Special Feature 01
Emacs活用と上達
Special Feature 02
最強Fusion-ioの秘密
2013 November
11
Software Design
2013年11月18日発行
毎月1回18日発行
通巻343号
（発刊277号）
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価
1,280円
Mega Special in Autumn !!

[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

特盛
「秋の
大増
ページ!」

思考を
コード化
する
道具
エディタ

Emacs is

我が友 Emacs

Special Feature 01

設定ファイルの行数は覇気に比例する?!

My Best Friend.

コンピュータの加速装置

Special Feature 02

サーバサイドフラッシュ Fusion-io全方位解説

サーバマシン・データセンターを支えるテクノロジー

Article

ソーシャルゲームのDevOpsを支える技術
魔法少女リリカルなのはINNOCENTの舞台裏（前編）

新連載

OS/Network

後藤大地の渾身執筆
FreeBSD活用「チャーリー・ルートからの手紙」

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。

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。

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。

第59回

IT エンジニア必須の 最新用語解説

TEXT：㈲オングス 杉山 貴章 SUGIYAMA Takaaki
takaaki@ongs.co.jp

テーマ募集
本連載では、最近気になる用語など、今後取り上げてほしいテーマを募集しています。sd@gihyo.co.jp 宛にお送りください。

HTTP 2.0

HTTP 2.0 に至る経緯

「HTTP 2.0」（プロトコルの正式な識別名は「HTTP/2.0」）は、IETFのHTTPbis WGによって策定が進められている次世代のHTTP規格です。現在はまだドラフトの段階にありますが、将来的には現行のHTTP 1.1に代わってWebを構成する基盤技術となるもので、HTTP 1.1が抱えるさまざまな問題を解決するべく議論が重ねられています。

HTTP 2.0に関する検討が正式に始められたのは2012年で、その背景には、昨今のWebの急激な進化にHTTP 1.1では対応し切れなくなってきたという事情があります。ハードウェア性能や通信速度の向上に伴って、Webサイトにおけるコンテンツの大容量化も急速に進んでいます。既存のHTTPはそのような大容量のコンテンツの扱いに最適化されておらず、アプリケーションのパフォーマンスに悪影響を及ぼすケースが出てきました。

HTTP 1.1の具体的な問題点としては、1接続につき1つのリクエストしか処理できないことや、ヘッダフィールドが冗長でサイズが大きいことなどが挙げられています。リクエストの多重化については、同時接続数を増やすなどといった対策を取ることが一般的ですが、これにはサーバやネットワークの負荷が上昇するというデメリットがあります。

このような問題を解決する目的で登場したのが、本誌2012年4月号で紹介した「SPDY」です。SPDYはGoogleによる独自のプロトコルですが、すでにいくつかのWebサーバやWebブラウザが対応するなど、徐々に実用レベルに近づいてきています。

HTTP 2.0に関する議論は、このSPDYをきっかけとして始まりました。HTTPにSPDYの要素を取り込むことで、HTTP 1.1の問題を解決する新しい標準仕様を作ろうという取り組みがスタートしたわけです。このような経緯からHTTP 2.0ではSPDYの考え方を多く引き継いではいますが、決してSPDYをそのまま標準化しようというわけではなく、議論が進むにつれて少しずつ独自の姿を見せ始めています。

HTTP 2.0 で何が変わるか

HTTP 2.0は、既存のWebベースのアプリケーションとの互換性を可能な限り保持するように設計されます。アプリケーションのリクエストとレスポンスのしくみについては従来どおりの方式が保たれるため、GET/PUTをはじめとするHTTPメソッドは引き続き利用できます。

大きく変わるのは転送のしくみです。HTTP 2.0にはおもに次のような内容が取り込まれる予定になっています。

- **リクエストとレスポンスの多重化**
  ストリームの概念が導入され、1つの接続の中で複数のリクエストとレスポンスを並列的に取り扱うことが可能になる
- **バイナリフレーム**
  リクエストとレスポンスは固定長のフレームにエンコードされ、通信はフレーム単位で行われる
- **ヘッダの圧縮**
  ヘッダフィールドはヘッダ用に最適化された符号化方式で圧縮してフレーム化される。圧縮方式としてはHTTP/2.0 Header Compression（HPACK）という新しい仕様が検討されている
- **サーバプッシュ**
  クライアントからのリクエストを待たずにレスポンスを送るサーバプッシュ型の通信がサポート
- **Upgrade ヘッダによるネゴシエーション**
  サーバがHTTP 2.0に対応しているか不明な状態でhttpリクエストを送る場合には、まずHTTP 1.xで通信を開始する。その際にUpgradeヘッダにHTTP/2.0を指定すれば。サーバがHTTP 2.0に対応していれば、以降の通信はHTTP 2.0に切り替わる
- **TLS-ALPN によるネゴシエーション**
  サーバがHTTP 2.0に対応しているか不明な状態でhttpsリクエストを送る場合には、TLS-ALPNによるネゴシエーションによって使用するプロトコルを決定する。TLS-ALPNはクライアントから使用可能なプロトコルを提示し、サーバ側でその中から選択する方式で、TLS WGで仕様策定が行われている

HTTP 2.0の標準化作業は2014年春の仕様化完了を目指して進められています。仕様自体はGitHub上で管理されており、標準化が進む様子をリアルタイムに確認することができるようになっています。SD

最新のドラフト
https://datatracker.ietf.org/doc/draft-ietf-httpbis-http2/
GitHub リポジトリ
https://github.com/http2/

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。

Contents

Software Design Nov. 2013

1

第1特集

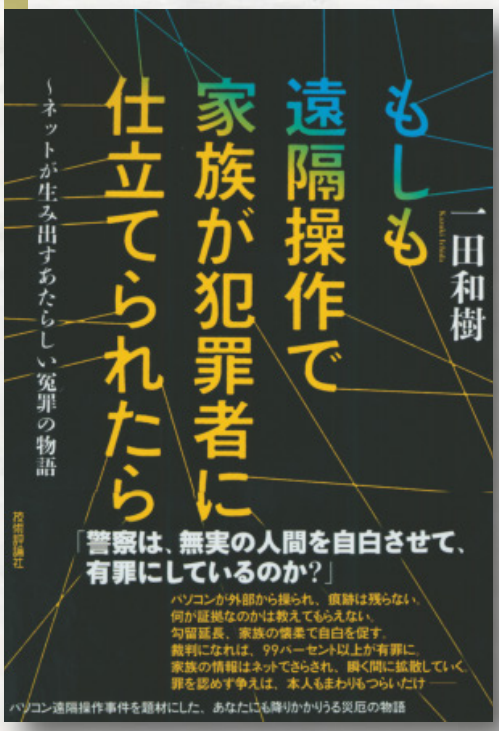
我が友 Emacs

思考をコード化する道具（エディタ）

Special Feature 01

設定ファイルの行数は覇気に比例する?!

ネットが生み出す　あたらしい冤罪の物語



もしも遠隔操作で家族が犯罪者に仕立てられたら

パソコンが外部から操られ、痕跡は残らない。
何が証拠なのかは教えてもらえない。
罪を認めないと、勾留延長、家族対策で自白を促す。
裁判になれば、99 パーセント以上が有罪に。
家族の情報はネットでさらされ、瞬く間に拡散していく。
罪を認めず争えば、家族も本人もつらいだけ ――

パソコン遠隔操作事件を題材にした、
あなたにも降りかかりうる災厄の物語。

一田和樹　著／四六判／ 288 ページ
定価 1,659 円（1,580 円＋税）
ISBN978-4-7741-6004-7

Linuxサーバエンジニア　必携の一冊！

最短突破 LPIC レベル 1 問題集

［101 試験／ 102 試験　Version 3.5 対応］

LPIC レベル 1 に対応した、最新の対策問題集です。
Version 3.5 の出題範囲に対応し、101 試験と 102 試験の両方の試験範囲を 1 冊で網羅。実績ある講師による厳選された練習問題を繰り返し解くことで，資格取得に必要な力が自然と身に付いていきます。解答・解説ページでは、「なぜ正解なのか」はもちろんのこと、「なぜ誤りなのか」も丁寧に解説しています。苦手な分野や科目を克服し、得点力がアップすること間違いなし！
LPIC レベル 1 の取得を目指すすべての人にお勧めできる、一番新しい問題集です。

河原木忠司　著／ A5 判／ 520 ページ
定価 3,129 円（2,980 円＋税）
ISBN 978-4-7741-5949-2

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

ソフトウェア・デザイン［目次］

Contents

Software Design Nov. 2013

2

第2特集 Special Feature 02

コンピュータの加速装置

サーバサイドフラッシュFusion-io全方位解説

サーバマシン・データセンターを支えるテクノロジー

一般記事 Article

巻頭Editorial PR Editorial PR

SD Special Report SD Special Report

アラカルト A La Carte

広告索引 AD INDEX

広告掲載企業への詳しい資料請求は、本誌Webサイトからご応募いただけます。 http://sd.gihyo.jp/

言語のセオリーを徹底解説

パーフェクトシリーズ

[改訂3版] パーフェクトC#

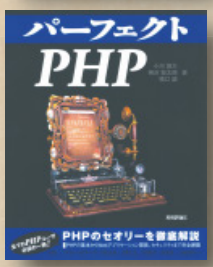
斎藤友男、醍醐竜一　著／B5変形判／608ページ　定価3,780円（本体3,600円）
ISBN 978-4-7741-5680-4

C#で.NET開発を行う人へのバイブル的1冊です。概要／基礎から実践までを幅広く学習でき、C#を扱ううえで知っておきたい知識は、この一冊に網羅されています。基本文法，Webアプリケーション開発はもちろん、C#5.0から可能になったWindowsストアアプリケーション開発の実践方法から、C#、.NETの内部動作まで幅広いテーマをあつかっており、この一冊でC#の知識は完璧といえる内容をめざします。C#5.0に対応。

パーフェクトJava

井上誠一郎、永井雅人、松山智大　著／B5変形判／640ページ　定価3,780円（本体3,600円）
ISBN 978-4-7741-3990-6

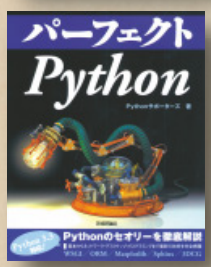
本書はJavaで開発を行う人へのバイブル的1冊です。Javaの基本から説明していますが、プログラミング一般の考え方や技法まで解説しています。なぜそれらが必要なのかを説明しながら、様々な技法やパターンについて、考え方や背景を含め理解できることを目的とした書籍です。本書の構成は3つのPartに分かれています。Part1でJava言語の基礎を、Part2はサーバプログラミング、Part3はJava GUIと代表的な応用分野の解説をしています。Part2とPart3は実践的な解説に力点を置いています。



パーフェクトPHP

小川雄大、柄沢聡太郎、橋口誠　著／B5変形判／592ページ　定価3,780円（本体3,600円）
ISBN 978-4-7741-4437-5

1冊で言語仕様から最新の技術までを網羅した内容。網羅的に解説されているだけでなく、各技術に関しては基本からしっかり解説し、フレームワークなどを利用したWebアプリケーション開発の解説などは、内部処理が裏で何をしているのかを掘り下げて解説してあるため、PHPを体系的に学びたい方はもちろん、より深い知識を得たい中～上級者にもお勧めの一冊です。



パーフェクトPython

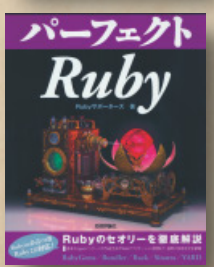
Pythonサポーターズ　著／B5変形判／464ページ　定価3,360円（本体3,200円）
ISBN 978-4-7741-5539-5

1冊で言語仕様から最新の技術までを網羅した内容です。網羅的に解説されているだけでなく、各技術に関しては基本からしっかり解説し、必要な箇所では、内部処理が裏で何をしているのかを掘り下げて解説してあるため、体系的に知りたい初心者はもちろん、中級者にもお勧めの一冊です。最新のPython3.3に対応。

パーフェクトJavaScript

井上誠一郎、土江拓郎、浜辺将太　著／B5変形判／544ページ　定価3,360円（本体3,200円）
ISBN 978-4-7741-4813-7

1冊で言語仕様から最新の技術までを網羅した内容です。本書はJavaScriptで本格的なWebアプリケーションを作りたい人を対象に、前半でJavaScriptの言語仕様を掘り下げて解説し、後半で今求められるJavaScriptの応用分野として、クライアントサイドJavaScript、HTML5、Web APIの利用、サーバサイドJavaScriptの解説を丁寧に行っています。



パーフェクトRuby

Rubyサポーターズ　著／B5変形判／640ページ　定価3,360円（本体3,200円）
ISBN 978-4-7741-5879-2

本書は1冊でRubyの言語仕様から最新の技術までを網羅した内容となっています。また、網羅的に解説されているだけでなく各技術に関しては基本からしっかり解説しており、体系的に知りたい初心者はもちろん中級者以上の方にもRubyを書く際に手元に置いておくと重宝する内容です。
Ruby 1.9.3とRuby 2.0対応。

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

ソフトウェア・デザイン［目次］

Contents

Software Design Nov. 2013

3

Column

Development

OS/Network

Inside View

Logo Design ロゴデザイン ＞ デザイン集合ゼブラ+坂井 哲也
Cover Design 表紙デザイン ＞ Re:D
Cover Photo 表紙写真 ＞ Frank Greenaway/Getty Images
Illustration イラスト ＞ フクモトミホ、高野 涼香、中川 悠京
Page Design 本文デザイン ＞ 岩井 栄子、近藤 しのぶ、SeaGrape、安達 恵美子
［トップスタジオデザイン室］轟木 亜紀子、阿保 裕美、佐藤 みどり
［BUCH+］伊勢 歩、横山 慎昌、 森井 一三、Re:D、［マップス］石田 昌治

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

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

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

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

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。

http://gihyo.jp/dp

GIHYO Digital Publishing 未来の"普通"を、今。

Google play

amazon.co.jp

※販売書店は今後も増える予定です。

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

紙面版
A4判・16頁
オールカラー

電脳会議

DENNOUKAIGI

一切無料

新規購読会員受付中!

『電脳会議』は情報の宝庫、世の中の動きに遅れるな!

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

今が旬の情報満載!

新規送付のお申し込みは…

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

電脳会議事務局　検索

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

一切無料!

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

毎号、厳選ブックガイドも付いてくる!!

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

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

Software Design plus

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

OpenFlow実践入門
高宮 安仁、鈴木 一哉 著
定価 3,200円+税　ISBN 978-4-7741-5465-7

はじめてのOSコードリーディング
青柳 隆宏 著
定価 3,200円+税　ISBN 978-4-7741-5464-0

Webサービスのつくり方
和田 裕介 著
定価 2,180円+税　ISBN 978-4-7741-5407-7

Androidアプリケーション開発教科書
三苫 健太 著
定価 3,200円+税　ISBN 978-4-7741-5189-2

プロのためのLinuxシステム・10年効く技術
中井 悦司 著
定価 3,400円+税　ISBN 978-4-7741-5143-4

サーバ/インフラエンジニア養成読本 管理・監視編
Software Design編集部 編
定価 1,980円+税　ISBN 978-4-7741-5037-6

サーバ/インフラエンジニア養成読本 仮想化活用編
Software Design編集部 編
定価 1,980円+税　ISBN 978-4-7741-5038-3

業務に役立つPerl
木本 裕紀 著
定価 2,780円+税　ISBN 978-4-7741-5025-3

Apache[実践]運用/管理
鶴長 鎮一 著
定価 2,980円+税　ISBN 978-4-7741-5036-9

プロになるためのデータベース技術入門
木村 明治 著
定価 3,180円+税　ISBN 978-4-7741-5026-0

データベース技術[実践]入門
松信 嘉範 著
定価 2,580円+税　ISBN 978-4-7741-5020-8

2週間でできる! スクリプト言語の作り方
千葉 滋 著
定価 2,580円+税　ISBN 978-4-7741-4974-5

PCのウイルスを根こそぎ削除する方法
本城 信輔 著
定価 1,980円+税　ISBN 978-4-7741-4867-0

Androidエンジニア養成読本
Software Design編集部 編
定価 1,880円+税　ISBN 978-4-7741-4859-5

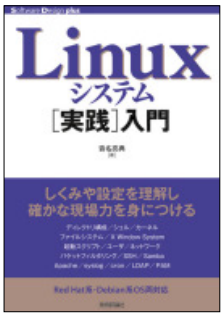
Vyatta入門
近藤 邦昭、松本 直人、浅間 正和、大久保 修一(日本Vyattaユーザー会) 著
定価 3,200円+税　ISBN 978-4-7741-4711-6

プロのためのLinuxシステム・ネットワーク管理技術
中井 悦司 著
定価 2,880円+税　ISBN 978-4-7741-4675-1

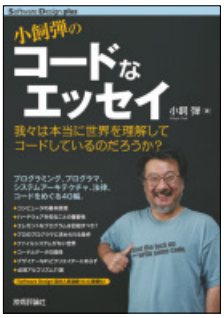
ニコラ・モドリック、安部 重成 著
A5判・336ページ
定価 2,780円(本体)+税
ISBN 978-4-7741-5991-1

水野 操、平本 知樹、神田 沙織、野村 毅 著
B5判・128ページ
定価 2,480円(本体)+税
ISBN 978-4-7741-5973-7

中井 悦司 著
B5変形判・384ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-5937-9



沓名 亮典 著
A5判・416ページ
定価 2,880円(本体)+税
ISBN 978-4-7741-5813-6



小飼 弾 著
A5判・200ページ
定価 2,080円(本体)+税
ISBN 978-4-7741-5664-4

青木 直史 著
A5判・288ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-5522-7

Japanese Raspberry Pi Users Group 著
B5変形判・256ページ
定価 2,380円(本体)+税
ISBN 978-4-7741-5855-6

菅野 裕、今田 忠博、近藤 正裕、杉本 琢磨 著
B5変形判・336ページ
定価 3,200円(本体)+税
ISBN 978-4-7741-5567-8

河村 嘉之、川尻 剛 著
B5変形判・480ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-5438-1

㈱マピオン、山岸 靖典、谷内 栄樹、本城 博昭、長谷川 行雄、中村 和也、松浦 慎平、佐藤 亜矢子 著
B5変形判・256ページ
定価 2,580円(本体)+税
ISBN 978-4-7741-5325-4

データサイエンティスト養成読本編集部 編
B5判・152ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-5896-9

Software Design編集部 編
B5判・176ページ
定価 1,880円(本体)+税
ISBN 978-4-7741-5888-4

データベースエンジニア養成読本編集部 編
B5判・136ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-5806-8

WINGSプロジェクト 著
B5判・352ページ
定価 2,580円(本体)+税
ISBN 978-4-7741-5611-8

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

ネットワークエンジニア虎の穴

『自宅ラックのススメ』

文／tomocha（http://tomocha.net/diary/）

第7回 サーバの選び方

イラスト：高野涼香

タワー（デスクトップ）型、それともラックマウント式？

今回はサーバ選びについて超軽く書いてみましょう。

自宅で使うサーバはどういうモノを選んでいますか？　NTT-X Storeなどで出ている1万円前後のタワー型のサーバを選んでいますか？　それとも、Yahooオークションやe Bay、東京だと秋葉原にある「コンピューターのおっと[注1]」などで考えて、ラックマウント式のサーバを選んでいますか？

筆者にとっては、いずれも貴重な購入ルートですが、NTT-X Storeで販売されるHP ML110/115シリーズやNEC Express/S70などは、サーバとしてはあまり買いません。どちらかというと、ちょこっとした検証用かデスクトップ用途でしょうか。おもにサーバとして使うのは、ラックマウント式のものが多いですね。

前者のメリットは、比較的FANの音が静かということと、そこそこの自由が利くということで手元にあってもあまり困りませんが、サーバラックに搭載するためには、棚板などが必要になることなど、収容スペースに対するメリットがまったくありません。それならば「もう、いっそのこと中古でも良いのでは」と思いはじめ、一定の世代以降のモノをサーバ用に選定しています。簡単に独断と偏見に満ちあふれた筆者の基準について、本記事では紹介してみましょう。この基準ならば、結構手軽に導入できると思います。サーバ用途として使用するのであれば、NICは最低2ポート欲しくなることや、メモリをたくさん搭載したい、といったニーズもでてくるでしょう。また、ラックに搭載する楽しさ、すばらしさ、ロマンなども満たされるでしょう。タワー型（デスクトップ型）の激安サーバではこのあたりは満たされないと思っています。

注1）http://www.ottoserver.com/

CPUの型番をチェックしていますか？

決して筆者はIntelの回し者ではありませんが、中古で買うのであれば、Intel Xeon搭載のNehalem（5500番台）以降のが良いと思います。それ以前のHarpertown（5400番台）については、非常に安価に出回っていますがメモリの入手性が良くなく、FB-DIMMの採用、またDDR2であること、性能、そして消費電力や排熱などをふまえるとメリットがありません。現在それら各社製品はリプレース中のものになるでしょう。おそらくあまり欲しがる人はいないと思います。

Nehalem世代から、未使用のコアへの電流の遮断、アクティブなコアへの電流の供給といったTurbo Boost（自動的に定格の動作周波数より高速で動作させる機能）など電力管理の面は、このあたりから良くできていますし、仮想化に着目されはじめた時期のモデルですので、自宅で手軽に仮想化環境を構築するにもお勧めです。また、そろそろ3～5年前後でリプレースがぼちぼちと出てきていることから狙いどころではないでしょうか。CPUクロック、性能を非常に要求する使用の仕方でない限り、とてもお勧めだと思います。値段についても、激安サーバを買って、

メモリやCPUの換装などを行うのとあまり変わらない程度の価格帯で入手できるでしょう。また、こうしたサーバの場合、マシンルームなどで使用されることが多く、さらに外気を使ったデータセンターが流行る少し前のモデルですので、サビや埃など傷んでいるケースも少ないです。ただし、消耗品だと考えられるHDDだけは、ちゃんと新品の物を手配しましょう（SASディスクの場合は結構長生きしますけど……）。

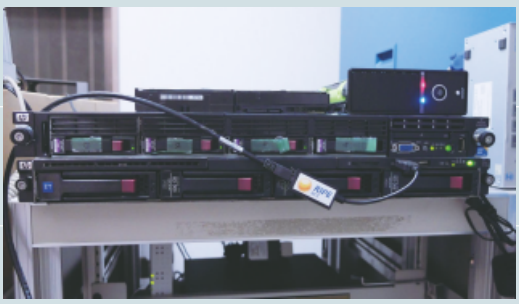
メーカー製のほうが良いか?

その他中古で購入する場合は、ホットスワップ対応のシャーシかどうか、ディスク搭載可能本数、空きベイに対するマウンタの有無・入手性、対応するインターフェース（SATA/SAS）、ディスクのサイズ（2.5/3.5インチ）、ラックマウントレールがあることも重要ですね。あとは、メモリの規格（unbuffered、ECC対応、Registered対応）などいくつか種類がありますので、このあたりは入手性、搭載したいメモリ容量、CPUの構成、お値段などで要確認です。メーカー製の場合は、構成ガイドといった資料で確認ができます。

欲を言えば、Westmere以降（5600番台）や、Sandy Bridge（E3、E5シリーズ）世代も狙いどころですが、なかなか手出しのできる価格帯ではありませんので、まだまだ先のようです。――ていうか、まだ後者は現行モデルですね。

こういった条件から、メーカー製のサーバを選んでいます。メーカー製サーバの良いところは、サーバの電力管理機能、ハードウェアの管理機能、IPMI/BMCなどが充実していること、FANの制御などよくできているためですね。あとは、自分のニーズにあったものを選んでみましょう。

▼写真1　レール固定をしておきましょう



もちろんレールに固定しましょう!

そんなわけで、筆者の環境では、Nehalem以降のものにし、古い機種はなるべくリプレースをしています。

参考に自宅ラックの写真を出しておきますが、決してキレイな環境ではなく、邪魔だから積み上げているといったダメなパターンです。なお、レールで固定しないと、地震があったときに落下する可能性があるので、真似をしてはいけません（写真1）。

レールがない場合や、ケージナット対応でない場合、棚板の上に積み上げる形になってしまいます（写真2）。個人的には、最近発表されたIntel ATOMのAvotonやRangeleyあたりがとても気になっているこのごろです。これらならば省スペース、省エネなお手軽サーバが期待できそうです。

おわりに

さて、筆者は転職先を常時募集中です！　もし採用いただければ、役に立たないかもしれないノウハウ、AS番号などももれなく一緒についてきます！ SD

▼写真2　ラックの中に積み上げてしまう

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

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

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

DIGITAL GADGET

安藤 幸央 —— Yukio Ando ——
EXA CORPORATION
[Twitter] >> @yukio_andoh
[Web Site] >> http://www.andoh.org/

Volume 179 >>>>>

コンピュータグラフィックスの祭典 SIGGRAPH 2013[後編]

～ディズニーランドの街アナハイムで出会った アート、デジタルガジェット編

メイカーズブームも後押し。工夫と実用性が目立った展示

2013年7月21～25日の5日間、コンピュータグラフィックスとインタラクティブ技術に関する世界最大の学会・展示会である、第40回 SIGGRAPH 2013が開催されました。先月号に続き、デジタルガジェット的視点でとらえた数々の展示・発表をご紹介しましょう。

昨今のメイカーズブームで、3Dプリンタへの注目が広く一般からも集まるようになってきました。もちろんSIGGRAPHでは、ブームになるはるか昔から3Dプリンタ製品や、それにまつわる技術、サービスなどは知られていました。さらにこのブームに乗って、多岐にわたったきめ細やかな技術が発展し、今まで培ってきた3次元CG技術の応用や活躍の場がますます広がってきたと感じました。

また、デジタルカメラやビデオカメラの性能もここ数年で飛躍的に向上しました。スマートフォンに搭載されるカメラから映画撮影に用いるハイエンド製品まで、撮影された写真・映像素材をいかに便利に素早く活用するかといった、実用面での研究も進んでいます。

数多くの基礎研究、実用研究、応用研究がなされているSIGGRAPHですが、より良い研究のためには、いくつかポイントがあると言われています。

- 解くべき的確な問題を見つけ出し、それを自分にしかできない方法で的確に解決すること
- 何かテーマや課題に迷ったら、既存の技術を逆にするだけでも意味のあるものになること（たとえば、カラーをグレイスケールにする技術があれば、それを逆に実行できる技術を開発するなど）
- 経験のある人と一緒に研究し、製品や作品をすでに作っているグループと一緒に作業すると良いこと
- 過去の研究は可能な限り調べ、可能な限りたくさんの論文を読む。そこ

« Iron Man 3のメイキングセッション会場に持ち込まれた1/3サイズのフィギュア

» 先進技術を紹介するEmerging Technologiesの一角。水面プロジェクションAquaTop Displayの展示

« 会場に浮かぶProject Skyeの巨大風船。iPhoneのアプリから動きや搭載カメラをコントロールできる

» 素材や道具が用意され、デジタルからアナログまでさまざまな物作りに挑戦するSTUDIO参加者の様子

※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。

にヒントや解決策があること

- エレガントで完璧な研究なんてないのだから、とにかく考え、時間をかければ新しい知見がもたらされること
- 体調管理も大切。創造性は健康度に比例する。閉じこもっていないで、外に出て人と話そう！

研究と実用の密接な関係。SIGGRAPHの先進技術展示

PAPILLON : Expressive Eyes for Interactive Characters

ディズニー・リサーチが展示していた「PAPILLON」は、感情によって目の表情や瞳が変化する、インタラクティブなキャラクター研究です。目の場所に位置するピンポン球のような眼球に、マンガ風の瞳と眉毛が表示されます。

うれしいときには、逆U字、悲しいときには、ハの字眉毛、困ったときには絵文字の（><）のような目など。目の表示だけでとても表情豊かで感情的な振る舞いが伝わってきます。

http://www.disneyresearch.com/project/papillon/

Project Wurm Hole

SIGGRAPH Studioのスキャンコーナーでは、オバマ大統領にもプレゼンした工作少年Joey Hudy君による、Kinectを活用した手作り感あふれる3Dスキャナ「Skanect」の展示と、短時間で人物丸ごと3Dスキャン可能な巨大3Dスキャナを設置した「Project Wurm Hole」の体験コーナーが用意されていました。

Project Wurm Holeは自動回転する12フィート（約3.6メートル）の球形スキャナで、その中に立ち、数分動かずに黙っているとスキャンが完了します。プロジェクトの名前は、奇妙なアート作品を作るErwin Wurm氏に由来しているそうです。

http://skanect.manctl.com/

An Autostereoscopic Projector Array Optimized for 3D Facial Display

この展示は、テキサスインスツルメンツ社製のLED光源の小型プロジェクタ「DLP Picoプロジェクタ」を72台用いた立体視表示装置です。レンチキュラーレンズ加工された30cm四方の大きさに640×480の解像度で立体映像が投影できます。

おもに顔の表示に特化した表示装置で、左右の視野角は広く110度。消費電力が少なくコストメリットが大きいそうです。ほぼ1度ごとに個別のCG映像が作られているので、少し顔の位置を変えただけでは映像がジャンプすることもなく、顔の移動とともに滑らかな表示を見ることができました。Kinectを用いて顔の位置を検出し、2人が同時に見るための映像を生成／表示するのも可能とのこと。

http://gl.ict.usc.edu/Research/PicoArray/

Near-Eye Light Field Displays

3Dグラフィックスチップメーカーの NVIDIAからは、立体視でき、目のピントも合いやすい次世代のヘッドマウントディスプレイの試作機が展示されていました。中身はSony ECX332A OLEDマイクロディスプレイ、1280×720の解像度のものが使われており、これとレンズアレイを組み合わせることで立体視メガネの液晶シャッターのような役割を果たします。

現在は解像度が粗いものしか実現できていませんが、ハードウェアの進歩とともに、まだまだ高解像度になる余地はあるそうです。個別に調整するため、近視や遠視の人も、自分のメガネなしで、ピントの合う立体視映像が楽しめます。

https://research.nvidia.com/publication/near-eye-light-field-displays

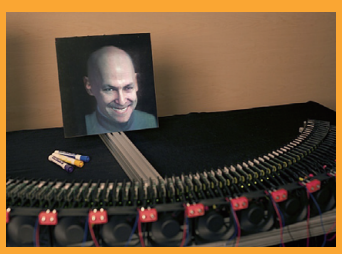
CG業界の求人・求職の場

SIGGRAPHは、CG論文の発表の場、CG関連製品の展示の場であるとともに、求人・求職の場でもあります。企業は優秀な人材を確保するため、アーティストや技術者、研究者は、ポートフォリオ、作品集、履歴書などを手にさまざまな人たちにアプローチしていきます。昨今の特徴としては、大規模な仕事であればあるほど、仕事はとても細分化されていることです。

また、CG/VFX産業も中心となる米国だけでなく、カナダやシンガポール、オーストラリアの各地に広がってきていることです。旧来は大規模なデータや映像をやり取りする必要性から1カ所で作業せざるを得なかった事柄も、ネットやツールの充実で距離を超えた遠隔地間の作業環境も整ってきました。各地間の映像確認作業に使える「cineSync」（http://www.cinesync.

PAPILLON実験用のキャラクタ人形。目に注目

木製の歯車が回って全身の3Dスキャンを実施



小型のプロジェクタを72台並べた立体表示装置

Near-Eye Light Field Displays装置

com/）というツールは、遠隔地間での映像の同期再生や、映像へのコメント記述が可能なツールです。また、大規模で大量のデータ転送には、「Aspera」（http://asperasoft.com/）などを使うことでネットワーク帯域を最大限に活用した協調作業ができるようになったとのこと。

デジタル技術の進歩によって、ある種の職が失われるのかというと、そうでもなく、実写との合成技術や、背景画像を描く画家、CGの世界でも自然な動きを表現するためのカメラマンの技、カラースクリプトと呼ばれる色彩設計などは貴重な職種です。単なるツールであれば、ある程度の時間をかけて覚えれば使えるようになりますが、伝統的な技術や経験は、短期間で身につけられるものでもなく、業界内ではトラディショナルな基礎的能力も重宝される傾向があるようです。

来年のSIGGRAPH 2014（http://s2014.siggraph.org/）は2014年8月10〜14日、カナダのバンクーバーでの開催となります。また、その前の11月19〜22日には、SIGGRAPH ASIA 2013が香港で開催されます。開催前より、SIGGRAPH ASIAで発表される予定の、1枚の写真からとても簡単な操作で3次元形状を抽出するイスラエルの技術「3-Sweep:Extracting Editable Objects from a Single Photo」（http://www.faculty.idc.ac.il/arik/site/3Sweep.asp）という論文が話題になっており、今から期待が膨らみます。SD

各社が求人ブースを設けたJob Fairの様子

gadget 1

Beam telepresence

http://www.suitabletech.com

会場を闊歩していたテレビ電話ロボット

Beamは遠隔操作可能なテレビ電話ロボット。アメリカの各所からはもとより、ヨーロッパ各国、日本、フィリピン、ニュージーランド、インドなどからも遠隔でSIGGRAPHに参加していました。テレビ電話の画面がちょうど人間の顔の高さにあるため、人とのコミュニケーションに適していること、人用の展示物も、同じ目線の高さで見られるのが特徴です。17インチのスクリーン、HDカメラ、6つのマイクアレイで、1秒間に約1.3mのスピードで移動できます。

gadget 2

Drawing Machine

http://roberttwomey.com/drawing-machine/

XYプロットアート

Drawing Machineはワシントン大学のRobert Twomey氏の作品。今年のアートギャラリーのテーマ"XYZN: Scale"にも合致した作風となっています。建築家が緻密な図面や設計図を描いている様子をプロットしていく描画マシンです。2方向から吊るされた描画装置が、高精度でスケッチ描画していく様子、過程そのものが作品となっています。マシンそのものはハイテクなのに、描画に使っているのはその辺に転がっていそうな普通のペンなのが印象的でした。

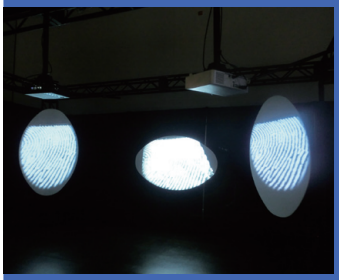
gadget 3

Digiti Sonus

http://www.yoonchunghan.com/project/digiti-sonus-2012/

指紋の模様を使ったアート作品

Digiti Sonusは、カリフォルニア大学Yoon Chung Han氏らによる指紋の模様をテーマにした作品です。自分の指紋をカメラにかざすと、指紋のもようから独特の音パターンを自動生成し、周囲に流します。1人1人が持つ固有の指紋と、1つとして同じものがない音楽との共鳴感を楽しむ作品でした。映像としても、指紋のもようが山脈のように立体的に映し出され、自分の指と映像との連動が感じられました。



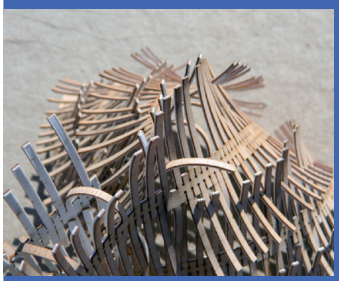
gadget 4

Water Columns

http://maurerweston.com/

水分で変化するオブジェ

Water Columnsは南フロリダ大学のMark Weston氏の作品。複雑にカッティングされた木製のオブジェが水蒸気で湿らされたあと、乾燥した展示会場で、ゆっくりと時間をかけて変形していく様子そのものを表現した作品です。コンピュータでコントロールできるレーザーカッターによってカットされた複雑な加工と、木材の変形を予想した形状と連続性が印象的でした。将来的には建築物の外壁などにも応用していきたいそうです。



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

結城浩の再発見の発想法 6

On Demand

On Demand—オンデマンド

オンデマンドとは

オンデマンド(on demand)とは「要求があるごとにサービスを提供する」という形態のサービス提供方法です。

「要求があるごとにサービスを提供する」とあらためて説明するとかえってわかりにくいですが、オンデマンドなビデオレンタルが一般的ですから、言葉の意味はよくわかるでしょう。

一般のテレビ放送はオンデマンドではありません。番組の内容や放送時間は視聴者の要求に応じて提供しているものではないからです(**図1**)。

それに対して、たとえばHuluのようなビデオサービスはオンデマンドです。視聴者は並んでいる番組の中から好きなものを好きな時間に選んで視聴することができるからです(**図2**)。サービスは、ユーザから「この映画をいま観たい」という要求があるごとに提供されます。

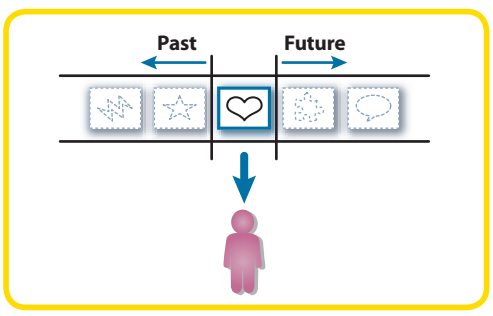
もともと“on demand”の“demand”は「要求」という意味で、“on”は「……に密着して」という意味です。ですから“on demand”は「ユーザからの要求に密着して」行われるという意味になります。

オンデマンドがなぜうれしいか

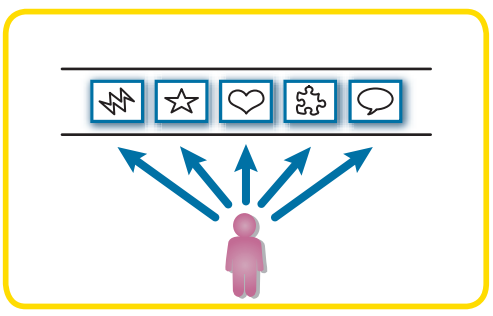
ユーザの「サービスを受けたい」という要求の発生に合わせてサービスが提供されれば、もっとも効果的にユーザの要求を満足させることができます。「見逃し」や「待ち」がない。これがオンデマンドのうれしい理由です。

放送を録画するビデオと、オンデマンドの動画配信サービスを比較するとおもしろいことがわかります。放送されている番組をビデオで録画して観るのは、放送という非オンデマンドなサービスはそのままで、自分の要求を満たそうとする行為です(**図3**)。それに対してオンデマ

▼図1　放送しているものだけ視聴可能



▼図2　オンデマンドで視聴可能



ンドのサービスでは、サービスの側からユーザに歩み寄って要求を満たそうとしているのですね。

オンデマンドに必要なもの

オンデマンドなサービスにはまず、要求を出す**ユーザ**が必要です。ビデオの例でいえば視聴者です。次に、要求を伝える**インフラ**とサービスを提供するインフラが必要です。これはケーブルテレビやインターネットです。そして、ユーザが要求を出すための**品ぞろえ**が必要です。ビデオでいえば視聴できる番組です。

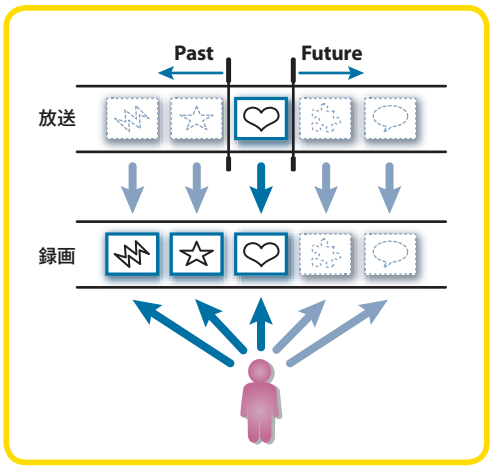
オンデマンドな動画配信サービスで技術はどこに注がれているでしょうか。単純に動画だけに注目すると、サイズを大きく、高画質でという方向に技術を注ぎたくなります。

しかし動画は視聴者がいてこそ価値を産みます。オンデマンドの動画配信サービスは「番組を観る視聴者」のユースケースに注目し、**視聴者の自由度**を高める方向に技術を注ぎ、価値を産んでいることになります。

「観たい動画を、観たい時間に観られる」だけがユーザの要求ではありません。家ではTVで、通勤中は電車の中でスマートフォンで、会社のお昼休みにはPCで観たい。つまり「観たい動画を、観たい時間に、観たいデバイスで」というのがユーザの要求です。そのためサービスは動画と時間とデバイスの自由度を高めようとしています。

ユーザの要求を見極めないと、オンデマンドにしても意味がないのです。

▼図3　録画でオンデマンドを作る



オンデマンド出版

オンデマンドなサービス提供はビデオだけではありません。本の出版でもオンデマンドなサービスが可能です。**オンデマンド出版**です。

オンデマンドな出版ではユーザの要求に合わせて印刷を行いますから、一般的な出版のような在庫を持ちません。これで在庫の費用を抑えることができます。最近話題の電子書籍はある意味で究極のオンデマンド出版になりますね。

日本では利用できませんが、Amazonの Kindle Direct Publishing には CreateSpace というサービスが付随しており、ユーザの要求に応じて「紙の本」が作れます。

オンデマンドな印刷の1つの形態に**バリアブル印刷**というものがあります。これは印刷物の一部分を**可変(バリアブル)**にしたものです。たとえば、ダイレクトメールで顧客の名前を入れ替えて印刷するサービスや、テンプレートに従ってユーザごとに身分証明書を印刷するサービスです。絵本で主人公の名前を子供の名前に入れ替えて印刷するサービスも見かけますね。オンデマンドにすることで、ユーザの自由度を上げ、サービス提供の直前でなければわからない変化に対応していると言えます。

ユーザの要求を見極めること、そして、その要求は何を変化させることを求めるのかを考えるのが有効です。

◆ ◆ ◆

あなたの周りを見回して、オンデマンドになっていないことで不便なことはあるでしょうか。そのときのユーザの要求は何でしょうか。そのユーザの要求に応える技術はあるでしょうか。ぜひ考えてみてください。SD

enchant ～創造力を刺激する魔法～

(株)ユビキタスエンターテインメント　清水 亮　SHIMIZU Ryo
URL http://www.uei.co.jp

第7回 デザインと哲学

ロンドンのデザイナー

その日、僕たちはパリのガレ・デ・ノルド(北駅)を発ち、ロンドンを目指しました。

ドーバー海峡を横断する超特急ユーロスター。

その客室には、個別に使える電源が確保されています。海底トンネルの殺風景な感じを想像していたのですが、不思議とそうした閉塞感はほとんど感じることなく、いつの間にかロンドンに到着していました。

そもそも、僕たちがなぜここにいるのか、話は半月ほどさかのぼります。

立ち消えになったプロジェクト"K"の後継プロジェクトとして、新たな極秘計画が立ち上がりました。その計画に必要な人物として白羽の矢が立ったのが、ロンドン在住のとあるデザイナー、N氏でした。

彼に会うため、僕たちははるばるロンドンまでやってきたというわけです。

ストーン・ヘンジを後にして、そこからロンドン郊外にあるN氏のデザイン事務所へと向かいました。ざっくばらんな作りの建物で、とてもここに世界的に有名なデザイナーが居るとは思えない場所でしたが、中に入るとN氏がにこやかに迎えてくれました。

「あなたに新プロジェクトのデザインをお願いしたい」

僕はたどたどしい英語で、しかし単刀直入に言いました。「そのためにここまで来た」

N氏は言いました。

「デザインとは何か、貴方はどう考えていますか、ミスターシミズ？」

まるで射抜くような瞳でN氏は僕をまっすぐ見つめました。

「デザインとは……」

僕は考えました。デザインとは何か。難しい質問です。とくにその質問が世界的に有名なデザイナーから発せられるとき、それは僕に発注者としての資格があるかどうか問われているということです。

「デザインとは、コミュニケーションです。……違いますか？」

僕は答えました。

N氏はパッとにこやかな表情になりました。

「Exactly, デザインは、コミュニケーションであり、哲学であり、真理の追求です。宗教でさえある。私があなたの仕事を引き受けるということは、単にデザインを提供するということではありません。あなたとコミュニケーションし、あなたとともに新しい哲学を創りだすということです。したがって私は誰とでも仕事をするというわけではありません」

「私はあなたと仕事をする資格がありますか？」

「もちろん。ちなみに先ほどの質問にもしあなたが、"美しさ(beauty)"や"カッコ良さ(Coolness)"

と答えていたら、この仕事はお断りしました。貴方の答えは悪くない。お引き受けしましょう」

そしてN氏との“コミュニケーション”が始まりました。手始めに、N氏に簡単なロゴデザインを依頼することにしました。我々がいまやろうとしていること、その背景となるもの、哲学、考え方、すべてを共有しました。

「フェイス・トゥ・フェイス。コミュニケーションをとるうえで、これはとても大切なことです。コンピュータ企業の経営トップが、これを理解していることは私にとっては嬉しい事実ですね」

N氏は別れ際にそう言いました。

僕たちが日本に戻ると、早速N氏からいくつかのデザインが上がってきました。さすがと唸るものでした。

ところが僕たちの関係が良好だったのはここまででした。プロジェクトKの後継となるプロジェクトのデザインを依頼しようとしたとき、N氏のマネージャーが契約を飲まなかったのです。大企業がスポンサーということもあり、契約金は非現実的な額に釣り上げられ、結局、この仕事をN氏に依頼することはできなくなってしまいました。せっかくのデザインも使えなくなってしまったのです。

特撮監督　樋口真嗣

そのころの僕は、週刊アスキー編集部との共催で、「天下一カウボーイ大会」というイベントを主催していました。

おもにコンピュータ関係の科学者や技術者が集まるイベントだったのですが、第参回となる大会のキーノートスピーカーとして、業界の外の人を呼びたい、という意向を伝えると、週刊アスキーの福岡俊弘総編集長が白羽の矢を立てたのが、『ローレライ』『平成ガメラシリーズ』で知られる、映画監督の樋口真嗣氏でした。

世田谷にあった樋口監督の事務所を訪ねると、監督は気さくに応対してくれました。大会の主旨を一通り説明すると、監督は言いました。

『平成ガメラシリーズ』『のぼうの城』などで知られる樋口監督

「それさ、俺じゃなくて細田守さんに頼んだほうがいいんじゃないんですか？　『サマーウォーズ』を観ました？」

「いいえ、まだ拝見していません」

「絶対観たほうがいいですよ。キーノートはお引き受けしますけど、絶対観たほうがいいです」

樋口監督があまりに薦めるので、その日の夕方には、僕はすっかりサマーウォーズを観る気分になっていました。

サマーウォーズ

果たして、『サマーウォーズ』はあらゆる意味で予想を裏切る映画でした。とくに目を見張ったのは球形で表現されたユニークなインターネット空間「OZ（オズ）」でした。あまりの内容に感動した僕は、それから毎夜のように新宿バルト9に通い詰めました。樋口監督があれほど薦めるのもよく解りました。

とくにUIまわりのデザインが素晴らしく、どこか懐かしい感じもしながら、現在の日本の延長線上にありそうなデザインで、まさにこれぞ未来のコンピュータだ、という気持ちにさせるものでした。

こんなデザインを作る人と一緒に仕事をしたい。何より、こんな素晴らしい映画を作る人と会ってみたい。そんなふうに思っていると、福岡さんから意外な申し出がありました。

「細田守監督に会ってみない？」

僕が毎晩のようにサマーウォーズを見に行っているという噂をどこかで聞きつけたのでしょ

うか。もちろん断る理由もなく、僕は二つ返事で細田監督に会いに行きました。

細田監督は素晴らしい人格者で、クリエイターで、尊敬するべき点の多い方でした。

「細田さん、あのサマーウォーズのUIみたいな、ああいうデザインをお願いしたいとしたら、どうすればいいんでしょうか」

「あれはね、私が描いたというよりも、制作の現場が描いたものですからね。現場と話をするのがいいと思いますよ。私はコンピュータとかには実は疎いんですよ」

僕は驚きました。しかし同時に、それはそうかもしれない、とも思いました。もしコンピュータに詳しかったら、人間が暗算できる程度の暗号なんて前提はちょっと思いつかないでしょうし、そういう知識と物語を感動的に演出する能力に長けているほうが監督としてより相応(ふさわ)しいはずです。

Nendo

細田監督の仰った「現場の人間」は意外なところにいました。

1993年に「Nendo」というグラフィックユニットを立ち上げたグラフィックデザイナー、藤本健太郎氏が、実はサマーウォーズに登場するサイバースペース“OZ”のUIデザインの一部を担当していたのです。僕がOZのデザインを「どこか懐かしい」と思うのも当然で、高校生のころ、Nendoを中心としたアーティストたちが集まって作った伝説的なデザイン雑誌『GAS Book』を愛読していたのです。いわば僕のデザインに対する関心が生まれたルーツが、そもそもこの藤本氏だったわけです。

僕は藤本氏を早速紹介していただきました。まず何かのロゴをお願いしたい、と思ったのですが、さて何がいいだろう、ということになりました。2010年の暮れのことです。

そのとき閃(ひらめ)いたのは、ちょうど僕が構想していた新組織のロゴでした。

その組織は、社内にあって燦然と輝く象徴であり、社員の誰もが憧れ、誇りに思い、社員だけでなく社外のあらゆる人々から好意と敬意を持ってもらえるような、知性の象徴となるような組織でした。

そうした組織には、それに相応しいロゴが不可欠です。人々は、組織の名前でも、組織の内容でもなく、組織を表現する紋章に忠誠を誓い、憧れ、紋章を胸に抱くことを誇りとするのです。

僕は新組織の名前を密かにARC(アーク)と決めていました。これは、数々の偉大な発明が生まれたXEROX PARCのPの字をとったものであり、また、ダグラス・エンゲルバートがビットマップディスプレイやマウス、ハイパーテキストというものを産み出したARC(Augmentation Research Center)の頭文字でもあります。

しかして藤本さんに依頼する最初のロゴデザインは、このARCの紋章に決まりました。

「どんな組織で、どんなデザインの方向性にしたいのですか？」

藤本さんは打ち合わせでそう聞いてきました。僕はN氏とのやりとりを思い出しました――デザインとはコミュニケーションである。ただ格好いいだけや、美しいだけではいけない。哲学を反映させなくては。

そこで僕は、UEI/ARCをどのような組織にしたくて、どのように作ろうとしているのか、それを説明しました。そこは当代一の研究機関を目指すこと。成果を広く世界に使ってもらうこと。ただ研究するだけでなく、人を育てる育成の機能を持っているということ。つまり研究と教育を兼ね備えた機能を持つ、社内にある大学・大学院のようなものであるということ。

そしてここから新しい21世紀の歴史を創りだしたい、という大それた夢まで語りました。

すると藤本さんは、

「だとすると、何か引用できるような事物や言葉があるとデザインがしやすいですね。探してみていただけますか？」

と言いました。

僕は普遍的な知性についていろいろ考えた挙げ句、『ソクラテスの弁明』を思い出しました。

『ソクラテスの弁明』では、デルファイの神殿で「地上で最も賢い人物」という神託を受けたソクラテスが、数々の賢者と呼ばれる人物を歴訪し、なぜ自分が最も賢い人物なのかということについて追求します。その結果、得た結論が「無知の知」であり、「自分が知らないことを知っている」ほうが、「知らないことも知っている」と思う人間よりもわずかに賢いということを見いだします。

ところでこの『ソクラテスの弁明』を著したのは、ソクラテスの弟子の一人であるプラトンでした。ソクラテス自身は文章を書かなかったので、ソクラテスが直接遺した言葉というのはほとんどなく、弟子によって書かれた書物にのみその形をとどめています。

さて、僕たち、そして僕の目指す新組織ARCとはどのようにあるべきでしょうか。

そのとき、ソクラテスのようにあるべきではない、と僕は思いました。組織というのは器であり、ソクラテスとはどれだけ賢かったとしても一人の個人に過ぎません。であれば、むしろソクラテスのようであるべきではなく、ソクラテスのような賢人の言葉を綴り、思想を熟成し、人を育てた弟子プラトンのアカデメイアのようであるべきだろうと僕は考えたのです。

アカデメイアとはギリシャの英雄に因(ちな)んで名付けられたアカデモスの森にあった学校で、プラトンによって設立されました。これが現在の学校を意味する「アカデミー」の語源です。

アカデメイアで教えられていたのは、算術、幾何学、天文学で、それらすべてを修めると、より上位の学問である哲学を学ぶことができたそうです。この哲学優位の考え方は現在の欧米社会にも引き継がれ、多くの場合、大学教授はPh.D.を持っていることが要求されますが、このPh.D.とは、ラテン語のPhilosophiae Doctor、つまり哲学博士を意味しています。

アカデメイアではとくに幾何学が重視され、その入り口には「幾何学を知らぬ者、入るべからず」とただし書きがしてあったそうです。

そこでARCのロゴは、アカデメイアをモチーフとすることにしました。

出来上がったロゴは、ARCという頭字語を取り囲むオリーブの葉、頭字語の下には古代ギリシャ語で「幾何学を知らぬ者、入るべからず(ΑΓΕΩΜΕΤΡΗΤΟΣ ΜΗ ΕΙΣΙΤΩ)」と書かれています。このロゴを僕は大変気に入り、鉄板にレーザー刻印した看板も作りました。

そのARCで、早速作られたのがenchant.jsで、このロゴも藤本さんにお願いしました。今回は時間がなかったので、「ゲーム用のライブラリである」ということだけ伝えると、いまのロゴ案ができてきました。三角形のタイルを組み合わせたこのロゴも非常に気に入り、enchant.jsの紋章として、あちこちで活躍することになります。

こうして2つのロゴが生まれたのです。SD

ARCのロゴが掲げられた代官山蔦屋書店

enchant.jsのロゴ。複数のタイルを組み合わせて「e」の文字になっている

コレクターが独断で選ぶ!

偏愛キーボード図鑑

株式会社 創夢
濱野 聖人 HAMANO Kiyoto
khiker.mail@gmail.com
Twitter：@khiker

第7回

手間暇かけて完成した珠玉の一品

Truly Ergonomic Mechanical Keyboard

写真1　Truly Ergonomicモデル209

はじめに

今回はTruly Ergonomic Mechanical Keyboard(以下Truly Ergonomic、**写真1**)を紹介します。発売が遅れに遅れたことで一部で話題になったキーボードです。筆者は2011年1月末に、支払いを含むプリオーダーを行いましたが、実際に商品が発送されたのは2012年12月中頃でした[注1]。エルゴノミックで、メカニカルで、プログラマブルと謳っていましたが、発売当初、プログラミングツールは配布されていませんでした。しかし、つい先日の8月に配布され、やっと一通りの機能がそろいました。このように聞くと、不安を覚える方もいらっしゃるかもしれませんが、キーボード自体は非常にできが良いです。

Truly Ergonomic Mechanical Keyboard

カナダのTruly Ergonomic社が開発したキーボードです[注2]。

発売当初は104、105、109という3種類のモデルがあり、スイッチはCherryの茶軸、青軸、赤軸とありました。104が英字環境向け、105と109がInternational環境向けです。その後、モデルチェンジがあり、現在では207と209というモデルが販売されています。スイッチはCherryの茶軸ですが、209のみ赤軸も販売されています。207は英字環境向けで、209がInternational環境向けです。

特徴

大きな特徴として、

- エルゴノミクスキーボード
- メカニカルスイッチ
- Fully-Programmable

といった点が挙げられます。筆者は109と209しか所持していませんが、これらであればOSの設定をとくに必要としないという特徴もあります。OSのキーボード設定が英字であろうと日本語であろうと、すべてのキーが入力できます。

多くのエルゴノミクスキーボードと同様に格子状のキー配列を採用しています。キーが縦一列にならんでおり、左右対象です。入力時には、ほとんど手を動かさずに指を動かすだけで大半のキーを入力できます。

大きさもそう大きくなく、横幅は通常のテンキーレスキーボードよりも小さいです。右利きの人がマウスを右側に置いた場合、手からマウスまでの距離が短くなるため、マウスに手が伸ばしやすくなっています。

また、パームレストは取り外しが可能です(**写真2**)。パームレストを

写真2　モデル209(パームレストなし)

注1) プリオーダー自体は2010年6月初頭からやっていました。当初の発送予定は2010年12月でした。
注2) http://www.trulyergonomic.com/

外すと縦幅がHappy Hacking Keyboardよりキー2つ分ほど大きいぐらいと非常に小さくなります。

スイッチにはCherryを採用しています。その中でも茶軸、もしくは赤軸が使われています。Cherryのスイッチのストローク(キーを押し込める深さ)は4mmですが、半分の2mmまで押し込めば押したと認識されます。茶軸はキーを押したと認識する直前に一番荷重が強くなるポイントがあるのが特徴です。赤軸はキーを深く押せば押すほど線形に荷重が強くなります。両軸ともに認識する個所での荷重が45cNでCherryの軸では軽い軸です。

プログラミング用のツールが配布されており、キーボードの配列を自由自在に変更できます。設定はキーボード側に保存されるので、あらゆる場面で設定した配列を使えます。

配列を変更するには、次の手順で自分の好きな配列を作成し、キーボードに適用します。

①Layout DesignerのWebページを開く[注3]
②Layout Designerで自分の好きなキーボード配列を作成し、Firmwareを保存する
③Firmware更新用のツールを使い、Firmwareの更新をする

Layout Designerはブラウザ上で動作するツールで、最新のFirefoxとChromeにのみ対応しています(**図1**)。Firmware更新ツールは現在のところ、Windows用のみが配布されています(**図2**)。もちろんFirmwareを更新したあとはLinuxやMac OS X上でも作成したキーボード配列が使えます。詳細はLayout DesignerのWebページを参照してください。なお、Dvorak配列などのサンプルのFirmwareがサイトで配布されています[注4]。


図1　Layout Designer

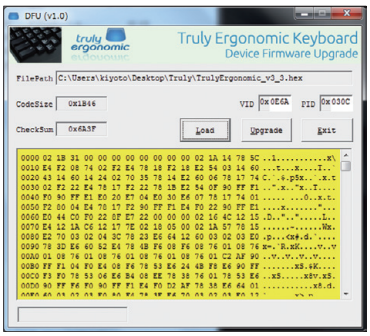


図2　Firmware更新ツール

また、簡易的な設定の切り替えであれば、キーボード裏面にあるDIPスイッチで変更ができます(**写真3**)。たとえば、Mac OS X上で使いやすくするために、左右の Ctrl をCommandキーとし、左右の印字のないキーを Ctrl に変更できます。Firmwareの変更に関するロックもこのDIPスイッチで行えます。

写真3　DIPスイッチ

入手方法

日本ではダイヤテックが国内正規販売代理店となっています。同社のオンラインショップ[注5]より購入ができます。販売しているのはキートップに印字のある209モデルです。赤軸/茶軸両方のモデルを販売しています。値段は2万円ほどです。

207や209でキートップに印字のないモデルは、Truly ErgonomicのWebサイトで販売しています。値段は$248で、Webサイトによると世界中に発送するそうです。

また、104や105、109モデルは、稀にオークションサイトに出ています。興味のある方は、eBayやYahoo!オークションを定期的に探してみてください。

◆ ◆ ◆

発売に時間がかかったり、発売後もプログラミングツールがなかなか配布されなかったりといろいろありましたが、その分良いキーボードに仕上がっています。

エルゴノミクスキーボードに興味はあるが、KINESISのようなあまりに形状が異なるものは手が出しづらいという方に、まずは最初の一歩としてお勧めします。エルゴノミクスキーボードを使い始めても、最初は慣れないのが普通ですので、使ってみる場合は最初は我慢してまずは1週間使ってみてください。SD

注3) http://www.trulyergonomic.com/store/layout-designer--configurator--reprogrammable--truly-ergonomic-mechanical-keyboard/
注4) http://www.trulyergonomic.com/store/alternate-layouts--truly-ergonomic-mechanical-keyboard
注5) http://www.diatec.co.jp/shop/index.php

秋葉原発!

第37回

はんだづけカフェなう

DIPマイコンとmbed

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

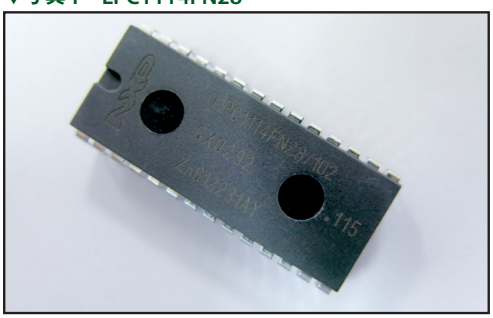
はじめに

ちょうど1年ほど前になりますが、本連載の25、26回で「ARMマイコンをはじめてみよう」ということで、LPC1114FN28（以降LPC1114と略）というDIPのマイコン（**写真1**）での開発を紹介しました。今回、このDIPマイコンにmbed SDKの移植をし、mbed.orgにあるオンラインコンパイラでも利用ができるようになったので紹介します。

mbedとは

mbedが何であるか、知れば簡単なことなのですが、把握するのが難しいもののようです。mbedはCPUの設計図を売っているARM社が提供している開発環境です。ARM社が売っているのは、青い基板の「mbed LPC1768」と黄色い基板の「mbed LPC11U24」の2種類です。この2種類の製品の箱には、この基板に搭載されているマイコンを製造しているNXPのロゴも刷られているためか、mbedはNXPのプロジェクトだと思っていらっしゃる方も多数見受けられますが、それは誤りです。NXPはmbedのリードパートナーという立ち位置であって、あくまでmbedというプロジェクトはARM社が行っているプロジェクト（製品）です。

▼写真1　LPC1114FN28



当初はARM社が販売している、先述の2種類の基板のみがmbed環境（mbed SDK）で開発可能な基板でした。mbed環境を使って開発したソフトウェアは、ほかの基板での使用も許可されています。ですので、正確に言えばこれらの基板に搭載されている2種類のマイコンの開発ができる状況だったのです。

しかし、今年、mbed 2.0となったことで状況が変わってきました。ソースコードが公開されていなかったmbedですが、mbed 2.0となり、mbed SDKがオープンソースになりました。mbed SDKのソースコードはmbed.orgでも公開されていますが、GitHubにもリポジトリ[注1]があります。

ほぼ同時期にフリースケール社から「黒い基板のFRDM-KL25Z」というボードの開発もmbed環境で行えるようになりました。初めてのNXP製品以外のマイコンを搭載したボードかつ、ARM以外のメーカーが販売するmbedのボードの登場です。このFRDM-KL25Z以来、mbed環境で開発が可能なボードやマイコンの種類は徐々に増えてきています。

移植のきっかけ

今年の6月、mbedの生みの親であるARM社のChris Styles氏が来日しました。そのとき、筆者が秋葉原を案内したのですが、そこで売っているARMコアのマイコンとしてLPC1114

注1）　https://github.com/mbedmicro/mbed

を紹介しました(**写真2**)。このときまでLPC1114FN28を目にしたことがなかったようで、DIP[注2]パッケージで手軽にブレッドボードで試せるこのマイコンにとても興味を持ったようです。そこでコミュニティの力でこのマイコンの開発をmbed SDKを使って行えるように移植できないだろうかという話になったのです。

筆者のように、mbedを使ったり、NXPのサンプルコードをもとに開発をしている段階ではスタックやメモリといった深い部分を考える必要はありません。このこともあって、筆者はmbedの移植をするほどのCortex-Mに関する知識を持っていませんでした。そこで、Cortex-Mやコンパイラに関する深い知識を持った仲間が必要だということでToyomasa Watarai氏(@toyowata)に相談をし、強力に開発を進めていただきました。

LPC1114のうれしさ

ここまで紹介してきたmbedですが、なぜLPC1114だとうれしいのでしょう。たとえば青い基板のmbed LPC1768と黄色い基板のmbed LPC11U24は基板ですが、DIPのように足が生えていてブレッドボードを使っていろいろな試作ができるようになっています。しかし、これらの基板はともに4～5千円台で売られており、ちょっと気軽にmbedを試してみるには思いきりのいる価格でした。

一方で、LPC1114は、なぜか日本では安価で販売されており、数百円も出せば購入できます。後述しますが、LPC1114にプログラムを書き込むにはUARTというシリアル通信のインターフェースを用意しなければなりません。しかし、こういったインターフェースも青や黄色のmbedからすると、さほど高い品ではありません。Arduinoなどを使っていれば持っているであろう部品に、数百円のCPUを1個買ってきて足すだけでmbedを始められるのが1つのメリットです。

移植の流れ

ターゲットとなるLPC1114に近いマイコンのコードをもとに移植作業を進めよう、ということでLPC11U24という黄色いmbedのコードをベースに移植を行いました。LPC1114とLPC11U24の間でGPIOの操作方法が当初の予想より大きく異なっていたのが想定外でしたが、GitHubのリポジトリをforkして開発を進めました。

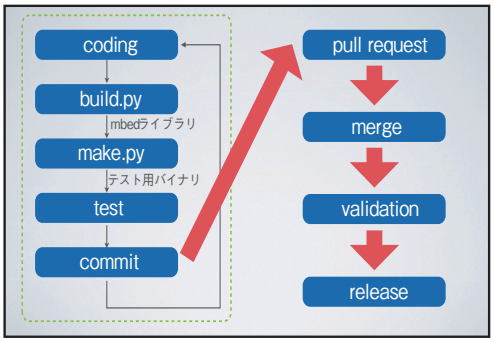
開発中、コードは手元でコンパイルしなければならないのですが、これらのコードはPythonで書かれたビルドシステムを利用することが可能になっています。詳しい説明はmbed.orgにあるHandbook[注3]などに記されていますが、private_settings.pyを書き変えて、MDK-ARM

注2) Dual In-line Package。ICのピンの形で、表面実装ではなくブレッドボードに刺さるような足が出たもの。

注3) http://mbed.org/handbook/mbed-SDK-porting

▼写真2　秋葉原でmbedを見つけて喜ぶChris氏

▼図1　おおまかな開発の流れ



という開発環境を用いて開発を行いました。MDK-ARMは有償のソフトウェアですが、32kBまでの開発が行えるMDK-Liteというエディションが配布されています。LPC1114FN28はFlashのサイズが32kBですので、Lite版でも十分に開発を行うことができます。

図1に大まかな開発の流れを記しましたが、基本的には手元の開発環境でコーディングとビルドを行い、実機(といってもブレッドボードの上で組んだ回路)でテストを行う工程を繰り返して機能を一通り実装していく作業を行います。リリースできるだろうという段階に達すれば、pull requestを出し、本家のリポジトリにmergeしてもらいましょう。mbedチームによる評価が行われ、多くの人に有用だと判断されればオンラインコンパイラで使えるようにリリースされるかもしれません。

LEDを点滅させてみよう

LPC1114は取り扱いやすいDIPですので、ブレッドボードで回路を組むこともできます。簡易的にブレッドボードで組む場合、図2のように接続します。

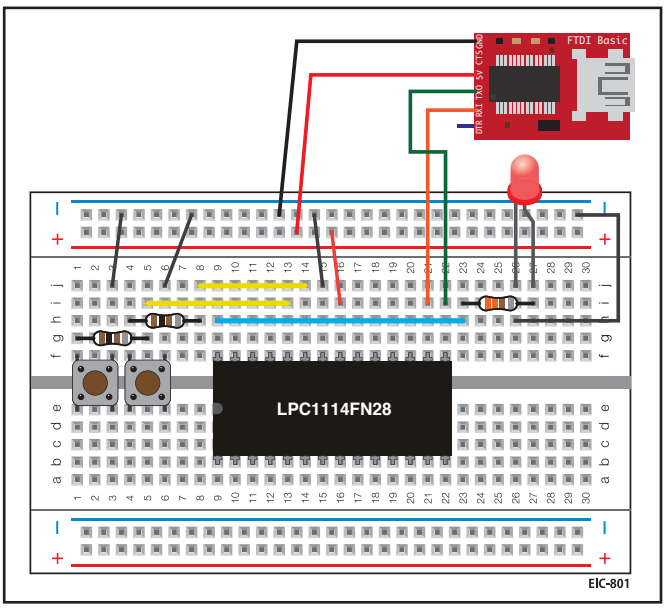
LPC1114、100Ωの抵抗2本、タクトスイッチ2個、330Ωの抵抗1本、LED 1個、UART変換基板で組んでみました。本来はコンデンサなどを追加したほうが良いのですが省略してあります。UART変換基板は、3.3Vのものが必要です。最近、筆者は5Vと3.3Vを切り替えできる、スイッチサイエンスの「FTDI USBシリアル変換アダプタ(5V/3.3V切り替え機能付き)」[注4]を愛用しています。秋月電子通商の「FT232RL USBシリアル変換モジュール」[注5]でも同様のことができるはずです。

mbed LPC1114の開発

mbedを使ったことがない方は、まずmbed.orgのアカウントを作成してください。するとオンラインコンパイラと呼ばれる、開発環を使えるようになります。残念ながら日本語でコメントを書くことはできませんが、なかなかよくできたエディタをブラウザ越しで使うことができます。ここで、コンパイラの画面の右上にターゲット(開発対象)とするCPUを選ぶことができるボタンがありますので、これをクリックします。ここで、「Add a device」というボタンをクリックし、「LPC1114FN28」を選択してください。すると、LPC1114のページに飛びますので、ここで「Add to mbed Compiler」をクリックします。

こうすることで、コンパイラで使うことができるターゲットの一覧に「LPC1114FN28」が追加されるので選択し、ダイアログの右上にある「Select Platform」をクリックすることで、ターゲットをLPC1114にすることができます(図3)。

▼図2 ブレッドボード上のmbed-LPC1114 FN28とその他部品の配線図



注4) http://www.switch-science.com/catalog/1032/
注5) http://akizukidenshi.com/catalog/g/gK-01977/

次にmain.cppにコードを書いてみましょう。今回は、LPC1114の28番ピンに接続したLEDを点滅させてみるので、**リスト1**のようなコードを書いてみました。コード内の「dp28」という値で、28番ピンを使うようにしています。

書き終えたら「Compile」というボタンをクリックしてみます。問題がなければ無事にコンパイルを終え、バイナリファイルのダウンロードが始まります。

プログラムの転送

書き込みを行うにはいくつかの方法があるのですが、今回はFlashMagicを使ってみます。FlashMagicには、Windows版とMac OS X版があり、Embedded Systems Academy社が配布していますので、同社のWebサイト[注6]からダウンロードし、インストールしてください。

また、FTDI USBシリアル変換アダプタを初めて使用する際には、ドライバのインストールも必要です。FTDI社のWebサイト[注7]からダウンロードし、インストールしておきましょう。

次にコンパイラから出力されたバイナリをHEXファイル（テキスト形式の16進ダンプ）に変換します。FlashMagicはHEXファイルでなければ扱えません。バイナリをHEXファイルに変換するには、BIN2HEXといったアプリケーションを使います。

最後にこのバイナリをLPC1114に転送しましょう。まず3.3VのUSBシリアル変換アダプタとパソコン、それからマイコンをすべて接続します。次にマイコンボードのISPスイッチ（**図2**では左側）を押したままRESETスイッチ（**図2**では右側）を押して離します。RESETスイッチから指が離れたらISPスイッチから指を離してください。つまりISPスイッチが押されている状態でリセットを行います。

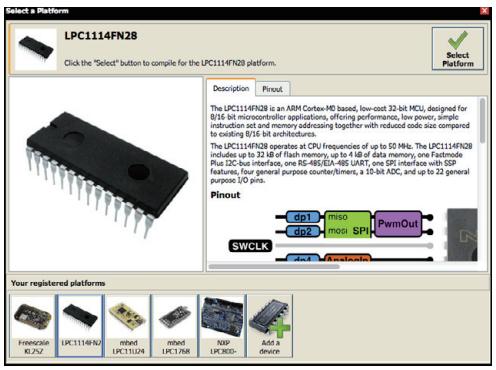
次にFlashMagicを起動し、「Connection」タブでUSBシリアル変換アダプタのポートを選択し、「Identify Device」をクリックして、「Device」が「LPC1114/102」となれば正常に接続ができ、マイコンがISPモードになっています。「Firmware」タブで先ほどのHEXファイルを選択し、「Start」ボタンをクリックするとマイコンへの転送が開始されます。「Completed」と表示されて転送が終了したところで再度ボード上のRESETスイッチを押してリセットを行うとスケッチの実行が開始され、LEDが点滅を繰り返します。

まとめ

駆け足で説明してきましたが、このような手順でmbed環境によるLPC1114の開発ができます。誌面の都合で省略してしまった点などについては、筆者のmbed.orgにあるNotebook[注8]に記してありますので参照してください。SD

注6） http://www.flashmagictool.com/
注7） http://www.ftdichip.com/Drivers/VCP.htm

▼図3　ターゲットを選択するダイアログ



注8） http://mbed.org/users/ytsuboi/notebook/mbed-lpc1114-blinky-ja/

▼リスト1　LEDを点滅させるコード

```
#include "mbed.h"

DigitalOut myled(dp28);

int main() {
    while(1) {
        myled = 1;
        wait(1);
        myled = 0;
        wait(1);
    }
}
```

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 2013 年 11 月 17 日です。プレゼントの発送まで日数がかかる場合がありますが、ご了承ください。

ご記入いただいた個人情報は、プレゼントの抽選および発送以外の目的で使用することはありません。アンケートの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 ミニステーション データ引越ソフト添付 USB3.0 用ポータブル HDD

1 名

PC のデータの引越しを簡単、安心、便利に行えるソフトウェア「ファイナルパソコンデータ引越し 9 プラス」を添付したポータブルハードディスク。高速な USB3.0 接続のため大量のデータもスムーズに転送できます。USB ケーブルをつなぐだけで、別途電源ケーブルが不要なバスパワー駆動に対応。容量は 500GB です。

提供元 バッファロー **URL** http://buffalo.jp

02 ESET FAMILY SECURITY

5 名

Windows、Mac、Android の端末を合計 5 台まで 1 年間利用可能なマルチパッケージの総合セキュリティソフト。「EVALUATION」（評価版）とのラベル付きですが、製品版とバージョンや機能に違いはありません。1 年間使用後は製品版同様に契約更新も可能です。

提供元 キヤノン IT ソリューションズ **URL** http://www.canon-its.co.jp

03 ツボラボ 足マッサージャー

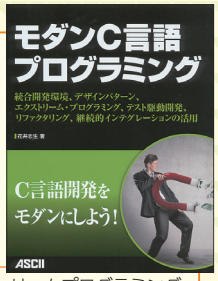
2 名

足裏を快適に刺激する足つぼ刺激マッサージャーです。外回りで足が棒になったとき、デスクワークで疲れたときなどに、椅子に座った状態のまま足裏でコロコロ踏んでマッサージすれば、心身ともにリフレッシュできます。

提供元 リーブコーポレーション **URL** http://lieb.co.jp

04 モダン C 言語 プログラミング

2 名



花井 志生 著／
B5 変形判、280 ページ／
ISBN = 978-4-04-891309-6

統合開発環境、デザインパターン、エクストリームプログラミング、テスト駆動開発、リファクタリング、継続的インテグレーションなどのモダンな開発スタイルを組み込み開発に適用する方法を解説。

提供元 アスキー・メディアワークス **URL** http://asciimw.jp

05 進化する魚型ロボットが僕らに教えてくれること

3 名

ジョン・H・ロング 著、松浦 俊輔 訳／
四六判、336 ページ／
ISBN = 978-4-7917-6718-2

絶滅した初期の魚に似たロボットを作り、そのロボット同士に配偶者や資源をめぐって争わせて、遺伝子を変異させる。そんな異例の手法で進化研究に挑んだバイオロボット工学者の挑戦の記録。

提供元 青土社 **URL** http://www.seidosha.co.jp

06 いちばんやさしい アルゴリズムの本

2 名

みわよしこ 著／
A5 判、224 ページ／
ISBN = 978-4-7741-6013-9

探索、ソート、選択、マージを小難しいと感じる人向けにとことんやさしく解説します。小学校の算数を基礎知識として、これから本格的にアルゴリズムを学ぶための基盤を作ることをめざします。

提供元 技術評論社 **URL** http://gihyo.jp

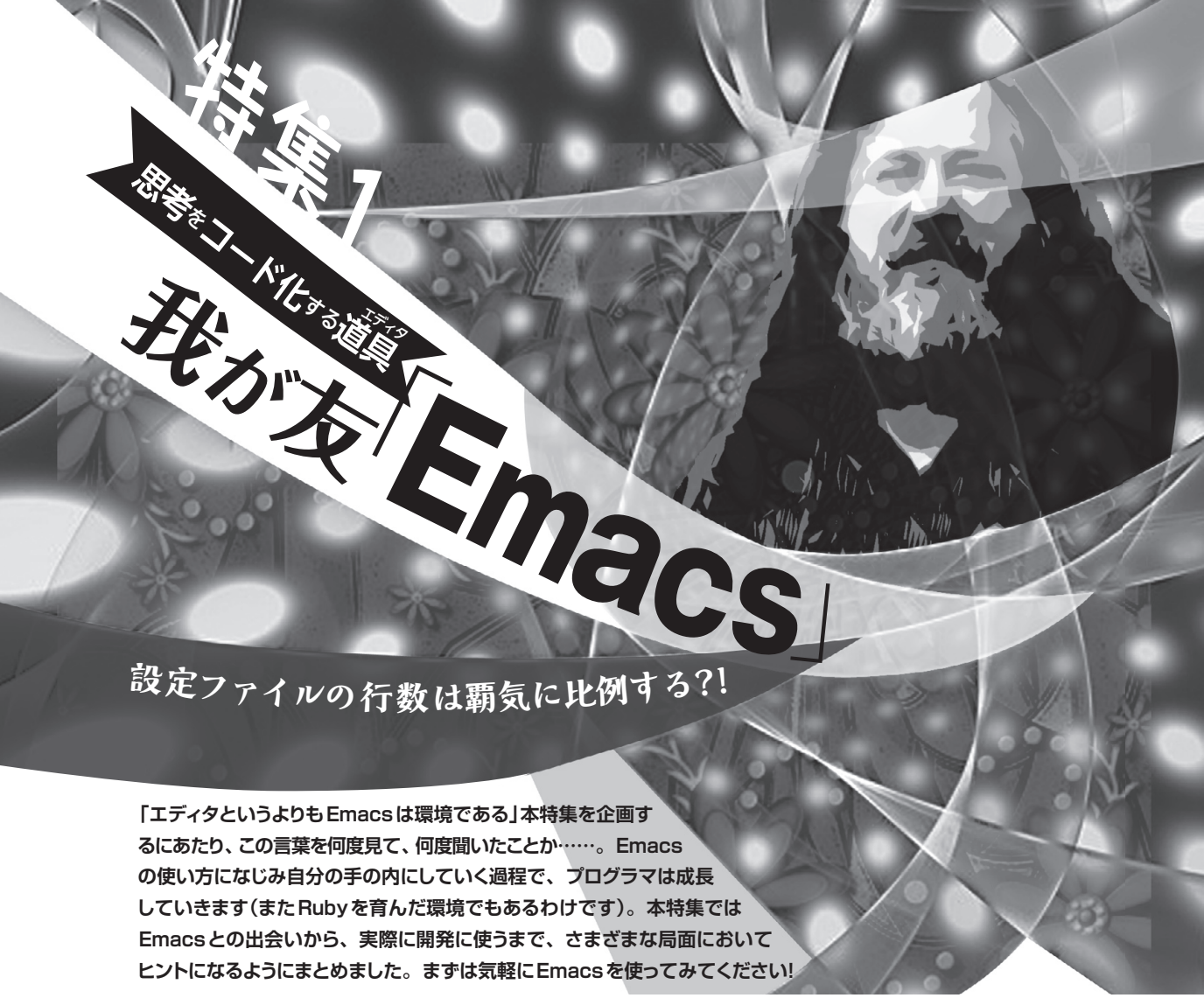
07 おいしい Clojure 入門

2 名

ニコラ・モドリック、安部 重成 著／
A5 判、336 ページ／
ISBN = 978-4-7741-5991-1

環境構築、Ruby との連携、NoSQL で Cassandra、遺伝的アルゴリズム、JBoss も MQ も、Heroku も、さらには Arduino で組み込みまで、Clojure プログラミング技術のフルコースを紹介します。

提供元 技術評論社 **URL** http://gihyo.jp



特集1 思考をコード化する道具（エディタ）我が友「Emacs」

設定ファイルの行数は覇気に比例する?!

「エディタというよりもEmacsは環境である」本特集を企画するにあたり、この言葉を何度見て、何度聞いたことか……。Emacsの使い方になじみ自分の手の内にしていく過程で、プログラマは成長していきます（またRubyを育んだ環境でもあるわけです）。本特集ではEmacsとの出会いから、実際に開発に使うまで、さまざまな局面においてヒントになるようにまとめました。まずは気軽にEmacsを使ってみてください!

本特集でEmacs特有のキー操作を表記する場合は次のように記述します。

- `C-n`
- `C-x C-c`
- `M-x コマンド名`

`C-n`は、Ctrlキーを押しながらnキーを押すということです。同様に2つめの操作は、Ctrlを押しながらxを押したあと、続けてCtrlを押しながらcを押します。

Ctrlのほかに Alt（Macの場合はoption）もよく使います（EmacsではMetaキーと呼ばれます）。この場合の表記は`C-`の代わりに`M-`となります。

このほか複雑な表記がありますが、`-`でつながっている場合は「押しながら」、空白で区切られていたら「一度手を離す」と考えれば（おおむね）よいでしょう。

Chapter 1

Emacsとの「出会い」はある日突然に

～yak shavingと寄り道について～

Emacsもしくはある程度互換性があるエディタを使い始めてから、はや10年以上が経ちます。これから紹介するように、凝ったカスタマイズはまったくしていません。それでもまさに「手に馴染む」エディタとして日々Emacsを使ってコードを書いています。タイピング時のホームポジションが崩れない、プログラミング用のエディタ独特のキーバインドに慣れてしまうととても快適になってしまうのです。今回は僕の個人的なストーリーとして、Emacsとの出会いからさかのぼりつつ、現在の利用法について触れます。

Writer 和田 裕介（わだ ゆうすけ）

●Emacsの設定ファイル行数：150

●㈱ワディット取締役（Twitter@yusukebe）1981年生まれ。Webアプリケーションエンジニア。未踏ユース準スーパークリエータ。㈱オモロキではCTOとして開発を担当。代表作「君のラジオ」。人気お笑いサイト「ボケて」など。

イラスト：高野涼香

はじめに「ご注意」

本章ではEmacsの高度な使い方は紹介していません。僕がわりと素の状態から使っているからです。便利とされているorg-modeやflymake、補完機能などはいっさい利用していません。ちなみに僕の「~/.emacs」ファイルは現在で150行ほどでその多くは「mode」の設定になっています。

その代わり、プログラミングエディタの代表格の1つであるEmacsをなぜ使うことになったのか？――を紹介することにより、未だ触ったことがないという方への使うキッカケになれば幸いです。ではいってみましょう。

情報処理の授業でいきなりEmacs

僕が大学生生活を過ごした慶應義塾大学の湘南藤沢キャンパス、通称「SFC」では1年生の最初から「情報処理」という授業があります。当時その授業の実習が、今思えばかなりスパルタでかつ硬派なものでした。というのも、実習のために特別教室に入るとコンピュータが並べられているわけですが、すべてのマシンがSolaris OSのSun端末なのです。DOS/VとWindowsしか触ったことがない僕にとっては衝撃的でした。UNIXと言えば映画ジュラシックパークのラストシーンで登場人物の女の子が「わたし、UNIXなら使えるわ」という台詞を話していたことくらいしか印象がありません。

秀丸エディタじゃだめなの？

授業内容はUNIXのコマンドを覚えることから始まり、期末にはHTMLを書いてホームページを作って、それが評価されるというこれもまた硬派な感じ。当然ながら、HTMLを書くには何かしらのエディタが必要になるのですが、そこで出会ったのがEmacsでした。

「え、Windowsのメモ帳とか秀丸エディタじゃだめなの？」

内心そう思いながら、Emacsに触れることになります。

Emacsで一番最初に困ったのが「終了のしかた」です。ターミナル上のEmacsを終了させて違う作業をしたいのにコマンドがわからず抜けられない状態になるのです。いまだと手癖で「C-x C-c」と打てますが、その当時は教えられても頭に入らず、すぐに忘れてしまいます。そこで、毎回そのたびに授業を手伝ってくれているティーチングアシスタント、略して「TA」の方を呼んで、教えてもらいました。そういえば、当時、TAの人にほかの困ったことも頼むと、CUIのコマンドを打ちながら問題を解決してくれて、すっごいカッコいいな！って見ていましたが、今の自分ならば自力で解決できるかもしれないと思うとなんだかうれしいような変な気分ですね。

こうして言うならば「Emacs強制ギプス」な出会いでしたが、やはり印象は「なぜこんな難しいことをテキスト書くのに使わなくてはいけないんだ」という感じで、授業が終わってからは一度Emacsをエディタとして使うことから離れて行くのです。

Emacsっぽいエディタ達

それから1年弱が経ちました。SFCでは何年生からでも研究室に所属できたので、僕はとある研究室に大学2年生から正式にジョインし、そこまで本格的じゃないにしろ、FlashのActionScriptやPHPなどのプログラミングを始めました。

xyzzyでEmacsに目覚める?

当時はWindowsのノートパソコンをおもに使っていて、たしか、秀丸エディタをメインのエディタとして使っていました。ただ、そのときすごく気になっていることがあって、とある先輩方のPCでのコーディングの様子を見るとエディタに「亀」のマークが。

「なんだこのエディタ……」

プログラミング初心者にとってコードを書ける人が使っているエディタには興味が湧くものです。より深く覗き込むと亀のアイコンの横には「xyzzy」と記されているではありませんか。さっそく検索してソフトウェアをダウンロードしてインストール[注1]。使ってみるとメニューが充実しているために秀丸のようなエディタと同じような感覚で使えます。が、いろいろといじっていると以前スパルタ的に覚えたEmacsのキー

注1）xyzzy公式サイト：http://xyzzy-022.github.io/

バインドが使えるじゃないか！　そこで、わからないコマンドはメニュー上からマウスで操作し、最低限わかる範囲はマウスを使わずにキーバインドやコマンドだけでなんとかxyzzyを使ってみたのです。すると、

・昔習得したキーバインドはそのまま使う
・わからない操作はマウスを使いメニューから選択
・時折マウスを使うのがめんどくさくなる
・目的とする使い方を検索する。基本xyzzyはEmacsっぽいキーバインド
・たとえばC-kで行単位でカット＝kill-lineできることがわかる
・これ便利だわ～

という流れができたのです。こうしてxyzzyを使うことで、徐々にEmacs特有の操作を覚えていくことになりました。ひとまず、**C-p**が上、**C-n**が下、**C-f**が右、**C-b**が左へカーソル移動するのがいちいちキーボードの右下の端にある矢印キーを使わなくても可能になるのがとても便利な印象でした。とくにプログラムをタイピングしているときに指や手を大きく動かすことはストレスになりますからね。そんなこんなで、ある程度マスターすると、当時プロジェクトを一緒にやっていた後輩にもxyzzyを勧めていました。当初は困惑していた彼も僕に質問したりしつつ、最後にはxyzzyを使いこなしていました。

また、よりEmacsっぽいWindowsのエディタ「Meadow」も触ったりしました。Emacsには「mode」という概念があり、たとえばPerlを書く場合は「cperl-mode」を使い、するとコードに色づけをしてくれるシンタックスハイライトや、インデントの設定ができたりします。こうしたmodeがMeadowだとよりEmacsと互換性が高いのです。

OS XでのEmacs環境

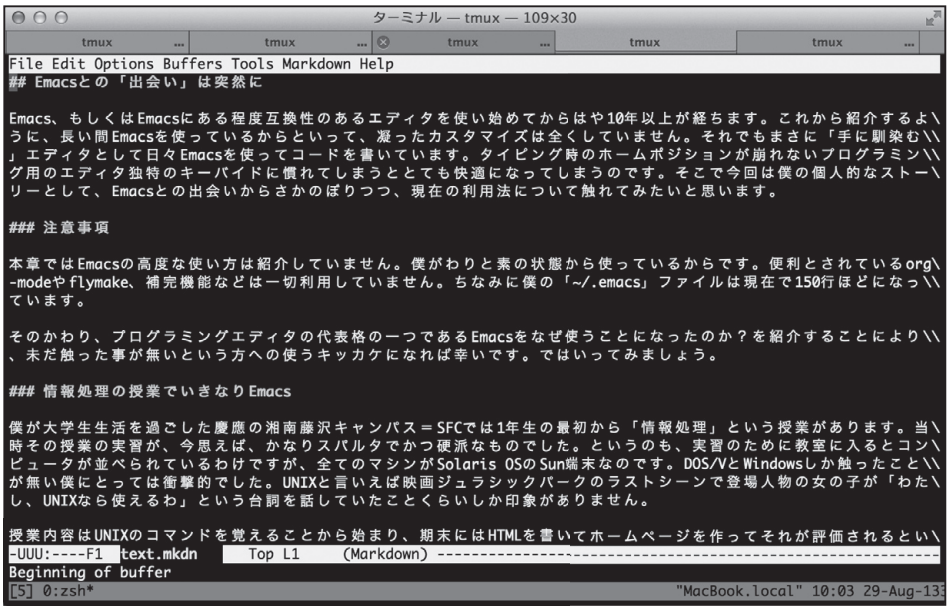
このようにEmacsやそれに近いエディタを使い始め、今ではコードを書く際には手放せないエディタはEmacsです。とはいえ、Emacsに求めていることはキーバインドとmodeとその設定くらいなので、Vimでもいいかもしれないですし、かなり簡易的な使い方をしています。

現在はメインのマシンをMac OS Xにしているので公式なEmacsを使っています。今手元で確認したところ、Homebrewで入れた「GNU Emacs 24.3.1」がそれになります。Mac OS XでEmacsを使っている他のエンジニアの様子を見るとGUIの「Cocoa Emacs」を常用するケースが多いようです。それに対して僕はちょいと特殊な使い方をしていて、Terminal上のシェルから直接CUIのEmacsを「emacs hoge.pl」として立ち上げています。また、Emacs上から直接シェルの操作ができますが、僕の場合は、screenやtmuxなどを使ってEmacsからもう1つのWindow内のシェルに切り替えてプログラムを実行するなどしています（**図1**）。

ちなみに、Emacsの設定ファイルである「**~/.emacs**」では次のようなことだけを記述しています。

・日本語関係の処理
・色づけを有効にして設定する
・タブを4スペースに

▼図1　シェルからEmacsを起動



・C-hをバックスペース、C-\をアンドゥに
・各種modeの読み込みと設定
・Perlコードを整形するPerltidyを有効に

僕は「dotfiles」と呼ばれるような~/.emacsを含めたユーザごとの各種設定ファイルをDropboxに入れておいて、新しいマシンをセットアップするときにはそのまま流用するようにしています。Dropboxではなくともなどをリモートリポジトリとして Gitで管理しておくのもよいかと思います。

日々の業務とEmacs

僕は普段の仕事は、Webアプリケーションの開発がメインで、サブとして今回の雑誌への寄稿のような執筆作業をたまにやっています。WebアプリケーションのほうではプログラミングのコードやHTML、CSSなどを編集する際に当然ながらEmacsを使います。1つの環境上で複数のターミナルを立ち上げることができるtmuxを利用し、Emacsやシェルなどを Webブラウザのタブ機能のように切り替えています。すると次のような操作をすばやく行ったり来たりすることになります。

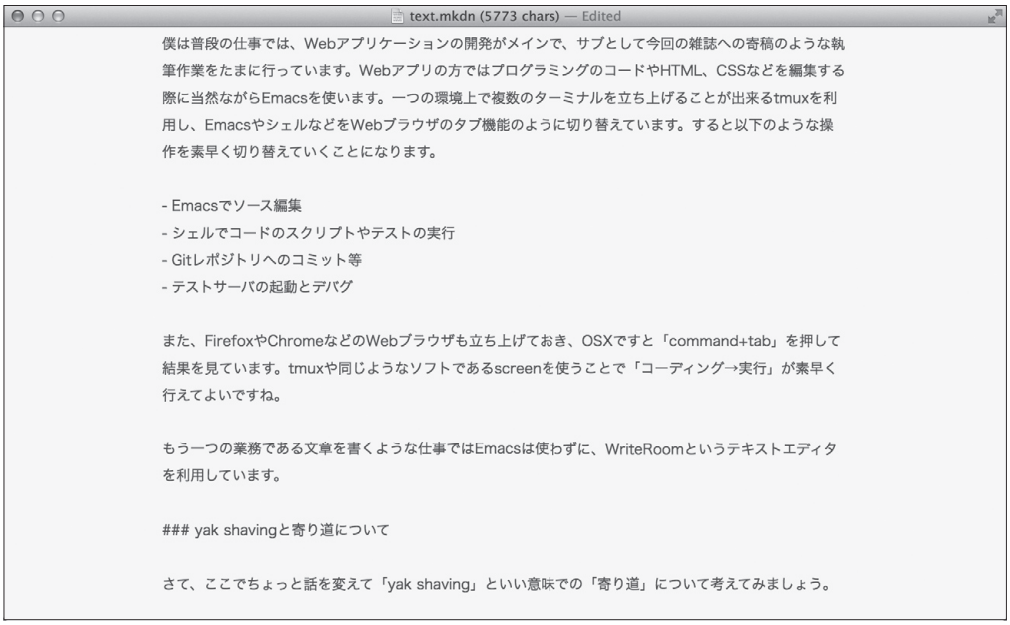
・Emacsでソース編集
・シェルでコードのスクリプトやテストの実行
・Gitリポジトリへのコミットなど
・テストサーバの起動とデバッグ

また、FirefoxやChromeなどのWebブラウザも立ち上げておき、OS Xですと⌘+Tabを押してウィンドウを切り替え、HTMLが描画された結果を見ています。tmuxや同じようなソフトウェアであるscreenを使うことで「コーディングしてから実行する」行為がすばやく行えて良いですね。

もう1つの業務である文章を書くような仕事ではEmacsは使わずに、最近ではWriteRoomというテキストエディタを利用しています(**図2**)。

OS XのCocoaベースで作られたアプリケーションでは「Emacsもどき」のキーバイドが使えるので、Emacsに限らないで、より文章の執筆に特化したソフトウェアでいいじゃないか！

▼図2 WriteRoom（http://www.hogbaysoftware.com/products/writeroom）



——と思い使い始めたのですが、これがなかなか文章を書くのに集中できてお勧めです。余談ですが、ケースによって、DropboxやGitリポジトリなどを利用すると編集者にすぐ原稿を見せることができて便利ですね。

yak shavingと寄り道について

さて、ここでちょっと話を変えて「yak shaving」という意味での「寄り道」について考えてみましょう。

yak shavingという言葉を初めて聞く方も多いかもしれませんが、この言葉自体は昔から存在しているようです。それを最初にプログラミングという文脈で使い、プログラマ界隈で話題になるキッカケになったのが、おそらく「高林哲さん」のブログ記事「yak shavingで人生の問題の80%が説明できる問題[注2]」です。また、最近になって当の高林さんが宮川達彦さんのPodcast[注3]に出演した際に話題として取り上げられ、ふたたび注目されている概念です。

高林さんの記事から引用させていただくと、yak shavingとは

> 一見無関係に見えるけど、真の問題を解くのに必要な問題を解くのに必要な（これが何段階も続く）問題を解くのに必要な活動

注2） http://0xcc.net/blog/archives/000196.html
注3） http://rebuild.fm/

となります。このyak shavingにおける「問題を解くのに必要な活動」の1つが我々にとっての「エディタの使いこなし」に当たるかと考えています。プログラムを書くにはまずそれを書くための道具が必要になりますからね。さて、エディタを使おう！——という初期の段階ではこんなエピソードに遭遇することがあります。

何かの問題を解決するためにプログラムを書きたい！　ただそのためには、書くためにはエディタが必要になる。世の中にはVimとかEmacsとか最近では「Sublime Text」なんて一のもあるけどどれがいいのかな？　調べてみよう。へ？　Sublime Textはすぐ使いこなせそうだなぁ。でも世の中のハッカー達はVimかEmacsを使っているよね。どっちにしようかな？　VimとEmacsで宗教戦争があるってどういうこと……？

こうした活動がyak shavingの定義における「問題を解くのに**本当に**必要なもの」に当たるかは、議論があるので、ここでは「寄り道」という言葉を使いますが、エディタを含む環境構築には寄り道を含め、時間を費やしてしまいがちです。

そこでいかに寄り道にかける時間を少なくし、目的を達成するかが理想のように思えます。事実、僕は目的達成を目指すためのプログラムを書くことを最優先するため、エディタにおいては最小限の設定にしているふしがあります。よくViやVimをメインエディタにしている人は「どこにでもデフォルトで入っているから」という理由で利用している方も多く、そうした方々は「寄り道率」が低い感じの印象です。

ただ、寄り道が「悪」であるわけではまったくなく、むしろ最適な寄り道をすることで本来の目的達成をより効率的に行ったり、寄り道から得られる知識やスキルなどもあるでしょう。また、それ自体が楽しみといったこともあり得ますね。すると「寄り道」という言葉がふさわしくないほど有益になります。

yak shavingにおける活動や寄り道という行為はバランスを保ちながら行うことが重要ですし、人それぞれどこに労力を費やしたいかという意図が個性にもつながるところだと思っています。だからこそ、エディタはEmacsじゃなくてもよいでしょうし、Emacsのカスタマイズ具合もさまざまでよいかと思います。yak shavingはプログラムを書くのに必要な行為であり、それがプログラマたるもの必須な通り道。寄り道も加えて適度に行うとよいでしょう。

まとめ

僕の個人的なEmacsとの出会いからyak shavingを取り上げたエディタに対する考え方を紹介してきました。プログラミング言語の取捨および習得と同じで「誰かが使っているから！」「かっこいいから！」という単純な理由で始めてもよいですし、本章の次から紹介されるカスタマイズ性の高さなどEmacs本来の魅力に惹かれて使うのもアリでしょう。どのように使うのであれ、Emacsは手に馴染むエディタであることはたしかです。Emacsを使って楽しくコードを書きましょう！SD

Chapter 2

Emacsユーザの生産効率をアップしてきたカスタマイズの第一歩

～定番になるにはワケがある!～

Emacsとカスタマイズとは切っても切れない関係です。そこで、本章ではEmacsのカスタマイズの魅力や定番の設定について紹介します。

Writer 高石 諒（たかいし りょう）

●Emacsの設定ファイル行数：1,900行
●アリエル・ネットワーク㈱のエンジニア。学生時代に使いやすいエディタを探していて、Emacsに辿りつきました。最近はVimも使えるように特訓中です。好きなEmacs拡張はorg-mode。Twitter@r_takaishi
Mail：takaishi_r@ariel-network.com

Emacsカスタマイズの魅力

皆さんはEmacsの魅力といえば何を思い浮べるでしょうか？　筆者は、やはり好きなようにカスタマイズができる、その自由度ではないかと思います。Emacsに限った話ではありませんが、カスタマイズしようと思えばどこまでも凝ったことが可能です。これはメリットでもありデメリットでもあります。

カスタマイズしすぎることによるデメリットとして、他の環境（他人のマシンやサーバなど）では不便になるということが挙げられます。カスタマイズした環境が使えないので、普段より不便な手段で作業することになり、効率が悪くなるのです。

このデメリットは見方を少し変えると必ずしもデメリットではありません。他の環境で使えるエディタの操作方法も覚えておけば、とくに不便になることはありません。手元の環境では自分の手になじむように設定したものを使えばよいのです。筆者はGUI上のEmacsとコンソール上でのVimを並行して使っており、他の環境などでの作業はVimを使ったりしています（場合によってはEmacsのTRAMP[注1]を使って編集することもあります）。

また、Emacsという限られた環境でいろいろできるようにカスタマイズする、というのは楽しいものです。必ずしも実用的ではないかもしれませんが、盆栽のようにお気に入りの設定を育てていく楽しみがあります。ただ、くれぐれも「手段が目的と化してしまう」ことにならないようお気をつけください。どこまでもカスタマイズできるため、「仕事の息抜きに設定していたら夕方になっていた」などという事故が起こることがあります。いわば、勉強の休憩につい部屋の掃除をしてしまったり、引越しの準備中に出てきた本をうっかり最後まで読んでしまうような中毒性があるのです。

設定方法について

本節ではEmacsの設定を変更する方法について説明します。なお、GNU Emacs 24.3.50.1 for Mac OS Xをもとに執筆しています。

Emacsの設定には、GUIで行う方法と自分で設定ファイルを書く方法の2種類があります。GUIの場合、設定できる項目をカテゴリ別に見たり、検索したりすることが可能です。「こういう設定ができるかどうか」を調べる場合、GUI上でカテゴリから絞り込んでいくのも1つの手です。また、設定済みの項目を一覧で見ることも可能です。一方、自分で書く場合は、標準では満足できないときに自分で動きを変えるような、

注1）Chapter 5を参照。

かなり凝ったことができます。

普通に使う分にはGUIでの設定で十分ですが、GUIでの設定に対応していないパッケージを使う場合には自分で設定を書くしかありません。また、自分で設定を書く場合は、多少なりともEmacs Lispの書き方を知っておく必要があります(Emacs LispについてはChapter 3を参照)。どちらの方法も一長一短ですので、自分に合った方法を利用してみてください。

GUIで設定を行う

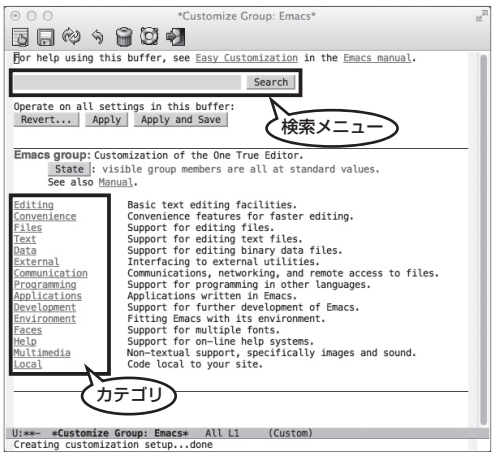
Emacsは自分で設定をすべて書く必要があるエディタだ、と思っている人もいるかもしれませんが、GUIで設定を行う手段も提供しています。GUIと言ってもテキストファイル上でボタンなどが使えるだけなので、イメージと違う人もいるかもしれません。とはいえ、自分で設定を書く際に必要になるEmacs Lispを使うことなく、かなり広い範囲の設定を行うことができます。

カスタマイズ機能(以降、customizeとします)を呼び出すには、ツールバーの[Options]→[Customize Emacs]→[Top Level Customization Group]を選択して呼び出します[注2]。

実行すると、グループと呼ばれる単位で整理されたメニューが表示されます(**図1**)。自分でインストールした外部パッケージでも、customizeに対応していればここから設定できます。初めてEmacsの設定を行う場合や、どこから手をつけていいかわからない場合には便利でしょう。

customizeを実行すると、画面の上部に検索メニューが表示されています[注3]。たとえば、インデントに関する設定をしたいけど設定項目がわからない、という場合は「indent」と入力して検索すると、その語を含む設定項目の一覧が表示されます。

▼図1　M-x customize



検索メニューの下にある2つのボタンは、そのバッファで設定した項目の適用や保存、リセットです。

- [Revert]……そのバッファで触った設定をリセットします
- [Apply]……バッファ内で変更した設定を適用します。この操作による適用は、Emacsを終了すると、次の起動時には引き継がれません
- [Apply and Save]……バッファ内で変更した設定を適用し、ファイルに保存します。Emacsを終了しても、起動時に引き継がれます

customizeに関する他の機能としては、**表1**のようなものがあります。

さて、customizeで実際に設定を行ってみましょう。customizeを呼び出して、検索ワードに「linum-mode」を指定して検索します。すると、**図2**のような画面に切り替わります。

8行目の「▽」の横、「Global Linum Mode」は設定名です。そして、その横に[Toggle]というボタンがあり、これをクリックすることで設定を切り替えます。Toggleボタンの右側には設定の

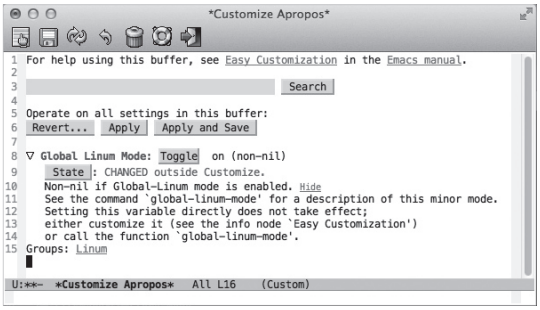
注2) Mac OS Xの場合はツールバーの[Emacs]→[Preference]、もしくは[Command]+[,]で呼び出し可能です。ちなみにコマンドの`M-x customize`でも同様です。

注3) `M-x customize-apropos`でも検索することが可能です。

▼表1　customizeに関係する操作

[Options]→[Customize Emacs]→[All Setting Matching...]	指定したパターンにマッチするオプションとフェイス、グループを一覧表示する
[Options]→[Customize Emacs]→[Browse Customization Groups]	オプションを折り畳み／展開できるツリーとして表示する
[Options]→[Customize Emacs]→[Custom Themes]	テーマー覧を表示する
[Options]→[Customize Emacs]→[Saved Options]	ファイルに保存された設定の一覧を表示する

▼図2　検索結果



▼リスト1　設定結果（.emacs）

```
(custom-set-variables
 ;; custom-set-variables was added by ➥
Custom.
 ;; If you edit it by hand, you could ➥
mess it up, so be careful.
 ;; Your init file should contain only ➥
one such instance.
 ;; If there is more than one, they ➥
won't work right.
 '(global-linum-mode t))
```

有効／無効が表示されています。

設定を反映するには、[Apply]ボタンをクリックします。[Apply and Save]の場合は設定を反映し、.emacsに保存します（.emacs.d/init.elがある場合はそちらに保存されます）。また、設定として何か入力する必要がある場合は、テキスト入力エリアが表示されます。設定を保存した後に.emacsを見ると、**リスト1**のように設定が追加されています。

自分で設定を書く

Emacsの標準機能やcustomize用のインターフェースを持つパッケージの設定を行う場合、customizeは大変便利です。しかし、インターフェースを持たないパッケージの設定や、自分で機能を拡張したりする場合はcustomizeで設定することができません。この場合、自分で設定やコマンドを設定ファイルに書く必要があります。

Emacsの設定を書くファイルは何種類か選ぶことができます。Emacsは起動時に「~/.emacs」か「~/.emacs.el」、「~/.emacs.d/init.el」を順番に探し、読み込みます。ですので、これらのファイルに設定を書いておけば、その設定が有効になります。

ちなみに、筆者は「~/.emacs.d/init.el」派です。.emacs.dディレクトリの中に設定ファイルをすべてまとめて管理できるので、バックアップや他の環境との同期が楽なためです。

たとえば、GUI設定で例にしたlinum-modeを有効にする設定を書く場合は、.emacsや.emacs.d/init.elなどに次のように記述します。

```
(global-linum-mode t)
```

これでcustomizeで有効にしたのと同じ結果となります。次節から紹介する定番の設定例は、設定ファイルに書き込めば使えるようになっていますので、まずはここから慣れていってください。

定番&お勧め設定

さて、Emacsにおける設定方法についてはおおまかに理解していただけたかと思います。この節では、Emacsを使っている人なら設定して

Note 設定ファイルを文芸的プログラミングする

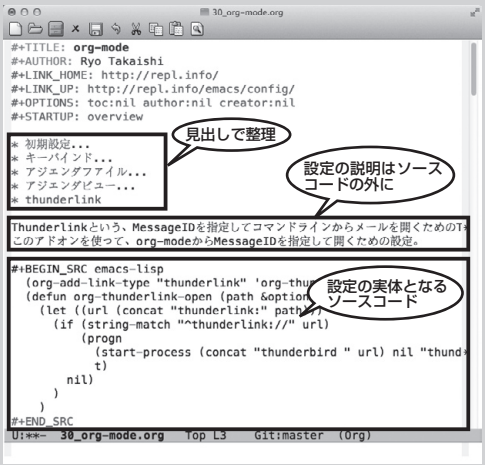
プロフィールにも書いたのですが、筆者はorg-mode[注a]が大好きで、Emacsの設定ファイルもorg-modeで書いています。具体的には**図A**のようにテキスト内に設定ファイルのコードを記述していて、ここからEmacs Lispのファイルを生成し、設定ファイルとして読み込んでいます。

なぜこのような方法をとっているのかというと、ソースコードとドキュメントを分離したいからです。詳細は割愛しますが、いわゆる文芸的プログラミング[注b]をEmacsの設定ファイルで行っているわけです。

しくみですが、org-modeにはテキスト中のプログラムコードだけを抽出して、ソースコードとして別ファイルを生成する機能があります。この機能を使って、各設定ファイル(orgファイル)がEmacsの起動時にソースコード(elファイル)へコンパイルされます。さらに、バイトコード(elcファイル)にコンパイルされ、最終的にロードされます。

org-modeで設定を書いておくことで、org-modeの機能を使ってHTMLやPDFなど、いろいろなフォーマットに変換できるようになるのが便利です。実用的かどうか、と言われるとそうでもないかもしれませんが、楽しみの1つとして取り入れてみてはいかがでしょうか?

▼図A　org-modeで設定



注a）Chapter 5参照。
注b）http://ja.wikipedia.org/wiki/文芸的プログラミング

いるのでは?というような設定を紹介していきます。これからEmacsを使ってみようという方や、使い始めたばかりでどういう設定をするのが良いのかわからない方は、まずこれらの設定をしてみてはいかがでしょうか。

見た目に関する定番設定

ここでは、設定のオン／オフで目に見える変化があるものを紹介します(**リスト2**)。

▼リスト2　設定すると目で見て違いがわかるもの

```
;;メニューバーを非表示にする
(menu-bar-mode 0)  ←①
;;ツールバーを非表示にする
(tool-bar-mode 0)  ←②
;;スクロールバーを非表示にする
(scroll-bar-mode 0)  ←③
;;モードラインにカーソル位置の行番号を表示する
(line-number-mode 1)  ←④
;;モードラインにカーソル位置の列番号を表示する
(column-number-mode 1)  ←⑤
;;カーソル位置の行を強調表示する
(global-hl-line-mode t)  ←⑥
;;起動時のメッセージを非表示にする
(setq inhibit-startup-message t)  ←⑦
;;yesかnoではなく、yかnで答えられるようにする
(defalias 'yes-or-no-p 'y-or-n-p)  ←⑧
;;ダイアログボックスを使わないようにする
(setq use-dialog-box nil)  ←⑨
;;対応する括弧を強調表示する
(show-paren-mode t)  ←⑩
```

①menu-bar-mode

フレームにメニューバーを表示するかどうかを決める設定。設定ファイルに書く際は、引数を1にすると表示を有効に、0にすると無効

②tool-bar-mode

フレームにツールバーを表示するかどうかを決める設定。menu-bar-modeと同様、設定ファイルに書く際は、引数を1にすると表示を有効に、0にすると無効

③scroll-bar-mode

スクロールバーを表示するかどうかをトグル設定。menu-bar-modeと同様、設定ファイルに書く際は、引数を1にすると表示を有効に、0にすると無効

④**line-number-mode**
モードラインに行数を表示するかどうかをトグル設定。menu-bar-modeと同様、設定ファイルに書く際は、引数を1にすると表示を有効に、0にすると無効

⑤**column-number-mode**
モードラインに列数を表示するかどうかをトグル設定。menu-bar-modeと同様、設定ファイルに書く際は、引数を1にすると表示を有効に、0にすると無効

⑥**global-hl-line-mode**
カーソルがある行をハイライト表示。今、画面のどこにカーソルがあるのかわからないというトラブルを防ぐことができる

⑦**inhibit-startup-message**
起動時のメッセージを表示しないようにする設定。inhibit-startup-screenのエイリアス

①～③は、使わないものは非表示にして画面をすっきりさせる設定です。使い始めたばかりだったり、これから使ってみようというユーザの場合は有効にしておいて、使わなくなってきたら非表示にするとよいかと思います。

⑦で設定できる起動時のメッセージですが、この画面からチュートリアルやマニュアルを参照できたり、新しくファイルやカスタマイズ画面を開くことができます。①～③と同様、不要になったら無効にすればよいでしょう。

Emacsを使っていると、"yes-or-no-p"と聞かれてyesかnoかを入力する機会があります。しかし、3文字もしくは2文字を毎回入力するのは少し面倒です。この対処法として、"yes-or-no-p"を"y-or-n-p"のエイリアスにしてしまうテクニックが⑧です。エイリアスとすることで、すべて「y」か「n」の1文字で回答できるようになります。

⑨のようにuse-dialog-boxの変数をnon-nil[注4]に設定すると、ダイアログから「はい」か「いいえ」を答える質問やマウスでファイル選択する操作が、ダイアログではなくミニバッファからの操作となります。

⑩のshow-paren-modeを有効にすると、カーソルが括弧上（括弧開きの場合は括弧の上、括弧閉じの場合は括弧の次）にあるとき、対応する括弧をハイライト表示してくれます。括弧が入れ子になるような場合、非常に便利な設定です。

▼リスト3　見た目は変わらないが設定しておくと便利なもの

```
;;ガベッジコレクションを実行するまでの
;;割り当てメモリの閾値を増やす
(setq gc-cons-threshold (* 50 gc-cons-ﾌ
threshold))  ←⑪
;;kill-lineで行末も削除する
(setq kill-whole-line t)  ←⑫
;;ログの記録量を増やす
(setq message-log-max 10000)  ←⑬
;;履歴存数を増やす
(setq history-length 1000)  ←⑭
;;重複する履歴は保存しない
(setq history-delete-duplicates t)  ←⑮
```

その他の定番設定

ここでは見た目はほとんど変化しませんが、設定しておくと便利なものを紹介します（**リスト3**）。

⑪のgc-cons-thresholdは、ガベッジコレクションを実行した後、次に実行するまでにLispオブジェクトに割り当てるメモリのバイトサイズです。閾値であり、このサイズを越えるとガベッジコレクションが実行されます。初期状態では40,000です。この値を大きくするとガベッジコレクションの実行回数が少なくなり、結果としてEmacsの動作が速くなるようです。

⑫**kill-whole-line**
この変数をnon-nilに設定することで、kill-line（C-kで実行できる1行削除するコマンド）で行末の改行文字も削除する

⑬**message-log-max**
メッセージログバッファが保持するログの最大行数を設定。標準では1,000だが、筆者は10,000行ほど保持するようにしている

注4）nil以外。通常、「t」を指定します。

⑭**history-length**

ミニバッファのヒストリを保存する数。標準では30だが、筆者は大きめに確保している

⑮**history-delete-duplicates**

non-nilにすることで新しくヒストリ追加する際に重複するものがあれば追加しない

定番のキーバインド

Emacsの設定といえば、キーバインド設定は欠かせません。どのキーにどのコマンドを割り当てるか、Emacsユーザなら日々悩んでいるはずです。また、キーバインドの設定は人によってかなり差が出ると思うので、いろいろな人のキーバインド設定を見るのはかなりおもしろいものです。ここでは定番のキーバインドについて紹介します(**リスト4**)。

⑯の設定は定番中の定番キーバインドではないでしょうか。**¥C-h**には初期状態ではヘルプコマンドが割り当てられています。しかし、入力しやすいこのキーバインドをヘルプに使わせるのはもったいないものです。そこで、カーソル前の1文字を削除するコマンドである**delete-backward-char**を割り当てます。ホームポジションのままBackward操作が行えるのは非常に便利です。

⑰の**backward-kill-word**はカーソル前の1語を消します。カーソル後の1語を消す**kill-word**に対応したコマンドです。たとえば、**delete-char**に**C-d**、**delete-backward-char**に**C-h**、**kill-word**に**M-d**、**backward-kill-word**に**M-h**を割り当てるときれいに対応させることができます。

⑱は指定した行にジャンプするコマンドです。

⑲の**new-line-and-indent**は、新しく空行を挿入して、メジャーモードに合わせてインデントを調整します。筆者は一石二鳥なこのコマンドがお気に入りです。

定番のパッケージ

ここではEmacs本体に同梱されているパッケージで、設定しておくと便利なものを紹介します(**リスト5**、**図3**)。

▼リスト4　定番キーバインド設定

```
;;カーソル前の文字を1文字消す
(global-set-key "¥C-h" 'delete-backward-char) ←⑯
;;カーソル前の1語を削除する
(global-set-key (kbd "M-h") 'backward-kill-word) ←⑰
;;入力した行にジャンプする
(global-set-key (kbd "M-g") 'goto-line) ←⑱
;;改行して、インデントの調節を行う
(global-set-key (kbd "C-m") 'newline-and-indent) ←⑲
```

■バッファに行番号を表示(linum-mode)

linum-modeはバッファの左側に行数を表示するパッケージです。今ファイルの何行目あたりを編集しているのかがひと目でわかり、大変便利です。人によっては、**line-number-mode**を有効にしているので不要、という人もいるかと思います。

⑳**global-linum-mode**

Emacs全体でlinum-modeを有効にする場合、引数に1を渡す。全体で無効にするには0を渡す

㉑**linum-format**

行数をどのようにして表示するかの設定。デフォルトは「dynamic」で、桁数に応じて幅が変わる。"%4d"のような文字列を設定すると最初から4桁分の幅を確保する

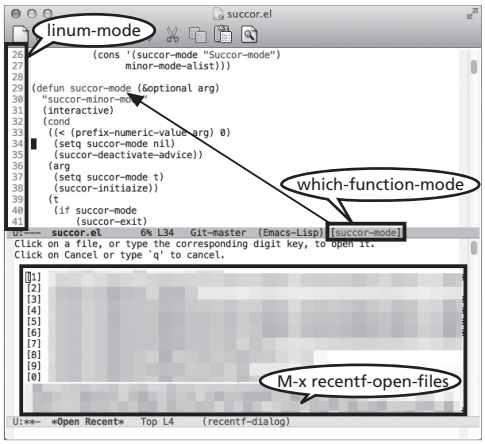
■今いるのはどの関数の中かモードラインに表示する

プログラムを書いているとき、今カーソルがあるのは何という名前の関数の中なのかを知りたいことがあるかと思います。**which-function-mode**は、モードラインに今カーソルがある関数の名前を表示してくれます。ちなみに、org-modeやoutline-modeだと、見出しを表示してくれます。

㉒**which-function-mode**

1を引数に渡すと有効に、0を渡すと無効になる

▼図3　定番パッケージ



■最近使ったファイルを記録する（recentf）

recentfは、最近開いたファイルやディレクトリを記録するパッケージです。履歴は"~/.recentf"に保存されます（**リスト4の㉓**）。保存先は**recentf-save-file**で設定できます（**リスト4の㉔**）。筆者は、"~/.emacs.d"以下を自宅と職場の端末で共有しているので、初期設定のまま（~/.recentf）にしています。

M-x recentf-open-filesでファイル一覧が表示されます。開きたいファイルで ENTER を押すことで、ファイルを開くことができます。直近10個にはショートカットが用意されており、1～0で開けます。anything.elやhelm[注5]と組み合わせるとかなり便利になると思いますが、ここでは割愛します。

㉘の**(run-with-idle-timer 30 t 'recentf-save-list)**では履歴の自動保存を設定しています。recentfがファイルに履歴を保存するのは、初期状態ではrecentf-modeを起動したときか、Emacsの終了時です。これだとEmacsが異常終了した場合には保存されません。そこで、**run-with-idle-timer**を使って、定期的に**recentf-save-list**を実行します。例では、30秒に一度保存しています。

注5）Chapter 3参照。

▼リスト5　定番パッケージの設定

```
;;常にバッファ左に行番号を表示する
(global-linum-mode 0)  ←⑳
;;行番号の表示領域として、
;;4桁分をあらかじめ確保する
(setq linum-format "%4d")  ←㉑
;;カーソルがどの関数の中にあるかを
;;モードラインに表示する
(which-function-mode 1)  ←㉒
;;最近使ったファイルを記録する
(require 'recentf)  ←㉓
(setq recentf-save-file "~/.recentf")←㉔
(setq recentf-exclude '("~/.recentf"))←㉕
(setq recentf-max-saved-items 5000)  ←㉖
(setq recentf-auto-cleanup '10)  ←㉗
(run-with-idle-timer 30 t 'recentf-save-list)  ←㉘
(recentf-mode 1)  ←㉙
```

㉖recentf-max-saved-items

直近何個の履歴を保存するかを設定する。初期状態は20個

㉗recentf-auto-cleanup

履歴リストをいつクリーンアップ（重複や除外の整理）するかを設定する。デフォルトはmodeでモードはオン。neverは自動的にクリーンアップしない。数値の場合は指定間隔ごと。"12:00am"のように時刻を文字列で指定するとその時刻にクリーンアップする

㉙recentf-mode

recentfを有効にする

おわりに

Emacsの設定方法と、定番の設定・パッケージについて紹介しました。GUIで設定を行うcustomizeは結構多機能なので、ぜひ一度使ってみてください。また、定番の設定ということで、比較的多くの人が設定しているであろう項目を紹介しましたが、もちろん人によっては「この設定は絶対有効にしない」というものもあると思います。使う人ごとに全然違うエディタになるのがEmacsです。ぜひ、こだわりの設定を見つけて、快適なエディタライフを送ってください。SD

Column 1

GNU 30周年とアジアのEmacs事情から考えたこと

Writer 井上 誠一郎(いのうえ せいいちろう)
アリエル・ネットワーク㈱
Mail inoue@ariel-networks.com
●Emacsの設定ファイル行数:400行

■GNUプロジェクト30周年

この原稿を書いている今、GNUプロジェクト30周年が自分のまわりで話題です。GNUのWebサイト[注1]によると、今から30年前の1983年9月27日にリチャード・ストールマン氏がGNUプロジェクトの開始宣言をネットニュースに投稿したようです。今こうして自分がEmacsに関する記事を書いていると、末端とはいえ、30年前のストールマン氏の思いを受け継ぐ重みを感じます。

■アジアのEmacs事情

個人的な話になりますが、ここ3年ほど、縁あってインド・中国の多くの大学生と話す機会があります。延べ100人以上と話をしました。全員IT系の有名大学の学生です。インドではインド工科大学(通称IIT)やIIIT(トリプルアイティー)、中国では精華大学や浙江大学、上海交通大学などの学生です。彼らが大学でどんな勉強をしているのか、どんなプログラミング言語を好んでいるか、どんな開発ツールを使っているのかなどの話をしました。当然、開発ツールの代表格はテキストエディタなので、彼らの好みのテキストエディタの話を聞く機会も多くあります。

結論から書くと、彼らに人気のテキストエディタはVimです。100人超に聞いたうち半数強がVimユーザでした。

残り半数がEmacsユーザです、と言いたいところですが、残念ながら残りの大半は統合開発環境(おもにEclipse)でコードを書き、テキストエディタを開発に使う習慣がありませんでした。使用テキストエディタを聞くとgedit(GNOMEのデフォルトエディタ)をあげる程度です。

100人超のうち、Emacsユーザには過去4人しか会えていません。Emacs特集にとっては悲しい現実ですが、これがインドと中国の実情です。2025年には世界の人口の4人に1人がインド人か中国人になると言われる中、Emacsにとっては暗い未来が見えてきそうです。

■彼らに伝えたいEmacsの価値

しかし未来は変えられます。Emacsの良さを彼らに伝えたい善意の心が半分、自分の得意スキルが陳腐化するのは面白くないと思う自己中心的な思惑が半分で、彼らにEmacsの良さを説こうと考えています。

Emacsの良さは拡張可能なアーキテクチャと自由なソフトウェアの意志の2つです。

ソフトウェアアーキテクチャとライセンスは本質的には直交しています。しかしEmacsの価値を考えるには2つを同時に考える必要があります。そして世界中にこの価値に共感した大勢のハッカーたちがいます。彼らが30年近い年月で作り上げた資産がEmacsです。Emacs本体はGNUプロジェクトの成果ですが、Emacsを語るとき、その周辺のEmacsアプリの資産とハッカー文化が暗黙に含まれています。これら、Emacs周辺のソフトウェア資産はもはや文化的遺産と呼んでもよいはずです。長い歴史と文化を持つインド人と中国人にもきっとこの価値は通じるはずです。

■最後に誤解のないように

VimもまたUnixツールボックスの偉大な文化的遺産です。筆者はVimを否定するために彼らにEmacsを勧めるわけではありません。Vimとは別の世界があることを伝えたいだけです。今年のインド・中国行きが楽しみです。GNUプロジェクト40周年に向けた種まきが始まります。SD

注1) http://www.gnu.org/gnu/initial-announcement.html

Chapter 3

Emacsを俺の嫁にする冴えたやり方

～Emacsは仮想的なLispマシン～

マルチコア化が進んでも、古強者Emacsは存分に存在感を見せつけてくれます。本稿ではEmacsライフを充実させるelispの使いこなしの手がかりを紹介します。EmacsはテキストエディタのLispマシンです。これをカスタマイズするのがelispです。パッケージをそろえ、お勧めのhelmやeshellを導入すれば、理想の「嫁」ができあがります。

Writer るびきち
(rubikitch) twitter@rubikitch

●Emacsの設定ファイル行数：13,601行、それでもダイエットしました。
●理想のEmacsを追い求めて17年。elisp好きです。S式好きです。括弧好きです。Emacs秘密サークル運営中！仲間は随時募集しています。長時間乾電池駆動モノクロ液晶軽量モバイルEmacsマシンを作るのが夢。
Blog■http://d.hatena.ne.jp/rubikitch/

Emacs充してますか

御無沙汰しています。るびきちです。Emacs充してますか？　本稿ではelispを紹介しますが、その前に近況を話させてください。

先日10年間愛用していたPCが故障して買い換えることになりました。シングルコアのPentium4から最新Core i7になったことで別次元の性能になりました。とくにLinuxカーネルのコンパイルは3時間以上かかっていたものがわずか10分、agによるカーネルソース検索もキャッシュなしで数秒で出るほどです。10年間の性能向上に驚くばかりです。

しかし、Emacsに関してはさほど速くなっていないのです。期待が大きかった分「まあまあ速くなったかな」程度なもので劇的な差はありませんでした。生憎突然故障してしまったため、計測データがないのは残念です。

原因は容易に想像できます。10年前からCPUのクロック周波数が頭打ちになり、その代わりマルチコアの路線にシフトしました。単一コアあたりの性能向上は数倍くらいでしかないので、シングルスレッドでしかないelispプログラムの速度向上はさほど期待できないのでしょう。おそらくCore i7とi3で比較しても大差ないと思います。

むしろPentium4のPCにSSDを導入したときの変化のほうが劇的でした。SSDは小さいファイルのランダムアクセス速度がHDDの数十～数百倍なのでEmacsの起動が超高速になったのです。

そして、Emacsを本格的に使い倒したいのならばOSはWindowsよりもGNU/Linuxだと実感しました。筆者はAtomのネットブック（メモリ1GB）も持っていますが、XPからDebian GNU/LinuxのSSDに換装しました。APTのおかげであっさりシステムが構築できました。おまけにファイルアクセスが速く、Unisonによる同期は数分かかっていたものが数秒で終わります。WindowsだとAPTがないうえにSSLや文字コードなどでいらない苦労をたくさんしましたが、余計なストレスがないのはうれしいものです。WindowsをUNIX的に使うのはどうしても限界があります。低性能と馬鹿にされて衰退してしまったネットブックも、今や電源投入後わずか20秒でEmacsが立ち上がる爆速起動マシンと化しました！

SSDにGNU/Linuxを入れれば、たとえほかの部分が貧弱であってもEmacsは超快適に使えます!!　Emacsに限らず体感速度には歴然とした差があります。

elisp！――チューニングの限りを尽くして

Emacs使いはなぜelispを書くのでしょうか？

setqひとつ、requireひとつ、コピー&ペーストであっても立派なelispです。

それは、自分にとって使いやすい環境を構築するためにほかなりません。好きなelispを選び、好きな変数を設定し、自作コマンドを書いたり……自作？　そう、自作です！

自作といえば、おそらく自作プログラムや自作PCを思い付く人が多いでしょう。Emacsの環境設定を自作PCのノリでやると、もっと楽しめます。なぜ自作したいのかというと、与えられたものをそのまま使うのではなくて、自分でコントロールしたいからですよね。せっかく自作PCをやるならばフリーでチューニングの限りを尽くせるソフトウェアを使うことはもちろん、OSもGNU/Linuxなどにしないともったいないです。

Emacsはそれ自体が自作環境、自作Emacsなのです！　Emacsはテキストエディタの顔をしたLispマシンです。いつでも好きなときに好きなelispプログラムを読み込め、自由に組み合わせられます。変数を設定すれば再コンパイルすることなく即時Emacsに反映されます。ほぼすべてをユーザがコントロールできます。

さあ、理想の環境の扉を開きましょう。

elispは基本的にカスタマイズとグルー言語

elispはご存じのとおりEmacsでの強力なカスタマイズ言語です。変数を設定することや自作コマンドで自分の使いやすいようにEmacsを変形できます。

そればかりかelisp単体でそれなりのアプリケーションが作れます。GnusやMewなどのメーラ、eshellというシェル、テトリスや五目並べといったゲーム、orgなどの巨大なメジャーモードがあります。けれども、elisp単体ではマルチコアを活かせないため、大きなデータを処理するには荷が重いです。

そのため、Emacsで大量のデータを処理するには外部プロセスとのやりとりが重要になってきます。たとえばM-x grepはgrepを呼ぶことで大量の文書から検索を行い、その結果をEmacsから簡単に開けるようになっています。emacs-w3mもw3mを呼ぶことで実用的な速度のWebブラウザになっています(**図1**)。このように外部プログラムで本処理を行い、elispはユーザインターフェースを担当するのが、Emacsで処理速度と操作性を両立させるコツです。elispのみでgrep相当のプログラムは書けますが何百倍も遅いです。餅は餅屋です。

▼図1　emacs-w3m(テキストブラウザ)

これからelispを覚えるならば、キーカスタマイズやちょっとしたコマンドを作ってみてください。これだけで十分快適になります。それらを通してLispという言語を好きになれば、堂々と「Emacsは俺の嫁」宣言してください(笑)。

自作Emacsに何を載せる？

Emacsは仮想的なLispマシンです。すでにCPU、メモリなどが装着してあり、これだけでも動作する小さなPCをイメージしてください。そして、このPCの特徴は、無限個の拡張スロットが存在することです。Emacsの環境構築とは、この仮想的な自作PCを構築することです。

自作PCならば好きなパーツを選べます。たとえばマザーボードにサウンド機能(オンボードサウンド)がついていますが、より高音質を楽しみたいならば外部サウンドカード・オーディオ

インターフェース・DACなどを増設します。それと同様にEmacsの標準機能に満足しなければ外部のelispプログラムを導入することになります。そして、UEFI(BIOS)でパラメータチューニングするように、変数やキーバインドを調整していきます。

では何を入れればいいのでしょうか？……これは人それぞれ好みや背景が異なるので一概に言えないのですが、筆者の長年の経験から主観を交じえてお勧めを紹介していきます。

パッケージの設定

まずやっておくべきことは、elispパッケージシステムが使えるようにすることです。Emacs24からpackage.elが標準添付となり、外部elispプログラムが簡単に導入できるようになりました。Debian系のAPT、PerlのCPAN、RubyのGemに相当するものが、Emacsの世界にもやっと登場したのです。

デフォルトではELPAしか有効になっていないので、事実上標準のMarmaladeとMELPAも有効にしておきます。これらも有効にすることでインストールできるパッケージの幅が一気に広がります(**リスト1**)。

パッケージをインストールする前には**M-x package-refresh-contents**あるいは**M-x list-packages**を実行して最新情報に更新しておく必要があります。

M-x list-packagesからは直感的にパッケージをインストール・削除できます。**i**でインストールするパッケージを選択、**d**で削除するパッケージを選択し、**x**でインストールします。

とりわけパッケージ更新機能は便利で、**U**で最新パッケージのインストールと(現在インストールされている)古いパッケージを削除を同時にしてくれます。

helm——Emacsの「舵」

そして、今Emacsを使うのならば忘れてはならないのがhelmです。helmはanything.elの進化形で、細かい違いはあれど同じような使い勝手です。

helmは、Emacsの「舵」と名乗るだけあり、Emacsのパラダイムを逆転させます！　通常はコマンドのあとに処理対象を指定するのですが、helmは処理対象を指定してからアクションを指定するのです。たとえば、**C-x C-f test.txt**で**test.txt**を開きますが、helmを使うと**test.txt**を指定したあとに「開く」というアクションを指定します。もし、最近使ったファイルに**test.txt**が含まれているならば、**test**と入力すれば**test.txt**が出てくるのでそのまま Return を押せば開けます。アクションは対象を選択したあとにTABを押せば一覧できます。処理対象とアクションは独立しているので、「最近使ったファイルXをview-modeで開く」のように元のコマンドが存在しないアクションも行えます。言葉で説明するよりも、実際に使えばいかに便利かがわかることでしょう。

```
M-x package-install helm
M-x package-install helm-migemo
```

helmの設定はinit.elの後のほうに追加してください。なぜなら、helmは多数のほかのelispに依存しているからです(**リスト2**)。

▼リスト2　helmの最小設定

```
(require 'helm-config)
;; migemoが設定されているなら有効になる
(require 'helm-migemo)
```

▼リスト1　パッケージを有効にする

```
(require 'package)
(add-to-list 'package-archives '("marmalade" . "http://marmalade-repo.org/packages/"))
(add-to-list 'package-archives '("melpa" . "http://melpa.milkbox.net/packages/") t)
(package-initialize)
```

初めてhelmを使うのであれば、バッファと最近使ったファイルを選択できる**M-x helm-mini**を使ってみましょう。基本的な使い方は画面に出てくるので自ずとわかってきます。カレントディレクトリのファイルやブックマークなども選択できる**M-x helm-for-files**もあります。

helmについて詳しく解説すると、1冊の本になるレベルなので特別に便利なコマンドをいくつか紹介しておきます。

M-x helm-occurは**occur**と**isearch**を合体させたようなもので、極めて強力な機能です。**migemo**も効きます。このためだけにhelmをインストールしても損はしません。

M-x helm-aproposはコマンド・関数名・変数名を検索し、説明を表示したり定義へジャンプしたりできます。

M-x helm-colorsは色名・RGB・フェイスを選択できます。フェイスのカスタマイズもそこからできます(**図2**)。

他にも無数のコマンドが定義されており、「helmを制する者がEmacsを制する」と言っても過言ではありません。基本的には**anything.el**を踏襲しているので、**anything.el**の解説も参考になります。

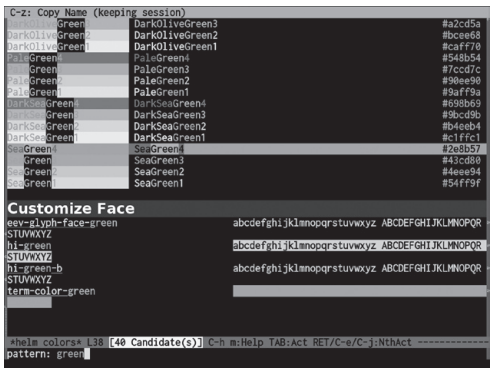
open-junk-file——保存できる*scratch*バッファ

scratchバッファはEmacsを起動したときに作られている落書き用バッファです。メジャーモードは**lisp-interaction-mode**で、elisp式を即評価できるようになっています。コードを殴り書きするためのものです。

しかし、***scratch***バッファにはEmacsを終了したら消えてしまう欠点があります。PCの電源を切ると内容が消えてしまうRAMディスクのようなものです。殴り書きとはいえせっかく書いたコードが消えてしまうなんて、とてももったいないことです。将来同じような問題に出喰わしたときに再び解かないといけないのは時間の無駄です。

scratchバッファを保存するelispは存在しますが、より便利な解決策を用意しました。特定のディレクトリに殴り書き用のコードを全部保存すればいいのです。そうすればあとでgrep検索できます。

▼図2　RGBのカスタマイズ



▼リスト3　open-junk-file.elの設定

```
(require 'open-junk-file)
(setq open-junk-file-format "~/junk/%y ↵
%m%d/%H%M%S.")
(global-set-key (kbd "C-x C-z") 'open- ↵
junk-file)
```

これを行うのが拙作**open-junk-file.el**です。**~/junk**以下に日付をベースとしたファイル名を開くだけの小さなelispです。パッケージからインストールします。

```
M-x package-install open-junk-file
```

そして、**リスト3**の設定を行います。

open-junk-file-formatはファイル名のフォーマットを指定します。途中のディレクトリは必要ならば自動で作成します。**2013/12/25 12:34:56**に**M-x open-junk-file**を実行したら、ミニバッファに「**~/junk/131225/123456.**」というファイル名が出てきます。拡張子を入力して[Return]を押せばファイルが用意されます。もちろんファイル名は自由に変更できます。たったこれだけですが、サッとファイルを開いて任意の言語で殴り書きできるのは便利ではないでしょうか。

global-set-keyは、Emacs全体で有効にな

るキーにコマンドを割り当てます。キーの指定方法はいろいろありますがkbdを使うのが手軽かつ確実です。

eshell——OS非依存のEmacsシェル

Emacsひきこもり生活をしている筆者が使っているシェルはeshellです(**図3**)。名前が示すようにelispで書かれたシェルで、Emacsと融合しています。フルelispのため、elispを書けば完全に自分の思いどおりの挙動にカスタマイズできます。通常の端末上のシェルでのカスタマイズは限定的なのに対して、eshellには無限の可能性があります。気に入らない挙動はelispで修正できるので、自分好みのeshellを構築できます。eshellのパワーはelispのスキルに比例して大きくなります。理想のシェル環境を追い求めるのもelispプログラミングの動機としてアリです。

フルelispにはOSに依存しないという付随的メリットがあります。UNIX系OSに馴染んでいる人がWindowsマシンを使わされている場合でも、eshellならば何の障害もなく使えます。

eshellにはelisp式を評価する機能があります。elispの評価方法は`*scratch*`や`M-:`や`M-x ielm`などがありますが、eshellも仲間に加えてください。`M-x shell`よりもEmacsとの親和性が強く、かつelisp式が評価できるのがeshellです。helmとeshellはEmacs上の2大インターフェースではないでしょうか。

シェルコマンドの出力をバッファや変数にリダイレクトするeshellならではの機能もあります。ファイルの内容を変数に設定するのはとても簡単です。

eshellのエイリアスは一度定義するとファイルに保存されるので、他のeshellでも即使えるうえ、Emacs終了後にも有効になります。

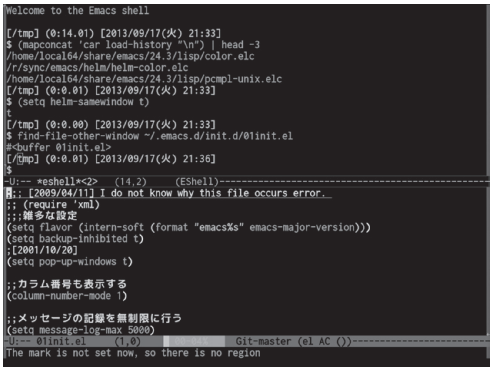
ここまでほめちぎったものの、eshellにも欠点が2つほどあります。カバーできるので安心してください。

1つは、w3mやalsamixerなどの画面志向のプログラムがそのままでは動作しないことです。端末を作成して呼び出す、あるいはGNU Screenやtmuxで新たなウィンドウを作成するaliasを定義すれば対処できます。速度を求めないならばelispで書かれたターミナルエミュレータ`M-x term`を使う方法もあります。

もうひとつは、リダイレクトやパイプの実装が甘いことです。「`cat < file`」といった入力リダイレクトが実装されていません。さらに出力リダイレクトとパイプがバイナリデータや大容量のデータに使えません。Emacsのバッファを一時保存領域として使っていることからくる制限です。ポータビリティとEmacsとの親和性が高い反面、性能が犠牲になっています。とはいえ小容量のテキストを扱うには十分です。eshellのリダイレクトで扱えない場合は、通常のシェルを呼び出すことで解決できます。

eshellは標準添付なので設定なしでも使えます。`M-x eshell`でeshellを立ち上げます。`C-u`を付けると新しいeshellになります(**図4**)。

▼図3 OS非依存のEmacsシェルeshell



elispと戯れたいなら

elispを書くならば、ぜひとも入れておきたいパッケージがあります。いずれもpackageからインストールできます。

・lispxmp……式の後ろに値を注釈する

・paredit……Lispの括弧の対応に沿った編集モード
・auto-async-byte-compile…保存時に自動バイトコンパイル
・rainbow-delimiters……括弧に色をつける

```
M-x package-install lispxmp
M-x package-install paredit
M-x package-install auto-async-byte-compile
M-x package-install rainbow-delimiters
```

lispxmpは行末の「**; =>**」の後にその行の式の値を注釈するものです。Rubyの**xmpfilter**をelispに移植しました。実行結果を外部に書くのではなく、ソースコードに埋め込むのでとても見やすくなります。elispの学習では必携のツールです。

pareditはLispの括弧を空気にしてくれるelispです。たとえば、開括弧を入力したら閉括弧も入力されてその間にカーソルが行きます。閉括弧だけを消すことはできなくて()の間にカーソルが行ったときに両方の括弧を同時に消すようになっています。elispでは括弧の対応が崩れたときのダメージが甚大なので、parediteを使うことで崩れないように保護します。

auto-async-byte-compileは保存と同時にバイトコンパイルをしてくれます。elispを更新してもバイトコンパイルを忘れると変更が反映されないという困った仕様なのでバイトコンパイル忘れを自動的に予防します。名前が示すとおり、バイトコンパイルはバックグラウンドで行われ、作業が中断しません。

rainbow-delimitersは括弧の階層ごとに色が付きます。どれに対応する括弧なのかがひと目でわかるようになります(**リスト4**)。そして、画面が楽しくなります(笑)。

今後……

1975年に最初のEmacsが産声を上げて40年近くになりました。今日広く使われているGNU Emacsも来年で30才になります。そして今なお開発は続けられています。本物のLispインタプリタを搭載したこのテキストエディタは筆者含め熱狂的なファンを数多く生み出してきました。Emacsは不滅であり、Emacsのスキルは一度習

▼図4　eshellの実行

```
Welcome to the Emacs shell

~ $ ls ~/.emacs.d/init.el
/r/.emacs.d/init.el

エイリアスを定義
~ $ alias ll 'ls -l $*'
~ $ ll ~/.emacs.d/init.el
-rw-r--r-- 1 rubikitch users 190  8月 28 04:23 /r/.emacs.d/init.el

パイプ
~ $ cat ~/.emacs.d/init.el | wc
      5      14     190

入力リダイレクトは未対応なのでshを呼び出す
~ $ sh -c 'wc < ~/.emacs.d/init.el'
  5  14 190

バイナリのパイプも扱えないのでshを呼び出す
~ $ sh -c 'wget -O- http://download.savannah.nongnu.org/releases/ratpoison/ratpoison-1.4.6.tar.xz | tar xJvf -'
……(省略)……

バッククォートは {}
~ $ ll { which emacs-24.3 }
-rwxr-xr-t 1 root root 14453433  7月 16 18:48 /usr/local/bin/emacs-24.3

変数にリダイレクト
~ $ cat ~/.emacs.d/init.el | wc > #'a
~ $ echo $a
      5      14     190

w3mはeshellでは動かないのでターミナルエミュレータから起動
~ $ urxvt -e w3m http://www.gnu.org/software/emacs/

elisp評価機能
~ $ (setq x 100)
100
~ $ (+ x 1)
101
~ $ (mapconcat 'identity load-path "¥n")
/r/.emacs.d/elpa/all-1.0
/r/.emacs.d/elpa/auto-complete-20130330.1836
/r/.emacs.d/elpa/bm-20130226.1440
……(省略)……
```

▼リスト4　elisp環境設定

```
;; emacs-lisp-modeでC-c C-dを押すと注釈
(require 'lispxmp)
(define-key emacs-lisp-mode-map (kbd "C-c C-d") 'lispxmp)
;; 括弧の対応を取りながら編集
(require 'paredit)
(add-hook 'emacs-lisp-mode-hook 'enable-paredit-mode)
;; ~/junk/以外で自動バイトコンパイル
(require 'auto-async-byte-compile)
(setq auto-async-byte-compile-exclude-files-regexp "/junk/")
(add-hook 'emacs-lisp-mode-hook 'enable-auto-async-byte-compile-mode)
;; 括弧に色付け
(add-hook 'emacs-lisp-mode-hook 'rainbow-delimiters-mode)
```

得すればずっと使えます。筆者は新しいコンピュータには真っ先にEmacsをインストールし、eshellが使えるようになったら一安心します。大工にとっての道具箱のような存在だからでしょう。

Emacsの何が人を惹き付けるのでしょうか？

恐らく、Emacsが醸し出す自作PCのような雰囲気や育成ゲーム要素でしょう。Emacsにおける不満は、elispを書けばたいてい解決できます。そして、チューニングを施しつつ長く使っていくうちに自分の手足の一部になってきます。

Emacsとの相性はelispの好感度に比例します。とはいえ、elispの深みに無理して入る必要はないです。すでに素晴しいelispプログラムはたくさん存在するので、それらをうまく活用できれば十分です。キーカスタマイズができれば理想の環境に近付けます。余裕があればコマンドを作ったり、アドバイスを書けばよいです。

elispは手軽に始められます。アイデアがすぐに実装できる楽しい言語です。コマンドが自作できるようになれば、プログラミングの楽しさを再発見できます。同時に、elispは本物のLispなのでLisp入門にも適しています。今使われている言語もLispを参考にしたものが多いので、常用している言語を違った角度でとらえることもできます。

終わりに

最後に宣伝させてください。筆者はEmacsの秘密サークルを運営しています。メルマガで即戦力となるスキルを配信し、読者の疑問に個別対応する活動をしています。メルマガ『Emacsの鬼るびきちのココだけの話』を講読することで参加できます。お互い真剣に責任を持って活動していくため、月々512円となっています。有料にすることで、筆者は質の高い情報を毎週配信する義務が生じます。読者は無料情報よりも真剣に受け取るようになり、上達がグーンと早くなります。Emacsという一生モノのスキルに投資[注1]してみませんか？

Emacsが誕生してからかなりの年数がたっているのに、人々を熱狂の渦に誘うelispプログラムが誕生しているのは、とてもすばらしいことです。ここでは紹介しきれなかった、bm、org-babel、auto-complete、calfw、magitなどもお勧めです。

「Emacsはとても楽しい！」

ということで筆を置きたいと思います。SD

注1）http://www.rubyist.net/~rubikitch/melmag.html

Chapter 4

いつもの環境がどこでも使える! 絶妙の引きこもり型エディタ

～OSを渡り歩くユーザも安心～

OSが起動したらEmacsを実行するだけで、仕事はすべてEmacsの中ですませる。そんなスタイルだって不可能じゃないと思わせるほどの多機能ぶり。日本語入力システムまでEmacsに内包してしまえば、OS間で異なるインプットメソッドの扱い方をも解消できます。本章を読んで、Emacsの懐の広さをあらためて感じてください。

Writer 水野 源
(みずの はじめ)

●Emacsの設定ファイル行数：約1,000行
●高校生の頃「なんかカッコいい」という中二心あふれる理由から、MS-DOSではなくBSDを使ったのがEmacsと出会ったきっかけでした。それ以来、Emacsは生活の一部になっています。

はじめに

Ubuntu Japanese Teamの水野です。筆者は日常的にEmacsを利用しており、gihyo.jpのWeb連載「Ubuntu Weekly Recipe」においても、何度かUbuntuとEmacsに関する紹介記事を執筆しています。そんな関係もあってか、今回光栄にもEmacs特集に執筆の機会をいただけることになりました。

筆者はEmacsそのものやEmacs Lispプログラミングについてはそれほど詳しいわけではないため「本当に自分なんかの記事でいいのか？」という思いが正直なところです。ですがせっかくの機会ですので、筆者なりに感じているEmacsの魅力や、Debian/Ubuntuの中の人が考えるEmacs環境(の一部)といった面から、Emacsを紹介したいと思います。

Emacs is Environment

Emacsとは、ご存じのとおりUnixライクなOSにおいて、Vimと人気の双璧をなすテキストエディタのようなものです。Emacsはまぎれもなくテキストエディタとして使えるアプリケーションなのですが、世間のEmacsユーザは、Emacsをもっと大きなものとしてとらえているようです。筆者も周囲に「Emacsとはなんぞや？」と聞いてみたところ、

「Emacsは宇宙だ」
「人生に必要なものはすべてEmacsの中にある」
「わたしの、最高の友達」

などなど、さまざまな答えが返ってきました。おそらく読者の皆さんの周りにいらっしゃるEmacsユーザも、おおむね似たような意見をお持ちなのではないかと思います。

Emacsのサイト[注1]では、Emacs自身を「GNU Emacs is an extensible, customizable text editor - and more.」と説明しています。Emacsのキモは、この「extensible」の部分にあります[注2]。

Emacs Lispというプログラミング言語があります。Emacs Lispはその名のとおり、Emacsで利用されているLispの方言です。EmacsはEmacs Lispプログラムを評価することによって、自分自身の機能を拡張することができます。実際、Emacs自身は非常にコアな部分の機能しか持っておらず、テキストエディタ部分をも含む機能の大部分は、Emacs Lispプログラムとして実装されています。そしてホームディレクトリに置かれるEmacsの設定ファイルもまた、Emacs Lispプログラムそのものです。Emacsの設定とは、Emacs Lispプログラムを記述して、

注1) http://www.gnu.org/software/emacs/
注2) あるいは「and more」の部分かもしれません ;p

自身の拡張を行うことに他なりません。このことからEmacsとはつまり、Emacs Lispを実行するためのプラットフォームであると言えるでしょう。

Emacs Lispで実装された、Emacs上で動作するアプリケーション（Emacsの拡張機能）は数多く存在します。Emacsのことをよく知らない人であれば、テキストエディタのプラグイン機能のようなものを想像するかもしれません。しかしEmacsのそれは、一般的なプラグインの範疇に到底収まりきるものではありません。ファイラーやメールクライアント、Webブラウザ、IRCクライアント。最近ではTwitterクライアントや、動画編集機能まであります。一般的なコンピュータ上で利用される機能はおおよそ実装されているのではないか。そうとさえ思えるその多機能ぶりは、1つのOSのようだと言われることもあるほどです。

Emacsは単なるエディタではなく、環境である

「Emacs」の名前の由来は「Editing MACroS（エディタのマクロ）」だそうですが、今現在「Emacs」の「E」は、「Environment」の「E」と言えるかもしれません。

▼写真1　Ubuntu Touch上の端末で動作しているEmacs

いつもの環境を、あらゆる場所で

当然の話ですが、Emacs Lispで記述された拡張機能は、Emacsが動けば動きます。Emacsが移植されている環境であれば、普段使っている拡張機能をそのまま利用することができるわけです[注3]。

たとえば、UbuntuではモバイルデバイスOS向けに「Ubuntu Touch」というOSを開発中です。Ubuntu Touch上ではまだ日本語インプットメソッドが動作していないため、日本語を入力することができません。ですが、ターミナルはすでに実装されています。Ubuntu TouchではARM用のUbuntuのパッケージがそのまま利用できるため、Emacsをインストールし、ターミナル上で動作させることができます。ですのでEmacs上で日本語入力システムのDDSKK（後述）を利用すれば、Ubuntu Touchでも日本語入力が可能になるわけです[注4]（**写真1**）。もちろん、Ubuntu Touchのソフトウェアキーボードでは、実際問題としてEmacsを実用レベルで運用することはできません[注5]。ですがこのように未完成なOS上であっても、Emacsさえ動けばいつもの環境で作業できるというのは、Emacsの特徴がよくわかるエピソードではないかと思います。

Emacsさえあれば、いつもの機能がすべて使える

多機能さ、Emacsさえ動けばなんとかなると

注3）もちろん外部プログラムをキックするような拡張機能はそれらのプログラムに依存するため、この限りではありません。

注4）ちなみにUbuntu Japanese Teamのリーダーである小林さんはこれを見て、「Emacsの中でできたからといって、そのOSでできたことにはならんだろう」と言っていました。やはりEmacsという環境は、ある意味隔離された特別な場所のようです。

注5）Ubuntu Touchのソフトウェアキーボードには独立したCtrlキーがなく、Ctrl＋任意のキーという入力を行うことができないためです。C-x（一般的にはカット）やC-c（一般的にはコピー）などは特殊な操作で入力できるため、Emacsを終了することはできますが、嬉しくもなんともありません。

いう安心感、そして絶妙の引きこもり感[注6]が、Emacsの魅力の1つではないかと筆者は考えています。

愛用のEmacs拡張

まっさらな状態のEmacsを、ビルトインの機能のみで使うなどということは、まずありえないでしょう。筆者のEmacsにも自分向けのカスタマイズがされており、日常的に多くの拡張機能を利用しています。

ここでは筆者が利用している拡張機能の中から、「筆者の生活に必要不可欠なもの」「あまりメジャーではないけれども、筆者が気に入っているもの」をいくつか紹介します。

twittering-mode

デスクトップの隅にはTwitterクライアントが常駐し、常にタイムラインをチェックしている……。本誌の読者であれば、そのような人も珍しくないと思います。

一般的なEmacsユーザであれば、Twitterやメールのようなテキストの読み書き作業は、すべてEmacs上で完結させたいと考えるのが自然です。そこで登場するのが、Emacs上で動作するTwitterクライアント「twittering-mode」です。twittering-modeは、ツイートの投稿やタイムラインの表示といった基本的な機能はもちろん、ハッシュタグやツイートの検索、複数のタイムラインをまとめて表示するマージタイムライン機能などを搭載しています。**リスト1**はtwittering-modeの設定例です。

twittering-modeは**M-x twit**を実行して起動します。デフォルトではhomeタイムラインがバッファに表示されますが、最近のワイドディスプレイを搭載したPCであれば、ウィンドウを分割して複数のタイムラインを表示したくなるかもしれません。これはラッパー関数を定義すれば[注7]、簡単に実現できます。**リスト2**では、ウィンドウを左右に分割した後にtwittering-modeを起動し、それぞれのウィンドウでhomeタイムラインと、repliesタイムラインを表示する、my-twit関数を定義しています。また新着ツイートを受信した際のフックを定義することも可能です。マニュアル[注8]にあるとおり、**リスト3**のようなフックを定義すると、Ubuntuの Notyfi OSDを利用して、新着ツイート数をデスクトップに表示させることができます(**図1**)。

このように、ちょっとした工夫で自由に処理を拡張できるのも、Emacsのよい所ですね。

▼リスト1　twmodeの設定の例

```
(setq twittering-timer-interval 90) ↵
;; 90秒ごとに自動更新
(setq twittering-display-remaining t)  ↵
;; モードラインにAPIの残数を表示
(setq twittering-icon-mode t)  ↵
;; アイコン画像を表示
(setq twittering-edit-skeleton ↵
'inherit-any)  ;; 返信時、返信先に↵
含まれるハッシュタグと@をすべて引き継ぐ
```

▼リスト2　起動時に複数のタイムラインを表示する関数

```
(defun my-twit ()
  (interactive)
  (delete-other-windows)
  (split-window-horizontally)
  (balance-windows)
  (twit)
  (cond
   ((twittering-account-authorized-p)
    (switch-to-buffer ":home")
    (other-window 1)
    (twittering-visit-timeline ":replies")
    (other-window 1))
   (t
    (delete-other-windows))))
```

注6）「ログインしたらとりあえずEmacsを起動し、あとはEmacsの中ですべての作業を行う」といったスタイルで作業を行うエンジニアも少なくありません。とくにGUIを持たないサーバ環境ではなおさらでしょう。

注7）参考：第4回 関西Emacs勉強会 初心者向けtwittering-modeのススメ
http://www.slideshare.net/masutaka/twitteringmode

注8）http://www.emacswiki.org/emacs/TwitteringMode-ja#toc26

▼リスト3　notify-osdの設定

```
(if (locate-library "notify-send" nil exec-path)
    (add-hook 'twittering-new-tweets-hook
              #'(lambda ()
                (let ((n twittering-new-tweets-count))
                  (start-process "twittering-notify" nil "notify-send"

"-i" "/usr/share/emacs/24.3/etc/images/icons/hicolor/48x48/apps/emacs.png"
                                 "New tweets"
                                 (format "You have %d new tweet%s"
                                         n (if (> n 1) "s" ""))))))
```

▼図1　Notyfi OSDを利用したポップアップ表示

▼図2　org-modeで書かれている本原稿のテキスト

org-mode

「org-mode」は、Emacs用の強力なアウトラインエディタです。筆者はすべての原稿を、org-mode上で書いています。もちろん本稿も例外ではありません（**図2**）。

org-modeでは階層化されたツリー構造のテキストを簡単に記述することができます。その機能は膨大でとても書ききれませんが、筆者はメモや原稿執筆に、おもに次の機能に魅力を感じて利用しています。

- 見出し単位で階層の上げ下げ、移動が行える
- 脚注を簡単に挿入できる
- 表の作成ができる
- 画像やリンクの挿入ができる
- HTMLやLaTeXとしてエクスポートできる

詳細は本特集のChapter 5も参考にしてください。

popup-select-window

Emacsでは、フレームの中に複数のウィンドウを同時に表示することができます[注9]。ウィンドウを分割している状態で**C-x o**を押すと、カレントウィンドウを次のウィンドウに切り替えます。しかしこれでは、シーケンシャルにしかウィンドウを切り替えられません。また、2ストロークのキーバインドであるためキーの連打がしにくく、3つ以上のウィンドウが存在している場合、切り替えに非常に苦労することになります。

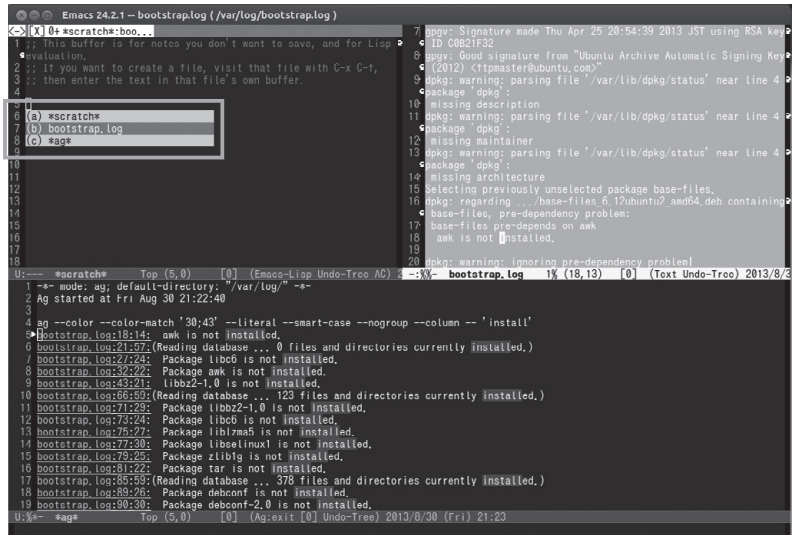
このようにEmacs標準のウィンドウ操作コマンドはお世辞にも使い勝手が良いとは言えません。これを改善するための拡張機能はいろいろ

注9） Emacsのウィンドウとは、一般的なGUIにおけるウィンドウとは異なるものを指します。Emacsでは、いわゆるGUIにおけるウィンドウのことをフレーム、フレームの中でバッファを表示している領域のことをウィンドウと呼んでいます。

▼リスト4　popup-select-windowの設定例

```
(global-set-key "¥C-xo" 'popup-select-window)  ;; C-x oにpopup-select-windowをバインド
(setq popup-select-window-popup-windows 2)  ;; ウィンドウが2つ以上存在する際にポップアップ表示する
(setq popup-select-window-window-highlight-face '(:foreground "white" :background "orange")))
;; 選択中のウィンドウは、背景をオレンジにして目立たせる
```

▼図3　3つに分割したウィンドウをポップアップで選択している状態



と存在するようですが、その中でも筆者は、直感的でわかりやすい「popup-select-window」を愛用しています。

popup-select-window[注10]は、現在開いているウィンドウの一覧をポップアップ表示し、**C-n**/**C-p**や英字キーで選択することを可能にします。変数popup-select-window-popup-windowsには、ポップアップを表示するウィンドウの最小個数を設定します。デフォルト値は「3」なので、ウィンドウが2個しかない場合は標準のother-windowが実行され、3個以上のウィンドウが存在した場合にのみ、ポップアップが表示されます(**図3**)。

このように空気(ウィンドウ数)を読んで挙動を変えてくれるため、**C-x o**にpopup-select-windowをバインドし、標準のother-windowと置き換えてしまうのがお勧めです(**リスト4**)。筆者はpopup-select-window-popup-windowsに「2」を設定し、ウィンドウが複数存在する場合には必ずポップアップ動作を行うようにしています(**リスト4**)。これは「ある機能は常に同じ動きをしてくれる」方が筆者の好みだからです。

popup-select-window-window-highlight-faceでは、選択している切り替え先ウィンドウの背景色と文字色を設定できます。筆者は通常の背景色と極端に違う色を設定して、選択中のウィンドウを目立たせています(**リスト4**)。

ag.el

「The Silver Searcher[注11]」(またの名をag)という、高速なgrepの代替プログラムが最近話題

注10) http://www.emacswiki.org/emacs/popup-select-window.el

注11) https://github.com/ggreer/the_silver_searcher

になっています。agそのものはgrepと同様、コマンドラインから実行するユーティリティプログラムで、「ag.el[注12]」は、このagをEmacs上から利用するためのフロントエンドにあたります。

Emacs上で**M-x ag**を実行すると、ミニバッファに「Search string:」と表示されますので、検索したい文字列を入力します。続いてミニバッファに「Directory:」と表示されたら、検索対象とするディレクトリへのパスを入力してください。なおデフォルトで“現在開いているバッファのファイルが存在するディレクトリ”が指定されています。検索が完了すると「*ag*」というバッファが作成され、検索結果が表示されます。このバッファ上で編集したい行にカーソルを動かして**C-c C-c**を押すと、該当するファイルを新しいバッファで開き、該当個所へカーソルがジャンプします(**図4**)。

ag.elには「ag-project」という関数も用意されています。これは現在開いているファイルがバージョン管理下にある場合、そのコードリポジトリのルートディレクトリを、自動的に検索対象に指定するという動作を行います。つまり検索対象ディレクトリの指定を省略して、コードツリー全体を検索することができるわけです。git cloneしたソースコードをいじっているような場合は、**M-x ag**よりも**M-x ag-project**を利用すると便利でしょう。

ちなみにDebianにおいては、筆者がagとag.elのパッケージメンテナンスを行っています。詳細は、筆者によるUbuntu Weekly Recipeの記事も参照してください[注13]。

鬼軍曹.el

C-n/p/f/bでのカーソル移動をはじめとしたEmacsのキーバインドは、Emacs以外の場所でも多く採用されています。たとえばシェルが代表的な例でしょう。このように、UnixライクなOS上で作業するのであれば、たとえEmacsそのものは使わなくても、Emacsキーバインドに慣れておくほうが便利です。

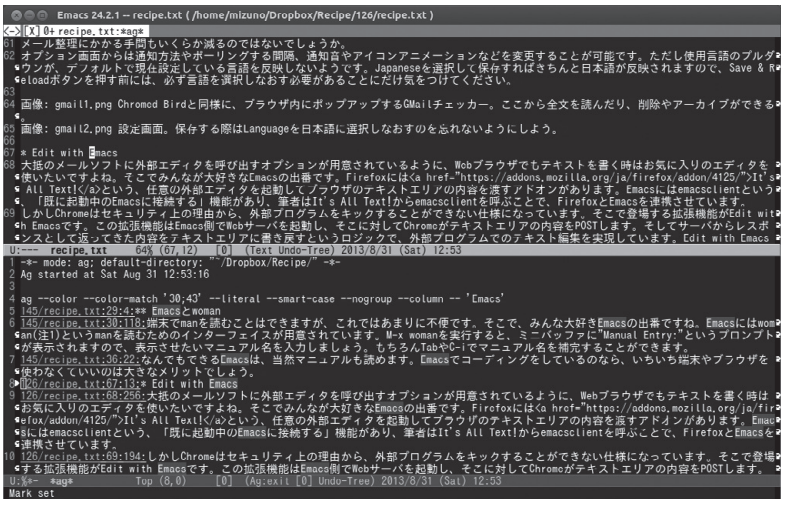
「鬼軍曹.el[注14]」は、Emacsキーバインドを強制するマイナーモードです。具体的には鬼軍曹を有効にした瞬間から、カーソルキー、ENTERキー、TABキー、BackSpaceキーといった、Emacs的には必須ではない(しかし一般的には主要な)

注12) https://github.com/Wilfred/ag.el

注13) http://gihyo.jp/admin/serial/01/ubuntu-recipe/0287

注14) https://github.com/k1LoW/emacs-drill-instructor/wiki

▼図4 ag.elの検索結果(下)と、検索結果から開いた該当箇所(上)



キーが使用できなくなり、使用すると警告メッセージが表示されるようになります(**図5**)。

実は筆者も長い間**C-i**と**C-m**を使うクセが身につかず、TAB、ENTERキーを直接叩いていました。そんな筆者ですが、鬼軍曹によるブートキャンプのおかげで、今ではEmacsを正しいキーバインドで使えるようになりました(**図6**)。Emacsのキーバインドを身につけたい人には、まさにうってつけの拡張と言えるでしょう。

「カーソルキーなんて地の果てにあるキーを使ってると、右手がホームポジションに戻ってくる前に戦争が終わっちまうぞ!」

「Sir! Yes, Sir!」

Emacsと日本語入力

Emacs上で動く日本語入力システム

私たちが普段利用している日本語はキーボードから直接入力することができないため、何らかの日本語入力システムを利用する必要があります。Ubuntuではインプットメソッドに IBus、日本語変換には Anthy[注15]が採用されており、デスクトップアプリケーションに日本語を入力する場合は、まず**C-SPC**か半角/全角キーでIBusを起動し、入力したかなを Anthy で漢字に変換していきます。これはEmacsも例外ではなく、Emacsのウィンドウや Emacsが起動しているターミナルエミュレータに、IBusで日本語を入力することが可能です。

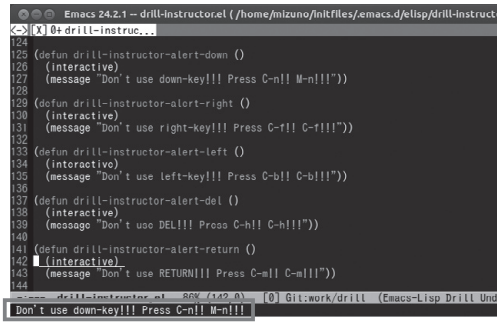
筆者はWindows、Mac、Linuxと、複数の環境を利用して作業を行っています。Emacsはマルチプラットフォームのアプリケーションですので、どのOSでも利用することができますが、日本語入力システムはそうはいきません。WindowsであればMS-IME、Macであればことえりというように、環境によって使える日本語入力システムには違いがあります。日本語入力システムの操作性の違いは作業効率に直結するため、環境が変わることによるコストは無視できません。そこで登場するのが、Emacs Lispで書かれた、Emacs上のモードとして動作する日本語入力システム「DDSKK[注16]」です。

Ubuntu Touchの例でも紹介しましたが、Emacs上で動作するDDSKKは、当然ながらOSの違いによる影響を受けません。Emacsさえ動けばどこでも、同じ操作で日本語入力ができます。これがDDSKKを使う最大のメリットです。

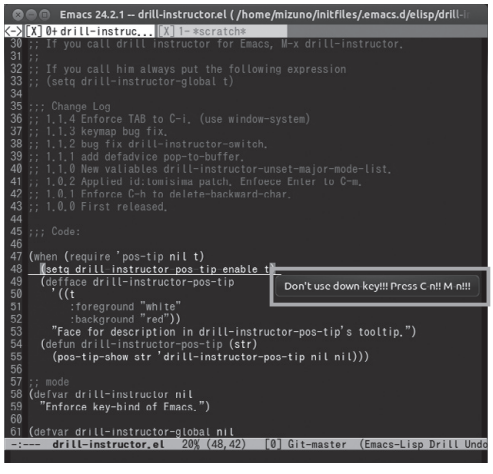
注15) 日本語RemixではMozcも使えるようになっています。

注16) もともとSKKと呼ばれる、Emacs上で動作する日本語入力システムが存在しました。DDSKKはその後継の実装となります。

▼図5 「カーソルキーを動かす前と後にサーと言え!」「Sir! C-n, Sir!」



▼図6 筆者の環境には、警告をポップアップするように改造した、pos-tip対応版がインストールされている



Emacsの拡張機能であるため、当然Emacsとの親和性も抜群です。他のインプットメソッドを使うと後述のコラムのように、Emacsのキーバインドをインプットメソッドが横取りしてしまうことがありますが、DDSKKならばそのような心配は無用です。

ほかにもよくあるケースとしては、日本語入力ができない端末を使って、ssh越しに日本語入力をしたい場合などでも、EmacsとDDSKKは役に立つでしょう。

UbuntuでDDSKKを利用するには、ddskkパッケージをインストールします。Emacsを起動したら、**C-x C-j**でskk-modeがオンになります。

SKKの特徴

一般的なインプットメソッドでは、入力された文章を自動で形態素解析し、品詞を特定し、部分ごとに変換を行います。しかしどんなに精度の高いプログラムであっても、「構文解析ミス」をゼロにすることはできません。そこでSKKは「漢字と送り仮名の区切りをユーザが手動で入力して指定する」というアプローチを取っています。これが他のインプットメソッドにはない、SKK最大の特徴です。

また入力中に、シームレスな辞書登録や登録削除が行えるのも便利な特徴です。辞書になくて変換できない単語に遭遇したときも、専用の辞書登録ツールなどを使用する必要はなく、変換中にその場で登録ができ、その瞬間から変換が可能になります。

誤解のないよう言っておくと、SKKはそれほど効率のよい日本語入力システムではありません。少なくとも速度に関して言えば、ATOKやMozcといった一般的な日本語入力システムには勝てないでしょう。しかしSKKの魅力は速度ではなく、プログラムによる解析ミスがそもそも存在しないため、変換時に文節区切りを指定しなおすような手戻りが存在しない（プログラムのミスのせいでイラっとすることがない）という点。そして繰り返しになりますが、Emacs上で動作するという点です。

▼リスト5　SKKサーバを利用する設定

```
(setq skk-server-host "localhost")
(setq skk-server-portnum 1178)
```

SKKサーバを使う

SKKは、システムで共有している書き換え不能な辞書と、ユーザごとに用意される個人辞書を併用して変換を行っています。この個人辞書には個人が登録した単語だけでなく、変換履歴も追記されていくのがポイントです。そのためSKKでは、使用頻度の高い単語ほど優先的に候補に挙げられ、結果として高いヒット率を誇ります。

DDSKKは辞書をバッファに読みこんで変換を行うため、辞書のサイズによってはEmacsのメモリの使用量が多くなることがあります。また使用する辞書を変更する際に、Emacsを再起動するというのも面倒です。この問題は辞書をフロントエンドで持たず、SKKサーバを利用することで解決できます。ここではSKKサーバとして、「yaskkserv」を利用する例を紹介します。

Ubuntuでyaskkservを利用するには、yaskkservパッケージをインストールします。また変換にSKKサーバを利用するため、.emacsに**リスト5**の内容を記述します。

yaskkservはインストール時に、システムにインストールされているSKK辞書を変換し、"/usr/share/yaskkserv"以下に取り込みます。しかしデフォルトの状態では、ある程度の人名地名や複合語を含んだ一般的な辞書である「SKK-JISYO.L.yaskkserv」のみが使用される設定になっています[注17]。他の辞書を同時に使用したい場合は[注18]、"/etc/default/yaskkserv"に使用する

注17）この際辞書ファイルが直接指定されるのではなく、alternativesを利用して、間接的に使用する辞書を指定するようになっています。

注18）yaskkservパッケージをインストールした際に、郵便番号、人名、法律用語、駅名などといった特殊な変換する辞書のコレクションである、skkdic-extraパッケージがインストールされています。

▼図7　複数の辞書を使用する例

```
$ sudo vi /etc/default/yaskkserv
PKG_DICS="/etc/alternatives/SKK-JISYO ¥
SKK-JISYO.zipcode ¥   ←コメントアウトを解除して郵便番号辞書を追加
SKK-JISYO.station ¥   ←コメントアウトを解除して駅名辞書を追加
"                     ←二重引用符を閉じる

$ sudo /etc/init.d/yaskkserv restart   ← yaskkservを再起動
```

▼リスト6　server completionの設定

```
(add-to-list 'skk-search-prog-list
          '(skk-server-completion-search) t)
(add-to-list 'skk-completion-prog-list
          '(skk-comp-by-server-completion) t)
```

▼リスト7　server completionの結果を個人辞書に追加しない

```
(add-hook 'skk-search-excluding-word-pattern-function
       #'(lambda (kakutei-word)
           (eq (aref skk-henkan-key (1-(length skk-henkan-key)))
               skk-server-completion-search-char)))
```

辞書を列挙します(**図7**)。

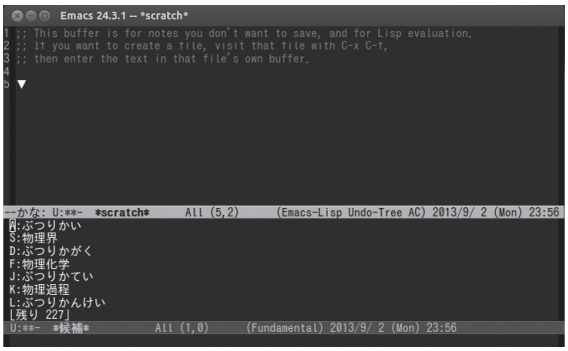
設定が完了したら、yaskkservを再起動しておきましょう。今後も使用する辞書を変更した際はyaskkservを再起動すればよく、Emacsの再起動は必要ありません。

server completionを使う

DDSKKからSKKサーバを使うメリットとして、yaskkservが対応しているserver completionを紹介します。これは辞書サーバを前方一致で全検索する機能です。この機能を利用するには、.emacsに**リスト6**の設定が必要です。この設定方法や機能の詳細は"/usr/share/emacs/site-lisp/ddskk/skk-server-completion.el"の先頭部分に記述されていますので、こちらも参照してください[注19]。

設定を完了したら、変換モードで読みの後ろにチルダ(~)をつけて変換を行ってみましょう(**図8**)。server completionが辞書サーバを前方一致検索し、マッチした語句を変換候補として列挙します。またserver completionの結果を個人辞書に追加したくない場合は、.emacsに**リスト7**の設定を追加します。

▼図8　「ぶつり~」と入力して変換してみた例



Emacsとパッケージ

ここまで、さまざまなEmacsの拡張機能を紹介してきました。そこで、Emacs拡張の入手と

注19) ただし筆者が試したところ、Ubuntu 12.10と13.04では動作しませんでした。おそらくEmacs 24.1/24.2に何らかの問題があるところまでは切り分けたのですが、原因の究明には至っていません。Ubuntu 12.04(Emacs 23.3)やUbuntu 13.10(Emacs 24.3)では動作することが確認できています。

Note GUI環境でIBusを無効にするには

Emacsでは、デフォルトで**C-SPC**がマークセットにバインドされています。しかしUbuntu 13.04のデスクトップでは、インプットメソッドであるIBusのオン／オフにもこのキーが割り当てられています。そのためデスクトップ上でEmacsを利用していると、**C-SPC**でIBusが起動してしまい、マークが設定できないという問題があります。

マークが設定できないのはEmacsとして致命的ですので、キーバインドを変更する必要があります。あるいはDDSKKを利用する人（＝IBusを使用しない人）であれば、Emacs上ではIBusの起動そのものを無効化してしまいたいところです。

▼リストA　IBusの無効化（.Xresourcesに記述）

```
Emacs24*useXIM:false
```

▼図A　.Xresourcesの設定を反映する

```
$ xrdb -merge ~/.Xresources
```

IBusの起動を無効にするにはいくつかの方法がありますが、ホームディレクトリの.Xresourcesに**リストA**の内容を記述しておくのが簡単です。その後はxrdbコマンドを実行して、設定した内容を反映します（**図A**）。あるいはXを再起動してしまっても構いません。

ここで気をつけるのは、「Emacs」ではなく「Emacs24」と記述する点です。これは、UbuntuではEmacsパッケージに独自のパッチが当たっており、WM_CLASSが「Emacs」から「Emacs24」に変更されているためです[注a]。

余談ですが、次期リリースであるUbuntu 13.10では、IBus 1.5が採用されています。IBus 1.5ではオン／オフの概念がなくなり、**C-SPC**を利用しなくなったため、この問題は発生しません。

注a）https://bugs.launchpad.net/ubuntu/+source/emacs23/+bug/949126
なお、Debianであれば「Emacs」と記述すれば問題ありません。

管理について、少し考えてみましょう。

Emacs拡張の配布方法は、作者によってさまざまです。個人のWebページやブログで公開している人もいれば、Emacs Wiki[注20]に置いている人もいます。最近ではGitHubが利用されるケースも多いようです。筆者個人の感想ですが、日本のEmacsユーザの間では、auto-install.elを使ってEmacs Wikiからインストールするというケースが、従来では多かったような気がします。

また最近のEmacsには、ELPAと呼ばれる標準のパッケージシステムが搭載されています。とはいっても、ELPAだけで必要なEmacs拡張がそろうとは限りません。前述のとおり、Emacs拡張の配布方法は作者次第ですので、究極的には「ユーザが何らかの方法でEmacs Lispのソースをダウンロードし、自分自身のホームディレクトリ内で管理する」しかありません。筆者も例外ではなく、いくつかの拡張は.emacs.d以下に抱え込んでいます。

この「環境をどうやって管理するか」というのは、Emacsに限らず、独自の世界を持っているシステムであれば必ずぶつかる悩みだと思います。たとえばプログラミング言語で言えば、PerlもRubyも、昨今のLinuxディストリビューションであればパッケージからインストールすることが可能です。ですがディストリビューションのパッケージメンテナンスの都合上、必ずしもPerlやRubyのプログラマが望むバージョンが、ディストリビューションから提供されるとは限りません。またPerlであればCPAN、Rubyであればgemの管理も必要です。そこでPerlbrewやrbenvといった管理システムを使い、OSからは独立して、ユーザが環境を丸ごと抱え込むのが最近のトレンドのようです。

Emacsの場合も、そういった手段を取るのが

注20）http://www.emacswiki.org/emacs/

現実的な解であるとは思います。ですが筆者は敢えて、ディストリビューションが提供するパッケージのみを使いたいと考えています。その理由は次のようなものです。

・拡張機能のアップデートを自分で管理しなくていい
・Ubuntuは半年ごとにバージョンが上がるのでそれなりに新しいパッケージを使える
・そもそもOS内に複数のパッケージ管理システムが混在するのが気に入らない
・自分自身がDebianのパッケージメンテナである

やはりUbuntuを使っているならば、apt-getコマンド1つで環境がそろえられるに越したことはありません。前述のように「パッケージのバージョンが古い」「そもそも自分の使いたい拡張のパッケージがない」ということもあるかと思いますが、それなら自分でパッケージを作って、解決してしまえばよいのです[注21]。実際筆者はそうしていますし、将来的には自分が使っているEmacs拡張をすべてパッケージ化し、apt-getのみで環境の構築ができるようにするのが目標です。

Debianのパッケージングのお作法については、本誌の連載「Debian Hot Topics」でも紹介されていますので、興味のある方はぜひチャレンジしてみてはいかがでしょうか。SD

注21) ここで言う「パッケージを自分で作る」とは、野良パッケージを作るという意味ではなく、Debianプロジェクトの公式リポジトリに入れ、メンテナンスする状態まで持っていくという意味です。

Chapter 5

Emacs環境100%全開テクニック

～高い拡張性を活かしていますか?～

Emacsをエディタとして使うだけでなく環境として見たとき、その豊富な機能にプログラマだったら、開発魂を駆り立てられるでしょう。本章では、elispを利用しTRAMPでssh接続してみたり、org-modeで出力をHTMLやTexにしてみたり、環境をフルに使うテクニックを紹介します。そして標準で含まれているWebブラウザ「eww」を通し、Emacsの未来を考察します。

Writer 濱野 聖人
(はまの きよと)

●Emacsの設定ファイル行数：3,313行(.emacs.d/init.elの行数)
●㈱創夢。EmacsとPCキーボードをこよなく愛するDebian GNU/Linux使い。
Twitter@khiker
Mail：khiker.mail@gmail.com

はじめに

Emacsは環境であるとよく言われます。この言葉はあながち間違いではありません。EmacsはEmacs Lisp(以降elisp)というプログラミング言語のインタプリタが本体で、その機能のほとんどはelispによって実現されています。Emacsが高い拡張性を誇る理由もelispで好きなプログラムを書くことができるためです。

Emacsではどんな便利なelispを追加するか、書くかということに注目しがちですが、すでにEmacs標準に便利なelispや機能が多数含まれています。たとえば、emacsclient、デーモン機能、TRAMP、org-mode、package.el、gnus、calcなどなど挙げたらキリがありません。今回はその中でもemacsclient、デーモン機能、TRAMP、org-modeを紹介します。

emacsclient

皆さんは、普段テキストエディタをどのように使っているでしょうか。ファイルごとに複数、エディタを起動していますか？　Emacsにはその配布物にemacsclientという別個のプログラムが含まれています。Emacsでは、このコマンドを利用することですでに起動しているEmacsに接続して、そのEmacsで新しくファイルを開き、編集できます。

起動済みのEmacsに接続できると次の利点があります。

・高速にファイルを開ける
・起動済みのEmacsのリソースを使いまわせる

Emacsはその設定の数が多くなるとその起動がどんどん遅くなります[注1]。そこでemacsclientを使うと、新規にEmacsを起動せず、既存のEmacsに接続するだけなので、高速にファイルを開けます。また、emacsclientでは、既存のEmacsに接続するので、「保存していない編集途中のバッファ」にアクセスすることもできます。

利用方法

emacsclientを利用するには、次の設定を.emacsに記載してください。

```
(require 'server)
(unless (server-running-p)
  (server-start))
```

そのうえで起動済みのEmacsがある状態で、ターミナルで次のようにemacsclientを実行すると起動済みのEmacsでファイルを開けます。emacsclient 開いたファイルを閉じる場合は、Emacs上で**C-x #**と入力します。

注1) **M-x emacs-init-time**とするとEmacsの起動にかかった時間がみられます。

```
$ emacsclient ファイル名
```

この場合、EmacsをGUIで起動していたならば、そのGUIのEmacs の画面で指定したファイルを開きます。

ターミナル上で開く

EmacsはGUIだけでなくターミナル上でも使えます。当然emacsclientもGUIで起動したEmacsでファイルを開くのではなく、ターミナル上で開くこともできます。次のようにEmacsをGUIなしで起動する場合と同様に**-nw**オプションを付けてemacsclientを実行することで、ターミナルでファイルを開くことができます。この方法で起動したemacsclientから抜ける場合、**C-x #**だけでなく、通常のEmacsを終了するコマンドである**C-x C-c**も使えます。

```
$ emacsclient -nw ファイル名
```

これはEmacsを1つGUIで起動している状態で、ターミナル上で作業をしているときに便利です。たとえば、何かの設定ファイルを一時的に編集する必要があるとき、**emacsclient -nw**を使えば、ターミナルから別ウィンドウのEmacsに移ることなく作業を継続できます。

デーモン機能

デーモン機能は、その名のとおり、Emacsをデーモン化します。これはEmacs 23で追加された機能で、デーモン化することで、Emacsをシステムに常駐させることができます。デーモン化したEmacsにアクセスするためには、emacsclientを使います。そのため、毎度、システムにログインするたびにEmacsを起動する必要はなくemacsclientで高速にファイルを開くことができます。

Emacsをデーモン化する利点は、Emacsをシステムに常駐させておけることです。たとえば、sshなどで出先からシステムに入ったときでもEmacsに接続できます。この場合、emacsclientの項で挙げた例と同様に、Emacsに未保存の編集途中のバッファがあったとしても、当然、そのバッファを開くことができます。

利用方法

Emacsをデーモンとして起動するには以下のようにdaemonオプションを付けてEmacsを起動します。実行するといくつかメッセージが出たあと、デーモン化したEmacsが起動し、プロンプトが返ってきます。

```
$ emacs --daemon

...
("emacs")
Starting Emacs daemon.
$
```

デーモン化したEmacsにアクセスするには次のように先述のemacsclientを使います。

```
$ emacsclient -nw ファイル # terminal で起動
$ emacsclient -c  ファイル # GUI で起動
```

デーモン化したEmacsを終了する場合大きく2つ方法があります。1つはターミナルから終了する方法です。Linuxであれば次のようにEmacsのプロセスに対してTERMシグナルを送ります。

```
$ pkill -TERM emacs
```

Emacsのコマンドを実行して終了する方法もあります。emacsclientでデーモン化したEmacsに接続して**M-x save-buffers-kill-emacs**とするか、単に**M-x kill-emacs**とします。前者は、**C-x C-c**と同じ意味で、保存していないバッファを保存するか確認されます。後者はその確認をしません。

なお、デーモン機能は現在のところWindowsバージョンのEmacsではサポートされていません[注2]。

注2) Cygwinバージョンであれば使えるようです。

TRAMP

デーモン機能ではリモートのマシン上でEmacsを起動しておき、外部からそのマシンにアクセスして使うというユースケースを示しました。別のケースで、ローカルにEmacsはありますが、リモートにEmacsはなく、しかし、リモートのファイルもEmacsで編集したいということも考えられます。この場合、Linuxであれば、たとえばsshfsを使って手元のファイルシステムにリモートのマシンをマウントするという方法でも解決できますが、EmacsにはTRAMPというelispが含まれており、もっと簡単に解決できます。TRAMPを使うと、リモートにあるファイルがあたかもローカルにあるかのように編集できます。

ssh越しの利用方法

TRAMPは通常のファイルを開くコマンド**C-x C-f**において、TRAMP用の入力をすればすぐさま使えます。たとえば、リモートホストremoteのファイル"/path/to/file"を編集するには、次のようにします。

```
C-x C-f /ssh:user@remote:/path/to/file
```

この場合、TRAMPはsshコマンドのラッパとして動作するので、当然、sshでリモートホストremoteにログインできる必要があります[注3]。

なお、先頭のメソッド**ssh:**は省略できます。省略すると変数**tramp-default-method**に登録されているメソッドが使われます。Emacs 24.3標準のTRAMPでは**scpc**となっています。このメソッドはメソッドsshとほぼ同義で、sshコマンドに**-o ControlMaster=yes**を付与したものとなります。**tramp-default-method**の値を**ssh**とするには次のように.emacsに記載します。

```
(setq tramp-default-method "ssh")
```

また、TRAMPはリモートにあるファイルがあたかもローカルにあるように編集できるだけではありません。Emacsの**dired**[注4]を使って、リモートのファイルをローカルにコピーしたり、**M-x grep**コマンドを使って、リモートのファイルを**grep**したりもできます。

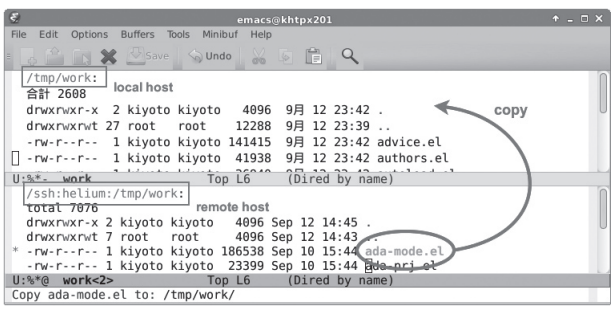
diredは通常のdiredと同じように使います。リモートホストのディレクトリをdiredで開き、**m**コマンドでコピーしたいファイルを選択し、**C**コマンドを実行してコピーする際、コピー先をローカルホストの適当なディレクトリとすればコピーがなされます。実行例は、**図1**のようになります。

grepも通常の**M-x grep**コマンドと同じように使います。この場合、リモートホスト上で**grep**コマンドが実行されます。実行例は**図2**のようになります。

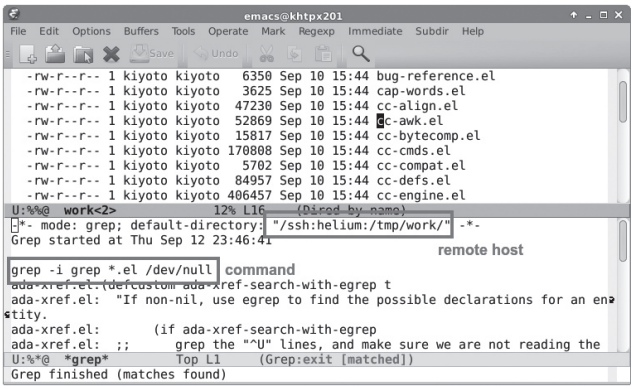
注3） そのため、実行するとパスワード(パスフレーズ)の入力が促されます。

注4） diredとはEmacsの標準搭載のファイラーです。

▼図1　diredでリモートのファイルをコピー



▼図2　リモートマシン上でM-x grepの実行



多段SSH

sshメソッドは、sshコマンドのラッパであるため、“~/.ssh/config”に記載した内容がそのまま使えます。たとえば、リモートホストgatewayを通さなければログインできないホストremoteに一度でログインするために、次のような多段sshをするための設定が“~/.ssh/config”に記載があったとします。その場合、**C-x C-f /ssh:remote:~/foo.txt**のような入力がEmacsでも使えます。

```
Host gateway
  HostName gateway
  User     bar

Host remote
  HostName remote
  User     foo
  ProxyCommand ssh gateway exec ↵
nc %h %p
```

もちろん、“~/.ssh/config”に書かずともTRAMPだけでも多段接続ができます。multi-hopと呼ばれる機能です。これには変数**tramp-default-proxies-alist**に設定をします。上記と同じようにホストgatewayを踏み台にしなければログインできないホストremoteにログインしたい場合、.emacsに次のように記述します[注5]。

```
(require 'tramp)
(add-to-list 'tramp-default-proxies-↵
alist '("remote" nil "/ssh:gateway:"))
```

この記述のうち、**("remote" nil "/ssh:gateway:")**は、左から順番に「ホスト」、「ユーザ」、「ホスト名に接続するための接続方法」を示します[注6]。これであとは、**C-x C-f /ssh:remote:~/foo.txt**とコマンドを実行することで、ホストremote上のファイルfoo.txtにアクセスできます。

ただ、多段sshの設定だけであれば、“~/.ssh/config”に記述することをお勧めします。“~/.ssh/config”に記載するとTRAMPからだけでなく、通常のsshコマンドでも使えるため、より便利です。

suとsudo

TRAMPはsshのためだけのものではありません。sudoやsuコマンドのラッパとしても使えます。

これは、“/etc”以下のファイルを編集しなければならないが、「rootユーザのEmacsは何も設定していなくて使いづらい」、「Vimやnanoの

注5）**("remote" nil "/ssh:gateway:")**の**gateway**のあとに必ず**:**が必要なことに注意してください。

注6）詳しい記法については、TRAMPのInfoを参照してください。

使い方がよくわからない」といった場合で力を発揮します。Emacsから抜けることなくそのまま“/etc”以下のファイルをrootユーザとして編集できます。

suメソッドを使う場合は次のように入力します。

```
C-x C-f /su::/etc/gitweb.conf
```

sudoメソッドの場合は次のように入力します。次のuser@localhostは省略できます。省略した場合、userはrootが指定されたものとみなされます。

```
C-x C-f /sudo:user@localhost:/etc/➥
gitweb.conf
```

両者ともに実行すると、ターミナルからsuやsudoコマンドを実行した場合と同様にパスワードを聞かれます。suならば、rootユーザのパスワード、sudoならば、自分のパスワードです。

sshとsudoを組み合わせる

sudoメソッドは、前述のmulti-hop機能を使うとリモートにsshでログインして、かつ、リモート上でrootユーザとなりファイルを編集するといったことができます。この場合、次のように設定を記載します。

```
(add-to-list 'tramp-default-proxies-➥
alist '("remote.alias" nil "/ssh:user➥
@remote:"))
```

あとは、次のように入力するとリモートホストremoteのファイルをrootユーザで編集できます。

```
C-x C-f /sudo:root@remote.alias:~/
```

また、rootユーザではなく、別のユーザfooで作業したいならば、以下のように入力すれば良いです。

```
C-x C-f /sudo:foo@remote.alias:~/
```

これらは、リモートホストremoteにsshでログインしたあとに「sudo -u ユーザ」を付与してシェルを実行していると考えれば良いです。

その他の機能

TRAMPのメソッドは、このほかにもftpやsftp、smbなどに対応しています。ほかにも機能や設定が多数あります。詳しくは、TRAMPのInfoを参照してください。

org-mode

org-modeは、プレーンテキストに記載した文章からHTMLやLaTeXなどのドキュメントを生成できるメジャーモードです。ドキュメント生成だけでなく、TODOリストを管理したり、スケジュールを管理したりもできます。

プレーンテキストからドキュメントを生成する同様のツールとしては、Emacsではありませんが、reStructuredText[注7]からドキュメントを生成するsphinxが有名です。sphinxと比較してorg-modeには次の利点があります。

- Emacsさえあれば使える
- Emacsに組み込まれているため、エディタ支援が充実している

org-modeを使いはじめるのは非常に簡単で、Emacs上で拡張子orgのファイルを作成するだけで良いです。そうすると、Emacsはファイルをorg-modeで開きますので、すぐにはじめることができます。

記法

org-modeは、Emacsのoutline-modeの記法を拡張したような独自の記法を持ちます。**表1**に筆者が比較的よく使う記法を示します。

見出しは、M-RETキーを押すと、現在のレベルと同じレベルの見出しを挿入できます。また、リストや番号付きリストの中にリストや番号付きリスト自身を書く場合は、ただインデントをすれば良いです。記載例を次に示します。

注7） EmacsでreStructuredTextの編集にはrst-modeがあり、標準に含まれています。

▼表1　org-mode の記法(一部)

大見出し	行頭に*1つと見出し名
中見出し	行頭に*2つと見出し名
小見出し	行頭に*3つと見出し名
リスト	-から行をはじめる
番号付きリスト	1.のような数字で行をはじめる

```
* 大見出し
ここは大見出しの中です。
1. 番号つきリスト
   - 番号つきリストの中にリスト
2. 番号つきリスト
   1. 番号つきリストの番号つきリスト
- リスト1
  - リストのリスト
- リスト2
** 中見出し
ここは中見出しの中です。
```

ソースコードなど文をそのままを載せたい場合は、#+begin_exampleの行と#+end_exampleの行の間に文を書きます。これにはショートカットがあり、「<e」と入力した後にTabキーを押せば、「<e」が#+begin_exampleと#+end_exampleに置き換わります。

表は次のように書きます。表を記載する場合もショートカットキーがあります。「|」キーを入力した後に表の項目を記載し、ただTabキーを押すだけで良いです。 それだけで表が作れます。表の新しい列を作成する場合もただ、Tabキーを押せば作れます。表の枠線を書く場合も「|-」と入力した後にTabキーを押すだけで良いです。

```
|------+------|
| 左上 | 右上 |
| 左下 | 右下 |
|------+------|
```

その他にも_で下付き文字、^で上付き文字を書けます。ただ、これは通常、日本語の文章を書く際はあまり使わないと思います。その場合はorgファイルの頭に次の行を記載しておけば

▼表2　org-mode のショートカット(一部)

C-c C-f	次の同じレベルの見出しに移動する
C-c C-b	前の同じレベルの見出しに移動する
C-c C-p	次の見出しに移動する
C-c C-n	前の見出しに移動する
Tab	見出し上の「*」上で押すとその見出しの内容を表示・非表示をする

この記法を無効化できます。

```
#+OPTIONS: _:{}, ^:{}
```

ショートカット

org-mode には編集する際に利用すると便利なショートカットが数多くあります。その一例を表2に示します。

このほかにもさまざまなショートカットがあります。詳しくはInfoを参照してください。

エキスポート

org-modeはさまざまな形式に出力することができます。TXT、HTML、LaTeX、ODTなどなどです。ここでは筆者が比較的よく使うHTMLとLaTeXについて、出力方法と設定例の一部を紹介します。

■HTMLへ出力

HTMLに出力するためには、C-c C-e hと入力します。実行すると拡張子がhtmlのファイルが生成されます。

HTMLへ出力する場合、HTMLを綺麗に見せるためにCSSファイルを1つ作成しておくと良いです。出力するHTMLにCSSファイルの読み込み部を埋め込むには、orgファイルの先頭に次を記載しておきます。

```
#+STYLE: <link rel="stylesheet" ➲
type="text/css" href="org-style.css" />
```

CSSのサンプルは、検索すれば数は少ないですがいくつか見つかります。

▼リスト1　Latexへ出力するための.emacsサンプル

```
(require 'org-latex)
(add-to-list 'org-export-latex-classes
             '("jsarticle"
               "¥¥documentclass[a4paper,10pt]{jsarticle}
¥¥usepackage[dvipdfm]{graphicx,color}
¥¥usepackage{hyperref}
 [NO-DEFAULT-PACKAGES]"
               ("¥¥section{%s}"       . "¥¥section{%s}")
               ("¥¥subsection{%s}"    . "¥¥subsection{%s}")
               ("¥¥subsubsection{%s}" . "¥¥subsubsection{%s}")
               ("¥¥paragraph{%s}"     . "¥¥paragraph{%s}")
               ("¥¥subparagraph{%s}"  . "¥¥subparagraph{%s}")))
```

■LaTeXへ出力

LaTeXに出力するためには、C-c C-e lと入力します。実行すると拡張子がtexのファイルが生成されます。

ただし、日本語のLaTeXに出力する場合、多少設定が必要です。何も設定しない状態でorg-modeでLaTeXを生成すると、生成されたtexファイルではarticleクラスが使われます。日本語を扱うのであれば、通常、jsarticleクラスを使いたいと思います。そのためには、**リスト1**の設定を.emacsに追記します。

orgファイルの先頭には次を記載しておきます。

```
#+LATEX_CLASS: jsarticle
```

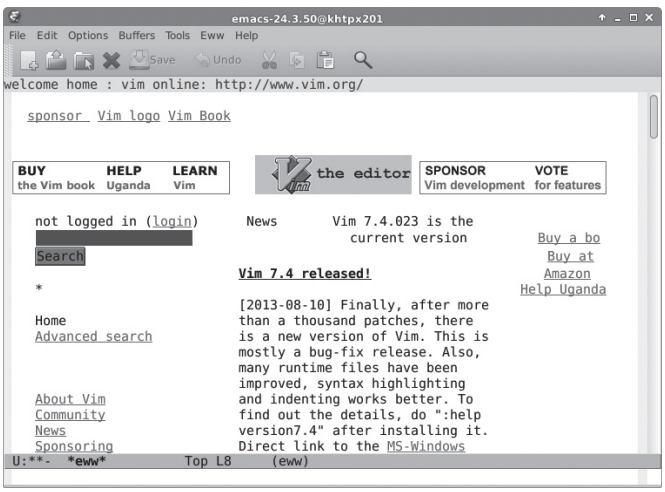
あとは、生成されたtexファイルを**platex**コマンドで処理すればPDFファイルが生成できます。

org-modeのまとめ

設定や記法の一部を駆け足で紹介しました。このほかにもorg-modeはHTMLやLaTeXに出力する際に使用できる記法などさまざまな機能・設定があります。詳しくは、Infoを参照してください。

また、org-modeは、本家で活発に開発されています。現在Emacs開発版に含まれているorg-modeのバージョンは、7.9.3fですが、本家ではすでに8.1がリリースされています。バージョン8系列では、一部記法やnamespaceの変更など大きな改修がなされており、上記に挙げた設定が使えなくなっています。8系列を使用する際は注意してください。

▼図3　ewwで敵状視察(?)



Emacsのこれから

Emacsは現在でも活発に開発がなされています。次のリリースに向けての開発が行われており、いくつもの新機能やバグ修正が行われています。新機能は、ACLのサポートや、Linuxであればinotifyを利用したfile変更の監視、Webブラウザの追加などがあります。その中でも今回はWebブラウザ「eww」について紹介します。

なお、Emacs はそのリポジトリを bzr で管理しており、以下のコマンドで開発版ソースコードを入手できます。 ビルド方法については、リポジトリ内の INSTALL ファイルを参照してください。

```
$ bzr branch http://bzr.savannah.gnu.org/r/emacs/trunk/
```

Emacs Lispで実装されたWebブラウザeww

ewwはEmacs Lispで実装されたブラウザです。Webブラウザといっても皆さんが普段使っているであろうGoogle ChromeやFirefoxのような高機能なものではありません。画像こそ表示できますが、JavaScriptもCSSも解釈できません(**図3**)。

Emacs上でWebブラウジングするためのelispではemacs-w3mが非常に有名ですが、eww には次の利点があります。

・Emacs標準に含まれている
・外部プログラムに依存していない

Emacs標準に含まれていること、これは非常に大きいです。ewwは機能面ではemacs-w3mに劣りますが、ユーザがとくにインストールせずとも、何も設定せずとも、Emacsを起動して**M-x eww**とするだけで使えます。非常に簡単です。

emacs-w3mはテキストブラウザである外部プログラムのw3mに依存していますが、ewwは依存していません。ewwはEmacsがlibxml2サポートが有効な状態でビルドされてさえあれば良いです。

通常、elispの処理速度は遅く、外部プログラムのw3mを使わないewwはページのレンダリングが遅いのではないかという疑念が湧きますが、そうではありません。elispの関数の中にもCライブラリのラッパとなっている関数がいくつか存在し、それらはelispだけで実装された関数よりも高速に動作します。**libxml-parse-html-region**もその1つです。これは前述のCライブラリlibxml2のラッパーとなっており、この関数がレンダリングの要であるHTMLを解釈するという部分を担っているため、十分に高速に動作します。Google検索のようなちょっとしたWebブラウジング程度ではストレスを感じることはありません。

ewwは、まだまだ開発版ということもあり、不安定な部分もありますが、標準でWebブラウザが搭載されたことにより、ドキュメントの作成中にちょっとGoogleで調べ物[注8]といったありがちな内容がEmacsの中でシームレスに行えるようになります。また、ブラウザであるということは、HTMLをレンダリングできるということなので、HTMLで書かれたマニュアルの参照や、sshでログインした先でHTMLで書かれたドキュメントの確認などで大きく力を発揮することになると予想されます。

おわりに

Emacs標準に含まれている機能のうちの筆者がよく使う極一部の機能を紹介しました。EmacsはInfoというマニュアルが標準で含まれており、ドキュメントが充実したエディタであるとも言えます。今回紹介したTRAMPやorg-modeのような大きめの機能は、それだけでInfoのマニュアルが存在します。時間のあるときにでもInfoを眺めてみると新しい発見があるかもしれません。SD

注8) なお、標準の検索エンジンは現在のところDuckDuckGo(https://duckduckgo.com/)です。

Column 2 もしEmacsがなかったら？ その❶

Writer まつもとゆきひろ　Emacs設定行数：718（外出し分を含めると1,230行）

■Emacsとの出会い

本特集のコラムとしてEmacsを紹介する必要はないでしょう。Emacsとの出会いは私の人生を大きく変えました。

大学在学中、UNIXを使うようになった私はUNIX上で使えるスクリーンエディタとしてEmacsを知りました。それまで使ったことのあるMS-DOSのエディタとは使い勝手が異なる「変わったエディタ」だと思いました。その時点では、それほど優れたエディタとは思わなかったというのが本音です。しかし、Emacs Lispと呼ばれるLisp方言によってさまざまなカスタマイズを行えるという点を知り、カスタマイズの幅の広いプログラマ好みのエディタだな、と感じました。

■禁じられたEmacs

しかし、当時、学部生であった私たちはEmacsを使うことが禁じられていました。当時私たちは1台のSun 3ワークステーションを1年から3年まで200人以上で共有していたため、当時は貴重なメモリを大量に消費するEmacsを学部生が使うことはご法度だったのです。禁止されると知りたくなるのが人間の性です。Emacsを使うことは禁じられていましたが(勝手に使うと管理者にプロセスをキルされました)、フリーソフトウェアをダウンロードすることは別に禁止されていませんでした。そこで、物好きな私はEmacsのソースコードをダウンロードして、中身を調べ始めたのです。

■Emacsは何でできているのか？

Emacsを調べているうちに驚くべきことに気がつきました。エディタとしてのEmacsは、その基本的機能(画面表示や文字の挿入・削除)だけはCで実装されていますが、それ以外の部分のほとんどはEmacs Lispで実装されています。各種言語をサポートする機能や、どのキーがどの働きをするかを決める部分などユーザから見てエディタの本質だと感じる部分までEmacs Lispによって実現されていました。つまり、Emacsは、Cによって実装されたEmacs Lisp処理系と、そのアプリケーションとしてのエディタという構成になっていたのです。

Emacsのソースコードを研究することで、私はプログラミング言語の実装技術の多くを学びました。たとえば、ポインタにタグ付けをして整数に埋め込む技法、マーク・アンド・スイープ技法によるガベージコレクションの実装、プログラミング言語処理系とユーザ定義C関数とのインターフェースなどなど。それらの技術はすべて後のRubyの実装に活用されました。

■Emacsが与えてくれた自由

その後、私は大学4年生になり研究室に配属されたことで、Emacs禁止令から解放されました。とうとうEmacsユーザになったのです。Emacsは完全に私好みのエディタというわけではありませんでしたが、気に入らなければなんでも変更できるのです。Emacsは私に「プログラマは自由である」ことを教えてくれました。プログラミングができない人は、既存のソフトウェアを与えられるままに使うしかありませんが、プログラマは自分の思うままにソフトウェアを改造し、自分のアイデアを自由に試すことができるのです。これは私にとって大きな発想の転換でした。

■プログラミング言語Rubyの誕生

大学を卒業し、就職してしばらくたった1993年、ふとしたことからプログラミング言語を開発することになりました。学生時代にEiffelという言語にかぶれていた私は、endキーワードでブロックを区切る文法の言語が良いのではないかと考えました。しかし、私はEmacsユーザです。快適な

プログラミングのためには、Emacsの言語モードがどうしても必要です。オートインデントのないような言語でどうしてプログラムが書けるでしょうか。しかし、当時endでブロックが終わるような言語のためのモードで、オートインデントをサポートしているものはありませんでした。

そこで、私は数日間、Emacs Lispと格闘し、endがある言語でもオートインデントができるようなRubyモードのプロトタイプを完成させました。これによって、Rubyを現在の文法にする決心ができたのです。もし仮にこの時点でRubyモードの開発に成功していなかったら、RubyはCやJavaのようなブレースを使った「当たり前」の文法になっていたかもしれません。

■もしEmacsがなかったら？

さて、「もしEmacsがなかったら……」ですよね。もし、Emacsがなかったら、Rubyは今の文法ではなかったでしょう。なにより、私がRubyを実装するのに必要な技能を身につけることもなかったでしょうし、プログラマの持つ自由にも、そのパワーにも気がつかず、今のような熱意を持ってプログラミングできていたのかさえ疑問です。

さらに言えば、Emacsがなければ、ストールマンはFSFを立ち上げることもなかったでしょうし、世界はフリーソフトウェアの闘士を失い、続くオープンソースムーブメントもなかったことでしょう。

つまり、Emacsがなかったら、世界は今のようではなかったはずなのです。Emacsがあって良かった！ SD

Chapter 6

Web開発の舞台裏とEmacs

～ELPA、quickrun、Web-mode、Flymake、Gitの決め技～

本章では、Emacsの本領とも言える開発について、最初に最新のEmacsにおける拡張の導入方法を解説したあと、便利な拡張を紹介しながら説明していきます。本当はもっと多くの事例を紹介したいのですが、誌面が限られていますので、まだWeb上に解説が少ない新しめの話を中心にしていきます。

Writer 大竹 智也（おおたけ ともや）

●Emacsの設定ファイル行数：1,864行
●1983年生まれ。2003年にインターネットに触れ、ブログを通じWeb技術を吸収、2010年にフィリピンにてオンライン英会話「ラングリッチ」を設立。代表取締役。

ELPAによる拡張インストール

ELPA（エルパと読みます）とはEmacs Lisp Package Archiveの略で、Emacs24から導入されたEmacs Lispパッケージマネージャです。このELPAはRed Hat LinuxのyumやMacのhomebrew、またはPerlのCPANなどのように、拡張機能のインストールや削除などを行ってくれるたいへん便利なツールです。

これまでパッケージマネージャがなかったため、人力による拡張の導入や管理が必要でしたが、本体機能として提供されるようになったため、とても便利になりました。

本章の動作確認環境

本章で解説している動作確認は、次のものになります。ターミナル環境についてもMacのターミナルを用います。

- Mac OS X Lion（Xcodeインストール済み）のEmacs.appの24.3

なお、次の環境でも動作確認をしています。

- Windows XP Emacs 24.3（オフィシャルビルドとNTEmacs）
- Windows 7 Emacs 24.3（オフィシャルビルドとNTEmacs）

ELPAの導入

ELPAは本体機能のため、インストールは必要ありません。ですが、拡張をダウンロードするリポジトリを追加する必要がありますので、その設定を解説します。

設定にはすでにChapter3で解説があったかと思いますが、**~/.emacs.d/init.el**に設定を記述します。init.elのできるだけ前のほうに**リスト1**の設定を追加して保存します。基本的には

▼リスト1　ELPAの設定例

```
;; ELPAの設定
(when (require 'package nil t)
  ;; パッケージリポジトリにMELPAと開発者運営のELPAを追加
  (add-to-list 'package-archives
               '("melpa" . "http://melpa.milkbox.net/packages/"))
  (add-to-list 'package-archives
               '("ELPA" . "http://tromey.com/elpa/"))
  ;; インストールしたパッケージにロードパスを通してロードする
  (package-initialize))
```

これだけでELPAを使う準備が完了します。

ELPAの使い方

それではさっそくELPAを使ってみましょう。まずはパッケージ一覧を取得する次のコマンドを実行します。

```
M-x list-packages
```

少し起動に時間がかかるかもしれませんが、ELPAに登録されているパッケージ一覧が表示されます(**図1**)。この画面はpackage-menu-modeというメジャーモードになっており、操作は**表1**のとおりです。

ELPAによる拡張のインストール

それでは、ELPAに登録されているパッケージを実際にインストールしてみましょう。

たとえば、Emacsのバッファを表示どおりにHTML化するhtmlizeという拡張機能をインストールしたい場合、htmlizeの行で**i**を押します。するとhtmlizeがインストール候補としてマークされます(**図2**)。そして**x**を押すと、ミニバッファに実行確認の**yes/no**が表示されるので、**yes RET**と答えるとパッケージのインストールが開始します。

インストール先は、標準の設定では**~/.emacs.d/elpa**以下となっています。パッケージのダウンロード→バイトコンパイルの順番で進行していき、バイトコンパイルが開始される際、Emacs上で開いているすべての未保存ファイルを保存するか質問されます。保存して先に進むときは**y**を入力して**Enter**を押しましょう。バイトコンパイルが終了するとインストール完了です。

なお、インストールされた拡張機能は、すぐに利用できるようにはなっていません。利用するためには、ELPAからインストールした拡張機能を読み込むためのコマンドを実行します。そのコマンドが、**リスト1**の設定の最後に記述したpackage-initializeなのです。**init.el**への記述はEmacs起動時に一度実行されるだけですので、Emacsの再起動をすることなく拡張機能を読み込むには、次のコマンドを実行しましょう。

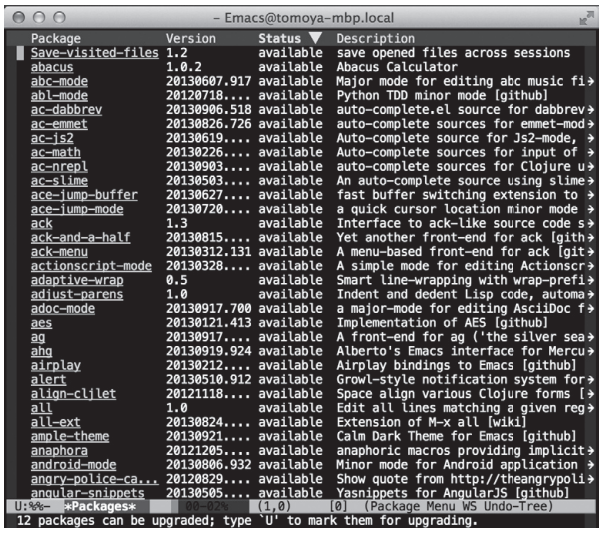
```
M-x package-initialize
```

すると、先ほどインストールしたhtmlizeの機

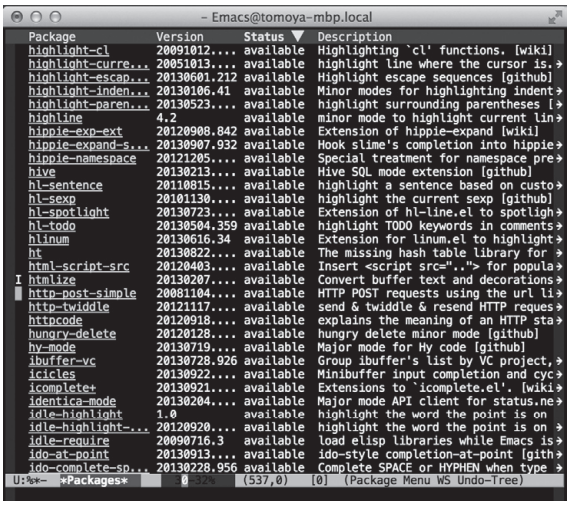
▼表1　ELPAの操作一覧

キー	説明
h	ミニバッファに操作ヘルプを表示する
p	前の行へ
n	次の行へ
?、RET	パッケージの説明を取得する
i	インストール候補としてマークする
U	アップデート可能なパッケージをすべてマークする
d	削除候補としてマークする
DEL	1行上のマークを外す
u	現在行のマークを外す
x	マークしたパッケージをイントール／削除
r	パッケージ一覧をリフレッシュ
q	ウィンドウを閉じる

▼図1　M-x list-packages



▼図2　htmlizeをマーク（x→yesでインストール可能）



▼表2　quickrunが提供するコマンド一覧

コマンド	説明
quickrun	現在のバッファを実行する
quickrun-region	選択中のリージョンだけ実行する
quickrun-shell	シェルから対話的に実行する
quickrun-compile-only	コンパイルのみ実行する
quickrun-replace-region	リージョンを実行結果で置換する
quickrun-with-arg	ミニバッファから引数をとって実行する

能を利用できるようになります[注1]。

ELPAを利用してインストールした拡張機能には初期設定ファイルが同梱されており、基本的には**init.el**に設定を記述することなく利用できるようになっています（ただし、すべてではありません）。

ELPAによる拡張の削除

ELPAからインストールした拡張を削除する方法はインストールとほぼ同じになっています。パッケージ一覧の画面で削除したいパッケージの行で**d**を押して削除候補としてマークしたあと、**x**を押して実行確認でyesと答えるだけです。

ただし、インストールと異なる点としてはパッケージを削除しても、あなたのEmacsにすでに読み込まれた機能はそのままとなっているため、パッケージを削除した場合は再起動することをお勧めします。

それでは、ELPAの使い方を覚えた次は、本章のメインとなる拡張を使った開発についての解説に入っていきます。

注1） **M-x htmlize- TAB**とタイプすると、htmlizeの提供するコマンドが一覧できます

quickrunによる開発

Emacs上でプログラムコードを書いているとき、現在のコードを実行してみたいと思うことはないでしょうか？　そんなときに便利なのがsyohex氏[注2]が開発したquickrunという拡張です。

quickrunはEmacsのバッファをさまざまなプログラムで実行できる拡張でGitHub上で公開されています[注3]。

quickrunの導入

quickrunはMELPAリポジトリを追加済みのELPAからインストールできます。

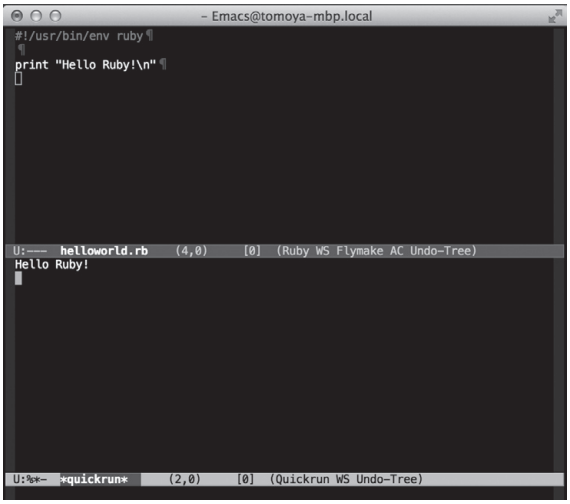
また、ELPAにはパッケージ一覧を表示しなくても、個別にインストールする方法がありますので、次のコマンドを実行しましょう。

```
M-x package-install RET quickrun RET
```

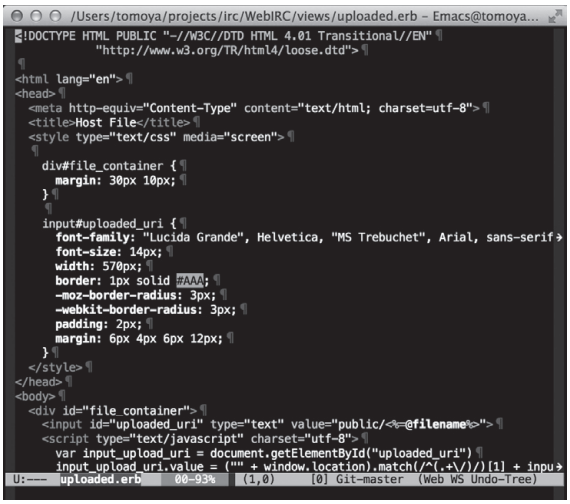
インストールが完了したら、すぐにquickrunが提供するコマンドが使えるようになります。

注2） http://d.hatena.ne.jp/syohex/
注3） https://github.com/syohex/emacs-quickrun

▼図3　quickrunの実行
（上：編集中のコード／下：実行結果）



▼図4　Web-modeの画面
（HTML＋CSS＋JavaScript+Ruby）



quickrunが提供する機能

quickrunが提供する機能はおもに**表2**になります。実際によく利用するコマンドは、quickrunとquickrun-regionになるでしょう。

quickrunの使い方

quickrunの使い方はとても簡単です。プログラムを書いて実行したくなったときに、**M-x quickrun**と実行するだけです。

quickrunコマンドを実行すると、現在のメジャーモードからコンパイラやインタプリタを自動的に判別し、カレントバッファに書かれているコードの実行結果が*quickrun*バッファに表示されます(**図3**)。

quickrunを利用すると、これまでわざわざターミナルからプログラムを実行していた操作が、Emacs内で行えるようになり、より開発に集中することが可能となりますので、まだ導入されていない方はぜひ導入しましょう。

Web-modeによる開発

Web開発者はHTMLやCSSやJavaScript、そしてPHPやPerl、Rubyなど、さまざまな言語を扱います。Emacsにはいろいろなメジャーモードがあるため、これらすべてをEmacs上で開発できますが、HTMLを中心として、複数の言語が混在するテンプレートファイルを編集する場合、逆にメジャーモードが足枷となる場合があります。こういったケースでも便利に編集する方法はないのでしょうか？ その1つの答えが最近登場したWeb-mode[注4]です(**図4**)。

Web-modeはその名のとおり、HTMLを中心としたテンプレートファイルをストレスなく開発するWeb開発に特化したメジャーモードです。

Web-modeの導入

Web-modeはMELPAリポジトリを追加済みのELPAからインストールできますので、次のコマンドを実行しましょう。

```
M-x package-install RET Web-mode RET
```

インストールが完了したら、**M-x Web-mode**でメジャーモードを切り替えることができるようになります。Web-modeを選択するとモード

注4）http://Web-mode.org/

ラインにWebと表示されます。

Web-modeが提供する機能

複数の言語が混在するファイルを編集するメジャーモードというと、昔からEmacsを利用されている方は、MuMaMo[注5]やMMM Mode[注6]などの複数のメジャーモードをバッファ内で自動で切り替える方式を想像されるかと思いますが、Web-modeはそれらとは異なり、単一のメジャーモードとして作られているため、動作が軽いのが特徴です。

Web-modeが提供する機能はおもに次のようになっており、HTMLやJavaScript、PHPやRubyなどの言語が混在する環境でも適切に動作するのが最大の特徴となっています。

- 複数の言語の同時編集
- 自動インデント
- HTMLタグの自動挿入（おもに閉じタグ）
- シンタックスハイライト
- コメントイン／コメントアウト
- コードの折り畳み

基本となる編集についてはとくに意識することなく使えるようになっていますが、**表3**のキー操作を覚えることで、より編集が楽になるかもしれません。

注5）http://Web.emacswiki.org/emacs/MuMaMo
注6）https://github.com/purcell/mmm-mode

▼表3　Web-modeによる代表的なコマンド一覧

キー	説明
C-;or M-;	カーソル行、およびブロックをコメントイン／コメントアウトする（注：リージョン選択不要）
C-c C-n	開始タグ、および閉じタグへカーソルを移動する
C-c C-f	タグ内の文字列を折り畳む／折り畳みを解除する（注：折り畳まれた個所は下線が表示されます）
C-c C-i	カーソル行、もしくはリージョンをインデントする
C-c C-e r	カーソル付近のHTML開始タグと閉じタグを一括リネームする
C-c /	閉じタグを挿入する
C-c C-s	スニペットを挿入する
C-c C-d	HTMLの文法をチェックする

▼リスト2　Web-modeの設定

```
(when (require 'Web-mode nil t)
  ;; 自動的にWeb-modeを起動したい拡張子を追加する
  (add-to-list 'auto-mode-alist '("\\.phtml\\'" . Web-mode))
  (add-to-list 'auto-mode-alist '("\\.tpl\\.php\\'" . Web-mode))
  (add-to-list 'auto-mode-alist '("\\.ctp\\'" . Web-mode))
  (add-to-list 'auto-mode-alist '("\\.jsp\\'" . Web-mode))
  (add-to-list 'auto-mode-alist '("\\.as[cp]x\\'" . Web-mode))
  (add-to-list 'auto-mode-alist '("\\.erb\\'" . Web-mode))
  (add-to-list 'auto-mode-alist '("\\.mustache\\'" . Web-mode))
  (add-to-list 'auto-mode-alist '("\\.djhtml\\'" . Web-mode))
  ;;; Web-modeのインデント設定用フック
  ;; (defun Web-mode-hook ( )
  ;;   "Hooks for Web mode."
  ;;   (setq Web-mode-markup-indent-offset 2) ; HTMLのインデイント
  ;;   (setq Web-mode-css-indent-offset 2) ; CSSのインデント
  ;;   (setq Web-mode-code-indent-offset 2) ; JS, PHP, Ruby などのインデント
  ;;   (setq Web-mode-comment-style 2) ; Web-mode内のコメントのインデント
  ;;   (setq Web-mode-style-padding 1) ; <style>内のインデント開始レベル
  ;;   (setq Web-mode-script-padding 1) ; <script>内のインデント開始レベル
  ;;   )
  ;; (add-hook 'Web-mode-hook  'Web-mode-hook)
)
```

Web-modeの設定

特定のファイルを自動的にWeb-modeで開きたい場合は、**リスト2**の設定をinit.elに追加しましょう。

なお、インデントは標準でスペース2つに設定されています。

Web-modeの使い方

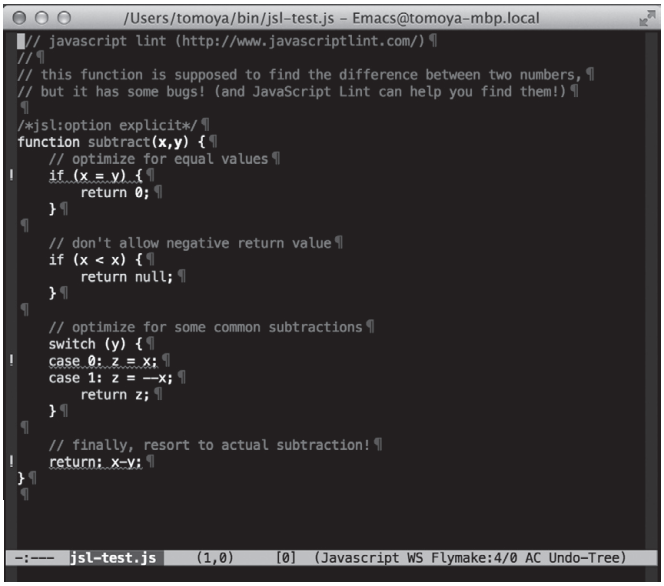
Web-modeの使い方はとくに意識する必要がありません。上記の設定で自動判別されるようになっていれば、あとは、気のむくままにファイルを編集しましょう。

Flymakeによる文法チェック

FlymakeとはEmacsに標準で搭載されているリアルタイム文法チェック機能です。各種プログラミング言語の構文チェッカと連携して、文法をリアルタイムでチェックし、その結果を現在編集中のバッファに視覚的に表示してくれます。

外部プログラムとの連携を前提に作成されて

▼図5　Flymakeの画面（jshint）



▼リスト3　Flymake/jshintの設定

```
;; JS用Flymakeの初期化関数の定義
(defun flymake-js-init ( )
  (list "jshint"
        (list (flymake-init-create-temp-buffer-copy
               'flymake-create-temp-inplace))))

;; JavaScript編集でFlymakeを起動する
(add-to-list 'flymake-allowed-file-name-masks
             '("\\.js\\'" flymake-js-init))

(add-to-list
 'flymake-err-line-patterns
 '("\\([^ \n]+\\): line \\([0-9]+\\), col \\([0-9]+\\), \\(.*\\)"
   1 2 3 4))
```

▼リスト4　Flymake/csslintの設定

```
(defun flymake-css-init ( )
  (list
   "csslint"
   (list "--format=compact" (flymake-init-create-temp-buffer-copy
                             'flymake-create-temp-inplace))))

(add-to-list 'flymake-allowed-file-name-masks
             '("\\.css\\'" flymake-css-init))

(add-to-list
 'flymake-err-line-patterns
 '("\\([^ \n]+\\): line \\([0-9]+\\), col \\([0-9]+\\), \\(\\(Warning\\|Error\\) - .*\\)"
   1 2 3 4))
```

いるため、新しい言語が登場したとしても、すぐに導入できるのが最大の特徴です（**図5**）。

jshintの導入

jshint[注7]は数あるJavaScript用の構文チェッカの中でも新しい構文チェッカであり、人気が高まっています。詳細なインストール方法については誌面の都合上省略しますが、Node.jsをインストールしている状態で、`npm install -g jshint`でインストールし、環境変数パスに/usr/local/share/npm/binを追加している前提で解説を進めます。

Flymake/jshintの設定

jshintがインストールされている環境であれば、Flymakeからの利用はとても簡単です。

リスト3の設定をinit.elに追加し、.jsファイルを開くとjshintによる構文チェックが可能になります。

csslintの導入

csslint[注8]も比較的新しいCSSの構文チェッカです。こちらも、jshintと同様に、Node.jsをインストールしている状態で、`npm install -g csslint`でインストールし、環境変数パスに/usr/local/share/npm/binを追加している前提で解説を進めます。

Flymake/csslintの設定

こちらも**リスト4**の設定をinit.elに追加し、.cssファイルを開くとcsslintによる構文チェックが可能になります。

Flymakeはマイナーモードとして定義されているため、`M-x flymake-mode`でオン／オフの切り替えができます。ほかにもさまざまな言語で使うことができるので、もっと詳しく知りたい方は、技術評論社より発売中の書籍『Emacs実践入門』の第7章をぜひご覧ください。

Emacsから Gitを使う

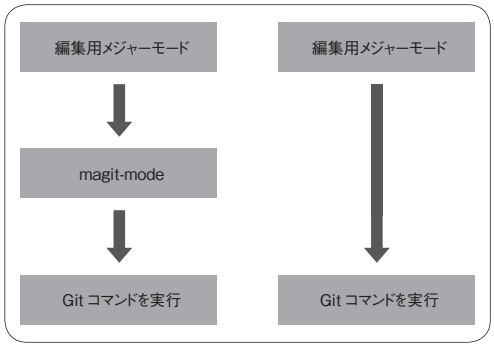
開発者にとって、バージョン管理システム（以下、VCS）はなくてはならないツールです。組織によってそれぞれのVCSが選択されていると思いますが、本章ではGitに焦点を当てて解説をします。

Gitは非常にパワフルな機能を提供してくれるツールですが、操作を覚えるのが難しいと言われています。ですが、基本となるコミットやブランチ作成などの操作をEmacsから行えるようになれば、Emacsを使っているあなたにとっては、思ったより簡単に感じることは間違いないでしょう。

それでは、さっそくEmacsからGitを利用す

注7） http://jshint.com
注8） http://csslint.net/

▼図6　magitの操作フロー



る方法を解説していきます。

magitの導入

Emacsは標準でGitをサポートしていますが、標準機能だけでは、Gitの操作をすべてサポートしているとは言い難い状態です。そこで、magitというEmacsからGitをより便利に利用するためのフロントエンド拡張を導入します。

ELPAから次のコマンドでインストールできます。なお、ELPAからインストールされるmagitは、リポジトリでソースコードが公開されている最新バージョンです。そのため、magitのサイトでバージョンが振られているもの(執筆段階における1.2.0)とは、動作が異なりますのでご注意ください。

```
M-x package-install RET magit RET
```

インストールが完了したら、すぐにmagitが提供するコマンドが使えるようになります。

magitが提供する機能

magitからはGitが提供するほとんどの機能を使うことができます(Gitコマンドがそうであるように)すべてを覚える必要はないので(筆者もすべては覚えていません)、まずは**表4**の基本コマンドを覚えましょう。

これから解説する操作は、おもにこれら基本コマンドから操作することになりますが、この

▼表4　magitの基本コマンド

コマンド	説明
magit-status	git status相当
magit-branch-manager	git branch相当
magit-log	git ログ相当
magit-reflog	git reflog相当

▼表5　magitポップアップメニュー

コマンド	説明
c	コミットポップアップメニューを表示
l	ログポップアップメニューを表示
f	フェッチポップアップメニューを表示
b	ブランチポップアップメニューを表示
P	プッシュポップアップメニューを表示
m	マージポップアップメニューを表示
z	スタッシュポップアップメニューを表示
M	リモートポップアップメニューを表示

基本コマンドを実行すると、magitによるGit操作専用のモード(以下、magit-modeと呼びます)を起動します。magitは、このmagit-modeからの対話的操作と、ミニバッファからコマンドを直接実行する直接的操作の2つに分かれています(**図6**)。

magit-modeの操作は(当然ながら)すべてキーボードから行うことになりますが、機能が多いため覚えるのがたいへんです。そこで、これらの基本コマンドを実行中にメニューを表示する**表5**のコマンドをぜひ覚えましょう。

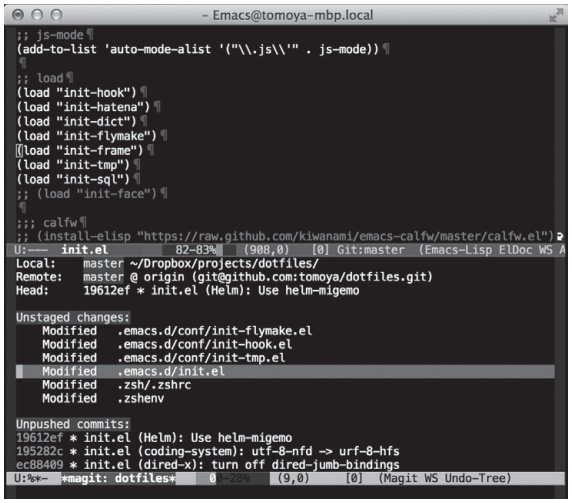
なお、magitによる操作中は**q**を押すことで、キャンセルできますので、こちらもあわせて覚えておきましょう。

Gitリポジトリの作成(M-x magit-init)

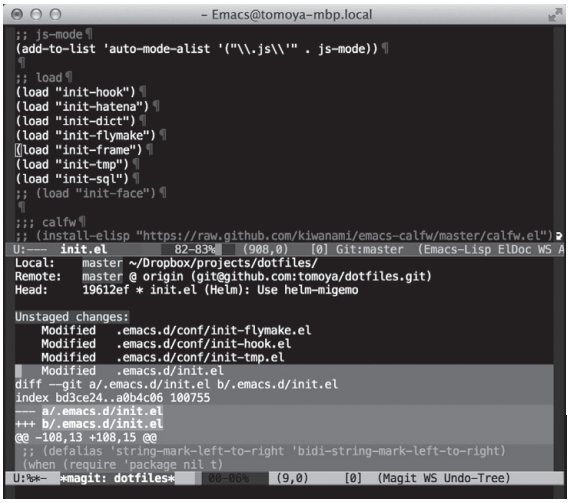
まだGitリポジトリを作成していないときに、Emacsからリポジトリを作成するには、**M-x magit-init**を実行します。

すると、Gitで管理する起点となるディレクトリを聞かれますので、そのままでよければ、Enterを入力します。すると、**git init**による初期化が完了します。とても簡単ですね。

▼図7　magit-statusの画面



▼図8　部分ステージングの画面



magitによるコミット（M-x magit-status）

リポジトリを作成した次はコミットでしょう。Gitでコミットするには、まずはコミットの対象となるファイルをステージングします。そのためには、**M-x magit-status RET**を実行します。

すると、**図7**のようなgit statusを実行したときと似た画面が表示され、こちらから対話的にステージングを行えます。

■ステージング／アンステージング

この画面では、**p**、**n**で上下に移動できるようになっています。ステージングしたいファイルの行で**s**を入力すると、そのファイルがステージングされます。また、ステージングされたファイルの行で、今度は**u**を入力すると、今度はアンステージできます。

すべてのファイルを一括でステージングしたい場合は、**S**（大文字ですので Shift ＋ S ）を入力すると、現在アンステージとなっているファイルすべてがステージングされます。

■部分ステージング

Gitではファイルではなく、特定の編集個所だけをステージング（つまりコミット）できます。それを部分ステージングと呼びますが、これもmagitから行えます。

部分的にステージしたいファイルの行で**M-s**を押すと、ファイル名の表示から変更差分が展開表示されます。この表示でも**p**、**n**によって差分ブロックを移動でき、先ほどのファイルをステージングするのと同様に、今度は変更ブロックで**s**を入力するとファイルの一部分だけがステージングされます。なお、**M-h**で展開を閉じることができます（**図8**）。

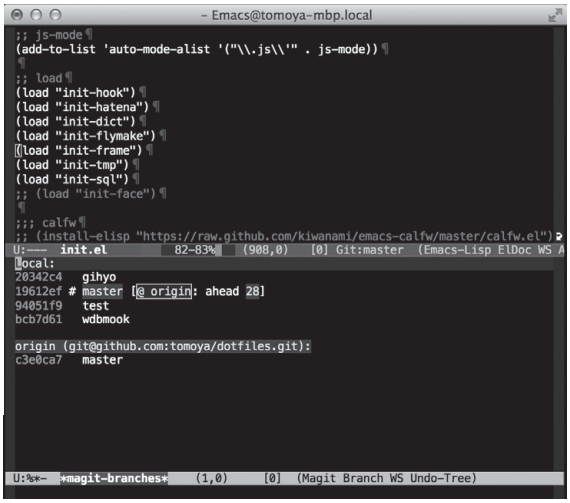
■コミット

ステージングが完了し、いよいよコミットする場合は**c**を入力します。すると、コミットポップアップメニューが開くので、さらに**c**を入力します。次にコミットログを入力する画面が表示されるので、こちらにコミットログを書いて**C-x C-s**で保存して**C-x #**を入力します。これでGitリポジトリへのコミットが完了します。

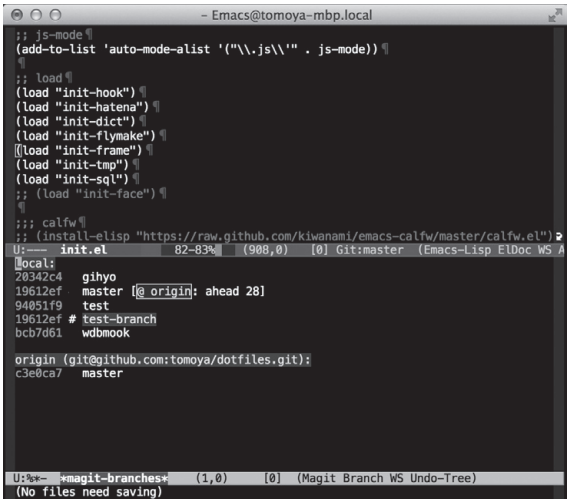
magitによるブランチ作成

magitからブランチを作成するには**M-x magit-branch-manager**を実行します。する

▼図9　magit-branch-managerの画面



▼図10　ブランチ作成後のmagit-branch-managerの画面



と、図9のようなgit branchに似た画面が表示されます。

ブランチを作成するには、この画面でcを押します。すると、ミニバッファで作成するブランチ名を聞かれるので、ブランチ名(ここではtest-branchとしましょう)を入力して Enter を押します。これで、図10のようにブランチが作成され、作成されたブランチへと切り替わります(つまりgit co -b test-branch相当のコマンドが実行されたわけです)。

■ブランチの切り替え

現在のブランチは、ブランチ名の前に#が表示されています。この画面もp、nで移動できるようになっており、切り替えたいブランチ名の行で Enter を入力するだけで、ブランチを切り替えられます。

magitによるプル／プッシュ

リモートリポジトリがある場合、そちらに対してプル／プッシュを行うことになるでしょう。その操作も、もちろんmagitから行うことができます。

リモートブランチの登録はすでに設定済みという前提で、magitからの操作のみ解説すると、M-x magit-pullというコマンドだけでリモートブランチからプルできます。プッシュは、M-x magit-pushです。とても簡単ですね。

終わりに

ここまで駆け足でWeb開発におけるEmacsの便利な使い方について解説しましたが、いかがでしたでしょうか？　ここで紹介した内容は、もちろんEmacsのすべてではなく、その一端にすぎません。

今回紹介したような、拡張機能を使って便利に使う方法もあれば、Emacs標準の機能を駆使して、純粋に編集を強化することもできます。ある意味、後者のほうが効率を上げるという意味では効果的かもしれません。

Emacsにはエディタという枯れたソフトウェアならではのおもしろさと深さがあり、我が友というタイトルに偽りなく一生の付き合いとなる可能性が十分ありますので、興味を持った方は、ぜひお試しください。SD

Column 3 もしEmacsがなかったら？ その❷

Writer 小飼 弾（こがい だん）Twitter@dankogai

「もしEmacsがなかったら、私はどうなっていただろうか」

プログラマとなることはなかったはずである。私にとってそれは、筆記用具なしで作家になれというのにも等しいのだから。しかしそれは世界にとってささやかな違いでしかない。Emacsはそんなささやかなものなのだろうか？

「もしEmacsがなかったら、世界はどうなっていたのだろうか」

WYSIWYGという考えそのものが普及しなかったのではないか。

“What you see is what you get.”

今でこそそれは、Xerox Altoが開き、MacintoshとWindowsが普及させたGUI = Graphical User Interfaceを連想する言葉となったが、テキストだけが表示される世界で、それは何を意味するか？

キーを打つと、その上に刻印された文字がそのまま入力され表示されるという意味である。aを打てばa、eを打てばe、iを打てばi、oを打てばo、uを打てばu……タイプライターであれば当たり前だったこの常識は、実はEmacsの普及以前は常識ではなかった。vi(m)使いにとって、それは今なお常識ではない。aを打って“a”が入力されるか現在のカーソルの次の位置にカーソルが移動するかはモードしだいだからだ。

コマンド入力と文字入力。Emacs以前、それは「切り替える」ものだった。ESCでコマンド入力に。a、i、oなどで文字入力に……つまりユーザは自分がどちらのモードにいるかを常に把握していなければならない。Vimでこそ文字入力モード中は`-- Insert --`と底に表示されるようになったけれども、viの時代はそれすらなかった。

Emacsにおいて、コマンド入力は「特別なキーを押しながら」文字キーを押すものとなっている。Emacsという言葉が何を意味するかは未定義だそうだが、「Escape、Meta、Alt、Control、Shiftの略」という説はかなり説得力がある。別の言い方をすれば、これらの特別なキーを押していないときには、押したキーのトップに書かれた文字がそのまま入力され、表示されるということです。この原則ふつうにバッファに文字を入力される場合のみならず、`M-x`を押してミニバッファにコマンド名を入力しているときにもそうである。

EMACSにも「モード」という言葉はあるけれども、それはvi(m)とは意味するところがまるで異なる。EMACSにおけるモードとは、何を編集しているか、どこを編集しているか、つまりcontextualに自動で決まるものであり、ユーザーがどのモードにいるのかをなるべく意識させないために存在する。常にどちらのモードにいるかを把握せねばならないvi(m)とは真逆なのだ。

Emacs最大の功績、それは「エディタ機能も備えた完備なLisp環境」ではなく、「コマンドは特別なキーを押しながら」というキーボードショートカットという概念そのものを普及させたことにあると私は考えている。tcshやbashやzshといった、コマンドヒストリー編集可能なシェルのデフォルトのキーボードショートカットがEmacsゆずりなのも、Macが「コマンド」キーを必ず備えているのもむべなるかな。余談ではあるが、Windowsにどうしてもなじめない理由も、同OSがCtrlをCmdの代わりに割り当ててしまったことが一番大きい。

私自身、Emacsを使いつづけて四半世紀が経っているけれど、「高機能エディタ」以上の使い方はしていない。折角のShellモードすら、`ctrl-Z`で切り替える私には宝の持ち腐れかもしれない。しかしviのモード地獄から救われて以来、私の`EDITOR`がEmacsのままなのは、私もまた“the rest of us”である証拠なのかもしれない。SD

第2特集

コンピュータの加速装置

サーバサイドフラッシュ Fusion-io全方位解説

サーバマシン・データセンターを支えるテクノロジー

今、サーバの高速化を実現するためにフラッシュストレージを導入する事例が増えています。中でもFusion-io社の技術／製品に注目が集まっています。本特集では、これまでのフラッシュストレージの変遷と今後の活用スタイル、フラッシュメモリの動作原理、新技術の性能検証など……さまざまな側面から、Fusion-ioを中心としたサーバサイドフラッシュについて解説を行います。

長所／短所をきちんと理解しよう

第1章 NANDフラッシュのしくみと特徴

長谷川 猛（はせがわ たけし）
フュージョンアイオー㈱ ソリューションアーキテクト
thasegawa@fusionio.com

ノートPCやデスクトップPCのHDD置き換えだけでなく、サーバにフラッシュストレージを載せて高性能化する風潮が見られます。まずはフラッシュを基礎知識からおさらいして、自信をもってストレージ選定ができるインフラ担当者になりましょう。

フラッシュストレージとは

ハードディスクの誕生から現在まで

ハードディスクドライブ（HDD）は1955年に登場したIBM 350ディスクストレージから始まり、1973年にほぼ現在のデザインであるウインチェスター型HDDが出てきました。IBM 350ディスクストレージは洗濯機ほどの大きさで5MBの情報を記録する、今となってはたいへん低密度なストレージでしたが、50年以上に渡るディスクストレージは進化の結果、今や2.5インチのドライブで何テラバイトものデータが保存できるようになりました。

このように、HDDはさまざまな技術革新により発展を遂げていますが、記憶密度の向上による大容量化が中心であり、転送レートや秒間あたりのI/O性能やI/Oあたりの所要時間はそれほど劇的な改善が図られていません。プロセッサなどコンピュータの基本部分はムーアの法則に従って18ヵ月に2倍の勢いで性能向上してきましたが、HDDはその進化から取り残されており、コンピュータの中でも最もボトルネックになりやすい部分となっています。

HDDを高速化するには、メディアの回転速度を増して「回転待ち時間」を短縮したり、アームを制御するサーボを高性能化して「シーク時間」を減らしたりすることが考えられます[注1]が、メディア—ヘッド間のすき間がごく僅（わず）かのため[注2]接触によりメディアクラッシュの危険があったり、メディアが高密度化していることから回転速度やシーク速度を上げることが難しい、などの課題があります。

モバイル端末はフラッシュへ、そしてサーバも

一方、フラッシュメモリは技術革新により短期間で大幅な容量向上を果たしました。フラッシュメモリは大容量化しやすい半導体メモリ技術で、従来のハードディスク（HDD）のようなメカニカルドライブと違って、半導体部品で二次記憶を実現できるため、シンプル、高性能、低消費電力が両立できる点が特徴です。20年前にはフロッピーディスクなどの記憶メディアが栄えていましたが、今ではフラッシュメモリのほうが容量密度、転送速度などさまざまな点において、利便性が高い記憶媒体となりました。

とくに本特集で扱うNANDフラッシュはデジカメ、スマートフォン、USBフラッシュメモリをはじめとするコンピュータ関連用品で広く採用されているほか、ソリッドステートドライブ（SSD）などHDDの代替品としての用途も広がっ

注1）HDDのアクセス時間＝シーク時間＋回転待ち時間＋データ転送時間。
注2）「ジェット機が高度1メートル以下を数百km/hで飛行しているようなもの」なのだそうです。

ています。読者のみなさんも、さまざまなシチュエーションでNANDフラッシュ製品から多くの恩恵を受けているのではないでしょうか。そして最近では、NANDフラッシュ技術によるストレージをサーバに組み込む流れが本格化しています。

スマートフォンやタブレット端末などが普及し、とくにWebサービス業界が捌(さば)くクエリ数は大幅に増加し続けています。これらはフラッシュメモリを搭載した「スピンドルレス」な端末がどんどん送信してくるリクエストです。これに対してサーバ側はどうでしょう？　まだハードディスクを使っている方も多いのではないでしょうか。

ディスクフラッシュによる端末側のスペックが高性能化するのに併せて、サーバにも同様のパフォーマンス向上が求められています。サーバがHDDベースで低速なままでは、端末側の性能に遅れをとってしまいます。サーバへのフラッシュ適用の必然性は明白でしょう。

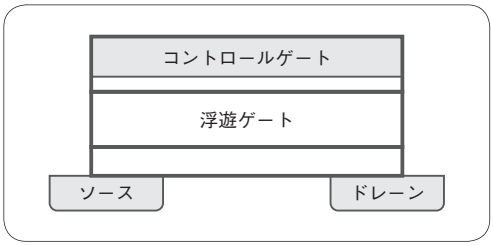
フラッシュメモリのしくみ

さて、みなさんにとっても身近な存在となったフラッシュメモリとは、どんなしくみの記憶メディアなのでしょうか？　フラッシュメモリといってもいくつかの種類がありますが、今、フラッシュメモリという場合には、1987年当時、東芝に在籍されていた舛岡富士雄氏が発明し、現在ではデジカメやUSBメモリなどをはじめとする多くの記憶メディアで使われる日本発の技術である、NANDフラッシュを指していることがほとんどです。

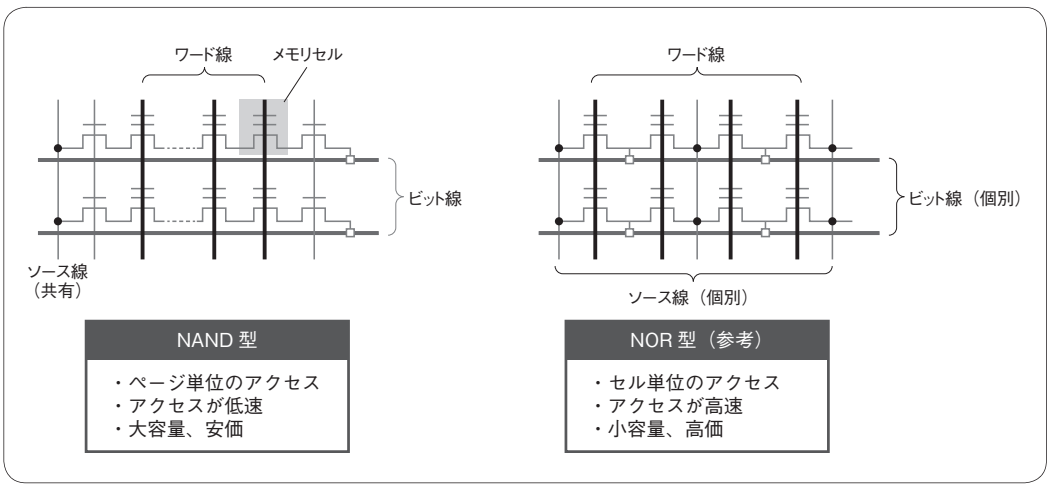
セルの構造とライフサイクル

NANDフラッシュは、**図1**に示すようなメモリセル回路の集合体でできています。NANDフラッシュの特徴として、類似技術であるNOR型のものと比べ、配線量が削減され、記憶密度がとても高いことが挙げられます。NOR型の場合はソース線とビット線が各メモリセルへ結線されているため配線量が多いのに対し、NANDフラッシュではソース線、ビット線をメモリセル

▼図1　NANDフラッシュのメモリセル



▼図2　NAND型とNOR型の比較



間で共有することにより容量密度が増し、今日のデジタル機器が要求する膨大な記憶容量ニーズに応えるものとなりました(**図2**)。

フラッシュメモリはメモリセルに電子を溜めることでデータを記憶します。セルのソース—ドレイン間に高電圧をかけると、トンネル効果により電子が浮遊ゲートに入ってくる現象を用いてメモリへのプログラムを行います(**図3-①**)。プログラムされたメモリセルはその浮遊ゲートに入っている電子の量を電圧として読み取れるため、この特性を利用してリードを行います[注3](**図3-②**)。最後に、プログラム済みのメモリセルのソース側に高電圧をかけると、トンネル効果により電子が浮遊ゲートから抜け出て、またプログラム可能な状態に戻り、イレースが行われます(**図3-③**)。

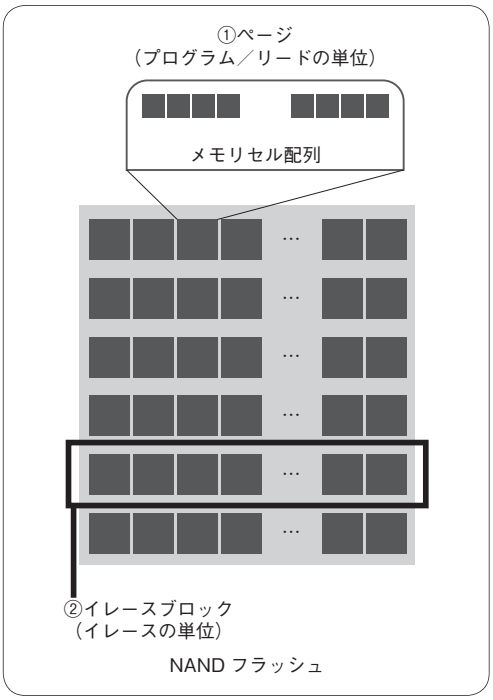
フラッシュメモリはメモリセルに通電していなくてもデータが持続する「不揮発」の特性がありますが、実際には繰り返し利用するうちにメモリセルを構成するトンネル酸化膜部分が劣化して電子が漏れるようになり、記憶素子としての寿命を迎えます。

注3) フラッシュメモリのメモリセルはイレース状態が1の値を指し、プログラムによって値を0に変化させます。

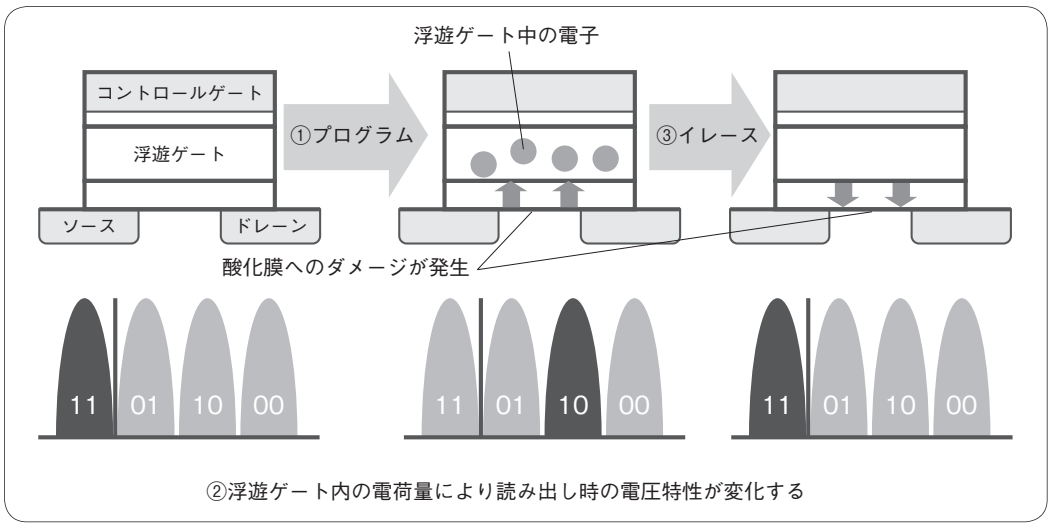
ページサイズとイレースブロック

NANDフラッシュでは、データに対するプログラム／リードの単位がページ単位(**図4-①**)で行われるという特徴があります。NOR型はバイ

▼図4 ページとイレースブロック



▼図3 プログラムとイレース



ト単位のアクセスができるのに対して、NAND型はソース線、ビット線の共有による大容量化と引き替えにアクセスがページ単位となりました。ページサイズは製品仕様によりさまざまですが、4KB、8KB、16KBなどのページサイズ仕様のものが流通しており、大容量の製品ほどページサイズが大きくなる傾向にあります。

NANDフラッシュは複数ページに一括でイレース操作を行う仕様となっています。1つのページを消去したくても、イレースブロック(EB)の単位で消去しなければなりません(**図4-②**)。やはりEBのサイズも製品仕様しだいですが、たとえば8KBのページサイズに対してEBのサイズは2MBといった仕様になっています。

ページサイズやEBのサイズが大きくなれば、メモリセルを増やしやすくそれだけ大容量化が実現しますが、引き替えに小さな単位での読み書きにおいてハードルが高くなります。今後はページサイズ16KB、EBサイズ4MBといった仕様のチップが主流になるでしょうから、安価に大容量なデバイスが作れるようになる半面、コントローラ側の負担は増していくことでしょう。

メモリセルの種類

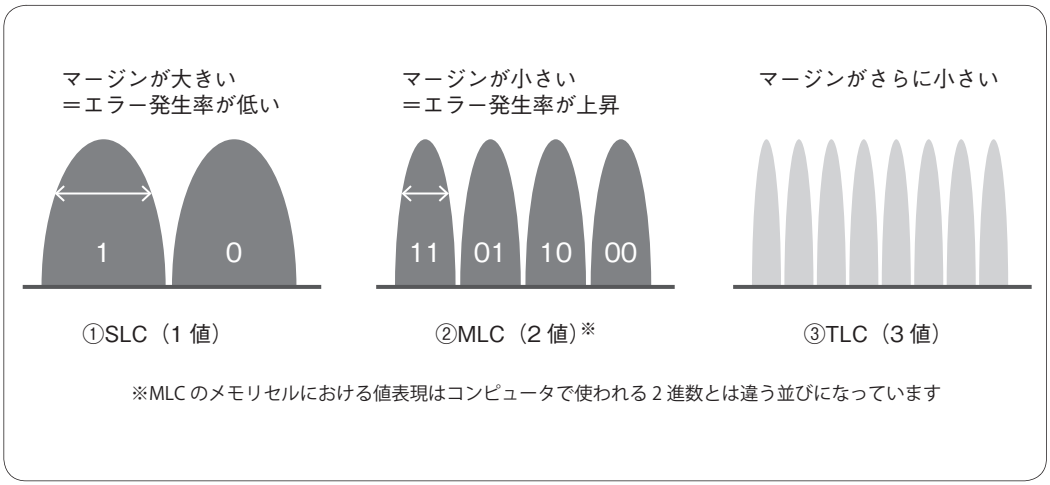
NANDフラッシュメモリのセル種類は、大きくSLC、MLCとTLCの3種類があります(**図5**)。

最も基本的なNANDフラッシュがSLC(Single Level Cell)です。SLCは1つのセル中に電荷がどれだけ貯まっているか――つまりメモリセルの電圧が一定値を超えていれば0、そうでなければ1と判断して1ビットの情報を記録します(**図5-①**)。SLCはその単純さゆえに高速で、メモリセルの書き込み耐性が強く、ビットエラーの発生も少ない点が特徴です。バイト単価が高く利用用途が限定されていることから生産量が少なくなりましたが、現在でもエンタープライズ向けのSSDなどに使われていることがあります[注4]。

現時点では、セルあたり2ビットの情報を記録するMLC(Multi Level Cell)が主流となっています。セルの物理的な構造はSLCと一緒ですが、セル内に複数のしきい値を持つことで複数ビットの情報を記録して大容量化を実現し、ビッ

注4) Nintendo 3DS用ソフト「どうぶつの森」のカードリッジにもSLCのNANDフラッシュメモリが使われたようです。圧倒的な人気とフラッシュチップの供給不足により一時期はカードリッジ版の入手がたいへん困難でした(筆者はダウンロード版で購入しました:-))。

▼図5　メモリセルの多値化



ト単価も低減されています(**図5-②**)。今どきのデジタル機器のほとんどはMLCを採用しています[注5]。

さらに、最近ではTLC(Triple Level Cell)と呼ばれるしきい値をさらに増やし、1つのセルで3ビットを記録する仕様のチップが供給されています。TLCはMLCからさらに容量密度が高くなりますが、ビットエラーが増加するうえ速度も遅くなるため、低価格な製品で使われていることが多いようです[注6]。

NANDフラッシュの性能と課題

セルの信頼性

半導体の製造プロセスの微細化、またメモリセルの多値化によって、NANDフラッシュは「より小さなセル」で「より多くのデータ」を記録できるようになってきています。しかし実際には、微細化の影響によりメモリセル内の電子量が減少しているうえ、リーク電流などの要因によりメモリセル間の干渉も発生します。このため高密度なチップほどビットエラー発生率は高くなります。

エラーの検出、修正のため、メモリコントローラはエラー訂正(ECC)を行い、失われたデータを復元しなければなりません。SLCと比べてMLCはエラーが発生しやすく、コントロールが難しくなるうえに書き換え耐性が低くなります。TLCではさらにハードルが高くなります。そのためNANDフラッシュチップの微細化/多値化は、それに合わせた高度なコントローラ技術が求められます。

アクセス速度

とあるNANDフラッシュメモリの具体的なスペックを見てみましょう(**表1**)。100マイクロ秒未満でリードできるのに対して、プログラムの性能は数百マイクロ秒と、1桁の差があります。ブロックイレースはさらに時間がかかる操作で、HDD同様にミリ秒オーダーの時間がかかります。

また、NANDフラッシュは半導体の製造プロセスやメモリセルの多値化に伴い、アクセス速度が低下する傾向にあります。より正確にプログラムできるように途中でベリファイ(検査)するなど複雑化してきていることから、メモリセルへのプログラム時間はますます長くなってきています。またリード時には、MLCやTLCは電圧のしきい値が複数化し、判定に時間がかかるほか、ビット化けを起こしたデータを復元するために繰り返しECC回路でエラー訂正を行います。フラッシュ技術が高密度化すればするほど信頼性が下がりますので、今後もエラー訂正コードは高度化/複雑化していくことでしょう。

帯域幅の観点では、フラッシュメモリ単体の転送速度は、実はそれほど速くありません。たとえば1つあたり30MB/sの転送速度であれば、それをストライプすることで60MB/s(2倍速)、240MB/s(8倍速)といった性能を実現しており、パッケージ内のダイ数が増えるとともに高速化します[注7]。現在ではパッケージあたり8階層、もしくはそれ以上のダイが積層化される傾向にあります。今後も積層化が進めば大容量化、帯域幅性能が進むものと思われます。SD

注5) それぞれの個体には品質差があるため、MLCの中でも良質なMLC個体にはeMLCというブランドおよびプレミアがつき、信頼性が求められるサーバ製品向けデバイス用などとして出荷されることもあります。

注6) 実際にコンピュータサプライ品売り場では、TLCが利用されている(と思われる)、ローエンドのメモリカードやSSDも市場で見かけるようになりました。

▼表1 とあるフラッシュチップにおけるアクセスレイテンシ

操作	レイテンシ	リードとの比較(%)
リード	25 μs	100%
プログラム	200 μs	12.5%
ブロックイレース	1500 μs	1.6%

注7) SDカードでも転送速度が異なる製品ラインナップが存在します。同様に、高性能を実現する場合には複数のフラッシュメモリに並列アクセスしなければないらないため、同じSSDでもフラッシュ搭載量が少ない低容量モデルは、高容量モデルと比べて性能差が生じます。

SSDならではのHDDとの違いを納得

第2章 フラッシュストレージ大解剖

長谷川 猛(はせがわ たけし)
フュージョンアイオー㈱ ソリューションアーキテクト
thasegawa@fusionio.com

第1章では、NANDフラッシュを用いたメモリ技術の概要について説明しましたが、本章では、SSDの全体像やコントローラの役割について説明します。

ストレージデバイスとしてのフラッシュ

コンピュータとNANDフラッシュは直接接続できず、コントローラを媒介する必要があります。コントローラはフラッシュの管理やホスト側とのプロトコル変換など、さまざまな機能を提供しており、とても複雑な造りになっています。フラッシュストレージ自体が1つのコンピュータシステムのようなものだと言っても過言ではないでしょう。

SSDコントローラ

コンピュータ側からのディスクI/Oには、SATAやSASなどのストレージプロトコルを用いるのが一般的です。しかし、フラッシュメモリ自体はストレージプロトコルを利用できません。このため、SSDにはプロトコル変換のための組み込みプロセッサが搭載されており、HDDを前提としたハードウェア／ソフトウェアがそのまま動作するしくみとなっています。

しかし後方互換性を重視した結果、プロトコル変換が遅延を引き起こしたり、フラッシュメモリの特性を活かしたデザインとならず、性能が活かし切れない課題が生じています。

Fusion-ioではPCI Express経由でデバイスを直結していますが、SATA/SASに代わるインターフェースとしてSCSI ExpressやNVM Expressといったフラッシュ接続用のインターフェースも策定されました。最近のノートパソコンではPCI Express接続のSSDが搭載されたモデルも登場していますし、HDD用のものからフラッシュに適したインターフェースへの移行が進んでいくでしょう。

Flash Translation Layer

現在のオペレーティングシステムは、HDDのアクセス仕様から512バイト(もしくは4KB)セクタに対するランダムアクセスを前提としてストレージI/Oが設計されています。しかしこのアクセス仕様は、4KBや8KB、もしくは16KBの物理ページサイズで読み書きするフラッシュメモリは互換性がありません[注1]。また、現時点のOSやファイルシステムはフラッシュのブロックイレースの制限を考慮せずに作られています。

そこで、OSとフラッシュメモリの間を埋める存在がFlash Translation Layer(FTL)です(**図1**)。FTLは、HDDとは異なるフラッシュメモリの特性を隠蔽し、HDDのようなセクタ単位のランダムアクセスを可能とするためのリマッピング機能を提供します。多くの場合、FTLの内部ではログ構造ファイルシステム、もしくは

注1) 既存のソフトウェア資産の多くが512バイトセクタを前提としたコードになっているため、なかなか解決が難しい課題です。ソフトウェア開発者のみなさんは、今後は4KBもしくはそれ以上のI/Oが最小単位となることも想定したコードを書きましょう!!

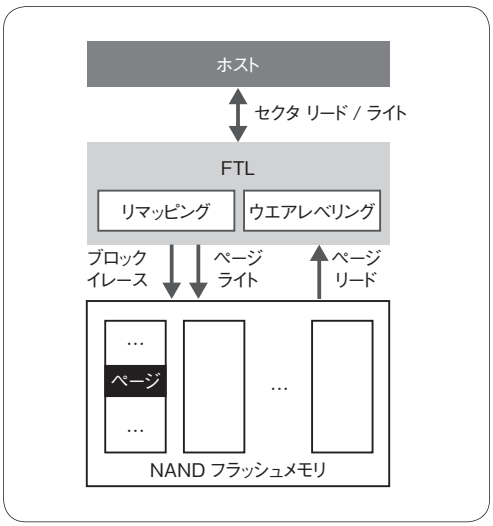
相当のデータ構造を持ち、512バイトセクタのメディアをエミュレーションします。

FTLの処理はSSD内の組み込みプロセッサおよびDRAM上で行われますが、後述の課題を考慮しながら何百GBにもおよぶ物理／仮想アドレス空間をやりくりするたいへん高負荷なものとなります。このため組み込みプロセッサのクロック向上、マルチコア化を宣伝するSSDベンダもあるようです。

ウエアレベリング

フラッシュメモリは、同じブロックに対してプログラム／イレースを繰り返すうちにメディアとしての信頼性が下がります。さらに書き込みを繰り返せば、ウエアアウト(wear-out)が発生して記憶素子としての役目を果たさなくなります。このため、FTLは配下にあるフラッシュメモリのプログラム回数を最大限に活用するため、内部のデータ配置を動的に変更します。たとえば、プログラム回数が少ないブロックに更新の少ないデータを置いておくと、そのブロックの書き込み可能サイクル数を活用できません。このため、使い込まれたブロックには更新頻度が低いデータを配置し、繰り返し書き込みが可能なブロックには一時的なデータを割り当てる、

▼図1　Flash Translation Layer



などのやりくりが行われています。

ガベージコレクション

フラッシュメモリは、イレースをしないと新しいデータを書き込めないという特性があります。しかしイレースにはミリ秒単位の時間を要するため、プログラムを行う前にあらかじめ、計画的にイレース処理を終えて空き領域を確保しなくてはなりません。FTLは常に書き込み可能なブロックを確保するために、イレース予定のブロックから、まだ必要なデータをほかのブロックにコピーし、ブロック全体が不要となってからイレースします(**図2**)。この処理はフラッシュデバイス内部で書き込みが発生するためデバイスの寿命を縮める可能性がありますが、デバイスへの書き込みを受け付けるためにはブロック単位のイレースは避けられず、いかに効率的なガベージコレクションを行えるかによって、フラッシュストレージとしての寿命に大きな影響を与えます。また、このようなデバイス内処理の複雑さが、SSDの不安定な書き込み性能[注2]の一因となっています。

オーバープロビジョニング

オーバープロビジョニングとは、フラッシュストレージがユーザの見えないところに隠し持っている予約領域のことです。フラッシュストレージではデバイスの広告容量に加えていくらかの余剰容量を搭載しておき、ブロックイレースの遅延を隠蔽したり、メモリセルに異常が発生した場合に置き換えたりするために利用します。予約領域の使い方はコントローラによってまちまちのため単純な比較は難しいですが、一般的に、オーバープロビジョニングを少なくすれば信頼性や書き込み速度が犠牲になり、多くすればデバイスが高価格化したり実効容量が目減り

注2) とくに初期のSSDは書き込みスループットが10分の1近くまで低下したり、何秒間にも渡り書き込みが遅延する「プチフリ」が多くみられました。現在ではある程度改善されているにしろ、SSDは高いライト負荷を苦手とする傾向があります。

したりするため、性能、容量、ガベージコレクションの動作効率や書き込み耐性、価格といったさまざまな面でバランスをとる必要があります。

RAIDによる冗長化と耐障害性

SSDは半導体部品で構成されているため故障が少ないと謳われることも多いのですが、HDDの代替として設計されたSSDですから、単体レベルでの冗長性はないに等しい状況です。SSD内では複数のフラッシュメモリにデータをストライピングしているため、内部部品の不具合が機能停止につながります。安定した動作を求めるならば、複数のSSDでRAIDによる冗長化を組む必要がありますが、それにはさまざまな考慮すべき事項があります。

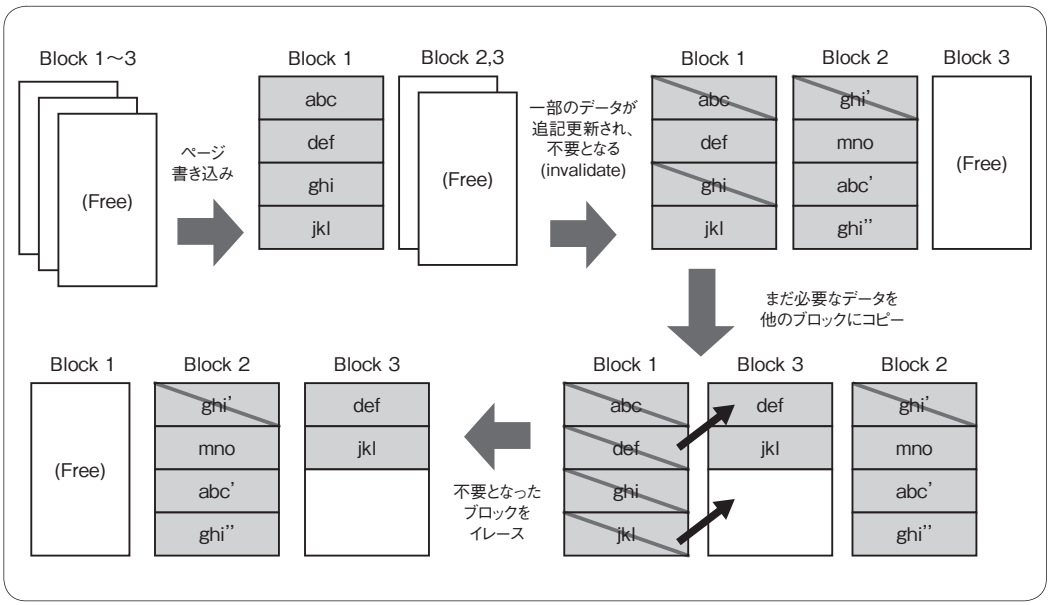
1つ目の課題がRAIDコントローラの性能面です。1発せいぜい数百IOPSのHDDに対して、現在のSSDは1基で万単位のIOPS性能を実現するためRAIDコントローラへの負荷も桁違いに大きくなり、高性能ながら高価格なコントローラチップが求められます。またRAID5/6のような分散パリティによる保護は、パリティ計算のためにRAIDコントローラがボトルネックにもなり、実効性能の観点から推奨されません。

HDDを用いたアレイの場合でも利用年数によって物理故障が増えますが、フラッシュストレージの場合は書き込み量による故障にも注意が必要です。SSDの設計やアクセスパターンによりデバイスの書き込み寿命がいつ訪れるかが変わりますが、同一モデルのSSDをミラーリングした場合、故障時期がおおよそ一緒となり、筐体内でSSDを冗長化しても故障時期が重なってしまいます。また、SSDが不具合や書き込み寿命が原因でリードオンリーモードとなったときにRAIDコントローラが正しくハンドリングできず、データ破損につながったり、アレイ全体に影響が出たケースもあり[注3]、RAIDコントローラとSSDの組み合わせには注意が必要です[注4]。

SSDをストライピングする場合、RAIDコントローラが複数のSSDに書き込みを分散させたとしても、アレイ内のSSDがガベージコレクショ

注3）“SSDで痛い目にあったから”という理由でデバイスレベルで冗長化されており、信頼性に優れたFusion-io製品を検討いただくケースも少なくありません。
注4）アレイコントローラとの互換性はHDDでも要注意ですので、特別SSDに限ったことでもありませんが……。

▼図2　ガベージコレクション



ンで一時停止すればアレイ全体の応答速度に響き、性能が悪化することがあります[注5]。また、データベースのログなどはHDDの特性を活かすためにシーケンシャルで書き込みされることが多いですが、ストライプ幅の都合により書き込み負荷が一部SSDに偏り、性能が出づらいといった現象が起きます。とくに書き込み性能を期待してSSDを複数ストライピングする場合は、期待通りの効果が得られるか注意が必要です。

モニタリング

フラッシュストレージがHDDと異なる点として、書き込み量に制限があるため、安心して運用するには定常的な利用状況のモニタリングが必要です。SATAなどで接続するSSDにはSMARTによるデバイス状態の確認ができますが、現状フラッシュストレージが報告するメディア状態の値やその基準について、統一されているわけではないようです。

パワーカット対策

フラッシュメモリへのデータ書き込みには時間がかかります。また半導体の製造プロセスの進化および多値化が進むにつれ、メディアへの所要時間は増加傾向にあります。このため、エンタープライズ利用を想定したSSDでは予期せぬパワーカットに対応するため[注6]、DRAMにバッファリングした書き込みデータを保護すべくキャパシタを搭載したり、アレイの整合性を確保するためにRAIDコントローラのライトキャッシュをバッテリでバックアップしたりします。しかしバッテリには経年劣化、放電時にライトキャッシュが効かないなどの問題があったり、大型のキャパシタも液漏れや爆発などの原因となります。SD

注5) HDDで言えばディスクアレイ中でドライブ群が「カッコーン、カッコーン」とシークエラーを起こしながら動いている姿を思い描いてみてください。HDDであればディスク交換で解決するでしょうが、SSDではガベージコレクションなどによる書き込み性能のスパイクが日常的に発生するのが難しいところです。

注6) 意外とパワーカット時の安心を謳(うた)ったり、きちんと対策できているSSDは少ないようです。USENIX FAST '13での発表結果によれば、15台のSSDに対して最大で数千回のパワーカットテストを行ったところ、13台に異常が発生したとのこと。Understanding the Robustness of SSDs under Power Fault(http://tinyurl.com/b9k7zl5)

COLUMN SSDはどれぐらい信頼できるのか?

SNIA Solid State Storage Initiative(SSSI)が、SSDが実現すべき信頼性について定義しています。この基準によれば、コンシューマ用途のSSDでは、10^{15}セクタあたり1セクタ未満のビットエラー発生率を要求しています。エンタープライズ用途のSSDではさらに厳しい、10^{16}セクタあたり1セクタ未満という品質基準があります。なお、この仕様はすべてのSSDが準拠しているわけではありませんし[注7]、メーカーによってはほかの基準を独自に持っているケースもあると思われます。

この基準は素のフラッシュメモリの信頼性では到底達成できるものではなく、コントローラ側が読み取ったデータを複数のエラー訂正回路に繰り返しかけ、元データを復元することで達成される信頼性です[注8]。

注7) 上記基準を満たしていないエンタープライズ向けSSDなども販売されていました。この値はSSDのデータシートなどにUBER(Uncorrectable Bit Error Rate)などとして掲載されていることがあります。
注8) Fusion-ioの場合、複数のECCロジックを独自開発し、USERもSNIA基準から何桁も低い値で設計/テストしています。

ioDrive2が選ばれる理由を解明

第3章 Fusion-io ioDrive2入門

長谷川 猛(はせがわ たけし)
フュージョンアイオー㈱ ソリューションアーキテクト
thasegawa@fusionio.com

第1章と第2章で、NANDフラッシュのメモリ技術と、フラッシュストレージについて説明してきましたが、本章では実際の製品としてのFusion-io ioDrive2を取り上げ、その活用法を説明します。

ioDrive2とは

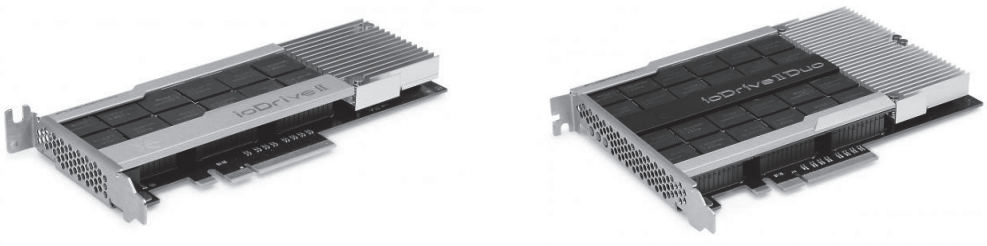
Fusion-io ioDrive2は、SSDが抱えるさまざまな問題を解消し、メモリとストレージの融合を目指して設計された、PCI Express接続のフラッシュデバイスです。第二世代のioDrive2(**写真1**)は2011年10月に発表されて以来、国内外でシステムの高性能化を支えてきました。本誌出版時点で"満2歳"の製品ですが、NANDやコントローラ技術の進歩の中、フラッシュデバイス製品としての性能や信頼性、内部の冗長性など、業界をリードする存在であり続けています。

高性能を実現する「カットスルーアーキテクチャ」

ioDrive2は、HDDからの思想ではなく、メモリティアの思想で設計されました。ストレージプロトコルを介さずにNANDフラッシュを接続し、ホスト側からコントロールする「カットスルーアーキテクチャ」を採用した(**図1**)、いわばNANDフラッシュを用いたRAMカードです。ストレージの域を超えた性能がアプリケーションやミドルウェアの高速動作につながり、システムの性能問題を解決します。

カットスルーアーキテクチャでは、ホストプロセッサがFTL(Flash Translation Layer)の役割を果たします。現在のホストプロセッサはマルチコア高クロックなうえ、高速なDRAMも接続されていますので、この上でFTLの処理をしたほうが速くシステム全体の処理性能向上につながります。このため、ioDrive2の基板上には、フラッシュメモリと、ホストプロセッサからの直接アクセスを可能にする「データパスコントローラ」程度しか載っていません。しかし、コンピュータにioDrive2を搭載し、OS側に「ioMemory Virtual Storage Layer」(以後VSL)と呼ばれるソフトウェアをインストールすると、HDDやSSD同様にブロックデバイス

▼写真1　左：ioDrive2、右：ioDrive2 Duo



として、しかし非常に高速なデバイスとして利用できるようになります。このしくみにより、とくに書き込み応答速度は15マイクロ秒という超レイテンシー特性が実現されており、データベースや仮想化などI/Oを酷使するワークロードとの相性がよくなっています。

「ホスト側のCPUやメモリを消費するのがもったいない」とたびたび言われることもあるのですが、複数の小さな組み込みコントローラで分散処理する「ストレージ的発想」よりも高効率、かつ部品点数が減るため、SSDなどと比較すると投資額あたりの性能が高い点が特徴です[注1]。同じDRAMでも新しいプロセッサと組み合わせればより性能が伸びるように、ioDrive2のソフトウェアコントローラはコンピュータの計算能力やバス能力にあわせて性能向上が期待できるほか[注2]、ソフトウェア実装なゆえ性能向上や機能追加のアップデート余地もあります。

カットスルーアーキテクチャにより、「ストレージボトルネックがほぼないコンピュータ環境」が実現されます。従来、ストレージ律速となりプロセッサ性能を使い切れていなかったアプリケーションやミドルウェアが、プロセッサやネットワークの限界にあたるまで高速化し、ストレージがボトルネックとならなければ何がボトルネックとなるかまで明確に見えるようになるでしょう。

カード単体で冗長化を実現

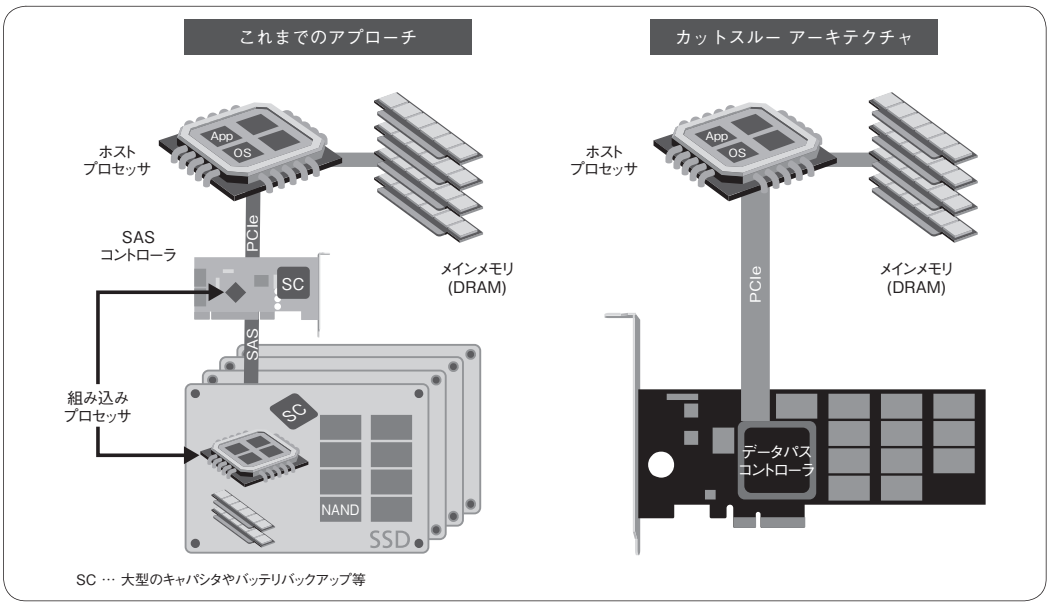
ioDriveは基板上に組み込みプロセッサやメモリなどが不要なため、製品自体がたいへんシンプルかつ故障が少ないデバイスです。しかし、記憶メディアであれNANDフラッシュ自体の信頼性はそれほど高くないため、カードレベルで耐障害性を確保しています。

仮にフラッシュメモリのパッケージが欠損しても、独自のAdaptive FlashBackと呼ばれる冗長化技術でカードが保護されており、自動的にセルフヒーリング機能が働きます。このため、通常は運用中に障害で交換したり、手動でリビルド作業する必要もありません(**図2**)し、内部が冗長化されたカードゆえSSDのようなRAIDによる障害対策はあまり意味がありません。

Adaptive FlashBackでは、n個のチャンネル

注1) Xeonの周りに低速なマイコンやメモリチップを並べる方法こそ、もったいないと思いませんか:-)
注2) 筆者の手もとのIvyBridge機では、書き込み15マイクロ秒の製品仕様のioDrive2が11マイクロ秒で動作しています。新型のサーバ用プロセッサの登場に期待がふくらみます。

▼図1 カットスルーアーキテクチャ



があった場合に、n-1個のデータチャンネルと1個のパリティチャンネルが構成されます。RAID5のディスクアレイと非常に似たイメージでデータ保護をしつつ、障害時にはセルフヒーリングがなされます(**図2**)。もしフラッシュメモリ上に障害が起きると、まず、パリティ情報から障害チャンネルのデータを復元し、正常なブロックに書き込みます。また、障害が発生したチャンネルを除いた構成でパリティを再計算、保存します。この結果、冗長化された状態へと復帰するため、デバイス単体で繰り返しの障害に耐えられます。容量が目減りしそうに見えますが、縮退は約4万分の1の単位で行われる上、オーバープロビジョニングされた領域から容量が補填されるため、デバイスの実効容量に影響はありません。

たとえ突然の電源断があったとしても、カード上の残存電力だけでデータパスコントローラが受け取ったデータはすべて書き込みを完了します。もともと遅延書き込みを行わない仕様のため、カード上の小型なコンデンサの電力だけで安全にシャットダウンできる設計です。カード単体で冗長化がなされているためRAIDコントローラのバッテリバックアップなども不要です。

余裕ある書き込み許容量設計

ioDriveは、書き込みに強いフラッシュストレージとして知られています。データベースのロギングなど、「書き込みを高速に処理できる」という意味だけでなく、書き込み続けても長期間に渡り利用できるだけの「書き込み許容量」が備わっています。Fusion-io製品で一番許容書き込み量が少ないMLCモデルでも、410GBあたり2PBの物理書き込み許容量があり、毎日1TBの書き込みが発生しても少なくとも5年以上持つ計算ですが、サーバ向けのioDrive2ではそれ以上の書き込みができます。このため、データベースなどの用途でFusion-io製のデバイスの寿命が数年で尽きるような事態は考えづらいでしょう[注3]。

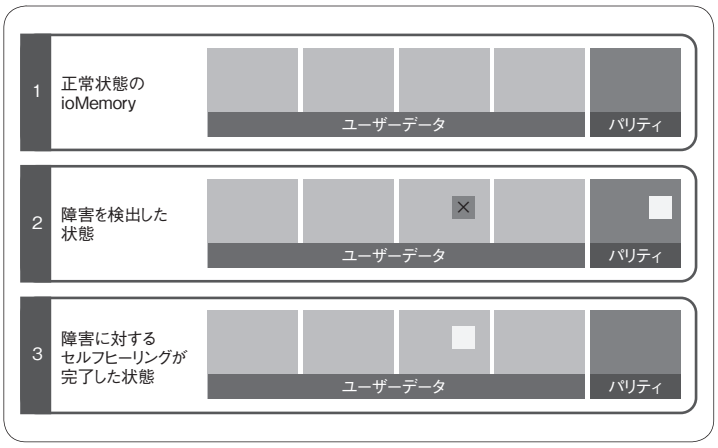
さまざまなモニタリング手法

ioDrive2の場合は、コマンドラインツールの表示により予備領域の何パーセントが残っているかを示すメディアステータスのほか、リード／ライトの総量などを定量的に把握できます。また、製品仕様として保証する書き込み量の何パーセントを消費したかなどの情報も表示されます。

ステータス情報はCUIツールから参照できるほか、SNMP、SMI、WMIといった管理インターフェース、また無償で利用提供のFusion-io

注3) 国内のソーシャルゲーム、コミュニケーションサービス事業者との打ち合わせ中に「書き込み量が多いのでそろそろ心配」とのコメントをいただくことがありますが、実際にカードの書き込み状況を確認しながら「あと10年以上大丈夫ですね」といった話になることがほとんどです。

▼図2　Adaptive FlashBackによるセルフヒーリング



製統合管理ツール「ioSphere」(**写真2**)を使えば、ブラウザ上で複数のホストに分散したioDrive2を集中管理できます。

充実した国内向け供給体制

フラッシュストレージ検討の際に気になるのが、購入後のアフターサポート体制でしょう。ioDriveは故障率がたいへん低いデバイスですが、万が一の故障も想定し、安心して導入できるかは重要なポイントです。

幸いにも、ioDriveはさまざまなサーバベンダからOEM供給体制が整っています(**表2**)。各社のサーバモデルとの組み合わせ動作が検証済みであるほか、各社の保守部品ネットワークや24時間オンサイトサービスなどの保守スキームでioDriveもカバーされる利点があります。また、OEMモデルによっては製品保守がサーバ保守費用範疇で提供されており、SANストレージなどのような年間保守費用が別途発生しないため、たいへんお得です。

▼写真2　ioSphere

▼表1　各ベンダのOEM供給体制

ベンダ	製品名
CISCO	Fusion ioDrive2 for C-Series Servers
HP	I/Oアクセラレータ
DELL	ioDrive2 Powered by Fusion-io
富士通	PCIe-SSD
日立	Fusion-io ioDrive2
IBM	HighIOPS PCIe Adapter
SUPERMICRO	Fusion-io ioDrive2
東京エレクトロンデバイス	Fusion-io ioDrive2

また、日本国内ではさまざまなホスティングサービスでioDrive搭載モデルを利用できます。ハードウェアを手許で持ちたくない、それでも1台あたりの性能を最大限に引き出したいという場合には、ioDrive搭載モデルのホスティングサービスを検討してはいかがでしょうか。

さまざまなioDrive2活用法

ioDriveは、国内向けの有名ソーシャルサイトや動画配信サイト、ソーシャルゲームなどで数十枚～数百枚単位で利用されていることをご存じの方も多いかと思いますが、今回はその他の事例や構成などを紹介します。

データベースの高速化

クラウドサービスを提供する某企業は、PostgreSQLによるデータベースを運用しています。このデータベースはサーバ2台のレプリケーション構成でしたが、日常のページ表示速度やVACCUM処理に時間がかかるなどの課題があったため、ストレージ部分だけをHDD RAID10構成からioDriveへアップグレードしました。DBサーバはOEM版ioDriveに対応したモデルだったため、お客様はOEM版ioDriveを追加で購入し、既存のサーバに搭載し、単純にデータベースの保存先をHDDからioDriveに移行しました(**図3**)。これによって性能問題が解決したほか、30分～1時間かかっていたVACCUM処理が5分程度で完了するようになったとのことです。少ない担当者で複数のシステムを運用されている都合、運用の負担がずいぶん楽になったようです。

日本ではMySQLのマスタ／スレーブ構成での利用例がWeb2.0系でよく知られていますが、ほかにはSQL Serverのレプリケーション構成での利用、Oracle Databaseの総コスト削減のためにご検討いただくケースが多くあります。その他、レプリケーション

機能付きHAソフトウェアでフラッシュストレージ向けのチューニングがされたものも出てきました。たとえば、サイオステクノロジーのLife KeeperとioDrive2を組み合わせると、サーバ内蔵フラッシュの性能を活かしながら冗長化できるため、性能と信頼性の両立が求められる企業向けアプリケーションで人気を博しており、公開事例も出てきています[注4]。

システムメモリの拡張として

フラッシュメモリの利点には容量単価の安さがあります。HDDと比べればやはり高いのですが、同量のDRAMを搭載した場合と比較すると容量単位あたりのコストは5分の1から半額程度になります(**表2**)。この特性を活かして、ioDrive2を大容量データのキャッシュとして利用するニーズが増えています。

HBaseには、高速なフラッシュデバイスをキャッシュ利用できる新機能、Bucket Cacheが登場しました[注5]。フラッシュデバイスをHBaseのDRAMバッファに続くキャッシュティアとして利用し、擬似的に大容量メモリを実現します(**図4**)。メモリティアに収まり切らないほどのデータ量があっても、ホットデータがDRAMとフラッシュメモリに自動的にキャッシュされるため運用も簡単です。Fusion-ioのBucket Cacheテスト結果では全DRAMヒット時の半分程度の性能が出ているようですので、性能あたりにかかるコストはおよそ半額になる計算です。

ストレージのキャッシュとしての利用

ioDrive2はサーバ内蔵型のフラッシュデバイスでありHDDやSSDの代替となる製品ですが、最近では仮想化環境での集約やストレージ性能のニーズから、高速ながらDRAMより安価なメモリティアとして、キャッシングに利用するケースが増えてきています。

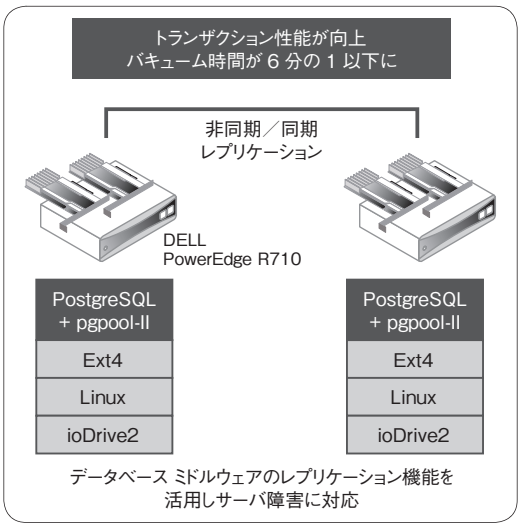
弊社が提供するキャッシングソフトウェアioTurbine(**図5**)はもともとVMware環境専用でしたが、バージョンアップに伴いWindows、Linuxの物理環境でも利用できるようになりました[注6]。仮想化環境のマシン集約のほか、データ量が膨大になりながらI/O傾向が偏りがちなシステムでioDriveをリードキャッシュとして使う(たとえばメール、グループウェア、ファイルサーバやデータベースなどの性能向上、ディスクアレイの負荷低減)ために利用されています。

SANストレージの代替として

ioNデータアクセラレータ(以後ioN)は、ioDrive2を使った、超高速・低価格な共有型フラッシュストレージを構築するためのソフトウェアです(**図6**)。ioNをインストールしたサーバはFibre Channel、InfiniBandやiSCSIのターゲットとして振る舞うため、Oracle RACのような

注4) 大日本印刷による採用事例 http://i.sios.com/news/press/lkdnp-20130827.html

▼図3　PostgreSQLをioDriveで高速化



▼表2　DRAMモジュールとioDrive2のバイト単価

サーバ向けDRAM(32GB×12)計384GB	600万円	1.56万円/GB
ioDrive2 OEM 365GB	参考価格100万円	0.27万円/GB

注5) Bucket Cache：A solution about CMS,Heap Fragment and Big Cache on HBASE：https://issues.apache.org/jira/browse/HBASE-7404

注6) ioTurbineはIBMよりFlashCache Storage Acceleratorとリブランドされたもの OEM版の提供が始まりました。サーバやOEM版カードと併せてお求めいただけます。

共有型ストレージのアーキテクチャを利用している際に、これまでの構成を維持したままioDrive化できます。

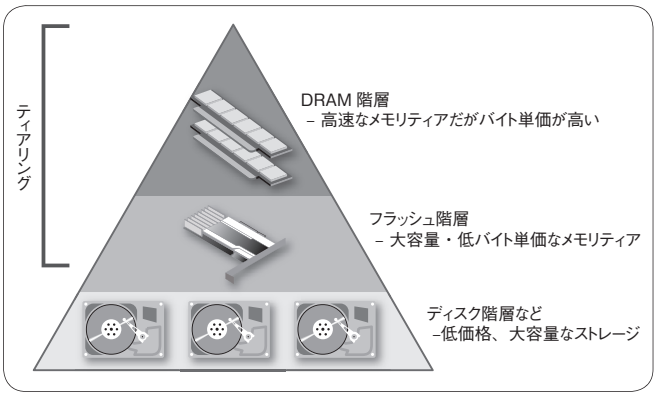
ioNはVMwareのHCL(互換性ガイド)にリストアップされています。Linuxカーネルと SCST[注7]の組み合わせに加えて、ioDrive専用ならではの強みとして性能のチューニングや相互運用性の強化やテストが行われており、ベンダのサーバ性能ベンチマークなどでもioNが活躍します[注8]。

ioDriveを活用しコストを削減しよう

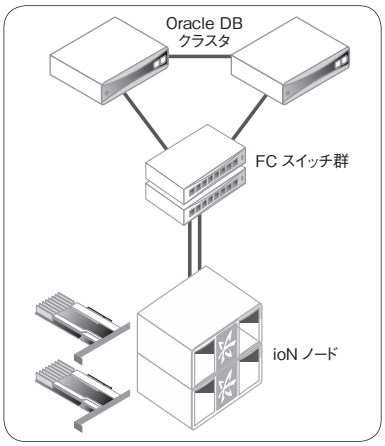
ioDrive2の導入をされるお客様は、性能が伸び悩んだり、性能を達成するために機材の導入費用／維持費用がかさんだり、メンテナンスやバックアップの運用に手間がかかったりといった悩みがあるときに、ioDrive2の性能・信頼性に加えて運用保守面など、トータルのバランスを見て決められる方がほとんどです。また、その規模は数百台単位のサーバクラスタから1〜2台のシステムまでさまざまです。「性能が足りない、足りなくなるかもしれない」「性能は足りているがサーバ台数が増え過ぎた」「アプリケーションをチューニングするにしても何人月かかるか……効果の度合いも不安だ」──そんなときは、ioDrive2を検討いただくときかもしれません。SD

注7) http://scst.sourceforge.net/
注8) 2013年、Fusion-ioはSCST開発者が多数在籍するid7を買収し、SCSTの商用サポートによるノウハウがioNにも適用されました。

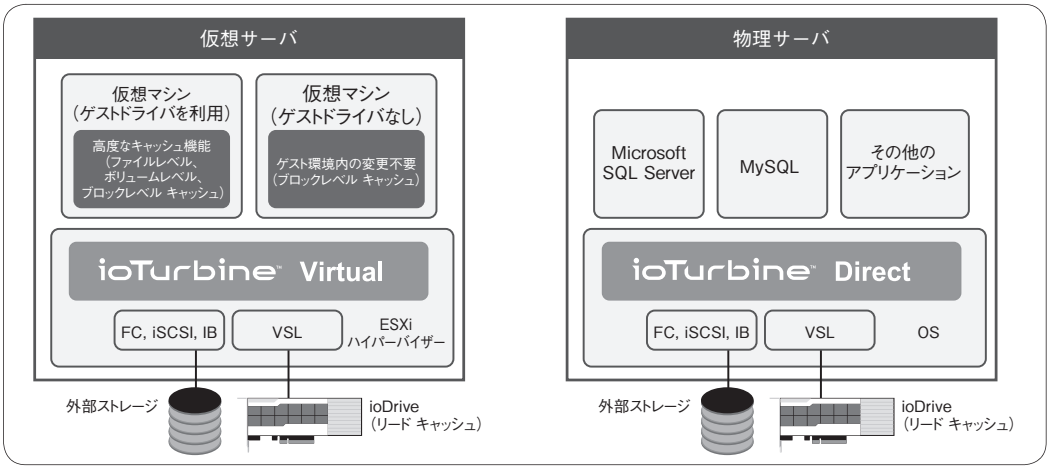
▼図4 DRAMとフラッシュメモリのティアリング



▼図6 ioN-SANストレージの代替として



▼図5 ストレージのキャッシュとしての利用



より効率的なフラッシュメモリ活用をめざして

第4章 OpenNVMの機能と可能性

長谷川 猛(はせがわ たけし)
フュージョンアイオー㈱ ソリューションアーキテクト
thasegawa@fusionio.com

フラッシュストレージ界隈の新たな動向として、OpenNVMというオープンソースプロジェクトの活動があります。本章では、当プロジェクトが提供している機能を紹介するとともに、どんな活用方法が考えられているかを解説します。

OpenNVMとは

OpenNVM[注1]は、ソフトウェアからフラッシュメモリをはじめとする不揮発メモリ(NVM)に対するアクセスインターフェースを構築するオープンソースプロジェクトです。本章ではOpenNVMの概要紹介、およびOpenNVMを通じてどのようなメリットが得られるようになるかを紹介しましょう。

本原稿の執筆時点では、同プロジェクトのリポジトリには**表1**のコンポーネントが含まれます。

ミドルウェアではMariaDBやPercona ServerがOpenNVMで想定するインターフェースを前提に、リリース版のソースコードやバイナリに不揮発メモリサポートが含まれた状態となっています。しかし、ハードウェア/ドライバの実装は(OpenNVMに実装はなく)ベンダからの提供となるため、現時点ではOpenNVM対応のデバイスやドライバをハードウェアベンダから個別に調達する必要があります[注2]。

NVMプリミティブ

現在、フラッシュデバイスへのアクセス方法はHDDではセクタのリード/ライト、シークを前提に設計されたブロックI/Oが基本ですが、フラッシュメモリの特性はHDDと違いますし、FTLはさらに高レベルなリクエストを受けられます。このリクエストを開発に利用するためのAPIがNVMプリミティブです(**表2**)。

注1) http://opennvm.github.io/

注2) Fusion-io ioDrive2でOpenNVMの機能を利用する場合は、Fusion-io(米国)の窓口へ連絡しOpenNVM対応版のVSLを入手する必要があります。現時点ではOEMベンダにて提供されていないバージョンのVSLが必要となるため、OEM版ioDrive2製品についてはOpenNVMを利用できるようになるまでもう少しかかりそうです。

▼表1 OpenNVMリポジトリ中のコンポーネント

コンポーネント	概要
NVMプリミティブ	ブロックインターフェースに加えFTLへの高度な操作をAPI定義
NVMKVライブラリ	NVMプリミティブ上で構築されたKey-Valueストアライブラリ
Flash-aware Linux Swap	不揮発メモリを前提としたスワップの拡張

▼表2 NVMプリミティブでできる主な操作

操作	説明
アトミックライト	複数のLBAに対してアトミックにデータを書き込む(マップする)
アトミックトリム	複数のLBAをTRIMする(アンマップする)
バッチ更新	アトミックライト、アトミックトリムをまとめて実行する
LBAのマップ状態取得	指定したLBAがマップ状態(書き込み済み)かどうかを取得する

NVMプリミティブはブロックI/O同様にLBA(Logical Block Address)を介してデータの書き込みを行ったり、FTLが管理するマップ状態を取得したりできます。アトミックライトはデータの整合性を保証しながら複数のLBAにまたがった更新を行える機能であり、データベースなどの書き込みロジックを大幅に単純化できます[注3]。TRIM操作は「書き込み済みのデータが不要である」とFTLに通知しガベージコレクションさせる操作として知られていますが[注4]、NVMプリミティブではさらに「LBAがマップされているか」という情報自体を開発者が活用することが可能です。

また、ioDrive2はデバイスレベルのスパースに対応しており、ioDrive2を実際より容量の大きなデバイスのように扱えます。

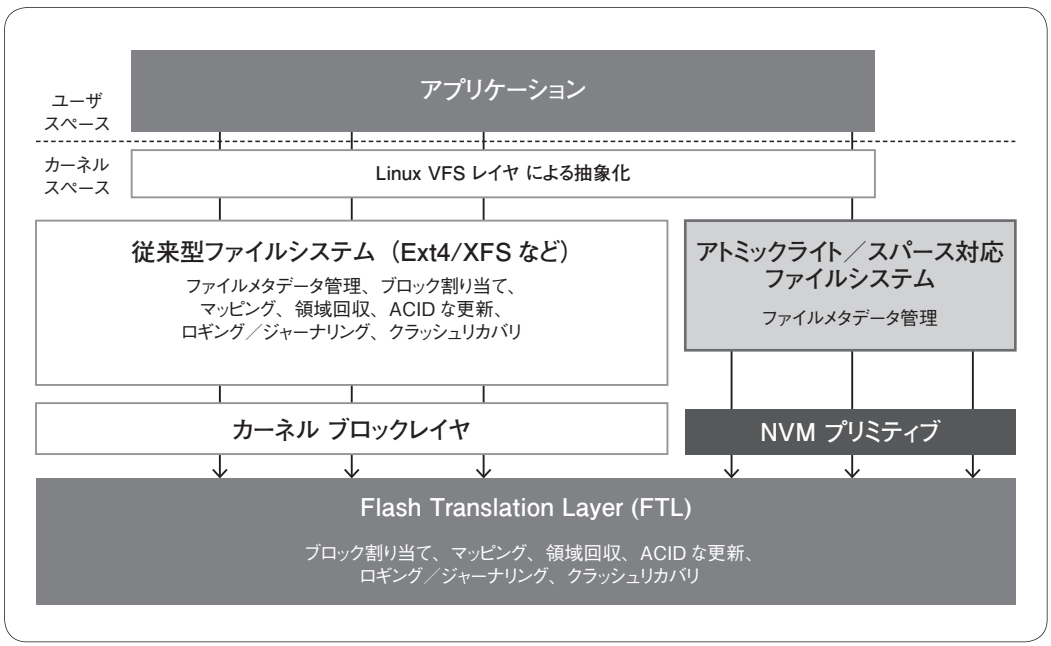
スパースのしくみを持たないHDDやブロックデバイスでは容量分のアドレス空間しか使えず、データを読み書きする際にはファイルシステムやミドルウェアによる論理アドレス(LBAのブロック)割り当てやリマッピングなど、複雑なロジックが必要でした。

しかし、このブロック割り当てやリマッピング処理はデバイス側でFTLも行っており、二重の実装となってしまいます。スパース機能を持つデバイスであれば、これらの処理をFTL側に任せられるため、従来のようなブロック管理のロジックが不要となり、コードのシンプル化やバグ発生のリスクの低減、効率アップや処理速度の向上が期待できます(**図1**)。

スパースモードでは、1.2TBの容量のioDrive2は144PBの超大容量デバイスに見え、実際の容量が溢れるまで、自由なLBAにデータを書き込みできる状態となります。たとえばファイルシステムをデザインする場合、iノード番号をもとにデータの保存先LBAを計算式で求められるようにする、といった設計が可能になります。

注3) アトミックライトについては、第5章で詳しく解説します。
注4) SSDではファイル削除時などに空き容量を回収するなど、将来の書き込みを高速化のためにTRIMが使われてきました。

▼図1 NVMプリミティブを利用したファイルシステムのデザイン



NVMKV――ネイティブKVSライブラリ

NVMKVはアトミックライトとスパースのしくみを組み合わせて作られた、Key-Valueストア(KVS)のライブラリです(**図2**)。先述のNVMプリミティブとスパース空間を活かし、キーのハッシュ値から求めたLBAにデータを保存するしくみとなっているため[注5]、GET/PUT操作ともにフラッシュデバイスに対して1IOPSの負荷しかかからず、デバイスの性能を活かした処理が可能です。

NVMKVはC言語向けのバインディングが提供されており、また、筆者はテスト用に独自のPerl向けバインディングを製作しています。ioDrive2と組み合わせたテストを手許で実施してみたところ、Perlで書いたスクリプトより60万OP/秒のPUT操作、10万OP/秒のGET操作ができており、デバイスの素のIOPS特性をそのまま反映した形となっています。また、NVMKVライブラリ自体はシステムメモリ上にデータをキャッシュしないので、上位レベルでキャッシュ処理を行うとさらに高速になるでしょう。

アトミックライトでInnoDBを高速化

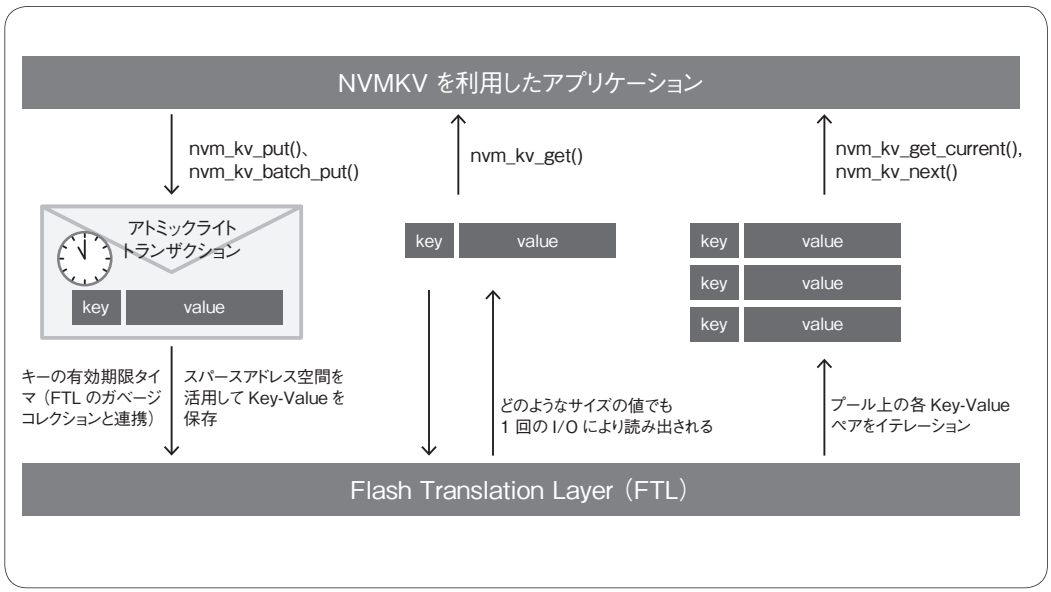
MySQLの亜種であるMariaDBやPercona Serverはアトミックライトに対応しており[注6]、ダブルライトなしでACIDの信頼性を維持できます。

従来、MySQLのストレージエンジンであるInnoDBは、更新時にデータ破損を起こさないようにするため、ダブルライトと呼ばれるアプローチをとってきました。まずメディア上のダブルライトバッファに「何を書き込むか」を書き込んでおき、その後にデータ領域を更新します。万が一、ミドルウェアやサーバが停止しデータ領域の更新が完了しなかった場合でも、ダブルライトバッファから書き込みデータの復元が可

注5) 従来型のKey-Valueストアであれば B-Treeなどを使ってデータブロックを管理する手法が一般的です。

注6) MariaDB本家のブログにて、アトミックライト機能とベンチマーク結果が紹介されています。
"MariaDB Introduces Atomic Writes" http://blog.mariadb.org/mariadb-introduces-atomic-writes/

▼図2 NVMKV



能なため、データ領域の論理破壊を防ぐことができます。

しかし、実際には、Ext4やXFSといったファイルシステムもジャーナリングなどによる保護を行っているため、効率がよくありません。アトミックライトを利用すると一度の書き込みで整合性を保証できるため、無駄な処理が減少しレイテンシ改善やスループット向上[注7]、フラッシュメモリへの書き込み量削減にもつながります(**図3**)。

InnoDBでアトミックライトを有効化するためには、OpenNVMのインターフェースを提供するストレージデバイス、アトミックライト対応ファイルシステム、そしてDBミドルウェアを組み合わせて利用します。アトミックライトに対応したファイルシステムとしてはFusion-io独自のものがあるほか、btrfsやExt4、XFSなどの既存ファイルシステムにアトミックライトを追加したり、Linuxカーネルのシステムコールでアトミックライトを標準化する提案などが行われているようです[注8]。

終わりに

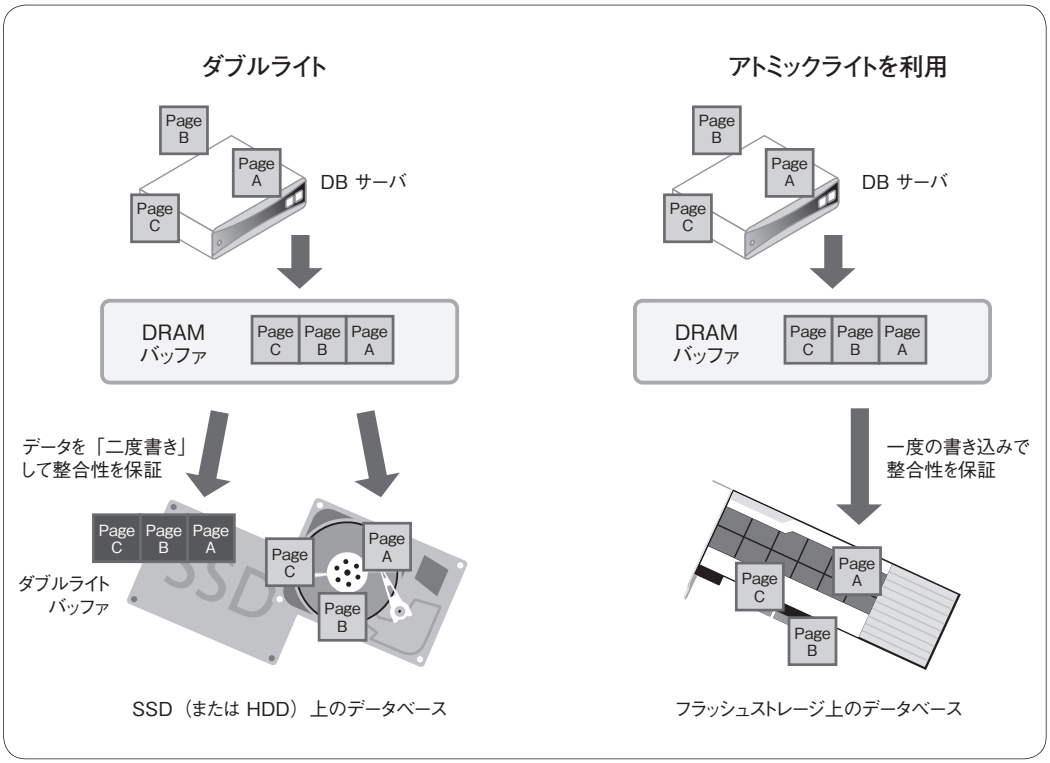
新たなインターフェースによりフラッシュメモリを利用できるようになると、ソフトウェアの開発が容易になったり性能が向上することを感じていただけましたでしょうか。

OpenNVMのWebサイト(http://opennvm.github.io)ではNVMプリミティブのAPIマニュアルや、NVMKVライブラリのコードが公開されています。興味をお持ちの方は、ぜひ上記サイトを訪れてみてください。SD

注7) インサート/アップデートクエリが多く書き込み性能ネックになりがちな利用環境などで、とくにテーブルサイズが大きくなってくるとアトミックライトの効果が発揮されるようです。

注8) "Atomic I/O operations" http://lwn.net/Articles/552095/

▼図3 データベースへのアトミックライト適用



ioDriveの性能をさらに引き出す

第5章 検証!アトミックライト+DFS

桑野 章弘(くわの あきひろ)　(株)サイバーエージェント
akihiro.kuwano@gmail.com　Twitter @kuwa_tw
URL http://d.hatena.ne.jp/akuwano/
長谷川 壮介(はせがわ そうすけ)　(株)サイバーエージェント

第4章でアトミックライトによりデータベースの高速化を図れることを紹介しましたが、どの程度のものなのか気になった方も多いのではないでしょうか。本章では、ioDriveにおけるアトミックライトの効果検証について報告します。

はじめに

はじめまして、筆者達は渋谷のサイバーエージェントという会社でサーバサイドエンジニアとして、おもにサーバの構築／運用などをしています。今回はioDriveについての特集ということで、筆者達が以前に行ったioDriveのアトミックライトとDirect File System(DFS)の検証について話をしたいと思います。

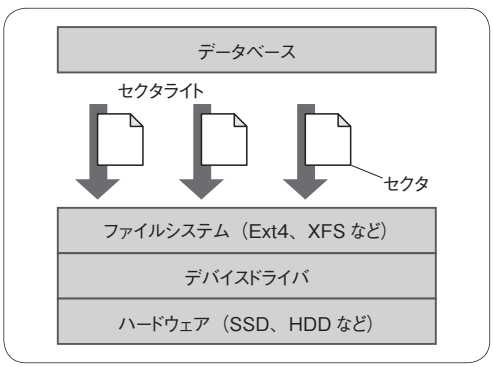
アトミックライトとは?

最初にアトミックライトってなんでしょう。初めて耳にした方もいらっしゃるかと思います。一言で言うとアトミックライトはioDriveの性能を限界まで引き出すしくみ、と考えれば良いと思います。

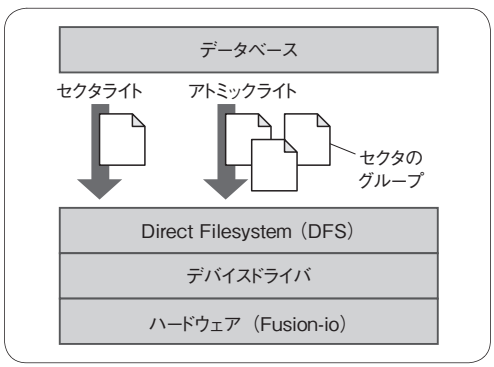
通常、ハードディスクを使う場合は、データは512バイトの「セクタ」の単位で更新されます(**図1**)。しかし、MySQLなどのソフトウェアはたくさんのセクタを更新しながら動作します。また、データ更新の途中で電源が落ちたりするとデータが壊れてしまいます。さらに、HDD、SSD、ioDriveなど、異なるハードウェア間で同じファイルシステムやソフトウェアが使えるしくみは便利ですが、その分オーバーヘッドも発生しており、ioDriveの性能を引き出せていないとも言えます。

そこで、ioDriveを使うときにアトミックライトを併用すると、ソフトウェアの書き込み要求をグループ化して発行できるため、オーバーヘッドを少なくすることができます(**図2**)。また、通常であればダブルライトで障害に備えるのです

▼図1　これまでの書き込み操作



▼図2　アトミックライトによる書き込み



が、アトミックライトを使えばioDrive自体がハードウェアレベルで書き込みの整合性を保証してくれるため、書き込みの整合性を気にする必要がなくなります。

DFS(Direct File System)

アトミックライトはいいことずくめのように聞こえますが、これまではとてもハードルが高い技術でした。なぜならアプリケーションをアトミックライトに対応させるためには、コード修正が必要なうえ、そもそもアトミックライトに対応したファイルシステムがなかったのです。

ですが、アトミックライトに対応したファイルシステムが登場したことで、状況が変わりました。Linux上から通常ファイルシステムとしてマウントしたり、データを載せられるようになったりと、試しやすくなりました。

ソフトウェアでの対応

現在、MariaDB 5.5.31やPercona Server 5.5.31-30.3以降のバージョンにおいて、標準でアトミックライトを使用できます。設定ファイルmy.cnfで、次の設定を入れると使用できます(MySQLにも同様のパッチがあるのでそれを当てることで使用可能)[注1]。

```
innodb_use_atomic_writes = 1
```

今回はDFS +アトミックライトの場合と、通常のファイルシステム+ダブルライトの場合でデータベースのパフォーマンス比較を行いました。

パフォーマンス比較

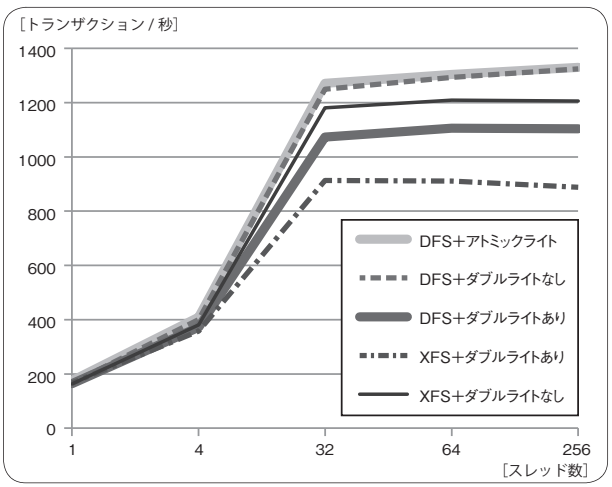
2Uのサーバに ioDrive2を搭載し、MariaDBを使用してテストを行いました。サーバの設定の細かい部分は割愛しますが、とくに次の2パターンを比較しました。これはアトミックライトを使用している場合はハードウェアレベルで書き込み整合性が保たれることから、書き込み量やデータ保証の条件を合わせて比較するためです。

①ioDrive＋DFS＋アトミックライト
②ioDrive＋XFS(ダブルライト)

sysbench——単純なクエリでの比較

最初は、sysbenchを使用して複雑なトランザクションなどのない単純なクエリのテストを行いました(**図3**)。アトミックライトを有効にすると、通常のXFS＋ダブルライトを使用した場合と比べ、約1.3倍以上の性能が得られています。

▼図3　sysbench結果



tpcc-mysql——複雑なクエリでの比較

次は、トランザクションがあるような複雑なクエリのテスト結果を紹介します(**図4**)。このテストにはtpcc-mysqlというTPC-C規格[注2]でベンチマークを取るソフトウェアを使用しました。実際のアプリケーションではいろいろなクエリが流れますので、こちらのほうが実環境に即していると考えられます。

注1) https://code.launchpad.net/~tmathiasen/mysql-server/mysql-5.5-fio

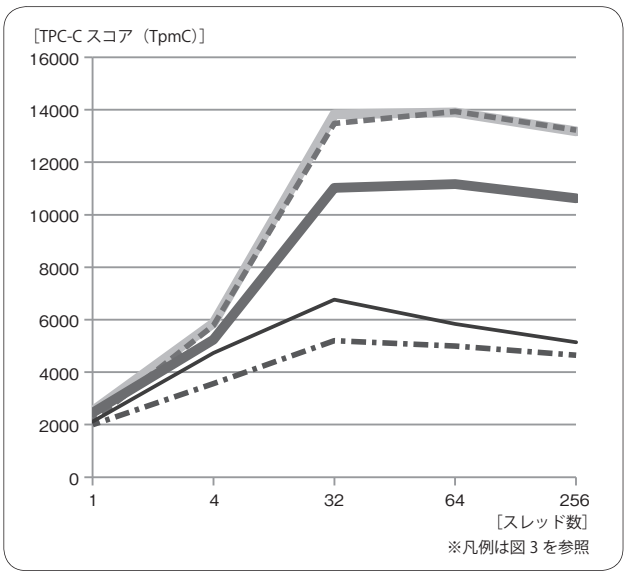
注2) 卸売業の注文、支払いなどの業務をモデルにしたトランザクションを実行した場合の性能測定をする方法。

アトミックライトを有効にすると、通常のXFS＋ダブルライトを使用した場合と比較し、4スレッドの場合で約1.6倍、32スレッド以上では約2.6倍以上の性能が得られています。単純なクエリの場合と比較して、アトミックライトとダブルライトの差が顕著になりました。理由としては書き込み量の差(後述)や、ロック時間の減少などが効いているのではないかと考えられます。

読み込み量／書き込み量比較

次に、データベースに同じ負荷をかけたときの読み込み量と書き込み量の比較を行いました。**図5、図6**を見るとわかるように、データの読み込み量は若干減少しており、書き込み量はアトミックライトの効果により半分以下になりました。書き込み量が半分になるということは、ioDriveの寿命が2倍に伸びたということになります。データ読み書き量の削減は、速度の確保と同時にioDriveの耐用年数に影響するため重要なポイントです。

▼図4　tpcc-mysql結果

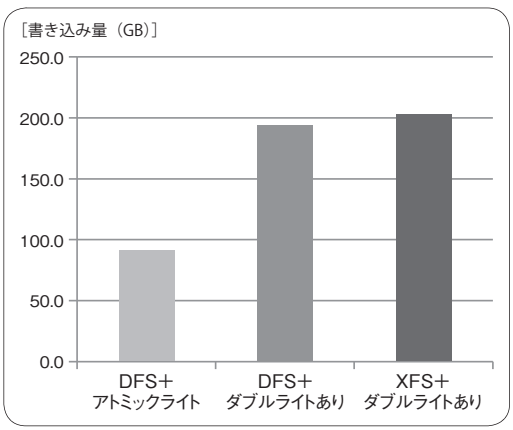


まとめ

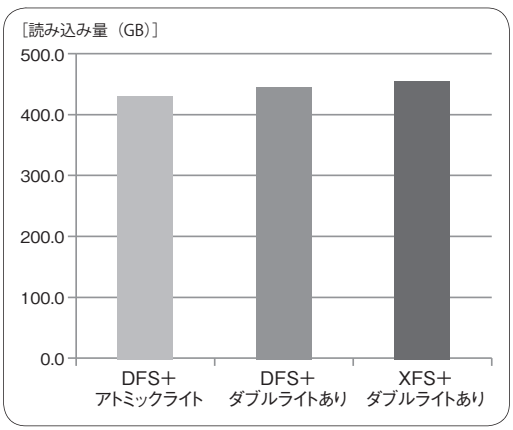
アトミックライトは速度向上に効果があると聞いていても、試すにはなかなかハードルが高い技術でした。しかし、DFSの登場やアトミックライト対応のソフトウェアがリリースされたことにより、ハードルが下がってきています。

今後、パフォーマンスがほしいデータベース環境の場合には検討の1つに入ってくると思いますし、パフォーマンスが上がるということはシステムのコストが下がるということでもあります。要件にマッチする場合には使用を検討してみるのも良いかと思っています。SD

▼図5　書き込み量比較



▼図6　読み込み量比較



次期LTSリリースをにらんだ、
野心的なマイルストーン

Ubuntu 13.10 “Saucy Salamander”

Ubuntu Japanese Team
長南 浩(ちょうなん ひろし)　chonan@progdence.co.jp

本記事では、2013年10月17日にリリースされたUbuntu 13.10“Saucy Salamander”での新機能と今後の方向性について説明します。

Ubuntu 13.10“Saucy Salamander”

Ubuntu 19回目のリリースとなる、13.10 “Saucy Salamander”が10月17日[注1]にリリースされました。今回のマスコットは「Salamander」[注2]で、サンショウウオと紹介されることの多い小動物です。

今回のリリースはLTS(長期サポート)リリースの合間の「standard」リリースで、14.04 LTSの直前のリリースです。前回のLTSからは1年半のタイミングで、通常であればLTSへ向けたブラッシュアップとしてのリリースとなる例が多いのですが、今回は12.10、13.04がそれほど大きな変化がなかった反動からか、比較的大きな変更が加えられました。とくにディスプレイサーバの変更は、長い間デスクトップUNIX/Linux環境で広く使われていた「X Window System」を置き換えるためのマイルストーンとなりそうです。

とはいえ強烈な変化は、おもにディスプレイサーバとIBus回りに集中しています。ディスプレイサーバの変更は単にXから脱却するだけでなく、スマートフォンやタブレットを統合するための布石としての意味を持つものとなっています。

サポート期間は9ヵ月

13.04から、LTSではない「standard」リリースのサポート期間は9ヵ月へ短縮されました。13.10でもこの方針通り9ヵ月のサポート期間が設定されています。

サポート期間の変更が行われる前にリリースされた12.10は、2014年1月にサポート期間が終了しますが、同じ月に13.04もサポート終了となります。その後「standard」リリースは9ヵ月のサポート期間となるので、LTSリリースに加えて1〜2個の「standard」リリースが同時にサポートされることとなります。

「standard」リリースを使う場合には、次のリ

▼表1　現在サポートされているリリースとサポート終了時期

リリース	サポート終了時期
13.04 Raring Ringtail	2014年1月
12.10 Quantal Quetzal	2014年1月
13.10 Saucy Salamandar	2014年7月
10.04.4LTS Lucid Lynx	2015年(Serverのみ)
12.04.2LTS Precise Pangolin	2017年

注1) 執筆時点ではリリースされていないので、開発中の「Daily Build」やその環境上での動作を基に執筆しています。
注2) サンショウウオだけでなくイモリの仲間も含めて陸に棲息する傾向の強いものがサラマンダーと呼ばれます。「Saucy」は「気の利いた」「しゃれた」「生意気な」「ずうずうしい」という意味があります。

リースに必ずアップデートする必要があるので、サーバなどの用途に使う際には注意が必要です(**表1**)。

Ubuntu 13.10の新機能

それではUbuntu 13.10での新機能を紹介していきましょう。デスクトップ環境では、もはやUbuntuの特徴と言っても過言ではないUnityのブラッシュアップと、日本語を扱うユーザにとっては影響が大きいIBusの変更が大きな変化になりました。

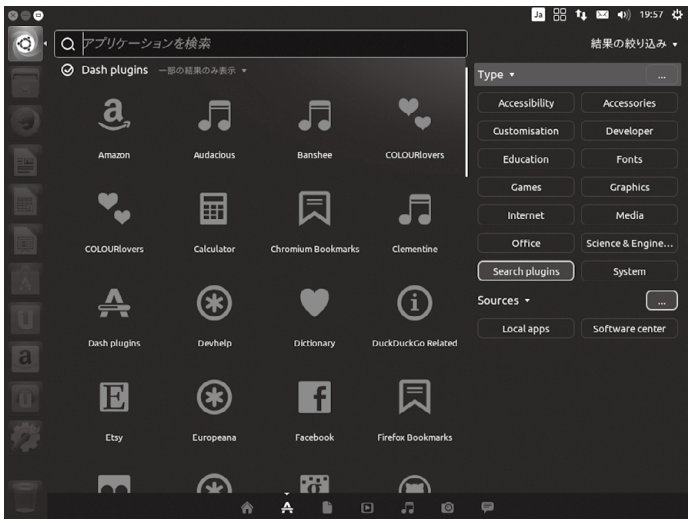
インターフェースの変更

■ Unity 7.1

Ubuntu 13.10上のUnityが7.1系列にバージョンアップされました。このバージョンでは13.04で間に合わなかった「Smart Scopes」が実装され、「Home Dash」でも検索結果をカテゴリや情報源別に絞り込む「フィルタ」が利用できるようになりました(**図1**)。また、「アプリケーション」ではプラグインを検索できるようになっています(**図2**)。

▼図1　「ホームdash」でもカテゴリや情報源による検索結果の絞込ができるようになった

▼図2　利用できる「dashプラグイン」も検索可能となった



■ IBus1.5

AnthyやMozcなどのインプットメソッドを管理するIBusが、1.5系列にアップデートされました。これに伴って、キーボード回りの設定画面が更新され、設定画面がシンプルになりました。

しかし、13.04で使われていたIBus 1.4とは大きく概念が変わり、「漢字変換機能を持った入力ソース」に切り替えるという考え方となりました(**図3**)。

プログラマが行う定番のキーボードカスタマイズの「CapsLockとCtrlの入れ替え」はibus-anthyの設定画面、すなわち「入力ソースごとに」行うようになりました(**図4**)。

そのようなIBus 1.5ですがAnthyはともかくとして、日本語入力にMozcを使う場合には「Mozcツール」を起動する方法がないなどの問題が生じること

がわかっています[注4]。

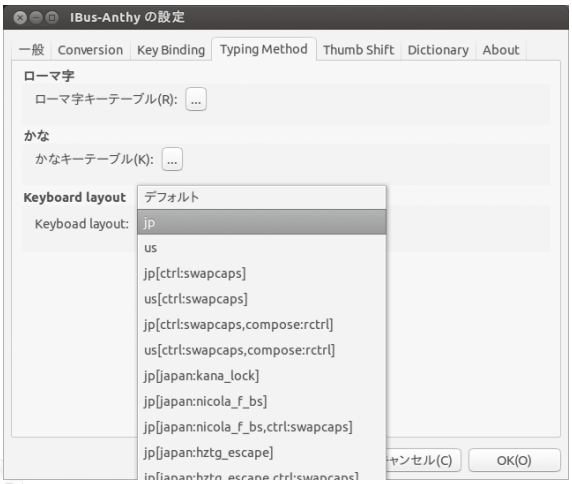
執筆時点では、本誌でもお馴染みの、あわしろいくや氏を中心にUbuntu Japanese Teamメンバで対処法を検討しているところですが、具体的にどのような方法でこの問題を解決すべきかは決まっていません。通常の方法での修正はリリーススケジュール的に間に合わないため、Japanese Remixで対策を講じる方向で議論が進んでいます。

注4）IBus 1.5が採用される前まではJapanese Team内部で日本語のインプットメソッドをMozcに変更する提案を出したらどうかという議論を行っていました。今回のリリースでユーザが一番影響が大きそうな部分と言えるかもしれません。

▼図3　iBus1.5での「入力ソース」選択画面

※MacintoshやOADG 109Aの表記に並んでAnthyが並んでいる。インプットメソッドとハードウェアキーボードの種類が同列に扱われるようになった。

▼図4　Ctrlキーや CapsLockキーなどの設定はIBus-Anthyの設定で行うことになった[注3]



注3）余談だが筆者が頻繁に使う“ctrl:nocaps”の設定は用意されていなかった。

ここ最近は、Japanese Remixなしでも実用の場面ではそれほど困ることがない状態が続いていましたが、13.10に関してはJapanese Remixが重要になるリリースとなることが予想されます。Japanese Remixは、Ubuntu本体のリリースに続いて1〜2週間ほどでリリースされるのが通例ですので、リリース情報には注意が必要です。

■デフォルトアプリケーション

デフォルトアプリケーションについてはデフォルトのWebブラウザをChromiumに変更することが議論されましたが、今回も結局Firefoxがデフォルトとなることに落ち着きました。過去に何度か変更された音楽プレーヤも13.04同様Rhythmboxが採用されています。

また、今年5月に一部メディアで話題になったMySQLの代替プロダクトについても13.10では具体的な変更は行われませんでした。

その他のアプリケーションについても順当にバージョンアップを重ねている状況です。

コア部分の変更

■カーネル

Ubuntu 13.10では、Linux Kernel 3.11に独自の変更が加えられたカーネルを利用します。サーバ向けARM SoCのECX-1000/2000や、OEM機種のサポートや各種デバイスドライバ向けのパッチ、AppArmorの調整などが加えられています[注5]。

また、Ubuntu Touch向けには、3.4.0ベースのカーネルが別に準備されています。

■ディスプレイサーバの変更

今回のリリースで最大の変更はディスプレイサーバの変更です。Ubuntuでは2010年11月に創始者のMark Shuttleworthが、自身のブログに記事[注6]を書いて以来、Xに代わるディスプレイサーバの実装が進めら

注5）https://wiki.ubuntu.com/KernelTeam/Specs/SaucyKernelDeltaReview
注6）http://www.markshuttleworth.com/archives/551

れてきました。当初はWaylandを使った実装が進められていましたが、紆余曲折を経て最終的にMirを採用することになりました。

13.10では、「Mir/XMirがデフォルトとなり、Mirが動作しないハードウェアでは従来のXを使ったセッションがフォールバックとして使用される」というロードマップが示されていますが、執筆時の状況では「Xセッションがデフォルト」という状況でした。リリースまでにこの挙動は変更される可能性がありますが、

```
$ sudo apt-get update
$ sudo apt-get install mir-demos unity-system-compositor
```

として再起動することで、Mir/XMirを試すことができます。とはいっても、見た目上の変化はほとんどなく、

```
$ ps ax | grep "unity"
```

とした際にunity-system-compositorプロセスが存在していればMir/XMir環境で動作していることになります。また、執筆時の段階ではマルチモニタサポート(XRANDR)が不十分だという状況もあるので注意が必要です。

また、LubuntuやXubuntuなどの「Ubuntu Flavours」では13.10でのMir/XMir採用を見送っているところが多く、13.10でMir/XMir付きでリリースされるのはUbuntuのみとなる見込みです。

Mir/XMirは今後14.04LTSでドライバの整備を完了させ、フォールバックXセッションの削除、14.10以降でXMirからMirへの移行を行い、既存のXアプリはMir上のルートレスウィンドウで動作させるというロードマップが示されています。しかし、ハードウェアベンダとの交渉や外部プロジェクトとの連携をどう進めていくのかという課題が残る状況です[注7]。

■ Upstart 1.10

起動時の処理を行うinitデーモンとしてUbuntuが採用しているUpstartは1.10にアップグレードされました。デーモン起動時には緩いAppArmorプロファイルを使い、起動後により厳しいAppArmorプロファイルに切り替えるといったことができるようになりました。

また、initプロセスやユーザセッションとの連携を容易にするlibupstart、dconfやgsettingsによるGNOMEの設定変更が行われたときにイベント発呼を行うupstart-dconf-bridge、D-Busシグナルでイベント発呼を行うupstart-dbus-bridge、UNIXソケット経由でイベント発呼を行うupstart-local-bridgeが追加されました。

その他の話題

インストール時の留意点

インストール、アップグレード方法は従来のリリースとほぼ同様の手順です。GUIでのインストーラ(Ubiquiy)にはUbuntu Oneへのログイン画面が追加されました(**図5**)が、Ubuntu Oneアカウントの設定は必須ではありません。

また、13.04では「Windows 8が動作するUEFIマシン」の場合にUbuntu Desktopの64bit版が推奨されていましたが[注8]、13.10では64bit版のUbuntu Desktopが推奨される条件が変更され、デフォルトでダウンロードされるISOイメージファイルが変更される可能性があります。この話題は8月に開催されたvUDS[注9]でも話題とされた[注10]項目ですが実際にどうなるのかは13.10がダウンロードできるようになるまで

注7) GNOMEではWaylandへの対応が進められています。MirもWaylandもXを置き換えるディスプレイサーバとして開発が進められていますが、今のところ両者が競合している状況です。

注8) Ubuntu Serverはすでに64bit版が推奨バージョンとなっています。

注9) Ubuntu Developer Summitの略。Ubuntu開発の起点となるイベントでオフラインで行われていましたが、現在はGoogle+のHangoutを使ったオンライン会議になっています。オンラインのUDSをvirtualの頭文字を付けvUDSと呼んでいます。

注10) https://blueprints.launchpad.net/ubuntu/+spec/client-s-32v64-bit

わかりません。

13.04での条件か、もしくはインストール対象マシンのメモリ搭載量が4GB以上あるような場合には64bit版を選ぶのが良いでしょう。

慎重なインストール／アップデート計画を

また、インストールに際しては予期しない不具合にあたる可能性があります。とくに今回のリリースは大きな変更が入っているため、リリースノートに十分目を通して、バックアップやテスト環境を作ってテストしたあとに移行するのがお勧めです。

とくに現在12.04LTSをサーバ用途などに使っていてとくに不満を感じていない場合、次のLTS(14.04LTS)がリリースされるまでアップグレードを見合わせるのも有力な選択肢です。繰り返しになりますがサポート期間が9ヵ月であることは、「少なくとも1年以内に必ずアップデートしないといけない」ことを示しています。

Unity 8のテスト環境

13.10ではUnity 7.1が採用されていることはすでに述べましたが、その一方で、「Ubuntu Touch」向けもしくは次期デスクトップ向けの「Unity 8」のデモ環境が13.10で利用できます。13.10の環境で

```
$ sudo apt-get install unity8
```

として必要なパッケージをインストールしたあとに、

```
$ export UBUNTU_ICON_THEME=ubuntu-mobile
$ unity8 -mousetouch
```

とすると、Qt/QMLベースのUnity 8をテストできます(**図6、7**)。現在では基本的な画面の遷移と、ウィンドウサイズを横に広げたときに自動的にタブレット用のUI/UXに切り替わる様子を確認できます。

このテスト環境は画面上に「EARLY ALPHA NOT READY FOR USE」という文字が表示されているとおり、開発の初期段階であるものの、Ubuntu Touchが動作するデバイスを用意しなくても気軽にUnity 8を体験できる状態になっています。

SteamとMir

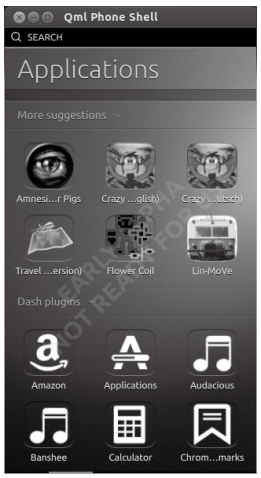
Ubuntuでのディスプレイサーバが変更されるという話題は早くも各方面で議論になっているようです。

2013年2月に正式版がValveからリリースさ

▼図5　デスクトップ版インストール時にUbuntu Oneのアカウント情報が質問される

※インストールするために必須の項目ではないので無理にアカウントを作る必要はない。

▼図6　Unity 8を使ってみた



※開発の初期であることを主張している背景の文字が表示される。

れたゲーム配信システムの「Steam」でのフォーラムには「UbuntuがMirに完全移行したあとはどうなるのか」というトピックが立てられ話題を呼んでいます[注11]。冷静に考えると、Steamの推奨環境は12.04LTSもしくは12.10ですので、2017年までは推奨OS環境がサポートされることになっていますが、ディスプレイサーバの変更はこういった方面にも影響を及ぼし得るものであることを知らされました。

Ubuntu 14.04 LTSへの見通し

Bug#1のクローズ

Ubuntuのバグトラッキングシステムの最初に登録された、「デスクトップPCの市場でMicrosoftが大多数の市場シェアを占めている」という「Bug#1」が、iOSやAndroidを搭載するデバイスの普及やクラウドコンピューティング技術の隆盛といったコンピューティング環境の変化を理由にMark Shuttleworth自らの手で5月にクローズされました[注12]。

このことはUbuntu の目標である「誰にでも使いやすい最新かつ安定したOSを提供すること」の対象を、これまでのデスクトップPCやサーバから組込みデバイスやスマートフォン、タブレットにも広げていこうとする意思表示ととることができます。

Linuxディストリビューションから総合コンピューティング環境へ

今回のリリースは、タブレット、スマートフォン、TVなどといったデバイスと従来のPC上でのデスクトップ環境をすべてUnityに統合しようという意思が現れたリリースとなりました。

次期バージョンのUnity 8は現在スマートフォンとタブレット向けの「Ubuntu Touch」上で動作するものが開発されていますが、13.10から14.10にかけて、Unity 8にデスクトップPC向け機能が実装される予定になっています。

また、明言されているわけではありませんが、この時期に開発者向けではない、製品としてのUbuntu Touchがリリースされることが予測されます。

Ubuntuはどこまでその輪を広げられるのか

Ubuntu Touchを含んだUnity向けのアプリケーション開発環境も、Ubuntu SDKやQtCreatorを使いQt/QMLベースのアプリケーションが開発されていくことが期待されます。

現在のiOSやAndroid開発と同じような規模で、多くの技術者がUbuntu向けにアプリケーションを書く時代がやってくるのかどうかは今のところ想像できませんが、サーバからモバイルデバイスまでのすべてのデバイスをUbuntuという1つのOSでカバーできる日がもしかしたらやってくるかもしれません。

次の14.04 LTSは、この目標がどの程度まで実現されるのか、「PC向けLinuxディストリビューション」という枠から飛び出してしまうのか、デスクトップと組み込みデバイスの差をどう埋めていくのかを占うリリースとなりそうです。SD

注11) http://steamcommunity.com/app/221410/discussions/0/864961629629332051/
注12) https://bugs.launchpad.net/ubuntu/+bug/1/comments/1834

▼図7　Unity 8のウィンドウを横に広げると、タブレットでのレイアウトが表示される

マニュアルどおりでホントに使える?

小規模プロジェクト現場から学ぶJenkins活用

最終回 日頃のメンテも大事です

最終回となる今回は、Jenkinsの引っ越しとバックアップについてのお話。そしてネットワークドライブなどを使う場合に役立つ、1台でもできるスレーブの使い方を紹介します。

嶋崎 聡(しまざき さとし)(株)XVI Twitter @sato_c

PCが壊れた?

本稿を書くほんの少し前の話になりますが、プロジェクトで利用しているJenkinsサーバの調子が悪くなりました。JOB実行中に反応しなくなったり、長くても20分で終わるはずの成果物コピーが3時間かかったりと普段と違う挙動が続きました。ほかにも電源を入れたら、通常のWindowsではなくスタートアップ修復ばかり起動する症状もあり、まともに使える状態ではなくなりました。

ちょうどこの連載でバックアップについて書こうと思ってはいたのですが、バックアップすらしてないこんなときにトラブルとかタイミング良すぎるだろ……と思いつつPCのカバーを開けて起動してみました。どうやら起動前にHDDの起動が失敗している感じ。調べた結果見事、HDDが壊れてかけていることが判明しました。これはちょっと……。

プロジェクトも始まったばかりでJenkins環境も作ったばかり。バックアップは後回しでもいいか……と考えていたところでした。こういうときだけ「転ばぬ先のバックアップ」とかいう言葉が頭をよぎりましたが、壊れてから言っても仕方ありません。壊れかけたHDDを別のWindows環境につないでみたところ、どうにか中身は見られたので、ファイルを移動することで環境の修復を試みました。なお、今回の話もWindows OSをもとに解説をします。

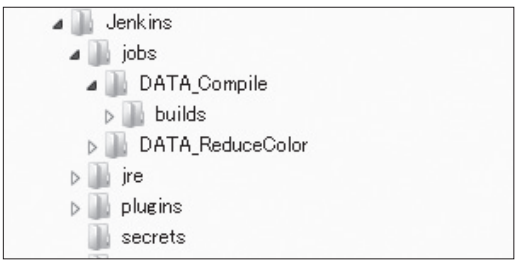
HDDが壊れてしまった(交換する)場合

今回の筆者の例のように壊れたHDDを交換する場合は、Jenkinsフォルダの修復の前に、新調したストレージにOSやJenkinsのビルドに必要なツールなどを一通りインストールする作業を行いましょう(当たり前ですね)。Jenkinsが動く環境ができたら、JOBを復元するための作業に移ります。このとき、元のデータにアクセスできなかった場合は環境を1から作り直すしかありません。筆者の場合はフォルダを開くまでしばらく待てばなんとかアクセスできるような状況だったので、壊れかけたHDDからJOBに関するファイルを救出していきました。

●JOB設定ファイルのコピー

JOB保存フォルダのデフォルトは、Jenkinsがインストールされているフォルダにある「jobs」フォルダです(**図1**)。この下に各JOBごとにデータフォルダがJOBの数だけあります。コピーするHDDにも念のため同じようにJOB名でフォルダを作って、

●**図1 フォルダ構成**



そこにデータフォルダにあるconfig.xml（以下、JOB設定ファイル）をコピーしておきます。最低限このファイルだけあれば大丈夫です。

言わずもがなとは思いますが、壊れたHDDからファイルをコピーする先は、一時的なものでいいので、別のHDDやUSBメモリなどにしてください。ネットワークにつながっている環境であれば、オンラインストレージ（Dropboxなど）を使ってもいいでしょう。

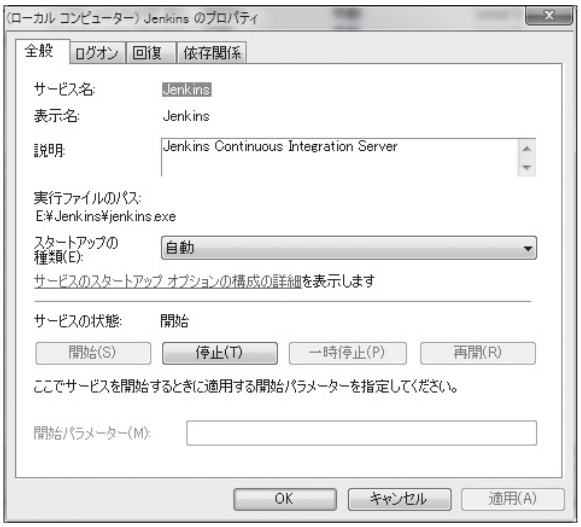
●引っ越し先の準備

新しいHDDが準備できたら、先ほどコピーしたJOB設定ファイルを新しいHDDにコピーしましょう。引っ越し先のJenkinsのjobsフォルダに、先ほど作った各JOBのJOB設定ファイルが入ったJOB名フォルダごとコピーします。このとき、サービスで起動しているJenkinsを止めておくこと（**図2**）を忘れないでください。

コピーが終わったら、サービスからJenkinsを起動します。正しくコピーされていれば、ダッシュボードが表示されます。

もし、Jenkinsのバージョンがこれまで運用していたものと新規のものとで差があったり、必要なプラグインがインストールされていない場合には、Jenkinsの設定画面を開いたときに警告が出ることがあります（**図3**）。ここで［管理］ボタンを押すと、どういったエラーが出たのか、どうすればいいかなどがまとめられたページが表示されます（**図4**）。プラグインがインストールされていないのであれば、あとでプラグインをインストールし直してから、再

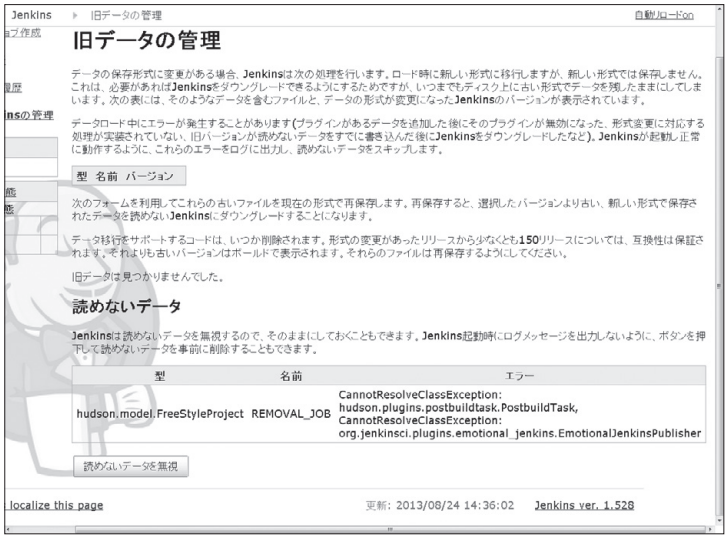
●図2　サービスからJenkinsを停止する



●図3　古い設定ファイルを読むと管理画面で警告が出る

度JOBを設定すれば大丈夫です。Jenkins本体のバージョン違いについては自動的に新しい形式にコンバートされて読み込まれますし、読み込めないデータは無理に読まないようになっていますので、Jenkinsが起動したら、再度JOB設定を行えばいいでしょう。

●図4　エラー、警告についての画面



●JOBの編集

とくにエラーもなく起動したら、引っ越したJOBのフォルダを開いてください。Jenkins起動時にJOBの実行に必要なファイルやフォルダが作られています。ファイルが作成されていない場合はJOB設定画面を開いて、そのまま[保存]ボタンを押してください。JOBの実行にはworkspaceも必要になりますが、こちらはSubversionなどのバージョン管理システム(VCS)を設定していれば、JOB実行時にその時点の最新状態に更新されます。

次は、通常どおりJOBを起動します。ログを見て、JOBが一通り実行されているかを確かめましょう。インストールできていないツールなどがないか、ファイルが正しく取れているかなどをチェックしていきます。JOB実行中のエラーで多いのは元の環境に存在していたツールがないことです。元々インストールする必要があるツールをチェックできていれば問題はないですが、エラーが出た場合はコンソールログを順番に確認して必要なものをインストールするなど、正しく動く環境を作っていきましょう。

HDDを増設する場合

HDDを増やして、ファイルを新しいHDDへ移動する場合はもう少し簡単です。まず、サービスとして起動しているJenkinsを停止して、次にJenkinsフォルダをそのまま新しいドライブにコピーします。ドライブ名が変わってしまうときはJenkinsを新しいドライブのJenkinsフォルダにインストールするか、今までのドライブのフォルダとしてマウントします。

今までのドライブのフォルダにマウントする場合は、「コンピュータの管理」にある「ディスクの管理[注1]」からマウントするか、**mklink**コマンドを使ってシンボリックリンクかジャンクションを作成します。たとえば、Eドライブの Jenkinsフォルダのシンボリックリンクを Cドライブに作成するには、次のようにします。

```
mklink /d c:¥jenkins e:¥jenkins
```

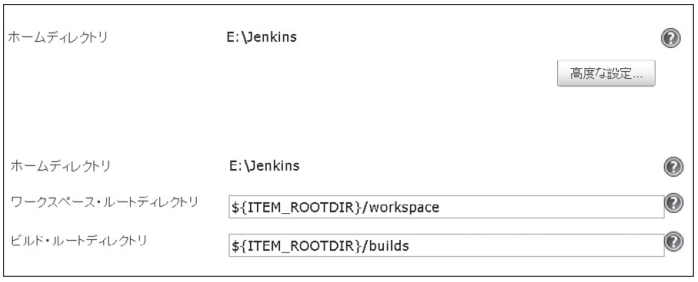
また、Jenkinsの「管理」からワークスペース・ルートディレクトリやビルド・ルートディレクトリを変更(**図5**)すれば、データの保管場所のみを変更できます。

バックアップとは言うけれど

次に、安心環境を作るための普段のバックアップについてです。バックアップが必要だということは

注1)[コントロールパネル]→[システムとセキュリティ]→[管理ツール]→[コンピュータの管理]→[記憶域]→[ディスクの管理]

●図5　Jenkinsのデータフォルダを個別に設定する



よく言われています。しかし、いざバックアップとなるとその量にうんざりする人もいるのではないでしょうか。そういう人は、引っ越しの項で書いたJOB設定ファイルだけでも定期的にバックアップしていきましょう。そうすれば、いざというときにJOBの動作だけでも復旧できます。VCSでJOBに関するファイルを管理していれば、より復旧作業も楽になるでしょう。

すべてのファイルをバックアップしておいてくださいと頼まれている場合は、これだけだとダメと言われるかもしれません。すべてのデータをバックアップする場合に手っ取り早いのは、Jenkinsをいったん止めて、Jenkinsフォルダ以下のすべてのファイルを別の場所（HDDやネットワークドライブなど）にコピーしてしまうことでしょう。差分バックアップを行いたい場合については今回は詳しく書きませんが、Windowsバックアップを使ってドライブの完全なバックアップをとって、あとから差分バックアップを繰り返す方法もあります。

ファイルの場所と定期実行

JOBフォルダには直近のJOBに関する情報が保存された「builds」フォルダ、JOB番号を記載した「nextBuildNumber」ファイルなどがあります。buildsフォルダは成功／失敗にかかわらずビルドに関する情報を保存していますので、JOB設定で定期的に削除する設定を行わない限り、すべて蓄積されています。ビルド情報はフォルダ名が日付になっています。ビルド番号ごとのフォルダやlastSuccessfulBuildやlastFailedBuildといった最新の成功／失敗ビルドは、この日付の付いたフォルダの別名フォルダとして作られていきます。ですから、これだけのフォルダすべてを保存するとなると結構な容量です。どうしても完全なバックアップで残したいということであれば、容量と時間が自由になるという条件はありますが、別のHDDやファイルサーバに領域を用意して、そこにすべてをコピーするのはどうでしょうか。

バックアップがとれれば、あとはアーカイブするなども別のPCからできますし、Jenkinsを止めている時間を短くできます。今回は一例としてWindows 7に標準でついている「ROBOCOPY」でフォルダをミラーリングする方法を紹介します。

●ROBOCOPY

ROBOCOPYはコマンドプロンプトで次のように起動します。

```
> robocopy
```

このように何もパラメータを指定しなければ、簡単な説明が出ます。最低限必要なパラメータはコピー元とコピー先です。単純にドライブ間でミラーリングなら、例に出てくる/MIRオプションを付けるだけでできます。**リスト1**のようなバッチファイルを作って、AT[注2]で定期実行すれば、自動バックアップのようなこともできます。

●リスト1　バッチファイルの例

```
001 : curl -o NUL http://[Jenkins の URL とポート ]/quietDown
002 : robocopy "Jenkins のフォルダ " " バックアップ先 " /MIR
003 : curl -o NUL http://[Jenkins の URL とポート ]/cancelQuietDown
```

注2）モデムのコマンドではなく、Windowsのタスクスケジューラをコマンドプロンプトから設定するためのコマンド。

JOBが走っていない時間帯しか使えませんが、例ではJenkinsをシャットダウンモードに移行してJOBが動いていない状態を作り、ROBOCOPYでファイルをコピーしてから通常モードに復帰させます。ROBOCOPYはまとめてコピーする用途に合うよう、マルチスレッドでファイルコピーを行うオプションもあります。詳細はROBOCOPYのヘルプにありますので、そちらを見ながら環境に合わせていろいろと設定してください。

あとはこうしてミラーリングしたデータを圧縮するなりして保存してください。シャットダウンモードで本当にJOBが動いていない状態か不安がある場合は、自動実行されていない時間にバックアップを行うか、手動でサービスを止めてからバックアップを行いましょう。

ネットワークドライブへバックアップできるようにする

Windowsのサービスで起動しているJenkinsは、ネットワークドライブにドライブを割り当てたドライブ名でアクセスできずエラーになります。Windowsの仕様なので仕方ないのですが、ネットワークドライブを成果物の保存やデータのコピー先に使いたいときもあります。これを実現するためには、Jenkinsが動いているPC上で(Jenkinsの)スレーブを起動すれば、JOBからネットワークドライブへアクセスできるようになります。

スレーブを起動してみよう

この連載では、単体のPCでJenkinsを使う方法をメインで紹介していましたので、いままでスレーブを使う方法を説明してきませんでした。ここでの解説は本来の使い方とは違うスレーブの使い方になりますが、起動方法から順を追って紹介していきます。

実際にスレーブを起動したいユーザアカウントでログインして作業を行います。

ログイン後、ブラウザでJenkins設定メニューの中からノードの管理にある新規ノード作成を選び、ノード名を入力して、ダムスレーブを選んでください(**図6**)。

同時ビルド数はあとから変更できますので、まずは「1」にします。リモートFSルートは、元のJenkinsと被らないように“d:¥Jenkins_slave”といった、区別のつく名前にしておきましょう。また、あとで使いますので、ラベルにこのスレーブの名前を付けておきます。それ以外の項目についてはデフォルトのままで構いません。[保存]を押してこの画面を抜けます。

次にこのスレーブの詳細画面に入ります。ここにスレーブの起動方法が書かれていますので確認してください。slave.jarを起動するのでJREが必要になりますが、Javaのバージョン違いがあると面倒なので、Jenkinsが使っているJREを使います。次のように環境変数PATHにJREフォルダを追加します。

```
PATH=%PATH%;[Jenkinsのドライブとフォルダ]¥jre¥bin
```

追加したら、コマンドラインから先ほどのスレー

●図6　ノード設定画面

ブのページのなかほど「スレーブでコマンドラインから起動:」にあるコマンドを入力して起動してみましょう。

例）筆者の環境でのスレーブ起動

```
javaws http://localhost:8080/computer/ ↵
slavenode/slave-agent.jnlp
```

アプリケーションのデジタル署名などについて警告が出ることもありますが、OKを押して先に進めてください。無事起動すると小さなウィンドウが開いてConnectedと表示されているはずです（**図7**）。

ここまでの一連の動作をバッチファイルにして保存しておくといいでしょう。

例）筆者の環境でのスレーブ起動用バッチファイル

```
PATH=%PATH%;d:¥jenkins¥jre¥bin;
javaws http://localhost:8080/computer/ ↵
slavenode/slave-agent.jnlp
```

ここまでの設定でユーザアカウントでスレーブが起動するようになりました。次回以降起動させる場合は、設定したユーザアカウントでログインしてからスレーブを起動する必要があります。

起動用バッチファイルをスタートメニューのスタートアップの中にコピーしておけば、ログインしたあとにスレーブを起動できます。

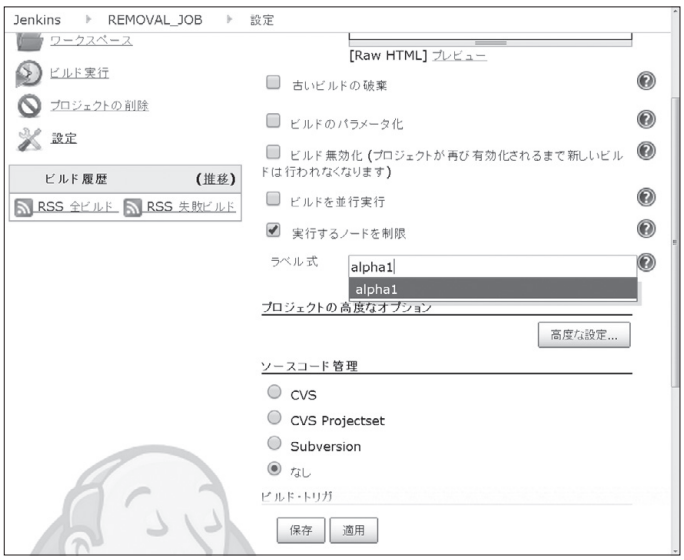
スレーブでJOBを動かす

スレーブが起動していることが確認できたら、次にスレーブでJOBを実行するための設定を行います。JOBの設定にある「実行するノードを制限」にチェックを入れて、ラベル式に先ほどスレーブに設定したラベル（名前）を設定します（**図8**）。

これでこのJOBはスレーブでのみ実行されるようになりました。これによって、ネットワークドライブ経由でデータへアクセスできるようになります。とくに成果物をネットワークドライブにコピーするような場合に便利です。

●図7　スレーブの起動画面が表示される

●図8　実行するノードを制限の設定



まとめ

運用に関する話というよりは設定に関する話になりましたが、バックアップは「やろうと思っていてもなかなか重い腰があがらない」というのがあります。そこでまずは負担の軽い、フォルダとJOB設定ファイルをバックアップする方法から始めてみてはどうでしょうか。どんなにめんどうくさがりな人でも何十分もかからないバックアップなら、筆者のようにHDDが壊れて「あーどうしよう……」とならない対策ができますし、そうなって困る人が少しでも減ってほしいと思います。

スレーブに関しては本来の使い方とはちょっと違いそうですが、こういう方法でどうにかなったというまとめの1つとして参考にしてください。

というわけで、今回で本連載は終わりです。ありがとうございました。SD

分散データベース「未来工房」

第5回 Riak on EC2でデータベースをスケールさせる

――可用性、拡張性、運用性を自在に管理

Writer 上西 康太（うえにし こうた） Bashoジャパン株式会社 kota@basho.com

Riakをクラウド環境で利用できるのはご存じのとおりだ。本稿では、スケールアウトするための構成例を挙げ、スタンダードからハイパーフォーマンスを例にそれぞれ使いどころを解説する。AWS上で使えるVagrantといったツール群の紹介も行う。

注）本稿は、筆者の意向により常体で表記している。

概要

Riakは可用性、拡張性、運用のしやすさが特徴のデータベースだ。マシンを追加すればしただけスケールアウトするし、ダウンタイムはほぼ0で、コマンド体系もごくシンプルでオペレーションミスはなかなか起きない。

だが少し待ってほしい。いくらソフトウェアがよくても、そのソフトウェアが動作するハードウェアやネットワークが高可用で、スケールして、運用しやすいものでないと意味がない。そして、そういった環境を構築するのはとても骨の折れる仕事だ。

もしも、可用性、拡張性、運用のしやすさを備えたクラウドサービスを利用すれば、Riakの可用性、拡張性、運用のしやすさは飛躍的に向上するだろう。

可用性、拡張性、運用のしやすさとは?

スマートフォンのアプリケーションをリリースしたら、想像した以上にユーザが拡大し人気が出てしまったが、そのトラフィックを受けきるようにシステムを拡張できない……ということは大いにあり得るだろう。通常は、負荷の増大よりも速くシステムを拡張しなければならない。これはどんなソフトウェアを利用していても必要なことで、Riakを使ってスケールアウトができるようになっていたとしても、サーバが調達できなければシステムをスケールさせることはできない。サーバを調達できないためにシステムを拡張できず、トラフィックをさばけないままサービスが衰退していく……ハードウェアの調達は、いくらRiakでも解決できない拡張性の問題だ。

同様に、サーバを調達してシステムを構築するのもかなりのコストだ。電源を確保し、ケーブリングしてネットワークを用意し、サーバをラックにマウントしてOSをインストールする……これらをある程度自動化するだけの環境は整ってきたが、どうしても人手を避けられない部分があり、リードタイムは長く安定運用の障害になっていた。これも、Riakでは解決できない運用の問題だ。

また、データベースのダウンタイムが発生すると、ユーザ体験に大きく影響する。もしもユーザへ何らかの課金をするようなアプリケーションだった場合は、データベースのダウンタイムが延びた分だけ機会損失になってしまう。たとえばデータベースが入っているラックだけはネットワーク障害に備えて、ネットワークを二重化しておくなどの対策をとることになるが、バックエンドのシステムごとに毎度冗長化構成を構築するのは非常にコストがかかる。これも、Riakだけでは解決できない可用性の問題だ。

クラウドサービスの優れたところ

タイトルにもあるとおり、これらの問題を解決する方法があ**る**。それは、仮想マシン提供型

(IaaS[注1])のクラウドサービスを利用することだ。一般的に、仮想マシンをサービスとして提供するクラウドの特徴は次のようなものだ。

- **短いリードタイム**
  ……仮想マシンの調達にかかる時間は基本的にリクエストを出してすぐだ。ユーザが待つのはOSがブートする時間だけである
- **プログラマブルな管理API**
  ……仮想マシンの管理はAPIを通じて行う。人手を介することがないので自動化が容易
- **高い可用性**
  ……ネットワークや実マシンなどのインフラは冗長化されており、すべてが同時に利用不可能になることは基本的にない

これをそのままRiakにあてはめると、オートスケールできる高可用なデータベースを実装できる、ということになる。

一方で、クラウドサービスを利用するデメリットもある。それは、管理APIに対するロックインであったり、従量型課金のためにコストが予測しにくくなったり、クラウドサービスそのものの障害に対しては無力であったりと、また別の観点の問題である。Riakをうまく使うことでこれらのデメリットも回避できる場合があるので、その方法も紹介したい。

以降では、代表的なクラウドサービスとしてAWS[注2]を例にしていく。

構成例

ここから、AWS上でいくつか構成例を例示していくことにする。AWSはEC2をはじめさまざまな部品があらかじめ用意されており、これらを組み合わせてAPIを通じてシステムを構築できる。AWSがとくに優れているのは、単に仮想マシンを貸して終わりではなく、ほかにもさまざまな便利なモジュールを用意しているためである。Riakを利用する場合には、おもにEBS[注3]を利用することになるだろう。また、バックアップ用途にS3を利用する場合もある。

基本的なポイント

Riakを安定稼働させるための最小構成台数は5台で、ネットワークが性能に影響する。メモリは8GB程度あったほうがよいだろう。CPUは最低でも2コア必要で、4コア以上が推奨だ。また、ディスクの性能がRiakそのものの性能に影響するので、これも必要に応じてグレードアップするとよい。Riakがデータを読み書きするパーティションについては、Provisioned IOPS(I/O性能が保証されたパーティション、以下PIOPS)つきのEBSを使うことが必須だ。PIOPSのないEBSを利用する場合は安定したI/O性能が得られないために、Riakのプロセスが過負荷になり、データを守るために停止する場合がある。

検証目的であったり、多少データを失ってもよい場合[注4]はマイクロインスタンスを利用したり、ストレージサイズを節約することでコストを抑えることができるだろう。ただし、その場合は物理メモリの不足や過負荷でプロセスが停止する場合があるので注意が必要だ[注5]。

ほかに最も重要な設定のポイントはセキュリティグループである。Riakのノード同士が通信するためには**表1**に示すポートを開けておく必要がある。

あとは構築や運用のために、必要に応じてSSHのポートを開けておく必要がある。OSは初回で説明したとおり、UbuntuなどのLinux

注1) Infrastructure as a Serviceの略。おもに仮想マシンなどITシステムの基本的な部品をサービスとして提供する事業。AWSやAzureはこちらにあたる。

注2) Amazon Web Services：クラウドサービスの元祖

注3) Elastic Block Storage：クラウド上で仮想化された物理ディスクのようなものだと思えばよい。この上にOSをインストールしたり、データを保存して利用する。EBSが便利なのはスナップショットなど便利な機能が多く用意されているためである。

注4) たとえば、セッションストアなどにRiakを利用するなど永続化が目的でない場合。

注5) 分散システムは、少しでも異常があった場合にノードを停止する設計にしておくと非常に設計がシンプルになる。これはStop Failureといって、分散システムの故障モデルの中でも最も単純なものだからである。

がよいだろう。今回の執筆の際には"ami-fe6ceeff"のAMI[注6]で動作確認をしている。

注意

Riakは32ビット環境では動作しないので、t1.microを使う場合でもきちんと64ビットOSを使おう！

Riakのノードが落ちた場合の選択肢として、Riakの前段にHAProxyなどのロードバランサを挟む場合もある。これを挟むことで、Riakノードの故障を検知してHAProxyがそちらにリクエストを送らないようにできる。この構成の問題点はHAProxyの動作するマシンのネットワークがボトルネックになることであった。しかし、Riakの公式クライアントのほとんどは、Riakサーバを複数指定しておいてクライアント側での自動的な再接続を実行できるようになっているから、あまり推奨はしない。

RubyやJavaなどのクライアントを利用する場合は、HAProxyがなくても自動的に別のRiakノードに再接続できるようになっている(**図1**)。PHPなどいくつかのクライアント実装ではこれができないものもあるので、注意が必要だ。

スモールスタート

まずは、実際のサービスを動作させるための小さめの構成を説明する。もちろん、Riakを動作させる最小構成はもっと小さなものだが、Riakの可用性が保証されるのは5台からである。ダウンタイムをある程度許容するならそれより少なくてもよい。

注6) Amazon Machine Image

▼表1　Riakが動作するセキュリティグループで必要なポート

	ポート番号	用途
tcp	4369	Distributed Erlang
tcp	6000～7999	Riakのノード間通信
tcp	8087	RiakのProtocol Buffers API
tcp	8098	RiakのHTTP API
tcp	8099	Riak内部のデータ通信

m1.largeを使うので、個々のマシンは2コア、7.5GBメモリという環境で動作[注7]できる(**表2**参照)。EBSの容量をC_{EBS}とすると、1レコードあたりのオーバーヘッドを無視すれば、5台でディスク容量のマージンを20%とすると、

$$Riak\text{に入るデータ量} = C_{EBS} \times 5 \times 0.8/3$$

でざっくりと計算できるので、たとえば128GBのEBSボリュームを用意すると170GB程度のデータが入ると考えてよい。

EBSのProvisioned IOPSはかなり低めにしてある。もしもディスクがボトルネックになっているようだったらこの数値を変更するとよいだろう。

スタンダード

スタンダードな構成では、m1.xlargeを使うことで15GB程度のメモリと4コアの構成にする。実際の物理マシンでもこの程度のスペックが標準的な構成だろう(**表3**参照)。また、

注7) データベースでは永続性が重要なのでスポットインスタンスは推奨しない。

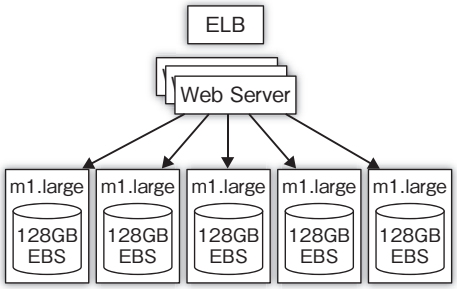
▼表2　スモールスタートの構成例

インスタンス	m1.large
ディスク	EBS(1,000～PIOPS)
台数	5

▼表3　スタンダードな構成

インスタンス	m1.xlarge
ディスク	EBS(2,000～PIOPS)
台数	5～10

▼図1　スモールスタート向けの典型的なWeb3層の構成



EBSはスループットを確保するために、PIOPSを高めに設定するとよいだろう。

EBSのなかでも最も便利な機能の1つがS3へのスナップショットの書き出しである。スナップショットの書き出しでも不十分な場合は、LVMの書き出しなど高速化する方法をある程度検討するとよいだろう。

m1.xlarge程度の規模で、EBSにPIOPSをつけるくらいがスタンダードな構成である。ここからCPUなどの性能を多少上下する場合もあるが、トレードオフを把握して変更するならまったく問題ないだろう。

ハイパフォーマンス

EC2には、2種類のストレージ最適化インスタンスが用意されている。それぞれ、飛び抜けたI/O性能と容量を提供するものだ。これもアプリケーションのユースケースに基づいて選択すればよい。単位時間の処理性能が必要であれば**hi1.4xlarge**を、データ量が必要であれば**hs1.8xlarge**を使えばよい(**表4**参照)。前者には総量2TBのランダムアクセスに強いストレージ(SSD)が、後者には48TBのシーケンシャルアクセスに強いストレージ(おそらくHDD)がある。

これだけのハイパフォーマンス構成を組むと、EBSのスナップショットが使用できなくなる。その場合には、通常のRiakのバックアップ方法(LVMのスナップショット、またはtarballの作成と退避)を用いるのがよいだろう。

システムの拡張と縮小

負荷が上がってきた場合に簡単に拡張できるのがRiakの特徴である。拡張にはスケールアップとスケールアウトの2種類がある。CPUやメモリ、耐障害性などより優れたマシンにリプレースすることをスケールアップといい、単にマシンを追加して台数を増やすことをスケールアウトという。

いわゆる分散型のソフトウェアはマシンを追加してスケールアウトできるのが特徴だが、スケールアウトの際にはマシンあたりのオーバーヘッドも追加されるので、基本的に全体の効率はよくない。トータルでの効率を求める場合には、台数を減らしてマシンをスケールアップするのが正しいが、これには1)システムを停止せずにマシンを高性能なものに入れ替えていくには時間がかかる、2)Riakはすべてのノードを対等に扱うのでクラスタ全体の性能設計は最も小さなマシンに合わせる必要がある、の2点の問題があるため、短期的にはスケールアウトを選択するべきだ。つまり、

- 急いで拡張しなければならない場合はマシンを追加するスケールアウトがリードタイムが短くてよいが、
- 計画可能な場合は、マシンを高性能なものにリプレースするスケールアップがトータルのコストは低い

もちろん、無限にスケールアップすることはできないし、強力なマシンは高価である。したがって、どこかでマシンスペックと台数のトレードオフから最適解をユースケースごとに導く必要がある。通常、スケールアップは難しいが、CPUやメモリを強力なものにしたい場合、EC2なら簡単にできる。EBSのボリュームをそのままにして、新しいインスタンスを立ち上げるだけでよい。もちろんIPアドレスが変わる可能性があるので、**riak-admin replace**のコマンドを利用してクラスタに追加する必要がある。ディスク容量を増やしたい場合はどうしても、いったんマシンを停止してアンマウントする必要があるだろう。

また、スケールアウト型のソフトウェアの重要な特徴として、必要に応じてシステムをスケールダウンする能力も必要である。この場合もスケールアウトとスケールアップと同様の考え方ができる。短期的に縮小したいのであればマシ

▼表4　ハイパフォーマンスな構成

インスタンス	hi1.4xlarge
ディスク	SSD
台数	5〜

ンの台数を削ればよいが、中長期的にコストダウンも狙いたい場合には、より安価なマシンへ徐々にリプレースしていくのがよいだろう。ここで注意しなければならないのは、アプリケーションサーバのようにいきなり規模を縮小することはできないということだ。データを持たないタイプのサーバと異なり、縮小の際にはRiakノードが持っているデータを他のRiakノードへ委譲しなければならないため、この処理にかかる時間の分だけ慎重に計画・設計する必要がある。たとえば、負荷の少ない早朝は5台で運用して、負荷の大きい午後～深夜は15台で運用するというのは**現実的ではない**。

さらに冗長構成

ハイパフォーマンス構成にすると、EBSを利用しないので、スナップショット機能を使ったバックアップもできなくなってしまう。

また、1つのRiakのクラスタを複数AZ[注8]に分散できないため、データセンターレベルでの冗長化はできない。その対処として、商用版Riakには、複数のRiakクラスタを同期するレプリケーション機能が含まれている。これを使えば、**図2**のように複数のクラスタを用意することができる。この用途としては、

・データのバックアップ用クラスタ
・データセンターレベルでの負荷分散
・複数クラウドサービス(またはオンプレミス環境)に跨(またが)ったレプリケーション
・実データを使ったステージング環境、分析用クラスタ

などがあるだろう。

●データのバックアップ用クラスタ

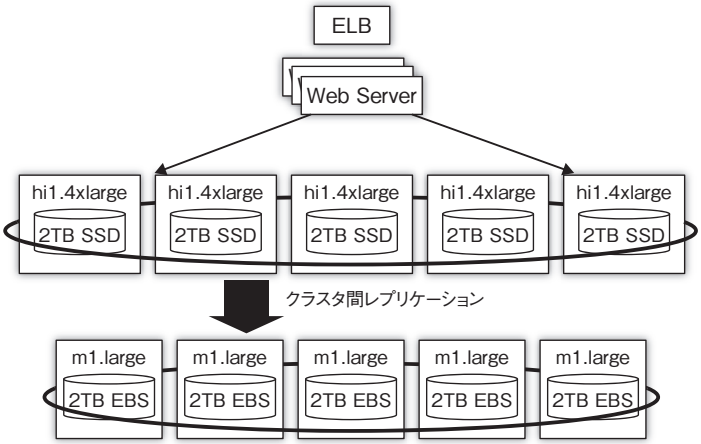
AWSであれば、複数のAZまたはリージョンに跨ってクラスタを用意することで、AZやリージョン単位での障害があった場合でもデータを失うことなく、サービスを停止することなくフェイルオーバーするシステムを組むことができる。S3などにバックアップを保存しておくこともできるが、復旧までの間のダウンタイムをほぼ0にするためにはリアルタイムのレプリケーションがよいだろう。たとえば、バックアップ用のクラスタは通常低負荷なので、スペックの劣るマシンを用意してコストを削減することもできる。

●データセンターレベルでの負荷分散

複数のリージョンに跨ってレプリケーションをすると、たとえば日本のアクセスは**ap-northeast-1**へ、アメリカのアクセスは**us-east-1**へといったふうに、地理的に最も近いリージョンへリクエストを誘導できる。今まで

注8) Availability Zone

▼図2　クラスタ間のレプリケーションつき構成



はCDN(Contents Delivery Network)を利用して静的コンテンツだけを地理分散することしかできなかったが、これによってレイテンシの改善に大きく寄与するだろう。

●複数クラウドサービス(またはオンプレミス環境)に跨ったレプリケーション

これはデータのバックアップと同様の使い方であるが、たとえばAWSとAWS以外のクラウドサービス、または自社のオンプレミス環境を組み合わせられる。複数のRiakクラスタがそれぞれ別々にホストされることで、クラウドサービスそのものの障害によるダウンタイムのリスクを回避できる。

●実データを使ったステージング環境、分析用クラスタ

Riakのクラスタ間レプリケーションではAct-Actの構成を組むのが一般的だが、Act-Sbyの構成を組み、マスターのデータベースとは別にテスト用・ステージング用のクラスタを用意できる。これによって、ステージング環境でデータの更新を行うアプリケーションなどをカジュアルにテストすることができる。もしくは、マスター環境に負荷をかけるのを避けるために、Sby上にあるデータで分析用のバッチジョブを実行するといった用途も考えられる。レプリケーションは非同期であるが、ほぼリアルタイムにデータは同期されるので、鮮度の高い分析ができるだろう。

ツール紹介

RiakをAWS上で使うためのツールにはさまざまなものが用意されている。

Vagrant

もっとも手軽にしたいのであれば、Vagrant[注9]のAWSプラグインがよいだろう。VagrantのAWSプラグインをインストールするには、

```
$ vagrant plugin install vagrant-aws
```

とすればよい。EC2の情報をVagrantleに設定し[注10]、仮想マシンを起動するためには

```
$ vagrant --provider=aws up
```

とする。仮想マシンが起動してから、基本的な操作は、

```
$ vagrant ssh "実行したいコマンド"
```

として、Riakのマシン数の分だけ操作を続けていくことでクラスタの構築や管理ができる。ここから先は通常のRiakのインストールと同様に行えばよい。

Vagrantはもともとローカルの仮想マシンに簡単にアクセスするためのものなので、少数のマシンで簡単に試すならこれがよいだろう。

Ansible

Ansibleは、Chef、Puppetなどに近いプロビジョニングツールである。Playbookと言われる設定ファイルをYAMLで記述することにより、複数のサーバの構成管理を自動化できる。

AnsibleはAWSの各種サービスのPlaybookがあり、これを使えばAnsibleを通じてEC2上の仮想マシンの操作や設定が可能になる。AWS上でマシンを起動する注意点は特にはないが、これまで説明してきたように、PIOPSとセキュリティグループの設定を正しくやっておけば問題ない。

また、Ansibleの公式ライブラリにはRiakのPlaybookも含まれているので、Ansible公式の構成例[注11]をもとにして、EC2上にRiakの構成を行うことができる。

基本的な流れとしては、

注9) 開発環境の仮想マシン上へのセットアップを支援するツール(http://www.vagrantup.com/)。

注10) 詳しい情報はAWS プラグインのレポジトリにある(https://github.com/mitchellh/vagrant-aws#quick-start)

注11) https://github.com/ansible/ansible-examples/tree/master/riak

・EC2でマシンを複数台起動し、インベントリファイルにサーバ一覧の名前を書き出す（SSHで入るための証明書をあらかじめ用意しておくこと）
・書きだした一覧の名前をAnsibleのインベントリファイルとして、RiakのPlaybookをその上で実行する

となる。Ansibleは大規模な環境でも多くの実績があるから、Vagrantよりも大きいクラスタを管理しやすいだろう。

もちろん、両者を組み合わせて、EC2管理の部分をVagrant、仮想マシン内部の構成にAnsibleを使用してもよい。

CloudFormation

また、テスト用ではあるがCloudFormationの部分的なテンプレートも用意されている[注12]。これを利用すれば、面倒なカーネルパラメータの設定なども含めて10分ほどで任意の規模のクラスタを簡単に構築できるだろう。

関連するクラウドサービス

Riakの開発元のBasho社は、現在3社のクラウドサービス事業者と提携しており、そこでのRiakの動作をサポートしている。

まずは本稿のもとにもなったAmazonのAWSである。これについてはAmazonからホワイトペーパーが公開されており、そこでもRiakをAWS上で稼働させるための基本的な考え方が解説されている[注13]。さらに、AWSのマーケットプレイスにBashoの公式AMIが公開されている[注14]。これを用いれば、はじめからRiakがインストール・設定されたマシンを利用できて便利だ。

次に、Microsoft Azureである。こちらも、BashoとMicrosoftが提携し、Azureの上でRiakが動作ためのテストなどを行った[注15]。Riakのドキュメントにも、詳しい手順が公開されている[注16]。

また、PaaS[注17]のサービス上でもRiakを利用できる。PaaSベンダーのEngineYard[注18]では、MySQL、PostgreSQLと並んでRiakを選択できる。EngineYardではさらに構築の手間が削減されていて、プルダウンメニューで台数を選択すれば、それでクラスタとして自動的に稼働する。自由度は下がるが、構築や運用のコストは劇的に下がるので、やりたいことが限られていて手間をかけたくない場合にはこちらがよいだろう。

まとめ

クラウドでRiakを使うと可用性、運用性、スケールアウトのすべてが強化されることを解説した。また、RiakをEC2上で運用や構築するためのノウハウ、ツール群を紹介した。

RiakはPCサーバ上で動作するデータベースなので、AWSに限らず基本的な機能をもったIaaSならどこでも動作する。読者が自身で使っているクラウドサービスがあれば、そちらでの構築を試してもらいたい。SD

注12）https://github.com/basho/cloudformation-riak
注13）AWS上でRiakを実行する―新しいホワイトペーパーを公開（ http://aws.typepad.com/aws japan/2013/07/running-riak-on-ec2-new-white-paper.html）
注14）https://aws.amazon.com/marketplace/pp/B00AMRXCQA
注15）http://blogs.msdn.com/b/windowsazure/archive/2012/10/09/riak-is-now-available-on-windows-azure.aspx
注16）http://docs.basho.com/riak/latest/ops/building/installing/azure/
注17）Platform as a Serviceの略。Webサービスを支える、いわゆるWeb3層構造をサービス化して提供する。Herokuなどがこれにあたる。
注18）https://www.engineyard.com/

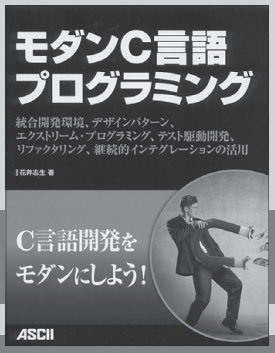
BOOK no.1 マスタリングTCP/IP OpenFlow編

あきみち、 宮永 直樹、 岩田 淳 【著】
B5判、 264ページ／価格＝2,940円 （税込） ／発行＝オーム社
ISBN＝978-4-274-06920-8

本書は潔くよくSDN（Software Defined Network）をばっさりとカットし、OpenFlow1.0と1.3のプロトコル解説にフォーカスしている。その理由は、SDNの定義があいまいだからだというもの。至極当然である。寄り道をせずに直球で解説するため、OpenFlowの概要を知りたい方にとって必須な情報が掲載できている。第4章「L2スイッチの機能再現から学ぶOpenFlow」は、VLANの基礎とそのしくみを復習することでOpenFlowがどんな動作をするものなのか具体的に理解が進むだろう。OpenFlowはネットワークエンジニアにとって応用問題だと言えよう。独自実装の解説にこだわることがないので極めて情報の純度が高く有用だと感じた。

BOOK no.2 モダンC言語プログラミング

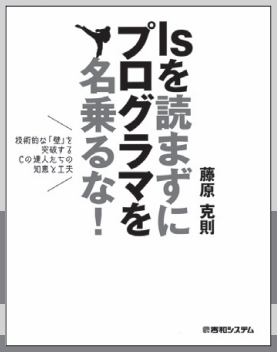
花井 志生 【著】
B5変形判、 280ページ／価格＝2,940円 （税込） ／発行＝アスキー・メディアワークス
ISBN＝978-4-04-891309-6



本書は古典的なC言語を、最新のサーバサイドプログラミングで用いられている手法を取り入れて、モダンなC言語として利用するノウハウを解説している。統合開発環境（Eclipse）の構築から、オブジェクト指向、デザインパターン（GoF）、エクストリーム・プログラミング、リファクタリング（Google Testによるテスト駆動開発の実施と実践）、インテグレーションとデプロイ（Jenkins利用、CIサーバによる自動化、Valgrindによるメモリチェック）など、モダンな開発スタイルについて実際の利用方法が説明されている。全体的に中級者以上向けで、組込み系視点からの解説だが、それ以外の開発でC言語を扱っている人にも役立つだろう。

BOOK no.3 lsを読まずにプログラマを名乗るな!

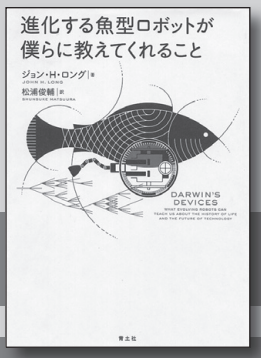
藤原 克則 【著】
B5変形判、 240ページ／価格＝2,520円 （税込） ／発行＝秀和システム
ISBN＝978-4-7980-3943-5



lsコマンドのコードリーディングの手引書とも言える1冊。lsにはBSD系とGNU系があるが、本書は読むのが難解と言われるGNU系のlsを対象としている。本書に限らずlsのソースコードは若手プログラマの教材としてよく使われるが、C言語の文法を知っているだけでは、コードを十分に理解するのに苦労するだろう。本書はコードのどこから読み進めればいいか、実行性能や利便性を高めるためにどんなテクニックが使われているかなどを示しつつ、最低限必要なOSの機能などについてはコラムで随時、解説してくれる。巻末にはコードを読む際に便利なツールや技の紹介もある。初学者にはlsのソースコードを読む際の副読本としてぜひ使ってほしい。

BOOK no.4 進化する魚型ロボットが僕らに教えてくれること

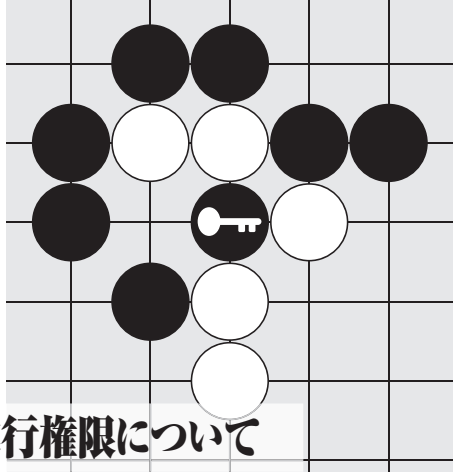
ジョン・H・ロング 【著】 ／松浦 俊輔 【訳】
四六判、 336ページ／価格＝2,520円 （税込） ／発行＝青土社
ISBN＝978-4-7917-6718-2



ロボットを使った進化論研究の取り組みを紹介した本。絶滅した初期の魚に似たロボットを複数作って、餌を巡って競争させる。その結果をもとに次の世代のロボットを作ってまた競争させる。それを何回も繰り返して進化の方向性を調べる。進化研究にこんなアプローチがあるというのは、純粋に驚いた。進化の過程をシミュレーションするためには、ロボットの形や素材、実験環境、測定方法などあらゆることを考えないといけないが、その選択肢や判断基準などがこまかく説明されている（技術面だけでなく労力面についての言及もあって生々しい）。地味な試行錯誤と緻密な検証作業が繰り返される最先端の研究の現場の空気が感じられる内容だった。

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

【第五回】UNIXのファイルアクセス権限および実行権限について

今回はUNIXのファイルのアクセスや実行権限について基本から解説します。さらに先日起こった大手レンタルサーバ事業者の改ざん事件を題材に、セキュリティ的な落とし穴はどこに発生しやすいのか議論していきたいと思います。

WordPress大規模改ざん事件

先日、大手レンタルサーバ事業者が提供しているサービスで、WordPressが大量に改ざんされる事件がありました。業者発表によると8,438件の改ざんが発見されたとのことです[注1]。Twitter上での話題など[注2]もあり、特定の大手レンタルサーバ事業者の不手際の問題としてクローズアップされましたが、実際には似たような問題はよく出会います。

この要因の1つにUNIXのファイルアクセス権限および実行権限についての問題があります[注3]。ここでは個別のケースに特化して議論するのではなく、より一般化して見てみたいと思います。そうすることで、本質的に同様な問題を抱えているサイトへのヒント、あるいは、今後このような問題を作り込まないための手助けになればと考えます。

UNIXのファイルのアクセス制御

UNIXのファイルやディレクトリへのアクセスの基本は「所有者」「グループ」「それ以外」という3つの区分けがあり、そこに対し「読む」「書く」「実行する」という3つの許可を与えるモデルです。

まず所有者の意味は、ファイルの持ち主であるユーザアカウントです。次のグループですが、UNIXではグループという形で、複数のユーザからアクセスできるアクセス制御の方法が使えます。自分がどんなグループに属しているか知りたいときは、コマンドgroupsを実行します。筆者の場合は**図1**のようになっています。

あるファイルのアクセス権として、これらのグループに「読む」「書く」「実行する」が許可されていると、その設定に応じてアクセスできます。foo.datファイルをコマンドddを使って作成し、lsで情報を表示してみます(**図2**)。

先頭の"-rw-r--r--"がファイルのパーミッション(アクセス権)の状態を表しています。次の"1"がリンクカウント、最初の"hironobu"が所有者(owner)、

◆図1　自分の所属グループを調べる

```
$ groups
hironobu adm dialout fax cdrom floppy tape audio dip  以下略
```

◆図2　ファイルのアクセス権を調べる

```
$ dd if=/dev/urandom count=3 > foo.dat
$ ls -l foo.dat
-rw-r--r-- 1 hironobu hironobu 1536 Sep 16 08:12 foo.dat
```

注1) ㈱paperboy & co.レンタルサーバサービス「ロリポップ！レンタルサーバー」 2013/09/09「第三者によるユーザーサイトの改ざん被害に関するご報告」 http://lolipop.jp/info/news/4149/。

注2) 「ロリポップ改ざん被害のお詫び。私の発言についてのお詫びとご説明」 http://www.kumagai.com/?day=20130908

注3) シンボリックリンク時のアクセス制御の問題もありますが、今回はファイルアクセス権限および実行権限にテーマを絞ります。

その次の“hironobu”がグループ(group)、ファイルサイズが1,536バイト、書き込んだ日時が“Sep 16 08:12”という意味になっています。

図3のように“-rw-r--r--”の一番最初のシンボルは性質を表します。たとえばディレクトリであれば“d”、シンボリックリンクであれば“l”などです。ファイルであれば指定はなく“-”となります。次から「所有者」「グループ」「それ以外」のブロックになります。各々に3つの表示があり、最初が「読み」、次が「書き」、そして「実行」を意味します。この3つのブロックに関しては「読み」「書き」「実行」の各々に1ビットずつ与え、そのビットをON/OFFするモデルにしています。その値として8進数で表すモードと呼ばれる方式を使っています(**表1**)。

ここの部分は、UNIXの初期のころのシンプルなモードの考え方から、いろいろな属性の設定などがあとからアドホックに加わり、妙にわかりづらい体系になってしまっています。

たとえばスティッキー・ビット[注4]というのがあります。「それ以外」の「実行」(つまりお尻の部分)に“t”の表示が現れますが、これを設定するモードは1000(8進数)です。

すでに体系が崩れてしまったこのやり方が、UNIXのパーミッションや実行権限の理解を難しくし、ひいてはセキュリティの問題を複雑にしている原因ではないかと思うこともしばしばです。そこでもう一度話を整理するために、ファイルはテキストファイルと限定し、説明を試みたいと思います。

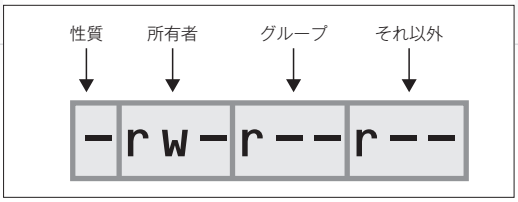
テキストファイルにおけるパーミッション

ファイル作成時のパーミッション

端末上でファイルを作ったときに、デフォルトでどんなパーミッションになるか調べてみましょう。

ファイルを作る際のデフォルトのパーミッションは、親プロセスから引き継がれたumask値を使います。それと0666(ディレクトリの場合は0777)の値とマスク(AND演算)を取り、その値をパーミッションとして使うようになります。

◆図3　パーミッション表示の意味



◆表1　読み書きのビットパターンと8進数表現

利用レベル	読み	書き	実行	ビットパターン	8進数表現
所有者	○	○	×	110	6
グループ	○	×	×	100	4
それ以外	○	×	×	100	4

“-rw-r--r--”をモードで表現すると644(8進数)となり、「所有者」に読み書き許可、「グループ」と「それ以外」には読みのみ許可となる。

```
$ touch sample.txt
$ ls -l sample.txt
```

筆者の利用環境ではデフォルトは“-rw-r--r--”となっています。umask値は自分でも変更できます。

```
$ umask 000
$ touch foo.txt
$ ls -l foo.txt
 000のビットパターンは 000000000
 パーミッションは -rw-rw-rw-になる
$ umask 022
$ touch foo.txt
$ ls -l foo.txt
 022のビットパターンは 000010010
 パーミッションは -rw-r--r--になる
$ umask 077
$ touch foo.txt
$ ls -l foo.txt
 077のビットパターンは 000111111
 パーミッションは -rw-------になる
```

umaskだけを実行すると現在のumaskの値が表示されます。またumaskで“-S”オプションを指定するとシンボルで表示されます。

ファイル作成後のパーミッション変更

ファイルを作成したあとに、パーミッションを変更するのは、コマンドchmodを利用します。chmodは引数にシンボルを使う表現と8進数を使う表現が

注4) UNIX 5th Edition(1974)から導入された権限ビットで、主記憶メモリ上になるべく確保しておく機能を表していました。現在ではディレクトリで作成ユーザ以外はファイルやディレクトリを削除できないことを表しています。

あります。オプションで使えるシンボルは次のようになります。

- 誰に対しての許諾か
  - u：所有者
  - g：グループ
  - o：それ以外
  - a：すべて
- 適用
  - +：パーミッションを加える
  - -：パーミッションを外す
  - =：パーミッションをそのまま設定
- パーミッション
  - r：読み込み
  - w：書き込み
  - x：実行（ディレクトリへのアクセス）

たとえばfoo.txtが600（"-rw-------"）だとします。ここにグループのみ読み取りの許可を加えるならば次のようにします。

```
$ chmod g+r foo.txt
$ ls -l foo.txt
-rw-r-----   省略
```

ファイルへのアクセスの許諾

自分が作成するファイルについて

ファイルのパーミッションに関してはumask 077として、デフォルトで「グループ」にも「それ以外」にもアクセスさせないのが最も安全です。ですが、デスクトップ環境の場合、今どきのGNU/Linuxだとユーザフレンドリーなグラフィカルユーザインタフェースのためにいろいろなデーモンがバックグラウンドで走っており、すべてを読み出し不可にするとうまく動作しないものが出てくる可能性もあります。

グループによるアクセスの問題点

先ほどユーザhironobuはadmのグループに属していることをgroupsで確認しました。では、そのadmのグループ権限でファイルの内容にアクセスするとどうなるか確認してみます。/varディレクトリ以下のadmグループで読み込み可能なものを探してみます（**図4**）。このようにグループの権限で参照することが可能だということがわかると思います。

グループによるアクセスの許諾の問題点は、そのグループに属しているユーザすべてに許諾を与えて

◆図4　admグループで読み込み可能になっているファイルを探す

```
$ find /var -group adm -perm -g=r  -exec ls -ld {} ¥;
 →UbuntuやDebianでは/var/log/syslogがadmのグループ権限で読み込みを許していたので確認する
$ ls -l /var/log/syslog
-rw-r----- 1 syslog adm 113691 Sep 18 06:10 /var/log/syslog
$ less /var/log/syslog
 以下略
```

◉umaskのデフォルト値について

GNU/Linuxディストリビューションにおけるユーザのumaskのデフォルト値は、0002と0022の2つの流儀があります。手元にあるディストリビューションをチェックしてみたところ、Ubuntu 12.04やCentOS 6.4が0002で、Debian 7.1.0が0022でした。

筆者のコンピュータ環境は、複数のディストリビューション間でファイルをNFSで共有しますので、Debianのデフォルトであるumask 0022に統一するため、.bashrcで明示的にumaskを0022を設定しています。

CentOSの場合、/etc/bashrcでユーザIDを値が200より小さい場合は0022、そうでない場合は0002と設定を変えていたりするので要注意です。

またumaskの設定がうまくいっていないサーバプログラムなどは、ファイルを作成するときに、パーミッションが適切ではない場合がありますので要注意です。

しまうことです。たとえばhironobu、malloy、apacheという3つのアカウントがあり、hironobuとmalloyはapacheにファイルを読み込ませることを許可したいとします。そしてグループとしてapacheがあり、そのグループにhironobu、malloy、apacheが属しているとします。

図5を見てください。ユーザapacheは、hironobuのファイルであるfoo.txtとmalloyのファイルであるbar.txtの両方を読むことができます。やりたいことはそこまでで、malloyとhironobuは互いにファイルの中身を相手に見せたくありません。しかし、この古典的なグループを使った方法では、malloyはfoo.txtを、hironobuはbar.txtを読めてしまいます。

ACLの利用

ACL(Access Control Lists)はオリジナルのUNIXには存在しておらず、あとから加えられたアクセス制御方式です。その仕様に関してはPOSIXで定義されています。Linuxでは、Linuxカーネル2.6以降で採用されています。現在ではLinuxの主要ファイルシステムで利用可能です。

ACLはグループなどいちいち設定せずに、直接、その特定のユーザのみのアクセスを許すという方式です。**図6**のようなファイルがあるとします。ACLの設定状況を表示するコマンドgetfaclを使ってfoo.txtの状態をチェックしてみます。

◆**図5　グループに対する読み込み許可の例**

```
$ ls -l foo.txt bar.txt
-rw-r----- 1 hironobu apache 5 Sep 18 08:41 foo.txt
-rw-r----- 1 malloy   apache 6 Sep 18 08:42 bar.txt
```

◆**図6　ACL適用前**

```
$ ls -l foo.txt
-rw------- 1 hironobu hironobu 6 Sep 18 08:59 foo.txt
```

◆**図7　ACLでapacheのみ読み込みを許す**

```
$ setfacl -m user:apache:r foo.txt
$ ls -l foo.txt
-rw-r-----+ 1 hironobu hironobu 6 Sep 18 08:59 foo.txt
```

```
$ getfacl foo.txt
# file: foo.txt
# owner: hironobu
# group: hironobu
user::rw-
group::---
other::---
```

まだACLの設定を何もしていないので、先ほどlsでチェックしたのと同じ状態です。では、foo.txtにACLの機能を使ってapacheのみ読み込みを許す設定をします(**図7**)。

ファイルのパーミッションの表示に"+"がつきましたが、これがACLの設定がされているという意味です。

```
$ getfacl foo.txt
# file: foo.txt
# owner: hironobu
# group: hironobu
user::rw-
user:apache:r--
group::---
mask::r--
other::---
```

先ほどの表示と比べるとapacheのユーザに対してr--(読み込み)が設定されていることがわかります。これでユーザapacheのみに読み込みが許されるということになりました。このようにACLを使えば、特定のユーザにファイルへのアクセスを許すことが可能になります。

プロセスの実効UID

実行時に、その権限を別のユーザとして実行することです。rootで実行している最中にシステムコールsetuid(uid)を呼ぶことでプロセスの実効(effective)IDが変化します。簡単にいうとsetuidで指定された別のユーザとして実行されます。

多くの場合、実行時にユーザ権限を適切なものにすることで、セキュリティを確保するためのものです。ApacheサーバではsuEXECでCGI

などの実行に適切なユーザ権限を割り振ります。suPHPはphpプログラムを実行するときにsuEXECと同じ役割を果たします。

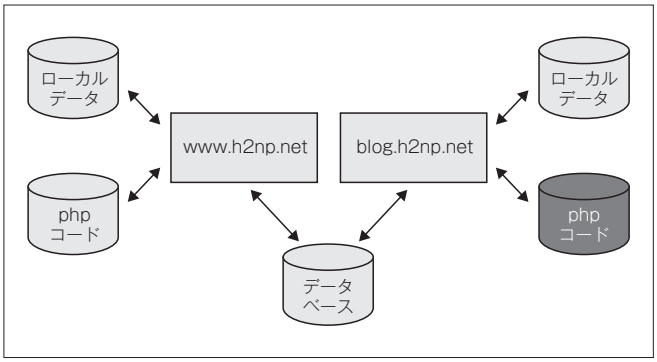
たとえば、httpdサーバはサーバプログラムがスタートするときはrootで実行されますが、実際に実行されているプロセスはwww-dataやapacheといった別の権限で動作します。これはroot権限で動かしていた場合、バッファオーバーフローなどで、httpdサーバ権限が乗っ取られる（システムがまるごと乗っ取られる）ことになってしまうからです。被害を最少にするために適切な権限で実行します。

バーチャルホスト環境における問題点

やっと本題です。冒頭の改ざん事件はバーチャルホスト環境で起こりました。バーチャルホストの機能を使えば、1つのサーバで複数のドメインを管理できます。たとえばwww.h2np.netとblog.h2np.netを1つのホストで運用するとします。そして、おのおのに別々のWordPress環境がインストールされ、利用しているデータベースは共通のものを使っているとします（**図8**）。

いくつものWordPressが別々の管理をされていて、インストールされているプラグインも別々だとしましょう。このようなバーチャルホスト環境でいくつものWebサービスをホストするというのは、ごく普通に見かける風景です。

◆図8　1サーバで複数のWordPress環境を運用する例



バーチャルホストの機能を使っているので、概念的には別々のホストのように見える。概念的にデータベースのホストは同一のものを使っている。実際には、これらはすべて同じマシン上で動作している。

さて、この状態でどこかのWordPressのプラグインに脆弱性があり、任意のファイルをWordPress環境の任意の場所におけるとしましょう。これまでにも、画像ファイルをアップロードするプラグインに脆弱性があり、WordPressのPHPのコードを上書きできてしまう、あるいはPHPのコードを追加できる、などというものがありました。

脆弱性を突く

このような脆弱性への最初の攻撃は完全にスクリプト化し、インターネット上にあるすべてのWebサーバに対して網羅的に攻撃を行うのがセオリーです。そして脆弱性を修正していないメンテナンスの悪いWordPress環境を狙います。

任意のPHPコードが実行できますから、まず、WordPressが利用しているデータベースへのアクセスが可能になります。あとはデータベースの中にあるWebコンテンツを自動的に書き換えられますし、勝手にユーザを追加し好き勝手にログインし、Webのコンテンツを加えることもできます。

この対策は確実にセキュリティアップデートをしておく、ということにつきます。WordPressにはセキュリティを高めるためのプラグインもあって、それなりに効果がありますが、やはりどこかに脆弱性が残っていれば攻撃されてしまいます。とくにアップロードで使うプラグインを追加するときは、十分に気をつけてください。またアップデート情報の入手もこまめにしたほうが良いでしょう。

ちなみに筆者は、WordPressのPHPファイルが書き換わった場合に、自動的に管理者にメールが届くようにしています。WordPressのプラグインにもありますし、Tripwireのようなファイルの改ざん検知ツールを導入するのも良いでしょう。ただ、ゼロデイ攻撃には耐えられないので、ある程度の覚悟は必要です。

WordPress改ざん事件を読み解く

例の改ざん事件の最大の問題は、Word

Pressのデータベースの接続アカウントとパスワードが書かれているwp-config.php[注5]の設定について、グループへ「読み」を許していたため、自分以外のユーザも読めてしまったことです。

WordPressはバーチャルホストで運用されることも多く、その前提で攻撃スクリプトを作成すれば最大限の効果をあげられます。攻撃ステップは次のような感じになるはずです。

①Apacheのバーチャルホスト設定ファイルを探す
/etcの下に固定的なパスで存在していて、普通は誰でも読めるパーミッションになっている

②バーチャルホストのディレクトリのリストを作成
やみくもにホストの中を探すわけではないので効率があがる

③候補のディレクトリ以下にwp-config.phpがないか探査する
普通はhttpdサーバが読めるパーミッションになっているので内容がわかる

④データベースにアクセスし改ざんを行う
データベースもアクセス先もパスワードも明らかになっている

このパターンで処理するスクリプトですから、あとはどれだけのバーチャルホストが影響下に存在しているかだけの話です。たとえ数千件の改ざんが発生しても、それは短い時間で終了しているはずです。

筆者のバーチャルホストの環境も、この攻撃には耐えられません。なぜならばWordPress環境の所有者はhttpdサーバの実効IDと同じapache（www-data）[注6]だからです。

suPHPやsuEXECを用いて実行時にユーザ権限を変更する（さらにchrootなどを設定する）方法がありますが、それらの環境で必ずしもうまく動くという保証はありませんので、それなりの手間とリスクを覚悟する必要があります。

この攻撃に耐える環境とは

かなり限定的になると考えます。

①まず、バーチャルホストごとに対応する個別のユーザを作成して、そのユーザのファイルとしてPHPやRubyなどのプログラムコードを設置する
②それらのコードを実行させるときは、必ずsuPHPやsuEXECを使い、httpdの実効ユーザIDでは動作しないようにする（またchroot設定の適用を検討する）
③バーチャルホスト以下のコードに関して、基本的にはユーザ以外読み書きできなくする
④それ以外のhttpdサーバ関連のファイルに関しては、apache（www-data）が参照できるようにしておく。グループやそれ以外のユーザがアクセスできるパーミッションを設定するのではなくACLのような特定のユーザのみからアクセスできる設定が望ましい

今回、大手レンタルサーバ事業者の改ざんに対する対策が迷走していたころ、どこかのブログで「アクセス制御のためファイルのパーミッションを404にする」という話題がありました。たぶん、これは登録されているユーザはすべて同じグループに属していたのでしょう。その前提に立てば、パーミッションを404とすれば「ほかの自分と同じように登録されているユーザ」からはアクセスできません。たぶんhttpdの実効ユーザIDは、www-dataといったユーザとは異なるグループだったのでWebアクセスはうまく動いたのだと思います。確かに一見正しいように見えますが、それ以外のユーザからはアクセスできるので、「頭かくして尻隠さず」みたいな状況になります。

◆ ◆ ◆

今回扱った問題は、常に最適な答えが見つかるわけではないし、落としどころを探すとしてもたいへんです。間違えると影響は大きい。例に挙げた改ざん事件も他人事ではないと理解していただけたら、筆者としては幸いです。SD

注5）自分でソースコードを入手しインストールした場合、httpdサーバのumaskを引き継ぐので注意のこと。最悪ファイルのパーミッションが666になっている可能性もある。
注6）httpdサーバはDebian系ではwww-data、CentOS系ではapacheのユーザID。

プログラム知識ゼロからはじめる
iPhoneブックアプリ開発

第7回

インタラクティブ性を持たせよう(その1)分岐

GimmiQ(ギミック;いたのくまんぼう&リオ・リーバス)
URL http://ninebonz.net/
URL http://www.studioloupe.com/
イラスト●中川 悠京

プログラミングをしたことのない方にもアプリを作る楽しさを味わってもらいたい本連載。今回はページを分岐させる方法を解説します。

より「アプリ」らしく

今月から2回にわたってアプリにインタラクティブな要素を追加していきたいと思います。紙の書籍ではなかなかできないことをアプリに追加することによって、よりアプリであることの可能性を広げていきましょう!

「その1」の今回は「分岐」です。これまではページをめくって表示される次のページは毎回必ず同じで、ちょうど1本の道を進むようなイメージでした。分岐を入れるということは、読者の操作によって次に表示されるページを選べるということです。道のイメージで表すと途中で道が2本や複数本に枝分かれしていて、どちらに進むのか決めることができるということです。

分岐先を増やす

ステップ1

それではまずは分岐先となるページを増やしましょう。

2ページ目から分岐するようにしようと思いますので、これまでの連載を参考に**図step1**のようになるように「Page2B」を作成しましょう。詳しい手順は過去記事に記載されているとおりですが、おおまかな手順を列挙すると次のようになります。

①「Object library」から「View Controller」を追加
②画像をページに追加
③ページ移動用のボタンを2つ追加
④タイトルと本文のUILabelを追加
⑤ソースファイル「Page2BViewController.h」「Page2BViewController.m」を追加
⑥ソースファイルに「1ページ目に戻れるように」「ページめくりの音が鳴るように」「文字の装飾が追加されるように」するためのコードを追加

step1

ちなみに作例では「金装飾」の写真を配置しましたので、ページのタイトルは「Gold」とし、本文も金に関するものとしてあります。

分岐を作る

ステップ2

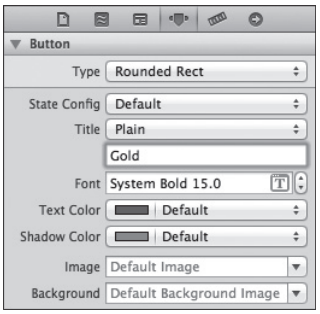
分岐用のボタンを追加しましょう。

「Object library」から「Round Rect Button」を1ページ目にドラッグ&ドロップし貼り付けてください（**図step2-1**）。このボタンは「Page2B」に移動するボタンとなります。

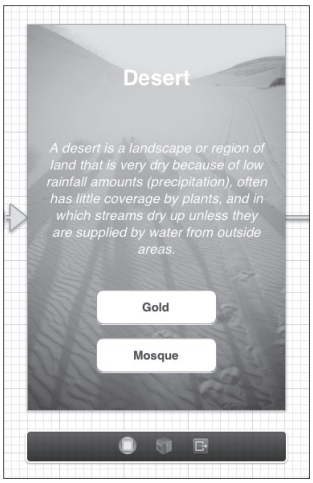
ですので、ボタンのタイトルは「Gold」としましょう。Xcode右カラムのアトリビュートインスペクタアイコンをクリックし、「Title」に入力されている「Button」という文字列を「Gold」に変更します（**図step2-2**）。

元々あった「Page2」に移動するためのボタンも同様に追加し、ボタンのタイトルを「Mosque」としました。これで、1ページ目は**図step2-3**のようになったかと思います。

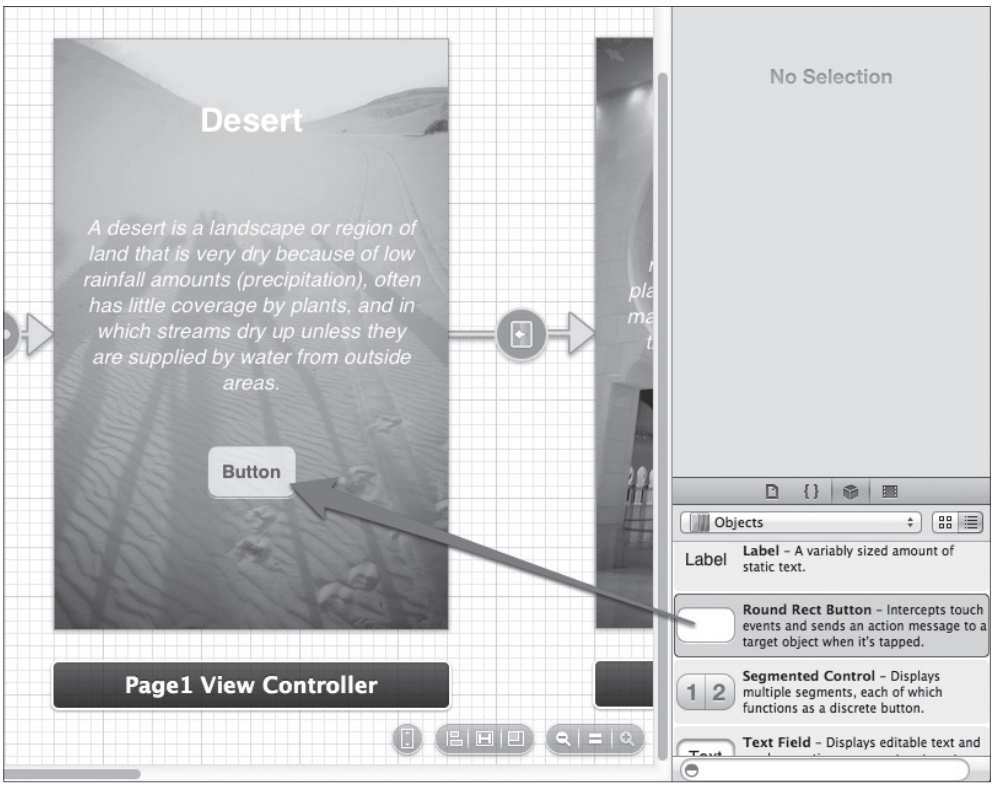
step2-2



step2-3



step2-1



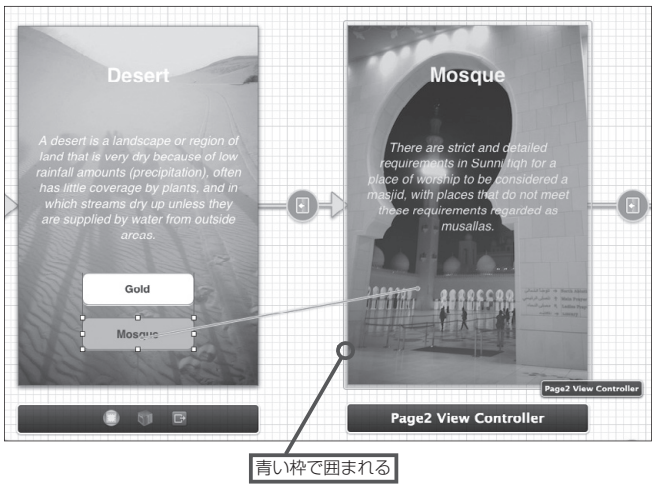
ステップ3

次に各ボタンをそれぞれの分岐先のページにつなげましょう。

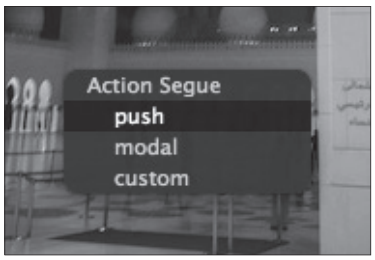
「Mosque」のボタンをcontrolキーを押しながらドラッグし、表示された青い線を「Page2 View Controller」まで引っ張り、2ページ目が青い枠(誌面ではわかりませんが)で囲まれたらマウスのボタンを放します(**図step3-1**)。ボタンを放したときに出る接続方法のメニューは「push」を選択してください(**図step3-2**)。

同様に「Gold」ボタンも「Page2B View Controller」につなげましょう(**図step3-3**)。

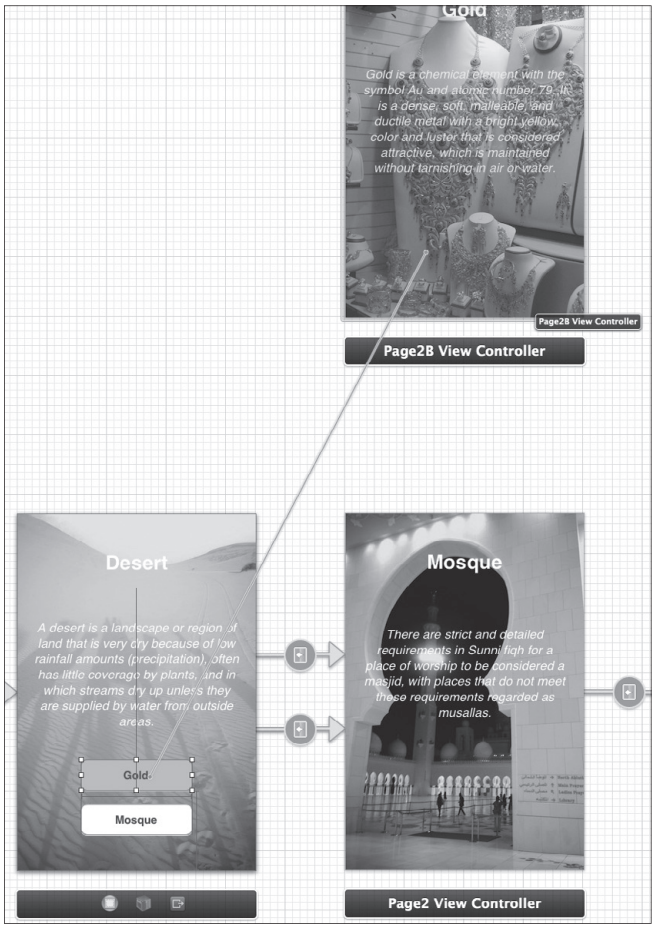
step3-1



step3-2



step3-3



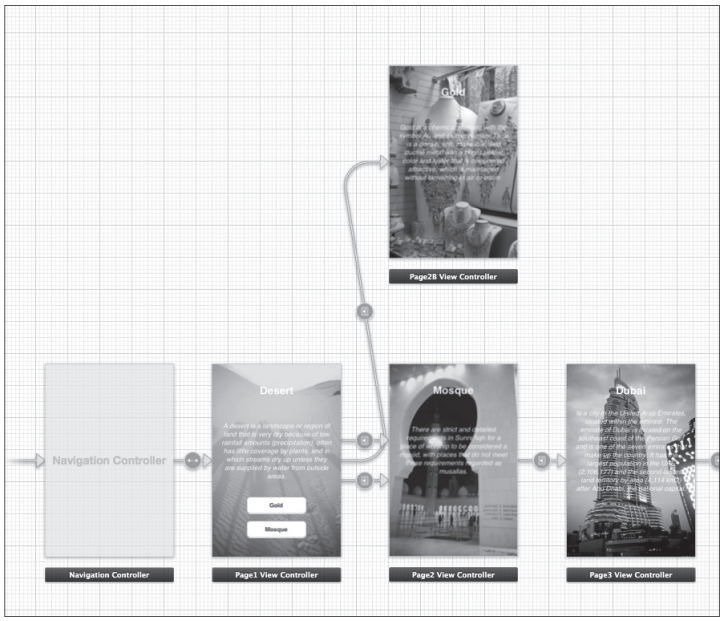
ステップ4

ここまでできると、ストーリーボード上では**図step4-1**のように分岐が矢印で視覚的に確認できると思います。

1ページ目から「Page2」への導線が2本あるのは、以前設置したページめくり用のボタンから出ているものと、今回新たに設置したボタンから出ているものがあるからです。分岐用のボタンからのみつなげている「Page2B」へは導線は1本しかありません。「Page2」へも分岐用のボタンからしか移動させたくなければ、1ページ目に設置してある通常のページめくり用のボタンは削除してしまってもかまいませんが、とりあえず現状のまま先へ進みましょう。

それでは、実行して動作を確認してみましょう。**図step4-2**のように1ページ目に分岐用のボタンが並び、タップするとそれぞれのページが表示されたでしょうか?

step4-1



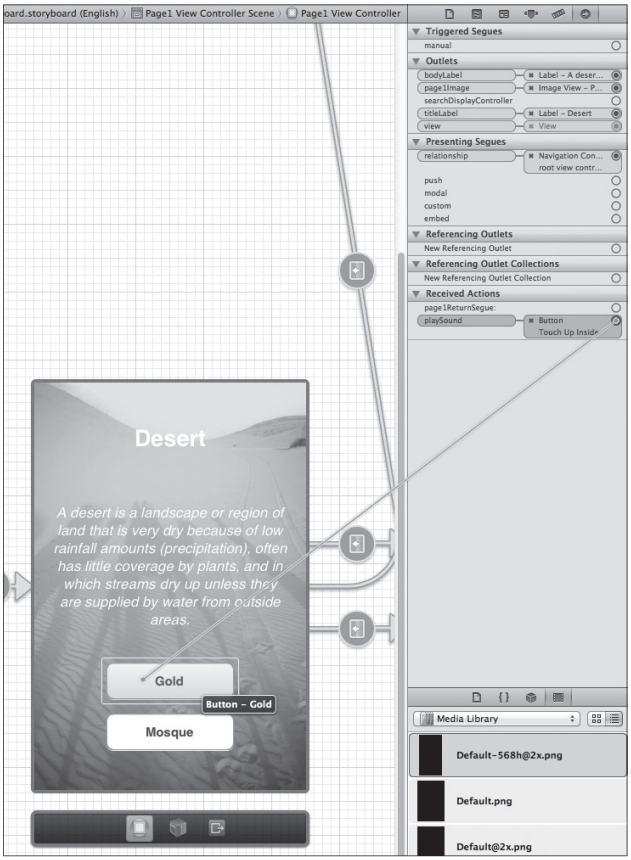
step4-2

ステップ5

無事分岐するようになりましたが、分岐用のボタンでページ間を移動したときにはページめくりの際の効果音が鳴っていませんね。このままでは少し寂しいと思いますので、分岐用のボタンでちゃんと効果音が鳴るようにしましょう。

ストーリーボード上で「Page1 View Controller」を選択した状態で、Xcodeの右カラムから「コネクションインスペクタ」を選んでください。分岐用のボタンを control キーを押しながらドラッグし、青い線を「コネクションインスペクタ」内の「playSound」横の○につなげてください（**図 step5-1**）。表示されたイベント発動条件のメニューは「Touch Up Inside」を選びましょう（**図 step5-2**）。

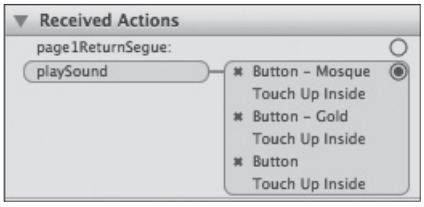
step5-1



２つの分岐用のボタンを同じように「playSound」へつなげれば効果音を鳴らすための設定は完了です。「コネクションインスペクタ」は**図step 5-3**のようになったはずです。実行してページ移動時に効果音が鳴ることを確認しましょう。

step5-2

step5-3



分岐ができると、可能性が広がる

さて、これで書籍の流れを分岐させることができるようになりました。分岐ができるとアプリの可能性が広がります。今月作った「Page2B」に、さらに続きのページを追加してみるのもよいでしょう(**図1**)。こうすると1つのアプリに2つの物語が入っているような状態になりますね。1ページ目のボタンを増やして目次のようにし、どの話を読むのか選ばせることも可能です。

また、より多くの分岐を設定し、複雑な変化をする物語を紡ぐことも可能でしょう(**図2**)。

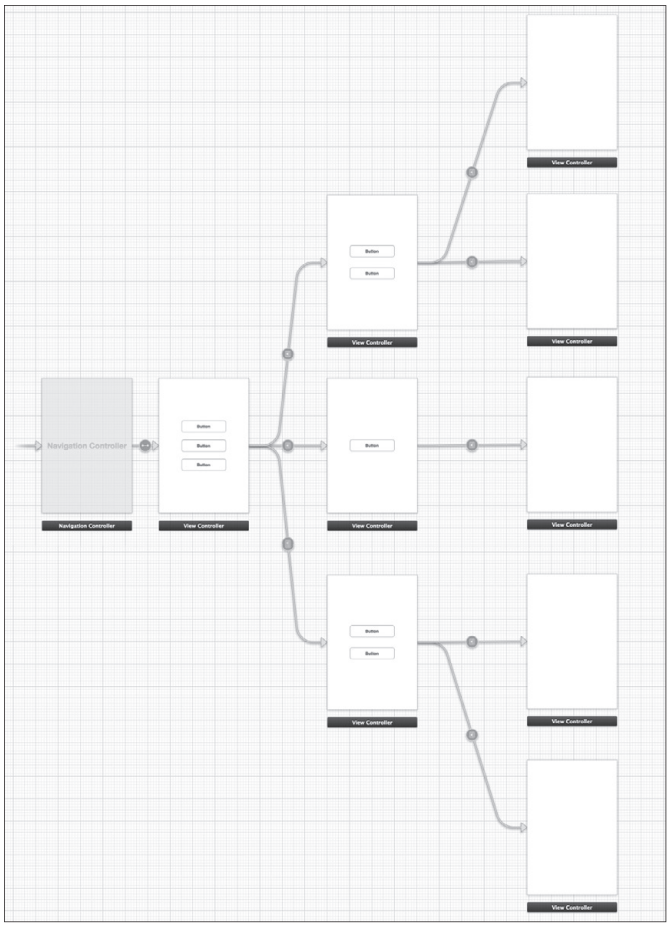
物語の先の展開を選ばせる手法もいろいろ考えられます。たとえば推理小説にすれば、読者の推理によって話が変わるようにすることもできるでしょう。ノベルゲームと言われるゲームアプリに仕上げることも不可能ではありません(**図3**)。本文を出題文にし、選択肢を答えにすることでクイズゲームにもできます(**図4**)。

ブックアプリをメインテーマとしたこの連載ですが、“インタラクティブ性”という武器を手に入れるとデジタルコンテンツの可能性は大きく広がることがおわかりいただけたのではないでしょうか。あなたのアイデア次第で可能性は無限大なのです！

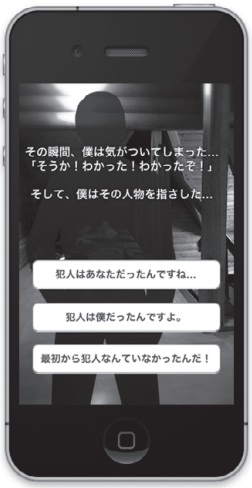
ということで、次回はよりアプリの可能性を広げるための機能を追加したいと思います。SD

▼図1　分岐の続きを追加したイメージ

▼図2　分岐を使って物語に変化をつける



▼図3　読者が参加できる推理小説



▼図4　クイズゲーム

本連載のサポートページで記事の補足説明をしています。あわせてご活用ください！
› http://www.gimmiq.net/p/sd.html

いたのくまんぼう／Itano Kumanbow
Twitter @Kumanbow

神奈川工科大学非常勤講師。リオさんとはGimmiQ名義で「MagicReader」（手を使わずにページがめくれる電子書籍ビューワ）をリリース。個人ではNinebonz名義で「Crop It Cam!」（おしゃれな切り抜き写真カメラ）、「i列車の車窓からーそうだ! 京都行こう!ー」（バーチャル旅行アプリ）など。アプリ紹介サイト「あぷまがどっとねっと（http://appmaga.net/）」の技術サポータ。

リオ・リーバス／Leo Rivas
Twitter @StudioLoupe

iOSアプリ開発を中心に電子絵本作家・漫画家として活動中。代表作は、KDDI株式会社に社内導入され、世界で40万ダウンロードを記録する革新的な電卓アプリ「FusionCalc」と、国連主催のWSA Mobile 2013を受賞した、顔の動きで電子書籍が読める「MagicReader」。電子絵本はiBookstore/Kindleストア共に児童書カテゴリ総合1位を獲得。

第42回

[アプリ開発2013] 3 手書きメモアプリを作成しよう

モバイルデバイス初のオープンソースプラットフォームとして、エンジニアから高い関心を集めるGoogle Android。いち早くそのノウハウを蓄積したAndroidエンジニアたちが展開するテクニックや情報を参考にして、大きく開かれたAndroidの世界へ踏みだそう！

野田 悟志　NODA Satoshi
日本Androidの会 神戸支部、GDG神戸
URL http://kobegdg.blogspot.jp/ Mail scarviz@gmail.com

はじめに

こんにちは。日本Androidの会神戸支部の野田悟志です。前々回、前回に引き続き、みなさんが今からAndroid開発を始めるうえで、楽しめるような内容をお伝えします。

今回は実際のアプリ開発により役立つように、Viewをカスタマイズして「手書きメモ」アプリを作成したいと思います。また、Google I/O 2013で紹介された「Navigation Drawer」というスライド式のオプションパネルも組み込んでいきます。カスタマイズや新しい機能を組み込むとなると難しいように思われるかもしれませんが、やってみるととても簡単で、実装方法がわかれば、出来上がったアプリを改造や改良したり、一部の機能を流用して新しいアプリを開発したりできると思います。

そのため、本稿では実際に一緒に開発しながら読み進めていく形式で記載しています。細かい部分までは記載できませんが、今回作る手書きメモアプリのサンプルプログラム[注1]のほうに、段落ごとのファイルなどを用意していますので、そちらも一緒にご参照ください。

注1）本誌サポートサイトからダウンロードできます。
URL http://gihyo.jp/magazine/SD/archive/2013/201311/support

手書き部分（カスタムView）を実装する

まずは手書き部分のカスタムViewを作成していきます。後で使い回しができるように、カスタムViewをFragment（後述）で表示するようにしたいと思います。前回までの記事を参考に、プロジェクトを新規作成してください。今回、アプリケーション名、プロジェクト名は「HandWriting」、パッケージ名は「sd201311.handwriting」、最小SDKバージョンは14（Android 4.0）としています。ActivityはBlankActivityで、名前もそのままで構いません。

カスタムViewをFragmentで表示させる

Fragmentとは、ビューとロジックを持ち、Activityに対して設置したり、切り離したりできるようになっているものです。Fragmentを使うことの利点は、たとえば、1つのActivityに対して複数の機能が存在する場合、そのActivityにロジックを書いていくと可読性が下がったり、テストがしづらくなります。そこで各機能を1つ1つFragmentにしていくと、Activityのロジックはすっきりし、後で使い回しがしやすくなります。

それではカスタムViewのクラスと、Fragmentのクラスをそれぞれ用意しましょう。Eclipseのパッケージエクスプローラ上から、[src]→

▼リスト1　カスタムViewの配置（main_fragment.xml）

```
<LinearLayout xmlns:android=
  "http://schemas.android.com/apk/res/android" …略… >
    <!-- カスタムView -->
    <sd201311.handwriting.View.DrawView ――――❶
    …略… />
</LinearLayout>
```

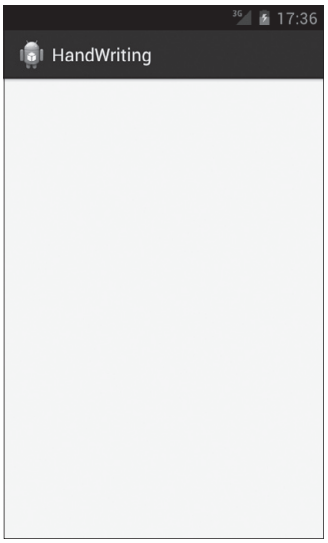
▼リスト2　Fragmentのレイアウト設定（WritingFragment.java）

```
public View onCreateView(LayoutInflater inflater,
  ViewGroup container, Bundle savedInstanceState) {
    return inflater.inflate(R.layout.main_fragment,
      container, false); ――――――――❶
```

▼リスト3　Fragmentの配置（activity_main.xml）

```
<fragment
    …略…
        android:name=
          "sd201311.handwriting.Fragment.WritingFragment" />
```

▼図1　カスタムViewの表示（初期状態）



［sd201311.handwriting］を右クリック→［New］→［Folder］を選択して「View」という名前のフォルダを追加してください。さらにそのフォルダを右クリック→［New］→［Class］を選択し、今度はクラスを追加します。クラス名を「DrawView」、ベースクラスを「android.view.View」とします。このクラスに引数2つのコンストラクタ（DrawView(Context context, AttributeSet attrs)）を実装してください。

同様の手順でFragmentクラスを追加します。「Fragment」というフォルダを追加し、クラス名を「WritingFragment」、ベースクラスを「android.app.Fragment」とします。そして、WritingFragmentクラスをパッケージエクスプローラ上で選択した状態で、メニューの［Source］→［Override/Implement Methods］から、onCreateViewメソッドを選択して追加してください。今後、オーバーライドメソッドを追加する場合は同手順で追加してください。

Fragmentのレイアウトファイルとして、layoutフォルダに「main_fragment」という名前のXMLファイルを追加します。このXMLに**リスト1-❶**のように、「パッケージ名.クラス名（DrawView）」という形式のタグを記述することで、カスタムView（DrawView）を配置します。このレイアウトをWritingFragmentクラスに反映させるには、**リスト2-❶**のように実装してください。

ActivityにFragmentを設置するには、activity_main.xmlを**リスト3**のようにfragmentタグでWritingFragmentを指定します。

これでカスタムViewを表示できるようになりました。一度実行してみましょう。実行すると**図1**のように表示されると思います。LogCatにはエラーは出ていませんが、何も実装していないので、真っ白だとしかわかりませんね。

試しに、main_fragment.xmlのカスタムViewタグに「android:background="#ff0000"」を追加して実行してみましょう。今度は真っ赤になっていると思います。ちゃんとカスタムViewが表示できているようですね（確認したらこのコードは消しておいてください）。

カスタムViewで手書き処理を実装する

DrawViewクラスにオーバーライドメソッドとして、onDrawメソッドとonTouchEventメソッドを追加してください（前述の手順同様に）。onDrawメソッドは描画処理を、onTouchEvent

▼リスト4　手書き処理の実装（DrawView.java）

```
Paint mCnvsPaint = new Paint();  // Canvas描画用
private int mDrawColor = Color.WHITE;  // 描画色
private int mStrokeWidth = 5;  // 描画線の幅
ArrayList<Point> mDrawPoint = new ArrayList<Point>();  // 描画位置

public DrawView(Context context, AttributeSet attrs) {
    super(context, attrs);
    mDrawColor = Color.RED;
    mStrokeWidth = 5;

    // Canvas描画の設定 ―――――――――――――――――❸
    mCnvsPaint.setAntiAlias(true);
    mCnvsPaint.setColor(mDrawColor);
    mCnvsPaint.setStrokeWidth(mStrokeWidth);
    mCnvsPaint.setStyle(Paint.Style.FILL);
}

@SuppressLint("DrawAllocation")
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    Point back = new Point(-1, -1);

    for(Point item : mDrawPoint){
        if (item.x >= 0) {
            if (back.x < 0) { back = item; }
            canvas.drawLine(back.x, back.y, item.x, item.y, mCnvsPaint); ――――❹

            int circle = mStrokeWidth/2;
            canvas.drawCircle(item.x, item.y, circle, mCnvsPaint); ――――――――❺
        }
        back = item;
    }
}

@Override
public boolean onTouchEvent(MotionEvent event) {
    mDrawPoint.add(new Point((int)event.getX(),(int)event.getY())); ――――――❶

    if (event.getAction() == MotionEvent.ACTION_UP) {
        mDrawPoint.add(new Point(-1, -1)); ――――――――――❷
    }

    // 再描画
    invalidate();
    return true;
}
```

メソッドは画面のタッチ時処理を行います。コンストラクタでの初期設定と、それぞれの処理を実装すると、**リスト4**のようになります。importは適宜追加していってください。

onTouchEventメソッドではタッチした座標をmDrawPointに追加していきます（**リスト4-❶**）。もし指を画面から離した場合は-1の値をmDrawPointに追加することで、連続した座標位置との区別ができるようにしています（**リスト4-❷**）。最後にinvalidateメソッドを呼ぶことで描画処理イベントが発生し、onDrawメソッドが呼ばれるようになります。

onDrawメソッドでは、mDrawPointに格納した座標をcanvas（画面）に描画していきます。描画する色や形や幅はmCnvsPaintで指定しています（**リスト4-❸**）。**リスト4-❹**では、指定した

2点の座標を直線でつなぐように描画します。**リスト4-❺**では直線の2分の1の幅を半径として円描画することで、曲線や曲がり角などを滑らかな線に見せるようにしています。

ここまで実装ができたら、実行してみましょう。画面を指でなぞってみてください。赤い線が描画されると思います(**図2**)。

カスタム属性を作成する

手書きのカスタムViewでは描画する色や線幅をコンストラクタで指定していました。後で使い回す場合、その都度クラスを触ると無駄なテストが発生してしまうので、レイアウトファイルから色や線幅を指定できるように、カスタム属性を作成しましょう。

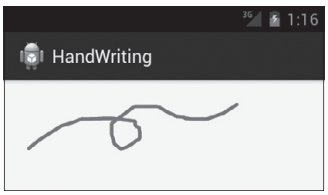
valuesフォルダに「attrs」という名前のXMLファイルを追加します。resourcesタグの中を**リスト5**のように実装してください。colorが色を、stroke_widthが線幅を指定する属性になります。

作成したカスタム属性をレイアウトファイルで使用するには、main_fragment.xmlに**リスト6**を追加します。LinearLayoutタグの中で、「drawAt」という名前空間で指定し(**リスト6-❶**)、カスタムViewタグでcolorとstroke_widthの値を指定しています(**リスト6-❷、❸**)。

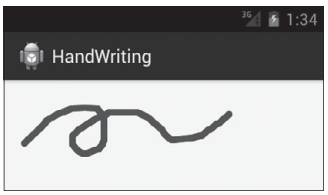
レイアウトファイルで設定した値を読み込むには、DrawViewクラスのコンストラクタで、描画色と描画線の幅の設定をしていた個所を、**リスト7**のように置き換えます。**リスト7-❶**でカスタム属性の設定値を読み込みます。**リスト5**のdeclare-styleableタグで設定した名前(DrawViewAttr)に対応した配列が取得されます。定義順でインデックスが振られるので、インデックス0がcolor、インデックス1がstroke_widthとなります。colorは色情報なのでgetColorメソッド(**リスト7-❷**)、stroke_widthは整数値なのでgetIntメソッド(**リスト7-❸**)で設定値を取得してください。最後にrecycleメソッドで破棄するのを忘れないようにしましょう(**リスト7-❹**)。

ここまで実装できたら、実行してみましょう。

▼図2　赤線を描画



▼図3　赤線(太)を描画



▼リスト5　カスタム属性の定義(attrs.xml)

```
<declare-styleable name="DrawViewAttr">
    <attr name="color" format="color" />
    <attr name="stroke_width" format="integer" />
</declare-styleable>
```

▼リスト6　カスタム属性の使用(main_fragment.xml)

```
<LinearLayout
    xmlns:drawAt="http://schemas.android.com/apk/res/sd201311.handwriting" ―――――❶
    …略… >
    <sd201311.handwriting.View.DrawView
        …略…
        drawAt:color="#AA0000" ―――――❷
        drawAt:stroke_width="10"/> ―――――❸
```

▼リスト7　カスタム属性の設定値読み込み(DrawView.java)

```
// attrs.xmlに定義したスタイル
TypedArray drawAt = context.obtainStyledAttributes(attrs, R.styleable.DrawViewAttr); ―――――❶
mDrawColor = drawAt.getColor(0, Color.WHITE); ―――――❷
mStrokeWidth = drawAt.getInt(1, 5); ―――――❸
// 不要なので破棄する
drawAt.recycle(); ―――――❹
```

色は同じ赤ですが、線幅は太くなっているのがわかると思います(**図3**)。

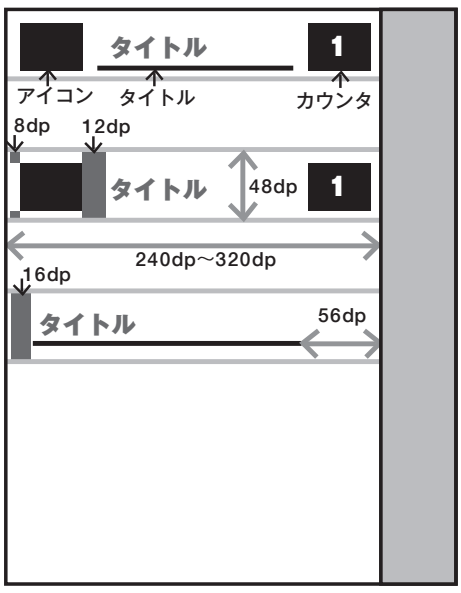
画像ファイルを読み込めるようにする

手書きメモなので、保存時は画像ファイルとして出力し、編集時はその画像ファイルを読み込むようにすれば良いと思います。画像ファイルの入出力処理などはWritingFragmentクラスに実装するとして、読み込んだ画像を描画する処理はDrawViewクラスに実装しましょう。

WritingFragmentクラスでの画像の入出力処理や、それらの処理をMainActivityのActionBar[注2]から呼び出したりする処理は、本稿では省略します。ここまでの内容に画像入出力処理を加えたものを、サンプルプログラムの「HandWriting_Chapter3」に用意しています。以降の解説では、画像入出力処理の実装が終わっていることを前提に進めますので、このサンプルプログラムをご用意ください。

注2) ActionBarについてはSoftware Design 2013年10月号の本連載記事をご覧ください。

▼図4　Navigation Drawerの概要



Navigation Drawerを実装する

皆さんはGoogle+やFacebook、Evernoteのアプリを使っていますか？　これらのアプリには、画面の左端から右へスワイプしたり、ActionBarのアプリアイコンをタッチすると、横からリスト項目のパネルがスライドしてくるようになっていますね。このスライド式のオプションパネルは今までは各アプリが独自で実装していたため、若干動きが違っていたりします[注3]。

このスライド式オプションパネルは「Navigation Drawer」という名前で正式にAndroidのデザインパターンとして取り入れられ、Google I/O 2013で紹介されました[注4]。

今回はNavigation Drawerの実装方法を簡素化するため、保存した画像ファイル名のリストを表示して、選択することで読み込み処理を行うようにしてみたいと思います。

Navigation Drawerの概要

Navigation Drawerを実装する前に、まずはNavigation Drawerがどういうものか押さえておきましょう(**図4**)。Navigation Drawerの項目は次の3要素から構成されます(タイトル以外はオプションです)。

- タイトル
- アイコン(タイトルの前に表示するもの)
- カウンタ(メールの新着数を表示するなどのカウンタ)

Navigation Drawerのレイアウトガイドライン[注5]では次のように定められています。

- 項目の幅は240dpから320dpの間に収めな

注3) 8月時点では、Google+は画面に覆いかぶさるようにパネルを表示し、Facebook、Evernoteでは画面を押し出すようにパネルを表示しています。ほかにも動きが違う点がありますね。

注4) https://developers.google.com/events/io/sessions/326301704

注5) http://developer.android.com/intl/ja/design/patterns/navigation-drawer.html#Style

ければならない
- 項目の高さは48dpを下回ってはならない
- タイトルは左端から16dp空ける。タイトルの前にアイコンが存在する場合はアイコンから12dp空ける
- タイトルは右端から最小56dp分マージンをとる
- アイコンやカウンタは周囲から8dp空ける

また、Navigation Drawerを表示してもActionBarには影響しない(覆いかぶさらない)ため、ユーザがActionBarの項目をNavigation Drawerとは関係のないものだと判断できない可能性があります。そのため、Navigation Drawerの表示中には、ActionBarの項目を隠すようにする必要があります。ただし、検索などのどこでも使うような項目や、ナビゲーションの対象になる設定やヘルプなどのメニューは、残しておいても構わないとのことです。

Drawer Layoutを実装する

Navigation DrawerのリストはDrawer Layoutを使用して実装します。activity_main.xmlを**リスト8**のように置き換えます。ここでいくつか気をつけなければならない点があります。

メインになるコンテンツはDrawer Layoutの最初の子にする必要があります(**リスト8-❶**)。また、layout_width、layout_heightは「match_parent」を指定しなければなりません。

Navigation Drawer の ListView の layout_gravityは、horizontalな値を設定しなければなりません。もしRTL言語[注6]をサポートする場合は、「start」を指定する必要があります(**リスト8-❷**)。また、layout_widthには240dp～320dpの値を(**リスト8-❸**)、layout_heightは「match_parent」を指定する必要があります(**リスト8-❹**)。

注6) アラビア語などの、テキストを右から左方向に記述する言語のこと。

▼リスト8　Drawer Layoutの実装(activity_main.xml)

```
<android.support.v4.widget.DrawerLayout
    …略… >
    <fragment …略… /> ―――――――――――――❶
    <!-- Navigation Drawerのリスト -->
    <ListView
        android:layout_width="240dp" ―――――❸
        android:layout_height="match_parent" ―――❹
        android:layout_gravity="start" ―――――❷
        …略… />
```

リストのレイアウトを追加したので、次はリスト項目のレイアウトも作成しておきましょう。layoutフォルダに「drawer_list_item」という名前でXMLファイルを追加してください。リストにタイトル以外を表示できるようにするには、カスタムAdapterクラスを用意します。クラス名を「NaviDrawerAdapter」、ベースクラスを「android.widget.ArrayAdapter<T>」として追加してください。よくあるリスト実装なので、本稿での説明は省略します。サンプルプログラムを参照してください。

リストの初期化はActivityのonCreateメソッドで行います。**リスト9**のように追加してください。

リスト選択時処理を実装する

リスト9-❶のDrawerItemClickListenerを実装し、リストの項目を選択したときの処理を行います。項目選択時には、カスタムViewへ画像をロードさせ、ActionBarのタイトルを画像ファイル名に置き換える処理を実装しましょう。これらの中身についてはサンプルプログラムを参照ください。

ActionBarのアプリアイコンでNavigation Drawerを開閉する

ActionBarのアプリアイコンをタッチすることで、Navigation Drawerのリストをオープン／クローズさせるには、ActionBarDrawerToggleを使うことで実装できます。ActionBarDrawerToggleの実装はMainActivityクラスのonCreateメソッドで行います(**リスト10-❶**)。

リストのオープン／クローズ時にインジケー

▼リスト9　リストの初期化（MainActivity.java）

```
protected void onCreate(Bundle savedInstanceState) {
    …略…
    // Navigation DrawerのListViewの生成
    mDrawerList = (ListView) findViewById(R.id.navi_drawer);
    // リスト項目選択時イベントの登録
    mDrawerList.setOnItemClickListener(new DrawerItemClickListener());  ❶
    // DrawerLayoutの取得
    mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
    …略…
}

private void ReLoadFileList() {
    …略…
    ArrayList<BindData> dataObj = new ArrayList<BindData>();
    for(int i = 0; i < mFileNameList.length; i++){
        // 画像ファイル名をタイトルに、サンプルとして
        // ランチャーアイコン画像をタイトルアイコンに設定
        dataObj.add(new BindData(mFileNameList[i],R.drawable.ic_launcher));
    }

    // リストに再設定する
    mDrawerList.setAdapter(new NaviDrawerAdapter(getApplicationContext(), dataObj));
}
```

▼リスト10　ActionBarDrawerToggleの実装（MainActivity.java）

```
protected void onCreate(Bundle savedInstanceState) {
    …略…
    // ActionBarDrawerToggleの生成
    mDrawerToggle = new ActionBarDrawerToggle(  ❶
        this, mDrawerLayout,
        android.R.drawable.ic_menu_sort_by_size,  ❷
        R.string.drawer_open, R.string.drawer_close ){  ❸
        // リスト非表示時処理
        public void onDrawerClosed(View view) { …略… }  ❹
        // リスト表示時処理
        public void onDrawerOpened(View drawerView) { …略… }  ❺
    …略…
```

タ（drawer indicator）として使用する、Navigation Iconが公式サイトで用意されています[注7]。これをダウンロードし、**リスト10-❷**を置き換えてください。また、オープン／クローズについての説明文を設定する必要があります。strings.xmlに任意の説明文を定義し、ActionBarDrawerToggleのコンストラクタの引数に設定します（**リスト10-❸**）。

オープン／クローズ時処理に、ActionBarのタイトルを画像ファイル名に切り替えるのと、ActionBarの項目の表示切替処理も実装します（**リスト10-❹、❺**）。表示切替処理はonPrepareOptionsMenuメソッドをオーバーライドして実装しましょう。

ActionBarのアプリアイコンのタッチ時処理は、onOptionsItemSelectedメソッドの最初で行い、ActionBarDrawerToggleのonOptionsItemSelectedメソッドを呼ぶようにします。何も処理されない（アイコンタッチ時でなかった）場合は、そのまま従来の処理を実施するようにします（**リスト11**）。

DrawerLayoutにsyncする処理（**リスト12-❶**）、ActionBarDrawerToggleのデバイスの状態（回転状態など）の変更時処理（**リスト12-❷**）を、onPostCreateメソッドとonConfigurationChangedメソッドをオーバーライドして、それぞれの処理を呼び出すように実装します。

ここまで実装できたら、実行してみましょう。ActionBarのアプリアイコンでNavigation Drawerを開閉できましたか？　画像を保存すると、リストに画像ファイル名とタイトルアイ

注7）http://developer.android.com/downloads/design/Android_Navigation_Drawer_Icon_20130516.zip

▼リスト11　アプリアイコンのタッチ時処理（MainActivity.java）

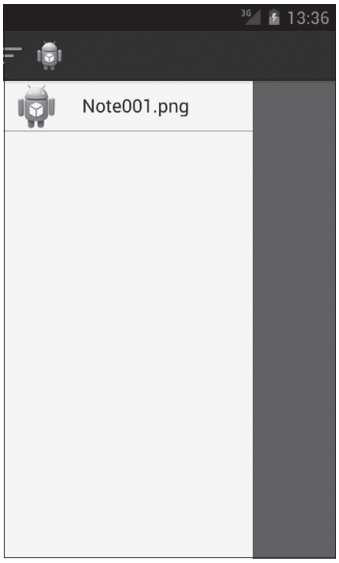
```
public boolean onOptionsItemSelected(MenuItem item) {
    // アイコンボタン押下時処理
    if (mDrawerToggle.onOptionsItemSelected(item)) {
        return true;
    }
…略…
```

▼リスト12　sync処理とデバイス状態の変更時処理（MainActivity.java）

```
@Override
protected void onPostCreate(Bundle savedInstanceState) {
    super.onPostCreate(savedInstanceState);
    mDrawerToggle.syncState(); ❶
}

@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    mDrawerToggle.onConfigurationChanged(newConfig); ❷
}
```

▼図5　Navigation Drawerの表示



コン（サンプルとしてランチャーアイコン）が表示されると思います（図5）。

最後に

いかがでしたか？　実装手順がわかれば、お決まりのパターンなので、意外と簡単に実装できたと思います。今回実装したものは、ほかのアプリでも使い回しができるようなものなので、ぜひ使ってみてください。また、今回のアプリ自体に何か手を加えてみるのも面白いと思います。たとえば、消しゴム機能をつけたり、Navigation Drawerの画像ファイルリストは、展開行を1つ用意して、その展開行で表示するなどです。

本稿が少しでも皆さんのお役に立てれば幸いです。これからもAndroid開発を楽しんでくださいね！SD

Column

開発を始める、その前に

早く何か作ってみたい！と思ってる方も少なくないと思います。しかし、頭の中で思いつくままに作り上げたアプリは、最初に思い浮かべていたイメージと全然違う「コレジャナイ」感満載の残念なアプリになってしまうことが多々あります。

そういう「コレジャナイ」アプリを作ってしまう原因は、最初のイメージが頭の中にしかないので、実装していくうちに変わってしまうことにあります。そのため開発を始める前に、一度どういうアプリを作るのかを書き出しておきましょう。どういうレイアウトか、どういう機能があるか、どういう動作をするのか、という点をメモ書きでも良いので簡単に書き出しておきます。今回の「手書きメモ」アプリでは、筆者はメモ帳へ手書きで書き出しました。レイアウトしやすいツールを使ったり、Excelやパワーポイントを使ったりする方もいるそうです。自分が使いやすいツールを利用すると良いと思います。

こうしておくと、機能追加やレイアウト変更時に、後で見返してみて、ほかに実装する機能などと矛盾しないか、全体的にまとまっているかなどがチェックできます。

野田 悟志（のだ さとし）

日本Androidの会 神戸支部、GDG神戸で活動中。最近はGo言語にもはまり、GDG神戸ブログ（http://kobegdg.blogspot.jp/）でいくつか記事を書いています。良かったら見てください。

ハイパーバイザの作り方

ちゃんと理解する仮想化技術

第14回 ハイパーバイザの作り方・総集編（下）

浅田 拓也（ASADA Takuya）Twitter @syuu1228

はじめに

前回は、Virtqueueとこれを用いたNIC（virtio-net）の実現方法について解説しました。前号に引き続き総集編として、2回にわたって仮想化システムの全体像を振り返ります。

x86アーキテクチャにおけるデバイスI/O

I/OマップドI/Oとメモリマップド I/Oの2種類の方式があり、通常アーキテクチャごとにどちらかの方式をとります。しかし、x86アーキテクチャでは、歴史的な事情により両方式を併用しています。

I/OマップドI/Oでは、メモリ空間とは独立したデバイス専用のアドレス空間（I/O空間）が存在しており、ここに各デバイスのハードウェアレジスタを割り付けます。なお、どのデバイスをどの番地にするかは、固定的に決まっているアーキテクチャと動的に決まるアーキテクチャとがあります。I/O空間へは専用の命令（IN命令、OUT命令）を用いてアクセスを行います。

一方、メモリマップドI/Oでは、各デバイスのハードウェアレジスタをメモリ空間の一部に割り付けます。こちらも、どのデバイスをどの番地にするかは、固定的に決まっているものと動的に決まるものとがあります。ハードウェアレジスタへのアクセスにはMOV命令など通常のメモリアクセスと同様の手順を用います。

ゲストマシン上で各種デバイスをサポートするには、この2つの種類のI/Oを仮想化し、アクセスされたデバイスに応じたエミュレーション処理を行う必要があります。

VT-xにおけるI/Oエミュレーション

総集編・上で解説したとおり、次のようなループの繰り返しによりデバイスI/Oのエミュレーションを行います。

① [ゲスト] ゲスト環境上でデバイスへのI/Oが実行される
② [ゲスト] I/Oの実行を契機にVMExit発生
③ [ハイパーバイザ] アクセス先のデバイス、アクセス幅、アクセス方向、書き込み先・読み込み元などを特定
④ [ハイパーバイザ] デバイスI/Oのエミュレーション処理を行う
⑤ [ハイパーバイザ] VMEnterしてゲストを再開させる
⑥ [ゲスト] I/O実行の次の命令から実行再開

この処理は、ハードウェアレジスタへの読み書きが行われるごとに繰り返されます。したがって、1回のハードウェアアクセス処理に必要なハードウェアレジスタへのアクセス回数が多ければ多いほどオーバーヘッドが大きくなります。

なお、VT-xではI/OマップドI/Oとメモリマップド I/Oのどちらの方式にも対応していますが、処理の仕方が異なるため、次では2つに分けて解説します。

VT-xにおけるI/Oマップド I/Oのハンドリング方法

VT-xにおいてI/OマップドI/Oをハンドリングするには、まずVMCSへ設定を行ってI/Oポートへのアクセス時にVMExitを発生させる必要があります。設定には2通りあり、すべてのI/OポートへのアクセスでVMExitを発生させる設定と、特定のI/OポートへのアクセスにのみVMExitを発生させる設定です。すべてのI/OポートへのアクセスでVMExitを発生させるには、VMCSのVM-Execution Control FieldsのUnconditional I/O exitingを1にします。

特定のI/OポートへのアクセスにだけVMExitを発生させるには、VMCSのVM-Execution Control FieldsのUse I/O bitmapsを1にして、VMExitを発生させるポートにビットセットしたビットマップテーブルを作成、VM-Execution Control Fields のI/O-Bitmap Address A・Bにアドレスをセットします[注1]。ゲストOSがデバイスドライバ経由でI/Oポートへアクセスを行うと、VMExit Reason 30（I/O Instruction）のVMExitが発生します。VMExit ReasonはVMCSのVM-Exit Information FieldsのVMExit Reasonフィールドに書かれています。

しかし、VMExit Reasonだけでは何番のI/Oポートへアクセスされているのか、アクセス方向が読み込みだったのか、あるいは書き込みだったのかがわかりません。これらの情報は、VM-Exit Information FieldsのExit qualificationフィールドで提供されます。ハイパーバイザはこの情報を見てどのようなハードウェアアクセスがあったのか把握し、これに応じた仮想デバイスのエミュレーション処理を行います。

VT-xにおけるメモリマップド I/Oのハンドリング方法

VT-xにおけるメモリマップドI/Oは、メモリ仮想化がソフトウェアで行われている（シャドーページング）か、ハードウェアで行われている（EPT）かで2通りあります。

ここでは、EPTによるハンドリング方法のみ紹介します。

EPT環境においてメモリマップドI/Oをハンドリングする場合は、デバイスがマップされたアドレスへのアクセスが発生した時に、ページへのアクセス違反でVMExitさせる必要があります。そのために、Extended Page Tableからデバイスがマップされたアドレスに対応するページテーブルエントリを探し、Read/Write権限を外します。これによってゲストマシンがデバイスがマップされたアドレスへアクセスした時にVMExit Reason 48（EPT violation）が発生するようになります。

ハイパーバイザはVMCSのVM-Exit Information FieldsのVMExit Reasonフィールドを確認してEPT violationによるVMExitだと確認し、VMCSのVM-Exit Information Fieldsにあ るGuest-physical addressを読み込みます。このアドレスがI/O先アドレスになります。さらに、VM Exit qualificationフィールドを参照するとアクセス方向を得ることができます。しかし、I/Oをエミュレーションするにはアクセスサイズ、データの書き込み先・読み込み元などの情報が足りません。

VMCSからこれらの情報を得ることはできないため、ハイパーバイザはVMExit時に実行していた命令[注2]をソフトウェアエミュレーションしてどのようなアクセスがあったか調べる必要があります。

x86アーキテクチャにおける割り込みのしくみ

外部割り込み、内部割り込み

割り込みは、外部割り込みと内部割り込みの2種類に分けられます。外部割り込みとは、ハードウェアからCPUへイベント通知を行うのに用いられます。外部割り込みは、さらにマスク可能な割り込みとソフトウェアからマスク不可能な割り込み（NMI）の2種類に分類されます。

内部割り込みとは、CPU内部の要因で発生する割

注1）I/O-Bitmap Aは0000h - 7FFFhまでの範囲を表すビットマップテーブルで、I/O-Bitmap Bは8000h - FFFFhまでの範囲を表すビットマップテーブルになっています。

注2）VMCSからゲストマシンのRIPを取得し、RIPが指すゲストメモリ上のアドレスにアクセスすることにより取得できます。

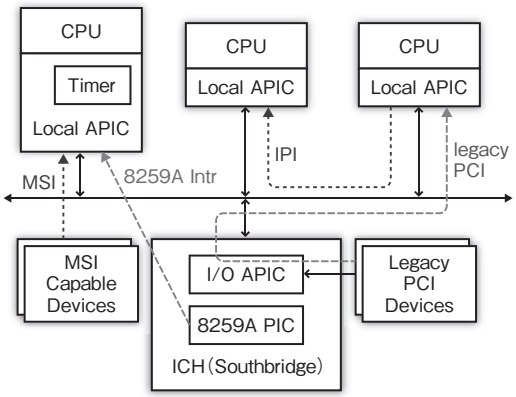
り込みのことです。この内部割り込みは、ソフトウェア割り込みと例外に分類されます。ソフトウェア割り込みとは、割り込みを起こす命令（INT命令）により発生する割り込みでシステムコールの実装に用いられます。例外とは、プログラムを実行した結果として何らかのエラーが発生したときなどに起きる割り込みです。

ベクタ番号とIDT

すべての種類の割り込みに0-255のベクタ番号が割り当てられます。

割り込みを使用するにはIDT（Interrupt Descriptor Table）と呼ばれる割り込みハンドラのアドレスを格納する最大255エントリのテーブル（配列）を作成し、IDTのアドレスをIDTRレジスタに設定する必要があります。割り込み発生時、CPUはIDTRの値からIDTを参照し、指定された割り込みハンドラを実行します。

▼図1　Local APICとI/O APIC、外部割り込みの関係



割り込みのマスク・アンマスク

EFLAGSレジスタのIFフラグをクリアすると割り込みを禁止、IFフラグをセットすると割り込みを有効にすることができます。プログラムからIFフラグをクリア／セットするには、CLI命令／STI命令を使用します。

外部割り込みと割り込みコントローラ

Pentium以降のx86アーキテクチャでは、外部割り込みの管理を行う割り込みコントローラはCPUごとに存在しCPUに内蔵されているLocal APICと、ICH（Southbridge）に搭載されているI/O APICから構成されています（**図1**）。

Local APICは、ローカル割り込みのベクタ番号設定、割り込みベクタ番号通知、EOI（割り込み終了）通知など、一般的な割り込みコントローラの役割を受け持ちます。また、タイマー／温度センサ／パフォーマ

表1　Local APICの主なレジスタ

IRR（Interrupt Request Register）	割り込み要求レジスタ（Read-only）：未処理の割り込みを管理するレジスタ。Local APICに外部割り込みが着信するたびにベクタ番号に対応するビットがセットされる
ISR（In-Service Register）	インサービスレジスタ（Read-only）：次に発行される割り込みの候補を管理するレジスタ。EOIレジスタへ書き込まれるとLocal APICによってIRRから最高優先度のビットがコピーされる
EOI（End Of Interrupt）	割り込み終了レジスタ（Write-only）：割り込み処理の終了をLocal APICに通知するためのレジスタ
TPR（Task Priority Register）	タスク優先度レジスタ（Read/Write）：外部割り込みに対する実行中タスクの優先度を設定するレジスタ。TPRに書き込まれた優先度より低い優先度の割り込みはマスクされる
PPR（Processor Priority Register）	プロセッサ優先度レジスタ（Read-only）：実際の割り込み着信時にマスクを行うか否かの判定に使われるレジスタで、TPRとISRに連動して更新される
Local APIC ID Register	LAPIC IDレジスタ（Read/Write）：システム全体でCPUを一意に特定するためのIDであるLAPIC IDを格納しているレジスタ。LAPIC IDはI/O APICから外部割り込みを転送するときやIPIを送るときなどに使用される
ICR（Interrupt Command Register）	割り込みコマンドレジスタ（Read/Write）：IPI（プロセッサ間割り込み）を送信するためのレジスタ
LDR（Logical Destination Register）	Logical APIC IDを指定
DFR（Destination Format Register）	Logical Destination Modeのモデルを指定（Flat model/Cluster model）

ンスモニタリングカウンタなどのデバイス、IPI（プロセッサ間割り込み）の送受信機能を内蔵しています。

一方、I/O APICは外部デバイスから割り込みを受け取り、I/O APIC上の設定に基づいて割り込み配信先のLocal APICを選び割り込みをリダイレクトする機能を受け持ちます。

このような構成を取ることによって、SMP環境下で複数のCPUで外部割り込みを処理できるようになっています。

Local APIC

表1に割り込みの処理で利用されるLocal APICの主なレジスタを示します。

割り込み着信時のLocal APICおよびCPUの挙動を簡単にまとめると、次のよう流れになります。

①Local APICが割り込みを受信したら、IRRに対応するベクタ番号のビットをセットする。CPUが割り込みをブロックしている場合はここで処理は終わり
②IRRにセットされた最高優先度のビットをクリア、同じビットをISRにセット、同じ優先度の割り込みをCPUへ発行する
③CPUで割り込みハンドラが実行される
④CPUで実行された割り込みハンドラがEOIレジスタに書き込み、割り込み処理の終了を伝える
⑤EOIレジスタへの書き込みを受け取ると、ISRにセットされた最高優先度のビットをクリア。まだIRRにビットが残っていたら②から繰り返し

表2　I/O APICのRedirection Table Entry

63:56	Destination	宛先Local APICの指定
16	Mask	割り込みマスク
15	Trigger Mode	エッジトリガ／レベルセンシティブ
14	Remote IRR	レベルセンシティブモードで割り込み送信中／EOI着信済み
13	Interrupt Pin Polarity	レベルセンシティブモードでhigh/lowのどちらを割り込みリクエストとするか
12	Delivery Status	割り込みペンディング中かどうか
11	Destination Mode	Physical Mode／Logical Mode
10:8	Delivery Mode	Fixed／Lowest Priorityなど
7:0	Vector	ベクタ番号

ただし、タスク優先度／プロセッサ優先度という機能があり、これによって割り込みに対する実行中タスクの優先度が制御でき、タスクの優先度が受信した割り込みより高い場合、その割り込みはマスクされます。

I/O APIC

I/O APICには外部デバイスからの割り込み線が接続され、24本の割り込みをサポートしています。各割り込みの割り込み先はI/O APIC上のRedirection Tableに設定され、システムバスを通じてCPUのLocal APICへ転送されます。Redirection Tableの各エントリのフィールドの詳細を**表2**に示します。

Redirection Table EntryではDestination Modeを指定する必要があります。Destination ModeにPhysical Destination Modeを指定した場合、Destinationフィールドに Local APIC IDを指定することにより、宛先CPUを一意に特定します。

Logical Destination Modeを指定した場合、Destinationフィールドで複数の宛先CPU群をビットマスク指定します。

Logical Destination Modeにより複数のCPUが割り込み先に指定されたときの挙動については、Delivery Modeで設定します。Fixed Modeでは、Destinationに指定されたすべてのCPUに割り込みます。Lowestpriority Modeでは、Destinationのうち Local APICのTPRが最も小さいCPUへ割り込みます。TPRの値が最も小さいCPUが複数存在する場合は、ラウンドロビンで割り込みが分散されます。

APIC IDのアドレス幅は8bitですので、Logical Destination Modeでは最大8CPUしかサポートできません。このため、Nehalem世代よりx2APICと呼ばれる拡張版APICが導入され、アドレス幅は8bitから32bitへ拡張されました。

MSI・MSI-X割り込み

I/O APICを経由する従来のPCIデバイスの割り込みは、割り込みベクタの割り当てが物理配線に依存して

おり、限られたIRQを複数デバイスで共有していました。

この制限を解消するため、物理配線を用いずPCIバス経由のメッセージとして割り込みを送るMessage Signalled Interrupt(MSI)・Extended Message Signalled Interrupt(MSI-X)が導入されています。PCIではオプション機能として提供されていますが、PCI expressでは必須とされています。

MSI・MSI-X対応PCIデバイスでは、割り込みはIO-APICを経由せず直接Local APICに転送されます。このとき、宛先CPUの設定は各PCIデバイスのコンフィギュレーションスペースに設定されます。コンフィギュレーションスペース内の割り込みの設定フィールドでは、Redirection Table Entryに近い内容が割り込みごとに設定できます。

割り込みの仮想化

VT-x環境においては、内部割り込みはCPUがすべて処理を行うため、基本的にハイパーバイザが介入する必要がありません。一方、外部割り込みについては、ハイパーバイザの介入が必要となります。

CPUへの割り込みの挿入

仮想CPUで任意の割り込みを発生させるには、VMCSのVM-Entry Control FieldsにあるVM-entry interruption-information fieldにベクタ番号と割り込みタイプを書き込みます(**表3**)。ただし、このフィールドに値をセットして外部割り込みを発生させるだけでは、Local APICのレジスタ値は適切に更新されず、ハイパーバイザが新しい値を計算しセットする必要があります。

▼表3　VM-entry interruption-information field

ビットポジション	内容
7:0	ベクタ番号
10:8	割り込みタイプ:通常は0(外部割り込み)を使用
11	スタックに例外のerror codeをpush
31	有効化ビット

外部割り込みの仮想化

外部割り込みは、ハイパーバイザが割り込みを送り込みます。これは、デバイスがソフトウェア的に、ハイパーバイザ内に実装されているためです。

I/O APICを通して割り込む場合

まずあらかじめ、ゲストOSが割り込みを初期化するときにI/O APICのRedirection Table Entryへ宛先Local APICが設定されます。

デバイスエミュレータからの割り込みを受け、ハイパーバイザはデバイスに対応するRedirection Table Entryの値を参照し、宛先の仮想CPUを選びます。宛先の仮想CPUが決定されたら、ハイパーバイザは宛先CPUのLocal APICのIRRレジスタを更新し、VMCSに割り込みの挿入を設定します。割り込み挿入が設定された仮想CPUがVMEnterされると、以降は内部割り込みと同様に、実機とほぼ同じ手順で割り込みの受付が行われて割り込みハンドラが起動されます。

EOI書き込みに関しては、Local APICのEOIレジスタへのアクセスを、ハイパーバイザが介入してエミュレーションを行う必要があります。

まとめると、外部割り込みを仮想化するには、ハイパーバイザでI/O APIC・Local APICのエミュレーションを行い、仮想CPUへ割り込みを挿入する必要があります。

MSI/MSI-X割り込みを用いて割り込む場合

MSI/MSI-X割り込みを用いる場合の違いは、割り込み先がI/O APICのRedirection Table Entryに書いてあるのではなく、PCI Configuration Spaceに書いてある、という点だけです。これを仮想化する場合、ハイパーバイザで宛先の仮想CPUを選択するときの参照先が変わりますが、あとはI/O APICを通じた割り込みと同じです。

Local APICの仮想化

Local APICはメモリマップドI/Oでアクセスするため、基本的には通常のメモリマップドI/Oの仮想化手法を使うことができます。しかし、高速化のた

めにVT-xにはLocal APICへのアクセスを特別扱いしてハンドルする機能が用意されています。

APIC access VMExit

APIC access VMExitを使うには、VMCSのVM Execution fieldsでVirtualize APIC accessesを有効化し、Local APIC用にアロケートした物理ページのアドレスをVMCSのVM-Execution fieldsへ設定します。これにより、APIC access pageへアクセスが発生した際に、VMExit Reason 44（APIC access）が発生するようになります。このとき、VM Exit qualificationを参照すると、アクセスのあったレジスタとアクセス方向がわかります。

これだけの情報では命令エミュレーションが避けられないレジスタもありますが、EOIレジスタに関しては使い方がきわめて限定的に決まっているので命令エミュレーションをスキップしてハンドリングを完了させることができます。これだけでもそれなりのオーバーヘッド軽減が見込めるようです。

TPR shadow

TPRレジスタのしくみ上、VMExitを用いたハイパーバイザからの介入が必要なのは、ある値より優先度を下げる方向の書き込みだけです。

それ以外のケースでは、ゲストOSから読み書きが正常に行えさえすればよく、VMExitを発生させる必要がありません。

このような挙動を実現させるため、VT-xではTPR shadowという機能を用意しています。TPR shadowを使うには、VMCSのVM-Execution fieldsでTPR shadowを有効にし、Virtual APIC Pageと呼ばれる物理ページをTPRの値を格納する場所として用意し、VMExitを発生させるしきい値をTPR thresholdというパラメータで指定します。

ゲストからTPRへのアクセスが発生した時、TPRの値がTPR thresholdを下回った時のみVMExit Reason 43 TPR below thresholdでVMExitします。

I/O APICの仮想化

I/O APICもメモリマップドI/Oでアクセスされますが、Local APICと異なり特別な仮想化支援機能は存在しないため、通常のメモリマップドI/Oの仮想化手法を使用して仮想化します。

CPUへの割り込みの挿入

仮想CPUで任意の割り込みを発生させるには、VMCSのVM-Entry Control FieldsにあるVM-entry interruption-information fieldにベクタ番号と割り込みタイプを書き込みます。これにより、VMEntry時にゲストマシンはIDTで指定した割り込みハンドラを実行します。

ただし、このフィールドに値をセットして外部割り込みを発生させるだけではLocal APICのレジスタ値は適切に更新されないため、VMEntry前にハイパーバイザが新しい値を計算しセットする必要があります。

まとめ

今回は、今までの総集編として仮想化システムの全体像を振り返りました。

第1回は、「x86アーキテクチャにおける仮想化の歴史とIntel VT-x」でした。第2回では「Intel VT-xの概要とメモリ仮想化」を扱い、第3回から5回までは、I/O仮想化として「デバイスI/O編」と「割り込み編をとりあげました。第6回から10回までは、Intel VT-xを用いたハイパーバイザの実装を、「仮想CPUの実行処理」と「vmm.koによるVMEntry」「vmm.koへのVMExit」、「ユーザランドでのI/Oエミュレーション」について解説してきました。第11回と12回までは、virtioによる準仮想化デバイスとして、「virtioの概要とVirtio PCI」、「VirtqueueとVitio-netの実現」まで言及してきました。

次回は、PCIパススルーとIntel VT-dについて解説します。SD

テキストデータならお手のもの

開眼シェルスクリプト

USP友の会／産業技術大学院大学　上田 隆一 UEDA Ryuichi　Twitter @ryuichiueda

第23回 文章の表記揺れ／綴りをチェックする ――コマンドを自作する時は単機能で

はじめに

前回は「文章を扱う」というお題でコマンドの操作をいくつか紹介しました。今回は「文章を扱う道具」、つまりコマンドをシェルスクリプトで作ってみます。

作るコマンドは語尾のチェックコマンドとスペルチェックのコマンドです。いずれのコマンドも、既存のコマンドをうまく組み合わせて、短いものを作ります。コマンド作者となるわけですから、GancarzのUNIX哲学[注1]を全部頭に叩き込んで、先にお進みください。とくに、

- 各プログラムが一つのことをうまくやるようにせよ
- 全てのプログラムはフィルタとして振る舞うようにせよ

が大事です。

注1）http://ja.wikipedia.org/wiki/UNIX哲学

環境など

筆者のMacに溜まった本連載の原稿をサンプルにして、文章をいろいろとチェックするコマンドを作っていきます。MacにはGNU sed(gsed)とGNU awk(gawk)がインストールされているものとします。**図1**に環境を示します。

原稿はテキストファイルです。reStructured Textという形式でマークアップされていますが、それはあまり気にしなくて大丈夫です。拡張子は**図2**のとおり「.rst」です。**図3**のように、原稿はだいたい30字ごとに改行を入れています。

コマンドなど、作ったものはディレクトリSD_GENKOUの下にbinというディレクトリを作ってそこに置くことにします。

ですます／だであるチェック

まず、表記ゆれの基本中の基本、「ですます調」と「だである調」のチェックを行うシェルスクリプトを作ってみましょう。「です。」「ます。」などの数を数え、次に「だ。」「である。」などの数を

▼図1　環境

```
$ uname -a
Darwin uedamac.local 12.4.0 Darwin Kernel Version 12.4.0: Wed May  1 17:57:12 PDT 2013; ➥
root:xnu-2050.24.15~1/RELEASE_X86_64 x86_64
$ gsed --version
gsed (GNU sed) 4.2.2
  (略)
$ gawk --version
GNU Awk 4.1.0, API: 1.0
Copyright (C) 1989, 1991-2013 Free Software Foundation.
```

数え、どちらも1以上だったら怪しいという簡素なものです。

まずは**リスト1**のように作ってみましょう。これだと、たとえば1行に「です。」が2回出てきても1とカウントされますが、自分で使うには十分でしょう。これを公開しようとすれば、いろいろ細かく修正が必要です。

コードの説明をしておくと、5行目で標準入力を$tmp-textに一度溜めています。$tmp-textは/tmp/下のファイルですが、このコードだと/tmp/下にファイルが残ってしまう可能性があります。不特定多数の人々が使うUNIX環境のときは、別のところに一時ファイルを置きましょう。今はあまりそういうこともないでしょうが、一応お断りを。また、5行目の**< /dev/stdin**は、このシェルスクリプトの標準入力を読み込むリダイレクトですが、書かなくてもcatが標準入力を読んでくれます。筆者はそれだとわかりにくいので、明記しています。

7、8行目で、正規表現を作ります。語尾はたくさん種類があるので、ここにずらずら並べておきます。全部網羅することは難しいですし、きりがないので自分の困らない範囲で列挙しておけば良いでしょう。ただ、まったく対処できないかと言うとそうでもなく、**図4**のようにワンライナーで語尾を抽出して、あとから解析することはできます。身も蓋もないことを言うと、形態素解析のコマンドをシェルスクリプトの中で使うと完璧に近いものができるかもしれません。それがシェルスクリプトの良いところですので、使えるものは何でも使いましょう。

では、deathmath1を実行してみましょう(**図5**)。原稿はですます調で書かれていますが、1.5%程度、ですます調に従っていない部分があるように見えます。

▼図4　語尾を抽出

```
$ cat ../*.rst |
gsed 's/....。/¥n&¥n/g' | grep 。|
sed 's/。.*/。/' | sort -u
（略）
(縦)軸。
(?)を。
)を作れ。
)を知る。
:私です。
```

▼図5　deathmath1の実行例

```
$ cat *.rst | ./bin/deathmath1
ですます 2468
だである 38
```

▼図2　原稿のファイル

```
$ ls *.rst
201201.rst        201210.rst        201306.rst
201202.rst        201211.rst        201306SPECIAL.rst
201203.rst        201212.rst        201307.rst
（略）
```

▼図3　原稿の中身

```
$ tail -n 5 201302.rst
lsとwcを使えば事足ります。captiveでないので、なんとかなります。

  今回は正直言いまして、
かなりエクストリームなプログラミングになってしまいましたので、
次回からはもうちょっとマイルドな話題を扱いたいと思います。
```

▼リスト1　語尾を数えるコマンド(deathmath1)

```
01 #!/bin/bash
02
03 tmp=/tmp/$$
04
05 cat < /dev/stdin > $tmp-text
06
07 death="(です。|ます。|でした。|ました。|でしょう。|ません。)"
08 da="(だ。|である。|ない。|か。)"
09
10 grep -E "$death" $tmp-text      |
11 wc -l                          |
12 tr -d ' ' > $tmp-death
13
14 grep -E "$da" $tmp-text         |
15 wc -l                          |
16 tr -d ' ' > $tmp-da
17
18 echo "ですます" $(cat $tmp-death)
19 echo "だである" $(cat $tmp-da)
20
21 rm -f $tmp-*
22 exit 0
```

コマンドを書き直して該当個所を出力させる

さて、だである調がちょっと混ざっているようなのですが、今度はどこを修正しなければならないのか知りたくなってきます。grepを使えば良いのですが、deathmath1では7、8行目で複数の語尾を指定しているので、これをいちいち手打ちするのは面倒くさいし、いちいち覚えてられません。コマンドを新たに作るか、deathmath1を拡張するか、どちらかをしたほうが良いでしょう。

ここで「コマンドの作り手」はとても悩みます。コマンドは単機能にしておくほうが、あとから手を入れるときに楽です。肥大化もしません。これはGancarzのUNIX哲学にもあります。たとえば、deathmath1を拡張するとなると、オプションを新たに設けなければいけませんし、追加個所と既存個所を分けるために、関数で分けなければなりません。つまり、余計な情報を追加しなければなりません。

一方でコマンドを新たに作るとなると、deathmath1の7、8行目にある変数をうまく共有するしくみが必要です。しかし今のところ、わざわざ辞書ファイルを外に出すほどのものでもありません。

筆者の出した答えは次のようなものです。

- 行数ではなく、当該個所を出力するように書き直す
- 行数を数えたかったら、ほかのコマンドで

つまるところ、筆者はdeathmath1が「作り過ぎ」だったと判断しました。せっかく作ったdeathmath1を放棄して、手戻りをします。普通、雑誌にコードを掲載するときはこういう放棄はしないものですが、こういう考え方でツールを改善するという例ですので容赦ください。

リスト1のようにgrepを使うと中間ファイルを作らざるを得ず面倒ですので、作りなおし版ではawkで印を付ける方式に変更します(**リスト2**)。

リスト2のコードについて補足説明しておきます。まず、6行目の**"$@"**は、deathmath2がもらったオプションをそのままawkに渡すための方法です。**"$*"**だと、複数のファイル名がオプションに入っている場合、うまくいきません。ファイル名がずらずら並んだ文字列を1個のオプションとみなすからです。**図6**の例では、「**201311.rst 201211.rst**」が1つのファイル名だと解釈されるため、catがエラーを出します。ところで、この方法はおもしろいことに、ファイル名を指定せずに標準入力からの文字列を入力しても動作します。このときは、**"$@"**が空になり、gawkはオプションなしと判断して標準入力を読みにいきます。

次に、検索で引っ掛ける文字列は3、4行目で、bashの変数として定義しています。これを7行目でgawkに渡しています。正規表現を変数に渡しているわけですが、**/**は不要なようです。

6行目のFNRは行番号が格納された変数ですが、NRと違って、読み込んだファイルごとの行番号が格納されます。ですので、この例のように「あるファイルの何行目」を出力するときに便利です。

deathmath2を使ってみましょう。**図7**のよう

▼リスト2　該当個所を出力するコマンド(deathmath2)

```
01 #!/bin/bash
02
03 death="です。|ます。|でした。|ました。|でしょう。|ません。"
04 da="だ。|である。|ない。|か。"
05
06 gawk '{print FILENAME ":" FNR ":" ,$0}' "$@"    |
07 gawk -v death=$death -v da=$da ¥
08      '$0~death{print "+",$0}$0~da{print "-",$0}'
```

▼図6　"$*"でうまくいかない例

```
$ cat ./bin/hoge
#!/bin/bash
cat "$*"
$ chmod +x ./bin/hoge
$ ./bin/hoge 201311.rst 201211.rst
cat: 201311.rst 201211.rst: No such file or directory
```

な出力が得られました。行頭が+のものがでます調、-のものがだである調です。実際に使う場合は、deathmath2の出力から**grep "^-"**で、だである調の行を抜き出し、目で検査することになるでしょう(**図7-①**)。もしこれでわからなければ、ファイル名と行番号が書いてあるので、当該のファイルを開いて前後の文脈を見れば良いことになります[注2]。

図7-②のように、deathmath2の出力をawk、sort、uniqで加工すると、deathmath1のような答えが得られます。deathmath1のほうがコマンド一発で数を数えてくれるので一見よさそうですが、

- コマンドが2つに分かれると使う側として覚えるのが面倒
- コードが汚くなるのは作る側が面倒
- そもそも数は最初に述べたように不正確

ということで、筆者はdeathmath2のほうが良いかなと考えます。UNIXのコマンドを作ったときの善し悪しは、ほかの主要なコマンドとの連携のうえで決定されます。

英単語をチェックする

次に、英単語のスペルチェックを行うスクリプトを作ってみましょう。スペルチェッカーは通常、エディタから読み出して使いますが、ここではコマンド仕立てにします。作ると言っても、単にラッパーを作るだけですからご安心を。

まず、スペルチェッカーをインストールします。一昔前、筆者の周辺の人はIspellというスペルチェッカーを使っていましたが、今はGNU Aspellというツールを用いるようです。MacだとHomebrewで次のようにインストールできました。

```
$ brew install aspell
```

シェルスクリプトからAspellを使いたいので、対話形式ではなく、フィルタとして使えるか(標準入出力だけで使えるか)どうか調べます。manで調べると、**図8**のような記述と、ほかにパイ

注2) 細かいですが、「でしょうか。」が「だである調」に分類されています。ただ、疑わしいものを抽出するという意味では、これでもいいでしょう。納得いかない場合は、「第一種過誤」と「第二種過誤」で検索を。

▼図7　deathmath2の実行例

```
$ cat *.rst | ./bin/deathmath2 | tail -n 3
+ -:13184:         //--dont-suggestを指定すると、候補が出てきません。
+ -:13195: エディタを開かなくてもどこに疑わしい単語があるかチェックできます。
+ -:13197: エディタから独立させておくと、思わぬところで助けられることがあります。
 ↑標準入力から読み込むとファイル名は「-」となる
$ ./bin/deathmath2 *.rst | tail -n 3
+ 201311.rst:338:         //--dont-suggestを指定すると、候補が出てきません。
+ 201311.rst:349: エディタを開かなくてもどこに疑わしい単語があるかチェックできます。
+ 201311.rst:351: エディタから独立させておくと、思わぬところで助けられることがあります。
$ cat *.rst | ./bin/deathmath2 | awk '{print $1}' | sort | uniq -c  ←②
2468 +
  38 -
$ ./bin/deathmath2 *.rst | grep "^-" | head -n 3  ←①
- 201202.rst:562: 寒さに負けず端末を叩いておられますでしょうか。
- 201202.rst:565: ドア用の close コマンドがないものか。
- 201202.rst:607: * プログラマの時間は貴重である。(略)
```

▼図8　manでAspellのオプションを調査

```
$ man aspell
 (略)
pipe, -a
        Run Aspell in ispell -a compatibility mode.
```

プについての記述が見つかりました。どうやら-aを指定するとフィルタとして使えるようです。試してみましょう。**図9**のように環境変数LANGをデフォルトのCにしないと動きません。

さて、これを使って、疑わしき単語のある行を行番号付きで出力するラッパーのシェルスクリプトを書いてみましょう。これができれば、とりあえずエディタを開かなくてもどこに疑わしい単語があるかチェックできます。結局、エディタで開いて修正しなければならないかもしれませんが、エディタから独立させておくと、思わぬところで助けられることがあります。

まず、補助的なコマンドとして、疑わしいスペルのリストを表示するコマンドを**リスト3**のように作ります。これはこれで独立で使えます。Aspellはバッククォートなどの記号類にも反応することがあり、また、日本語が入ると何が起こるかわかったもんじゃないので、3行目のsedで単語に使う文字だけ残してあとは空白に変換しています。

henspell-listを使ってみましょう。**図10**のようにまともな単語も引っかかりますが、これはAspellの辞書にこれらの単語を登録することで、出なくなります。

辞書ファイルはいろいろ種類があるようですが、とりあえず**図11**のように、1行目におまじないを書いて、あとは引っかかった正しい単語をひたすら書いていくと作れます。これをhenspell-listに読み込ませます。パスの指定が面倒ですが、コマンドのパスと一緒のところに置くなら、**図12**のようにdirnameというコマンドを使って指定できます。

次に、このコードを利用して、もとの原稿のどこに変なスペルがありそうなのかを表示します。**リスト4**に作成したコマンドを示します。このコードでは、標準入力から文字列を入力する場合と、ファイル名をオプションで指定する場合について、場合分けをせざるを得ませんでした。9行目と12行目のgrepのオプションですが、-wは単語の検索を行う[注3]、-nは行番号を入れる、

注3) つまり検索語がmarchでも、部分一致のdeathmarchは引っかからないということです。

▼リスト3　疑わしいスペル抽出コマンド（henspell-list）

```
01 #!/bin/bash
02
03 sed "s/[^a-zA-Z0-9']/ /g" "$@"  |
04 LANG=C aspell -a --dont-suggest |
05 awk '/^#/{print $2}'            |
06 sort -u
```

▼図10　henspell-listの実行例

```
$ ./bin/henspell-list 201311.rst
FILENAME
FNR
 （略）
berong
 （略）
```

▼図11　Aspellの辞書ファイル

```
$ head -n 5 ./bin/dict
personal_ws-1.1 en 0
FILENAME
FNR
GENKOU
Gancarz
 （略）
```

▼図9　Aspellをフィルタモードで使う

```
$ echo "All your base are berong to us." | LANG=C aspell -a
@(#) International Ispell Version 3.1.20 (but really Aspell 0.60.6.1)
 （略）
*
& berong 25 18: Bering, bronc, belong, Behring, bearing,  （略）
 ↓--dont-suggestを指定すると、候補が出てきません
$ echo "All your base are berong to us." | LANG=C aspell -a --dont-suggest
@(#) International Ispell Version 3.1.20 (but really Aspell 0.60.6.1)
 （略）
*
# berong 18
 （略）
```

-f <FILE>が検索対象の文字列を<FILE>から読み込む、です。

利用するときは図13のようにlessで受けて、本当にスペルミスがないか探すことになるでしょう。図中のミスは「仕込み」です。

終わりに

今回は、シェルスクリプトで文章チェックのためのコマンドを作ってみました。文章の仕事はその時々で特殊な作業が必要になることが多いので、今回のようにシェルスクリプトでコマンドを作ることを覚えると、1日かけていた作業が数秒で終わるという幸運なことに何回か巡り会うことができます。シェルスクリプトでコマンドを作るとほかのコマンドも呼び出せますから、この方法はお勧めです。

一方、今回のようにコマンドを自作しても、後日使い回すことはあまりないかもしれません。ニッチな自作コマンドなど、すぐに使い方を忘れてしまうものですが、それはそれでいいと思います。もし100個自作して、そのうちの1個がお気に入りのコマンドになれば、そのコマンドは何年にも渡って永続的に力になるわけですから、たとえ生存率が1/100であっても、御利益はあるのです。

次回は、実は最終回になりますが、crontabの使い方あれこれをやって、感動のフィナーレを飾りたいと思います。いや、話題が地味ですので何気なく終わることが予想されます。SD

▼図12　辞書ファイルを読み込むようにhenspell-listを改良

```
$ cat ./bin/henspell-list
#!/bin/bash

dict=$(dirname $0)/dict

sed "s/[^a-zA-Z0-9']/ /g" "$@"             |
LANG=C aspell -p "$dict" -a --dont-suggest |
awk '/^#/{print $2}'                       |
sort -u
  ↓使う
$ ./bin/henspell-list 201311.rst
berong
da
deathmeth
dirname
zA
```

▼図13　henspellの実行例

```
$ ./bin/henspell 201311.rst | less
14:Macには、GNU sed(``gdes``)がインストールされているものとします。
87:     da="(だ。|である。|ない。|か。)"
  (略)
217:``deathmarch2`` がもらったオプションをそのまま ``awk``
  (略)
```

▼リスト4　疑わしいスペルがある行を出力するコマンド（henspell）

```
01 #!/bin/bash
02
03 tmp=/tmp/$$
04
05 if [ "$#" -eq 0 ] ; then
06         cat < /dev/stdin                |
07         tee $tmp-stdin                  |
08         $(dirname $0)/henspell-list > $tmp-list
09         grep -w -n -f $tmp-list < $tmp-stdin
10 else
11         $(dirname $0)/henspell-list "$@" > $tmp-list
12         grep -w -n -f $tmp-list "$@"
13 fi
14
15 rm $tmp-*
16 exit 0
```

新連載

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第1回 ps(1) ～TT、STATってなんだか知ってますか?～

◉著者プロフィール
後藤 大地（ごとう だいち）
BSDコンサルティング（株） 取締役／（有）オングス 代表取締役／FreeBSD committer
エンタープライズシステムの設計、開発、運用、保守からIT系ニュースの執筆、IT系雑誌や書籍における執筆まで幅広く手がける。カーネルのカスタマイズから業務に特化したディストリビューションの構築、自動デプロイ、ネットワークの構築など、FreeBSD/Linuxを含め対応。BSDコンサルティングでは企業レベルで求められるFreeBSDの要求に柔軟に対応。

FreeBSDの魅力をもっと知ってもらいたい!

今月号からFreeBSDの連載を始めることになりました。しばらくはFreeBSDのコマンドをメインに「あ、こんな使い方できるんだ」「この使い方は便利だなぁ」「ここの値ってそういう意味だったんだ」といったポイントに絞り、Unixに対する土台作りをしながらFreeBSDの魅力を紹介していきたいと思います。FreeBSDを使われている方にもそうでない方にも、新しい知識を獲る場所として使ってもらえるような内容になるように頑張ります。

意外と知らないps(1)コマンド

動きがおかしくなったアプリケーションやサービスを停止するのには**kill(1)**コマンドが使われます。携帯ごしに「暴走したかぁ……壊れてるんだコイツ……殺しといてもらえる？ 1時間で戻るから」といった不穏当な会話が飛び交うのがサーバ運用や管理、保守の現場です。

kill(1)コマンドには終了させたいプロセスのユニーク番号、いわゆるプロセスIDを指定する必要があります。このプロセスIDを調べるのに使われることが多いのが**ps(1)**コマンドです。

たとえば**図1**の出力ですと、ユーザが実行しているプロセスがzsh(1)とps(1)の2つであることがわかります。zsh(1)のプロセスIDは81790、ps(1)のプロセスIDは81796です。ここまでは皆さんよくご存じではないかと思います。**kill(1)**に指定するのはこの値ですね。

▼**図1 ps(1)の実行例**

```
% ps
  PID TT  STAT    TIME COMMAND
81790  1  Ss   0:00.06 -zsh (zsh)
81796  1  R+   0:00.00 ps
%
```

では「TT」の項目の1は何を意味しているかご存じでしょうか。「STAT」のSsやR+もまるで呪文がごとく。こうしたちょっとしたことを知るだけで、FreeBSDはもっと使えるツールになります :)

TTは制御端末、いまではあまり意味がないけれども

TTはプロセスの制御端末（control terminal）を意味しています。現在は「制御端末」と呼ばれる機械を使ってFreeBSDを使うことはありませんので、若い技術者には想像が難しいところがあります。シリアルポート経由でログインするといった操作は制御端末に近いところがありますが、これも最近では特定の管理業務や組込み開発といったシーンを除いて使われることは少なくなりました。

ですので、若干語弊がありますが、ターミナルアプリケーションがこの「制御端末」に相当するものだと思ってください。

先程**ps(1)**を実行したサーバに、別のターミナルアプリケーションを起動してログインしてみま

す。**ps(1)**コマンドを実行すると**図2**の結果が得られます。

TTが2のプロセスが2つあることがわかります。1つ目のターミナルアプリケーションからログインしたときは、制御端末として1が割り当てられました。2つ目のターミナルアプリケーションからログインしたときは、制御端末として2が割り当てられています。

実際に制御端末のデバイスは存在しませんので、そこは「擬似端末」という擬似的なデバイスがあるものとして扱われます。FreeBSDでは/dev/pts/0、/dev/pts/1、/dev/pts/2のように必要に応じて/dev/pts/の下に擬似端末が追加されるしくみになっています。1つ目のzsh(1)は/dev/pts/0を端末デバイスとして、2つ目のzsh(1)は/dev/pts/1を端末デバイスとして扱っている、ということになります。

制御端末がある意味ってなに?

デバイスとしての制御端末を使うことがない今、制御端末という概念を持ち続けることに意味があるのか疑問に感じる方も少なくないと思います。後方互換性とか歴史的背景といった言葉で流すこともできますが、制御端末について知っておくといくつか役に立つことがあります。

▼図2　別のターミナルからps(1)を実行

```
% ps
  PID TT  STAT    TIME COMMAND
81790  1  Is+  0:00.06 -zsh (zsh)
82093  2  Ss   0:00.05 -zsh (zsh)
82099  2  R+   0:00.00 ps
%
```

▼図3　TTとSTATを組み合わせて状況を読み取る

```
% (sleep 100 | sleep 200 | sleep 300) &
[1] 82496
% ps
  PID TT  STAT    TIME COMMAND
81790  1  Is+  0:00.11 -zsh (zsh)
82093  2  Ss   0:00.06 -zsh (zsh)
82496  2  SN   0:00.00 sleep 300
82497  2  SN   0:00.00 sleep 100
82498  2  SN   0:00.00 sleep 200
82499  2  R+   0:00.00 ps
%
```

制御端末という概念はもともと必要があって――必要があってというか、この概念を持っておいたほうが何かと便利なことが多かったので追加された概念なのですが、「プロセスをグループ化できる」点にあります。**図3**のコマンドを見てみましょう。

sleep(1)コマンドが3つ実行されているなぁ、ということがわかります。制御端末を知らないとそこ止まりですが、制御端末を知っているとさらに深いところまで読み取ることができます。

2つ目のzsh(1)、sleep 300、sleep 200、sleep 100、ps(1)には制御端末2が割り当てられています。つまり、同じターミナルアプリケーションで実行されているコマンドだ、ということがわかります。詳しくは後ほど説明しますが、STATの値を組み合わせると、ps(1)が直前に実行されていること、zshがインタラクティブに操作できる状態にあること、sleepはバックグラウンドプロセスとして実行されているであろうこと、などまで読み取れます。

STATを読んでみよう!

STATの項目は呪文のようになっていますが、ちゃんと意味があります。最初の1文字がプロセスの実行状態を表しています。**表1**のような感じです。

2文字目以降は付加情報です。**表2**のような文字を指定できます。

つまり、Ssのzsh(1)はスリープ状態(何も実行していない)でセッションリーダー、sleep(1)もスリープ状態でさらにCPUスケジューリングプライオリティが引き下げられたプロセスであること、ps(1)は

▼表1　STATの1文字目の意味

プロセスの実行状態	意味
R	プロセスは実行状態(RUNNABLE)
T	プロセスは停止状態(STOP)
I	プロセスは20秒以上のスリープ状態(IDLE)
S	プロセスは20秒未満のスリープ状態(SLEEP)
D	プロセスはディスク待ち状態(DISK)
L	プロセスはロックの取得待ち状態(LOCK)
W	プロセスはアイドル割り込みスレッド
Z	プロセスはゾンビ

▼表2　STATの2文字目の意味

追加情報	意味
L	ロックされたページを保持しているプロセス
W	スワップアウトしているプロセス
+	制御端末のフォアグラウンドプロセスグループに所属している。インタラクティブ状態にあるシェルや実行中のコマンドなどが該当する。パイプラインでフォアグラウンドプロセスとして実行されるコマンド達もこの状態として認識される
s	セッションリーダーのプロセス
<	CPUスケジューリングプライオリティが引き上げられたプロセス
N	CPUスケジューリングプライオリティが引き下げられたプロセス
J	jail(2)環境で実行されているプロセス
V	vfork(2)中にサスペンドしたプロセス
X	トレースされているプロセスまたはデバッグされているプロセス
E	終了しようとしているプロセス

今まさに実行され、制御端末のフォアグラウンドプロセスに所属していること、がわかります。zsh(1)でps(1)コマンドが実行された、ということはsleep(1)はそれ以前にバックグラウンドプロセスとして実行されたこと、などがわかるわけですね。ちょっとした知識を得るだけで、**ps(1)**コマンドの出力してくれるデータはとても情報量にあふれたものになるわけです。

制御端末を持たないプロセスもある—サービスやGUIアプリケーション

すべてのプロセスが制御端末を持っているかというと、そうでもありません。Webサーバやメールサーバは制御端末が必要ありませんし、GUIアプリケーションも制御端末を持つ必要はありません。**ps(1)**コマンドは制御端末を持たないプロセスを表示するために**-x**というオプションを提供しています(**図4**)。

ホストにssh(1)でログインしているので、この受け口になっているsshd(1)プロセスが制御端末を持たないプロセスとして表示されています。TTの項目が**??**になっていますね。これが制御端末を持たないという意味です。

▼図4　制御端末を持たないプロセスも表示

```
% ps -x
  PID TT  STAT    TIME COMMAND
81789 ??  S    0:00.01 sshd: daichi@
pts/1 (sshd)
81790  1  Ss   0:00.11 -zsh (zsh)
82824  1  R+   0:00.00 ps -x
%
```

プロセスの親子関係も表示する-dオプション

FreeBSDに限らず、Unix系の設計思想を引き継いだOSにはプロセスの間に親子関係があります。想像できないかもしれませんが、シェルでコマンドを実行するというのは、シェルプロセスが自分自身をコピーしたのち(これが子供のプロセス)、コピーされたプロセスを指定されたコマンドに置き換えて(別の人に転生して人生をやり直すようなもの)、そのあと処理を実行する(その人として仕事する)、という処理を毎回毎回律儀に実行しているのです。

ps(1)コマンドにはこの親子関係をわかりやすく表示する**-d**というオプションがあります(**図5**)。どこで実行されたコマンドなのか調べたいときもありますので、知っておくと便利です。

ユーザのプロセスじゃないプロセスを表示してみよう

ps(1)コマンドは何もオプションを指定しないと、コマンドを実行したユーザのプロセスのうち、制御端末を持つプロセスのみを表示します。ほかの

▼図5　プロセス間の親子関係を表示

```
% ps -d
  PID TT  STAT    TIME COMMAND
81790  1  Ss   0:00.12 -zsh (zsh)
82911  1  S    0:00.05 - zsh
82917  1  S    0:00.05 `-- zsh
82923  1  R+   0:00.00   `-- ps -d
%
```

ユーザのプロセスを表示するには-aオプションを指定します。

図6を見れば、自分のほかにもrootユーザがシェルで作業していることがわかります。

カーネルプロセスを表示させてみよう

FreeBSDにはカーネルの中でだけで動作する特殊なプロセスが存在します。これをカーネルプロセスと呼びます。スワップの管理や割り込みの処理など、OSとして動作するのに欠かせない機能はカーネルプロセスとして実行されています。

カーネルプロセスは制御端末を持たないプロセスで、ユーザ以外のプロセスですから、-xと-aを同時に指定すると表示させることができます(図7)。

いきなりたくさんのプロセスが表示されたと思います。プロセス名が[]で囲まれたプロセスがカーネルプロセスです。大半のカーネルプロセスにはとても高いスケジューリングプライオリティが与えられ、OSとしての機能をまっとうしています。

実はもうプロセスは存在しない、その実体はスレッド

いままで何度も何度もプロセスという言葉を使ってきましたが、実はFreeBSD 5以降、FreeBSDには処理の実体としてのプロセスは存在していません。しくみ上これまでと同じように扱えるようにしてありますが、処理の実体はマルチプロセッサ、マルチコア、メニーコアに対応するために「スレッド」に変わりました。ちょっとだけ覗いてみましょう(図8)。

プロセスIDが同じで、さらにプロセス名も同じプロセスがいくつも見つかったと思います。これがカーネルスレッドです。実際にはこの単位で処理が行われています。

これだけは覚えておこう「ps auxww」

ちょっとそんないきなりはオプションは覚えられないよ、という方は、今回はこれだけ覚えていってください。「ps auxww」です。ps(1)にはたくさんのオプションがありますので、ここで紹介しきれなかったさまざまな角度からデータを表示させることができます。

なかでもよく使われるオプションの組み合わせがauxwwです。システムで動作しているプロセスをすべて詳細に表示せよ、といったオプションだと思ってください。ps(1)コマンドはオプション引数の-を省略できますので、「ps auxww」と入力することが多いです。

おわりに

今回の記事から何か新しい知識は得られましたでしょうか。毎回「お、なるほど」と思わせる小ネタを紹介していきたいと思います。今後ともよろしくお願いいたします。SD

▼図6　ほかのユーザのプロセスも表示

```
% ps -a
  PID TT  STAT    TIME COMMAND
81790  1  Is+  0:00.13 -zsh (zsh)
83264  2  Is   0:00.06 -zsh (zsh)
83280  2  I    0:00.01 - su
83281  2  I    0:00.00 `-- su (sh)
83282  2  S    0:00.00   `-- ps -ad
%
```

▼図7　カーネルプロセスも表示

```
% ps -ax
  PID TT  STAT          TIME COMMAND
    0 ??  DLs        6:38.91 [kernel]
    1 ??  SLs        0:06.29 /sbin/init --
    2 ??  DL         0:00.00 [ctl_thrd]
 …略…
81790  1  Ss         0:00.13 -zsh (zsh)
83425  1  R+         0:00.00 ps -ax
%
```

▼図8　スレッドとして表示

```
% ps -axH
  PID TT  STAT          TIME COMMAND
    0 ??  DLs      87:26.65 [kernel]
    0 ??  DLs       0:00.00 [kernel]
    0 ??  DLs       0:00.00 [kernel]
    0 ??  DLs       0:00.00 [kernel]
    0 ??  DLs       0:00.00 [kernel]
    0 ??  DLs       0:00.00 [kernel]
 …略…
81790  1  Ss         0:00.13 -zsh (zsh)
83504  1  R+         0:00.00 ps -axH
%
```

9

Debian Developer　やまねひでき　henrich@debian.org

DebConf13開催! そして7.2リリースがやってくる

Debian Hot Topics

「DebConf13」開催

さて、前回、前々回と2回続いたDebianのRubyの話題はいかがでしたでしょうか？今回はまたDebian界隈のさまざまなトピックについてお送りしていきます。

◆ ◆ ◆

8月11日～17日の1週間、スイスのヌーシャルテル湖そばのヴォマルキュのキャンプ場にて、300名近くのDebian関係者を集めたDebianカンファレンス「DebConf13」[注1]が開催されました。例年どおりの2トラックのトークセッション用と、期間中に突発的に登録／開催されるBoFセッション用の2部屋の計4部屋が用意され、さまざまな話題が飛び交いました（**写真1**）。

今年の特色としては、昨今のクラウド機運の高まりに合わせてAWSとGoogle Compute Engine関連のセッションが多数開かれ、とくにAWSのセッションはAWSソリューションアーキテクトでDebian開発者のJames Bromberger さんが精力的にさまざまなセッションを開くなどして目を引きました。そのほかクラウド関連では、OpenStackを含めたパッケージのメンテナがこのDebConf13で議論を行って新規メーリングリストを立ち上げたことをアナウンスしています。現状のdebian-cloudメーリングリストはクラウド関連の一般的な話題を、新規に設立されたcloud-packagesメーリングリスト[注2]ではパッケージに特化した話を行っていくようです。

ほかのディストリビューション関係者との交流も「Why Debian needs Upstart」「Why Debian should (or should not) make systemd the default」などのセッションという形で行われました。「initシステムとしてUbuntuで採用されているupstartを採用しよう」、「いやいやFedora

注1）URL http://debconf13.debconf.org/　スイスという涼やかな避暑地での開催ということもあり、多くの開発者はバカンスモードに入っていました（ヨーロッパ在住の人の中には家族連れで、という人もちらほらといました）。

注2）Mail cloud-packages@lists.alioth.debian.org

▼写真1　DevConf13会場の様子

で採用されているsystemdを使うべきだろう」という話を、upstartとsystemdのそれぞれの開発者が語るというなかなか贅沢なセッションです。同様なものとしては、MySQLのフォークである「MariaDB」の作者自身がMariaDBの説明を行うセッションも開催されました。まだ結論めいたものは出ていませんが、関心を持っている開発者らへの良い情報インプットとなったのではないでしょうか。

そのほか、DebianのLinuxカーネルパッケージメンテナでもあり、upstreamのkernel.orgでの3.2カーネルのメンテナでもあるBen Hutchingさんが、「What's new in the Linux kernel ? and what's missing in Debian」というセッションを行い、Debianの安定版で採用している3.2カーネルではサポートしていない新機能の説明を行いました。いくつかはカーネル側の対応だけではなくユーザランド側にも新規で追加パッケージや更新が必要になるので、安定版へ適用とはいきづらいようです。そのような機能を利用したい場合はtestingやunstableを使うか、stable版でバックポートされたカーネルや関連パッケージの利用をすると良いでしょう。

また、別のセッション「Hardware support in Debian stable」では、カーネルパッケージのアップデートに伴うリグレッションを極力起こさないよう、今後、更新ドライバモジュール用のパッケージを収録する予定であることが明らかにされました。これは安定版をサーバなどで利用するユーザにとっては変更を極小にできるので喜ばしいのではないでしょうか。

上記を含めた基本のトークセッションはビデオ録画されてフランスのIRILLのサイト[注3]で閲覧できるようになっていますので、ご興味のある方はぜひどうぞ。なお、興味深かったのは、例年とは違ってハックラボ内で作業にいそしむだけではなく、室外で数人のグループが集まって芝生の上に座り込んで議論をしている姿があちらこちらで見られたことです(快適な避暑地で開催した効果でしょうか?)。後述の20周年記念パーティの件もあり、いつもより参加者同士の交流が活発(=ソーシャル)なイベントの側面が大きかったように思われます。

Debianが20周年を迎えたということでパーティを開いて巨大ケーキでお祝いをしました。当日はあいにくの雨模様になったのですが、雨が去ったあとには見事に虹が輝き、まるで天がDebianを祝福してくれたようでした。また今回、DebConf13のスポンサーであるぷらっとホームさんからOpenBlocksを提供いただいたので、raffles(富くじ)の景品とさせていただきました(ありがとうございます)。

来年以降のDebianカンファレンス

そして来年のDebConf14ですが、アメリカのポートランドで開催されることが正式に発表されました。ポートランドは日本からの直通便があり非常に楽に行けるのに加え、知人曰く「アメリカでイベントを開催するなら最高の場所」とのことですので、今から来年が楽しみです。

そして、再来年(!)のDebConf15ですが、ベルギーとドイツが候補地として立候補しました。ベルギーは世界最大のFLOSSイベントである「FOSDEM」[注4]の開催地であることからイベント開催のノウハウがあることを、ドイツは世界でもっともDebian開発者が在住している国であり豊富な人手と層の厚さを、それぞれアピールしています。

なお、「日本でもDebConfを開催しないのか」という話はたびたび挙がるのですが、開催に向けては「local government」「local team」「money」の3本柱をそろえる必要があり、なかなか容易

注3) URL http://www.irill.org/videos/debconf13 YouTubeに転載もされているようですので、そちらで「debconf13」で検索いただいても結構です。また、ビデオ録画以外のメモなどはgobbyというツールでgobby.debian.orgに接続いただければ閲覧が可能になっています。

注4) Free and Open Source Development European Meeting URL https://fosdem.org/

ではない……というのが正直なところです[注5]。とはいえ、開催に向けては前向きな姿勢でいますので、まずはもう少し小さめのイベント「MiniDebConf」を開いていこうかと考えています。

MiniDebConf

MiniDebConfはその名のとおり「ミニ」サイズのDebianイベントです。世界各地域のDebianコミュニティが自主的に開催しています。11月には次の2つのMiniDebConfが開催される予定です。

- **MiniDebConf in Taiwan（11月9日〜11日）**[注6]
  LXDE開発者のAndrew Leeさんらが主催するアジアで開かれる数少ないDebianイベントです。下記のUKイベントに比べると少人数になるかと思いますが、台湾は日本からも近いため、できれば筆者も参加してみたいと考えています

- **MiniDebConf in UK（11月14日〜17日）**[注7]
  現状4、50人規模の参加者が予想されています。こちらは主催者がARM社に勤めている元Debian Projectリーダーの Steve McIntyreさんで、開催地がARM社オフィスということもあり、おもにARM周り（とくに64bitARM、AARCH64アーキテクチャ）のハックが進められるのではないかと推測しています

bit from release

次に、イベント以外の話題もいくつか取り上げてみましょう。

残念ながら筆者は参加できなかったのですが、DebConf13中にリリースチームのBoF[注8]が行われ、その話し合いの成果が「bit from release」という形[注9]で出てきました。4点ほどの提案がありましたが、その中でも重要な2点について取り上げたいと思います。

パッケージ更新スピードの加速

まず、現状のパッケージのアップデート状況を説明すると、パッケージが更新されるとunstableにアップロードされます。そこで10日間[注10]の間に重大なバグ（RCバグ）が登録されなければ更新されたバージョンがtestingへコピーされます。この10日という時間は、多くの問題を洗い出して、testingという次のリリースへ向けたマネジメントを行う（大きな問題がないパッケージのみを選別して更新する）という意味があります。しかし、この手法は人手に頼っており、またどうしても10日間のタイムラグが発生することから開発のスピードにブレーキをかけている点も否めません。

今回提案されたのは、パッケージにautopkgtestを利用したパッケージ単体テスト[注11]を導入して既知の問題についてのリグレッションを防ぐことで品質をある程度担保し、そのようなパッケージは2日間でtestingへコピーするようにして開発スピードを上げよう、というものです。現状、Ubuntuでは一部でautopkgtestを活用しているようですが、Debianにはautopkgtestのノウハウがほとんど溜まっておらず、次のリリースまでにどの程度この動きが進められるかに注目したいところです。

注5）ご協力いただける組織、企業などに心当たりがございましたら、ぜひご連絡ください（Twitter：@debianjp、またはメール：board@debian.or.jp）。

注6）URL http://wiki.debian.org/DebianTaiwan/MiniDebConf2013

注7）URL http://wiki.debconf.org/wiki/Miniconf-UK/2013

注8）「birds of a feather」の略で特定の話題についての気軽な集まり。とくに進行も定めないでワイワイとやる集まりのことを指します。開催がビデオ録画がない部屋だったのが悔やまれます。

注9）URL http://lists.debian.org/debian-devel-announce/2013/08/msg00006.html

注10）これはchangelogエントリ中のurgencyの値に依存します。通常はurgency=lowで10日間、mediumだと5日、highだと2日となります。

注11）URL http://dep.debian.net/deps/dep8/で「autopkgtest - automatic as-installed package testing」として提案されたものです。

不具合パッケージの自動削除

ご存じの方も多いかと思いますが、Debianの安定版は必ず「なんとかして」RCバグ(リリースクリティカルバグ、そのままリリースするには問題となる重大なバグのこと)を0にしてリリースに漕ぎ着けます[注12]。しかし、昨今のリリースはパッケージの総数が非常に多くなってきたこともあり、RCバグの数も増大し、リリースまでの時間が若干遅延気味になってきました。リリースチームはこれに対して現状半自動で行っているRCバグ持ちのパッケージ削除を「全自動」にして改善しよう、という提案を出してきたのです。

ただ、単にRCバグを持っているパッケージをtestingから削除すると依存関係の問題が出てきたり、重要なパッケージがいきなり使えなくなったりと問題が出てきてしまいます。その対応としては、Lucas Nussbaumさんによる「キーパッケージ」という考え方[注13]をベースにして、「キーパッケージに影響するもの以外」のRCバグについてパッケージを自動的に削除するようにしよう、ということになっています。現在登録されているRCバグの多くはキーパッケージには影響しない(実際、unstableにある1300個のRCバグのうち、キーパッケージに影響を及ぼすのは250個程度)ので[注14]、もし導入されればリリース直前のRCバグ潰しはかなり楽になってリリースの遅延も少なくなるのではないか、と思われます[注15]。

Perl 5.18 transition

Unstableに久々に大きな波がやってきました。Perlのバージョンが5.14→5.18と大きく上がり、多数の依存パッケージが芋づる式にアップデート、再ビルド、修正が必要な状態となったのです[注16]。

なお、変更の影響は大きかったものの幸いなことに破壊的な状態にはなっていません(有名どころで言うと一時的にgit-svnあたりが壊れたぐらいでしょう)。もしかしたら、まだ報告されていない問題が隠れているかもしれませんので、何かおかしなことがあったら本件を念頭に調査をしてみてください。

7.2アップデートリリース

この号の発売と前後しますが、Debian 7 "wheezy"の2回目のアップデートが10月12日～13日の間で実施予定であることがdebian-releaseメーリングリストで明らかになっています。このアップデートでは例のごとく

- これまでにリリースされたセキュリティ修正
- 7.1リリース後に発見／修正された大きな問題への修正
- ドライバのアップデートや新規追加を含むカーネルの更新

が含まれています。多くの場合はとくに気にすることなく更新を実施可能ですが、運用中のサーバなどで慎重を期する必要がある場合はご注意ください(以前も本連載で言いましたが、できればリリース前のproposed-updatesの時期にテストを重ねておくのがお勧めです)。

また、同様に6.0 "Squeeze"についても更新(6.0.8)がきます。こちらは予定では10月19日～20日となっています。忘れずに適用してください。SD

注12) 基本はバグ修正ですが、最後の最後には重要度を下げて「先送り」にする場合もあります。

注13) URL http://lists.debian.org/debian-devel/2013/05/msg00496.html

注14) Ultimate Debian Databaseで閲覧できるようになっています。URL http://udd.debian.org/bugs.cgi

注15) その代わり、バージョンがやけに古いものがリリースされたりするかもしれませんが……。

注16) このようにバージョンアップによってライブラリのバージョン(soname)が更新され、依存パッケージの再ビルドが必要になる状況をDebianでは「library transition」と呼んでいます。現状進められているtransitionについてはhttp://release.debian.org/transitions/で状況の確認ができます。なお、筆者はこれをよく把握せずにパッケージの更新を行ってしまい、派手に依存関係をぶっ壊して怒られた経験があります。

第14回

気持ちのrootにあるもの

梅谷 琢磨
Umeya Takuma
レッドハット㈱
グローバルサポートサービス
スーパーバイザー

注）本稿は執筆者の意向により、常体にしている。

サポートエンジニアであることとは

「明けない夜はないという気持ちを持て」

しばらく前に、もうだめだと思ったときに父にかけられた言葉だ。相談事を人に打ち明けることは今も昔も多くはないので、あのときは大分困っていたのだと思う。今となってはそのときの気持ちをつぶさに表現するには記憶があやふやだけれども、喉がカラカラに渇いてうまく言葉が出てこない、そんな心持ちであったことを覚えている。

筆者はRed Hatのテクニカルサポートエンジニアを経験したあと、サポート部門のマネージメント職に就いた。グローバルサポートサービス・プロダクションサポートチームのスーパーバイザーというのが与えられた役割になる。さて、そのサポート部門は電話とWebツールを利用し、顧客に製品についての技術サポートを提供することを業務としている。業務内容の多くは自ずとトラブルなどの厄介事についての相談だ。サポートメンバの業務がスタートする時点で事態はすでにヨロシクナイ状況にある、というとこの部門の置かれている状況が理解しやすいだろう。職責というのは承知か否かにかかわらず執り行われるべきとはいえ、メンバが難しい状況に果敢に挑むのを目の当たりにして気持ち抜きにはいられない。

オープンソース製品のサポートを生業とするRed Hatのサポート部門では、フロントエンドからバックエンドまでどのポジションで作業しているかによらず、ソースコードレベルの解析が許されている。コード解析に関係する詳細なデバッギングや解析などは多くのサポートエンジニアが日常的に行う作業だ。また、顧客が自前で調べているケースもあるため、豊富な知識量がなくてはサポート業務は務まらない。こうした状況だから、サポートエンジニアは日々勉強に勤しまなければ今日明日の職責を全うすることはできない。なお、海外のエンジニアとのコミュニケーションはすべて英語。Red Hatのサポートではエスカレーションモデルとは異なるコラボレーションモデルを採用しており、積極的にエンジニア同士の会話や議論に加わることが望まれている。エンジニアとしてのキャリアを積むということに考えをめぐらせたとき、Red Hatでサポートエンジニアを経験することはその人のキャリアにとって大きなチャンスになることをご理解いただけたかと思う。

そうはいっても技術的な難易度やコミュニケーションの観点からケースの進捗が思わしくなく、頭を悩ませながら作業に従事するメンバにとって、これがチャンスなのか苦行なのかは紙一重といったところ。キャリアとして魅力的だから、という気持ちだけでRed Hatでサポートエンジニアの仕事を長く続けることは難しいように感じる。

ずっと整理したかったこと

話が逸れる。——かなり以前の話になるのだが、筆者は自立したエンジニアになりたいと思っていた。内容を噛みくだくと、他人の力を借りないで自分だけで完結できるエンジニア、とい

う何ともカンチガイ甚だしい向きだった。そして自分のようなエンジニアが居場所や食い扶持を得るためには、自分が競争に勝ち続けなければならないとも思っていた。1日がそうした勝ち負けに終始することにモヤモヤを感じながらもさしたる疑問も持たずに胸を張って歩いていたのは、偏(ひとえ)に筆者の了見の狭さの表れであったのだと思う。1人よがりであるがために自分の物差しだけが基準になり、思うように運ばない物事を目の当たりにしては心の中で憤慨し、人とのちょっとの衝突で妙に傷ついていたように思う。自立するということが目標なのに、生活にはあべこべに自分の軟弱さが透けて見えていた。本当は自立したかったのではなく、自分の辛抱のなさや心の弱さと向き合うことを避けて、他人との交渉事を回避しながら手前勝手に生活することを追認したかったのかもしれない。こうした精神的傾向はエンジニアの業務以外の自分にも過分に言えたことなのではないかと思う。

なお、父と言葉を交わしたのはこの頃だった。仕事やそれ以外、生活のすべてが行き止まりまで来てしまったように感じたときがあった。でも、どう後戻りするのか、どうやって取り返すのかが皆目見当がつかなかった。何より不安を口にする相手もなかったのかもしれない。

以上は個人的な事情が影響している話なのだが、それはともかくとして。自分とディスプレイとだけがそこにいて1日中対話しているような、エンジニアの作業は孤独な環境に意図せずとも陥りがちであり、そうした暮らしが永続的か、人間的か、魅力的か、前進的かについて思うところありとする私の今の気持ちに賛同いただけるエンジニア諸兄は決して少なくないのではないだろうか。

今になって思うのは、人はたった今の時間の充実があって、やっと未来に目を向けるんじゃないだろうかということだ。たった今の充実は、話し合って、助けて、助けられて、感謝して、安心して、あるいは意見して、失敗して、憤慨されて、謝って、反省して、などなど、心の行き交い、つまりはその人が社会的であろうとする意思が大きく作用するのではないか。だから、窓は外に向かって広く開け放たれていてほしい。

我々が身を置く社会は弱肉強食なコンバットゾーンではない。互いに失敗から学んで、次の成功のために助け合ってやり直すことが許されている。そうした経験はエンジニアがまた明日も頑張ろうとする姿勢に貢献するのだと思う。苦楽はあれども有意義だと思えるたった今の暮らしがあってこそ、キャリアなど先々の自分について前向きな考えを持つことができるんじゃないかと思う。

孤独は何も生まない

現在部門に従事しているメンバにはいくつかの点を心がけてもらえるようそれとなく気をつけている。自分の意見を大切にして行動すること、正直かつ誠実であること、謙虚な心を持つこと、互いの理解に努めること、辛抱強くあること、確信はなくとも自分を信じて行動すること、などなど。どこかで聞いたような言葉ばかりだが、エンジニアが社会的であることに関係する大切な点だと思う。サポートのメンバが常に社会的であることを心がければ、作り出される風紀はやがてお客様の立場を理解して可能な限りの努力を提供しようとするメンバの行為の後ろ盾となってくれると思う。社のキャッチフレーズに“We are Red Hat”というのがある。筆者はメンバ全員に、たった今この部署に従事していることを喜びに感じて、お客さまに可能な限りのサービスを提供する、オープンソースサポートならではの素晴らしいエンジニアリングを提供してほしいと考えている。所属するエンジニアが、ここでの経験を通してラジカルな飛躍ができますように。Red Hatの製品を利用しているお客さまが我々のサービスを購入してよかったと思ってもらえますように。これからも可能な限りの努力を費やしたいと考えている。

SD

第43回

Ubuntu Monthly Report

Inside Ubuntu Touch

Ubuntu Japanese Team／(株)創夢
柴田 充也　SHIBATA Mitsuya

スマートフォン／タブレット向けに開発されている「Ubuntu Touch」。今回はその中身について紹介します。

Ubuntu Touchとは

Ubuntu Touchは、スマートフォンやタブレットに代表されるマルチタッチ対応のモバイルデバイス向けに開発されているUbuntuの一種です。2013年の初めにプロジェクトが公表されて以降、「第3のモバイルOS」の一角として注目されています。

一言で「Ubuntu Touch」と言っても、その中身についてはいくつかのコンポーネントから成り立っています。

■カーネル・ドライバ

現在のUbuntu Touchのサポートデバイスは、GoogleのNexusシリーズです。このためカーネルやドライバはAOSPをベースにしたCyanogenMod 10.1のそれを一部改変する形で使用しています。

■ミドルウェア

グラフィック／オーディオのドライバはAndroidのそれを使う必要があります。そこで描画はX Window SystemではなくAndroidのドライバモデルを使える「Mir」を採用する予定です。ただし2013年9月初めの時点ではまだMirが完成していないため、AndroidのSurfaceFlingerで代用しています。オーディオについては当初AudioFlingerを使っていましたが、8月にALSAへの移行が完了しました。ちなみにAndroidのドライバやライブラリにアクセスするために、hybrisも使用しています。アプリケーションやGUIシェルには、Qt 5.xをツールキットとして採用しています。AndroidのDalvik VMは同梱していません。

■GUIシェル

デスクトップ版のUbuntuは、あまりマルチタッチデバイス向けのインターフェースにはなっていません。また、モバイルデバイスのような小さなサイズのインターフェースを考慮していません。そこでQtを使って既存のデスクトップUIであるUnityと同じようなインターフェースをフルスクラッチで作成しています。ちなみに、このGUIシェルはUnityの次のバージョンとして14.04以降のデスクトップ版のUbuntuにも採用される予定です。

■アプリ・パッケージ

モバイルOSにとって重要なのが「アプリストア」に代表されるエコシステムです。Ubuntuにはもともとソフトウェアセンターからインストールできる豊富なパッケージが存在しますが、今のところ大半のデスクトップ系パッケージはUbuntu Touch上では使えません。そこでQtアプリを作りやすいように「Ubuntu SDK」と呼ばれる開発環境を提供することになりました(**図1**)。またモバイルOS向けにカスタマイズしたパッケージングシステムである「Click」も開発してい

ます。ちなみにGUIを必要としないパッケージであれば通常のUbuntuと同じものがそのまま使えます。

今回はこれらの中から、各コンポーネントがつまったイメージファイルの構成と、カーネルが起動してからGUIシェルが立ち上がるまでに参照するファイルについて紹介します。

Ubuntu Touchを試してみる

Ubuntu TouchはGoogleのNexusデバイスとUbuntu PCがあれば実際にインストールできます[注1]。詳細な手順や注意点はWikiにあるインストールガイド[注2]を確認してもらうとして、ここでは簡単な手順を説明しておきましょう。

あらかじめNexusデバイスのブートローダをアンロックして、さらにUSBデバッグを有効にしておいてください。なおブートローダをアンロックするとハードウェア保証が無効になります。さらにAndroidなどのデータはクリアされますので、必要なデータは適宜バックアップをとっておいてください。ちなみにUbuntu Touchをインストールしたあとにアンドロイドをインストールしなおすことはできます。

次にUbuntu PCとNexusデバイスをUSBケーブルで接続したうえで、**図2**を実行すると必要なイメージをダウンロードしてインストールします。インターネットの接続環境にも依存しますが、30分ほど放置するとUbuntu Touchが立ち上がっているはずです(**図3**)。

最後の2つのコマンドはいずれか1つを実行しましょう。cdimage-touchは、通常のUbuntuと同じような構成でインストールできます。インストール後はapt-getコマンドなどでパッケージのアップデートやインストールができます。ubuntu-systemはルートパーティションを読み込み専用としてインストールします。このためそのままではapt-getコマンドは使えません。ただしスマートフォンとしての用途を考えると後者のほうがセキュアなのと、差分アップデートが可能であることから、現在はubuntu-systemのほう(システムイメージ版)の利用が推奨されています。

インストールイメージの構成

システムイメージ版の場合、インストールする際にいくつかのファイルをサーバ[注3]から「~/ダウンロー

注1) Nexus7(2013)には未対応です。Nexus7(2012)の3Gモデルにはwifiモデルと同じイメージを使うため3G部分は使えません。

注2) https://wiki.ubuntu.com/Touch/Install

注3) https://system-image.ubuntu.com/

図1 Ubuntuアプリを開発できるUbuntu SDK

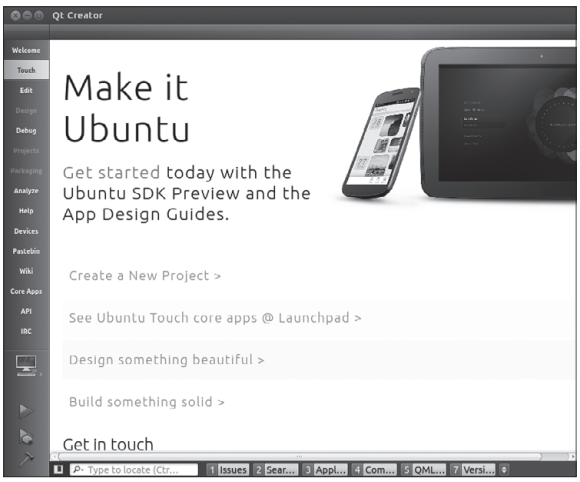


図2 Ubuntu Touchのインストール方法

```
$ sudo add-apt-repository ppa:phablet-team/tools
$ sudo apt-get update
$ sudo apt-get install phablet-tools

(以下の2つのうちいずれかを実行)
$ phablet-flash cdimage-touch -b
$ phablet-flash ubuntu-system
```

図3 インストール完了後は自動的にUbuntu Touchが立ち上がる

ド/phablet-flash/」以下にダウンロードします（**図4**）。そのうち重要なのは次の2つです。

- grouper-20130905.1.full.tar.xz
- ubuntu-20130905.1.full.tar.xz

前者にはUbuntu Touchのカーネルイメージや Ubuntu Touchをインストールするために必要なリカバリイメージなど、デバイスに依存したデータが含まれています（デバイスイメージ）。このためデバイスによって名前が変わります。たとえば上記の「grouper」はWiFi版Nexus 7（2013）のコードネームです。

後者にはUbuntu Touchのルートファイルシステムが含まれてます。フォーマットはただのアーカイブファイルでデバイスに関係なくすべて同じイメージが使われます。

図4　ダウンロードされるファイル一覧

```
imageupdates
└── daily
    ├── grouper
    │   ├── grouper-20130905.1.full.tar.xz
    │   ├── grouper-20130905.1.full.tar.xz.asc
    │   ├── grouper-20130905.1.full.tar.xz.asc.lock
    │   ├── grouper-20130905.1.full.tar.xz.lock
    │   ├── version-4.tar.xz
    │   ├── version-4.tar.xz.asc
    │   ├── version-4.tar.xz.asc.lock
    │   └── version-4.tar.xz.lock
    └── ubuntu
        ├── ubuntu-20130905.1.full.tar.xz
        ├── ubuntu-20130905.1.full.tar.xz.asc
        ├── ubuntu-20130905.1.full.tar.xz.asc.lock
        └── ubuntu-20130905.1.full.tar.xz.lock
```

図5　デバイスイメージの中身

```
partitions
├── boot.img
└── recovery.img
system
└── var
    └── lib
        └── lxc
            └── android
                └── system.img
```

図6　イメージファイルの展開方法

```
$ sudo apt-get install abootimg
$ abootimg -x boot.img
```

図7　システムイメージをマウントする

```
$ sudo mount -t ext4 system.img /mnt/
```

デバイスイメージを分解する

デバイスイメージ自体は単なるtarアーカイブなので普通に展開できます。さらにその中身は、単純なイメージファイルの集合体です（**図5**）。このためAndroidのイメージと同じく、abootimgコマンドで展開できます。中身を確認したい場合は**図6**のようにabootimgパッケージをインストールしておきましょう。

boot.imgはカーネルイメージとinitrdファイルが存在します。通常はこのイメージを使って起動します。recovery.imgはリカバリ領域に書き込むイメージで、中身はAndroidでよく使われるClockworkMod ROM Managerにいくつか Ubuntuの改造を加えたものです。Ubuntu Touchインストール時はまずこのrecovery.imgが書き込まれたあとに、必要なデータを次々書き込んでいきます。ちなみにこれらのファイルに入っているinitrd.imgは、abootimg-unpack-initrdコマンドで展開できます。

system.imgだけはAndroidのブートイメージではありません。これはUbuntu Touchのルートファイルシステムに展開される、Androidに依存したデータをまとめたものです。

中身を確認したい場合は**図7**のようにマウントしてしまうのが一番簡単でしょう。たとえばカーネルモジュールやデバイスドライバ、Androidサービスを実行するために必要なデータが含まれます。

ここで紹介したデータはすべてカーネルに依存します。カーネルが更新されたとき、つまりboot.imgが入っているデバイスイメージが更新されたときに同時にアップデートされなければならないため、ルートファイルシステムのイメージではなく、デバイスイメージに入っているのです。

Ubuntu Touchのサービス

Ubuntu Touchの起動シーケンスはPC版Ubuntuとそれほど違いがあるわけではありません。しかしながらモバイルデバイス向けに、いくつかPC版とは異なるサービスが起動しています。

まず起動の流れとして、カーネルが起動してからルートファイルシステムがマウントされ、init（Ubuntuの場合はUpstart）が呼ばれるというところは同じです。カーネルはCyanogenModのカーネルをベースにUbuntu独自の変更を加える形であるため、デバイスごとにカーネルのバージョンは異なります。

ルートファイルシステムはubuntu-systemの場合ほとんど読み込み専用で/homeにマウントされる/userdataや/tmpなどだけが書き込み可能という設定になります。また、/userdata/.writable_imageという空ファイルを作ってから再起動すると、読み書き可能な状態でルートファイルシステムをマウントできます[注4]。ファイルシステムのマウントについては、/usr/share/initramfs-tools/scripts/touchを読むと良いでしょう。

Upstartが呼ばれたら順次サービスを起動していきますが、X Window SystemやLightDMは立ち上がりません。その代わりLXCでAndroidサービスを立ち上げたり、Upstartのユーザセッションから直接Unity 8を立ち上げています。

Upstartが立ち上げるサービスのうち、Ubuntu Touch特有のサービスを見て行きましょう。

adbd

adbd（android-tools-adbd）はAndroidでおもにデバッグ用に使われるadbをUbuntuにそのまま移植したデーモンです。これによりUbuntu TouchではAndroidと同様にUSBケーブルでデバイスに接続したうえで、ホスト側からadbコマンドを使って基本的な操作を行えます。

たとえば単純にUbuntu Touchにログインしたい場合は、「adb shell」コマンドを実行してください。デバイスを再起動するなら「adb reboot」です。ファイルの転送もできます。デバイス上でスクリーンショットを撮影してホストに転送する場合は、**図8**を実行します。

注4） この場合システムイメージのアップデートはできなくなり、通常のUbuntuと同様にapt-getコマンドでアップデートを行う必要があります。

後述するように、Ubuntu Touchの中ではAndroidサービスも動いています。こちらのAndroidサービスの中ではadbdは動いていませんので、もしAndroidのほうのadbdを起動して接続したい場合は、**図9**を実行してください。

LXC

Ubuntu TouchはAndroidのドライバを使ってデバイスを操作します。そのため、デバイスによっては、Androidのサービスを立ち上げる必要があります。そこでUbuntu Touchでは、Ubuntuが起動したあとにLXC（Linux Container）を用いて、Androidの仮想マシンを内部に立ち上げ、その中でAndroidのサービスを起動してます。

それを起動しているのが、/etc/init/lxc-android-config.confです。この中でlxc-startを実行してAndroidを起動しています。Androidのrootfsは/var/lib/lxc/androidの下にあります。ここには「デバイスイメージを分解する」のsystem.imgが展開されています。

Android自体は必要最低限のサービスしか動かさないようにしています。Android側へadbでログインしたうえで、psコマンドを実行した結果を**図10**に挙げておきます。おもにデバイス管理系のサービスしか動いていないことがわかります。

powerdとoFono

powerdはモバイルデバイス向けの電源管理サービ

図8　adbコマンドでスクリーンショット

```
$ adb shell /system/bin/screencap -p /tmp/ss.png
$ adb pull /tmp/ss.png ss.png
```

図9　Androidへのadbログイン

```
$ adb forward tcp:5555 tcp:5555
$ adb shell（デバイスにログイン）
# setprop service.adb.tcp.port 5555
# exit
$ adb connect localhost:5555

（以降、-sオプションでAndroidのadbdに接続）

logcatしたい場合：
$ adb -s localhost:5555 logcat
```

スです。サスペンドや復帰、バッテリーの統計情報の取得といった電源系の操作だけでなく、環境光によってディスプレイの明るさを調整したり、デバイスが高熱になったらシャットダウンシーケンスに入るといった操作もpowerdが担当します。

oFono[注5]は電話通信部分のAPIを提供するサービスです。WiFi版のNexus 7には必要ありませんが、Nexus 4などでは適切なSIMカードがささっていれば、このサービスにより通話できるようになります。

いずれのサービスもアプリ側でもコントロールするための、Qtインターフェースが備わっています。

ubuntu-touch-session

ubuntu-touch-sessionはユーザセッションそのものです。通常のUbuntuは、LightDMからログインすることでログインマネージャがユーザセッションごとにUpstartを起動し、ユーザにとって必要なサービスを立ち上げます[注6]。しかしUbuntu Touchは現在のところこのような処理を行うログインマネージャが存在しないので、このubuntu-touch-sessionサービス経由でユーザセッションを立ち上げるようにしています。

ユーザセッションで立ち上がるサービスは/etc/initではなく/usr/share/upstart/sessions/で設定されます。セッションリストは**図11**でも確認できます。

図11の結果にUbuntu Touchでは必要ないunity7が含まれているのは、Unity 8が依存するライブラリにUnity 7の起動スクリプトが含まれているためです。Ubuntu 13.10ではデスクトップ版はUnity 7を使うので、このまま残る予定です。

最後のmaliit-serverはMeeGo由来のインプットメソッドです。モバイル向けのソフトウェアキーボードをプラグインとして提供しており、Nokia N9などでも採用されていました。Ubuntu TouchではIBusではなく、このMaliitを使用しています。2013年9月の時点では日本語入力・変換の機能が実装されていないため、残念ながらUbuntu Touch上ではまだ日本語入力はできません。

注5） https://ofono.org/

注6） Upstartはシステムに必要なサービス（システムジョブ）とユーザ単位で必要なサービス（セッションジョブ）を区別し、セッションジョブについてはログインするタイミングで別プロセスのセッションinitを立ち上げてそこで管理するようにしています。

図10　Androidマシンのプロセス（抜粋）

```
shell@android:/ $ ps
USER     PID          NAME
root      1           /init
root      2           /sbin/ueventd
system    4           /system/bin/servicemanager
root      6           /system/bin/debuggerd
system    7           /system/bin/surfaceflinger
drm       8    中     /system/bin/drmserver
media     9           /system/bin/mediaserver
install   10          /system/bin/installd
root      11   略     /system/bin/ubuntuappmanager
system    16          /system/bin/sensorservice
```

図11　ユーザセッションのサービス一覧

```
$ export XDG_RUNTIME_DIR=/run/user/$(id -u phablet)
$ export UPSTART_SESSION=$(initctl list-sessions | cut -d' ' -f1)
$ initctl list
...
unity7 stop/waiting
...
unity8 start/running, process 908
...
maliit-server start/running, process 900
```

まとめ

これまで見てきたように、Ubuntu Touchは既存のUbuntuの資産をなるべく流用できるようなプラットフォームを構成したうえで、コミュニティベースでは対応しづらいハードウェア部分でAndroidの資産を流用するという手段をとっています。このためAndroidが動くデバイス（とくにCyanogenModがサポートしているデバイス）であれば、比較的ポーティングも容易ですので[注7]腕に覚えのある方はぜひ試してみてください。SD

≪参考文献≫

http://developper.ubuntu.com/get-started
https://wiki.ubuntu.com/Touch/Building

注7） https://wiki.ubuntu.com/Touch/Porting

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。

2013年10月号

第1特集
Vimを使いこなしてますか?
Vim至上主義

第2特集
ネットワーク技術力のパワーアップ
ルータの教科書

一般記事
・Key Value Storeをゼロから創る
・小規模プロジェクト現場から学ぶJenkins活用（4）

1,280円

2013年9月号

第1特集
今からはじめる
sed/AWK再入門

第2特集
開発するなら やっぱりMacですよね?
9人9色のデスクトップ拝見＋新OS傾向と対策

一般記事
・小規模プロジェクト現場から学ぶJenkins活用（3）
・最終段階に入ったFedora 19

1,280円

2013年8月号

第1特集
理論から実践まで
3Dプリンタが未来を拓く理由

第2特集
バグを狙い撃つ技術
システムを見通す力でソフトウェア開発を楽にしませんか!

一般記事
・小規模プロジェクト現場から学ぶJenkins活用（2）

1,280円

2013年7月号

第1特集
データの価値を見直しませんか?
ここからはじめるデータ分析学習

第2特集
自分の定規を持っていますか?
ボトルネックを探れ! 「ベンチマーク」活用テクニック

一般記事
・小規模プロジェクト現場から学ぶJenkins活用

1,280円

2013年6月号

第1特集
わかった人だけメキメキ上達
ちゃんとオブジェクト指向できていますか?

第2特集
研修じゃ教えてもらえない!?
あなたの知らないUNIXコマンドの使い方

一般記事
・リアルタイム分散処理「Storm」ほか

1,280円

2013年5月号

第1特集
IT業界ビギナーのための お勧め書籍55+α
新しい季節に君へ

第2特集
覚えておきたい、ちゃんと使いたい!
正規表現をマスターしていますか?

一般記事
・バーチャルネットワークコントローラ2.0開発の実際

1,280円

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%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。

Linux カーネル観光ガイド

第20回

カーネルパニック時のログを保存する ～pstoreのしくみ～

Text：青田 直大　AOTA Naohiro

カーネルハックをしている方は、一度はカーネルパニック（やOops）に襲われたことがあるかと思います。パニック時のログは、パニックの原因を探るためにたいへん便利なものです。カーネルパニックが起こっているので、ログファイルへの書き込みがうまくいかないこともあります。代わりに「再起動しても消えないメモリ」上の領域にログを書いておき、あとでそれを読むというpstoreというしくみが導入されています。Linux 3.12には、このpstoreでのデータの読み書きが自動的に圧縮／展開されるコードが入りました。今回はこのpstoreについて見ていきます。

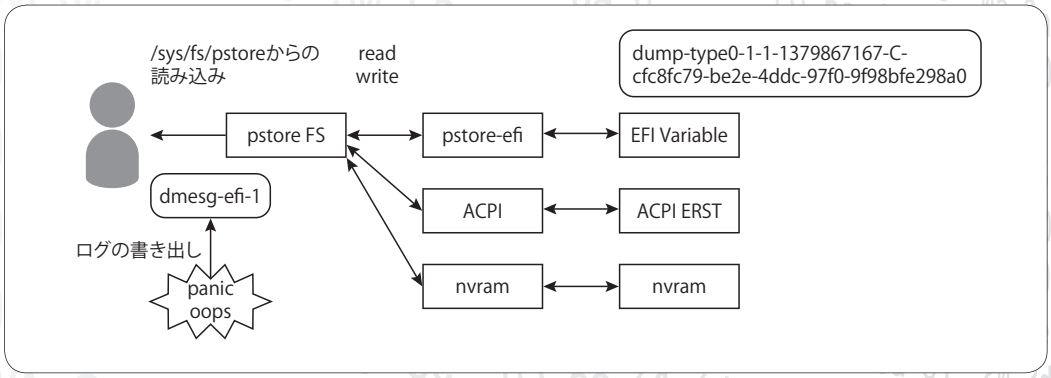
pstorefs

pstoreはファイルシステムの形でユーザランドからの読み出しを提供する部分と、個々のマシンに存在する不揮発メモリとのやりとりを行うドライバの部分に分かれています（**図1**）。まずはpstorefsの部分から見ていきましょう。

このpstorefsはfs/pstore/inode.cで実装されています。このモジュールが読み込まれると、/sys/fs/pstoreディレクトリが作られます。一般に、このディレクトリにpstorefsをマウントしておきます。カーネルパニックが起きた場合、ここにdmesg-<driver名>-<数字>といったファイルが作られます。これを**図2**のように数字逆順にcatするとパニック時のログが得られます。

さて、このpstorefsがマウントされると関数pstore_fill_super()から、pstore_get_records()が呼び出されます。ここでは“struct pstore_infopsinfo”に、ドライバ部分が登録する関数

▼図1　pstoreの構造



▼図2　数字逆順にcatする

```
$ mount -t pstore pstore /sys/fs/pstore
$ cd /sys/fs/pstore
$ % cat dmesg-efi-{5,4,3,2,1}
Oops#1 Part5
<3>[ 7812.595096] ata1.00: cmd c8/00:80:19:8f:c4/00:00:00:00:00/e4 tag 0 dma 65536 in
<3>[ 7812.595096]          res 51/40:00:19:8f:c4/00:00:00:00:00/04 Emask 0x9 (media error)
<3>[ 7812.595098] ata1.00: status: { DRDY ERR }
<3>[ 7812.595099] ata1.00: error: { UNC }
<6>[ 7812.698763] ata1.00: configured for UDMA/133
<6>[ 7812.704314] ata1.01: configured for UDMA/100
<6>[ 7812.704331] sd 0:0:0:0: [sda] Unhandled sense code
<6>[ 7812.704333] sd 0:0:0:0: [sda]
<4>[ 7812.704334] Result: hostbyte=DID_OK driverbyte=DRIVER_SENSE
<6>[ 7812.704335] sd 0:0:0:0: [sda]
<4>[ 7812.704337] Sense Key : Medium Error [current] [descriptor]
<4>[ 7812.704339] Descriptor sense data with sense descriptors (in hex):
<6>[ 7812.704340]         72 03 11 04 00 00 00 0c 00 0a 80 00 00 00 00 00
<6>[ 7812.704346]         04 c4 8f 19
<6>[ 7812.704348] sd 0:0:0:0: [sda]
<4>[ 7812.704350] Add. Sense: Unrecovered read error - auto reallocate failed
<6>[ 7812.704351] sd 0:0:0:0: [sda] CDB:
Oops#1 Part4
<4>[ 7812.704352] Read(10): 28 00 04 c4 8f 19 00 00 80 00
<3>[ 7812.704358] end_request: I/O error, dev sda, sector 79990553
<3>[ 7812.704360] btrfs: bdev /dev/sda2 errs: wr 0, rd 144, flush 0, corrupt 0, gen 0
<3>[ 7812.704366] btrfs: bdev /dev/sda2 errs: wr 0, rd 145, flush 0, corrupt 0, gen 0
<6>[ 7812.704387] ata1: EH complete
<6>[ 7812.704400] btrfs csum failed ino 3678248 extent 40955228672 csum 2248845604 wanted➥
668385290 mirror 0
<1>[ 7812.704442] BUG: unable to handle kernel paging request at ffff87fe78208290
<1>[ 7812.704473] IP: [<ffffffff81320edb>] __btrfs_map_block+0x2fb/0x1000
<4>[ 7812.704507] PGD 0
<4>[ 7812.704519] Oops: 0000 [#1] SMP
<4>[ 7812.704535] Modules linked in: nls_iso8859_1 vfat msdos fat ext3 jbd ext2 macvtap ➥
macvlan ebtables xt_CHECKSUM iptable_mangle hwmon_vid nfsd auth_rpcgss oid_registry nfs_acl➥
lockd sunrpc i915 snd_hda_codec_hdmi snd_hda_codec_realtek i2c_algo_bit intel_agp intel_gtt➥
 drm_kms_helper i2c_i801 drm snd_hda_intel agpgart snd_hda_codec pata_via
Oops#1 Part3
<4>[ 7812.704682] CPU: 1 PID: 24916 Comm: btrfs-endio-3 Tainted: G        W    3.11.0-rc1+ ➥
#275
<4>[ 7812.704707] Hardware name: System manufacturer System Product Name/P8H67-M PRO, BIOS ➥
3806 08/20/2012
<4>[ 7812.704734] task: ffff88041c6b0000 ti: ffff88016779a000 task.ti: ffff88016779a000
<4>[ 7812.704767] RIP: 0010:[<ffffffff81320edb>]  [<ffffffff81320edb>] __btrfs_map_
block+0x2fb/0x1000
<4>[ 7812.704804] RSP: 0018:ffff88016779ba50  EFLAGS: 00010286
<4>[ 7812.704822] RAX: ffff880166d873d8 RBX: ffff880166d873c0 RCX: 0000000000010000
<4>[ 7812.704842] RDX: ffff87fe78208268 RSI: 000000000000f000 RDI: 00000000afea5500
<4>[ 7812.704863] RBP: ffff88016779bb18 R08: 00000000000169c0 R09: ffff880166d873c0
<4>[ 7812.704884] R10: 000000000000379b R11: 0000000000000000 R12: 00000000afea5500
<4>[ 7812.704905] R13: ffff8805fa288a80 R14: 0000000000000000 R15: 000000000000f000
<4>[ 7812.704936] FS:  0000000000000000(0000) GS:ffff88061f240000(0000)
knlGS:0000000000000000
Oops#1 Part2
<4>[ 7812.704962] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
<4>[ 7812.704981] CR2: ffff87fe78208290 CR3: 000000000ec0c000 CR4: 00000000000427e0
<4>[ 7812.705002] Stack:
<4>[ 7812.705012]  ffffffff8173b107 000000000000379b 0000000000010000 00000000379c0000
<4>[ 7812.705041]  0000000000000000 000000000000379c 0000000000000000 ffffffff00000000
<4>[ 7812.705072]  000000000000379b ffff8805fa3becf8 ffff8805fa3bc000 ffff88016779bb98
<4>[ 7812.705102] Call Trace:
<4>[ 7812.705118]  [<ffffffff8173b107>] ? _raw_spin_unlock+0x27/0x30
<4>[ 7812.705140]  [<ffffffff812f09d0>] ? __btrfs_lookup_bio_sums.isra.9+0x420/0x550
<4>[ 7812.705166]  [<ffffffff81325c21>] btrfs_map_bio+0x71/0x570
<4>[ 7812.705186]  [<ffffffff813402d5>] btrfs_submit_compressed_read+0x355/0x4c0
<4>[ 7812.705221]  [<ffffffff812ffc71>] btrfs_submit_bio_hook+0x1d1/0x1e0
<4>[ 7812.705255]  [<ffffffff811fd0ae>] ? bio_alloc_bioset+0x12e/0x1e0
<4>[ 7812.705285]  [<ffffffff8131996d>] end_bio_extent_readpage+0x4dd/0xa90
Oops#1 Part1
```

次ページに続く

前ページの続き

```
<4>[ 7812.705319]  [<ffffffff811fb93d>] bio_endio+0x1d/0x30
<4>[ 7812.705345]  [<ffffffff812f3957>] end_workqueue_fn+0x37/0x40
<4>[ 7812.705375]  [<ffffffff813297ad>] worker_loop+0x12d/0x530
<4>[ 7812.705404]  [<ffffffff81329680>] ? btrfs_queue_worker+0x310/0x310
<4>[ 7812.705437]  [<ffffffff810adcca>] kthread+0xea/0xf0
<4>[ 7812.705464]  [<ffffffff810adbe0>] ? kthread_create_on_node+0x140/0x140
<4>[ 7812.705498]  [<ffffffff81743f5c>] ret_from_fork+0x7c/0xb0
<4>[ 7812.705525]  [<ffffffff810adbe0>] ? kthread_create_on_node+0x140/0x140
<4>[ 7812.705558] Code: 8b 55 98 41 8d 3c 14 85 d2 7e 3c 0f 1f 44 00 00 49 63 4d 10 49 63 d4 ➪
4c 89 fe 48 8d 14 52 41 83 c4 01 48 83 c0 18 49 8d 54 d5 00 <48> 03 72 28 49 0f af ca 48 01➪
f1 48 89 48 20 48 8b 52 20 48 89
<1>[ 7812.705757] RIP  [<ffffffff81320edb>] __btrfs_map_block+0x2fb/0x1000
<4>[ 7812.705792]  RSP <ffff88016779ba50>
<4>[ 7812.705811] CR2: ffff87fe78208290
<4>[ 7812.958956] ---[ end trace d59ec9ca9f2ec22f ]---
```

open、read、closeを使ってpstorefs内のファイルが作られます。read関数は次のような情報を返します。

ファイルデータの種類	type
同じ種類の中での識別	id
ファイルデータ	data
ファイルサイズ	size
ファイルが圧縮されているかどうかを示す	compressed
ファイルのタイムスタンプ	time

typeには、パニック時のデータを保管していることを示すPSTORE_TYPE_DMESGや、パニック時以外の一般的なカーネルのコンソールログを保管していることを示すPSTORE_TYPE_CONSOLEといったものがあります。こうしたtypeやidからファイル名が決定されます。たとえば、typeがPSTORE_TYPE_DMESGである場合、“dmesg-<ドライバ名>-<ID>”といった名前となります。ファイルデータが圧縮されている場合、このあとに“.enc.z”がつきます。保管されているものがPSTORE_TYPE_DMESGであり、データが圧縮されていた場合には透過的に展開されたデータが(圧縮されていたデータそのものの代わりに)ファイルに保管されるので、“.enc.z”がつくのは(展開用のメモリ領域の不足などを理由とする)データの展開に失敗した場合のみとなります。

また、pstorefs内のファイルを削除すると対応するデータがpsinfoに登録されているerase関数によって削除されます。

efi-pstore

始めにふれたようにpstoreはファイルシステム部分とドライバ部分とが分かれています。ドライバには現状EFI variableを使うもの、ACPIのERST(ErrorRecord Serialization Table)を使うもの、PowerPCのnvramを使うものがあります。ここではEFI variableを使うefi-pstoreについて見ていきます。

EFI(UEFI)とはBIOSを置き換えることを目的とした新しいOSとファームウェアのやりとりのための仕様です。この仕様の中にEFI variableというものがあります。これは「名前」と「GUID」とをキーとして、任意のバイナリの値を不揮発性メモリ上に記録し、(再起動後にも)参照できるようにするものです。UEFIではこれを利用してブートオーダーの決定などを行っています。たとえば、名前が“Timeout”で、GUIDが“8be4df61-93ca-11d2-aa0d-00e098032b8c”(EFI_GLOBAL_VARIABLE_GUID)をキーとしてUEFIのブートマネージャのタイムアウト時間が設定されています。

efi-pstoreでは、このEFI variableを使ってpstoreのデータ保持を実現しています。現在のバージョン(Linux 3.12)でデータ保持に使うEFI variableのGUIDは“cfc8fc79-be2e-4ddc-97f0-9f98bfe298a0”(LINUX_EFI_CRASH_GUID)で、名前は“dump-type<type>-<part>-<count>-

<time>-<compressed>"となります。typeは前述のread関数が返すtypeに対応し、partが同じくidと対応しています。countの部分は複数のoopsを記録するためにoopsごとに違う値になります。compressedはデータが圧縮されて保持されている場合には"C"で、そうでない場合には"D"となります。以前のバージョンではcountやcompressedの部分がない名前も使われていましたので、システムによってはそういう名前のEFIvariableも見つかるかもしれません。

efivarfs

それではEFIにログが保存されている様子を見てみましょう。ユーザランドからEFIを読むには/sys/firmware/efi/varsまたは/sys/firmware/efi/efivarsを用います。前者はsysfsの一部として実装されており、後者はefivarfsとして独自のファイルシステムとして実装されています。もともとEFI variableにはその値が1024byteまでという制限があったのですが、後に撤廃されました。そのため、sysfsとしてEFI variableの読み書きを提供することが難しくなりefivarfsが作られたという経緯があります。

/sys/firmware/efi/varsの下には各EFI variableに対応した"<名前>-<GUID>"というディレクトリがあります。そのディレクトリの中には次の5つのファイルが入っています。

- attributes
- data
- guid
- raw_var
- size

attributesにはそのEFI variableの属性が記載されています。NON_VOLATILEがシステムをリセットしてもEFI variableが残ることを、BOOTSERVICE_ACCESSはOSブート前の環境からこのEFI variableにアクセスできることを、RUNTIME_ACCESSはOSからこのEFI variableにアクセスできることを示しています。dataにはEFI variableの値が入っていて、sizeには値のサイズが書かれています。raw_varはEFI variableの名前、GUID、データ、属性といった情報の生のデータが入っています(**図3**)。

efivarfsのほうも名前とGUIDとでアクセスできるのは同じですが、ディレクトリではなく通常ファイルの形でのアクセスになります。最初

▼図3　EFI variable

```
efi % pwd
/sys/firmware/efi
efi % ls vars/Timeout-8be4df61-93ca-11d2-aa0d-00e098032b8c/
attributes  data  guid  raw_var  size
efi % sudo cat vars/Timeout-8be4df61-93ca-11d2-aa0d-00e098032b8c/attributes
EFI_VARIABLE_NON_VOLATILE
EFI_VARIABLE_BOOTSERVICE_ACCESS
EFI_VARIABLE_RUNTIME_ACCESS
efi % sudo hexdump -C vars/Timeout-8be4df61-93ca-11d2-aa0d-00e098032b8c/data
00000000  01 00                                             |..|
00000002
efi % sudo cat vars/Timeout-8be4df61-93ca-11d2-aa0d-00e098032b8c/guid
8be4df61-93ca-11d2-aa0d-00e098032b8c
efi % sudo hexdump -C vars/Timeout-8be4df61-93ca-11d2-aa0d-00e098032b8c/raw_var
00000000  54 00 69 00 6d 00 65 00  6f 00 75 00 74 00 00 00  |T.i.m.e.o.u.t...|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400  61 df e4 8b ca 93 d2 11  aa 0d 00 e0 98 03 2b 8c  |a.............+.|
00000410  02 00 00 00 00 00 00 00  01 00 00 00 00 00 00 00  |................|
00000420  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000820  07 00 00 00                                       |....|
00000824
efi % sudo cat vars/Timeout-8be4df61-93ca-11d2-aa0d-00e098032b8c/size
0x2
```

の4byteが属性を示し、そのあとにデータが続きます(**図4**)。

では、dmesg-efi-1に対応する部分を見てみましょう(**図5**)。dmesgなので"type0"でidとpartは1であり、タイムスタンプが1379867167なので、対応するEFIvariableの名前は"dump-type0-1-1-1379867167-C"になります。"C"がついているとおり、圧縮されているのでopensslを使って展開してみると確かにdmesg-efi-1に対応する部分が記録されていることがわかります。

pstoreへの書き込み

パニック時のpstoreへの書きこみはコールバック関数pstore_dump()によって行われています。パニックやOopsが起きるとこの関数がkmsg_dump()から呼び出されます。ログの圧縮がほどこされ、readと同様にpsinfoに登録されたwrite関数を用いて書きこみが行われます。pstoreがmountされている場合には、変数pstore_new_entryを1にしておきます。これでpstorefsから見えるファイルが更新されます。ただしこの更新はデフォルトでは無効になっています。pstoreのモジュールパラメータupdate_msに更新間隔(ミリ秒単位)を設定する必要があります。

その他の永続化

ここまでパニック時に書かれるログを中心に見てきましたが、pstoreにはほかにも保存できるものがあります。 1つは"console-efi"といった名前で保存されるパニック時以外にも書かれるカーネルログというだけでほとんどパニック時のものと変わりません。

pstoreに記録されるもう1つのデータがなかなかおもしろいものです。/sys/kernel/debug/pstore/record_ftraceに1を書くことで、pstoreにどの関数がどこのアドレスから呼ばれたかのデータが記録されるようになります。これをもとにして**図6**のような出力を得ることができます。パニックはせずにシステムがハングしてしまうような問題があるときに、どの関数が呼び出されていたかを知ることができるので問題の解決に役立てることができます。

command line partition

最後にLinux 3.12に追加された小ネタを1つ紹介しましょう。このバージョンではcommand line partition parserという機能も追加されています。カーネルの起動パラメータを使ってパーティションを指定するもので、おもに組み込みのFlashメモリに使われることを意図しているようです。たとえば"blkdevparts=mmcblk0:1G(data0),1G(data1),-"といったパラメータを指定すると、mmcblk0p1が1G、mmcblk0p2が1Gで残りの容量がmmcblk0p3となります。括弧の中はPARTNAMEとしてudevに送信されます。これを使ってユーザにわかりやすい名前のデバイスファイルを作ることができるというわけです。

まとめ

今回はカーネルパニック時のログを保存するpstoreのシステムを中心にpstoreのしくみについて解説しました。カーネルのデバッグに役立ててみてください。SD

▼図4　efivarfs

```
% hexdump -C efivars/Timeout-8be4df61-93ca-11d2-aa0d-00e098032b8c
00000000  07 00 00 00 01 00                                 |......|
00000006
```

▼図5　dmesg-efi-1に対応する部分

```
% ls -l --time-style=+%s /sys/fs/pstore/dmesg-efi-1
-r--r--r-- 1 root root 1779 1379867167 /sys/fs/pstore/dmesg-efi-1
% cat /sys/fs/pstore/dmesg-efi-1
Oops#1 Part1
<4>[   92.223539] RBP: ffff8805eb5d9ca8 R08: 0000000000000000 R09: ffff8800caf0d8c0
<4>[   92.223576] R10: 0000000000000000 R11: 0000000000000000 R12: ffff8805f171f5a8
<4>[   92.223612] R13: ffff8805f7d3abd0 R14: ffff8800caf0d900 R15: ffff8800caf0d8c0
<4>[   92.223650] FS:  0000000000000000(0000) GS:ffff88060600000(0000) knlGS:0000000000000000
<4>[   92.223691] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
<4>[   92.223722] CR2: 0000000000000000 CR3: 00000005e07b5000 CR4: 00000000000427e0
<4>[   92.223758] Stack:
<4>[   92.223772]  ffff8805f171f5a0 ffff8805f8a34000 0000000000000000 000000000003ffff
<4>[   92.223821]  ffff8805f7d3abd0 ffff8805f171f5a0 ffff8805f7d59b90 ffff8805eb5d9d88
<4>[   92.223868]  ffffffff81316127 ffff8805eb5d9d30 ffffffff00000050 ffffffff82095680
<4>[   92.223916] Call Trace:
<4>[   92.223934]  [<ffffffff81316127>] btrfs_finish_ordered_io+0x1e7/0xab0
<4>[   92.223970]  [<ffffffff81316c15>] finish_ordered_fn+0x15/0x20
<4>[   92.224002]  [<ffffffff81337e8d>] worker_loop+0x12d/0x530
<4>[   92.224033]  [<ffffffff81337d60>] ? btrfs_queue_worker+0x310/0x310
<4>[   92.224065]  [<ffffffff810ae9dd>] kthread+0xed/0x100
<4>[   92.224095]  [<ffffffff810ae8f0>] ? kthread_create_on_node+0x140/0x140
<4>[   92.224126]  [<ffffffff81763e6c>] ret_from_fork+0x7c/0xb0
<4>[   92.224158]  [<ffffffff810ae8f0>] ? kthread_create_on_node+0x140/0x140
<4>[   92.224190] Code: 75 b4 48 8b 45 c8 48 8b 40 08 49 39 c4 0f 95 c0 48 83 c4 10 5b 41 5c 41 5d ➥
41 5e 41 5f 5d c3 4c 89 e0 eb e6 c1 e8 1f 84 c0 74 b4 <0f> 0b 66 66 66 66 66 66 2e 0f 1f 84 00 00 00 ➥
00 00 66 66 66 66 90
<1>[   92.224392] RIP  [<ffffffff8130bdc0>] record_extent_backrefs+0xd0/0xe0
<4>[   92.224429]  RSP <ffff8805eb5d9c70>
<4>[   92.232153] ---[ end trace a477e7e5ba88ba55 ]---
% sudo cat /sys/firmware/efi/vars/dump-type0-1-1-1379867167-C-cfc8fc79-be2e-4ddc-97f0-9f98bfe298a0/➥
data|openssl zlib -d
Oops#1 Part1
<4>[   92.223539] RBP: ffff8805eb5d9ca8 R08: 0000000000000000 R09: ffff8800caf0d8c0
<4>[   92.223576] R10: 0000000000000000 R11: 0000000000000000 R12: ffff8805f171f5a8
<4>[   92.223612] R13: ffff8805f7d3abd0 R14: ffff8800caf0d900 R15: ffff8800caf0d8c0
<4>[   92.223650] FS:  0000000000000000(0000) GS:ffff88060600000(0000) knlGS:0000000000000000
<4>[   92.223691] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
<4>[   92.223722] CR2: 0000000000000000 CR3: 00000005e07b5000 CR4: 00000000000427e0
<4>[   92.223758] Stack:
<4>[   92.223772]  ffff8805f171f5a0 ffff8805f8a34000 0000000000000000 000000000003ffff
<4>[   92.223821]  ffff8805f7d3abd0 ffff8805f171f5a0 ffff8805f7d59b90 ffff8805eb5d9d88
<4>[   92.223868]  ffffffff81316127 ffff8805eb5d9d30 ffffffff00000050 ffffffff82095680
<4>[   92.223916] Call Trace:
<4>[   92.223934]  [<ffffffff81316127>] btrfs_finish_ordered_io+0x1e7/0xab0
<4>[   92.223970]  [<ffffffff81316c15>] finish_ordered_fn+0x15/0x20
<4>[   92.224002]  [<ffffffff81337e8d>] worker_loop+0x12d/0x530
<4>[   92.224033]  [<ffffffff81337d60>] ? btrfs_queue_worker+0x310/0x310
<4>[   92.224065]  [<ffffffff810ae9dd>] kthread+0xed/0x100
<4>[   92.224095]  [<ffffffff810ae8f0>] ? kthread_create_on_node+0x140/0x140
<4>[   92.224126]  [<ffffffff81763e6c>] ret_from_fork+0x7c/0xb0
<4>[   92.224158]  [<ffffffff810ae8f0>] ? kthread_create_on_node+0x140/0x140
<4>[   92.224190] Code: 75 b4 48 8b 45 c8 48 8b 40 08 49 39 c4 0f 95 c0 48 83 c4 10 5b 41 5c 41 5d ➥
41 5e 41 5f 5d c3 4c 89 e0 eb e6 c1 e8 1f 84 c0 74 b4 <0f> 0b 66 66 66 66 66 66 2e 0f 1f 84 00 00 00 ➥
00 00 66 66 66 66 90
<1>[   92.224392] RIP  [<ffffffff8130bdc0>] record_extent_backrefs+0xd0/0xe0
<4>[   92.224429]  RSP <ffff8805eb5d9c70>
<4>[   92.232153] ---[ end trace a477e7e5ba88ba55 ]---
```

▼図6　pstoreに記録されたftraceの関数呼び出しログ

```
0 ffffffff8101ea64  ffffffff8101bcda  native_apic_mem_read <- disconnect_bsp_APIC+0x6a/0xc0
0 ffffffff8101ea44  ffffffff8101bcf6  native_apic_mem_write <- disconnect_bsp_APIC+0x86/0xc0
0 ffffffff81020084  ffffffff8101a4b5  hpet_disable <- native_machine_shutdown+0x75/0x90
0 ffffffff81005f94  ffffffff8101a4bb  iommu_shutdown_noop <- native_machine_shutdown+0x7b/0x90
0 ffffffff8101a6a1  ffffffff8101a437  native_machine_emergency_restart <- native_machine_restart+0x37/0x40
0 ffffffff811f9876  ffffffff8101a73a  acpi_reboot <- native_machine_emergency_restart+0xaa/0x1e0
0 ffffffff8101a514  ffffffff8101a772  mach_reboot_fixups <- native_machine_emergency_restart+0xe2/0x1e0
0 ffffffff811d9c54  ffffffff8101a7a0  __const_udelay <- native_machine_emergency_restart+0x110/0x1e0
0 ffffffff811d9c34  ffffffff811d9c80  __delay <- __const_udelay+0x30/0x40
0 ffffffff811d9d14  ffffffff811d9c3f  delay_tsc <- __delay+0xf/0x20
```

November 2013

NO.25

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

初の試みがめじろ押し!　LLまつり開催!

11年目を迎えた軽量プログラミング言語の祭典Lightweight Languageイベントですが、今年は「Lightweight Language Matsuri」(通称：LLまつり)と題して開催しました。参加者は395人でした。今回は初めて複数トラックのプログラムを編成しましたので、トラック別に内容を紹介します。

Lightweight Language Matsuri

■Lightweight Language Matsuri
【日時】2013年8月24日(土) 10:30～20:00
【会場】すみだ産業会館

■ディスカッショントラック

例年行っているパネルディスカッションのトラックです。今年は次の4セッションを実施しました。

- 顕在化する？○○言語のリスクとは？
- とびだせ！LL～リアル世界をプログラミングする～
- Infrastructure as LL
- Beyond JS～JavaScriptの改良に未来はあるか？

「顕在化する？○○言語のリスクとは？」は、現在使っている言語がこの先も生き残っていけるかどうかを議論するもので、本イベントでは初めてDelphiやD言語の登壇者が登場しました。「とびだせ! LL」では、ニコニコ学会の事例や、新郎の位置情報を新婦に送り続ける指輪デバイスなど、LLを使ったフィジカルコンピューティングの世界が紹介されました。「Infrastructure as LL」はLLを用いたインフラ運用をテーマとするセッションで、クックパッドなど大規模サイトの事例紹介や運用上の問題点の検討がなされました。「Beyond JS」はJavaScriptを改良しようとするさまざまな活動を総覧するもので、言語の開発に携わる方々同士の濃い議論が展開されました。

■プレゼンテーショントラック

このトラックは各自の取り組みを20分で発表するもので、LLイベントでは初めて実施しました。

- GNUプロジェクト、30周年
- 勉強会アンチパターン
- Javascriptエンジンによるアプリの動的実行環境を作っている話
- ニコニコ学会βとIT勉強会の幸せな関係
- Lightweight Language“Go”
- 関数型志向Python
- LL好きにおすすめ：こんなこともできるWindows PowerShell
- A language for the rest of us
- HTML5とか今風の技術を使ってWebサービスを作る？　wri.peの作り方
- Rubyで創るOpenFlowネットワーク
- プロトコルの変化からみるInternetの未来予測
- クロスサイト・スクリプティング再入門
- 加速器制御とLL
- 天文学におけるLL利用の過去と現在
- 会場ネットワーク提供の憂鬱

公募によるエントリーと実行委員会からの依頼に

よる発表を織り交ぜた結果、GoやWindows Power Shellなど本イベント初登場の言語の紹介、wri.peの作り方など新しいサービスの紹介、OpenFlowやクロスサイト・スクリプティングなど関連分野の技術動向、加速器制御や天文学など異業種におけるLLの活用事例、ニコニコ学会βや勉強会の動向など、これまでには実現できなかったバラエティ豊かな発表が集まりました。また、「関数型志向Python」は発表者のエキサイティングなパフォーマンスが好評でした。

■チュートリアルトラック

初学者もLLイベントに参加してもらえるようにという意図で初めて導入した企画です。

- 第6回チャンピオンシップシェル芸ランナー勉強会in LLまつり
- モダンPHPチュートリアル～PHP4時代の知識を捨て、PHP5.5時代へ～
- RailsGirls, More! & Rails寺子屋出張版

LLイベントではこれまであまり初心者を考慮したセッションを組んでこなかったので、どれぐらいの参加者が集まるか不安でしたが、始まってみるとどのセッションも20人前後の参加があり、チュートリアルとしては適正な人数で実施できてよかったです。Railsのチュートリアルでは、予定されていたコーチ陣に加えてRubyの作者やコミッターが教えるというぜいたくな場面も見られました。

■ライトニングトーク

毎年イベントの掉尾（ちょうび）を飾るセッションで、今年は10件の発表がありました。演題を次に掲載します。

- Lua組み込みプログラミング
- Pythonで機械制御
- LLから無線通信で電子工作
- みんなのはなもげら
- シェルスクリプトで簡易ping監視
- JavaScriptで行う時系列解析
- 5分で絶対に分かるPyCon APAC 2013 in Japan
- PHP Matsuriの光と闇
- 軽量のススメ
- LL Maker

今回は「Pythonで機械制御」「LL Maker」などハードウェアとLLを絡めた発表と、「PHP Matsuriの光と闇」のような各言語のイベント紹介が多かったです。また、「みんなのはなもげら」は百人一首風の短歌を自動生成するサービスを作った話で、生成される短歌がなんとも言えないおもしろさを醸し出していて好評でした。

■その他の催し

このほかにLLまつりでは次の催しを行いました。

- **書籍の販売／展示**
  今年は、書籍販売は5社、見本誌展示は6社が参加しました
- **LL QUIZ**
  会場でちょっとしたプログラミングを楽しんでもらおうという企画です。送信された数字の羅列に隠された文字を浮かび上がらせるという課題が出されました
- **参加者向け無線LANの提供**
  今年は多少のトラブルに見舞われたものの、おおむね無事に運用できました。今や大規模イベントにおける参加者向けインターネットの供給は重要かつ難しい課題の1つですが、その点でもLLイベントは良い先駆者になっていると考えています

◆ ◆ ◆

LLイベントも10回を超え、新たな展開を模索する中で、今年は今までにない試みを多数採り入れたチャレンジングなイベントとなりました。上述したプログラム編成以外の部分でも、チケット販売方法の変更、ペア券の導入、当日スタッフの公募などを行いました。今後も試行錯誤を繰り返しながら、より良いイベントを目指していきたいと思います。SD

Hack For Japan
エンジニアだからこそできる復興への一歩

●Hack For Japanスタッフ
及川 卓也　OIKAWA Takuya
Twitter @takoratta

第23回 Spending Data Party 2013と「税金はどこへ行った?」の広がり【その2】

社会的課題をテクノロジで解決するためのコミュニティ、Hack For Japanの活動をレポートする本連載。今回は前回に続き、税金の使い道を可視化するための取り組みについて紹介します。

今回も前回に引き続き、7月20日と7月21日に行われた「Spending Data Party 2013」のレポートを続けます。また、石巻市版の開発の模様とその後の「税金はどこへ行った?」プロジェクトの状況についてもご紹介します。

Spending Data Partyの成果

7月20日の後半と7月21日は、自治体対応を行うグループと新機能開発を行うグループ、そして今後の取り組みなどについて議論するグループの3グループに分かれて作業が進められました。

自治体対応

自治体対応グループは、7月21日のうちに、15の自治体がOpenSpendingにデータの登録を完了し、さらに8つの自治体がサイトの開設を完了しました。7月21日中の完成にはこぎつけなかったもののその後開設した自治体を合わせると、原稿執筆時で46都市が対応しています。

その中には、独自のユーザインターフェースですでにサイトを持っていた鳥取県[注1]や静岡県[注2]のように都道府県レベルで対応する自治体も含まれています。これだけ多くの都市が対応しはじめていると、自分の自治体が対応しているかどうかを探すのも大変になってきたので、そろそろ都道府県から絞り込む機能や地図から辿っていける機能などが望まれるとの要望も出ていました。実際その後、日本版「税金はどこへ行った?」の総合サイトであるspending.jpのリニューアルが行われ、都道府県別に分類はされるようになりました。

新機能開発

新機能開発としては、ソーシャル機能の追加が挙げられます。現状の「税金はどこへ行った?」は可視化された自治体のデータを住民が見るという一方向です。住民が税金の使い道について意見があったとしてもサイト上では伝えるしくみがありません。もし、ここにFacebookの「いいね!」やGoogle+の「+1」が付けられたり、コメントを付けられたりすれば、さらに住民参加が進むかもしれません。

Spending Data Party開催前のOpenSpending側との会議で、米国オークランドの例[注3]が紹介されたのですが、その中身は実に簡単でした。コメントはDISQUS[注4]、共有などはAddToAny[注5]を使っています。どちらのサービスも質問に答えていくことで生成されるJavaScriptコードを埋め込むだけで利用できます。

これを現在、東京都小金井市版[注6]に実験的に埋め込んでみています(**図1**)。同様のサービスはほかにもあるので、もしかしたら日本向けには別のサービスを組み込んだほうが良いかもしれないと思っています。

また、現在の「税金はどこへ行った?」は本家OpenSpendingの英国版を元にしたものとなっていますが、本家側でも世界各地の自治体対応を容易に

注1) http://tottori.spending.jp/
注2) http://shizuoka-ken.spending.jp/
注3) http://openbudgetoakland.org/mayor_13-15_proposed.html
注4) http://disqus.com/
注5) http://www.addtoany.com/
注6) http://koganei.spending.jp/

◆図1　小金井市版でソーシャル機能を組み込んだ例

するための改善が行われています(**写真1**)。

OpenSpendingプロジェクトは、予算や決算情報を可視化するための元データを登録するopenspending.orgを中心とした、いくつかのサブプロジェクト(モジュール)により構成されています。「税金はどこへ行った?」のサイトの見える部分は、現在ではSatellite Templateというモジュールで実現されるものとなっています。また、税金処理を計算するモジュールはTaxmanというものになります。余談になりますが、Taxmanという名前の由来はビートルズが当時の英国の高い税率を皮肉って歌った同名の曲から来ています。まとめると**表1**のようになります。

日本版もいつかこれらの新しいモジュールに移行するために、いくつかの作業が必要となります。Taxmanに日本の税処理を追加する作業は神保嘉秀氏が対応し、8月に無事修正がマージされています[注7]。Satellite-templateについては、西林孝氏と高野光弘氏によりマルチテナント対応が試みられました。

現在の「税金はどこへ行った?」ではほとんどソースコードが共通にもかかわらず、自治体を追加しようと思ったらGitHub上でプロジェクト全体をフォークし、独自にGitHub Pagesを立ち上げなければいけません。プログラム本体は共通とし、自治体独自のデータをパラメータで指定するだけで複数の自治体をサポートできるようにしようというのが、マルチテナント対応です。残念なことに、プロジェクトが使っているGitHub PagesのJekyll(Rubyを用いたサイトジェネレータ[注8])では複数のインスタンスをサポートすることがそのままでは難しく、まだ実現されていません。ただ、西林氏はGoogle Analysisのトラッキングコードがハードコードされている部分を修正するなど本家側への貢献を行い、同日には同じようにSpending Data Partyに参加している他国のエンジニアをIRCでサポートするなどもされていました。

◆写真1　OpenSpendingのモジュール構成を説明する西林孝氏

◆表1　OpenSpendingのサブプロジェクト(一部)

モジュール名	役割	GitHubの場所
Satellite Template	「税金はどこへ行った?」(Where Does My Money Go?)サイトを作るためのテンプレート	https://github.com/openspending/satellite-template
Taxman	汎用税金処理モジュール	https://github.com/openspending/taxman
Openspending	openspending.orgの機能モジュール	https://github.com/openspending/openspending
Openspending JS	openspendingのJavaScript。dailybreadという1日あたりの使途を表示する部分やbubblemapという使途別予算額を実現する部分のJavaScriptなど	https://github.com/openspending/openspendingjs

注7) https://github.com/openspending/taxman/pull/4

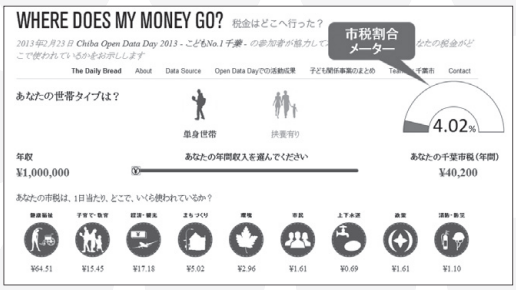
注8) http://jekyllrb.com/

今後の取り組み

今後の取り組みを検討したグループでは、いくつかのアイデアが議論されました。その1つが、可視化をさらに進め、所得額に占める税金の割合を「市税メーター」という形でわかりやすく表示するアイデアです（**図2**）。スライドバーで年収を変化させると、それに応じてメーターも変化するようにします。これによって収入と税負担の関係性がよりわかりやすくなると考えられます。

もう1つのアイデアが、実際に行政コストを削減した場合に、それで何が行えるかを示すアプリケーションです（**図3**）。村上龍氏の著作に「あの金で何が買えたか」というものがありますが、それと同じように市民の努力で行政コストを削減すると、別の何にそれを活用できるかが示されます。

◆**図2　市税メーターのアイデア（松島隆一氏の資料より）**



◆**図3　行政コストの削減でできることを可視化するアイデア（江口晋太郎氏の資料より）**

その後のプロジェクト

Spending Data Partyの後もサポートされる自治体は増え、冒頭でも紹介したように本誌原稿執筆時点で46自治体が対応されています。また、島根県版を開発された一般財団法人島根総合研究所により、日本全国版も提供されました[注9]。これは次のような特徴を持っています（開発者ブログ[注10]より）。

- 日本全国1800市区町村に対応。地図から簡単に検索可能
- 最大3市区町村まで比較可能
- 歳出（*目的別）に加え、歳出（*性質別）や歳入（*財源の確認）、地方債現在高などに対応
- 市区町村民税に加え、都道府県民税や国税（所得税と消費税）、社会保険に対応。それぞれについて税負担率を円グラフで可視化
- 税額計算においては、年収・年齢・配偶者・扶養家族を考慮したうえで、各種控除などを含めて計算
- 人口ピラミッドなどの人口統計を可視化
- 関連度の高い類似する市区町村を自動的に表示。初期設定は人口総数だが、100以上のカテゴリから選択可能
- 市区町村ごとの固定ページを用意

以前より自治体間の比較は希望の多かった機能なので、このサイトが提供されたことは大きな意味を持つことではないでしょうか。

さらに本家への貢献としては、ドキュメントの翻訳対応などが行われています。Facebookグループ[注11]やGoogleグループ[注12]で開発者間のコミュニケーションが行われていますので、興味を持った方はぜひとも参加してみてください。SD

注9）http://spending.souken.or.jp/
注10）http://ma-bank.com/item/1491
注11）https://www.facebook.com/groups/spendingjp/
注12）https://groups.google.com/forum/#!forum/spendingjp-dev

Column 税金はどこへ行った? 宮城県石巻市版

Spending Data Partyの前の6月27日に、宮城県石巻市版の「税金はどこへ行った?」のサイト[注1]が開設されました。サイトを制作したのは石巻復興プロジェクト[注2]のメンバーですが、筆者もその作業をサポートしました。

石巻の復興を支援することを目的とする石巻復興プロジェクトは、東京都大森に「石巻マルシェ@大森ウィロード山王店」という石巻の物産を販売する店舗を持っています。この石巻マルシェはコミュニティスペースとしても機能しているのですが、そこで5月に行われた交流会に呼ばれたのをきっかけとして、石巻市版の制作は開始されました。

実は、筆者は石巻市版の制作の前に地元自治体用のサイトを制作したのですが、その際は自治体のWebサイトに掲載されていた財源データの内容をほぼそのまま採用しました。筆者自身「税金はどこへ行った?」のサイト制作が初めてだったために、複雑なことを避けたというのも理由の1つではあったのですが、それよりも市の財政上注目すべき点がどこにあるのか個人では判断が難しかったためです。

一度作業を行ったことのある技術者ならば、「税金はどこへ行った?」の自治体対応を行うのは実はさほど難しくはありません。難しいのは、財政データの分析と分類です。自治体によって、サイトに掲載されている財政内容がどのような形式でどの程度まで細かいかが異なります。また、住民がどのような点に注目しているかも自治体によってまったく異なります。石巻市版の制作にあたっては、この作業を石巻復興プロジェクトのメンバーと、大森の石巻マルシェで行いました。

具体的には、まず、石巻市のサイトに掲載されている平成25年度の予算データ[注3]を見ました。石巻市が公開している予算書は大変細かい内容まで公開されているため、これをまず理解し、次に石巻市としてはどのような区分けにすべきかをメンバーで話し合いました。

いろいろな意見が出ましたが、最終的には、やはり震災からの復興途中である被災地であることを考え、復興やまちづくりに注目するようにしました。これを決めてからは、マジックペンと付箋紙というアナログツールを使って、一度予算書に書いてある項目を分解し、再度まとめます。メンバーが考えた区分けは「震災関係」「健康・福祉・環境」「公債」「子育て・教育」「その他」「まちづくり」「産業」「安全」という8つです。最初の「震災関係」の中には、残りの7つの区分けが入ります。これは、それぞれの区分けごとに「震災関係」のために特別実施されている事業があるためです。

区分けが決まったら、ここからはデジタルデータとしての入力です。数名で分担してデータを入力したところで、補正予算の扱いをどうするかと疑問が浮かんできました。できるだけ最新の正確なデータにするべきと考えたので、補正予算による補正を行ったデータができたところでその日の作業は終了とし、あとはオンラインでプロジェクトを続けることとしました。

その後、サイト自体はデータの入力が完了してから1週間ほどでメンバー向けには立ち上がったのですが、改めて予算書を見ていたときに疑問がわいてきました。それは、一般財源以外に国庫支出金や県支出金を財源とする予算があることです。とくに震災復興関係の予算にそれらは多くありました。果たして、石巻市民からの税金の使い道を可視化するという目的を考えた場合に、一般財源以外のものを含めて見せることが正しいのだろうか。そんな疑問が出てきたのです。

一般財源のみを予算とするのか、それともすべてを含めて構わないのか。どちらが正しいのかはわかりませんでした。「税金はどこへ行った?」自体が税金の流れをかなり簡略化して可視化していることもあり、おそらく正解というものはないのだと思います。そこで、石巻市版では、「一般財源のみ」と「一般財源と国庫支出金・県支出金」を財源とする2つのモデルを可視化できるようにしました。

また、アイコンも石巻に合ったものにしたいと、3つほど追加を行いました。アイコンはSVGで用意するのですが、使っているライブラリ(Raphael)の制限などもあり、Illustratorで作ったものをかなり手直ししました。

実は、この2つの作業に思ったよりも時間がかかってしまったため、データが用意できてからサイトが開設されるまで1ヵ月近くかかってしまいました。かなり手がかかったこともあって、筆者もメンバーもより愛着のあるものとなっています。このときの様子は石巻復興プロジェクトのブログ、石巻復興プロジェクト発!「税金はどこへ行った?」石巻版[注4]にも掲載されていますのでぜひご覧ください。

注1 http://ishinomaki.spending.jp/
注2 http://reishinomaki.net/
注3 http://www.city.ishinomaki.lg.jp/d0030/d0020/d0170/d0010/d0020/d0050/index.html
注4 http://reishinomaki.net/archives/3767

温故知新
IT むかしばなし

第27回

FUJICとパラメトロン

北山 貴広　KITAYAMA Takahiro　Twitter：@kitayama_t　kitayamat@gmail.com

はじめに

今回は、日本でコンピュータが生まれたころ（1950～1960年代）の特徴的な2つの出来事について歴史を振り返ってみたいと思います。

コンピュータの世代

初期の電子式コンピュータでは、コンピュータの論理回路を実現するために使われた素子のテクノロジで世代が分類されています。第1世代の真空管、第2世代のトランジスタ、そして第3世代のIC（Integrated Circuit）です。第1世代の真空管と第2世代のトランジスタはまったく異なった素子のため明確に定義されていますが、第3世代以降はIC、LSI（Large Scale IC）、VLSI（Very Large Scale IC）と集積度の違いで質的な変化はないため、第4世代をVLSIとしたり電卓用に作成した4004から発展したマイクロプロセッサとしたりと定義がいろいろあるようです。

ENIACとEDVACで知られている第1世代の真空管式コンピュータは、1940年代から1950年代にかけて製造され、1950年代に主流となりました。その後1960年代に入ると信頼性に優れ消費電力の少ない第2世代のトランジスタにとって代わられました。

第2世代のトランジスタは1950年代に開発が始まり1960年代には安定したシリコントランジスタが使われて主流となりました。1960年代は第3世代のICによる実装も始まっていて、第2世代と第3世代との混在した時代でした。

今回紹介するFUJICは第1世代の真空管コンピュータ、パラメトロンは第1世代の真空管と第2世代のトランジスタに挟まれた時期に日本独自で開発されたコンピュータ素子です。

FUJIC

日本で最初に稼働した電子式コンピュータは、富士写真フイルムの岡崎文次氏が開発した真空管コンピュータのFUJICで、1949年に開発に着手し1956年3月に稼働しました。現在、東京の上野にある国立科学博物館の地球館2Fに常設展示されています。

FUJIC開発の目的は、カメラレンズの設計で必要な大量で複雑な計算を自動化することで、高級レンズでは計算だけで数ヵ月かかった設計期間を短縮する効率化でした。実際に設計速度は人手の千倍から二千倍ぐらいに上がったそうです。

FUJICの特徴は、故障の多い不安定な真空管の本数を極力減らすために論理回路を工夫した点にあります。ENIACの真空管本数は17,468本だったのに対し、FUJICは1,700本と1桁少ない本数でした。また、真空管にかける電圧も極力下げていました。

また入出力装置はカードリーダー入力と電動タイプライタ出力で、当時としては非常に充実した装備でした。電動式タイプライタのキーにワイヤをひっかけて、ワイヤを引っ張ることで自動的にタイピングして印字出力する非常にユニークなしくみを備えていました（このユニークなしくみを書籍で知ってから国立科学博物館でFUJICの実物を見たときはテンション上がりました（笑））。

また、FUJICは一企業の一個人が開発したことが非常に特徴的です。第2次世界大戦中に開発を始めたENIACの記事がニューヨークタイムズに掲載され、世の中にコンピュータの存在が知られ

たのは1947年でした。このころ、世界に限らず日本でもあちらこちらでコンピュータの開発が始まりました。開発形態の主流は、政府や企業、大学の研究機関などで行う組織的なプロジェクトです。それに対してFUJICは手動の計算機で、人手で何百日もかかるカメラのレンズ計算を自動で高速に計算させるニーズから、本業のかたわら一個人がこつこつと部品を買いそろえて作ってしまったところにおもしろさを感じます。文献が少なかったから調査する時間が少なかったとか、1人で開発したため会議や調整に手間取らなかったことも手伝って、当時の技術で何がどこまでできるかを見極めて、コンパクトな仕様で実装まですべて行った岡崎文次氏のすごさをあらためて感じました。

パラメトロン

パラメトロン(Parametron)とは、1954年に当時東京大学大学院（理学部高橋秀俊研究室)の学生であった後藤英一氏によって発明された日本独自のコンピュータ素子で、パラメータ励振という原理を利用することから名付けられました。

パラメトロン素子はフェライトコア2個とコンデンサ1個で構成されます。フェライトコアに2本の電線をコイル状に巻き電流を流すことで電気的な振動が発生し励起現象で振動が増幅されます。2分の1分周で生じる180度異なる位相を利用して2値を取る特性を利用します。

当時は真空管もトランジスタも非常に高価で寿命も短く不安定でしたが、パラメトロンで使用するフェライトコアは非常に安価でした。またパラメトロンは安定性が高かったため長時間の連続稼動が可能という特徴がありました。

パラメトロンを使ったコンピュータは、1957年に電気通信研究所のMUSASINO-1（通称M1）が最初に稼働しました。その後、東大の高橋研究室で1958年にPC-1（パラメトロン4,300個、36ビット語長)、1961年にPC-2（パラメトロン13,000個、48ビット語長）が開発されました。企業でも日立製作所で「HIPAC MK-1」「HIPAC 103」、日本電気で「NEAC 1101」などが開発されました。

真空管からトランジスタが使われるまでの数年間、パラメトロンは日本のコンピュータで実装され、非常に遅い速度ながらも数日間安定稼動したことから実用に供されていました。その後、高速で消費電力が小さいトランジスタコンピュータが出てきたため、パラメトロンコンピュータは1960年代には開発が打ち切られました。

パラメトロンコンピュータが出荷されたのは2年間と非常に短い期間でしたが、日本では確かにパラメトロンコンピュータという時代が存在していました。

終わりに

コンピュータの演算素子が真空管からトランジスタに変わるころ、真空管の本数を極限まで減らして電圧も下げてコンパクトに実装したFUJICと、不安定な真空管とトランジスタが安定するまでのわずかな数年の間安定稼働したパラメトリック。その時代に利用可能な素子をよく理解してシステムを稼働させた実績は、日本のコンピュータの黎明期に誇れる存在だと思います。

先日、書店で『新装版 計算機屋かく戦えり』（遠藤諭著、アスキー、ISBN978-4756146786；**写真1**）を見つけて読んだところ、TK-80EやApple IIなど筆者が経験として知っているマイクロプロセッサ開発後の歴史以前の日本のコンピュータ開発史が、関係者のインタビューにより成功談も失敗談も経験による実感がこもった文章でつづられていました。筆者は4年ほど前から東京に在住してるため、以前は実感のなかった上野の国立科学博物館や武蔵野にあるNTT技術史料館も身近になりました。開発者のインタビューを読んでから国立科学博物館を訪問してFUJICの実機に会えたときはとても感慨深いものがありました。武蔵野のNTT技術資料館の一般公開日は時期により平日の午前か午後と限られていますが、機会を見つけてパラメトリック計算機のMUSASINO-1Bにも会いに行きたいと思います。SD

▼写真1

CyberAgentのエンジニアに直撃！

インターネットサービスのこれから未来を創る人たち28

Android端末の動作検証の課題を解決（前編）

取材／文：㈱インサイトイメージ 川添 貴生　KAWAZOE Takao　mail@insightimage.jp　　撮影：中村 俊哉　NAKAMURA Toshiya

サイバーエージェントでは現在数多くのAndroid向けアプリを提供していますが、問題となるのはそれぞれのAndroid端末を個別に検証しなければならないこと。この課題を解決するべく立ち上がった、ブルンナー グンタ氏（**写真1**）とキンヌネン シモ氏（**写真2**）に具体的にどのような課題があるのかを伺いました。

写真1 ブルンナー グンタ氏

多数の製品が存在するAndroidにおける検証の苦労

インターネットに接続してさまざまなアプリやコンテンツを楽しめる新たなモバイルデバイスとして、スマートフォンが普及しています。このスマートフォンはiPhoneとAndroidの2強対決となっていますが、それぞれで製品展開の形は大きく異なります。

iPhoneはアップルがハードウェアからソフトウェアまで一括提供しており、モデル数もけっして多くありません。一方Androidは、基本的にオープンソースでコードが公開されていて、さまざまなハードウェアベンダが独自にAndroidをカスタマイズし、さらにハードウェアスペックも異なる製品を発売しています。このため、Android向けのWebアプリやネイティブアプリは製品によって挙動が異なることがあるため、機種ごとに個別に動作検証をしなければなりません。これが大きな負担となっているのです。

もちろん、日本向けのサービスであれば、日本市場で一般に流通しているAndroid端末に絞ることも可能でしょう。ただ、それでも数十もの製品があるため、検証作業はけっして容易ではありません。

こうした課題を解決すべくブルンナー グンタ氏とキンヌネン シモ氏が開発したのが、複数台のAndroidをいっせいに操作することで検証作業の負荷を大幅に軽減するツールです。このプロジェクトはどのように始まったのでしょうか。

「私たち2人は別々のゲームを作っていたのですが、プロジェクトの移動で別の部署に異動し、2人で何をするか相談していました。最初は、2人でアプリ開発を支援するライブラリを作ろうと考えました。しかし、ライブラリを作れば、当然その検証が求められます。とくに、汎用的に使われるライブラリでは検証がたいへんで、多数の端末で調査しなければなりません。それをどうにかできないかと考えたのが最初です（グンタ氏）」

ブラウザが持つバグの影響を受けやすいWebアプリ

こうした検証は、携帯通信事業者のオフィスに“缶詰”になってテストを繰り返すことになると言います。

「サイバーエージェントにも検証用のスマートフォンがいくつかありますが、すべてではないので、携帯通信事業者に出向いて1日ずっと各端末を触りなが

ら検証するという作業を行っていました。これが本当にたいへんだったのです（グンタ氏）」

さらに、Android端末では予想外のバグに出くわすことも多いようです。

「あるメーカーの製品には、大量のバグが潜んでいました。たとえばHTML5 Canvasを使うと、いきなり端末がクラッシュするなど致命的なバグが放置されていたんです。しかし開発者としては、そういった製品でもサポートしたい。それは本当にジレンマなんです（シモ氏）」

またバグの出方は、Webアプリとネイティブアプリによっても違うとグンタ氏は説明します。

「Androidの場合、ネイティブアプリであればOSに起因するバグはそこまで出ません。しかしWebアプリでは、メーカーごとにWebブラウザをカスタマイズしています。それはメーカーごと、あるいは端末ごとの独自色を出したいということだと思います。ただ、そのカスタマイズによって多くのバグが混入してしまっているんです。このため、Webアプリの検証では、多くのWebブラウザのバグに直面することになります（グンタ氏）」

シモ氏は、あるAndroid端末における複雑な状況を説明します。

「たとえば世界中で人気のあるAndroid端末は、実はWebブラウザが3つあります。デフォルトブラウザ、Google Chrome、そしてアプリからWebブラウザを利用するWebViewと呼ばれるしくみです。これらが各端末で共通ならいいのですが、実は違うんです。ある端末はGoogle Chromeのソースコードをフォークして手を入れ、それをデフォルトブラウザのアイコンで表示しています。なので、デフォルトブラウザでもAndroid端末によって実体が違うという状況が起きているわけです（シモ氏）」

製品ごとに異なる操作方法を覚えることも時間の無駄

さて、Androidのようなデバイスでデバッグを行うとき、エミュレータを使うという手がまず考えられます。しかし、検証用途で使うのは厳しいようです。

写真2 キンヌネン シモ氏

「まず大きな問題として、ものすごく遅いんですね。我慢できれば何とかできるレベルではありますが……。そしてそれ以上に問題なのは、メーカーが手を入れているところはエミュレータで再現できないことです。つまりエミュレータで動作しても、実機では動かないという現象に遭遇することになります（グンタ氏）」

そうすると、やはり物理的なデバイスを手元に用意して検証せざるを得ません。グンタ氏が先に話したとおり、数多くの端末を検証するには携帯通信事業者のオフィスで検証作業を行いますが、慣れない環境で検証を行うのはストレスが大きいでしょう。さらに、端末ごとに異なる操作方法に慣れることもたいへんだとシモ氏は話します。

「たとえば、物理的なボタンは製品によって異なりますよね。慣れた端末であれば問題ありませんが、初めて使う端末はまず操作方法から理解しなければなりません。それが本当に時間の無駄だと感じます（シモ氏）」

こうした検証の手間は、当然開発コストにも大きな影響を与えることになります。しかしエンジニアとしては、あきらめるという選択肢はありません。そこで2人が取り組んだのが、PCに接続したスマートフォンをネットワーク経由でリモートコントロールするしくみです。次回はその内容について詳しく解説していきます。SD

特別企画

ソーシャルゲームのDevOpsを支える技術 前編

～魔法少女リリカルなのはINNOCENTの舞台裏～

モバゲーから提供されているソーシャルゲーム『魔法少女リリカルなのはINNOCENT』は、企画・開発・運用を株式会社ユビキタスエンターテインメントが行っています。このサービスを支えるインフラは物理サーバと仮想サーバのハイブリッド構成となっており、株式会社IDCフロンティアのクラウドサービスが採用されています。本稿はこの両社から、人気ソーシャルゲームならではのデータを交えながら、DevOps実現に向けての取り組みを披露してもらいます。

㈱ユビキタスエンターテインメント　取締役　CTO　水野 拓宏(みずの たくひろ)
㈱ユビキタスエンターテインメント　R&Dディビジョン 第2セクション　宮嶋 史尋(みやじま ふみひろ)
㈱IDCフロンティア　カスタマーサービス本部 ソリューションアーキテクト　藤城 拓哉(ふじしろ たくや)

どんなゲームなの?

『魔法少女リリカルなのはINNOCENT』は2013年3月にモバゲーにてリリースされたソーシャルゲーム[注1]です。原作者の都築真紀氏がこのゲームのために構築した世界観、従来のカード型ソーシャルゲームのシステムを踏まえつつも他作品に類を見ないリアルなバトルシステムと豊富なスキル(100種類超)、そして作品内の進行度によって開放される完全書きおろしストーリーが大変好評をいただいており、毎月10万人以上のユーザがプレイする作品となりました。

この作品の企画的に大きな特徴の1つは「ゲーム内のキャラクター達も、プレイヤーとまったく同じカードゲーム『ブレイブデュエル』をプレイしている」ということです。このため「プレイヤーが参加している期間限定ゲーム内イベントに、ゲーム内のキャラクター達も参加して一緒に遊びながら、時には感想を聞いたり、時には共に成長したりできる」という臨場感を持ったおもしろさがあります。

ユビキタスエンターテインメント(以下、UEI)はセブン・アークス様と共に、この作品の企画・開発・運用すべてを担当しており、筆者、水野[注2]は企画ディレクションと開発・運用アーキテクチャの設計を行いました。

本稿では前後編に分け、本作品がDevOpsの考え方に基づいてシステム設計されていることとその概要、運用担当の宮嶋からはインフラ設計と実際の運用に際して収集データを基にどういったことを裏側で行っているのか、さらに次号の後編では、そのインフラを支えているIDCフロンティアのクラウドサービスにまで踏み込

注1) スマートフォン、フィーチャーフォン用。
URL http://sp.pf.mbga.jp/12010803/

注2) URL https://facebook.com/takuhiro.mizuno
URL https://twitter.com/mizunon

んで、ソーシャルゲームの裏側のお話をしたいと思います。

ソーシャルゲームは眠らない

最近あらためて「DevOps」という言葉をよく見聞きします。DevOpsとは2009年にベルギーで始まった「DevOps Days」からポピュラーになってきた言葉で、筆者自身は2011年のサンタクララで行われたWebパフォーマンスと運用に関するコンベンション「O'reilly Velocity Conference」で初めて耳にしました。

それは日本語で直訳してしまうと「運用開発」ではありますが、それでは言い尽くせないほどに深い概念です。

WikipediaのDevOpsの項目は最近のものでは変わってしまいましたが、初期の記述に「DevOpsは、開発部門(アプリケーションエンジニア／ソフトウェアエンジニア)と技術運用部門の、コミュニケーション・連携・調律を行うための、手法とシステムが一連となったプロセスである。」[注3]「その目的は、良い品質のソフトウェアプロダクトやサービスを的確に素早く提供することによって、ビジネスゴールへと到達することを容易にするためにある。」[注4]と書かれていました。

このように、DevOpsは「サービスのリリースサイクルを速めようぜ！」とか「開発と運用が仲良くしようぜ！」という単純なものではなく、全体をとらえて「ビジネスゴールへ到達することを容易にしよう！」という目的を見据える概念であるのが深いところです。

筆者は、いままでにオンラインゲームやWebサイト、スマホアプリの開発に100以上かかわってきました。その経験から、本作品のようなソーシャルゲームの運用開発における、その目的と典型的な4つの特徴は次のようであると考えています。

【目的】

プレイヤーに、常に新鮮なイベントやシステムと、成長素材を提供し、安定してゲームを遊んでもらうこと。

【4つの特徴】

①1ヵ月に数十回の小規模開発＝「カード追加」「UI改善」「小機能追加」のリリースが発生すること
②1ヵ月に数回の中規模開発＝「ゲームイベント」のリリースが発生すること
③数ヵ月に一度の大規模開発＝「新規ゲームイベント」のリリースが発生すること
④技術者の調査が必要なユーザサポート問い合わせが日に数件～数十件必ず発生すること

特徴の4つを見てわかることは、とても複雑かつ密接に開発と運用が結びついているということです。そのため、長期にわたってトラブルなく運用することは非常に難しく、実際のところ、筆者もそれは完全にはできていません。そこでプレイヤーの皆さんにおなじみ(？)の「補填」や「詫び石」が発生する事態になったりする場合があります。

しかし、トラブルを完全になくすことはできなくても、少なくすることにエンジニアリングで挑戦するのが技術者のはずです。ソーシャルゲームにおいても開発と運用は密接な関係にあり、どちらかに寄ってしまったり完全に切り離すことはできないことがわかりました。それならば、戦場の狙撃兵が観測手と組み、一体となってその目的を達成するように、本作品の目的のためには開発と運用を統合して考えなければなりません。そのための技術としてDevOpsを必要としたのです。

注3) 原文：DevOps is a set of processes, methods and systems for communication, collaboration and integration between departments for Development (Applications/Software Engineering) and Technology Operations.

注4) 原文：Its purpose is to facilitate meeting business goals by producing good quality software products and services in a timely fashion.

さて、ここからは弊社内「Ops」側担当の宮嶋から、インフラ整備と運用に関するエンジニアリングを事例を交えて紹介します。

[運営サイド] インフラ構成について

フィーチャーフォンやスマートフォンのブラウザでプレイされるソーシャルゲームは、ゲーム専用機のパッケージで提供されるゲームとは異なり、ゲームに必要なデータをほぼすべてサーバ側で保持しています。そのためサーバに要求される性能や機能の水準も高くなります。また同時利用者数が多く、イベントなどが実施されると普段の数倍以上の利用者数になることもあり、サーバ負荷が変動しやすい性質をもっています。そのためシンプルな構成を保ちながら運用しやすく、負荷に応じて柔軟に性能を変化させやすいインフラ構成を検討する必要があります。

LAMP構成中心の設計

少人数での運営作業は、運用コストを見直し、低下させる方針が重要となります。自分たちが使い慣れた、動作実績があり設定が把握しやすいミドルウェアを利用すると、運用コストの削減につながります。

また、ソーシャルゲームのプラットフォーム側で提供されている機能(画像キャッシュ機能)など、利用可能なものは自前で構築しないなどの割り切りも行います。我々が本作品で利用したミドルウェアと利用方針は次のようになります。

■ロードバランサとしてのNginx

ロードバランサとして、NginxをWebサーバの上段ネットワークに配置します。ここで参照頻度の高い画像配信と、リバースプロキシサーバとして機能させ、Webサーバにトラフィックの分散を行います。

■アプリケーションサーバとしてのApache

Webサーバ兼アプリケーションサーバとして、Apache HTTP Serverを利用します。PHPで記述されたゲームプログラムを実行し、Flashのリアルタイム合成や、カード画像などの一部画像ファイル名のハッシュ化も行います。

ファイル名のハッシュ化を行うとファイル名が推測しにくくなり、不正利用の防止につながります。

■データストアとしてのMySQLとMemcached

ゲームユーザごとのプレイデータとゲームプログラムで利用するゲームマスタデータはデータベースサーバのMySQLで保持しています。Memcachedは利用頻度の高いゲームマスタデータのキャッシュに利用し、MySQLへの接続頻度の低下を図っています。

MySQLは最新安定版であるMySQL 5.6を採用しています。MySQL 5.6は高速であるだけでなく、mysqlfailoverコマンドを利用したフェイルオーバー機能も備えています。しかし、このフェイルオーバー機能は利用できるストレージエンジンが限られるなど制限もあり、運用面では課題が残る部分もあります。

MySQL 5.6での特筆すべき機能として、遅延レプリケーションがあります。遅延レプリケーションでは、通常のレプリケーションとは異なり、SQLの実行を事前に設定した時間の間で遅延させることができます。

たとえば意図しないデータ削除などのオペレーションミスが生じた場合、「削除処理がまだ実行されていない状態のデータ」を容易に入手できるため、バックアップからの復元といった負担の大きな作業を行わなくてもよいメリットとなります。

■トラフィック削減のためデータ通信の圧縮

サーバのCPUコア数やメモリはクラウド環境を利用することで柔軟に変化させることがで

きますが、ネットワークインターフェースの通信速度はサーバのスペックに依存する部分を除いては柔軟な変更を行いにくい部分となります。このように変更を行いにくい部分はボトルネックになりがちです。

そのためデータ通信の圧縮を行うと、トラフィックの削減を図ることができ、ゲームのレスポンス向上にもつながります。

Apacheではmod_deflateモジュールを利用してコンテンツの圧縮を行い、PHPでMySQLとMemcachedの接続時に圧縮する設定や、MySQLレプリケーション通信間の圧縮設定を行うなど、さまざまな個所で圧縮通信設定を行っています。

クラウド環境を選ぶ理由

使い慣れたミドルウェアを利用するため、『魔法少女リリカルなのはINNOCENT』ではIaaSクラウド環境を利用しています。IaaSクラウド環境は、クラウド事業者によって利用できる機能が異なります。サーバスペックに現れない部分は実際に利用しなければわかりません。できるだけさまざまなクラウド環境を試して、選択肢を多くすることをお勧めします。

向きと不向き

IaaSクラウド環境では仮想サーバが利用できます。仮想サーバはCPU性能やメモリ容量によって料金が異なりますが、仮想サーバは物理ホストサーバの上で動作する以上、サーバの性能を専有できず、さまざまな影響によって性能にバラつきが生じます。我々の場合、公開されているベンチマークデータを鵜呑みにせず、自分でSysBenchやfio、mysqlslapなどのベンチマークツールを用いて確認したあとは、厳密な性能の測定にはこだわらず、利用想定、仮想サーバの負荷状況に応じてサーバスペックや台数を柔軟に変化させる方針を取っています。

仮想サーバは簡単に調達でき、自分でミドルウェアをセットアップしたあとのサーバイメージをコピーすることも容易に行えます。

Webサーバはアクセス状況によって負荷が変動しますので、仮想サーバを利用した構成を取ると、急なアクセス増にも楽に対応が行えます。

DBサーバはI/Oアクセスが多いため、安定したI/O性能がないとサーバ性能への影響が大きくなります。I/O性能は仮想サーバでは安定して確保しにくいため、クラウド事業者によっては通常のストレージとは別に、高速なSSDや安定したIOPS（Input/Output Per Second）を供給するストレージを提供するところもあります。

しかしどうしても安定した性能を確保したい場合は、物理サーバの利用も選択肢になります。物理サーバは高速なI/O性能を誇るioDriveを搭載したものや、大容量のメモリが選べるものがあり、それらの性能が専有できるため、仮想サーバと比較して性能面でのバラつきが小さくなります。反面、仮想サーバのように調達が容易に行えず、納期に時間がかかったり、故障の影響を受けやすいなど柔軟な運用が行いにくいデメリットもあります。

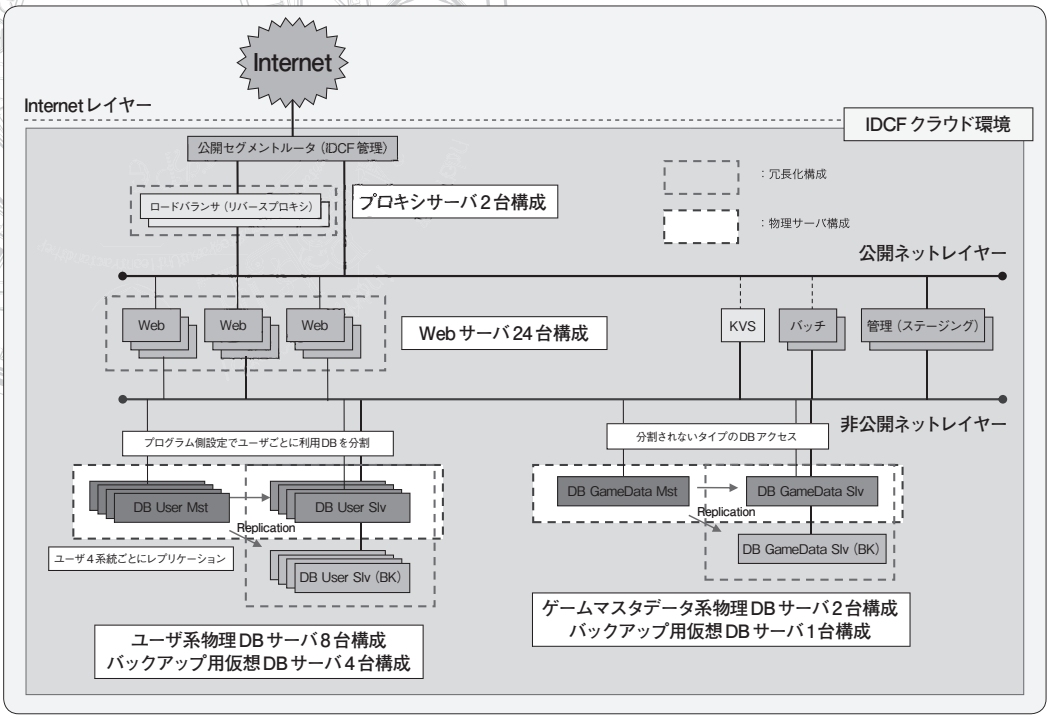
本作品では事前登録会員数が10万人を超え、ユーザ活性が非常に高いことが想定されたため、すべて仮想サーバを利用した構成とせず、DBサーバは物理サーバを利用する構成をとりました。初期構成が**図1**となります。

仮想サーバと物理サーバはL3接続されているため、ルーティング設定を行えば仮想サーバと物理サーバの区別なく相互接続が可能となります。

ソーシャルゲームでのクラウド利用

小さい構成からはじめて、利用状況に応じて増強する「スモールスタート」構成はクラウド環境が向いていると言われます。ソーシャルゲームもリリース後の状況が読みにくい部分はありますので、リリース初期はクラウド環境を利用

▼図1 『魔法少女リリカルなのはINNOCENT』サーバシステム初期構成



し、その後ユーザ数の増加に伴い安定した性能を確保するためにオンプレミス環境へという方法を選択されている他社SAPの方も少なくありません。

もちろんオールクラウド環境だと不安定で利用に耐えないということはなく、そのソーシャルゲームをどのように成長させていくかという想定のもと、稼働環境を検討することが重要です。

本作品もそれにならい、仮想サーバと物理サーバのハイブリッドクラウドから開始し、最終的にはオンプレミス環境を利用するという成長過程の想定をしています。

クラウド事業者の違い

IaaSクラウド環境を提供する事業者はAWSをはじめ、国内にも多数あります。仮想サーバ機能はどこでも利用できますが、機能面でベンチマークとなるのはAWSです。AWSで当たり前に利用できる機能が、他社では搭載されていないことはよくあります。そのため、AWS以外のクラウド事業者を利用する場合は、ある程度機能面での不足の割り切りを行ったうえで、強みや独自性が生かせる事業者を選択しています。

本作品では、サーバとネットワーク部分はIDCフロンティアクラウドを利用し、DNSやバックアップストレージにAWSのRoute53とS3を利用しています。

IDCフロンティアクラウドは仮想サーバの費用対性能が高く、iDC（Internet Data Center）も運用し、クラウド環境と物理サーバのハイブリッド利用オプションがあるためソーシャルゲームの成長過程の想定に合致していました。

AWSは基本的にインターネット越しにはなりますが、AWS外からもサービスの利用が可能なため、クラウド事業者間でハイブリッド利用することが可能です。

IaaSクラウドはベンダーロックインのような縛りがほとんどないため、事業者ごとに有利

な部分を選択的に使うと運用面でも費用面でもコストを下げることができます。

イベント中の運営で何が起きているのか？

『魔法少女リリカルなのはINNOCENT』では現状、月に1回程度イベントを実施しています。イベントは約1週間程度実施され、その期間中の活動ポイントによって報酬が異なります。イベント期間中は普段の数倍のトラフィックが生じることも珍しくなく、イベントの参加者だけでなくイベント運営側にとっても、大変忙しい期間となります。

イベント準備期間では

本作品では原作者の方が監修し、企画後に毎回イベントを新規に実装しています。そのため、使い回しのイベントがほぼありません。企画ごとに新規実装しているため、企画内容によってサーバにかかる負荷パターンが異なります。

たとえば、Flashアニメーションを多用したイベントの場合は、Flashのリアルタイム合成を行っているWebサーバに負荷がかかり、ユーザ間対戦型イベントの場合はユーザのプレイデータを保持するDBサーバに負荷がかかるなど、増強するサーバタイプが異なる性質を持ちます。

どのサーバをどのようなタイミングで増強してゆくのかは悩ましいところではありますが、DBサーバに物理サーバを選択しているため、場合によっては調達に時間がかかる恐れがあります。増強に遅れを生じさせないためには、できるだけ事前の負荷想定が重要になってきます。

イベント時の負荷推定を行うためには、日頃からサーバとネットワークの負荷状況を把握しておく必要があります。リソースの利用状況の把握にはZabbixを利用し、可視化しています。

イベントの開始

イベントの開始から終了までのロードバランシングサーバのトラフィックの一例が**図2**のグラフとなります。イベント直前の50Mbps前後のトラフィックから、7月20日18時から19時のイベント開始直後に、一気に200Mbps超のトラフィックに急増しています。

このとき運営側では、事前想定と負荷状況がどの程度違うかを調査しています。調査ではFluentdを利用したサーバのログ収集と、MySQLのクエリ状況を中心に確認しています。

ソーシャルゲームではWebアプリケーションのレスポンスタイムが重要な指標となります。Apacheのログフォーマットを変更し、「%D」を付与するとマイクロ秒単位でレスポンスタイムの記録が可能です。Combinedフォーマットを利用している場合の変更例は次のようになります。

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

レスポンスタイムに5秒以上要しているレスポンスが一定数以上あると、ソーシャルゲームのプラットフォーム側から強制的にメンテナンス状態に移行される恐れがあります。この状況を避けるために1秒以上レスポンスに時間を要しているものは調査対象に分類します。

MySQLでは処理に時間がかかったクエリを記録することができ、そのクエリをスロークエリログに記録することができます。スロークエリの記録しきい値はMySQL 5.1以降であれば、動的に変更することができます。通常は0.5秒程度をしきい値に設定していますが、より詳細に調査したい場合は次のようにしきい値をゼロ秒に設定し、一時的にすべてのクエリ状況を記録することも行います。

```
mysql>set global long_query_time=0;
```

すべてのクエリを記録すると負荷が大きいた

め、確認後は必ず元のしきい値に戻しておくことが重要です。

見つかった問題点は開発側にフィードバックし修正を行います。残念ながらイベント開始後に発覚した問題点には、改善に時間を要するものもあります。その場合は、ゲームの一部機能停止や、影響が大きい場合は緊急メンテナンスを行い、イベントの一時休止を行うこともあります。

イベントの中盤

イベントの中盤ではリソース監視を継続し、ユーザのイベント参加率を考慮しながら、サーバの増強効果を確認します。この時点ではトラフィックが急増することもほとんどないため、イベント準備期間と開始直後に問題点の改善が済んでいれば比較的落ち着いた状態が保てます。

とはいえ、いつまでも落ち着いた状態のままではいられないため、イベント終盤に向けての計画を確認し、すでに指摘済みの改善点の漏れがないか、イベント開始直後に見つかった一時的にレスポンスが遅いリクエストの再確認を行います。

レスポンスが遅い原因がデータベースへのクエリにある疑いが生じた場合、処理時間が短いが、発行回数が多いクエリを疑います。単純にスロークエリの記録しきい値を変更しても、発行回数が多く、問題があるタイプのクエリの発見は難しいため、MySQL付属のmysqldumpslowコマンドを利用します。

```
mysqldumpslow -s t mysql-slow.log
```

上記のコマンドを調査したいスロークエリログに対して実行すると、クエリ処理時間と実行回数を計算した合計処理時間が、上位のものからソートした結果として得られます。このコマンドの結果を分析し、問題が見つかった場合は開発側にフィードバックを行います。

イベント中盤ではユーザの振る舞いも安定しているため、この時点で発見された問題を無理に改善しようとすると、結果として必ずしも最適にならない恐れがあります。そのため修正するかどうかはイベント終盤時に問題になりうるかどうかという視点で協議し、判断を行います。

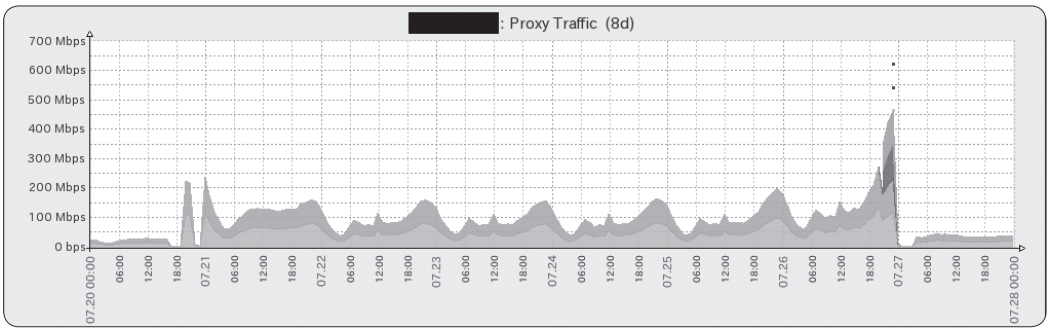
イベントの終盤

イベントの終盤が迫り、とくに最終日になるとイベント参加者も追い込みに入ります。

図2の7月26日18時ごろから見られるように、イベント終了間際になると200Mbps超のトラフィックがさらに急増し、600Mbps超にもなります。

トラフィック急増の原因は、このイベントの場合はユーザによるWebサーバ上でのFlashリアルタイム合成要求数の増加なのですが、合成処理によるCPU利用率の上昇がレスポンスタ

▼図2　イベント開始から終了までのロードバランシングサーバのトラフィック例



イムに大きな影響を与えないように、Webサーバのロードアベレージを確認します。

図3のグラフがイベント中のWebサーバのロードアベレージになります。イベント終了の1～2時間前ぐらいからトラフィックが急増する傾向があるため、サーバ起動時間を考慮して事前プロビジョニングを行います。**図4**のグラフでは16時と20時にサーバを追加し、最終的にはWebサーバの数は60台になりました。

イベント期間中は定期的にイベントランキングの公開を行っていますが、イベント終了間際では公開を停止しています。これは参加者による駆け込みアクセス増の防止を目的としています。

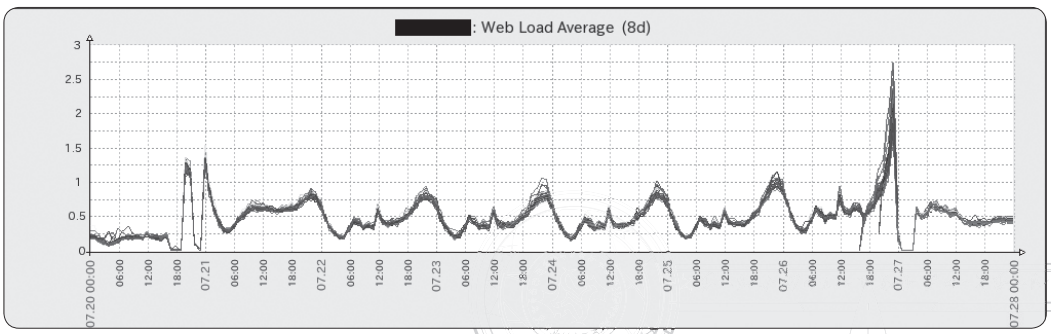
“駆け込みアクセス増”とは、イベント終盤になると、ランキング報酬が獲得できる順位争いが加熱化し、ランキングを取得する目的で集計間際に大量のポイントを獲得されることがあります。フィーチャーフォンやスマートフォンのブラウザでの利用を想定したゲームではありますが、PCなどからツール等を利用して参加する一部ユーザが存在します。ツールを利用するとフィーチャーフォンやスマートフォンのブラウザでは実行不可能な速度でゲームへの参加が可能になるため、対策として駆け込みアクセスなどの連続アクセスにはしきい値を強化するなど監視を強化し、ユーザ間で公平なリソース利用となるように注意しています。

終了間際のリアルタイム最適化

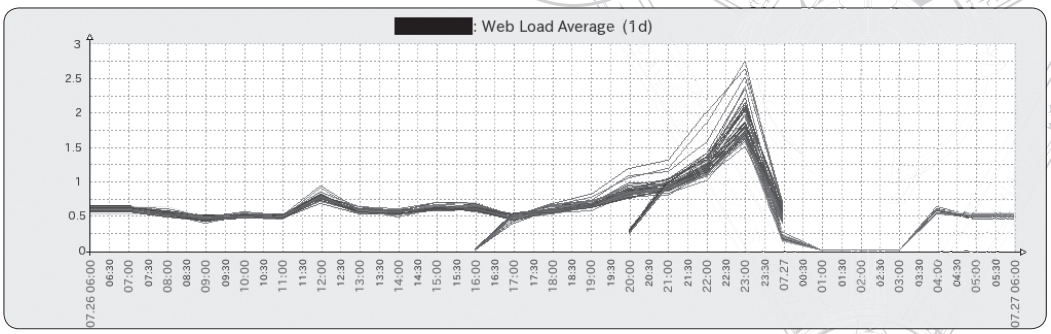
イベントの終了が近くなると、これまで行った最適化の効果を検証し、イベントを無事に終了させるためにリソース監視だけでなくDBサーバで実行中のプロセスを確認し、問題の早期発見に努めます。

MySQLで実行中のプロセスを確認するには次のコマンドを実行します。

▼図3　イベント中のWebサーバのロードアベレージ例



▼図4　図3のイベント終了間際を拡大



```
mysql>show full processlist;
```

イベントの終盤ではユーザの行動ログが蓄積され、データが肥大してきます。データが肥大してくると、これまでに行ったスロークエリログを利用したクエリの改善が有効でなくなるケースが存在します。

図5のグラフはユーザのプレイデータが保持されているDBサーバ上で、Select Scanされた数を記録したものです(これまでのサンプルとは別のイベントのもの)。Select Scanはテーブルやインデックスの先頭行から全件検索を行った回数を意味します。

SELECT文を用いたクエリのチューニングではインデックスを使用して探索する行数の範囲を少なくするのがセオリーですが、全件探索を行う場合もあるため、一定数のSelect Scanは存在します。しかし、Select Scanが意図せず急増している場合はMySQLが利用するインデックスが変化した可能性があるため、調査を行います。

MySQLのオプティマイザはデータ量と利用頻度によって利用するインデックスを変化させるため、事前のクエリ実行計画結果と利用されるインデックスが違ってしまう場合があります。その場合は`force index(インデックス名)`でインデックスのヒントをオプティマイザに与え、利用するインデックスを変更するように発行するSQL文を変更します。

場合によっては、クエリの実行速度を再優先するためにインデックス追加や削除などをイベント中に最適化することもあります。MySQL 5.6ではインデックスの追加と削除がオンラインで可能なため、イベント終盤でゲームの中断を行わずに済みます。

イベント後

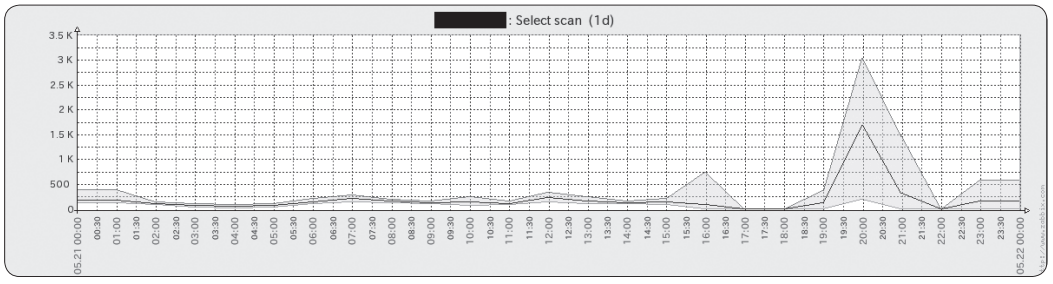
無事にイベントが終了できると、運営側もやっと一息がつけます。イベント終了時刻を超えるとトラフィックが急減するため、イベント中に増強したサーバを停止し、課金を抑えるといったイベント終了作業を行います。

また、想定よりも悪かった部分や良かった部分を次回のイベントに生かせるように抜き出し、調査項目にあげるなどの振り返りを行います。振り返りは地道なことですが、地道なことを継続し、経験を蓄積することが、企画・開発・運用が協力した少人数での運営には重要になります。

次回予告

今回はDevOpsの考え方を実践するソーシャルゲーム『リリカルなのはINNOCENT』のインフラ構築の方針、データ分析による継続した改善について我々の取り組みを紹介しました。後編では、DevOpsのおもに「Dev」に重きを置いたソーシャルゲーム開発の裏側と、クラウド事業者のIDCフロンティアから見たソーシャルゲームトラフィックのさばき方を紹介したいと思います。SD

▼図5　DBサーバ上でSelect Scanされた数を記録したもの



Hardware

ぷらっとホーム、
PoEや3Gに対応したM2Mゲートウェイに最適なマイクロサーバ「OpenBlocks A7」を発売

ぷらっとホーム㈱は、企業向けマイクロサーバ「OpenBlocks Aファミリ」の最新モデルとなる「OpenBlocks A7」とそのオプションモジュールを9月12日に発表、同日より販売を開始した。

OpenBlocks（以降、OBS）シリーズは2000年にスタートし、以来培ってきた技術と経験、そしてユーザからの要望をフィードバックすることで進化してきた。OBS A7は、ARM CPUを搭載したAファミリの上位モデルであるOBS AX3と、エントリーモデルのOBS A6の間に位置する製品。ユーザからの要望が多かった「4万円前半の価格帯でGbEポートを2基搭載」という要件を満たし、本体サイズも一世代前のOBS 600と同じにすることでリプレースの需要にも応えやすくなっている。

▲OpenBlocks A7

同時に発売された「3G通信モジュール」（W-CDMA/GSM/GPRS対応）は、ネットワーク回線の敷設が難しい場所での用途に対して、「RS-485モジュール」は、複数のセンサーデバイス管理への対応として、いずれも昨今求められることの多い用途向きへの拡張を可能にする。

▼主なハードウェア仕様

品名	OpenBlocks A7
CPU	ARMADA 310（88F6283）600MHz
メインメモリ	1GB（DDR3 SDRAM）
Flash ROM	256MB（NAND）
ストレージ	SATAII（2.5inch SSD（別途接続ケーブルが必要）、HalfSlim SSD を搭載可）
内部インターフェース	USB2.0×2、RS-232C×2
外部インターフェース	10/100/1000BASE-T×2（PoE 受電可） USB 2.0（Type-A）×2 RS-232C（RJ-45）×2（1port はコンソールと排他）
筐体サイズ	81（W）×133.5（D）×32（H）（ゴム足含まず）
重量	約 230g
動作条件	温度：0 − 55℃（PoE 受電時は 0 − 50℃）、湿度 20 − 80% Rh（結露なきこと）AC アダプタ含む
電源	AC アダプタ 5V または PoE 受電 DC48V
OS	Debian GNU/Linux

CONTACT ぷらっとホーム㈱
URL http://www.plathome.co.jp

Topic

LPI-Japan、
「HTML5認定試験」の開発に着手、2014年初春に配信開始予定

OSS/Linux技術者認定機関として「LPIC」および「OSS-DB技術者認定制度」を実施する特定非営利活動法人エルピーアイジャパンは、「HTML5認定試験」の開発に着手したことを発表した。

本試験はNTTソフトウェア㈱、HTML5開発者コミュニティ「html5j」、LPI-Japan理事企業の協力のもと、オープンコミュニティの枠組みにて開発しており、2014年1月の配信開始を目標にしている。現時点での試験概要は次のとおり。

■試験概要案

本認定プログラムはHTML5に関する基本知識や静的なコンテンツを制作できるスキルを認定する「Silver『Markup Professional』」と、応用知識や動的なアプリケーションを制作できるスキルを認定する「Gold『Application Development Professional』」の2つのレベルの試験から構成されます。

■試験名案（日本語名）

- HTML5認定試験Silver（Markup Professional）
- HTML5認定試験Gold（Application Development Professional）

■受験料案

Silver：15,750円（税込） Gold：15,750円（税込）

■受験方法

CBT（コンピュータベーストテスティング）

■出題範囲（案）

- Silver（Markup Professional）
  Webの基礎知識／CSS3／要素／レスポンシブWebデザイン／オフラインWebアプリケーション（概要とマニフェスト）
- Gold（Application Development Professional）
  JavaScript、Selectors API（Level1、Level2）、マルチブラウザ対応／Canvas、SVG、マルチメディアコンテンツ／オフラインアプリケーションAPI／Session History and Navigation／表示／ストレージ／通信／Geolocation API／Web Workers／パフォーマンス

CONTACT 特定非営利活動法人エルピーアイジャパン
URL http://www.lpi.or.jp

Topic

LPI-Japan、『Linuxセキュリティ標準教科書』を公開、無償での配布を開始

特定非営利活動法人エルピーアイジャパンは10月1日、『Linuxセキュリティ標準教科書』を公開した。

PDF版、EPUB版、Kindle版、書籍版での提供となり、PDF版とEPUB版は右記URLより無料でダウンロード可能。同教材はLinuxにおけるセキュリティを学習／再認識するために最低限必要となる知識を体系的にまとめた内容となっており、「LPICレベル3 303試験」の教育および学習にも役立つという。

■Linuxセキュリティ教科書概要

- タイトル：『Linuxセキュリティ標準教科書』
- 著者：伊本貴士、面和毅、藤森健、小石川雅紀、安田恭行、嶋中賢佑
- 配信開始日：2013年10月1日（火）より
- 提供価格：無料
- データ形式：PDF（180ページ）、EPUB（6.5MB）
- 言語：日本語
- 公開URL：http://www.lpi.or.jp/linuxtext/security.shtml

CONTACT 特定非営利活動法人エルピーアイジャパン
URL http://www.lpi.or.jp

Event

The Linux Foundation、「Enterprise User's Meeting 2013」を12月11日に開催

The Linux Foundationは、国内外のLinux/OSSのユーザ企業向け技術カンファレンス「Enterprise User's Meeting 2013」を12月11日に、横浜赤レンガ倉庫にて開催することを発表した。

同イベントは、Linux活用企業とエンドユーザ、OSSコミュニティ間の協業と交流を促進する活動の一環として2011年より定期開催しており、今回で3回目。

今回は初めて発表者を日本で一般公募（Call for Participation：CFP）する。発表案の受付は10月13日まで特設サイトにおいて行う。CFP発表案およびセッションでの講演はすべて日本語で行うほか、すべての講演はUstreamでライブ配信する予定。

▼開催概要

開催日	2013年12月11日
会場	横浜赤レンガ倉庫
Webサイト	http://events.linuxfoundation.jp/events/enterprise-users-meeting-japan
プログラム発表	11月1日（金）
参加登録	11月1日（金）開始予定
参加費	無料（上記Webサイトにて参加登録が必要）

CONTACT The Linux Foundation
URL www.linuxfoundation.jp

Software

エンバカデロ、iOSアプリに加えてAndroidアプリ開発も可能になった「RAD Studio XE5」を販売開始

エンバカデロ・テクノロジーズは、マルチデバイスソフトウェア開発ツールの最新版「RAD Studio XE5」の販売を9月12日より開始した。RAD Studio XE5は「Delphi XE5」「C++ Builder XE5」「HTML5 Builder XE5」の3製品が含まれたツールスイート。

WindowsとMac OS X用ソフトウェアのクロス開発がDelphiおよびC++言語で可能だった本製品だが、前バージョンのXE4（2013年4月22日国内発売）でiOSアプリケーションの開発に対応し、今回のXE5ではAndroidアプリケーションの開発も行えるようになった。現状ではどちらもDelphi言語でのみ（Delphi XE5）の対応となっているが、近くC++言語での開発にも対応する予定。

RAD Studio XE5により、これまで複数のプラットフォーム向けに別々の設計・実装を行っていた負担が大幅に軽減されるうえ、独自のコンパイラにより各OSに最適化されたネイティブアプリとして生成されるため、パフォーマンスの低下もないことが特徴。トライアル版も提供されている。

CONTACT RAD Studio製品サイト
URL http://www.embarcadero.com/jp/products/rad-studio

Service

シーズホスティングサービス、Web改ざん対策（リアルタイムスキャン）オプションを提供開始

シーズホスティングサービスは10月1日より、専用サーバサービス、VPSサービスのオプションとしてWeb改ざん対策ソフトウェア「GIDEONリアルタイムスキャン」の提供を開始した。

GIDEONリアルタイムスキャンは、サーバへインストールし、ファイルの変更動作を監視することで、不正なファイル変更を検知し、自動修復するソフトウェア。これを利用することで、Web改ざん、ウィルス混入を防止し、安全なWebサイトの運用が可能となる。

フルマネージドプラン、ライトマネージドプランの場合は、同社にてインストールと基本設定を行うため、手間なくすぐに利用できる。セルフマネージドプランもコマンド1つで簡単にインストールできるとのこと。

同オプションの料金は、フルマネージドプランとライトマネージドプランの場合、初期費用15,000円、月額費用15,000円。セルフマネージドプランの場合は初期費用15,000円、月額費用7,500円（いずれも税抜価格）。2013年12月末日までに申し込みすれば、初期費用が無料となる導入キャンペーンを実施している。

CONTACT シーズホスティングサービス
URL http://www.seeds.ne.jp

Service

さくらインターネット、「さくらのクラウド」で1時間9円からの時間課金をスタート

さくらインターネット㈱は、「さくらのクラウド」において、1時間あたり9円からの時間課金を導入し、2013年10月1日より提供開始した。

同サービスは、CPUとメモリを自在に組み合わせて使えるクラウドサービス。1コア1GBから最大12コア128GBまでの豊富なサーバ選択肢と、SSDによる圧倒的なストレージ性能を提供する。

これまでは20日未満の利用の場合は日割料金での精算となり、20日の時点で月額料金が適用される料金体系だった。また、1時間の利用でも日割料金がかかっていた。だが、これからは1時間単位での時間課金となるため、最大で約90%安くなるとのこと。時間課金はおよそ10時間未満の利用の場合に適用され、それより多くの時間利用すると従来の日割金額となる。短時間利用の場合はより安く、長く使ってもこれまでと変わらない日割金額が上限となる。

時間課金は全サーバプランで利用可能で、1コア1GBプランが1時間9円、12コア128GBのマシンでも1時間あたり474円で使える。

CONTACT さくらインターネット㈱
URL http://www.sakura.ad.jp

Service

at+link、at+linkアプリプラットフォーム最大3ヵ月利用料無償キャンペーンを実施

㈱リンクと㈱エーティーワークスが共同で展開するホスティングサービス「at+link」は、スマホゲームやアプリ、高負荷Webサービス向け専用サーバパッケージ「at+linkアプリプラットフォーム」において、9月19日より最大で3ヵ月利用料が無償になるキャンペーンを実施している。

同サービスは、複数のWebサーバとioDriveを搭載したDBサーバ、大容量回線やネットワーク機器のほか、アプリの運用に必要な機能をまるごとパッケージにし、初期費用無償で提供するサービス。このキャンペーンにより、ユーザ企業は実質、最大で3ヵ月間インフラコストをかけずに自社サービスの運用が可能となる。

▼キャンペーン概要

名称	at+link アプリプラットフォーム最大 3 ヵ月利用料無償キャンペーン
実施期間	2013 年 9 月 19 日（木）～ 10 月 31 日（木）
内容	期間中に申し込みすると、最大で 3 ヵ月間利用料が無償となる（10 社限定）。※ioDrive 搭載サーバプランのみの適用
申込方法	http://www.at-link.ad.jp/appli_platform/ から問い合わせのこと

CONTACT at+link
URL http://www.at-link.ad.jp

Letters from Readers

女性読者が急増中!!

「急増中」とは少々大げさでした。みなさんが想像されるほど爆発的に増えてはいません。ですが、9月号はここ最近で、アンケートに答えてくれた女性読者が一番多かったです。なぜなのでしょう? 読者プレゼントがおしゃれなヘッドフォンだったせい? いや、むしろ書籍『プログラミング言語AWK』を希望している女性が結構多いです……ってことはsed/AWK特集だったから!?

2013年9月号について、たくさんのお便りありがとうございました!

第1特集 今からはじめるsed/AWK再入門

sedとAWKはUNIXの伝統的なツールですが、多くのスクリプト言語がある最近では、あまり使っていないという人も多いのではないでしょうか? 端末からすぐに使える、ほかのコマンドと組み合わせて使えるなど、習得すればいろんな場面で役に立ちます。sed/AWKの基本的な使い方から、「こんなことまでできるの?」という技まで幅広く紹介しました。

入門として非常にわかりやすかったです。見逃しているのかもしれませんが、どういった処理は苦手なのかという情報もあると良かったです。

岡山県／佐藤さん

sed/AWKの復習を兼ねて本誌を購入。結果、新しい情報はほとんどなかったが、AVD(AWK Visual Debugger)には興味を持った。

千葉県／山下さん

sed/AWKは使えたら便利そうだと思っているが、なかなか習得しないので、これを機に使えるようになろうと思う。

神奈川県／くまーさん

sed/AWK特集、良かったです。私は学生で、大学でsedとAWKを使っていますが、年上の先生からは「いまだに使われているの!?」としばしば驚かれます。

東京都／遊佐さん

sed/AWKは日々利用しているので、再確認できて、とても良かったです。

東京都／psiさん

AWKの奥深さには恐れ入る。とくにネットワーク機能の使い方はおもしろかった。

東京都／hiddenさん

「今さらsed/AWK?」という批判もあるかも……と、特集を組んでみたところ、意外と読者の反応は良かったです。UNIX、Linuxに標準で入っているツールだけあって、みなさんにとって身近なものだったのでしょう。

第2特集 開発するならやっぱりMacですよね?

今やIT勉強会などに行くと、参加者が持参しているPCの大半がMacです。しかし、同じMacでも、最前線で活躍しているエンジニアはどんな使い方をしているのでしょう? そこで9人のエンジニアの方々に自分の開発環境(デスクトップ)を紹介してもらいました。

Macの解説本は多いが、開発をメインに特集してくれたことはうれしい。さっそくいくつか試そうと思う。

東京都／藤田さん

Macの環境は持っていないが、いずれ導入したいと思っていた。そんな中、他人がどのような環境で開発をしているかを載せてもらって、自分もこのような環境を持ちたいと思った。

愛知県／長屋さん

Windowsは会社でしか使わないため、Windows環境とMac環境を両立させるニーズはとくに感じませんでした。

大阪府／しろたんさん

意外だったのは、執筆者の方々の多くが、デスクトップの壁紙をデフォルトの銀河画像のままにしていたこと。一流エンジニアは使いやすさ重視で、見た目にはあまりこだわらないってことなのでしょうか。

短期連載 小規模プロジェクト現場から学ぶJenkins活用

短期連載の第3回目。プラグインの探し方や導入方法、便利なプラグインのいくつかを紹介しました。

ちょうど自分の置かれている環境と似通っているので参考になった、バックナ

ンバーもチェックしたい。

長野県／清水さん

最近、ようやくJenkinsをインストールした。この連載に「小規模でも導入していいんだ」と後押しされた形。

東京都／binaさん

多数のプラグインがあっても、あまりに多いと、何から使っていいか迷うもの。著者のプロジェクトで実際に導入されているプラグインが紹介されましたが、参考になったでしょうか?

連載

「はんだづけカフェなう」のArduino、mbed、Raspberry Piの比較がわかりやすくて良かったです。比較してみるとRaspberry Piの安さはすごいなぁと思いました。

神奈川県／drepoxyさん

各製品の開発環境、性能などとともに著者独自の選択基準なども挙げられていたのが良かったですね。製品を選ぶ際には参考になりそうです。

「偏愛キーボード図鑑」を興味深く読んでいます。キーボードだけではなく、「キーボードの○○キーを押しながらマウスの××キーをクリック」をマウスだけでできるような多機能マウスも紹介してほしいです。

大阪府／澤下さん

主要なキーボードを一通り紹介し終わったら、ほかの入力デバイスを取り上げるのもいいかもしれませんね。ご意見ありがとうございました。

フリートーク

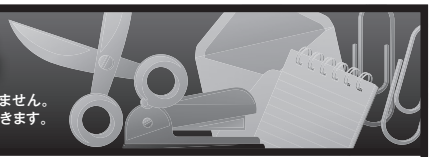
Jenkinsもプラグインで拡張できるのが魅力なのですが、最近使っているいろんなソフトウェアもプラグイン対応です。もし今の環境が壊れたら再現できるか不安になります。Firefoxしかり、Eclipseしかり、Internet Explorerしかり。何かいい管理方法はないものでしょうか。

奈良県／机の上も片付かないさん

ソフトウェアによっては環境を丸ごとバックアップ／リストアできるプラグインがあるようですよ。ただ、ツールごとに管理しないといけないですし、確かにやっかいな問題ですね。

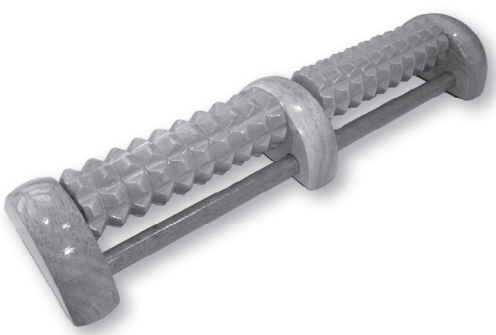
エンジニアの能率を高める一品

仕事の能率を高める道具は、ソフトウェアやスマートフォンのようなデジタルなものだけではありません。
このコーナーではエンジニアの能率アップ心をくすぐるアナログな文具、雑貨、小物を紹介していきます。



ツボラボ 足マッサージャー

893円（税込）／㈱リーブコーポレーション　http://lieb.co.jp



以前このコーナーで腰や脚を揉んでくれるマッサージクッションを紹介しましたが、今回は足の裏のマッサージ器具です。「ツボラボ 足マッサージャー」は椅子に座った状態で、同器具を足裏で踏んだり、ゴロゴロ転がしたりしながら使います（写真）。音もならないので、デスクの下に忍ばせておけば、仕事の合間に使ってリフレッシュできます。竹踏みの竹よりも細いので、足の指のつけ根もほぐせますし、踏む部分には凹凸が付いているので、軽く足裏のツボも刺激できます。かかとをゴリゴリ刺激するのが意外と気持ち良くてお勧めです。エンジニアは普段、座りっぱなしですので、たまに長時間歩いたり、立ちっぱなしだったりすると、すぐに足裏が痛くなると思います。そんなときにこそ足マッサージャーはよく効きますよ。

（読者プレゼントあります。16ページを参照）

▲写真　使用している様子

9月号のプレゼント当選者は、次の皆さまです

①オーバーヘッドフォン KOTORI201 ..... 長野県　高橋俊雄様
②裸族の頭 USB3.0 SATA6G ..... 大阪府　濱田義勝様

★その他のプレゼントの当選者の発表は、発送をもって代えさせていただきます。ご了承ください。

次号予告

2013年12月号
定価1,280円　176ページ

11月18日発売

［第1特集］SDNの普及はいかに?

仮想ネットワークOpenFlow/SDNを使っていますか?

SDN/OpenFlowの普及が進む今、OpenFlowについてさまざまな実装を比較することで、SDNの現状を理解し、クラウドコンピューティングにおける立ち位置を明解にします。ホットな最新技術情報の紹介に加え、今現在どのようにSDN/OpenFlowを使いこなしていくべきなのか、本特集ではわかりやすく伝えます。

［第2特集］下手でも好印象で効果絶大

「エンジニアの伝わる図解術」

仕様書・顧客へのシステム説明書など、エンジニアにとって作図能力は欠かせないものです。しかし、いざ顧客にとってわかりやすい図を描こうとすると、ピントがはずれた図になってしまいます。開米瑞浩先生による特別講義で図解の達人になろう!

［一般記事］基礎の基礎だけど案外知らない

LinuxとFreeBSDのジャーナリングファイルシステムの良いところ悪いところをご存じですか?

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

休載のお知らせ

「システムで必要なことはすべてUNIXから学んだ」（第10回）、「IPv6化の道も一歩から」（第10回）は都合によりお休みいたします。

SD Staff Room

●エディタ特集を2回連続で構成しましたが、いかがでしょうか。自分好みのエディタはありましたか。ところで私が愛用しているのはWZEditorです。色分けや見出しの階層化表示など強力な編集支援機能があって文章を書いたり推敲したりするうえでは最高の道具です。編集者にはWZなのです。（本）

●Windows、Mac、Android、iOSと、OSのバージョンアップは心ときめくものがあった。でも最近のOSはそういったものがあまり感じられない。Linuxディストリも以前ほどの感動がない気がする。歳とったせいなのかな。新しいハードウェアに便利なアプリの方が楽しそうだ。（幕）

●Emacs特集だったせいか、enchantMOONってちょっとEmacsに似てるんでは? と思う節が。手書きツール（エディタ）としての側面は氷山の一角で、水面下にある本質はMOON BlockやHTML5＋JSの実行環境ととらえれば。あ、最初に使い手を選ぶのも……ね（^^;（キ）

●家庭菜園をしていると外出中でも畑に雨が降ったかどうか気になります。そんなときは東京アメッシュを見ています。畑のある地域にどれくらいの量で何分間雨が降っていたかがわかるので便利。あまり降っていなかったら後で水やりに行きます。天気の予測以外にこんなふうにも使えるのは新発見でした。（よし）

●韓国でタッカンマリにはまりました。丸鳥とじゃがいも、葱、トッポギが入った鍋。これを醤油、酢、マスタード、タテギを混ぜたつけだれで食べます。これが絶品! でも大きい鳥1羽なので、3人で1羽がやっと。横の韓国人は1人1羽を完食。なのに私達の方がデブって……遺伝子を恨みます。（北）

●ワインはあまり得意ではないのですが、勝沼ぶどうの丘にあるワインカーヴに行きました。タートヴァンという試飲専用の容器を購入すると約180種類のワインを自由に試飲できるという施設。お気に入りを見つけようと思うのですが、種類がありすぎてどれがいいかわからず、結局今年も買わずじまい……。（ま）

ご案内

編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［FAX］
03-3513-6173
※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

［E-mail］
sd@gihyo.co.jp

Software Design
2013年11月号

発行日
2013年11月18日

●発行人
片岡　巌

●編集人
池本公平

●編集
金田富士男
菊池　猛
吉岡高弘

●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。