

日本の方による、HTTP/2実装やHTTPbis WGのメーリングリストでの議論もたくさんありました。それらの活動の結果、こうしてRFCの謝辞に載っています。

また日本では有志での勉強会／ハッカソンが2013年8月ごろより活発に行われてきました。中にはissue-thonといったHTTPbis WGで議論されているissueについて情報交換をするハッカソンなども行われました。それら勉強会の様子はIETFでも報告されています[注E]。

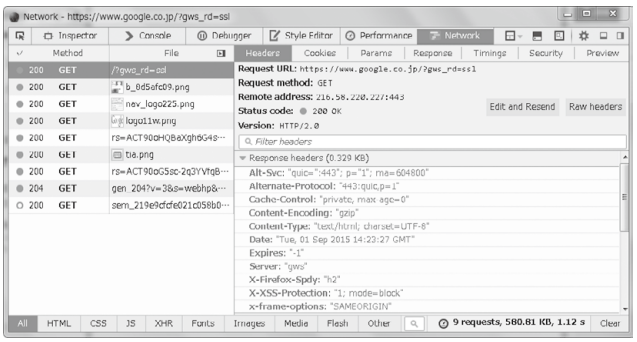
国内の過去のイベント情報についてはhttp://http2.info/をご覧ください。

注E）URL https://www.ietf.org/proceedings/89/slides/slides-89-httpbis-2.pdf
URL https://www.ietf.org/proceedings/91/slides/slides-91-httpbis-1.pdf

実験的に既存のインフラに導入できる状況となっています。AkamaiといったCDN（Contens Delivery Network）でも対応が進められています。また、CurlやWiresharkといったツール類もHTTP/2対応が済んでいます。

HTTP/2はまさに今、適用のフェーズへと移ろうとしているのです。SD

▼図8　Firefoxで、HTTP/2の接続状況を確認



第1特集

第2章

図解&例題でわかるHTTP/2プロトコル

Author 後藤 ゆき(ごとう ゆき)／Twitter @flano_yuki　http2study

パフォーマンスの課題を解決するために、HTTP/2では「ストリーム」「フレーム」「ヘッダ圧縮」など、さまざまな機能が取り入れられています。それでいて、従来のWebアプリケーションは変更しなくても動作するような配慮もなされています。HTTP/2で何が変わり、何が変わらないのか、本章で整理します。

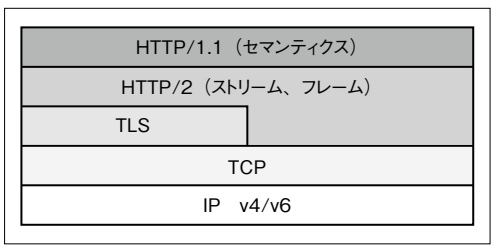
HTTP/2の概要

HTTP/2は、HTTP/1.1のセマンティクスを維持したままパフォーマンスとセキュリティを向上させています。

大きな特徴としては、HTTP/1.1のメッセージを効率良く転送するため、新たに「ストリーム」という概念と、「フレーム」というメッセージを導入しています。HTTP/1.1のようにHTTPリクエスト／HTTPレスポンスを伝えるというしくみ自体には変わりはなく、その伝え方に変化があります。HTTP/2はバイナリのプロトコルであり、その変化の大きさに驚かれた方も多いようです。しかし、先ほど挙げた2つの新しい概念の上で、今までどおりにHTTP/1.1のメッセージをやりとりしているとイメージすると理解しやすいでしょう。

図1はプロトコルスタックのイメージ図です。

▼図1　HTTP/2のプロトコルスタック

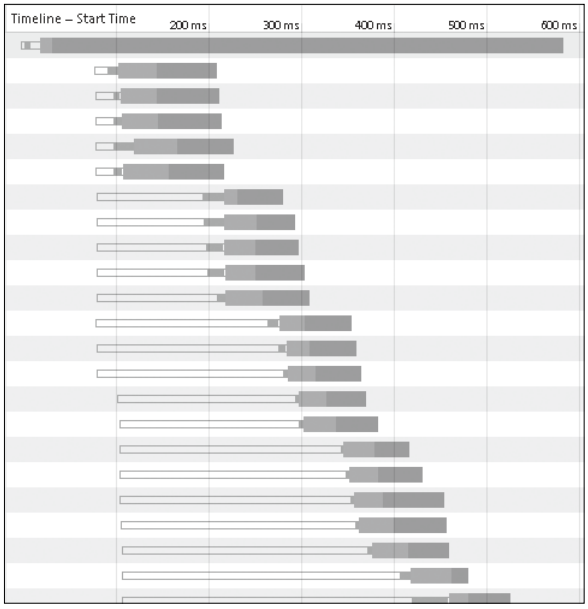


・HTTP/1.1と同様、URLで「http://」が指定されるとTCPにより平文で通信し、「https://」が指定されるとセキュアなTLSを用いて通信する。TCPとTLSの上にHTTP/2が載るような形をしている
・HTTP/2の上で、今までどおりのHTTP/1.1相当のメッセージが扱われる。HTTP/1.1では「User-Agent:〜」や「Referer:〜」といったメッセージをテキストで扱っていたが、HTTP/2ではこれらがすべてバイナリになる。ただし「意味」としては、今までのHTTP/1.1相当のHTTPリクエスト／HTTPレスポンスのメッセージとして扱える

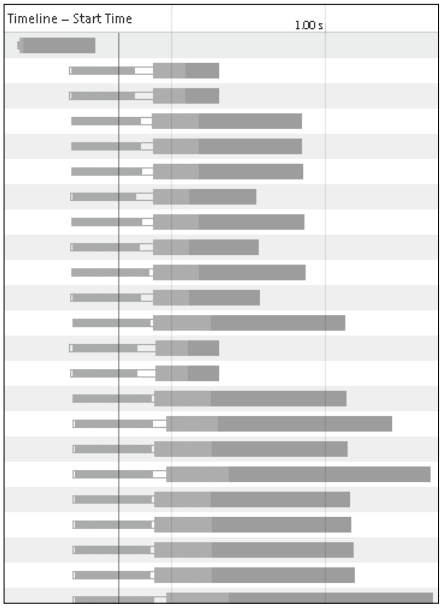
セマンティクスの維持

HTTP/1.1で使用していたGET、POST、PUTといったメソッドやUser-Agent、Cookieといったヘッダは、HTTP/2においても今までどおり使用されます。HTTP/2を通信するサーバやブラウザは、もちろんHTTP/2に対応している必要があります。しかし、サーバやブラウザがHTTP/1.1相当のメッセージを取り出し、上位のアプリケーションに渡す形で処理してくれるため、サーバの上で実行されるWebアプリケーションや、ブラウザ上で実行されるJavaScript側は、とくに変更なくそのまま使えるという利点があります。これが最初にも述べた「セマンティクスの維持」です。

▼図2　HTTP/1.1の場合



▼図3　HTTP/2の場合

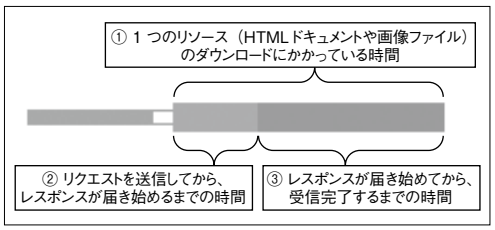


改善点

先述のとおり、HTTP/2ではHTTP/1.1のメッセージを効率良くやりとりするため、ストリームという新しい概念を導入しています。ストリームにより、HTTPのメッセージは多重化されて送受信されます。HTTP/1.1では、同一オリジンのHTTPリクエストの並列数は制限されていましたが、HTTP/2ではHTTPリクエストの並列数はクライアントとサーバの設定次第になります[注1]。多重化されることで、HTTP/1.1のKeepAliveであったような、特定のHTTPレスポンスの処理に専有され、ほかの処理が待ち状態になってしまうようなことは起こりません。

この多重化の様子は、Webページを表示する際に画像、CSS、JavaScriptファイルをどのようにダウンロードしているかを確認するとわかりやいでしょう。Chromeのデベロッパーツールを用いて、画像の多いサイトを表示する際の様子をHTTP/1.1とHTTP/2で比べてみます(**図2、3**、各グラフの見方は**図4**を参照)。

▼図4　図2、3のグラフの見方



HTTP/1.1は同時に送れるHTTPリクエストの数が6つに制限されているため、後続のダウンロードは先行しているダウンロードが終わるまで待機しています(**図2**)。一方、HTTP/2ではそのようなことはなく、一斉に処理が開始されていることが、図から読み取れるでしょう(**図3**)。

また、HTTP/2では1つのTCPコネクションを極力使いまわすようにしています。「http://」のURLでは同一のIPアドレスの場合に、「https://」のURLでは同一IPアドレスかつ証明書が正当な場合に、既存のTCPコネクションを使

注1) とはいえ、無制限というわけでもなく、仕様上ではストリームの並列数としては100以下に設定しないことが推奨されています。

用してリクエストを送信します。たとえば、証明書のsubjectAltNameが「*.example.com」であれば、「https://a.example.com/」や「https://b.example.com/」へのリクエストはTCPコネクションの再利用が行えます。

このTCPコネクションの再利用は、クライアント単位で可能です。ブラウザはタブやウィンドウが別であっても、条件さえ満たしていれば、1つのTCPコネクションを再利用し続けて使用できます。

このようなしくみが、従来のHTTP/1.1で問題になっていた通信オーバーヘッドの問題を解決してくれます。TCPコネクションを再利用するため、HTTPリクエストのたびにTCPスリーウェイハンドシェイクをする必要がありません。同様に、1つのTCPコネクションの生存時間は長くなるため、スロースタートなTCPでもより通信路を効率良く使用できると言われています。

逆に言えば、HTTP/1.1では当たり前のように行われているドメインシャーディングを、HTTP/2で行うと、コネクションの使い回しができなくなってしまう場合があります。HTTP/1.1に最適だったサーバ構成は、HTTP/2にとっても最適とは言えないのです。

そのほかにも多くの機能があり、簡単に取り上げても次のような特徴があります。

- ヘッダ圧縮：HPACKというしくみでヘッダの圧縮を行う
- 優先処理：クライアントはリクエスト時に優先度を通知できる
- フロー制御：多重化された各ストリームのフロー制御[注2]を行う
- サーバプッシュ：サーバはHTTPリクエストに先行してHTTPレスポンスを行える
- 最終メッセージ通知：コネクション切断時に、切断理由やどのメッセージまで処理を行ったのかを相手に通知する

これ以降、いくつか取り上げて説明していきます。

注2）フロー制御とは、送信者が、受信側が受け取れる以上のデータ量を送信しないようにするしくみのこと。

COLUMN TLSの利用における制限

「https://」から始まるURLにアクセスする際にTLSを利用するのは、HTTP/2でも同様です。ただし、HTTP/2ではTLSの利用に条件があります。

- TLS 1.2以上を使用しなければならない
- SNI（Server Name Indication）拡張に対応していなければならない
- TLSでの圧縮を無効にしなければならない
- HTTP/2通信が始まってからの再ネゴシエーションを無効にしなければならない
- HTTP/2の仕様中で禁止された暗号スイート（RC4などのような脆弱性が指摘されている暗号など）を使用してはいけない

などの条件があります。これらの条件が満たされていない場合は、INADEQUATE_SECURITYエラーとしてコネクションをただちに切断しても良いとされています。Chromeではこれらの条件が満たされない場合、「ERR_SPDY_INADEQUATE_TRANSPORT_SECURITY」というエラー画面になり、Webサイトを表示することはできません。

これらの圧縮の無効化や再ネゴシエーションの無効化といった条件の多くは、TLS 1.3ではそもそも使用できないものが多く、HTTP/2はTLS 1.3を先取りするような形になっています。

実際にHTTP/2サーバを動作させる場合は、OpenSSLなどの暗号ライブラリのバージョンを確認する必要があります。バージョンが低い場合は、ALPN（後述）をサポートしていなかったり、許可されている暗号スイートが使用できなかったりする場合があります。試そうとした際に、うまくいかない原因となり得るので注意が必要です。

ストリームとフレーム

HTTP/2では、送受信するメッセージはストリームという単位で管理されます。そして、HTTPメッセージはフレームと呼ばれるメッセージ形式で送信されます。これらは重要な概念ですので、順番に説明していきます。

ストリーム

ストリームは通信を管理する単位であり、HTTP/1.1におけるHTTPリクエストとHTTPレスポンスのやりとりが1つのストリームになります。

図5のように1つのTCPコネクションの上で、HTTPリクエストとそれに対するHTTPレスポンスが1つのストリームという単位で送受信されるイメージです。各HTTPリクエストとそれに対するHTTPレスポンスはストリーム単位で並列的に送受信され、準備できたものから送信されます。HTTP/1.1のように、同時に送信できるHTTPリクエストの数に制限があったり、ほかのレスポンスの処理が終わるのを待ったりすることはありません。

ストリームはHTTPリクエストを送信する都度、新しく生成されます。各ストリームはストリームIDという識別子にて、重複しないよう管理されます。ストリームID：0は特殊で、特定のストリームではなく、コネクションそのものを指します。たとえば、HTTP/2にはコネクションの生存を確認するPINGフレームというメッセージがありますが、これはストリームID：0で送信されます。

また、ストリームは再利用ができません。1回のリクエスト／レスポンスが終了したら、そのストリームは使用することができなくなります。

このようなストリームのしくみが、TCPコネクションの切断を不要にしています。HTTPリクエストを送信するときには、すでに使用しているTCPコネクションがあればそれを再利用し、ストリームを生成します。

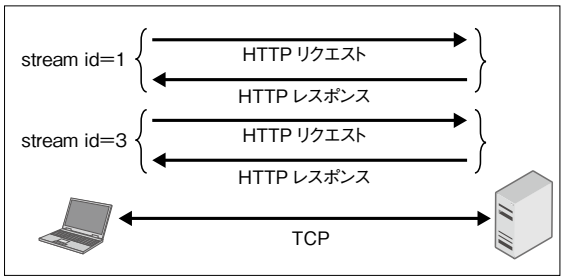
フレーム

フレームはHTTP/2におけるメッセージの形式です。

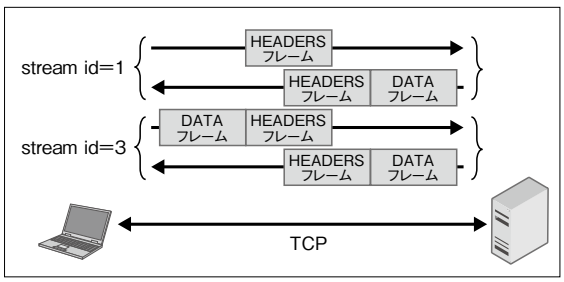
ストリーム上でHTTPリクエスト／HTTPレスポンスがやりとりされると述べましたが、正確に言えばフレームというバイナリ形式のメッセージが送受信されます。HTTPリクエストやHTTPレスポンスは、HEADERSフレームとDATAフレームにより表現されます。HEADERSフレームにはHTTPヘッダが格納されており、DATAフレームにはHTTPリクエストにおけるPOSTデータや、HTTPレスポンスにおけるHTMLファイルや画像データが格納されています。これをイメージ図にすると、**図6**のようになります。

理解を深めるために、ChromeがHTTP/2

▼図5　ストリームのイメージ（1つのTCPコネクション上で並列的に通信が行われる）



▼図6　フレームのイメージ（ストリーム上ではフレームがやりとりされる）



で通信している様子を確認してみることにしましょう。第1章で述べたように、ChromeのURLバーに「chrome://net-internals/#http2」と入力し、IDの欄にあるリンクより通信の中身を見てみます。

HTTP/2はさまざまなメッセージを扱いますが、**図7**では、一部だけを抜粋して取り上げています。メッセージ中の細かい表記はのちほど説明しますので、まずはおおまかに、**図7**のサンプルでは3つのメッセージが送受信されているのを確認してみてください。これらは通信時にはバイナリ形式で取り扱われていますが、ツールの支援によって人間に読めるようにテキスト形式で表示されています。

これらは、HTTP/1.1でも当たり前のように行われていた、ブラウザからサーバへのリソース読み込みを要求している様子をモニタリングしたものです。HTMLドキュメントや画像ファイルをサーバから取得していると考えてみてください。

各メッセージの意味を少し説明すると、

①HTTP2_SESSION_SEND_HEADERS
ストリームID：3にて、HEADERSフレームを送信している。この中にはHTTPリクエストヘッダが格納されている

②HTTP2_SESSION_RECV_HEADERS
ストリームID：3にて、HEADERSフレームを受信している。この中にはHTTPレスポンスヘッダが格納されている

③HTTP2_SESSION_RECV_DATA
ストリームID：3にて、DATAフレームを受

▼図7　HTTP/2の通信の中身（一部抜粋）

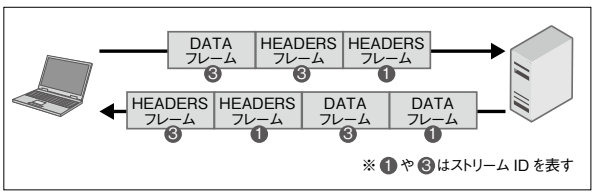
```
t=2741 [st=0]  HTTP2_SESSION_SEND_HEADERS  ←①
               --> fin = true
               --> :authority: 192.168.0.104
                   :method: GET
                   :path: /
                   :scheme: https
                   accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/
webp,*/*;q=0.8
                   accept-encoding: gzip, deflate, sdch
                   accept-language: ja,en;q=0.8
                   cache-control: max-age=0
                   upgrade-insecure-requests: 1
                   user-agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/44.0.2403.157 Safari/537.36
               --> priority = 0
               --> stream_id = 3
               --> unidirectional = false

t=2743 [st=2]  HTTP2_SESSION_RECV_HEADERS  ←②
               --> fin = false
               --> :status: 200
                   accept-ranges: bytes
                   content-length: 5
                   content-type: text/html
                   date: Tue, 01 Sep 2015 15:15:07 GMT
                   etag: "55e5c0e1-5"
                   last-modified: Tue, 01 Sep 2015 15:14:41 GMT
                   server: nginx/1.9.4
               --> stream_id = 3
t=2744 [st=3]  HTTP2_SESSION_RECV_DATA  ←③
               --> fin = false
               --> size = 5
               --> stream_id = 3
(..以下略..)
```

信している。Chromeの画面上では表示されていないが、HTTPレスポンスボディが格納されている

このように、HTTP/2でも1つのストリームで、HTTP/1.1で行われていたようなHTTPリクエスト／HTTPレスポンスが行われます。上記の例はシンプルな1つのリクエスト／レスポンスですが、大きなファイルを扱う場合は、複数のDATAフレームに分割されて送られることもあります。また実際には、HTTPリクエスト／HTTPレスポンスのペアの分だけストリームが生成され、並列的にHTTPリクエスト／HTTPレスポンスがやりとりされます。並列的とは言っても、もちろんTCPコネクション上ではフレームは順々に送られています(**図8**)。

先に紹介したHEADERSフレーム、DATAフレーム以外に、仕様で定義されているフレームは10種類あります。**表1**にて簡単に紹介します。

▼図8　実際のTCPコネクション上での送受信の様子



リクエストの優先度（プライオリティ）

HTTP/1.1では各々のHTTPリクエストは独立しており、それぞれのリクエストに優先度というものはありませんでした。すべてのリクエストは平等に処理されます。しかし現実には、ブラウザはWebサイトを表示するにあたり、レンダリングを開始するのに必要なデータとそうでないデータがあったりと、優先度が違う場合があります。

HTTP/2ではクライアントからサーバにリクエストの優先度を伝えることができます。これにより、「CSSやJavaScriptファイルは優先的に送ってほしい」「画像は後からほしい」というブラウザ側の希望をサーバに伝えることができます。また、アクティブではないタブやウィンドウにおけるHTTPリクエストを低優先度にするといったこともあり得るでしょう。

HTTP/2における優先度とは、割り当てるリソースの割合を意味しています。しかし、IETFの仕様では、リソースという言葉に明確な定義は与えられていなかったりします。サーバの使用するネットワーク帯域や、サーバの計算資源ととらえると良いでしょう。また、サーバは必ずしも、この優先度を守らなければい

▼表1　フレームの種類

Frame type名	Type番号	説明
DATA	0x0	HTTPリクエストのPOSTデータ／アップロードファイル、HTTPレスポンスのボディを転送する
HEADERS	0x1	HTTPリクエスト／HTTPレスポンスのヘッダを転送する
PRIORITY	0x2	特定のストリームの優先度を変更する
RST_STREAM	0x3	エラーの際などにストリームの終了を通知する
SETTINGS	0x4	通信に関するパラメータを通知する
PUSH_PROMISE	0x5	サーバプッシュのためにストリームを予約する
PING	0x6	コネクションの生存を確認する。PINGフレームを受け取った場合は、ACKフラグの付いたPINGフレームを返さなければならない
GOAWAY	0x7	コネクション自体を切断するために、新しいストリームの生成を停止するよう通知する。どのストリームまで処理を行ったかの情報も含まれる
WINDOW_UPDATE	0x8	フロー制御のために使用する
CONTINUATION	0x9	たくさんのHTTPヘッダを送信する際などに、HEADERフレームに継続して送信される

けない、というわけではありません。

あくまでHTTP/2では優先度付きの機能を提供しているだけであり、その使い方は定義していません。クライアント側／サーバ側の戦略や実装により、使用のされ方は大きく違う可能性があります。速くなるのも、遅くなるのも、使っているミドルウェアしだいです。Webサイトの表示速度に与える影響は大きいのでしょうが、同時にそれはこれからどんどん改善していくことでしょう。したがって、Webアプリケーションを作るエンジニアが気にする機会は、そこまで多くはないように思います。

優先度によるリソースの割り当て方

優先度は、クライアントがHTTPリクエストを送信する際のHEADERSフレームに付加できます。明示的に指定しない場合でも、デフォルトの値が設定されます。優先度はストリームごとに管理され、Weight(重み)とStream Dependency(依存関係)により優先度が計算され、割り当てるリソースが決定されます。

- **Weight**
  1～256で表されるストリームの重み。デフォルト値は16
- **Stream Dependency**
  依存するストリームのストリームIDが指定される。依存されているほうのストリームが優先されるようになる。たとえば、リクエストAがリクエストBに依存している場合は、リクエストBの処理が終わってからリクエストAの処理が行われる。デフォルト値は、ストリームID：0(Dependencyを管理するのに使用される存在しないストリーム)

Stream Dependencyによって、ストリームの依存関係を示す木構造ができあがります。この依存関係の木はDependency Treeと呼ばれます。簡単な例で、Dependency Treeがどのように構築され、どのようにリソースが割り当てられるかを確認していきましょう。

例1：

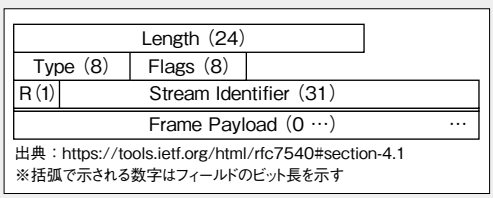
- データAをWeight＝16でリクエスト
  (ストリームID：1)

COLUMN フレームのバイナリフォーマット

フレームはバイナリのメッセージであると紹介しました。簡単にどのようなものか紹介します。

フレームは9バイトの固定長のフレームヘッダ部分と、それに続くペイロード部分に分かれています(**図A**)。

▼図A　フレームのフォーマット



出典：https://tools.ietf.org/html/rfc7540#section-4.1
※括弧で示される数字はフィールドのビット長を示す

● フレームヘッダ

すべてのフレームにある固定長9バイトの領域です。次のフィールドから構成されます。

- Length：ペイロードの長さ
- Type：フレームタイプ
- Flags：フラグ。各種Frame typeごとに用途は異なる
- R：予約済みの1ビット領域。値は0である
- Stream Identifier：ストリームID

● Frame Payload

ペイロードは、各フレームごとに必要なデータが入ります。

- HEADERSフレームであれば、優先度とHPACKで表現されるヘッダ
- DATAフレームであれば、HTTPメッセージのボディ
- SETTINGSフレームであれば、SETTINGSパラメータ

・データBをWeight＝8でリクエスト
（ストリームID：3）
・データCをWeight＝4でリクエスト
（ストリームID：5）

依存するストリームを指定しなかった場合は、ストリームID：0に依存しているとして管理されます(**図9**)。ストリームID：0のストリームは実際には存在せず、ツリーのルートとして使用されます。リソースが割り当てられることはありません。それぞれのリクエストで生成されたストリームはDependency Treeの中で、ストリームID：0の子、つまり兄弟関係になります。兄弟関係にあるストリームには、Weightに比例してリソースが割り当てられます。

図9の兄弟関係にある3つのストリームについて、優先度は次のように決定されます。

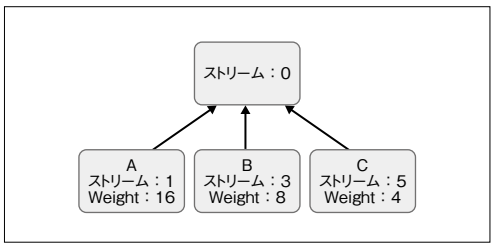
・データA(ストリームID：1)は、
16/(16＋8＋4)＝4/7
のサーバリソースが割り当てられる
・データB(ストリームID：3)は、
8/(16＋8＋4)＝2/7
のサーバリソースが割り当てられる
・データC(ストリームID：5)は、
4/(16＋8＋4)＝1/7
のサーバリソースが割り当てられる

処理が終わったストリームにはサーバリソースは割り当てられませんので、ほかのストリームにサーバリソースは分配されます。

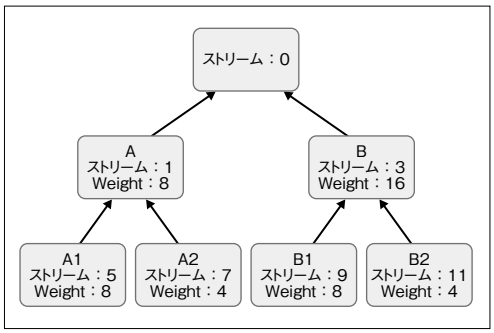
次に、依存関係を含む少々複雑な例を見ていきましょう。データAに依存するA1、A2とデータBに依存するB1、B2という形でリクエストしたとします。

例2：
・データAをWeight＝8でリクエスト
（ストリームID：1）
・データA1をWeight＝8でリクエスト
（ストリームID：5）

▼図9　例1におけるDependency Tree



▼図10　例2におけるDependency Tree



・データA2をWeight＝4でリクエスト
（ストリームID：7）
・データBをWeight＝16でリクエスト
（ストリームID：3）
・データB1をWeight＝8でリクエスト
（ストリームID：9）
・データB2をWeight＝4でリクエスト
（ストリームID：11）

Dependency Treeは**図10**のようになります。

依存関係がある場合、依存されているほうの処理が終わるまで、依存している側へはリソースは割り当てられません。そのため、最初は次のようにデータAとデータBのストリームにリソースが分配されます。

・データA(ストリームID：1)は、
8/(8＋16)＝1/3
のサーバリソースが割り当てられる
・データB(ストリームID：3)は、
16/(8＋16)＝2/3
のサーバリソースが割り当てられる

データBをレスポンスしてデータBのストリームが終わった場合、データBのストリームに依存しているストリームへリソースが割り当てられます。このとき、データBのストリームへ割り当てられていたリソースをB1とB2のストリームで分割します。そのため、次のようにリソースが分配されます

・データA(ストリームID：1)は、
8/(8＋16)＝1/3
のサーバリソースが割り当てられる
・データB(ストリームID：3)に割り当てられていた
16/(8＋16)＝2/3
をB1とB2で分配する
- データB1(ストリームID：9)は、
16/(8＋16)×8/(4＋8)＝4/9
のサーバリソースが割り当てられる
- データB2(ストリームID：11)は、
16/(8＋16)×4/(4＋8)＝2/9
のサーバリソースが割り当てられる

また、ストリームの処理が完了していないのに、そのストリームの処理が進められない場合(ディスクI/Oやネットワーク I/Oによる待ち状態など)は、ほかのストリームにリソースが割り当てられます。なお、一度設定したWeight や Stream Dependency は PRIORITY フレームで変更することも可能です。

このように優先度はWeightとStream Dependencyで構成されるDependency Treeという構造を持ち、さらに処理が進むにつれリソースの配分が変わるため、非常に複雑なしくみになっています。

この優先度付けの効率の良い構成方法は、今後も議論が続けられていくことでしょう。

ヘッダ圧縮(HPACK)

HTTP/1.1では、プロトコル上は状態を持ちません。そのため、クライアントは同じサーバにHTTPリクエストを送信する場合でも、毎回、内容がそこまで大きく変わらないようなHTTPヘッダを送信しなくてはいけませんでした。

たとえば、User-AgentヘッダやAccept-Encodingヘッダといったヘッダは、1つのセッションであれば、さほど内容が変わらないヘッダです。これらのヘッダをすべてのHTTPリクエストで送信することはたいへん冗長です。HTTPレスポンスヘッダでも、いくつかのヘッダには同じことが言えます。

そこでHTTP/2では、ヘッダを圧縮することにしました。HTTP/2の元となったSPDYでは、deflateと呼ばれる圧縮アルゴリズムを用いていましたが、CRIMEと呼ばれる攻撃手法が発見されました。そこでHTTP/2では、deflateとは違った圧縮方法を用いたHPACKという新しいしくみを導入しています。HPACKは、HTTP/2の仕様とは別にRFC 7541として仕様化されています。

HPACKでは大きく分けて2種類の方法でヘッダを圧縮します。

・ハフマン符号化
・テーブル
- 静的テーブル
- 動的テーブル

HTTPヘッダをHEADERSフレームなどで送信する際は、このHPACKのしくみに則ってヘッダを表現します。

ハフマン符号化

ヘッダ名やヘッダ値は文字列を使って表現されます。文字列表現はハフマン符号化して圧縮することも可能です。ハフマン符号化は、出現頻度の偏りに合わせて「文字」と「ビット列」の割り当てを変更することで、データ量を削減します。たとえば、**表2**のように、HTTPヘッダにおいて出現頻度の高い「a」や「c」といった文字にはビット表現が短いもの、出現頻度の低い「!」や「<」といった文字にはビット表現の長

いものが、割り当てられています。

このハフマン符号を用いて「www.example.com」を表現すると**表3**のようになります。

ハフマン符号の対応表はRFC 7541の仕様中で定義されています。クライアントとサーバはあらかじめ同じ対応表を持っているので、ハフマン符号化されたヘッダを受け取っても復号できます。

テーブル

HTTP/2では、テーブルと呼ばれる、頻繁に使用するヘッダの辞書データのようなものを持っており、そのインデックスを指し示すだけでヘッダを表現することができます。

▼表2　ハフマン符号化の例

文字	ビット表現	長さ(ビット)
a	00011	5
b	100011	6
c	00100	5
!	11111110 00	10
<	11111111 1111100	15

静的テーブル

HPACKでは事前に頻出度の高いヘッダを格納した静的テーブルというものを定義しています(**表4**)。たとえば、そこに含まれている次のようなヘッダ

```
accept-encoding:gzip, deflate
```

は、テーブルのindex = 16に格納されており、「16」(1バイトの数値)と指定するだけで表現できます。

静的テーブルは**表4**のように61のエントリが定義されており、リクエストとレスポンスで共通に使用されるテーブルです。

「:」で始まるヘッダは擬似ヘッダと呼ばれ、HTTP/2で使用されるヘッダです。HTTP/1.1においてリクエストラインやステータスラインで表現される値、たとえばメソッド名やステータスコードはHTTP/2では擬似ヘッダとして表現され、HPACKで圧縮することができます。

▼表3　「www.example.com」をハフマン符号化する場合／しない場合

ハフマン符号化	16進数表現	長さ
なし	7777 772e 6578 616d 706c 652e 636f 6d	15バイト
あり	f1e3 c2e5 f23a 6ba0 ab90 f4ff	12バイト

▼表4　HPACKの静的テーブル

インデックス	ヘッダ名	ヘッダ値
1	:authority	
2	:method	GET
3	:method	POST
4	:path	/
5	:path	/index.html
6	:scheme	http
7	:scheme	https
8	:status	200
9	:status	204
10	:status	206
11	:status	304
12	:status	400
13	:status	404

インデックス	ヘッダ名	ヘッダ値
14	:status	500
15	accept-charset	
16	accept-encoding	gzip, deflate
(途中省略)		
53	retry-after	
54	server	
55	set-cookie	
56	strict-transport-security	
57	transfer-encoding	
58	user-agent	
59	vary	
60	via	
61	www-authenticate	

動的テーブル

動的テーブルは、HTTPリクエスト／HTTPレスポンスをやりとりしていく中で使用したヘッダが追加されていくテーブルになります。クライアントやサーバは送信したヘッダを動的テーブルに追加するように指示することができます(ヘッダは静的テーブルの領域の後ろ、インデックス62番から使用されます)。これにより、静的テーブル以外のヘッダもインデックスを指し示すだけで表現できるようになります。ただし、動的テーブルにはサイズ上限があり、上限を超す場合は古いものから順番に消されていきます。

動的テーブルは1つのコネクション中でのみ有効で、再度コネクションした場合は、まっさらな状態からになります。

HPACKの実例

ここまでHPACKの基本的な圧縮のしくみを説明しましたが、ここからは、それらがどのように使用されるかを具体的に説明していきます。HPACKには、1つのヘッダを表現する「ヘッダフィールド表現」がいくつか定義されています(**表5**)。それぞれテーブルのインデックスを使うか、文字列表現を使うか、動的テーブルに追加を行うかといった違いがあります。どれを使うかは、ヘッダの送信者が決めます。

具体的に、例を挙げて説明します。まず次のヘッダリストを送信する場合を考えます。

例A
```
:method: GET
:scheme: http
:path: /
:authority: www.example.com
```

これをHPACKでヘッダ圧縮をすると、それぞれどのように表現できるかを**表6**に示します。このリクエストによって、受信側の動的テーブルに**表7**のエントリが追加されます。

COLUMN 擬似ヘッダ

HTTP/2では擬似ヘッダという、「:」で始まるヘッダが定義されています。これらはほかのHTTPヘッダと同じように扱われますが、HTTP/2でのみ利用できます。

擬似ヘッダはHTTP/1.1におけるリクエストライン／ステータスラインと呼ばれる部分で定義されている値を表すのに使用されます。

HTTP/1.1のリクエストラインの例
```
GET /index.html?top HTTP/1.1
host: example.com
(以降、HTTPヘッダが続く)
```

HTTP/1.1のステータスラインの例
```
HTTP/1.1 200 OK
(以降、HTTPヘッダが続く)
```

● リクエスト擬似ヘッダ

- **:method**
  リクエストラインで指定されるHTTPメソッド(GETやPOSTなど)を示すのに使用される
- **:scheme**
  URIのスキーム部分(httpやhttpsなど)を示すのに使用される
- **:authority**
  URIのauthority部分を示すのに使用される。authority部分とは、ドメイン名とコロンで区切られたポート番号で構成される部分のこと。たとえば「example.com:8080」。hostヘッダの代わりに使用される
- **:path**
  URIのパス部分を示すのに使用される。パス部分とは「/index.html_top」などのこと

● レスポンス擬似ヘッダ

- **:stauts**
  HTTPレスポンスのステータスコードを示すのに使用される

▼表5　ヘッダフィールド表現

ヘッダフィールド表現		説明
Indexed Header Field Representation		テーブルのインデックスを1つ用いてヘッダ名／ヘッダ値を表現する
Literal Header Field Representation		ヘッダを表現する際に文字列表現を使用して表現する。ヘッダ名のみインデックスを使いヘッダ値は文字列で表現する方法と、ヘッダ名とヘッダ値の両方を文字列で表現する方法の2通りがある。文字列の表現はハフマン符号化する場合と、しない場合がフラグによって示される。そのヘッダを動的テーブルに追加するように指示するかどうかで、さらに次の3つに分類される
	Literal Header Field with Incremental Indexing	動的テーブルに追加する
	Literal Header Field without Indexing	動的テーブルに追加しない
	Literal Header Field Never Indexed	動的テーブルに追加しない。さらにプロキシサーバなどの中継者も、このヘッダを動的テーブルには追加してはいけない(同じように表現しなければいけない)
Dynamic Table Size Update		これは特殊で、ヘッダを表現するわけではなく、動的テーブルサイズの変更に使用される

▼表6　例Aのヘッダを HPACKで圧縮した場合

ヘッダ	ヘッダフィールド表現	長さ	説明
:method: GET	Indexed Header Field Representation	1バイト	このヘッダは静的テーブル(index＝2)に存在するので、インデックスを使い1バイトで表現される
:scheme: http	Indexed Header Field Representation	1バイト	このヘッダは静的テーブル(index＝6)に存在するので、インデックスを使い1バイトで表現される
:path: /	Indexed Header Field Representation	1バイト	このヘッダは静的テーブル(index＝4)に存在するので、インデックスを使い1バイトで表現される
:authority: www.example.com	Literal Header Field with Incremental Indexing	17バイト	このヘッダは、ヘッダ名のみ静的テーブルに存在する(index＝1)ので、ヘッダ値のみを文字列で表現する(今回はハフマン符号化は行っていない)。ヘッダ名を示すindexの1バイトと、ヘッダ値の長さを示すLengthの1バイトと、ヘッダ値15文字の15バイトで、計17バイトとなる。また、このヘッダを動的テーブルに追加する

▼表7　例Aのリクエスト直後の動的テーブル

インデックス	ヘッダ名	ヘッダ値
62	:authority	www.example.com

▼表8　例Bのヘッダを HPACKで圧縮した場合

ヘッダ	ヘッダフィールド表現	長さ	説明
:method: GET	Indexed Header Field Representation	1バイト	このヘッダは静的テーブル(index＝2)に存在するので、インデックスを使い1バイトで表現される
:scheme: http	Indexed Header Field Representation	1バイト	このヘッダは静的テーブル(index＝6)に存在するので、インデックスを使い1バイトで表現される
:path: /	Indexed Header Field Representation	1バイト	このヘッダは静的テーブル(index＝4)に存在するので、インデックスを使い1バイトで表現される
:authority: www.example.com	Indexed Header Field Representation	1バイト	このヘッダは動的テーブル(index＝62)に存在するので、インデックスを使い1バイトで表現される
cache-control: no-cache	Literal Header Field with Incremental Indexing	10バイト	このヘッダは、ヘッダ名のみ静的テーブル(index＝24)に存在するので、ヘッダ値のみを文字列で表現する(今回はハフマン符号化は行っていない)。ヘッダ名を示すindexの1バイトと、ヘッダ値の長さを示すLengthの1バイトと、ヘッダ値8文字の8バイトで、計10バイトとなる。またこのヘッダを動的テーブルに追加する

例Aのあとに続けて、次のヘッダリストを送信する場合を考えます。

例B
```
:method: GET
:scheme: http
:path: /
:authority: www.example.com
cache-control: no-cache
```

これをHPACKでヘッダ圧縮すると、**表8**のようになります。例Bのリクエストのあとの動的テーブルは、**表9**のようになります[注3]。

サーバプッシュ

HTTP/1.1では、クライアントからリクエストがあって初めて、サーバはレスポンスを返すことができます。逆に言えば、サーバ側から何かを送るということはできませんでした。HTTP/2はサーバプッシュという機能があり、クライアントからのリクエストがなくてもレスポンスを返すことができます。この機能も、HTTP/2では機能を提供するだけであり、どのような使い方をするかは定義していません。効率的な使い方についてはこれからも議論が続くでしょう。

サーバプッシュと言うと、WebSocketのようにメッセージを通知するものを連想されるかもしれません。しかしそれとはやや異なり、基本的には「HTTPリクエストを待たずに、HTTPレスポンスを送信する」というしくみになっています。サーバプッシュのイメージとしては、**図11**のようになります。

HTTP/2では、どのようにサーバプッシュを行っているのかを説明します。サーバはサーバプッシュをする際、PUSH_PROMISEフレームを使用します。ちなみに、クライアントはプッシュ機能を使うことはできません(サーバはPUSH_PROMISEフレームを受け取るとコネクションエラーを返します)。

PUSH_PROMISEフレームには、Promised Stream IDとHeader Block Fragmentという2つのパラメータを含みます。

- **Promised Stream ID**
  プッシュするデータを送信するために予約するストリーム
- **Header Block Fragment**
  想定するHTTPリクエストのヘッダ。サーバはHeader Block Fragmentに示されるHTTPリクエストを受け取ったと想定して、HTTPレスポンスを返す。この情報をもとにクライアントはプッシュされたデータをキャッシュする

図11のように、index.htmlをリクエストしてimage.jpgのプッシュを受け付けるまでの流れを、フレームに注目して図示すると**図12**のようになります。

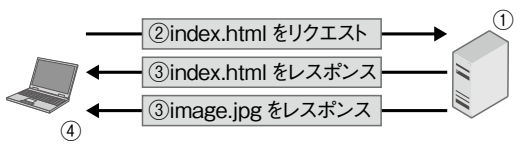
①:クライアントは、通常どおりindex.htmlをリクエストするためのHEADERSフレームを送信する

注3) 動的テーブルを追加する場合は、テーブルの先頭(62番)に新しいエントリが挿入されます。

▼表9 例Bのリクエスト直後の動的テーブル

インデックス	ヘッダ名	ヘッダ値
62	cache-control	no-cache
63	:authority	www.example.com

▼図11 サーバプッシュのイメージ



①サーバ上にindex.htmlがあり、そこにはimage.jpgを表示するimgタグが含まれている。そのため、サーバは「index.htmlを要求したクライアントが次にimage.jpgを要求してくる」ことが想定できる
②クライアントがindex.htmlをリクエストする
③サーバは、index.htmlをレスポンスすると同時に、image.jpgへのリクエストを待たずしてimage.jpgをプッシュする
④クライアントは、プッシュされたimage.jpgをキャッシュとして保存する。通常のHTTPのキャッシュのしくみと同様、image.jpgをリクエストをする際は、キャッシュしたimage.jpgを使用する

②-a：サーバは、受け取ったHEADERSフレームのストリームと同一のストリームで、まずPUSH_PROMISEフレームを送信する。このPUSH_PROMISEフレームでimage.jpgを送信するためにストリームID：2を予約する。サーバはimage.jpgのリクエストを受け取ったと想定して、この予約したストリームIDでプッシュしようとしているわけだが、Header Block Fragmentにはその想定しているリクエストの内容が記述されている

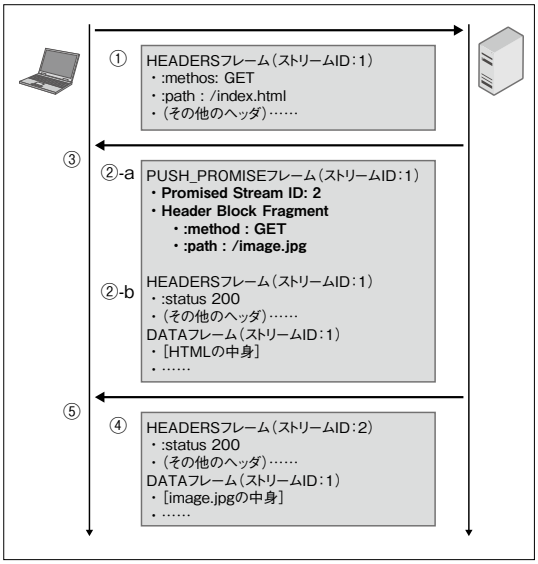
②-b：サーバはPUSH_PROMISEフレームに続いて、クライアントが送信したHEADERSフレームに対するHTTPレスポンスとなるHEADERSフレームとDATAフレームを送信する

③：クライアントは受け取ったPUSH_PROMISEフレームの中身を確認する。すでに該当ファイルをキャッシュしているなどの理由によりプッシュを受ける必要がなければ、予約されたストリームでRST_STREAMフレームを送信することでサーバプッシュを拒否することもできる。図12においてサーバプッシュを拒否する場合は、ストリームID：2が予約されているので、ストリームID：2でRST_STREAMフレームを送信することになる

④：サーバは予約したストリームID：2で、image.jpgをプッシュする（HEADERSフレームとDATAフレームを送信する）

⑤：クライアントはサーバプッシュで受け取ったHTTPレスポンスをキャッシュする。これは通常のHTTPのキャッシュのしくみと同様であり、あらためてHTTPリクエストをする際に、キャッシュがあればそれを使用する

▼図12　サーバプッシュ時のフレームの流れ



HTTP/2を開始する手順

ネゴシエーション

HTTP/1.1とHTTP/2はメッセージに互換性がないにもかかわらず、使用するポート（http://の場合は80番ポート、https://の場合は443番ポート）は同じです。そのため、通信を開始する前には、相手がどちらに対応しているかを確認する必要があります。HTTP/2の仕様では次の3つの手順が挙げられています。

・ALPN（Application Layer Protocol Negotiation）を使用する
・HTTP/1.1で接続してからアップグレードする
・HTTP/2に対応していることが、事前にわかっている場合

お互いに合意がとれたあと、最終確認としてコネクションプリフェイスと呼ばれるメッセージを送信し、HTTP/2の通信が開始されます。上記の3つの手順について順番に説明をしていきます。

ALPNを使用する

現在、おもに使用されているのはALPNであり、主要なブラウザはHTTP/1.1からのアップグレードは対応していません。ALPNは、TLSハンドシェイク中で、使用するプロトコ

ルをネゴシエーションします。この方法は、ネゴシエーションのために余分なRound Trip(パケットの往復)を発生させません。通信の流れは**図13**のとおりです。これにより、お互いに対応しているプロトコルで通信を開始することができます。

よく似た手法にNPN(Next Protocol Negotiation)というしくみもあります。もともと同じようにネゴシエーションをする必要があったSPDYのために考案されたしくみです。このNPNもTLSハンドシェイク中にネゴシエーションを行います。ALPNとのおもな違いは、サーバからプロトコルのリストを提出し、クライアントが選択するという点です。

HTTP/2の仕様中ではNPNについて記述されていませんが、NPNでもHTTP/2のネゴシエーションが可能な実装はあります。

HTTP/1.1で接続してからアップグレードする

HTTP/1.1で接続してから、Upgradeヘッダを用いてHTTP/2の通信に切り替えます。この方法は、WebSocketでも使用されています。

クライアントは、まずHTTPリクエストでConnectionヘッダに「Upgrade」と指定し、HTTP2-Settingsヘッダを送信します(**リスト1**)。HTTP2-Settingsは、SETTINGSフレームのペイロード(設定項目)をbase64urlエンコードした値です。

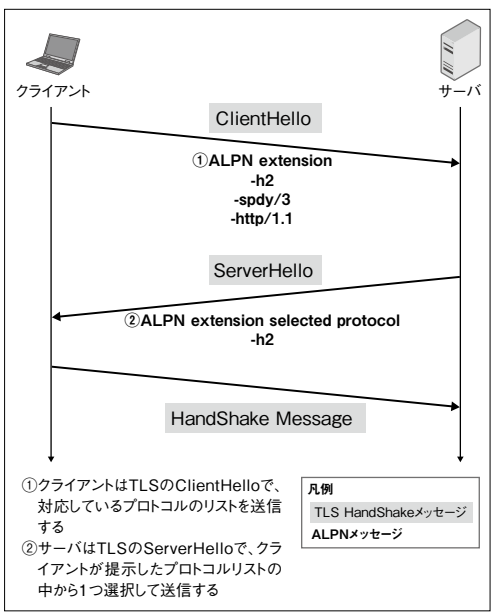
サーバは、プロトコルの切り替えをする場合には「101 Switching Protocols」のHTTPレスポンスを返します(**リスト2**)。以後HTTP/2の通信となります。

HTTP/2に対応していることが、事前にわかっている場合

サーバがHTTP/2に対応していることが事前にわかっていれば、最初からHTTP/2で通信を開始することもできます(ダイレクト接続と呼ばれます)。

たとえば、ネゴシエーションする前に、ほかの方法でHTTP/2に対応していることがわかっている場合です。仕様中では一例としてALT-SVCを用いる場合が挙げられています。

▼図13　ALPNの通信の流れ



▼リスト1　HTTP/1.1からHTTP/2にアップグレードする際のHTTPリクエスト

```
GET / HTTP/1.1
Host: server.example.com
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c
HTTP2-Settings: <HTTP/2 SETTINGSのペイロードをbase64urlエンコード>
```

▼リスト2　HTTP/1.1からHTTP/2にアップグレードする際のHTTPレスポンス

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: h2c

(以降、HTTP/2の通信)
```

ALT-SVCは現在策定中の別の仕様であり、提供しているサービスを別のサーバ／別のプロトコルでも提供できることを通知するしくみです。

プロトコル識別子

少し話は変わりますが、ALPNなどでHTTP/2を示す識別子は次のとおりです。

- h2：TLS上でHTTP/2を使用することを示す識別子
- h2c：TCP上で平文のHTTP/2を使用することを示す識別子

ログ情報を出力するプログラムでは、この識別子に則ってh2、h2cといった情報をログに出力するものもあります。

HTTP/2の仕様がドラフト段階のときは、「h2-16」といったようにハイフンで区切り、その実装のドラフトバージョンを付与して使いました。実装によってはいまだに使用しているものもあります。

最終確認(コネクションプリフェイス)

HTTP/2ではネゴシエーションが終わったあと、サーバとクライアントは通信を開始する最終確認として、コネクションプリフェイスと呼ばれるメッセージをお互いに送信します。コネクションプリフェイスの中身は、クライアントとサーバでそれぞれ異なっています。

クライアントコネクションプリフェイスでは、まず次のような文字列(\r\nはそれぞれ改行文字を意味する)を送信します。

```
PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n
```

この文字列は、HTTP/1.1のリクエストラインのフォーマットをしていますが、HTTP/2.0独特の内容になっています。この文字列を送る理由は、間違ってHTTP/1.1のサーバと接続しようとした際に、切断されるようにするためです。この文字列のあとに続いて、SETTINGSフレームを送信します。

サーバコネクションプリフェイスはSETTINGSフレームだけであり、HTTP/2の接続で最初に送信される必要があります。

このコネクションプリフェイスのあと、実際のWebサイトやWebアプリケーションのデータがHTTP/2でやりとりされます。SD

COLUMN SETTINGSフレーム

SETTINGSフレームは通信に関するパラメータを通知するものです。HTTP/2の仕様の中では、次のようなSETTINGSパラメータが定義されています。

- **SETTINGS_HEADER_TABLE_SIZE**
  ヘッダ圧縮に使用するテーブルサイズをバイト値で相手に通知する
- **SETTINGS_ENABLE_PUSH**
  おもにサーバプッシュを無効化するのに使用する。デフォルト値は1(有効)だが、値を0で通知することで相手にサーバプッシュを送らないように通知できる
- **SETTINGS_MAX_CONCURRENT_STREAMS**
  同時に使用できるストリームの並列数を相手に通知する
- **SETTINGS_INITIAL_WINDOW_SIZE**
  ストリームごとのフロー制御のために使用する、送信ウィンドウサイズの初期値を通知する
- **SETTINGS_MAX_FRAME_SIZE**
  受け入れられるフレームの最大ペイロード長を通知する
- **SETTINGS_MAX_HEADER_LIST_SIZE**
  受け入れられるヘッダのリスト長を通知する

第1特集

第3章

HTTP/2環境の構築とモニタリング手法

Author 後藤 ゆき(ごとう ゆき)／Twitter @flano_yuki　http2study

HTTP/2の基礎が把握できたところで、実際にそれを体験してみましょう。いくつかの実装でHTTP/2サーバを構築します。ただ、その前にまず大前提となる、HTTP/2を用いたサーバサイドのアーキテクチャについてお話しておきましょう。

HTTP/2導入時のアーキテクチャ

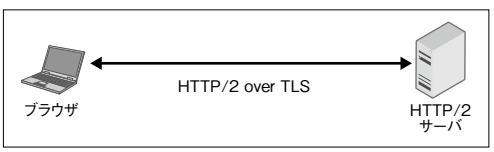
HTTP/2を導入する際のサーバ構成についてはいくつか考えられます。今回はWebサイトでのHTTP/2利用を想定し、3つのパターンを紹介します。

ほとんどのWebブラウザは、平文でのHTTP/2利用をサポートしていません。そのため、クライアントからの通信はHTTP/2 over TLSということを前提にします。もちろん、ネイティブアプリやそのほかのツールで利用する際は平文でのHTTP/2通信を利用できるため今回紹介する構成以外もあり得るかと思います。

①HTTP/2 Webサーバで直接受け付ける

ブラウザのアクセスをそのままHTTP/2 over TLSで受け付けます(**図1**)。とてもシンプルな構成です、HTTP/2を簡単に試すとき使えます。ただし各サーバに証明書をセットする必要があります。

▼図1　HTTP/2 Webサーバで直接受け付ける



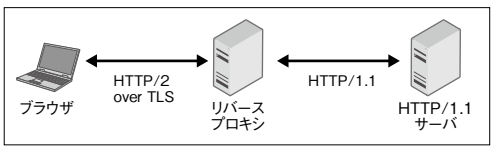
②リバースプロキシでHTTP/2 over TLSをHTTP/1.1に変換する

アプリケーションサーバの前段にリバースプロキシを設置し、HTTP/2 over TLSの通信をHTTP/1.1に変換します(**図2**)。バックエンドのアプリケーションサーバはHTTP/2対応する必要がなく、比較的導入しやすい構成になります。NginxやApache 2なども徐々に対応している段階ですので、すでにリバースプロキシとしてこれらのミドルウェアを利用している場合はリバースプロキシの設定なども多くの部分が流用できるでしょう。ただし、一部モジュールやプラグインの相性などで正しく動作しない可能性もあるので注意が必要です。

③TLS終端のみを途中で行う

中間装置でTLSの終端のみを行い、中身をそのままバックエンドのサーバに送信する構成です(**図3**)。HTTPの通信はブラウザとバックエンド間で行われます。そのため優先処理やプッシュなどがバックエンドサーバ(アプリケーションサーバ)の都合で制御しやすくなります。また、

▼図2　リバースプロキシでHTTP/2 over TLSをHTTP/1.1に変換する



HTTP/2対応は行うものの、HTTP/2 over TLSまではサポートを行わないというミドルウェアでも、ブラウザからのHTTP/2 over TLS通信を受け付けることができます。ただし、TLS終端にはALPN(Application Layer Protocol Negotiation)を用いたHTTP/2のネゴシエーションができる必要があります。さらにALPNでネゴシエーションされたプロトコルに合わせてバックエンドサーバを選択できる必要があります。

たとえばHAProxyではALPNおよびバックエンドの選択が可能であり、こちらの構成がとれます。

環境準備

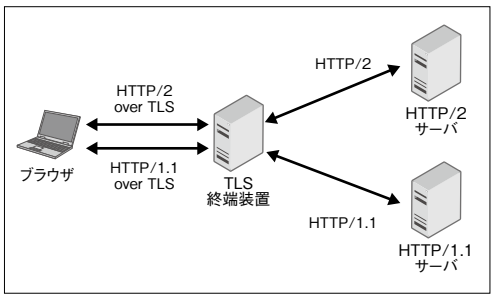
今回はUbuntu Server 15.10(Wily Werewolf)を使用して構築を行いました。執筆中ではUbuntu 15.10は開発版のISOイメージ[注1]を取得しました。正式版[注2]がリリースされたら、そちらからダウンロードしてください。GUI環境の方が使い慣れている方は、Desktop版でもかまいません。15.10では「apt-get install」するだけで、OpenSSL 1.0.2dが取得できます。

Ubuntu 14.04 LTSでも構築できますが、OpenSSLのバージョンが古く、ALPNなどが使用できない場合があります。別途最新版OpenSSLをインストールするか、静的リンクする必要がある点にご注意ください。

また必要な方は事前にGitをインストールしてください。

今回はhttpsでのアクセスを許可するために

▼図3　TLS終端のみを途中で行う



サーバ証明書が必要になります。事前にサーバ証明書(自己署名証明書)を用意しておきます(**図4**)。

nghttp2による構築

tatsuhiro-t氏によって開発されたnghttp2は、SPDYのC++ライブラリも実装されているHTTP/2のC++ライブラリになります。nghttp2には、そのライブラリを用いて作成されたHTTP/2 Client、HTTP/2 Server、HTTP /2 Proxyの実装が含まれています。また、Pythonバインディングも提供されています。詳しくは、作者の公式ドキュメント[注3]をご覧ください。

それではnghttp2を使ってHTTP/2サーバを動作させていきます。今回は次の流れでソースコードよりビルドを行います(**図5**)。

・ビルドに必要なパッケージをインストール
・GitHubよりソースコードの取得
・ビルド
・./src/ 以下に実行ファイルが生成される

注1) URL http://cdimage.ubuntu.com/ubuntu-server/daily/current/
注2) URL http://www.ubuntu.com/server http://www.ubuntu.com/desktop
注3) URL https://nghttp2.org/

▼図4　サーバ証明書の取得

```
$ openssl ecparam -out server.key -name prime256v1 -genkey

↓ いくつか入力を求められる。指示にしたがって入力すること
$ openssl req -new -key server.key > server.csr

$ openssl x509 -days 30 -req -signkey server.key < server.csr > server.crt
```

サーバ

nghttpdがHTTP/2サーバとして動作します。次のように起動することができます。

```
↓ nghttpd <PORT> <PRIVATE_KEY> <CERT>
$ sudo nghttpd 443 ./server.key ↵
./server.crt
```

「-v」オプション付きでサーバを起動し、Listenしたポートに対してブラウザ「https://」のURLでアクセスしてみると、**図6**のようにログが表示され、HTTP/2でアクセスできていることが確認できます。

nghttpdでおもに使用するオプションに、次のようなものがあります

- -D：デーモンとして起動
- -d：サーバのルートディレクトリを指定
- -v：詳細ログの表示

リバースプロキシ

nghttpxがプロキシとして動作します。いくつかのモードがあり、リバースプロキシとして動作させたり、フォワードプロキシとして動作させたりできます。今回は、リバースプロキシとして動作させます。

```
↓ nghttpx <PRIVATE_KEY> <CERT>
$ sudo nghttpx  ./server.key ./server.crt
```

nghttpdでおもに使用するオプションに、次のようなものがあります。

- -f：フロント側のIP、ポート指定
- -b：バックエンドサーバのIP、ポート指定
- -L：ログレベルの指定
- -D：デーモンとして起動
- --conf：confファイルの指定
- --accesslog-file：アクセスログファイルのpath
- --accesslog-format：アクセスログのフォーマット指定
- --errorlog-file：エラーログファイルのpath

図7のようにプロキシを起動し、アクセスログを確認できます。また、ログレベルを上げることで、HTTP/2の通信内容を確認することもできます。

h2oによる構築

h2o[注4]は高品質かつ便利なHTTPサーバ実装を提供することを目標として、DeNAの奥一穂氏によって開発されたHTTPサーバ実装です。

注4） URL https://github.com/h2o/h2o

▼図5　nghttp2をソースコードよりビルドする

```
↓ 必要なものをインストール
$ sudo apt-get install make binutils autoconf  automake autotools-dev libtool pkg-config \
  zlib1g-dev libcunit1-dev libssl-dev libxml2-dev libev-dev libevent-dev libjansson-dev \
  libjemalloc-dev cython python3.4-dev python-setuptools t

↓ コードの取得
$ git clone -b v1.3.0  https://github.com/tatsuhiro-t/nghttp2.git

↓ ビルド
$ cd ./nghttp2
$ autoreconf -i
$ automake
$ autoconf
$ ./configure
$ make

$ ./src/nghttp --version
nghttp nghttp2/1.3.0
```

生成されるバイナリレベルでの最適化が施されており、非常に高速に動作します。詳しくは作者のドキュメント[注5]をご覧ください。

注5） URL https://h2o.examp1e.net/

サーバ

ではh2oでもHTTP/2サーバを動作させていきます(**図8**)。

▼図6　ログによるHTTP/2でアクセスできているかの確認

```
IPv4: listen 0.0.0.0:443
[ALPN] client offers:
 * http/1.1
 * spdy/3.1
 * h2-14
 * h2
The negotiated protocol: h2
(...中略...)
[id=3] [  4.321] recv (stream_id=3) :authority: 192.168.0.102
[id=3] [  4.321] recv (stream_id=3) :method: GET
[id=3] [  4.321] recv (stream_id=3) :path: /hoge
[id=3] [  4.321] recv (stream_id=3) :scheme: https
[id=3] [  4.321] recv (stream_id=3) accept: text/html,application/xhtml+xml,application/➥
xml;q=0.9,image/webp,*/*;q=0.8
[id=3] [  4.321] recv (stream_id=3) accept-encoding: gzip, deflate, sdch
[id=3] [  4.321] recv (stream_id=3) accept-language: ja,en;q=0.8
[id=3] [  4.321] recv (stream_id=3) cache-control: max-age=0
[id=3] [  4.321] recv (stream_id=3) upgrade-insecure-requests: 1
[id=3] [  4.321] recv (stream_id=3) user-agent: Mozilla/5.0 (Windows NT 6.1; WOW64) ➥
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.2403.157 Safari/537.36
[id=3] [  4.321] recv HEADERS frame <length=25, flags=0x25, stream_id=3>
          ; END_STREAM | END_HEADERS | PRIORITY
          (padlen=0, dep_stream_id=0, weight=256, exclusive=0)
          ; Open new stream
```

▼図7　リバースプロキシとして起動し、アクセスログや通信内容を確認

```
$ sudo ./nghttp2/src/nghttpx   ./server.key ./server.crt  -f *,443 -b localhost,➥
80  --accesslog-file ./access

$ sudo cat ./access.log
192.168.0.230 - - [03/Sep/2015:23:42:15 +0900] "GET / HTTP/2" 200 612 "-" "Mozilla/5.0 ➥
(Windows NT 6.1; WOW64) AppleWebKit/537.➥
36 (KHTML, like Gecko) Chrome/44.0.➥
2403.157 Safari/537.36"
```

▼図8　h2oでのHTTP/2サーバの起動

```
↓ 必要なものをインストール
$ sudo apt-get install git cmake libssl-dev

↓ ソースコードの取得
$ git clone -b v1.4.4 https://github.com/h2o/h2o.git

↓ ビルド
$ cd ./h2o
$ cmake -DWITH_BUNDLED_SSL=on .
$ make
$ sudo make install

↓ 起動
$ /usr/local/bin/h2o -c examples/h2o/h2o.conf
```

・ビルドに必要なパッケージをインストール
・GitHubからソースコードを取得
・ビルド、インストール

読み込む設定ファイルおよび、証明書はexamplesディレクトリに最初から含まれています(**リスト1**)。

ブラウザで8081ポートに対し、「https://URL」でアクセスすると、**図9**のようにアクセスログが標準出力に出力されます。

リバースプロキシ

リスト2のようにpathsに、proxy.reverse.urlを指定することでリバースプロキシを構成することが可能です。

▼リスト1　examples/h2o/h2o.conf

```
listen: 8080
listen:
  port: 8081
  ssl:
    certificate-file: examples/h2o/server.crt
    key-file: examples/h2o/server.key
hosts:
  "127.0.0.1.xip.io:8080":
    paths:
      /:
        file.dir: examples/doc_root
    access-log: /dev/stdout
  "alternate.127.0.0.1.xip.io:8081":
    listen:
      port: 8081
      ssl:
        certificate-file: examples/h2o/alternate.crt
        key-file: examples/h2o/alternate.key
    paths:
      /:
        file.dir: examples/doc_root.alternate
    access-log: /dev/stdout
```

▼リスト2　h2oをリバースプロキシとして動作させる設定

```
listen:
  port: 8081
  ssl:
    certificate-file: examples/h2o/server.crt
    key-file: examples/h2o/server.key
hosts:
  "192.168.0.102:8081":
    paths:
      /:
       proxy.reverse.url: http://127.0.0.1/
    access-log: /dev/stdout
```

Nginx HTTP/2パッチによる構築

Nginxを使用されている人も多いのではないでしょうか。Nginxも「Early Alpha Patch for HTTP/2」としてパッチが公開されていましたが、バージョン1.9.5で本体に組み込まれました。

サーバ

NginxでHTTP/2を使うためには、Nginxが提供しているパッケージ[注6]を使うか自分でソースコードをダウンロードしてビルドをする必要があります(**図10**)。

主な流れとしては、

・必要なパッケージをインストール
・本体のソースコードの取得
・パッチのダウンロード及び反映
・ビルド

になります。

次に/usr/local/nginx/conf/nginx.confを**リスト3**のように設定します。HTTPS serverの設定全体がコメントアウトされているので、コメントアウトを外します。

listenの部分にhttp2を指定することでHTTP/2を使用できるようになります。「ssl」を指定しない場合はhttp://のURLでもHTTP/2を使用できますが、**HTTP/1.1からのアップグレードには対応しておらず**ダイレクト接続のみの通信になります。

Nginxを使ううえで、ssl_prefer_server_ciphersを設定している場合は、TLSネゴシエーション時に選択される暗号スイートはサーバの設定が優先されます。そのため、HTTP/2で使用

注6) URL http://nginx.org/packages/mainline/ubuntu/pool/nginx/n/nginx/

できる暗号スイートが選択されるように考慮する必要があるので注意してください。たとえば、次のようにssl_ciphersのパラメータは、AESGCMの暗号スイートを優先する必要があります。

```
ssl_ciphers AESGCM:HIGH:!aNULL:!MD5;
```

リバースプロキシ

リバースプロキシとして動作させる設定は**リスト4**のとおりです。

HTTP/2だからといって特殊な設定はなく、HTTP/1.1と同様「proxy_pass」パラメータを使

▼図9　h2oで出力されるアクセスログ

```
10.0.2.2 - - [06/Sep/2015:02:37:38 +0900] "GET / HTTP/2" 200 177 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/537.36"
10.0.2.2 - - [06/Sep/2015:02:37:38 +0900] "GET /favicon.ico HTTP/2" 404 9 "https://localhost:8081/" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/537.36"
```

▼図10　NginxでのHTTP/2サーバの起動

```
↓ 事前準備
$ sudo apt-get install build-essential libc6 libpcre3 libpcre3-dev libpcrecpp0v5 libssl1.0.0 libssl-dev zlib1g zlib1g-dev lsb-base

↓ 本体ダウンロード
$ wget http://nginx.org/download/nginx-1.9.5.tar.gz
$ tar zxvf nginx-1.9.5.tar.gz

↓ ビルドする。必要があれば、--with-openssl= でopensslを指定する
$ cd ./nginx-1.9.5
$ ./configure --with-http_ssl_module \
 --with-http_v2_module \
 --with-debug
$ make
$ sudo make install

↓ confを編集する（リスト3参照）
$ vim  /usr/local/nginx/conf/nginx.conf

↓ 起動する
$ sudo ./obj/nginx
```

▼リスト3　/usr/local/nginx/conf/nginx.confを変更する

```
# HTTPS server
# 以下のコメントアウトを外す
server {
    listen       443 ssl http2; # http2を追加する
    server_name  localhost;
    ssl_certificate      cert.pem; # 証明書を指定
    ssl_certificate_key  cert.key; # 鍵を指定

    ssl_session_cache    shared:SSL:1m;
    ssl_session_timeout  5m;
    ssl_ciphers  HIGH:!aNULL:!MD5;
    #ssl_prefer_server_ciphers  on; # コメントアウトする
    location / {
        root   html;
        index  index.html index.htm;
    }
}
```

用することでリバースプロキシとして動作します。

アクセスログにHTTP/2でのリクエストなのかを出力する場合は、log_formatにて$http2を指定することで出力されます(**リスト5**)。

「https://URL」のときはh2、「http://URL(平文の通信)」のときはh2cと表示されます(**図11**)。

そのほかのHTTP/2に関わる設定項目は、Nginxのドキュメンテーションページ[注7]に記載されています。

Apache 2 mod_h2による構築

Webサーバとして Apache 2を使っている方もいるでしょう。Apache 2の場合はHTTP/2モジュールとしてmod_h2が公開されています。作者の公式ページ[注8]より、開発状況が確認できます。

mod_h2には、Sandbox Installationという必要なライブラリ群もすべてダウンロードを行い、サーバに影響を与えないようにビルドするインストール手順があります。このインストール手順を用いることで、簡単にApacheでのHTTP/2環境を試すことができます(**図12**)。

以前は動作したのですが、執筆時に確認したところ大きな変更が入ったため正しく動作しないようです。そのため、今回は簡単な解説のみとします。

また、Apache 2.5の開発版にはmod_h2がすでに含まれています。Apache 2.5ではProtocolsという設定項目が増え、そこでh2を指定することができます。

```
Protocols h2 http/1.1
```

またこれと合わせて、mod_h2の設定項目については公式のドキュメント[注9]をご覧ください。

Wiresharkによるモニタリング

通信の中身をパケットレベルで確認、解析するのに、Wiresharkを使用される方は多いのではないでしょうか? Wireshark(Development Release 1.99.8)もHTTP/2に対応しています

注7) URL http://nginx.org/en/docs/http/ngx_http_v2_module.html
注8) URL https://github.com/icing/mod_h2
注9) URL http://httpd.apache.org/docs/trunk/ja/mod/mod_h2.html

▼リスト4 Nginxをリバースプロキシとして動作させる設定

```
location / {
  proxy_pass http://localhost:80;
  proxy_set_header Host $http_host
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
}
```

▼リスト5 アクセスログにHTTP/2であることを表示させる設定

```
log_format  h2 '$remote_user [$time_local] "$request" '
                       '$status "$http_referer" '
                       'h2:$http2';
access_log  /var/log/nginx/access.log  h2;
```

▼図11 アクセスログの表示例

```
- [23/Aug/2015:29:20:30 +0900] "GET / HTTP/1.1" 200 "-" h2:h2
- [23/Aug/2015:29:21:11 +0900] "GET / HTTP/1.1" 200 "-" h2:h2c
```

ので、HTTP/2の通信の中身を解析できます。Wiresharkを使う主な利点としては次のようなものがあります。

・パケットのバイナリも確認できるため、トラブルシューティング、学習に非常に役立つ
・IP、TCP、TLS、HTTP/2とプロトコルスタックすべてを通して確認できる
・フィルタ機能を用いることで特定のパケットストリームのみ抜き出すことができる
・統計機能が使用できる

Wiresharkの Stable Release 1.12.7(執筆時点)ではHTTP/2に対応しているものの、h2-13(2014年6月ごろの草案)での対応のため、現在のHTTP/2を正しくパースすることはできません。HTTP/2の解析のためにWiresharkを使う場合はDevelopment Release 1.99.8を使用する必要があります。Wireshark公式ページ[注10]からインストーラが提供されていますので、ご自身の環境に合わせてインストールしてください。

また、HTTP/2をモニタリング・パケットキャプチャする注意点があります。TLSを用いた暗号通信をWiresharkで解析する場合は、その暗号を復号する必要があります。HTTP/2でTLSを使用する場合は、Forward secrecyという特性のある鍵交換方式が用いられる必要があります。

Forward secrecy特性のある鍵交換方式では、その接続でのみ有効な一時鍵が共有され、通信が暗号化されますので、サーバの秘密鍵では通信を復号できません。その通信で暗号化に使用される鍵であるMaster Secretという値を直接Wiresharkに指定する必要があります。ブラウザに特殊な設定をするか、サーバ側のデバッグログよりMaster Secretの値を取得する必要があります。

FirefoxやChromeでは、SSLKEYLOGFILEという環境変数を使うことでTLS通信のMaster Secretを取得できます。Macの場合は次のように、コンソールより環境変数つきでブラウザを起動します。

```
$ SSLKEYLOGFILE=/tmp/tls_key.log ➪
"/Applications/Google Chrome.app/➪
Contents/MacOS/Google Chrome"
```

注10) URL https://www.wireshark.org/download.html

▼図12　ApacheでのHTTP/2環境を試す手順(参考用)

```
↓ 各種設定アプリ
$ sudo apt-get install git gcc g++ libpcre3-dev libcunit1-dev libev-dev libjansson-dev ➪
libjemalloc-dev cython make binutils autoconf automake autotools-dev libtool pkg-config ➪
zlib1g-dev libssl-dev libxml2-dev libevent-dev python3.4-dev libevent-openssl-2.0-5 ➪
python-setuptools curl git

↓ ソースコードの取得
$ git clone -b v0.9.7 https://github.com/icing/mod_h2

↓ ビルド
$ cd ./mod_h2
$ autoreconf -i
$ automake
$ autoconf
$ ./configure --enable-sandbox
$ make
$ sudo make install

↓ 起動 h2cが12345ポートで、h2が12346ポートで起動する
$ make start
```

Windowsの場合はシステムのプロパティより環境変数の指定ができます(**図13**)。ブラウザでhttpsでサイトに接続を行うと、指定したファイルに**図14**のようにClient Randomと続いてMaster Secretが出力されています。

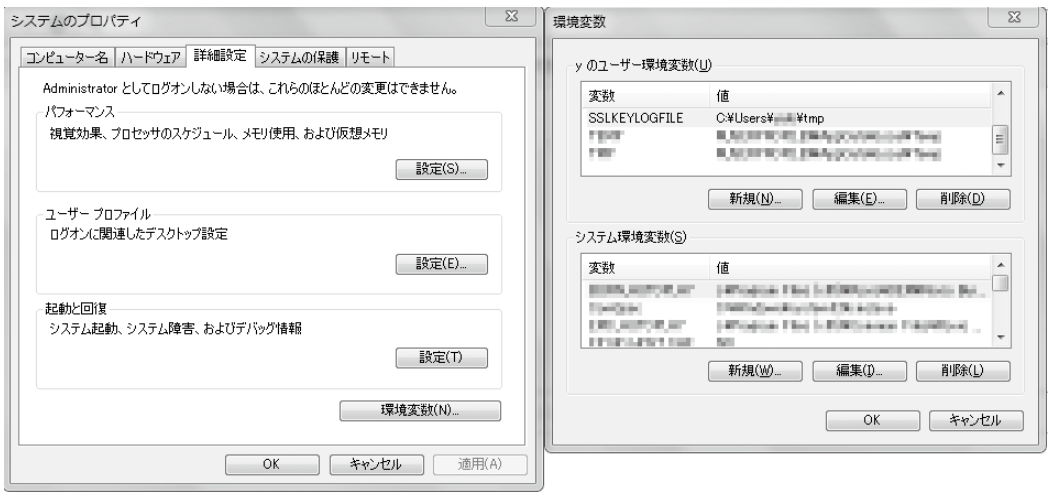
出力されたファイルを、WiresharkのPreferencesからSSLを選択し、(Pre)-Master-Secret log filenameを指定することで該当通信のTLS通信を復号できます(**図15**)。

平文であれば、そのままWiresharkで解析することもできます。

パケットリスト画面

パケットキャプチャをすると、**図16**のようにProtocolがHTTP2と識別され、Infoにフレームタイプが表示されます。この際に該当パケットがHTTP2として認識されない場合は、分析(Analyze)→……としてdコード(Decode

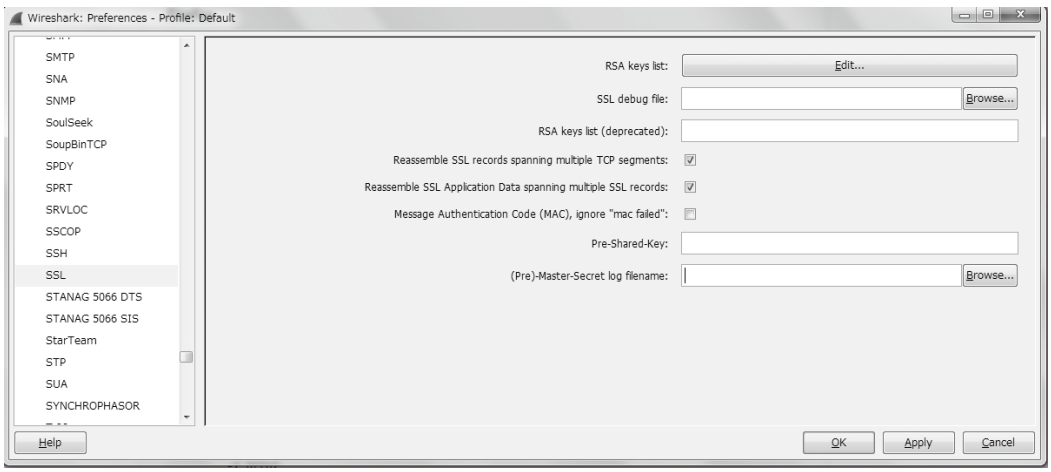
▼図13　Windowsの場合の環境変数の設定



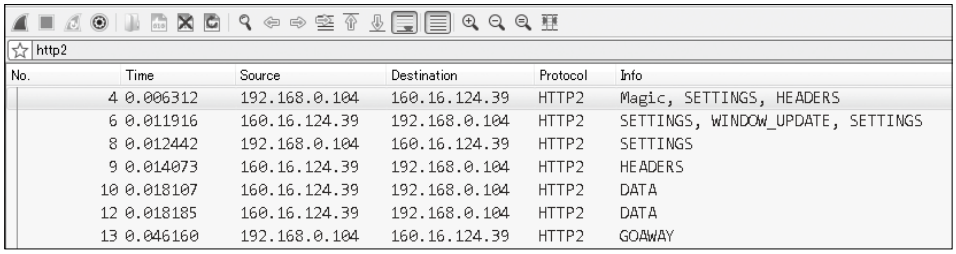
▼図14　Master Secretが出力される

```
CLIENT_RANDOM 5db2a8ece8c6e5ce5b69194af9093adc9ec9273081d620a29a623159721e4332 6b76331fb12
da10165a0d1bc30bb17097cafa2c3ce302a1741d1b1229a04ea3b90b416f18538b17d31b3220af85f72ae
```

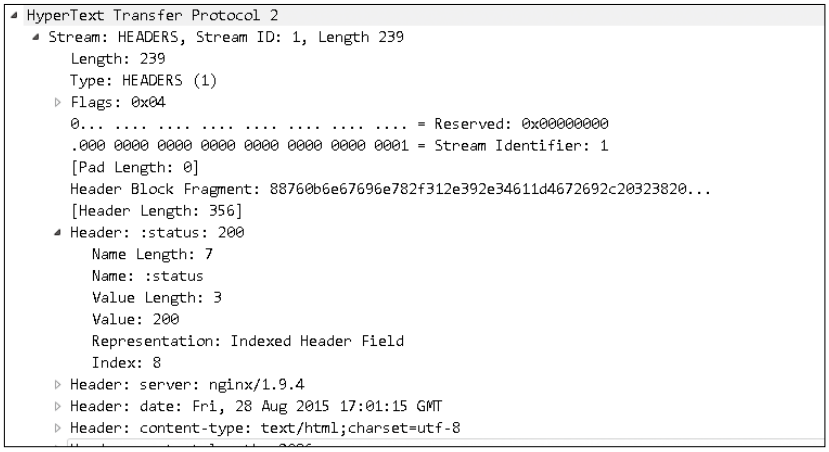
▼図15　TLS通信の復号指定



▼図16　パケットキャプチャ時の表示



▼図17　キャプチャしたパケットの詳細画面



As...)からTransport欄よりHTTP2を選択することで、該当パケットをHTTP2としてデコードさせることができます。

図16では上部分にあるフィルタ欄にhttp2と入力することで、HTTP/2の通信のみ表示しています。http2.に続いてさまざまな指定をすることでさらに絞り込むこともできます。

- **「http2.headers」**
  フレームタイプとしてheaders、data、go away、unknownなどが指定できる
- **「http2.streamid == 1」**
  ストリームIDを指定できる
- **「http2.header.name == "accept"」**
  特定のHTTPヘッダ名を指定できる
- **「http2.flags.priority」**
  特定のフラグを指定できる

パケット詳細画面

続いて、キャプチャしたパケットの詳細画面を見ていきます(**図17**)。WiresharkがHTTP/2に対応している場合は「HyperText Transfer Protocol 2」と表示されます。仮に古いバージョンのWiresharkを使用していると、HTTP/2のドラフト番号が併記されます。

HEADERSフレームの場合は詳細画面は、**図17**のようにフレームタイプ・フラグ・長さといったフレームの基本情報および、HTTPヘッダの名前と値がパースされており、読める形で表示されます。

Wiresharkのコマンドライン版であるTsharkでも同様にパケットの解析が行えます。SD

第1特集

第4章

HTTP/2を取り巻くWeb標準技術とIETFの今後

Author 川田 寛(かわだ ひろし)／Twitter @_furoshiki 東京Webパフォーマンス
Author 後藤 ゆき(ごとう ゆき)／Twitter @flano_yuki http2study

Webの技術はHTTPだけではありません。本章では、HTMLやWeb周りのAPIにも目を向けてみます。HTTP/2と組み合わせることで効果を発揮するタグやAPIが登場してきています。また、IETFではHTTP/2のさらに先の技術も検討され始めています。その一端を紹介しましょう。

Web標準技術の動向

本章では少しだけ、通信のしくみの上位に目を向けてみましょう。HTTPよりもさらに上、HTMLの話です。ここまではIETF標準の話でしたが、ここからはW3C(World Wide Web Consortium)や WHATWG(Web Hypertext Application Technology Working Group)[注1]といったWeb標準で、HTTP/2がどのように扱われるのかをお話します。

そもそも、「パフォーマンスに関する技術について、HTTP/2がすべてをカバーできるわけではない」というのがWeb標準としての答えで、そのためにJavaScriptやHTML側からネットワークの制御を行うための機能が提供されています。HTTP/2の時代を想定した仕様が追加されていたり、HTTP/2では対応できなかったような課題をフロントエンドで解決しようという動きがあったりします。

たとえば、これまでにずっと議論として続いているのが、HTMLのlinkタグを用いたネットワークとリソースの制御です。また、画像の遅延読み込みを行うためのlazyload属性についても、Resource Prioritiesと呼ばれるスペック(仕様)で議論を進めていました。しかし、GoogleのIlya Gritorik氏がResource Hintsと呼ばれるスペックを提案したことにより、そこに先ほどのResource Prioritiesがマージされることになりました。そして、lazyloadはその実装面の困難さからスペックから削除されてしまいました。linkタグの柔軟さ、ユースケースとの整合性の高さから、linkタグベースの方法のみが生存することになります。

2015年現在、Webページの読み込みパフォーマンスに関連したスペックは、Resource HintsとPreloadの2つになりました。このあたり、HTTP/2の影響で興味深い進化をしているため、少し深く掘り下げてみます。

Resource Hints

Resource Hintsは、Webページの読み込みなど、将来的にサーバに対して何かしらのアクションを行う可能性があるような場合、linkタグを使ってヒントを与えるためのスペックです。リソースという名称こそついていますが、実際にはリソース以外にもさまざまな制御が行え、HTTP/2の特性にあった機能も追加されています。

現在、扱えるヒントは次のとおりです。

注1) HTMLやWebアプリケーションのためのAPIの開発に取り組んでいるコミュニティ。Web技術者のニーズを軽視しがちであったW3Cに不満を持ったApple、Mozilla、Operaのメンバーによって設立された。W3Cにより勧告された技術の中には、HTML5のようにWHATWGが作成した仕様がベースとなっているものもある。

▼リスト1　Resource Hintsの指定例（HTMLドキュメント上でヒントを指定する）

```
<link rel="dns-prefetch" href="//widget.com" pr="0.75">
<link rel="preconnect" href="//cdn.example.com" pr="0.42">
<link rel="prerender" href="//example.com/next-page.html" pr="0.75">
<link rel="prefetch" href="//example.com/logo-hires.jpg" as="image">
```

・dns-prefetch：指定のドメインのDNS名前解決を先行して実行し、キャッシュする
・prefetch：指定のURLの画像やJavaScriptといったリソースを先読みし、キャッシュする
・prerender：指定のURLのWebページの内容を先読みし、キャッシュする
・preconnect：指定のドメインのサーバへ接続する

dns-prefetch、prefetch、prerenderは、2014年に起きたIlyaの大変革の前から存在していた仕様で、今もなお互換性を保ったまま利用できます。多くのブラウザで、すでに実用的に扱えます。一方でpreconnectは、ここ最近になってから追加されたヒントです。

preconnectは、DNS名前解決やTLSの鍵交換に加え、TCPのスリーウェイハンドシェイクまでをあらかじめ行い、サーバと接続済みの状態にします。ここまででも説明してきたとおり、HTTP/2はTCP接続を柔軟に制御できるプロトコルです。複数のタブ／ウィンドウ間で1つのTCPコネクションを共有できるのみならず、別のWebページに遷移してもコネクションを切らなくても良いなど、とにかく常に接続しっぱなしを前提にできます。つまり、preconnectを使えば、HTTP/2によってリソースが読み込まれるかなり前から、将来的に接続される可能性をふまえて事前接続しておくことを可能にします。preconnectはまさに、HTTP/2の時代を見据えて生まれた仕様とも考えられます。

このスペックは、さまざまな用途が想定できます。一番簡単な例だと、たとえば、Googleの検索結果のような、リダイレクトを活用してユーザの行動などのサービス分析を行っているケースです。リダイレクトはTCPやHTTP、TLS鍵交換のRTTがダイレクトに響くしくみであるため、リダイレクトサーバをpreconnectしておくとパフォーマンスは劇的に変わるでしょう。

Webページ読み込み後に、JavaScriptで何かしらのリソースを遅延読み込みするようなケースでも、preconnectを使って先にサーバと接続しておけば、レスポンスタイムは改善されるでしょう。JavaScriptに限らず、HTMLドキュメント上でサーバ接続を行うことが明白となっている場合も、効果が見込めるかもしれません。こうしてアプリケーションレイヤも含めて考えると、HTTP/2の使いどころは非常に多様にも見えます。

Resouce Hintsは「未来」に目を向けて事前にアクションを行う機能なのですが、そこにフォーカスする以上、避けて通れないのが、未来の「可能性」です。アクセスするかもしれないけど、しないかもしれない。将来的に使うもしれないけど、使わないかもしれない。このあたりの可能性については、pr（Probability）と呼ばれる属性で指定することができ（**リスト1**）、可能性の高さを数値の大きさで、0から1の間で実数にて指定します。この値は優先度ではないのがミソで、HTTP/2のPriorityと同様、何がどう優先されるのか、その実装方法は定義されていません。

Resource HintsはHTTPヘッダ内からも扱えます（**リスト2**）。HTMLドキュメントのheadタグで扱うよりも早くブラウザはその意味を解釈できるため、オーバーヘッドをさらに小さくすることも期待できるでしょう。また何よりも、ミドルウェア側から扱えるというのも大きく、インフラエンジニアとアプリケーションエンジニアの双

▼リスト2　Resource Hintsの指定例（HTTPレスポンスヘッダ内でヒントを指定する）

```
Link: <https://widget.com>; rel=dns-prefetch
Link: <https://example.com>; rel=preconnect
Link: <https://example.com/next-page.html>; rel=prerender;
Link: <https://example.com/logo-hires.jpg>; rel=prefetch; as=image;
```

▼リスト3　Preloadの指定例（HTMLドキュメント上で指定する）

```
<link rel="preload" href="/styles/other.css" as="style">
```

▼リスト4　Preloadの指定例（HTTPレスポンスヘッダ内で指定する）

```
Link: <https://example.com/other/styles.css>; rel=preload; as=style
```

方で共有できる柔軟さはとても魅力的です。

Preload

Preloadはリソースの先読みを行うためのスペックです。

基本的な考え方は先ほどのResource Hintsと同じで、linkタグからも扱えるうえに、HTTPレスポンスヘッダ内で指定することもできます（**リスト3、4**）。ただ、Resource HintsはWebサイトの中で何度もWebページを遷移することが前提であり、狙うところは次のWebページの読み込み時のパフォーマンス改善という感じが強かったりします。しかし、このPreloadは、どちらかといえば、今アクセスしているWebページのパフォーマンス改善です。

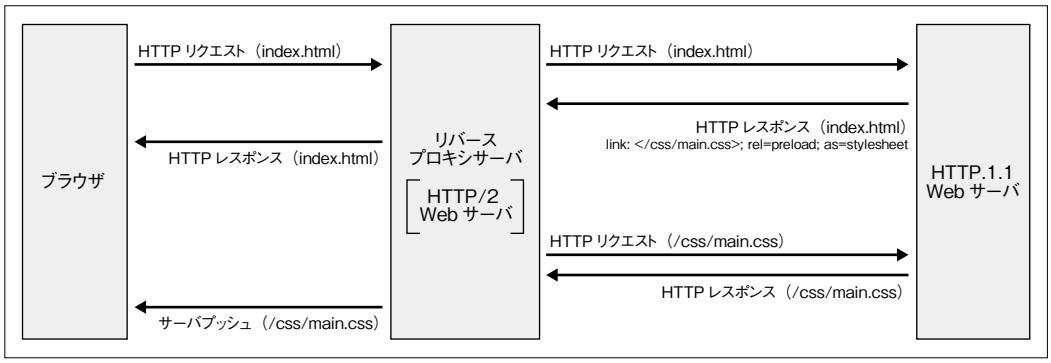
もう少しだけ踏み込んで話しておきましょう。そもそも、W3CのWeb標準にまで至っていない、WHATWGで扱われているようなレベルの仕様に「Fetch」と呼ばれるJavaScript APIがあります。この仕様は何かと言うと、XMLHttpRequest[注2]ほど複雑な処理や制約を排除し、気軽にサーバからリソースを取得しクライアントのキャッシュに読み込ませる機能です。リソースをキャッシュさせる、たったそれだけのことが、Webのフロントエンドをアプリケーションのように扱うようになった昨今では、重要だったりします。

しかし、XMLHttpRequestやFetchを使ったJavaScript上からの読み込みは、パフォーマンスにとって大きな痛手となることがあります。ブラウザはJavaScriptやCSSによってレンダリングをブロックされる特性を持つため、ネットワークを効率的に扱おうと、プリロードパーサ（もしくはプリロードスキャナー）というしくみが動きます。これは、HTMLドキュメント内のタグの中からURLを探し、先行してサーバからリソースを読み込もうとする機能なのですが、JavaScriptやCSSの中までは追ってはくれません。したがって、タグにURLを指定するのが、パフォーマンスにとっては最も効果的なのです。それを実現しようというのが、このPreloadというスペックです。

PreloadもResource Hintsと同様に、アプリケーションエンジニアとインフラエンジニアの両方から扱うことが可能です。ただ、この仕様はResource Hintsのほかの仕様とは異なる点が1つあります。「リソースをキャッシュに読み込む」このしくみはHTTP/2にもあったことを覚えていますでしょうか？　そう、サーバプッシュです。Preloadは、HTTP/2を扱えるミドルウェアが、HTTP内にサーバプッシュが含まれていることを、ミドルウェア間で共有するために活用しています。この点で、

注2）スクリプト言語を使ってサーバにHTTPリクエストを行うためのAPI。一度読み込んだWebページからJavaScriptなどを実行することで非同期にHTTPリクエストを送り、XMLやJSONのデータを受信することができる。これによりページ遷移を伴わずに、ページ内容の書き換えが可能になる。

▼図1　サーバプッシュにおけるPreloadの活用例



Resource Hintsとは大きく異なります。

たとえば、**図1**のような例が考えられます。HTTP/2で動作しているWebサーバが終端にリバースプロキシとして立ててあり、HTTP/1.1で動作している既存のWebサーバからリソースを取得する場合、サーバプッシュの存在をリバースプロキシ側へ通知します。これを受け、HTTP/2のWebサーバ側はブラウザにPUSH_PROMISEフレームを送信し、HTTPリクエストを使わずにブラウザへリソースを読み込ませます。

モニタリング

ブラウザには、JavaScript APIを経由してパフォーマンス情報を取得するためのAPIがあります。Webページの遷移のタイミングを利用してパフォーマンス情報をどこかのサーバへ送る、あるいは監視専用のサーバ内でブラウザを動かして計測するなど、さまざまな方法で活用されています。とくにNavigation Timingは、Internet Explorer 9の時代に実装されたということもあり、パフォーマンスメトリクスとして広く活用されるようになりました。しかし、最近は徐々に世代交代の時期が近づきつつあります。

現在、Performance Timelineと呼ばれる汎用的なインターフェースを通じて、Webページの読み込みに限らず、さまざまなパフォーマンスのメトリクスを取得できるよう改善が進められています。それこそ個々のリソースの読み込み速度、フレーム速度、任意のJavaScriptコードの実行速度など対象は幅広く、またこれらのパフォーマンスをブレークダウンして可視化できるようにしています。こうした流れもあり、次世代のNavigation Timing 2は、互換性が完全に失われています。旧式のNavigation Timingは今後、姿を消していくことになるでしょう。

ブラウザのモニタリング用APIの動向がある程度見えたところで、HTTP/2による変化にも目を向けてみましょう。HTTP/2のパフォーマンスをそのまま計測できるようなAPIは、Web標準としては提供していません。しかし、先ほどの「Navigation Timing 2」と、2014年の秋ごろから実用性を増しつつあるスペック「Resource Timing」には、HTTP/2とHTTP/1.1の双方の違いを明確に意識せざるを得ない仕様が追加されています。

Navigation Timingは、Webページの読み込み速度を計測するためのAPIです。ハイパーリンクやブックマークといったさまざまな機能を通じて指定されたURLに対して、HTMLドキュメントを要求しWebページを完成させる過程を計測するものです。

一方、Resource Timingは、リソースごとの読み込み速度を計測するAPIです。たとえばimgタグで読み込むような画像ファイルが、いつ名前解決を始めて、いつリダイレクトを始

めて、いつサーバとTCPスリーウェイハンドシェイクを始めたのかといった情報を記録し、あとでPerformance Timelineの仕様に沿ったJavaScript APIを通じて取得する機能です。Resource Timingの使用例を**リスト5**に示します。

どちらも速さを求めると、必然的にネットワークパフォーマンスを改善することが求められます。実際の改善作業を行う際には、判断材料として、HTTP/2を意識しなくてはいけません。

第2章において、HTTP/1.1とHTTP/2のどちらの通信プロトコルを使うべきかという判断を行うネゴシエーションのフェーズでは、ALPNと呼ばれる方法が主流であると説明をしましたが、Navigation Timing 2とResource Timingでは、その情報をnextHopProtocolと呼ばれるプロパティを通じて取得できるようになります。また、HTTP/2から新たに追加された概念であるフレームごとのオーバーヘッドについても、transferSizeと呼ばれるプロパティを通じて計測値を取得できるようになります。

ついこの前までHTTP/2に対する動きがまったくなかったモニタリング分野ですが、情報や概念の整理が徐々に進められ、HTTP/2を最適に扱えるよう改善が進められそうです。

IETFの動向と今後

再び通信側のレイヤに話を戻し、IETFの動向について最近の中から大きなものを紹介します。

IETF 93 プラハ

IETFは年に3回オフラインの会合が行われます。直近のIETFはチェコのプラハで2015年7月19～24日にかけて行われました。

ワーキンググループごとにセッションが開かれ、発表／議論が行われています。HTTPを扱うHTTPbisワーキンググループもセッションが開かれました。アジェンダや議事録はGitHubで公開されており誰でも閲覧することが可能です。

HTTP/2はRFCとしてすでに公開されたので、HTTP/2に関してはアジェンダには盛り込まれませんでした。おもにワーキングドラフトと呼ばれる段階となっている、以下の仕様の議論が行われました。

- **Alternative Services**[注3]
  - オリジンのリソースを別のサーバ、別のプロトコル、別のポートで提供可能にする仕様
  - 別のサーバに通信を切り替えるように指示できるため、TCPコネクションの生存時

注3）URL https://tools.ietf.org/html/draft-ietf-httpbis-alt-svc-07

▼リスト5　Resource Timingの使用例

```
// imgタグを由来として読み込みが行われたリソースのパフォーマンスを計測し、
// その読み込み時間をコンソールへ出力する

// リソースのパフォーマンス情報を取得する
var resources = performance.getEntriesByType("resource");
for (i = 0; i < resources.length; i++)
{
   // imgタグが由来だった場合
   if (resources[i].initiatorType == "img")
   {
      // 読み込み時間を計算し、URLとセットで出力する
      var time = resources[i].responseEnd - resources[i].startTime;
      console.log( time + " -> " + resources[i].name);
   }
}
```

間が長いHTTP/2ではロードバランシングを実現するのに使用できる

- **Opportunistic Security**[注4]
  - 相手を認証しないで行う暗号通信のこと。日和見(ひよりみ)暗号とも呼ばれる。近年の広域盗聴行為などにより、プロトコルにおける暗号化に対するモチベーションは高まっている
  - 「http://」のURLでも暗号通信を行う
- **451**[注5]
  - HTTPレスポンスのステータスコードとして「451 Unavailable For Legal Reasons」を追加する仕様
  - 法律上の理由などにより、表示できないときに使用するエラーコード

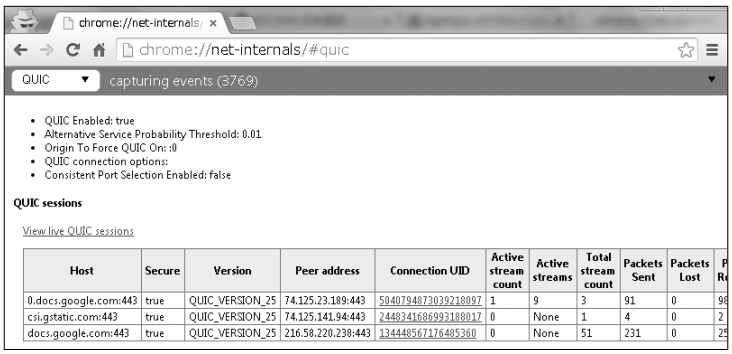
ワーキングドラフトはこれら以外にもあり、IETFのHTTPbisのページ[注6]より確認することができます。

QUIC

HTTPbis WGではありませんが、TSV WG(Transport Area Working Group)では、QUIC(Quick UDP Internet Connections)に関する発表がありました[注7]。また、公式なプログラム外の時間で関係者が集まり議論を行うBar BOFも開かれ、議論が行われました。

QUICは2013年ごろよりGoogleが試験的にブラウザ／サーバに実装したプロトコルであり、

▼図2　QUICによる通信の一覧



Host	Secure	Version	Peer address	Connection UID	Active stream count	Active streams	Total stream count	Packets Sent	Packets Lost
0.docs.google.com:443	true	QUIC_VERSION_25	74.125.23.189:443	5040794873039218097	1	9	3	91	0
csi.gstatic.com:443	true	QUIC_VERSION_25	74.125.141.94:443	2448341686993188017	0	None	1	4	0
docs.google.com:443	true	QUIC_VERSION_25	216.58.220.238:443	13444856717648536O	0	None	51	231	0

すでにChromeやGoogleのサービスでは使用されています。HTTP/2のときと同様、ChromeでURLバーに「chrome://net-internals/#quic」と入力することにより、現在、QUICを使っている通信の一覧を確認できます(**図2**)。GoogleのサービスでQUICが使用されていることが確認できます(通信環境によってはQUICが使用されない場合もあります)。

このQUICもHTTPのメッセージを効率良く転送するためのプロトコルです。その大きな特徴はUDP上のプロトコルであり、ちょうどTCPの機能である通信の信頼性、TLSの機能である通信の安全性といった機能を提供します(**図3**)。QUICでは上位層にはそのままHTTP/2のメッセージを使用します。HTTP/2では仕様としてTCPを使うことになっていますが、QUICというしくみ中で効率的にHTTP/2のメッセージをやり取りします。

- **初回のみ1回のRTT、次回からは0回のRTTでコネクションを確立**
  TCPおよびTLSのハンドシェイクは、初回であれば3RTTが必要になるが、QUICは初回のみ1RTTで、以後は0RTTでの接続も可能としている
- **柔軟な輻輳(ふくそう)制御**
  QUICでは、TCPよりも柔軟な輻輳制御が行える。さまざまな工夫があるが、たとえば、再送パケットに対してオリジナルのものとは別の

注4) URL https://tools.ietf.org/html/draft-ietf-httpbis-http2-encryption-02
注5) URL https://tools.ietf.org/html/draft-ietf-httpbis-legally-restricted-status-01
注6) URL https://datatracker.ietf.org/wg/httpbis/documents/
注7) URL https://www.ietf.org/proceedings/93/slides/slides-93-hopsrg-6.pdf

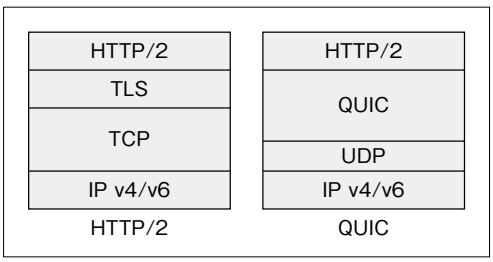
シーケンス番号を付与することで、届いたパケットが再送されたものかどうか判別できる

- **ヘッダとペイロードの認証および暗号化**
  TCPではTCPヘッダは平文でやりとりされる。ウィンドウサイズやシーケンスナンバーも平文でやりとりされている。QUICのパケットは一部を除いて常に暗号化されており、改ざんされていないことを確認することで、通信を保護する
- **前方誤り訂正**
  損失したパケットの再送を待つことなくパケットを回復するために、QUICでは前方誤り訂正(FEC：Forward Error Correction)をサポートしている。FECグループを定義し、そのグループの中で誤りが1つだけであれば、ほかのパケットから損失したパケットをリカバリすることができる
- **接続のマイグレーション**
  TCPのコネクションは、送信元IPアドレス／送信先IPアドレス／送信元ポート番号／送信先ポート番号の4つで値によって識別される。移動中に端末の接続がWi-Fiから4G回線に切り替わった場合、通常ならばIPアドレスが変わりTCPコネクションは切断される。QUICは下位層にUDPを使ったうえで、別途、コネクションIDというものでコネクションの管理を行う。そのため、IPアドレスが変わったとしてもコネクションはそのまま維持できる

詳しくはIETFのドラフトの仕様に記述されています[注8]。

まだまだIETFでもHTTP、Webに関する議論は続くでしょう。次回のIETFの会合は、2015年11月1日より横浜で行われます。日本の方が参加する絶好のチャンスかと思います。興味のある方は参加されてみてはいかがでしょうか。

▼図3 HTTP/2とQUICの比較



The HTTP Workshop

2015年7月23～30日にかけてドイツのミュンスターで、The HTTP Workshopが行われました。このWorkshopは各HTTPにかかわるさまざまなコミュニティ／開発者が、具体的な取り組みやしくみについて議論する場となりました。実際に大手企業、ミドルウェア開発者が参加者として名を連ねています。当日の様子や資料などはGitHub上で公開されており[注9]、誰でも閲覧可能です。

より実践的な議論が多く、h2oやmod_h2などの実装者の発表や、TCPやTLSなどのHTTPの下位層のプロトコルと関連した発表、プロキシ／キャッシュといったHTTPのしくみ／機能に関する発表、セキュリティ／プライバシーの観点の発表、さらにQUICの発表など実にさまざまな発表が行われました。

さらに今後将来へのアイデアとして、キーワードがWikiに記述されています[注10]。

HTTP/3、HTTP/2 over TCP/TLS、HTTP/2 Extensions、HTTP/* Extensionsと見出しを並べるだけでも興味深いものとなっています。

興味のある方はぜひ実際にWebページにアクセスしていただき、今後の議論を追いかけてみてはいかがでしょうか？ SD

注8) URL http://tools.ietf.org/html/draft-tsvwg-quic-protocol-01
URL http://tools.ietf.org/html/draft-tsvwg-quic-loss-recovery-00
注9) URL https://github.com/HTTPWorkshop/workshop/wiki/2015-Report
注10) URL https://github.com/HTTPWorkshop/workshop/wiki/HTTP-Ideas

第2特集

攻撃を最前線で防ぐ

ファイアウォールの教科書

ネットワークの境界に設置し、アクセスを選り分ける“検問所”の役割を担うファイアウォール。Webサービスの提供者、エンドユーザ双方が知っておくべきセキュリティ対策です。

まずは、ファイアウォールを理解するうえで重要な「セキュリティゾーン」の考え方を解説します。次にファイアウォールの2つの機能、パケットを読み取ってポートを開閉するしくみ「ステートフルインスペクション」、プライベートIPとグローバルIPを相互変換する「IPマスカレード」という2つの機能について見ていきます。基礎知識について解説したあとは、パケットフィルタ型のファイアウォールの実践として、Linux標準の「iptables」とRHEL7/CentOS7で新しく登場した「firewalld」の設定方法を紹介します。

最後の章では、パケットのヘッダ・ペイロードを見て攻撃を判別し、Webアプリを守るWAF(Web Application Firewall)について、アプライアンス製品「BIG-IP ASM」の設定を具体例に、導入・運用の勘所をお伝えします。

Part 1

ゾーンの設計がセキュリティ確保の第一歩

ファイアウォールの基礎知識

ネットワークセキュリティを考えるうえで、ファイアウォールの設置は当然のように行われます。それは、対象となる情報が何から守られる必要があるのか、という根本的な考え方を「ゾーンの分離」という形で実現できるためです。ファイアウォールの詳細に踏み込む前に、本章ではこのゾーンの考え方をおさらいしておきましょう。

Author 中井 悦司(なかい えつじ) レッドハット株式会社 Twitter @enakai00

セキュリティゾーンとファイアウォール

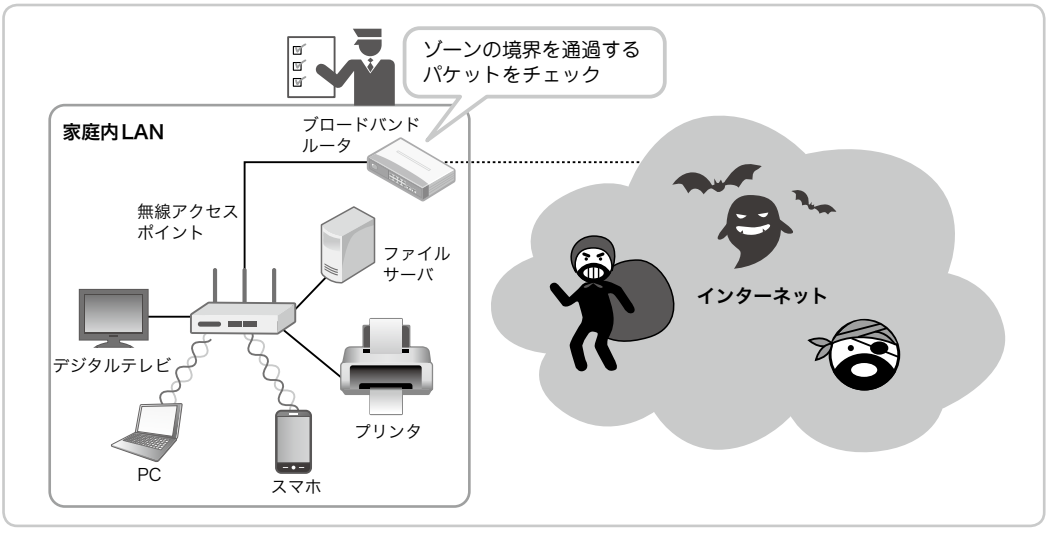
みなさんご存じのとおり、ファイアウォールの役割はネットワークセキュリティの確保です。それでは、ファイアウォールがあると、なぜセキュリティが確保できるのでしょうか？——その答えは、「セキュリティゾーンの分離」にあります。

ブロードバンドルータを介して、家庭内LANをインターネットに接続する場合を考えてみましょう(**図1**)。家庭内にどのような機器が設置されていて、誰が利用するかは、あなた自身で管理することが可能です。一方、インターネット上での活動は、こちらで管理することはできません。中には、悪意を持ってサーバやPCから情報を盗みだそうとする悪者もいるでしょう。家庭内LANが平和な「安全地帯」だとすれば、インターネットはなにが起きるかわからない「ワイルドウェスト」です。

このように、管理レベルの異なる領域、すなわち「セキュリティゾーン」を分離して、それぞれのゾーンの間に「検問所」を設けることがネットワークセキュリティの基礎となります。検問所を通過しようとするネットワークパケットは、係員のチェックを受けて、許可を受けたものだけが通過していきます。家庭内LANのサーバから家計簿ファイルや家族の写真など、大切なデータを盗みだそうとする不正アクセスのパケットは、ここで遮断するというわけです。

そして、この検問所の役割を果たすのがファイアウォールに他なりません。**図1**の例では、ブロードバンドルータに内蔵されたファイア

▼ **図1 ファイアウォールはセキュリティゾーンの「検問所」**



ウォール機能が家庭内LANとインターネットの間を通過するパケットのチェックを行います。ファイアウォールを直訳すると「防火壁」ですが、実際の役割は、隣家の火災から自宅を守るという、単なる防火壁とは少し異なります。

セキュリティゾーンの設計

管理レベルの異なるセキュリティゾーンを分離することが、ネットワークセキュリティの基礎であることはわかりました。それでは、実際には、どのようにゾーンを分割すればよいのでしょうか。

図1の例では、「安全地帯」と「ワイルドウェスト」の2つのゾーンに分割されていますが、これは、**図2**のように表現することができます。正式な用語ではありませんが、「ブルーゾーン」「レッドゾーン」のように管理レベルの違いを色で表現するとわかりやすくなります。家庭内LANからインターネットへの接続は許可するけれど、インターネットから家庭内LANへの接続は禁止するという、最もシンプルな考え方です。

一方、一般的な社内ネットワーク(イントラネット)の環境では、**図3**のような、3種類のゾーン分割が行われます。社内に設置したWebサーバを外部に公開するために、専用のイエローゾーンを用意しています。これは、一般に「DMZ(De-Militarized Zone：非武装地帯)」と呼ばれるゾーンになります。

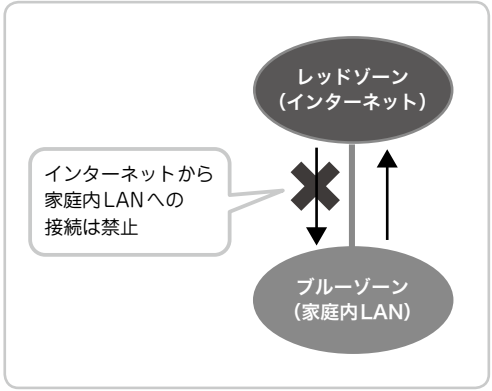
これにより、インターネットから社内への接続は禁止しつつ、必要な情報だけは社外に公開することができます。もちろん、インターネットからDMZへの接続は、すべてのアクセスを許可するのではなく、指定のWebサーバへのアクセスだけを許可します。

また、DMZから社内ネットワークへのアクセスは禁止されますので、万一、Webサーバに侵入されても、そこからさらに社内ネットワークへと侵入することは阻止できます。

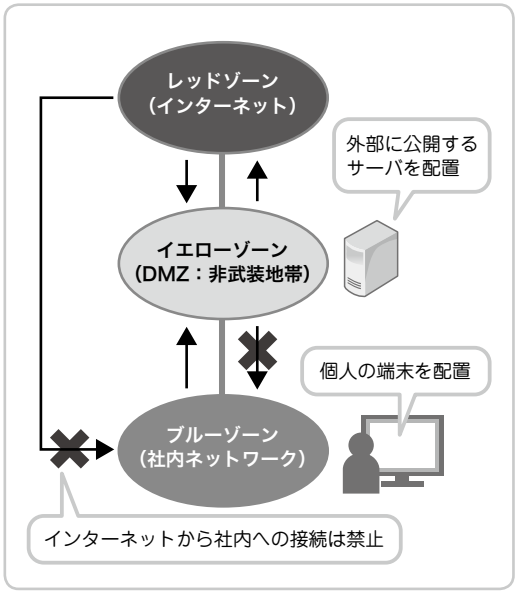
それでは、最後に、勤怠管理システムなど、社内向けのアプリケーションを提供するサーバは、どこに配置するべきでしょうか？　インターネットからアクセスする必要のないサーバですので、イエローゾーン(DMZ)に配置するのは危険です。このような場合、**図4**のように、グリーンゾーンを追加して、そこに配置する方法が考えられます。

ここで、イエローゾーンとグリーンゾーンの相互接続には、注意が必要です。たとえば、外部に公開するWebシステムが、内部的に社内システムと連携する場合があります。社内シス

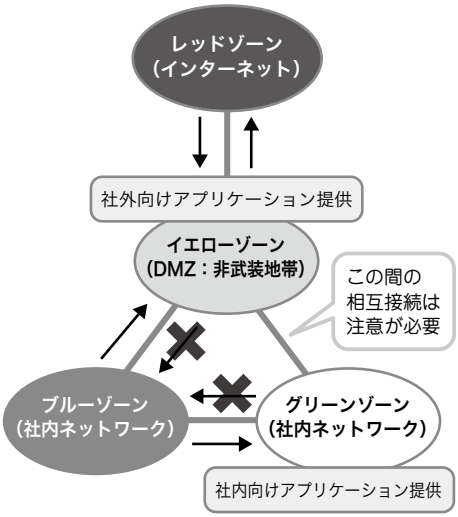
▼図2　最もシンプルなゾーン分割



▼図3　DMZを用いたゾーン分割



▼図4　社内向けシステム用のゾーンを追加した構成



テムに保存されたデータの一部を社外に公開するような場合です。この場合、イエローゾーンからグリーンゾーンへの接続を限定的に許可することも考えられます。ただし、イエローゾーンのサーバに侵入された結果、グリーンゾーンの社内データを盗み出される危険性は高くなります。このような危険性を避けるには、公開が必要なデータをグリーンゾーンからイエローゾーンに定期的に転送するという方法もあります。

このように、ゾーン間でのアクセスをどのように許可するかは、意外と難しい問題です。どのようなデータがどのゾーンで必要なのか、そのデータはどのレベルで保護するべきものなのか、事前にポリシーを定めて管理していく必要があります。

IPパケットの階層構造

ここで、IPパケットの階層構造を復習しておきましょう。これは、この後のファイアウォールの分類を理解するために必要となります。

まず、IPネットワークを流れるパケットには、宛先と送信元のIPアドレスを示した「IPヘッダ」が付与されていることはご存じでしょう。その後ろには、「TCP/UDP/ICMP」などの転送プロトコルに対応したヘッダが続いて、最後に実際のアプリケーションデータを含んだ「ペイロード」があります（**図5**）。

ここで、TCPとUDPの主な違いは、パケット再送機能の有無にあります。TCPヘッダには、「SYN/ACK」などのフラグのほか、送信パケットの順序を示す「シーケンス番号」が記載されています。2つの機器がTCPで通信する際は、**図6**のように、「SYN」「SYN + ACK」「ACK」の3種類のフラグのパケットを交換して、お互いの存在を確認した後に、アプリケーションデータの送信を開始します。これを3ウェイ・ハンドシェイクと呼びます。

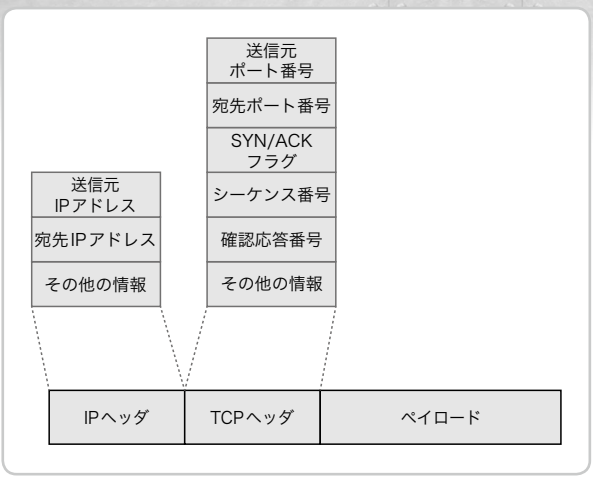
Column　立場で変わるセキュリティゾーンの「危険性」

図4では、社内向けアプリケーションを提供するサーバを専用のグリーンゾーンに配置しました。しかしながら、社内向けのシステムであれば、ブルーゾーンに配置しても構わない気もします。ブルーゾーンへの配置には、何か問題があるのでしょうか？

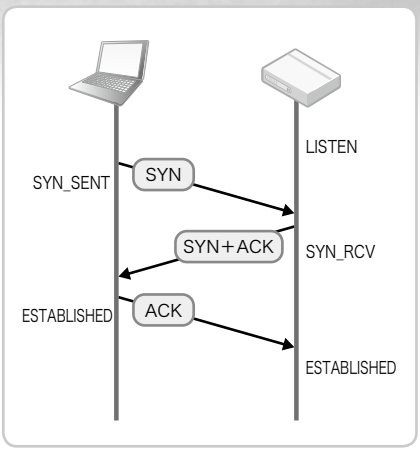
これは、社内システムの管理者の気持ちになればわかります。社内向けのシステムといえども、すべての社員に公開していい情報だけが入っているわけではありません。給与情報などは、第一級（？）の機密情報です。それを一般社員の端末と同じゾーンに配置するわけにはいきません。社内システムの管理者からすれば、ブルーゾーンは、何をしでかすかわからない連中がいる「危険地帯」にほかなりません。きちんとゾーンを分離して、必要なアクセスのみを許可するのが正解です。

このように、セキュリティゾーンの位置づけは、立場によって変わります。それぞれのゾーンの「役割」と「守るべきもの」を明確に定めたうえで、ゾーン間のアクセスポリシーを決定することが大切です。

▼図5　IPパケットの階層構造（TCPパケットの場合）



▼図6　3ウェイ・ハンドシェイクによるTCP接続の開始



その後の通信において、パケットを受け取った側は「確認応答番号」として、受信パケットのシーケンス番号を次に送信するパケットに埋め込んで返信します[注1]。これにより、それぞれの機器は、どのパケットが相手側に到達したかの受信確認ができます。受信確認ができないパケットについては、ネットワーク上でパケットが消失した（あるいは、受信側の機器がパケットを取りこぼした）可能性があるため、同じパケットを自動的に再送します。このためTCP接続は、サーバにログインするSSH接続やWebサーバへのHTTP接続など、接続の安定性が重視されるアプリケーションで利用されます。

一方、UDP接続ではこのような受信確認は行いません。受信側に到達しなかったパケットは、そのまま失われてしまいます。このためUDP接続は、DNSの検索など、交換するパケット数が少ない通信や、動画や音声のストリーミング配信など、一部のパケットが失われても影響の少ないアプリケーションで用いられます。

なお、TCPおよびUDP接続では、SSH、HTTPなどのアプリケーション用途ごとに、サーバ側で使用するポート番号が決まっており、**図5**に示したように、TCP/UDPヘッダには宛先ポート番号と送信元ポート番号が記載されています。

最後のICMPは、pingコマンドによる疎通確認など、ネットワークの状態を確認／通知するために用いられます。一般のアプリケーションデータを転送することはできません。

ファイアウォールの分類

それでは、ここでファイアウォールの分類について解説します。Wikipediaで「ファイアーウォール」の項を参照すると、次のような分類が紹介されています。

① パケットフィルタ型
② サーキットレベルゲートウェイ型
③ アプリケーションゲートウェイ型

まず、動作上の違いで大別すると、①のフィルタ型と②③のゲートウェイ型に分かれます。フィルタ型の場合は、パケットのヘッダの内容を見て、ゾーン間でのパケットの通過を許可するかどうかを決定します。ちょうど、冒頭で説明した「検問所」のイメージに対応するしくみです。

この際、ヘッダの内容をどの程度詳しく分析

注1）正確には、「次に受信を期待するシーケンス番号」を返信します。

するかによって、ファイアウォールとしての機能が変わってきます。たとえば、**図2**の最もシンプルな構成であれば、最低限、次のようなルール設定が必要です。

・家庭内LAN→インターネット：すべてのパケットの通過を許可
・インターネット→家庭内LAN：家庭内LANから送信したパケットに対する返信パケットの通過を許可

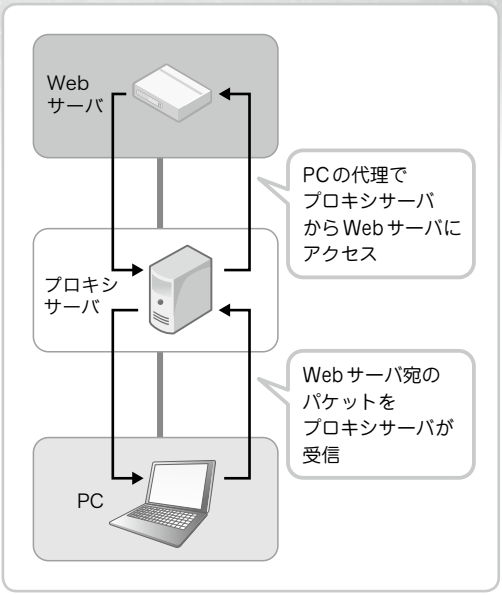
インターネットからの接続については、すべてのパケットを遮断すればよいわけではない点に注意してください。家庭内LANのPCからインターネット上のWebサーバに接続するには、「Webサーバからの返信パケット」については、遮断せずに通過させる必要があります。インターネット側からのパケットが「返信パケット」であることを判定するしくみについては、Part2で解説します。

次のゲートウェイ型は、一般には、「SOCKS」や「プロキシ」と呼ばれるしくみです。SOCKSは、「②サーキットレベルゲートウェイ型」、プロキシは、「③アプリケーションゲートウェイ型」にそれぞれ対応します。これらは、ゾーン間でダイレクトにパケットを通過させる代わりに、中継地点となるサーバが代理でパケットを送受信します。

たとえば、**図3**のDMZを用いた構成において、社内ネットワークのPCからインターネット上のWebサーバに接続する場合を考えます。DMZ内部に「Webプロキシサーバ」を設置して、PC上のWebブラウザが送信するパケットは、Webプロキシサーバが受け取るように設定しておきます[注2]（**図7**）。

PCからのWebサーバ宛てパケットを受け取ったプロキシサーバは、PCの代わりに、あらためて、自分自身からWebサーバにパケットを送信します。そしてWebサーバから受け取った返信パケットを元のPCに返送します。

▼図7　プロキシサーバによるWebアクセスの中継



それでは、なぜこれでセキュリティが保護されるのでしょうか？　たとえばインターネット上のWebサーバからは、PCの情報を盗みだすウイルスが仕込まれたパケットが返送される可能性があります。プロキシサーバにおいて、ウイルスが含まれたパケットをチェックして、怪しいパケットはPCに返送しないようにすることが可能です。Webプロキシサーバは、Webアクセス専用に設計されているので、特定のWebサイトへのアクセスを禁止したり、成人向けのコンテンツを検出して閲覧を禁止するなどの処理も可能になります。

もう一方のSOCKSは、特定のアプリケーションに限定せずに、任意のTCP/UDP接続を中継するように設計されています。DMZに配置したSOCKSサーバが、社内ネットワーク上のPCからインターネットへの接続を中継します。このとき、特定の宛先ポート番号だけ中継を許

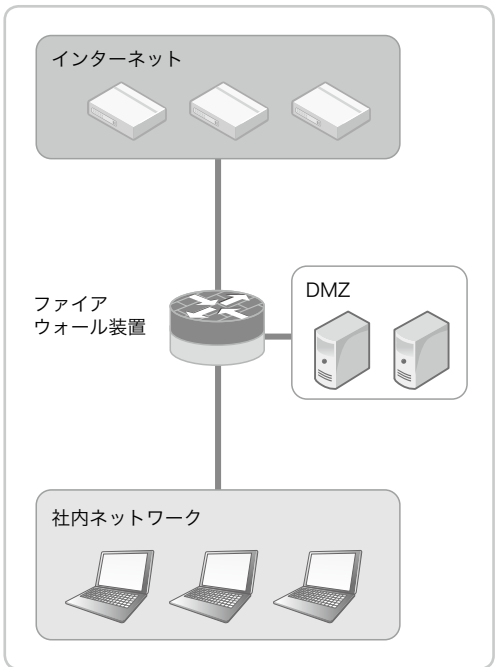
注2）プロキシサーバを使用する際は、PC上のブラウザにおいてプロキシサーバのホスト名を明示的に指定する方法と、インターネット宛のパケットを強制的にプロキシサーバに転送するようにネットワーク上のルータを設定しておく方法があります。後者の場合はPC側の設定が不要で、これを「透過型プロキシ」と呼びます。

可することができます。たとえば、Webサーバへの接続(宛先ポート80番)は許可するけれど、インターネット上のサーバへのSSH接続(宛先ポート22番)は禁止するなどの設定が可能になります。

また、プロキシやSOCKSは、社内ネットワークからインターネットへのアクセスを記録して、インターネット接続に関する監査目的に使用することもできます。あるいは、NAT(IPマスカレード)の代替として利用されることもあります。家庭内LANや社内ネットワークでは、通常、プライベートIPアドレスを使用していますが、送信元IPアドレスがプライベートIPのパケットをインターネットに送信することは禁止されています。

プロキシサーバやSOCKサーバは、グローバルIPアドレスを持っていますので、これらがアクセスを中継することで、プライベートIPの問題を気にする必要がなくなります。プロキシやSOCKSでアクセスを中継しない場合は、Part2で解説するNAT(IPマスカレード)で、送信元IPアドレスをファイアウォールが持つグローバルIPアドレスに変換する設定が必要になります。

▼図8　3脚構成のファイアウォール接続



ファイアウォールの物理接続

最後に、ファイアウォール装置の物理接続について補足しておきます。たとえば、**図3**の構成では、「レッドゾーンとイエローゾーン」、および、「イエローゾーンとブルーゾーン」、それぞれの間にファイアウォールを設置すると考えるかもしれません。そのような構成も可能ですが、実際には、**図8**のように、1つのファイアウォール装置に対して、3つのネットワーク(ゾーン)を接続する構成もよく利用されます。これを「3脚構成」と呼びます。

この構成の場合、ファイアウォールは3つのネットワークを相互接続するルータの役割を果たします。ただし、ファイアウォール内部の設定により、レッドゾーンとブルーゾーンの間では、ルーティングを行わないようにしておきます。ルーティングが許可されたゾーン間でパケットを転送する際に、フィルタリングの処理を行います。

まとめ

Part1では、ネットワークセキュリティの基礎となるセキュリティゾーンの分割について説明しました。また、ファイアウォールの分類として、ゾーン間でパケットをフィルタリングする「パケットフィルタ型」と、プロキシサーバやSOCKSサーバでゾーン間でのパケット転送を中継する「ゲートウェイ型」を紹介しました。

この後のPart2では、とくにパケットフィルタ型のファイアウォールについて、その機能を掘り下げて解説します。その後のPart3、Part4では、Linuxのファイアウォール機能である、iptablesとfirewalldについて、それぞれの使い方や典型的な設定例を紹介していきます。SD

Part 2

返信パケットのみを通過させるしくみ

ファイアウォールの高度な機能

パケットフィルタによる不正アクセスを防ぐしくみを理解するには、パケットがファイアウォールを通過するとき、どのような処理を行っているのかを知る必要があります。本章では、返信パケットのみを通過させるしくみであるステートフルインスペクションと、プライベートIPとグローバルIPを相互変換するIPマスカレードについて解説します。

Author 中井 悦司(なかい えつじ) レッドハット株式会社 Twitter @enakai00

ステートフルインスペクション

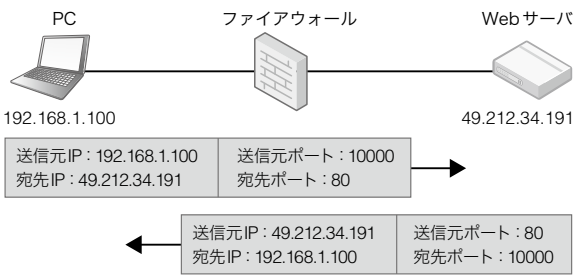
パケットフィルタ型のファイアウォールでは、ファイアウォールを通過しようとする個々のパケットについて、**Part1の図5**(p.63)に示した「IPヘッダ」と「TCPヘッダ」(UDP/ICMPの場合は、それぞれ「UDPヘッダ」と「ICMPヘッダ」)の情報を見ることで、通過を許可するかどうかを決定します。そして、**Part1の図2**(p.61)に示した最もシンプルな構成の場合、最低限、次のルール設定が必要であることを説明しました。

・家庭内LAN→インターネット:すべてのパケットの通過を許可
・インターネット→家庭内LAN:家庭内LANから送信したパケットに対する返信パケットの通過を許可

ここで、インターネット側からのパケットが「返信パケット」であることを判定するしくみが「ステートフルインスペクション」になります。これは、家庭内LANのPCから、インターネット上のWebサーバに向けたパケットが送信されたことを記憶しておき、インターネット側からのパケットが、対応する返信パケットかどうかを判定する機能を提供します。具体的な動作例を示すと、次のようになります(**図1**)。

まず、家庭内LANのPC(IPアドレス:192.168.1.100)がインターネット上のWebサーバ(IPアドレス:49.212.34.191)に最初の接続パケットを送信したとします[注1]。このパケットのヘッダには、送信元/宛先のIPアドレスに加えて、送信元/宛先のポート番号が記録されています。Webサーバへの接続なので、宛先ポート番号は80番です。送信元ポート番号は、PC上の空きポートがランダムに選択されますが、ここでは例として10000番としています。このパケットの通過を許可したファイアウォールは、確立済みセッションの

▼図1 ステートフルインスペクションのしくみ



●ファイアウォールのセッションテーブル

送信パケット	返信パケット
送信元IP:192.168.1.100 送信元ポート:10000 宛先IP:49.212.34.191 宛先ポート:80	送信元IP:49.212.34.191 送信元ポート:80 宛先IP:192.168.1.100 宛先ポート:10000

注1) プライベートIPアドレスを持った家庭内LANのPCからインターネットにアクセスするには、本来は、この後で説明するNAT(IPマスカレード)の処理が必要です。ここでは説明を簡単にするために、プライベートIPのままで通信するものと仮定しています。

情報として、送信元／宛先のIPアドレスと送信元／宛先のポート番号を「セッションテーブル」に記録します[注2]。

この後、Webサーバから返送されるパケットのヘッダは、IPアドレスとポート番号において、送信元と宛先がそっくり入れ替わっているはずです。そこで、このようなヘッダのパケットについては、先ほどのセッションテーブルに記録したパケットに対する「返信パケット」として、家庭内LANへの通過が許可されます。

また、セッションテーブルに記録された情報には、カウントダウンタイマーが付いており、一定時間、該当のセッションに対応するパケット（送信パケット、もしくは、返信パケット）が通過しないと、セッションテーブルから削除されます。該当セッションのパケットが通過するとタイマーがリセットされて、再びカウントダウンが始まります。

このようにしてパケットのヘッダ情報を用いることで、家庭内LANからインターネットへの接続のみが許可されることがわかります。インターネット側からのパケットは、セッションテーブルに記載があるもの以外は通過が許可されないため、インターネット上のサーバから家庭内LANに勝手に接続されることはありません。

ただし、あくまでヘッダの情報を用いているだけですので、Webサーバから返信されたパケットの中身まではチェックされません。怪しげなWebサイトに接続して、ウイルスが仕込まれたパケットが戻ってきた場合でも、そのままPCへと届けられます。パケットの中身（ペイロード部分）を解析してウイルスの存在を検知するには、Part1で紹介したプロキシサーバを用いる必要があります[注3]。

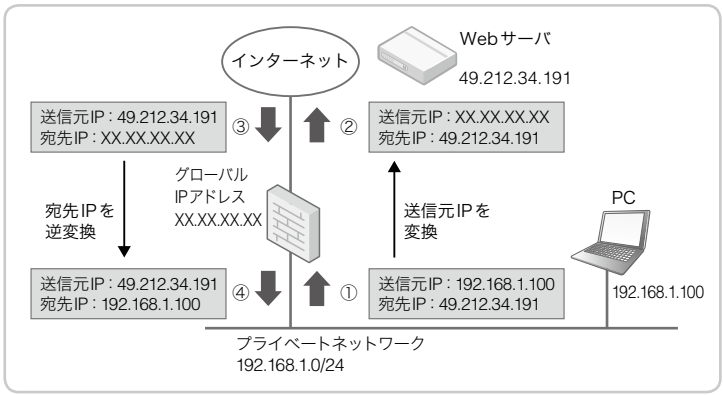
NAT（IPマスカレード）

パケットフィルタ型のファイアウォールが提供する重要な機能に、NAT（Network Address Translation）があります。ここではとくに、プライベートIPアドレスを使用するネットワークからインターネットに接続する際に必要となる、IPマスカレードについて説明します。たとえば、家庭内LANにある多数のPCがブロードバンドルータ経由でインターネットに接続する際は、ブロードバンドルータに割り当てられたグローバルIPを共有してインターネットに接続します。これが、IPマスカレードの機能になります。

IPマスカレードが必要となるのは、送信元IPアドレスがプライベートIPのままでは、インターネットに接続できないためです。そのようなパケットをインターネットに送出することは禁止されており、仮に送出してもインターネットからの応答パケットは返ってきません。

IPマスカレードの概要は、**図2**のようになります。

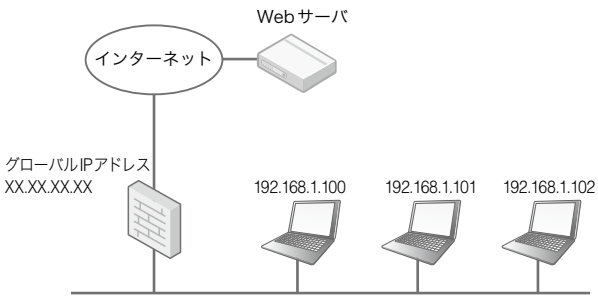
▼図2　IPマスカレードの概要



注2）TCP接続では、Part1の図6（p.63）に示した「3ウェイ・ハンドシェイク」の処理が終わった状態のセッションを「確立済み（ESTABLISHED）」と呼びますが、ここで言う「確立済みセッション」は、これとは関係ありません。最初のパケットが通過した段階で、「確立済み」としてセッションテーブルに記録されます。

注3）たとえば、オープンソースで提供されるウイルス検知ソフトウェアのClamAV（http://www.clamav.net/）とプロキシサーバソフトウェアのSquid（http://www.squid-cache.org/）を組み合わせることで、ウイルス検知に対応したプロキシサーバを構築することができます。

▼図3　IPマスカレードのNATテーブル



NATテーブルの例

送信元IPアドレス	送信元ポート番号	変換後の送信元ポート番号
192.168.1.100	3021	3021
192.168.1.101	4008	4008
192.168.1.102	3021	9000

これは、プライベートネットワーク上のPCがインターネット上のWebサーバにアクセスする想定です。はじめに、送信元IPアドレスにプライベートIP「192.168.1.100」がセットされたパケットがファイアウォールを通過する際に、ファイアウォールは、それを自身の持つグローバルIP「XX.XX.XX.XX」に変換します(①→②)。このパケットを受け取ったWebサーバは、送信元IPと宛先IPを入れ替えて、ファイアウォールのIPアドレス宛にパケットを返信します。これを受け取ったファイアウォールは、今後は、宛先IPアドレスをPCのプライベートIP「192.168.1.100」に逆変換して、PCに送り返します(③→④)。

IPマスカレードは、「送信元IPアドレスを変換する機能」と説明されることもありますが、返信パケットについては、宛先IPアドレスの逆変換も行っている点に注意してください。この際、自身のIPアドレス「XX.XX.XX.XX」に宛てたパケットを受け取ったファイアウォールは、これを正しく逆変換するために、PCのIPアドレス「192.168.1.100」を覚えておく必要があります。

これは、**図3**の「NATテーブル」を用いて行われます。この例では、3種類のプライベートIPを持つPCがインターネットに接続しています。ファイアウォールは、あるPCからのパケットについて送信元IPアドレスを変換する際に、送信元IPアドレスと送信元ポート番号をセットで記憶します。**図3**の一番上の行では「192.168.1.100」と「3021」になります。この後、Webサーバからは、宛先IPアドレス「XX.XX.XX.XX」、宛先ポート番号「3021」の返信パケットが戻るので、宛先ポート番号「3021」を見て、対応するIPアドレス「192.168.1.100」を発見します。

ここで、**図3**のNATテーブルに「変換後の送信元ポート番号」という列がある点に注意してください。それぞれのPCが送信するパケットの送信元ポート番号は各PCがランダムに選択するので、偶然に同じポート番号が使用されることもあります。そうなると、ポート番号から対応するIPアドレスを一意に決めることができなくなります。このような際は、送信元IPアドレスと併せて、送信元ポート番号も変換してパケットを転送します。

図3の例では、一番下の行で、送信元ポート番号「3021」が一番上の行と被っているため、送信元ポート番号をNATテーブル内で未使用の「9000」に変換しています。その後、Webサーバから宛先ポート番号「9000」の返信パケットを受け取ったファイアウォールは、宛先IPアドレスを「192.168.1.102」に変換すると同時に、宛先ポート番号も元の「3021」に変換して、対応するPCにパケットを転送します。

このように、IPマスカレードでは、IPアドレスに加えてポート番号も変換するため、NATではなく、NAPT(Network Address and Port

Column　NATテーブルのポート番号が不足する問題

本文で説明したように、**図3**のNATテーブルには送信元ポート番号を変換した結果が記録されていきます。新たに変換する際はNATテーブル内で未使用のポート番号を選んで変換する必要がありますが、ポート番号の数は有限です。NATテーブルのエントリーは一定時間で削除されるようになっていますが、それでも膨大な数のPCがある環境では、使用できるポート番号が不足してIPマスカレードの処理ができなくなることがあります。

これはIPマスカレードのしくみ上、避けられない問題です。ネットワーク機器の種類によっては、メモリ容量の問題でNATテーブルに記録できるエントリー数が制限される場合もあります。大規模にIPマスカレードを使用する際は、NATテーブルの使用状況に注意を払うようにしてください。

number Translation)と呼ぶこともあります。

WAFとDPI

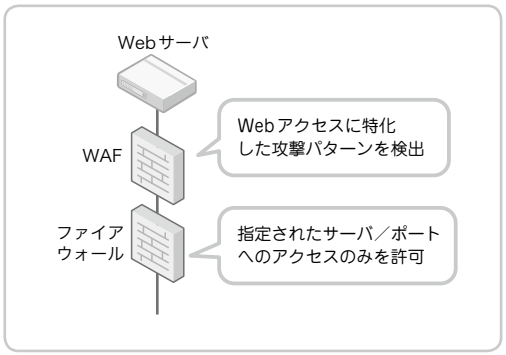
ここまでは、ファイアウォールの役割について、家庭内LANや社内ネットワークをインターネットからの不正アクセスに対して防御するという視点で捉えてきました。一方、Part1のコラム(p.62)でも触れたように、立場が変われば、守るものも変わってきます。インターネットに公開するWebサーバや社内向けのアプリケーションシステムを管理する立場で考えると、家庭内LANや社内ネットワークからやってくるアクセスこそが、予想のつかない危険な存在です。

外部からのアクセスに対して、Webサーバを防御するしくみに、WAF(Web Application Firewall)があります。これはファイアウォールを通過するパケットのヘッダ情報だけではなく、その中に含まれるペイロード部分を解析して、通信内容そのものを分析します。SQLインジェクションやクロスサイトスクリプティングなどの不正アクセスのパターンを発見して、そのようなアクセスを遮断することができます。**図4**のように、通常のファイアウォールの背後に配置して、ファイアウォールで許可されたアクセスに対して、さらに、そのアクセス内容を分析する形で使用します。

あるいは、WAFの考え方を高度に発展させたしくみとして、DPI(Deep Packet Inspection)があります。DPIでは、特別なプログラミング言語を用いて、分析内容そのものを作りこむことが可能です。既知のウイルスの攻撃パターンを発見したり、Webサーバ以外のシステムに対するアクセスを分析することも可能になります。

ただし、DPIの過度な使用については、通信内容の秘密が守られないというプライバシーの問題を引き起こす可能性も指摘されています。実際、各国の政府機関において、検閲や通信傍受のためにDPIが使用されていることもあります。

▼図4　WAFによるWebシステムの保護



まとめ

Part2では、ファイアウォールの機能の中でもとくに、「ステートフルインスペクション」と「NAT(IPマスカレード)」のしくみを詳しく解説しました。次のPart3では、Linuxのiptablesを用いて、これらの機能を実際に適用してみます。WAFについての詳細は、本特集のPart5で取り上げています。SD

Linuxサーバで構築してみよう!

Part 3

iptablesで理解するファイアウォールのしくみ

前章までで、ネットワークを流れるパケットの通過を制御するファイアウォールのしくみがおわかりいただけたでしょうか。本章では、Linuxに標準で備わるiptablesを使ってパケットフィルタリングを行うことで、Linuxサーバを簡易ファイアウォールとして利用する設定方法を解説します。

Author 中井 悦司(なかい えつじ) レッドハット株式会社 Twitter @enakai00

ルータ用Linuxサーバの準備

iptablesは、Linuxが標準機能として提供するパケットフィルタリングのしくみです。Linuxサーバに対する外部からのアクセスを制限するほかに、Linuxサーバを複数のネットワークに接続して、簡易的なファイアウォール装置として使用することも可能です。本章では**図1**のような環境を想定して、iptablesのしくみ／使い方を解説します。

これは、プライベートネットワークにWebサーバとPCを接続して、ルータ用のLinuxサーバを介してインターネットに接続するという構成です。Webサーバとルータ用Linuxサーバのそれぞれで、iptablesを用いたファイアウォールを構成します。Webサーバについては、主にはプライベートネットワークのPCから閲覧することを想定していますが、DNAT(Destination NAT)を用いてインターネットからアクセス可能にする方法も紹介します。Linuxディストリビューションは、Red Hat Enterprise Linux 7(RHEL7)、もしくは、CentOS7を使用します。

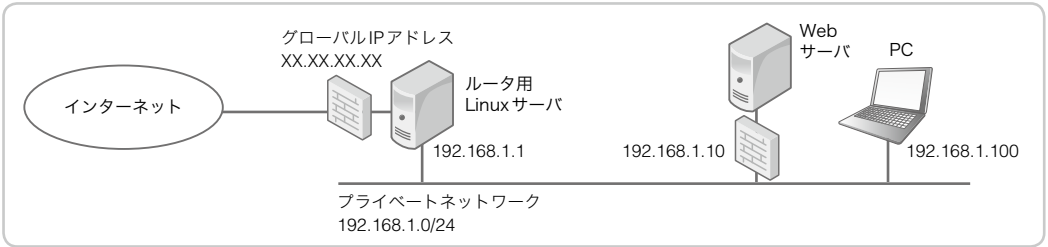
ここではまず、ルータ用Linuxサーバをセットアップします。はじめに、RHEL7/CentOS7を最小構成でインストールして、**図1**のIPアドレスを設定します。インターネット側のグローバルIPアドレスは、環境に合わせて設定してください。

RHEL7/CentOS7のデフォルトでは、firewalldサービスによるパケットフィルタリングが有効になっていますので、ここでは次のコマンドで、iptablesサービスを使用するように変更します[注1]。

```
# yum -y install iptables-services
# systemctl stop firewalld.service
# systemctl mask firewalld.service
# systemctl start iptables.service
# systemctl enable iptables.service
```

さらに、ルータとして使用するためにパケット転送を許可したうえで、iptablesに対して、IPマスカレードの設定を行います。まず、パケット転送を許可するように、カーネルパラメータを設定します。

▼図1 想定するネットワーク構成



注1) ここで「mask」サブコマンドで無効化したfirewalldサービスについて、再度、有効化する際は、「systemctl unmask firewalld.service」を実行します。

```
# echo "net.ipv4.ip_forward=1" >> /etc/sysctl.conf
# sysctl -p
```

次に、IPマスカレードの設定を行います。iptablesの設定内容についてはこの後で解説します。

```
# iptables -F
# iptables -A POSTROUTING -t nat -s 192.168.1.0/24 -j MASQUERADE
# iptables-save > /etc/sysconfig/iptables
```

この後、プライベートネットワーク上のPCにIPアドレスを割り当てて、ルータ用LinuxサーバのIPアドレス「192.168.1.1」をデフォルトゲートウェイに設定すると、インターネットと通信できるようになります。

iptablesの動作原理

ここで、iptablesの基本的なしくみを解説しておきます。iptablesが有効化されたLinuxサーバに出入りするネットワークパケットは、**図2**に示したいくつかの「チェーン」を通過していきます。

図2の左は、**図1**のWebサーバのようにLinux上のアプリケーションが外部ネットワークと通信する場合です。アプリケーションが受信するパケットは「INPUTチェーン」、アプリケーションが送信するパケットは「OUTPUTチェーン」を通ります。**図2の右**は、**図1**のルータ用Linuxサーバのように、2つのネットワークの間でパケットを転送する場合で、「PREROUTING」「FORWARD」「POSTROUTING」のそれぞれのチェーンを通過します。

また、それぞれのチェーンには、設定を記録する各種の「テーブル」が用意されており、設定の種類に応じて、適切な「チェーン」と「テーブル」を選択する必要があります。典型パターンとしては、下記の4種類を覚えておくとよいでしょう。

・アプリケーション受信パケットのフィルタリング
→「INPUTチェーン」の「filterテーブル」
・ルータが転送するパケットのフィルタリング
→「FORWARDチェーン」の「filterテーブル」
・IPマスカレードの設定
→「POSTROUTINGチェーン」の「natテーブル」
・DNATの設定
→「PREROUTINGチェーン」の「natテーブル」

これらのテーブルに設定を追加するには2種類の方法があります。1つは、iptablesコマンドで現在有効な設定を直接に変更した後に、その結果を次のコマンドで設定ファイルに保存します。

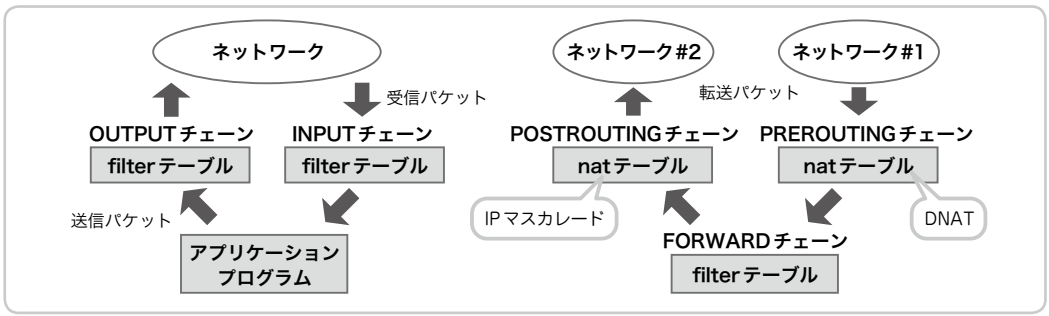
```
# iptables-save > /etc/sysconfig/iptables
```

もう1つは、設定ファイル/etc/sysconfig/iptablesをエディタで編集した後に、次のコマンドで設定変更を反映します。

```
# systemctl restart iptables.service
```

ここでは設定ファイルを編集する方法を用いて説明を進めます。先ほどの手順でセットアップしたルータ用Linuxサーバでは、/etc/sysconfig/iptablesは、**リスト1**のようになっています(コメント行は省略)。テーブルごとにセクショ

▼図2　iptablesのチェーンとテーブル



▼リスト1　iptablesの設定ファイルの例（/etc/sysconfig/iptables）

```
*nat
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
-A POSTROUTING -s 192.168.1.0/24 -j MASQUERADE
COMMIT

*filter
:INPUT ACCEPT [10:776]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [6:736]
COMMIT
```

natテーブルの設定
各チェーンのポリシー
IPマスカレードの設定
filterテーブルの設定
各チェーンのポリシー

ンが分かれており、それぞれのセクションの先頭には、各チェーンの「ポリシー」が指定されています。これは、どのパケットフィルタリング条件にもマッチしなかったパケットの処理方法を指定するのものです。この例では、すべて「ACCEPT」になっており、パケットフィルタリング条件で明示的に拒否されなかった場合は、そのパケットの通過が許可されます[注2]。

ルータ用Linuxサーバの設定例

それでは、ルータ用Linuxサーバのiptablesについて、必要な設定を追加していきます[注3]。**リスト1**では、IPマスカレードの設定がすでに用意されていますが、これは冒頭の準備作業において、iptablesコマンドで設定したものになります。設定ファイル/etc/sysconfig/iptablesでは、各テーブルのセクションに対して、次の書式でエントリーを追加します。

```
-A <チェーン> <マッチング条件> -j <アクション>
```

図2に従ってそれぞれのチェーンを通過するパケットは、該当チェーンのエントリーについて、設定ファイルに記載した順にマッチング条件が評価されます。そして、最初にマッチしたエントリーのアクションが適用されます。

リスト1にある「IPマスカレードの設定」では、マッチング条件として「-s」オプションで送信元IPアドレスのサブネットを指定しています。プライベートネットワーク「192.168.1.0/24」から送信されたすべてのパケットが条件にマッチします。そしてアクションの「MASQUERADE」は、IPマスカレードを適用するという指示になります。そのほかに指定できる主なアクションは、**表1**になります。

設定を追加する

ここで、プライベートネットワークからインターネットへのパケット転送のみを許可して、インターネットからプライベートネットワークへのパケット転送を拒否する設定を追加します。filterテーブルのセクションに対して、**リスト2**の設定を追加した後に、先ほどのsystemctlコマンドで設定変更を反映します。

```
# systemctl restart iptables.service
```

リスト2では①～③の設定を追加していますが、それぞれの意味は次のとおりです。まず、②は送信元IPアドレスのサブネットが「192.168.1.0/24」、すなわち、プライベートネットワークからの転送パケットを許可しています。①は、Part2で説明した「ステートフルインスペクション」に関連するものです。プライベートネットワークのPCがインターネット上のWebサーバに接続する場合、Webサーバからの返信パケットはプライベートネットワークに転送を許可する必要があります。①のマッチング条件（「-m」オプション、および「--state」オプション）は、このような返信パケットにマッチするもので、これで返信パケットの転送を許可しています。

注2）ポリシーの後ろにある[0:0]などの数字は、iptablesが処理したパケットの統計情報を示します。iptablesの動作には影響ありませんので、この数字はとくに変更する必要はありません。
注3）ここでは、説明を簡単にするためにパケット転送のフィルタリングのみを設定しています。実環境で使用する際は、ルータ用Linux自身に対する接続も制限する必要があるので注意してください。

▼リスト2　一方向のパケット転送のみを許可する設定(/etc/sysconfig/iptables)

```
*filter
:INPUT ACCEPT [10:776]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [6:736]
-A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT  ①
-A FORWARD -s 192.168.1.0/24 -j ACCEPT                    ②
-A FORWARD -j REJECT                                      ③
COMMIT
```

▼表1　iptablesの主なアクション

アクション	説明
ACCEPT	パケットを許可
REJECT	パケットを拒否
DROP	パケットを破棄
LOG	パケットの情報をロギング
MASQUERADE	IPマスカレードを適用
DNAT	DNATを適用

①②のどちらにもマッチしなかったパケットは、③のエントリーで転送が拒否されます。③にはマッチング条件が指定されていないため、すべてのパケットがマッチします。

Webサーバ用の設定例

続いて、**図1**のWebサーバについてiptablesを設定していきます。動作確認用に、次のコマンドでApache HTTPサーバを起動しておきます。

```
# yum -y install httpd
# systemctl start httpd.service
# systemctl enable httpd.service
```

続いて、ルータ用Linuxサーバと同様に、デフォルトのfirewalldサービスから、iptablesサービスを使用するように変更します。

```
# yum -y install iptables-services
# systemctl stop firewalld.service
# systemctl mask firewalld.service
# systemctl start iptables.service
# systemctl enable iptables.service
```

▼リスト3　iptablesのデフォルト設定(/etc/sysconfig/iptables)

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT          ①
-A INPUT -p icmp -j ACCEPT                                      ②
-A INPUT -i lo -j ACCEPT                                        ③
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT ④
-A INPUT -j REJECT --reject-with icmp-host-prohibited           ⑤
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

このとき、iptablesのデフォルトの設定ファイルは、**リスト3**のようになっています。INPUTチェーンのfilterテーブルに対する設定で、このサーバ上のアプリケーションへの接続パケットをフィルタリングしています。

まず、①は、**リスト2の**①と同じで、このサーバから送信したパケットに対する返信パケットを許可するものです。②③はそれぞれ、ICMPパケットとループバックアドレス(127.0.0.1)宛のパケットを許可します。ループバックアドレスは、同じサーバ上のアプリケーションが相互に通信する際に使用するアドレスです。

④は、SSHサーバに対する最初の接続パケットを許可します。「-p tcp」はTCPパケット、「-m state --state NEW」は最初の接続パケット、そして、「-m tcp --dport 22」はTCP22番ポート宛のパケットにマッチします。ここで、「最初の接続パケット」というのは、Part2のステートフルインスペクションで説明した「セッションテーブル」にエントリーのないパケットという意味です。最初の接続パケットが通過すると対応するエントリーがセッションテーブルに記録されるので、それ以降のパケットについては①の条件で許可されるようになります[注4]。

設定を追加する

Webサーバ用に設定を追加したものが**リスト**

注4) iptablesのセッションテーブルの内容は、特殊ファイル「/proc/net/nf_conntrack」をcatコマンドで表示することで確認できます。

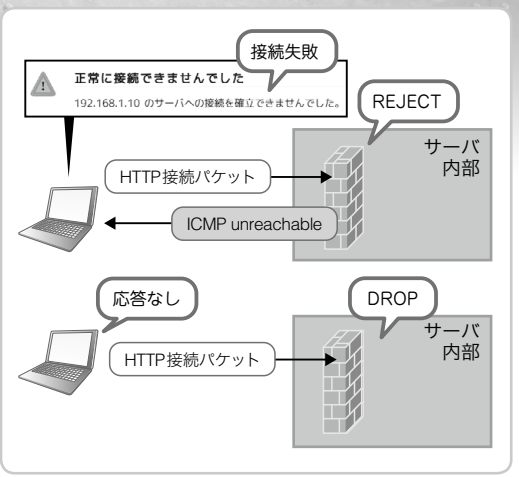
4になります。⑥⑦が追加部分で、その前後は、追加する場所がわかるように記載してあります。⑥は、Webサーバの80番ポート(HTTP接続)、および443番ポート(HTTPS接続)への接続を許可します。ここでは、「-m multiport」オプションを使用することで、複数のポートをまとめて記載しています。

⑦は、受信を許可されなかったパケットについて、その情報をシステムログ(/var/log/messages)に記録するエントリーです。必須ではありませんが、不正アクセスの状況をチェックするために利用できます。アクションが「LOG」のエントリーにマッチしたパケットは、その情報をシステムログに記録した後、さらに次のエントリーへと評価が進みます。なお、「-m limit」は、ログの出力が連続して発生した場合に、一時的に出力を抑制するオプションです。不正なパケットを大量に受信した際に、ログ出力でサーバの性能が低下するのを避けるために付けてあります。

受信を許可されなかったパケットは、最終的に**リスト3の**⑤のエントリーにマッチして、受信を拒否されます。アクション「REJECT」でパケットを拒否した場合、パケットの送信元に対して、受信拒否を示すICMPメッセージが返送されます。ここでは、「--reject-with」オプションでメッセージの種類を指定しています。デフォルトでは、「icmp-proto-unreachable」が送信されます。

表1を見ると、「REJECT」のほかに「DROP」というアクションがあります。これらはどちらもパケットの受信を拒否しますが、DROPを指定した場合はICMPメッセージは返送しません。送信パケットは黙って破棄されることになります(**図3**)。WebブラウザからWebサーバに接続して拒否された場合、REJECTであれば拒否メッセージが送信されるのでブラウザにはエラーメッセージが表示されます。一方、DROPの場合、Webサーバからは何も応答がないため、ブラウザはタイムアウトするまで応答待ち状態になります。セキュリティ保護の観点では、Webサーバの存在すらわからなくするという意味で、DROPのほうがセキュリティ的に強固な設定と言えます。

▼図3 REJECTとDROPの違い



また、⑤のエントリーの代わりに、どの条件にもマッチしなかったパケットは該当チェーンのポリシー(デフォルトのアクション)で拒否するという方法もあります。ただし、ポリシーで拒否する場合は、DROPのみが指定できます。REJECTで拒否したい場合は、⑤のようにREJECTを指定したエントリーを明示的に追加する必要があります。

DNATの設定

最後に、DNATについて補足しておきます。ルータ用LinuxサーバにDNATの設定を追加すると、**図1**のWebサーバにインターネットから

▼リスト4 Webサーバ用の追加設定(/etc/sysconfig/iptables)

```
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp -m multiport --dports 80,443 -j ACCEPT  ⑥
-A INPUT -j LOG -m limit --log-prefix "[INPUT Dropped] "  ⑦
-A INPUT -j REJECT --reject-with icmp-host-prohibited
```

▼リスト5　DNATの追加設定

natセクションに記載(任意の場所に追加)

```
-A PREROUTING -p tcp -d XX.XX.XX.XX -m tcp --dport 80 -j DNAT --to-destination 192.168.1.10
```

filterセクションに記載(リスト2の②の直後に追加)

```
-A FORWARD -p tcp -d 192.168.1.10 -m tcp --dport 80 -j ACCEPT
```

接続できるようになります。具体的には、ルータ用Linuxサーバの設定ファイルのnatセクションとfilterセクションに、**リスト5**のエントリーを追加します。natセクションのエントリーは、宛先IPアドレスがルータ用LinuxサーバのTCPパケット(-p tcp -d XX.XX.XX.XX)で、かつ宛先ポート番号がTCP80番(-m tcp --dport 80)のものについて、宛先IPアドレスをWebサーバ(--to-destination 192.168.1.10)に変換して転送するという指定です。filterセクションのエントリーは、WebサーバのTCP80番ポート宛のパケットについて、転送を許可しています。

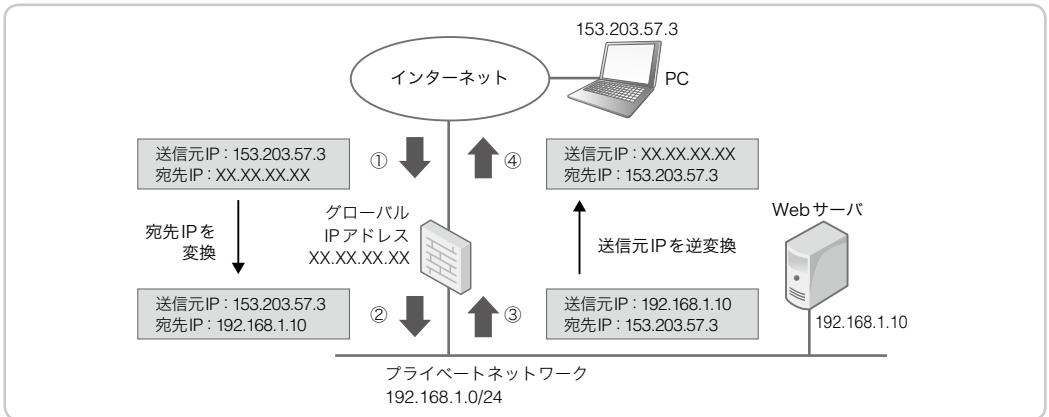
これにより、インターネット上のブラウザからルータ用LinuxサーバのTCP80番ポートに接続すると、プライベートネットワーク上のWebサーバにパケットが転送されて、その応答を受け取ることができます。**図4**に示したように、宛先IPアドレスを変換してWebサーバにパケットを転送した後、Webサーバからの返信パケットは送信元IPアドレスをルータ用LinuxサーバのIPアドレスに逆変換します。その結果、外部のブラウザからは、ルータ用LinuxサーバがWebサーバであるかのように見えることになります。ここでは、TCP80番ポート宛のパケットのみを転送していますが、TCP443番ポート宛のパケットも転送する際は、同様のエントリーを追加して対応します。

まとめ

Part3では、Linuxのiptablesを用いたファイアウォール機能の設定例を紹介しました。iptablesを用いると、**図1**のように、複数のネットワークを相互接続するファイアウォールとして利用したり、あるいは、単体のWebサーバについて外部からのアクセスを制御することが可能になります。Linuxサーバのセキュリティ保護の基本となる機能ですので、しっかりとマスターして使いこなしてください。

続くPart4では、RHEL7/CentOS7で新しく登場した、firewalldサービスについて解説を行います。SD

▼図4　DNATの動作のしくみ



Part 4

ゾーンごとの柔軟な設定が可能に

ファイアウォールの新機能「firewalld」をマスター

Red Hat Enterprise Linux 7からデフォルトのファイアウォール機能となったfirewalldは、ゾーンにあわせたフィルタリング条件の設定・変更がやりやすくなりました。Part1で説明したファイアウォールの基本であるゾーンの考え方にも重なり、より管理がしやすくなったと言えるでしょう。本章ではこのfirewalldの使い方を解説します。

Author 中井 悦司(なかい えつじ) レッドハット株式会社 Twitter @enakai00

firewalldのサービスモデル

Part3で触れたように、RHEL7/CentOS7では、デフォルトではfirewalldサービスによるファイアウォール機能が有効になっています。firewalldサービスは、その背後ではiptablesを利用していますが、設定における考え方はまったく異なります。iptablesをバックエンドにした、新たなファイアウォール機能を提供するものと考えるとよいでしょう。

最近のサーバは、複数のNICを用いて、複数のネットワークに同時に接続することがよくあります。このような際に、接続先のネットワークごとにファイアウォールの設定を変えたくなることがあります。たとえば、サービス用ネットワークからはWebサーバ(TCP80番ポート)へのアクセスのみを許可して、管理用ネットワークからはすべてのアクセスを許可するような場合です。

firewalldサービスでは、**図1**のように、サーバに搭載したNICのインターフェースごとにファイアウォールを設置して、それぞれに設定を適用していきます。この際、設定内容をまとめた「ゾーン」を用意しておき、それぞれのインターフェースに対して適用するゾーンを指定します。

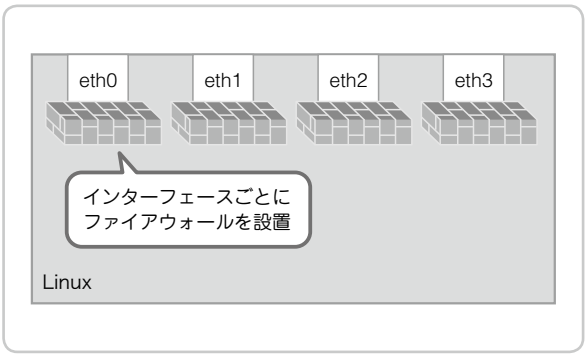
ただし、firewalldサービスは設定の考え方が異なるだけで、従来のiptablesサービスではできなかった機能が追加されるわけではありません。これまでの設定方法に慣れている場合は、Part3の冒頭で紹介した方法で、iptablesサービスに切りかえて使用しても問題ありません。

firewalldサービスはデフォルトで有効化されていますが、手動で有効化/無効化する場合は次のsystemctlコマンドを使用します。

```
# systemctl enable firewalld.service
# systemctl disable firewalld.service
# systemctl start firewalld.service
# systemctl stop firewalld.service
# systemctl restart firewalld.service
```

enable/disableはシステム起動時のサービス自動起動を有効化/無効化するもので、start/stop/restartはその場でサービスを起動/停止/再起動します。

▼図1 firewalldサービスの設定モデル



ゾーンの考え方

デフォルトで用意されているゾーンには、**表1**のものがあります。「drop」「block」「trusted」以外のゾーンは設定を

▼表1　デフォルトで用意されているゾーン

ゾーン	説明
drop（設定変更不可）	あらゆるパケットを黙って破棄する。内部から外部にパケットは出せるが、返信パケットが破棄されるので実質的には通信できない
block（設定変更不可）	あらゆるパケットを受信拒否して、ICMP Prohibitedメッセージを返す。内部から開始した通信の返信パケットは通るので、内部から外部への通信はできる
public	デフォルトでは「ssh」「dhcpv6-client」のみが許可される
external	デフォルトでは「ssh」のみが許可される。IPマスカレードが有効
dmz	デフォルトでは「ssh」のみが許可される
work	デフォルトでは「ssh」「ipp-client」「dhcpv6-client」のみが許可される
home	デフォルトでは「ssh」「ipp-client」「mdns」「samba-client」「dhcpv6-client」のみが許可される
internal	デフォルトでは「ssh」「ipp-client」「mdns」「samba-client」「dhcpv6-client」のみが許可される
trusted（設定変更不可）	あらゆるパケットが受信許可される

▼図2　デフォルトで用意されているサービスとICMPタイプ

```
# firewall-cmd --get-services ↵ Enterの入力（以下同）
RH-Satellite-6 amanda-client bacula bacula-client dhcp dhcpv6 dhcpv6-client dns ftp high-
availability http https imaps ipp ipp-client ipsec kerberos kpasswd ldap ldaps libvirt libvirt-
tls mdns mountd ms-wbt mysql nfs ntp openvpn pmcd pmproxy pmwebapi pmwebapis pop3s postgresql
proxy-dhcp radius rpc-bind samba samba-client smtp ssh telnet tftp tftp-client transmission-
client vnc-server wbem-https

# firewall-cmd --get-icmptypes ↵
destination-unreachable echo-reply echo-request parameter-problem redirect router-
advertisement router-solicitation source-quench time-exceeded
```

変更することができるので、必要な際はこれらの設定を変更して使用します。独自のゾーンを定義ファイルで追加することもできますが、ほとんどの場合は既存のゾーンを設定変更するだけで十分でしょう。ゾーンの名前から、それぞれのゾーンの使用目的が決まっていると思うかもしれませんが、実際の使い方に制限はありません。

デフォルトで用意されたゾーンの定義ファイルは、ディレクトリ「/usr/lib/firewalld/zones」の下にあります。「public.xml」のようにゾーン名に対応するファイル名のXMLファイルが用意されています。定義ファイルの書式は、manページ「firewalld.zone」に記載があります。

独自のゾーンを追加する場合は、ディレクトリ「/etc/firewalld/zonez」の下に同じ書式で定義ファイルを作成します。ゾーンだけに限らず、一般に定義ファイルを追加／変更した際は、次のコマンドで変更を反映します。

```
# firewall-cmd --reload
```

それぞれのゾーンでは、接続を許可する「サービス」と受信を禁止するICMPタイプ、そして、IPマスカレードの有効／無効を指定します。TCP/UDP接続については、明示的に許可したサービスのみに接続できて、ICMPについては、明示的に禁止しないものはデフォルトで許可される形になります。

デフォルトで定義されているサービスとICMPタイプは、**図2**のコマンドで確認することができます。それぞれのサービスの定義ファイルはディレクトリ「/usr/lib/firewalld/services」の下にあるので、具体的な設定内容を確認する際は、定義ファイルの内容を参照することができます。リ

▼リスト1　サービス「ssh」の定義ファイル（/usr/lib/firewalld/services/ssh.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>SSH</short>
  <description>Secure Shell (SSH) is a protocol for logging into and executing commands on
remote machines. It provides secure encrypted communications. If you plan on accessing your
machine remotely via SSH over a firewalled interface, enable this option. You need the openssh-
server package installed for this option to be useful.</description>
  <port protocol="tcp" port="22"/>
</service>
```

▼リスト2　サービス「ftp」の定義ファイル（/usr/lib/firewalld/services/ftp.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>FTP</short>
  <description>FTP is a protocol used for remote file transfer. If you plan to make your FTP
server publicly available, enable this option. You need the vsftpd package installed for this
option to be useful.</description>
  <port protocol="tcp" port="21"/>
  <module name="nf_conntrack_ftp"/>
</service>
```

▼図3　各ゾーンの設定確認

```
# firewall-cmd --list-all-zones ↵
block
  interfaces:
  sources:
  services:
  ports:
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:
…（中略）…
public (default, active)
  interfaces: eth0 eth1 ←このゾーンを適用したインターフェース
  sources:
  services: dhcpv6-client ssh
  ports:
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:
…（以下省略）…
```

リスト1の例にあるように、サービス名に対応するファイル名のXMLファイルが用意されています。この例では、「sshサービス」はTCP22番ポートへの接続に対応することがわかります。

独自のサービスを追加する際は、ディレクトリ「/etc/firewalld/services」の下に同じ書式で定義ファイルを追加します。定義ファイルの書式は、manページ「firewalld.service」に記載があります。iptablesでは、機能拡張用のカーネルモジュール（ヘルパーモジュール）が使用できますが、サービスの定義ファイル内で使用するヘルパーモジュールを指定することもできます。たとえば、**リスト2**に示した「ftpサービス」では、アクティブモードのFTP通信に対応するためのヘルパーモジュール「nf_conntrack_ftp」が指定されています[注1]。

注1） ヘルパーモジュール「nf_conntrack_ftp」の役割については、次のWeb記事を参考にしてください。
『「iptablesとFTP」の困った関係』（http://www.school.ctc-g.co.jp/columns/nakai/nakai40.html）

▼図4　デフォルトゾーンの確認と変更

```
# firewall-cmd --get-default-zone ↵ ←デフォルトゾーンの確認
public
# firewall-cmd --set-default-zone=external ↵ ←デフォルトゾーンの変更
success
```

そして最後に、それぞれのゾーンの既存設定は、**図3**のコマンドで確認します。この例では、インターフェースeth0とeth1に対して、ゾーン「public」が適用されていることがわかります。

ゾーンの適用手順と設定変更手順

NICのインターフェースにゾーンを適用する手順を説明します。まず、デフォルトゾーンが1つ決められており、新しく追加したインターフェースなど、明示的にゾーンが指定されない場合はデフォルトゾーンが適用されます。デフォルトゾーンの確認と変更は、**図4**のコマンドで行います。

明示的にゾーンを指定する場合は、ディレクトリ「/etc/sysconfig/network-scripts」の下にあるインターフェースの設定ファイルに「ZONE=<ゾーン名>」という指定を追加します。たとえば、ゾーン「external」をeth1に適用する場合は、設定ファイルifcfg-eth1に「ZONE=external」を追加します。その後、サーバを再起動するか、次のコマンドで設定変更を反映します。

```
# nmcli c reload
# nmcli c down eth1; nmcli c up eth1
```

ゾーンで許可するサービス、および禁止するICMPタイプを変更するには、**図5**のコマンドを使用します。「--zone」オプションを省略した場合は、デフォルトゾーンが対象になります。

なお、**図5**のコマンドは実行中の設定を変更するだけで、サーバを再起動すると設定は元に戻ります。起動時の設定を変更する際は、「--permanent」オプションを追加してください。実行時と起動時の両方を変更する際は、「--permanent」オプションあり／なしの両方でコマンドを実行する必要があります。

また、Part3の「ルータ用Linuxサーバ」のようにルータとして使用する場合は、IPマスカレード、およびDNATの設定をゾーンに追加する

▼図5　ゾーンの定義変更手順

```
# firewall-cmd --list-services --zone=public ↵ ←許可されているサービスを表示
dhcpv6-client ssh
# firewall-cmd --add-service=http --zone=public ↵ ←許可するサービスを追加
success
# firewall-cmd --query-service=http --zone=public ↵ ←指定のサービスが許可されているか確認
yes
# firewall-cmd --remove-service=http --zone=public ↵ ←許可するサービスを削除
success

# firewall-cmd --list-icmp-blocks --zone=public ↵ ←禁止されているICMPタイプを表示
# firewall-cmd --add-icmp-block=echo-request --zone=public ↵ ←禁止するICMPタイプを追加
success
# firewall-cmd --query-icmp-block=echo-request --zone=public ↵ ←指定のICMPタイプが禁止されているか確認
yes
# firewall-cmd --remove-icmp-block=echo-request --zone=public ↵ ←禁止するICMPタイプを削除
success
```

▼図6　IPマスカレードとDNATの設定

IPマスカレードの設定

```
# firewall-cmd --query-masquerade --zone=public ↵ ←現在の設定を確認
no
# firewall-cmd --add-masquerade --zone=public ↵ ←IPマスカレードを有効化
success
# firewall-cmd --remove-masquerade --zone=public ↵ ←IPマスカレードを無効化
success
```

DNATの設定

```
# firewall-cmd --list-forward-ports --zone=public ←現在の設定を確認
# firewall-cmd --add-forward-port=<変換ルール> --zone=public ←変換ルールを追加
# firewall-cmd --query-forward-port=<変換ルール> --zone=public ←変換ルールの存在を確認
# firewall-cmd --remove-forward-port=<変換ルール> --zone=public ←変換ルールを削除
```

こともできます。IPマスカレードについては、該当ゾーンを適用したインターフェースから送信するパケットについて適用されます。**Part3の図1**であれば、インターネット側に接続したインターフェースに、IPマスカレードを有効化したゾーンを適用します。DNATについては、該当ゾーンを適用したインターフェースで受信したパケットについて適用されます。具体的な設定手順は、**図6**のようになります。

DNATの変換ルールは、たとえば、TCP80番ポート宛のパケットを192.168.1.10のWebサーバに転送する場合は次のようになります。

```
port=80:proto=tcp:toaddr=192.168.1.10
```

実際にオプションを書くと「--add-forward-port=port=80:proto=tcp:toaddr=192.168.1.10」という変な書き方になりますが、これで正しく設定されます。設定を永続化する際は、**図5**のコマンドと同様に「--permanent」オプションが必要です。

Part3のルータ用Linuxサーバであれば、インターネット側のインターフェースに「ZONE=external」、プライベートネットワーク側のインターフェースに「ZONE=internal」を適用した後、次の設定を行うことで、IPマスカレード、およびDNATの設定を再現することができます。設定を永続化する際は、「--permanent」オプションを付けて実行してください。

```
# firewall-cmd --add-forward-port=port=
80:proto=tcp:toaddr=192.168.1.10 --zone
=external
```

なお、Linuxサーバをルータとして使用する際は、パケット転送を許可するようにカーネルパラメータ「net.ipv4.ip_forward=1」を設定する必要があります。firewalldでは、このパラメータは必要に応じて自動設定されるようになっています。

まとめ

Part4では、firewalldサービスを使ったファイアウォールの設定方法を紹介しました。firewalldサービスでは、このほかにも「Rich Language」と呼ばれる特別な書式を用いることで、iptablesと同等の詳細な設定を行うことができます。ただし、firewalldサービスのメリットは、ゾーンを用いて、インターフェースごとのフィルタリング条件を簡単に設定できることにあります。複雑な設定を行う場合は、従来どおり、iptablesサービスを使用するほうがよいでしょう。SD

Part 5

設置したら終わり、ではない

チューニングしながら運用するWAF

世界中の攻撃的なアクセスからWebアプリを守る必要がある「WAF」。本章では、Webアプリに対するいくつかの脅威を紹介したあと、アプライアンス製品「BIG-IP」の機能「Application Security Manager」の設定を具体例に、WAFの導入・運用について解説します。

Author F5ネットワークスジャパン株式会社　鈴木 賢剛(すずき ただよし)

なぜWAFが必要か

Webアプリケーションを守るファイアウォール

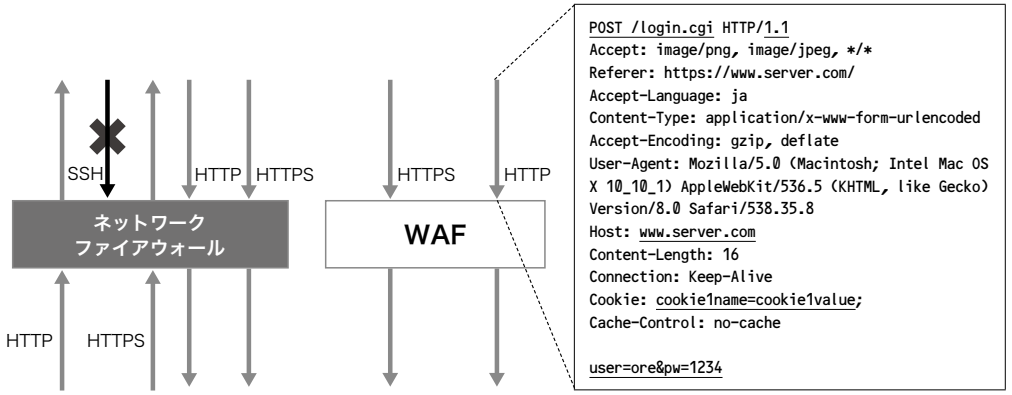
通常のネットワークファイアウォールはネットワーク境界に置かれ、IPアドレスとポートに対してACL(アクセス制御リスト)を設定し、許可した通信のみを通す動作をします。これに対してWebアプリケーションファイアウォール(以下、WAF)は、Webアプリケーション(以下、Webアプリ)で使われるHTTPの通信の中身(ヘッダおよびペイロード)を見て、あらかじめ設定された「セキュリティポリシー」に従ってアクセスの可否を判断するWebアプリ専用のファイアウォールです(**図1**)。

Webアプリをセキュアにする難しさ

Webアプリは、全世界に公開されるWebサーバ上で動作します。この特性上、動作させたら24時間365日全世界にさらされる状態になり、また、いつでもどこからでも入力を受け付けるしくみになっています。

Webアプリをセキュアにするためにセキュアコーディングを実施することは簡単ではありません。あらゆる値が改ざんされて送られてくる可能性があり、想定外のデータを受けた場合にも適切な対処を施すためのプログラムをきちんと作っておく必要があります。また、セキュアコーディングが適切になされていることを証明することも簡単ではありません。ある程度は脆弱性診断で行うことができますが、機械的な診断ツールでは見つけられないものもあり、専

▼図1　ネットワークファイアウォールとWAF



ネットワークファイアウォールで許可されているHTTP(S)通信の"内容"をチェックする

門家によるコードレベルでのレビューが理想的です。しかし、Webアプリに変更を加えるごとに行う必要があったり、定期的に行わないと新しい脆弱性の問題に対処できなかったりといった問題もあり、“どこまで何をやっても確実にならない”性質のものだと言えます。

現在ではさまざまな電子商取引(Eコマース)のWebサービスがあり、ユーザ名とパスワードによって誰がアクセスしているのかを特定するしくみが設けられています。その裏側にはユーザ名とパスワードといった資格情報、住所、氏名やクレジットカード番号などの個人情報がデータベースに置かれています。これに関連して、クレジットカードの加盟店・決済代行事業者が取り扱う会員のカード情報・取引情報を守るために、大手カード会社5社が共同で策定した、クレジット業界におけるグローバルセキュリティ基準「PCIDSS(Payment Card Industry Data Security Standard)」があります。PCIDSSでは、電子商取引を行うWebアプリに対しセキュリティ対策が適切に施されていることが求められ、その代表的な手法はセキュアコーディングとWAFによる保護です。欧米では、この基準を満たしていない電子商取引サイトでは手数料が割高になるなどの具体的なペナルティもあることから、WAFの導入も進んできています。

Webアプリをとりまく脅威

Webアプリは、公開されたあとさまざまな攻撃にさらされます。インターネットの世界では良識ある市民も犯罪者も等しくIP到達可能な距離にあり、こちらが避けたくても否応なく犯罪者によるアクセスはやってきます。犯罪者が狙うものとしては、次のようなものがあります。

踏み台として利用できるリソース

Webアプリが動作するサーバには、ネットワークリソースと計算機能力リソースがあります。これを不正に支配下に置いて任意のプログラムを動作させることで、踏み台となるほかのサーバを探したり、DDoS攻撃に荷担させたりします。また、アクセスしたユーザに対するマルウェア感染を引き起こすサイトや資格情報を盗み取るフィッシングサイトの構築、迷惑メールの送信などに使われます。

個人情報

データベースから盗み出した資格情報やメールアドレスなどは、オンラインバンキングや電子商取引、オンラインゲームのリアルマネートレードなど金銭に絡むものほど高く売れるため、常に犯罪者に狙われています。

著作物などのコンテンツ

音楽や映画などの著作物が不正に公開されていないか、またのぞき見する目的で、自宅用無線カメラの画像が誤って公開されていないか探し回る場合もあれば、検索エンジンの開発者がbot・クローラにWebサイトを自動巡回させて検索エンジンの基礎情報とする場合もあります。

Webアプリに対する攻撃例

具体的にはどのような脅威があるのか、Webアプリに対する代表的な2つの攻撃を紹介したいと思います。

SQLインジェクション

不正に細工された文字列を、文字列を適切にチェックしていないWebアプリに渡すことで、Webアプリ制作者の想定しないSQLリクエストを発生させ、その結果データベース上の情報を抜き出す攻撃手法です。

具体的にどのように攻撃が発生するのか、その詳しいしくみを見ていきましょう(**図2**)。たとえば、あるWebアプリでは、ログインボタンを押すとusernameフィールドに入力された値とパスワードフィールドに入力された値の両方が一致するユーザをデータベースから引いてきて、ログインを許可するとします。このとき、データベースを検索するときにSQL構文として、

```
select * from table where username = ↵
'ユーザ名' and password='パスワード';
```

のような問い合わせを行い、それぞれ「ユーザ名」「パスワード」の部分にユーザが入力した内容が入ります。正常なアクセスでは、たとえばusernameフィールドにtaro、passwordフィールドにpasswordと入力され、SQL構文は、

```
select * from table where username = ↵
'taro' and password='password';
```

となり、結果としてユーザ「taro」の情報をデータベースから引き出してログインできます。一方、悪意のあるアクセスでusernameフィールドに、

```
' OR 1 = 1 #
```

のような入力がされたとします。username =のあとにこの文字列が入るので、次のようなSQL構文になります。

```
select * from table where username = ↵
'' OR 1 = 1 #' and password='password';
```

この細工されたSQL構文では、#よりあとはコメントとして無視され、usernameが空白または1 = 1のユーザが返されます。このとき、ユーザ情報と無関係に1=1は真ですので、SQLでは全ユーザがこの条件に該当することになります。これにより、悪意ある攻撃者は正しいユーザ名、パスワードを知らなくてもログインに成功するだけでなく、実装によってはこのWebアプリを利用するすべてのユーザの情報を取り出せます。

これを防ぐには、外部からの入力情報――ここではusernameやパスワード――に対し書式のチェックを厳密に行い、記号を利用できないようにしておくなどの対策が必要となります。

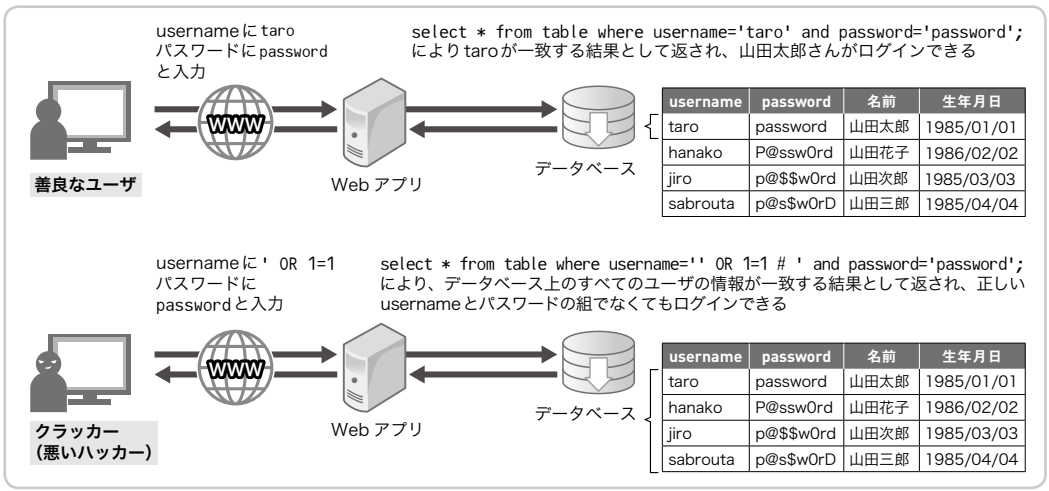
クロスサイトスクリプティング

たとえば掲示板のようなWebアプリで、ユーザが入力したデータを適切に処理せずにHTML上にそのまま出力させると、ユーザが入力したデータに含まれる悪意あるJavaScriptコードなどがそのまま実行されます。これにより、不正なセッションハイジャック、フィッシングなどの二次的な攻撃が成立してしまいます。またPHPコードを埋め込むことにより、外部から任意のプログラムをサーバ側で実行させられるのが最大の問題です。**図3**では、

```
<script>alert("HELLO WORLD");</script>
```

が実行されて単に警告ウィンドウが表示される

▼図2　SQLインジェクションの起きるしくみ



username	password	名前	生年月日
taro	password	山田太郎	1985/01/01
hanako	P@ssw0rd	山田花子	1986/02/02
jiro	p@$$w0rd	山田次郎	1985/03/03
sabrouta	p@s$w0rD	山田三郎	1985/04/04

username	password	名前	生年月日
taro	password	山田太郎	1985/01/01
hanako	P@ssw0rd	山田花子	1986/02/02
jiro	p@$$w0rd	山田次郎	1985/03/03
sabrouta	p@s$w0rD	山田三郎	1985/04/04

▼図3　クロスサイトスクリプティングのしくみ

掲示板の自由記入フォーム(textareaタグなど)にHTMLではタグとして解釈されるような内容、たとえば、JavaScriptやPHPなどを直接記入して投稿すると、そのままそれが実行されてしまう

<script>alert(“HELLO WORLD”);</script>を投稿すると、次にアクセスしたときにそれがJavaScriptとして解釈され実行されてしまう

だけですが、JavaScriptではクライアント側のアクセス履歴を取得したり、別のWebサイトへ強制遷移させたりとさまざまな処理をしかけることができます。またPHPが実行されるとデータベースへのクエリとその結果の表示もできるため、情報漏えいにつながる危険もあります。

これを防ぐには、HTMLタグなどで使用される文字をHTMLへ出力させる前に、

```
< → &lt;  > → &gt;  & → &amp;  " → &quot;
```

のように正しく無害化(サニタイズ)するなどの対策が有効です。

「OWASP Top 10」以外の脅威

昨今のおもな脅威は、Webアプリのセキュリティ向上を目的とするオープンなグローバルコミュニティ「OWASP(Open Web Application Security Project)」にてプロジェクトでまとめられており[注1]、インジェクションやクロスサイトスクリプティングも含まれています。これ以外にも最近の攻撃としては次のようなものがあります。

DoS/DDoS攻撃

大量アクセスを行いWebサイトを意図的にダウンさせる攻撃です。金銭を支払うように脅迫したうえで行われるもの、政治的な目的のもとに行われるものがあります。

Bruteforce攻撃／リスト型攻撃

認証に必要なユーザ名とパスワードの組み合わせを総当たりで試す攻撃です。最近では、ほかのサイトで漏えいした資格情報に基づいて、IPアドレスを頻繁に変えながらアクセスを試すリスト型攻撃もあります。利用者の観点からは資格情報の使い回しを避けるのが望ましい対策となります。

◆ ◆ ◆

このほかの攻撃として「クリックジャッキング」などが大きな脅威として挙げられ、IPAのサイト[注2]で詳しく解説されています。

WAFの持つ機能

WAFはこれまでに述べてきたさまざまな脅威からWebアプリを強力にプロテクトします。ここでは、F5ネットワークス社の「BIG-IP ASM」を具体例として挙げ、WAFの設定の流れを説明したいと思います。

BIG-IPは基礎機能として高度な負荷分散機能を持つフルプロキシ型のアプライアンスです。ASM(Application Security Manager)などのソフトウェアライセンスを追加することでWAF機能も付加できる製品で、次のような機能を持ちます。

・Virtual Server(VS)
BIG-IPが監視するIP・ポート。通常グローバルIPアドレスまたはそこからNATされるIPアドレスを設定できる。ここにASM機能を付加することで、WAFとして動作する

注1) OWASP Webアプリケーションリスクのトップ10(日本語版) URL https://www.owasp.org/images/7/79/OWASP_Top_10_2013_JPN.pdf
注2) URL https://www.ipa.go.jp/about/technicalwatch/20130326.html

・pool
実サーバのIP・ポートを複数個まとめてpoolとして定義できる

vSphere ESXiハイパーバイザなどの仮想環境で動作するBIG-IPのVirtual Edition(VE)は、AskF5[注3]よりアカウント登録のうえ、無料でダウンロードできます。ASMを使用する場合は、製品ページ[注4]から、[Full-Featured Evaluation License]を選択して進めることで評価ライセンスを取得し、ASMを評価できます。

ここでは、「BIG-IP ASM」の機能のうち代表的なものをいくつか紹介します。

Attack Signature

Webアプリに対する攻撃パターンをシグネチャとして持っておくことにより、それにマッチした攻撃をブロックできます。この機能だけでもWebアプリを狙うSQLインジェクション、クロスサイトスクリプティング、強制ブラウズなどさまざまな攻撃の7〜8割は防ぐことができます。自分でカスタムシグネチャを作成することで不要なアクセスを遮断するものや、セキュリティの強化を実施できるものもあります。簡易なブラックリスト型WAFでは、この機能のみを持っているという場合もあります。

Evasion Technique

HTTP通信ではASCII Roman Setのアルファベット、数字、URIで予約されていない記号以外の文字はパーセントエンコードして表現します。

たとえば、UTF-8での「あ」という文字は、「%E3%81%82」のように表現されます。このように文字コードをそのまま数字で表現できるパーセントエンコードでは、これを1回あるいは複数回デコードしたあとに悪意あるコマンドとなるように細工されたものがあります。パーセントエンコード以外にもHTMLエンコードなどさまざまな方式があり、これら指定回数以上エンコードされた怪しいものをブロックできるほか、エンコードされた文字列に対してもデコードしてAttack Signatureなどにマッチングして値を評価する動作をします。

プロトコル違反検出

HTTPプロトコル違反、RFCに準拠しない異常なトラフィックをブロックできます。たとえば、ヘッダのみで値を持たないものや、Content-Lengthが負の値であるもの、HTTP/1.1であるのにHostヘッダがないリクエストなど、明らかにHTTPとして正しくないリクエストを検出します。WebアプリやOSの未知の脆弱性を突く攻撃のほとんどは異常なリクエストであるため、これにより未知の脆弱性による攻撃もある程度防ぐことができます。

パラメータ改ざん検出

あらかじめWebアプリがクライアントに送出したパラメータのハッシュ値を、Cookieに合わせて付与してクライアントに渡しておくことで、クライアント側でパラメータが不正に改ざんされてきた場合に検出できます。

フロー制御

あらかじめ定義したログインページを踏んでからでないとアクセスしてはいけないURL、たとえばユーザのプロフィール表示のページなどを定義しておくことで、ログインせずにアクセスしてきた場合にそれを検出できます。ログアウトページも合わせて登録しておくことで、ログアウト後のアクセスも抑止できます。

Data Guard(クレジットカード情報漏洩対策)

万一SQLインジェクション攻撃などが成功してクライアント側にクレジットカード情報が表示されそうになった場合でも、HTML上に含まれるクレジットカード番号を検出してマスクできます。

注3) URL https://downloads.f5.com
注4) URL https://f5.com/jp/products/trials/product-trials

ホワイトリスト

次の3種類のホワイトリストを定義しておくことができます。

①File Types：Webアプリで使われているオブジェクトの種類

例）php、htm、html、gif、jpg

Webサーバ上に存在するファイルの拡張子のみを登録しておくことで、存在しない拡張子へのアクセスをブロックできます。たとえばApacheでは通常.aspは存在しえないので、.asp拡張子を持つURLへのアクセスはブロックできます。

②URLs：Webアプリに実際に存在するオブジェクト

例）/index.php、/images/logo.gif

ここで登録されているもの以外へのアクセスを許さないようにすることができます。Webアプリが固定されている場合を除き、設定が面倒ですのであまり使われることはありません。

③Parameters：リクエスト・レスポンス内に確認するパラメータの名前、種類

例）username、password、JSESSION

Cookie、HTTP POSTなどのformパラメータ、HTTP GETのパラメータが定義され、全体に対して有効なGlobalパラメータ、特定のURLにひもづくURLパラメータ、そして特定のURL遷移にひもづくフローパラメータを個別に設定することができます。個々のパラメータに対して文字列の長さのチェック、数値の最小・最大値、使用して良い文字、個別のAttack Signatureの有効無効といった設定をすることができます。本来Webアプリが行うべき入力内容のチェックを代行する機能と言えます。

DoS/DDoS対策

同一のIPアドレス、あるいは同一URL宛ての通信が突然増大したり、あらかじめ指定した閾値（しきいち）を超えた場合にDoS/DDoS対策とみなしアクセスを制限する機能です。サーバ側のレスポンスが悪くなった場合に攻撃されているとみなしてトラフィック流量を下げる方式や、クライアントにJavaScriptを送ってアクセスを抑止するなどの能動的な対策もできます。

BruteForce対策

あらかじめ設定してあるログインフローに対して同一IPから大量のログイン試行が行われた場合、あらかじめ指定した閾値でアクセスを抑止する機能です。

Geolocation

アクセス元のIPアドレスに基づく国、地域によりアクセスの許可・拒否を決めることができる機能です。完全に日本国内の日本人向けの日本語のみのWebサイトであれば、日本以外からのアクセスを遮断するという極端な設定もありでしょう。不要であれば中国、ロシア、ウクライナなどからのアクセスを遮断することで攻撃リクエストをかなり減らすことができます。また、WAFに到達するまえにNATされているといった場合は、「X-Forwarded-For:」や「X-Real-IP:」ヘッダなど元のIPアドレスの書かれているヘッダをソースIPとみなす設定もできます。

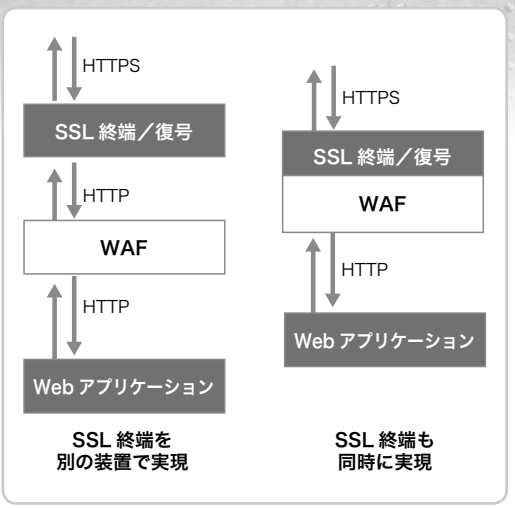
WAFを導入する

配置計画

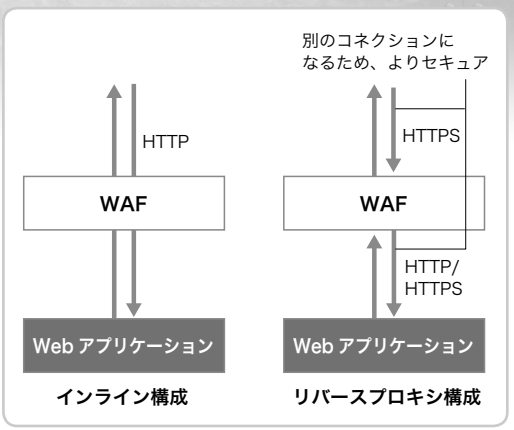
WAFは、SSLで暗号化された通信の中身を直接見ることはできませんので、SSLを終端して暗号化を解いてから内容を見られる場所に配置する必要があります。WAFの中にはBIG-IP ASMのようにSSLアクセラレータ、負荷分散機能、ネットワークファイアウォール機能を併せ持つものもあります（**図4**）。

また、インラインで配置する方式と、WAFでいったんコネクションを終端するリバースプロキシとして配置する方式があります。インライン構成の場合、既存の機器のIPアドレスや

▼図4　WAFの配置計画（SSL終端をどこで行うか）



▼図5　WAFの配置計画（インライン・リバースプロキシ）



構成の変更が最小限で済むメリットがあります。アプリケーションプロキシの場合は、クライアントからの通信がいったんWAFで終端され、WAFから見てクライアント側とサーバ側のコネクションが分離できるので、よりセキュリティ的に安心できる構成となります(**図5**)。

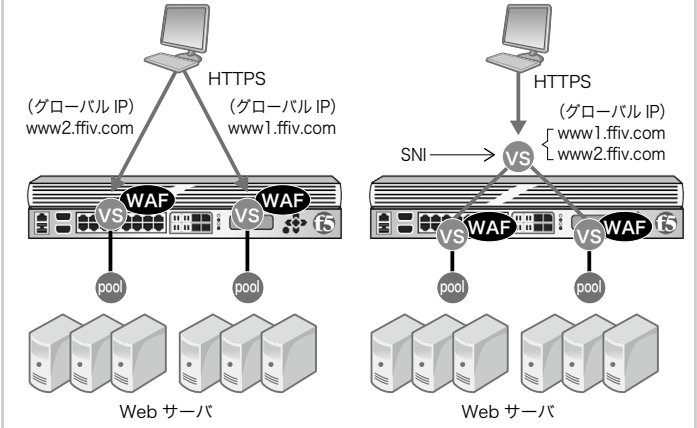
BIG-IP ASMでは、あらかじめpoolとしてWebサーバ群の実IPアドレス、Virtual Server(VS)としてグローバルIPアドレスを設定し、そこにWAFのセキュリティポリシーを割り当てる形で設定を行います(**図6**)。WAFの設定は、Webアプリ単位で行われます。同一のIPアドレスで複数のFQDNの名前解決が行われるケースでは、SNI(Server Name Indication)機能も併用してSSL終端をさせると良いでしょう。

セキュリティポリシー

WAFはさまざまな機能を組み合わせてWebアプリに合うセキュリティポリシーを設計、策定し、実トラフィックを通して検出・誤検出・非検出を確認して設定を最適化していく、いわゆる「PDCAサイクル」を回しながらポリシーチューニングを行っていくことでセキュリティを高めていきます。

WAFの設定を始める前に、まずWebアプリ自体の調査を行います。使用されている言語、OS、Webサーバ、Webアプリ技術、変数名とその内容、使われるHTTP Methodなどさまざまな情報を収集します。そしてWebアプリがどんな国、人、デバイスを対象にしたものなのかをあらかじめ定義しておくことで、WAFのポリシーチューニングがしやすくなります。これにより、初期設定の段階である程度誤検出が起き

▼図6　WAFの配置計画（BIG-IP ASMの場合）



1台のWAFでリソースの許す限りWebアプリごとに異なるセキュリティポリシーを設定できる。複数のHTTPSサイトが同一IPアドレスでもSNIやHost:ヘッダによる分離で、Webアプリごとに異なるポリシー設定ができる

にくい設定を作ることができます。

たとえばセキュリティ関連のブログや掲示板では、SQLインジェクションの例を投稿し「' OR 1=1 #」のような内容をHTTP POSTするような場面もありますが、WAFがそれをブロックしてしまいます。そのため、該当するTEXTAREA ParameterではAttack Signatureの設定をゆるめ、同時にWebアプリ側で無害化されていることを確認する必要も出てきます。同様に機種依存文字も、たとえば「①」「②」など厳密にはShift JISコードから外れるキャラクタもFailed to convert character、文字コードの違反としてブロックされてしまうので、Shift JISが使われているWebアプリではこれはブロックせずに許可する調整が必要な場合もあります。

Webアプリで使用される変数名に日本語が使用されている場合はパーセントエンコードされるため、「%」記号も許可する必要があったり、またJBoss環境では変数名に「:(コロン)」が一般的に使用されるので許可する調整も必要になります。

初期設定の例

BIG-IP ASMでは実用的なWAFのポリシー設定を簡単に作成できます。BIG-IPの管理画面から[Security]→[Application Security]→[Security Policies]→[Active Policies]と移動し、[Create]から新規作成を行うことで、ウィザード形式でポリシーの初期設定ができます。

まず適用先VSの設定ですが、あらかじめVSとpoolが設定されている場合は、[Existing Virtual Server](既存のVS)を選択し、次にHTTPかHTTPSかのプロトコルを選択のうえ、ASMを適用するVSを選択します。

このあとに出てくる[Deployment Scenario]では、[Create a security policy automatically (recommended)]が推奨となっていますが、[Create a security policy manually or use templates(advanced)]を選択し、セキュリティポリシーの編集に移ります。ここでは、Webアプリの扱う文字コードを選択し(今回はUTF-8)、あらかじめWAFの保護対象のWebアプリが、OWA(Outlook Web Access)やSharePointなどと決まっている場合はそれを選択、そうでない場合は[Rapid Deployment security policy]を選択し、[Enforcement Readiness Period](機能を有効にするまでの猶予期間)は「0days」とします。

[Attack Signature]は、対象システムに合わせて必要な項目を選択します。不要な検出、誤検出が頻発しますが、どのような攻撃があるのか可視化することが目的であれば、すべてのAttack Signatureを割り当てます。

Staging機能はAttack Signatureで検出してもブロックしないように個別設定できる機能ですが、同様の設定はTransparentモード(透過モード)でも可能ですので、複雑にしたくない場合は無効にしておきます。

図7が以上の設定の一覧になります。初期ポリシーができたところでは、Transparentモードになっています。これをBlockingモードにすることで検出したときにWAFが実際にアクセスを遮断します。WAFのメイン設定画面では、モードの選択のほか、クレジットカード番号のマスク、許可するHTTPレスポンスコード、X-Forwarded-For:などのソースIPが書かれているヘッダを見る場合の設定などを設定できます(**図8**)。

これら設定をしたうえで、試験的にトラフィックを流してみてブロックされないかを確認します。

▼図7 設定の一覧

Security » Application Security : Security Policies » Deployment Wizard : Security Policy Configuration Summary

Security Policy Configuration Summary

Local Traffic Settings

HTTPS Virtual Server	DummyVS

Security Policy Properties Configuration

Security Policy Name	DummyVS
Application-Ready Security Policy	Rapid Deployment security policy
Application Language	Unicode (utf-8)
Enforcement Readiness Period	0 days

Attack Signatures Configuration

Systems	General Database, Various systems, System Independent
Signature Sets	**Automatically assigned set(s):** Generic Detection Signatures
Signature Staging	Disabled
Apply Signatures to Responses	No

```
https://<VSのIPアドレス>/%00
https://<VSのIPアドレス>/%5c
```

のようなリクエストを投げると、HTTP Parser Attack、Injection Attemptとして検出できました。

すべてのAttack Signatureを設定すると、

```
https://<VSのIPアドレス>/phpmyadmin/
https://<VSのIPアドレス>/etc/passwd
```

といったリクエストも検出されます。WAFにより検出される項目は管理画面でそのリクエスト詳細を確認できます(**図9**)。

また、すべての検出リクエストはWAFの管理画面で確認できます。設定によってはアクセス違反にならなかったものも含めてすべて記録することもできます。リクエスト詳細確認画面では検出理由のほか、どのような文字列が問題になったのかを確認でき、HTTPヘッダの内容も確認できるため、正常なアクセスか攻撃と見受けられるアクセスかをある程度判断できます(**図10**)。ここで[Learn]ボタンを押すと、この検出が誤検出だった場合は次回以降は検出しないように設定を変えていくことができます。

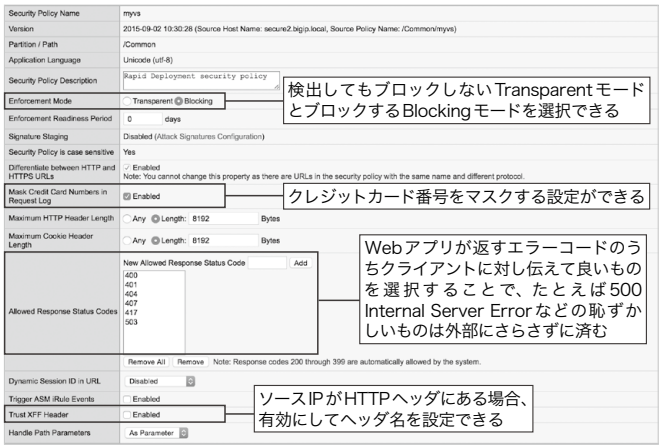
WAFを運用する

誤検出の問題と試行運用期間

WAFを運用するうえで、あらかじめ想定した正常系のトラフィックに対してポリシー設定を行ったとしても、実環境でのトラフィックでは予期しない誤検出が発生します。そのため、試行運用期間を設けて運用開始直後しばらくはTransparentモードで様子見をして、誤検出をある程度つぶしてからBlockingモードに変更して運用開始する、というのが望ましい運用方法です。

また、WAFによる検出があったときにアクセス者のWebブラウザに対して任意のHTMLを表示できます。このときに攻撃をブロックしたことを知らせる程度ならまだしも、過度に悪者扱いして攻撃者を挑発するメッセージを含め

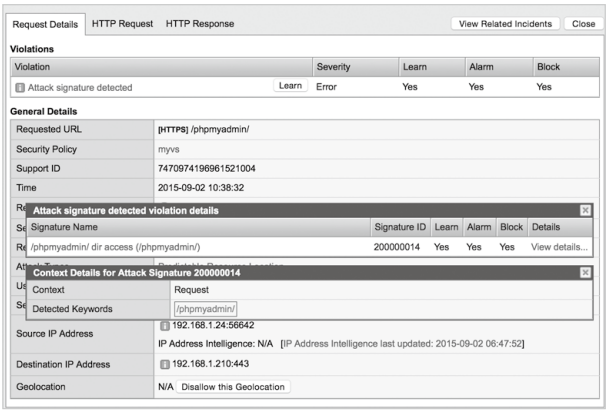
▼図8　WAFのメイン設定



▼図9　WAFの検出例

Status	Time	Severity	Source IP	Response Code	Requested URL
	10:38:38	Error	192.168.1.24	N/A	[HTTPS] /example/
	10:38:32	Error	192.168.1.24	N/A	[HTTPS] /phpmyadmin/
	10:38:24	Error	192.168.1.24	N/A	[HTTPS] /etc/passwd
	10:38:16	Critical	192.168.1.24	N/A	[HTTPS] /
	10:38:11	Error	192.168.1.24	N/A	[HTTPS] /0x0

▼図10　リクエスト詳細確認画面



ると、さらなる攻撃の元になりかねません。また、電子商取引のサイトで誤検出になった場合、利用者を不快にさせないようなメッセージにする工夫が必要になります。

正常アクセスと攻撃アクセスの見分け方

WAFは機械的・設定どおりにセキュリティポリシーに従わないものを検出する働きをすることで、Webアプリに対する攻撃を防ぎますが、これが正常な検出だったのか誤検出だったのかを最終的に判断するのは人です。

たとえば、WAFによるブロックが発生する前に、同一のIPアドレスを持つHTTP USER AGENTからリクエストがあり、Refererヘッダも正しく追随していて、CookieやHTTP POSTパラメータが明らかに正常な場合は、正常なリクエストを誤検出している可能性が高いと判断できます。この場合、ポリシーの調整が必要になります。

一方、不正なReferer:ヘッダ、たとえばトップページからリンクしていないにもかかわらずトップページのURLがReferer:ヘッダにあり、とくにその直前にトップページへアクセスしていないうえ、GETやPOSTリクエスト内に異常な文字列や見知らぬユーザ名・パスワードがあるような場合はブロックしたままでかまわないリクエストだと判断します。この場合は、能動的に放置しておけます。

また、検索エンジンのbot・クローラには、大手検索エンジン以外にもさまざまなものがあり、各々のHTTP USER AGENTを持ちます。通常、検索エンジンに対してはアクセスの可否の意思表示としてrobots.txtファイルを置いて設定を行うのが暗黙の了解となっていますが、これに従わない検索エンジンはマナー違反の悪質なものだと判断できます。筆者の経験上、とくにahrefsbot、mj12bot、mail.ru_botといったものはrobots.txtを無視して走査するため、たとえばmj12botに対しては次のようなカスタムAttack Signatureを追加設定してブロックすることも有効な対策です。

```
headercontent:"mj12bot"; nocase;
pcre:"/^User-Agent:.*?mj12bot.*?/Hmi";
```

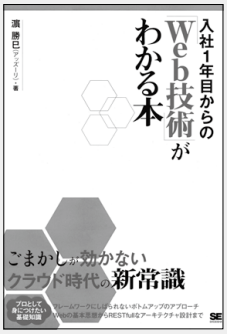
WAFは、ネットワークファイアウォールやUTMなどと同じ要領で導入し、設置したらそれで終わりという製品ではありません。Webアプリに合わせたチューニングを行いセキュリティポリシーのPDCAサイクルを回しながら運用することで、Webアプリそのものと利用者を最新の犯罪からでも守ることができます。

おわりに

このようにWAFの設定は、IP・ポートといったネットワークに関する理解だけではなく、HTMLを始めとするWebアプリやHTTP通信のしくみに対する理解に加え、それに対する攻撃と防御法に関する知識が不可欠で、適切な運用と設定は簡単ではありません。WAFの初期設定から運用も含めてすべてプロのコンサルタントに任せるのも1つの選択です。また、自社でWAFの運用ができる体制を構築するためのトレーニングを受けるのも1つの手です。

筆者が属するF5ネットワークス社では、BIG-IP ASMのトレーニングのほか、ユーザのWebアプリケーションの調査からASMの導入、運用までを包括的にサポートする体制も用意してあり、電子商取引サイトや金融機関、政府機関での豊富な実績を持っています。

またクラウド型のWAFサービスとして、DNS書き換え、またはルーティング経路変更によってトラフィックを迂回させ、WAFでチェックすることでユーザのWebアプリを守る「SilverLine」というサービスを提供しています。BIG-IP ASMはアプライアンスだけではなくクラウド上で動作させることもできるため、オンプレミス環境、クラウド環境、両方を利用したハイブリッド環境でも対応できます。SD



入社１年目からの「Web技術」がわかる本

濱 勝巳 著
A5判／296ページ
2,480円＋税
翔泳社
ISBN＝978-4-7981-4244-9

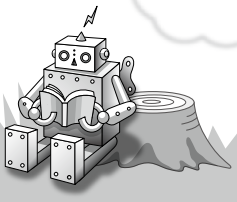
新米技術者の学習の入り口となるように書かれた本である。現時点で標準化されているWeb技術について、特定の製品・企業などの説明をできるだけ省いて解説している。Webに限らないITの専門用語についても、ページの欄外で都度説明されており親切に感じる。本の構成としては、1章でWebを支える技術全体を俯瞰したあと、各技術の詳細を章ごとに解説していくというもの。章立ては、HTML5、CSS、ECMAScript、REST/HTTP、JSON、Webのセキュリティ（暗号化、認証など）、インターネットプロトコルスイートと、基礎的なものから最新のものまで幅広い。Webの“フロントエンド”についての本なので、“バックエンド”（サーバサイド）については、ほかの書籍をあたると良いだろう。

初めてのSpark

Holden Karau、Andy Konwinski、Patrick Wendell、Matei Zaharia 著、Sky株式会社 玉川 竜司 訳
B5変形判／312ページ
3,200円＋税
オライリー・ジャパン
ISBN＝978-4-87311-734-8

Sparkは最近人気の分散処理プラットフォーム。並列実行に特化したキャッシュ機構「RDD」を持ち、簡単に分散処理を実装できる。本書はSparkについて、基本操作から高度なプログラミングまで網羅的に解説している。著者はSparkのユーザとして２種類を想定している。データを変換・分析して知見を得る「データサイエンティスト」と実稼働させるデータ処理アプリを開発する「データエンジニア」だ。前者は特定目的のプログラムをその都度組むことが多いため、Sparkの開発スピードの速さがメリットになる。後者に対しては、分散システムの複雑なプログラミングが隠蔽されるので、開発の手間を減らせる。自分がどちらの用途でSparkを使うのかを意識しながら読むことで、よりためになる１冊となるだろう。

SD BOOK REVIEW



たのしいインフラの歩き方

齊藤 雄介（外道父）著
A5判／672ページ
3,200円＋税
技術評論社
ISBN＝978-4-7741-7603-1

本書は、まったく知識のない新人を一人前のインフラエンジニアに育てるために書かれた「愛情の込められた解説書」である。なんと600を軽く越えるページ数なのだ！ところがその内容は、説明口調で淡々と技術解説するありがちなものではなく、あたかも上司が部下を現場で教えるかのよう。かといって冗長過ぎず、実践的にITインフラの構築技術を説明している。まず興味のあるところから読み進めれば、このボリュームも苦痛ではないだろう。ハードウェアからレイヤ7のアプリまで、さらにはオンプレミスだけでなくクラウド環境まで、カバーする領域は広い。多くの技術書で書かれている内容を補完している、といっても過言ではないので、復習としても有用だ。やさしく書かれているので営業やマーケティングの方にもお勧め。

データサイエンティスト養成読本 機械学習入門編

比戸 将平、馬場 雪乃、里 洋平、戸嶋 龍哉、得居 誠也、福島 真太朗、加藤 公一、関 喜史、阿部 厳、熊崎 宏樹 著
B5判／192ページ
2,280円＋税／技術評論社
ISBN＝978-4-7741-7631-4

機械学習の入門書は、アルゴリズムを解説する学術書と人工知能など機械学習の応用について語る読み物の２つに大別される。それぞれは機械学習の理論とトレンドを把握するうえでは役立つが、機械学習を応用したサービスを検討しているエンジニアにとって有用な書籍ではないかもしれない。その点本書は、これまでの書籍とは異なり、機械学習に取り組むときの入門書に相応しいと言って良いだろう。前半では機械学習の全体像や手法を解説し、話題の深層学習を取り上げる。後半ではPythonによる機械学習、推薦エンジン、画像認識、異常検知などのトピックをサンプルコードを見ながら解説する。「データサイエンティスト」と冠しているが、職種にかかわらず真剣に機械学習を使いたいと考えるエンジニアにお勧めしたい。

手がかりを探せ！

SMB実装をめぐる冒険

特別企画 短期集中連載

File System for Windowsの作り方（前編）

探す、調べる、ソフトを作る喜び

こんにちは。よういちろうです。つい最近、ChromeOS向けに「Windows共有フォルダをChromeOSのファイルアプリにマウントする」ことができるChromeアプリを開発してリリースしました。これを開発するためには、SMB（Server Message Block）と呼ばれるプロトコルを理解し、SMBプロトコルを話すクライアントコードをJavaScriptで書くことが必要でした。これは「File System for Windows」という名前でChromeウェブストアにて無料で公開していますので、Chromebookを持っている方はぜひ使ってみてください。今回のこの記事の内容は、そんなSMBプロトコルの実装に挑戦した話を、探偵が真相に迫っていくような感じで紹介します。

Author 田中 洋一郎（たなか よういちろう） Blog https://www.eisbahn.jp/yoichiro/ Twitter @yoichiro

第1部 SMBプロトコルの研究

手がかりを入手しろ

僕は2014年に「Chrome MySQL Admin」というChromeアプリの開発をしていました。その中で、MySQLサーバに接続してSQL文の実行依頼と処理結果の取得を行うドライバコードをJavaScriptだけで書き上げ、さらにSSH2 Port Forwardingでの接続を実現するために、libssh2というライブラリを利用したNaClモジュールをC++で作成しました。Chrome MySQL Adminの開発が一段落しようとしていたある日、知人から次の一言を言われました。

「Port Forwarding書けたなら、sshfsも簡単なんじゃない？」

これと同時に、僕はChromeOSにFile System Provider API（FSP API）がBeta公開されていることを知りました。助言どおり、sshfsのようなlibssh2を使ったSFTPの処理をNaClモジュールとして実装し、それをFSP APIで包んで“SFTP File System”というアプリ名でChromeウェブストアにリリースしました。その後、矢継ぎ早にDropbox版、OneDrive版、WebDAV版と開発を進め、気がつくと4種類のアプリを書き上げていました。

しかし、これらのリリースをするたびに、1つの要望ばかりがコメント欄に投稿されてきました。それは、

「SMB実装はまだですか？」

でした。本当にこのコメントが多かったのです。つまり、Chromebookを購入したほぼ全員が、WindowsやSambaのファイルサーバにアクセスしたがっていたのです。

欲しい気持ちはわかります。僕だって欲しい。この時点で、SMBプロトコルという存在自体はもちろん知っていました。そして、SMBプロトコルの中身はまったく知らなくても、「とにかく複雑なプロトコルのようだ」ということも漠然と知っていました。一筋縄では行かないであろうSMB実装の開発に着手すべきかどうか、数日間悩みました。考え抜いた末の結論、それは「世界中の人が僕に期待しているんだ、チャレンジしてみよう」でした。もし当時の自分に助言できるとしたら……「やめとけ！」と言うかもしれません。

とにかくググってみた

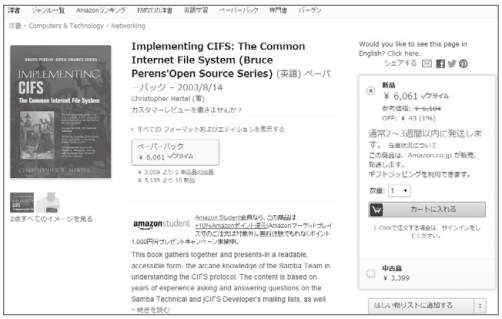
SMBとはどんなプロトコルなのか、WindowsやSambaのファイルサーバに正しい手順で潜入していくにはどうしたらいいのか、まずはGoogleで検索することから始めました。キーワードとしては、「SMB」と「CIFS」です。CIFSは

「Common Internet File System」の略であり、簡単に言ってしまうと、SMB1（SMB2が登場する前に使われていた仕様）とほとんど同じ仕様を指す名称です。「きっと過去にSMBやCIFSを調べた人が書いた記事とかヒットするに違いない」と期待していましたが、残念ながらその期待は外れました。少なくとも日本語の記事は「皆無」でしたし、英語で書かれた記事すらも、ほとんどヒットしません。その中でも「そのときに読んで理解できた記事」は、たった1つだけでした（**図1**）。

図1は、2004年の記事です。実に11年も前に書かれた記事になります。そして、書籍の存在もわかりました。洋書で1冊だけありました（**図2**）。

▼図1　［第20回　ファイル共有プロトコルSMB／CIFS（その1～3）- @IT］（http://www.atmarkit.co.jp/ait/articles/0410/29/news103.html）

▼図2　Implementing CIFS（http://www.amazon.co.jp/dp/013047116X）



@ITの記事から得られたこと

上記の@ITの記事は、SMBプロトコルの基本的なことをていねいに説明した、僕にとっては救世主的な内容でした。僕はその記事から次のことを学びました。

- SMBプロトコルにはダイアレクト（方言）があること
- 大昔はNetBIOSというインターフェースで会話する必要があったが、今では445ポートを使って直接SMBプロトコルでサーバと話せること
- ファイルアクセスまでには、Negotiation→Session Setup → Tree Connect → File Accessという手順を踏む必要があること
- MySQLプロトコルは［送信バイト数］［送信内容］というパケット構造だったが、SMBプロトコルも同じようなパケット構造であること
- 送受信されるパケットは、「ヘッダ」「パラメータ」「データ」の3つから構成されていること。ヘッダは32バイト固定、ワードデータとバイトデータは可変長であること

これらだけでも、SMBプロトコルが「顔も名前もまったくわからない誰か」から「名前と職業程度はわかった」くらいの進歩があります。NetBIOSを把握する必要はなくなりましたし、きっとものすごい難解だけど、ファイルアクセスまでに必要な手順もわかりました。そして何よりも、送受信されるバイナリデータの形式をこの時点で知ることができたのは、非常に大きなことでした。基本的な形式さえわかってしまえば、「理解できそうだ」という気持ちが芽生えてくるからです。

今振り返ると、この@ITの記事に掲載されていたダイアレクト一覧では、“NT LM 0.12”が最新のダイアレクトでした。つまり、まだSMBプロトコルのバージョン2が登場する前の記事ということになります。

秘伝の書『Implementing CIFS』の購入

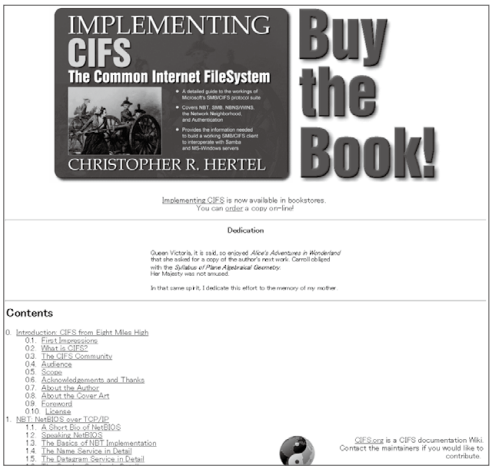
Google検索をしていたときに使っていたキー

ワードは「CIFS」でしたが、それについて書かれた書籍があれば、これ以上ない情報源になります。洋書ですが、1冊だけCIFSについて詳しく書かれた本を見つけることができました。それは『Implementing CIFS』という名前の本です(**図2**)。

本のタイトルからして、まさに僕がこれからやろうとしていることが説明されているのではないか、と期待は膨らみます。すぐにAmazonにて購入しました。たしか追加料金を払って、お急ぎ便で注文したと記憶しています。本が手元に来たときに「これで勝てる!」と喜んだのですが、この本の内容がすでにインターネット上で公開されていることを知ったのは、それからたった数日後のことでした(**図3**)。

この本は、Sambaプロジェクトに所属しているエンジニアが書いた本になります。昔からCIFSのJava実装「jCIFS」があることは耳にしていたのですが、この本の著者はjCIFSのメンバーでもありました。ざっとこの本を読んでいくと、随所にSMBプロトコルを調査していたときの苦労話が出てきます。「結局この値の意味は最後までわからなかった」という記載も多く、Sambaの開発がいかに困難を極めていたかが、涙が出るほど伝わってきました。

▼図3 Implementing CIFS——ubiqx.org (http://ubiqx.org/cifs/)



ターゲットはSMB1/CIFS

調査結果として、@ITの記事、そしてImplementing CIFS本、この2つの情報源を手にできました。これらを武器にして、実装を進めていくことにしました。どちらの情報も、SMB1/CIFSの範囲でしたので、実装対象もその範囲になりました。SMB1やCIFSのダイアレクトである"NT LM 0.12"はWindows 95ですでに実装されていたものですので、「20年も前に実用化されていたプロトコル」を実装するということになったわけです。

とはいえ、今も現役で使われているプロトコルですし、NASによってはSMB2以降は未サポートなものもいまだ多そうなので、十分価値がある作業だと信じることにしました。さらに、いざとなればjCIFSのコードを読めばそこに正解がある、という点でも、調査段階でかなり安心したことを覚えています。

相手にすべき敵は決まりました。早く通信してみたい、そんな気持ちがどんどん大きくなります。

ちなみに、仕様書であるMS-SMB[注1]やMS-CIFS[注2]も見つけていましたが、まったく読む気になれず、しばらく無視することにしました。

第2部 3つの武器をそろえろ!

必要そうな文献を得たところで、敵とのコンタクトを取ってみたい気持ちが高まって来ていましたが、少し我慢して事前準備をもう少し行いました。やり過ぎは良くないですが、基本的に用意周到であることは良いことです。ここで、過去にChrome MySQL Adminを作った際に得た経験を活かせるのではないか、と考えました。

それは、「盗聴」です。

注1) https://msdn.microsoft.com/en-us/library/cc246231.aspx
注2) https://msdn.microsoft.com/en-us/library/ee442092.aspx

パケットキャプチャ

何らかのプロトコルを実装する際に、果たしてどんな内容が送受信されているのか、とても気になります。HTTPのような文字ベースのプロトコルであれば簡単なのですが、SMB1/CIFSでやりとりされる情報は、バイナリです。サーバから受け取った結果が文字列であれば、その内容を標準出力に出せばすぐに確認ができますが、バイナリとなるとそうもいきません。もちろん、実装作業が進むにつれて16進コードを読めるようになってくるのですが、限界はあります。

そこで役に立つ道具として、パケットキャプチャツールがあります。これは、ネットワークインターフェース（Linuxであればeth0など）を通るパケットの内容を出力してくれるツールです。自分で作ったプログラムが送った内容や、サーバが返してきた内容を「ある程度わかりやすいように」出力してくれます。僕が使ったことがあるツールは次の2つです。

・tcpdumpコマンド
・Wiresharkアプリケーション

この2つのツールは、バイナリプロトコルの実装をするうえで強い武器になります。実際の開発では、サーバから何度もエラーレスポンスを受け取りながら、クライアントが送る情報を細かく変えていきます。その際に、サーバからの結果を受け取るためのコードを書かなくても、パケットキャプチャツールによって処理が成功したかどうかを知ることができます。

また、実際に通信ができている状況の送受信内容を「盗聴」することも重要です。つまり、送受信されている内容を覗き見ることで、「正解がわかる」わけです。SMBについては、残念ながら事前にすべてを文献から勉強することは非現実的です。「動いているコードが仕様だ」という言葉がありますが、「送受信できている内容が仕様だ」という気概で最初から取り組んだほうが気が楽だ、とChrome MySQL Adminを作っていてMySQL Serverとの通信内容を盗聴しているときに悟りました。

どこでも使えるtcpdumpコマンド

パケットキャプチャと聞いてまず思い浮かぶツールは、tcpdumpでしょう。UNIX系のOSであれば、まずインストールされているコマンドだと思います。rootユーザになる必要がありますが、最も手軽に送受信されている内容を覗き見できます。たとえば、開発にMacBook Airを使っていて、接続相手にLinuxで動作しているSambaサーバを選んだ場合は、どちらでもtcpdumpを利用できるでしょう。

tcpdumpをオプション指定なしで起動すると、パケットのヘッダだけが表示されます。しかし、いくつかのオプションを指定して起動することで、tcpdumpは「人間に見やすいように」そして「送受信されている内容をある程度理解して」パケットの内容を表示してくれるようになります。僕が気に入っている指定は次のものです。

```
tcpdump -vv -X -s 0 port 445
```

“-vv”は、送受信される内容について、自動的にプロトコルを判断して、そのプロトコルに特化した情報表示を行うためのオプションです。“v”を2つ重ねることで、tcpdumpはSMBプロトコルを理解してくれるようになります。次の“-X”は、パケットの内容を16進数とASCIIで表示するためのオプションです。“-s”に0を指定することで、送受信される情報すべてが表示対象になります。最後に、パケットキャプチャ対象のポート番号を“port”オプションで指定します。**図4**は、これらのオプションを指定してtcpdumpを起動した際のSMBプロトコルのパケットキャプチャ内容の例です。

tcpdumpがSMBプロトコルをちゃんと理解しているのがなんとなくわかると思います。たとえば、Error classおよびError codeは、サーバが

▼図4　tcpdumpの実行例

```
yoichiro — yoichiro@cynos: ~ — -zsh — 80×36
        0x0040:  c754                                           .T
06:18:35.327808 IP (tos 0x0, ttl 64, id 5252, offset 0, flags [DF], proto TCP (6
), length 125)
    192.168.0.11.52555 > freenas.eisbahn.jp.microsoft-ds: Flags [P.], cksum 0xdc
2e (correct), seq 1:74, ack 1, win 4117, options [nop,nop,TS val 437731148 ecr 4
043425620], length 73
SMB PACKET: SMBnegprot (REQUEST)
SMB Command   =  0x72
Error class   =  0x0
Error code    =  0 (0x0)
Flags1        =  0x8
Flags2        =  0x1
Tree ID       =  65535 (0xffff)
Proc ID       =  1 (0x1)
UID           =  65535 (0xffff)
MID           =  0 (0x0)
Word Count    =  0 (0x0)
smb_bcc=34
Dialect=NT LM 0.12
Dialect=SMB 2.002
Dialect=SMB 2.???


        0x0000:  50e5 493e 41df 98e0 d99b 84ff 0800 4500  P.I>A.........E.
        0x0010:  007d 1484 4000 4006 a3f1 c0a8 000b c0a8  .}..@.@.........
        0x0020:  00aa cd4b 01bd a9c2 4658 d65G 7ae9 8018  ...K....FX.Vz...
        0x0030:  1015 dc2e 0000 0101 080a 1a17 3f4c f101  ............?L..
        0x0040:  c754 0000 0045 ff53 4d42 7200 0000 0008  .T...E.SMBr.....
        0x0050:  01c8 0000 0000 0000 0000 0000 0000 ffff  ................
        0x0060:  0100 ffff 0000 002Z 0002 4e54 204c 4d20  .......".NT.LM.
        0x0070:  302e 3132 0002 534d 4220 322e 3030 3200  0.12..SMB.2.002.
        0x0080:  0253 4d42 2032 2e3f 3f3f 00              .SMB.2.???.
06:18:35.328746 IP (tos 0x0, ttl 64, id 12281, offset 0, flags [DF], proto TCP (
6), length 52)
    freenas.eisbahn.jp.microsoft-ds > 192.168.0.11.52555: Flags [.], cksum 0xb1d
2 (correct), seq 1, ack 74, win 8316, options [nop,nop,TS val 4043425622 ecr 437
```

送ってきたエラーコードです。この場合は、値が0なので、サーバは正常に処理を行うことができたことを示しています。このように、自分のプログラムが送った内容を相手のサーバが正しく解釈できたかどうか、パケットキャプチャを行うことで簡単に把握することが可能になります。

しかし、tcpdumpによるSMBプロトコルの理解度では、僕がSMB実装を行ううえで足りませんでした。開発を進めていく中で、SMBプロトコルはそれだけでなく、その中に別のプロトコル(サブプロトコル)が入ってくることがあることがわかりました。しかし、残念ながらtcpdumpコマンドはそこまでは理解してくれず、サブプロトコルの部分は単なる16進数表示になってしまいました。tcpdumpだけでは、十分に敵の姿を見ることができなかったのです。

最強のツール Wiresharkアプリ

サブプロトコルも含めて解析可能なパケットキャプチャツールを探していたのですが、そこで出会ったツールがWireshark[注3]です。これは、tcpdumpと違い、GUIアプリケーションです。そのため、送受信されている内容を視覚的に見やすい形で表示してくれます。

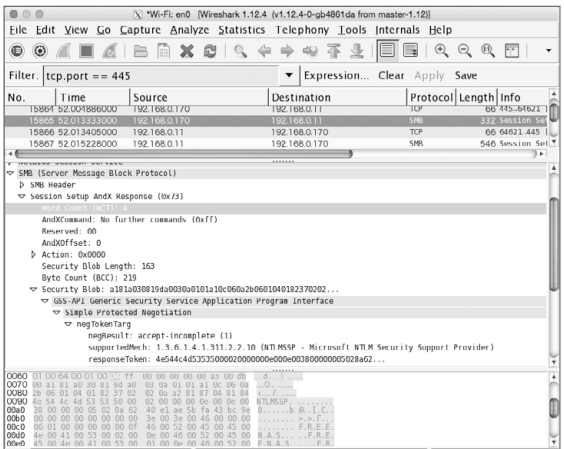
Wiresharkの最も大きな利点は、800以上のプロトコルに対応しているということです。SMBプロトコルも例外ではなく、SMBプロトコル内で使われているサブプロトコルについても、ちゃんと認識してくれます。たとえばサーバから共有リソースの一覧を取得したい際には、SMBプロトコル内でDCE/RPCという別のプロトコルを利用することが必要なのですが、Wiresharkであれば**図5**のようにちゃんとDCE/RPCプロトコルの内容まで表示されます。

WiresharkはWindows、Mac OS X、そしてUNIX系OSで利用できます。普段は手軽なtcpdumpを使い、深い解析が必要な場合にWiresharkを使うと良いでしょう。

Sambaサーバのプロトコルバージョンの限定

最後にそろえた道具、それは「自由が利く通信相手」です。たとえば、スポーツでも「本当の相手」と普段戦うことはできないので、その相手と似ている特徴がある人や、その相手のプレースタイルを真似た「仮想の相手」を準備して練習に取り組むことがあると思います。その仮想の相手に対していろいろと試すことができるし、どんどん失敗することもできます。さらに、そ

▼図5　Wiresharkの実行例



注3）https://www.wireshark.org/

の仮想の相手から「今のプレーはどうだった？」と聞くことさえできるわけです。そうやって入念な準備をすることで、実際の相手の攻略をしていきます。

SMB実装の開発は、通信相手のサーバをいかに攻略するか、これにかかっています。つまり、相手をよく知ること、これが本当に重要です。そのために、自宅にあるサーバマシンに仮想マシンを1つ追加し、Sambaをインストールしました。今回のSMBプロトコルの実装対象は"NT LM 0.12"ダイアレクト、つまりSMB1/CIFSレベルのものですので、インストールしたSambaに対して、SMB2以上のバーションを話すことができないように制限をかけました。具体的には、smb.confファイルに次の行を追加します。

```
server max protocol = NT1
```

Mac OS XやWindowsから先ほどインストールしたSambaサーバに接続する際に、Sambaサーバに「"NT LM 0.12"以上話すな」と言っておけば、Mac OS XやWindowsは会話レベルをサーバに合わせます。つまり、tcpdumpやWiresharkを使ってパケットキャプチャした際に、いつでもSMB1/CIFSレベルの「正しい」通信内容を入手することが可能になります。言わば「おとり捜査」ができるわけです。

今回は「送受信できている内容が仕様だ」という取り組み方をしたので、実際の通信内容からいつでも正解を得られる環境を構築しました。盗聴のためのツールと、おとりとなるサーバ、これで武器はそろいました。

敵を知る

知識と武器がそろったところで、いよいよ敵とコンタクトを取りましょう。仮想の相手となるSambaサーバと、SMBプロトコルを使って通信をしていきます。まずやることは、通信路を作ることです。つまり、ソケット通信における「socket()→connect()」をします。

プロトコルによっては、接続した時点でサーバから何か送られてくることがあり、接続するだけで「おぉぉ！きたぁ！」と喜ぶことができます。たとえば、MySQL Serverとの通信の場合は、クライアントが接続した時点でまずサーバからパケットが送られてきますので、クライアントプログラムはその内容を解釈することが第一歩になります。しかし、SMBプロトコルの場合は、接続後にまず話しかけるのはクライアントからになります。何も話しかけなければ、サーバはそっと接続を切ってしまうことでしょう。

ファイルアクセスまでの流れ

クライアント／サーバ型の通信では、ほとんどの場合「接続、認証、やりたいことの要求」という手順を踏むことになります。SMBプロトコルの場合も同様です。ただし、SMBプロトコルはもう少し多い手順を踏む必要があります。この手順は、@ITやImplementing CIFSにて紹介されていました。具体的には①～⑤のようになります。

①TCPにてサーバのポート番号445に接続する
②ネゴシエートする（使用するダイアレクトが決定する）
③認証する（User IDが決定する）
④使用する共有リソースに接続する（Tree IDが決定する）
⑤ファイルアクセスする

各手順を踏むことで、必要な識別子を順に入手していく、という作業になります。言葉にするととても簡単な印象を持つかもしれませんが、とくに認証と共有リソース関連は、本当に、本当に、困難な道のりが待っていました。敵のアジトに潜入するには、数多くの関門を突破しなければならなかったのです。

パケットの構造

誰かと会話するには、双方が共通して理解できる言語が必要になります。日本語、英語などですね。それと同じで、クライアントとサーバが会話をする際にも、同じような約束事が必要です。プロトコルとは、まさにそのための約束事の総称です。ここからいよいよSMBプロトコルについての具体的な話に入って行きましょう。パケットやメッセージの構造を説明するための方法として、C言語の構造体のような表現をここでは使います。値の型は**表1**のように表現します。

まず、パケットの構造は、送信するメッセージの長さと、実際のメッセージのバイト列の2つで構成されます。具体的には、**リスト1**のようになります。

メッセージの長さが4バイトで書かれるのですが、重要な注意事項があります。それは、次の2点です。

・バイトオーダー：ビックエンディアン
・最大のサイズ長：
0xFFFFFF (16,777,215)

ビッグエンディアンは、たとえば0x12345678という数値がそのままの並び(0x12 0x34 0x56 0x78)で指定される方式です。これに対して、リトルエンディアンの場合は逆順(0x78 0x56 0x34 0x12)になります。つまり、最大のサイズ長が3バイト分でビッグエンディアンであれば、SMBパケットの最初の1バイトは常に0になります。たとえば、サーバから32バイト分の情報が送られてきたときは、**図6**のようなSMBパケットになっています。

サーバから送られてきたSMBパケットをクライアントが読み込む際には、ソケットからまず4バイト分読み込み、最初の1バイトが0であることをチェックします。そして、次の3バイトからメッセージのサイズ長を知ります。最後に、ソケットからメッセージのバイト列をサイズ長だけ読み込みます。このSMBパケットは、クライアントからサーバに送信する際にも、まったく同じ構造となります。

実は、ここで僕は敵の罠にまんまと引っかかりました。先ほど調べた文献には「SMBプロトコルはリトルエンディアンを採用している」と書かれていました。てっきり上記のサイズ長もリトルエンディアンだと思い込んでしまいました。たとえば、SMBパケットのサイズ長として0x12 0x34というバイト列が来た時、ビッグエンディアンであれば10進数で4,660ですが、リトルエンディアンとした場合は13,330となってしまうわけです。この場合は、クライアント

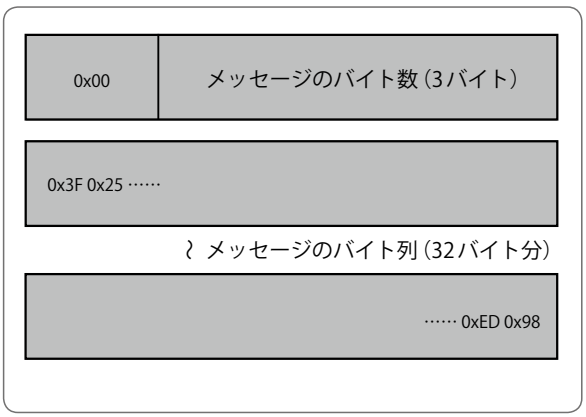
▼表1　本記事でのパケット、メッセージの各種値の型

型名	長さ(バイト長)	範囲
UCHAR	1	0～255
USHORT	2	0～65,535
SHORT	2	-32,768～32,767
UINT	4	0～4,294,967,295
ULONG	8	0～18,446,744,073,709,552,000
ANY	任意または指定バイト数	----

▼リスト1　SMBパケットの構造

```
SMB_PACKET {
  UINT packet_length; // メッセージのバイト長
  ANY message; // メッセージ(packet_length分のバイト長)
}
```

▼図6　32バイトのメッセージを持つSMBパケット



は「まだメッセージのバイト列がサーバから来るはずだ」として、来るはずがないバイト列を延々と待ち続けることになってしまいます。逆に、0x34 0x12とサイズ長が来た場合は、クライアントはSMBパケットの途中までしか読み込まず、「あれ、メッセージが欠損してる……」とエラー扱いすることになってしまいます。

ビッグエンディアンは、このサイズ長「だけ」です。この後はすべてリトルエンディアンが使われます。

メッセージの構造

SMBパケットに乗って送受信されるメッセージの構造は、文献を読むと意外ときっちり決められていて、実は比較的単純なものでした。**図7**のように、メッセージは3つの区画から成り立っています。先ほどのパケットの構造と同じように、メッセージの構造も送信と受信の両方が同じ構造となっています。

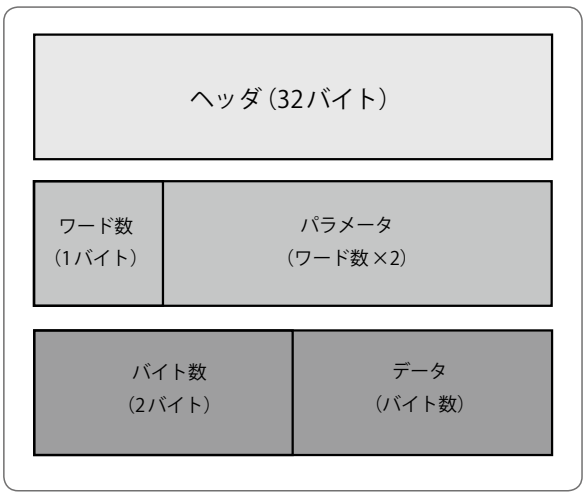
ヘッダは完全に決まりきった内容になっていて、メッセージの種別が何であれ、同じような内容となります。それに対して、パラメータとデータはメッセージの種別に応じてまったく異なる内容になります。すべてのメッセージにおいてヘッダは同じ内容ですし、説明もしやすく、ヘッダ内の各項目の値の意味も明確です。そのためか、@ITの記事やImplementing CIFS本の中でも、ヘッダの内容はかなり細かく説明されていました。

ヘッダは全体が32バイトのバイト列です。その中に、すべて長さが決められた複数の値が含まれています。ヘッダを構成する各値について、**リスト2**に示します。

最初のmessage_idは固定値です。1バイト目の0xFFに続く3バイト（0x53 0x4D 0x42）はASCIIコードで"SMB"となります。つまり、SMBのメッセージはすべて"0xFF+SMB"で始まるのです。これはSMBプロトコルをパケットキャプチャした際に、大きな目印になります。たいていパケットキャプチャする際には16進数とASCII文字列を同時に見れるようにしますので、0xFFに続いて"SMB"と出力された個所から始まるSMBのメッセージを簡単に見つけることができます。SMBプロトコルはやっかいな敵だと思っていましたが、わざわざ目印を残してくれるなんて意外でした。

次の1バイトはコマンドです。これは、クライアントがサーバに処理を要求する際の処理種別となります。SMB1/CIFSプロトコルでは、数多くのコマンドが定義されています。MS-CIFSには64個のコマンド値が定義されていて、

▼図7　ヘッダの区画



▼リスト2　ヘッダの各値

```
SMB_HEADER {
  UINT message_id; // 固定値 0xFF 0x53 0x4D 0x42
  UCHAR command; // コマンド
  UINT nt_status; // エラーコード（成功時は0）
  UCHAR flags; // フラグ1
  USHORT flags2; // フラグ2
  UCHAR[12] reserved; // 予約領域（12バイト分）
  USHORT tree_id; // ツリーID
  USHORT process_id; // プロセスID
  USHORT user_id; // ユーザID
  USHORT multiplex_id; // マルチプレックスID
}
```

サブコマンドも含めるともっと多くなります。DOS全盛の頃からWindows Vista以前まで、必要に応じて次々とコマンドが追加されてきました。似たようなコマンドが複数あったりするのを見ると、SMBプロトコルの歴史の深さを感じます。このコマンドに応じて、パラメータやデータの内容が変わってきます。つまり、コマンドの数だけパラメータやデータの形式がそれぞれ存在するわけです。途方に暮れそうです。

クライアントからの処理要求を正しくサーバが処理できたかどうか、その結果はnt_status値として返却されます。正常に処理された場合は0、何かエラーが発生した場合は0以外の数値がnt_statusにセットされます。nt_statusにセットされる可能性がある値は、MS-ERREFという文書の「2.3.1 NTSTATUS values[注4]」に掲載されています。クライアントの開発は、このnt_status値とのにらめっこです。Try & Errorを繰り返し、nt_status値が0のレスポンスをサーバから得るべく、試行錯誤をひたすら行うという苦行です。"0xFF+SMB"から近いので、**図8**のように、パケットキャプチャの結果から一瞬でレスポンスがエラーだったのかどうか判別ができます。

ヘッダの中で最も困った項目、それはflagsとflags2です。これらの値はビット単位で意味があるのですが、1ビットでも間違えるとSMBプロトコルでのやりとり全体に影響してきます。たった1ビットの間違いで、解決に数日を費やしたこともありました。"0xFF+SMB"なんてわかりやすい目印を晒している割には、非常に細かなところにトラップがあるという、敵の執念さが伝わってきます。

flagsとflags2の合計24ビット分のフラグ値について、パケットキャプチャしてMac OS XやWindowsなどのクライアントOSからサーバへの通信を覗き、さらに自分で試した結果導き出した僕なりの値は**リスト3**となります。

「長いパス名やUNICODEなんて当たり前だろ」と思うかもしれませんが、SMBプロコトルがDOSの時代から使われていたことを忘れてはなりません。**リスト3**の設定を怠ると、サーバはDOS時代の環境(8.3のパス名やASCIIのみ文字列など)を想定した挙動になってしまいます。

よほど古いサーバでなければ、そのサーバからのレスポンスに含まれるヘッダのflags、flags2の値は、クライアントで指定した値とほとんど同じ値を返してくるはずです。ただし、flagsのみ8番め(一番左)のビットを立てて返却してきます。上記のコード例であれば、0x98になります。この理由は、8番めのビットが「0:クライアントから

▼図8　パケットキャプチャにおけるnt_status値

```
0x0000:  98e0 d99b 84ff 50e5 493e 41df 0800 4500  ......P.I>A...E.
0x0010:  00b7 fdb6 4000 4006 ba84 c0a8 00aa c0a8  ....@.@.........
0x0020:  000b 01bd f567 b029 536b 2570 7251 8018  .....g.)Sk%prQ..
0x0030:  2086 4b6d 0000 0101 080a 6ad2 ce36 18e9  ..Km......j..6..
0x0040:  6142 0000 007f ff53 4d42 7200 0000 0088  aB.....SMBr.....
0x0050:  01c8 0000 0000 0000 0000 0000 0000 ffff  ................
0x0060:  0100 ffff 0000 1100 0003 3200 0100 ffff  ..........2.....
0x0070:  0000 0000 0100 0740 0000 fdf3 8080 b69c  .......@........
0x0080:  4151 e8c7 d001 e4fd 003a 0066 7265 656e  AQ.......:.freen
0x0090:  6173 0000 0000 0000 0000 0060 2806 062b  as.........`(..+
0x00a0:  0601 0505 02a0 1e30 1ca0 0e30 0c06 0a2b  .......0...0...+
0x00b0:  0601 0401 8237 0202 0aa3 0a30 08a0 061b  .....7.....0....
0x00c0:  044e 4f4e 45                             .NONE
```

▼リスト3　採用したflags, flags2の値

```
flags = 0x18
      SMB_FLAGS_CANONICAL_PATHNAMES (0x10) // 正規化されたパスを利用
    | SMB_FLAGS_CASELESS_PATHNAMES (0x08) // 大文字小文字区別なし
flags2 = 0xC803
      SMB_FLAGS2_UNICODE_STRING (0x8000) // UNICODE利用[注5]
    | SMB_FLAGS2_32BIT_STATUS (0x4000) // NT_STATUSを利用
    | SMB_FLAGS2_EXTENDED_SECURITY (0x0800) // 拡張セキュリティを利用
    | SMB_FLAGS2_EAS (0x0002) // 拡張属性を利用
    | SMB_FLAGS2_KNOWS_LONG_NAMES (0x0001) // 長いパス名を利用
```

注4) https://msdn.microsoft.com/en-us/library/cc704588.aspx
注5) SMBプロトコルのUNICODE符号化方式は、UTF-16LEです。

の要求 1:サーバからの応答」という意味があるからです。

process_idは、サーバに接続しているクライアントが複数のプロセスから要求を出している際に指定しておくと良い値です。1つのプロセスからシーケンシャルに通信を行っているだけであれば、この値は接続から切断まで1つの値(0でも可)を指定しておけば良いです。multiplex_idもprocess_idと同じような用途になりますが、こちらは接続から切断まで、要求を出すたびに0から1ずつ値を増加させて指定しておけば良いです。サーバからのレスポンスのヘッダに同じ値が入ってくるので、どのリクエストに対するレスポンスなのかわかるようになっています。

最後のuser_id、tree_idは、サーバから発行してもらう値です。SMBプロトコルという敵を攻略すること、それはuser_id、tree_idを入手すること、と言い換えても過言ではありません。

ネゴシエートする

SMBパケットとヘッダの構成がわかれば、いよいよ実際にサーバにパケットを送って結果を得ることができます。SMBプロトコルで最初に行うことは、ネゴシエートです。これによって、サーバとクライアントで使用するダイアレクトが決定されます。これは比較的簡単なメッセージ構成なので、Implementing CIFS本にも明確に説明されていました。

ネゴシエートするためには次のメッセージをサーバに届けます。

・command：SMB_COM_NEGOTIATE(0x72)
・パラメータ：なし
・データ：クライアントがサポートするダイアレクト一覧

ダイアレクトは、プレフィックスに0x02、サフィックスに0x00(Null-terminated String)を付けます。もし複数のダイアレクトをサポートできる場合は、“0x02……0x00”を複数並べます。たとえば、“NT LM 0.12”のダイアレクトをサポートするクライアントは、**リスト4**のようにパラメータおよびデータを指定します。

リスト4の内容を持つSMBパケットは、**リスト5**のようになります。

ヘッダには、flags、flags2は先ほど紹介した値を、tree_id、user_id、process_id、multiplex_idは0を指定しています。“NT LM 0.12”のASCIIコードの前後に0x02、0x00を付けて、この長さ0x000cをデータバイト数としています。パラメータは何もないので、単に0x00を指定しています。コードを見てわかるように、0x000cを0x0c 0x00というようにリトルエンディアンにしています。flags2も同様です。

クライアントからサーバの445ポートにTCPで接続し、**リスト5**の内容を送信します。すると、**リスト6**のようなSMBパケットが返ってきます。

“ff53 4d42 72”の後に“00”が4つ続いていま

▼リスト4　SMB_COM_NEGOTIATEリクエストの構造

```
SMB_PARAMETER {
  UCHAR word_count = 0
}
SMB_DATA {
  USHORT byte_count = 36
  bytes {
    ANY dialect = "¥x02NT LM 0.12"
  }
}
```

▼リスト5　ネゴシエート要求のSMBパケット

```
0000 002f ff53 4d42 7200 0000 0018 03c8
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 000c 0002 4e54 204c 4d20 302e
3132 00
```

▼リスト6　ネゴシエート結果のバイト列

```
0000 007f ff53 4d42 7200 0000 0088 03c8
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 1100 0003 3200 0100 ffff 0000
0000 0100 3577 0000 fdf3 8080 6965 84cc
b1c8 d001 e4fd 003a 0066 7265 656e 6173
0000 0000 0000 0000 0060 2806 062b 0601
0505 02a0 1e30 1ca0 0e30 0c06 0a2b 0601
0401 8237 0202 0aa3 0a30 08a0 061b 044e
4f4e 45
```

す。つまり、サーバはネゴシエート要求を正常に受け付けて、nt_status=0、つまり成功したことを返してきました。敵とのファーストコンタクトに成功したようです。サーバと正しく会話できたときは、いつでも感動します。

リクエストの約3倍の長さのバイト列が返ってきました。この中身を紐解いていきましょう。ヘッダ部分はリクエストの内容とほとんど変化がないので、パラメータとデータについて、**リスト7**に構造とレスポンスの値を示します。

いろいろ値が返ってきましたが、とくに重要な値は次のようになります。

- dialect_index：サーバが選んだダイアレクト
- security_mode：サーバが期待するユーザ認証の方法
- capabilities：サーバがサポートする機能のフラグ
- security_blob：ユーザ認証時に使用されるバイト列

たとえばリクエストでダイアレクトに[Samba, NT LM 0.12, CIFS]を指定して、サーバが"NT LM 0.12"までサポートしているSambaだった場合は、dialect_index値は1が返却されます。ダイアレクトによってサポートされていないコマンドがありますので、クライアントはサーバが選んだダイアレクトから利用可能なコマンドを選択して、処理要求をサーバに投げていくことになります。

Sambaをインストールしたことのある方は、smb.confファイルで"security = user/share/domain"といった設定をした経験を持っていると思います。このsecurity設定によって、SMBプロトコル上でのユーザ認証の方法が変わってきます。security_mode値は、そういったサーバの設定値が反映されています。とくにsecurity_mode値の右から1ビット目、2ビット目は重要です。今回の例の場合は、security_mode=3、つまりUSER_LEVELはUserであり、Challenge/Responseでのユーザ認証処理が求められています。

- 1 bit：USER_LEVEL (0: Share, 1: User)
- 2 bit：CHALLENGE_RESPONSE (0: Plain text password, 1: Challenge/Response)

security_blobバイト列は、ユーザ認証時のChallenge/Responseを行う際に使用されます。もしサーバがPlaintext passwordを選択していた場合は、このsecurity_blobバイト列ではなく、パスワードを暗号化する際に利用するencryption_keyバイト列が送られてきます。

ネゴシエートのやりとりで、かなり敵に近づいた印象がありますね。しかし、問題はここからでした。(以下、次回に続く!)SD

▼リスト7　ネゴシエートレスポンスのパラメータとデータ

```
SMB_PARAMETER {
  UCHAR word_count = 17; // 34バイト
  words {
    USHORT dialect_index = 0;
    UCHAR security_mode = 3;
    USHORT max_mpx_count = 50;
    USHORT max_number_vcs = 1;
    UINT max_buffer_size = 65535;
    UINT max_raw_size = 65536;
    UINT session_key = 0x7735;
    UINT capabilities = 0x8080f3fd;
    UINT system_time_low = 0xcc846569;
    UINT system_time_high = 0x01d0c8b1;
    SHORT server_time_zone = 0xfde4;
    UCHAR encryption_key_length = 0;
  }
}
SMB_DATA {
  USHORT byte_count = 58;
  bytes {
    UCHAR[16] guid = 0x667265...000000;
    UCHAR[42] security_blob = 0x602806...4f4e45;
  }
}
```

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10月号

第1特集
多層防御や感染後対策を汎用サーバに実装
攻撃に強いネットワークの作り方

第2特集
Webメールの教科書
クラウドサービス利用か？　自社で構築か？

特別付録
・創刊300号記念　Vim&Emacsチートシート

定価（本体1,220円+税）

2015年9月号

第1特集
特講
正規表現・SQL・オブジェクト指向
苦手克服のベストプラクティス

第2特集
メールシステムの教科書
日本語もバイナリもちゃんと届くのはなぜか

特別企画
・なぜ俺の提案は通らないのか？

定価（本体1,220円+税）

2015年8月号

第1特集
Lispより始めよ、されば救われん！
なぜ関数型プログラミングは難しいのか？

第2特集
安全な通信を確保する
SSL/TLSの教科書

短期連載
・AWSで始めよう！　モダンなJavaアプリケーション開発

定価（本体1,220円+税）

2015年7月号

第1特集
あなたにもできる！
ログを読む技術［セキュリティ編］

第2特集
黒い画面（tmux）の使い方
プロになるためのターミナル活用術

第3特集
6人の先駆者に訊く
スペシャリストになる方法

定価（本体1,220円+税）

2015年6月号

第1特集
新人さん歓迎特集
Git&GitHub入門

第2特集
OpenLDAPの教科書
ユーザ／ネットワーク管理の基本と活用例

一般記事
・SambaによるActive Directoryの機能性と移行性を検証する

定価（本体1,220円+税）

2015年5月号

第1特集
テキスト処理ベーシックレッスン
手を動かしてデータを操ろう！

第2特集
ファイル共有自由在在
［徹底入門］最新・Sambaの教科書

特別付録
・3分間HTTP&メールプロトコル基礎講座［特別編］

定価（本体1,300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。

Googleが作った OS Xクライアント管理ツール「munki」

後編●クライアントのインベントリ情報を収集する

Author やまねひでき　Mail henrich+munki@gmail.com

前編ではオープンソースのMacのクライアント管理ソフト「munki」について、その導入方法とソフトウェアパッケージの配布のしかたを取り上げました。今回は、管理対象のクライアントの状態とそのインベントリ情報の取得について説明をします。

インベントリの取得とダッシュボード

munkiを入れることによって、ソフトウェアの配布は容易になりました。これがちゃんとできるようになると、次はどの程度のクライアントがきちんと更新を適用しているかなどの「現状把握」をしたくなるのではないでしょうか。munkiではクライアントからさまざまなインベントリデータを取得できますが、ここではインベントリデータをまとめてわかりやすく「ダッシュボード」として表示するツールである「munkireport」を紹介します。

munkireport

munkireport[注1]は、Arjen van Bochoven氏らによって開発されている、PHPで記述されたmunkiのインベントリダッシュボードです。**表1**にあるような非常に多彩な情報を、Bootstrap[注2]を使ったモダンな見やすいインターフェースで確認することができます(**図1**)。

munkireportの設定

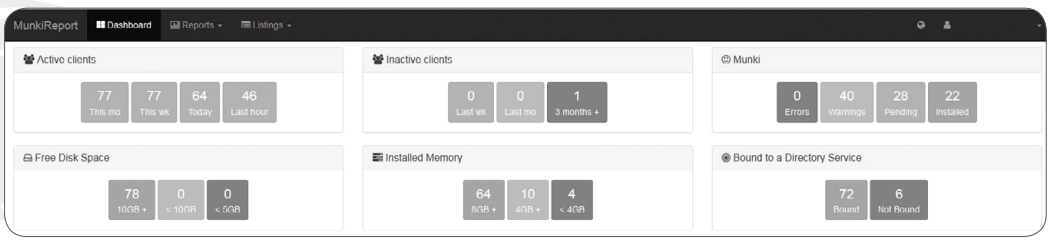
サーバ側の設定

ここではmunkiをセットアップしたMacサー

▼表1　munkireportでチェックできる情報(一部)

マシン名
ユーザ名
シリアル番号
MACアドレス
インストールしているソフトウェア一覧
FileVaultによる暗号化の状態
Time Machineの動作状況
ActiveDirectoryへのバインド設定
稼働時間
空きディスク容量
マシン型番(例：「MacBook Pro(Retina, 13-inch, Late 2013)」)
搭載CPU、メモリ、ディスク容量、接続しているディスプレイ
保証期間の有無(AppleCare加入かどうかを含む)
Bluetooth情報
インストール待ちになっているソフトウェアの名前
munkiのバージョン情報
バッテリの状態

▼図1　munkiのトップページ



注1) URL https://github.com/munkireport/munkireport-php
注2) WebサイトやWebアプリケーションを作成するためのCSSフレームワーク。HTML、CSS、JavaScriptから成るデザインテンプレートが収録されている。

バの/Users/Shared以下にmunkireportを配置することにします。GitHubよりソースをcloneして設定を行います(**図2**)。

次にApacheの設定を変更します。必要なのは、

- PHPモジュールを有効にする
- munkireportディレクトリの.htaccessを有効にする

の2点です。**リスト1**のように、/private/etc/apache2/httpd.confファイルを編集してください。

編集が終わったら「sudo apachectl restart」と実行して、変更を反映させてください。

ここまでできたら、クライアントから「http://<サーバ名>/munkireport-php/index.php」にアクセスすると、ユーザ名とパスワードの設定を求められます(**図3**)ので、適当なユーザ名とパスワードを指定して「Generate」をクリックします。

入力情報をもとにして「$auth_config['<ユーザ名>'] = '<ハッシュ値>';」という行が生成されて表示される(**図4**)ので、この行をconfig.phpに追記します(**リスト2**)。

これで再度「http://<サーバ名>/munkireport-php/index.php」にアクセスすれば、指定したユーザ名とパスワードでmunkireportが表示できるよ

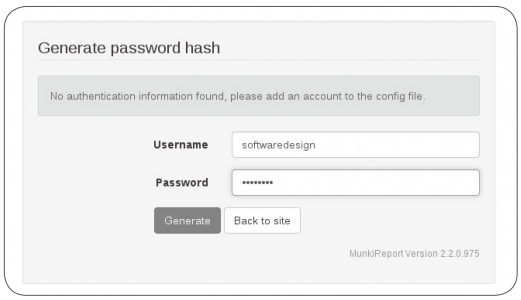
▼図2　munkireportを入手し、権限などの設定を行う

```
$ cd /Users/Shared
$ git clone https://github.com/munkireport/munkireport-php.git
$ cd munkireport-php
$ echo "<?php" > config.php
$ sudo chown -R _www app/db  ←httpdの動作ユーザで読み取りできるように権限を変更
 ↓ApacheのRootDocumentからシンボリックリンクで参照できるようにする
$ sudo ln -s  /Users/Shared/munkireport-php /Library/WebServer/Documents/
```

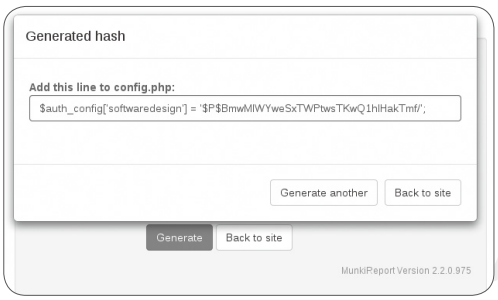
▼リスト1　httpd.confを変更する

```
 ↓行頭のコメントを外す
#LoadModule php5_module        libexec/apache2/libphp5.so  ←変更前
LoadModule php5_module         libexec/apache2/libphp5.so  ←変更後

 ↓以下を追加する
<Directory "/Library/WebServer/Documents/munkireport-php">
    AllowOverride All
</Directory>
```

▼図3　パスワードのハッシュ値の生成(その1)



▼図4　パスワードのハッシュ値の生成(その2)



▼リスト2　図4の場合のconfig.phpの中身(2行)

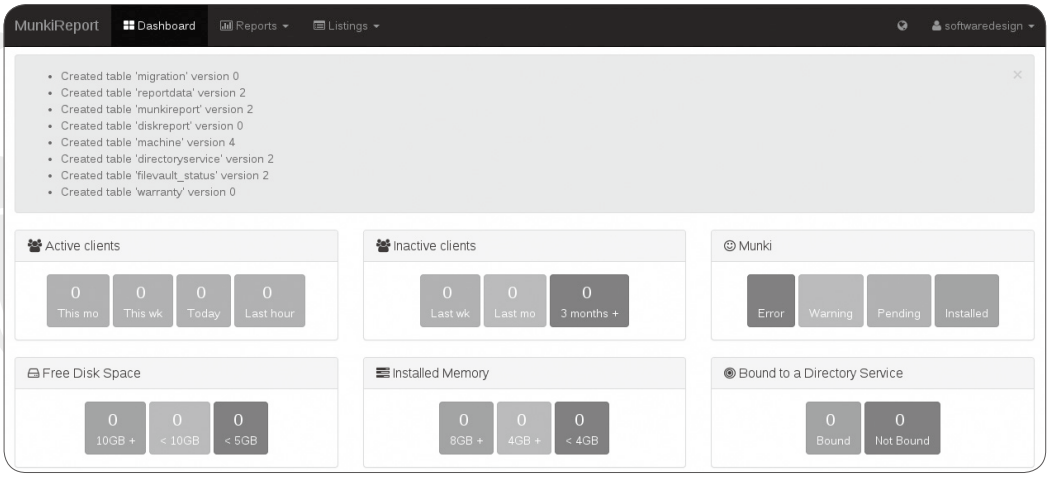
```
<?php
$auth_config['softwaredesign'] = '$P$BmwMIWYweSxTWPtwsTKwQ1hlHakTmf/';
```

うになります[注3]。**図5**の画面が出ればOKです。

クライアント側の設定

munkireportサーバの設定は以上で完了したので、次にクライアントがmunkireportにインベントリ情報を追加するように設定しましょう。munkitoolsをインストールしてあるOS Xクライアントで、**図6**のように「sudo /bin/bash -c

▼図5　初回アクセス時のmunkireportの画面



▼図6　クライアントに、munkireportの導入に必要なモジュールをインストールする

```
$ sudo /bin/bash -c "$(curl -s http://localhost/munkireport-php/index.php?/install)"
Preparing /usr/local/munki/ and /Library/Preferences/MunkiReport
BaseURL is http://localhost/munkireport-php/
Retrieving munkireport scripts
Configuring munkireport
+ Installing ard
+ Installing bluetooth
+ Installing directory_service
+ Installing disk_report
+ Installing displays_info
+ Installing filevault_status
+ Installing installhistory
+ Installing inventory
+ Installing localadmin
+ Installing munkireport
+ Installing network
+ Installing warranty
Installation of MunkiReport v2.2.0 complete.
Running the preflight script for initialization
    Munkireport: # Executing scripts in preflight_abort.d
    Munkireport: # Executing scripts in preflight.d
    Munkireport: Running bluetooth.sh
    Munkireport: Running directoryservice.sh
    Munkireport: Running disk_info
    Munkireport: Running displays.py
    Munkireport: Running filevaultstatus
    Munkireport: filevaultstatus Error: /usr/local/munki/preflight.d/filevaultstatus: line
15: fdesetup: command not found 注4
    Munkireport: Running localadmin
    Munkireport: Running networkinfo.sh
    Munkireport: Running warranty
    Munkireport: Running submit.preflight
```

注3） 誰でも閲覧できるようにしたい場合は、config.phpのハッシュ行を削除して「`$conf['auth']['auth_noauth'] = array();`」を追加します。また、指定したパスワードがわからなくなったら、config.phpの設定行を削除して、再度アクセスして生成すれば問題ありません。

注4） この実行の場合は、OS X 10.6環境のため、FileVault2の検知でエラーが出ている。

"$(curl -s http://<munkiサーバの名前>/munkireport-php/index.php?/install)」と実行して、munkireportが必要とするモジュールをインストールします(たまにエラーが出るので、その場合は再度実行してください)。

以上でインベントリ情報がサーバに送信され、munkireportで確認できるようになりました。munkireportの画面左上で「Active clients」の数字が増えたことと「New clients」でクライアントが登録されたことを確認してください(**図7**)。

ここまでできたら、あとは順次、munkireportの設定をほかのクライアントにも展開していくだけです。次の手順で、munkiを使って各クライアントへ展開します(**図8**)。

①サーバ上で「curl -s http://<サーバ名>/munkireport-php/index.php?/install/plist -o /Users/Shared/munki_repo/pkgsinfo/MunkiReport.plist」として、設定をMunkiReport.plistファイルに保存(ここでは/Users/Shared/munki_repo/がmunkiのリポジトリ)

②サーバでmakecatalogsを実行して、①のMunkiReport.plistファイルに保存したmunkireport参照設定をパッケージとしてcatalogに反映

③manifestutilsでmunkireportの設定が強制インストールされるように設定

munkireportの画面で徐々にクライアントで適用されていくのを確認し、問題があるようであればそのクライアントでのmunkiの動作に異常がないかをチェックしてください。

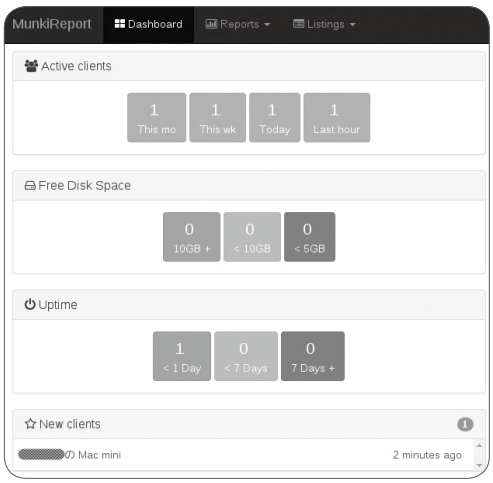
ダッシュボード画面

munkireportでは「ダッシュボード」「レポート」「一覧」の3つの情報を表示できます(**図9**)。まずはトップ画面で表示されているダッシュボードの情報について説明します。

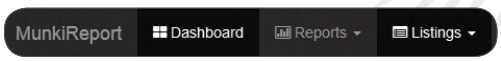
Active clients

今月／今週／今日／この1時間以内、に何台のmunkiクライアントがmunkiサーバにアクセスしにきているかを表示します(**図10**)。

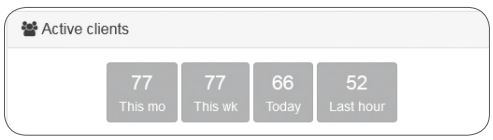
▼図7　munkireportの画面でインベントリ情報を確認できるようになった



▼図9　munkireportのメニュー



▼図10　Active clients



▼図8　munkiを使って、各クライアントにmunkireportの設定を展開する

```
$ curl -s http://<サーバ名>/munkireport-php/index.php?/install/plist -o /Users/Shared/munki_repo/pkgsinfo/MunkiReport.plist ←①
$ makecatalogs ←②
$ manifestutils ←③
> add-pkg MunkiReport --manifest testing --section managed_installs
```

Inactive clients

Active clientsの逆で、今週／今月／この3ヵ月、にアクセスがないmunkiクライアントの台数を表示します(**図11**)。ここに表示されるクライアントを確認して、0に近づけましょう。

Munki

Munkiでは配布するアプリケーションについて、エラー／警告／インストール予定(Pending)／直前で実施されたインストール、の数を表示します(**図12**)。エラーと警告は内容を確認し、必要に応じて設定を変更して0に近づけるようにしましょう。

Free Disk Space

空きディスク容量について表示します(**図13**)。

Installed Memory

搭載されているメモリ容量について表示します(**図14**)。

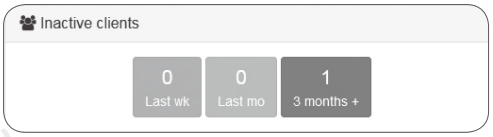
Bound to a Directory Service

ActiveDirectory(以下、AD)などのディレクトリサービスに参加(バインド)しているクライアント数を表示します(**図15**)(ADへの参加のしかたはコラムTips 2を参照)。

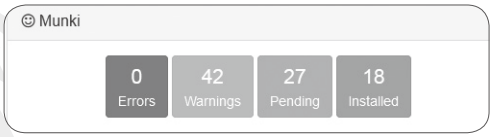
Uptime

連続稼働時間を表示します(**図16**)。

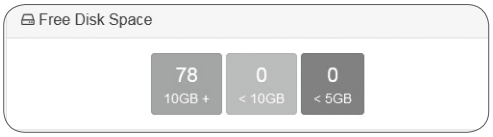
▼図11　Inactive clients



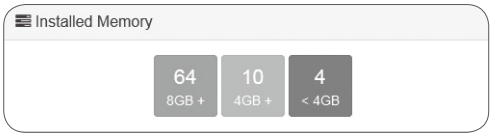
▼図12　Munki



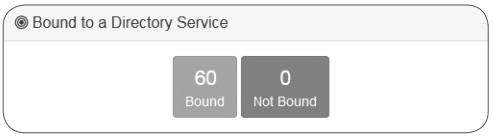
▼図13　Free Disk Space



▼図14　Installed Memory



▼図15　Bound to a Directory Service



COLUMN　Tips 1　munkiでダウンロードについてのWarningが出る場合

筆者がmunkiを使用しているときに何度か経験したのですが、とくに設定が間違っていないはずなのにソフトウェアの配布が実行されないことがありました。munkireportを入れたことでWarningに気づき、ログ(**リストA**)を確認してやっと原因がわかったのですが、ファイル権限が適切に設定されない関係でダウンロードができないことがあるようです。

これはmunki_repo/pkgs/foo/xxxxx.dmgの読み取り属性がなぜか落ちていることが原因ですので、「`chmod +r munki_repo/pkgs/foo/xxxxx.dmg`」などとして、対象のファイルの読み取り権限を付けてあげてください。

▼リストA　munkireportに表示されていたログ

```
Download of xxxxx failed: HTTP result 403: forbidden
```

COLUMN　Tips 2　OS XをWindowsのようにADのクライアントにする

Macを Active Directoryに参加(バインド)させると、ローカルに管理用アカウントを追加しなくとも、いつもWindows端末の管理に使っているのと同じADのアカウントで、Macへのログインや管理作業ができるようになります。ADに参加させるにはクライアントのMacで次の手順を行います。

①アプリケーションの「システム環境設定」から「ユーザとグループ」を選択
②「ログインオプション」を選択
③「ネットワークアカウントサーバ」で「接続」ボタンをクリック
④「サーバ」欄にWindowsドメイン名をFQDN形式で入力
⑤自動的にWindowsドメインへ接続し、図Aの画面が開く。「AD管理ユーザ」[注5]と「パスワード」を入力して「OK」をクリックし、マシンアカウントを登録

マシンアカウントの登録後に、ADの適当なグループに参加しているユーザにも管理者権限を与えます。

▼図A　マシンアカウントの登録

⑥再度、「ネットワークアカウントサーバ」で「接続」ボタンをクリック
⑦ディレクトリユーティリティで「Active Directory」をダブルクリック
⑧ディレクトリユーティリティの画面で詳細が隠れているので表示させ(図B)、「管理」タブで「管理を許可するユーザ」にチェックを入れる

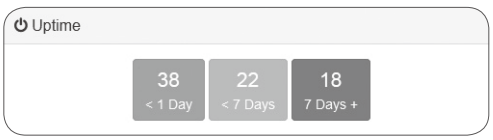
これで、管理クライアント(Mac)にローカルユーザを登録していなくても、Windows管理の際に利用しているADユーザ[注6]でログインできます。アプリケーションのインストール、システム環境設定の変更などのローカルの管理者アカウントが必要な作業も実施することができます。

▼図B　ユーザに管理者権限を付与

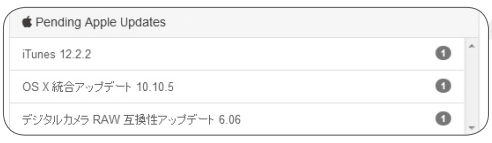
Pending Apple Updates

AppStoreアプリで表示されるOSアップデートをmunkiで管理するように設定した場合(コラム Tips 3参照)、Pendingになっているアップデートを表示します(**図17**)。

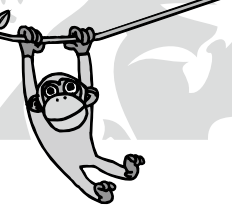
▼図16　Uptime



▼図17　Pending Apple Updates



注5）ここで入力するアカウントは、ドメインにマシンを追加できるActive Directoryの管理アカウントである必要があります。
注6）domain adminsならびにenterprise adminsグループに参加しているユーザ。



COLUMN Tips 3 OS X自体のアップデートをmunkiでトラッキングする

AppStoreアプリだけではなく、Managed Software Center(以下、MSC)にも、OS Xのアップデートを表示させる方法があります[注7]。

そのためには、まず各クライアントで**図C**のとおりに実行します[注8]。設定の確認には、**図D**を実行し、出力で「InstallAppleSoftwareUpdates = 1」となっていればOKです。

これでMSCを使ってアップデートのチェックを行うと、これまでAppStoreで行っていたOSのアップデートもまとめてチェックできるようになり、munkireportでも適用状況のトラッキングが可能になります。

ただ、この設定には2つほど欠点もあります。それまで筆者の環境では、AppStoreでの更新については、OS Xサーバの「キャッシュサービス」機能を使ってトラフィックを削減し、ダウンロードスピードを上げていました。ところが、このMSCの設定を行うと、全クライアントがLAN内のOS Xサーバからではなく、Appleからアップデートを直接ダウンロードすることになります。そのため、キャッシュサービスの恩恵を受けられなくなり、時間が大きくかかるようになりました。これが1つめの問題です。

この対処はOS Xサーバで「ソフトウェア・アップデート・サービス」を動作させて、クライアント側でOS Xサーバを参照するように**図E**のような設定を行います。

もう1つの問題は、MSCでアップデートチェックを行う際、Apple Software Updatesのチェックに大きく時間がかかる場合があることです。サーバ側のアップデートリストカタログが少しでも変更されていたり、クライアント側でアップデートを行って状態が変わったりすると、数秒程度で終わっていたチェックに数分あるいは数十分以上かかるようになりました。この問題については、対処法が見つけられませんでした。

▼図C MSCでOS Xのアップデートを表示させるように設定する

```
$ sudo defaults write /Library/ ➪
Preferences/ManagedInstalls.plist ➪
InstallAppleSoftwareUpdates -bool True
```

▼図D 設定後の確認

```
$ defaults read /Library/Preferences/ManagedInstalls. ➪
plist
{
    AppleSoftwareUpdatesOnly = 0;
    ClientIdentifier = "testing";
    DaysBetweenNotifications = 1;
    InstallAppleSoftwareUpdates = 1; ←こうなっていればOK
 (...略...)
```

▼図E クライアントがOS Xサーバを参照するように設定する

```
$ sudo defaults write /Library/Preferences/com.apple. ➪
SoftwareUpdate CatalogURL http://su.example.com:8088/ ➪
index.sucatalog
 (su.example.comはOS Xサーバ のURL)
```

Pending Installs

クライアントでインストール／アップデート対象になっているが、まだインストールが完了していないソフトウェアの一覧[注9]とその数を表示します(**図18**)。なるべく数が0になるようにしましょう。

▼図18 Pending Installs

注7) MSCはOSのアップデートは配信しません。トラッキングして表示するだけです。
注8) 似たような名前のオプションで「AppleSoftwareUpdatesOnly」というのがありますが、こちらは通常のアプリケーションのアップデートチェックをいっさい行わないものになりますので、間違わないようにしてください。
注9) インベントリ情報はソフトウェアのインストール前に送信されます。そのため、Managed Software Centerを実行してソフトウェアをインストールした直後はまだ「Pending」として報告されてしまいますので、ご注意ください。

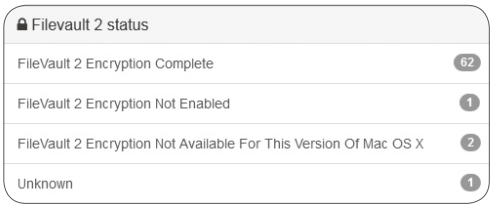
Filevault 2 status

FileVaultによるディスク暗号化が実施されているかを、完了した／有効になっていない／そもそもこのバージョンのOS Xでは使えない／不明、などというステータスで表示します(**図19**)。なるべくすべてのクライアントが暗号化を実施するようにしたほうが良いでしょう。

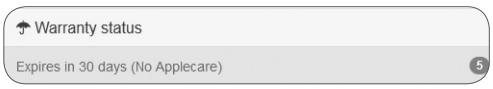
Warranty status

Appleによるハードウェア保証が1ヵ月以内に切れるクライアント数を表示します(**図20**)。

▼図19　Filevault 2 status



▼図20　Warranty status



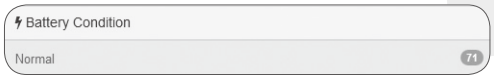
Battery Condition / Battery Health

バッテリ状態を表示します(**図21、22**)。異常が出ているようであれば、最悪、発火などの危険性もありますので即座に対応しましょう。

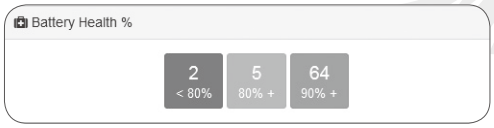
各クライアントの詳細表示

ダッシュボードデータからドリルダウンしてクライアント名をクリックすると、各クライアント情報のサマリを表示できます(**図23**)。筆者は、故障したMacを修理に出す際に、この表示のおかげで背部のシリアルナンバー情報をいちいち手打ちで入力しなくていいので、重宝しています。

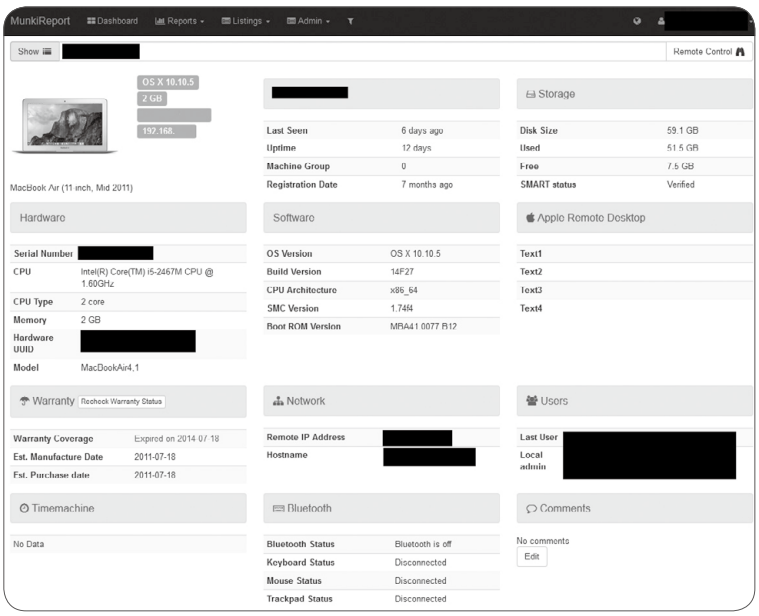
▼図21　Battery Condition



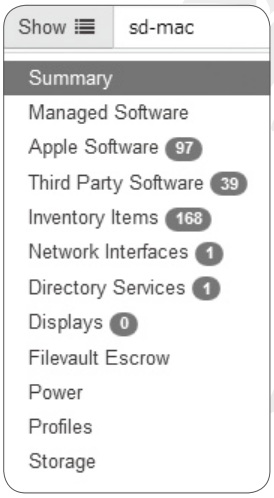
▼図22　Battery Health

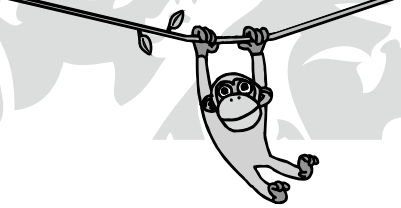


▼図23　各クライアント情報のサマリ画面



▼図24「Show」のクリック時に表示されるリスト





また、同一セグメントにいないためにBonjour[注10]で名前解決できないMacについては、この画面右上の「Remote Control」から画面共有やSSHですぐ接続できるのも便利です。

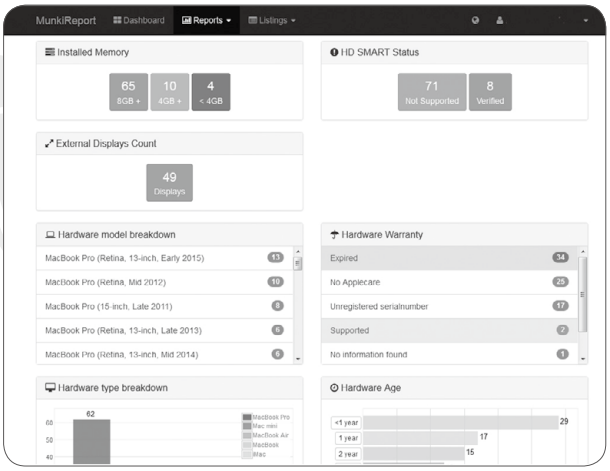
左上の「Show」となっている部分をクリックすると、クライアントに関する各情報のリストが出てきます(**図24**)ので、必要に応じて内容を確認しましょう。

ほかの画面 ——「Reports」「Listings」

「Reports」ではある1つの物事に合わせた分析結果が表示されます(**表2**)。たとえば、「Hardware report」(**図25**)では、今、利用しているMacの搭載メモリ量は何なのか、外部ディスプレイはいくつつながっているのか、どのモデルが使われているのか、などが表示されます。数字とグラフが表示されるので、感覚的に把握する際に便利です。

「Listings」ではクライアントが一覧表示されます。ここから右上の「search」欄で絞っていき、必要な情報を得る形になります。たとえば、「Clients」でクライアント一覧(**図26**)を出して、そこからバージョンが古いものを

▼図25　Hardware reportの画面



▼図26　Clients reportの画面

▼表2　「Reports」で表示できる内容

名称	内容
AppVersions	config.phpで指定したアプリケーションのバージョンを表示する。初期状態ではSafariのみ
Backup report	今日バックアップが完了したクライアント数、1日あるいは1ヵ月以上バックアップされていないクライアント数
Client report	クライアントのハードウェア(MacBook Proなのか Airなのか)のグラフやOSバージョンのグラフなど
Configuration report	重複したコンピュータ名、コンピュータ名とADで登録されたコンピュータ名の不一致の確認、ADにバインドした数など
Hardware report	搭載メモリ量、ハードウェアの経年数、ハードウェア保証など
Munki report	munkiのエラー／警告数、「pending」になっているアプリケーション、munki自体のバージョンなど
Network report	config.phpで指定したネットワーク情報に合わせてクライアントの配置を表示
Power report	バッテリの状態

注10) AppleによるZeroconf実装の1つで、人手による操作を介さず、かつ特別なサーバを使わずに、利用可能なIP機器一覧を検出／作成する。Linuxなどでは同様の実装としてAvahiがある。

「search」欄に入れてアップデートが必要なクライアントと利用者を確認する、というようなことができます。

「Listings」で表示できる内容の一部を**表3**に挙げます。

最後に

配布ツールであるmunkiに、ダッシュボードmunkireportを組み合わせることで、管理対象となるOS Xクライアントの現状が把握できるようになります。クライアントの状態、導入されているアプリケーションの種類とバージョン、暗号化／バックアップ／ディレクトリへの参加などの作業漏れも容易にわかるようになります。

これから複数のOS Xクライアントを管理する必要がある場合、ここで紹介したツールの導入を検討してみてください。きっと日常業務の負担が軽減されることでしょう。SD

▼表3　「Listings」で表示できる内容（抜粋）

名称	内容
Apple Remote Desktop	Apple Remote Desktopで設定されているTextメッセージ
Bluetooth	Bluetoothの有効／無効、キーボード／マウス／トラックパッドの接続有無、バッテリの残量
Certificates	発行した証明書と証明書の期限切れについての情報
Clients	一般的なクライアント情報
Directoryservice	バインドしているディレクトリ情報
Disk	ディスク利用パーセンテージと残量など
Displays	内蔵／外部ディスプレイ情報
Hardware	Macのスペック情報など
Inventory	インストールされているアプリケーション情報
Munki	munkiの動作情報
Network	接続デバイス、MACアドレス、IPアドレス情報
Power	バッテリの状態、温度など
Security	暗号化の状態など
Timemachine	最後に成功したバックアップがどのぐらい前なのか、どのぐらい時間がかかったのか、エラーメッセージなどの情報
Warranty	保証状態といつ保証が切れるかの情報

COLUMN　もう1つの実装「munkiwebadmin」

もう1つのインベントリ管理ソフトに「munkiwebadmin」[注10]があります。こちらはmunkiと同じGoogle製のプロダクトで、Django（PythonのWebアプリケーションフレームワーク）で作られた実装です。munkireportと比較すると閲覧可能な情報は少なめですが、「webadmin」という名前のとおり、munkiに関する設定の変更がブラウザから一部可能になっています。

注意点として、munkireportとmunkiwebadminはそのままでは併用できません。お互いのクライアント用インストーラスクリプトが同じ/usr/local/munki/{pre,post}flight.dを上書きしあうからです。どちらかを選んで利用しましょう。

本稿執筆時点で筆者がmunkireportとmunkiwebadminを両方使った感想としては、munkireportのほうがインベントリ情報が充実していると思いますが、munkiwebadminは2015年1月に新しい実装[注11]が出てきています。先ほどのmunkireportと同じようにbootstrapを使ったモダンなインターフェースになっていますので、今後の動向を注目してみてください。

注10）URL https://github.com/munki/munkiwebadmin
注11）URL https://github.com/SteveKueng/munkiwebadmin

短期集中連載

Jamesのセキュリティレッスン

パケットキャプチャWiresharkの新展開

第6回

キャプチャファイルからポートスキャンの結果を推測してみよう

Writer 吉田 英二（Eiji James Yoshida）
合同会社セキュリティ・プロフェッショナルズ・ネットワーク（http://www.sec-pro.net/）

はじめに

みなさん、こんにちは！　10月に息子とサカナクションのライブを観に行くEiji James Yoshidaです。息子はサカナクションをよく聴いていて、「10月のライブに行ってみたい！」と言い出したので、筆者も一緒に行くことにしました。息子とライブを観に行く日が来るなんて思っていなかったので、音楽好きの親としては涙が出そうなくらいうれしいです。

さて、前回はWiresharkのディスプレイフィルタの基本的な使い方を説明したので、今回はディスプレイフィルタの特訓としてキャプチャファイルからポートスキャンの結果を推測してみたいと思います。

環境説明

本稿を書く際に使用した環境はWindows 8.1、Wireshark 1.12.5です。キャプチャファイルは筆者のブログからsd1511.pcapngをダウンロードしてください。

・Software Design 短期集中連載「Jamesのセキュリティレッスン」用キャプチャファイル
http://d.hatena.ne.jp/EijiYoshida/20140907/1410071296

あとは何かBGMでも流しましょう。筆者はライブの予習としてサカナクションの「ホーリーダンス」や「years」などを聴きながら本稿を書いています。そうそう、ついさっき“ホーリーダンス”が“放り出す”と掛かっていることに気づきました。

ポートスキャンとは

まずはポートスキャンについて説明しましょう。ポートスキャンとは**ポートにパケットを送信して、反応の違いからポートの開閉を調べる行為**のことです。たとえば、ここに192.0.2.3というIPアドレスが設定されたサーバがあるとします。さて、TCPポートは何番が開いているかわかりますか？　当然、IPアドレスを教えてもらっただけでは、TCPポートの何番が開いているかなんてわかるわけがありません。では、どうすればわかるのかというと、ここでポートスキャンの出番となります。192.0.2.3の1/tcpが開いているか閉じているか知りたければ、ポートスキャンに使えるパケットを1/tcpに送信して、その反応からポートの開閉を判断します。つぎに2/tcpが開いているか閉じているか知りたければ、同じようにパケットを送信して、反応からポートの開閉を判断します。これを繰り返すことで192.0.2.3で開いているTCPポートを調べ上げることができます。この行為がポートスキャンであり、自動化したものはポートスキャナと呼ばれます。

▼図1　192.0.2.3の開いている1/tcpにSYNパケットを送ったときの反応

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.0.2.1	192.0.2.3	TCP	60	1780→1 [SYN] Seq=0 Win=512 Len=0
2	0.006496000	192.0.2.3	192.0.2.1	TCP	60	1→1780 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
3	0.006560000	192.0.2.1	192.0.2.3	TCP	60	1780→1 [RST] Seq=1 Win=0 Len=0

▼図2　192.0.2.3の閉じている2/tcpにSYNパケットを送ったときの反応

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.0.2.1	192.0.2.3	TCP	60	2677→2 [SYN] Seq=0 Win=512 Len=0
2	0.000147000	192.0.2.3	192.0.2.1	TCP	60	2→2677 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

SYNポートスキャン

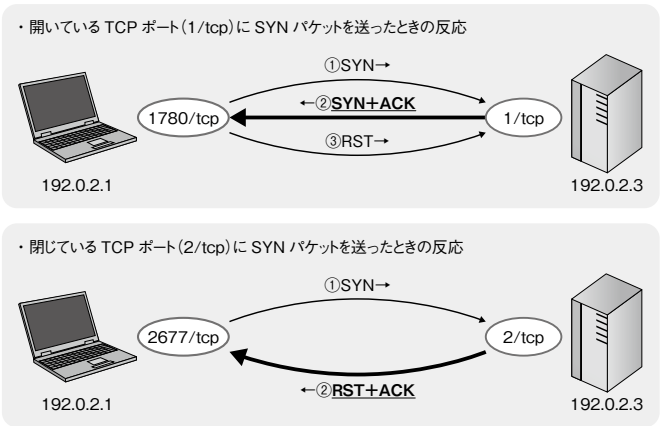
TCPポートの開閉を調べる基本的なポートスキャンとして、「SYNポートスキャン」があります。これは、開いているか閉じているかを知りたいポートにSYNパケットを送信して、反応の違いからポートの開閉を判断するというポートスキャンの手法です。実際に、192.0.2.3の開いている1/tcpにSYNパケットを送ったときの反応(**図1**)と、閉じている2/tcpにSYNパケットを送ったときの反応(**図2**)を見比べてみましょう。

図1について、No.1を見ると、ポートの開閉を調べるために192.0.2.1の1780/tcpから192.0.2.3の1/tcpにSYNパケットを送信しています。No.2を見ると、そのSYNパケットに反応して192.0.2.3の1/tcpから192.0.2.1の1780/tcpにSYN+ACKパケットを返信しており、No.3でRSTパケット(接続を中断する際に送られるパケット)を送っています。

一方、**図2**のNo.1でもポートの開閉を調べるために、192.0.2.1の2677/tcpから192.0.2.3の2/tcpにSYNパケットを送信していますが、次のNo.2を見ると192.0.2.3の2/tcpから192.0.2.1の2677/tcpにRST+ACKパケットを返信しています。つまり、開いているポートにSYNパケットを送信するとSYN+ACKパケットが返信されますが、閉じているポートにSYNパケットを送信するとRST+ACKパケットが返信されるのです(**図3**)。SYNポートスキャンは、この"SYNパケットに対する反応の違い"を利用してポートの開閉を判断しています。

いきなりですが、ここで問題です。192.0.2.4の80/tcpが開いているか閉じているか知りたいので、192.0.2.1からSYNパケットを送信しました。その際に送受信されたパケットが**図4**になります。192.0.2.4の80/tcpは開いているでしょうか？それとも閉じているでしょう

▼図3　"開いているポート"と"閉じているポート"のSYNパケットに対する反応の違い



▼図4　192.0.2.4の80/tcpにSYNパケットを送信したときの反応

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.0.2.1	192.0.2.4	TCP	60	1088→80 [SYN] Seq=0 Win=512 Len=0
2	0.000003000	192.0.2.4	192.0.2.1	TCP	60	80→1088 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
3	0.000005000	192.0.2.1	192.0.2.4	TCP	60	1088→80 [RST] Seq=1 Win=0 Len=0

か？　先ほどの解説をもとに考えてみてください。

◆　◆　◆

それでは答え合わせをしましょう。まず**図4**のNo.1を見ると、192.0.2.1の1088/tcpから192.0.2.4の80/tcpにSYNパケットを送信していて、次のNo.2を見ると、192.0.2.4の80/tcpから192.0.2.1の1088/tcpにSYN+ACKパケットを返信していることがわかります。つまり192.0.2.4の80/tcpはSYNパケットに対してSYN+ACKパケットを返信しています。あとは**図3**を見て、SYNパケットに対してSYN+ACKパケットを返信しているのは“開いているポート”なのか“閉じているポート”なのかを調べるだけです。結果は、開いているポートにSYNパケットを送信するとSYN+ACKパケットが返信されることから、**図4**の192.0.2.4の80/tcpは“開いている”ということになります。

ここまでのまとめとして、SYNポートスキャンの判定方法を**表1**にまとめてみました。最終行に“返信なし、またはICMPパケット”と書かれているのは、パケットフィルタリング(ファイアウォール)でパケットをDROPしている場合は返信がないことや、REJECTしている場合にICMPパケット(Type:3、Code:13など)が返信されることを利用して、“パケットフィルタリングで保護されている”と判断するためのものです。

▼表1　SYNポートスキャンの判定表

SYNパケットに対する反応	判定
SYN＋ACK	開いている
RST＋ACK	閉じている
返信なし、またはICMPパケット	パケットフィルタリングで保護されている

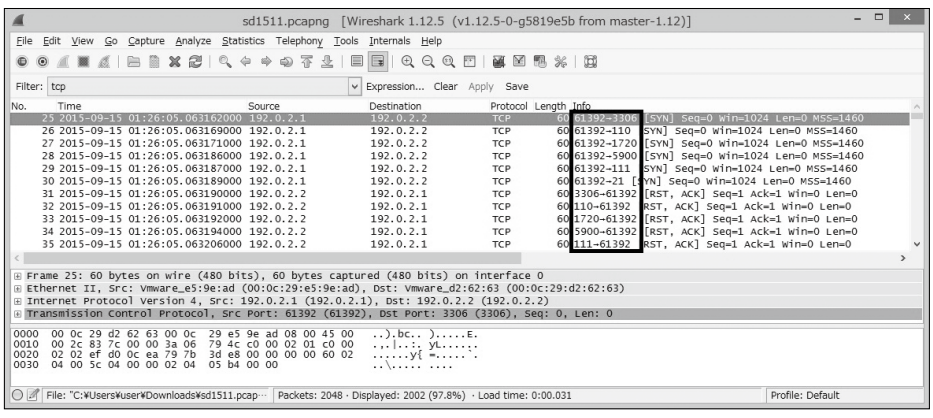
キャプチャファイルからSYNポートスキャンの結果を推測してみよう

筆者のブログからsd1511.pcapngをダウンロードしてWiresharkで開いたら、[View]メニューから[Time Display Format]を選んで、[Date and Time of Day]を選択してください。

次に、ディスプレイフィルタとして**tcp**を設定して、TCPパケットのみ表示してみましょう(**図5**)。もし[Packet List]ペインにあるInfo列が「61392→3306」という表示ではなく、「61392→mysql」という表示になっている場合は、[View]メニューから[Name Resolution]を選び、[Enable for Transport Layer]のチェックを外して、ポート番号で表示するように設定してください。ちなみに「61392 → 3306[SYN]」は、“61392/tcpから3306/tcpに送信されたSYNパケット”という意味になります。

表示されている内容を見ると、同じIPアドレス宛に送信された大量のSYNパケットが目立ちます。とくに[Packet List]ペインにあるInfo列に注目すると、

▼図5　ディスプレイフィルタでtcpを設定



SYNパケットの送信先ポート番号を3306/tcp、110/tcp、1720/tcpといったようにガチャガチャ切り替えていることがわかります。このように、同一IPアドレスの複数のポート宛に連続してパケットを送信している場合は、ポートスキャンの可能性が高いでしょう。192.0.2.1から192.0.2.2の複数のポート宛にSYNパケットが連続して送信されていることから、192.0.2.1から192.0.2.2に対してSYNポートスキャンが行われたと考えられます。

それではディスプレイフィルタを使ってSYNポートスキャンの結果を推測してみましょう。ディスプレイフィルタについては本連載第5回(2015年10月号)を参照してください。

● ポートスキャンの送信元と送信先IPアドレスで送受信されたパケットのみ表示

192.0.2.1から192.0.2.2に対してSYNポートスキャンが行われたと考えられるので、まずは192.0.2.1と192.0.2.2で送受信されたパケットのみ表示させましょう。ディスプレイフィルタは次のとおりです。

```
ip.addr == 192.0.2.1 &&
ip.addr == 192.0.2.2 &&
tcp
```

● ポートスキャンされている時間のパケットに絞り込む

表示されている内容を見ると、SYNポートスキャンは2015年9月15日の01:26:05.063162から同日01:26:05.108062まで行われたと考えられるので、その時間のパケットに表示を絞り込みます。ディスプレイフィルタは次のとおりです。

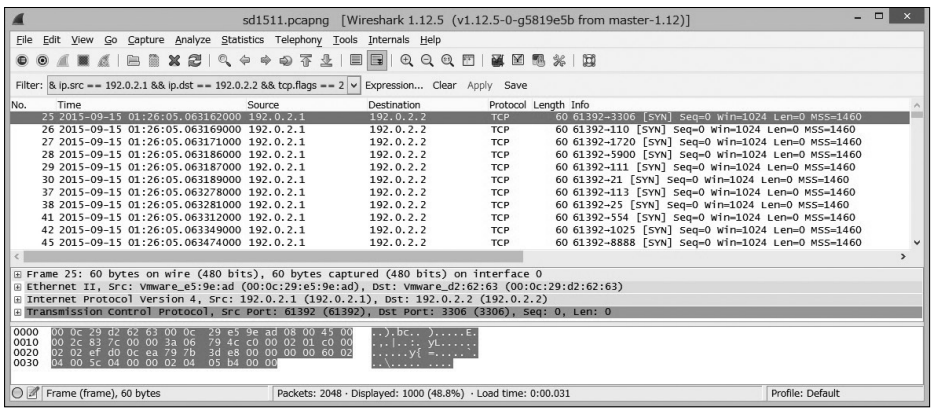
```
frame.time >= "2015-09-15 01:26:05.063162" &&
frame.time <= "2015-09-15 01:26:05.108062" &&
ip.addr == 192.0.2.1 &&
ip.addr == 192.0.2.2 &&
tcp
```

● スキャン対象のポート番号を調べる

ここまでのディスプレイフィルタでSYNポートスキャンのパケットだけが表示されているので、次はスキャン対象となったポート番号を調べてみましょう。SYNポートスキャンでは、スキャン対象のポート番号宛にSYNパケットが送信されるので、ディスプレイフィルタを使ってSYNパケットのみ表示させます(**図6**)。ディスプレイフィルタは次のとおりです。

```
frame.time >= "2015-09-15 01:26:05.063162" &&
frame.time <= "2015-09-15 01:26:05.108062" &&
ip.src == 192.0.2.1 &&
ip.dst == 192.0.2.2 &&
tcp.flags == 2
```

▼図6　SYNポートスキャンで送信されたSYNパケット



先ほどのディスプレイフィルタに似ていますが、**ip.addr**が**ip.src**や**ip.dst**に変更されている点と、**tcp**が**tcp.flags == 2**に変更されている点に注意してください。

あとは[Packet List]ペインのInfo列を見るとSYNパケットが3306/tcp、110/tcp、1720/tcp、5900/tcpなどに送信されていることから、これらがSYNポートスキャンのスキャン対象になったポート番号であることがわかります。

● 開いていると判断されたポート番号を調べる

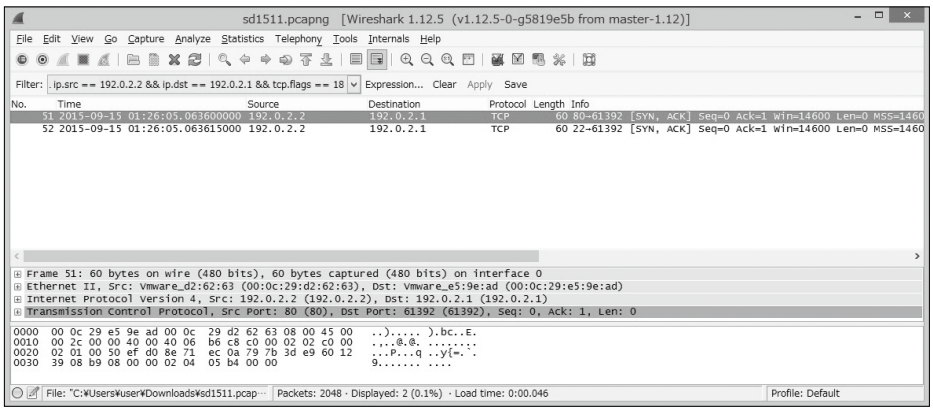
SYNポートスキャンでは、SYNパケットに対してSYN+ACKパケットを返信したポートを“開いているポート”と判断することから、スキャン対象にこのような動作をするポート番号があれば、ポートスキャナはそのポートを開いていると判断するはずです。さっそくディスプレイフィルタを使って192.0.2.2から192.0.2.1に返信されたSYN+ACKパケットのみ表示させましょう(**図7**)。ディスプレイフィルタは次のとおりです。

```
frame.time >= "2015-09-15 01:26:05.063162" &&
frame.time <= "2015-09-15 01:26:05.108062" &&
ip.src == 192.0.2.2 &&
ip.dst == 192.0.2.1 &&
tcp.flags == 18
```

図6のディスプレイフィルタに似ていますが、IPアドレスとtcp.flagsの値が違うので注意してください。

結果として、80/tcpと22/tcpからSYN+ACKパケットが返信されていることから、SYNポートスキャンでは80/tcpと22/tcpが開いていると判断されたことが推測されます。

▼図7　SYNポートスキャンで返信されたSYN+ACKパケット



▼図8　SYNポートスキャンを行ったポートスキャナの結果

```
# nmap -n -Pn -sS 192.0.2.2

Starting Nmap 6.47 (http://nmap.org) at 2015-09-14 16:26 UTC
Nmap scan report for 192.0.2.2
Host is up  (0.00013s latency).
Not shown:  998 closed ports
PORT    STATE SERVICE
22/tcp  open    ssh
80/tcp  open    http
MAC Address:  00:0C:29:D2:62:63 (VMware)

Nmap done:  1 IP address (1 host up) scanned in 0.09 seconds
#
```

答え合わせ

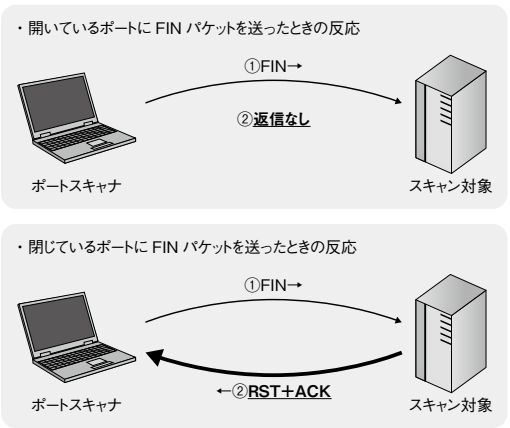
筆者はこのSYNポートスキャンを行ったポートスキャナの結果を持っているので、ポートスキャナが80/tcpと22/tcpを開いていると判断したのかどうか確認してみましょう(**図8**)。これを見ると、22/tcpと80/tcpのSTATEがopenと表示されていることから、キャプチャファイルの内容から推測したように、ポートスキャナは80/tcpと22/tcpを開いていると判断したことがわかります。

FINポートスキャンでも結果を推測できる

今回解説した技術は、SYNポートスキャンだけではなく、ほかのポートスキャンでも結果を推測できます。試しに今度はFINポートスキャンの結果を推測してみましょう。FINポートスキャンの判定表は**表2**で、FINポートスキャンで送受信されたパケットは**図9**です。

LinuxなどのOSでは、開いているポートにFINパケットを送信しても何も返信しませんが、閉じているポートにFINパケットを送信すると、RST+ACKパケットを返信します(**図10**)。この、開いているときと閉じているときの反応の違いを利用してポートの開閉を判断するのがFINポートスキャンです。

▼図10　"開いているポート"と"閉じているポート"のFINパケットに対する反応の違い



このようなFINポートスキャンのしくみを念頭に置いて**図9**を見ると、192.0.2.1から192.0.2.2にFINパケットが複数送信されていて、さらに192.0.2.2から192.0.2.1にもRST+ACKパケットが複数返信されていることから、確かにFINポートスキャンであることがわかります。FINパケットの送信先ポート番号を[Packet List]ペインのInfo列で調べると10003/tcp、10002/tcp、10004/tcp、10005/tcp、10001/tcpとなっているので、これらがスキャン対象のポートになります。

あとはFINポートスキャンの判定表に基づいて開いているポートを探すだけですが、SYNポートスキャンに比べてFINポートスキャンは開いているポートを探すのがたいへんだったりします。SYNポートスキャンではSYN+ACKパケットが見つかれば送信元のポート番号は開いていると判断できますが、FINポートスキャンでは開いているポートからパケットは返信されないので、"開いている場合に返信されるパ

▼表2　FINポートスキャンの判定表

FINパケットに対する反応	判定
返信なし	開いている、またはパケットフィルタリングで保護されている
RST＋ACK	閉じている
ICMPパケット	パケットフィルタリングで保護されている

▼図9　FINポートスキャンで送受信されたパケット

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.0.2.1	192.0.2.2	TCP	60	38480→10003 [FIN] Seq=1 Win=1024 Len=0
2	0.000017000	192.0.2.1	192.0.2.2	TCP	60	38480→10002 [FIN] Seq=1 Win=1024 Len=0
3	0.000019000	192.0.2.1	192.0.2.2	TCP	60	38480→10004 [FIN] Seq=1 Win=1024 Len=0
4	0.000069000	192.0.2.1	192.0.2.2	TCP	60	38480→10005 [FIN] Seq=1 Win=1024 Len=0
5	0.000071000	192.0.2.1	192.0.2.2	TCP	60	38480→10001 [FIN] Seq=1 Win=1024 Len=0
6	0.000257000	192.0.2.2	192.0.2.1	TCP	60	10002→38480 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
7	0.000260000	192.0.2.2	192.0.2.1	TCP	60	10004→38480 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
8	0.000269000	192.0.2.2	192.0.2.1	TCP	60	10005→38480 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
9	0.000270000	192.0.2.2	192.0.2.1	TCP	60	10001→38480 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
10	1.101771000	192.0.2.1	192.0.2.2	TCP	60	38481→10003 [FIN] Seq=1 Win=1024 Len=0

ケットを探す”という方法は使えないのです。

ではどうすれば良いかというと、スキャン対象のポート番号のうちRST+ACKパケットやICMPパケットが返信されたポートを消去していって、最終的に残ったポート番号を“開いている、またはパケットフィルタリングで保護されている”と判断する方法があります。スキャン対象のポート番号である10003/tcp、10002/tcp、10004/tcp、10005/tcp、10001/tcpのうち、RST+ACKパケットを返信したポート番号は10002/tcp、10004/tcp、10005/tcp、10001/tcpですので消去していくと、最終的に10003/tcpだけが残ったと思います。このポートがFINポートスキャンでは“開いている、またはパケットフィルタリングで保護されている”と判断されます。

答え合わせとして、FINポートスキャンを行ったポートスキャナの結果である**図11**を見ると、10003/tcp の STATE が、**open|filtered** となっていることから、ポートスキャナはFINポートスキャンで10003/tcpを“開いている、またはパケットフィルタリングで保護されている”と判断していることがわかります。

このようにFINポートスキャンでも、判定表をもとに結果を推測できました。そもそもポートスキャナは返信されたパケットをWiresharkのようにキャプチャしてポートの開閉を判断しているので、ポートスキャンの判定方法さえわかっていればポートスキャナと同じことができるわけです。

おわりに

今回は、Wiresharkのディスプレイフィルタを駆使してキャプチャファイルからポートスキャンの結果を推測する技術について解説しました。この技術を習得しておくと、ポートスキャンで送受信されたパケットさえ残っていれば、攻撃された当時のサーバが攻撃者にはどのように見えていたのか知ることができます。実際、パケット解析をしていると「なんで攻撃者はこっちのサービスには攻撃しなかったんだろう？」と思ったりしますが、ポートスキャンで送受信されたパケットからポートスキャンの結果を推測すると「なるほど、そのサービスで使っているポートがパケットフィルタリングで保護されていて攻撃できなかったのか」といったことがわかるので、攻撃者の行動を理解するのに役立ったりします。またポートスキャンと同様の技術であるOS推測についても、OS推測に使われる判定表があれば、今回解説した技術を応用して同じようなことができたりします。

本稿の内容が少しでもみなさんのお役に立てば幸いです。SD

▼図11　FINポートスキャンを行ったポートスキャナの結果

```
# nmap -n -Pn -sS 192.0.2.2 -p  10001-10005

Starting Nmap 6.47 (http://nmap.org) at 2015-09-15 08:26 UTC
Nmap scan report for 192.0.2.2
Host is up  (0.00025s latency).
PORT         STATE            SERVICE
10001/tcp  closed           scp-config
10002/tcp  closed           documentum
10003/tcp  open|filtered  documentum_s
10004/tcp  closed           emcrmirccd
10005/tcp  closed           stel
MAC Address:  00:0C:29:D2:62:63 (VMware)

Nmap done:  1 IP address (1 host up) scanned in 1.27 seconds
#
```

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8回 プロセス間状態通知

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の持つリンク、モニタ、そしてスーパーバイザといったプロセス間状態通知とそれを利用した耐障害性の確保について紹介します。

リンクによる双方向状態通知

システムとしての耐障害性を実現するためには、個々のプログラムや入出力デバイスなどで発生した予測し得ない事態を迅速に遅滞なく通知するしくみが必要です[注1]。Erlangではこの基本的な機能を「リンク」という双方向の通知機能で実現しています[注2]。

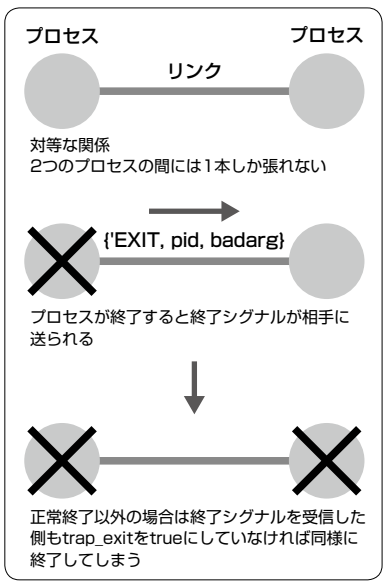
リンクは2つのプロセス間の間で1本だけ張ることができます。通常はプロセスを起動する際の組込み関数(BIF)spawn/{1,2,3,4}と不可分(アトミック)な動作としてリンクを張るspawn_link/{1,2,3,4}を使います[注3]。一度張ったリンクはBIFのunlink/1で解消できま

注1) UNIX系OSではこれを「シグナル」というしくみで実現しています。また多くのプログラミング言語では「例外」、そしてハードウェアでは「割り込み」という形で同様の処理を実現しています。

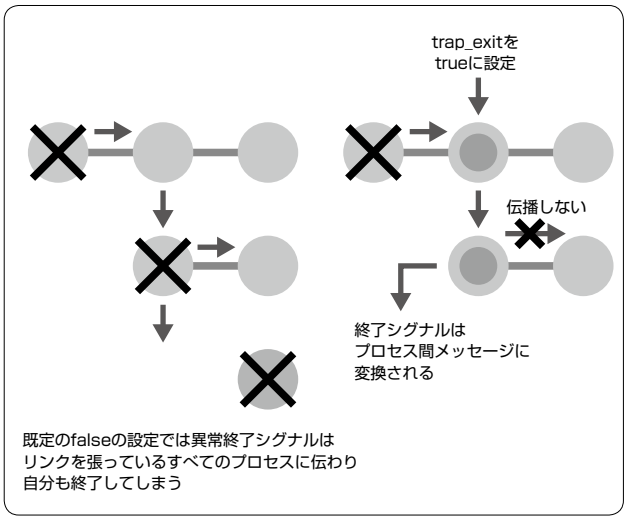
注2) リンクはErlangの仮想マシンBEAM外との通信チャネルの1つである「ポート」に対しても張ることができますが、今回はポートとのリンクについては割愛します。

注3) 個別にBIFのlink/1として張ることもできますが、spawnによるプロセス起動とlinkを実行する間に張ろうとした相手のプロセスが終了してしまうという動作を避けるために、通常はspawn_link/{1,2,3,4}を使うことが推奨されています。

▼図1 リンクの基本動作



▼図2 trap_exitフラグの設定による異常終了時の終了シグナルの扱いの違い



▼リスト1　リンク説明用の各種関数をまとめたlink_exampleモジュール

```
リンクに関する各種関数の紹介
-module(link_example).
-export([wait_spawn_link/2,
         wait_spawn_link/1,
         trap_link_msg/0,
         gen_prob_true/1,
         spawn_prob_link/1]).
この関数ではNミリ秒待ったあとに理由を変数Statusとして終了するプロセスを起動してリンクを張る
-spec wait_spawn_link(pos_integer(), term()) -> pid().
wait_spawn_link(N, Status) when N >= 1 ->
    spawn_link(fun() ->
                       timer:sleep(N),
                       exit(Status)
               end).
この関数は上記関数の正常終了版
-spec wait_spawn_link(pos_integer()) -> pid().
wait_spawn_link(N) ->
    wait_spawn_link(N, normal).
この関数ではtrap_exitをtrueとしたプロセスを起動してリンクを張る。この関数はメッセージを1つ受信したら、その旨標準出力に出す。
こうすることでリンクの先のプロセスにどのような終了シグナルが伝わっているかを確認できる
-spec trap_link_msg() -> pid().
trap_link_msg() ->
    spawn_link(
      fun() ->
              process_flag(trap_exit, true),
              receive X ->
                      io:format("Process ~p received: ~p~n",
                                [self(), X])
              end
      end).
この関数はtrueとfalseをランダムに選んでN個の要素を持つリストにする（出現確率は1/2に近いがrand:uniform/1で決めているだけなので、
N個の中で厳密には決めていない）
-spec gen_prob_true(pos_integer()) -> [true | false].
gen_prob_true(N) when N >= 1 ->
    [rand:uniform(2) =:= 1 || _ <- lists:seq(1, N)].
この関数では前述のランダムにtrue/falseを出すリストを使って、N個のプロセスを起動する。これらのプロセスは無期限に待つだけで何もしない。
そしてtrueになっているものにはリンクを張り、falseであるものにはリンクを張らないという動作をする。
関数の戻り値{P, L}のうちPはpidのリスト、Lは対応するtrue/falseの別のリストである
-spec spawn_prob_link(pos_integer()) -> {[pid()], [true | false]}.
spawn_prob_link(N) ->
    L = gen_prob_true(N),
    P = [case C of
             true ->
                 spawn_link(fun() -> timer:sleep(infinity) end);
             false ->
                 spawn(fun() -> timer:sleep(infinity) end)
         end || C <- L],
    {P, L}.
```

す。リンクは張ったどちらかのプロセスの終了とともに消滅します。リンク上を伝わるメッセージである「終了シグナル」は、{'EXIT', プロセスID, 理由を示す項}というフォーマットで送られます。理由の部分がnormalというアトムであれば、正常終了とみなされます。そうでなければ異常終了となり、受信したプロセスは終了します（**図1**）。複数のプロセス間でリンクを張ることで、終了シグナルを相互にやりとりするプロセスのグループを作ることもできます。

リンク上の終了シグナルを受け取ったプロセスには2つの動作の可能性があります。既定の動作としては、正常終了以外の場合、自らも終了してほかのリンクを張ったプロセスに終了シグナルを転送します。一方、もう1つの動作としては、あらかじめprocess_flag(trap_exit, true)というBIFを実行することで、終了シグナルをプロセス間メッセージに変換して受信し、ほかのリンクには終了シグナルを転

送しないという動作も選択できます[注4](**図2**)。

Erlangのシェルでリンクを扱う際は終了シグナルの扱いに注意が必要です。インタラクティブ入力で発生した各種エラーでは、シェルのプロセスが一度終了して再起動され別のプロセスになるだけでなく、リンクしているプロセスにも同じ終了シグナルが伝わってしまいます。これを防ぐにはリンクを張らないか、unlinkで切るという作業が必要です。

リスト1に、リンクの使用例をまとめています。**図3**にErlangシェルがリンクによってどのような影響を受けるかを示しています。また、リンクを張っているかいないかによっての違いを**図4**に示しました。

注4) このtrap_exitをtrueに設定した場合、プロセス間メッセージとして異常通知を捕捉し個別に例外処理ができるのは大きな利点ですが、その一方で外部から送られた終了シグナルではプロセスを強制停止できなくなります。

▼図3 リンクによる異常終了時の終了シグナルの伝搬とErlangシェルの挙動

```
Eshell V7.0.3  (abort with ^G)
BIFのself/0でシェル自身のpidを表示する
1> self().
<0.33.0>
リンクを張ったプロセスを起動する
2> link_example:wait_spawn_link(1000, normal).
<0.36.0>
normalは正常終了を意味するので、1秒経過しても何も起きない。
self/0を見るとシェルのpidも変わらない
3> self().
<0.33.0>
abendという理由で異常終了を発生させる
4> link_example:wait_spawn_link(1000, abend).
<0.39.0>
1秒後に次の表示が出る
** exception error: abend
シェルのpidも変わって再起動されているのがわかる
5> self().
<0.41.0>
trap_exitをtrueに設定する
6> process_flag(trap_exit, true).
この関数の仕様で設定前の値が戻ってくる
false
もう一度確かめる
7> process_info(self(), trap_exit).
確かにtrueになっている
{trap_exit,true}
異常終了を発生させてみる
8> link_example:wait_spawn_link(1000, abend).
<0.148.0>
この場合何も表示はされない。プロセス間メッセージをはき出すコマンドを実行すると終了シグナルがメッセージとして蓄積されていたのがわかる
9> flush().
Shell got {'EXIT',<0.148.0>,abend}
ok
```

リンクに関する詳しい説明はBIFのマニュアル[1]、ならびにプロセスの扱いに関するマニュアル[2]を参照してください。

モニタによる一方的ダウン監視

リンクは便利なしくみですが、2つのプロセス間に1つしか張ることができないのと、一度張ったら自分の状態も相手に伝わるため、単純な死活監視用には適しません。リンクに代わって、プロセスが「ダウン」したかどうかの監視機能を提供するのが「モニタ」です。

モニタではダウンの監視をしたい相手先を指定して接続します。具体的にはmonitor(process, プロセス識別名[注5])という関数で接続ができます。2つのプロセスの間で複数個接続することもでき、各モニタの接続には個別のリファレンス[注6]が付きます。spawn_link同様、spawn_monitorというBIFも用意されています。また、モニタ接続を解除するためのdemonitor/{1,2}というBIFもあります。

モニタの動作としては、モニタ接続で終了監視されているプロセスが終了したか、存在しなくなったか、分散ノード間の接続が切れたかというダウン監視の条件が満たされた場合に、{'DOWN', モニタのリファレンス, process, 発信元のプロセス識別名, 理由を示す項}というプロセス間メッセージがモニタの接続元であるプロセスに送られます(**図5**)。これらのメッセージにはとくに高い優先度は設定されないため、終了シグナルよりは伝わるのは遅くなる可能性があります。

モニタを使ったダウン監視の例を**図6**に示しました。プロセス間メッセージが受信できていることがわかります。

注5) ここでいう「プロセス識別子」とは、名前を持たないプロセスではプロセスID(pid)、名前を持つ登録済みプロセス(本誌2015年7月号の連載第4回を参照)ではその名前を指します。

注6) リファレンスはBIFのmake_ref/0で作られる参照子で、重複する確率は非常に低いため、複数のモニタの接続を区別するのに実用上問題なく使えます。

モニタはリンクと同様Erlang/OTPのマニュアルではBIFとプロセス関連の事項として説明されています。また登録済みプロセスとの関連については、Fred Hébertによる解説[3]が参考になります。

スーパーバイザによる死活監視と再起動、そしてアプリケーションの統合管理

プロセスは予期しない理由で異常終了することがあります[注7]。その意味で死活監視や再起動を自動で行うしくみは運用システムでは不可欠[注8]です。Erlang/OTPではリンクやモニタといったしくみを使って状態通知を行うことができますが、これを個別にコーディングするのは

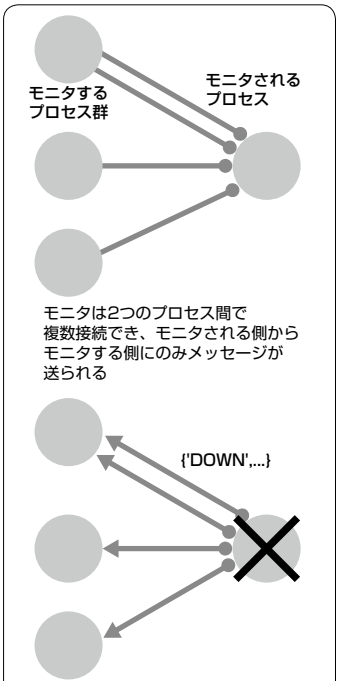
注7) 100%完璧でバグのないソフトウェアであったとしても、ハードウェアの故障、あるいは人間の操作ミスからは逃れることはできません。

注8) UNIX系OSでは、アプリケーションの死活監視と再起動の自動化や簡素化を行う運用ツールとして、daemontools（http://cr.yp.to/daemontools.html）が広く使われています。daemontoolsの設計思想はOTPのスーパバイザによく似ているように筆者は思います。

▼図4　シェルの実行エラーによる終了シグナルがspawnしたプロセスにリンクを通じて伝搬する例

```
Eshell V7.0.3  (abort with ^G)
10個プロセスをspawnして起動する。trueとなっているものにはリンクが張られfalseとなっているものにはリンクは張られない
1> {P, L} = link_example:spawn_prob_link(10).
{[<0.35.0>,<0.36.0>,<0.37.0>,<0.38.0>,<0.39.0>,<0.40.0>,
  <0.41.0>,<0.42.0>,<0.43.0>,<0.44.0>],
 [false,false,false,false,true,false,false,false,true,true]}
シェルで実行エラーを起こす。この実行エラーはspawnされた各プロセスにも伝搬する
2> 1/0.
** exception error: an error occurred when evaluating an arithmetic expression
     in operator  '/'/2
        called as 1 / 0
プロセスの状態を見るとリンクを張ったものは皆存在しない（終了している）。リンクを張っていないものは終了せず残っている
3> lists:zip3(P, L, [erlang:process_info(Pid, status) || Pid <- P]).
[{<0.35.0>,false,{status,waiting}},
 {<0.36.0>,false,{status,waiting}},
 {<0.37.0>,false,{status,waiting}},
 {<0.38.0>,false,{status,waiting}},
 {<0.39.0>,true,undefined},
 {<0.40.0>,false,{status,waiting}},
 {<0.41.0>,false,{status,waiting}},
 {<0.42.0>,false,{status,waiting}},
 {<0.43.0>,true,undefined},
 {<0.44.0>,true,undefined}]
```

▼図5　モニタの基本動作



▼図6　モニタを使ったダウン監視とメッセージの例

```
モニタの実行例を示す。10秒待ってabendという理由で異常終了するプロセスを10個作り、それぞれをモニタするようにした。戻ってくるのはPidとモニタのリファレンスの組のタプルである
1> [spawn_monitor(fun() ->
        timer:sleep(10000),
        exit(abend) end) ||
    _ <- lists:seq(1,10)].
[{<0.56.0>,#Ref<0.0.8.186>},
 {<0.57.0>,#Ref<0.0.8.187>},
 {<0.58.0>,#Ref<0.0.8.188>},
 {<0.59.0>,#Ref<0.0.8.189>},
 {<0.60.0>,#Ref<0.0.8.190>},
 {<0.61.0>,#Ref<0.0.8.191>},
 {<0.62.0>,#Ref<0.0.8.192>},
 {<0.63.0>,#Ref<0.0.8.193>},
 {<0.64.0>,#Ref<0.0.8.194>},
 {<0.65.0>,#Ref<0.0.8.195>}]

10秒経過してもとくにシェルには表示されないが、たまっているプロセス間メッセージを表示するシェルコマンド c:flush/0 を実行すると次のとおり、メッセージが来ていることがわかる。順番が揃っていないことに注意してほしい

2> flush().
Shell got {'DOWN',#Ref<0.0.8.186>,process,<0.56.0>,abend}
Shell got {'DOWN',#Ref<0.0.8.195>,process,<0.65.0>,abend}
Shell got {'DOWN',#Ref<0.0.8.187>,process,<0.57.0>,abend}
Shell got {'DOWN',#Ref<0.0.8.188>,process,<0.58.0>,abend}
Shell got {'DOWN',#Ref<0.0.8.190>,process,<0.60.0>,abend}
Shell got {'DOWN',#Ref<0.0.8.191>,process,<0.61.0>,abend}
Shell got {'DOWN',#Ref<0.0.8.192>,process,<0.62.0>,abend}
Shell got {'DOWN',#Ref<0.0.8.193>,process,<0.63.0>,abend}
Shell got {'DOWN',#Ref<0.0.8.194>,process,<0.64.0>,abend}
Shell got {'DOWN',#Ref<0.0.8.189>,process,<0.59.0>,abend}
ok
```

▼リスト2　スーパーバイザ用msgcounter_supモジュール

```
スーパーバイザの起動用モジュール
-module(msgcounter_sup).
-behaviour(supervisor).
-export([start/0,
         start_from_shell/0,
         start_link/0,
         start_link/1,
         init/1]).
?MODULEは自分のモジュール名(ここではmsgcounter_sup)。ローカルノードでスーパーバイザを
別プロセスとしてspawnして起動する
-spec start() -> pid().
start() ->
    spawn(fun() ->
              supervisor:start_link(
              {local, ?MODULE}, ?MODULE, [])
          end).
シェルから起動するときのためにリンクを切っておく
-spec start_from_shell() -> true.
start_from_shell() ->
    {ok, Pid} = supervisor:start_link(
                  {local, ?MODULE}, ?MODULE, []),
    unlink(Pid).
関数supervisor:start_link/3への橋渡しをする
-spec start_link(list()) -> {ok, pid()}.
start_link(Args) ->
     supervisor:start_link(
         {local, ?MODULE}, ?MODULE, Args).
上記関数で引数がない場合の定義をする
-spec start_link() -> {ok, pid()}.
start_link() ->
     start_link([]).
スーパーバイザを初期化する
-spec init([]) -> {ok, {term(), term()}}.
init([]) ->
    ここではスーパーバイザ配下のプロセスが異常終了したときにどのような動作をするかを指定する
    SupFlags =
        #{strategy => one_for_one,
          intensity => 3,
          period => 10},
    ここでは配下のプロセスをどのように起動するかを指定する。gen_serverなどであればそのまま起動
    できる。このスーパーバイザではmsgcounter_gen_serverのプロセスを4つ起動するように設定する
    ChildSpecs = [
        #{id => counter1,
          start => {msgcounter_gen_server, start_link, []},
          shutdown => brutal_kill},
        #{id => counter2,
          start => {msgcounter_gen_server, start_link, []},
          shutdown => brutal_kill},
        #{id => counter3,
          start => {msgcounter_gen_server, start_link, []},
          shutdown => brutal_kill},
        #{id => counter4,
          start => {msgcounter_gen_server, start_link, []},
          shutdown => brutal_kill}
        ],
    % 返り値として初期設定を返します
    {ok, {SupFlags, ChildSpecs}}.
```

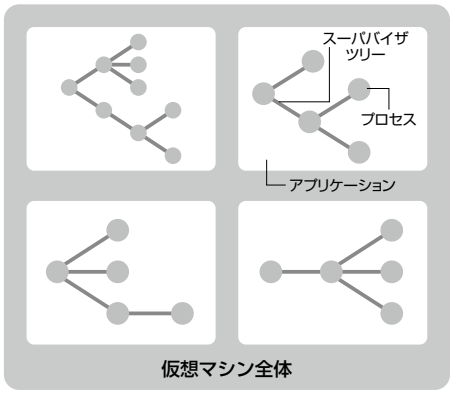
容易なことではありません。

OTPには、あらかじめ死活監視をして、異常終了したプロセスを再起動するためのビヘイビア(本誌2015年8月号の連載第5回を参照)であるスーパーバイザ(supervisor)が用意されています。スーパーバイザからはほかのスーパーバイザ、あるいはgen_serverなど標準のビヘイビアに従ったプロセス(ワーカ)を起動できます。これによって階層的な死活監視や部分再起動を行うことができます。

スーパーバイザは単独でも起動できますが、その真価は上位概念である「アプリケーション」として起動したときに発揮されます。Erlang/OTPのアプリケーションは複数のモジュールの集まりであり、一括して複数のモジュールを扱うことができます。アプリケーションは中にスーパーバイザを持つことができ、これによって階層的な死活監視を行えます。また、アプリケーションは起動時にBEAMの持つアプリケーションコントローラに登録され、プロセス間の起動に関する階層構造や内部状態を把握できるため、デバッグが容易になります(**図7**)。

アプリケーションを作るためには、アプリケーションの起動と終了を制御するためのモジュール、そしてアプリケーションがどんなモジュールを含むかなどの情報を記した設定ファイル

▼図7　アプリケーションとスーパーバイザ



仮想マシンの中では複数のアプリケーションが動作しており、各アプリケーションの中ではスーパバイザツリーによる階層的死活監視が行われている。
各アプリケーションが終了した場合、仮想マシン全体を止めるかそうでないかについては起動時に選択できる(既定値では他のアプリケーションには影響しない)。

▼リスト3　アプリケーション用msgcounter_appモジュール

```
msgcounter_supモジュールをスーパバイザとするアプリケーションのためのモジュール。これをapplicationモジュールからロードすることでErlangのアプリケーションとして認識される
-module(msgcounter_app).
-behaviour(application).
-export([start/2, stop/1]).

start(_Type, _Args) ->
    msgcounter_sup:start_link().

stop(_State) ->
    ok.
```

▼リスト4　msgcounter_appアプリケーション用設定ファイルmsgcounter_app.app.src

```
アプリケーションmsgcounter_appの設定ファイル。このファイルはrebar用のソースとしてmsgcounter_app.app.src という名前にしているが、実際の設定ファイルは msgcounter_app.app という名前になる。設定全体は1つのタプルとしてまとまっている
{application, msgcounter_app,
 [
  アプリケーションの説明
  {description, "Message counter application"},
  バージョン番号
  {vsn, "1"},
  アプリケーションに必要なモジュール名を列挙する
  {modules, [msgcounter_app,
             msgcounter_sup,
             msgcounter_gen_server]},
  登録済みプロセスの名前を列挙する
  {registered, [msgcounter_sup]},
  動作に必要な他のアプリケーションの名前を列挙する。
  OTP標準アプリケーションのkernelとstdlibは最低限必要
  {applications, [kernel, stdlib]},
  コールバックするモジュール名を列挙する
  {mod, {msgcounter_app,[]}}
 ]}.
```

▼図8　msgcounter_appアプリケーションとmsgcounter_supスーパーバイザによる実行例

```
msgcounter_appの実行例
Eshell V7.0.3  (abort with ^G)
アプリケーションをロードして起動する
1> application:load(msgcounter_app).
ok
2> application:start(msgcounter_app).
ok
登録済みプロセスの名前を見ると、スーパーバイザのmsgcounter_supが登録されていることがわかる
3> registered().
[erl_prim_loader,inet_db,msgcounter_sup,code_server,
 standard_error_sup,application_controller,rex,user_drv,
 error_logger,standard_error,user,init,kernel_sup,
 global_name_server,global_group,file_server_2,
 kernel_safe_sup]
スーパーバイザがどのプロセスを管理しているかを表示する
4> supervisor:which_children(msgcounter_sup).
[{counter4,<0.43.0>,worker,[msgcounter_gen_server]},
 {counter3,<0.42.0>,worker,[msgcounter_gen_server]},
 {counter2,<0.41.0>,worker,[msgcounter_gen_server]},
 {counter1,<0.40.0>,worker,[msgcounter_gen_server]}]
管理している各プロセスがどのように起動されたかもわかる
5> supervisor:get_childspec(msgcounter_sup, counter1).
{ok,#{id => counter1,
      modules => [msgcounter_gen_server],
      restart => permanent,
      shutdown => brutal_kill,
      start => {msgcounter_gen_server,start_link,[]},
      type => worker}}
counter1 (Pid <0.40.0>) のカウンタを操作してみる
6> msgcounter_gen_server:inc(pid(0,40,0)).
1
正常に動作しているのがわかる
7> msgcounter_gen_server:inc(pid(0,40,0)).
2
このcounter1に相当するプロセスを強制終了する
8> exit(pid(0,40,0),kill).
true
スーパバイザの管理プロセスを見るとcounter1に相当するプロセスが再起動されているのがわかる
9> supervisor:which_children(msgcounter_sup).
[{counter4,<0.43.0>,worker,[msgcounter_gen_server]},
 {counter3,<0.42.0>,worker,[msgcounter_gen_server]},
 {counter2,<0.41.0>,worker,[msgcounter_gen_server]},
 {counter1,<0.55.0>,worker,[msgcounter_gen_server]}]
当然ながら再起動をしたが過去のプロセス同様カウンタの状態は失われている
10> msgcounter_gen_server:val(pid(0,55,0)).
0
```

(.appファイル)が必要です。

一例として、**リスト2**に、本連載で使用してきたgen_serverビヘイビアを使ったカウンタ(連載第5回を参照)をスーパーバイザの下でプロセスとして起動するためのモジュールを示しました[注9]。また、**リスト3**では、msgcounter_supモジュールをアプリケーションとするのに必要な最低限の関数だけを含めたアプリケーション用モジュールを示しています。このアプリケーションの設定ファイルは**リスト4**に示しています。

▼図9 observerによるアプリケーション実行表示の様子

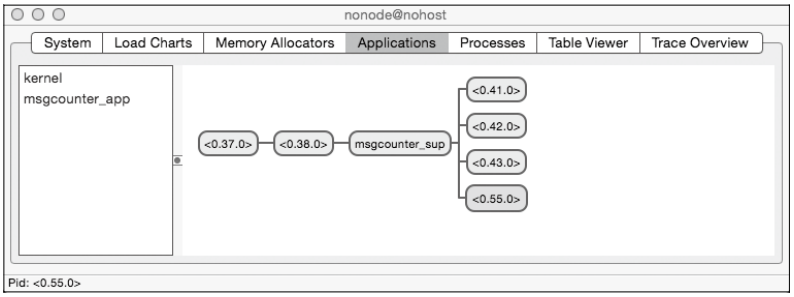


図8に、msgcounter_appアプリケーションの実行例を示しました。管理下にある各カウンタのプロセスを強制終了しても、新しいプロセスが自動的に立ち上がってくるのがわかります。アプリケーションの様子はobserverを通じて見ることもできます(**図9**)。

ここで注意しなければならないのは、スーパーバイザは何らかの原因で終了したワーカプロセスを再起動はしてくれますが、終了したワーカプロセスの中の状態までは再現してくれません。状態を回復したい場合は、何らかの方法でワーカプロセスの外部に状態を記録しておく必要があります[注10]。

スーパーバイザの挙動についてはOTPのDesign Principlesに説明があります[4]。管理用のsupervisorモジュールの各関数の説明[5]も参照してください。また、アプリケーションについての同様の説明があります[6][7]。アプリケーションの設定ファイルについては.appの説明が別途あります[8]。

まとめ

今回はErlangのプロセス間状態通知の手法について紹介しました。次回はErlangの持つプロセス辞書、ならびにOTPの持つETSやDETS、Mnesiaデータベースといったプロセス間情報共有の手法について紹介します。

ソースコードとサポートページ

連載の記事で紹介したソースコードなどGitHubのレポジトリに置いています(https://github.com/jj1bdx/sd-erlang-public/)。ぜひご活用ください。SD

注9) コールバック関数のmsgcounter_sup:init/1では、18.0からの推奨設定としてマップによる初期値を与えています。17.x以前のバージョンでは、別途タプルとして与える必要があります。詳しくは参考文献[5]を参照してください。

注10) 一般的なクライアント=サーバモデルによるシステム設計の場合、サーバ側に状態を持たせるのか、クライアント側に状態を持たせるのかは難しい問題です。Erlangでのワーカプロセスはサーバとして使われることが多いということを考えると、サーバ側で持つ状態を極力少なくするのが、耐障害性を上げる1つのコツといえるでしょう。

参考文献

[1] http://www.erlang.org/doc/man/erlang.html

[2] http://www.erlang.org/doc/reference_manual/processes.html

[3] http://learnyousomeerlang.com/errors-and-processes(日本語訳はFred Hébert, 山口能迪(訳)、「すごいErlangゆかいに学ぼう!」、オーム社、2014年、ISBN978-4-274-06912-3、pp.151-164(第12章「エラーとプロセス」)を参照)

[4] http://www.erlang.org/doc/design_principles/sup_princ.html

[5] http://www.erlang.org/doc/man/supervisor.html

[6] http://www.erlang.org/doc/design_principles/applications.html

[7] http://www.erlang.org/doc/man/application.html

[8] http://www.erlang.org/doc/man/app.html

書いて覚える Swift 入門

第9回 Swift 2で、何が変わるのか？

Writer 小飼 弾（こがい だん）　twitter @dankogai

収穫の秋

毎年9月末の「秋のApple祭り」は、毎年恒例になった感があります。新しいiOS、新しいiPhone、そして新しいXcode。iOS 9は日本時間9月17日未明にリリースされ、Xcode 7もそれとほぼ同時にApp Storeでの配布が開始されました。同時にリリースされる予定だったwatchOS 2のリリースこそ5日遅れましたが、あとは9月25日にiPhoneが届けば祭りもクライマックスといったところでしょうか。本稿が読者のみなさんの手に届くころには、iOS 9もiPhone 6s/6s plusもすっかり馴染んでいることでしょう。

しかしXcode 7、そしてSwift 2はどうでしょうか？　以前紹介したとおり、今年中にはAppleはSwiftのオープン化を公約しています。Swift 2に習熟するには絶好の期間かもしれません。

そのXcode、本物ですか？

そのXcodeですが、気になるニュースが1つ。中国で改竄され配布された通称XcodeGhost[注1]で開発、デプロイされたアプリが、マルウェアに感染した状態でApp Storeに登場したのです。

Apple ID さえあればβ版も無料で入手でき、ましてや安定版がMac App Storeから無料で入手できるXcodeのパチモノをわざわざ使うなんて一見ありえなさそうですが、さすがGreat Firewall of Chinaの向こうだけあって、かの国ではAppleの公式サーバとの接続も日本のように安定しているとも言えず、よって（違法に）二次配布するサイトも点在しているようです。飛行機を乗っ取るために空港を乗っ取るようなもので、不謹慎ながら感嘆を禁じ得ませんでした。

Appleもこの件に関しては、異例の注意喚起[注2]を行っています。

そのXcodeが本物か否かは、Terminal.appで`spctl --assess --verbose /Applications/Xcode.app`を実行すればOKです。Xcodeの巨大さゆえ、コマンドが終了するのにしばらく待たされるのですが、問題なければ

```
/Applications/Xcode.app: accepted
source=Mac App Store
```

のようにacceptedを含んだ表示が出るはずです。

Apple「すべてのSwift 1.xプロジェクトを、生まれる前に消し去りたい」

それでは気を取り直して本題へ。iPhone 6sのキャッチコピーは、「唯一変わったのは、そのすべて。」だそうですが、Swiftの変貌ぶりはそれどころではありません。なにしろSwift 1.xのコードが全然そのまま動かないのですから。

注1）http://japanese.engadget.com/2015/09/20/app-store-xcodeghost/
注2）https://developer.apple.com/news/?id=09222015a

Python 2.xとPython 3.xは別言語ですが、Swift 1.xとSwift 2.xの別言語ぶりはそれをも凌駕します。Python 3の普及が遅れている一番の理由はまさにそこ、「同じ言語じゃない」という点にあるのですが、AppleはそれをXcodeでひっくり返しました。Swift 1.xで書かれたプロジェクトを開くと、XcodeがSwift 2.xに翻訳してくれるのです。その一部始終は前々（第7回）に紹介したとおりですが、まるで「すべてのSwift 1.xプロジェクトを、生まれる前に消し去りたい」といわんばかりのこの機能には「そんな祈りが叶うとすれば、それは時間干渉なんてレベルじゃない。因果律そのものに対する反逆だ」とインキュベーターでなくとも叫びたくもなるというものです。Python 3どころかPerl 6とか、「人類には早すぎる言語」との付き合いが浅からぬ筆者には、「さすがApple！おれたちにできない事を平然とやってのけるッそこにシビれる！あこがれるゥ！」という冷やかしすら無理で、ただ伏してメメタァするしかありませんでした。

前々回で私は「Swift 2への移行は正式版が出た後で、ただしSwift 2正式化以後はSwift 1の後方互換性サポートも捨てるのがよさそうです。筆者がGitHubに上げているプロジェクトはそうするつもりです」と述べましたが、ほぼそのようになりました。Swift 2正式版がXcodeとともにリリースされてまだ一週間も経っていないのに、筆者はすでにSwift 1がどうだったか思い出せなくなりつつあります。

What's new in Swift 2

というわけで、後方非互換性はXcode 7の力であっさりと乗り越えてしまいましたが、これからがやっと俺たちの戦い。Swift 2は何が変わったのでしょうか？　完全に書き直されたThe Swift Programming Language[注3]でそれを追うのは、サン・ピエトロ寺院を見てコロッセオがどうだったかを思い起こすほど難しいかもしれません（コロッセオは石切場だったのです。これ豆）。筆者の場合、Swift 1から2への変遷を追うのに一番ありがたかったのはWWDC 2015のビデオ、What's new in Swift[注4]でした。本稿でもそれを改めて駆け足で追いかけた後、Error HandlingやProtocol Extensionといった重要な機能追加に関しては記事を改めてじっくり紹介していく予定です。逆に、println()からprint()など、Xcode 7が自動で片付けてしまう変更点に関しては、軽く触れるにとどめます。

最低限文化的なenum

Swiftのenumは、enumというよりむしろunion（共用体）に近いもので、本当はSwift 1の頃からもっとバリバリ使いたかったのですが、classやstructと比べると微妙に使いづらいものでした。

デフォルトのdescription debugDescriptionが直感的に

たとえば、

```
enum Langs {
    case C, Cplusplu, ObjectiveC, Swift
}
```

というenumがあったとして、print(Langs.Swift)はそのままではそっけなく（Enum Value）と表示されるだけでした。“Swift”と表示させるためには、いちいち、

注3）https://ja.wikipedia.org/wiki/ラムダ計算
注4）https://developer.apple.com/videos/wwdc/2015/?id=106

```
extension Langs : Printable {
  var description: String {
    switch self {
    case C:
          return "C"
        case Cplusplus:
          return "C++"
        case ObjectiveC:
          return "Objective-C"
        case Swift:
          return "Swift"
      }
  }
}
```

などという具合にdescriptionプロパティを定義しておく必要があったのですが、これと同様のことを自動でやってくれます。

総称型enumがまともに

Swift 1では次のコードがクラッシュしていました。

```
enum Either<L,R> {
    case Left(L)
    case Right(R)
}
```

総称型のEnumでは型変数が1つしか取れなかったので、

```
enum Optional<T> {
  case None
  case Some(T)
}
```

は定義できてもEitherのような型は作れなかったのですが、やっと作れるようになりました。

再帰的enumが可能に

Swift 1では、これもだめでした。

```
enum Cons<T> {
    case Nil
    case Atom(T)
    case Pair(Cons, Cons)
}
```

enumは値型(value type)なので、「固定的なサイズを持たなければならないのに、自己参照していては無限にメモリが必要になってしまう」とはSwiftのパパ、Chris Lattnerの弁。「今のデバイスでは無理。まあ来年あたりなら」というジョークが見事に滑っていましたが、indirectを次のようにつけることで再帰的enumが実現可能になりました(**図1**)。

```
enum Cons<T> {
    case Nil
    case Atom(T)
    indirect case Pair(Cons, Cons)
}
```

もしくは、

```
indirect enum Cons<T> {
    case Nil
    case Atom(T)
    case Pair(Cons, Cons)
}
```

ここまでそろっていたなら、筆者もswift-json[注5]をenumベースで実装したんですがねえ……。

repeat { } while

Swift 1までの「必ず1回はブロックを実行する」ループはdo { } whileだったのですが、repeat { } whileになりました。do { }は今後後述のdo { } catch { }などでも用いられるようになるのと、ブロックの頭を見ただけで条件分岐がブロックの後ろにあることが直感的にわかるということでそうなったようです。

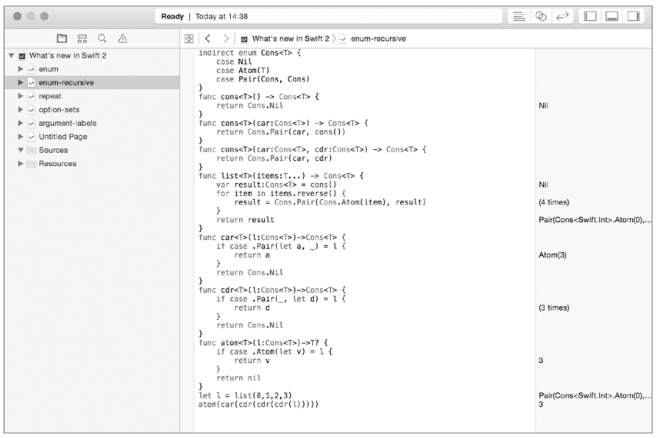
Option Sets

これはSwift 1.2から導入された重複なしの集合、Setとは別物です。

Swift 1ではビットマップのフラグの扱いは、たとえば、

注5) https://github.com/dankogai/swift-json

▼図1　再帰的enum




```
let swiftGoals:Goals = .Refined | .Safe |
.Expressive
```

のように、Bitwise命令で操作していたのですが、わかりづらいということで次のようにOptionSetTypeプロトコルに適合したstructを使ってより直感的にフラグを立てたり折ったりできるようになりました。

```
struct Goals : OptionSetType {
    let rawValue : Int
    static let Refined    = Goals(rawValue:1)
    static let Safe       = Goals(rawValue:2)
    static let Expressive = Goals(rawValue:4)

}
let swiftGoals:Goals = [.Refined, .Safe,
.Expressive]
if swiftGoals.contains(.Expressive){}
```

関数とメソッドとラベルと

Swiftでは関数もメソッドもfuncで定義しますが……、

```
func f(first:String, second:Bool){ }
swtruct S {
  func m(first:String, second:Bool){ }
}
```

Swift 1では関数は、

```
f("Swift", second:true)
```

と最初の引数のラベルが省略可能だったのに対し、

```
var o = S()
o.m(first:"String", second:true)
```

といった具合にメソッドで最初の引数のラベルは省略不能でした。Swift 2では関数の場合と同様、どちらもデフォルトでは最初の引数のみ省略という形におさまりました。_をラベルの前につけることでそのラベルを省略可能にできるのは今までどおりです。

使用する変数に警告

これはあくまでもXcodeの機能であってSwift 2の仕様ではありませんが、varで宣言したはずなのに一度も変更を加えていない変数は、letで定数にするように警告するようになりました。

@testable

ユニットテストの際に1つ困るのは、publicなシンボルにしかアクセスできなかったことですが、ユニットテスト中で、

```
@testable
import MyApp
```

と宣言することで、internalなシンボルにもアクセスできるようになりました。

Markdown in Rich Comments

リッチコメントの中でMarkdown記法が使えるようになりました。画像すら入れられます。

guardでガード

if let記法だと、条件が煩雑だとネストが深くなりがちでした。

```
if let name = json["name"] {
  if let year = json["year"] {
    handlePerson(name, year)
  } else {
    handleError("year is missing")
  }
} else {
  handleError("name is missing")
}
```

たとえばPerlだったら、同様の場合はこう書くところです。

```
my $name = $json->{name} or handlError("name
is missing");
my $year = $json->{year} or handlError("year
is missing");
handlePerson($name, $year);
```

どちらが読みやすいかは一目瞭然ですが、これと同様のことがguard文の導入である程度できるようになりました。

```
guard let name = json["name"] else {
  handleError("name is missing")
}
guard let year = json["year"] else {
  handleError("year is missing")
}
handlePerson(name, year)
```

else { }がまだうざくはありますが、だいぶすっきりしました。elseというのもやや違和感がありますが、これがないと

```
guard let something = someFunction
  { /*...*/ }
```

と書いてしまった場合、

```
guard let something = someFunction({ /*...*/ })
```

と解釈されてしまうおそれがあるのでこのような形になったのでしょう。

case、beyond、switch

パターンマッチは便利なものですが、switchがうざく感じませんでしたか?

```
switch lang {
case .Swift(let version) where version < 2.0:
  updateIt(version)
default: break
}
```

こういう場合に、if caseが使えるようになりました。

```
if case .Swift(let version) = lang where
version = < 2.0 {
  upadteIt(version)
}
```

caseはifだけでなくforでも使えます。

```
for n in numbers where numbers % 2 == 0 {
  handleEven(n);
}
```

if #available

APIバージョンによる条件コンパイルに、#availableが使えるようになりました。

```
override func awakeFromNib() {
  if #available(OSX 10.10.3, *) {
    dropButton.springLoaded = true
  }
}
```

今までextensionをかけられたのはstruct、enum、classといった実装を伴った型のみでしたが、protocolにextensionをかけられるようになりました。

```
extension CollectionType {
  func countIf(match: Element -> Bool) -> Int {
    var count = 0
    for value in self {
      if match(value) { count++ }
    }
    return count
  }
}
```

これは大進歩です。大進歩なので次回以降詳

しく取り上げます。

Exception、ahem、error handling

おそらく巷で最も騒がれているSwift 2の新機能が、try {} catch {}かもしれません。しかしこれはAppleが慎重にexceptionという言葉を避けているように、Javaなどの言語の「例外」からするとかなり限定的な機構です。これも解説は次回以降と、とりあえずこんな感じというものを、あえて何も解説せずに(**図2**)。

String is not just a sequence of characters

String自体がSequenceTypeでなくなり、そのように扱いたければ.characters、.utf8、.utf16を必ず使うようになりました。

Swiftによる文字列の扱いもまたそのための記事を割くだけの意味があるものなので、次回以降あらためて取り上げます。

まとめ

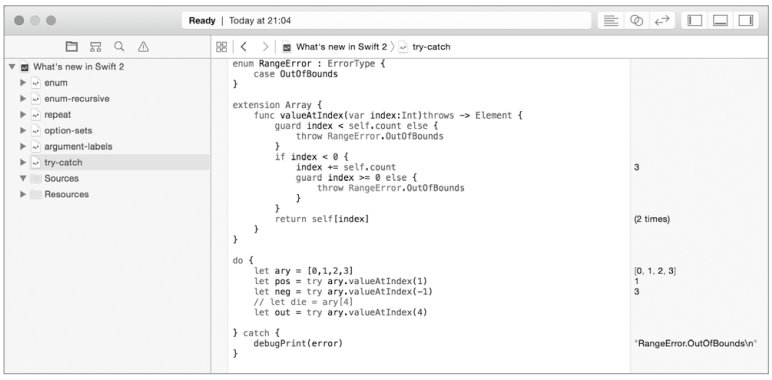
そのほかにも、

- 破壊的ソートだったsort()がsortInPlace()になり、非破壊的なsorted()がsort()になったり
- find()がindexOf()になったり
- .extend()が.appendContentsOf()になったり

などとAppleが変えたい放題変えたSwift 2ですが、世代や性能や機能が変わってもそれがiPhoneであることは見間違えないように、一目見ればそれがSwiftのコードだという特徴をSwift 2は備えているように感じています。シンボルの改名にしても、英語的な洗練を捨ててまで非ネイティブな人でも誤解しにくい名前にしているのは日本のSwiftプログラマにとっては朗報かもしれません。

というわけで今回はSwift 2の変更点と新機能を駆け足で見てきたわけですが、ここで今年の産業界のニュースNo.0ほぼ間違いなしのビッグニュースが飛び込んできました。Appleがいよいよ自動車を再発明する？　いえ。VW社のディーゼル不正事件です。EA189という「クリーンディーゼル」の制御コンピュータに、当局がガスチェックしているときだけエンジンの動作モードを変えるという通称“defeat device”が組み込まれているのを米国環境保護局(EPA)が見つけたというもので、リコール対象50万台、制裁金2兆円というのは序の口で、これまで同エンジンが搭載されたクルマは全世界で1,100万台も売れており、今年上半期トヨタを抜いて世界最大販売台数を達成した同社は、通年でそれを達成する前にそれをはるかに上回る創業以来の危機が到来しています。ソフトウェアとハードウェアの組み合わせで世界を変えるという点で自動車メーカーも実はAppleと同じような立場にあったわけで、それが負の方向に働くとどれほど恐ろしいことになるのか、一エンジニアとして恐々としています。世界を変えるなら、より良い方向に変えていきたいものです。SD

▼図2　try-catch……例外処理？



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19回 同時押し&リピート　少し特殊なキー設定

前回に引き続き、Emacsの操作性を決定づけるキーバインド設定についてお話します。前半では「同時押し」というEmacsには今までなかった操作を導入するパッケージ「key-chord」を、後半では繰り返し入力するようなコマンドを短くできるパッケージ「smartrep」を紹介します。

前回のおさらい

ども、るびきちです。“Emacsの使い勝手は操作性で決まる”というテーマで、前回はキーバインドの設定を中心にお話しました。Emacsはキーボード操作が中心ですので、いかに賢くキーバインドを設定するかが操作性を決定付けます。

言うまでもありませんが、よく使うコマンドはキーに割り当てて然るべきです。そのために、標準でglobal-set-keyとdefine-key関数が用意されています。ですが、個人的にキーバインドの設定をする際には、bind-keyパッケージの同名のマクロを使用するほうが賢明です。短く記述でき、モードに上書きされない強制設定(bind-key*)ができ、個人用キーバインドの列挙(M-x describe-personal-keybindings)もできるからです。

キーバインドの資源は有限ですので、なんでもかんでもキーに割り当てるわけにはいきません。regionやC-uの有無で挙動を変化させるコマンドを定義するmykieパッケージを使って、キーバインドを有効活用しましょう。これを使えば、たとえばC-u C-wやregionなしのC-wに機能を割り当てられるからです。mykieで設定できる“状況”はとても多いので、突き詰めていけば1つのキーにたくさんの機能を割り当てられます。

あまり使用頻度が高くないコマンドは、キーバインドよりも名前で覚えることが多いでしょう。M-xを非常に使いやすくするsmexパッケージを使えば、そういったコマンドをすぐに呼び出せます。また、長い名前のコマンドや押しづらいキーバインドのコマンドはdefaliasで短い別名を付けると、M-xから呼び出しやすくなります。

今回は一風変わったキーバインド設定方法を紹介します。

同時押し・連打にキーを割り当てるkey-chord

キーボード同時押しという新たな境地

割り当てられるキーを増やすには、自分用のプレフィクスキーを用意するのがEmacs的にもっとも自然な方法です。複数個プレフィクスキーを用意したり、あまり割り当てられていないプレフィクスキー(M-sやM-g)を使ったりすれば、かなりの数のコマンドをキーに割り当てられます。

そんな中、ちょっと変わったキー割り当ての方法を提供するのがMELPAのkey-chordパッケージです。chordとは和音であり、key-chordとはキー同時押しを意味します。たとえば、K

▼リスト1　key-chordの設定

```
(require 'key-chord)
;;; タイムラグを設定
(setq key-chord-two-keys-delay 0.04)
(setq key-chord-one-key-delay 0.15)
(key-chord-mode 1)
;;; 設定例
(key-chord-define-global "kl" 'view-mode)
(key-chord-define emacs-lisp-mode-map "df" 'describe-function)
(key-chord-define-global "vv" 'find-file)
```

とLを同時に押すことでview-modeを実行する……なんてことができるようになります。当然、同時押しによるコマンド発動はEmacs的にはまったく新しい発想であり、どのパッケージにも使われていません。同時押し対象の2つのキーを自由に選べばいいのです。マウスジェスチャーが、クリックとドラッグしか能がないマウスに新しい風を吹き込んだように、key-chordは新たな操作性をEmacsに提供します。

本当の「同時押し」はない

少し考えてみればわかることですが、本当に寸分の狂いもなく同時にキーを押すことは不可能です。どうしても0.01秒単位のタイムラグが生じます。そこで、2つのキーが押された間隔が一定時間よりも短い場合を「同時押し」とみなしています。その間隔の設定は実際に使ってみて試行錯誤する必要があります。間隔が短過ぎれば同時押ししたつもりでもコマンドが発動しません。逆に長過ぎれば別々のキーのつもりが同時押しとみなされてコマンドが誤爆してしまいます。

たとえば、JとKの同時押しはホームポジション上ですので押しやすいです。通常のローマ字入力では使わないので極めて有効な選択となります。しかし、拡張ローマ字AZIKでは「じん」「くん」と入力するときに使うので誤爆の可能性が出てきます。AZIKユーザはVや@を使えば誤爆しにくくなります。

すばやく連打する

同時押しの2つのキーとして、同一のキーを選択することもできます。そのときは同時押しではなくて、すばやく連打することでコマンドを発動させます。

この場合でも当然、誤爆のリスクと背中合わせです。2つのキー(文字)が立て続けに登場する確率が低くなるキーを選択する必要があります。間隔も適度に短く設定しなければなりません。

さらに、文字キーを連打キーに設定した場合、入力してから表示されるまでのタイムラグが生じます。連打によるコマンド発動を待っているのだから当然です。次の文字をすばやく打てば問題ありません。

key-chord.elの導入

M-x package install key-chordでインストールしたら、設定をします(**リスト1**)。タイムラグの設定は必須です。ここでは同時押しで0.04秒、連打で0.15秒に設定しています。連打は本気ですばやく打たないと発動しません。

関数名から推測されるようにグローバルマップに作用するkey-chord-define-globalと任意のキーマップに作用するkey-chord-defineがあります。それぞれglobal-set-keyとdefine-keyに相当します。

リピートを活用する

立て続けに実行されるコマンドは1ストロークで

Emacsのコマンドには立て続けに実行され得

るものと、そうでないものがあります。カーソル移動やウィンドウのリサイズは連続的に実行されやすいです。そういうコマンドは1ストロークであればとても実行しやすいです。C-fやM-fは連打するだけで移動を繰り返せます。

一方、ウィンドウのリサイズを行うC-x {やC-x }は連続的に実行され得るにもかかわらず2ストロークです。ウィンドウの大きさを視覚的に確認しながらリサイズするにはC-x {を繰り返す必要があって面倒です。

よって、連続して実行され得るコマンドは1ストロークのキーに割り当てることが望ましいです。M-<**英大文字**>は空いているので、そういうコマンドを割り当てると良いでしょう。

また、グローバルマップではM-pとM-nにはコマンドを割り当てていません。それはメジャーモードにとって相応しいカーソル移動コマンドを割り当てられる余地を残しています。

repeatを活用する

C-x {のように繰り返し実行される複数ストロークコマンドにも救済措置が用意されています。それはC-x zに割り当てられているrepeatコマンドです。

C-x zは直前のコマンドを繰り返します。さらに繰り返すのにプレフィクスキーC-xは不要で、Zを押すだけです。C-x zは次項で紹介するsmartrep的なコマンドです。これによりC-x {を3回実行する場合はC-x { C-x z zと、1ストローク節約できます。

とはいえ、"立て続けに実行されるコマンドは1ストロークに割り当てる"という原則にならえば、repeatも1ストロークのキーに割り当て直したほうが快適です。たとえば、C-,に割り当てたとすれば、C-x {×3は、C-x { C-, C-,と、2ストローク節約できます。

M-x repeatをC-,に割り当てる設定は、

```
(global-set-key (kbd "C-,") 'repeat)
```

または、

```
(require 'bind-key)
(bind-key* "C-," 'repeat)
```

です。

プレフィクスキーを省略させるsmartrep

特定のコマンドを繰り返し実行するときに、プレフィクスキーを省略させるsmartrepパッケージがMELPAにありますので、M-x package install smartrepでインストールしましょう。smartrepは「smart repeat」の略で、繰り返しを快適にします。

repeatを1ストロークに割り当てれば、直前のコマンドのみのプレフィクスキーを省略できますが、smartrepを導入すれば省略対象のコマンドを複数個持てます。hydraという類似品が最近登場しましたが、本稿の対象読者を考慮して日本人作かつユーザ数の多いsmartrepを紹介します。

smartrepを使うには、requireしたあとにsmartrep-define-key関数を使います。それでは2つの具体例からsmartrepの使い方を見ていきましょう。

org-modeで見出し移動を快適にする

org-modeには見出し移動コマンドがたくさん用意されています(**表1**)。一度見出し移動コマンドを実行したら、次も見出し移動コマンドが実行される可能性があります。これらのコマンドを実行中に、C-cを省略できるようにするには、**リスト2**の設定をします。

これによりC-c C-u C-c C-f C-c C-nがC-c C-u C-f C-nのように操作できます。実際にやってみると、見出し移動がとても快適になることがわかります。smartrep状態を解除するにはC-gを押すか、登録されていないキーを押します。登録されていないキーを押すと、元のコマンドが実行されます。たとえばAを押すとそのまま「a」が入力されます。

▼表1　org-modeでの見出し移動コマンド

コマンド	動作
C-c C-n	次の見出しへ
C-c C-p	前の見出しへ
C-c C-u	上の見出しへ
C-c C-f	同じレベルの次の見出しへ
C-c C-b	同じレベルの前の見出しへ

▼表2　ウィンドウ操作コマンド

コマンド	動作
C-x o	隣のウィンドウに切り替える
C-x 0	ウィンドウを削除する
C-x 1	他のウィンドウをすべて削除する
C-x 2	上下分割する
C-x 3	左右分割する
C-x {	ウィンドウを横方向に縮める
C-x }	ウィンドウを横方向に広げる
C-x +	ウィンドウの大きさをそろえる
C-x ^	ウィンドウを縦方向に広げる
C-x -	ウィンドウを縦方向に縮める

ウィンドウ操作をひとまとめにする

ウィンドウを操作する代表的なコマンドはC-xをプレフィクスキーにしています。ウィンドウの分割状態を作成するには、分割したりリサイズしたりするコマンドを連続的に使う必要があります。そういう場合に、smartrepは大きな威力を発揮します。

リスト3では**表2**のウィンドウ操作コマンドをsmartrepで扱えるようにします。ついでにC-x -を、ウィンドウを縦方向に縮めるコマンドにしておきます。もともとこのコマンドは、そのウィンドウの行数に合わせてウィンドウを縮めるものですが、ここではC-x ^の対になるコマンドがほしいという理由で設定しています。

使ってみればわかりますが、この設定を施したあとは、ウィンドウを適切な大きさに分割させることが楽しくなることうけあいです。

その他の方法

操作性を快適にする方法はほかにもいろいろあります。

key-comboパッケージは同じキーを連続的に使用したときの挙動を変えられます。C-aを2回でバッファ先頭に移動したりといったことができます。

mag-menuパッケージは強力なメニューインターフェースを提供します。メニューを経由して複数の機能をひとまとめにできます。

▼リスト2　見出し移動コマンドのC-cを省略

```
(require 'smartrep)
(require 'org)
(smartrep-define-key org-mode-map "C-c"
  '(("C-n" . org-next-visible-heading)
    ("C-p" . org-previous-visible-heading)
    ("C-u" . outline-up-heading)
    ("C-f" . org-forward-heading-same-level)
    ("C-b" . org-backward-heading-same-level)))
```

▼リスト3　ウィンドウ操作コマンドのC-xを省略

```
(require 'smartrep)
(smartrep-define-key global-map "C-x"
  '(("o" . other-window)
    ("0" . delete-window)
    ("1" . delete-other-windows)
    ("2" . split-window-below)
    ("3" . split-window-right)
    ("{" . shrink-window-horizontally)
    ("}" . enlarge-window-horizontally)
    ("+" . balance-windows)
    ("^" . enlarge-window)
    ("-" . shrink-window)))
```

おわりに

今回も前回に引き続きキーバインド特集でしたが、今回は一風変わった方法を紹介しました。

筆者はサイト「日刊Emacs」を運営し、毎日パッケージの紹介記事を書いています。マイナーなものも紹介しているので、新たなパッケージを求めている人の役に立てば幸いです。また、Emacsユーザの QOL を上げるための厳選した情報を週間メルマガで配信しています。Emacsについてはもちろんのこと、ライフハックなどいろいろな分野について書いています。SD

登録はこちら➡http://www.mag2.com/m/0001373131.html

一歩進んだ使い方のためのイロハ

Vimの細道

第2回

VimでJavaを使う（Eclim編）

mattn
twitter:@mattn_jp

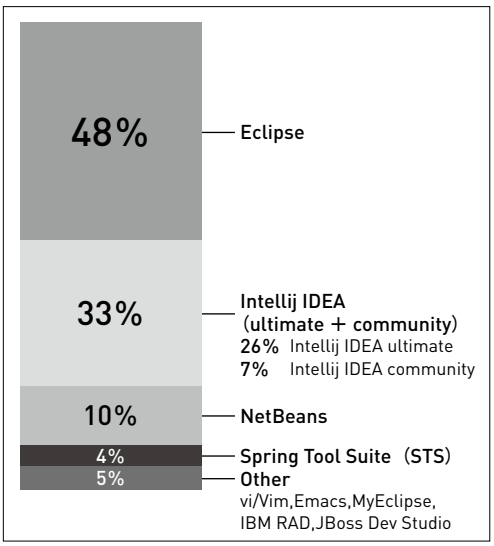
Javaでの開発にはEclipseといったIDEを使うのが一般的ですが、普段から使い慣れているVimを使って開発したい、といった人も多いはず。今回は、Eclipseをバックグラウンドで動かしながら、Eclipseと同等の機能をVimに提供する「Eclim」を紹介します。

VimでJava、は果たして絶滅か？

突然ですが、皆さんはJavaで開発をするとき何を使っていますか？　IDEですか？　テキストエディタですか？　Eclipse？　IntelliJ IDEA？　それともVim？　Emacs？

2014年にZEROTURNAROUND社が発表した「Java Tools and Technologies Landscape for 2014」[注1]という調査資料によると、ユーザがJavaで開発する際のツールの使用率は**図1**のようになっています。この調査によるとユーザの48%がEclipseを使い、33%がIntelliJ IDEAを、10%がNetBeansを使っています。VimやEmacsやそのほかのIDEという言葉でまとめられた部分は5%です。つまり、Vimを使ってJavaを開発している人は超極小ユーザなのです。

▼図1　Javaで開発するときのツールの使用率



ところでみなさんのEclipseはどれくらいの時間で起動しますか？　数秒？　数十秒？　もしかして分オーダーですか？　せっかくいいアイデアが浮かんだのにEclipseを起動しているあいだに頭の中から飛んで行ってしまった、なんてことはありませんか？　また、最近は見なくなりましたが、「書くぞー！」と気合を入れたはいいが、操作中に「OutOfMemoryError: Perm Gen space」なんてエラーとともに再起動を余儀なくされた経験はありませんか？

「だってVimはエディタでしょ？」「入力補完できないからVimでJavaをコーディングするのはキツいよね」と思っておられる方も多いと思います。果たしてVimやEmacsではJavaを開発することはできないのでしょうか？　やっぱりVimはエディタなのでしょうか？　実はそんなことはないのです。VimでJavaの開発をするには、大きく分けて2つの方法があります。

・Eclimを使い、Eclipseと同じ機能をVimから

注1）URL http://zeroturnaround.com/rebellabs/java-tools-and-technologies-landscape-for-2014

使う
・mavenやgradleをプロジェクト管理に使い、java-complete2などの補助プラグインを使う

今回はこの2つの方法のうち、Eclimを使って開発を行う方法を紹介したいと思います。

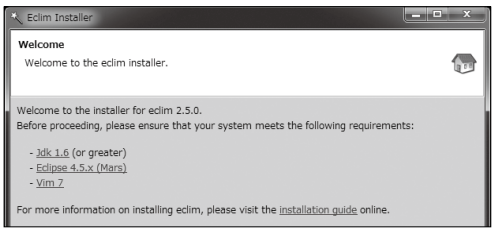
Eclimとは

Eclim[注2]はコード補完、検索、コード検証といったEclipseが持ち合わせている機能を、Eclipseと同等レベルでVimに提供するためのプラグインです。もちろんJavaだけではなく、C++、HTML/CSS、JavaScript、PHP、Python、Ruby、XML/DTD/XSD、そしてなぜだかVim scriptに対してEclipseが提供する機能をVimから使えるようにできます。機能の詳細は公式ページ[注3]を参照してください。

Javaに関連するところであればAnt/Mavenといったプロジェクト連携はもちろん、Androidの開発も行えます。次はEclimが提供するJavaに関する機能です。

・自動コード検証
・コンテキストを意識したコード補完
・サジェスト形式のコード修正提案
・クラスコンストラクタを生成
・Java Beanのゲッタおよびセッタを生成
・デリゲートメソッドを生成
・Javaソースと Javaドキュメントの検索
・インターフェースやスーパークラスからスタブメソッドを生成
・jUnitテストのためのスタブメソッドを生成
・import文の敏速な除去・並び替え。新しいクラスのimport
・ロガー初期化コードの自動生成。さらには初

▼図2　Eclimの GUIインストーラ



回の使用説明
・パッケージ、クラス、フィールド、メソッド、その他に関するJavadocの生成
・Javaの正規表現テスト
・Checkstyleのサポート
・log4j XMLファイルの検証

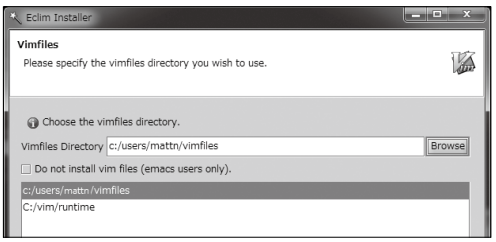
とまぁ盛りだくさんです。これだけの機能すべてがVimだけで実現されているわけではありません。Eclimは名前からも想像がつくとおりEclipseと連携するプラグインです。使用するためにはEclipseを導入している必要がありますし、VimでJavaを編集する際にはEclipseの起動が必要になります(実際にはバックグラウンドで起動させます)。Linuxの場合、バックグランドで起動するとはいえ、X Window Systemの環境が必要になります。ただしこれについては「Xvfb」を使って起動する方法もあるので、どうしてもリモートからEclimを使ってJava開発したいという奇特な人でも安心です。

Eclimをインストールする

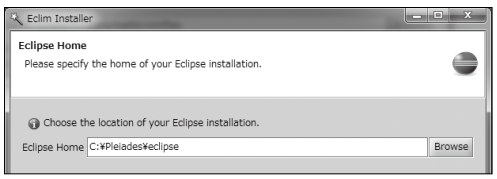
ダウンロードページ[注4]からeclimのインストーラ(jarファイル)をダウンロードします。執筆時点でのバージョンは2.5.0です。推奨環境はJDK1.6、Eclipse 4.5.x (Mars)、Vim7となっています。筆者は Windows 上に「Pleiades All in One 」の Eclipse 4.5 Mars をインストールしました(インストール先は`C:\Pleiades`)。「eclim_2.5.0.jar」を起動すると Eclim の GUIインストーラが起動します(**図2**)。自身の .vim ディレクト

注2) URL http://eclim.org
注3) URL http://eclim.org/features.html
注4) URL http://eclim.org/install.html

▼図3　vimfilesの場所を選択



▼図4　Eclipse のインストール場所を選択



リ(Windows であれば vimfiles でも可)が自動判定されるので選択します(**図3**)。Eclipse のインストール場所を聞かれるので`C:\Pleiades\eclipse`を選択します(**図4**)。最後にインストールする開発環境を聞かれるので、Java Development と Web Development をチェックしてインストールを完了させます。eclipse コマンドと同じ位置にeclimdというコマンドがインストールされます。これらは同じインストーラを使ってアンインストールできますので、インストーラはどこかに置いておくと良いです。

Eclimを起動する

まず、Eclipseをインストールしたフォルダにeclimdというコマンドがインストールされるので、それを起動します。これによりバックグラウンドでEclipseが起動します。

```
$ eclimd
```

▼リスト1　Javaのコード例

```
package io.gitub.mattn;

public class Foo {
        public static void main(String[] args) {
        System.out.println("hello world");
        }
}
```

そして別の端末でVimを起動し、次のコマンドを実行します。

```
:PingEclim
```

正しくセットアップできているのであれば次のようにEclimとEclipseのバージョンが表示されるはずです。

```
eclim   2.5.0
eclipse 4.5.0
```

なお、Windowsでencodingをutf-8にしてVimを使っておられる方は注意が必要で、あとで紹介するflymake機能などでjavaコマンドの出力メッセージが文字化けします。この場合、Windowsの環境変数の設定画面で`ECLIMD_OPTS`に`-Dfile.encoding=utf-8`を設定したあとでeclimdを起動します。ただ、Windowsで Javaの開発をするならば、Eclimに関係なく`eclipse.ini`に`-Dfile.encoding=utf-8`を足しておいたほうが、いろいろと問題が解決する場合があります。

Eclimを使う

プロジェクトを開く

新規にプロジェクトを作る場合は次のように実行します。

```
:ProjectCreate /path/to/my_project -n java
```

またEclimのワークスペースにプロジェクトが取り込まれていない場合は、次のようにしてプロジェクトを取り込んでください。

```
:ProjectImport /path/to/my_project
```

これで準備は完了です。プロジェクトのフォルダでVimを起動し、プロジェクトを開きます。

```
:ProjectOpen
```

クラスファイルを新規に作成する場合は次のように実行します。

```
:JavaNew class io.github.mattn.Foo
```

空のクラスが生成されるので、**リスト1**のようにコードを入力して:Javaを実行するとmain関数が実行されます。

コード補完

コード補完を使うには次の設定が必要です。

```
let g:EclimCompletionMethod = 'omnifunc'
```

vimrcに追加しておきましょう。これで<C-X><C-O>をタイプすると入力補完が効くようになります(**図5**)。補完エンジンもEclipseと同じものですので、入力位置に対する候補(コンテキスト)もバッチリ扱えています。

また、EclimではSuperTab、AutoComplPop、neocomplcache、YouCompleteMeといった補完プラグインとも連携できるようになっています。詳しくは公式ページ[注5]を参照してください。

import文の操作

importされていないクラスからimportを追加するには、クラスにカーソルを合わせ、

```
:JavaImport
```

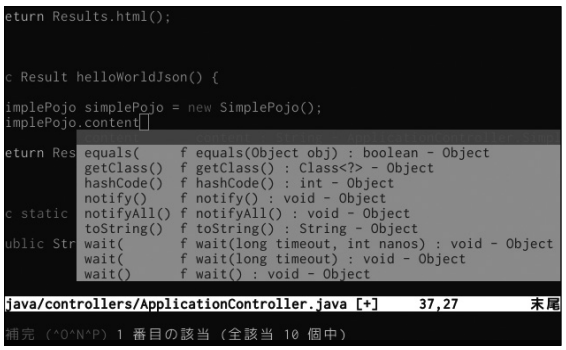
を実行します。候補が一意に決まらない場合は**図6**のように選択リストを表示してくれます。また、不要なimport文を消したり体裁を整えたりする場合は、

```
:JavaImportOrganize
```

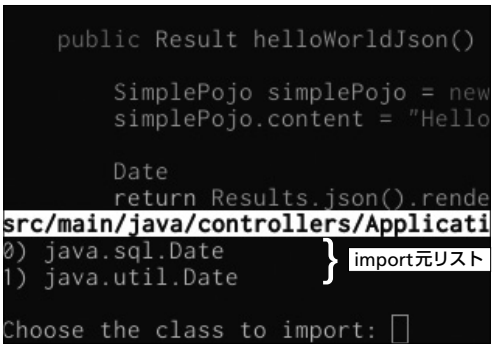
を実行します。

注5) URL http://eclim.org/vim/code_completion.html

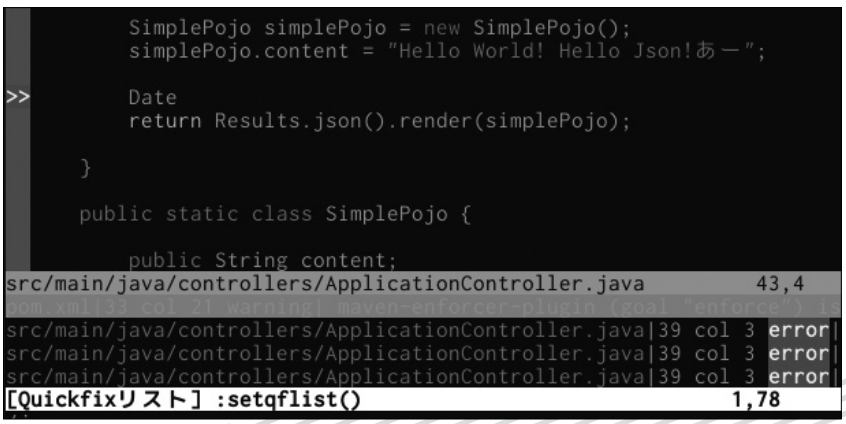
▼図5　Eclimのコード補完



▼図6　import元の候補を表示



▼図7　flymake(画像は:ProjectProblemsを実行後)



リファクタリング

リファクタリングもVimから扱えます。名前を変更したいシンボルにカーソルを合わせて、

```
:JavaRename <変更したい名前>
```

を実行します。プレビューも確認できるので、安心してリファクタリングを行えます。

▼表1　Eclimのプロジェクトに関するコマンド

名前	説明
:ProjectCreate <folder> [-p <project_name>] -n <nature> ... [-d <project_dependency> ...]	新しいプロジェクトを作成
:ProjectImport <folder>	Eclipseプロジェクトフォルダに存在するプロジェクトをインポート
:ProjectList	現在のプロジェクトを一覧表示
:ProjectSettings [<project>]	プロジェクトの設定を表示／編集
:ProjectDelete <project>	プロジェクトを削除
:ProjectRename [<project>] <name>	プロジェクトをリネーム
:ProjectMove [<project>] <dir>	プロジェクトを移動
:ProjectRefresh [<project> <project> ...]	現在のディスク上のファイルが含まれるプロジェクトのファイルリストを更新。プロジェクトが指定されない場合は現在のプロジェクトを更新
:ProjectRefreshAll	すべてのプロジェクトをリフレッシュ
:ProjectBuild [<project>]	現在(もしくは指定)のプロジェクトをビルド
:ProjectInfo [<project>]	現在(もしくは指定)のプロジェクト情報を表示
:ProjectOpen [<project>]	プロジェクトを開く
:ProjectClose [<project>]	プロジェクトを閉じる
:ProjectNatures [<project>]	1つもしくはすべてのプロジェクトに関する設定済みのNatureを表示
:ProjectNatureAdd <project> [<nature> ...]	1つもしくは複数のNatureをプロジェクトに追加
:ProjectNatureRemove <project> [<nature> ...]	プロジェクトから1つもしくは複数のNatureを削除
:ProjectProblems [<project>]	Eclipseの現在(もしくは指定)のプロジェクトに関するビルドエラーや警告をVimのquickfixに委任
:ProjectCD	現在のソースファイルが含まれるプロジェクトのルートディレクトリを全体のルートディレクトリとして変更(:cdを実行)
:ProjectLCD	現在のウィンドウのディレクトリを現在のファイルのプロジェクトルートに変更(:lcdを実行)
:ProjectTree [<project> <project> ...]	指定のプロジェクト(もしくは複数のプロジェクト)のファイルツリーを開く
:ProjectsTree	すべてのプロジェクトが含まれるツリーを開く
:ProjectTab <project>	プロジェクトツリーを新しいタブで開き、ローカル作業ディレクトリを指定のプロジェクトルートに変更
:ProjectGrep /<pattern>/ file_pattern [file_pattern ...]	現在のプロジェクトルートディレクトリでvimgrepを実行
:ProjectGrepAdd /<pattern>/ file_pattern [file_pattern ...]	現在のプロジェクトルートディレクトリでvimgrepaddを実行
:ProjectLGrep /<pattern>/ file_pattern [file_pattern ...]	現在のプロジェクトルートディレクトリでlvimgrepを実行
:ProjectLGrepAdd /<pattern>/ file_pattern [file_pattern ...]	現在のプロジェクトルートディレクトリでlvimgrepaddを実行
:ProjectTodo	プロジェクトファイルのTODO／FIXMEを検索し、ロケーションリストとして追加
:Todo	現在のファイルのTODO／FIXMEを検索し、ロケーションリストとして追加

コンパイルエラーの表示

ファイルを保存すれば、Eclipseと同様にflymake(文法チェック)が実行され、**図7**のようにエラーがわかりやすく表示されます。

よく使うコマンド

そのほか、Javaを開発するうえで便利なコマンドを紹介します。**表1**に示したプロジェクトに関するコマンドは、Javaに限らないコマンドですので、Eclimを開発環境として採用したい

▼表2　EclimのJavaに関するコマンド

名前	説明
:JavaGet	JavaBeansのゲッタを作成
:JavaSet	JavaBeans のセッタを作成
:JavaGetSet	JavaBeansのセッタとゲッタを両方作成
:JavaConstructor	クラスのコンストラクタを作成。選択されているクラスフィールドをベースにできる
:JavaCallHierarchy	カーソル下のメソッド階層を表示
:JavaHierarchy	型階層を表示
:JavaImpl	実装を表示。スーパークラスからの継承、インターフェースからの実装もできる
:JavaDelegate	カーソル下のフィールドに対するデリゲートメソッドを表示
:JUnit [testcase]	JUnitテストケースを実行
:JUnitFindTest	現在開いているファイルに対するテストを探す
:JUnitImpl	テストメソッドを作成(:JavaImplと同じ)
:JUnitResult [testcase]	テストケースの結果を表示
:JavaImport	カーソル下のクラスをインポート
:JavaImportOrganize	未定義の型のimportを追加し、不要なimportを削除。ソートし、体裁を整える
:JavaSearch [-p <pattern>] [-t <type>] [-x <context>] [-s <scope>]	クラス／メソッド／フィールド／その他を検索。カーソル下の単語をパターンとして与えられる
:JavaSearchContext	カーソル下のエレメントに対するコンテキストを考慮した検索
:JavaCorrect	現在のソースに対して可能な修正を提案
:JavaDocSearch	Javadocを検索。:JavaSearchと同様に動作
:JavaDocComment	カーソル下のエレメントのコメントを追加／更新
:JavaDocPreview	カーソル下のエレメントのJavadocをVimのプレビューウィンドウに表示
:JavaRename [new_name]	カーソル下のエレメントをリネーム
:JavaMove [new_package]	現在のクラス／インターフェースを別のパッケージに移動
:Java	プロジェクトのmainクラスを実行
:JavaClasspath [-d <delim>]	プロジェクトのクラスパスをパスセパレータ(もしくは指定のセパレータ)で分割してエコー表示
:Javadoc [file, file, ...]	すべてのソースファイル(もしくは指定のソースファイル)に対してjavadocを実行
:JavaListInstalls	インストールされているJDK/JREを一覧表示
:JavaFormat	Javaソースコードのフォーマット
:Checkstyle	現在のファイルに対してcheckstyleを実行
:Jps	現在実行中のJavaプロセスに関する情報を新しいウィンドウに表示
:Validate	ソースコードの検証を手動で実行
:Mvn	mavenプロジェクトの場合は引数のmavenコマンドを実行
:JUnit	JUnitテストを実行

ならば覚えておくべきです。**表2**はJavaに関するコマンドで、Eclipseで扱えるJavaの編集操作はひととおりVimから実行できるようになっています。Eclim上の:Javaコマンドは、プロジェクト内で設定されたクラスパスを元に起動するので、quickrunで実行するよりも高度な起動方法が可能です。

まとめ

今回はVimからEclipseと同じ機能が使えるEclimについて紹介しました。さすがEclipseがバックエンドというだけあって、かなり品質の高い機能が使えるようになっています。ただし、こんなにも便利なのですが、Webサーバを起動するための設定が一部Eclimだけでは行えなかったり、Maven Projectからのインポート機能がEclimから使えなかったりと、Eclipseを起動しないと設定できない項目がいくつかあります。そこさえEclipseで設定してしまえば、あとはVimだけでもJavaを開発することは十分に可能です。ですがやはり起動が遅い点や、sshでログインした先でeclimdが起動できないのも残念な部分です(Xvfbによる回避方法もありますが……)。

次回はEclimとは別のアプローチを使ったJava開発用Vimプラグイン「javacomplete2」を使う方法を紹介します。SD

Vim日報

シームレスな入力補完

Vimのコード補完機能は、オムニ補完と呼ばれ一般的なテキストエディタやIDEが持っている補完機能とは少し異なる動作をします。たとえば一般的なものは.や→をタイプした際に自動で補完が開始されますが、Vimではデフォルトでそのような動作は行われず、<C-X><C-O>をタイプしなければなりません。さらに、補完を行うと候補一覧の先頭のアイテムが勝手に選択されてしまい、なおかつ編集部分にその候補が入力されてしまいます。

たとえばGo言語で、fmt.Prinまでタイプして補完を行うと、Print、Println、Printfという3つの候補の表示だけされてほしいのですが、実際にはPrintが入力されてしまいます(**図A**)。もちろん<C-N>や<C-P>でほかの候補を選択できるのですが、ほかのテキストエディタやIDEと大きく挙動が異なるため、戸惑ってしまう人もいます。またこれらを改善するための、SuperTab、AutoComplPop、neocomplete、YouCompleteMeといったVimプラグインも存在します。

バージョン7.4.775で取り込まれたnoinsert・noselectオプションを使うことでこの問題を回避できます。勝手に入力させないためにはオプションcompleteoptにnoinsertを足します。

```
:set completeopt+=noinsert
```

さらに先頭の候補が初期状態で選択されないようにするには、noselectも追加します。

```
:set completeopt+=noselect
```

これで一般的なテキストエディタやIDEと同じ動作になりました(**図B**)。

▼図A　最初の候補が入力されてしまう

```
func main() {
    fmt.Print(
}       func Print(a ...interface{}) (n int, err error)
        func Printf(format string, a ...interface{}) (n int, err
        func Println(a ...interface{}) (n int, err error)
~
~
```

▼図B　3つの候補を表示

```
func main() {
    fmt.Prin
}       func Print(a ...interface{}) (n int, err error)
        func Printf(format string, a ...interface{}) (n int, er
        func Println(a ...interface{}) (n int, err error)
~
~
```

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第22回 オプションの魅力

某シェル芸の師範代は、かなりレアなオプションまで知悉しているので驚愕しますが、さらに→←→→→A＋Bとか裏コマンドまであるので愕然としました（嘘）。

to be Continued

UNIXコマンドは、実行に成功すれば何も出力しないことが多く、はじめてコマンドラインを触れる人にとってはとっつき難いかもしれません。現在の状態ぐらいは表示が欲しいときもありますね。そういう場合は「冗長モード」の「-v」オプションなどを使ってみましょう。mvやcp等では実行するアクションを出力してくれます。「本当に削除していい？」という確認がないと不安になる症候群の皆さまには、rmやmvでも「-i」オプションを使えば削除、上書きするかを確認してくれます。これらの確認、出力オプションは、そのコマンドごとに用意されているマニュアルで確認できるのでmanで見ましょう。あ、そのコマンドたちは、メンテナ達の「愛」でできてるよ。

Sphinxで始めるドキュメント作成術

熊谷 章治 Kumagai Shoji Twitter @shkumagai

第8回 HTMLテーマをカスタマイズしてみよう ——ドキュメントの見た目を変える

HTMLテーマとは

前々回の「Webサイトを作ろう（前編）」では、Sphinxの機能の1つであるHTMLテーマを利用した見た目の変更方法に触れました。今回はさらに一歩踏み込んで、HTMLテーマのしくみと作成方法、配布方法について紹介します。

はじめにHTMLテーマ機能とはどんなものなのかについて触れておきましょう。HTMLテーマとは「HTMLテンプレートとCSSを使ってドキュメントの見た目をそこそこ自由にカスタマイズ可能にする機能」です。

Sphinxをインストールすると、最初から12種類のHTMLテーマを使うことができます（ここでは組み込みテーマと呼ぶことにします）。初期状態ではalabasterという白を基調にしたシンプルなHTMLテーマが適用されます（**図1**）。これを別のHTMLテーマに変更するにはドキュメントプロジェクトのconf.pyで次のように設定を変更し、make htmlを実行します。

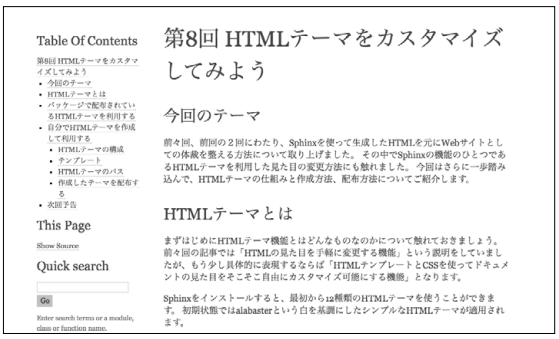
```
変更前
html_theme = 'alabaster'
変更後
html_theme = 'bizstyle'
```

すると、**図2**のようにテーマが適用されます。

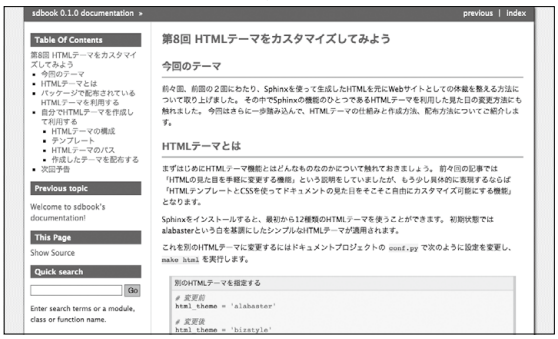
また、いくつかのHTMLテーマにはオプションが利用可能なものがあります（オプションがないHTMLテーマもあります）。たとえば、**図2**のbizstyleテーマにはrightsidebarオプションがあり、このオプションを有効にするとサイドバーが画面の右側に表示されるようになります。このオプションを利用するにはconf.pyの中でhtml_theme_optionsというパラメータの値を次のように設定します。

```
変更前
#html_theme_options = {}
変更後
html_theme_options = {
  "rightsidebar": true,
}
```

▼図1 HTMLテーマ「alabaster」



▼図2 HTMLテーマ「bizstyle」



html_theme_optionsの記述形式はPythonの辞書型です。複数のオプションをキー・バリューの形式で記述できます。conf.pyの変更を保存したらmake htmlを実行してドキュメントをビルドしなおしましょう。すると、**図3**のようになります。

ここまでに説明した情報はSphinx公式ドキュメントのHTMLテーマのページ[注1]やその日本語版[注2]にまとまっています。このページにはそれぞれの組み込みテーマのスクリーンショット画像と利用可能なオプションについての説明が載っています。組み込みテーマをちょっとカスタマイズして使いたい場合には、ここを参考にしながら変更してみると良いでしょう。

パッケージで配布されているHTMLテーマを利用する

Pythonには Python Package Index[注3](以下、PyPI)と呼ばれる公式のパッケージリポジトリがあります。Pythonを使っている方ならすでにご存じの名前だと思いますが、ここに登録されているパッケージの中にはSphinxのHTMLテーマも数多く含まれています。サイトトップ右上の検索窓にsphinx themeと入れて検索すると登録されているHTMLテーマのパッケージ一覧が見られるので、その中から探してみると良いでしょう(**図4**)。

気に入ったHTMLテーマのパッケージを見つけたらpipでインストールしましょう。

sphinxtheme.readabilityのインストール例
```
$ pip install sphinxtheme.readability
```

パッケージで配布されているHTMLテーマの利用方法は、組み込みのテーマとほぼ同じです。前節でも紹介したように、HTMLテーマを変更するときはconf.pyの中でhtml_themeの値にHTMLテーマの名前を指定するのですが、それだけではHTMLテーマが読み込めないものもあります。そのようなパッケージはたいていの場合、パッケージのドキュメントにインストール方法と利用手順について記載があるので、その内容に従って必要な設定をすれば大丈夫です。先述のsphinxtheme.readabilityの場合だと、conf.pyに**リスト1**のように指定します。

またパッケージ配布されているHTMLテーマ

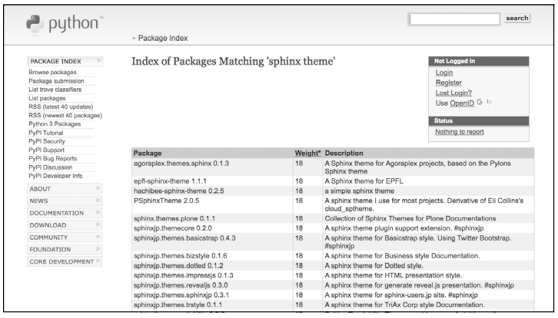
注1) http://sphinx-doc.org/theming.html
注2) http://docs.sphinx-users.jp/theming.html
注3) https://pypi.python.org/pypi

▼リスト1　パッケージのHTMLテーマを使う場合の設定例

```
import sphinxtheme

readability_path = os.path.dirname(os.path.abspath(sphinxtheme.__file__))
relative_path = os.path.relpath(readability_path, os.path.abspath('.'))

html_theme = 'readability'
html_theme_path = [relative_path]
```

▼図3　bizstyleテーマのrightsidebarオプション適用後

▼図4　PyPIのHTMLテーマのパッケージ一覧



にも、オプションを持つものがあります。これらの使い方も組み込みテーマのオプションと同じで、conf.pyのhtml_theme_optionsに設定を記述することで変更ができます。こちらもsphinxtheme.readabilityを例に記述方法のサンプルを**リスト2**に示します。

自分でHTMLテーマを作成して利用する

Sphinxの組み込みのテーマとPyPIからインストールできるHTMLテーマだけでも相当の種類があります。そのため、どれか1つくらいは目的に適ったHTMLテーマが見つかるものですが、ときには思ったとおりのものが見つからない場合もあります。そんなときはHTMLテーマを自分で作ってみましょう。

HTMLの作成に際してはHTMLやCSSの知識は必要になりますが、最初は組み込みテーマのコピー&ペーストと変更から始めて少しずつ理解を広げていけば良いので、ぜひチャレンジしてみてください。ここではsd_themeというHTMLテーマを作成するものとして、HTMLテーマの構成を説明します。

HTMLテーマの構成

HTMLテーマの内部はいくつかの決められたファイルを配置したディレクトリ構造をしています。sd_themeの場合、**図5**のような構造になります。

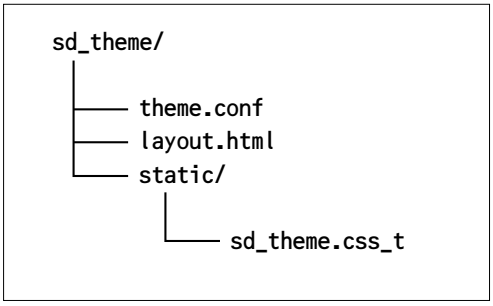
- theme.confはINI形式のテキストフォーマットで記述されたHTMLテーマの定義ファイル。HTMLテーマには必須のファイルで、このHTMLテーマがどのような設定かを記述する
- layout.htmlはHTMLテンプレートファイル。ここでは1ファイルのみだが、必要に応じて複数ファイルを作成／提供することもできる
- staticはHTMLテーマの中で利用する静的ファイルを配置するディレクトリ。画像ファイル、スタイルシート、スクリプトファイルなどを配置しておくと、ドキュメントのビルド時に出力ディレクトリにコピーされる
- sd_theme.css_tはこのHTMLテーマのスタイルシートとして利用するファイル。Sphinxの HTMLテーマ機能が持つ静的テンプレート[注4]という機能を利用することで、ドキュメントのビルド時にHTMLと同じようにテンプレートから静的ファイルを生成する

では、実際にsd_themeのtheme.confをどのように記述するか見てみましょう(**リスト3**)。

ファイル先頭の[theme]はテーマの定義に関するセクションであることを宣言します。

次のinheritは継承元のテーマ名もしくはnoneを指定します。継承元のテーマを指定しておくと、継承元テーマのテンプレートが使用されます。このため継承元のテーマが持っているテンプレートと同じものをすべて提供する必要

▼図5　テーマディレクトリの構造



▼リスト2　パッケージのHTMLテーマのオプションを指定する方法

```
html_theme_options = {
    'readabilitystyle': 'novel',  ←テーマ内のstyleに'novel'を指定
    'nosidebar': True,            ←サイドバーを非表示
}
```

▼リスト3　theme.confの記述例

```
[theme]
inherit = basic
stylesheet = sd_theme.css
pygments_style = friendly

[options]
rightsidebar = true
```

注4) http://docs.sphinx-users.jp/theming.html#static-templates

はありません。

stylesheetはHTMLヘッダから参照されるCSSファイル名を指定します。この値はconf.pyのhtml_styleで上書きできます。

pygments_styleはシンタックスハイライトに使用するPygments[注5]のスタイル名を指定します。この値はconf.pyのpygments_styleで上書きできます。

その次の[options]はオプションを定義するセクションです。このセクションにはオプション名とそのデフォルト値のペアを記述していきます。**リスト3**ではrightsidebarオプションのデフォルト値としてtrueを設定しています。これらの設定はconf.pyのhtml_theme_optionsを記述することで上書きできます。またテンプレートの中からはtheme_<名前>という形式でこの設定値にアクセスできます(後述)。

テンプレート

SphinxはHTMLテンプレートとしてJinja2[注6]というテンプレートエンジンを利用しています。Pythonを使ってWebアプリケーションを作成したことのあるエンジニアの方であれば、どこかで名前だけでも目にしたことがあるかもしれません。Jinja2に関する具体的な説明は長くなってしまうので割愛します。Sphinxの中でJinja2のテンプレーティングを行う場合に必要な情報は公式ドキュメントのテンプレートガイド[注7]を参照してください。

sd_themeのテンプレートも可能であれば一から作成していくほうが良いのですが、ここでは便宜的に既存のテーマを継承したシンプルなテンプレートの例として、組み込みのテーマsphinxdocのテンプレートファイルを拝借してくることにします。sphinxdocのテンプレートはlayout.html[注8]の1ファイルのみで、内容は**リスト4**のとおりです。

前半の{#から#}までは複数行コメントで、生成されるHTMLファイルには出力されません。次の{% extends "basic/layout.html" %}の部分で継承元のテーマであるbasicのテンプレートを継承しています。こうすることで継承元テーマのテンプレート内で宣言されている内容をそのまま流用することができます。さらに、それ以降の部分で必要に応じて宣言内容を上書きしたり、追加の宣言を定義したりしています。

このテンプレートに記述されている内容から、「sphinxdocテーマのテンプレートは継承元であるbasicテーマのテンプレートにほぼ変更を加えずに利用している」ことがわかります。これをsd_themeディレクトリの直下に置きます。

そしてもう1つ、スタイルシートもsphinxdocのテーマから拝借することにしましょう。CSS

注5) http://pygments.org/
注6) http://jinja.pocoo.org/docs/dev/
注7) http://docs.sphinx-users.jp/templating.html#jinja-sphinx-templating-primer
注8) https://github.com/sphinx-doc/sphinx/blob/master/sphinx/themes/sphinxdoc/layout.html

▼リスト4　layout.htmlの内容

```
{#
    sphinxdoc/layout.html
    ~~~~~~~~~~~~~~~~~~~~


    Sphinx layout template for the sphinxdoc theme.

    :copyright: Copyright 2007-2015 by the Sphinx team, see AUTHORS.
    :license: BSD, see LICENSE for details.
#}
{%- extends "basic/layout.html" %}

{# put the sidebar before the body #}
{% block sidebar1 %}{{ sidebar() }}{% endblock %}
{% block sidebar2 %}{% endblock %}
```

全体の内容はリポジトリ注9の中を参照していただくとして、いくつかポイントとなる部分がわかるように引用します(**リスト5**)。

先頭の複数行コメントの直後にある`@import url("basic.css");`という記述は、継承元のテーマのスタイルシートを読み込むことを示しています。このようにすることで継承元のテーマがすでに定義しているスタイルをそのまま利用できます。ただしsphinxdocが継承しているbasicテーマは、レイアウトや色などのスタイルがほとんど定義されていません。basicを直接継承元のテーマとして使う場合は、必要なスタイル定義を自分で記述する必要があります。

初めてテーマを作成するときや既存のテーマに少しだけ変更を加えるだけのテーマを作成するときなどは、既存のテーマのテンプレートやスタイルシートをコピーして必要な部分だけを書き換えるようにすると良いでしょう。

ところで、このスタイルシートに記述されている定義の中に`{{`と`}}`で囲われた部分があるのにお気づきでしょうか(**リスト5-**①)。これはもう1つのポイントである、静的テンプレートを利用する場合の記述方法です。

静的テンプレートとは、テーマのディレクトリ内のstaticディレクトリ(またはユーザ指定の静的ファイルパス)にある、拡張子の末尾に`_t`

注9) https://github.com/sphinx-doc/sphinx/blob/master/sphinx/themes/sphinxdoc/static/sphinxdoc.css_t

▼リスト5　sphinxdoc.css_tの内容(一部抜粋)

```
/*
 * sphinxdoc.css_t
 * ~~~~~~~~~~~~~~~
 (..略..)
 */

@import url("basic.css");

/* -- page layout ----------------------------------------------------------- */

body {
    font-family: 'Lucida Grande', 'Lucida Sans Unicode', 'Geneva',
                 'Verdana', sans-serif;
    font-size: 14px;
    letter-spacing: -0.01em;
    line-height: 150%;
    text-align: center;
    background-color: #BFD1D4;
    color: black;
    padding: 0;
    border: 1px solid #aaa;

    margin: 0px 80px 0px 80px;
    min-width: 740px;
}

div.document {
    background-color: white;
    text-align: left;
    background-image: url(contents.png);
    background-repeat: repeat-x;
}

div.bodywrapper {
    margin: 0 {{ theme_sidebarwidth|toint + 10 }}px 0 0;  ←①
    border-right: 1px solid #ccc;
}
 (..以下略..)
```

と付いているファイルをテンプレートエンジンで処理する機能のことです。**リスト5-**①のように記述すると、Sphinxがドキュメントを生成する際に、theme.confやconf.pyで定義した変数の値をCSSやJavaScriptなどの静的ファイルに埋め込めます。たとえばサイドバーの幅や位置、ヘッダ／フッタの高さ、文字の大きさなどの値や、フォントファミリーの指定などをconf.pyから変更できるようになります。

なお、静的テンプレートで利用できる変数名は前節のtheme.confのところで触れたように、[options]のセクションで定義したものと、継承元のテーマで定義されているオプション名が対象となります。

では、このファイルをsd_theme/staticの下にsd_theme.css_tという名前に変更して配置します。

HTMLテーマのパス

ここまでででHTMLテーマを構成するために必要なものについて触れてきました。今の時点でsd_themeの構成は、先に示した**図5**の構造のようになっていると思います。

説明の便宜上、テンプレート(layout.html)とスタイルシート(sd_theme.css_t)は組み込みのテーマから拝借する形になりましたが、これらを自作した場合でも構成は上記と同じだと考えてかまいません。

さて、このテーマで実際にドキュメントのビルドをしてみましょう。自作のHTMLテーマをSphinxから参照できるようにする方法はいくつかありますが、もっともシンプルな方法はsd_themeのディレクトリを、このHTMLテーマを利用したいドキュメントプロジェクト内のconf.pyと同じディレクトリに配置する方法です(**図6**)。

このように配置した状態でconf.pyには次のように指定をします。

```
html_theme = 'sd_theme'
html_theme_path = ["."]
```

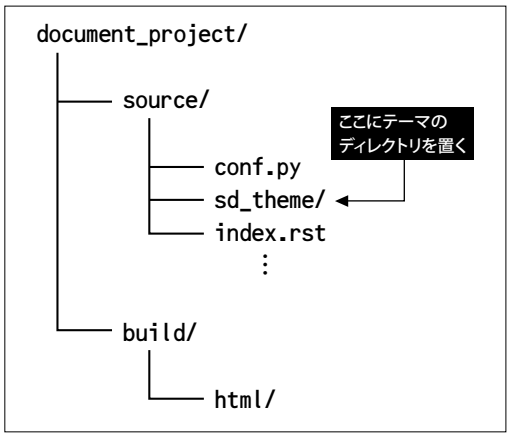
このときhtml_theme_pathには**conf.pyからの相対パス**を指定します。

設定ができたらいよいよドキュメントをビルドしてみましょう。「テーマが参照できない」といったエラーが出ることなくHTMLドキュメント(**図7**)が生成できれば、自作HTMLテーマの適用は無事完了です。

作成したテーマを配布する

作成したHTMLテーマを自分1人で使うのならディレクトリのまま保持しておいて、適用したいドキュメントプロジェクトと一緒にリポジトリにコミットしてしまえば良いでしょう。テーマのCSSに変更を加えたとしても、ドキュメントと一緒にソースコードが管理されているので安心です。

▼図6　自作HTMLテーマのディレクトリの配置場所



▼図7　sd_themeを適用したドキュメント

ですが、作成したHTMLテーマをほかの人と共有するには、どうしたらいいでしょうか。もしかしたらあなたの作成したHTMLテーマを気に入った人が、自分も使いたいと申し出てくるかもしれません。

手っ取り早くHTMLテーマを共有する方法として、HTMLテーマのディレクトリの中身をそのままzipアーカイブファイルにまとめて提供する方法があります。sd_themeのテーマを例にすると、まずsd_themeディレクトリの配下をzipコマンドでアーカイブ化します(**図8**)。zipアーカイブが作成できたら中身を確認してみます(**図9**)。zipアーカイブ化するときのポイントとしては、HTMLテーマのディレクトリをzipアーカイブに含めないことです。

このようにしてzipアーカイブ化したHTMLテーマは、ディレクトリのままのときと同じように扱えます。前節の**図6**と同じディレクトリ構成で例を示すと**図10**のようになります。

このときconf.pyの記述は、変更の必要はありません。ディレクトリのままで配置していたときと同様に、Sphinxが読み込んでくれます。

zipアーカイブでまとめておけば1ファイルを受け渡しするだけで利用できるので、ディレクトリのままの状態よりも管理が容易になります。同じチーム内でテーマを共有するなどの用途であれば、zipアーカイブで配布するだけでも必要十分でしょう。

このほかにPythonパッケージを作成して配布するという方法もあります。zipアーカイブのままでもPyPIに登録することはできますが、Pythonパッケージを作成するとpipでインストールができるので配布がより簡単になります。Pythonパッケージの作成方法はSphinx-Users.jpの参考手順[注10]を参照してください。

Sphinx-Users.jpのメンバーには筆者を含め、PyPIにSphinxの自作テーマを公開している人がいます。HTMLテーマのパッケージングに興味がありましたら、Sphinx-Users.jpのメーリングリスト[注11]で質問をしてみてください。

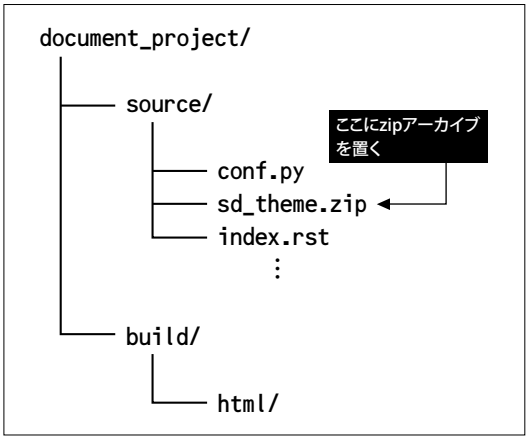
▼図8　sd_themeディレクトリをzipアーカイブファイルにする

```
$ cd path/to/sd_theme
$ zip -r sd_theme.zip .
  adding: layout.html (deflated 41%)
  adding: static/ (stored 0%)
  adding: static/sd_theme.css_t (deflated 73%)
  adding: theme.conf (deflated 19%)
```

▼図9　zipアーカイブファイルの中身を確認する

```
$ unzip -l sd_theme.zip
Archive:  sd_theme.zip
  Length     Date   Time    Name
 --------    ----   ----    ----
      384  09-08-15 00:37   layout.html
        0  09-08-15 00:11   static/
     6266  09-08-15 00:37   static/sd_theme.css_t
      124  09-08-15 00:10   theme.conf
 --------                   -------
     6774                   4 files
```

▼図10　自作HTMLテーマのzipアーカイブの配置場所



次回予告

今回はHTMLテーマの作成方法、配布方法について取り上げました。次回はわかりやすい文書を作るうえでは欠かせない画像について、Sphinxで便利に扱う方法を紹介します。SD

注10) http://sphinx-users.jp/cookbook/makingwebsite/application.html#python

注11) http://sphinx-users.jp/howtojoin.html#mailinglist

COLUMN

PyCon.MY 2015とPyCon.KR 2015

Author 清水川 貴之

本連載執筆陣の1人、清水川です。

2015年8月21～23日にかけてマレーシアのクアラルンプールでPyCon Malaysia 2015（以下、PyCon.MY）[注A]が開催されました。また、翌週の8月29～30日にかけて韓国のソウルでPyCon Korea 2015（以下、PyCon.KR）[注B]が開催されました。

筆者は、両方のカンファレンスに参加し、Sphinxの発表を行ってきました。各イベントについて、gihyo.jpに参加レポートを掲載していますので、ぜひご参照ください[注C]。

本コラムでは、それぞれの地域においてSphinxがどのように利用されているのか、また、Sphinxに対するカンファレンス参加者の様子を紹介します。

■マレーシアでのSphinxへの反応

筆者は、PyCon.MYで2つの発表を行いました。「Easy contributable internationalization process with Sphinx」ではSphinxのドキュメント翻訳サポート機能（i18n）について紹介しました。「Sphinx autodoc: automated API documentation」では、APIドキュメントの自動生成（autodoc）について紹介しました。

マレーシアではほとんどのPython技術者が英語を扱えるということもあり、i18n機能の発表ではあまり参加者が集まりませんでした。もう一方のautodocの発表には、40名近くが参加してくれました。

全体としては、ドキュメンテーションには興味があるものの、書くことはあまり一般的ではないようで、Sphinxの利用例もあまりないのだと感じました。今回の発表が、ドキュメンテーションを始めるきっかけになってくれるとうれしいですね。

■韓国でのSphinxへの反応

PyCon.KRでは、autodocについて紹介しました。韓国ではPythonへの興味が高まりつつあり、カンファレンス全体で約650名の参加者があり、Sphinxの発表には100名近くが参加してくれました。

発表後のQ&Aでは、紹介したSphinxの機能についてだけでなく、とても多くの質問をいただきました。これは、2014年末に翻訳出版された「The Hacker's Guide to Python」[注D]という本の影響が大きいそうです。この本で、ドキュメンテーションの方法としてSphinxが紹介されており、ちょうど関心が高まったタイミングだったようです。

発表ではSphinxのautodoc機能を使うと、Pythonプログラムの充実したドキュメントを手軽に作れることを紹介しました。Q&Aでは、その機能をさらに活用するために、autodocで生成されたドキュメントをSwagger[注E]などのREST API生成サービスと連携させたいという質問がありました。

Swaggerは、API仕様ドキュメントからREST APIのエンドポイントを自動生成して、APIを利用者向けのクライアントライブラリやスタブAPIを提供するサービスです。このサービスを利用すれば、実際のAPIがリリース前でも、クライアントはアプリケーションの実装や検証を行えます。

Sphinx公式ではSwagger連携機能を提供していませんが、第三者によるSphinx拡張機能が公開されているようです。このように、APIドキュメントを自動生成するだけでなく、その先の連携まで考えた質問があったことにはとても驚きました。

ほかにも、「日本でSphinxを使って出版した事例があるか[注F]、その際にSphinxを使うメリットは何か？」「Sphinxなどのオープンソースの開発にはどうやったら参加できるのか？」といった質問がありました。また発表以外の時間でも、韓国におけるドキュメンテーション事情など、多くの議論を交わしました。

詳しくは、先述の海外PyCon発表修行レポート2015の第6回で紹介しています。そちらもご参照ください。

注A）PyCon.MY 2015　http://www.pycon.my/

注B）PyCon.KR 2015　http://www.pycon.kr/2015/

注C）海外PyCon発表修行レポート2015
http://gihyo.jp/news/report/01/overseas-pycon-presentation-training-2015/

注D）『The Hacker's Guide to Python』
https://julien.danjou.info/books/the-hacker-guide-to-python

注E）http://swagger.io/

注F）Sphinxを使って出版した事例
http://www.sphinx-users.jp/book_usage.html

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9回 Mackerelのアーキテクチャを知る

今回は番外編、Mackerel内部の話題です。ユーザとエージェントという性質の異なる2種類のアクセスを受けるMackerelならではのサーバ構成を紹介し、データストアを構成するPostgreSQL・Redis・Graphite、それぞれの選定理由を解説します。

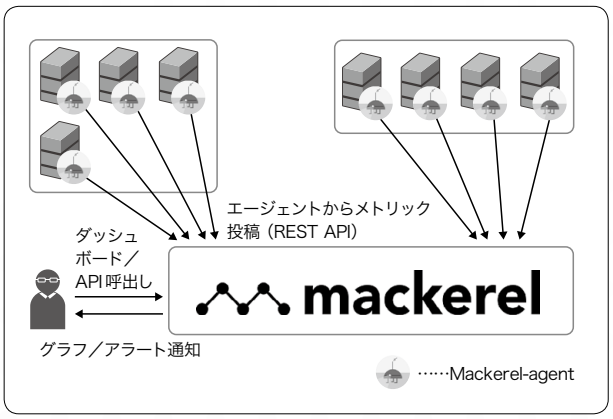
本連載ではこれまでMackerelを用いたサーバ管理のノウハウを紹介してきました。今回は番外編として、Mackerelそのもののシステムの構成と運用事情を紹介します。

Mackerelの内部システム概要

本連載の第1回で、Mackerelの基礎的なアーキテクチャについて紹介しました。**図1**のように、Mackerelのアーキテクチャはいわゆる Push 型であり、各サーバのmackerel-agentがメトリック値をMackerelのサーバへ定期的に送信します。Mackerelのシステムの負荷の大部分は、このmackerel-agentからのメトリック送信によるものです。これはMackerel特有の性質であり、いわゆるピークタイムが存在せず、常に一定量以上の書き込み要求が送られてきます。

一方で、ユーザのアクセスは常時一定というわけではなく人間の活動時間に左右されます。さらに、サーバメトリックのグラフ表示など、mackerel-agentによるアクセスとはワークロードが大きく異なります。

▼図1 Mackerelのアーキテクチャ



Mackerelサーバ構成

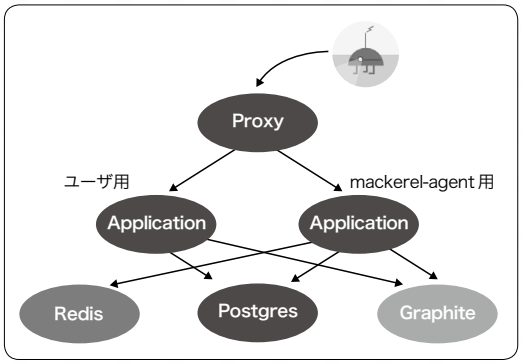
前述のように、Mackerelを運用するうえではmackerel-agentによるアクセスとユーザのアクセスを切り分けて考えており、サーバの構成を工夫しています（**図2**）。

基本はリバースプロキシ・アプリケーション・データベースの3層構成であり、通常のWebサービスと同様です。ただし、データストアとして、PostgreSQL・Redis・Graphite[注1]を併用している点が特殊です。とくに時系列データベース[1]としてGraphiteを使用しているのは、少なくとも国内ではあまり例がないと思います。アプリケーションの開発言語がScalaであることも特徴的と言えるでしょう。Mackerelで使用している各ミドルウェアについては後述します。

ユーザのアクセスとmackerel-agentのアクセスのワークロードの違いを考慮して、アプリケーションサーバのロー

注1） URL http://graphite.readthedocs.org

▼図2　Mackerelのサーバ構成



ルを2つに分割しています。リバースプロキシの振り分けにより、ユーザとmackerel-agentのアクセスをそれぞれ別のアプリケーションサーバ群に分割します。ユーザ用アプリケーションサーバとmackerel-agent用アプリケーションサーバでは細かな違いがあります。

まず、mackerel-agentのアクセスではRedisを使用することがありますが、ユーザのアクセスではRedisを使用しません。これにより、仮にRedisに障害が発生したとしても、ユーザ用のアプリケーションサーバは影響を受けないため、ユーザの画面操作に影響を与えません。

アプリケーションからGraphiteへのリクエストもユーザ用とmackerel-agent用で処理を分けています。ユーザのアクセスは、グラフ表示のためにGraphiteのデータ読み出し用のデーモンである「graphite-web」に投げられ、mackerel-agentのアクセスは時系列データの保存のためにGraphiteのデータ書き込み用のデーモンである「carbon」に投げられます。したがって、graphite-webに障害が起きてもmackerel-agentのアクセスには影響がなく、carbonに障害が起きてもユーザのアクセスには影響がありません。Graphiteのアーキテクチャについては後述します。

さらに、mackerel-agentのアクセスはユーザのアクセスに比べて多いため、アプリケーションサーバの個数やハードウェア性能を大きめにしています。

そのほか、データベース接続のためのコネクションプーリング[2]のパラメータをmackerel-agent用とユーザ用でそれぞれ異なる設定にするなど、ロールごとにパラメータをチューニングしています。

このように、ワークロードごとにロールを分割することにより、問題が発生したときの影響範囲を小さくし、個別のチューニングがしやすくなるようにしています。より一般的なWebアプリケーションの構成については、筆者のブログで解説しています[3]。

Mackerelで使用しているミドルウェア

ここまでMackerelの全体的なサーバ構成を簡単に紹介しました。次に、Mackerelで使用しているミドルウェアについてそれぞれ詳しくみていきます。

PostgreSQL

はてなでは既存のサービスのほとんどで、RDBMSとしてMySQLを採用してきました。MackerelであえてPostgreSQLを採用したのには理由があります。

Mackerelはサーバ管理のためのサービスであるため、はてなの既存のサービスと比較してデータ構造が非常に複雑になり得ます。たとえば、ホストという概念に結びつくデータはたくさんあります。ホスト名やIPアドレスはもちろん、インターフェース、デバイス、ハードウェア情報、サービス、ロール、オーガニゼーション、メトリック、各種監視の設定、アラートなどです。このように静的にスキーマを決定できるデータに加えて、ユーザの運用事情にあわせて柔軟に情報を持たせることがあるのです。たとえば、AWSのEC2におけるタグのような、ホストやロールに対して任意のキーバリュー値を設定し、API経由で利用するといったものです。

そこで、JSON型を備えたPostgreSQLに着目しました。PostgreSQLのJSON型はJSONのキーに対してインデックスを作成できるという高機能ぶりです。そのほか、NOTIFY文を

用いたPublish/subscribe型のメッセージ通信など、MySQLにはないNoSQL的な機能を備えています。

PostgreSQLは接続を受け付けるごとに接続用のプロセスをforkするため、接続コストが高いと言われています。ですので、コネクションプーリングのような、接続を永続化するようなデータベースクライアントと併用するのが望ましいでしょう。

JavaやScalaのようなJVMベースの言語の場合、BoneCP[注2]やHikariCP[注3]のようなJDBCのコネクションプール実装が使えるのが良いところです。

Perlのようなマルチプロセス型のWebサーバであれば、pgpool-II[注4]やPgBouncer[注5]のようなプロキシ型のコネクションプール実装[2]を併用する必要があります。

Graphite

Graphiteは、タイムスタンプ、メトリック名、メトリック値という3つの値の組を複数個受け取ってグラフ化するというシンプルな機能を、ネットワークサービスとして提供しています。

Graphiteはデータの書き込みと読み出しでインターフェースが異なります。TCPベースの独自のテキストプロトコルで時系列データを書き込み、書き込んだ時系列データをHTTPで取得します。コマンドラインからデータを書き込む場合、次のようなコマンドになります。

```
$ PORT=2003
$ SERVER=graphite.your.org
$ echo "local.random.diceroll 4 `date ➥
+%s`" | nc -q0 ${SERVER} ${PORT}
```

時系列データはグラフ画像もしくはJSON形式で取得できます。単に時系列データを取得できるだけでなく、時間範囲指定やアグリゲーションなどの機能を備えています。詳細は公式ドキュメント[注6]を参照してください。

MackerelのようにクライアントサイドJavaScriptでグラフを描画する場合、アプリケーションサーバがGraphiteからJSON形式でデータを取得します。

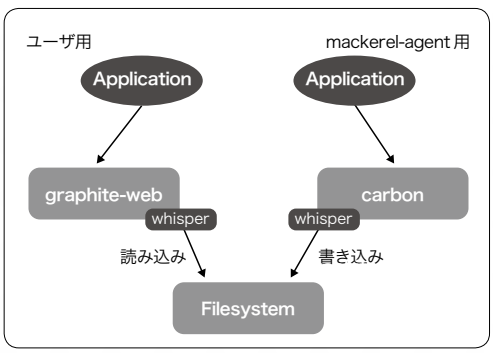
Graphiteはおもに次の3つのコンポーネントで構成されています。

- whisper：ラウンドロビンデータベースファイルを作成・更新するためのライブラリ
- carbon：書き込み要求を受け付けるためのデーモン
- graphite-web：読み込み要求を受け付けるためのWebアプリケーション

図3に各コンポーネントとMackerelのアプリケーションの関係を示します。メトリック値を保存するときはアプリケーションからcarbonに向けて前述の形式で書き込み要求を投げて、メトリック値を取得するときはアプリケーションからgraphite-webに読み出し要求を投げることになります。

carbonとgraphite-webが分かれているのは、サーバモニタリングのような書き込み要求の数が圧倒的に多く、読み出し要求の数は比較的少ないというユースケースに合わせて、それぞれ別々のデーモンで処理するためだと筆者は考えています。実際、carbonは大量の書き込み要

▼図3　Graphiteのアーキテクチャ



注2）URL https://github.com/wwadge/bonecp
注3）URL https://github.com/brettwooldridge/HikariCP
注4）URL http://www.pgpool.net/mediawiki/jp/index.php
注5）URL https://wiki.postgresql.org/wiki/PgBouncer
注6）URL http://graphite.readthedocs.org/en/latest/render_api.html

求を想定した作りになっています。

carbonに書き込み要求を投げると1つのメトリックにつき、1つのwhisperファイルが作成されます。したがって、メトリックの数だけファイルが作成され、大量のファイルをファイルシステム上で管理することになります。実に素朴なしくみですが、素朴な分だけ内部を把握しやすく、問題が起きても運用でカバーしやすいといえます。

Redis

RedisはWeb業界で一般的に使用されているオンメモリのデータストアです。Macekrelではおもに一時的な監視用途のデータを格納するために使用しています。

Redisは高速ですが、オンメモリ型であるため格納するデータサイズを小さく抑えるのが楽に運用するコツです。半永久的に保存しなければならないかつ、サイズがどんどん増えていくデータを格納するにはあまり向いていないと筆者は考えています。

サーバの監視条件というのは基本的に直近のデータに対して適用されるものであるため、古いデータを削除できるという性質を考慮して、Redisに格納することに決めました。

Mackerelでは受動監視、メトリック監視、外形監視などの各種監視機能がありますが、もっとも負荷が支配的なのがメトリック監視です。メトリック監視をするために、監視条件が設定されたメトリックすべてをRedisに格納する必要があり、mackerel-agentから投稿されるメトリックの多くがRedisに書き込まれるためです。

Redisは高速ですが、1スレッドで動作するためマルチコアスケールしないという欠点があります。サービスが成長するにつれて、遠からず1つのCPUコアでさばき切れなくなります。今のところクロックレートの高いCPUを載せたマシン上で動作させることで問題を先送りにしています。

また、Linuxカーネルの機能であるRPS/RFS[注7]を用いて、ソフト割り込みを受け付けるカーネルスレッドとRedisが動作するユーザスレッドを別々のCPUコアに割り当て、できるだけ複数のコアに処理を分散させています[4]。

今後スケーラブルなしくみを作るためにRedis 3.0から追加されたRedis Cluster[注8]によるシャーディングも検討しています。

まとめ

今回はオペレーションエンジニアからみたMackerelの内部システムの構成と運用事情を簡単に紹介しました。Mackerelは国内のWeb企業において、サービスの性質や使用しているミドルウェアが独特なものとなっていますが、通常のWebアプリケーションと基本部分は同じだと考えています。アプリケーション開発寄りの技術については、Mackerelの公式サイト内「Mackerelで採用している技術一覧とその紹介」[注9]を参照してください。

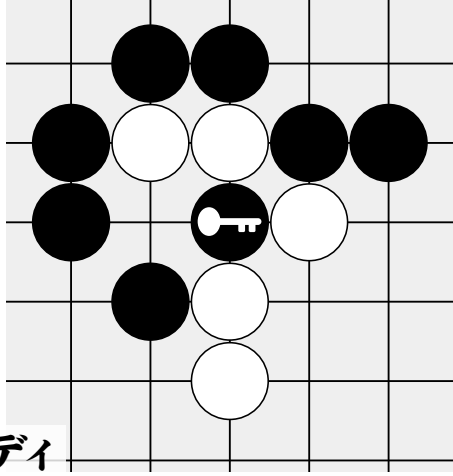
なお、今回紹介した各々の技術の詳細は筆者のブログで詳しく扱っているので、興味のある方は次の参考文献を参照してください。SD

● 参考文献（筆者ブログ：http://yuuki.hatenablog.com）
[1]Mackerelを支える時系列データベース技術
http://yuuki.hatenablog.com/entry/high-performance-graphite
[2]Webシステムにおけるデータベース接続アーキテクチャ概論
http://yuuki.hatenablog.com/entry/architecture-of-database-connection
[3]はてなで大規模サービスのインフラを学んだ
http://yuuki.hatenablog.com/entry/large-scale-infrastructure
[4]Linuxでロードバランサやキャッシュサーバをマルチコアスケールさせるためのカーネルチューニング
http://yuuki.hatenablog.com/entry/linux-networkstack-tuning-rfs

注7）URL https://www.kernel.org/doc/Documentation/networking/scaling.txt
注8）URL http://redis.io/topics/cluster-spec
注9）URL http://developer.hatenastaff.com/entry/mackerel-overview

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

【第二六回】パスワードクラックのケーススタディ

最近、Ashley Madisonという Webサイトで、パスワードや個人情報などの大量流出事件がありました。いろいろな観点からこの事件は話題になっていますが、筆者が注目しているのはハッシュ化されていたパスワードが解読されていることです。調べてみると、解読の背景にはアルゴリズムの危殆化、システム設計／メンテナンスなど、これまでの懸念がそのまま表面化したような状況が見て取れます。今回はパスワードはシステムとして、どうすべきなのかを考えてみます。

Ashley Madison 顧客情報流出事件

AshleyMadison.comはカナダのAvid Life Media社（以下、ALM社）が運営している一種のSNSサイトで[注1]、登録利用者は約3,700万人です。

自らを“The Impact Team”と呼ぶグループが、どのような方法を用いたかはわかりませんが、Ashley Madison.comが保持していた顧客情報データベース、サイト上にユーザがアップロードした各種データなどを入手しました。AshleyMadison.comを閉鎖しないと内容を公開すると脅迫し、後に個人情報も含むデータを公開しました。執筆時（2015年9月15日）までの事件の経過は次のとおりです。

①2015年7月15日、セキュリティーブログkrebsonsecurity.comが、AshleyMadison.comに不正アクセスがあったことを報じる[注2]
②同年7月20日、ALM社がシステムに不正アクセスされたことを認めるアナウンスを流す[注3]
③同年8月18日、The Impact Teamが顧客情報などのデータ10GB分を、Torネットワークを使って流出させる。続く8月20日に、企業に関する内部データ20GB分を流出させる[注4]

「AshleyMadison.comから300GBを越えるデータを入手した」とThe Impact Teamは主張しています。それらは、ユーザアカウントに関する個人情報、パスワード認証のためのデータ、ユーザの写真、文章、チャットの内容、従業員のメールとのことです。これを見る限り、まるで同サイトの内部システムで保有する管理データがまるごと流出したかのように見えます。

8月18日に流出したのは約3,600万人分の顧客情報で、その内容はユーザ名、氏名、メールアドレス、パスワードのハッシュ値、クレジットカード（の一部）、住所、電話番号、960万件分のクレジットカード利用記録などでした[注5]。

このうち1,100万件分のパスワードが解読されました（本稿執筆時点）。これはThe Impact Teamではなく、Cynosure Primeという別のチームが、流出した情報をもとにパスワードクラッキングを行いました。現在も解読を進めており、数字は増えていますので、最終的にどれくらいの割合でパスワードが

注1） そのサイトのサービス内容は本論とは関係ないので、とくには言及しません。
注2） http://krebsonsecurity.com/2015/07/online-cheating-site-ashleymadison-hacked/
注3） http://www.prnewswire.com/news-releases/statement-from-avid-life-media-inc-300115394.html?tc=eml_cleartime
注4） http://www.wired.com/2015/08/happened-hackers-posted-stolen-ashley-madison-data/
注5） http://thehackernews.com/2015/08/ashley-madison-accounts-leaked-online.html

解読できるかは現状ではわかりません。なお、Cynosure Primeは解読したパスワードの統計情報は公開していますが、個々のユーザのパスワードは公開していません。

Cynosure Primeは、一緒に流出したAshley Madison.comのシステムのソースコードを解析し、そこに大きなセキュリティの穴を見つけ、そこからパスワードを発見するという手順を踏んでいます。Cynosure Primeはブログで手法を解説しています[注6]が、セキュリティ的にも非常に興味深く、ここから我々はセキュリティについて学ぶべき点があるのではないかと筆者は考えています。

パスワードのしくみ

一般的なパスワードの考え方やしくみは本連載第1回で取り上げていますので、ここでは具体的なパスワードシステムのしくみについてのみ考えます。

ここでのパスワードは「ユーザのみが知っている秘密の文字列」と定義します。文字列は英数字と記号からなり、英字は大文字と小文字が使えるとします。パスワードの最小文字数は、ここでは規定しないこととします。

```
ここでのパスワード例
JQeo1Hroyi8J
MyPa55W0rds
#$%2912@2Ac
```

パスワードとする文字列は、暗号学的ハッシュ関数(以下、ハッシュ関数)を使い、ハッシュ値を取ります。そのとき、パスワード文字列にソルト(塩)と呼ばれる任意の値を与えます(**リスト1**)。これは、パスワード文字列が同じものであれば同じハッシュ値になるのを避けるために加えています。

システムはソルトとハッシュ値を記録しておき、ユーザが入力したパスワード文字列にソルトを加えハッシュ値を都度計算し、それが記録してあるハッシュ値と同じかどうかを比較します。同じであれば、同じパスワードが入力されたということになります。

◆リスト1　SHA-256によるハッシュ値

```
パスワード文字列“abcdef”の場合
  abcdef:  ae0666f1（略）bd366a57
“abcdef”にソルト文字列“1000”を加える
  abcdef1000:  0e6dd6d2（略）85029da0
“abcdef”にソルト文字列“0001”を加える
  abcdef0001:  8f8e2ca9（略）24d0dde1
```

ハッシュ関数の危殆化

どんなハッシュ関数が一般的か、RFC 5246を参考に、TLS 1.2で使用できるハッシュ関数を確認してみます。

MD5、SHA-1、SHA-224、SHA-256、SHA-384、SHA-512

また、最新のOpenSSL-1.0.2dでは、次のハッシュ関数が使えます。

MD4、MD5、MDC2、RIPEMD160、SHA、SHA-1、SHA-224、SHA-256、SHA-384、SHA-512、Whirlpool

OpenSSLでは、すでに安全でないものも含め、たくさんのハッシュ関数が用意されています。たとえば、MD4やMD5はすでに安全ではないので使うべきではありません。

この中でSHA-224、SHA-256、SHA-384、SHA-512、(SHA-1)以外はNIST(National Institute of Standards and Technology)の推奨する安全なハッシュ関数(Secure Hash Function)には挙げられていません[注7]。

当初、NISTは2010年以降にはSHA-1も規格から外す予定でしたが、SHA-2ファミリ(SHA-224/256/384/512)への移行が進まず、規格としてはまだ取り下げていません。しかし、SHA-1も新しくシステムを作る場合には使うべきではありません。

Microsoft社はSHA-1を使った証明書は2015年

注6) http://cynosureprime.blogspot.co.uk/2015/09/how-we-cracked-millions-of-ashley.html
注7) http://csrc.nist.gov/groups/ST/toolkit/secure_hashing.html

いっぱいで発行を中止し、2017年1月1日以降はSHA-1を受け付けないと宣言しています。Google社のChromeも、段階的に警告を行っており、2017年以降はSHA-1（の証明書）を使うHTTPSサーバは正規のものとして扱わないことになります。Mozillaも似たようなスケジュールです。ちょうど本年の2015年は、SHA-1からSHA-2ファミリにスムーズに移行する最後の準備期間と言える年です。

MD5を使うのは致命的

2004年の攻撃方法の発表（コラム「MD5が破られた瞬間」参照）から11年が過ぎ、現在は「md5(m) = md5(n)」であるmとnを見つけるコリジョン攻撃（Collision Attack）は、技術的に何ら問題なく可能で、MD5のデジタルフィンガープリント（電子指紋）の偽造は容易に行えます。

さらにGPU「NVIDIA Geforce 8800」を使ったMD5のプログラムが2009年にはすでにあり、約2億回/秒の計算が可能とのことです。さて、2015年、最新のGPUを大量に使うとどれくらいの速度が出るのでしょうか？　すでに「NVIDIA Geforce GTX 980」を同時に8台で動かして試した人がいるようです。その資料を読むと、おおよそ793億回/秒の計算が可能とのことです[注8]。ブルートフォースを考えれば極めて驚異的な計算能力と言えます。

さらに最近では、オンラインでMD5をクラックできるmd5crack.com[注9]というサイトまであります。ここ以外にも、いろいろなサイトがあります。これはMD5のハッシュ値を入力すると元の値を答えてくれるというものです。さて本当かどうか試してみます。次のように“ABCDEFG”という文字列をMD5で処理したハッシュ値をmd5crack.comに入れてクラックしてみました。

MD5の値
```
% echo -n 'ABCDEFG' | openssl dgst -md5
(stdin)= bb747b3df3130fe1ca4afa93fb7d97c9
```

md5crack.comには**図1**のように表示されました。これは簡単過ぎたようです。

たとえばパスワードを“p@ssw0rd”としているようなレベルであれば、辞書登録レベルですので一瞬で見つかります。

MD5の値
```
% echo -n 'p@ssw0rd' | openssl dgst -md5
(stdin)= 0f359740bd1cda994f8b55330c86d845
```

クラック結果は**図2**のとおりです。

なお、この手のサイトに入力した文字列は辞書に記録されるので、それ以降、その文字列はパスワードとしては使ってはいけません。

入力値とハッシュ値の対となるテーブルはRainbow Tableと呼ぶ手法で圧縮できます。しかし今日は、ビックデータに象徴される、ストレージも含むハードウェア能力、Key-Valueストアなどのストア技術、ネットワーク分散技術などが、飛躍的に向上しています。パスワード辞書が数億のエントリになっても、わざわざ特殊な技法を使わずにまるごとデータベース化してしまうことでしょう。

◉MD5が破られた瞬間

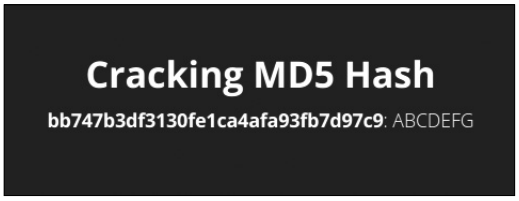
筆者は2004年にMD5が破られた論文[注A]を発表した会場に居合わせました。それまで世界でトップクラスの暗号学者でもMD5を破ることはできていませんでした。これは突然やって来た当時無名だった1人の中国人研究者によって完全に解読されるという劇的な発表でした。発表が終わったと同時に会場全員がスタンディングオベーションで拍手が鳴り止みません。まるでハリウッド映画の1シーンのような様子を今でも覚えています。

当時の様子は、本誌2004年11月号に記事として書いています。それをベースにした内容は自分のサイトに公開していますので、ぜひご覧ください[注B]。

注A）Xiaoyun Wang, Dengguo Feng, Xuejia Lai, Hongbo Yu, “Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD”, August 17, 2004.
https://eprint.iacr.org/2004/199.pdf
注B）http://h2np.net/docs/crypto2004.html

注8）https://gist.github.com/epixoip/c0b92196a33b902ec5f3
注9）http://md5crack.com/

◆図1　“bb747b3df3130fe1ca4afa93fb7d97c9”のクラック結果



◆図2　“0f359740bd1cda994f8b55330c86d845”のクラック結果

Ashley Madisonのパスワードクラック

Cynosure Primeの解説によれば、AshleyMadison.comはPHPを使って書かれています。ログインのパスワードはBCryptによってハッシュ化されて格納されています。

それ以外にもシステム内にこのパスワードを使った認証をかけています。なぜそんなメカニズムが必要だったのかは、断片的な情報からはわかりません。「内部的に使うだけだから複雑さは必要ない」と思ったのか、次の例のように全部を小文字に変換して利用するという奇妙な処理が行われています。

Cynosure Primeの資料より

```
md5(lc($username)."::".lc($pass))
```

確認のために手順を書き上げると次のとおりです。

(1) ユーザ名を全部小文字に変換する
(2) パスワードを全部小文字に変換する
(3) その2つを真ん中に“::”を加えて連結する
(4) その文字列をmd5でハッシュ化する

これは安全でしょうか？　ユーザ名がソルトの値ですので、事前に対応テーブルを作って見つけることは難しいです。そこで、ブルートフォースを考えてみます。パスワードは大文字が小文字に変換されているため、英字のみなら“abc～xyz”の26文字、数字を加えても36文字の組み合わせになります。

先ほどの、MD5を約793億回/秒で処理できるシステムを想定してみます。もし、英字だけの6文字から成るランダムなパスワードを想定した場合、その組み合わせは$26^6 = 308,915,776$となり、ほぼ瞬時にパスワードを発見できます。毎秒約500件程度のパスワードを見つけていくことになります。

世間で出回っている、かなり高い確率でヒットすることが予想されるクラック辞書では15億単語程度のサイズです。先ほどのMD5が約793億回/秒で処理できるシステムを使えば、おおよそ毎秒50件程度のパスワードを発見していくことになります。

パスワードが8文字から成るランダムな英数字でも、$36^8 = 2,821,109,907,456$となり、1件あたり平均時間は約18秒、最悪約36秒で解けてしまいます。

正しいパスワードを見つけるには、このMD5から解読したもとのパスワード（ただし小文字パターン）を使い、BCryptで使われている最終的なパスワードを推定します。

パスワードのパターン

Cynosure Primeの資料を読む限り、AshleyMadison.comではパスワードに最低限の長さや、「必ず大文字、小文字、数字、記号を含まなければならない」といった制約を課していません。そのためユーザは制約なく任意の文字列を選べるようです。しかし、それは良い方向には向かわず、悪いパスワードの使用を許すことになっていることがわかります。

統計データからは1～4文字のパスワードも存在し、5文字パスワードが約85万件、6文字パスワードが約300万件あるのがわかります。極めて弱いパスワードです。Cynosure Primeは6文字パスワードにブルートフォースをかけたと発表しています。ですから、文字にどんなものを選ぼうと、6文字パスワードを使っているユーザはすべてパスワードをクラックされたことになります。

クラックされたパスワードにどんな文字パターンが使われていたかの統計的な結果もグラフから読み

取ることができます。

パスワードに使われる文字

- **小文字のみ　45%**
- **小文字と数字　38%**
- **数字のみ　12%**

良いパスワードは小文字、大文字、数字、記号が含まれていることが求められますが、その条件を満たしたパスワードでクラックされたものは極めてわずかです。3,600万件をターゲットにし、1,500万件を処理し、その中の1,170万件で正しいパスワードを見つけることができましたが、330万件は失敗しています。ユーザの中の約5人に1人は安全なパスワードを選んでいることがわかります。

逆を言えば、パスワードを自由に選ばせた場合、ユーザの78%は弱いパスワードを利用するということです。そのようなユーザですからパスワード管理も期待できないでしょう。そして、この中にはかなりの確率で、ほかのサイトでも同じパスワードを使い回すユーザがいることが容易に想像できます。

サイトのパスワードシステムに求められること

ユーザがパスワードを決めるときに、弱いパスワードを選択させないことが重要です。パスワード変更のとき、次のチェックを行うべきです。

- **長さは最低でも8文字にすること**
- **小文字、大文字、数字、記号を必ず含むこと**
- **パスワード辞書を用意し、選んだパスワードはパスワード辞書に存在しないのを確認すること**

これで大幅にリスクが低減します。とくにパスワードの長さは探査時の計算量に直結しますので、なるべく長くすべきです。

さらに、プログラミングとセキュリティの両方に関して十分な能力を持つ技術者が、コードのレビューをすることで、利用するアルゴリズムや使い方に関して問題がないかをチェックすべきです。AshleyMadison.comの事例で存在していた次の問題点に関しては、簡単に指摘できるはずです。

- **2015年においてもMD5をセキュアハッシュ関数として利用している**
- **わざわざ小文字に変換し、検索空間を意図的に小さくしている**

今回の事例では、ログイン時のパスワードチェックはBCryptで行っており一定の安全性は確保しているにもかかわらず、それ以外のところでなぜかパスワードを使用し、さらにその使い方が間違っているので、そこが穴となり、攻撃側にパスワードを解析させてしまうという誤りを犯しています。

あちらこちらでパスワードを使いまわすのは、そもそも基本設計として正しいとは言えません。付け足し付け足しでサイトのプログラムが拡大していく中で、あまりセキュリティを考えていなかった部分が、そのまま残ってしまったような印象を受けます。

パスワード向けハッシュ関数について求められること

Webアプリーケーション、とくにPHPを使っているサイトではパスワードのハッシュ関数としてBCryptが広く使われています。BCryptのアルゴリズムは1999年のUSENIXで提案されたもので、内部ではブロック暗号アルゴリズムBlowfishを使用しています。Blowfishは、AESが使われる現在において、完全に時代遅れな暗号です。BlowfishはBruce Schneier氏が1993年に作った暗号でDESやIDEAといった過去の暗号との比較はできても、今日の暗号と比較することはできません。内部であれ、このような暗号を使い続けることに筆者は懸念を示したいと思います。

パスワード向けハッシュ関数で標準化されているものは、唯一PBKDF2だけです[注10]。しかし、これも内部でHMAC-SHA1を使っているので、SHA-1の寿命が尽きようとしている今日、いつまで使い続けられるかは不確実です。ほかにも提案されている

注10）"NIST Special Publication 800-132　Recommendation for Password-Based Key　Derivation"
http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf

パスワード向けハッシュ関数が多数あり、十分な安全性検証もなく使われている現状があります。

そのような現状から、2013年に有志らの手によりPassword Hashing Competitionというパスワード向けハッシュ関数のコンペティションが立ち上がりました。24件が受け付けられ、2015年7月20日にその中からArgon2が優勝となります。また、Argon2だけではなくCatena、Lyra2、Makwa、yescryptも優秀なパスワード向けハッシュ関数としてリストアップされています[注11]。

今後、これらのパスワード向けハッシュ関数の利用が増えるでしょう。しかし、これらの活動は必ずしも標準化とは直結していません。

ブロック暗号やハッシュ関数（認証やデジタル署名に使われるもの）などは十分な検証を受けたうえで標準化されています。一方、パスワード向けハッシュ関数だけは重要なコンポーネントにもかかわらず、関心が低いと言わざるを得ません。AESやSHA-2で行ったのと同様に、NISTや標準化団体が責任をもってコンペティションを開催し、安全性や性能などを公開の場で討論し、最後に標準化に結びつける必要を筆者は強く感じています。

ユーザに求められること

筆者は2010年以降、GNU/Linuxが入っているノートPCのログインパスワードは12文字にしています。慣れると意外に苦にならないものです。これは良い悪いではなく慣れの問題だと思います。また、Webサービスで使うパスワードは、自分で考えずにパスワード生成プログラムで12文字パスワードを生成し、それをブラウザに覚えさせています。

今回の流出事例を見てわかるように、自分でどんなにがんばっても登録しているサイトから流出しパスワードを解析される状況では、ランダムで長いパスワードを選ぶ以外、ユーザ側としてできることはあまりありません。また、パスワードが他者に知られたとしても、自分が利用しているほかのサイトに被害が及ばないように、いずれのサイトでもそれぞれ異なるランダムな文字列を選ぶ必要があります。

セキュリティのアドバイスとして“p@ssw0rd”などといった既存の単語を入れ替えた方法を推奨するケースが見受けられますが、パスワード辞書攻撃の範疇（はんちゅう）にほぼ収まってしまい、瞬時に見つけられることを覚悟しなければなりません。

また、言葉遊びで作るパスワードを推奨する向きもありますが、いろいろなバリエーションを考え、それを変化させたパスワードの組み合わせを異なるサイトごとにいくつも覚えられるほどの記憶力が良い人間はどれだけいるのか疑問です。ブラウザに覚えさせたり、パスワード管理のセキュリティツールに任せたりするなら、初めから十分に長いランダムな文字列にしても同じはずです。

今回の記事中でパスワードを見つけるWebサイトを紹介しました。繰り返しになりますが、そのようなサイトで自分のパスワードをチェックしないでください。そこはパスワードを集め、辞書を作るためのサイトでもあるのです。

◆ ◆ ◆

銀行のサイトなどはワンタイムパスワードを使ったり、信頼性の高いサイトは二重認証を使ったりするなど、徐々に単純なパスワード認証ではなくなってきています。サイトごとにパスワードを必要とする時代から、OAuthやOpenIDのように統一したユーザ認証を必要とする時代へ変化しつつあります。

しかし、AshleyMadison.comのようにセキュリティ技術に関心がない、あるいはセキュリティ技術がおざなりなレベルの低いサイトがあるのも事実です。それを前提にWebサービスを使っていく覚悟が必要だということを、今回の件は教えてくれます。

筆者は完全なシステムなど期待しているわけではありませんが、今回の件を振り返るにあたり、ある一定のセキュリティレベルは担保されるような枠組の必要性を感じざるを得ませんでした。SD

注11）Password Hashing Competitionサイト　https://password-hashing.net/
Argon2サイト　https://www.cryptolux.org/index.php/Argon2
“Password Hashing Competition - Survey and Benchmark”（評価論文の1つ）　https://eprint.iacr.org/2015/265.pdf

Red Hat Enterprise Linuxを極める・使いこなすヒント

.SPECs ドット・スペックス

第16回 Planner in JBoss BRMS 6.1（その2）

2015年4月にレッドハットミドルウェア製品であるJBoss BRMS 6.1がリリースされました。Business Rule Management Systemとして歴史を刻んできた同製品には、新たにPlannerと呼ばれる機能が追加されフルサポートが開始されました。前回はサンプルを実行することでPlannerを直感的に理解いただきました。今回はPlannerに関する技術的な話題を取り上げます。

Writer レッドハット（株）サービス事業統括本部
プラットフォームソリューション統括部ソリューションアーキテクト部長　藤田 稜（ふじた りょう）

NP完全問題

前回、Plannerを「Business Resource Planningを扱う機能」と簡略に紹介したのですが、より正確には「NP完全問題を解くプログラム」と言うことができます。“NP”はNon-deterministic Polynomial timeの略で「非決定性多項式時間」と訳されます。「非決定性」とは同一の入力に対して異なる出力が得られること、「多項式」とは問題の入力サイズがnであるとき処理時間がnの多項式になることを意味する計算量理論における用語[注1]です。もっとわかりやすく書けば、「現実的な時間で解を求めることができない問題の集合」が「クラスNP」で、「クラスNPに分類される問題のうち最も難しいもの」が「NP完全問題」です。

矛盾してるような……

「現実的な時間で解を求めることができない」のがNP完全問題なのに「PlannerはNP完全問題を解くプログラム」とは、これいかに？――と思いませんか。

一般にコンピュータプログラムではアルゴリズムを用いてある問題を解きます。たとえば「1から100までの総和を求める」という問題であれば、

(1)ループ処理で1から100まで足す

```
for(i = 1; i <= 100; i++){
    sum += i
}
```

(2)1と100、2と99……という和が101となるものを50組作る

```
sum = (1 + 100) * 100 / 2
```

というアルゴリズムがすぐに思いつきますが、いずれであっても現実的な時間で解を求めることができます。

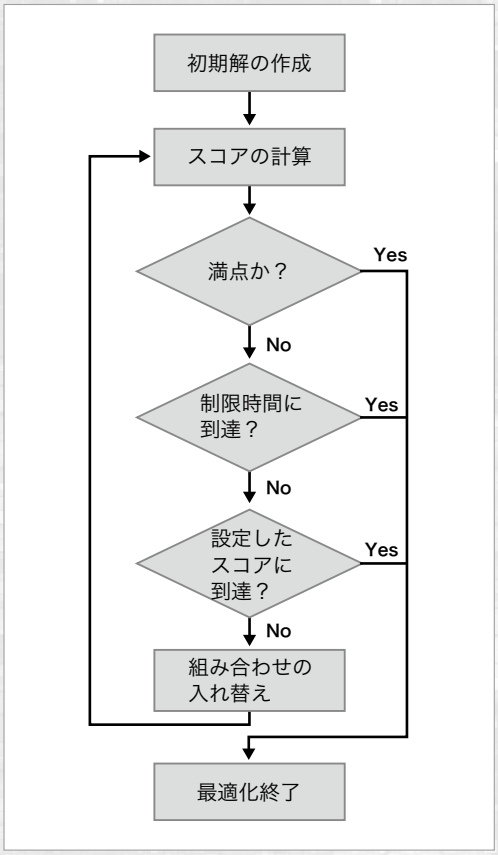
ところがこのようなアルゴリズムを用いてNP完全問題を解くことはできません。そこでPlannerは現実的な時間内に妥協できる解を可能な限り効率の良いアルゴリズムで得ることになり、「最適化をする」というほうがより適切な表現と言えます。

最適化のフロー

では、Plannerはどのように最適化を行うの

注1）バブルソートでは要素数nを比較し並び替える回数が、n(n - 1)/2、nの多項式になる。O記法での計算量は$O(n^2)$。

▼図1 Plannerの概観フロー



でしょうか。キーとなるのはメタヒューリスティックという解法で、要するに経験則、トライ&エラーを繰り返してより良い解を探す、という単純作業です。ここで「より良い解」というのは「ある評価式」によって得られる点数・スコアが良い組み合わせを指します。つまり、

1.初期解(組み合わせ)を作成する
2.組み合わせのスコアを計算し、最良のスコアか否かを判定する
3.組み合わせを入れ替える

という作業の2.と3.の手順を繰り返し(図1)、

(1)満点が得られた[注2]
(2)制限時間内で最高のスコアとなった
(3)事前に設定したスコアに到達した

いずれかの組み合わせを最適な解とします。

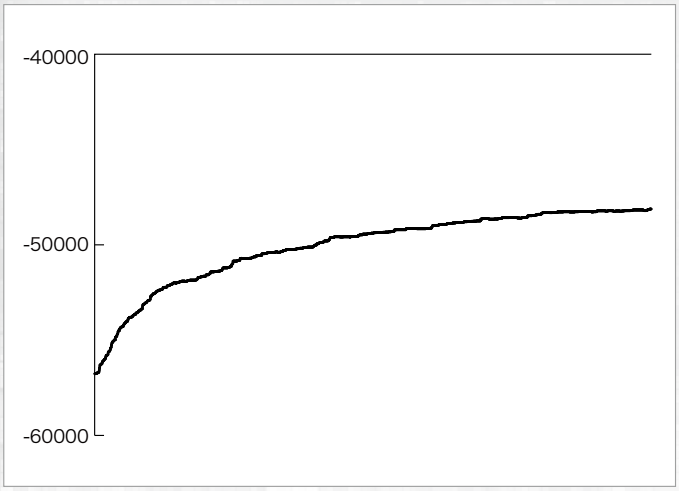
上で「ある評価式」と書きました。Plannerは制約条件をHard、Medium、Softの3種類で設定[注3]し、ある組み合わせが制約条件に違反すると段階に応じて減点方式で評価[注4]します。Hard制約条件はある組み合わせが絶対に満たさないとならないものです。たとえば配車計画であれば生鮮食料品を輸送するトラックは冷蔵機能が必須である[注5]、といった制約条件が該当します。一方、Soft制約条件の例として、配車計画においてドライバーの勤務時間が平準化しているほうが良い、といった条件が挙げられます。PlannerはHard制約条件をすべて満たすように組み合わせを入れ替え、そのあと、Medium制約条件、Soft制約条件を可能な限り満たすことでスコアが改善されていきます(図2)。

アイデアしだいで応用範囲は無限大

Plannerは広範囲のビジネスの課題を扱うことができます。前回のサンプルの中には「まさにこれで悩んでる！」というものもあるかもしれませんし、「似てるけどちょっと違う」というレベルの課題もあるかもしれません。いずれにしても「組み合わせ問題」を解決することが可能なソリューションですのでアイデアしだいで使い途はいろいろです。ぜひ社内のブレストのネタなどに使ってみてください。

注2) 満点が得られたということは完全な解が得られたということであり通常はあり得ない。
注3) Hard制約条件を複数設定し、条件ごとにウェイトを変えることも可能。
注4) 一般的には減点方式、つまりマイナスのスコアが加算される設定で評価することが多い。
注5) ビジネス上の制約条件をHard/Medium/Softに分類するのであって、生鮮食料品が腐ってもかまわないのであれば、Hard制約条件に設定する必要はない。

▼図2　Hospital Bed Planningにおけるスコアの上昇例（Plannerの出力をもとに作成）



Red Hat Tech Exchange

Plannerの話題から離れてRed Hatのイベントを2つほど紹介します。

Red HatのAPACにおける最大の技術系年次イベントがRed Hat Tech Exchangeです（**写真1**）。今年はベトナムのホーチミン市[注6]のシェラトンホテルを会場とし、約300名のエンジニアが9月7日から9日の技術セッション、10日・11日の実技セッションに参加しました。セッションのテーマとして目立ったのはやはりOpenStackとDockerで、次いでIoT（Internet of Things）に関する話題が幅を占めました。Red Hatのパートナー企業のうち約30社からもエンジニアが参加して、熱心にセッションに聞き入る様子が見られました。またガーラ[注7]ディナーではテーブルごとに分かれてのゲームなど、エンジニア同士の交流を深める催しも行われました。

▼写真1　Red Hat Tech Exchange

Red Hat Forum Tokyo

Red Hatの日本における最大のイベントがRed Hat Forum Tokyoです。今年は11月4日に恵比寿のウェスティンホテル東京で開催されます。OpenStackのセッションはもちろんのこと、実際に手を動かしてRed Hatの製品に触れられるハンズオンラボなど、エンジニアの方にも役立つような内容を数多く用意していますので、ぜひ当社Webページ[注8]から申し込んで参加してください。SD

注6）予定ではタイのバンコクだったが爆弾テロの影響で急遽変更に。
注7）galaは特別なお祭りのこと。
注8）http://redhat.events/forum/tokyo/

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第25回 ❖ログ整形にも役立つdate(1)コマンドの小技

◉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。

便利なdate(1)コマンド

アクセスログの解析であるとか、何かしらのデータの解析においては日付データを処理することがしばしばあります。こうした処理にはdate(1)コマンドが使用できるのですが、意外とこのコマンドの使い方が知られていないように思います。今回はdate(1)コマンドの使い方を紹介します。

date(1)コマンドの使い方としてもっともよく使われるのは現在の日時を出力させる機能です。date(1)コマンドに何も引数を指定せずにコマンドを実行すると、次のように現在の日時データが出力されます（日本語ロケールが設定されている場合）。

```
# date ↵
2015 年 9 月 10 日 木曜日 17 時 58 分 37 秒 JST
```

date(1)コマンドはシステムの現在日時を設定することもできます。次のように引数に桁をそろえるようにゼロ埋めした「年月日時分秒」を指定すると、その日時を現在日時に設定します（root権限が必要です）。

```
# date 201509101800 ↵
Thu Sep 10 18:00:00 JST 2015
```

ただし、現在ではこの方法でシステムの日時を設定することはあまりないでしょう。この方法だといきなりシステムの日時を大幅に変えることができてしまうため、のちのち日時に関するデータで不整合が発生する可能性があります（たとえば、古いほうのファイルがなぜか新しい日時データになっていたりとか）。通常はntpd(8)デーモンを実行するなどして自動的に日時を調整して使っていることでしょう。

フォーマットを指定して出力

日時を表示するdate(1)コマンドの基本的な使い方をおさえておきましょう。date(1)コマンドは引数に+からはじまる文字列を指定すると、それは出力する日時のフォーマットとして扱われます。指定できるフォーマットは%からはじまる文字が特別な意味を持っています。指定できる%文字はstrftime(3)のマニュアルにまとまっているので参照ください。

次に、いくつかフォーマットを指定して日時を出力した場合の例を紹介しておきます（**図1～5**）。改行やタブも含めることができますし、出力される形式はロケールの指定によっても変わってきます。

出力する日時のフォーマットが指定できるということは、引数に指定する日時に関しても「年月日時分秒」以外のフォーマットを指定できるということです。ここから以降がとくに今回紹介しておきたい機能です。

日時を更新しないオプション-j

date(1)コマンドに日付を指定すると、その値で

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。

チャーリー・ルートからの手紙

▼図1　フォーマットを指定して出力

```
# date "+%Y-%m-%d %H:%M" ↵
2015-09-10 18:42
```

▼図2　タブと改行を使用

```
# date "+ 日付 :%t%Y-%m-%d%n 時刻 :%t%H:%M" ↵
日付 :      2015-09-10
時刻 :      18:45
```

▼図3　省略形とそうでない場合で出力が変わる例（日本語ロケールだと省略形とそうでない場合の出力が同じなので、LANG=Cにして英語ロケールにしてある）

```
# LANG=C date "+%A %d %B, %Y" ↵
Thursday 10 September, 2015
```

▼図4　データとして保持するための日時出力の例

```
# date "+%Y%m%d%H%M%S" ↵
20150910185110
```

▼図5　UNIX時間での出力

```
# date +%s ↵
1441878714
```

システムの日時データを更新しようとしますが、これを抑制するために-jというオプションが用意されています。-jオプションを指定すると、**図6**のように一見すると動作しているように見えるものの、実際にはシステムの日時は更新しないというように処理が変わります。

このオプションとほかのオプションを組み合わせることで、日付フォーマットの変換や日付データのシフトといった処理ができるようになります。

フォーマットを変換する

出力するフォーマットは+からはじまる文字列で指定しましたが、指定する日時のフォーマットは-fオプションで指定します。実際に動作を見てみるのがわかりやすいでしょう。次のようにコマンドを実行すると、「Thursday 10 September, 2015」を「2015-09-10」へ変換できます[注1]。

```
# LANG=C date -j -f "%A %d %B, %Y" \
"+%Y-%m-%d" \
"Thursday 10 September, 2015" ↵
2015-09-10
```

-fで指定する日時のフォーマットを指定、+からはじまる文字列で出力する日時のフォーマットを指定しています。

もうちょっと具体的な例を見てみましょう。Nginxのログデータは設定ファイルでフォーマットを変更しなければ、「03/Apr/2015:12:31:37」といったような日付が出力されます。date(1)コマンドでこ

▼図6　-jを指定するとdate(1)コマンドは日時の更新を行わない

```
# date ↵
Thu Sep 10 18:04:41 JST 2015
# date -j 201509101800 ↵
Thu Sep 10 18:00:00 JST 2015
# date ↵
Thu Sep 10 18:04:44 JST 2015   ←更新されていない
```

▼図8　日付データが含まれたファイルの例

```
# cat data ↵
11/Sep/2015:12:03:54
11/Sep/2015:12:03:59
11/Sep/2015:12:05:21
11/Sep/2015:12:16:30
11/Sep/2015:12:16:55
11/Sep/2015:12:17:26
11/Sep/2015:12:18:02
11/Sep/2015:12:18:03
11/Sep/2015:12:18:05
13/Sep/2015:17:16:25
```

▼図7　「03/Apr/2015:12:31:37」を「20150403123137」へ変換（Aprのように英語表記なのでLANG=Cで実行しています）

```
# LANG=C date -j -f "%d/%b/%Y:%H:%M:%S" "+%Y%m%d%H%M%S" "03/Apr/2015:12:31:37" ↵
20150403123137
```

注1　LANG=Cを指定しているのはSeptemberといった表記を処理するためです。フォーマット指定が機能していることを示すためにあえて英語表記のほうを変換してみました。

れを「年月日時分秒」に変換するには**図7**のようにdate(1)コマンドを実行します。

ログデータは何行にも渡って出力されますので、複数行まとめて処理する方法を考えてみましょう。まず、**図8**のように日付データだけ入ったファイルを考えます。

どう処理してもよいのですが、たとえば**図9**のように、xargs(1)コマンドを使うとまとめて処理できます。-L1は1行ごと処理せよ、という指定です。-I@で@が対象の1行のデータに置き換わります。この場合、@が日付データに置き換わりますので、個別にdate(1)コマンドが実行されたような動きになります。

Nginxログデータを空白区切りに変換する

もうちょっと発展させてみましょう。実際には**リスト1**のようなデータがNginxのアクセスログデータです。このデータを**リスト2**のようなデータに変換して保存しておきたいケースを考えます。「日時（年月日時分秒）アクセス元IPアドレス HTTPリクエスト HTTPステータスコード」が空白で区切られたデータです。HTTPリクエスト内部の空白は「_」に置き換えてあります。

どう処理してもよいのですが、まずダブルクォーテーションで括られた文字列を単一の文字に変換するコマンドを作ってしまいましょう。ここがクリアできると処理が簡単になります。この手の処理はC

▼**図9　一括して変換する例**

```
# cat data | LANG=C xargs -L1 -I@ date -j -f "%d/%b/%Y:%H:%M:%S" "+%Y%m%d%H%M%S" "@"
20150911120354
20150911120359
20150911120521
20150911121630
20150911121655
20150911121726
20150911121802
20150911121803
20150911121805
20150913171625
```

▼**リスト3　bq2wordコマンドのソースコード**

```
#include <unistd.h>
#include <stdio.h>
#include <sysexits.h>

int
main(int argc, char *argv[])
{
    FILE *fp;
    char *p, buf[BUFSIZ];
    int inbq;

    fp = fopen("/dev/stdin", "r");

    inbq = 0;
    while (NULL != (fgets(buf, BUFSIZ-1, fp))) {
        p = buf;
        while (*p != '\0') {
            switch (*p) {
                case '"':
                    if (inbq)
                        inbq = 0;
                    else
                        inbq = 1;
                    break;
                case ' ':
                    if (inbq)
                        putchar('_');
                    else
                        putchar(' ');
                    break;
                default:
                        putchar(*p);
                    break;
            }
            ++p;
        }
    }
    fclose(fp);
    return (EX_OK);
}
```

▼**リスト1　Nginxのアクセスログの例**

```
192.168.1.39 - - [03/Apr/2015:12:37:26 +0900] "GET /mynavi/20150403-jpnicgtld/typescript.html HTTP/1.1" 200 6544 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.68 Safari/537.36"
```

▼**リスト2　変換後のログデータ（空白区切り）**

```
20150403123726 192.168.1.39 GET_/mynavi/20150403-jpnicgtld/typescript.html_HTTP/1.1 200
```

言語で書いてしまえばよいでしょう。たとえば、bq2word(double quotation to word)というコマンドを**リスト3**のように書きます。

このコマンドを実行すると、**図10**のようにダブルクォーテーションで囲まれた文字列が_でつながった単一の文字列になります。

このコマンドを使って、Nginxのアクセスログのデータを空白区切りの目的とするデータに変換するスクリプトを作成します。これもどう書いてもよいのですが、たとえば**リスト4**のようなスクリプトを用意します。

このスクリプトを実行すると、**図11**のように変換後のデータを得ることができます。

date(1)コマンドの使い方を知らないと日時データの変換にもうちょっと煩雑なコードが必要になりますが、使い方を知っていれば簡単ですね。

また、コマンドを開発したほうが処理が簡単になると判断したら、コマンドを作ったほうがいいと思います。スクリプトや正規表現が難しくなると、のちのメンテナンスが困難なものになりがちです。単機能でシンプルなソースコードで済むなら、Cでコマンドを書いてしまうのはロジカルな選択肢です。

1週間後、1年前といったように日時をシフトする

これに加えてdate(1)コマンドは、特定のデータだけを指定した値に変換したり、または特定の値だけを指定した分だけ未来に、または過去にシフトさせるといったことができます。この機能を使うと、指定した日付の前後1週間のデータが欲しいとか、指定された日付の週の火曜日の午後2時とか、そういった日時データを簡単に取得できるようになります。

この指定のために使うオプションが-vです。**表1～3**のように指定することで、それぞれ値の調整が可能です。

たとえば、先ほど整理したログデータから、指定した日時データの前後10分間のログデータを一緒に表示させたい、といった要望があるとしましょう。**リスト5**のようにdate(1)コマンドで前後10分の日時データを生成して、awk(1)で切り分ければそれを実現できます。実行すると**図12**のようになります。

日付データはさまざまな場面で必要になります。その日付は業界ごとにさまざまだったりします。たとえばはじまりが月第1週の水曜で、締めが翌月第1週の火曜で、といったものです。date(1)コマンドを使うと、こうした日付データをシンプルに生成できます。

使い方アレコレ date(1)コマンド

日時データの処理はログデータやさまざまなデー

▼図10　bq2wordの動作例

```
# cat log | bq2word ↵
192.168.1.39 - - [03/Apr/2015:12:37:26 +0900] GET_/mynavi/20150403-jpnicgtld/typescript.html_↵
HTTP/1.1 200 6544 - Mozilla/5.0_(Macintosh;_Intel_Mac_OS_X_10_10_2)_AppleWebKit/537.36_(KHTML,↵
_like_Gecko)_Chrome/42.0.2311.68_Safari/537.36
```

▼図11　スクリプトの動作例

```
# log2ssv.sh /var/log/nginx-access.log | tail ↵
20150911121726 192.168.1.106 GET_/mynavi/20150911-google/typescript.html_HTTP/1.1 200
20150911121726 192.168.1.106 GET_/mynavi/20150911-google/images/001raw.jpg_HTTP/1.1 304
20150911121802 192.168.1.106 GET_/mynavi/20150911-google/typescript.html_HTTP/1.1 200
20150911121803 192.168.1.106 GET_/mynavi/20150911-google/images/001raw.jpg_HTTP/1.1 304
20150911121805 192.168.1.106 GET_/mynavi/20150911-google/typescript.yd_HTTP/1.1 200
20150913171625 192.168.1.39 GET_/softwaredesign/201511/freebsd/typescript.html_HTTP/1.1 200
20150913171625 192.168.1.39 GET_/favicon.ico_HTTP/1.1 404
20150913173624 192.168.1.39 GET_/softwaredesign/201511/freebsd/typescript.html_HTTP/1.1 200
20150913173753 192.168.1.39 GET_/softwaredesign/201511/freebsd/typescript.html_HTTP/1.1 200
20150913174547 192.168.1.39 GET_/softwaredesign/201511/freebsd/typescript.html_HTTP/1.1 200
```

タ処理で1つのかなめとなる部分ですが、意外とdate(1)コマンドの使い方が知られていないように思います。便利なコマンドですので、ぜひ一度使ってみてください。

なお、date(1)コマンドは実装系で差が大きいコマンドの1つです。フォーマットの指定はある程度共通なのですが、それ以外のオプションの挙動が異なりがちです。Linuxディストリビューションで採用される傾向が強いGNU coreutilsのdate(1)コマンドを使いたい場合は「pkg install coreutils」のようにしてGNU coreutilsをインストールしてください。gdate(1)というコマンドでGNU coreutilsのdate(1)コマンドが使用できるようになります。SD

▼表1　未来へシフト

オプション	内容
-v+数字y	年を指定した年数分、未来へシフト
-v+数字m	月を指定した月数分、未来へシフト
-v+数字w	週を指定した週数分、未来へシフト（1週間ごと未来へシフト）
-v+数字d	日を指定した日数分、未来へシフト
-v+数字H	時を指定した時間分、未来へシフト
-v+数字M	分を指定した分数分、未来へシフト
-v+数字S	秒を指定した秒数分、未来へシフト

▼表2　過去へシフト

オプション	内容
-v-数字y	年を指定した年数分、過去へシフト
-v-数字m	月を指定した月数分、過去へシフト
-v-数字w	週を指定した週数分、過去へシフト（1週間ごと過去へシフト）
-v-数字d	日を指定した日数分、過去へシフト
-v-数字H	時を指定した時間分、過去へシフト
-v-数字M	分を指定した分数分、過去へシフト
-v-数字S	秒を指定した秒数分、過去へシフト

▼リスト4　ログデータを変換するスクリプトlog2ssv.sh

```
#!/bin/sh

# 一時ファイルを出力するディレクトリを用意
tmpd=/tmp/log2ssv_$$; mkdir $tmpd
trap "rm -rf \"$tmpd\"" EXIT

# 必要なデータのみを日付とそれ以外に分けて
# ファイルに出力
cat $1 | bq2word |
awk '{print $4}' | tr -d '[' > $tmpd/date
cat $1 | bq2word |
awk '{print $1,$6,$7}'       > $tmpd/access

# 日付のフォーマットを変換
cat $tmpd/date |
LANG=C xargs -L1 -I@ \
        date -j -f "%d/%b/%Y:%H:%M:%S" \
        "+%Y%m%d%H%M%S" "@"   > $tmpd/datetime

# 日付とそれ以外のデータを結合して出力
paste -d" " $tmpd/datetime $tmpd/access
```

▼表3　個別に設定

オプション	内容
-v数字y	年を設定
-v数字m	月を設定
-v数字w	週を設定
-v数字d	日を設定
-v数字H	時を設定
-v数字M	分を設定
-v数字S	秒を設定

▼リスト5　指定した日時の前後10分間のデータを表示するスクリプトzengo.sh

```
#!/bin/sh

str=$(date -j -f "%Y%m%d%H%M%S" -v-10M "+%Y%m%d%H%M%S" "$1")
end=$(date -j -f "%Y%m%d%H%M%S" -v+10M "+%Y%m%d%H%M%S" "$1")

awk '"'$str'"<=$1&&$1<="'$end'"{print}'
```

▼図12　zengo.shの実行例

```
# log2ssv.sh /var/log/nginx-access.log | zengo.sh 20150910181538
20150910180804 192.168.1.106 GET_/softwaredesign/201511/freebsd/typescript.html_HTTP/1.1 200
20150910181417 192.168.1.106 GET_/softwaredesign/201511/freebsd/typescript.html_HTTP/1.1 200
20150910181428 192.168.1.106 GET_/softwaredesign/201511/freebsd/typescript.html_HTTP/1.1 200
20150910181538 192.168.1.106 GET_/softwaredesign/201511/freebsd/typescript.html_HTTP/1.1 200
```

第67回 Ubuntu Monthly Report

Ubuntu 15.10とそのフレーバーの変更点

Ubuntu Japanese Team　あわしろいくや

今回は10月22日にリリースされる予定のUbuntu 15.10とそのフレーバーの変更点をお知らせします。

Ubuntuの現状

Ubuntuは偶数年の4月に5年間サポートを継続するLTS（Long Term Support）をリリースしています。すなわち、奇数年の10月リリース分にはLTS前に対応しておきたい大きな変更が入ります。通常、LTS自体にはあまり大きな変更は入れず、主としてバグフィックに注力したり、5年サポートを見越してアップストリームの都合に合わせたアップデートなどを行っています。

それはそれとして、Ubuntuは数年がかりの大きな変化の真っ最中です。とくに注力しているUbuntu Phoneは日本でも購入できるようになりました[注1]。電波法の関係で、国内では購入できても使えないという大きな問題はあるものの、少なくとも開発は順調に進んでいるということです。

UbuntuはディスプレイサーバのMirと、デスクトップ環境のUnity 8でスマートデバイスとPCのインターフェースを統一し、「コンバージェンス」を実現させるべく開発中です。これも15.10リリースサイクルでのリリースはなさそうですが、次期デスクトップ版もUbuntu Desktop Nextとして開発が進行しています。9月にはMir+Unity 8でX Window System用のアプリケーションを動作させるデモ動画を公開しました[注2]。

本誌3月号で取り上げたSnappy Ubuntu Coreにも注力しています。その特性上、IoTやサーバで広く採用されそうですが、デスクトップもUbuntu Desktop Nextとして開発されているバージョンからこのSnappy Ubuntu Coreをベースとすることになり、現状開発が進んでいます。

サーバ面でもいろいろな開発が行われていますが、特筆すべきは本誌7月号でも取り上げたLXDです。また、コンテナのためのネットワーク拡張技術であるFan Networking[注3]という新顔も登場しました。

いずれも現在のリリース、つまりはUbuntu 15.10にはあまり影響しません。結果的には順当なバージョンアップ、換言すれば変更点はさほど多くないということになります。

15.10概要

リリース日とコードネーム

Ubuntu 15.10は本誌の発売日より少しあと、10月22日にリリースされる予定です[注4]。コードネームは

注1）http://gihyo.jp/admin/clip/01/ubuntu-topics/201508/21

注2）https://www.youtube.com/watch?t=3&v=BwOGy_UPo14

注3）https://wiki.ubuntu.com/FanNetworking 日本語による解説はUbuntu Weekly Topics 2015年6月26日号（http://gihyo.jp/admin/clip/01/ubuntu-topics/201506/26）に詳しいです。

注4）したがって本記事も開発版で検証しており、リリース版と異なる部分がある可能性があります。

“Wily Werewolf”で、「したたかな狼男」と訳されます。Ubuntuのコードネームは動物縛りですが、狼男が動物なのかはよくわかりません。最も、それは14.10のユニコーンでも同様ですが。ちなみに最初のリリースである4.10のコードネームは“Warty Warthog”であり、15.10は2回目のWシリーズです。

15.10共通の変更点

gcc 5への移行

15.10での一番大きな変更点はこれです。長らく使用していたgcc 4から5へ移行（transition）作業が行われました。libstdc++にもABIの非互換が発生する変更が入っているため、C++で書かれたパッケージもリビルドの対象になりました。ツールチェインの移行作業は何度となく行われていますが、今回は最大規模のものだったと言っていいでしょう。

具体的にどういう作業が発生するのかというと、パッケージには依存関係があるので、それらをすべて満たせない限りはそのパッケージはビルドできません。もちろんパッケージのリビルドは失敗することもあるので、その修正が行われない限りほかのパッケージがビルドできないということになります。ビルドできないパッケージが発覚する、それを修正する、ビルドする、ビルドできないパッケージが発覚する……というモグラ叩き的な作業が延々と行われます。ユーザにとってもたいへんなのは同じで、依存関係を満たせなければインストールないしアップデートできなくなります[注5]。今回は約4週間ほどそんな期間が継続し、8月12日には完了宣言が出されました。

ユーザにはとくにメリットがあるわけではありませんが、かといって避けることもできず、またLTSの開発期間に実施するには規模が大き過ぎるため、このタイミングでなければいけない作業だったといえます。

Linux 4.2

カーネルのバージョンは8月末にリリースされた4.2になっています。15.04は3.19だったため、Ubuntuとしては4.0系列は初採用となります。とはいえ、14.10のカーネルは3.16であり、15.04へは3バージョンのスキップでした。15.10でも同じく3バージョンのスキップです。

Network Manager 1.0.4

Network Manager 1.0がリリースされたのは昨年12月ですが、ようやくUbuntuでも採用されました。15.04で採用していた0.9.10のバージョンアップであり、多数の新機能が追加されています。詳しくは英語ですがリリースアナウンス[注6]をお読みください。なお、次のバージョンのNetwork Manager 1.2はこの12月にリリース予定です。

ntpdateの削除

15.04から時刻合わせにsystemd-timesyncdが使われており、ntpdateが削除されました。systemd-timesyncdはどちらかといえばntpdと似ており、ntpdateのように随時実行ではなく、デーモンとして常駐しています。任意で実行しない限りは起動時にしか時刻を同期しなかったのが、常時同期されるようになり、再起動をあまりしない環境でも時刻のズレを気にする必要がなくなりました。

Ubuntuの変更点

GNOME 3.16からオーバーレイスクロールバー機能が実装されました。Ubuntuにも独自のオーバーレイスクロールバーがありますが、GTK+ 3.xアプリケーションに関してはこちらを使用するようになりました。すなわち、GTK+ 3.xアプリケーションだけオーバーレイスクロールバーが別のものになりました（**図1**）。Ubuntu独自のものに比べてGNOMEのはあまり目立ちませんので、気づかないかもしれません。“OMG! Ubuntu!”というUbuntuニュースサイトの該当記事[注7]を見ても、さまざまな意見があります。

注5） Ubuntuだと開発版が明確に存在するので、ユーザはそれを避ければいいのですが、Debian sideだとそうもいかないので、ユーザはたいへんです。なお、gcc 5への移行はDebianでも同じタイミングで行われました。

注6） https://mail.gnome.org/archives/networkmanager-list/2014-December/msg00030.html

注7） http://www.omgubuntu.co.uk

そもそもオーバーレイスクロールバーなどという機能は不要である、前（Ubuntu独自）のほうがよかった、新しいのはさりげないのがいい、などなどです。UI的には複数の実装が混じるのは美しくないため、メンテナンスの手間を省くためだとは思うのですが、ここまで賛否さまざまな意見を目にするとは思いませんでした。UI設計の難しさがよくわかります。

それ以外にはあまり大きな変更はありません。Unity 7もたくさんの不具合が修正されていますが、15.04とバージョン（7.3.2）は変わっていません。

Kubuntuの変更点

KDEは執筆段階でKDE Frameworks 5.13、Plasma 5.4.1、Applications 15.08にアップデートしています。リリースまでにさらなるアップデートがあるかもしれません。

それよりも、Kubuntuプロジェクトは15.10開発期間中にUbuntu Community CouncilがKubuntu Community CouncilのJonathan Riddellにリーダー職の辞任を要求、反発した有力開発者が脱退するという事態が発生しました[注8]。Jonathanは元Canonicalで、Kubuntuのサポートをやめることになって他社へ転職したものの、Kubuntuプロジェクト開始当初からずっと中心メンバでした。現在もKubuntuでの活動は継続しているものの、今後はどうなるかわかりません。16.04ではKubuntuがなくなるのではないかという話も出ているくらいで、今後の動向が気になります。

Xubuntuの変更点

ベースとなるXfceはメジャーバージョンアップはしていないものの、細かなアップデートは行われています。

一番大きな変更は、ワープロソフトがAbiWordからLibreOffice Writerに、表計算ソフトがGnumericからLibreOffice Calcに代わったことでしょう。LibreOfficeはThe Document Foundationが配布するオフィシャルのバイナリを使用したいという場合にはXubuntuが最適だったわけですが、この変更によりそうも言えなくなってしまいました。

同時にGIMPはデフォルトインストールから外れ、Xfce Panel Switchというパネルの設定を管理するツールが追加されました。

Lubuntuの変更点

LXDEは現在メンテナンスモードであり、その後継となるLXQtの開発が進んでいますが、Lubuntuでは15.10でもLXDEを継続して採用しています。おそらく16.04でも同様でしょう。15.10ではLXQtのパッケージがすべてそろっているわけではありませんが、16.04ではそろうはずですので、LXQt版Lubuntuもプレビュー的な位置づけでリリースされるかもしれません。

Lubuntu独自の変更点はとくにありません。デフォルトでインストールしていたインプットメソッドがIBusからFcitxになったくらいです。

Ubuntu GNOMEの変更点

ベースとなるGNOMEは3.16にアップデートし、たくさんの変更点があります。詳しくはリリースノート[注9]をご覧ください。

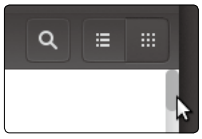
起動時にヘルプが表示されるようになり、チュートリアルが動画で見られるようになりました（**図2**）。英語ではあるものの、これで使い方がわからないということが減るでしょう。

GNOMEアプリケーションであるMusic（gnone-

注8）https://skitterman.wordpress.com/2015/05/26/information-exchange-between-the-ubuntu-community-council-and-the-kubuntu-council/

注9）https://help.gnome.org/misc/release-notes/3.16/index.html.ja

図1　GTK+ 3.xのオーバーレイスクロールバーはカーソルを近づけると広くなるだけです



music）と写真（gnome-photos）が追加され、Shotwellが削除されました。Rhythmboxはまだ残っています。筆者の個人的な感想としては写真がShotwellの役割を担えるとは思えませんが、いずれにせよ、リポジトリから削除されたわけではないので、必要になればいつでもインストールできます。

Ubuntu MATEの変更点

MATEデスクトップ環境は6月にリリースされた1.10にアップデートされました。古いソースコードの書き換え、不具合の修正、新機能の追加などが行われています。将来のGTK+ 3.xへの移行をふまえ、移植作業も行われています。ドキュメントビューアであるAtrilには、EPUB表示機能が実装されました[注10]。これはフォーク元のEvinceにもない機能です。

ubuntu-mate-welcomeという起動時にUbuntu MATEのさまざまな情報を表示するアプリケーションが追加されました（**図3**）。

インプットメソッド

「15.10共通の変更点」に含むべきではあるのですが、あまりにも変更が大きいので独立させることにしました。

ついデフォルトのインプットメソッドがFcitxになりました。15.04では中国語のみだったのですが、15.10では日本語を含むほかの言語でも追随しました。

そればかりか、変換エンジンはMozcになりそうです。バージョンはやや古い（1.15.1857.102）のですが、Anthyとは比較にならないほどの変換効率を誇ります。日本語ユーザには朗報ですが、残念ながら執筆段階ではまだ確定しているわけではありません。

日本語Remixの行方

Ubuntu Japanese Teamは結成以来Ubuntu日本語Remix[注11]をリリースしてきました。Ubuntuに日本語独自の修正を施しているのですが、当初はともかくここ最近の主な変更点は次のとおりです。

① インプットメソッドをfcitx-mozcに
② unzipにパッチを適用し、Windowsで作成されたZIPアーカイブを文字化けせずに解凍できるように
③ ライブイメージ起動時にも日本語の入力ができるように

①は前述のとおりUbuntuそのもののデフォルトになりそうで、②も修正は進んでいます[注12]。取り込まれるかどうかは未知数ではありますが。

残るは③ですが、今となっては取り立てて必要とも思えないので、近い将来に日本語Remixが不要になるのかもしれません。もしそうなればとても喜ばしいことです。SD

注10）執筆段階ではうまく動作しませんでしたが。

注11）名称は何回か変わっていますが。

注12）https://bugs.launchpad.net/bugs/1422290

図2　初回ログイン時「初めて使う方へ」が表示されるようになりました

図3　翻訳はされていないものの、ドキュメントのほかにもさまざまな情報にアクセスできます

第44回

Linux 4.1の機能
ディスクへの書き込みをリプレイするdm-log-writes

Text：青田 直大　AOTA Naohiro

予定どおり、Linux 4.2が8月30日にリリースされました。その後、さっそくLinux 4.3の開発が始まり、9月12日には4.3-rc1がリリースされ、一通りの4.3の機能が出揃っています。今回は、まず前回に引き続きLinux 4.0からいくつか残りの機能を紹介します。その後、Linux 4.1の新機能からdm-log-writesについて紹介します。

/dev/pmsg0

デバッグ時にはログが有力な情報となります。しかし、カーネルのデバッグ時には、ログを記録するファイルシステム自体に障害が発生し、ログを記録できなくなる場合があります。そこでLinuxカーネルではpstoreという簡易的なログ保存用の領域を用意しています。このシステムは、たとえば（NVRAMに保存される）EFI variableのような、電源が切れたあとでもその内容が残り、かつ読み書きがシンプルである領域を用いて、カーネルクラッシュのような緊急時のログを保存し、再起動後も読み取れるようにするものです。

これまでpstoreによるログの保存を使うのは、カーネル側のプログラムだけでした。すなわち、kernel panic時のBack traceなどのログ、カーネルのログ（dmesg）、ftrace（カーネルの関数トレース）の出力がpstoreを使用してログを出力していました。

Linux 4.0では/dev/pmsg0というデバイスファイルを通して、ユーザランドからpstoreを使用できるようになりました。syslogなどのログを/dev/pmsg0に記録することでデバッグがやりやすくなるでしょう。

ただし、この機能はEFIのNVRAMでは使うことができません。これはEFIのNVRAMがフラッシュメモリ上に実現されていて、書き換え回数が制限されているため、pmsg0やconsoleのログの対象にされてないからです。

gdbのヘルパスクリプト

もう1つ、Linux 4.0の新機能を紹介します。gdbはLinuxでよく使われるデバッガです。Linuxカーネルもgdbによってデバッグを行うことができます。Linux 4.0ではgdbによるデバッグを便利にするPythonのヘルパスクリプトが追加されています。このヘルパスクリプトによって追加されたコマンドについて見ていきましょう。

今回はQEmu上で起動しているLinuxを対象にします。そのVMにsshで接続して、“seq”コ

マンドを実行しておきます。

まずは、gdbをカーネルに接続します。QEmu上で動作しているLinuxを対象とする場合、qemuコマンドに引数"-s"(これは"-gdb tcp::1234"と同じ意味)で、TCP 1234番ポートでgdb serverが起動します。ビルドしたvmlinuxを引数にして、gdbを起動し、"target remote :1234"でgdbserverに接続します(**図1**)。接続されると、その時点でkernelが停止し、gdbのコマンドを実行できるようになります。

writeシステムコールを実行しているプロセスを取得してみましょう。"b sys_write"でwriteシステムコールにブレークポイントをしかけて、"c"でカーネルの実行を継続します(**図2**)。すると"seq"コマンドが数字を出力するために、write()を実行しているので、そこでgdbに制御が戻ってきます。ここでヘルパスクリプトの関数を使ってみます。たとえば、"$lx_currnet()"関数は、現在実行中のプロセスのプロセス構造体を取得する関数です。**図2**のように、"$lx_current().comm"をプリントすれば、"seq"コマンドの名前が出ていますし、"parent"をたどっていくことで、sshd -> bash -> seqというプロセスの親子関係も見えます。

次に、モジュール関連のコマンドを見てみましょう。まず、lsmodコマンド相当の働きを行う"lx-lsmod"コマンドを使って読み込まれているモジュールを見てみます(**図3**)。ここでは

▼図1　gdbの起動と接続

```
$ gdb vmlinux
GNU gdb (Gentoo 7.10 vanilla) 7.10
Copyright (C) 2015 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.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from vmlinux...done.
(gdb) target remote :1234                          # gdbserver への接続
Remote debugging using :1234
native_safe_halt () at /home/naota/src/linux/arch/x86/include/asm/irqflags.h:50
50      }
(gdb)
```

▼図2　gdbでseqの親子プロセス関係を見る

```
(gdb) b sys_write
Breakpoint 1 at 0xffffffff811a4f30: file /home/naota/src/linux/fs/read_write.c, line 577.
(gdb) c
Continuing.

Breakpoint 1, SyS_write (fd=1, buf=140558280798208, count=900) at /home/naota/src/linux/fs/r
read_write.c:577
577     SYSCALL_DEFINE3(write, unsigned int, fd, const char __user *, buf,
(gdb) p $lx_current().comm
$6 = "seq?000?000)?000?000oot?000?000?000?000"
(gdb) p $lx_current().parent.comm
$7 = "bash?000)?000?000oot?000?000?000?000"
(gdb) p $lx_current().parent.parent.comm
$8 = "sshd?000)?000?000oot?000?000?000?000"
```

“microcode”モジュールだけが読み込まれていてることがわかります。モジュール名がわかると、“lx_module”関数にモジュール名を指定してモジュール構造体を取得できます。

microcodeモジュール内の変数である“microcode_dev.nodename”の値をプリントしてみましょう。最初の状態では、モジュールのシンボルテーブルが読み込まれていないため、“microcode_dev”のシンボルを解決できないというエラーが出ています。ここで“lx-sysmbols”コマンドを使って、モジュールのシンボルテーブルを読み込みます。すると、“microcode_dev.nodename”の値がプリントできるようになりました。

そのほかのコマンドも簡単に見てみましょう。“lx_task_by_pid()”はプロセスIDを引数にとって指定したプロセスIDを持つプロセス構造体を取得する関数です(**図4**)。“lx-dmesg”はdmesgコマンド相当のカーネルログを取得するコマンドです。さらに、Linux 4.2以降のスクリプトであれば、“lx-ps”を使って実行中のプロセス一覧を取得できます。

dm-log-write

みなさんも、一度はマシンの電源が急に落ちたり、OSがハングしたりして再起動後fsckをかけることになったことがあるかと思います。ファイルシステムにおいて、いつ電源が落ちてもファイルシステムが壊れない信頼性を保持することは重要です。

いつ電源が落ちても、ファイルシステムが大丈夫かどうかをテストするために使える機能であるdm-log-writesがLinux 4.1には導入されています。これはdevice mapperを使って、指定したデバイスへの書き込みのログをとる機能です。

ではさっそくdm-log-writesを使ってみましょ

▼図3　モジュール関連コマンド

```
(gdb) lx-lsmod                                  # lsmodの取得
Address           Module                  Size  Used by
0xffffffffa0000000 microcode             24576  1
(gdb) p $lx_module("microcode").module_cores  # module構造体の取得
$9 = (void *) 0xffffffffa0000000
(gdb) p microcode_dev.nodename                  # microcode_devのアドレスを解決できていない
No symbol "microcode_dev" in current context.
(gdb) lx-symbols
loading vmlinux
scanning for modules in /home/naota/src/ktest/output
loading @0xffffffffa0000000: /home/naota/src/ktest/output/arch/x86/kernel/cpu/microcode/
microcode.ko
(gdb) p microcode_dev.nodename
$10 = 0xffffffffa0002326 "cpu/microcode"
```

▼図4　その他のコマンド

```
(gdb) p $lx_task_by_pid(732).comm
$10 = "cron?000)?000?000oot?000?000?000?000"
(gdb) lx-dmesg                                  # dmesgの取得
[    0.000000] b'Initializing cgroup subsys cpuset'
[    0.000000] b'Initializing cgroup subsys cpu'
[    0.000000] b'Initializing cgroup subsys cpuacct'
(...略...)
(gdb) lx-ps                                     # 実行中プロセスの取得（4.2以降）
0xffffffff81a10500 <init_task> 0 swapper/0
0xffff88001f0e8000 1 systemd
0xffff88001f0e8c00 2 kthreadd
0xffff88001f0e9800 3 ksoftirqd/0
0xffff88001f0eb000 5 kworker/0:0H
(...略...)
```

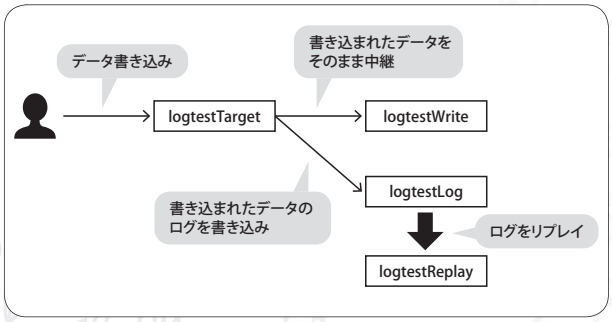
う(**図5**)。まずは、**図6**のようにlog-writesのデバイスを作ります。ここでは通常の読み書きを行うデバイスを"logtestWrite"として10GBで作り、ログの書き込みを行うデバイスを"logtestLog"として同じく10GBでLVMから作っています。ここでは、書き込まれたログがわかりやすいように、ログ書き込み用のデバイスのクリアも行います。次にdmsetupコマンドを使って、dm-log-writesデバイスである/dev/mapper/logtestTargetを作成します。ベースとなった/dev/vg2/logtestWriteと同じサイズのデバイスが作られています。

では、ログへの記録をしていきましょう。このデバイス上にext4のファイルシステムを作り、"dmsetup message"を使ってログに"mkfs"というマークをつけておきます(**図7**)。あとで、このマークを使うことで、この時点までのログを再生できます。このファイルシステム直下にランダムな内容で"file"というファイルを作り、SHA1チェックサムを取得しておきます。また、ファイルシステムのUUIDも確認しておきましょう。

次に記録されたログを見てみましょう。まず"dmsetup remove"を使って、/dev/mapper/logtestTargetを削除します(**図8**)。これですべてのログが/dev/vg2/logtestLogへとflushされます。また、同時に自動的に"dm-log-writes-end"というマークが記録されるます。

一度ログのdumpを見てみましょう。ログの形式はシンプルでデバイスの先頭のsuper blockのあとに、log entryが並ぶという形になっています。super blockは先頭から、マジックナンバー(8byte)、バージョン番号(8byte)、log entryの数(8byte)、セクタのサイズ(4byte)と並んでいます。その後、0x200byte目までzero埋めが続き、そこから"log entry #0"が始まります。0x200から0x400がentryのヘッダとなっており、書き込み先のセクタ番号(8byte)、書き込みセクター数(8byte)、フラグ(8byte)、データ長(8byte)と並んでいます。ここでいう「データ長」とは「書き込みのサイズ」ではなく、「マークデータの長さ」であることに注意してください。すなわち、こ

▼図5 dm-log-writesの動作



▼図6 dm-log-writesデバイスの作成

```
# lvcreate -L 10G -n logtestWrite vg2
  Logical volume "logtestWrite" created.
# lvcreate -L 1G -n logtestLog vg2
  Logical volume "logtestLog" created.
# dd if=/dev/zero of=/dev/vg2/logtestLog
dd: writing to '/dev/vg2/logtestLog': No space left on device
2097153+0 records in
2097152+0 records out
1073741824 bytes (1.1 GB) copied, 53.4529 s, 20.1 MB/s
# dmsetup create logtestTarget --table "0 $(blockdev --getsz /dev/vg2/logtestWrite) log-
writes /dev/vg2/logtestWrite /dev/vg2/logtestLog"
# fdisk -l /dev/mapper/logtestTarget
Disk /dev/mapper/logtestTarget: 10 GiB, 10737418240 bytes, 20971520 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
```

のlog entryはセクター0からセクター0x8000までの書き込みを記録したものになり、「データ長」は0ということになります。

dumpの末尾を見ると0x13465000byteから「マーク」を表すlog entryが見つかります。log entryのヘッダを見てみると、書き込みセクター番号・セクターサイズが0で、フラグが0x8(=LOG_MARK_FLAG)であることが示されています。このentryのdata sizeは17byteで、そのあとからdata(=マークの名前)である"dm-log-writes-end"が書かれています。

最後にlogのリプレイをしてみます。 logのリプレイには、**https://github.com/josefbacik/log-writes**で公開されているツールを使います。LVMからそれぞれ10Gで、"logtestReplayFile"と"logtestReplayMkfs"というデバイスを作ります(**図9**)。そして"replay-log"コマンドを使ってlogのリプレイを行います。"--log"の引数にlogを記録したデバイスである"/dev/vg2/logtestLog"を指定し、"--replay"の引数にリプレイ先のデバイスである"/dev/vg2/logtestReplayFile"または"/dev/vg2/logtestReplayMkfs"を指定します。さらに、"logtestReplayMkfs"のほうでは"--end-mark mkfs"をつけて"mkfs"マークまででログのリプレイを停止しています(**図9**)。

リプレイしたファイルシステムのUUIDで"tune2fs"コマンドで確認してみると、確かに同じUUIDのファイルシステムができていることがわかります。これらのファイルシステムはどちらも正常にmountできますし、"file"のSHA1チェックサムも一致し、ファイルの中身まで正しくリプレイされていることが分かります。また、mkfs直後まで止めた"/dev/vg2/logtestReplayMkfs"の方には"file"は作られていないことも確認できます。

▼図7　ログの記録

```
# mkfs.ext4 /dev/mapper/logtestTarget
mke2fs 1.42.13 (17-May-2015)
/dev/mapper/logtestTarget contains a ext4 file system
        created on Thu Sep 24 01:40:40 2015
Proceed anyway? (y,n) y
Discarding device blocks: done
Creating filesystem with 2621440 4k blocks and 655360 inodes
Filesystem UUID: 962a3b09-486d-49a1-84dd-babf813c6b3e
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

# dmsetup message logtestTarget 0 mark mkfs
# mount /dev/mapper/logtestTarget /mnt/test
# dd if=/dev/urandom of=/mnt/test/file bs=4k count=256
256+0 records in
256+0 records out
1048576 bytes (1.0 MB) copied, 0.136649 s, 7.7 MB/s
# sha1sum /mnt/test/file
a0658f0063e62ff7b37411d3762c090f981f6d9a  /mnt/test/file
# umount /mnt/test
# tune2fs -l /dev/mapper/logtestTarget
tune2fs 1.42.13 (17-May-2015)
Filesystem volume name:   <none>
Last mounted on:          /mnt/test
Filesystem UUID:          962a3b09-486d-49a1-84dd-babf813c6b3e
Filesystem magic number:  0xEF53
(...略...)
```

まとめ

今回はLinux 4.0からuserlandのログを永続化する/dev/pmsg0、gdbのヘルパスクリプト、そしてLinux 4.1からディスクへの書き込みのログを取るdm-log-writesについて紹介しました。dm-log-writesは、Btrfsの開発者が開発に関わっていたこともあり、ツールにはほかにもファイルシステムのテストに便利な機能が実装されています。ファイルシステムを作りたくなったら、ぜひ試してみてください。SD

▼図8　logのダンプ

```
# dmsetup remove logtestTarget
# hexdump -C /dev/vg2/logtestLog
                                                                  # 0x200 まで super block
00000000  72 68 73 77 66 73 6a 00  01 00 00 00 00 00 00 00  |rhswfsj.........|  # magic(8byte), version(8byte)
00000010  38 84 00 00 00 00 00 00  00 02 00 00 00 00 00 00  |8...............|  # entries(8byte), sector size(4byte)
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|  # 0x200(sector size)までzero埋め
*                                                                 # 0x400 まで entry header
00000200  00 00 00 00 00 00 00 00  00 80 00 00 00 00 00 00  |................|  # sector(8byte), #sectors(8byte)
00000210  04 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|  # flags(8byte), data len(8byte)
00000220  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400  00 80 00 00 00 00 00 00  00 00 40 00 00 00 00 00  |..........@.....|  # 書き込みデータ
00000410  04 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000420  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000600  00 80 40 00 00 00 00 00  00 00 40 00 00 00 00 00  |..@.......@.....|
(...中略...)
13465000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|  # sector=0, #sectors=0
13465010  08 00 00 00 00 00 00 00  11 00 00 00 00 00 00 00  |................|  # flag=0x8(MARK), data size=17
13465020  64 6d 2d 6c 6f 67 2d 77  72 69 74 65 73 2d 65 6e  |dm-log-writes-en|  # data
13465030  64 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |d...............|
13465040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
40000000
```

▼図9　logのリプレイ

```
# lvcreate -L 10G -n logtestReplayFile vg2
  Logical volume "logtestReplayFile" created.
# lvcreate -L 10G -n logtestReplayMkfs vg2
  Logical volume "logtestReplayMkfs" created.
# git clone https://github.com/josefbacik/log-writes
# cd log-writes; make
(...略...)
# ./replay-log --log /dev/vg2/logtestLog --replay /dev/vg2/logtestReplayFile
# ./replay-log --log /dev/vg2/logtestLog --replay /dev/vg2/logtestReplayMkfs --end-mark mkfs
# tune2fs -l /dev/vg2/logtestReplayFile                          # どちらのFSもUUIDが一致する
tune2fs 1.42.13 (17-May-2015)
Filesystem volume name:   <none>
Last mounted on:          /mnt/test
Filesystem UUID:          962a3b09-486d-49a1-84dd-babf813c6b3e
Filesystem magic number:  0xEF53
(...略...)
# tune2fs -l /dev/vg2/logtestReplayMkfs
tune2fs 1.42.13 (17-May-2015)
Filesystem volume name:   <none>
Last mounted on:          <not available>
Filesystem UUID:          962a3b09-486d-49a1-84dd-babf813c6b3e
Filesystem magic number:  0xEF53
(...略...)
# mount /dev/vg2/logtestReplayFile /mnt/test
# sha1sum /mnt/test/file                                         # fileのSHA1 checksumが一致する
a0658f0063e62ff7b37411d3762c090f981f6d9a  /mnt/test/file
# umount /mnt/test
# mount /dev/vg2/logtestReplayMkfs /mnt/test; ls -l /mnt/test    # mkfs後まで replay したデバイスには"file"がない
total 16
drwx------ 2 root root 16384 Sep 24 06:10 lost+found
```

November 2015

NO.49

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
岡松 伸太郎　OKAMATSU Shintaro　shintaro@jus.or.jp
りゅうちてつや　RYUCHI Tetsuya　ryuchi@ryuchi.org

手を動かし頭を使って腕を磨く、シェル芸漬けの夏の日

jus & USP友の会共催シェル勉強会

■jus & USP友の会共催シェル勉強会
【日時】2015年8月29日（土）10:00～17:00
【会場】（株）KDDIウェブコミュニケーションズ

USP友の会と共催でシェルスクリプトの勉強会を開催しました。この勉強会は毎回好評で参加者が多く、今回も事前申込の時点で満席となっていました。当日の参加者42名でした。今回の勉強会は午前の部と午後の部の2回に分けて開催されました。

今回の勉強会は、午前は座学、午後は講師の出した問題をワンライナーで解いてみる勉強会（通称「シェル芸勉強会」）という構成で開催されました。

午前の部「なぜシェルに仕事をさせてはいけないのか」講師：鳥海 秀一（USP友の会）

午前の部は鳥海さんによる勉強会でした。この勉強会のきっかけは、昨年12月に鳥取で開催されたjusのシェルスクリプトワークショップにてjusの幹事でもある鳥取環境大学の齊藤明紀先生が示したスライドにあった「シェルに仕事をさせてはいけない」という言葉だそうです。「なぜシェルに仕事をさせてはいけないかを知るためには、実際に仕事をさせてみたらいい」ということで、重めの処理を仕事として与えることにしました。その題材として選択されたのは「nクイーン問題」[注1]でした。

注1）n×nマスのチェスの盤上にn個のクイーンを置き、どの駒もお互いに取られないような配置を考える問題。

■nクイーン問題をBashで解くと遅い

nクイーン問題の説明のあと、鳥海さんが実際に17のプログラミング言語でnクイーン問題を解いた例が示されました。さらにそれを参考にしてBashでnクイーン問題を解いてみるよう指示があり、座学と思っていたところに課題が出されて驚かれた参加者もいたようです。みなさんしばらく実際に解いてみたり、鳥海さんが作成した例を読み解いて考えたりしていたようです。

その後、鳥海さんからBashでの解法の説明がありました。この際、nクイーン問題を解く前にまず4クイーンを解いてからその解法をnクイーンに拡張する、という方針が示されました。このように問題を解くにあたっての方針が示されたのは、そのあとの鳥海さんによるライブコーディングによる解法例を見ていた参加者の理解を助けたと思います。

Bashでの解法ができたところで、鳥海さんの環境で複数の言語での12クイーン問題の解答時間の比較が示されましたが、Bashは6分以上と圧倒的に時間がかかっていました。その理由として変数へのアクセス効率の悪さ、具体的には変数がすべて文字列となっている点、そして、配列が高速なランダムアクセスを実現していない点を挙げていらっしゃいました。

■シェルはglue langとして使う

ではどうしたらいいのかと考えるにあたり、あらためて齊藤先生のスライドに立ち返り「シェルにはglue langとしての役割をさせる」という方針が示されました。今度はawkのワンライナーをBashの関

数として定義し、パイプというglue(糊)を使ってつなげていくことで実装されました。この方法で実装したことにより、最初の実装方法では6分以上かかっていた12クイーン問題が11秒で解けました(講師の鳥海さんの環境上での計測結果)。

パイプを使ったシェルスクリプトの利点としては

- 処理速度が速い点
- スクリプトが組みやすい点
- 見通しが良い点

が挙げられ、欠点しては

- プロセス数を意識する必要がある点
- 手続き的な発想とは異なる発想が必要な点

が挙げられました。

さらにパイプを使ったシェルスクリプトでは、手続き型ではなく関数型プログラミングの発想が必要とされるという説明がありました。とくに抽象代数における閉包性を活用する点は共通点として強調されました。また、関数型プログラミングに慣れるための方法として何点か挙げられた中で一番推していらしたのは、シェル芸勉強会に参加することでした。そして、関数型言語としてのシェルスクリプトのメリットとしては、一般的な関数型プログラミングと違って、記述順と処理順が一致することを挙げていらっしゃいました。

nクイーン問題を実際に解くという課題を出されたり、「閉包性」など幅広い話題が取り上げられたりと、参加者は講師の話を聴くだけではなく、頭を使い、手を動かす午前の部となりました。午前の部のスライドはWeb上に公開されています[注2]。

午後の部「シェル芸勉強会」
講師:上田 隆一(USP友の会)

午後の部は、USP友の会がいつも開催しているシェル芸勉強会が開催されました。いつもは上級者が初心者の人をサポートしながら進めていけるようにグループ分けが行われるのですが、今回は午前中のスクール形式のレイアウトをそのまま使う形式で開催されました。

■シェル芸勉強会の進め方

勉強会は、講師の上田さんによる眠気覚ましを含んだスライドから始まり、USP友の会の近況についても報告がありました。そのあとシェル芸についての説明や進め方、問題を解くときの着目点などについての一通りの説明がありました。今回のシェル芸勉強会では8問の問題が用意されていること、1問を約15分で進めること、回答例はLinux環境で用意されていることなどが説明されました。

シェル芸の回答例として、最近はawkを使って回答されるケースが多くなってきているので、今回はなるべくawkを使わない方法で用意されたとのことですが、awkを使った回答も歓迎されていました。

■予想外の回答が出ることも

シェル芸勉強会では、講師の方も知らないコマンドの使い方があったり、会場から時折思いもよらぬ回答が飛び出したりすることがあります。今回の問題の1つに、テキストファイルから出力するときに重複する空行を出力しないようにする問題がありました。この問題は次のように入力すれば良いことが即座に会場から出てきました。

```
$ cat -s < text
```

この機能はcatコマンドですでに実装されており、-sオプションを使うことで重複する空行を抑制できます。そのほかcommコマンドなど、今まで気づかなかった便利なコマンドがたくさん出てきて、とても勉強になったシェル芸勉強会でした。

一通り問題が終了したあとは、同じ会場で有志による懇親会が行われました。ピザなどを食べ、飲み物を飲みながら、自己紹介やLTによるプレゼンが行われ、お互いに楽しい時間を過ごすことができました。SD

注2) URL http://www.slideshare.net/umidori/20150829-jus

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
及川 卓也　OIKAWA Takuya
Twitter @takoratta
鎌田 篤慎　KAMATA Shigenori
Twitter @4niruddha
清水 俊之介　SHIMIZU Shunnosuke
Twitter @donuzium
高橋 憲一　TAKAHASHI Kenichi
Twitter @ken1_taka

第47回 第4回 石巻ハッカソン——その2

前回に引き続き、2015年7月に開催された第4回目となった石巻ハッカソンについてレポートします。Hack For Japanのメンバーがかかわったチームを掘り下げる続きで、今回は及川と高橋それぞれのチームについて紹介したいと思います。

我が家の防災計画メーカー

筆者（及川）は今回で4回連続の石巻ハッカソンの参加となります。1年目と2年目はITブートキャンプ部門で講師を務めさせていただき、3年目となる昨年は、1年目に教えたうちの1人で、今ではイトナブの中心メンバーの1人として今年のITブートキャンプでは講師を担うまでになった、デルシオこと中塩成海さん[注1]のプロジェクト、「Disaster Survival Toolbox」に参加しました。これは発災後に必要となる知恵を事前に収集・共有するためのサービスです。このプロジェクトはまだ継続中ですが、中塩さんを始めとしたメンバーのスケジュール調整がつかず、少し停滞してしまっているので、そろそろ再開したいところです。

さて、4年目となる今回も昨年と同じように、一参加者としてハッカソン部門に参加しました。初日のアイデアピッチを聞いて参加を決めたのが、加藤拓明さんが提案していた「マイ避難所マップ」です。加藤さんとは、福島のITコミュニティであるエフスタ!![注2]の活動を通じて以前より知り合いでしたが、当日ピッチの後、所在なさげに賛同してくれる仲間を待っているのを見て、少し話でも聞いてみようと思ったのがきっかけでした。

プロジェクトの構想

加藤さんの「マイ避難所マップ」というアイデアは聞いてみると、2つのアイデアが合わさったものでした。1つは家族で防災を話し合い、災害時に自宅近くの避難所まで逃げる経路を自分たちで作るもの。もう1つは、この避難所までの経路を作成をするサービスを、普段プログラミングをすることのないようなお父さんでも、子どもたちと一緒に作ることができるようなツール。

後者は、小さいときにお父さんがアプリケーションを作ってくれたことからITの世界に興味が出始めたという、加藤さんの個人的な経験に基づくものでした。小学校低学年の子どもでもお父さんと一緒に作れるように、カードのような紙を組み合わせることでサービスが構築できるものを加藤さんは想定していました。たとえば、写真というカードや地図というカードなど、最終的に自分たち家族で必要となる避難所までの経路情報を表示するためのカードを好きな位置に配置します。カードですから、子どもでも好きな配置を考えられます。それぞれのカードの裏には、その機能を実現するためのコードや家族向けにカスタマイズするための変数が書かれているので、それを後でツールから打ち込むことで実際に紙のカードベースで組み上げたサービスが実現できるというアイデアです。

この加藤さんの考える2つのアイデアを一緒に他参加者にアピールしているうちに、2名のメンバーが集まりました。加藤さんと同じくエフスタ!!のスタッフである藤原裕也さんと、昨年も石巻ハッカソンに参加したテキストマイニングやソーシャル分析の研究者である村上明子さんです。

加藤さんのアイデアは、防災というアイデアとITに慣れ親しんでもらうツールというアイデアの2つが組み合わさったものでした。これはこれで魅力的ではあるものの、実質作業できるのは1日半という

注1　イトナブは変なあだ名を付けられるという、メンバーになるためのイニシエーションがあります。
注2　http://efsta.com/

限られた時間であることと、サービスの軸がぶれてしまう恐れがあるということで、話し合った結果、防災一本に絞ることとしました。ただし、家族で一緒に作るというコンセプトは守ることにします。

サービスのイメージを詰めていく

さて、皆さんは防災について家族で考えるとき、避難所までの経路以外にどのようなことを話し合うでしょう。私たちもまずはそのことを考えました。まずは、自宅近くの避難所の場所を確認。災害によって適切な避難所は異なるかもしれません。津波の際の避難の場合は、避難所までの道も普段使う最短の道ではないかもしれません。また、災害時に持ち出す荷物や自宅に常備しておくものも、家族で考える必要があるでしょう。お父さんやお母さんの連絡先も子どもたちにとっては大切です。

すでに自治体や各種団体などからハザードマップを始め、防災のための情報は多く提供されています。私たちの作るものは、完璧な情報を網羅するというよりも、このサービスを使って、家族が防災について話し合い、自分たちなりの防災計画を作るものに主眼を置くこととしました。プロジェクトの名前もそれに合わせて「我が家の防災計画メーカー」となりました。

サービスはWebアプリケーションの形を取り、ターゲットとなる小学校低学年の子どもでもわかるようなシンプルなユーザインターフェースを目指します（**写真1**）。バックエンドはPHPで各種コンポーネント間のデータの連携や外部サービスとのつなぎを担当します。時間も限られているので、今回のハッカソンではサービスのコンセプトが示せるようにと、データの入力から自分たち用の防災計画が表示されるまでの基本フローを実現することと、肝となる避難経路を作成・表示する部分に注力することにしました（**写真2**）。

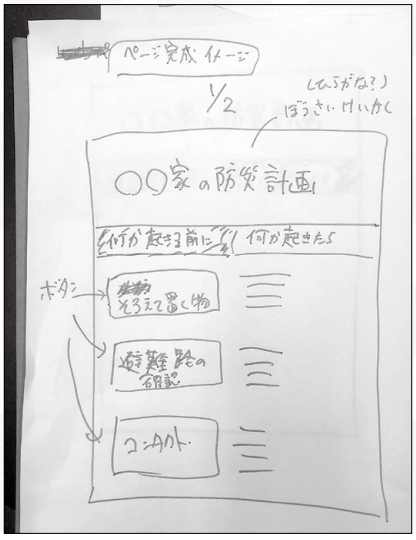
避難経路は、ITに慣れた私たちだと、Googleマップのような地図に経路をプロットすることを想定しがちです。しかし、道筋を示すには必ずしも地図で表示することだけが最善とは限りません。しかも、ターゲットは子どもです。

そこで私たちは避難経路を実際に確認し、その際に撮った写真をそのまま防災計画に取り込めるようなしくみを考えました（**図1**）。子どもと一緒に自宅を出て、最寄りの避難所までを実際に歩いてみます。「この信号を左折、坂を登って」と避難所までの道を子どもと確認しながら、撮った写真をサービスに取り込むと、その写真が撮影日時順に並び（後から変更可能です）、さらには撮影場所の緯度経度を元にGoogleマップのような地図上にも表示されます[注3]。撮影された写真を子どもとともに確認することで、自宅でももう一度道順を確認できます。

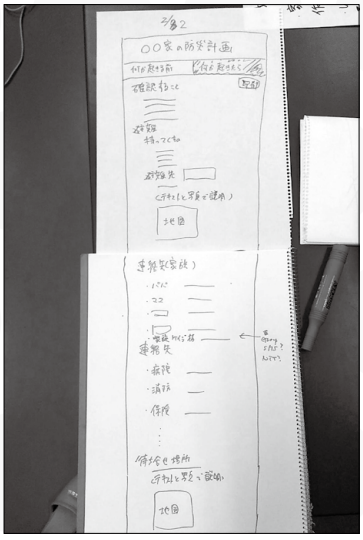
また、このサービスは何度も言っているように子どもがターゲットで、しかも災害時に使用されることを想定しています。そのため、作成はブラウザか

注3　そのため撮影にはスマートフォンを推奨しています。

◆写真1　データ入力を行うトップ画面のアイデア



◆写真2　各種情報をカスタマイズし、防災計画を固める画面イメージ



◆図1　平時にそろえておくもの、家族の連絡先、避難所の確認を入力します

◆図2　完成した防災計画は印刷できます

ら行いますが、その結果となる防災計画は紙に印刷して使うことにしました(**図2**)。

実質1日半となる開発時間でしたが、どうにか予定したコア機能を実現し、最終日の成果発表会に臨みました。

ハッカソンを終えて

成果発表会では残念ながら賞はいただくことはできませんでしたが、審査員の方々からは高い評価を得られました。「ぜひとも完成させて教育委員会などに持ち込み、実際の防災教育に役立ててほしい」との暖かいメッセージもいただきました。また、家庭にそろえておくものに関しては、「多くの家庭で必要となるものなどは同じこともあるので、そのような典型的な常備品については入力を容易にする工夫がほしい」とのフィードバックもいただきました。実は開発したサービスでは、入力項目ごとにマウスをホバーさせると、ヒントとなるような情報が表示されるようになっており、推奨される常備品についてもそこで示されてはいました。しかし、それはあくまでも表示されるだけで、再入力が必要ですし、マウスホバーで表示というユーザインターフェースも、必ずしも操作しやすいものではありません。実装が間に合わなかったというのが実情ですが、常備品や避難所などを選択するだけでデータ入力が完了する項目の操作性については今後の改善点です。

ハッカソンとしては成果発表会のデモで私たちのプロジェクトはいったん終了ですが、審査員の方に言われたように一般公開を目指したいと考えています。ハッカソンが終わると、モチベーションの低下などもあり、継続されないプロジェクトも多くありますが、審査員からの「このサービスはもう動いているみたいなんですけど、すぐ使えますか？」という質問に「はい！」と元気よく答え、他メンバーを慌てさせた加藤リーダーの根拠なき楽観のもと、このプロジェクトはぜひとも継続したいと考えます。家族で防災について話し合うことで防災意識を高めることこそが、もっとも災害に強い家族、そして社会を作っていくものと信じています。

色白族の楽園

筆者(高橋)も2012年の第1回目から継続して参加しており、これまでは講師やサポート役に徹してきたのですが、今年は一参加者として開発をすることができました(それができたのも、初回の頃はアプリ開発は初めてだったメンバーが自分たちだけで進められるまでに成長してくれたおかげです)。2013年の2回目に声をかけて以来、継続して参加してくれているAndroidコミュニティでの盟友adamrocker氏とチームを組み、今回のテーマである海をVR空間に実現することにしました。第2回のときにボクスケというアプリを開発して優勝しているadamrocker氏と、講師でもある筆者(高橋)が組むことで一部の方からは「この2人が組むなんてずるい!」という指摘もいただいたのですが、マインクラフトで何かやりたいというadamrocker氏と、Google Cardboard[注4]で何かやりたいという筆者(高橋)の純粋な思いが合致して誕生したチームです。

「色白族の楽園」というタイトルは、今回「海」がテーマとして与えられたことは、夏に海に行って黒く焼けることに縁遠い(=色白)我々開発者への挑戦状と捉え、それならば「バーチャルな空間に我々が思い描く海の楽園を作ろうじゃないか」という発想から来たものです。

構成はMac OS X上でマインクラフトを動かし、解析した地形データとリアルタイム情報(プレイヤーの位置やブロックの生成・破壊などの情報)をmodと呼ばれるしくみで連携できるように再構成しました。地形データはデータ自体が巨大なのでGoogle Cloud Storageを経由し、リアルタイム情報はGoogle App Engineで構築したサーバ経由でAndroid端末に転送、その端末をGoogle Cardboardにセットして VRとしての表示を行うというものです。Androidのアプリケーションは通常どおりJavaで実装し、3Dグラフィクスの表示にはRajawali[注5]というライブラリを使用したうえで、Google Cardboard SDKを使用して右目用と左目用に分けた表示や方向のトラッキングを行いました。

◆写真3 Cardboardにセットされた端末の画面に浮かぶ海(のようなもの)

サーバに用意したWeb APIを呼び出すと、全体のシーン構築情報とプレイヤーの現在位置の情報を取得できるようにしてあり、端末側ではそのAPIにより得られた圧縮ファイルを展開、シーン構築をする……ところまではできたのですが、広大な空間を実現するために必要なメモリの不足問題を時間内には解決できず、海のようなものがVR空間上にあるという程度のところでタイムオーバーとなってしまいました(**写真3**)。

理想とする楽園の完成には至りませんでしたが、2人ともこのマインクラフトの世界観とそれを実現する膨大な仕様に感心しながら、純粋に高度な技術的課題をお互いに課してハッカソンの醍醐味を楽しんだ2日半でした。

来年も開催決定

このように、高校生をサポートする役に回ったり、即興でできたチームに加わったり、技術的な興味の赴くままに開発をするなど、多様な参加の方向性があるのもこの石巻ハッカソンの魅力です。最後の発表のときに「今年はスキルが足りず開発に貢献できなかったけど、来年は開発する側に回りたい」と言ってくれた方がいました。すでに来年の開催日程は2016年7月22～24日と決定しています。この連載の昨年の記事を見て興味を持ったのがきっかけで参加してくれた方もいます。来年の夏、非常に濃い3日間をぜひ石巻で過ごしてみませんか。SD

注4 https://www.google.com/get/cardboard/
注5 https://github.com/Rajawali/Rajawali

温故知新
IT むかしばなし

第48回

磁気ディスクメディア
～フロッピーディスクの変遷と停滞～

速水 祐(HAYAMI You)　http://zob.club/　Twitter : @yyhayami

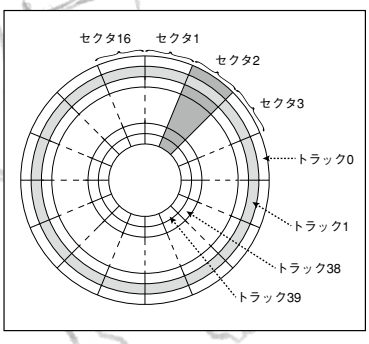
はじめに

筆者のPCには、いまだに3.5インチのフロッピードライブ（以下FDD）が前面スロットに収まっています。PCの進歩の激しい中、3.5インチフロッピーディスク（以下FD）は、つい最近まで現役のディスクメディアでした[注1]。その3.5インチFDが倉庫に多量にあり、さすがに処分しようと考えて整理していたところ、古い5インチや8インチFDさらにクイックディスク[注2]などの多様なディスクメディアが出てきました。中には、開発ソフトウェアツールやゲームなど、非常に貴重に感じられるものもありました。今回は、これらの磁気ディスクメディアについてお話します。

注1） 2011年3月生産中止。

注2） クイックディスク(Quick Disk)は、レコードの溝のように、渦巻状のトラックが1本だけ存在し、片面全部を順に一気に読み出しまたは書き込みするシーケンシャルアクセスのみが可能で、任意部分へのランダムアクセスはできません。片面64KB、A／B面を裏返して使用します。

▼図1　2Dフロッピーディスクの構造



最初のパソコンのFD

筆者が最初に個人で購入して使用した磁気ディスクメディアは、1982年末発売の富士通の8bitマシンFM-7用の5インチFDでした。それまではカセットテープでデータをセーブしたりロードしたりしていたので、FDのスピードは圧巻でした。それ以前にAPPLE IIのFDを使用する機会もあったのですが、単にロードスピードだけでなく、少ないメモリ上で大きなデータをやりとりするランダムアクセス機器としての魅力を強く感じており、アドベンチャーゲームのような多くのグラフィックデータを必要とするゲームの開発には必須だと思っていました。そのためFM-7用のFDが本体よりも高価だったにもかかわらず、思いきって購入したのです。5インチFDは正確には5.25インチの直径です。記録方式や容量の違いにより、いくつかの種類があって、FM-7用のものは2Dと呼ばれていました。2は両面（2 Side；ディスクメディアの両面にデータを読み書きできる）を表し、Dは倍密度（Double Density；2倍のデータを記録できる）を意味します。2D FDは**図1**のような構成になっており、その総容量は次の計算で求まります。

総容量＝セクタサイズ×1トラック内のセクタ数×トラック数×面数（片面なら1、両面なら2）

セクタサイズは、1980年代のパソコンのFDで256byte、その後MS-DOSで512byteになります。倍密度の場合、1トラック内のセクタ数は16セクタ、単密度なら8セクタです。トラック数は40で、両面ですので最後2を掛けます。

256byte × 16 × 40 × 2 ＝ 320KB

現在では、8GB以上のUSBメモリを当たり前のように使用していますが、当時は320KBを広大に感じたものです。いくらでもデータ、プログラムが格納できるという気がして、1枚の高価なメディア（1983年当時1枚1,400円程度）を何回も使い回しました。

1979年末発売されたにNECのPC-8001の周辺機器の5インチ

FDDは、当初片面倍密度の1D（総容量160KB）でしたが、直後に2DのFDDに変わっています。ですので、当時の8bitパソコンのFDは2Dと考えていいでしょう[注3]。

フロッピーディスクの容量の変遷

1983年発売のNECのPC-9801Fには、トラック数が2Dに比べて2倍の80になった2DD（両面倍密度倍トラック）FDDが搭載されました。これは、トラック数が倍なので、総容量は640KBです[注4]。

続いて1984年に発売されたPC-9801Mには、2HD（両面高密度。容量1MB）のFDDが搭載されました。筆者がFM-7の次に手に入れたマシンがこれでした。今まで使用していた2D FDの3倍以上の容量になり、2HDが10枚あれば、今まで作成したデータがすべて格納できると思いました。

セクタサイズはフォーマット[注5]によって異なりますが、DISK BASICフォーマットの2HDのセクタサイズは、256byteと2Dと同じでした。しかし、1トラックの記録密度が上がり、倍密度から高密度になって、セクタ数は16から26に増えました。また、トラック数は2DDの80トラックに対し、若干少ない77トラックでした。

256byte × 26 × 77 × 2 ＝
1,001KB（1MB）

初代PC-9801の標準FDDは8インチでした。5インチの2HD FDDは、その8インチ両面倍密度2DのFDDの構成をそのまま小さくしたイメージで、8インチFDをそのままディスクコピー[注6]できました。初代PC-9801の8インチFDDインターフェースに5インチ2HDのFDDをそのまま接続することもできたのです。ディスクの回転数も5インチ2Dや2DDの300rpm（1分間に300回転）から、8インチと同じ360rpmと高回転になりました。2HD/2DDを両方アクセスできるドライブは、この2種類の回転数を切り替える機能が要求されました。

5インチから3.5インチFDへ

2HDのMS-DOSフォーマットのFDは、容量がセクタ長1,024byte、セクタ数8、トラック数77で1.25MBで、NECや富士通のパソコンの標準フォーマットになりました。その後ディスクサイズがコンパクトになった3.5インチのFDでも、まったく同じフォーマットとなり、広く長く使用されるようになりました。

一方、海外のIBM-PCは、5インチから3.5インチへの切り替えの時期が、2DDが利用され始めたときと重なったため、5インチ2DD（PC-9801のMS-DOS 2DDと同じ構成）がそのまま3.5インチディスクになり、さらにセクタ数を1つだけ増やして9にした容量720KBの2DD[注7]も使われるようになりました。

この流れで2DDが2HDに発展するとなると、倍密度から高密度でセクタ数が2倍になり、セクタサイズ512byte、セクタ数18、トラック数80ですので、

512byte × 18 × 80 × 2 ＝ 1.44MB

となります。ディスクの回転数も2DDと同じ300rpmのままです。

8インチの互換性を活かして5インチ、3.5インチへの道を進んだPC-98、それに対して5インチの互換性を活かして3.5インチに進んだIBM-PCとは異なる道を進みました。

その後PC-98からDOS/Vに変わっていく中で、2つのFDの構成の違いに悩まされた方も多かったのではないでしょうか。

フロッピーディスクのその後

パソコンにおけるCPUの速度、メモリ・ハードディスクのアクセス速度と容量は、まさに倍々で増えていきましたが、FDの容量は、キーボードの入力速度と同じように、増えませんでした。時代はハードディスクへと移行していったのです。

1988年にNECが発売した究極のPC-8801シリーズの88VA3では、2TDと呼ばれる2HDの7倍以上の約9MBの容量のFDDが搭載されましたが、一時の話題だけに終わったようです。SD

注3） その前にSHARP MZ-80用の両面単密の2S（総容量140KB）というFDがありましたが、使用していた人はほんのわずかでしょう。

注4） MS-DOSはセクタサイズが2倍の512バイトになっており、その分セクタ数は半分になっています。

注5） ここでは物理フォーマットのこと。

注6） べたコピーとも言う。トラックやセクタなどの構造も含めてコピーすること。

注7） 当時は/9（スラッシュナイン）フォーマットなどとも呼ばれていました。

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

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

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

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

Software

トレンドマイクロ、総合サーバセキュリティ対策製品の最新バージョン「Trend Micro Deep Security 9.6」を提供開始

トレンドマイクロ㈱は10月1日、総合サーバセキュリティ対策製品の新バージョン「Trend Micro Deep Security 9.6」の提供を開始した。

Trend Micro Deep Securityは、サーバのセキュリティに求められる多様な機能（ウイルス対策、IPS/IDS、Webレピュテーション、ファイアウォール、ファイルやレジストリの変更監視、セキュリティログ）を1つのソリューションとして実装した、総合サーバセキュリティ対策製品。自分の環境に合わせ、各OSにインストールする「エージェント型」と、VMware ESX上にインストールして仮想マシンを保護する「仮想アプライアンス型」の2種類のモジュールを選択できる。

新バージョンである「9.6」の、仮想アプライアンス型では、ハイブリッドクラウドの基盤ソフトウェアの最新バージョンであるVMware vSphere 6.0に対応する。これにより、vSphere 6.0が提供する拡張性、可用性、耐障害性がさらに向上した仮想環境でのセキュリティ対策が可能になる。

エージェント型としては新しく、Windows Server Core、Debian 6/7、SLES 12、Oracle CloudLinux 7、Oracle Linuxのプラットフォームに対応する。

CONTACT
トレンドマイクロ㈱ URL http://www.trendmicro.co.jp

Software

アクロニス、個人向けバックアップソフトウェア「Acronis True Image」の最新版を発売

アクロニス・ジャパン㈱は9月9日、一般ユーザ向けバックアップソフトの新バージョン「Acronis True Image 2016」「Acronis True Image Cloud」を発売した。

Acronis True Imageは、Windows/Mac向けのバックアップソフト。データの転送時および保管時ともに、すべてのデータをAES256で暗号化する（復号する際はパスワードを入力する）。本製品のクラウド版であるAcronis True Image CloudはAcronis Cloudストレージを容量無制限で標準提供する、1年更新のサブスクリプションモデル。最新版ではモバイルデバイスのデータバックアップ／復元機能が強化されるとともに、Windows PCやMacのローカルドライブ上の利用頻度の低いファイルや大容量のファイルをクラウドで保管する「クラウドアーカイブ機能」が追加された。

●価格表（すべて税別）

	Acronis True Image 2016	Acronis True Image Cloud
通常購入	4,980円（1ライセンス） 7,980円（3ライセンス）	9,980円（4ライセンス［内訳：コンピュータ：1、モバイル：3］） 15,980円（13ライセンス［内訳：コンピュータ：3、モバイル：10］）
アップグレード	2,980円（1ライセンス） 4,980円（3ライセンス）	6,980円（4ライセンス［内訳：コンピュータ：1、モバイル：3］） 11,980円（13ライセンス［内訳：コンピュータ：3、モバイル：10］）

CONTACT
アクロニス・ジャパン㈱ URL http://www.acronis.com/ja-jp

Topic

アカマイ・テクノロジーズ、「XOR DDoS」について調査報告

アカマイ・テクノロジーズ合同会社は9月30日、「XOR DDoS」に関する調査報告を発表した。

XOR DDoSは、Linuxシステムの乗っ取りに使用する、トロイの木馬型マルウェア。アカマイによる調査では、XOR DDoSボットネットによるDDoS攻撃の帯域幅が、わずか数Gbpsから大規模な攻撃となる150Gbps超にわたることが判明した。ボットネットは1日に最大20件の標的を攻撃し、うち90%がアジア地域への攻撃となっている。ゲーム業界がもっとも標的とされやすく、次に教育機関が標的になっているとのこと。攻撃ベクトルとしては現在、SYNフラッド、DNSフラッドの2種類が観測されている。

かつてはWindowsマシンがDDoSマルウェアの標的だったが、攻撃者がLinuxへフォーカスを切り替え、攻撃件数が増えているようだ。

アカマイのサイト（**https://www.stateoftheinternet.com/xorddos**）で、XOR DDos攻撃の低減、該当マルウェア検出方法や、攻撃のペイロードシグネチャ、攻撃トラフィックに合わせたtcpdumpフィルタなどが公開されている。

CONTACT
アカマイ・テクノロジーズ合同会社
URL https://www.akamai.com/enja

Event

サイバーテック、XML・XML DBの国内最大級イベント「NeoCoreサミット」、開催

11月13日（大阪）・11月20日（東京）、XMLやXML DBの普及啓蒙を目的としたイベント「NeoCoreサミット」が、㈱サイバーテック主催で開かれる。

同イベントは今年で8回目で、今回は初めての2拠点での開催となる。イベントの目的は、ドキュメントの生産性向上・利用促進に大きく寄与するXMLやXML DBの普及と啓蒙。今回開催される「NeoCoreサミット2015」のメインテーマは、「Document Revolution元年～脱WORD・脱DTPの実現に向けて」。XMLや、NeoCore[注1]といったXML DBテクノロジの具体的な活用方法や事例など、ドキュメントの制作・管理などに課題を持つユーザに向けたセッションが多数用意される。

●開催概要

	大阪	東京
日付	2015年11月13日（金）	2015年11月20日（金）
時間	15:20～18:00（開場15:00）	
場所	アマゾンデータサービスジャパン㈱大阪支社セミナールーム	アマゾンデータサービスジャパン㈱セミナールーム
参加費	無料（事前登録制）	
定員	120名	

CONTACT
㈱サイバーテック　URL http://www.cybertech.co.jp

注1）サイバーテック社開発のXML DB。XML DBはRDBと比べ、冗長性が高いことが挙げられる。

Report

SAPジャパン、「SAP HANA Cloud Forum Tokyo」、開催

9月18日、DMM.make AKIBA（東京都千代田区）にて「SAP HANA Cloud Forum Tokyo」が開催された。

本イベントでは「これからの時代のアプリケーションとプラットフォームのあり方を考える」をメインテーマとして、慶応義塾大学の夏野剛氏などがセッション・ディスカッションを行った。また、当日はSAPが提供しているPaaS「SAP HANA Cloud Platform（以下、SAP HCP）」[注2]の技術セッションやハンズオンも行われた。

○IT革命の本質と社会構造変化～さらに大きくなるアプリケーションのインパクト（慶応義塾大学政策・メディア研究科特別招聘教授　夏野剛氏）

夏野氏が語ったのは、日本の技術力の高さと、それがうまく活用されていないという現状である。インターネットの普及によって、ビジネスのフロントラインがWebに移行した「効率革命」、情報へのアクセス方法が革新的に変わった「検索革命」、個人の情報発信力が急激に高まった「ソーシャル革命」という3つの革命が起きたが、日本ではそれが経済成長につながっていない。一方アメリカでは、ITを最大限に活用して近年も高い成長率を維持している。日本の技術力は十分に高く、人材・資金ともに潤沢だが、政府の判断の遅さや組織の古いシステムがそれに追いついていない、というのが夏野氏の考え。これからは若い世代・先進的なIT担当者が、経営層に対してIT導入のメリットをいかにマクロに説明して、採用を決めるかが重要だと語った。


▲夏野剛氏

○SAP HANA Cloud Platformとは（SAPジャパン）

このプログラムでは、SAP HCPと活動量計リストバンド「Fitbit」を組み合わせたIoTアプリのデモが行われた。Fitbitを腕につけた人が歩くと、歩数がスマートフォン（Android）を経由してSAP HCPに送られ、ブラウザから確認できるダッシュボード上にリアルタイムで更新される。歩数の情報はSAPの日本オフィス、シンガポールオフィスにいる数個のデバイスから常時送られ、アプリ上で集計・ランキング化されるというしくみ。このアプリは、インターンの学生が1週間ほどで開発したとのことで、開発の手軽さがアピールされた。

◆ ◆ ◆

ハンズオンセッションでは、オンプレミスの基幹システムとSAP HCPを接続したアプリ、デバイスとの連携アプリの開発といった演習が行われた。

CONTACT
SAPジャパン㈱　URL http://www.sap.com/japan

注2）インメモリコンピューティング技術を中核として、データベースサービスとアプリケーションの構築・運用環境を、クラウドサービスとして提供するPaaS。リアルタイム処理、大規模データ解析といった分野にとくに向いている。

ブロケード、NewIPのロードマップでSDN/NFVの価値をアピール

仮想ソフトウェアルータのVyattaの買収をはじめとして、ネットワーク機能のソフトウェア化を急速に推進しているブロケードの目指すところは何か？ SDN/NFVビジネス開発本部 The New IP推進部担当の尾方氏、米国本社のDirector, Product Managementであるジェームス・クオン氏から、SDN/NFVをベースにしたビジネスをどう進めていくのか話を伺った。

Tier1への展開

（クオン氏）以前、データセンターの成長はIT企業の成長の指標になっていました。それが今はクラウドにシフトしています。大手のサービスプロバイダがクラウド事業を推進するために変化したと言えます。そのデータセンターでは仮想化がさらに進んでいます。そこでは拡張性に対する要求も多く、そのためNFVとSDNが欠かせない状況になっています。さらにその流れはTier1、つまり通信業界にも波及しています。AT&Tのようにすでに通信網のIPネットワーク化が進んでいるのは、読者の皆さんご存じのとおりです。NFVを導入することで、サーバ1台あたりの処理密度が上がります。データセンターの効率化はサーバ台数で決まります。ビジネスがフォーカスしているところは、スループットからハードウェアの費用対効果に移っています。

ネットワークをAPIで管理する時代になる

（尾方氏）仮想化の導入が進めば、CLIで行っていたネットワークの管理をAPIで行えるようになります。これによって物理的な制約からの解放が実現されます。容易にスケールアウト可能でしかも高速なデリバリができます。NFV/SDNの導入によってリードタイムの削減ができるのです。レイヤー3からレイヤー7まで、ネットワークを思いのままに操ろうというのがNew IPの大きなメッセージです。そういう状況になったときネットワークエンジニアはコーディングだけでなく、ハイパーバイザの機能を学ぶべきですし、より高度な技術も求められるようになります。

CONTACT
ブロケードコミュニケーションズシステムズ㈱
URL http://www.brocadejapan.com

グレープシティ、モバイルアプリ開発用UIコンポーネント「Xuni」、Visual Studio用コンポーネントセット「ComponentOne Studio」を発売

グレープシティ㈱は、モバイルアプリ開発用コンポーネントセット「Xuni（ズーニー）」の最新バージョン「2015J v2」を9月16日にリリースした。

Xuniはモバイルアプリ開発用に、表や円グラフといった各種ユーザインターフェースをライブラリとして提供する開発支援製品。2015年5月にリリースしたバージョンではXamarinにのみ対応していたが、今回リリースの「2015J v2」では、AndroidおよびiOSアプリにも対応した。これにより、Android StudioやEclipse、Xcodeでの利用が可能になる。開発言語は、C#（Xamarin）、Java（Android）、Objective-C（iOS）、Swift（iOS）の4種類をサポートしている。

Xuniは定額制のサブスクリプション方式で販売されており、配布時のライセンス使用料金は無料。1ユーザライセンスの年間費用は162,000円、2015年12月末までに申し込んだ場合はスタートアップキャンペーン特別価格97,200円（ともに税込）で提供されるとのこと。

Xuniを利用して開発したアプリ「Xuni Explorer」が無料公開されており（**http://www.goxuni.com/jp/demos**）、Xuniの各コンポーネントの機能を確認できる。

同社はまた、Visual Studio用コンポーネントセット「ComponentOne Studio」の最新バージョン「2015J v2」を同日にリリースした。

ComponentOne StudioはWindowsフォーム、ASP.NET Web Forms、ASP.NET MVC、WPF、Silverlight、Windowsストアのアプリを開発できるコンポーネントを数多く収録したスイート製品。プラットフォームごとにデータグリッドやチャート、帳票、コンテナやナビゲーションといったUI部品を提供する。最新バージョンでは、Windows 10をサポートし、Visual Studio 2015や.NET Framework 4.6、ASP.NET 5といった最新環境に対応。

ComponentOne Studioは、1年定額制のサブスクリプション方式で販売されている。初回費用は、最上位エディションである「ComponentOne Studio Enterprise」の1ユーザライセンスで162,000円、1年単位の更新費用は初回費用の40%である64,800円（ともに税込）となっている。

CONTACT
グレープシティ㈱ URL http://www.grapecity.com

Service

ソラコム、
モバイルデータ通信とクラウドを一体化した IoTプラットフォーム「SORACOM」を提供開始

㈱ソラコムは9月30日、IoTデバイスに最適化された、従量課金制のIoTプラットフォーム「SORACOM」のサービス提供開始を発表した。

IoTデバイスから取得したデータを、インターネットを経由してサーバやクラウドに送って処理するIoTシステムの構築のためには、IoTデバイスとサーバとの経路をつなぐ通信ネットワークが必要不可欠。しかし、無線LAN（Wi-Fi）や有線LANは、場所の制約や設置コストの問題がついてまわる。一方、場所を選ばずに通信でき、設置コストのかからないLTE/3G回線を利用したモバイル通信ネットワークは、利便性は高いものの初期投資と通信費が高額で、柔軟な契約が難しいなどの課題があった。

ソラコムはNTTドコモとのMVNO（仮想移動体通信事業者）の契約（L2卸契約）を締結し、NTTドコモの基地局を利用することで、この課題を解決する。モバイル通信のコアネットワーク（パケット交換、回線管理、帯域制御など）とサポートシステム（顧客管理、課金）は、AWSのクラウド上に実装。大量のIoTデバイスからデータ通信が行われても、強固な信頼性と拡張性を持つよう設計されている。そのAWS上のパケット交換機能を、NTTドコモのモバイルネットワークと接続することで、モバイルデータ通信とクラウドを一体化したIoTプラットフォームが「SORACOM」である。

SORACOM上で展開される2つのサービス、「SORACOM Air」および「SORACOM Beam」の概要について紹介する。

SORACOM Air

SORACOM Airは、IoTデバイスに特化したモバイルデータ通信サービス。ユーザは、「SORACOM」サイト（**https://soracom.jp/start**）やAmazon.comからデータ通信SIMカード（ナノ／マイクロ／標準の3種類、それぞれデータ通信のみとSMS機能ありの2種類で、計6種類）を購入して自分のデバイスに挿入する。SIMを挿入したデバイスは、ブラウザもしくはAPIから、通信速度の変更、通信の休止／再開、通信の監視、イベントに応じた処理の設定などを、一括操作できる。数万を超えるIoTデバイスの回線も、集中的に管理できる。

初期費用は契約事務手数料580円／SIM1枚＋送料。基本料金はSIM1枚につき10円／日、データ通信量は0.2円／MBからの従量課金となっている。ダウンロードよりもアップロード、日中よりも深夜時間帯の通信料金が安価に設定されている。ダウンロードをほとんど行わずに小さなデータを定期的にアップロードし続けたり、昼のあいだに貯め込んだデータを夜間に一括でアップロードするようなIoTデバイスに対しては、非常に安価な料金設定になっている。

SORACOM Beam

SORACOM Beamは、データ通信SIMを搭載したIoTデバイスやタブレット、スマートフォンから得たデータに、認証や暗号化、プロトコル変換などを施して転送支援する高機能なサービス。これまで、コンピュータ資源が比較的限られたIoTデバイスにおいて、暗号化などの高負荷処理を実施したり、認証情報の事前設定を行ったりすることは、IoTシステムの構築や運用の難易度を高めていた。本サービスを利用することにより、IoTデバイス側に負担をかけることなく、SORACOM側で、暗号化、プロトコル変換、オンプレミスのサーバやクラウドサービスへの転送が可能になる。

またSORACOMプラットフォームはAWS上で動いているため、SORACOM Beamからのデータを直接AWS上のサービスを利用して処理することもできる。従来IoTシステムの構築には、デバイス、通信、インフラと別々に用意する必要があったが、SORACOM Beamを用いると、セキュアなIoTシステムをAWS上で容易に構築できる。

料金は完全従量課金制で、Beamへのリクエスト、Beamから転送先へのリクエストをそれぞれ1リクエストとカウントし、0.001円／リクエストとなっている。

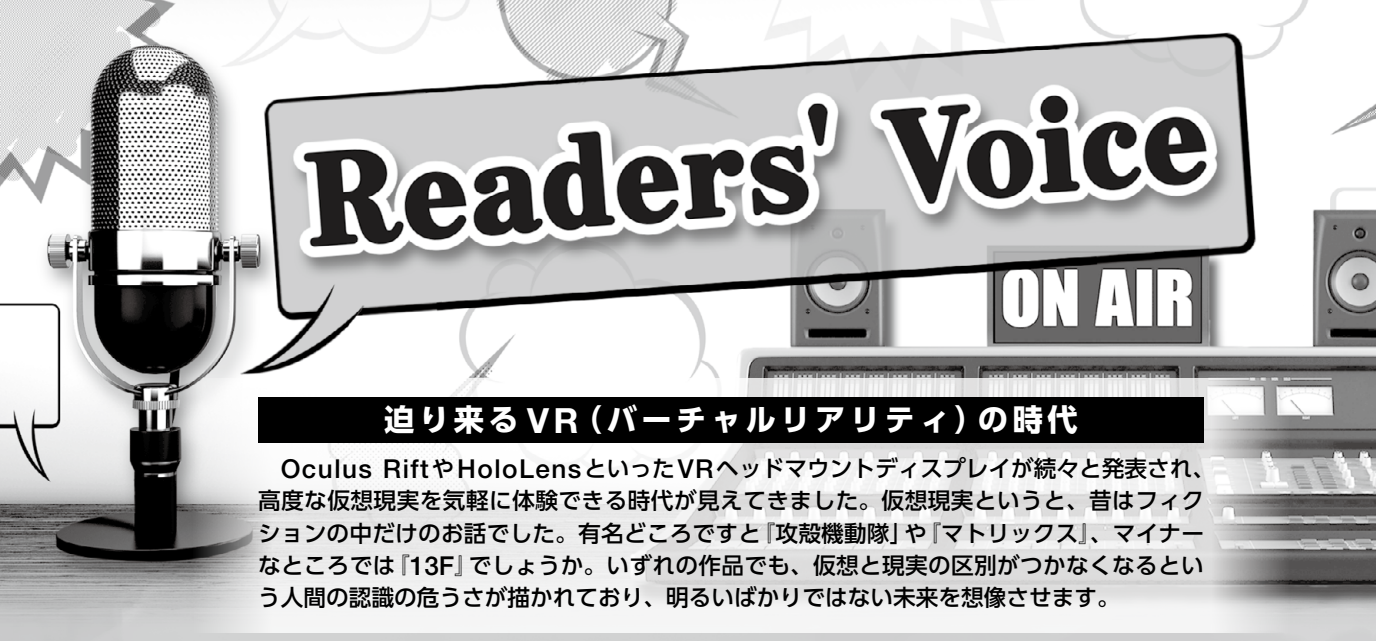
◆ ◆ ◆

開発者向けの情報サイト（**https://dev.soracom.io**）も公開されている。SORACOMの利用ガイド（Getting Started）、API利用ガイド、およびAPIリファレンスを提供する。また、クラウドサービスとの接続ガイドとして、Amazon Web Services連携、IBM IoT Foundation／Bluemix連携のガイド、さらにRubyのCLIを含むライブラリも公開している。

リリースの情報などは、ソラコムの公式ブログ「SORACOM Blog」（**https://blog.soracom.jp**）にて発表されていく予定。

▲右：㈱ソラコム代表取締役社長 玉川憲氏、左：同社最高技術責任者 安川健太氏

CONTACT
㈱ソラコム URL https://soracom.jp



迫り来るVR（バーチャルリアリティ）の時代

Oculus RiftやHoloLensといったVRヘッドマウントディスプレイが続々と発表され、高度な仮想現実を気軽に体験できる時代が見えてきました。仮想現実というと、昔はフィクションの中だけのお話でした。有名どころですと『攻殻機動隊』や『マトリックス』、マイナーなところでは『13F』でしょうか。いずれの作品でも、仮想と現実の区別がつかなくなるという人間の認識の危うさが描かれており、明るいばかりではない未来を想像させます。

2015年9月号について、たくさんの声が届きました。

第1特集　特講　正規表現・SQL・オブジェクト指向

ソフトウェアエンジニアが一度はつまずく3つの技術要素、正規表現・SQL・オブジェクト指向を、問題を解きながら勉強してもらう特集です。

SQLは少しやったことがある程度だったので内容的には難しかったのですが、おもしろかったです。

ももんがさん／静岡県

正規表現は便利なツールですね。ただ、表記のバラツキがあり混乱してしまうことと、自分で書いたものでもあとで見るとすぐに理解するのが難しく、保守性が低く感じるときがあります。機会があれば、保守性を高める記述方法についても取り上げてほしいです。

出玉のタマさん／大阪府

オブジェクト指向が協調動作を含む、とはちょっとおもしろく思った。

藤原さん／奈良県

どちらかというとすでに使い慣れたエンジニアの方から、「良い復習になった」「新しい発見があった」という声が多く寄せられました。実務に追われるなか、手軽に読んで勉強できる点が良かったのかもしれません。

第2特集　メールシステムの教科書

普段から当たり前のように使われる電子メール。その裏側とプロトコル・データ形式・セキュリティについて基本を解説しました。サーバ側とクライアント側双方の技術を学びました。

メールシステムについて、懐かしく思えたのは……最近、あまり管理していないから！？

南雲さん／埼玉県

第3章で例として紹介されていた「Mozilla Thunderbird」を自宅で使っています。無料だからなんとなく使っていましたが、高速化のための工夫として索引ファイル「.msf」ファイルが使われていると初めて知りました。これからは感謝して使おうと思います。

永作さん／東京都

メールのおさらい、すっきりした方も多いのではないかと思います。

赤間さん／大阪府

安定した伝送技術だからこそ、今日まで広く利用されている電子メール。当たり前過ぎて、中身についてはあまり詳しくないという人が多かったようです。

特別企画　なぜ俺の提案は通らないのか？

若いエンジニアに、会社にまつわるお金、経営のルール、事業計画について学んでもらう特別企画。なぜ提案が通らないのか、に答える形で提案の練り方を一から説明しました。本誌では珍しい、技術要素のまったくない記事でした。

技術者でも提案をしていかないとだめな世の中になっていくと思う。

tekitoizmさん／東京都

耳が痛いところがあった。

yuuiさん／兵庫県

SDでお金の記事は珍しいでしょうか？初めて読んだ気がします。私はエンジニアながら簿記の研修を受けたこともあって、たいへんおもしろく思いました。架空のITサービスを例に出した説明がわかりやすく、良かったです。そこまで深くなくても良いので、もう少し具体的なサービス例をいくつか出したうえで、連載化してほしいです。お金の感覚は大事だと思います。

Junkさん／神奈川県

9月号のプレゼント当選者は、次の皆さまです

①「Raspberry Pi 2」&「Camera module」セット
ふじ様（神奈川県）

②USBチャージ付き雷ガードタップ「P3U3-JP」
内藤亮介様（東京都）

③はてなTシャツ2015（Mackerel）
あかび。様（宮城県）、檜垣賢様（大阪府）、sunao様（大阪府）

④『The Art of Computer Programming Volume 1』
吉田和人様（東京都）、茂木和弘様（栃木県）

⑤『できるPRO Red Hat Enterprise Linux 7』
近藤元一様（山形県）、山下修様（東京都）

⑥『クラウドを支えるこれからの暗号技術』
牧秀亮様（大阪府）、石内博子様（福岡県）

⑦『Dockerエキスパート養成読本』
朝田貫太様（兵庫県）、大西尚利様（神奈川県）

※当選しているにもかかわらず、本誌発売日から1ヵ月経ってもプレゼントが届かない場合は、編集部（sd@gihyo.co.jp）までご連絡ください。アカウント登録の不備（本名が記入されていない場合、プレゼント応募後に住所が変更されている場合など）によって、お届けできないことがあります。2ヵ月以上、連絡がとれない場合は、再抽選させていただくことがあります。

エンジニアが経営の視点を持つこと、そして非エンジニアが技術について学ぶことの重要性を、最近いろいろな場所で聞くようになりました。知識の交流を行って仕事の無駄をなくすというのが、大きな目的のようです。

短期連載　Jamesのセキュリティレッスン【4】

新しいファイル形式「pcap-ng」も使えるようになったパケットキャプチャ「Wireshark」の使い方を紹介する短期連載です。今回はそのpcap-ngファイルの中身を、pcapファイルと比較しながら1つずつ見ていきました。

とても良い。ぜひ短期ではなく長期連載してほしい。
blackbirdさん／東京都

パケット構造は見ていて楽しい。
笹倉さん／千葉県

Wiresharkは、いつも同じ機能ばかり使うので、ほかの機能を試そうと思わせてくれるきっかけはありがたいです。
下平さん／東京都

Jamesのセキュリティレッスンは以前読んで続けてほしかったので、再開されてよかったです。pcap-ng以外にもWireshark関連で記事を取り上げてほしいです。
今井さん／千葉県

2014年11月号～2015年1月号まで連載していた「Jamesのセキュリティレッスン」ですが、今回からまた3号連続で連載します。再開を喜ぶ声が多く寄せられました。

短期連載　DevOpsで始めよう！モダンなJavaアプリケーション開発【2】

「ユーザの意見を積極的に取り入れ、1日に何百回もデプロイする」。Javaを使って、そんな"イマドキ"の運用を実現する短期連載です。第2回では、MavenとGitHubを使った安全なテストとスピーディなデプロイを紹介しました。

最近Javaの勉強をしているので、タイムリーで参考になりました。
入日さん／神奈川県

DevOpsにはちょうど興味があったので勉強になった。
ＹＹさん／神奈川県

「DevOps」という言葉はよく聞いても、実態はいまいちわからないという人が多いと思います。今回は具体的なツールを使って設定から解説したことで、勘所をつかめたのではないでしょうか。

短期連載　Kotlin入門【6】

Java仮想マシン上で動作するオブジェクト指向言語「Kotlin」についての短期連載。最終回では、Androidアプリの開発全般について紹介しました。

最終回、お疲れさまでした。
エゾモモンガさん／滋賀県

Kotlin 1.0がリリースされたらプロダクトに使いたいです。
binaさん／東京都

6回にわたって連載した「Kotlin入門」でしたが、今回で最終回です。ほかの媒体ではなかなか見られないKotlinという言語に注目した本連載は、毎月読んでくれていた読者の方が非常に多かったです。

コメントを掲載させていただいた読者の方には、1,000円分のQUOカードをお送りしております。コメントは、本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」からアクセスできるアンケートにてご投稿ください。みなさまのご意見・ご感想をお待ちしています。

次号予告

Software Design
December 2015

2015年12月号
定価（本体1,220円+税）
192ページ

11月18日発売

［第1特集］最新情報をキャッチアップ!

［決定版］Docker自由自在

実用期に入ったLinuxコンテナ技術

自由度の高いネットワークとシステム環境を実現できるDockerは注目を浴び続けています。Linuxコンテナ技術を実践するにあたり、すでに先端ユーザが使用して実績を上げているツール群などすみからすみまで紹介します。

■序　章　Dockerを取り巻く環境――HashiCorpが目指すもの
■第1章　Dockerのキホン――アーキテクチャとイメージ管理
■第2章　Dockerのオーケストレーション――Docker Machine/Docker Swarm
■第3章　Dockerの高度な操作――Docker API/Docker Trusted Registry/Docker Plugins
■第4章　Dockerの自動化ツール群――Vagrant/Packer/Terraform/Serf/Consul/Vault/Otto/Nomad
■第5章　Dockerツールを連携する――Atlas/Otto完全解説

［第2特集］ネットワーク・システム管理はおまかせ!

SNMPの教科書

基礎からわかる・すぐ応用できる

※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

「Debian Hot Topics」（第30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10月号 特別付録　Vimチートシート「モーション」欄、下から4行目
［誤］fx　　［正］tx
■2015年10月号　連載「Linuxカーネル観光ガイド」第43回
●P.170図3とP.173図7の解説図の中身が入れ違っておりました。

SD Staff Room

●機械学習理論の本を作っているんだけど、中身は確率論と偏微分と行列計算が満載。もう四半世紀前に使っていたものなので、忘却の彼方。でもがんばって作った。多くの入門本が避けていた部分をしっかりと、しかも確実に解説しているのは『ITエンジニアのための機械学習理論入門（中井悦司 著）』だけだ。お勧め!（本）

●9月上旬に遅い夏休みをとって、伊勢神宮→奈良→京都と巡ってきました。台風と重なり、おかげ横丁、石舞台や酒船石や大仏殿、平等院や伏見稲荷などカメラを出せないほどの大雨（涙）。しかし奈良や京都の外国人（主に中国）が多いこと。大丈夫か!?日本。めげずに京都で漬物をたんまり買い込んできました。（幕）

●フモフモさん（抱き枕）が我が家に。このメーカーのお客さん目線がすごい。ぬいぐるみは愛するほどに表面が汚れ、中のワタがかたよるので、洗濯やワタの交換というアフターサービスがある、だけじゃないんです!「入院中の我が子を撮影してくれる」んです!はー、いやされるわ。あ、もちろん娘用ですよ。（キ）

●専門学校時代の同級生の何人かが他社の雑誌編集部に勤めています。書店に行ったときには、彼らが作っている雑誌を手に取り、奥付に彼らの氏名が載っていることを確認しています。ちょっとした安否確認です。ついでに役職や氏名の掲載順序なども見て、出世したかどうかもチェックしています。（よし）

●秋葉原にある『スーパーポテト』というお店、ご存じですか?　レトロゲーのソフトやハードを専門に扱うゲーム屋さんで、最上階には駄菓子を食べながら昔の大型筐体が遊べるゲームスペースもあったりします。友人と久しぶりに『スパⅡ』で対戦してみると、“波動拳”がなかなか出なくて焦りました。（な）

●約4年在籍していたアシスタントから卒業することに。初めて書いた編集後記を見ると、Macを触ったこともなく、閉じるボタンの位置に戸惑っていたことが書いてあり、不慣れだったことを懐かしく思えるほどには成長できたみたいです。その後は編集後記といいながら、日記みたいな内容ばかり……。（ま）

本誌に記載の商品名などは、一般に各メーカーの登録商標または商標です。　© 2015 技術評論社

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［E-mail］
sd@gihyo.co.jp

［FAX］
03-3513-6173

※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

Software Design
2015年11月号

発行日
2015年11月18日

●発行人
片岡　巌

●編集人
池本公平

●編集
金田富士男
菊池　猛
吉岡高弘
中田瑛人

●編集アシスタント
松本涼子

●広告
中島亮太
北川香織

●発行所
㈱技術評論社
編集部
TEL：03-3513-6170
販売促進部
TEL：03-3513-6150
広告企画部
TEL：03-3513-6165

●印刷
図書印刷㈱

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

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