Special Feature 01
UNIXシェル活用
Special Feature 02
10ギガビットを支える技術
2014 January
01

Software Design

2014年1月18日発行
毎月1回18日発行
通巻345号
（発刊279号）
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
特別定価
1,380円

[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

UNIXシェルの
機能を学び、
スクリプトまで
基本技を磨く!

あなたの好きなシェルは何ですか?

WHAT'S YOUR FAVORITE UNIX'S SHELL ?

シェルがわかればシステムがわかる





Special Feature 01

Special Feature 02
未来を作るITインフラ
10ギガビットを実現するケーブリング技術

一般記事
ソーシャルゲームのDevOpsを支える技術（後編）
魔法少女リリカルなのはINNOCENTの舞台裏

新連載
小飼弾「SDでSF」　くつなりょうすけ「ひみつのLinux通信」



サーバ・ネットワーク・
システム安全
[特別付録]
法輪寺電電宮
情報安全
護符シール
Ver.2.0



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61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。

Presto

分散 SQL エンジン「Presto」

「Presto」は、Facebookが自社サービスのために開発し、2013年11月にオープンソースで公開した大規模データ向けの分散SQLエンジンです。問い合わせに対する結果をインタラクティブに取得するという目的に最適化されており、ペタバイト級のデータに対して即時性に優れた対話的な問い合わせを実現します。大量データを扱うためのオープンソースの分散処理システムとしてはHadoop MapReduceやApache Hiveなどがありますが、Prestoはこれら既存のソフトウェアに比べてCPU効率や処理スピードの点でおよそ10倍優れているそうです。

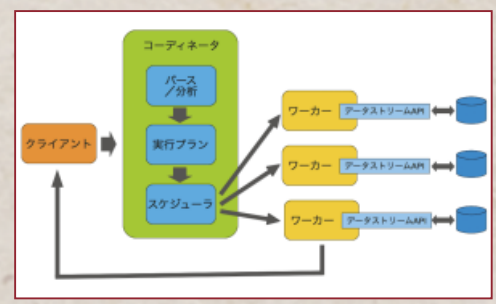
Facebookが Prestoの開発に着手した背景には、同社が抱える大量のデータに対して、リアルタイム性に優れた処理が必要になってきたという事情があるといいます。同社では300ペタバイトにものぼる大量のデータをHadoop/HDFS（Hadoop File System）ベースのクラスタによって保管し、バッチ処理やグラフ分析、機械学習、リアルタイムなインタラクティブ分析など、さまざまな用途に利用しています。従来から利用していたHadoop MapReduceやHiveは、大規模で信頼性の高い計算処理と、システム全体のスループットに最適化されており、十分に高い性能を備えています。しかしデータの規模がペタバイト級ともなると事情が異なり、クエリの遅延を減らすことに最適化されたシステムが必要になってきました。そのため、新しいしくみを持った独自のツールの開発を決めたとのことです。

Prestoのアーキテクチャ

Prestoでは、MapReduceとは異なる独自のアーキテクチャが採用されています（図1）。まず、クライアントはコーディネータと呼ばれるモジュールにクエリを送ります。Prestoのクエリ はANSI SQLに準拠しており、ジョインやアウタージョイン、サブクエリなどをサポートしています。コーディネータではクエリを分析して実行プランの組み立てを行います。

そうして作られた実行プランに基づき、スケジューラが実行パイプラインを組み上げて、データに近いワーカーに対してジョブを割り当て、データ処理の進捗を監視します。そして、クライアントはそれぞれの出力ステージからクエリの結果を引き出します。

▼図1　Prestoのアーキテクチャ



すべてのデータ処理はメモリとネットワーク上でパイプライン化されて実行されます。これによって不要な入出力を回避し、オーバーヘッドを削減しているとのこと。複数のステージをまとめて実行することも可能で、データをステージ間で順次ストリームすることによってエンド・ツー・エンドの遅延を最小限に抑えているそうです。また、拡張性の高さもPrestoの強みの1つで、シンプルなストレージ抽象化のしくみによって、コーディネータやワーカーは、HDFSをはじめとするさまざまなデータソースからメタデータの取得やデータのロケーション情報の取得、データへのアクセスを行えるようになっています。

Presto自身はJavaで開発されていますが、メモリ消費とデータ構造の扱いを慎重に行うことでメモリ割り当てやガベージコレクタの動作の安定性を確保しているとのこと。ソースコードはプロジェクトのWebサイトよりダウンロードでき、Java 7以上のJava仮想マシンで動作させることができます。ライセンスはApache License 2.0が採用されています。

クラウドベースのサービスを提供するうえでは、大規模データをどう分析しサービスにフィードバックしていくかが極めて重要な課題になります。そんな中でPrestoは、即時性の高い大規模データ処理を必要とするサービスプロバイダにとっての強い味方になるかもしれません。SD

Presto
http://prestodb.io/



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 Jan. 2014

1

第1特集 Special Feature 01

シェルがわかればシステムがわかる

あなたの好きなシェルは何ですか?





後藤 大地 019

第1章 シェル言語 020
なぜシェルは生まれたのか

第2章 きっちり押さえるシェルの基本 032
シェルの4大要素①
コマンド

第3章 きっちり押さえるシェルの基本 039
シェルの4大要素②
リダイレクト／変数とパラメータ／
パラメータ展開

第4章 ちゃんと知っていますか? 049
意外と知らない
「組み込みコマンド」の意味

第5章 もっと自由に! 055
制御構文でプログラミング

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

ソフトウェア・デザイン［目次］

Contents

Software Design Jan. 2014



2

第2特集 Special Feature 02

未来を作るITインフラ

10ギガビットを実現するケーブリング技術

入澤 徹 063

第1章 10GBASE-Tに求められるケーブリング技術 064

第2章 光ファイバ&ラックシステムにおける 将来を見据えたケーブルマネジメント 073

第3章 高速アプリケーションに最適化した 新型RJ-45開発記 079

一般記事 Article

法輪寺鎮守社電電宮 情報安全護符シールVer.2.0の使い方 017

ソーシャルゲームのDevOpsを支える技術（後編） ～魔法少女リリカルなのはINNOCENTの舞台裏～ 水野 拓宏、宮嶋 史尋、藤城 拓哉 162

巻頭Editorial PR Editorial PR

【連載】Hosting Department［第93回］ H-1

アラカルト A La Carte

ITエンジニア必須の最新用語解説［61］Presto 杉山 貴章 ED-1

読者プレゼントのお知らせ 016

SD BOOK FORUM 062

バックナンバーのお知らせ 087

SD NEWS & PRODUCTS 171

Letters From Readers 174

広告索引 AD INDEX

	広告主名	ホームページ	掲載ページ
ア	エーティーワークス	http://web.atworks.co.jp	P.4-5
サ	サイバーエージェント	http://www.cyberagent.co.jp/	裏表紙
	シーズ	http://www.seeds.ne.jp/	P.6
	システムワークス	http://www.systemworks.co.jp/	P.22
ナ	日本アイ・ビー・エム	http://www-06.ibm.com/systems/jp/x/highdensity/nextscale/	第1目次対向
	日本コンピューティングシステム	http://www.jcsn.co.jp/	裏表紙の裏
ハ	ハイパーボックス	http://www.domain-keeper.net/	表紙の裏-P.3

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/

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

ソフトウェア・デザイン［目次］

Contents

Software Design Jan. 2014

3

Column

digital gadget［181］	映像表現のためのプログラミング	安藤 幸央	001
結城浩の再発見の発想法［8］	Critical Section	結城 浩	004
enchant ～創造力を刺激する魔法～［9］	ソフトウェアを組み立てる『前田ブロック』	清水 亮	006
コレクターが独断で選ぶ! 偏愛キーボード図鑑［9］	Libertouch	濱野 聖人	010
秋葉原発! はんだづけカフェなう［39］	Maker Faire Tokyo 2013	坪井 義浩	012
SDでSF［新連載］	『ソードアート・オンライン』	小飼 弾	099
Hack For Japan～ エンジニアだからこそできる復興への一歩［25］	今年の夏もアツかった! ～石巻ハッカソン開催～［その2］	鎌田 篤慎、佐伯 幸治、高橋 憲一	154
温故知新 ITむかしばなし［29］	富士通の8ビットパソコン	佐野 裕	158
ひみつのLinux通信［新連載］	OMM killer	くつなりょうすけ	173



Development

サーバマシンの測り方［2］	データベース性能を測る	藤城 拓哉	088
分散データベース「未来工房」［7］	RiakアプリケーションとしてのRiak CS（2） ――基本機能から内部構造までじっくり解説	上西 康太	092
セキュリティ実践の基本定石 ～みんなでもう一度見つめなおそう～［7］	SSL/TLSデジタル署名の落とし穴	すずきひろのぶ	100
プログラム知識ゼロからはじめるiPhoneブックアプリ開発［9］	インタラクティブ性を持たせよう（その3） 卓上時計と写真の保存	GimmiQ（いたのくまんぼう、リオ・リーバス）	106
Androidエンジニアからの招待状［44］	［アプリ開発2013］5 ストレージのデータ保存基礎を学ぼう	重村 浩二	114
ハイパーバイザの作り方［16］	PCIパススルー その2「VT-dの詳細」	浅田 拓也	120
IPv6化の道も一歩から［最終回］	IPv6で外部接続した場合の動作確認方法	廣海 緑里、渡辺 露文、新 善文、藤崎 智宏	125

OS/Network

Be familiar with FreeBSD ～チャーリー・ルートからの手紙［3］	top(1)その2 ～スレッドの優先順位やI/O回数から見えるもの～	後藤 大地	132
Debian Hot Topics［11］	Init Systemとデスクトップ環境の動向／MiniDebConf in 台湾レポート	やまねひでき	136
レッドハット恵比寿通信［16］	Red Hat流OpenStack	小島 啓史	140
Ubuntu Monthly Report［45］	シェルスクリプトでもGUIのダイアログを使いたい!	あわしろいくや	142
Linuxカーネル観光ガイド［22］	ダイナミックトレースツール「ktap」	青田 直大	146
Monthly News from jus［27］	イノベーションの鍵をにぎる「デザイン思考」とは	波田野 裕一	152

Inside View

インターネットサービスの未来を創る人たち［30］	アドネットワークを支えるビッグデータ解析	川添 貴生	160

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 ＞ (c) Rob Reijnen/ Foto Natura/Minden Pictures /amanaimages
Illustration イラスト ＞ フクモトミホ、高野 涼香、中川 悠京　Seal Design シールデザイン ＞［Rocket Bomb］簑原 圭介
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。

技術評論社の 確定申告本

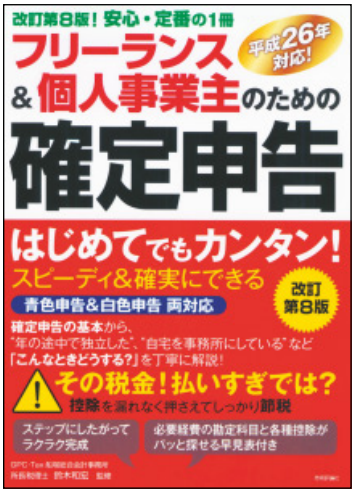
平成26年3月締切分



初めてでも大丈夫！ マネして書くだけ確定申告 平成 26 年 3 月締切分

山本宏　監修／ A4 判／ 176 ページ
定価 1,449 円（1,380 円＋税）
ISBN 978-4-7741-6059-7

家族が働いていたり副業があるために確定申告が必要なサラリーマンのために、確定申告に必要な書類の作成方法を、簡単にわかりやすくまとめました。サラリーマンだけでなく、フリーランスや不動産オーナー、年金で生活している方などが確定申告を行う場合の書類の作成方法についても、個別のケースごとに詳しく解説しています。本書を参考にすれば、該当するケースを探して、番号順にマネして書くだけで書類が記入できるので、忙しくて書類作成に時間をとれない方や、初めて確定申告を行う方などに、特におすすめしたい1冊です！



フリーランス＆個人事業主のための確定申告 改訂第 8 版

鈴木和宏　監修
A5 判／ 240 ページ
定価 1,554 円（1,480 円＋税）
ISBN 978-4-7741-6012-2

フリーランス＆個人事業主として働く方の確定申告はこの1冊で決まり！確定申告・帳簿付けの基本から、所得税・税額控除の計算法、申告書への記入の仕方まで、ステップにしたがって、確実にスピーディーに手続きができます。医療費や保険・共済など各種控除の情報も充実しており、節税ポイントもしっかり解説。青色申告・白色申告両対応です。支出や収入から勘定科目をすばやく検索できる早見表や、法人成りをした場合のメリット・デメリットなど、確定申告シーズン以外も、事業を営む皆さまのお役にたてる情報が満載です。

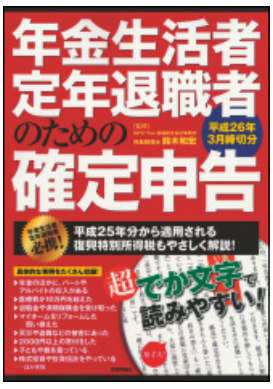


世界一ラクにできる確定申告

全自動クラウド会計ソフト「freee」で仕訳なし・入力ストレス最小限！

原尚美・山田案稜　著
A5 判／ 264 ページ
定価 1,659 円（1,580 円＋税）
ISBN 978-4-7741-6119-8

恐怖の3月15日 ―確定申告の締切を目前に、「ただでさえ忙しいのにもっとラクにできないの？」と思いませんか？そんな悩みを解決できるとして話題なのが、全自動クラウド会計ソフト「freee」。「めんどうな仕訳なし」「データを自動的に同期して入力ストレス最小限」という同ソフトで最大限ラクする方法はもちろん、「経費で落とせる取引と、落とせない取引の違いは？」といった悩みどころもフォローした、日本初の書籍です！



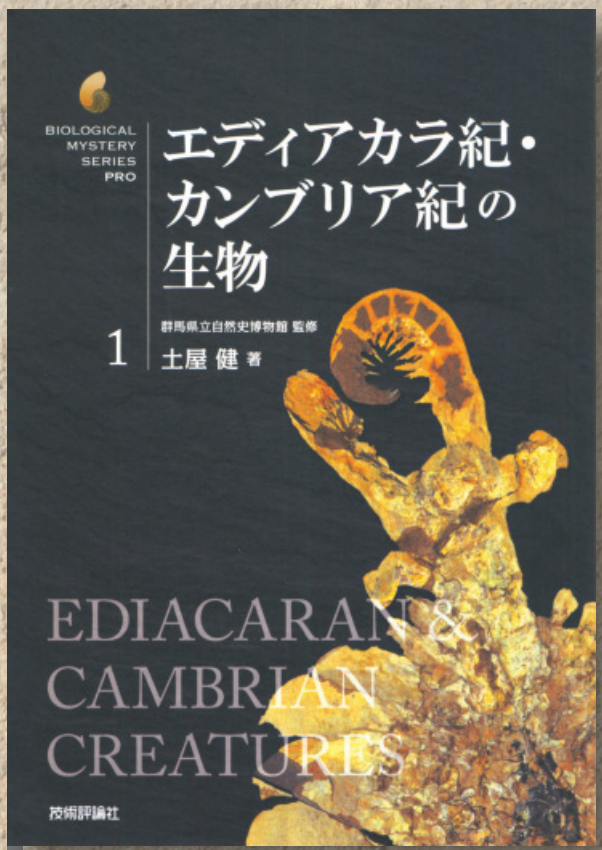
年金生活者定年退職者のための確定申告 平成 26 年 3 月締切分

鈴木和宏　監修／ A4 判／ 144 ページ
定価 1,554 円（1,480 円＋税）
ISBN 978-4-7741-6027-6

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

生物ミステリーPROシリーズ、創刊！

いまだから読みたい、古生物たちの世界



魅惑的な古生物たちの世界。

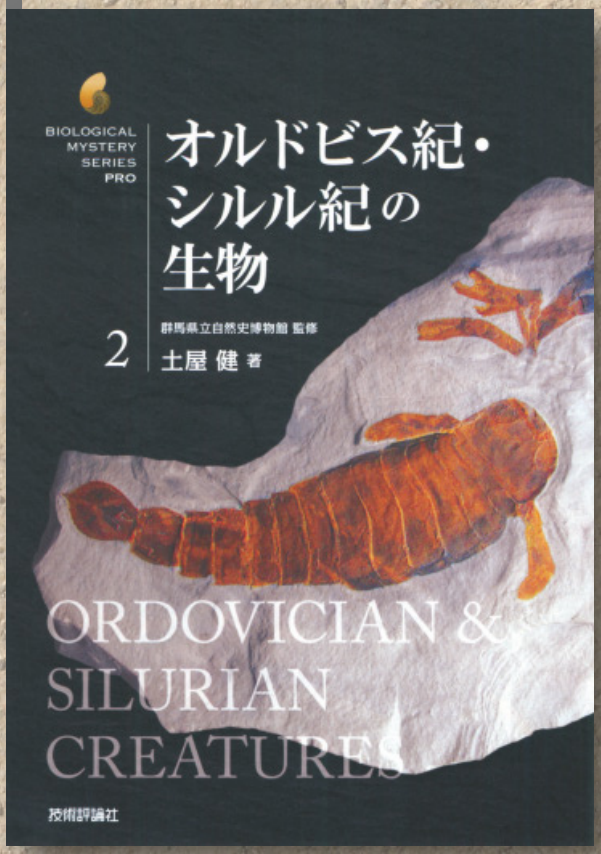
知的好奇心をくすぐり、知的探究心を呼び起こし、そして何よりシンプルに面白い。そんな世界を、みなさまにお届けします。

エディアカラ紀・カンブリア紀の生物

土屋健　著
群馬県立自然史博物館　監修
A5 判／ 192 ページ
定価 2,814 円（2,680 円＋税）
ISBN 978-4-7741-6084-9

オルドビス紀・シルル紀の生物

土屋健　著
群馬県立自然史博物館　監修
A5 判／ 144 ページ
定価 2,814 円（2,680 円＋税）
ISBN 978-4-7741-6085-6



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



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8回

ケーブリングのコツ

イラスト：高野涼香

はじめに

今までラック、電源、ネットワーク機器、サーバと続いてきました。今回はケーブリングについて少し話をしてみましょう。

ケーブリングとは?

ケーブリングとは、配線処理。すなわち、適切に処理がなされていると、利便性やメンテナンス性に優れ、作業の効率がアップし、ミスや障害を未然に防ぐことができます。ケーブリングのことを考え、スイッチや電源タップなどを配置し、適切な長さのケーブルを用いることが重要です。また、ケーブリングを行うためにいろいろなケーブル配線ソリューションも存在します。

これだけは気を付けよう

ケーブリングを行う中で重要なことを次にまとめます。

- 電源ケーブルとLANケーブルはなるべく同じルート上に置かない
- ケーブル同士が絡まらないように気を付ける
- タイラップなどで強く締め付け過ぎない
- 識別タグを付ける

ということです。

これらを順番に説明しましょう。

電磁誘導に気を付けよう

電源とLANを別々にする（分離させておく）理由は、電磁誘導を避けるためです。

電線に電流を流すと磁界が発生します。生じた磁界により、LANケーブル上に微量の誘導電流が発生し、ノイズとなります。これにより、データ化けなどによる通信不良などが発生する可能性があります。なるべく電源ケーブルとLANケーブルは、ラックの左側と右側などで分けると良いでしょう。

余談ですが、LANケーブルは、ツイストペアケーブルとも呼ばれ、2本1組で撚対線（よりついせん）になっており内部では互いに逆向きの信号を伝え、打ち消す形で電磁界やノイズを発生させないようにしています。撚（より）りが多いほど、発生しにくいようになっています。

ケーブル同士は絡まないように

ケーブルが絡まったり、引っかかったり、よじれたりしないように整理することが重要です。絡まっていると、不要なケーブルを撤去するだけでも一苦労ですし、引っ張ったら別のケーブルも一緒に引っ張られ、力がかかったり、ケーブルが抜けたりなど、障害に結び付くことがあります。また、絡まっていると、どのケーブルがどこに接続されているのか、ケーブルを追いかけるだけでも一苦労です。また、簾（すだれ）のように垂らして、クリアランス、エアフローの確保などを行うようにしつつ、不要なケーブルはなるべく撤去するなどしつつ、絡まないようにきれいにしましょう。また、ファイバの場合は損傷防止の

ため、固定するとき、角に当たる可能性のあるところなどは、スパイラルチューブなどを巻くなど保護も必要です。

強く締め付けない

LANケーブルの場合、通信に必要な周波数帯の信号を銅線に乗せて通信しています。強く締め付けてしまうと、周波数特性（帯域幅）が変わってしまったり、インピーダンス（抵抗）の変化につながり、通信不良が発生することがあります。強く締め付け過ぎないようにしましょう。お勧めはタイラップではなく、マジックバンドなどです。これらの使用をお勧めします（ちょっと値段は高いですが）。

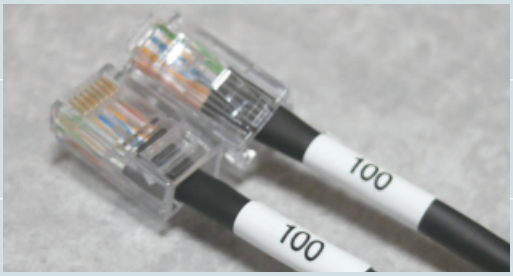
ケーブルタグを付ける

ケーブルタグというのも重要になってきて、線名札や識別タグなどとも呼ばれています。

スイッチはケーブルが高密度になればなるほど、どの機器に接続されているか、わからなくなりますので、タグを付けて認識しやすくします（写真1）。タグといっても、機器の名称を付ける方法から、ケーブルに一意な番号を振るという方法などがあります。

筆者は機器名称をタグに書くのではなく、番号で管理するようにしています。番号のメリットとしては、使い回しが効くということです。ケーブルの両端に書いておけば、お互い接続ポートを見ればわかりますし、管理表を作ることもできます。名称を書いてしまうと、用途が変わったときに外さないと混乱を生じますので、なるべく名称で管理しないようにしています。その中でも区別しないといけない物は色や別のケーブルタグを付けるなど、わかるようにしておくと良いでしょう。貼ったときにわかりやすいように上下（裏表？）の両面から見えるように2ヵ所に印字しておくとさらに便利です。

▼写真1　ケーブルにタグを付けておくと便利（筆者は番号で管理しています）



まとめ

上記のことをに気を付けて、ケーブルは用途ごとに色を分けたり（3色程度までのほうが無難です）わかりやすく工夫してみましょう。とくにスイッチ側のポートレイアウトを工夫することにより、より美しくきれいにするためのヒントです。

まずは、どのようなルートでケーブルを通すか、考えてみましょう。また、メーカー製サーバの場合は、ケーブルマネージメントアームなどもありますので、積極的に使ってみても良いですし、ケーブルマネージメント用の機材も広く出回っていますので、ぜひ検討しつつ、見栄えのよく効率の良いケーブリングを検討してみましょう。

ちなみに写真2のものは、ケーブルマネージャーという製品を使用しています。

具体的にどのようにやるのかは、ページの都合上紹介できなかったので、LANケーブル配線.com（http://www.lan-cabling.com/）や通信工業株式会社の技術情報（http://www.tsuko.co.jp/faq.html）を紹介しておきます。「ケーブリング」などで検索するとたくさんの情報が見つかるので、興味のある方は自分で調べてみてください。SD

▼写真2　ケーブルマネージャーでまとめておくとさらに便利





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

DIGITAL GADGET

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

Volume 181 >>>>> 映像表現のためのプログラミング

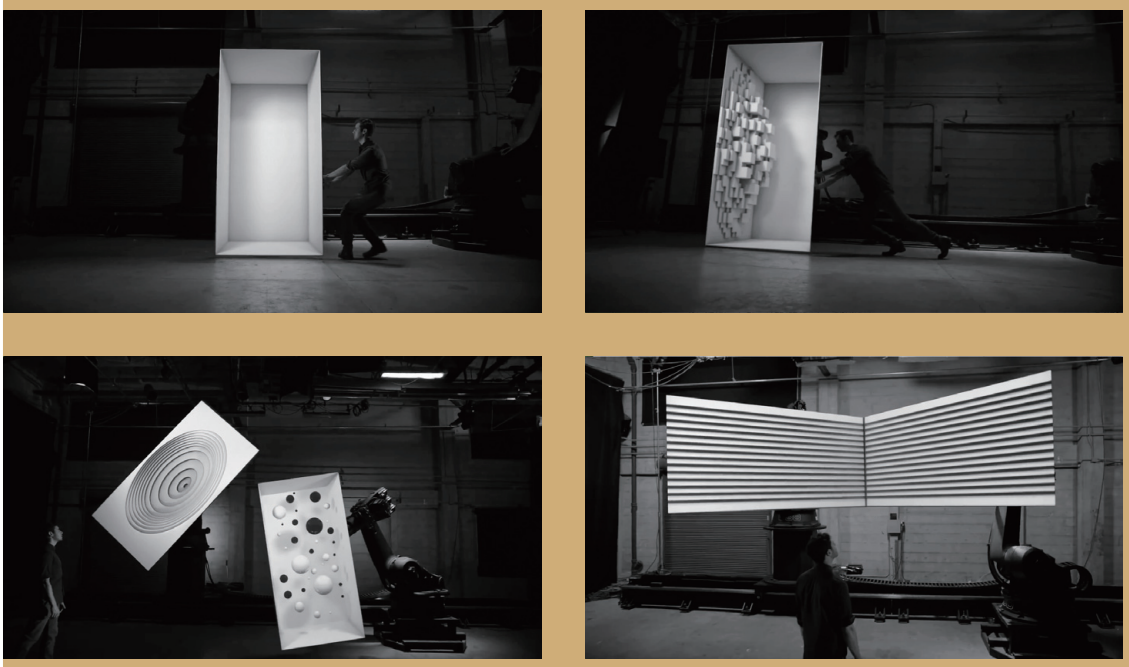
リアルタイムに生成される映像

ちまたで目を見張るような映像表現や、インタラクティブな（相互作用のある）映像表現を見たときに、どんな方法で作られているのか気になることはないでしょうか？ コンピュータを使えば何でもできると思いつつも、いざ実際に同じようなことを真似ようとすると意外と難しかったりします。

コンサートやライブの演出、最先端の技術力をアピールするようなTVCM、人に反応するデジタルサイネージ、プロジェクションマッピング技術を活用し巨大な建造物に映像をぴたりと投影するイベントなど、さまざまなところで素晴らしい映像表現が見られるようになりました。

身近なところでは、XBox Kinectに代表されるNatural User Interfaceの登場により、身体の動きを映像に逐次反映することが不思議ではなくなりました。ゲーム機の世界では身振り手振りなどの認識を活用し、ゲームコントローラの十字キーだけでは得られなかった楽しみ方、遊び方が提示されています。

ゲーム機のほかにも、これらのインタラクティブな映像制作には、それぞれの目的に適した専用のアート系プログラミング環境、映像コントロール用のプログラミング環境などが好んで使われています。もちろんゼロから一般的なプログラミング言語で構築できないわけではありませんが、よほど動作速度が重要視される局面でない限りあまり好まれません。それは必要なライブラリが煩雑になったり、環境構築が複雑になったり、本来の目的以外の



ロボットアームと巨大ディスプレイ、プロジェクションマッピング技法を組み合わせた映像表現
http://www.botndolly.com/box

※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。

ところで手間のかかる場合があるからです。

インタラクティブな映像に適したプログラミング環境は次のような特徴を持ち、いくらかの制限はありつつも、目的のことを素早く実現するのに向いています。

- 環境構築が平易
- 使いこなすために学ぶこと、学ぶ量が最小限で済む
- 多くの現場で使われている実績と信頼
- 実現したいことに対するノウハウの蓄積
- 利用者コミュニティが形成されていること
- 柔軟な拡張性を備えたアーキテクチャであること
- 最新技術の取り込みが早いこと

インタラクティブ系プログラミング環境

ここでインタラクティブ系プログラミング環境をいくつか紹介しましょう。

openFrameworks

http://www.openframeworks.cc

「oF」と省略して呼ばれる openFrameworks は、最近のインタラクティブ映像制作系ではもっとも引っ張りだこなプログラミング環境。グラフィックス、静止画、動画、オーディオ、フォント関連のライブラリが充実しており、必要なライブラリをプログラミングでつなぎ合わせていけば、目的の映像制作ができるようになっている。

シンプルなプログラミングかつ高速で動作し、プログラミングの初心者でも使え、拡張することも他者の拡張を取り込むことも容易である。世界のクリエーターによる作品群は CAN（http://www.creativeapplications.net）で見ることができる。

Cinder

http://libcinder.org

カンヌ国際クリエイティビティフェスティバル 2013 のイノベーション部門でグランプリを獲得したプログラミングフレームワーク環境。街頭広告系のインタラクティブ映像や、大規模イベントの映像コントロールに使われる傾向がある。しかしまだ少数派。iPad/iPhone にも対応していたり、スクリーンセーバーが開発できたり、マルチタッチに標準対応している点など、多くの機能が網羅されている。

Processing

http://processing.org

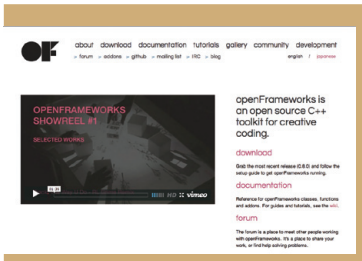
もともとは Design By Numbers（DBN）と呼ばれるアート表現向けの平易なプログラミング言語が進化したもの。Java ベースのプログラミング環境で、JavaScript 実装もある。プログラミングに親しんだ人でなくとも、スケッチするように見よう見まねで使い始められるところが最大の特徴かもしれない。また、作った作品を簡単に Web で公開することが可能なところも好まれている。

Quartz Compose

https://developer.apple.com/technologies/mac/graphics-and-animation.html

Mac OS X 限定。ノードベース（線をつないでいく）でパッチをつないでいくことで、インタラクティブな映像表現を構築していくためのツール。最新の Xcode には含まれないため、別途ダウンロード&インストールの必要がある。PixelShox Studio という VJ（Video Jockey）ツールを開発した Pierre-Olivier Latour 氏が中心となって開発したツール。

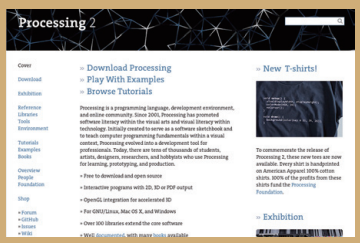
これらのほかにも、グラフや画像処理が得意な「NodeBox」（http://nodebox.net）、3Dアニメーションに強い「Polycode」（http://polycode.org）、データビジュアライゼーションに強い「D3」（http://d3js.org）、プロジェクションマッピングに強い「vvvv」（http://vvvv.org）、3Dグラフィックス表現に強い「PhiloGL」（http://www.senchalabs.org/philogl/）などなど、数多くの環境が存在します。それぞれ



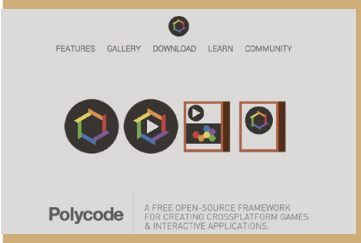
openFrameworks



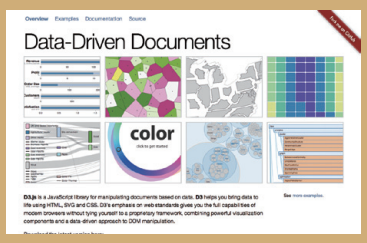
Cinder



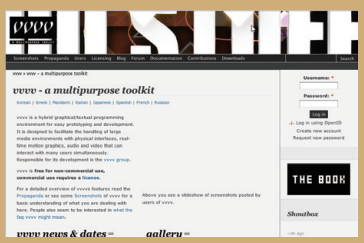
Processing



Polycode



D3



vvvv

得意・不得意があるので、ギャラリーや機能一覧、事例などで雰囲気をつかんだうえで活用するのが得策です。

より高度な映像制作のこの先

今では最先端の映像制作手法も、平易なものがオモチャで実現するようになったり、一般的なコンパクトデジタルカメラに機能が載ってきたり、スマートフォンのアプリで可能になる事柄も多くなってくるでしょう。新しい家庭用ゲーム機が出て、今は物珍しい映像表現も、すぐにどの家庭でも普通に遊ばれるような表現になるのかもしれないのです。

カメラの技術、ハードウェアの進化やコンピューティングパワーの巨大化、画像認識や映像加工技術の進化によっても映像表現が変わってくることでしょう。映像機器の潮流は、4K、8K といったハイビジョン以上の高解像度に移ってきており、高解像度ならではの映像表現、映像制作のニーズやコンテンツも考えられます。

その一方、旧来からある技術でも、切り口を変えた使い方によって目新しい表現が可能になる場合もあります。静止画の一部分が動画として動き続けるシネマグラフ手法や、最近また復刻しつつあるアニメーション GIF によるさまざまなパラパラアニメ風の表現もその一端かもしれません。

さらに、表現したい映像や手法があって、技術がまだ確立していない場合には、それらの技術を探し出したり、創り出す土壌も整ってきています。表現すべきデータや映像表現のアイデアがあって、それを可能にするツールやコンピューティングパワーはすでに存在しています。これから生まれる新しい映像表現は、普段見ている映画やテレビなどの映像表現の延長線上にはないのかもしれませんね。SD

gadget 1

Lego Calendar

http://vitaminsdesign.com/projects/lego-calendar/

スケジュール調整の架け橋となるブロック

本文でも紹介したopenFrameworksと、画像認識用のツールキットOpenCVを利用した実験プロジェクト。LEGOブロックで制作した、壁に貼付けた各人のスケジュールをカメラで撮影して自動認識します。さらにそのデータをオンラインのGoogleカレンダーと連携するしくみ。スケジュールの内容を示すのが単なる色のついたブロックなので、壁に貼っていてもプロジェクトの秘密が守れるのも良いところのようです。



gadget 2

ハコビジョン

http://www.bandai.co.jp/candy/hakovision/

映像を楽しむ食玩

スマートフォンの画面を活用した、小型映像投影玩具。食玩として、おもちゃ売り場や、雑貨店で手に入ることでしょう。投影対象となる立体的なフィギュアと、投影用のクリアプレートと、ガムがセットになったもの。簡易的なしくみではありますが、これをきっかけに多くの人が興味を持つようになれば素晴らしいことでしょう。



©東京ミチテラス 2012

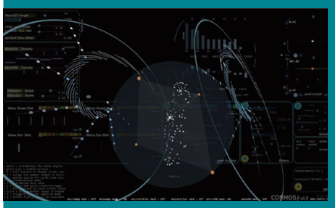
gadget 3

Cosmosf

http://www.sonic-lab.com/products-2

ビジュアルシンセ

Cosmosfは、太陽系や宇宙を喚起させる映像的、ビジュアル的なユーザインタフェース表現がなされた音響ツールです。楽器を演奏している際は「音」でしかその表現を認知することができませんが、音を作ったり、加工する過程では、音が見えたほうが、より適切な表現に近づくのが早くなるのかもしれません。このような映像表現である必然性は薄いですが、スタートレック風の操作パネルに不思議な魅力を感じるのではないでしょうか。



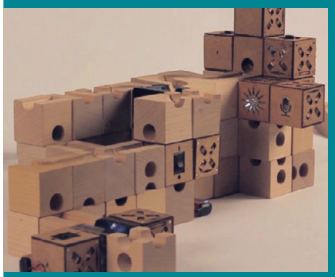
gadget 4

Cuboino

http://f3-h.de/cuboino/

デジタル積み木

Cuboinoはイギリスで古くから楽しまれている積み木cuboroをデジタル積み木として改造したものです。cuboroは積み木の中に空洞があり、積み木の組み方によって、玉が転がり落ちてくる道筋を作れるのが特長の積み木です。Cuboinoは木の積み木にセンサーやボタン、玉の通過スイッチなどのついた積み木を追加し、新しい可能性を付加しています。回路図などもすべて公開されています。



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

結城浩の再発見の発想法 8



Critical Section

Critical Section ——クリティカルセクション

クリティカルセクションとは

クリティカルセクション(critical section)とは、同時に複数のスレッドが入ってはいけない区間のことです。

コンピュータで処理を行うときに、効率を上げたり、動作の反応を良くしたりするために複数のスレッドを利用することは一般的に行われています。そのようなプログラムをマルチスレッドなプログラムと呼びます。

マルチスレッドなプログラムでは「この区間だけは同時に1つのスレッドしか絶対に入ってはいけない」という部分が作られます。それが今回お話しするクリティカルセクションです。

クリティカルセクションの例

クリティカルセクションの例として、銀行口座への入金処理を考えましょう。残高が10万円の口座に1万円の入金があったとき、それを扱うスレッドは次のような処理を行います。

(1)残高の取得
(2)入金額の加算
(3)残高の更新

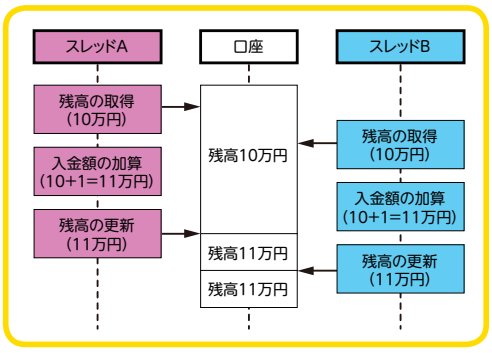
ここには問題はまったくありません。しかし、マルチスレッドなプログラムでは問題が発生します。それは上記の(1)(2)(3)の間に別のスレッドが処理を行う可能性があるからです。

残高が10万円の口座にAさんとBさんから1万円ずつ入金があったとします。Aさんの入金処理を行うスレッドAと、Bさんの入金処理を行うスレッドBとが同時に動作すると、**図1**のようになってしまう危険性があります。本来は12万円になるべきなのに、口座の残高は11万円になってしまいました。

こんなことが起きる理由は明らかです。スレッドAが残高の取得をしてから残高の更新を行うまでの間に、スレッドBが割り込んでしまったからです。

「わざわざ3段階に分離するから問題が起きるんじゃないか！」と思いますよね。そのとおりです。そこで1つのスレッドが「残高の取得→入金額の加算→残高の更新」を処理している間はほかのスレッドが入れないようにします。これはまさにクリティカルセクションを作っていることにほかなりません。

▼図1　残高が狂う場合



スレッドAがクリティカルセクションに入ると、入ろうとした別のスレッドBはいったん待たされます。そしてスレッドAがクリティカルセクションから出たタイミングでスレッドBが入れるようになります。その結果、処理は**図2**のように正しくなります。

クリティカルセクションの大きさ

銀行口座の例を考えると、なんだか当たり前の話をしているようですが、実際にはここに設計判断が必要になります。

クリティカルセクションを長くすれば致命的なことが起きる心配は減りますが、効率は落ちてしまいます。一方、クリティカルセクションを短くすれば効率は上がりますが、しっかり設計しないと致命的なことが起きてしまいます。つまり、ここにはトレードオフがあるのです。

共有リソースを守る

クリティカルセクションが守っている共有リソースを意識するのは大切です。共有リソースというのは、複数のスレッドが同時に読み書きを行う資源のことで、口座の例では顧客の口座そのものが共有リソースでした。

クリティカルセクションを的確に作るためには、複数のスレッドが、共有リソースをどんなときに読み書きするかをよく理解しなければならないのです。

道路上のクリティカルセクション

日常生活にもクリティカルセクションはたくさんあります。たとえば、交差点は重要なクリティカルセクションですね。青信号で1つの方向に車が流れているとき、それを横切る車の流れは赤信号でストップされます。両方青信号になったらたいへんな事故が起きてしまいます。信号が守っている共有リソースは交差点です。

会議でのクリティカルセクション

会議には司会者がいて、複数人が同時に話し出さないようにします。司会者が動的にクリティカルセクションを作っているのです。守っている共有リソースは、会議という場全体、あるいは参加者の耳でしょう。

誰か1人がずっと話し続けるのは長いクリティカルセクションに相当します。クリティカルセクションが長くなると、複数人がそこにいる意義が薄れてきます。

大人数の会議というのは巨大な共有リソースがあるシステムのようなものです。1人が発言すると、沈黙しなければならない人数が多くなりますから、効率はよくありません。

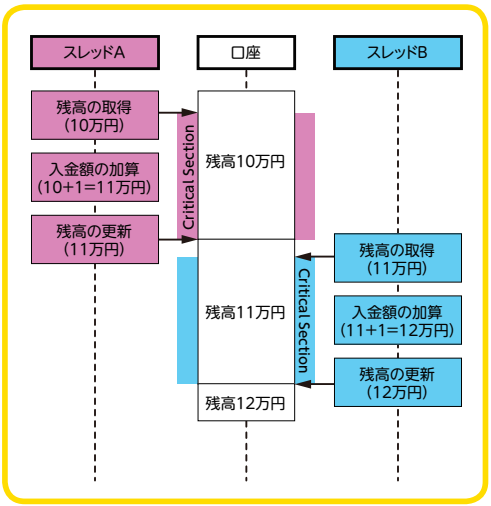
「複数人を投入すれば効率が良くなる」というのは単純過ぎる考えです。複数人が同時にアクセスする共有リソースが何であるか、どのようなクリティカルセクションが生まれるかを考えないと、かえって効率は悪くなります。

効率の良い会議を行うには、マルチスレッドのプログラムと同様に設計が必要なのです。

◆ ◆ ◆

あなたの周りを見回して、複数人が同時に動くことで起きるトラブルはありませんか。そこにはクリティカルセクションが必要ではないでしょうか。また、守るべき共有リソースはなんでしょうか。ぜひ考えてみてください。SD

▼図2　Critical Sectionで残高が正しくなる



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9回 ソフトウェアを組み立てる『前田ブロック』

enchant.jsの出だしの成功は、僕たちを大いに勇気づけてくれました。そして僕たちは、プログラミングをさらに多くの人に楽しんでもらうため、新たな挑戦を始めました。

レーサーミニ四駆を作った男

小学生の頃でした。

少年だった僕は、毎週土曜日の早朝に流れる番組「RCカーグランプリ」をいつも食い入るような目で見ていました。この番組は田宮模型(タミヤ)がスポンサードする番組で、その名のとおりRCカー(無線操縦の自動車模型)を競争させるレースシーンが目玉の番組でした。

司会はタミヤの前ちゃんこと、前田靖幸氏、初代のアシスタントは当時アイドル絶頂期だった日高のり子さん。この2人のコンビに加え、迫力のレース実況は小倉智昭氏。この番組は少年の頃の僕たちの胸を激しく揺さぶりました。

RCカーをやってみたい、そう思った僕はクリスマスプレゼントでグラスホッパー[注1]を手に入れ、早速組み立てようとしますが、難しくてとても作ることができません。父に泣きつくと、父はエンジニアらしい器用な手つきであっという間にRCカーを組み立ててしまいました。完成したRCカーを家の前の道路で走らせると、すぐにドブに落ちてしまい、泣きながら家に持ち帰ったのも、今ではいい思い出です。

このRCカー、しかし小学校には同じ話題で盛り上がれる友達は1人もいませんでした。この番組はみんなが見ていたものの、当時のRCカーは数万円と高価すぎて到底、小学生の手に届くものではなかったのです。

その数年後、タミヤは画期的な新商品を発売します。

それがレーサーミニ四駆です。数百円で購入することができ、組み立てもRCカーに比べると数段簡単です。組み立てて専用のサーキットを走らせるだけで簡単に友達と競争することができます。同じキットを同じように組み立てても、勝ったり負けたりするところがおもしろく、アフターパーツを工夫することでさらに遊び方に幅が出るようになっています。

この商品は大ヒットし、ミニ四駆ブームが到来しました。

この商品を企画した前ちゃんこと、前田靖幸さんと、僕は偶然出会うことになります。前田さんはタミヤからスクウェアに転職してゲームプロデューサーとなったあと、ゲーム会社を立ち上げ、楽天にバイアウトしていたのです。UEIの創業者の1人である武市智行さんは前田さんが働いていた頃のスクウェアで社長をしていて、その縁で僕は前田さんと会うことになりました。

「私のようなロートルにいまさらなにかお話で

注1) タミヤから発売されていたオフロード仕様のRCカー。

もあるのでしょうか」

　前田さんは謙虚にそう語りました。

「前田さん、僕はあなたにもう一度ミニ四駆を作ってほしいのです」

　僕はそう切り出しました。

「プログラミングはいまのところ、RCカーのようなものです。作るのが難しく、楽しめるようになるまでにもの凄い努力と特殊な才能が必要です。それをミニ四駆のように、簡単に、誰でも楽しめるようにしてほしいのです」

「……プログラミングですか。そりゃ難題ですね。なにしろこの前田もプログラミングに関しちゃ、てんで素人で……」

「だからこそです。素人でも楽しめるプログラミング言語を一緒に作っていただきたいのです」

　僕はアスキー・メディアワークス向けに書いたenchant.jsの入門書を手渡しました。前田さんは翌月からUEIにCHOこと、「チーフ・ホビー・オフィサー」としてやって来てくれることになりました。プログラミングを遊び(ホビー)にする、そういう使命を果たしてほしいと思って新設した役職です。これまで子供たちとともにあった前田さんにふさわしい肩書きだと思います。

電通マンにプログラミングを教える

　一方で僕は、SF作家としても知られ、電通の役員でもあった鏡 明さんから電通の若手社員向け研修の講師をしてほしいという依頼を受けていました。最初の数回はゲーム向けの企画の立て方などをレクチャーしていたのですが、電通の皆さんは飲み込みが早く、もっと複雑なことを教えることができるのではないか、と考えました。そこで、鏡さんにひとつ大胆な提案をしてみました。

「次の研修では皆さんにプログラミングをやってもらいましょう」

　鏡さんは眼を白黒させました。

「1日でプログラミングを教える。そんなことが可能なんですか？」

「そう難しいことはないはずですよ。そのためにenchant.jsがあるんです」

　それから僕はenchant.jsをより簡単に扱えるようにした簡易スクリプト言語「アトラスX」を作りました。次のような感じでプログラムを記述することができます。

```
プロローグ = function(){
      台詞("・・・ここは、どこだ?");
      一時停止();
      台詞("気がつくと見知らぬ小部屋に居た");
      一時停止();
      台詞("ひどい頭痛がする");
      一時停止();
      台詞("なんてこった・・・");
      一時停止();
      台詞("なにも・・・・");
      一時停止();
      台詞("思い出せない");

      次へ(シーン2);
};
```

　これは一見すると専用言語のように見えますが、れっきとしたJavaScriptです。JavaScriptでは識別子に日本語を使えるのでこういう書き方ができます。わずか7つの命令しかなく、すぐに使い方を飲み込めるように工夫されています。ここまで簡単なら、電通マンなら誰でも使いこなせるだろう、と思って研修に挑みました。

　いざ研修に挑むと、確かに電通マンはこのアトラスXの文法をすぐに飲み込みました。ところがいざプログラムを打ち込むと、「動きません!」「わかりません！」と大混乱が生じたのです。

　援護で来ていた伏見と高橋が見てみると、ダブルクォーテーションやアンダーバーといった記号の打ち方がわからない、全角と半角の区別がつかない、括弧が対応していない、など散々でした。つまり、論理構造以前のキーボード操作でのつまずきが想像以上に大きかったのです。

　それでも結局、なんとか8時間後には全員が作品らしきものを作ることができたのですが、if文やループを教えるとやはり大混乱に陥ることがわかりました。立派な成人にプログラミングを教えることですらここまで苦労するわけですから、子供にまでプログラミングを教えようというのは相当難しいぞ、と頭を抱えました。

仕様書を書くようにプログラミングする

僕たちは山梨の温泉に日帰りで合宿にでかけました。僕はこうしたアイデアソンを近藤誠とよくやっていました。帰りがけ、僕は1つアイデアを思いつきました。

enchant.jsをラップする、もう1つ上位の階層を作ったらどうか——それはもっと自然言語に近い形で、仕様書を記述するようにプログラムを記述するのです。我ながら、悪くないアイデアだと思いました。東京に戻って早速試作し、ひとまずpuppet.enchant.jsと名付けます。

たとえば、クマを画面に出して右に移動するだけのプログラムは、通常のenchant.jsではこう書きます。

```
enchant();
window.onload = function(){
    game = new Game(320,320);
    game.preload("char1.gif");
    game.onload = function(){
        var kuma = new Sprite(32,32);
        kuma.image = game.assets["char1.gif"];
        kuma.x=100;
        kuma.y=100;
        kuma.addEventListener(
            'enterframe',function(){kuma.x++;});
        game.rootScene.addChild(kuma);
    }
    game.start();
}
```

いかにもプログラムという感じです。もともとプログラミングの経験があれば、これはそれほど難しくは感じませんが、まったくプログラミングの経験がないと、この簡単なサンプルを理解するだけで相当な苦労が必要です。

そこでpuppet.enchant.jsでは次のような書き方をできるようにしました。

```
Puppet.create("くま", {
    filename: "chara1.png",
    w: 32,
    h: 32,
    behavior: ["一人で登場", "右へ動く"]
});
```

この「一人で登場」や「右へ動く」といった部分はビヘイビア(behavior)と呼ぶもので、内部的にはあらかじめ定義された振る舞いを定義するオブジェクトに変換されます。たとえば「右へ動く」は次のように定義されます。

```
window['右へ動く'] = 'moveRight';

    moveRight: {
        enterframe: function() {
            this.x += this.speed * 0.1;
        }
    },
```

これはenchant.jsが本来イベントリスナーの塊だけでプログラムを表現しようとしているという性質を、「振る舞い」に置き換えたものです。

puppet.enchant.jsでは、ビヘイビアをユーザが任意のタイミングで追加することができます。ビヘイビアを追加した場合、動作が置き換わるのではなく、動作が合成されます。たとえば、ジグザグな動きを実現するzigzagXとzigzagYは次のような定義になっていて、

```
    zigzagX: {
        enterframe: function() {
            this.x += Math.cos(this.age * 0.1) *
                      0.1 * this.speed;
        }
    },
    zigzagY: {
        enterframe: function() {
            this.y += Math.sin(this.age * 0.1) *
                      0.1 * this.speed;
        }
    },
```

この2つを合成すると、円運動になります。つまり、通常のプログラミングでは「どのように(how)」を記述しますが、puppet.enchant.jsが目指したのは「何を(what)」したいかを記述することを目指しているのです。

howを記述する部分をできるだけ少なくしようというのがこのpuppet.enchant.jsの発想でした。もし記述するとしても、既存のよく使う動きに関しては毎回howを説明する必要はなく、差分だけ書けば良いことになります。

前田ブロックの誕生

puppet.enchant.jsの最初のバージョンは半日で完成しました。基本的なアイデアを考えるの

に時間がかかっただけで、実装そのものは複雑ではなかったからです。puppet.enchant.jsはしかし同時に危険なプラグインでもありました。いや、プラグインですらないかもしれません。

window.onloadのようなJavaScriptの基本的な構文も隠蔽し、プログラムの中でデータを定義するのではなく、データの中でプログラムを定義するという逆転構造を持っています。これはいままでのような手続き型プログラミングのパラダイムに慣れた人たちからは敬遠されるだろうと思いました。

しかし、もともとの動機はプログラミングスキルがない人にできるだけ短いステップでプログラミングの魅力を伝えることだったので、思い切ってこのpuppet.enchant.jsをビジュアル言語でラップすることにしました。Googleがちょうど Blockly というHTML5ベースのビジュアル言語をオープンソースとして公開していたので、これを改造することでpuppet.enchant.jsをラップしたビジュアル言語ができました。

前田さんの入社と同時だったので、これに「前田ブロック」と名付けました。前田ブロックでは、たとえば画面にクマが出て来てマウスカーソルを追いかける、ということを**図1**のように記述できます。

このブロックは次のようなpuppet.enchant.jsの文法に変換されます。

```
Puppet.create("くま", {
    filename: "chara1.png",
    w: 32,
    h: 32,
    behavior: ["standAlone",
    {
        sceneStart: function() {
            this.interval = 30;
            this.initialNumber = 10;
        }
    }, "tapChase"]
});
```

早速前田さんに見せると、「あ、これなら子供にも説明できそうです」という好感触を得て、彼は早速、小学館のコロコロコミック編集部に赴きます。ミニ四駆時代に培った人脈を活かして、コロコロ編集部を説得するためです。

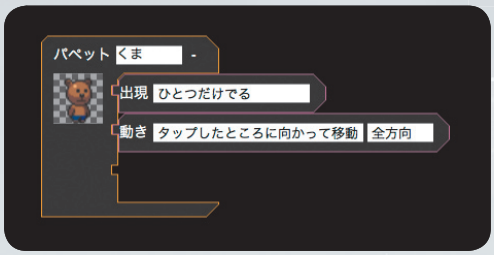


図1　前田ブロックのプログラミング例

僕たちは前田ブロックを使ったワークショップを、まずはロサンゼルスで開催しました。6歳から60歳まで、幅広い年齢層の人々がプログラミングを趣味として楽しみたいと考え、集まってくださったのです。その場でたくさんの人々がプログラミングの喜びにふれ、思わず顔をほころばせる場面を見て僕は、自分たちのやろうとしていることに間違いはないのだという確信を得ました。

しかし同時にそうしたワークショップの中で、「もっと違う絵を動かしたい」「自分の描いた絵を動かしたい」という要望を多くいただきました。

前田ブロックによってプログラミングの敷居そのものは下がったものの、根本的に、できることが用意されている素材やブロックに拘束されてしまうという点で、前田ブロックはまだ汎用的な言語とは言えませんでした。

さあどうしようか。

僕は頭を抱えます……となると、プログラミングを簡易にするだけでは、プログラミングの楽しさを伝えることはできても、夢中にさせることはできないということです。

プログラミングする環境そのものを根本的に変えなければなりません。どんな人でも自然にプログラミングを始めるような、日常的にプログラミングしながら何かをするような、そういうものを作らない限りは、プログラミングは週末の趣味で終わってしまいます。どうすればそんなことができるでしょうか。

日常の文脈の中にプログラミングを埋め込む。

この命題は、この先ずっと僕の中に残ることになります。SD

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9回

最高のキータッチを持つメンブレンキーボード

Libertouch



写真1　Libertouch

はじめに

今月は最高のメンブレンキーボードの1つであるLibertouch（リベルタッチ）（**写真1**）を紹介します。メンブレンキーボードと聞くと、PCを購入したときに付属するキーボードや、市販されている比較的安価なキーボードのような印象を持たれることが多く、キータッチもそれほど良くないと思われがちです。しかし、すべてがそうではありません。Libertouchは最高のキータッチを持つキーボードです。

Libertouch

Libertouchは、富士通コンポーネント㈱が開発したUSBキーボードです。「キータッチの心地良さを追求したキーボード」と謳われているとおり、打っていて非常に心地良いです。キーボード自体はテンキーのついたフルキーボードで、キーボードの色が白色のものと黒色のもの、キーボード配列が日本語配列のものと英語配列のものの合計4バージョンがあります。

特徴

Libertouchは、見た目が一般に販売されているキーボードとほとんど変わらず、多少チープな印象であることは否めません。しかし、その一番の特徴はやはりキータッチにあり、キー入力をするというキーボード本来の機能にこだわったキーボードです。

一般的なメンブレンキーボードは、メンブレンシートと呼ばれるシートの上にラバーカップが乗っかった構造が多いです（**写真2**）。

それに対し、Libertouchはラバーカップだけでなく、コイルバネも利用してキートップを保持しています（**写真3**）。このことでキーを押したと認識させるためにキーの底まで押し込む必要がなく、より軽く柔らかなキータッチが実現されています[注1]。

キーの荷重は45gとキーボードの中では比較的軽い部類です。メカニカルキーボードで採用されているCherryのキースイッチの中でよく使われている赤軸、青軸、茶軸、黒軸の中で一番軽い赤軸と同等と言え

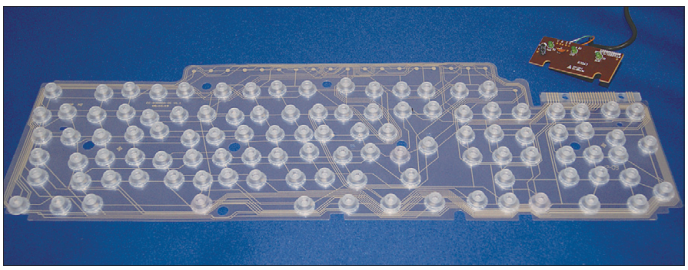


写真2　市販のメンブレンキーボードの分解

注1）キーの荷重がピークと底の中間地点で認識されます。http://www.fcl.fujitsu.com/release/2007/20070619.html

ば、軽いタッチであることがよくわかると思います。ただこれだけだと、45gでもまだ重いという方や、重いタッチのほうが好みという方にとっては使いづらいと思われるかもしれません。そこで、Libertouchにはキーの荷重を変更できる交換用のラバーカップが付属しています(**写真4**)。

交換用のラバーカップは2種類あり、キーの荷重は赤色のほうに交換すると35g、白色のほうに交換すると55gとなります[注2]。ラバーカップはLibertouch本体にそれぞれ15個しか付属しておらず、すべてのキーを交換するには足りませんが、50個入りのものが別売で販売されています。

標準の15個だけでも、力の弱い指／強い指で打つ一部のキーだけを交換するというアプローチは可能です。たとえば、力の弱い小指で打つキーは荷重を35gと軽くしておく、親指で打つキーは荷重を重くしておくといったカスタマイズが挙げられます。この指の強さに合わせたアプローチは、同じ高級キーボードであるRealforceの変荷重キーボードでなされていますが、Realforceの場合、軽いキーと重いキーが固定されており、細かくカスタマイズはできません。Libertouchではどのキーの荷重を軽くするか、どのキーを重くするかといったことを使う側が自由に決められます。たとえば、タイピングの運指(指づかい)が特殊で小指を多く使うのでより多くのキーを軽くしたいというケースや、中指を怪我していて中指で打つキーを軽くしたいというケースなどにも対応できます。

また、Libertouchはタッチの良さだけでなく、キー入力の静音性や、キーボード自体の重量があるといった特徴もあります。静音性は一般的なメンブレンキーボードと同様に優れており、メカニカルキーボードなどと比べるとはるかに静かで、打鍵音を気にしなければならない環境でも安心して使えます。キーボードの重量も十分あり、キー入力をしていてキーボードの位置がずれてしまうということがありません。

そのほか、左のCtrlとCaps Lockの交換用キートップが付属します(**写真5**)。これは左のCtrlとCaps Lockを入れ替えるためのものですが、このことからもプログラマのようなキーボードをよく使うユーザ向けの製品であるとわかります。しかし、Libertouch自体に左のCtrlとCaps Lockの入力を入れ替える機能は存在しません。キーを入れ替えるのであれば、OSごとに何か特別なアプリケーションを使ってキー入力も入れ替える必要があります。

入手方法

多くのネットショップで取り扱っています。少し大きめの量販店であれば店頭でも販売しています。キーボードの定価は18,000円ほどですが、安いところであれば12,000円ほどで買えるようです。交換用のラバーカップは、3,000円ほどで販売しているようです。

◆ ◆ ◆

今月はLibertouchを紹介しました。Libertouchは打っていて非常に気持ちの良いキーボードで、長く使うお供として最適なキーボードと言えます。筆者は、このLibertouchこそ最高のキータッチを持つキーボードの1つであると考えています[注3]。少し大きめの量販店などに行けば、実際に触って試せる店もありますので、ぜひとも一度触ってみてください。SD

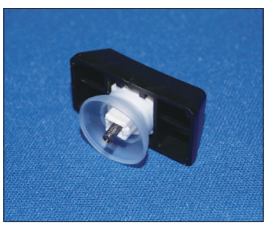


写真3　Libertouchのキー(ラバーカップとコイルバネ)

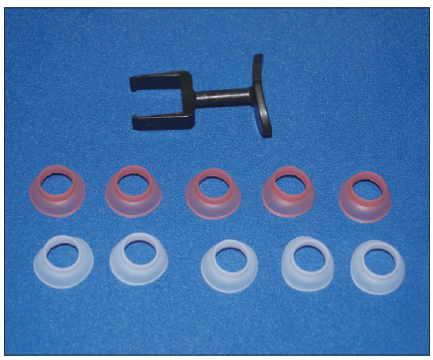


写真4　交換用ラバーカップとキー引き抜き工具

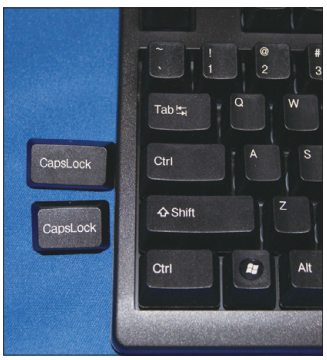


写真5　交換用キートップ

注2) ラバーカップを交換するために必要なキーの引き抜き工具もキーボードに付属しています。
注3) 筆者は普段テンキーをまったく使わないので、テンキーレスのバージョンを販売してほしいと願っています。

はんだづけカフェなう

Maker Faire Tokyo 2013

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/

Maker Faire Tokyo 2013

去る11月3日と4日、Maker Faire Tokyo 2013が開催されました。今年は昨年も開催された日本科学未来館に加え、その近隣のタイム24ビルも会場に利用され、さらに規模の大きなMaker Faireとなりました。この連載の第20回と第32回でも紹介した、中国の深圳でオープンソースハードウェアビジネスを行っているSeeed Studioのスタッフをアテンドをするなど、いろいろな予定が詰まっていたため、あまり会場を見てまわることができませんでした。筆者が見ることができた範囲ですが、いくつか展示されていたものを紹介させていただきます。

Galileo

今回、ついに日本のMaker Faireでもintelがスポンサーになったことが印象的でした（**写真1**）。前回紹介したGalileoの登場に合わせてのことでしょう。プレゼンの時間にGalileoの国内販売に関する発表もありましたが、12月中旬とのことでしたので本誌が読者の手元に届くころには発売されているでしょう。

筆者が、ローマでintelさんからいただいたサンプルを使ってみて、いくつか思うところがあったので基板を作ってみました。**写真2**は、シリアルコンソールへのアクセスをするケーブルがないという個人的な要望を満たすために作ったものです。ほかのいくつか作ってみたものと一緒にMaker Faire Tokyoの会場で展示を行いました。Galileoの発売開始と近い時期に、スイッチサイエンスから販売できるように準備を進めています。

GalileoはArduinoのスケッチがLinuxのユーザランドで走る構造です。GalileoはUEFIを採用しており、microSDからLinuxを起動することもできます。基板の裏面にはMini PCIeスロットもありますし、基板にはUSB端子が存在しますので、Linux側を触るとArduinoではなくLinuxの動くマイコンボードとして楽しむことができそうです。

Autodesk

AutoCADでお馴染みのAutodeskは、第34回で紹介したベイエリアのMaker Faireで展示

▼写真1　intelのプレゼン



▼写真2　Galileo用シリアルケーブル



されていた123D Cartchフォトブースの日本版を設置していました（**写真3**）。アメリカのものはデジタル1眼を並べたものでしたが、日本版はiPod touchを並べたものでした。アメリカで撮影してもらった筆者の3Dデータのできは残念なものでしたが、日本版ではどうなのでしょう。こちらでもたいへんな人気だったため、撮影してもらう機会は得られませんでした。

Autodeskは、数年前にInstructables[注1]というサイトを買収しています。このサイトはいろいろなものの作り方を共有するコミュニティサイトで、筆者もたまにおもしろい工作ネタを見つけて楽しんでいます。今回、このInstructablesに日本版が登場し、趣旨が発表されました。AutodeskのCEOも来日し、プレゼンが行われるという力の入りようです。

プロ用ソフトウェアのメーカーでありホビーユーザとは縁遠かったAutodeskですが、昨今ホビーユーザ向けソフトウェアやサービスが続々投入されていて目が離せません。

NSX-1

NSX-1という次世代音源LSIを10月下旬に発売したヤマハも、Maker Faireのスポンサーとして出展していました。人の歌声を合成する技術「VOCALOID」をNSX-1専用にカスタマイズしたeVocaloid音源として、あるいは、サンプリングと演奏技法をつなぎ合わせることでリアルなサウンドを出力可能なRAS（Real Acoustic Sound）音源として使うことのできるチップです。NSX-1はeVocaloidやRASと同時に、General MIDI音源も同時に使用することができます。

ヤマハのブースでは、MIDIキーボードでの演奏を、このRASでサックスの音として再生するデモンストレーションが行われていました（**写真4**）。また、NSX-1のプレスリリースで案内されていた、学研の「大人の科学」で来春出版される予定の歌うキーボードや、eVY1シールドの案内もなされていました。

eVY1シールドは、NSX-1のeVocaloid音源をArduinoやパソコンから手軽に使えるようにしたArduinoのシールドです。Arduinoからは、シリアル（UART）から（パソコンからはUSB-MIDIから）MIDI命令を送ることで歌わせることができます。最近、Web MIDI API（W3Cでドラフトとなっている状態です）がChrome Canaryで使えるようになり、ブラウザでMIDI命令が扱えるようになりつつあります。eVY1シールドをパソコンにつなぎ、Chrome Canaryでブラウザから歌わせるということも可能になっています。

GloGlobe

工作ユニット、HelloWORLDさんの展示でおもしろいものを見かけました。GloGlobeとい

▼写真3　Autodeskブース



▼写真4　ヤマハブース



注1）　http://www.instructables.com/ja/

う、LEDを使った球面ディスプレイです(**写真5**)。モーターを使って回転し、LEDをチカチカと点滅させながら動かすことで残像を使って絵を表示します。おもしろいのは、パソコンなどで使われるファンを改造して本体を回転させるモーターにしている点、部品の一部が3Dプリンタで作られているといった点です。また、リモコン基板から赤外線でGloGlobeに表示内容を転送することもできるそうで、実際に表示内容の書き換えをデモしていただきました。

GloGlobeはATmega88Aという Arduinoで使われているATmega328Pと同一シリーズのマイコンを使っていますが、筆者は最近ARMマイコンがお気に入りです。ARMマイコンを使って、GloGlobeのようなディスプレイを作ってみたくなりました。

mbed

ARMマイコンと言えば、本連載でも何度か紹介させていただきました。ARMマイコンを製造しているNXPのブースでは、第37回でも紹介をしたLPC1114FN28がmbedとして使える旨や、これを使ったうおーるぼっとminiなどを展示していました(**写真6**)。

うぉーるぼっとminiは、赤外線リモコンで操作できるロボットのベースボードです。そのままでも使うことができますが、mbed開発環境で開発して、自分だけのロボットを作ることができます。本体の下には磁石が取り付けられており、冷蔵庫の側面やホワイトボードなどの金属製の壁面に貼り付いて走ることもできます。

ブラウザでスペアナ

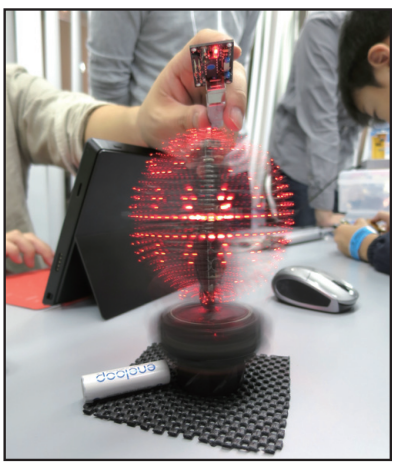
唐突ですが、スペアナ(スペクトラムアナライザ)をご存じでしょうか。スペアナは信号に含まれる周波数成分の分布を見るための装置で、ひと昔前までのオーディオ機器などで低音や高音の強さを表示するバーグラフのようなものです。GPS Laboは、このスペアナをHTML 5とJavaScriptを使いWebブラウザで実現したデモを行っていました(**写真7**)。

ここでは、JJYと呼ばれる日本標準時の標準電波(40kHz)をAmazonのダンボール箱に電線を巻いたアンテナで受信し、USBサウンドカードでパソコンに取り込んでグラフの描画と受信内容の解析がデモされていました。サウンドカードの入力のほかにもUSB接続できるタイプの受信機を使って、より高周波の受信と解析を行うこともできるようです。スクリーンショットなどは、GPS Laboのサイト[注2]にありましたので、興味のある方は見てみてください。

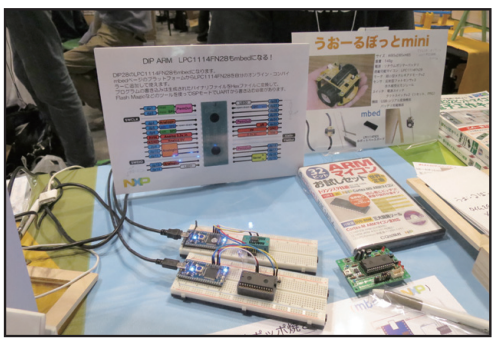
3Dプリンタ

3Dプリンタ関連の展示も数多くありました。

▼写真5　GloGlobe



▼写真6　NXPのブース



注2)　http://news.gps-labo.com/

第36回で紹介したRepRap Community Japanのブースでは、前回紹介したパラレルリンク機構の3Dプリンタ、Rostock miniが展示されていました(**写真8**)。会場では、樹脂を積層するのではなく、シリンジ(注射筒)からインクを出して紙に印刷するというデモが行われていました。3Dプリンタで基板にペースト状のはんだを印刷するということに挑戦しているそうです。

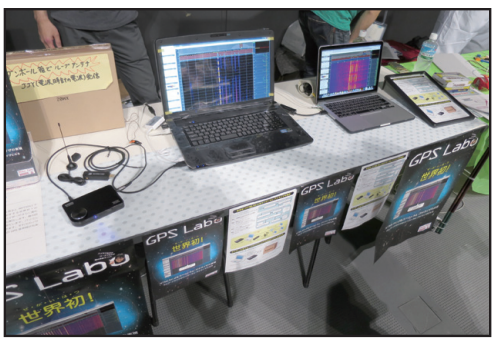
折りたためるRepRap、FoldaRapの展示もありました(**写真9**)。FoldaRapは輸送時に本体を折りたたむことができ、折りたたんでケースに入れてしまうこともできます(**写真10**)。設計した方は前回のローマに出展していらっしゃったのですが、日本にFoldaRapユーザがいるということをご存じでした。RP(3Dプリントした3Dプリンタの部品)を出力する条件がシビアで、なかなか作るのが難しいからユーザが珍しいようです。

Makerムーブメント

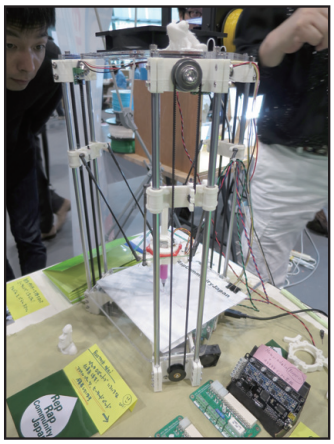
割とマイナーな趣味だったDIYが、昨年くらいからMakerムーブメントとして急に脚光を浴びるようになりました。今回のMaker Faireのスポンサーにも有名な企業が名を連ねるようになっています。筆者が今年参加したベイエリアでは、トヨタもスポンサーをしていました。正直なところ、Maker Faireのスポンサーをしたり、Maker向けのツールをリリースしてもこれらの会社の売上に与えるインパクトはごく小さいものでしょう。「企業の目的は、顧客の創造である」というドラッカーの有名な言葉がありますが、スポンサー企業は顧客の創造を行っているのでしょうか。

オンラインでは気づかなかったおもしろい作品が見られるのが、Maker Faireの楽しみだと筆者は考えています。今回のMaker Faireはそういった楽しみと、お祭りの楽しみ、両方が楽しめるイベントだったと感じました。SD

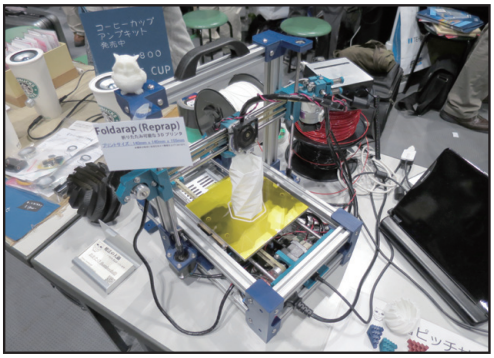
▼写真7　GPS Labo



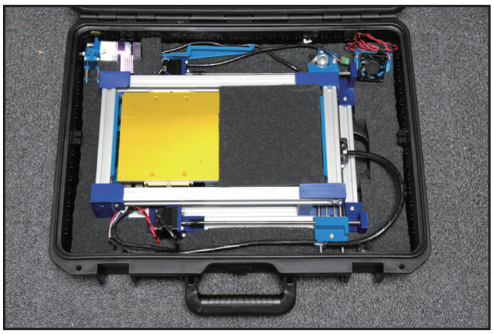
▼写真8　Rostock mini



▼写真9　FoldaRap



▼写真10　折りたたんだ状態



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2014年1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 Olasonic TW-BT5

1名



音響的に理想とされる卵型の高剛性キャビネットを採用し、クリアな音質を実現したOlasonicブランドの小型高音質スピーカー。そこにBluetoothワイヤレス接続の機能が加わった新製品です。Bluetooth対応のスマートフォンや音楽プレーヤなどの音源を1台のスピーカーだけで簡単に広い音場感を持った高音質で楽しめます。NFC対応。

提供元 東和電子
URL http://www.olasonic.jp

02 弥生会計 14 スタンダード

3名



2014年4月1日より施行される消費税率8%への引き上げに対応した会計ソフト。シンプルな操作で、あらゆる個人事業者／小規模法人の会計業務を効率化します。法人決算、個人決算（青色申告／白色申告）にも対応。Windows 8.1/8/7/Vista/XPで動作可能。

提供元 弥生 **URL** https://www.yayoi-kk.co.jp

03 ウォーク麺

2名



世界初のまったく新しい発想のカップ麺専用のホルダー。カップ麺を本体にセットして熱湯を注いだら専用のフタをするだけ。「熱くない」「フタができる」「こぼれない」と三拍子そろい、これで物が多くて手狭なデスクでも安心してカップ麺を食べられます。

提供元 ナイクウイン **URL** http://www.cupmenholder.com

04 ウイルスバスター モバイル（1年版）

2名

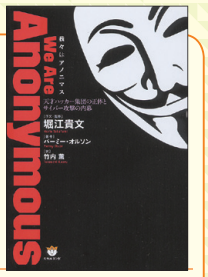


紛失／盗難対策、不正アプリ対策、Web脅威対策などの機能を備えたスマートフォンやタブレット専用の総合セキュリティ対策アプリ。Android 2.2/2.3/3.0/3.1/3.2/4.0/4.1/4.2/4.3/4.4で動作可能。

提供元 トレンドマイクロ **URL** http://www.trendmicro.co.jp

05 我々はアノニマス

4名



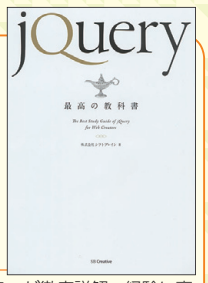
パーミー・オルソン 著、竹内 薫 訳、
堀江 貴文 監修／四六判、477ページ／
ISBN = 978-4-86471-157-9

アノニマスは言わずと知れた、世界に脅威を与える神出鬼没のハッカー集団。彼らはいったい何者なのか？　気鋭のジャーナリストが、首謀メンバーを直接取材。謎の集団の核心に迫りました。

提供元 ヒカルランド **URL** http://www.hikaruland.co.jp

06 jQuery 最高の教科書

2名



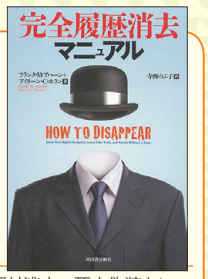
㈱シフトブレイン 著／
B5判、320ページ／
ISBN = 978-4-7973-7221-2

jQueryをこよなく愛するトップクリエイターが徹底詳解。経験に裏付けられた「わかりにくいポイント」を押さえた解説だから確かな基礎力と、実務で活かせる実践力を身につけられます。

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

07 完全履歴消去マニュアル

2名



フランク・M・アハーン、
アイリーン・C・ホラン 著、寺西 のぶ子 訳／
四六判、240ページ／
ISBN = 978-4-309-24639-0

ストーカーや探偵からの身の隠し方から、国外逃亡、死亡偽装といった失踪方法まで、インターネット時代のプライバシーや個人情報の守り方を、その道のプロが教える決定版。

提供元 河出書房新社 **URL** http://www.kawade.co.jp

新年号
特別付録!!

法輪寺鎮守社電電宮

情報安全護符シールVer.2.0の使い方



写真1　電電宮本殿

本誌2013年1月号で好評だった情報安全護符シールの第2段です。今年も京都府嵐山の虚空蔵法輪寺にご協力いただき、ご住職自ら開眼供養したお寺のお墨つき。大事なデータが入ったサーバ、PC、スマホなどいろいろなものに貼ってお使いください!

電電宮とは

13歳になった少年少女が虚空蔵菩薩に智恵を授かりに行く十三まいりで有名な虚空蔵法輪寺。その境内に、寺の鎮守のために建立された神社が「電電宮」です。ここには、大空(宇宙)の自然現象を広く包含する本尊虚空蔵大菩薩に関連した明星天子、雨宝童子、電電明神などが祀られています。中でも電電明神は雷、稲妻の神様で、今ふうに解釈すれば電気祖神、電波祖神と言えます。このことから長年、電力／電機／電波／電子各業界の事業者に、事業の成功や、電気／電波業界の発展を祈る場として参詣されています。

法輪寺住職 藤本高仝氏よりひとこと

昨今、情報通信の世界は目まぐるしい変化や著しい進歩が見られます。この中、設計や管理の仕事をされておられる方々はともすれば細部に目が行きがちと思われますが、大きな視野を持ち少しでも社会に貢献できるものを創造されることが神仏の意に適うことだと考えられます。どうか守護シールの御利益で、予想外のトラブルに見舞われることなく、順調に作業を進められ、少しでも世界に寄与できるよう頑張ってください。

虚空蔵 法輪寺　京都市西京区嵐山虚空蔵山町
http://www2.ocn.ne.jp/~horinji



写真2　シールのデザインデータが入ったCD-ROMを開眼供養し、そのデータで制作しました

シールの使用例

サーバ、PC、スマートフォンなどに貼ってお使いください。剥がしても跡が残りませんので、会社から支給されたPCにも安心して貼れます。





写真3　MacBook Proでの使用例。シールに描かれた梵字は、電電宮の本尊「虚空蔵大菩薩」を象徴する種字。大事なデータを不慮の事故から守ってくれますよ!

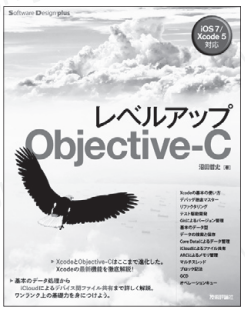
Software Design plus

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。

最新刊！



大谷 純、阿部 慎一朗、大須賀 稔、北野 太郎、鈴木 教嗣、平賀 一昭 著
㈱リクルートテクノロジーズ、㈱ロンウイット 監修
B5変形判・352ページ
定価 3,600円(本体)+税
ISBN 978-4-7741-6163-1



沼田 哲史 著
B5変形判・360ページ
定価 3,200円(本体)+税
ISBN 978-4-7741-6076-4



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



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



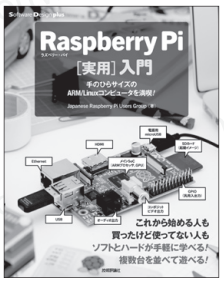
小飼 弾 著
A5判・200ページ
定価 2,080円(本体)+税
ISBN 978-4-7741-5664-4



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



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



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



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



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

JavaScriptライブラリ実践活用
WINGSプロジェクト 著
定価 2,580円+税　ISBN 978-4-7741-5611-8

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

日本一の地図システムの作り方
㈱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
定価 2,580円+税　ISBN 978-4-7741-5325-4

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

技術評論社　〒162-0846 東京都新宿区市谷左内町21-13
販売促進部　TEL 03-3513-6150 FAX 03-3513-6151

第1特集

シェルがわかればシステムがわかる
あなたの好きなシェルは何ですか？

Writer 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

sh、ash、dash、bash、ksh、csh、tcsh、zsh……

どういったシステムの構築や運用をしようとも、UNIX系OSの作業の要には常に「シェル」の存在があります。運用ツールはさまざまありますが、システムの状態を知り、リソースを明示的に活用するという点に関しては、カーネルと対話できるシェルでの操作はより多くのことを可能にしてくれます。

コマンドやパイプ、独特なスコープなど、シェルの基礎となる概念やしくみ、実装などを把握してシステムを意のままに操る術を身につけましょう。

CONTENTS

第1章 シェル言語 なぜシェルは生まれたのか 20
第2章 きっちり押さえるシェルの基本 シェルの4大要素① コマンド 32
第3章 きっちり押さえるシェルの基本 シェルの4大要素② リダイレクト／変数とパラメータ／パラメータ展開 39
第4章 ちゃんと知っていますか？ 意外と知らない「組み込みコマンド」の意味 49
第5章 もっと自由に！ 制御構文でプログラミング 55

COLUMN

» ソースコードで見えるカーネルとシェルの関係
» 東急ハンズや無印良品で活躍するシェルスクリプトの企業システム
» 常に求められるBSDライセンスでの実装とは
» 時短の極意、エイリアスを使い倒す
» /usr/bin/cdが存在する理由
» シェルスクリプト的発想ってどんなもの

記事執筆時の使用環境は、とくにことわりのない限り下記のものです。そのほかのシェルやインストール環境によっては実行結果が異なる可能性があることをご承知おきください。

OS：FreeBSD 9.2-RELEASE amd64
　　FreeBSD 10.0-BETA3 amd64
シェル：ashまたはdash

第1章 シェル言語
なぜシェルは生まれたのか

シェルには機能が豊富なインタラクティブ操作に向いたものと、処理が高速なシェルスクリプト向けのものがあります。本章ではさまざまなシェルの歴史や系譜から、それぞれがどういった用途に向いているのかを解説します。マルチコア/メニーコア時代の超高速開発に適したシェルの重要さを確認していきましょう。

シェルを再認識しよう!

UNIX系のオペレーティングシステム(OS)の管理や運用を任されている方であれば、シェルに触らない日はないのではないかと思います。シェルはUNIX系OSを管理するためのインターフェースであり、バッチ処理を実行するためのメカニズムを支えており、そうした必要性に応じて生まれたソフトウェアです。余談ですが、Windows Server系プラットフォームでもPower Shellから操作できる内容は増えており、シェルを経由したシステム管理への要求が高いことの表れでしょう。

シェルはUNIX系OSを人間が操作する場合の要となるソフトウェアです。シェルについて動作の理解を深め技術力を高めることは、そのままUNIX系OSの活用力の向上につながります。基礎となる概念やしくみ、実装などを把握し、システムを意のままに操る術を身につけましょう。

サーバ管理やシステム運用に欠かせないシェル

オンプレミスのサーバ、データセンターに設置されたサーバ、クラウドで提供される仮想サーバ環境、目の前のワークステーション(デスクトップ)、MacBookのターミナル、Windowsのターミナルアプリケーション……エンジニアはさまざまなレベルでシステムにログインして、なんらかの形でシェルを触っています。

ここ数年、サーバ管理の流行としてChefやPuppetといったシステム管理ツールを使って数十台から数百台、数千台といった規模のサーバを管理するやり方に注目が集まっています。こうしたシステムを構築する最初の段階で必要になるのはシェルであり、管理用のミドルウェアが動作するようにOSを動作させているのもシェルです。

サーバの管理では直接サーバのコンソールから作業することは希(まれ)です。たいてい、なんらかの方法でネットワークごしにログインする方法が提供されています。**ssh**(1)[注1]経由でログインするか、ssh(1)とVPS、これになんらかの認証方式を組み合わせてネットワーク経由でログインできる状態にしておき、それを使ってサーバにログインして運用上の管理を実施します(**図1**)。

システム開発の段階では頻繁にログインして作業をすることになりますが、いったんシステムが稼働をはじめたあとは、なんらかのワーニング(警告)が発生したときにログインして、モニタリングを実施したり問題を調査するといった作業に使われます。場合によっては、管理用のWebインターフェースを提供するソフトウェアを稼働させて管理の手間を自動化、または手引き書にしたがった操作ができるようにして管

注1) コマンドのうしろについている括弧書きの数字は、manコマンドで見ることのできるマニュアルに記載されている章番号を表しています。

理運用の担当者に任せるといった方法をとっているかもしれません。しかし、このレベルでの対応は調査できる内容に限界があります。対応できなくなればシステムにログインしてシェル経由で操作をすることになります。

また最近は、仮想サーバでシステムを構築することも増えました。こうした環境では「システムを修復する」といった発想ではなく、「壊れたら新しいインスタンスを起動する」という考え方でシステムを構築し運用します。つまり、ssh(1)でログインして修復するのではなく、システムがおかしくなったら不具合のあるインスタンスは捨てて、新しくインスタンスを作るといったやり方をします。この場合でも、最初のインスタンスを作る段階ではシェルを経由した作業が実施されます。

どういったシステムの構築や運用をしようとも、作業の要として常に「シェル」があります。

▼図1　ssh(1)経由でサーバにログインしての管理業務

```
daichi@pivot:/var/home/daichi (ssh)
last pid: 10439;  load averages:  0.11,  0.08,  0.08                    up 3+21:07:42  14:49:40
54 processes:  1 running, 53 sleeping
CPU:  0.0% user,  0.0% nice,  0.1% system,  0.0% interrupt, 99.9% idle
Mem: 13M Active, 620M Inact, 624M Wired, 819M Buf, 6606M Free
Swap: 24G Total, 24G Free

  PID USERNAME    THR PRI NICE   SIZE    RES STATE   C   TIME    WCPU COMMAND
10433 daichi        1  33    0 26856K  5332K pause   3   0:00   0.29% -zsh (zsh)
10439 daichi        1  22    0 19768K  2812K CPU3    3   0:00   0.29% top -a
 1031 www           1  20    0 38644K  8432K kqread  1   0:04   0.00% nginx: worker process (nginx)
  911 root          1  20    0 25328K  4364K select  2   0:04   0.00% /usr/sbin/ntpd -c /etc/ntp.conf -p
 1033 www           1  20    0 38644K  8460K kqread  0   0:04   0.00% nginx: worker process (nginx)
 1030 www           1  20    0 38644K  8324K kqread  1   0:04   0.00% nginx: worker process (nginx)
 1029 www           1  20    0 38644K  8448K kqread  0   0:04   0.00% nginx: worker process (nginx)
 1078 root          1  52    0 16520K  2140K nanslp  0   0:01   0.00% /usr/sbin/cron -s
 1011 root          1  20    0 33008K  4848K kqread  3   0:01   0.00% /usr/local/libexec/postfix/master
  824 bind          7  20    0 62400K 22924K kqread  0   0:01   0.00% /usr/local/sbin/named -t /var/name
  738 root          1  20    0 14432K  2012K select  2   0:01   0.00% /usr/sbin/syslogd -l /var/run/log
  922 root          1  20    0 25328K  4408K select  0   0:00   0.00% /usr/sbin/ntpd -c /etc/ntp.conf -p
 1013 postfix       1  20    0 33132K  4892K kqread  0   0:00   0.00% qmgr -l -t fifo -u
 6974 daichi        1  20    0 14744K  2392K select  0   0:00   0.00% /usr/local/bin/imapd /var/home/dai
 5466 root          1  20    0 12312K  1904K select  1   0:00   0.00% /usr/local/libexec/courier-imap/co
 5465 root          1  20    0 12268K  1904K piperd  1   0:00   0.00% [courierlogger]
 1021 postfix       1  20    0 33076K  5600K kqread  1   0:00   0.00% tlsmgr -l -t unix -u
 6973 root          1  20    0 18832K  4392K select  1   0:00   0.00% /usr/local/bin/couriertls -server
 6978 daichi        1  20    0 20908K  2360K select  3   0:00   0.00% /usr/local/bin/imapd /var/home/dai
 6980 daichi        1  20    0 20908K  2392K select  1   0:00   0.00% /usr/local/bin/imapd /var/home/dai
 1039 www           1  20    0 14372K  1896K accept  3   0:00   0.00% /usr/local/sbin/fcgiwrap -s unix:/
 5448 root          1  20    0 22796K  2248K select  0   0:00   0.00% /usr/local/libexec/courier-authlib
 5446 root          1  20    0 22796K  2248K select  3   0:00   0.00% /usr/local/libexec/courier-authlib
 5444 root          1  20    0 22796K  2248K select  1   0:00   0.00% /usr/local/libexec/courier-authlib
 5447 root          1  20    0 22796K  2248K select  1   0:00   0.00% /usr/local/libexec/courier-authlib
 5445 root          1  20    0 22796K  2248K select  0   0:00   0.00% /usr/local/libexec/courier-authlib
 6976 daichi        1  20    0 20908K  2364K select  1   0:00   0.00% /usr/local/bin/imapd /var/home/dai
PIVOT /var/home/daichi% _
```

シェルとは

シェルはOSを操作する特徴を持ったソフトウェアです。人間が対話的に操作してカーネルの提供している機能を利用したり、ほかのソフトウェアを実行するために使われます。中間的な機能を加えることが少なく、ダイレクトにカーネルの機能を呼び出す傾向があります。またシェルは、システムそのものを組み上げるソフトウェアとしての役割を担っています。

UNIX系OSではブートローダがカーネルを起動し、カーネルがファイルシステムを認識すると、次にシステム起動用のプロセスを起動します。このプロセスはシェルを起動して、システムの起動処理をすべて任せます。起動するやり方には何種類かあるのですが、典型的にはシェルスクリプトで構成された起動スクリプトを設定にしたがって順次実行するといったやり方がとられます(詳しくは後述)。

システムの起動が完了するとシステムはユーザのログインを待ちます。ユーザのログインは、ユーザの認証がとれたら、そのユーザ権限でシェルを起動する処理になります。ユーザはシェルを通じてほかのソフトウェアを実行したり、カーネルが提供する機能を使用します。この段階で使用されるシェルは、人間が操作しやすいようにさまざまな機能が追加されたシェルが使われる傾向があります。最初の起動したシェルが終了すると、それはログアウトということになります。

このようにシェルは2つの側面を持っています。ユーザが対話的に操作するときのシェルは「インタラクティブシェル」、システムを組み上げるために使われるソフトウェアとしての側面が強い場合には、シェルの操作をまとめたファイルを含めて「シェルスクリプト」と呼ばれます。

知るべき、シェルあれこれ

シェルにはいくつかの実装があります。ここ

では代表的なシェルをとりあげて紹介します。

sh / ash / dash

現在のUNIX系OSのシステムを構築しているシェルの源流にあたるシェルが「sh(Bourne Shell)」です。現在では/bin/shとしてBourne Shellと互換性のあるシェルがデプロイされていることがほとんどです。シェルスクリプトの基本となる機能はBourne Shellで実現されています。

Bourne Shellの互換実装はいくつかありますが、もっともオリジナルのBourne Shellに近く、現在でも広く使用されている実装がKenneth Almquist氏によって実装された「ash」です。現在のashはIEEE Std 1003.1("POSIX.1")のシェルの仕様を実装したものに、いくつかのバークレー機能拡張を追加したものになっています。

現存するシェル実装の中でもashはサイズが小さく軽量で、メモリの消費量が少なく動作も高速という特徴があります。システムを組み上げるためのシェルスクリプトとして使われることが多いほか、rootユーザのシェルとしても使われます。たとえば、FreeBSDの起動ファイルである/etc/rcの先頭部分を見てみると、**図2**のとおり確かにシェルスクリプトであることがわかります。

ashはFreeBSDやNetBSDなどのOSで採用されていますが、Linuxにも移植されていて、移植された実装は「dash」と呼ばれています。高速に動作するためシステムのシェルスクリプトに使われていて、DebianやUbuntuなどいくつかのLinuxディストリビューションではdashが/bin/shとして使用されています。**図3**を見ていただくとわかるとおり、/bin/shのシンボリックリンクとしてdashが指定されています。

bash

インタラクティブシェルとして広く使用されているシェルが「bash(GNU Bourne-Again SHell)」です(**図4**)。Bourne Shell互換の多機能シェルで、Mac OS Xや多くのLinuxディストリビューションでデフォルトのシェルとして

▼図2　システムスクリプトは/bin/shが使われる

```
$ head /etc/rc
#!/bin/sh      ←シェルスクリプトの開始宣言
#
# Copyright (c) 2000-2004  The FreeBSD Project
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
# 1. Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
$
```

▼図3　Linux(例はUbuntu)に移植されたash「dash」

```
daichi@ubuntu:~$ uname -a
Linux ubuntu 3.5.0-17-generic #28-Ubuntu SMP Tue Oct 9 19:32:08 UTC 2012 i686 i686 i686 ➲
GNU/Linux                              ↑実行環境
daichi@ubuntu:~$ ls -il /bin/sh
2621591 lrwxrwxrwx 1 root root 4  2月 26  2013 /bin/sh -> dash     ←シンボリックリンク
daichi@ubuntu:~$ ls -il /bin/bash
2621447 -rwxr-xr-x 1 root root 920796  9月 19  2012 /bin/bash     ←bashの存在を確認
daichi@ubuntu:~$ ls -il /bin/dash
2621473 -rwxr-xr-x 1 root root 96232  8月 15  2012 /bin/dash     ←bashとは別にあることが確認できた
daichi@ubuntu:~$
```

採用されています。シンタックスはBourne Shellに準拠していますが、cshの機能を取り込むなど、ほかのシェルが実現した便利な機能を取り込んでいます。

Bourne Shellと互換性があるためインタラクティブシェルとしてのみならず、シェルスクリプトとしても使われています。Linuxディストリビューションの多く、そしてMac OS Xはbashをシェルスクリプトのシェルとして利用しています(**図5**)。さまざまな拡張機能を持ち、現在もっともユーザが多いシェルの1つです。**図4**の例では、shには存在しない「ブレース展開」という拡張機能を使ってみています。

ashやdashと比較するとbashの動作は重めです。サイズも大きくほかのシェルと比べて重量感があります。また、bashで動作するように組まれたシェルスクリプトはashやdashでは利用できない機能を使っていることがあり、移植性を重視するケースでは利用を懸念される向きもあります。

ksh

Bourne Shellと後述するC shellの機能を融合させたようなシェルに「ksh(KornShell)」があります(**図6**)。kshもBourne Shellと互換性があり、とくに商用UNIXでデフォルトのシェルとして活用されています。オープンソースソフトウェアのOSではOpenBSDがkshをベー

▼図4　LinuxやMac OS Xでデフォルトのシェルbash

```
daichi@ubuntu:~$ uname -a
Linux ubuntu 3.5.0-17-generic #28-Ubuntu SMP Tue Oct 9 19:32:08 UTC 2012 i686 i686 i686
GNU/Linux
daichi@ubuntu:~$ cd tmp
daichi@ubuntu:~/tmp$ ls -al      ←↓/tmpの中身が空であることを確認
合計 8
drwxrwxr-x  2 daichi daichi 4096 11月 19 12:58 .
drwxr-xr-x 27 daichi daichi 4096 11月 19 12:56 ..
daichi@ubuntu:~/tmp$ touch {1..9}{a..c}{A,B,D}     ←shには存在しない拡張機能「ブレース展開」を実行
daichi@ubuntu:~/tmp$ ls      ←↓実行結果
1aA  1cA  2bA  3aA  3cA  4bA  5aA  5cA  6bA  7aA  7cA  8bA  9aA  9cA
1aB  1cB  2bB  3aB  3cB  4bB  5aB  5cB  6bB  7aB  7cB  8bB  9aB  9cB
1aD  1cD  2bD  3aD  3cD  4bD  5aD  5cD  6bD  7aD  7cD  8bD  9aD  9cD
1bA  2aA  2cA  3bA  4aA  4cA  5bA  6aA  6cA  7bA  8aA  8cA  9bA
1bB  2aB  2cB  3bB  4aB  4cB  5bB  6aB  6cB  7bB  8aB  8cB  9bB
1bD  2aD  2cD  3bD  4aD  4cD  5bD  6aD  6cD  7bD  8aD  8cD  9bD
daichi@ubuntu:~/tmp$
```

▼図5　Mac OS Xの/bin/shはbash

```
/Users/daichi% uname -a
Darwin Daichi-no-MacBook-Air.local 13.0.0 Darwin Kernel Version 13.0.0: Thu Sep 19 22:22:27
PDT 2013; root:xnu-2422.1.72~6/RELEASE_X86_64 x86_64
/Users/daichi% ls -il /bin/ | grep sh
494113 -r-xr-xr-x  1 root  wheel  1228240 10 24 09:39 bash
816492 -rwxr-xr-x  2 root  wheel   357984 10 24 09:49 csh
773603 -r-xr-xr-x  1 root  wheel  1315248 10 24 09:48 ksh
494114 -r-xr-xr-x  1 root  wheel  1228304 10 24 09:39 sh
816492 -rwxr-xr-x  2 root  wheel   357984 10 24 09:49 tcsh
817120 -rwxr-xr-x  1 root  wheel   530320 10 24 09:49 zsh
/Users/daichi% bash --version
GNU bash, version 3.2.51(1)-release (x86_64-apple-darwin13)
Copyright (C) 2007 Free Software Foundation, Inc.
/Users/daichi% sh --version      ←この実行結果と上のbashの結果を見比べると、shにbashを使っていることがわかる
GNU bash, version 3.2.51(1)-release (x86_64-apple-darwin13)
Copyright (C) 2007 Free Software Foundation, Inc.
/Users/daichi%
```

スとしたシェルをデフォルトに採用しています。

kshは後に登場した多機能インタラクティブシェル(tcsh、bash、zshなど)と比べると軽量で高速です。シェルスクリプトを動作させるのに必要な機能は実装しつつ、インタラクティブシェルとして機能するために必要になる機能をコンパクトにまとめあげた作りになっており、シェルスクリプトにもインタラクティブシェルにも利用できるシェルです。

kshは命令形がほかのシェルとちょっと違います。**図6**ではkshにしかない機能を紹介してみました。

▼図6　商用UNIXでデフォルトシェルに設定されることが多かったksh

```
$ cat<<EOF
> here documents
> EOF
here documents
$
$ cat<<<inline_here_documents
inline_here_documents
$
$ echo a=b c
a=b c
$ set -k
$ echo a=b c
c
$ getconf ARG_MAX
262144
$ getconf PATH
/usr/bin:/bin:/usr/sbin:/sbin
$
```

csh

Bourne ShellとともにUNIX系OSに強い影響を与えたシェルが「csh(Berkeley UNIX C Shell)」です(**図7**)。Bourne Shellとは異なるシンタックスを採用しています。開発された当時はOSの開発言語であるC言語に似せた文法を実現すること、それにインタラクティブシェルとして機能するために便利な機能を実装することを目指していました。C言語に似せたため、sh系シェルとは文法が異なります。たとえば**図7**の**foreach**や**switch**コマンドは、sh系だとそれぞれ**for**、**case**です(forやcaseについては第5章参照)。

cshの最大の特徴はコマンド履歴、ジョブコントロール、インタラクティブなパス名補完などの機能を実装した点にあります。これら機能は後続することになるksh、tcsh、bash、zshなどに強い影響を与えました。現在おもに使わ

▼図7　BSD系OSで採用されることが多かったcsh。sh系シェルとはシンタックスが大きく異なる

```
# set prompt = "% "
% foreach i (0 1 2 3)
? echo smartctl -a /dev/ada$i
? end
smartctl -a /dev/ada0
smartctl -a /dev/ada1
smartctl -a /dev/ada2
smartctl -a /dev/ada3
% switch ("a")
? case "a"
%     echo a
a
%     breaksw
? case "b"
?     echo b
?     breaksw
? default:
?     echo default
?     breaksw
? endsw
% time true
0.0u 0.0s 0:00.00 0.0% 0+0k 0+0io 0pf+0w
% set      ←設定もshとは大きく異なる
argv    ()
autoexpand
autolist        ambiguous
autorehash
cwd     /home/csh
filec
history 1000
home    /home/csh
i       3
mail    /var/mail/csh
path    (/sbin /bin /usr/sbin /usr/bin
/usr/games /usr/local/sbin /usr/local/
bin /home/csh/bin)
prompt  %
prompt2 ?
promptchars     %#
savehist        (1000 merge)
shell   /bin/csh
status  0
term    xterm
user    csh
%
```

れているインタラクティブシェルはBourne Shellの文法をベースにcshのインタラクティブ系機能を取り込んだもの、と説明できます。

登場当時は革新的なシェルでしたが、現在の多機能シェルと比べるとインタラクティブ機能は貧弱です。また、文法そのものが言語的な矛盾を抱えているほか、パーサの実装などに問題があり、想定しているようにはハンドリングできないシーンがいくつか存在します。cshでシェルスクリプトが組まれることもありますが、Bourne Shell系のシェルで組まれる割合と比べるととても少ないといえます。とくにファイルディスクリプタのハンドリングに難があり、シェルスクリプト向きではありません。

csh系の文法を好む場合でも、cshが現在でも使われているケースは希です。csh系を好む場合には次に解説するtcshというインタラクティブシェルが使われています。

tcsh

cshと互換性を持たせながら多機能化を進めたシェルが「tcsh」です。**図8**の**set**コマンドによる設定の出力をcshと比較すると、両者の結びつきが感じられるでしょう。コマンドライン機能、プログラマブルな補完機能、スペル補正機能、ジョブ制御機能などcshには存在しなかったいくつもの機能が追加されています。

csh系のシェルに慣れている場合には、tcshが現在のところ唯一に近い選択肢といえます。

zsh

Bourne Shell系の文法を採用しつつ、bashやtcsh、kshなどほかのシェルが実現した機能を積極的に取り込んだ多機能シェルが「zsh」です。現在採用されている多機能インタラクティブシェルとしては最後発のシェルとなります。多種多様な機能を提供するとともに、カスタマイズ性がとても高いという特徴があります。インタラクティブシェルとしての利用をきわめようとした場合、最後に到達する選択肢の1つがzshです。

bashとzshはよく似た機能を提供していますが、いくつかの点で異なる特性を持っています。とくにzshはコマンド履歴機能が強力です。zshでは複数行に渡るコマンドを改行データを

▼図8　cshのインタラクティブ性能を拡張したtcsh

```
tcsh@parancell:~ % set
_       cd

addsuffix
anyerror
argv    ()
autoexpand
autolist        ambiguous
autorehash
csubstnonl
cwd     /home/tcsh
dirstack        /home/tcsh
echo_style      bsd
edit
euid    1006
euser   tcsh
fignore (.so .a)
filec
gid     1006
group   tcsh
history 1000
home    /home/tcsh
killring        30
local   /usr/local
loginsh
mail    /var/mail/tcsh
owd     /usr/local/lib
path    (/sbin /bin /usr/sbin /usr/bin
/usr/games /usr/local/sbin /usr/local/bin
/home/tcsh/bin)
prompt  %N@%m:%~ %#
prompt2 %R?
prompt3 CORRECT>%R (y|n|e|a)?
promptchars     %#
savehist        (1000 merge)
shell   /bin/tcsh
shlvl   1
status  0
tcsh    6.18.01
term    xterm
tty     pts/6
uid     1006
user    tcsh
version tcsh 6.18.01 (Astron) 2012-02-14
(x86_64-amd-FreeBSD) options wide,nls,dl,
al,kan,sm,rh,color,filec
tcsh@parancell:~ %
```

保持したまま保存します。bashではコマンドはワンライナーにまとめた状態で履歴として保持します。zshでは改行情報を保存するため、**図9**のように複数行に展開された状態で履歴編集ができます。

シェル歴史と進化の流れ ~bashやzsh誕生の必然性

シェルはUNIX系OS操作の要です。人間の操作性を向上させる機能を実装したbashやzshといったシェルが登場したのは当然の要求だったといえます。あとから登場したシェルはそれまでに存在しているシェルの影響を受けていたり、より軽量な実装を実現しようという多機能へのアンチテーゼ的な取り組みになっています。

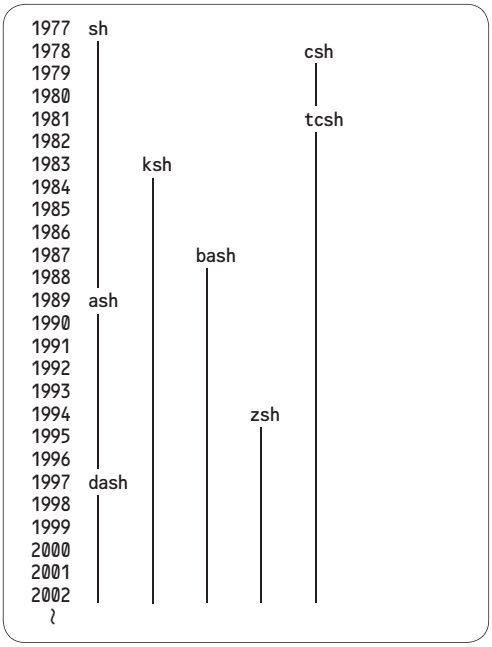
入手可能な資料に開発開始の年代の確認がとれないものがあるので正確な値とはなりませんが、ソースコードなどに含まれている年号などをまとめると、だいたい**表1**の年代にこれらシェルが登場したことになります。

図10には系譜をまとめてみました。それぞれのシェルの関係はsh、ash、dashが直系に近い関係にあります。cshとtcshも直系的な関係

▼表1　主要シェルの登場時期

登場した年	シェル
1977年	sh
1978年	csh
1981年[注2]	tcsh
1983年	ksh
1987年	bash
1989年	ash
1994年[注3]	zsh
1997年	dash

▼図10　登場時期と系譜



▼図9　とても強力な履歴機能を備えた多機能インタラクティブシェルzsh

```
PARANCELL /z/daichi% for i in 0 1 2 3
do
    sudo smartctl -a /dev/ada$i | grep ^SMART | grep health
done
SMART overall-health self-assessment test result: PASSED
SMART overall-health self-assessment test result: PASSED
SMART overall-health self-assessment test result: PASSED
SMART overall-health self-assessment test result: PASSED
PARANCELL /z/daichi% history -1
 3004  for i in 0 1 2 3\ndo\n    sudo smartctl -a /dev/ada$i | grep ^SMART | grep health\n
done                          ↑改行コード(\n)付きで保存されていることがわかる
PARANCELL /z/daichi%      ここでコマンド履歴のショートカットを押している
PARANCELL /z/daichi% for i in 0 1 2 3
do
    sudo smartctl -a /dev/ada$i | grep ^SMART | grep health
done
```

注2） Ken Greer氏が開発したパス名補完機能がcshにマージされた年をtcshの発祥と考えた場合。
注3） 履歴から確認できるのは1994年にリリースされたバージョン2.6-beta1まで。

といえそうです。それぞれ登場時期よりも前のシェルからなんらかの影響を受けています。

現在、インタラクティブシェルとしての機能はほぼ出そろった感じがあります。シェルスクリプトとして動作するのに必要になる機能も1980年代にはすでに固まっています。

カーネルとシェルの関係をご存じですか

OSではソフトウェアが問題なく動作するように、特定の処理はカーネルのみが実行できるしくみを採用しています。ユーザランドで動作する普通のソフトウェアには実行させないことで、安全に処理が実行されるようになっています。

ユーザランドで動作するソフトウェアは、カーネルで処理すべき状況になるとシステムコールを呼び出します。システムコールが発行されると処理がカーネルに移ります。カーネル内部で処理をして、結果をユーザランドへ戻します。

通常、ソフトウェアが直接システムコールを叩くような実装はあまりとられません。たいていのソフトウェアは処理を実現するためにいくつものライブラリを使用し、ライブラリの一部がシステムコールを呼んで処理を実行するといった作りになっています。それ以外の処理はユーザランドレベルで処理しています。

シェルはそのあたりの処理が普通のソフトウェアと大きく違います。シェルはカーネルの提供する機能をダイレクトに利用します。カーネルの提供する機能を積極的に活用するタイプのソフトウェアです。シェルのソースコードを読むとそのあたりがよくわかります。詳しくはコラムを参照ください。

今、なぜ企業でシェルが求められているのか

シェルはサーバやシステムの管理や運用に活用されていますが、この数年で企業におけるシステム開発に要求されるプログラミング言語としても採用されはじめています。理由の1つは、企業の意思決定からそれを情報システムに落とし込むまでの期間が短くなっており、従来のウォーターフォール型の開発ではシステム構築が困難なシーンが増えているためです。シェルを採用した企業システムはいわゆるアジャイル開発、最近の言葉でいえば「超高速開発」に分類される手法に入ります。

シェルないしはカーネルが提供するパイプ(**pipe**(2))の機能を使用するとストリーム型のコマンドを結びつけて、テキストデータを並列に処理させることができます。企業システムで要求されるデータ処理などはテキストの加工や集計処理で表現できることが多く、この点UNIX系OSが提供しているコマンドの組み合わせで処理するという方法と親和性が高いと評価されています。コマンドをパイプでつなげるだけの開発なので学習コストが低く、企業の内製化を実現しやすいという特徴もあります。

現在のマシンはマルチコア／メニーコア化が進んでいます。コマンドをパイプで組み合わせてテキストデータを処理する方法はこうしたマシンの性能を発揮させやすく、既存のシステムの高速化などにも寄与します。既存の開発方法と方法論そのものが違っているので導入している企業は比較的アグレッシブな企業に限定されていますが、この流れは今後も広がることが予測されます(**図11**)。

▼図11　個々の企業がそれぞれの手法で高速開発を実施している超高速開発コミュニティの発足



COLUMN ソースコードで見えるカーネルとシェルの関係

本文「カーネルとシェルの関係をご存じですか」で説明したカーネルとシェルの関係をソースコードから眺めてみましょう。ashまたはdashがとっかかりとしてはよいでしょう。たとえばシェルの提供している入出力機能であるリダイレクトの実装を追ってみましょう（**リストA**）。リダイレクトの処理はashのソースコードに含まれているnodetypesというファイルを見ると、その種類を確認できます。

▼リストA　ashの実装から：ファイルリダイレクトに関するリスト（nodetypesより抜粋）

```
NTO       nfile    # fd>fname
NFROM     nfile    # fd<fname
NFROMTO   nfile    # fd<>fname
NAPPEND   nfile    # fd>>fname
NCLOBBER  nfile    1# fd>|fname
NTOFD     ndup     # fd<&dupfd
NFROMFD   ndup     # fd>&dupfd
NHERE     nhere    # fd<<\!
NXHERE    nhere    # fd<<!
```

標準出力のリダイレクト**>**はよく使われる機能ですが、この機能にはコーディング上のラベルとしてNTOという名前がつけられています。該当する部分のソースコードはredir.cファイルにあります。該当部分の抜粋が**リストB**です。

オプションへの対応やエラーチェックなどで込み入ったように見えますが、あまり使われないオプション機能を取り除いて、かつ、エラーチェックなども抜いて処理を簡潔にすると**リストC**のようになります。open(2)、dup2(2)、close(2)のどれもシステムコールです。シェルで**>**によるリダイレクト処理を記述することは、このようにシステムコールを呼んでカーネルに処理を委託していることと同じです。

▼リストC　標準出力のリダイレクト処理

```
case NTO:
        f = open(fname, O_WRONLY, 0666);
        dup2(f, fd);
        close(f);
```

▼リストB　ashの実装から：リダイレクトを処理している部分（redir.cより抜粋）

```
case NTO:
        if (Cflag) {
                fname = redir->nfile.expfname;
                if (stat(fname, &sb) == -1) {
                        if ((f = open(fname, O_WRONLY|O_CREAT|O_EXCL, 0666)) < 0)
                                error("cannot create %s: %s", fname, strerror(errno));
                } else if (!S_ISREG(sb.st_mode)) {
                        if ((f = open(fname, O_WRONLY, 0666)) < 0)
                                error("cannot create %s: %s", fname, strerror(errno));
                        if (fstat(f, &sb) != -1 && S_ISREG(sb.st_mode)) {
                                close(f);
                                error("cannot create %s: %s", fname,
                                    strerror(EEXIST));
                        }
                } else
                        error("cannot create %s: %s", fname,
                            strerror(EEXIST));
                goto movefd;
        }
...
movefd:
               if (f != fd) {
                       if (dup2(f, fd) == -1) {
                               e = errno;
                               close(f);
                               error("%d: %s", fd, strerror(e));
                       }
                       close(f);
               }
               break;
```

OSを駆動させるシェルスクリプト

シェルはシステムをシステムとして動作させるためのスクリプトとしても機能するということは、これまでに何度か書きました。UNIX系OSはカーネルをローディングしたあとはシェルスクリプトにシステムの起動処理をゆだねます。システムの起動の方法はいくつか種類がありますが、ここではシンプルな例としてFreeBSD 9の起動システムを見てみましょう。

/etc/rcというファイルが起動スクリプトです。さまざまな判定処理が加わっていますが、本質的な部分だけを抜き出すと**リスト1**のようになります。

FreeBSDの起動メカニズムは**rc**(8)と呼ばれています。/etc/rc.d/に"1サービス1スクリプトファイル"の形式で起動、停止、再起動、状態報告などの処理を実施するファイルがまとまっています(**図12**)。

/etc/rcスクリプトの中で実行されている**rcorder**(8)というコマンドは、/etc/rc.d/以下のスクリプトファイルの起動順序を調査して、

▼リスト1　システム起動スクリプト/etc/rcの本質部分

```
#!/bin/sh

HOME=/
PATH=/sbin:/bin:/usr/sbin:/usr/bin
export HOME PATH

. /etc/rc.subr
load_rc_config 'XXX'

files=`rcorder /etc/rc.d/* 2>/dev/null`

for _rc_elem in ${files}; do
        run_rc_script ${_rc_elem} ${_boot}
done

exit 0
```

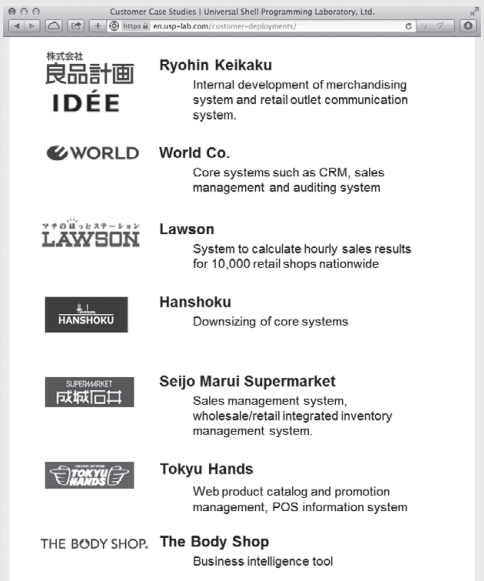
COLUMN　東急ハンズや無印良品で活躍するシェルスクリプトの企業システム

シェルスクリプトで企業システムを構築する企業はいくつかありますが、中でも注目度の高い企業にユニバーサル・シェル・プログラミング研究所を紹介できます。ユニバーサル・シェル・プログラミング研究所はシェルスクリプトを使って企業システムを開発する研究開発を実施するとともに、発足から本日まで良品計画、ローソン、成城石井、東急ハンズ、ロッテリア、三省堂書店などさまざまな企業のシステム開発にシェルで取り組んでいます(**図A**)。

ユニバーサル・シェル・プログラミング研究所では同社の開発手法を「ユニケージ開発手法」としてまとめ、考え方から開発のマナー、教育事業の提供も含めて包括的に提供しています。ユニケージ開発手法を採用したシステム開発では、最終的に企業が自社システムを内製する傾向が強いという点もポイントとして紹介できます。学習が容易ですぐに開発に適用でき、企業内での開発体制を構築しやすいという特徴があります。

ユニケージ開発手法に関してはUECというサイト(usp engineers' community site URL https://uec.usp-lab.com/)にも情報がまとまっていますので、興味がある方はこちらをご覧ください。

▼図A　さまざまな企業の勘定系システム、データ分析系システム、基幹システムなどをシェルスクリプトで開発している企業ユニバーサル・シェル・プログラミング研究所



リストで報告するというコマンドです。/etc/rc.d/以下のスクリプトファイルには「ファイルシステムがマウントされたあとで、○○というサービスが走っていることが起動の条件である」といったようにそれぞれ起動の条件が記載されています。rcorder(8)はこれらをすべてチェックして、スクリプトを実行する順序を算出し、結果を報告します。この処理だけ実行してみると、**図13**のような結果が得られます。

たとえばかなり初期の段階で/etc/rc.d/swapというシェルスクリプトが実行されていますが、このシェルスクリプトは**リスト2**のようになっています。このシェルスクリプトは**swapon**(8)というコマンドを実行してスワップ機能を有効にしています。このようにシステム起動時に必要になる処理(コマンドの実行)をシェルスクリプト経由で実行していくことで、OSとして稼働できる体になるようにしくみを構築しています。

方式の違いはありますが、UNIX系のOSはこのようにシェルスクリプトで起動処理を実施しています(専用のプログラムを開発してもっとタイトに起動しているシステムもあります)。/etc/rc.d/以下のシェルスクリプトは個別のサービスの制御スクリプトにもなっています。

▼リスト2　スクリプトの例:スワップ機能を有効化する/etc/rc.d/swap

```
#!/bin/sh
#
# $FreeBSD: stable/10/etc/rc.d/swap ➤
252310 2013-06-27 18:28:45Z hrs $
#

# PROVIDE: swap
# REQUIRE: disks
# KEYWORD: nojail shutdown

. /etc/rc.subr

name="swap"
start_cmd='/sbin/swapon -aq'
stop_cmd=':'

load_rc_config $name
run_rc_command "$1"
```

▼図12　/etc/rcから起動される個別のスクリプト

```
% ls /etc/rc.d/ | xargs -L4 echo | keta -- | head
DAEMON          FILESYSTEMS     LOGIN           NETWORKING
SERVERS         abi             accounting      addswap
adjkerntz       amd             apm             apmd
archdep         atm1            atm2            atm3
auditd          auditdistd      bgfsck          bluetooth
bootparams      bridge          bsnmpd          bthidd
ccd             cleanvar        cleartmp        cron
ctld            ddb             defaultroute    devd
devfs           dhclient        dmesg           dumpon
faith           fsck            ftp-proxy       ftpd
%
```

適材適所でシェルを選ぶ

現在おもに採用されているシェルの実装にはash/dash、ksh、tcsh、bash、zshなど複数の実装があることを紹介しました。ここではそれぞれどのようなシーンでどういったシェルを採用すればよいか、その指針を示します。

シェルスクリプト

シェルスクリプトを組む場合は、ashまたはdashがよいでしょう。kshも十分に要件を満た

▼図13　/etc/rcから起動されるスクリプトを順序ごとに出力(一部)

```
% rcorder /etc/rc.d/* | head
/etc/rc.d/sysctl
/etc/rc.d/hostid
/etc/rc.d/zvol
/etc/rc.d/dumpon
/etc/rc.d/ddb
/etc/rc.d/initrandom
/etc/rc.d/geli
/etc/rc.d/gbde
/etc/rc.d/ccd
/etc/rc.d/swap
% rcorder /etc/rc.d/* | tail
/etc/rc.d/inetd
/etc/rc.d/hostapd
/etc/rc.d/gptboot
/etc/rc.d/geli2
/etc/rc.d/ftpd
/etc/rc.d/ftp-proxy
/etc/rc.d/dhclient
/etc/rc.d/bsnmpd
/etc/rc.d/bluetooth
/etc/rc.d/bgfsck
%
```

すのですが、インストールされていない環境があります。ashまたはdash相当の機能を実装したシェルはUNIX系OSであればたいていインストールされています。パスも/bin/shで統一されていて扱いやすいところがあります。

ashとdashは必要最小限の機能しか実装していませんので、処理が高速で軽量という利点があるほか、移植性が高いという特徴があります。bashをベースにシェルスクリプトを組むと、Linux以外のOSにシェルスクリプトを移植する際に手間がかかります。シェルスクリプトが数十程度であれば手動の書き換えでも追いつくでしょうが、対象が数千ファイルということになってくると、できればそういった事態は避けたいところです。

移植性という点では、シェルスクリプト内部で利用するコマンドに関しても注意が必要です。同じ名前のコマンドでもOSごとに採用している実装が異なるものがあります。**awk**(1)、**sed**(1)、**date**(1)、**sort**(1)などよく使われるコマンドの実装はOSごとにばらばらです。可能な限りPOSIXで定められている機能で実装するようにしておくと、OSが変わっても修正なしで動作するシェルスクリプトができます。拡張機能を利用する必要がある場合には、その旨をどこかに書いておくか、コマンド名そのものを変えておくというやり方もあります(GNU sed(1)が必要な場合はgsedと書くなど)。

インタラクティブシェル

これからインタラクティブシェルを選ぶのであれば、bashかzshを選択しておけばよいでしょう。シェルスクリプトとインタラクティブシェルで同じシンタックスを扱ったほうが都合がよいことが多いので、とくに強い理由がない限りは今後はtcshを利用することの利点は低いといえます。

個人的には機能が豊富でカスタマイズも容易、さらに履歴機能が強力という点からインタラクティブシェルとしてはzshの採用をお勧めします。インタラクティブシェルはエンジニアの生産性に大きく影響を与えます。自らが必要とする機能をカスタマイズできるzshはなにかと便利です。SD

COLUMN 常に求められるBSDライセンスでの実装とは

オープンソースムーブメントの隆起とともにライセンスにGPLを採用したソフトウェアはその数を一気に増やしました。こうした動きがオープンソースソフトウェアを根付かせる要因になったことは間違いのないところです。

しかし、GPLでは都合がわるいという産業もあります。高性能アプライアンスや組み込み機器、ゲーム機器、高機能家電製品など、プロダクトとして多機能さを要求されながらも、ソースコードを公開することが困難な分野です。こうした分野では差別化がビジネス上重要になります。他社が実装していない機能をいち早く実装し、差別化を図って製品価値を引き上げることが競争力につながります。

このためBSDライセンスやMITライセンスといったライセンスでのソフトウェアも常に求められてきました。これらのソフトウェアは利用してもソースコードを公開する必要がないので、そういった製品で採用しやすいためです。最近出荷されたプロダクトではソニーのPlayStation 4がカーネルなどにFreeBSDカーネルなどを採用しています。

多機能のインタラクティブシェルとしてzshは、BSDライセンスを採用しているというところもポイントです。組み込みベンダーやアプライアンスベンダーはライセンス問題を懸念してBSDライセンスやMITライセンスのソフトウェアのみでプロダクトを構築する傾向がありますが、そういった場合にzshは採用しやすいシェルの1つとなります。

第2章 きっちり押さえるシェルの基本 シェルの4大要素① コマンド

シェルを4つの側面、「コマンド」「リダイレクト」「変数とパラメータ」「パラメータ展開」からとらえてみましょう。本章ではまず「コマンド」を取り上げます。コマンドと言ってもいくつかの種類があります。ファイルとして存在するソフトウェアや、最初からシェル内部で定義されているもの、あとから定義されたもの、コマンドを別名で呼び出すものなどがあります。

動作環境の想定

シェルはプログラミング言語という側面でとらえるとすれば、とてもシンプルな言語です。しかし、なんらかのやり方に沿って、ある程度抽象化された高級言語と異なり、シェルの動作はカーネルとのインタラクションという点を基本としており、既存のプログラミング言語とは概念が違っているところがあります。こうした基本点を押さえることで、シェルをもっと見通しよく理解できるようになります。

シェルとしてはashまたはdashを想定します。第1プロンプトには「% 」を使います。コマンドの入力時点では第1プロンプトが表示されています。第2プロンプトには「> 」を使います。制御構文を使う場合など2行以上の入力が必要になる場合にはこの第2プロンプトが表示されます。

本特集で取り上げる機能は、Bourne Shell互換のシェルであればそのまま利用できる機能がほとんどです。

コマンド

シェルで処理の主体となるのはコマンドです。コマンドはファイルシステム上に配置されたソフトウェアであることもあれば、シェルが最初から内包しているコマンド、あとからシェル内部で定義されたコマンド、または別のコマンドへの別名であるなどいくつか種類があります。ここではそれぞれを「コマンド」「シェル関数」「組み込みコマンド」「エイリアス」として解説します。

ファイルシステム上のソフトウェアを実行するために、それを呼び出すための文字列をコマンドと呼びます。たとえば次の例は、現在の日時を表示するdate(1)というコマンドを実行した場合の出力結果です。

```
% date
Tue Nov 19 15:39:23 JST 2013
%
```

dateという文字列が入力されると、シェルは指定されたディレクトリにdateという名前のソフトウェアが存在するかどうかチェックにいきます。ここでは/bin/dateというファイルとして存在していることがわかります(**図1**)。

シェルはこのプログラムを呼び出して実行し、結果を出力します。シェルが実施する機能の本質はこの作業です。

シェル関数

シェル内部で定義できるコマンドのような処理単位を「シェル関数」と呼びます。次のように記述した場合、「jstdate() {」という行から「}」という行までがシェル関数となります。シェル関数はコマンドのように呼び出して実行できます。

▼図1 date(1)コマンドは/bin/dateというファイルを実行したもの

```
% which date
/bin/date
% ls -il /bin/date
495220 -rwxr-xr-x  1 root  wheel  24496 10 24 09:39 /bin/date
%
```

```
% jstdate() {
> LANG=ja_JP.UTF-8 date
> }
%
% date
Tue Nov 19 15:41:31 JST 2013
% jstdate
2013年 11月19日 火曜日 15時43分40秒 JST
%
```

シェル関数の実態は、実行中のシェルのメモリの中に保持されている文字列です。シェル関数はシェルが終了すると消えます。永続的に存在してほしい場合にはシェルスクリプトに書き出す必要があります。

組み込みコマンド

シェルが最初から実装しているシェル関数のようなものを「組み込みコマンド」と呼びます。これはシェル関数のように文字列として保持されているのではなく、バイナリの状態でシェルの中に保持されています。組み込みコマンドの実行例は次のようになります。

```
% pwd   ←pwdは組み込みコマンド
/root
% cd /z/daichi/   ←cdも組み込みコマンド
% pwd
/z/daichi
%
```

組み込みコマンドは、シェルが内部的に実行する必要がある機能をコマンドの形で扱えるようにしたものです。たとえばcdコマンドは、ユーザから見れば「ディレクトリを移動するためのコマンド」のように振る舞いますが、実際には「シェルが指し示しているカレントディレクトリを置き換えるもの」といった処理しかしません。別にディレクトリの間を移動しているわけではなく、そのように「見せかける」ための処理をしているだけです。

pwdという組み込みコマンドは、この設定されたカレントディレクトリを表示する処理をします。これらはシェルの外部のコマンドを実行しても意味がなく、シェル自身が保持する必要のあるデータですので組み込みコマンド経由で操作します。

エイリアス

コマンド、シェル関数、組み込みコマンドにはそれぞれ別名を割り当てることができます。これは「エイリアス」と呼ばれる機能です(**図2**)。

コマンド入力のショートカットに使われることが多い機能で、インタラクティブシェルとして使っている場合に重宝される機能です。特定の処理を2文字または3文字の文字列に割り当てることで、処理入力時間の短縮などを実施できます。

実行の優先順位

コマンド、シェル関数、組み込みコマンド、エイリアスでそれぞれ同じ名前を持っている場合、どれが優先的に処理されるかを知っておく必要があります。名前の重複に気がつかないと、想定しているのとは別の処理が実行される可能性があります。

次のように同じ名前のコマンド、シェル関数、エイリアスを用意すると、エイリアスがもっとも優先順位が高いことがわかります。

▼図2　エイリアスの定義と使用例

```
% alias ll="ls -al"     ←llにls -alを割り当てている
% pwd
/root
% ll     ←llが展開されたls -alが実行される
total 65
drwxr-xr-x   4 root  wheel    10 11月 19 15:46 .
drwxr-xr-x  21 root  wheel    29 11月 19 13:00 ..
drwxr-xr-x   3 root  wheel     3 11月 10 12:45 .cache
-rw-r--r--   2 root  wheel   964 11月  4 04:49 .cshrc
-rw-------   1 root  wheel  4690 11月 10 09:35 .history
-rw-r--r--   1 root  wheel   147 11月  4 04:49 .k5login
-rw-------   1 root  wheel    90 11月 19 15:36 .lesshst
-rw-r--r--   1 root  wheel   295 11月  4 04:49 .login
-rw-r--r--   2 root  wheel   288 11月 10 09:31 .profile
drwxr-xr-x   3 root  wheel     6 11月 11 09:05 .subversion
% alias     ←エイリアスに登録されている内容を表示する
ll='ls -al'
%
```

```
% which date
/bin/date
% date() {
>     echo time
> }
% alias date="echo alias"
% date
alias
%
```

エイリアスを解除すると次はシェル関数が実行されます。コマンドよりもシェル関数のほうが優先度が高いことがわかります。

```
% unalias date
% date
time
%
```

エイリアスと組み込みコマンドを比較してみましょう。組み込みコマンドとしてのaliasではなく、alias組み込みコマンドで設定されたエイリアスのほうが優先して実行されていることがわかります。

```
% alias alias="echo alias_alias"
% alias
alias_alias
%
```

同じようにシェル関数と組み込みコマンドを比較すると、組み込みコマンドよりもシェル関数のほうが優先して動作していることがわかります。

```
$ alias() { echo alias_func; }
$ alias
alias_func
$
```

コマンド同士の優先順位は、環境変数PATHに定義されているパスで、より最初に一致したほうとなります(**図3**)。

次のようにエイリアスに関してはスラッシュ記号を含む文字列も定義できます。かなり強力な優先順位を持っていることがわかります。

```
% /z/daichi/bin/date() {
Syntax error: Bad function name
% alias /z/daichi/bin/date="echo alias"
% /z/daichi/bin/date
alias
% /
```

コマンド、シェル関数、組み込みコマンド、エイリアスの優先順位をまとめると次のようになります。

▼図3　コマンドは環境変数PATHの先に一致したものが優先される

```
% echo $PATH
/sbin:/bin:/usr/sbin:/usr/bin:/z/daichi/bin
% date     ←/bin:の方が優先されている
Tue Nov 19 15:57:44 JST 2013
% /z/daichi/bin/date     ←フルパスで指定して実行する
home/bin
%
```

①エイリアス
②シェル関数
③組み込みコマンド
④コマンド(環境変数PATHで先に一致したコマンドを優先)

エイリアスは便利な機能ですが優先順位が高いため、扱い方を間違えると問題を引き起こす可能性が高いこともわかります。

コマンド終了ステータス

コマンドは終了した段階で終了ステータスと呼ばれる数値を通知します。通常、正式に終了した場合には0、それ以外の場合には0以外の値が通知されます。このあたりはシェルでは$?でアクセスできます。

```
% true
% echo $?
0
% false
% echo $?
1
%
```

true(1)は0で終了するだけのコマンド、false(1)は1で終了するだけのコマンドです。それ以外の動作はしません。

フォアグラウンドプロセス

次のようにソフトウェアを実行すると、そのソフトウェアが終了するまでシェルはプロンプトを出力せず、入力も受けとりません。

```
% xclock
```

誤解されることが多いのですが、これはオペレーティングシステムのデフォルトの挙動ではなく、シェルが模擬的に作り出した状況です。カーネルではすべてのプロセスは後述するような「バックグラウンドプロセス」としての特性をもっています。カーネルからすればすべてのプロセスはバックグラウンドで並列して動作しているようなものです。

「フォアグラウンドプロセス」も実際には「バックグラウンドプロセス」です。シェルがこのプロセスの終了を待つという処理をしているため、ここで処理が止まったように見えているだけです。人間がインタラクティブにコマンドを実行する場合、このようにソフトウェアの実行が終了するまで待ってから次の処理を受け付ける動作をしたほうが扱いやすいことが多いというわけです。シェルスクリプトとしても1行1行処理が完了してから次に進むほうがスクリプトが組みやすくなります。

バックグラウンドプロセス

次のようにsleep(1)コマンドを実行すると、ターミナルは10秒間、プロンプトも表示せずに待ち状態になります。

```
% sleep 10
```

次のように&を指定してコマンドを実行すると、そのコマンドは「バックグラウンドプロセス」として処理されます。シェルと並列して動作します。このため、sleep 10実行中もシェルはインタラクティブに操作でき、別のコマンドを実行できます。10秒が経過するとsleep 10が終了した旨がシェルに通告されます。

```
% sleep 10 &
% ps | grep sleep
59779  6  S    0:00.00 sleep 10
59781  6  S+   0:00.00 grep sleep
% wait
[1]   Done                    sleep 10
%
```

バックグラウンドプロセスの活用は、マルチコアやメニーコアのシステムにおいてコアを使い切る手法として有益です。waitは組み込みコマンドです。waitが実行されると、すべての子プロセスが終了するまでその場で待機するようになります。フォアグラウンドプロセスの場合

と同じ待ち処理が入ります。バッチ処理を並列して実行して、必要になるポイントでwaitを実行して終了を待つという使い方をします。

サブシェル

のちほど解説しますが、見た目は同じシェルで実行されているように見えても、実際には子プロセスとしてfork(2)したシェルで動作していることがあります。このfork(2)後のシェルはサブシェルと呼ばれています。

次の例ですと、「while :; do done;」の処理がサブシェルで実行されています。サブシェルは現在のシェルをコピーしたシェルだと考えてください。

```
% (while :; do done;) &
% ps d
PID TT  STAT    TIME COMMAND
321  3  S    0:00.01 -su (sh)
325  3  R    0:02.00 - -su (sh)
326  3  R+   0:00.00 - ps d
%
```

シェル、サブシェル、ps(1)がそれぞれ別のプロセスとして実行されています。プロセスは仮想メモリ空間が完全に独立した実行単位です。このあたりがほかの汎用的なプログラミング言語には存在していないシェル特有の特性になります。

パイプライン

シェルはコマンドの出力を、次のコマンドへ結び付ける機能を提供しています。パイプラインと呼ばれる機能で、カーネルが提供するpipe(2)というシステムコールを使って実装されています。

```
% seq 1 10
1
2
3
4
5
6
7
8
9
10
% seq 1 10 | awk '$1>5 {print}'
6
7
8
9
10
%
```

pipe(2)を使うことで、「|」の左側に記載されたコマンドの標準出力を、「|」の右側に記載されたコマンドの標準入力へ結び付けるといった処理が行われます(**図4**)。

パイプはいくらでも接続できます。パイプで接続されたソフトウェアはそれぞれがバックグラウンドプロセスとして同時に並列して動作します。マルチコア／メニーコアを使い切るもっとも簡単な方法です(**図5**)。

パイプは一方通行で名前空間も限定されています。グローバルな名前空間でパイプと同じ機能を使いたい場合には名前付きパイプ(FIFO)という機能を使います。名前付きパイプはファイルシステムの名前空間を利用します。FIFOのように振る舞うファイルだと思うと理解しやすいと思います。 名前付きパイプはシェルスクリプトにおけるプログラミングで活用することがあります。

▼図4　コマンドは2つのみならず何個でも連結できる

```
% seq 1 10 | awk '$1>5 {print}' | awk '$1<8 {print}'
6
7
%
```

▼図5　パイプで接続されたコマンドは並列で動作するのでマルチコアの性能を使い切るのに都合が良い

```
% sleep 100 | sleep 101 | sleep 102 | sleep 103 | sleep 104 &
% ps | grep sleep
60373  6  S    0:00.00 sleep 100
60374  6  S    0:00.00 sleep 101
60375  6  S    0:00.00 sleep 102
60376  6  S    0:00.00 sleep 103
60377  6  S    0:00.00 sleep 104
60379  6  S+   0:00.01 grep sleep
%
```

グルーピング

シェルはコマンドをグループ化する2つの方法と提供しています。サブシェルでの実行と、同じシェルでの実行です。

次のように処理を()で囲むと、囲まれたコマンドはサブシェルで実行されます。コピーされたシェル(サブシェル)からさらにコマンドのプロセスが生まれていることがわかります。

```
$ (sleep 200 | sleep 100)&
$ ps d
  PID TT  STAT    TIME COMMAND
  321  3  S    0:00.02 -su (sh)
 1130  3  S    0:00.00 - -su (sh)
 1131  3  S    0:00.00 |-- sleep 200
 1132  3  S    0:00.00 `-- sleep 100
 1133  3  R+   0:00.00 - ps d
$
```

サブシェルが生成されているということは、()でグルーピングされた中で組み込みコマンドを実行しても、元のシェルには影響を与えません。別プロセスとして動作しているので、影響を与えることはできません。次のようにサブシェルの中でcdしたとしても、それはサブシェルの世界で閉じており、元のシェルには影響を与えないわけです。

```
% (cd /root; pwd)
/root
% pwd
/z/daichi
%
```

もう1つのグループ化が{}によるグループ化です。このグループ化は同じシェルで実施されます。複数のコマンドから出力される内容をまとめたい場合などに使われます。

```
% pwd
/z/daichi
% { cd /root;}
% pwd
/root
%
```

このあたりのスコープの考え方が、ほかのプログラミング言語にはないところです。スコープはプロセスとして分離しているかどうかが主な判断ポイントとなります。たとえば次のように{}&としてグループ化したものをバックグラウンドプロセスとして動作させる場合、バックグラウンドプロセス化した時点でサブシェルを生成して別プロセスになりますので、次のように()として処理をサブシェルとした場合と同じ挙動を示すことになります。

```
% pwd
/z/daichi
% { cd /root;} &
% pwd
/z/daichi
[1]   Done                    cd /root
%
```

{}はそのまま使うとしても、コマンドごとに&を指定したとすれば、その処理もサブシェルで実行されます。たとえば次の例ではcd /rootという処理がサブシェルで実行されます。{}自体は同じシェルで実行されてますが、cd /rootがサブシェルで実行されていますので、この処理はシェルのカレントディレクトリには影響を与えません。

```
% pwd
/z/daichi
% { cd /root&}
% pwd
/z/daichi
[1]   Done                    cd /root
%
```

このあたりのスコープが把握できるようになるとシェルやシェルスクリプトを使ってより高度なリソースの活用ができるようになります。

コマンド実行のしくみ — fork(2)とexecve(2)

ここでシェルが実際にどのようにしてコマンド、いわゆるソフトウェアを実行しているのかを把握しておきましょう。ここが理解できるようになるとシェルの本質をだいぶつかみやすくなります。

実施にどのような順序でシステムコールが呼ばれているのかを次のようなDTraceのスクリプトを用意して調査します。**リスト1**のスクリプトでashで実行されるfork(2)システムコール

▼リスト1　fork(2)とexecve(2)システムコールの流れを追うためのDスクリプト

```
syscall:freebsd:fork:entry      /execname == "sh"/ { printf("| %s %d %d syscall %
s()", execname, pid, timestamp, probefunc); }
syscall:freebsd:fork:return     /execname == "sh"/ { printf("| %s %d %d syscall %
s() -> %d", execname, pid, timestamp, probefunc, arg1); }
syscall:freebsd:exec*:entry     /execname == "sh"/ { printf("| %s %d %d syscall %
s(\"%s\")", execname, pid, timestamp, probefunc, copyinstr(arg0)); }
syscall:freebsd:exec*:return    /execname == "sh"/ { printf("| %s %d %d syscall %
s() -> %d", execname, pid, timestamp, probefunc, arg1); }
```

▼図6　fork(2)とexecve(2)のコールを確認

```
% kldload dtraceall
% dtrace -s TRACE.d
dtrace: script 'TRACE.d' matched 4 probes
CPU     ID                    FUNCTION:NAME
  7  55284                       fork:entry | sh 63520 366458590978285 syscall fork()
  7  55285                      fork:return | sh 63520 366458591129605 syscall fork() -> 
63808
  1  55398                     execve:entry | sh 63808 366458605136171 syscall execve("/
bin/date")
```

とexec系(2)システムコールを捕捉できます。

次のようにdtrace(8)でモニタリングを開始します。

```
% kldload dtraceall
% dtrace -s TRACE.d
dtrace: script 'TRACE.d' matched 4 probes
```

この状態でashで何かコマンドを実行します。

```
% date
Tue Nov 19 17:40:33 JST 2013
%
```

すると**図6**のようなモニタリング結果がdtrace(8)側に表示されます。

fork(2)はプロセスをコピーするシステムコールです。シェルは55284というプロセスIDが割り当てられたプロセスとして動作しているわけですが、ここからプロセスID 55285という番号のプロセスが生まれています。この段階ではプロセスIDが違うだけでまったく同じものです。次にexecve("/bin/date")というようにexecve(2)システムコールが呼ばれます。execve(2)システムコールは自分自身を指定したソフトウェアで上書きして実行を移すという処理をします。

つまり、シェルは次の2つの手続きをコマンドを実行するごとに毎回実施していることになります。

①自分自身をコピー(つまりサブシェルを生成)
②コピーで生み出されたほう(サブシェル)は自分を別のプログラムで上書きして実行をそちらへ移す

これはUNIX系オペレーティングシステムの特徴的な動作ですが、fork(2)で自分自身をコピーすることで権限情報や規制情報、環境変数など共有すべきデータの引き継ぎができるようになるほか、プロセス自身が木構造の親子関係を持つようになります。しくみが単純なうえに管理上都合が良いことが多いわけです。

生成されたサブシェルで実行すべき対象がコマンドではなく組み込みコマンドやシェル関数である場合、そのままサブシェルで実行されます。SD

第3章 きっちり押さえるシェルの基本 シェルの４大要素②　リダイレクト／変数とパラメータ／パラメータ展開

第２章ではシェルの「コマンド」を取り上げました。本章では、ほかの３つの要素「リダイレクト」「変数とパラメータ」「パラメータ展開」を取り上げます。また、複数行のテキストをシェル実行中に記載する機能「ヒアドキュメント」についても解説します。

リダイレクト

シェルの特徴的な機能の２つめが入出力のリダイレクトです。

標準出力リダイレクト「>」

もっともよく使われるのはコマンドが出力するデータをファイルへ書き込む標準出力リダイレクトです。次のように「>」で指定します。

```
% date
Wed Nov 20 16:12:05 JST 2013
% date > out
% cat out
Wed Nov 20 16:12:09 JST 2013
% echo a > out
% cat out
a
%
```

シェルは「>」のあとに指定されたファイルを新しく作成し、そこにデータを書き込みます。ファイルが存在する場合にはそのファイルの内容はクリアされ、新しく書き込まれた内容に置き換わります。

標準出力リダイレクト追記「>>」

「>」では既存のファイルの内容は消えてしまいますが、「>>」を使うと追記になり、ファイルの最後に追加されるようになります。

```
% cat out
a
% echo a > out
% cat out
a
% echo b >> out     →outへ追記する
% cat out
a
b
%
```

具体的には、open(2)システムコール時に指定するオプションを変更することで、このあたりの動作の違いが実装されています。

標準入力リダイレクト「<」

標準出力の先を向けるリダイレクトの逆版というか、特定のファイルを標準入力に指定するための「<」というリダイレクトも存在します。

```
% cat out
Tue Nov 19 17:50:47 JST 2013
% cat < out  ←標準入力へoutの内容をリダイレクト
Tue Nov 19 17:50:47 JST 2013
%
```

このようにcat(1)で内容を取り出すことと同じですが、cat(1)を経由するとパイプラインを挟むことになって別プロセスでの実行になります。こうした事態を避けるために、この標準入力リダイレクトが使われたりします。ターミナルやコンソール上で制御端末におけるフォアグラウンドプロセスとして実行されているプロセスで実行しないと意味のないコマンドなどがありますので、そういった用途で使われます。

たとえば次のような数字が格納されたデータファイルがあったとします。

```
% cat data
1
2
3
4
5
6
7
8
9
%
```

このファイルの中身の数値を全部足すような処理をシェルスクリプトで書くには、次のようします(通常こういう処理は別途コマンドを使ったりawk(1)を使います。動作の説明用だと思っておいてください)。入力は標準入力リダイレクトを使っています。

```
% i=0
% while read a
> do
>       i=$(( $i + $a ))
> done < data
% echo $i
45
%
```

ここの処理をcat(1)を使ってパイプ経由にしても同じことが処理されるように思いますが、パイプを経由させると想定しているようには動作しなくなります。

```
% i=0
% cat data |
> while read a
> do
>       i=$(( $i + $a ))
> done
% echo $i
0   ←パイプを挟むと思ったようには機能しない
%
```

なぜなら、これまで説明してきたようにパイプを挟んだ段階でwhileの処理はサブシェルで実行されるようになるからです。この段階でwhileの中の変数iとwhileのその外の変数iには連続性がなくなります。このためサブシェルが終了するとその値も失われることになります。

このようにシェルスクリプトではどのプロセスで実行されているのかを意識することが重要になってきます。

標準エラー出力リダイレクト「2>」

標準入力ではなく標準エラー出力をリダイレクトするには2>を使います。

```
% awk '$1>>1'
awk: syntax error at source line 1
 context is
         >>> $1>> <<<
awk: bailing out at source line 1
% awk '$1>>1' 2> /dev/null
%
```

標準エラー出力のファイルディスクリプタ番号は2ですので、その値が使われています。標準出力リダイレクトは「>」だったわけですが、これは「1>」と書いても同じです。1は標準出力のファイルディスクリプタ番号です。ほかのファイルディスクリプタ番号を指定して、そのディスクリプタのみを特定のファイルへ向けるといったことができます。

ファイルディスクリプタのコピー「2>&1」「1>&2」

リダイレクトでよく使われているものがあと2つあります。「2>&1」と「1>&2」です。この書き方は「>&」というリダイレクトの前後にファイルディスクリプタ番号が指定されたものです。dup2(2)システムコールが呼ばれます。

「2>&1」はファイルディスクリプタ2の出力先を、ファイルディスクリプタ1に変更せよ、という指定になります。このリダイレクトが指定されると標準エラー出力に出されたデータであっても標準出力に出力されたように振る舞います(**図1**)。

典型的には「> /dev/null 2>&1」の形で使われます。標準出力は/dev/nullへリダイレクトされていますから、標準出力に出てきたメッセージは削除されます。さらに「2>&1」の指定がありますので、標準エラー出力の内容も/dev/nullへ向くことになります。つまり、「> /dev/null 2>&1」は標準エラー出力も標準出力もすべて削除する、という指定になります。

「1>&2」は逆に標準出力の出力先を標準エラー

▼図1　標準エラー出力を標準出力と同じにする使い方

```
% printf 'echo echo; echoo' | sh -e
echo
echoo: not found
% printf 'echo echo; echoo' | sh -e > /dev/null
echoo: not found
% printf 'echo echo; echoo' | sh -e 2> /dev/null
echo
% printf 'echo echo; echoo' | sh -e > /dev/null 2> /dev/null
% printf 'echo echo; echoo' | sh -e > /dev/null 2>&1
%
```

▼図2　標準出力も標準エラー出力へ差し替える使い方

```
% printf 'echo echo; echoo' | sh -e 1>&2
echo
echoo: not found
% printf 'echo echo; echoo' | sh -e > /dev/null
echoo: not found
% printf 'echo echo; echoo' | sh -e 1>&2 | cat > /dev/null
echo
echoo: not found
%
```

出力へ向けるという指定になります。エラーメッセージの表示など、標準エラー出力として出力したい場合に使われます。

変数

シェルは変数を持っています。通常の「変数」以外に、子プロセスへコピーされる「環境変数」、1文字の変数で構成される「位置パラメータ」と「特殊パラメータ」、シェルの動作に影響を与える「特殊変数」があります。意味が違っていますが、基本的にはデータを対して名前を割り当てたものです。シェルは基本的に型という概念を持ちません。変数が指しているのはただのデータとして扱われます。

変数

変数は「a=a1」のように設定されます。aが変数名、a1が変数の示す値（データ）です。

```
% a=a1
% echo $a
a1
%
```

変数には「$a」または「${a}」のようにアクセスします。

環境変数

変数はシェルが保持するデータです。このため子プロセスには引き継がれません[注1]。

```
% a=a1
% echo $a
a1
% sh -c 'echo $a'
    ←変数は子プロセスへは引き継がれない
%
```

こうした制限を超え、子プロセスにもコピーされる変数が環境変数です。

```
% echo $a
a1
% sh -c 'echo $a'

% export a     ←子プロセスへ引き継がれる
% sh -c 'echo $a'
a1
%
```

注1）サブシェルには引き継がれます。execev(2)が実行されるまではfork(2)後で保持しているからです。

組み込みコマンドのexportで変数を指定すると、その変数は環境変数になります。環境変数は子プロセスに対してもコピーされます。

位置パラメータ

シェルそのものに渡されてくる引数は「位置パラメータ」と呼ばれる変数でアクセスします。次のように$1から$9まで、9の位置パラメータが用意されています(**表1**)。

次の例がシェルスクリプトやシェルの引数として位置パラメータを利用したものです。

```
% cat s1.sh
#!/bin/sh

echo "1: $1"
echo "2: $2"
echo "3: $3"
echo "4: $4"
echo "5: $5"
echo "6: $6"
echo "7: $7"
echo "8: $8"
echo "9: $9"
%
```

位置パラメータは次のように9個の引数までアクセスできます。

```
% ./s1.sh a b c d e f g h i j k l m n
1: a
2: b
3: c
4: d
5: e
6: f
7: g
8: h
9: i
%
```

▼表1　位置パラメータ

位置パラメータ	意味
$1	第1引数
$2	第2引数
$3	第3引数
$4	第4引数
$5	第5引数
$6	第6引数
$7	第7引数
$8	第8引数
$9	第9引数

位置パラメータには特徴があります。シェルは組み込みコマンドshiftを用意しています。shiftを実行すると、位置パラメータの内容がシフトします。$1の内容は破棄され、$1に$2の内容が、$2に$3の内容が、といったように値が1つずつ前にずれるような動きをします。$9には第10引数だったものが格納されます。

```
% cat s2.sh
#!/bin/sh

echo "1: $1"
echo "2: $2"
echo "3: $3"
echo "4: $4"
echo "5: $5"
echo "6: $6"
echo "7: $7"
echo "8: $8"
echo "9: $9"
shift; echo "10: $9"
shift; echo "11: $9"
shift; echo "12: $9"
shift; echo "13: $9"
%
```

こうすることで9個を超える引数を処理します。

```
% ./s2.sh a b c d e f g h i j k l m n
1: a
2: b
3: c
4: d
5: e
6: f
7: g
8: h
9: i
10: j
11: k
12: l
13: m
%
```

位置パラメータは、組み込みコマンドsetを使うと自分で定義することもできます。

```
% cat s3.sh
#!/bin/sh

set -- A B C D E F G H I J K L M N
echo "1: $1"
echo "2: $2"
echo "3: $3"
echo "4: $4"
echo "5: $5"
```

```
echo "6: $6"
echo "7: $7"
echo "8: $8"
echo "9: $9"
%
```

組み込みコマンドsetも特徴的なコマンドです。このコマンドは3つの異なる役割を持っています。位置パラメータの設定はそうした役割の1つです。

```
% ./s3.sh a b c d e f g h i j k l m n
1: A
2: B
3: C
4: D
5: E
6: F
7: G
8: H
9: I
%
```

特殊パラメータ

シェルは1文字で構成された特殊なパラメータである「特殊パラメータ」を提供しています。特殊パラメータはいくつかありますが、とくに重要なものをまとめると**表2**のようになります。

$0は位置パラメータのようでもあります。コマンドが実行されたときのコマンド名が格納されています。

```
% cat s4.sh
#!/bin/sh

echo $0     ←実行時にコマンドに展開される
%
```

実際にs4.shを実行すると次のようになります。実行のされ方しだいで$0の値は変化します。

```
% ./s4.sh
./s4.sh
% /usr/home/sh/s4.sh
/usr/home/sh/s4.sh
%
```

次の例では、$#には引数の個数が格納されています。

```
% cat s7.sh
#!/bin/sh

echo $#
%
```

これを実行すると、引数の個数が表示されます。つまり$#は位置パラメータの個数を表現しているのです。

```
% ./s7.sh a   b "c  d"
3
%
```

次の例では、$*には引数全体が格納されています。$*は空白区切りの引数全体へ展開されます。

```
#!/bin/sh

for i in $*
do
        echo "$i"
done
echo ----
for i in "$*"
do
        echo "$i"
done
%
```

▼表2　特殊パラメータ

特殊パラメータ	意味
$0	シェルまたはシェルスクリプトの名前。コマンドとして実行されたときのコマンドが格納されている
$#	引数の個数（つまり位置パラメータの個数）
$*	引数すべてを空白区切りの文字列へ展開。ダブルクォートでくくられた場合、個別引数の空白は保持される。ダブルクォートでくくらない場合、個別引数の空白も区切りの空白もどちらも区別されない
$@	引数すべてを空白区切りの文字列へ展開。ブルクォートでくくられた場合、個別引数の空白は保持されるとともに、引数としての個数も保持される
$?	直前のコマンドの終了ステータスを保持
$$	シェルのプロセスID

ダブルクォートでくくると個別引数の空白は保持されます。

```
% ./s5.sh a   b "c  d"
a
b
c
d
----
a b c  d
%
```

$@にも引数全体が格納されています。ダブルクォートで"$@"のようにクォートすると、引数としての情報も保持されます。引数全体を扱うときは基本的に"$@"を使うものだと思っておけば良いでしょう。

```
% cat s6.sh
#!/bin/sh

for i in $@
do
        echo "$i"
done
echo ----
for i in "$@"
do
        echo "$i"
done
%
```

ダブルクォートでくくると$*の動きに加えて、それぞれを個別の引数として扱います。

```
% ./s6.sh a   b "c  d"
a
b
c
d
----
a
b
c  d
%
```

$$にはそのシェルのプロセスIDが保持されています。

```
% echo $$      ←$$はシェルのプロセスIDを保持
64564
% ps | grep sh
64564  3  S    0:00.15 -su (sh)
69726  3  R+   0:00.00 grep sh
%
```

$?には直前に実行したコマンドの終了ステータスが格納されています。コマンド終了ステータスについては先に説明したとおりです。

特殊変数

特殊変数は変数の中でもシェルの挙動に影響を与えるものをいいます。特殊変数は環境変数になっていることもあります。代表的な特殊変数は**表3**のとおりです。

cdを引数なしで実行するとホームディレクトリへ移動しますが、この値はHOMEに格納されています。環境変数でもあり、ソフトウェアの動作に影響を与えます。IFS変数は入力フィールドのセパレータです。空白、タブ、改行が設定されています。文字列を任意の文字で分割したい場合などに使われますが、IFS変数を使用した文字列操作は理解しにくいので多用しないほうが無難です。文字列操作が必要な場合にはそれを得意としたソフトウェアを使ったほうが良いでしょう。

PATHは環境変数です。実行可能なソフトウェアが格納されたディレクトリの一覧が設定されています。

▼表3　特殊変数

特殊変数	意味
HOME	ホームディレクトリ。組み込みコマンドcdを引数なしで実行した場合、この変数が示すパスへカレントディレクトリを設定する
IFS	入力フィールドセパレータ(Input Field Separator)。デフォルトでは空白、タブ、改行に設定されている。IFS変数を変更することで文字列の区切り文字を変更できる
PATH	コマンドの検索に利用されるパス一覧
PPID	親プロセスのプロセスID

▼表5　パラメータ展開

パラメータ展開	内容
${parameter:-word}	パラメータが設定されていないかnullの場合に:-以降の文字列を展開する。変数のデフォルト値を指定する場合で使われる
${parameter:=word}	:-と同じ。ただしset parameter=wordも同時に実施される
${parameter%word}	wordパターンに一致する最短の文字列サフィックスを削除する
${parameter%%word}	wordパターンに一致する最長の文字列サフィックスを削除する
${parameter#word}	wordパターンに一致する最短の文字列プレフィックスを削除する
${parameter##word}	wordパターンに一致する最長の文字列プレフィックスを削除する

シェルパターン

シェルではパス名、制御構文、変数展開などのタイミングでパターンを指定できます。簡単な正規表現のようなものです。**表4**で示す4つの書き方を把握しておきましょう。

パス名展開ではパターンはスラッシュの間の文字列に対して適用できます。スラッシュを超えての指定はできません(多機能なインタラクティブシェルにはそういう機能も用意されています)。

展開

シェルは変数やパラメータ、その他の文字列を展開して使用します。展開には「チルダ展開」「パラメータ展開」「コマンド置換」「算術展開」「フィールドセパレータ分割」「パス名展開」などがあります。このあたりの展開規則や順序を理解しておくと、すっきりとしたシェルスクリプトが書けるようになります。

チルダ展開

IFS変数で指定された入力フィールドセパレータ(空白、タブ、改行)などのあとのチルダ「~」はホームディレクトリに展開されます。$HOMEと同じです。

```
% echo $HOME
/home/sh
% echo ~
/home/sh
% echo ~sh
/home/sh
% echo ~daichi
/z/daichi
% echo ~/daichi
/home/sh/daichi
% echo daichi~
daichi~
% echo daichi/~
daichi/~
%
```

▼表4　シェルパターン

パターン	意味
?	任意の1文字
*	任意の文字列
[文字列]	文字列内のどれか1文字
[^文字列]	文字列内の文字以外の文字

$HOMEの入力を省く効果があります。

パラメータ展開

変数やパラメータはアクセスする段階で特定の処理を実施できます。この処理は「パラメータ展開」と呼ばれます。とくによく使われるパラメータ展開は次のとおりです。

変数が設定されていないかまたはnullが設定されている場合、:-や:=などの指定でデフォルト値を代わりに展開できます。:-は展開するだけ、:=だと変数にデフォルト値を代入とともに展開も実施します。

```
% echo $b

% echo ${b:-B}
B
```

```
% echo $b

% echo ${b:=B}
B
% echo $b
B
%
```

あとは文字列の加工です。切り取り機能のみを提供しています。#と##が左側からのパターンに一致する文字列の削除、%と%%が右側からのパターンに一致する文字列の削除です。これらパラメータ展開はパスの加工やファイル名の切り出し、サフィックスの切り取りなどに使われます。

```
% echo $ENV
/home/sh/.shrc
% echo ${ENV%.*}
/home/sh/
% echo ${ENV%e*}
/hom
% echo ${ENV#*/}
home/sh/.shrc
% echo ${ENV##*/}
.shrc
%
```

シェルそのものは(とくにashやdashなどBourne Shellに近いシェルは)文字列の加工を目的としていません。文字列の加工はそれに長けたほかのソフトウェアを使ったほうが良いといえます。

コマンド置換

シェルにはコマンドを実行してその結果へ置き換えるという機能があります。これはコマンド置換と呼ばれ、$()でくくった文字列が対象となります。次の例ではコマンド置換によって末尾の改行の連続は削除されています。

▼表6　算術演算に使用できる演算子

演算子	種類
単項演算子	!、~、+、-
二項演算子	*、/、%、+、-、<<、>>、<、<=、>、>=、==、!=、&、^、|、&&、|
代入演算子	=、+=、-=、*=、/=、%=、<<=、>>=、&=、^=、|=
条件操作	?、:

```
% /usr/bin/printf 'a\nb\nc\n\n\n'
a
b
c


% echo $(/usr/bin/printf 'a\nb\nc\n\n\n')
a b c
% echo "$(/usr/bin/printf 'a\nb\nc\n\n
\n')"
a
b
c
%
```

コマンドの実行結果はダブルクォーテーションでクォートするかどうかでIFS変数の扱いが変わります。ダブルクォートでクォートすればIFS変数はそのまま保持されますし、そうでなければ空白区切りへフィールドセパレータ分割されます。

算術展開

シェルは簡単な算術演算も提供しています。$(())で囲まれた文字列が対象となります。おもにインデックス値をインクリメントして使うといった用途に使われます。演算結果は整数になります。

```
% cat s8.sh
#!/bin/sh

i=0
while [ -e /dev/ada$i ]
do
        echo smartctl -a /dev/ada$i
        i=$(( $i+1 ))
done          ↑インデックス値のインクリメント
%             に使われることが多い
```

これを実行すると次のように表示されます。

```
% ./s8.sh
smartctl -a /dev/ada0
smartctl -a /dev/ada1
smartctl -a /dev/ada2
smartctl -a /dev/ada3
%
```

シェルは数値計算を目的としたソフトウェアではありません。$(())の使用は必要最低限にしておいたほうが良いといえます。数値演算を目的としたソフトウェアはbc(1)などがあります

ので、それで計算したり、多くのほかのプログラミング言語がそうした機能を提供していますのでそちらの機能を使います。

フィールドセパレータ分割

展開結果にはIFS変数に設定されている文字を区切り文字としてフィールドセパレータ分割が実施されます。ダブルクォートで囲っている場合にはこの分割の対象外となります。次の例ではIFS変数の設定値に従ってフィールドセパレータ分割しています。

```
% a=$(printf "  a  b\n  c  \n\t ")
% echo $a
a b c
% echo "$a"
  a  b
  c

%
```

IFS変数に設定する値を変更すると、ここでの処理が変わります。

パス名展開

パス名にはパターンを含めることができます。シェルパターンについては先ほど解説しました。このパス名の展開が実施されます。

```
% ls /usr/local/l*/libz*
/usr/local/lib/libzip.a          /usr/local/lib/libzip.so
/usr/local/lib/libzip.la         /usr/local/lib/libzip.so.3
% ls /usr/local/???/libz*
/usr/local/lib/libzip.a          /usr/local/lib/libzip.so
/usr/local/lib/libzip.la         /usr/local/lib/libzip.so.3
% ls /usr/local/[bil][bil][a-z]/libz*
/usr/local/lib/libzip.a          /usr/local/lib/libzip.so
/usr/local/lib/libzip.la         /usr/local/lib/libzip.so.3
% ls /usr/local/[bil][bil][^a]/libz*
/usr/local/lib/libzip.a          /usr/local/lib/libzip.so
/usr/local/lib/libzip.la         /usr/local/lib/libzip.so.3
%
```

パス名の展開はスラッシュが含まれないことに注意してください。個別のディレクトリ名やファイル名に対して適用できるというもので、スラッシュで区切られたパス全体を指定することはできません(そういった機能を実装した多機能シェルもあります)。

展開の優先順位

文字列展開には優先順位があります。次の順序で展開が実施されます。

①チルダ展開
②パラメータ展開
③コマンド置換
④算術演算
⑤入力フィールドセパレータ
⑥パス名展開
⑦クォートの削除

このあたりはシェルスクリプトを組む場合に最初の障壁となる部分でしょう。とくに入力フィールドセパレータとクォートの関係などに苦慮すると思います。

シェルスクリプトは難しく組まない、いったんファイルに出力する、といった処理を経てシェルスクリプト自体はとてもシンプルな状態にしておくことがうまいシェルスクリプトの組み方になります。

ヒアドキュメント

シェルは複数行のテキストをシェル実行中に記載する機能「ヒアドキュメント」を提供しています。**図1**のように「<<区切り文字列」で指定した文字列だけの行までのデータを標準入力として指定したコマンドへ流し込みます。「cat<<eof」と記述すればeofが現れるまでの文字列が標準入力へ送られます。

ヒアドキュメントでは文字列展開の規則が異なります。ヒアドキュメントで適用される文字列展開はパラメータ展開、コマンド展開、算術演算、の3つです。区切り文字列をクォートするとこの文字列展開も実施されず、区切り文字だけの行までのデータがそのままリダイレクト

▼図1　ヒアドキュメントではパラメータ展開、コマンド置換、算術演算が実施される

```
% cat<<eof
                $(printf "|\t$HOME%d\n" $((1+1)))
eof     ↑この部分はタブです
                |       /home/sh2
%
```

▼図2　区切り文字列をクォートすると展開は実施されなくなる

```
cat<<'eof'
                $(printf "|\t$HOME%d\n" $((1+1)))
eof
                $(printf "|\t$HOME%d\n" $((1+1)))
%
```

▼図3　「<<-」を指定すると先頭の連続するタブが削除される

```
% cat<<-eof
                $(printf "|\t$HOME%d\n" $((1+1)))
                eof
|       /home/sh2
%
```

▼図4　「<<-」指定でさらに区切り文字列をクォートすると、先頭の連続するタブが削除されたうえで何も展開されずに表示される

```
% cat<<-'eof'
                $(printf "|\t$HOME%d\n" $((1+1)))
                eof
$(printf "|\t$HOME%d\n" $((1+1)))
%
```

の対象となります(**図2**)。

「<<-」のようにヒアドキュメントを指定すると、データの中で行頭から連続するタブが無視されるようになります(**図3**)。シェルスクリプトを記述している場合、このヒアドキュメントを使うことでシェルスクリプトのインデントの統一性を損なうことなくヒアドキュメントを活用できるようになります。

「<<-'eof'」のように区切り文字列をクォートした場合(**図4**)、行頭の連続するタブは削除されますが、それ以外の文字列は文字列展開の対象にはならずにそのまま出力されます。SD

注2) タブは Ctrl + v TAB で入力できます。

COLUMN 時短の極意、エイリアスを使い倒す

強力な機能であり扱い方も簡単ですが、うまく活用されていないことが多いのがエイリアスです。エイリアスはインタラクティブシェルとしてシェルを使う場合に時短に活用できる機能だということを知っておくと良いと思います。

たとえばエイリアスは~/.aliasesといったファイルに書いておきます。こうするとBourne Shell互換のシェルから共通のファイルとして読み込めるので便利です。

2、3回でも繰り返し同じコマンドを打っているということに気がついたら、それはもうエイリアスに登録してしまったほうが良いタイミングかもしれません。手癖として打ちやすい2、3文字の文字列へエイリアスを張って利用します。~/.aliasesには、時々使うし重要なコマンドなんだけどオプションの指定方法を必ず忘れる、といったたぐいのコマンドを記憶しておくためのメモとしても使用できます。

時短の要は積極的に手を抜くことを考えるということです。エイリアスを活用していないのなら、これから活用を考えてみてはいかがでしょうか。

▼図A　繰り返しコマンドを打っていることに気がついたら2、3文字の文字にエイリアスする

```
% cat ~/.aliases
alias gihyo="cd /z/daichi/Documents/fdt/;clear;ls"
alias digital="cd /z/daichi/Documents/digitalfreebsd/;clear;ls"
alias sd="cd /z/daichi/Documents/softwaredesign/;clear;ls"
alias sdd="cd /z/daichi/Documents/softwaredesign/201401/shell/;ls"
alias load="sudo make load"
alias unload="sudo make unload"
alias mount_dos="sudo mount_msdosfs -o large -L ja_JP.UTF-8 -D utf-8"
% . ~/.aliases
% sdd
Makefile        csvs            sources         typescript.xml
commands        images          typescript.html
%
```

第4章 ちゃんと知っていますか? 意外と知らない「組み込みコマンド」の意味

シェルには、シェルが内包している機能である組み込みコマンドが用意されています。シェルが実装していないと意味がない機能がこれに相当します。

真値で終了「:」

true(1)と同じ動きをする組み込みコマンドに「:」があります。次のように実行すると終了ステータス0で終了します。

```
% :
% echo $?      ←ステータスを表示
0
%
```

次のように使うと、終わることのないビジーループを実現できます。

```
% while :
> do
> done
```

この組み込みコマンドは、次のようにパラメータ展開と組み合わせて未設定の変数にデフォルト値を代入するといった用途でも使われます。

```
% echo $default_router

% : ${default_router:=192.168.1.1}
% echo $default_router
192.168.1.1
%
```

同じ処理を、「:」とパラメータ展開を使わない分岐構文で実装すると次のようになります。

```
% echo $default_router

% [ -z "$default_router" ] && default_
router=192.168.1.1
% echo $default_router
192.168.1.1
%
```

慣れの問題もありますが、「:」を使ったほうが簡潔に書けます。また、分岐構文で記述するよりも「:」のほうが処理速度が高速です[注1]。

フォアグラウンドとバックグラウンドの切り替え「fg」「bg」

ソフトウェアをフォアグラウンドプロセスとして実行するかバックグラウンドプロセスとして実行するかは、ソフトウェアを実行したあとから切り替えることができます。次のxclockはフォアグラウンドで動作し続けるプロセスです。

```
% xclock   ←デスクトップに時計が表示される
```

フォアグラウンドで実行した場合、ターミナルで Ctrl + Z と入力します。こうするとプロセスはサスペンドの状態になって処理が一時停止します。

```
% xclock
^Z[1] + Suspended                xclock
%
```

組み込みコマンドjobsで状態を表示させると「Suspended」となっていることが確認できますし、ps(1)で状態を調べると停止したプロセスである

注1) ただし、現在のashは[(1)もtest(1)も組み込みコマンドとして内部に取り込んでいるので、処理速度に大きな違いはありません。

ことを示す「T」のフラグを確認できます。

```
% jobs
[1] + Suspended                 xclock
% ps -x | grep xclock
79087  2  T    0:00.01 xclock
79134  2  S+   0:00.00 grep xclock
%
```

ここで組み込みコマンドbgを実行します。bgを実行すると最後にサスペンドしたプロセスがバックグラウンドプロセスとして実行されます。サスペンド中のプロセスが複数ある場合、bgを実行するごとにもっとも最後にサスペンドしたプロセスから順次バックグラウンドプロセスとして実行を再開します。

```
% bg
[1] xclock
%
```

バックグラウンドプロセスとして動作していることを確認するには次のようにします。

```
% jobs
[1] + Running                   xclock
% ps | grep xclock
79087  2  S    0:00.01 xclock
79294  2  S+   0:00.00 grep xclock
%
```

この状態でfgを実行すると、起動時刻(再開時刻ではなく起動された時刻)がもっとも新しいバックグラウンドプロセスがフォアグラウンドプロセスへ切り替わります。フォアグラウンドプロセスになったあとは、Ctrl + Cと入力すればそのプロセスを終了させることができます。

▼リスト1 ashのソースコードcd.cファイル内のupdatepwd()関数の処理

```
static void
updatepwd(char *dir)
{
        hashcd();  /* update command hash table */

        if (prevdir)
                ckfree(prevdir);
        prevdir = curdir;
        curdir = dir ? savestr(dir) : NULL;
        setvar("PWD", curdir, VEXPORT);
        setvar("OLDPWD", prevdir, VEXPORT);
}
```

```
% fg
[1] + Running                   xclock
^C
%
```

インタラクティブシェルとしてシェルを操作している場合に、プロセスを終了させることなく別の処理を実行したい場合などにこの操作(Ctrl + Z、bg、fg)が使われます。

ディレクトリの移動「cd」

ディレクトリを移動する組み込みコマンドがcdです。現在のディレクトリ(カレントディレクトリ)は組み込みコマンドpwdで表示させることができます。

```
% cd
% pwd
/z/daichi
% cd /root
% pwd
/root
% cd /usr/local/
% pwd
/usr/local
%
```

実際には指定されたディレクトリが存在し、そこに移動可能であるかどうかを調査し、可能であればそのディレクトリを現在いるディレクトリ(カレントディレクトリ)として設定する、という処理をしています。ashの場合、cd.cというソースコードのupdatepwd()という関数でその処理を確認できます(**リスト1**)。

savestr()という関数はmalloc()で文字列の長さ分だけメモリを確保して、指定された文字列をそこへコピーする処理をしていますので、この関数で指定されたディレクトリをカレントディレクトリとして保存し、PWD環境変数にカレントディレクトリを設定し、OLDPWD環境変数に1つ前のカレントディレクトリを設定しています。

パースと実行「eval」

指定された文字列をすべてスペースで結合し

たのち、パースして実行するという組み込みコマンドがあります。evalという名前で用意されています。ちょっとわかりにくいですが、おもに変数を含む変数を展開したい場合に使われる機能です。次の例を見るとわかると思います。

```
% a=1
% a1=b
% echo \$a$a
$a1
% eval echo \$a$a
b
%
```

a1やa2といった変数がある場合、この1や2の部分を別の変数が保持していることがあります。「eval echo \$a$a」はまずevalがパースと実行を実施するため「echo $a1」に展開されたあとにこれが実行されて$a1の中身が表示されます。

プロセスの置き換え「exec」

execve(2)システムコールに相当する組み込みコマンドがexecです。これを指定するとシェル自身を指定したソフトウェアで置き換えて実行するため、シェルとしての実行はその時点で終了します。

```
% cat s13.sh
#!/bin/sh

/usr/bin/printf "done\n"
/usr/bin/printf "done2\n"
%
```

これを実行すると、次のように動作します。

```
% ./s13.sh
done
done2
%
```

次のようにexecを指定したコマンドが実行されると、そこでシェルは終了するため、それ以降の処理は実行されません。

```
% cat s14.sh
#!/bin/sh

exec /usr/bin/printf "done\n"
/usr/bin/printf "done2\n"
%
```

次のようにシェルスクリプトを実行すると、2つめのprintf(1)は実行されなくなります。

```
% ./s14.sh
done
%
```

ラッパースクリプトなどで、本来実行したいとしているソフトウェアを実行するときに使われる機能です。完全に置き換わるためプロセスとしての無駄がなくなるほか、終了ステータスが本来動作させようとしていたソフトウェアのものになるなど都合が良いからです。

シェル終了「exit」

シェルを終了するには組み込みコマンドexitを使います。

```
% cat s15.sh
#!/bin/sh

/usr/bin/printf "done\n"
exit
/usr/bin/printf "done2\n"
%
```

途中に組み込みコマンドexitがあると、その段階でシェルは終了します。

```
% ./s15.sh
done
%
```

exitには引数として0から255までの終了ステータスを指定できます。不正終了であることを表す場合など、0以外の値を指定してシェルを終了させることができます。

変数を環境変数へ格上げ「export」

変数はexport組み込みコマンドを使うことで環境変数へ変更できます。すでにある変数に対しては次のように指定することで環境変数へ

の変更を実施できます。

```
export a
```

新しく環境変数を作る場合には次のようにも指定できます。

```
export a=b
```

オプションの処理「getopts」

シェルスクリプトでオプションを処理するために組み込みコマンドがgetoptsです。**リスト2**のように使用します。

getoptsのあとにオプションの内容を指定しています。「:」はそのオプションが引数を取ることを意味しており、引数はOPTARG変数に格納されます。shiftの使い方など含めてこのまま使い回すタイプのコーディングです。

▼リスト2　オプション指定の処理に使われる組み込みコマンドgetopts

```
while getopts cs:n:a:wVh?v option
do
    case "$option" in
    c)
        colordiff_mode=true
        ;;
    s)
        waitsec=$OPTARG
        ;;
    n)
        colorhistnum=$OPTARG
        ;;
    a)
        alert_mode=true
        alertcommand=$OPTARG
        ;;
    w)
        fitwindowwidth=false
        ;;
    V)
        header_mode=true
        ;;
    h|\?)
        echo "$usage_msg" 1>&2
        exit 0
        ;;
    v)
        echo "topless version $version"
        exit 0
        ;;
    esac
done
shift $(($OPTIND - 1))
```

標準入力からのデータの読み込み「read」

標準入力からデータを読み込むための組み込みコマンドがreadです。次のように使用します。

```
% read ans
yes
% echo $ans
yes
%
```

次のものはreadの活用例です。

```
% read -p "[y/N]? = " ans
[y/N]? = y
% echo $ans
y
%
```

readのあとに指定した変数に値が格納されます。readは複数の引数を指定にとることができ、IFS区切りで分割された内容が設定されます。

```
% echo 1 2 3 4 5 |
> while read a b c d e
> do
>       echo $a
>       echo $c
>       echo $d
> done
1
3
4
%
```

IFSとread、whileを組み合わせると任意の文字列の分割といった処理も実現できますが、そういった処理は文字列操作を得意とするプログラミング言語やcut(1)、awk(1)、sed(1)などのコマンドを使ったほうが良いでしょう。

変数の一覧表示と変数の削除「set」「unset」

シェルが保持している変数の一覧は、組み込みコマンドsetで表示できます。

```
% set
BLOCKSIZE=K
EDITOR=vi
ENV=/home/sh/.shrc
HOME=/home/sh
IFS='
```

```
'
MAIL=/var/mail/sh
OPTIND=1
PAGER=more
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/games:/usr/local/sbin:/usr/local/bin:/home/sh/bin
PPID=79942
PS1='% '
PS2='> '
PS4='+ '
PWD=/usr/home/sh
SHELL=/bin/sh
TERM=xterm
USER=sh
_=%
ans=y
%
```

環境変数だけ知りたいという場合にはenv(1)コマンドを実行します。

```
% env
PAGER=more
MAIL=/var/mail/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/games:/usr/local/sbin:/usr/local/bin:/home/sh/bin
EDITOR=vi
ENV=/home/sh/.shrc
PWD=/usr/home/sh
TERM=xterm
USER=sh
HOME=/home/sh
SHELL=/bin/sh
BLOCKSIZE=K
%
```

変数も環境変数も組み込みコマンドunsetで削除できます。

```
% unset BLOCKSIZE
% unset EDITOR
% unset ENV
% unset MAIL
% unset OPTIND
% unset ans
% set
HOME=/home/sh
IFS='
'
PAGER=more
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/games:/usr/local/sbin:/usr/local/bin:/home/sh/bin
PPID=79942
PS1='% '
PS2='> '
PS4='+ '
PWD=/usr/home/sh
SHELL=/bin/sh
TERM=xterm
USER=sh
_=ans
%
```

シェルオプションの切り替え「set -/+」

組み込みコマンドsetは3つの異なる役割を負っています。変数以外に、シェルに指定されるオプションで切り替わる機能の有効無効の切り替え機能を持っています。たとえば、ashは-eオプションを指定して起動されると、コマンドの終了ステータスが0以外だった場合にはただちにシェルを終了するといった動作をします。シェルスクリプトの開発段階などでよく使われるオプションです。

```
% sh -e
$ true
$ false   ←終了ステータスが1
%         ←シェルが終了した
```

一部のコマンドは0以外で終了することがあるがそれはバグではない(たとえばgrep(1)など)といった場合、次のようにset +eとすると、-eオプションで有効になった機能が無効になります。

```
% sh -e
$ set +e
$ true
$ false   ←+eが指定されたので1でも終わらない
$ set -e
$ true
$ false   ←-eが指定されたので1で終了
%
```

処理が終わったらset -eとすることで再び-eオプションが指定されたのと同じ状態になります。一時的に機能を変更したい場合に使われます。

位置パラメータの変更「set --」

組み込みコマンドsetは位置パラメータを設定できるという機能も持っています。

```
% cat s16.sh
#!/bin/sh

echo $1 $2 $3 $4 $5
```

```
set -- a b c d e
echo $1 $2 $3 $4 $5
%
```

このように位置パラメータを設定すると、実行時に位置パラメータを変更できます。

```
% ./s16.sh 1 2 3 4 5
1 2 3 4 5
a b c d e
%
```

この機能は繰り返し構文で同じ処理を実施する場合に使われます。繰り返し構文の最初の段階で毎回「set -- $(eval echo \$any_vars$i)」のようにして値を設定し、繰り返し構文の中では$1や$2で値にアクセスします。繰り返し構文forで代入できる変数は1つしかありませんので、このようなしくみを入れることで複数の変数を使用できるようにしています。

シェルスクリプトとしてはかなり難しい書き方ですので、使う必要がないなら使わないほうが良いと思います。

バックグラウンドプロセスの終了待ち「wait」

バックグラウンドプロセスとしてソフトウェアを実行した場合に、それらバックグラウンドプロセスが終了するまで待つための組み込みコマンドがwaitです。子プロセスのバックグラウンドプロセスがすべて終了するまで待機します。

```
% sleep 10 &
% sleep 11 &
% wait
[1]   Done                    sleep 10
[2]   Done                    sleep 11
%
```

複数の処理を同時に実行させ、それらが終了するまでwaitで待つといった使い方をします。簡単に処理の並列化を実現できます。SD

COLUMN /usr/bin/cdが存在する理由

OSによっては、/usr/bin/cdのようにファイルシステム上に組み込みコマンドと同じ名前のソフトウェアが配置されていることがあります。ちなみにMac OS Xにも入っています。

```
% type cd
cd is a shell builtin  ←組み込みコマンド
% which cd
/usr/bin/cd  ←cdというコマンドがある
%
```

これは次のようなシェルスクリプトになっています。Mac OS Xではファイルシステムの特性に対応するためにもうちょっと複雑なシェルスクリプトが配置されています。

```
% cat /usr/bin/cd
#!/bin/sh
# $FreeBSD: stable/10/usr.bin/alias/
generic.sh 151635 2005-10-24 22:32:19Z
cperciva $
# This file is in the public domain.
builtin ${0##*/} ${1+"$@"}
%
```

書き方はちょっと難しいところがありますが、これは基本的に**リストA**の処理と同じです。

このスクリプトは意味がありません。組み込みコマンドはシェル自身が必要としている機能ですので、別プロセスとしてシェルを起動し、そこで同じ組み込みコマンドを実行したとしても、処理はその中で閉じていてほかにまったく影響を与えません。このシェルスクリプトは意味がないスクリプトです。

同様のファイルを探すとほかにも見つかります。

意味のないシェルスクリプトが存在している理由はPOSIX.1にあります。POSIX.1では上記組み込みコマンドを「標準の組み込みユーティリティ」と分類し、これらユーティリティはexec(3)ファミリーから利用できることが要求されています。つまりファイルとして存在していることを求めています。この仕様に準拠するために、こうした意味のないシェルスクリプトが用意されているのです。

▼リストA /usr/bin/cdはこのスクリプトを実行しているのと同じ

```
#!/bin/sh
cd "$@"
```

第5章 もっと自由に! 制御構文でプログラミング

本章では、シェルが持っている制御構文や割り込み処理、シェル関数について、その使い方としくみを解説します。ちょっとした繰り返し処理の実行やシェルスクリプトを書くときに役立ちますが、その特性を知っておかないと思わぬ落とし穴にはまることもあります。しっかりと理解しておきましょう。

シェル上でプログラミングは必要?

シェルスクリプトは手続き型プログラミング言語のようにごりごりとアルゴリズムを実装したり数値計算を実施するタイプのソフトウェアではありません。処理の主体はあくまでもほかのソフトウェアです。それらソフトウェアを起動したり組み合わせたりするのがシェルの仕事です。

ただし、シェルでも手続き型プログラミング言語的な処理が記述できるように制御構文(分岐構文、繰り返し構文)、割り込み処理、関数(シェル関数)といった機能が用意されています。こうした機能があるおかげでシェルスクリプトとしてさまざまな処理を組み上げることが可能になっています。使いすぎない程度にこれら機能を使うことが、扱いやすいシェルスクリプトを構築する1つのポイントです。

制御構文

シェルには分岐構文と繰り返し構文の制御構文が用意されています。Bourne Shell系の制御構文はあくまでもコマンドの実行を主体とするものです。

コマンドを実行した結果を受けて、どちらに分岐するか

コマンドを実行した結果を受けて、いつまで繰り返すか

それを決定します。このあたりは手続き型のプログラミング言語としくみが大きく異なります。

コマンド[(1)とtest(1)

制御構文を取り上げる前にtest(1)コマンドを取り上げる必要があります。test(1)コマンドは比較を実施し、その結果をコマンド終了ステータスで表現するコマンドです。比較が正しければ0が、正しくなければ1を終了ステータスとします。シェルではこのコマンドを使って制御構文の式としています。

test(1)コマンドと実態がまったく同じコマンドに[(1)があります。一見すると括弧の始まり側の記号で、プログラミング上の記号のように見えますが「[」という名前のファイルが存在しています。

図1の結果を見ればわかるように/bin/[も/bin/testも同じi-node番号を持っています。実態が同じで名前だけが違うファイル、つまりハー

▼図1 [(1)コマンドとtest(1)コマンドの実態は同じファイル

```
% which [
/bin/[
% which test
/bin/test
% ls -il /bin/[
24306 -r-xr-xr-x  2 root  wheel  11312 ↵
Nov 10 12:39 /bin/[
% ls -il /bin/test
24306 -r-xr-xr-x  2 root  wheel  11312 ↵
Nov 10 12:39 /bin/test
%
```

ドリンクされたファイルになっています。testと[は呼び出されたときの名前で挙動を変えるしくみになっています。[で呼び出されたときには最後の引数として]を指定しなければならないというしくみになっています。結果、見た目はプログラミング上のしくみのように記述できる、というトリックになっています(**図2**)。

test(1)のソースコードを見てみると、自分の名前(シェル的に言えば特殊パラメータ$0で表現される部分、C言語で言えばargv[0]から指し示される文字列ですが)が[であるかどうかを調べて、[である場合には]が最後の引数にあるかどうかをチェックしています(**リスト1**)。つまり、**["1" = "1"]**というコマンドは**test "1" = "1"**というコマンドと同じです。

▼図2　ファイル名が[で呼び出されたときは引数の最後に]が必要というしくみになっている

```
% [ "1" = "1" ]
% echo $?
0
% [ "1" = "0" ]
% echo $?
1
% test "1" = "1"
% echo $?
0
% test "1" = "0"
% echo $?
1
%
```

▼リスト1　/usr/src/bin/test/test.cにその処理が記載されていることを確認できる

```
int
main(int argc, char **argv)
{
    int     res;
    char    *p;

    if ((p = strrchr(argv[0], '/')) == ↵
NULL)
        p = argv[0];
    else
        p++;
    if (strcmp(p, "[") == 0) {
        if (strcmp(argv[--argc], "]") ↵
!= 0)
            error("missing ]");
        argv[argc] = NULL;
    }
```

分岐構文「if」

if分岐構文にはif-then-fi、if-then-else-fi、if-then-elif-then-fiという形式があります。ifとelifのあとに式(つまりコマンド)を指定し、式の評価結果(つまりコマンドの実行結果)が0であればthenのあとのコマンドを実行、そうでなければ次へ処理を移します。if-then-fiの基本的な使い方は次のとおりです。

```
% if true
> then
>       echo 1
> fi
1
%
```

elseを追加すると、式の評価結果(コマンドの実行結果)が0以外だった場合にelse以降の処理が実行されるようになります。たとえば次のようになります。

```
% if false
> then
>       echo 1
> else
>       echo 2
> fi
2
%
```

elifのあとには式(コマンド)を記述できます。何度も比較する場合はelifを繰り返し重ねていくことになります。

```
% if false
> then
>       echo 1
> elif false
> then
>       echo 2
> else
>       echo 3
> fi
3
%
```

いずれの場合もfiでif制御構文を閉めます。

分岐構文「case」

文字列の比較で処理をまとめて分岐する場合には**case**制御構文が使われます。次の使い方

が基本となります。

```
% case a in
> a)
>       echo 1
>       ;;
> b)
>       echo 2
>       ;;
> esac
1
%
```

caseでは「どれにも一致しない場合」という指定として***)**を指定します。

```
% case a in
> c)
>       echo 1
>       ;;
> *)
>       echo 2
>       ;;
> esac
2
%
```

)の前にはパターンを指定できるようになっています。*)はすべてに一致するパターンですので、どれにも一致しない場合にはこの部分に一致することになります。

caseのあとにコマンド置換を指定すればコマンドの実行結果でパターン分岐できるようになります。caseで指定するパターンは「|」でor指定になります。次のようにホスト名で処理を切り替えるとか、そういったことを実施できます。

```
% case $(hostname -s) in
> host1|host2|localhost)
>       echo 1
>       ;;
> *)
>       echo 2
>       ;;
> esac
2
%
```

caseでよく使われるのは[Yy][Ee][Ss]|[Yy])や[Nn][Oo]|[Nn])を指定しておいて、ユーザに「[y/N]?= 」のようなプロンプトを表示した場合の入力値がYESでもYesでもyesでもNoでもnでも分岐できるように処理するといったものです。シェルでパターンによる分岐を実施できる構文なので、ifではなくcaseを好んで使う方もいます。

分岐構文「&&」

&&はもっとも簡単な分岐処理です。&&の左のコマンドの終了ステータスが0の場合、&&の右側のコマンドが実行されます。0以外だった場合には実行されません。

```
% true && echo 1
1
% false && echo 1
%
```

次のように、&&の右側の処理を**{}**でグルーピングすると、実行する処理をまとめることができます。こうなるとif制御構文で処理を分岐した場合と近い処理ができるようになります。

```
% false && {
>       command1
>       command2
> }
%
```

分岐構文「||」

||は&&の逆の分岐処理です。**||**の左のコマンドの終了ステータスが0以外だった場合に、**||**の右側のコマンドが実行されます。0だった場合には実行されません。

```
% true || echo 1
% false || echo 1
1
%
```

&&と同様、**{}**でグルーピングすれば複数のコマンドを実行することもできます。これは**||**の左側のコマンドの処理が失敗した場合に、右側に**exit**と書いてシェルスクリプトを終了するとか、フェールオーバーの処理を実施するといった場合によく使われます。

```
% true || exit 1
%
```

&&と**||**は短く読みやすいので、シェルスク

リプトを書く場合に1つのポイントとなる書き方です。

繰り返し構文「for」

繰り返し構文forはin以降に記載した文字列を、forのあとに記載した変数に代入しつつ、doとdoneの間の処理を繰り返すという制御構文です。次のように使います。

```
% for i in a b c d
> do
>       echo $i
> done
a
b
c
d
%
```

inのあとに指定する文字列をコマンド置換にしたりパス名展開にして使うことが多い制御構文です。パス名展開を使用すると次のようなことができます[注1]。

```
% for i in /dev/ada?
> do
>       echo smartctl -a $i
> done
smartctl -a /dev/ada0
smartctl -a /dev/ada1
smartctl -a /dev/ada2
smartctl -a /dev/ada3
%
```

for繰り返し構文は繰り返しの個数がそれほど多くない場合に使われる傾向がある制御構文です。

繰り返し構文「while」

whileの次の式(コマンド)の終了ステータスが0である間doとdoneの間の処理を繰り返す制御構文がwhileです。test(1)や[]を使うと手続き型プログラミング言語のような書き方ができます。

```
% i=1
% while [ $i -lt 5 ]
> do
>       echo smartctl -a /dev/ada$i
>       i=$(( $i + 1 ))
> done
smartctl -a /dev/ada1
smartctl -a /dev/ada2
smartctl -a /dev/ada3
smartctl -a /dev/ada4
%
```

標準入力からデータを読み込むreadという組み込みコマンドを併用すると、次のようにパイプで接続して処理できます。whileで任意の個数、とくに大量のデータを処理する場合などに使われる方法です。

```
% ls /dev/ada?
/dev/ada0       /dev/ada1
/dev/ada2       /dev/ada3
% ls /dev/ada? |
> while read dev
> do
>       echo smartctl -a $dev
> done
smartctl -a /dev/ada0
smartctl -a /dev/ada1
smartctl -a /dev/ada2
smartctl -a /dev/ada3
%
```

ここで1つ注意してください。whileをパイプで挟んだ段階で、この処理はサブシェルで実行されます。上記の例だとlsがシェルの子プロセスとなって実行され、whileはサブシェルで実行されます。シェルはシステムコールを発行して、これら2つの子プロセス(サブシェルもシェルの子プロセスです)が終了するのを待ちます。つまり、パイプを挟んだ段階でシェルとwhile制御構文まわりは別プロセスに分岐しますので、変数を共有するといったことはできなくなります。

この挙動を知らずに、いままでwhileで動作していたものが、パイプに挟んだときから動作しなくなるという問題に遭遇して解決できなくなる方がいます。シェルはコマンドを組み合わせるソフトウェアであり、パイプとはそういった挙動になるものだということを覚えておきましょう。

注1) smartctlコマンドは-aオプションでデバイスの情報を取得します。

慣れの問題もありますし、頭の回転が速い方はどんな複雑なコードを書いても読めるものですが、制御構文はそれほど多様しないほうがよいといえます。シェルの制御構文は読みやすいとは言いがたいところがありますし、プロセスへの分離など気をつけて読む必要があることがあります。制御構文には頼り切らず、構造的にもっとすっと、上から下へコマンドを実行するだけで済むような構造へ状況を持って行くほうが賢明です。

割り込み処理

シェルは割り込みを処理する方法を提供しています。割り込みがあった段階で実行するコマンドを設定しておくことができます。このあたりのプログラミングはほかのプログラミング言語と比較しても簡単です。シェルがカーネルに近い部分で動作するソフトウェアであるという特性が出ているところです。

割り込み処理「trap」

割り込みは組み込みコマンド**trap**を使って処理します。**trap "コマンド" 割り込みシグナル番号**といったように使います。次のシェルスクリプトは、シグナル1(HUPシグナル)を受けるとHUPedという文字列を標準出力へ出力します。whileのところはビジーループです。ずっとwhileが回り続けます。

```
% cat s9.sh
#!/bin/sh

echo $$
trap "echo HUPed" 1

while :
do
done
%
```

割り込みシグナルは**signal**(3)のオンラインマニュアルにまとまっています(**man 3 signal**を実行してマニュアルが読めます)。デーモンのいくつかはこのシグナルを受け取ると設定ファイルの再読み込みを実施します。そういった用途で使われることがあるシグナル番号です。

作成したtrap使用スクリプトを実行してみましょう。

```
% ./s9.sh
77064
```

このスクリプトを実行すると、このようにスクリプトのプロセス番号が表示されますので、別のターミナルからこのプロセスに対して1シグナルを送信します。次のように**kill -1 プロセス番号**と指定することで、シグナル番号1で割り込みをかけることができます。

```
$ kill -1 77064
$
```

実行させておいたシェルスクリプトは次のようにシグナルを受けるごとにトラップ処理が走り、文字列を出力します。

```
% ./s9.sh
77064
HUPed
HUPed
HUPed
HUPed
```

割り込み処理は終了依頼を受けた場合に終了処理をするとか、ターミナルのウィンドウサイズが変更された場合に出力するテキストの加工部分の処理を変えるとか、そういった処理に使用できます。

シェル関数

シェル関数に関してはこれまでに何度か紹介してきました。シェル関数はエイリアスの次に優先順位が強い実行の単位です。コードはシェルのメモリ上に存在し、シェルと同じプロセスとして実行されます。処理をまとめておくサブルーチンに近い存在です。シェル関数はスコープに注意して扱う必要があります。

図3のシェル関数と変数を**図4**のように実行してみましょう。

実行結果から、シェルの位置パラメータとシェル関数の位置パラメータは独立した別の存在であることがわかります。そして、変数はシェルとシェル関数で共有されていることがわかります。シェル関数はシェルの変数に影響を与えるため取り扱いに注意が必要です。

では次に、**図5**のようなシェルスクリプトを**図6**のように実行してみましょう。

この例では、シェル関数はサブシェルで実行されます。シェル関数fが実行される前に、シェルはfork(2)で自分自身をコピーします。コピーされたサブシェル(子プロセス)には引数としてシェルに与えられた引数が同じように与えられています。サブシェルのシェル関数fはこの引数を処理します。

このため位置パラメータに関してはパイプをはさまない場合と同じ動作になりますが、変数の扱いが違ってきています。サブシェルで変数を書き換えてもシェルには影響しないため、変数aの出力結果はaになっています。このようにシェル関数を扱う場合はスコープに注意しておく必要があります。

シェル関数は便利な機能ですがこういった扱いにくい側面も抱えているため、利用を推奨しない開発手法も存在します。シェル関数を活用するフレームワークもあればその逆もあります。適材適所で利用すべきものですが、慣れないうちはあまり使わないほうがよいでしょう。

▼図3　変数やパラメータのスコープに注意

```
% cat s10.sh
f() {
        a=b
        echo $0
        echo $1
        shift; shift
        echo $1
        echo ----
}

a=a
f "$@"
echo $0
echo $1
echo $2
echo $a
%
```

▼図4　シェル関数の位置パラメータはシェルの位置パラメータとは独立している

```
% ./s10.sh 1 2 3 4 5
./s10.sh
1
3
----
./s10.sh
1
2
b
%
```

▼図5　複雑なスコープの例

```
% cat s11.sh
#!/bin/sh

f() {
        a=b
        echo $0
        echo $1
        shift; shift
        echo $1
        echo ----
}

a=a
f "$@" | cat
echo $0
echo $1
echo $2
echo $a
%
```

▼図6　記述の仕方でそのシェルで実行されるかサブシェルで実行されるのかが変わる

```
% ./s11.sh 1 2 3 4 5
./s11.sh
1
3
----
./s11.sh
1
2
a
%
```

エンジニア冥利に尽きる、シェル

シェルはエンジニアにとって気持ちのよい存在です。システムを自在に操るための要、エンジニアが実現したいと思いついたアイディアを最速で実現させてくれる最強のツールだからです。そしてシェルはスキルを上げた分だけこちらの要望に応えてくれます。

UNIX系のOSはC/C++やObjective-Cなどで開発され、シェルとシェルスクリプトで組み上げられています。このためシェルやシェルスクリプトへの理解を深めることは、OSそのものに対する理解に直結しています。いままでわからなかったしくみがわかることへの純粋な快感。まさにシェル、エンジニア冥利に尽きるというわけです。SD

COLUMN シェルスクリプト的な発想ってどんなもの

シェルはソフトウェアを起動することを目的としたソフトウェアです。シェルそのものでプログラミングをするという発想はあまり持たないほうがよいと言えます。ほかのコマンドについてよく知り、そのコマンドをシンプルな構造で呼び出して実行する。シェルスクリプトはそういった目的で組み上げたほうがよいでしょう。

制御構文、割り込み処理、シェル関数といった機能が用意されているのでシェルスクリプトで複雑なプログラムを組み上げることもできますが、シェル特有のスコープと、プロセスとして分岐した瞬間に関係性が途切れるといった独特のしくみが、シェル単体でそうした複雑なしくみを組み上げることの複雑性を引き上げているように思います。

1つのシェルスクリプトのサイズが1,000行を超えるといった状況もあまり健全とは言えないように思います。その場合、ある処理単位は別のシェルスクリプトとして実装したり、特定の処理をするソフトウェアを開発してそちらを呼び出すようにしたりと、構造そのものを変えたほうがよい時期に来ていると言えます。

これはコーディングルールに分類されるので一概には言えませんが、たとえば**リストA**のような分岐処理を記述したとします。よくあるif分岐です。

シェルスクリプト的には**リストB**のように書くこともできます。一見するとtest(1)を4回実行することになる**リストB**のほうが処理の無駄のように見えますが、その行だけで処理が完結しているところがポイントです。上下のコーディングに目を通さなくてもその行だけで条件とコマンドがまとまっています。あとから読み返したとき、この書き方は読む範囲が少なくてすみます。

上から下へコードを組むこと、上に巻き戻るようなコーディングはしないこと、分岐は最小限に抑えること、シェルスクリプトのサイズは小さく抑えること、シェルスクリプトで処理をこなすのではなくコマンドで簡潔に処理をさせること、などを考慮するのがシェルスクリプトコーディングのポイントです。

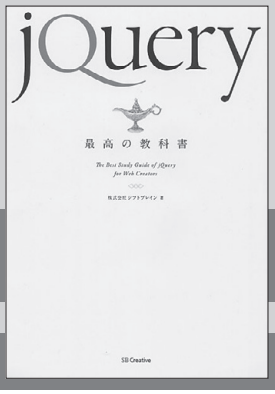
▼リストA　if分岐の実装例

```
if [ "1" = $a ]
then
        if [ "2" = $b ]
        then
                command1;
                ...;
        else
                command3;
                ...;
        fi
else
        if [ "2" = $b ]
        then
                command2;
                ...;
        else
                command4;
                ...;
        fi
fi
```

▼リストB　シェルスクリプト的なコーディングの例

```
[ "1"  = $a -a "2"  = $b ] && { command1; ...; }
[ "1" != $a -a "2"  = $b ] && { command2; ...; }
[ "1"  = $a -a "2" != $b ] && { command3; ...; }
[ "1" != $a -a "2" != $b ] && { command4; ...; }
```

BOOK no.1 jQuery最高の教科書



株式会社シフトブレイン 【著】
B5判、 320ページ／価格＝2,580円 （税別） ／発行＝SBクリエイティブ
ISBN＝978-4-7973-7221-2

本書はjQueryの初学者を対象とした入門書。JavaScriptの知識も前提としていないため、コードの書き方の基本（変数やthis、関数、制御文など）も解説している。これで十分というわけではないが、この1冊だけで知識ゼロからでもjQueryの魅力を体感できるはず。各所にあるコラムも初学者が迷いそうな事柄や補足情報をうまく拾い上げている。ある機能を学ぶために、まず基礎となるものを理解させ、そのうえで実践的なテクニックを習得できるよう段階をふむ。サンプルもモダンで実用的。章が進めばサンプルのクオリティも高くなり、モチベーションも上がってくる。学習ステップをしっかりと考えた作りが本書の持ち味だ。

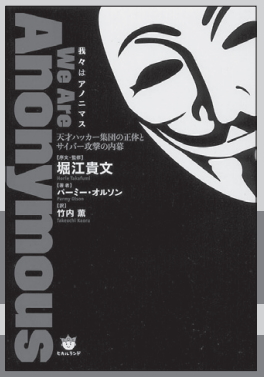
BOOK no.2 実例で学ぶゲーム開発に使える数学・物理学入門



加藤 潔 【著】
B5変形判、 328ページ／価格＝2,800円 （税別） ／発行＝翔泳社
ISBN＝978-4-7981-3086-6

物が落ちる、キャラクタが回転する、爆発して飛び散る、光が反射する……ゲームにはいろいろな要素があるが、実際にそれをどうプログラムすればいいのか。本書は、その関連付けを行えるように数学や物理学を利用するための導入書だ。扱っている処理は5つ（運動／スクロール／当たり判定／光線／画面切り替え）に分類され、それぞれに物理や数学の公式が学習でき、DirectXのサンプルプログラムがあるので動作を試すことができる。座標計算から微分・積分、三角関数、行列演算などをある程度知っていないと簡単に読み進めるわけにはいかないかもしれないが、ちゃんと勉強したい人のための1冊である。

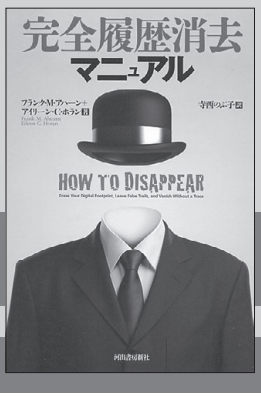
BOOK no.3 我々はアノニマス
天才ハッカー集団の正体とサイバー攻撃の内幕



パーミー・オルソン 【著】 ／竹内 薫 【訳】 ／堀江 貴文 【監修】
四六判、 477ページ／価格＝2,000円 （税別） ／発行＝ヒカルランド
ISBN＝978-4-86471-157-9

アノニマスの誕生から、最近のソニーへのサイバー攻撃やFBIとの攻防までを追ったノンフィクション。代表者もなく、決まったメンバーもいない。メンバー同士は互いの素性も知らない。そんな状況で、なぜあれだけ団結した行動を起こせるのか。本書では、彼らがことを起こす様子をつぶさに記述しているが、それでもそのメカニズムは謎だ。各人は正義感や興味本位、暇つぶしなど、それぞれの理由で攻撃に参加する。仲間割れもしょっちゅう起こる。だが、最終的には大手企業のサイトをダウンさせるような力を発揮する。まるで団体で1つの個性を成しているようだ。個々の技術的な話も興味深いが、組織の異様な生態にも注目してみてほしい。

BOOK no.4 完全履歴消去マニュアル



フランク・M・アハーン、 アイリーン・C・ホラン 【著】 ／寺西 のぶ子 【訳】
四六判、 240ページ／価格＝1,600円 （税別） ／発行＝河出書房新社
ISBN＝978-4-309-24639-0

スキップ・トレーサー（以下、ST）という職業をご存じだろうか？　人を探し出して個人情報を暴く職業のことだ。本書は元STの著者が自らのノウハウを公開し、いかにすればSTの手から逃れられるかを伝授する。結局、最も個人情報が漏れやすいのは、人の口から。STは追跡者本人や家族になりすまし、電話会社、銀行、役所などに電話し、個人情報を手に入れるという。我々は便利なサービスと引き換えに、いかに多くの個人情報を企業に与えているのかがよくわかる。誰かから逃れたいという人は、ぜひ本書を手に取ってほしい。けれど買うときにはくれぐれもクレジットカードは使わないように（足がつくから）。これも著者からのアドバイスだ。

第2特集

未来を作るITインフラ

10ギガビットを実現するケーブリング技術

クラウドコンピューティングの普及の影響で、ネットワークの広帯域／高速化が求められています。10GBASE-Tや光ファイバなど、10Gb/sを実現するネットワーク技術はいくつかあります。各技術の特性を理解し、適切な配線を行い、その能力を最大限に発揮できるようになりましょう。既存環境からのスムーズな移行、将来的な拡張、データセンタ内のエアフローなども考慮したケーブリング技術を解説します。

●入澤 徹（いりさわ とおる）
ザ・シーモン・カンパニー
URL www.siemon.com／www.siemon.co.jp

第1章 10GBASE-Tに求められるケーブリング技術 …… p.64

第2章 光ファイバ&ラックシステムにおける
将来を見据えたケーブルマネジメント …… p.73

第3章 高速アプリケーションに最適化した
新型RJ-45開発記 …… p.79

第1章 10GBASE-Tに求められるケーブリング技術

ギガビットの高速ネットワークにはいくつかの技術がありますが、そのなかでも低価格／省消費電力の10GBASE-Tが注目されています。しかし、その長所／短所を理解して正しく利用しなければ、十分な性能を発揮できません。本章では10GBASE-Tの特徴とケーブリングにおけるポイントを整理します。

入澤 徹(いりさわ とおる)　ザ・シーモン・カンパニー

求められるEthernetの高速化の要求と課題

コンピュータ同士を接続し、通信手段の1つとして使うことを提案したしくみが、Ethernetと呼ばれる通信規格です。厳密にはEthernetとは10BASE5のことを指します。

1980年代半ばから10BASE5や10BASE2と言われる同軸ケーブルを使ったEthernetが市場に現れました。次に10BASE5のスター型トポロジを取り込む形で、現在最も主流となったツイストペアケーブル上で通信を行う10BASE-Tが作られ、そして、現在の10GBASE-Tまで進化を遂げていくこととなります。

太く高価な同軸ケーブルから細くて扱いやすいツイストペアケーブルに進化し、その通信速度も20年間で10メガから100メガ、100メガから1ギガ、1ギガから10ギガと10倍単位で増加しています。これは市場が通信の高速化を顕著に求めた結果です。E-mailから始まったインターネットの発展が、大容量で高速な通信を求めています。国際的な通信規格について話し合う組織IEEE(米国電気電子技術者協会)では次世代の通信規格40ギガ／100ギガビットEthernetについて調整段階にあります。また、その規格の一部はすでに光ファイバ上での通信で達成されています。

本稿では、ますます高速化するEthernetと、高速化とともに高密度化を求められる配線の課題とその解決方法を考えていきたいと思います。

まずは、アプリケーション[注1]の速度とメディアの関係について**表1**をご覧ください。「XXBASE-T」の「XX」はスピードを、「T」は銅線ツイストペア上で送受信することを表しています。

表2はEthernetとファイバチャネルの各々のデータレートの許容最大ロス値と通信距離の関係を示しています。通信経路や接続形態(**表2**のチャネルリンク内のMTPカセット／MTPアダプタ数)を設計するうえで、この表を参考にOS2／OM3／OM4、または低損失タイプを選択し、各々の通信速度を確保することが大切です。第2章で説明する10Gb/sから40/100Gb/sにシステムを移行する場合は、40/100Gb/sの通信距離に注意する必要があります。

アプリケーションの高速化とともにケーブリングシステムは改良を重ねてきました。それは、高速アプリケーションが正常に動作するための導線を確保することでした。

漏話やNEXTという言葉を耳にしたことがあると思いますが、銅線ツイストペアケーブル伝送に最も悪影響を及ぼすのは送信信号が受信

▼表1　アプリケーションの速度とメディアの関係

アプリケーション	スピード	通信ペア数	カテゴリ
10BASE-T	10Mb/s	2	カテゴリ3/5e以上
100BASE-T	100Mb/s	2	カテゴリ5e以上
1000BASE-T	1Gb/s	4	カテゴリ5e以上
10GBASE-T	10Gb/s	4	カテゴリ6A/7/7A

注1) 本稿でのアプリケーションとは「アプリケーションソフトウェア」ではなく、「通信アプリケーション」(ケーブリングシステム上で動作するサービスのこと。具体的には、10GBASE-Tや40GBASE-Tなど)の意味で用いる。

▼表2　光ファイバMTPプラグアンドプレー標準損失タイプ＆低損失タイプの距離とチャネルロス一覧

シーモン社システム		コネクタグレード	MTP-LCカセット最大ロス値(dB)	MTPアダプタ最大ロス値(dB)	ケーブル減衰値(dB/km)	データレート	チャネルリンク内のMTPカセット／MTPアダプタ数と850nm(MM)/1310nm(SM)波長での最大チャネルリンク長(m)							最大チャネルリンク長(m)	最大チャネルロス(dB)
							2	3	4	5	6	7	8		
イーサネット	OM3 XGLO 300 50/125 μm(MM)	低損失	0.35		3.0	1GB	1000	1000	1000	910	800	680	560	1000	4.50
			0.35		3.0	10GB	300	300	300	283				300	2.60
				0.20	3.0	40/100GB	100	100	100	100	100	100	100	100	1.90
		標準損失	0.65		3.0	1GB	1000	850	630	410	200			1000	4.50
			0.65		3.0	10GB	300	116						300	2.60
				0.40	3.0	40/100GB	100	100	100					100	1.90
	OM4 XGLO 550 50/125 μm(MM)	低損失	0.35		3.0	1GB	1100	1100	1100	1000	900	780	650	1100	4.80
			0.35		3.0	10GB	550	550	550	450	350			550	3.15
				0.20	3.0	40/100GB	150	150	150	150	100	32		150	1.50
		標準損失	0.65		3.0	1GB	1100	950	730	510				1100	4.80
			0.65		3.0	10GB	550	300	90					550	3.15
				0.40	3.0	40/100GB	150	15						150	1.50
	XGLO OS2 9/125 μm (SM)	標準損失	1.0		0.4	10GB	10000	8000	5500	3000	500			10000	6.20
			1.0		0.4	40GB	10000							10000	6.70
			1.0		0.4	100GB	10000							10000	6.30
ファイバチャネル	OM3 XGLO 300 50/125 μm(MM)	低損失	0.35		3.0	8GFC	180	150	150	125	90	90	35	180	※
			0.35		3.0	16GFC	120	100	100	75	40	40		120	
			0.35		3.0	32GFC	80	70	70	60	45	15	15	80	
				0.20	3.0	32GFC	80	80	80	80	70	70	60	80	
		標準損失	0.65		3.0	8GFC	150	125	35					150	
			0.65		3.0	16GFC	100	75							
			0.65		3.0	32GFC	70	60	15					70	
				0.40	3.0	32GFC	80	70	60	60	45	15		80	
	OM4 XGLO 550 50/125 μm(MM)	低損失	0.35		3.0	8GFC	220	190	190	160	120	120	50	220	
			0.35		3.0	16GFC	150	125	125	100	50	50		150	
			0.35		3.0	32GFC	110	100	100	80	65	20	20	110	
				0.20	3.0	32GFC	110	110	110	110	100	100	80	110	
		標準損失	0.65		3.0	8GFC	190	160	50					190	
			0.65		3.0	16GFC	125	100						125	
			0.65		3.0	32GFC	100	80	20					100	
				0.40	3.0	32GFC	110	100	80	80	65	20		110	
	XGLO OS2 9/125 μm (SM)	標準損失	1.0		0.4	8GFC	1400							1400	2.60
			1.0		0.4	8GFC	10000	8500	6000	3500	1000			10000	6.40
			1.0		0.4	16GFC	10000	8500	6000	3500	1000			10000	6.40
			1.0		0.4	32GFC	10000	8250						10000	6.34

※ファイバチャネルのロスバジェットについては、ANSI INCITSのFC-PI-5の表B.3とFC-PI-6の表B.1のコネクションロス値を参照のこと。

信号を妨害することで、高周波になるほどその影響を受けやすくなります。また、PoE(Power over Ethernet)と呼ばれる電力を供給できるアプリケーションも登場し、市場はより便利にユーザフレンドリーに変化しています。一方で、完全に妨害を排除した高速通信の実現のために光ファイバの技術が発展してきました。こちらは信頼性の高い伝送でファイバチャネルなどさまざまなアプリケーションをサポートしています。

10Mb/sから10Gb/sや100Gb/sに高速化したうえに電力供給することでケーブリングシステムも大きく変化しています。ケーブリングシステムは短いライフサイクルで進化するアプリケーションを安定的に運用し、長期間に渡りサービスを提供できる強固な通信基盤でなければなりません。

表1で気をつけなければならないのは、チャネルリンクのカテゴリは、最下位カテゴリに準拠します。つまり、カテゴリ6Aのリンクにカテゴリ5eのパッチコードをつなぐとその通信チャネルはカテゴリ5eの通信性能を持つことになってしまいます。光ファイバリンクでも同じことです。高価な高速通信機器を購入してもパッチコード1本で下位アプリケーションとオートネゴシエーションしてしまうことに注意したいですね。

このようにケーブリングシステムは、期待した性能が得られないことが多く存在します。初期性能を保ったままケーブル設備を長期間運用するためのポイントとは何なのでしょうか？

COLUMN

ケーブリング基礎用語 ①

・パッチパネル
構内情報配線網のケーブル群を収納し接続するための配線盤。バックボーンLANやPBXから配線した1次側端子と、個々のユーザの席にあるLANや電話コネクタから配線をした2次側端子がある。1次側端子と2次側端子をパッチコードで接続する。パッチコードをつなぎかえるだけで配線を変更できる

・パッチコード
両端にモジュラプラグを取りつけたパッチパネルやスイッチ類との間を配線するケーブル。伝送機器を配線盤に接続する場合には「機器コード」、PCなどの端末機器を通信アウトレットに接続する場合には「ワークエリアコード」と呼ぶ

・通信アウトレット
ワークエリア(ユーザがPCなどの通信端末機器を扱うビル内の領域)で水平ケーブルを成端し、端末接続配線へのインターフェースを提供する接続部品

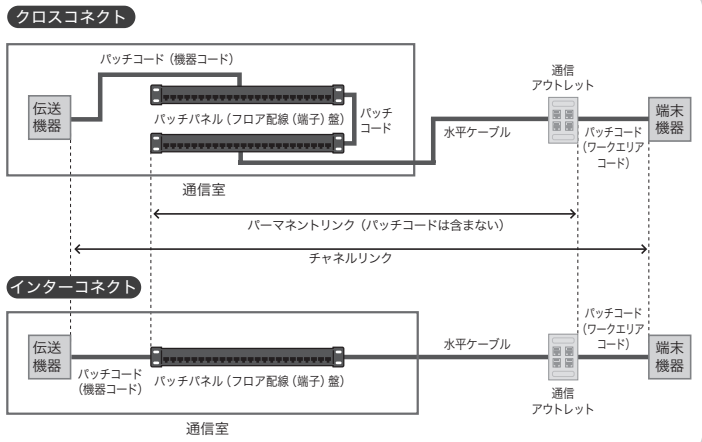
・水平配線
同一フロア内にて配線すること。フロア配線盤(パッチパネル)を通信アウトレットに接続するケーブルを水平ケーブルという

・クロスコネクト
水平ケーブルと機器コードを、パッチパネルを介したパッチコードなどで接続する方法(**図1**)

・インターコネクト
パッチパネルを介したクロスコネクトを省略し、ケーブル端末のジャック側からハブなどの機器類に直接接続する方法(**図1**)

・成端
モジュラージャック、モジュラープラグを取り付けること

▼図1　クロスコネクトとインターコネクトの概略図



10GBASE-Tとは

10ギガアプリケーションは、**表3**のように光ファイバやSFPコネクタを使ったインターコネクションなどがあり、近年多様化していますが、低価格／省消費電力で100メートルの通信距離をサポートする経済的な10GBASE-Tが今、一番注目されています。

10GBASE-Tはツイストペア上で10Gb/sの通信を行うEthernetアプリケーションです。2006年6月にIEEE 802.3anで定められました。ISO/IEC ClassEAまたはTIA/EIA Category 6A[注2]以上のケーブリングシステムで、100メートルの通信距離をサポートします。IEEE 802.at PoE+をサポートし、最高で57V/30Wの電力の転送が可能です。

拡張性を高める オートネゴシエーションの落とし穴

さらに注目すべき点は、オートネゴシエーション機能を持ち、10Gb/s以下のネットワークカードをスピードを選ばず1台のスイッチでマルチに通信ができることです。ほかのアプリケーションのようにスイッチとサーバまたは、ネットワークカードを同時に買い替える必要がなく機器の

▼表3　10ギガアプリケーションの種類

アプリケーション	メディアタイプ	伝送距離
10GBASE-T	カテゴリ6A UTP	100m
	カテゴリ6A F/UTP	
	カテゴリ7A S/FTP	
10GBASE-CR	インタコネクトSFP+	7m
10GBASE-S	OM3 マルチモード	300m/OM3
	OM4 マルチモード	400m/OM4
10GBASE-L	0.4dB/km シングルモード	10km
10GBASE-E	シングルモード5-11dB	40km

注2) ISOはInternational Organization for Standardization(国際標準化機構)、IECはInternational Electrotechnical Commission(国際電気標準会議)、TIAはTelecommunications Industry Association(米国電気通信工業会)、EIAはElectronic Industries Alliance(米国電子機械工業会)。

ライフサイクルに合った柔軟性に富んだネットワークを構築できます。

しかし、ネットワークカードが10Gb/s対応でもケーブルのカテゴリが対応していなければ期待したスピードは出ません。カテゴリ5eのケーブルでは1Gb/sの伝送速度で通信してしまうのは、言うまでもありません。

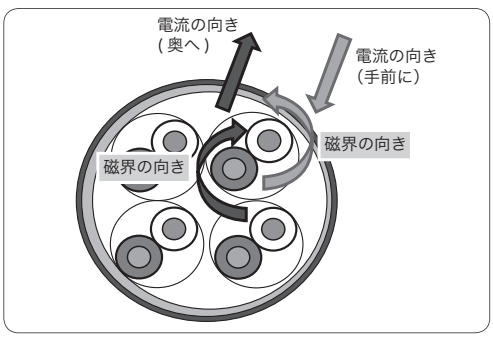
市場の要求

10GBASE-Tに対する市場からの要求は日に日に高まっています。それはほかの10ギガアプリケーションに比べて機器、ネットワークカード、ケーブル設備を含めて導入費用が非常にリーズナブルであることが挙げられます。マルチジェネレーションのサポートやLOM(LAN On Motherboard)チップの導入もその要求を加速させています。

10GBASE-Tの最新省電力技術

10GBASE-Tの普及が遅れた原因の1つに高い消費電力がありましたが、近年のPHYチップの微細化技術やWoL(Wake On LAN)、EEE(Energy Efficient Ethernet)、ショートリーチモードなどの技術を使い、電力消費量の削減が図られています。最新のPHYチップでは1ポートあたり1.5W以下の電力消費量に抑えられた製品も出荷予定があります。これは、開発当初のPHYチップと比べて約85%の電力削減になります。この技術により48ポートを収容する高密度タイプのスイッチが市場に登場しています。

▼図2　各導線における電流の向きと磁界の向き



10GBASE-Tを長期間運用するために

クロストーク／エイリアンクロストーク

10GBASE-Tに限らずクロストーク(漏話)は、通信にとって大きな問題です。一般にクロストークとは、任意のケーブル内で発生する送信信号が受信信号に悪影響を与える雑音のことです。近端漏話や遠端漏話とも呼ばれています。クロストークをいかに発生させないか、または、排除するかが、ケーブリングシステムにとって最も大きな課題です。

10GBASE-Tは、その符号化技術からケーブル間の雑音でさえ敏感に影響されてしまいます。それは、雑音と信号を区別するのが困難なほどの高周波帯域を使う通信アプリケーションだからです。このケーブル間のクロストークをエイリアンクロストーク(外部からのクロストーク)と呼んでいます。

理科の授業を思い出してみましょう。磁力線は、電線上を流れる電流の方向に時計回りで磁力が発生します(右ネジの法則)。

ケーブルの各ペアも同じです。ペアの各導線(コンダクタ)は、各々が右から左、左から右に電流が流れます(**図2**)。磁力線が発生するのと同じ原理でノイズが発生します。ペア間で発生するノイズをディファレンシャルモードと呼びます。この電気の流れがすべてのペア、すべての導線で一定の強さであれば導線間、ペア間、ケーブル間のバランス(平衡)は保たれ各々の磁界は打ち消し合う性質からクロストークは発生しません。しかし現実は、すべて打ち消し合うことはできません。ゆえに常に通信設備は、クロストークに悩まされるのです。それはどの地点どの時点においても常に一定の電圧、電流を得ることが現実的ではないからです。環境からの電磁波を含めてどのような状況でも常に同じ力を発生させることは不可能です。

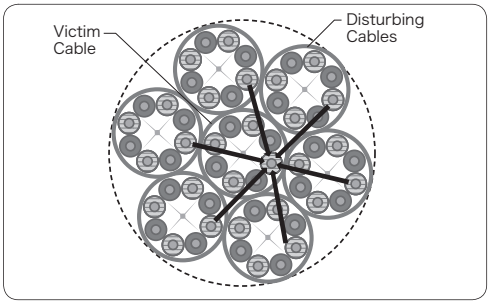
このため、ケーブリングの世界では、各ペア

をツイストさせる(撚る)ことにより電気的に平衡を保つように工夫をしています。それがクロストークを避けながらうまく付き合う方法です。カテゴリXXケーブルが平衡ツイストペア(Balanced Twist Pair)ケーブルと呼ばれるのは、このことからです。

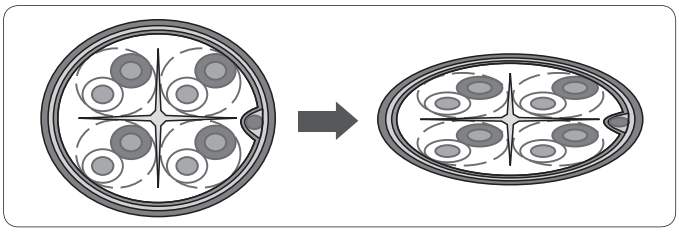
エイリアンクロストークは6 around 1(シックスアラウンドワン)と呼ばれる**図3**のような状況(影響を受けるケーブルが影響を及ぼすケーブルに囲まれている)で最も強いクロストークの影響を受けます。1つに結束されたケーブル束の本数が多ければ多いほど、その束の中に多くの6 around 1が存在することになります。同じ色のペアの影響を最も受けやすいと言われているのは、同じ色のペアは同じピッチでツイストされているからです。

また、困ったことにケーブル間のクロストーク、エイリアンクロストークは、周囲の環境にあるノイズを引き寄せる性質を持っています。電源ケーブルを含む電源設備、冷暖房空調設備、ケーブルテレビやセキュリティ設備、モーターなど影響を及ぼす物は無数にあります。より良い環境で通信を行うためには、このようなノイズの影響を受けない方法を考えなければなりません。

▼図3　6 around 1におけるエイリアンクロストーク



▼図4　ケーブルが押しつぶされるとツイストペアの形状も崩れる



それはケーブリングシステムの設計段階でいくつかの方法の中から選択していくことになります。

ケーブル形状を保つ

クロストークの影響を受けないために最も考慮する点は、ツイストペアのピッチを崩さないことです。ケーブルは平衡を保つために各ペアごとに一定の間隔でツイストされていますが、そのペアが崩れてしまえば平衡が保たれなくなり、性能が低下し通信障害が起きてしまいます。

ケーブルが押しつぶされて形状が変化してしまうとツイストペアの形状も崩れてしまい、著しく性能が低下します(**図4**)。これは経年劣化の特徴的な事象です。施工直後の性能検査に合格しても2年後、3年後に性能を満たしているとは限りません。径路の選択ではできるだけ梯子(はしご)タイプではなくメッシュ形状のケーブルトレーを選択するとともに、ケーブルメーカーに何ミリまでケーブルの積み上げが可能かあらかじめチェックし、設計の基準を設けることを勧めます。シーモン社では150ミリまで積み上げが可能で、その性能を20年間保証しています。

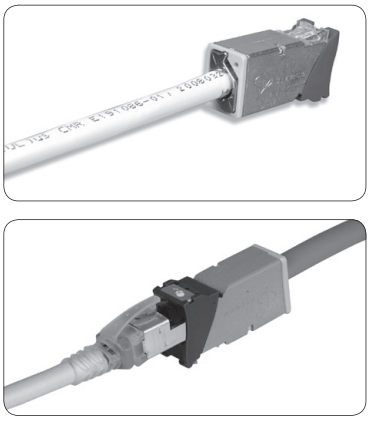
熱の影響

熱による性能低下にも注意が必要です。ケーブルはPoEなどの電力を流すことによって、その線径が細ければ細いほど導線が熱せられます。また、外部の環境に影響されて熱せられる

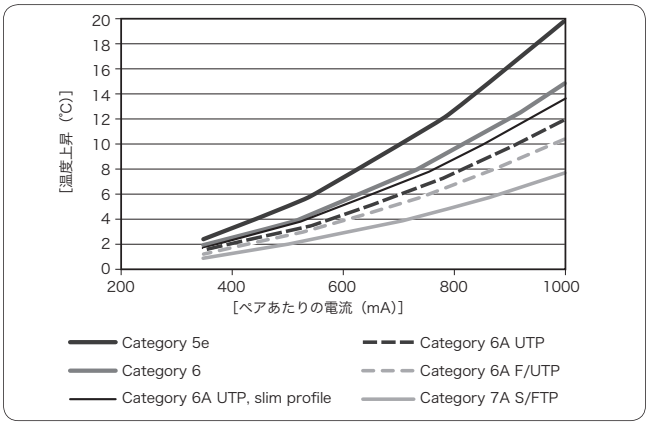
▶写真1　ケーブルを性能低下させる配線の例



▼ 写真2　シーモン社Z-MAXの接続点



▼ 図5　電流と温度上昇（ケーブル100本束）



ことも考慮しなければなりません。たとえばサーバや通信機器から排出される排気熱は、ケーブルの温度上昇を加速させます。

ISO/TIAでは一般的なUTPケーブルの温度が40℃から60℃の間でケーブル温度が1℃上昇するごとに挿入損失が0.6%増えると規定しています。ツイストペアケーブルの挿入損失が増えることによって、通信距離は短くなります。**写真1**のラック内配線のような状態では、ケーブルの性能はおろか機器の性能も低下し、さらにライフサイクルも短くなってしまいます。

接続点

モジュラーパッチコードやモジュラージャックの接続点（**写真2**）は、最もケーブルの平衡状態を崩しやすい点であることはすぐに想像できます。この接続点は人の手に多く触れられるので経年劣化の進みやすいポイントであることは間違いありません。このポイントをコントロールするには、信頼性の高いコネクタやパッチコードを使うことと、トレーニングを積み高い設計力と施工技術を持った施工者が必要条件となるでしょう。それでは、10GBASE-T、PoE+をサポートするコンポネント（配線材料）、設計と施工について考えていきましょう。

コンポネント（配線材料）

コンポネントはカテゴリ6A以上の性能を持つ材料で構成し、ケーブルは500MHz以上の帯域でエイリアンクロストークの影響を受けず、なおかつ熱消散に優れたシールドケーブルで導線が23AWG（0.57ミリ）以上の線径のものがお勧めです。

図5のグラフにあるようにUTPに比べシールドは熱消散性に優れています。また、シールドはその本来の役割としてエイリアンクロストークを遮断します。UTPではエイリアンクロストークの影響を遠ざけることはできても遮断することは適いません。

23AWGの導線は挿入損失を改善し、クロストークの影響を受けにくくすると同時にPoEによる発熱を抑えます。

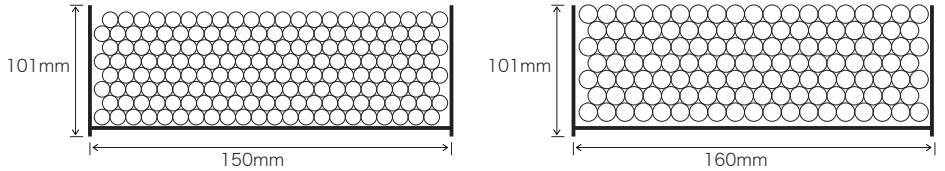
設計／施工

径路のケーブルの最大収容量は、次の式で求められます。

ケーブル断面積＝πr^2
＝3.14×（ケーブル外径/2）^{2}
ケーブルトレイ断面積＝幅×深さ
収容本数（計算値）＝（ケーブルトレイ断面積×50%）/ケーブル断面積

このときにトレイの内側の深さは、150ミリを超えてはいけません。このように設計に制限を加えることでケーブルの変形を防ぎます。具体的な例を**図6**に示します。

▼図6　径路のケーブルの最大収容量



150×101mm のケーブルトレイでは外径 7.4mm のケーブルの最大収容量は 176 本

160×101mm のケーブルトレイでは外径 8.9mm のケーブルの最大収容量は 130 本

▼表4　パッチコードの最小曲げ半径

パッチコード(シーモン社)	最小曲げ半径
Z-MAX 6A UTP	25mm
Z-MAX 6A シールド	50mm

ケーブルの最小曲げ半径は、TIAの規定ではケーブル外径の4倍です。長期間、期待した性能を保つための重要な要件です。

パッチコードの最小曲げ半径は、**表4**のようになります。パッチコードとパーマネントケーブルの最小曲げ半径は異なる点に注意しましょう。

システム設計

導入するケーブリングシステムの構成は、インターコネクトまたはクロスコネクトです。これは運用の安全性とラックの種類／スペースなどの物理空間によって決定されます。**図7-1〜7-4**をご覧ください。カテゴリ6Aの設計に必要なトポロジーごとの距離制限とコンポネントが記載されています。

コンソリデーションポイント[注3]はMAC(Move-Add-Change)つまり、機器の移設／追加／変更の頻度に基づき設計します。

ケーブル長の制限について各規格は、通信性能を確保するために次の規定を設けています。

・最長水平配線長：90メートル
・最短水平配線長：15メートル
・パッチコードの合計長：10メートル
・最短パッチコード長：2メートル

高い性能のコンポネント(シーモン社のカテゴリ6Aシステム)で構成した場合、この距離は**図7-1〜7-4**に示したケーブル長に変化します。ここで注目すべき点はケーブル長、パッチコード長ともに短くなっても性能を保証している点です。これはデータセンタ設備を意識してラック内やラック間で実際に必要な短距離のチャネ

▼図7-1　2コネクタ



	アイテム	カテゴリ6A規格値最小／最長	カテゴリ6A最小値	カテゴリ6Aショートリンク
機器コード	①Z-MAXパッチコード	2m/10m	1m	2m
水平配線	②Z-MAXパッチパネル	15m/90m	15m	3m
	③カテゴリ6Aケーブル			
	④Z-MAXアウトレット			
ワークエリアコード	⑤Z-MAXパッチコード	2m/10m	1m	2m

補足事項(図7-1〜7-4共通)
・チャネルリンクの最大長は100メートルを超えないこととします
・コンソリデーションケーブル＋水平配線ケーブルの最大長は90メートルを超えないこととします
・機器コード／クロスコネクトコード／ワークエリアコードの合計は10メートル超えないこととします

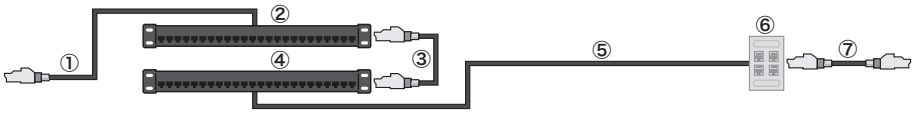
注3) パーマネントリンク内の中間コネクションポイント。図7-2でわかるように配線盤とアウトレットの任意の点にコネクションポイントを置きアウトレットやケーブル移設の際にケーブルにダメージを与えず短時間で施工が完了する。またアウトレットの新設の際に施工時間を短縮するための設備でTIA、ISOともに認めている正式な施工方法。

▼図7-2　3コネクタ／コンソリデーションポイント



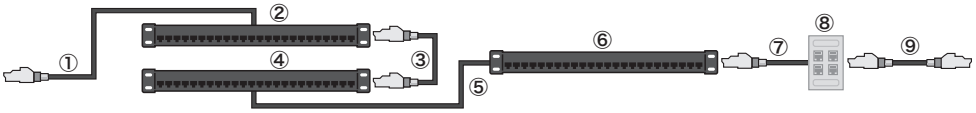
	アイテム	カテゴリ6A規格値最小／最長	カテゴリ6A最小値	カテゴリ6Aショートリンク
機器コード	①Z-MAXパッチコード	2m/10m	1m	N/A
水平配線	②④Z-MAXパッチパネル	15m/85m	15m	N/A
	③カテゴリ6Aケーブル			
コンソリデーションポイントケーブル	⑤Z-MAX単線パッチコード	5m/75m	5m	N/A
	⑥Z-MAXアウトレット			
ワークエリアコード	⑦Z-MAXパッチコード	2m/10m	1m	N/A

▼図7-3　3コネクタ／クロスコネクト



	アイテム	カテゴリ6A規格値最小／最長	カテゴリ6A最小値	カテゴリ6Aショートリンク
機器コード	①Z-MAX単線パッチコード	2m/10m	1m	3m
	②Z-MAXパッチパネル			
クロスコネクトコード	③Z-MAXパッチコード	2m/10m	1m	2m
水平配線	④Z-MAXパッチパネル	15m/90m	15m	3m
	⑤カテゴリ6Aケーブル			
	⑥Z-MAXアウトレット			
ワークエリアコード	⑦Z-MAXパッチコード	2m/10m	1m	2m

▼図7-4　4コネクタ／コンソリデーションポイント



	アイテム	カテゴリ6A規格値最小／最長	カテゴリ6A最小値	カテゴリ6Aショートリンク
機器コード	①Z-MAX単線パッチコード	2m/10m	2m	N/A
	②Z-MAXパッチパネル			
クロスコネクトコード	③Z-MAXパッチコード	2m/10m	2m	N/A
水平配線	④⑥Z-MAXパッチパネル	15m/85m	15m	N/A
	⑤カテゴリ6Aケーブル			
コンソリデーションポイントケーブル	⑦Z-MAX単線パッチコード	5m/75m	5m	N/A
	⑧Z-MAXアウトレット			
ワークエリアコード	⑨Z-MAXパッチコード	2m/10m	2m	N/A

ルリンクの性能を保証するためです。

経済的なショートリンクに注目

ショートリンクとは、データセンタのように15メートル未満のケーブル長で配線をすることを言います。シーモン社Z-MAXカテゴリ6Aショートリンクを利用すれば、隣接したラック間の短距離で15メートルのケーブル長ではなく、その距離に応じたケーブル長を選択できます。たとえば3メートルのケーブル長を利用した場合は、12メートルのケーブルを無駄にすることなくラック間の敷設ができます。これは、ラック間に24本のケーブルを敷設した場合に約288メートルのケーブルを節約できます。

300メートル巻きのケーブルであれば、1巻以上の無駄を削減することになります。グリーンな設備には必要不可欠な取り組みですね。

ケーブル径路の共有

一般に同一のケーブル束に細いケーブルが混在することにより、10GBASE-Tはエイリアンクロストークの影響を受けやすくなると言われてきました。その事実を検証する必要があります。シーモン社が提供するカテゴリ6A Z-MAXシリーズ UTPのコンポネントで検証した結果は、次のとおりです。

カテゴリ6A UTPの製品設計と施工方法はエイリアンクロストークの最小化に大きく影響します。シーモンラボでは継続的にカテゴリ6Aケーブル性能とエイリアンクロストークの研究をしていますが、カテゴリ6A UTPケーブルとほかのカテゴリ(5e、6)UTPケーブルを同一の束にし、どのような影響があるか評価試験を続けています。

パワースペクトル分析を行いワーストケース(4コネクトトポロジ)で、カテゴリ6A UTPケーブルとそのほかのカテゴリUTPケーブルを束にし、すべてのカテゴリUTPケーブルに10GBASE-Tの信号を伝送するときに、エイリアンクロストークの問題が発生するという結果が得られました。

カテゴリ6A未満のUTPケーブルが10GBASE-Tではないアプリケーション(たとえば1000BASE-T、100BASE-TX、10BASE-T、ATM、DSL、ベースバンドビデオ、RS 485、アナログやデジタルテレフォン)を通信し、カテゴリ6A UTPケーブルが10GBASE-Tの通信をしているときにはエイリアンクロストークの問題は発生しませんでした。

このため、低いカテゴリUTPケーブルが10GBASE-Tの信号を伝送しない限り、カテゴリ6A UTPケーブルとその他カテゴリUTPケーブルを同一の束にまとめて、同じ経路を共有できます。しかし、カテゴリ6A UTPとそのほかのカテゴリのUTPケーブルが同時に10GBASE-T信号を伝送する場合は、ケーブルトレイなどの経路は共有可能ですが、異なるカテゴリケーブルごとにケーブルを束にし共有径路内でカテゴリごとに離隔をとるか、別の配線ルートを設けてください。

上記の要求はUTPケーブルのみに適応されます。カテゴリ6A F/UTP(シールド)やカテゴリ7 S/FTP(完全シールド遮蔽)ケーブルを使用する場合には、対応する必要はありません。同じ通信ラック内に既存の1Gb/sスイッチと10Gb/sスイッチを導入する場合は、エイリアンクロストークの影響を考え、カテゴリ6AF/UTPを導入することをお勧めします。

カテゴリ6A未満で10GBASE-Tを運用する

カテゴリ6や低いカテゴリで10GBASE-Tの伝送を行う場合は、別途エイリアンクロストークの試験を行う必要があります。この試験は前述した6 around 1を仮想してサンプリングテストを行うのが一般的ですが、時間のかかるたいへんな作業です。注意しなければならないのは、時間の経過とともに10GBASE-Tの運用が増えていった場合、同一の束にすでに10GBASE-Tを伝送している低いカテゴリのチャネルリンクが混在しているとエイリアンクロストークの影響をテストするためにそのリンクを遮断するリスクが伴ってきます。低いカテゴリと6A UTPが同じ束にあり10Gを運用している場合も同様です。低いカテゴリで10GBASE-Tを運用する場合は、調査と試験に多くの時間を割かなければならないこと、伝送不具合のリスクの割合が6Aの設備に比べ高いことを考慮してください。また、各規格団体は10Gb/sの通信をカテゴリ6A未満の設備で運用することを推奨していません。カテゴリ6Aのコンポネントを使いチャネルリンク性能が規格値を満足していれば、このような試験は必要ありません。

詳細な試験方法は、エイリアンクロストークの試験機を製造販売しているメーカーに問い合わせてください。SD

第2章 光ファイバ&ラックシステムにおける 将来を見据えたケーブルマネジメント

本章では具体的なケーブリングノウハウを紹介します。1つめは、一度の設備投資で将来のアプリケーション導入にも対応できる光ファイバソリューション。2つめは、データセンタ内で機器の熱排気を効率よく循環させるためのラック内ケーブルマネジメントです。

入澤 徹(いりさわ とおる)　ザ・シーモン・カンパニー

光ファイバソリューション

最新の光ファイバソリューションでは、まず次世代の通信規格をサポートするケーブリングシステムに、どうやってスムーズに移行するかを考慮しなければなりません。第1章のツイストペアシステムでも書いたとおり、ケーブリングシステムは必ず下位互換性を維持しています。つまり、上位のケーブリングシステムを導入すれば、設備投資をアプリケーションごとに複数回することなく一度の投資で将来のアプリケーションをサポートできることになります。

光ファイバ次世代アプリケーション

次世代のアプリケーションとは、**第1章の表2**で示した「イーサネット」の40/100Gb/s、「ファイバチャネル」の32Gb/sのスピードに代表されるアプリケーションです。

現在主流のマルチモード10Gb/sアプリケーション(ファイバチャネルアプリケーション16Gb/s)は、1つのファイバチャネル(2芯ファイバ)で10Gb/s(16Gb/s)のスピードで通信しています。つまり、10Gb/sのデータをペアの1芯で各々伝送し、2芯を使って双方向の通信をしています。では、アプリケーションが40/100Gb/sに進化した場合はどうなるのでしょう。

40Gb/sの場合は、4つのファイバチャネルを使い送受信をします。今までと同じく1チャネルあたり10Gb/sの送受信をします。それは、たとえば1台のサーバに2芯のファイバチャネルを4本作ることです。それでは、今までと何ら変わらないことと思われるでしょう。しかし、新しい通信規格ができた背景の1つに次のような考え方があります。

「サーバや通信機器からネットワークカードを減らし、省電力で環境に優しい経済的な環境を整えよう」

機器の背面または、正面のケーブル数が少なくなれば当然排気効率も良くなり、エアフローも改善されます。エアフローが改善されることにより光熱費、電気料金を抑えることができ、経済的な運用が可能になります。グリーンITを実現するうえで重要なポイントですね。

新しいファイバソリューションの規格は、12芯のケーブル1本にMTPと呼ばれる12芯のコネクタを両端に取り付け、1本のケーブル、1つのインターフェースで40Gb/sと100Gb/sをサポートするように作られました。では、これらのインターフェースの違いをどのように解決し、2芯から12芯に移行していくのか例を挙げて説明していきたいと思います。

移行のポイント

図1のように、12芯のケーブルの両端がMTPコネクタで成端されたケーブル①と、MTPコネクタを2芯のコネクタ(仮にLCコネクタ)に変換するMTP-LCカセット②を、①の両端に取り付けます。②のLCコネクタには2

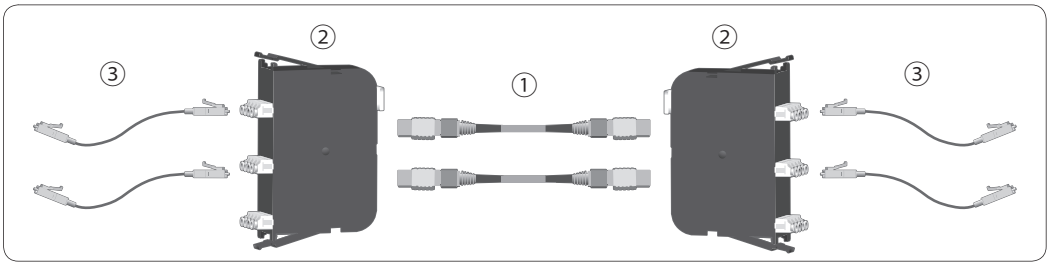
COLUMN

ケーブリング基礎用語 ②

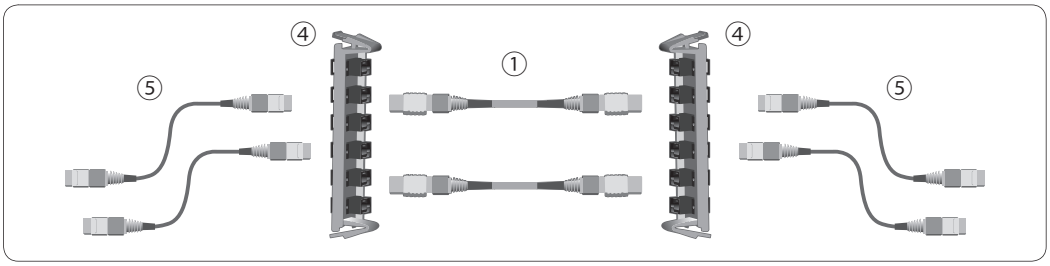
・ファイバチャネル
①コンピュータと周辺機器を結ぶための高速シリアルインタフェースの一種。おもにストレージネットワーク用に使用される。機器の接続には同軸ケーブル、光ファイバなどを用いる。本稿では、この意味で使う場合は「ファイバチャネルアプリケーション」と表記する。 ②ファイバ接続通信経路のこと。

・光ファイバ
ガラスやプラスチックの細い繊維でできた光を通す通信ケーブル。光の伝わるコアとその周囲のクラッドと呼ばれる部分の2層から成る。コアの直径によりマルチモード光ファイバ(MM)とシングルモード光ファイバ(SM)の2つの種類が存在する。一般的にシングルモードは高速で長距離伝送が可能だが高価であり、マルチモードは長距離伝送には不向きで、転送速度は落ちるもののコストパフォーマンスに優れている。光ファイバのケーブルは通常、送信用、受信用の2本1組で使われる

▼図1　10Gb/sの通信チャネルを確立



▼図2　40Gb/sと100Gb/sの通信チャネルを確立



芯のファイバ機器コード③を取り付けます。これで10Gb/sの通信チャネルが確立されます。**図1**の例では12芯ケーブルが2本で12チャネルが提供できます。

次に③を取り外し、そして②を取り外します。それぞれ12芯MTPファイバ機器コード⑤とMTP-MTPカプラ④に交換すれば**図2**のように40Gb/s(40GBASE-SR4)と100Gb/s(100GBASE-SR10)をサポートする通信チャネルが2チャネル(100Gb/sが1チャネル)[注1]のリンクに移行が完了します。

このように次世代アプリケーションをサポートすることで長期間運用できるシステムとなり、TCO(総所有コスト)やROI(投資利益率)を改善できます。

光ファイバプラグアンドプレーシステム

このように簡単に次世代のシステムに移行できる光ファイバソリューションを「光ファイバプラグアンドプレーシステム」、または「光ファイバアレーシステム」と呼んでいます。

このシステムは、次世代のシステムに移行できるだけではなく、機器のエアフローを改善する機能も持っています。たとえば、シーモン社製品では、従来よりケーブル径を30～40%細くし(**表1**)、狭い経路で曲げ半径が小さくなっ

注1) 40GBASE-SR4は8芯のファイバを使い各々4芯で送受信し、100GBASE-SR10は20芯のファイバを使い各々10芯で送受信を行います。IEEEでは、100GBASE-SR10のファイバを20芯の通信チャネルから8芯の通信チャネルへ移行する規格を策定中です。

ても通信性能に影響が出ない特別な技術を採用したケーブル「12芯MTPコネクタ付きRazor Coreケーブル」(**写真1**)を使い、通信室やデータセンタのラック内や室内のケーブル経路を縮小し、エアフローの改善に貢献します。このようなエリアからもグリーンITが実現できます。

長期間、安定した高性能を提供可能な理由

光ファイバプラグアンドプレーシステムが長期間、安定した高性能を提供できる理由は次のとおりです。

- 敷設サイトでの成端時および試験時にありがちな成端端面に異物が付着することを防ぎ、高性能を持続
- 性能試験では双方向で挿入損失と反射損失[注2]を2波長で試験し、そのデータをすべての製品に添付し、トレーサビリティーを確保
- ファイバ端面形状試験では、IEC61300-3-30 Ed 1.0に合格
- IEC、TIA/EIAの試験基準のEncircledFlux法を使用して嵌合(かんごう)性が高く挿入損失／反射損失の精度を高めている。その結果最大75%まで試験のバラツキを低減
- 独立した第三者機関Experior Labsで試験した結果、40Gb/sと100Gb/sの性能マージンがIEC1280-4-1とTIA/EIA-455-171Aを満たしていることを確認
- プラグアンドプレーモジュールおよびケーブルアセンブリは高性能なUSConnec社のMTPコネクタを採用
- LC/SCアダプタは、マルチモードとシングルモードともに高品質セラミック製アライメントスリーブを採用。これにより高い嵌合性能と耐摩耗性を向上

▼写真1　12芯MTPコネクタ付きRazorCoreケーブル



▼表1　RazorCoreケーブルのケーブル径

芯数	12	24	36	48	72	96	144
ケーブル径(mm)	3.0	3.8	8.7	8.7	9.4	13.2	13.2

ラックシステム

データセンタ、通信室、機械室でラックに最も求められている機能として、効率の良いエアフローがあります。**図3**のホットアイル／コールドアイル[注3]がその代表的なモデルです。

このモデルを有効に機能させる肝は、なんと言っても機器の吸気／排気能力をフル活用できるかにかかっています。効率よく冷気を吸い込み、機器内で熱せられた熱気を効率よくラック外へ排出すれば機器やケーブルが過度に熱せられて故障したり、障害が起きたりはしないはずです。

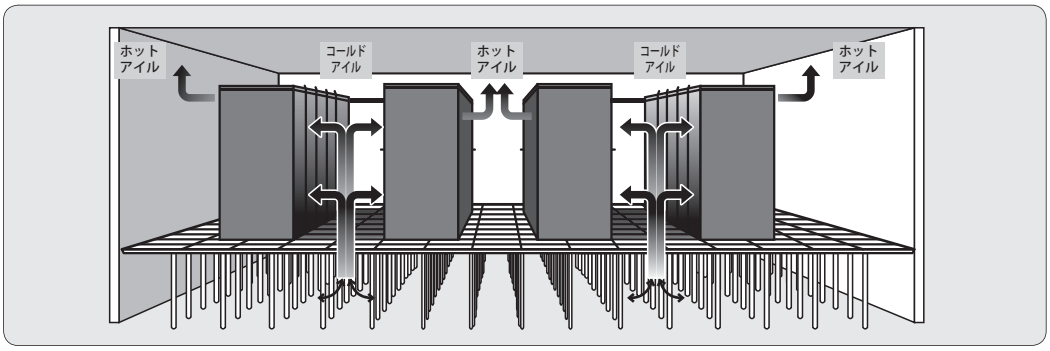
ここからは、ケーブリングシステム、ケーブルマネジメントを扱う立場からこれらの効率について改善策を提案していきたいと思います。みなさんが通信室やデータセンタに入ったときに**写真2**のような光景を目にしたことはありませんか。

ラック正面のコールドアイル側に面した機器は、ケーブルや配線支持設備などに妨害される

注2) 挿入損失(インサーションロス)とは光がファイバや接続点を通過するときの光の損失。反射損失(リターンロス)とは通信経路内のケーブルおよび嵌合部分での反射による損失。

注3) ラックを向かい合わせで設置することで、冷気を送り込む領域(コールドアイル)と機器の排熱が集まる領域(ホットアイル)を明示的に分ける方法。

▼図3　ホットアイル／コールドアイル



ことなく冷気を取り込みますが、背面のホットアイル側の機器の背面は、電源ケーブル、機器コードとそれらを支持するスイングアームで排気口を塞いでしまっています。これでは、熱気をラック外に排出できません。データセンタの熱対策は、まずこの点を改善することが大切です。

熱対策の代表的なソリューションとして次のようなものがありますが、高い効果を期待するならば、まずはラック内のエアフローを改善しましょう。

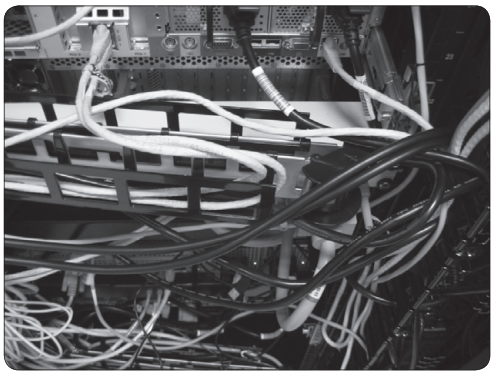
・高い発熱量のラックに冷却用のドアを設置
・ホットアイルまたはコールドアイルを囲うコンテインメントシステム
・ラックの天板に煙突やファンを取り付けたチムニーシステム

トップオブラックと呼ばれるデータセンタのイクイップメントディストリビューションエリア(サーバ機器を搭載したラックが集まるエリア)の通信方法がありますが、効果は期待できるでしょうか？

このシステムは、ラック間ケーブル本数を減らすことでエアフローを改善し、システムの導入期間を短縮できる利点があります。

しかし同時に、各ラックに通信機器(スイッチ)を搭載しなければならず、配線システムよりも消費電力が多く、ラックあたりの排出熱も高くなります。また、ラック内の上1/3に搭載されている機器は下2/3の機器よりも3倍故障率が

▼写真2　ケーブルマネジメントがなされていない状態



高いとUptime Instituteのレポートにあるように、ラック上部に置かれたスイッチのライフサイクルは短く常に交換機器の準備が必要になります。ラック単位で管理する点からスイッチの未使用ポートも多く存在し、多くの無駄な電力を使い、ラック間ケーブルの本数が少なくなっても肝心なラック内ケーブルの本数は変わらず、期待するエアフローの改善どころか、かえって熱に悩まされる結果になってしまいます。では、何が必要でしょうか？

ラック内のケーブルマネジメント

ラックのスペースを有効に使うことによってケーブルをマネジメントすることは、簡単にできます。1本のラックのマネジメントスペースはほんの僅かですが、2本のラック間のスペースを使えばケーブルをマネージするのに十分なスペースが確保できるではありませんか(**写真3**)。こうすることで、ラック間にパッチパネ

▼ 写真3　ラック間のスペースを使ってケーブルを収納





ルやファイバアダプタも取り付けられ、パネルからスイッチ間／サーバ間の距離も短くなり、ケーブルが機器のエアフローを妨げることがなくなります。そのうえ独立したリンクケーブル径路エリアを確保できます。

このように僅かなアイデアを実現した結果、機器周りの配線が乱れることなくケーブルマネジメントエリアに収まり、エアフローの改善はもちろん、データセンタの拡張／縮小に最も重要なMAC(移設／追加／変更)作業を計画のフェーズから円滑に進めることができます。

今までのデータセンタから聞こえてくる会話……

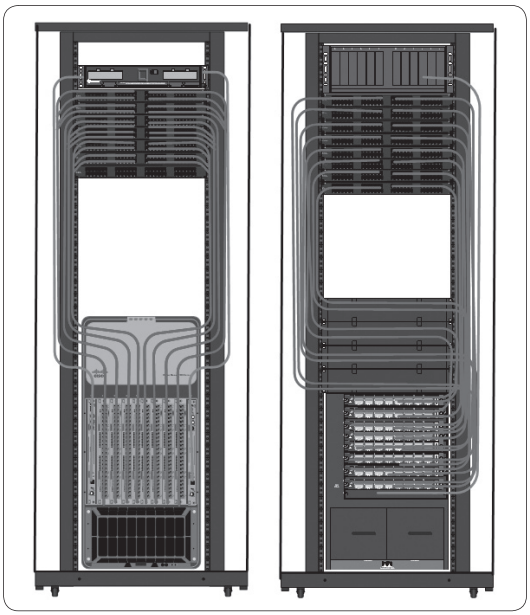
「このスイッチポートにつながっているTOKxxXXサーバはどのラック？」
「ラベル貼ってあるでしょ」
「見えない……」
「このケーブルは何かにつながってるの？」
「熱い……いや寒い……」

もうこんな捜索をする必要はなくなります。苦笑しているあなた、未来を現実にするチャンスです。

側面吸気／排気の機器

もう1つ悩まされてしまうのが、機器の側面で吸気／排気をするシステムです。大型のシャー

▼ 図4　側面で吸気／排気する機器の対応(机上での施工イメージ)



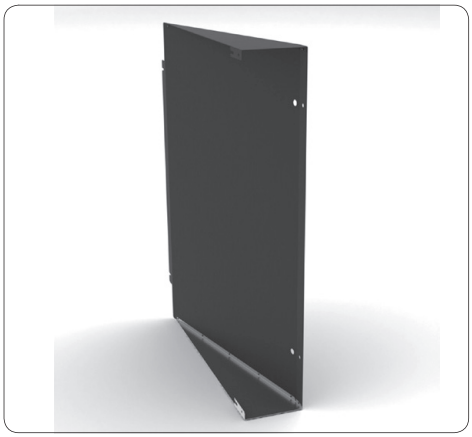
シ型スイッチによくこのタイプが見られます。どこから冷気を取り入れて、どの方向に排気を誘導するのか、頭を悩ませたエンジニアの方々が多数いらっしゃることでしょう。隣接のラックが熱くなってしまうことも経験されているのではないでしょうか。

優れたラックを使ってこんな感じにしようと絵に書きました(**図4**)が、現実はそんなに机上どおりに進まないものです。

▼ 写真4　側面で吸気／排気する機器の対応（シミュレーションと試行錯誤の結果）



▼ 写真5　バッフル



あるデータセンタでケーブルマネージャを駆使して1ラックに2台のスイッチを搭載したラックを2台連結しました。**写真4**は設計の段階で、ケーブルの本数と太さからケーブル束の太さを割り出しケーブルマネージャの収容数をシミュレーションし、多くの時間を割いて現場で試行錯誤の末に施工した結果です。

この環境では隣接ラックも熱くならず、もちろん隣接機器が熱気を吸い込んで能力が低下することもありませんでした。安全に運用できる満足な結果を得られました。しかし、ここまでたどり着くには、少し時間がかかり過ぎました。ご存じのようにプロジェクトは時間との戦いです、スケジュールどおりにシステムを引き渡さねばと内心すごく焦っていたのを思い出します。

しかし今では、設計／施工に時間をかけずに満足を得られるコンポネントがあります。**写真5、6**のようにサイドポッドとバッフルを組み合わせることで安定した側面吸気、側面排気を得られるようになりました。

写真6のとおり簡単な構造で吸気と排気をコントロールします。縦型のパッチパネルやケーブルマネージャを組み合わせると管理しやすい環境が整います。

▼ 写真6　サイドポッドとバッフルを組み合わせる



このように革新の技術で高性能に特化したケーブリングシステムや、高い機能を提供するケーブリングマネジメントシステム、ラックシステムを組み合わせることにより現在の高速なアプリケーション、また将来に起こり得る最新のアプリケーションのサポートが可能となります。

拡張／縮小を繰り返すITシステムの環境の中で、アプリケーションをサポートするためにアプリケーションごとにケーブルやラックを入れ替えますか？　それとも機器を乗せ換えるだけで最新のアプリケーションがサポートでき、長期間運用できるグリーンな環境を選択しますか？ SD

第3章 高速アプリケーションに最適化した 新型RJ-45開発記

これまで長年の間、LANケーブルなどのコネクタとして、RJ-45コネクタが当たり前のように使われてきました。しかし、10Gb/s高速ネットワークが実現してきたこの段階に来て、RJ-45を見直し、高速アプリケーションに最適化したコネクタが登場してきました。

入澤 徹(いりさわ とおる)　ザ・シーモン・カンパニー

RJ-45を再開発する

Ethernetポートを連想すると、まずRJ型モジュラープラグやジャックコネクタが頭に浮かんでくるでしょう。通信のための構造配線システムの概念が確立されてからずっとこのインターフェースは通信業界で著しく際立った存在です。

1960年代以後、2ペアバージョンのコネクタが電話機器に取り付けられました。それから現在に至るまで何の疑問を抱くことなく、RJ型モジュラープラグやジャックコネクタは存在してきました。しかし、データ転送速度が著しく進化を遂げ10Gb/sを超えようとしている現在、これらのコンポネントは将来のアプリケーションをサポートできるのでしょうか？

これらの問題を打開するためには、革新的なモジュラージャックとプラグのデザインが必要です。そこで、高速通信の本来あるべき性能に最適化したRJ-45の再開発が必要となりました。RJ型モジュラープラグとアウトレットの内部機能を完全に再設計し、全く新しい技術で、プラグとアウトレット接続でデータを劣化させないコネクタ開発が早急に求められました。

高い伝送マージンと信頼性を実現するために、モジュラープラグにプリント基板を採用する必要もありました。これらの技術はZ-MAXシリーズのカテゴリ6Aの接続ハードウェアとパッチングソリューションに活かされています。 現在のRJ型モジュラーコネクタは、カテゴリ5e全盛期には予測できなかった高性能とヘッドルーム(期待値とのマージン)を提供しています。その一方で、5年前には考えられない環境での使用に耐え得ることを要求されています。このインターフェースは500MHz間での内部的(ペア間)クロストークと外部的(エイリアン)クロストークを排除し、PoE+規格で各ペアに最大600mAまでの電力供給をサポートしています。また、幅広い環境条件でも高い信頼性を持つ頑丈な接続を要求され、環境に配慮した部材で作られることが期待されています。

COLUMN
ケーブリング基礎用語 ③

・コネクタ
装置間をケーブルでつなぐ際の接点となる部品。突起状になっている側をオス、受け口状になっている側をメスと呼ぶ。電気信号を伝えるコンタクトピンという金属の端子と内外を取り囲んで電気を絶縁するインシュレータと呼ぶ部品から構成される

・モジュラージャック
電話線やLANケーブルなどを接続するためのコネクタ(メス側)。一般に電話線用のモジュラージャックを「RJ-11」、LAN用を「RJ-45」という

・モジュラープラグ
電話線やLANケーブルを終端するコネクタ(オス側)

・RJ-45
EthernetやISDNのケーブルを接続するために用いられる8極8芯の通信用コネクタ。10BASE-Tや100BASE-TXなどのコネクタとして多く利用されている

・UTP(Unshielded Twisted Pair)
撚り対線がシールドで覆われていないケーブル。一般家庭やオフィスで使われる

・STP(Shielded Twisted Pair)
外部の電気的ノイズから守るためシールドと呼ばれるアルミ箔などで撚り対線が覆われたケーブル。ノイズの多い工場などで使われる

今までの製品に勝る次世代のカテゴリ6Aコネクタを設計するという要求に直面したエンジニアは、RJ型モジュラープラグやジャックを徹底的に分解し見直すこととなりました。シーモン社では次の重要なエリアに設計の改善ポイントを置きました。

＜電気的な強化ポイント＞

・現場や工場成端プロセスに関連した高い可変性を排除する
・モジュラープラグ性能を継続的にモジュラーアウトレットの「スイートスポット」の範囲に入れ、最適な結合性能を提供する
・ケーブル成端におけるペア割れとペアのクロスオーバー[注1]を取り除く
・入力と出力導線接続間の各ポイントにおいて、伝送モデリング技術の有限要素解析をし、大きさと位相を最適化する
・パッチパネルやフェースプレート[注2]の隣接コネクタ間を電気的に隔離し、エイリアンクロストークの影響を最小限に抑え、同時に偶発的なチャンネル間シールド接触を回避する

＜メカニカルな強化ポイント＞

・すべてのコンポネントにおいて伝達長さ[注3]や途切れを最小限にするために、接続と接触構造の配置を最適化する。隣接アウトレット間の導線接続距離を最長化し、対エイリアンクロストーク性能を改善する
・環境に配慮していない原料（ベリリウムや鉛など）を製品から取り除く
・導線を挿入する力を低減し、プラグ／ジャックの垂直抗力のバラつきを最小限にする
・電力供給やデータ伝送をする場合、接続／切断時に接続コンタクトピンの配置によって起きるアークゾーン[注4]に注目し、信頼性を高めるためにプラグとアウトレットの接続インターフェースからアークゾーンを取り除く
・モジュラーアウトレットの要求すべてに対応できる特徴を兼ね備えること。たとえば、パッチエリアで全9色のカラーバリエーションがそろっていること。フラット／アングル両用のハイブリット型とキーストーンアウトレットが利用可能なこと

電子レベルの強化

RJ-45モジュラープラグの性能バラつきを取り除く

ツイストペアケーブルは緻密なペア構成によって適切な伝送性能を確保しています。しかし、ケーブルとモジュラープラグ成端の際に、ペア構造を崩すことで相当な性能のバラつきをもたらします。さらに、データ伝送用4ペアT568AとT568B結線[注5]では、ピン3と6の結線はピン4と5の結線をまたぐように結線されており、導線の撚りを戻され分離されています（**図1**）。3番6番ピンに結線される緑色とオレンジ色のペアは、RJ-45プラグに結線されるときには、スプリットペアになります。このスプリットペアがRJ-45プラグ内のクロストークやカプリング[注6]の大きな影響源になっています。

そのうえ、ケーブルのどちらの末端を成端するかによって、ケーブルの配線方向は右手向きと左手向きが存在します（**図2**）。この事実はよく見落とされますが、この「ミラーイメージ」配線方向の影響により任意のペアはクロスオーバー

注1）「ペア割れ」とは、図1のT568AのPair2、T568BのPair3のように隣接したピンに結線せずにほかのペアをまたいで結線すること。これは規格で決まった結線方法で、過去のアプリケーションの利便性向上のためにペアを割っているのだが、ペアを割ってしまうことで性能が低下し、現代のアプリケーションの要求を満たすのが困難になっている。「クロスオーバー」とは、図2のようにケーブルの両端でペアの位置が違っているように、必ずどちらかの端のペア同士をクロスさせてポジションを入れ替えないとコネクタの結線ポジションにペアが収まらない。これも規格とコネクタの構造上の問題で性能が低下する一因となっている。
注2）壁から直接ケーブルを出すときに安全性や意匠性を高めるために取り付ける化粧板。
注3）伝達にかかる時間的な長さ。
注4）コネクタを接続する際、または切断する際にアーク放電が発生するエリア。
注5）米国規格協会（ANSI）で定めたLANケーブル終端における結線規格。正式名称はTIA/EIA-568-AとTIA/EIA-568-B。
注6）クロストークと同義。雑音が乗るという広義で使われる。

接続された状況でケーブル両端のピンやペア接続の平衡を実現しなければなりません。そのため、同じプラグ設計を利用するコードの両端で性能の整合性をとることは事実上不可能です。

オペレータがトレーニングを積み、正確にモジュラーコードを成端できるようになるまでは、多くの時間を必要とします。パッチコードは必ず100%伝送テストを行い、正確な構成でかつ基準を超える性能であることを保証しなければなりません。

プラグ成端の品質と整合性における伝送性能の信頼性を確認した結果、総合ケーブリングメーカーはカテゴリ5e以上の保証範囲からフィールド製造パッチコードを除外しました。製造やテストに厳しいプロセス制御がなければ、性能を保証することができないと判断した結果です。

モジュラープラグとそのケーブル成端は結合性が最も重要であり、Z-MAXの設計で最初に注目すべきポイントでした。シーモン社では次の点をもとにZ-MAXプラグを作り上げました。

- 既存のTIA/IEC仕様のアウトレットとメーカーにかかわらず、下位互換性またはメーカー間の相互互換性を持つこと
- スプリットペアと撚り戻しに関連する性能のバラつきを取り除く
- ケーブルの各末端の異なるカラーシーケンスが引き起こす性能のバラつきを取り除く
- すべての業界標準に一致し、TIAとISO/IECの規格性能範囲内に最大限の整合性をとり、結合性能を提供しながら、アウトレットとカテゴリに完全な互換性を持たせる

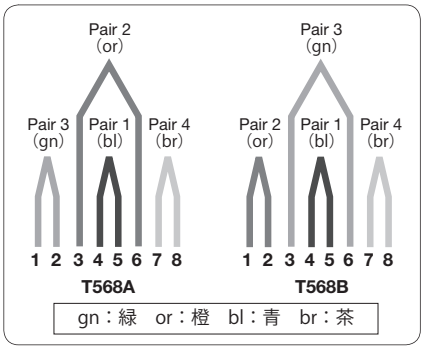
これらの目標を実現するため、シーモン社は特許取得済み、または特許出願中のプリントボード技術を採用し、Z-MAXモジュラープラグ本体に設置しました(**図3**)。

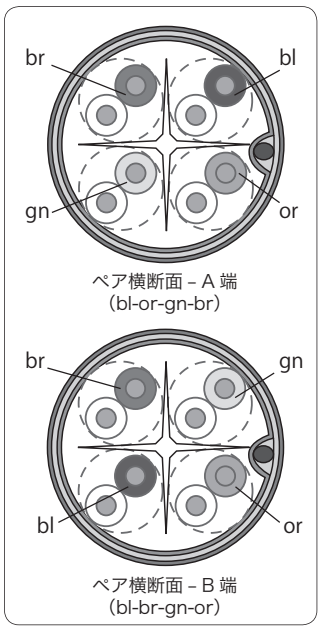
伝統的なプラグ成端プロセスに影響される性能のバラつきを取り除くため、プリントボード(PCB)はケーブルの各末端導線のペアカラーシーケンスを考慮し取り付けるデザインになっています。これにより、各ペアが位相のそろったユニットとしてペア割れやペア間のクロストークを取り除き、成端ポイントに至るまで性能を最適化しペアの整合性を保持します。各コードはユニークなプラグで成端され、各プラグは独立した回路設計と接続のしくみを持ち、安定性のある機能を提供します。TIAとIEC規格に一致する相互互換性のあるモジュラープラグとなっています。

スイートスポットを見極める

TIAとIECは、一貫した性能規格を保持す

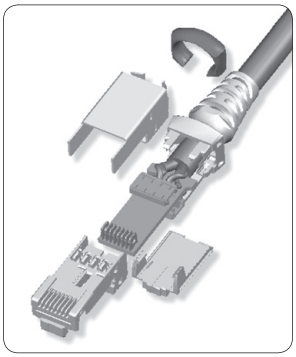
▼図1　T568AとT568Bの結線





◀図2　成端の違いで生じるケーブル配線方向の向きの違い

▼図3　Z-MAXモジュラープラグPCBベース設計



るためにNEXT loss(近端漏話)、FEXT loss(遠端漏話)、Return loss(リターンロス(反射損失))値の範囲を規定しました。この範囲が指定されたことによりカテゴリの下位互換性とメーカー間の相互互換性が保たれることになります。一方でアウトレットは、プラグ性能の範囲に対応できるように補償回路を使い設計されています。これらの互換性と補償回路が平均的な性能を生み出し、高いNEXTヘッドルームを持つプラグを開発できない原因になっていました。

シーモン社はプリント基板をプラグ内に装着させることにより、Z-MAXプラグの高い性能を最適化しました。Z-MAXプラグとと Z-MAXアウトレットで構成されるチャネルリンクは、その製品のライフサイクルに渡って高性能を保証し確実に機能します。

この革新的な「circuit-in-plug」コード(基板の入ったRJ-45プラグ付きパッチコード)の高性能のもう1つ根拠として注目すべき点は、フィールドテスターのパーマネントリンクアダプタコードがこの技術を採用していることです。さらにTIAとIEC接続ハードウェアテスト方法もテストプラグ構成はプリント基板の使用を規定し、研究室における測定反復性や再現性を確保しています。

アウトレット成端のバラつきを削減

モジュラーアウトレットはInsulation Displacement Contacts(IDCs)[注7]を利用してツイストペアに接続します。IDCsは丈夫で、気密性の高い接続で再成端ができます。これらのケーブル成端ポイントもペアのクロスやスプリットペアを起因とする性能のバラつきの原因になり、制御不可能なクロストークを引き起こします。これらの成端プロセスもアウトレット性能に悪影響を与えます。

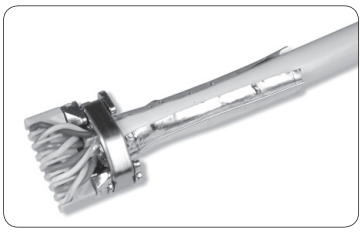
シーモン社ではZ-MAX成端フィールドを設計する際に、ケーブル両端相互のペアクロスする必要性を取り除き、ツイストペアの成端整合性を最大にすることに成功しました。Z-MAX Zero-Cross成端モジュールは各ペアを独立配置させ、ケーブルペア構成の問題を最低限にしました(**写真1**)。IDCコネクタは隣接アウトレット間でペアの離隔を最大化し、クロストークを削減する方向に正しく配列されています(**写真2**)。

補償回路

補償回路の役割は、結合プラグ―ジャック間インターフェースとケーブル接続間のクロストークを補正することです。シーモン社のエンジニアは転送中断をもたらす静電容量や、インダクタンスがコネクタに引き起こすクロストークのモデルを明確にしました。論理的に同等な振幅の静電容量やインダクタンスはお互いにクロストークを打ち消し合うことができます。このテクニックは「リアクタンスバランス」と呼ばれますが、実際にそれを機能させるのは次の理由により非常に困難です。

・クロストークレベルはプラグとIDC成端の

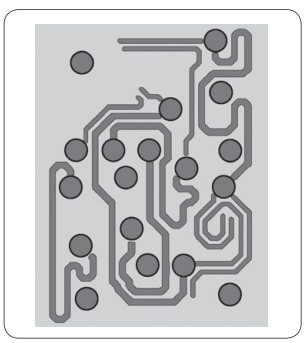
▼写真1　Z-MAX Zero-Cross成端モジュール



▼写真2　Z-MAX IDC方向



▼図4　Z-MAXアウトレットPCB



注7) ケーブルの絶縁被覆を剥かずに、絶縁にコンタクトを刺して接続する方法。圧接ともいう。

バラつきによって、著しく変動する
・補正エレメントは非線形周波数レスポンスを示すため、補償回路に100MHzでクロストークを減らし、500MHzでクロストークを増やさないことを確保する特別な対処が要求される
・4ペアコネクタが持つ多面性は、1つの点をほかの性能を低下させずに改良することが非常に困難

Z-MAXモジュラープラグとアウトレットIDC構成は、校正補償設計によって成端のバラつきを最適化します。安定したリピータブルな(同じ性能を繰り返せる)プラグやIDC性能によって、信頼性の高い補償回路を基礎に高性能な伝送モデルを開発しました。また、特許や特許出願中の技術を応用して、これらの要素をZ-MAXアウトレットに組み込みました。

Z-MAXアウトレットの高度な回路(**図4**)は、電気的な特徴をいかし、多種多様なテクニックを採用した結果できあがったものです。補償回路における振幅や位相に関する特許には長さ、幅、厚さなどPCBパラメータの精密制御が含まれます。これらのパラメータは最適化され、Z-MAXアウトレット全体に最小反射と平衡であることを保証します。これらの技術はシーモンZ-MAXアウトレットの補償回路に組み込まれており、Z-MAXシステムのマージンをサポートするほか、一般的なカテゴリ6Aより高い性能マージンをサポートします。

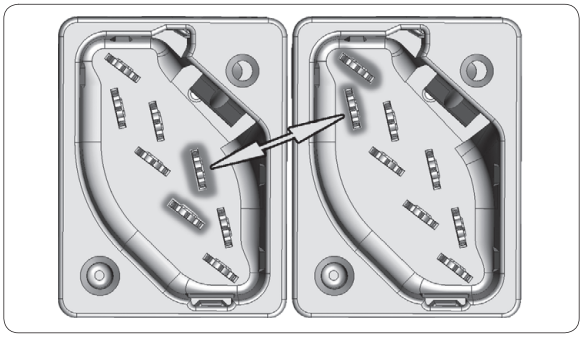
隔離

シールドは外部クロストークを含むあらゆるタイプの電磁妨害に対して効率的な遮蔽性を提供します。一方、UTPケーブルとコンポネントは、ペア線のバランスをより良くする(ペア線のツイストをよりきつく撚る、導線を太くする)ことでペア間を分離しクロストークを遠ざける効果が得られます。実際にケーブル径を太くしてケーブル内外でペア間の離隔をとることでクロストーク対策を達成します。Z-MAXアウトレットは、IDCsの対角線方向にある隣接Z-MAXアウトレットの背面のペア間でその最短距離で最適な距離空間を提供します(**図5**)。**図6**の従来製品と比べれば明確にその違いがわかります。

この設計の成果としてZ-MAXはUTPとシールドの両方で、1Uに24や48ポートのパッチパネル密度をサポートするという極めて高密度のポート配置を実現しました(**図7**)。

ケーブリングシステムを設計する際には、高密度を重点的に考慮すると同時に、隣接アウトレット間でシールド接触の危険を引き起こさないことを保証しなければなりま

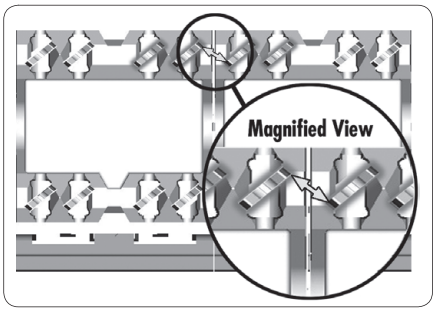
▼図5　Z-MAX対角線IDC方向による最大化したアウトレット間におけるペア分離



▼図7　Z-MAXの高密度のポート配置



▼図6　伝統的なIDC方向付きアウトレット間におけるペア分離



せん[注8]。この要求を満足するために、Z-MAXではカーネルとベゼル構造に非対称構造が用いられ、取り付けるときに横並びのシールドアウトレットがお互いに接触しないようになっています。

伝送ヘッドルーム(マージン)

Z-MAXモジュラープラグやアウトレットを構成する各々の要素は、安定かつ一貫したプラグ機能(リピータブルで低クロストークIDC成端と最新技術の補償回路など)を持っています。チャネルとパーマネントリンク構成において、Z-MAXソリューションは前例のないマージンを提供し、カテゴリ6A伝送要件を満足させています(**表1**)。これらの性能レベルは、簡単な微調整やカテゴリ6を調節するような製品単体の調整では実現できないものです。モジュラープラグとジャックシステムを含む各コンポネントを、完全かつ正確に見直した結果、性能向上につながりました。このシステムはリンクやチャネルの長さが15m未満でもカテゴリ6Aとクラス EA性能の要件を満足できます。

メカニカルな強化

ジャックピンコンタクトピン

モジュラージャックのコンタクトピンは電気的にプラグジャック結合し、接続の信頼性においても重要な役割を果たしています。過去には、モジュラージャックの製造は性能よりむしろコストを重点的に考慮してきました。

一般的にジャックコンタクトピンはPCBのジャック方向に適応するように設計され、スペック以外(4ポジション、6ポジションコンダクタピンのRJ-11)のプラグが挿入されたりすると、変形してしまいました。通常はベリリウム銅で構成され、それ自体は良好な機械属性を持ちますが、環境に配慮した材料ではありません。また、伝統的なモジュラージャックは簡単に製造できると同時に機能的な欠陥も潜在しています。たとえば、コネクタポジションの微小な動きには信頼性が薄く、挿入角度や電気的性能が影響を受けてしまいます。

Z-MAXジャックコンタクトピンの短いピン設計は、機能的な利点に加え、環境に配慮した材料と製造工程で安定した再現性を実現しています。スペック外のプラグ挿入などでもコンダクタが破損することなく、最適で有効な垂直方向挿入ができる材料を利用しています。銅合金はスプリングコネクタの稼動範囲を定義する弾力性の係数に対して、降伏応力の最適な割合を示し、ジャックコネクタ強度や耐久力において最適なバランスを確保しました。

Z-MAXの接点は特別に環境に配慮した高度なプレイティングプロセス[注9]で設計されています。通常のアウトレットはピンを接続する場合に、過去からの習慣で基板にハンダ付けしていますが、Z-MAXはPCBにピン接続技術(Press-fit、Eye-of-Needle)を施したことにより、ハンダ付けせずに基板との接続性を高めています。弾力性のあるピンの素材とそれを生かしたピンの形状で、基板にあるピン接続用の穴にピンを通して基板と接続しています。これにより、性能を安定させ、ハンダやベリリウム銅、

▼表1　Z-MAX保証チャンネルヘッドルーム(1-500MHz)

	Z-MAX(UTP)	Z-MAX(F/UTP)
挿入ロース	3%	0%
Nextロース	3dB	3dB
PSNEXTロース	3.5dB	3.5dB
ACR-F	7dB	7dB
PSACR-F	10dB	10dB
リターンロース	3dB	3dB
PSANEXTロース	1dB	10dB
PSAACR-F	1dB	5dB
ACR-N	6dB	6dB
PSACR-N	6.5dB	6.5dB

注8) これはシールドシステムの特別な点で、故意に接地していないアウトレット間で直接カプリングが発生しないことを完全に確保するためです。
注9) 結合コネクタ接触ゾーンの重要なエリアにおけるコネクト性能を完全に初期制御する。

鉛など危険な材料をアウトレット組み立てから完全に取り除くことができました。

アウトレットコンタクトピンをより簡単に偏差範囲に適応できる合金を利用することによって、結合モジュラープラグの高さ制限の全範囲をカバーでき、6ポジション／8ポジションコンダクタピンモジュラープラグに相互互換できるコネクタを設計しました(**図8**)。Z-MAXコンタクトピンはほかの製品に比べ、ピンの長さが最も短く、最も要求が高いライフサイクル要件を満たしています。Z-MAXモジュラージャックのコンタクトピンは、変形や歪みに強い弾性のある素材を採用することによりコンダクタピンの規定位置を保持します。この構造により完全にプラグと結合し隣接するコンタクトピン間でクロストークやカプリングを最小限にします。信頼性と恒久性の高いコンタクトピンがアウトレットに採用された結果、少なくとも2,500回の抜き差しに耐え得る構造を確保しました。

コンタクトワイプ

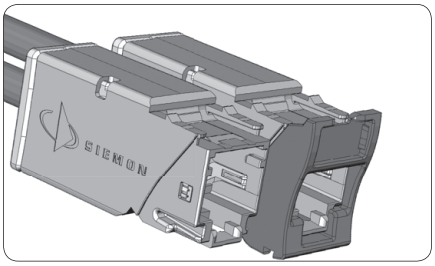
どんなに良い環境においても、プラグとジャックコネクタは外的環境からの液体や破片などで汚染される恐れがあります。Z-MAXモジュラープラグとアウトレットを結合するとき、各コンポネントのコネクタは最終的に接触するコネクションポイント以外のエリアから互いに接触し、付着物を拭い取ります。この拭い取り作用により、きれいで汚れのない接続が確保されます。

Z-MAXコンタクト設計では、回路積載電力[注10]がモジュラーアウトレットに接続／切断される場合に起こる、アーク放電による潜在的なコネクタ損害を回避するよう考慮されています。このアーク放電もコンタクトゾーンの外部の定められたエリアで処理できるように設計されています。

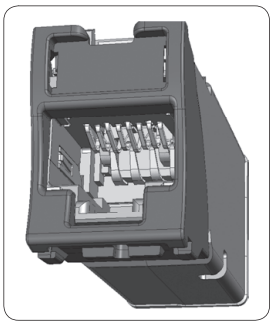
ハイブリット型モジュラー

高性能なだけでなく、ユーザの幅広いニーズに応えるため、Z-MAXフル機能のアウトレット「コア」と、「カーネル」というサブアセンブリ(製品構成部品)を開発しました。コアとカーネルとは設計思想上の言葉で、**図9**の部品の1つ1つがカーネルです。カーネルを組み立てることで幾種類かの製品(コア)ができあがります(**図10**)。具体的には、アウトレット本体(カーネル)に、パッチパネル／フェースプレート接続用のモジュラーベゼル(これもカーネルの1つ)を各々部品として開発しました。3種類×10色あるモジュラーベゼルの組み合せを変更するだけで、

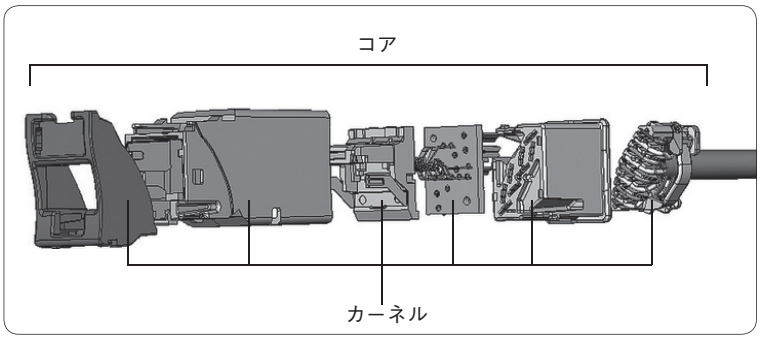
▼図10　カーネルを組み立てることで1つの製品(コア)ができる



▼図8　Z-MAXジャックコネクタピン



▼図9　Z-MAXのコアとカーネル



注10) たとえばPoEやPoE+アプリケーションなど。

アウトレット完成品(コア)を迅速に組み立てられるようになりました。たとえば、ハイブリット(フラット／アングル共用)、キーストーンアウトレットのカラーコーディングで適切なベゼルを装着し迅速に出荷できます。

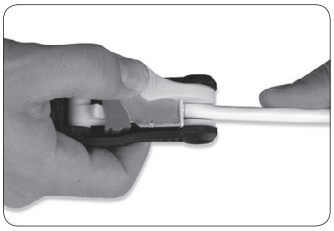
▼写真3　Z-MAXハイブリットフラット／アングルタイプ



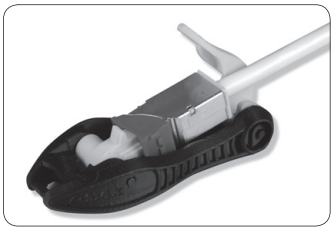
▼写真4　平行に各ペアを色分けし独立ペア割り付け



▼写真5　ワンステップでシールド成端が終了



▼写真6　Z-MAXアウトレットとZ-TOOL成端工具



UTP Z-MAXカーネルサブアセンブリがモールドハウジング、シールドZ-MAXカーネルサブアセンブリはダイキャスト金属ハウジングを特徴としていますが、この方法の採用で全製品の製造ラインがUTP／シールドの区別なく持続的に機能するようになりました。製造プロセスの変動を最低限にし、安定した性能の製品を効率よく出荷することに成功しました。また、製造時に出る無駄となる原材料を削減／リサイクルでき、グリーン環境に貢献しています。

ベゼルはZ-MAX設計の革新的な特徴で、ハイブリットバージョンは標準パネルへの取り付けができるのはもちろんのこと、フェースプレートではフラットとアングルに取り付けができます(**写真3**)。カラーコーディングはベゼルカラーオプションによって、シールド／UTPアウトレットで同色利用ができ、付属アイコンでさらに識別が可能になりました。このコンパクトカーネルとベゼル設計はアウトレットを前面または背面から取り付けることを可能にしました。パネルへ取り付けるためにラッチが付いていて軽く押すだけでロックを解除することができます。アイコンは見やすく再生利用可能な素材を採用しています。

速く／簡単に成端

現場でより速く成端できることは、コスト削減や迅速なトラブルシューティングにつながり、ネットワークインフラの構築に大きな利点をもたらします。多くの製品を導入したユーザからのフィードバックに基づき、その需要を取り込んでZ-MAXアウトレットはカテゴリ6Aで最短の成端時間を実現しました。

すべてのペア対称性を維持し、スプリットやクロスペアを取り除くテクニックを採用したことで、Z-MAXは本質的に短い時間で成端できるようになっていますが、それに加えて、シーモン社はそのZ-MAXアウトレットの成端時間を1分以内に短縮することを目標にしました。

写真4のようにケーブルの両端ともに平行に各ペアを色分けし独立ペア割り付けすることにより、ペアクロスを排除しました。**写真5**のようにワンステップでシールド成端、ペア線挿入、圧接が行われ、成端が終了します。またコネクタボディ自体がストレインリリーフの役割を果たしケーブルの曲げすぎやケーブルがコネクタから抜けてしまうことを防止します。今まで2回、3回かかっていた手作業がすべてワンステップで完了します。新しいZ-TOOL成端工具(**写真6**)を使い最適なワンステップ成端フローを実現しました。結果として、Z-MAXの成端時間はケーブル成端準備時間を含め、UTPアウトレットで55秒まで短縮、シールドアウトレットの場合は60秒に短縮しました。SD

バックナンバーのお知らせ　BACKNUMBER

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

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



2013年12月号

第1特集
SDN/OpenFlowの流れを総まとめ!
SDN/OpenFlowで幸せになれますか?

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

一般記事
・LinuxとFreeBSDのファイルシステムの良い・悪いところをご存じですか?
・「Mirama」a.k.a. VIKING ほか

1,280円



2013年11月号

第1特集
思考をコード化する道具
我が友 Emacs

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

一般記事
・小規模プロジェクト現場から学ぶJenkins活用（5）
・ソーシャルゲームのDevOpsを支える技術（前編）
・Ubuntu 13.10 "Saucy Salamander"

1,280円



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

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。



より速く、より莫大に、より高みへ！

サーバマシンの測り方

ベンチマークを極める実践テクニック

第2回　データベース性能を測る

第1回では、ディスクI/O性能のベンチマークについて取り上げました。ディスクI/O性能が必要な用途としてデータベースサーバが挙げられます。そこで今回と次回にわたりMySQLでデータベースを構築し、データベースのベンチマークをすることでサーバ性能を測定します。

Writer ㈱IDCフロンティア　ソリューションアーキテクト　藤城 拓哉（ふじしろ たくや）／Twitter@tafujish

定番ツールでDBサーバを測る

データベースの性能は、サーバーの次の要素の影響を大きく受けます。

- **CPU性能（動作周波数やコア数）**
- **メモリ容量**
- **ディスクI/O性能**

このようにサーバで必要とされる要因が複数関係し合い、かつデータベースはサーバでよく使われる利用用途でもあるので、データベースのベンチマークは、サーバの全体的な性能を測るのに適しています。連載第2回は、定番のRDBMSであるMySQLを例に取り上げ、ベンチマークの手順を紹介します。

MySQL Serverをインストール

MySQLをインストールするにあたり、OS標準のパッケージを利用したり、特定のバージョンとビルドオプションを用いて自分でビルドしたりするケースがあると思います。普段から決まった方法があれば、そのとおりに構築すればいいです。OS標準のパッケージでは、バージョンが古いこともあるので、ここではMySQL Community Server の Generally Available Releases[注1]を用いて、執筆時点の最新バージョンである5.6.14をRPMから入れました。

コンフィグファイル（/usr/my.cnf）は、数GBのメモリを搭載したサーバを想定し汎用的な設定としました。ベンチマークの目的に合わせて設定のチューニングや、普段から決まった設定があればそちらを利用すると良いでしょう。もし決まったコンフィグファイルがなければ、とくに次の値に注意してください。

`max_connections`で、データベースへの最大接続数を指定します。ベンチマークで接続する数が多い場合や実際に実行してコネクション数不足になった場合は、この値を大きくしてください。併せて、生成するスレッドをキャッシュさせるために次の設定値、`thread_cache_size`と`max_connections`と同じ値にしておくとよいでしょう。

`innodb_buffer_pool_size`でバッファプールを割り当てます。つまり、メモリにどれだけ割り当てるかです。メモリへの割り当てが少ないとその分ディスクI/Oが増えることにつながります。

これらのコンフィグの設定値による実際の動きは次号で紹介します。この後は本稿での具体的なツールを紹介します。

注1）（http://dev.mysql.com/downloads/mysql/）

mysqlslapを使う

mysqlslapはクライアントからの負荷をエミュレートするMySQL標準のツールです。そのため個別でのインストールが不要で、手軽に利用できます。クエリを指定して負荷をかけることができるので柔軟性が高い一方で、テーブルやクエリを自動生成して負荷をかけることができるので手軽です。

MySQLのクライアントが入っていればインストールは不要です。実行オプションについては**mysqlslap --help**や**man mysqlslap**で確認します。

自動生成を利用しベンチマークを実行した例が**図1**です。結果は、各クライアントにおいてすべてのクエリを実行するのにかかった時間(秒)の平均値、最小値、最大値が得られます。時間が短いほうが速いということになります。

一般的に使うオプションとしては、**--concurrency**は同時に実行するクライアント(スレッド)の数、**--iterations**は繰り返し実行する回数です。また、DBのユーザがrootでない場合は**--user**でユーザ名を指定、DBサーバがローカルホストでない場合は**--host**でDBサーバのIPアドレスやホスト名を指定します。

自動生成のオプションとしては、まず**--auto-generate-sql**を指定する必要があります。また、**--auto-generate-sql-load-type**にてread(テーブルのスキャン)、write(テーブルへの挿入)、mixed(挿入とスキャン半々)、ほかにもkey(主キーの読み取り)、update(主キーの更新)が指定できます。**--auto-generate-sql-add-autoincrement**を指定し、AUTO_INCREMENTカラムを追加します。**--auto-generate-sql-write-number**は、各クライアントが挿入するクエリの数でデータベースのサイズを大きくしたい場合はこの値を大きくします。**--auto-generate-sql-unique-query-number**は、それぞれ異なるクエリを生成する数です。**--auto-generate-sql-execute-number**は、各クライアントが実行するクエリの数です。

手軽に試したいときや、ベンチマークツールをインストールできない環境の場合はmysqlslapを試してみてはいかがでしょうか。

SysBenchを使う

SysBench[注2]はCPUやメモリなど多くの種類のベンチマークが可能なツールです。mysqlslapかSysBenchのどちらかを使えるとよいです。mysqlslapは手軽に利用でき、SysBenchは計測時間やリクエストの数を容易に操作できるので便利です。

SysBenchのインストールは**図2**のとおりです。mysqlslapと違い、ベンチマークを実行する前にテスト用のデータベースを作成しデータを生成する必要があります。**--oltp-table-size**を5000000としたとき、1.2GBほどのデータが生成されます。

実行例は**図3**です。結果として、何回クエリが実行したかや実行にかかった時間などが得ら

注2) http://sysbench.sourceforge.net

▼図1　mysqlslapの実行例

```
$ mysqlslap --concurrency=1000 --iterations=1 --auto-generate-sql --auto-generate-sql-add-
autoincrement --auto-generate-sql-load-type=mixed --auto-generate-sql-write-number=100000
--auto-generate-sql-unique-query-number=100000 --auto-generate-sql-execute-number=100
Benchmark
        Average number of seconds to run all queries: 10.628 seconds ← 結果
        Minimum number of seconds to run all queries: 9.843 seconds
        Maximum number of seconds to run all queries: 11.516 seconds
        Number of clients running queries: 1000
        Average number of queries per client: 100
```

れますが、秒間あたりのtransactionsの結果がわかりやすいと思います。この例では、3618.00 TPSとなり値が大きいほど高速です。

一般的に使うオプションとしては、**--test=oltp**を指定しOLTP(データベース)、**--num-threads**で同時に実行するクライアント(スレッド)の数、**--db-driver**でMySQLを指定し、**--mysql-user**でDBのユーザも指定する必要があります。DBサーバがローカルホストでない場合は**--mysql-host**でDBサーバのIPアドレスやホスト名を指定します。

テスト内容に関するオプションとして、**--oltp-table-size**は**図2**で生成したサイズを指定し、**--max-requests**でリクエスト数(トランザクション数)を指定します。今回の例では、**--max-requests**は0ですが、この場合はリクエストを投げ続けます、そして**--max-time**で指定した時間(秒)がきたら計測を終了します。つまり今回の例では600秒間計測する場合の例となります。

sysbench --test=oltp helpでほかのオプションが調べられます。SysBenchでCPUやメモリのベンチマークと併せて、データベースのベンチマークも一緒に実行されてはいかがでしょうか。

tpcc-mysqlを使う

mysqlslapやSysBenchよりも実際の利用を想定したデータベース操作や複雑なテーブル構成へのベンチマークを本格的に実施する場合、tpcc-mysqlが使えます。

tpcc-mysqlのインストールは**図4**のとおりです。SysBench同様、ベンチマークを実行する前にテスト用のデータベースを作成しデータを生成する必要があります。**tpcc_load**にて生成しますが、ここでwarehouse(倉庫)を指定し

▼図2　SysBenchのインストール

```
$ sudo yum -y install libtool make gcc
$ curl -L http://downloads.sourceforge.net/project/sysbench/sysbench/0.4.12/sysbench- ➲
0.4.12.tar.gz | tar zx
$ cd sysbench-0.4.12
$ libtoolize --force --copy
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
$ mysqladmin create sbtest ← テスト用のデータベースを作成
$ sysbench --test=oltp --db-driver=mysql --mysql-table-engine=innodb --oltp-table- ➲
size=5000000 --mysql-user=root prepare ← テスト用のデータベースにテストデータを生成
```

▼図3　SysBenchの実行例

```
$ sysbench --test=oltp --num-threads=100 --db-driver=mysql --mysql-user=root --oltp-table- ➲
size=5000000 --max-requests=0 --max-time=600 run
……(省略)……
OLTP test statistics:
    queries performed:
        read:                            30391774
        write:                           10854202
        other:                           4341681
        total:                           45587657
    transactions:                        2170840 (3618.00 per sec.) ← 結果 transactions/sec
    deadlocks:                           1       (0.00 per sec.)
    read/write requests:                 41245976 (68741.95 per sec.)
    other operations:                    4341681 (7235.99 per sec.)
……(省略)……
```

これがデータの規模となります。100としたときすべてのテーブル合わせ8.3GBほどのデータが生成されます。

実行例は**図5**です。最終的な結果として、New-Order処理のTpmC(分間あたりのトランザクション数)が得られます。また、途中経過としてもトランザクション数が得られますので、グラフ化して結果のブレを見て安定性を確認したりもできます。

実行時のオプションとして、**-w**でwarehouseのサイズを指定、**-c**でクライアントのコネクション数(スレッド数)、**-l**で実行時間(秒)、**-i**で途中経過の表示間隔(秒)が表示されます。**-w**でウォームアップの時間(秒)も指定できます。

一度計測すると中のデータが書き換わり、再度実行時に同じ計測となりません。計測後、**mysqladmin drop tpcc**でデータベースを削除し再度生成し直すか、計測前にバックアップを取得してください。

安定した測定値を出すには数十分以上計測することをお勧めします、またデータの生成にも数十分～数時間かかり計測に時間を要しますので手軽さはないですが、しっかりとした計測をする場合にご利用ください。



次回予告

今回は3つのデータベースベンチマークツールを紹介しました。データベースのベンチマークはサーバ全体の総合的な性能を計測できる一方で、メモリの容量やディスクI/Oなどの性能に依存し、またMySQLやベンチマークツールのパラメータにより値ががらりと変わります。ベンチマーク中にtopコマンドなどでCPUやディスクI/Oどちらがボトルネックかまたはリソースを活かせていないか確認すると良いでしょう。

次回は今回のツールを用いて、それぞれのパラメータにより結果がどう変わるか見ていきます。またioDriveのためのチューニングを施し実践的なデータベースのベンチマークを行います。SD

▼図4　tpcc-mysqlのインストール

```
$ sudo yum -y localinstall http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.
noarch.rpm
$ sudo yum -y install bzr
$ bzr branch lp:~percona-dev/perconatools/tpcc-mysql
$ cd tpcc-mysql/src
$ make
$ cd ..
$ mysqladmin create tpcc ← テスト用のデータベースを作成
$ mysql tpcc < create_table.sql
$ ./tpcc_load localhost tpcc root "" 100 ← テスト用のデータベースにテストデータを生成
$ mysql tpcc < add_fkey_idx.sql
```

▼図5　tpcc-mysqlの実行例

```
$ ./tpcc_start -d tpcc -u root -w 100 -c 30 -l 3600 -i 60
……(省略)……
3480, 30283(0):1.736|3.832, 30268(0):0.472|1.581, 3029(0):0.311|1.102, 3028(0):2.150|3.
105, 3027(0):5.189|6.920
3540, 30061(0):1.673|3.225, 30036(0):0.453|1.621, 3006(0):0.316|1.318, 3004(0):2.128|3.
868, 3006(0):5.023|5.892
3600, 30651(0):1.631|3.955, 30629(0):0.444|1.836, 3065(0):0.313|1.328, 3067(0):2.219|3.
078, 3066(0):4.971|6.871 ← -iで指定した時間ごとにその間の結果が表示される(1つめはNew-Orderの値)
……(省略)……
<TpmC>
                30935.217 TpmC ← 結果 transactions/min
```

分散データベース「未来工房」

第7回

Riakアプリケーションとしての Riak CS(2)

――基本機能から内部構造までじっくり解説

Writer 上西 康太(うえにし こうた)　Bashoジャパン株式会社　kota@basho.com

先月号に引き続き、Riakを用いてアプリケーションを設計するときの方法や考え方を、Riak CSを例にし解説する。後編となる本稿では、Riak CSオブジェクトのライフサイクルの最後の部分である削除や、Riakに入っているデータ一覧の取得の方法やインデックスなど、おもにデータを問い合わせる方法を中心に解説する。

注)本稿は、筆者の意向により常体で表記している。

はじめに

先月号に引き続き、SQLもトランザクションもないデータベースであるRiakで、どのようにしてアプリケーションを設計したらよいかという問題とその解答例を、実際にRiakを利用するアプリケーション[注1]の例としてRiak CSを用いて解説する。

先月号では、Riak CSにおけるオブジェクトのライフサイクルを途中まで解説した。まずはこの続きとして、途中で別のクライアントから同じものを読まれる可能性がある場合でもデータを安全に削除するための設計を解説する。

次に、Riak CSのアクセス履歴から時系列データの管理方法の例を解説する。同時に、アクセス履歴やストレージの使用量の計算に利用しているMapReduceの集計処理の実装を解説する。また、先月号ではオブジェクトの一覧表示について解説すると予告したが、内容を再考したところ状態管理や時系列データの扱いのほうが重要であると思われるので、こちらを先に解説したい。最後に、Riak CSでセカンダリインデックス(以降2iと表記する)を使っている個所と、その理由を簡単に説明する。

先月と合わせて、この2回の連載でRiak CSの設計をほぼすべて解説したことになる。Riakを使ってアプリケーションを作成する方も、Riak CSを使ってシステムやサービスを構築する方も参考になれば幸いである。

注1) Riak CSは、Riakをデータベースとする REST APIを持つWebアプリケーションの実装とみなすこともできる。

Riak CSのガベージコレクション:範囲選択と削除

先月号では、Riak CSにおけるオブジェクトは**図1**のように状態遷移することを説明した。これと、Siblingsと呼ばれるRiakのしくみによって、並列にアップロードや読み出しが行われてもメタデータ(マニフェスト[注2])とブロックの関係が破壊されることなく、アップロードに成功した最新のオブジェクトが整合性のある状態で見えることが保証される。

クライアントからのDELETEリクエストから、実際の物理削除の流れは次のとおりとなる。先月号で簡単に説明したものに、少し補足を加えてある。

1. クライアントからのDELETEリクエストを受けとると、pending_delete状態のマニフェストをあらためてPUTする。これも読み込み時の解決によって、pending_deleteが優先されるようになる
2. pending_delete状態に変更した直後に、

注2) Riak CSでは、オブジェクトのタイムスタンプやサイズなどのメタデータをまとめたものをマニフェストという。また、マニフェストには必ずUUIDが付属しており、それによってオブジェクトの世代を内部的に管理する。

GCバケットにもエントリを追加し、scheduled_delete状態に変更する

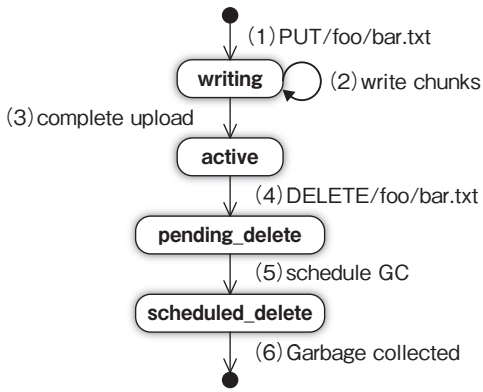
3. 定期的にGCプロセスがGCバケットのマニフェストを走査し、Riakでの削除操作が行われる。この時点でRiak CSから見た削除は完了する
4. 実際のRiak内部での削除は、このあとのbitcaskやleveldbのコンパクション時のタイミングでデータがファイルシステム上で削除されることとなる

pending_delete

クライアントからのDELETEリクエストを受け取ると、Riak CSは、そのオブジェクトにpending_deleteという状態を持ったマニフェストをPUTする。実際にはいったんマニフェストを読みだして、GETなど読み込み時の処理と同様に現状で最新のUUIDを決定する。それと同じUUIDを持ったマニフェストを、pending_deleteにマークしたものをRiakへPUTする。これが成功すると、そのUUIDに対応するマニフェストは無効化されたことになる。無効化により、読み込み時のSiblingsの解決の結果が変わり、このオブジェクトは削除されたように見えるだろう。

この状態は、その名のとおり、削除がpending(凍結中)という意味である。この状態で、処理を行っているRiak CSのプロセスが落ちたり、サーバが故障したりするとオブジェクトの状態は先に進むことなく、pending_deleteのまま残る。この状態であれば論理的には削除が成功したことになっている。このように、Riak CSはどのタイミングでどのサーバが故障してもデータ構造が壊れないように、アクセスやデータ構造が慎重に設計されている。

▼図1　オブジェクトの状態遷移の流れ



GCバケットへ登録

このあと、マニフェストのすべてのUUIDの複製を、riak-cs-gcという名前のRiakのバケットに、そのときのタイムスタンプをキーにしてPUTする。キーにはランダムな文字列がサフィックスとして付けられているため、キーが重複することはほとんどない[注3]。

仮にキーが重複しても、allow_multが有効になっているため上書きとはならず[注4]、その時刻で削除されたマニフェストがSiblingsとして併存することになる。ここでのポイントは、タイムスタンプをキーにしておくことだ。これによって、この後実際にGCプロセスが削除を行いやすくなっている[注5]。

scheduled_delete

GCバケットへの登録が成功すると、ようやくマニフェストの状態をscheduled_deleteに変更する。この状態変更も、マニフェストをRiakに直接PUTするだけで済む。これで、ようやくRiak CS上での論理削除が完了したことになる。そのあと、クライアントへ成功を返す。

GCによる物理削除

Riak CSのGCは、物理削除待ちとマークさ

注3) この機能は1.4.2で実装された。1.4.1以前のバージョンには、DELETEが集中するとGCバケットへのPUT集中しボトルネックになるという問題があった。

注4) Riakの設定ファイルapp.configに設定する項目の1つ。デフォルトはfalseになっている。

注5) ここでは詳細を説明しないが、実際にRiakにPUTされるタイムスタンプは、削除が実行された時間にLeeway secondsという値を足したものになる。直訳するとズレの時間という意味になるが、実際にGCによって削除が可能になるのはこの時間だけ後にずれる。これはDELETEが行われた瞬間にGETをしている別のクライアントがいたときに、そのオブジェクトのGETを失敗させないための工夫である。

れたマニフェストを実際に削除していく機能であり、ユーザからのリクエストとは異なるタイミングで、定期的に起動するバッチのように実行される。Riak CSプロセスの設定ファイルapp.configの中の`gc_interval`という項目で、秒単位で設定する。

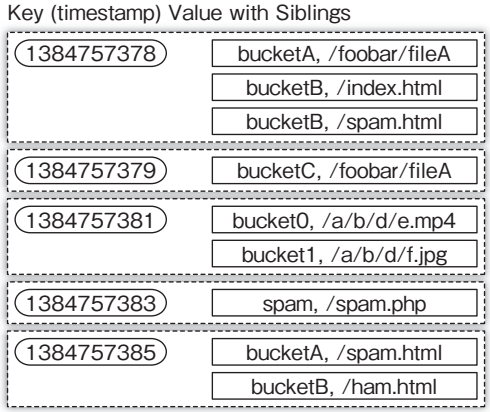
デフォルトでは900となっており、15分間隔でGCが実行されることを意味している。Riak CSの内部では、実際に削除をしているプロセスをGCデーモンと呼んでいる。

実際にGCデーモンは、GCバケットに対してキーをソートした状態で取得する。これは2i[注6]のAPIを普段とは異なる方法で使用する。インデックスに特殊なタグ**$key**を指定すると、キーを"0"から現在時刻まで範囲指定してソートされた状態で取得できる。これによって、ソートされた状態で、削除すべきマニフェストのキーを古いものから順に取得したことになる。

GCデーモンは、順に得られたタイムスタンプを見て、そこに入っているマニフェストをみながら、すべてのUUIDについてブロックの削除を行い[注7]、削除が完了すると、マニフェストを削除し、最後にGCバケット上のキーを削除する。これを、現在までのすべてのタイムスタンプについて繰り返すことで、Riak CS上での削除が完了する。

注6）セカンダリインデックス（Secondary Index）の略。キー以外のタグを付けることによってタグの値についてソートされた状態でキーを取得できる。

▼図2　GCバケットにおけるデータ構造



キーとしてタイムスタンプを利用し、値には1つのオブジェクトの複数のマニフェストが格納され、Siblingとして複数オブジェクトが格納される。単純化のため図には表記していないが実際には、1.4.2からはキーにランダムなサフィックスが付加され、DELETEの際にホットキーとなることを防いでいる。

時系列のデータとしてのGCバケット

GCバケット内のキーは、タイムスタンプベースの時系列データとみなすことができる。実際にセンサーなどのデータを高可用なデータストアに保存して利用したい場合には、この設計パターンを利用できる。つまり、

- **タイムスタンプがプレフィクスになったキー**
  たとえばタイムスタンプをキーにして、何らかのIDをサフィックスにする。書き込みはPUTするだけで完了する。キー空間はハッシュによって均等に分散しているため、最新のデータがあるところだけが負荷が高くなるようなことはない
- **$keyを用いた範囲指定のクエリ**
  タイムスタンプの区間を指定することで、必要なタイムスロットの時間を取得する。分析用であればそのまま用いることができるし、Riak CSのように古いデータから削除してもよい。さらに、ある範囲を指定して入力としてMapReduceで分析を分散処理することもできる

というパターンによって、時系列のデータを高速に書き込みまたは取得できる。読者がアプリケーションを設計する際にも、このパターンを覚えておくとよいだろう（**図2**）。

時系列データとしてのアクセス履歴

Riak CSには、ほかにも時系列データがある。それは、クライアントからのアクセス履歴だ。Amazon S3がデータの取り出し量に対して課

注7）Riakに対してDELETEを実行する。この時点でディスク上の容量がすぐに解放されるわけではなく、Bitcaskのコンパクションが起きるタイミングで容量が解放される。

金しているように[注8]、Riak CSでも、データの書き込み・読み出しの量を時系列でRiakに保存し、それをHTTP REST APIで取得できるようになっている。たとえば、2013年の4月1日から1ヵ月間の利用データを取得したい場合は、管理者アカウントで**リスト1**のようなURLにアクセスするとよい。

それぞれ、**example.com**はRiak CSが動作するホスト、**J_PJJS7NEHXJSQQV5XJH**は、アクセス履歴を取得したい対象のユーザのアクセスキー、**20140401T000000Z**は、取得したい期間の始点、**20140501T000000Z**は取得したい期間の終点である。

もしかすると、事前に**riak-cs-access flush**を実行しておかないと、利用履歴の集計が終わっていないかもしれないので、試す前に実行しておくとよい。Riak CSの管理者アカウントが設定されたs3cmdを使えば、**図3**のように任意のユーザのアクセス履歴を取得できる。

この機能の詳細は、Riak CSのドキュメントのアクセス統計の部分[注9]に詳細に記述されているのでそちらをご覧いただきたい。この機能を実現するための設計を解説する。

アクセス履歴の保存

まず、クライアントからのアクセスの履歴を、リクエストを処理した直後に毎回Riakに保存する方法が考えられるが、この方法では毎回アクセスがあるたびにRiakへのPUTが発生する。これではRiakにパフォーマンス上の影響があるため、RiakCSのメモリ上でユーザごとにアクセス履歴をある程度蓄積して定期的にRiakへまとめてPUTするようになっている。デフォルトでは**access_archive_period**という項目名で**app.config**上に3,600秒と設定されている。つまり、1時間ごとにアクセス履歴を書き出して永続化することを意味する。

このとき、Riakへの書き込みを行う際のキーは、ユーザごとに**<datetime>.<user_id>**となっている。アクセス履歴のデータは、JSONのオブジェクトになっており、複数のアクセス履歴について、サーバのID、アクセスの種類(書き込み、読み出し、エラー、他)、アクセスした量、アクセスした時間がそれぞれ記録されている(**図4**)。この時刻は、サンプリングされた区間を表す(英語ではSliceと呼ぶ)。

アクセス履歴の永続化は周期的に起こるので、おおよそその周期に合わせてキーを揃える。たとえば、3,600秒とすると、1日のうち0時～1時、1時～2時といったふうに一日を区切り、その間のアクセスは区間の開始時刻をキーとして、ユーザIDを末尾に追加したものをRiakに永続

注8) 厳密には、Amazonのデータセンターの外に転送して持ち出すトラフィックに対して課金している。

注9) "Querying Access Statistics" http://docs.basho.com/riakcs/latest/cookbooks/Querying-Access-Statistics/

▼リスト1　アクセス履歴を取得するURLの例

```
http://riak-cs.example.com/usage/J_PJJS7NEHXJSQQV5XJH/a/20140401T000000Z/20140501T000000Z
```

▼図3　s3cmdを使ったユーザのアクセス履歴の取得例
(ここでは表示を省略したが、結果はリクエストを受け取ったRiak CSのノードごとに、アクセスの種類に基づいて分類されている)

```
$ s3cmd get s3://riak-cs/usage/J_PJJS7NEHXJSQQV5XJH/a/20140401T000000Z/20140501T000000Z -
{
  "Storage": "not_requested",
  "Access": {
    "Errors": [],
    "Nodes": [ ... ]
  }
}
```

化する。このあたりの処理は複雑だが、実装は**riak_cs_access.erl, rts.erl**にあるので興味のある読者は参照されたい。

範囲指定でのアクセス履歴の取得

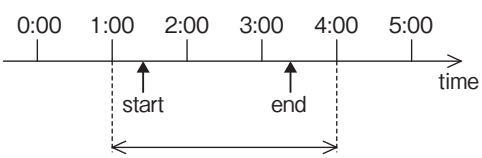
図3のようにリクエストを受け取ると、Riak CSは開始と終了時刻を含む区間を用いて、その間のキーとなるSliceを列挙し、**moss.access**というバケットの**<datetime>.<user_id>**となるすべてのキーについて順にGETを呼び出していく。これを結合して1つのJSONオブジェクトとしたものがそのままアクセス履歴となる。

ここでは、Riakのメジャーなデータアクセスの方法であるMapReduceを使っていない。その理由は、MapReduceのアクセスが広範囲に及ぶと、Riakに必要以上に負荷をかけることになり、それを避けられるケースではなるべく避けるほうが良いためである。仮にこれを2iで実現しようとしても、時間帯とユーザIDという2つのインデックスのタグでクエリをかける必要がある。しかし、どちらのタグも結果のデータ量を有意に絞ることが難しく、どちらかのタグについてはスキャンをしなければならなくなる。Riakが持つデータモデルにうまくフィットしないケースでも、キー設計とデータ構造を工夫することでスケールアウト可能な形にできるよい例だろう。

ユーザの使用量の集計：MapReduceを使って分散処理

最後に、ユーザのバケットの使用量の集計をMapReduceを使って行う方法を解説しよう。Riak CSには、ユーザが使用しているストレージ容量を定期的に集計する機能がある。これは、複数のRiak CSプロセスが起動している場合でも、どれか1つのプロセスでのみスケジュールされていなければならない。app.configの**storage_schedule**でそのスケジュールをセットできる。**storage_archive_period**というパラメータで、ストレージ集計の周期を設定できる[注10]。この機能を使用するためには、アクセス履歴と同様に、s3cmdを使えば**図5**のようにしてリクエストをRiak CSに送信できる。この集計はデフォルトでは動作しない[注11]ので、事前に**riak-cs-storage batch**を実行するか、app.configから設定して集計を一度動かしておく必要がある。

▼図4　クエリで指定した開始時刻と終了時刻が実際に選択するSliceの時間の範囲



このデータを作成する集計の処理は、ユーザがスケジュールするか、コマンドを用いて起動したタイミングで実行される。

まずは、**moss.users**というRiakのバケットから、全ユーザの情報を取得し、その中に列挙されているバケットひとつひとつについて、バケット内のすべてのキーを入力とするMapReduceの処理をバケットごとに起動する。バケット内のすべてのマニフェストをチェックして使用量を集計するという、スキャンに近いアクセスパターンであるため、MapReduceが非常に有効だ。MapReduceは、全サーバで分散処理されるため、データ量の増加に伴ってサーバの台数が増えても、処理時間は増えないというメリットがある。そして、1つのユーザアカウントが持っているすべてのバケットの集計処理が終了すると、それをアクセス履歴と同様に生成したキーでRiakにPUTする。

このバッチは、個々のユーザについて順番に集計が行われるため、ユーザ数が増加すると処理時間が伸びるという問題はある。これについては、今後の改善を期待しよう。

ストレージ使用量に対するクエリはアクセス履歴のしくみと同様、Sliceの単位がまとめて返るものになっているのでとくに説明はしない。

ここまで、時系列データに関する2つめのパターン、データ量が多い場合はある周期にデー

注10）周期というと語弊があるかもしれないが、アクセス履歴のSliceと似たような概念である。

注11）前述したように、二重に集計することを防ぐため。

タをアラインしてキーを決定論的にし、読み込み時のクラスタに対する負荷を下げるという工夫を解説した。

セカンダリインデックス

Riakには、PUT時にインデックスを指定して書き込む機能がある。これを使えば、バケットとキーの組み合わせがわからなくても、バケットとインデックスのタグからキーを取得できる。実際にアプリケーションを設計する際には、キーだけからデータを取り出すということはほとんどなく、インデックスからもデータを取得したいケースは非常に多いだろう。Riak CSでも、インデックスによるデータの取得を行っているところがいくつかある。

ユーザ情報

1つめは、ユーザ情報の取得である。ユーザは通常、アカウントのアクセスキーを基にGETして取得するが、それ以外にも、カノニカルIDやメールアドレスからユーザ情報を取得できるようになっている。これは、ACLでアクセスを許可する際にアクセスキーではなく、カノニカルIDやメールアドレスでアクセス権限をチェックするためである。

マルチパート

次のケースは、マルチパート情報の取得のためである。Riak CSでは、巨大なファイルを複数のHTTPのやりとりに分けてアップロードできる。これによって、TCPコネクションが途中で切れたり、サーバの故障などにより中断したのアップロードを、途中からやり直すことができる。再開するには、アップロードに成功したパート(アップロードしたいオブジェクトの断片)の一覧を確認してから、最後にアップロードが成功したパートの直後から、再送するデータを選ぶことになる。このとき、バケットの作成者(オーナー)が持っている一覧から、すべてのアップロード中のオブジェクトを表示する部分で、2iを使用している。

なぜここで2iを使うのか?

RiakCSにおいては、オブジェクトのPUT/GET/DELETEがホットパスであるから、先ほどの例のどちらもユーザからのアクセスが多いパスではない。これは、たまに呼ばれる種類のAPIリクエストを高速に処理したいというケースである。

こういうところで2iを使用する理由は、2iの実装にある。もともと、分散データベースでインデックスというしくみを構築する場合、Read/Write両方のワークロードで性能をスケールさせることはできない。それは、インデックスの更新とデータの更新が分散トランザクションでしか実行できないためだ。Riakではこのトレードオフにおいて、データ書き込みの可用性と性能に対して最適化する設計の選択をした。つまり、あるキーのインデックスは、同じマシンに保存されるようにPUTを設計し、インデックスからのデータの読み出しは、すべてのマシンに問い合わせを行って結果をマージするというしくみを選択した。このため、書き込みの性

▼図5　s3cmdを使ったユーザのストレージ使用量の取得例
(ここでは表示を省略したが、結果はユーザが作成したバケットごとに集計されたものが表示される)

```
$ s3cmd get s3://riak-cs/usage/J_PJJS7NEHXJSQQV5XJH/b/20140401T000000Z/20140501T000000Z -
{
  "Storage": {
    "Errors": [],
    "Samples": [ ... ]
  },
  "Access": "not_requested"
}
```

能はマシンを追加すればそれだけスケールするが、インデックスからの読み出しはマシンを追加してもスループットはスケールしない。これが理由でRiak CSでは一部のアクセスパターンだけを、インデックスを使って実装している[注12]。

まとめ

ここまで2回にわたって、Riak CSにみるRiakアプリケーションの設計技法を解説してきた。Stanchionのように排他制御を必要とするユーザ管理のしくみ、状態を持ったオブジェクトを安全に作成・更新・削除していく方法、時系列データのキー設計、それに対するクエリの出し方、MapReduceによる集計処理など多岐にわたるものとなった。

SQLもトランザクションもない世界で整合性のあるデータベースを設計するのはとても難しいが、使い方を限定するか工夫するかすると、従来のRDBMSでは成し得なかったような強力なデータベースを安価に構築できることがおわかりいただけたと思う。

少し難易度は高いかもしれないが、こういった技術を習得することで、読者の問題が解決し、データベースのスケーリングの問題で苦労することが減ることを望んでやまない。SD

注12）Riakでは、MapReduceの入力に2iのクエリ結果を利用できるようになっている。これは、同じようにすべてのノードにアクセスするパターンにはとても有効である。

Software Design plus　技術評論社



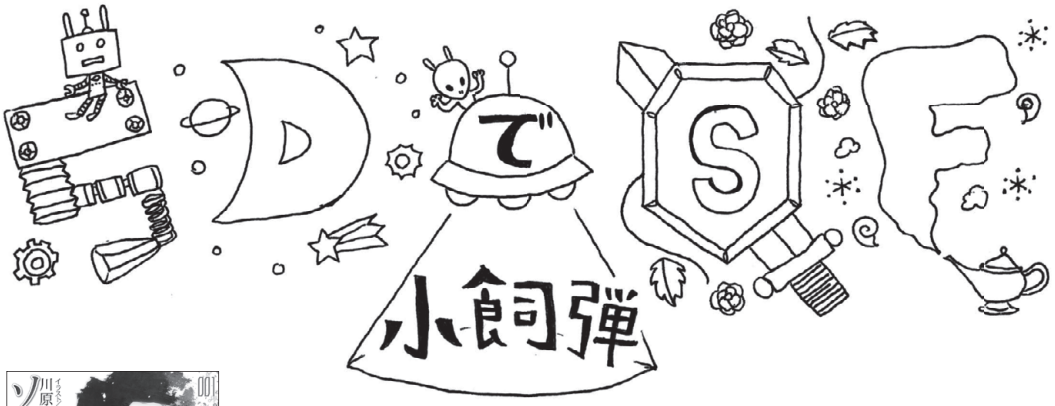
ニコラ・モドリック、安部重成 著
A5判／336ページ
定価2,919円（本体2,780円）
ISBN 978-4-7741-5991-1

大好評発売中！

おいしい Clojure 入門
関数型脳を育てる スペシャルメニュー

フランス人プログラマのニコラが、Clojureを包丁代わりにHadoopやRedisといった流行の素材を自由自在にプログラミングします。関数型プログラミングというと、敷居が高く扱うのが難しいのではないかと思い込んでいませんか。
本書は、環境構築からRubyとの連携をさらっと解説した後、NoSQLでCassandraも、遺伝的アルゴリズムも、JBossも、MQも、Herokuも、さらにはArduinoで組込まで、Clojureプログラミング技術のフルコースを紹介します。
プログラマの能力を大きく成長させる極上メニュー、ぜひ賞味してください！

こんな方におすすめ　関数型プログラミングに興味を持つエンジニア





『ソードアート・オンライン』（川原礫／電撃文庫）

というわけで始まりました新連載「SDでSF」。SFはSan Franciscoの略ではなく文芸の一ジャンルのSFです。

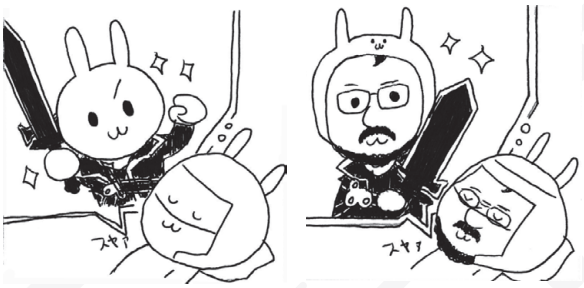
第1話に取り上げるのは、『ソードアート・オンライン』（川原礫／電撃文庫）。「え、SFじゃないの？」――その質問、待っていました。連載第1話に本作を取り上げた理由が、まさにそこにあるからです。「SFとは何か？」という設問は、SFというジャンルを愛する人々にとって「最高のエディタはEmacsかVimか？」に匹敵する答えなきFAQ。本作はその意味でまさに境界上にあります。

本作の出だしは単純にして明快です。仮想現実で実現された剣と魔法のゲーム世界で遊んでいたら、ログアウトできなくなった。しかもゲーム世界内で死んだら、現実のプレイヤーも死ぬ。『Hunter × Hunter』（冨樫義博）のグリード・アイランドもそうですが、デスゲームというのは今ではそれ自体がサブジャンルとして成立するほどファンタジーでよく使われる設定で、本作の登場以降は小説投稿サイトがこれで埋め尽くされるほどよく使われるようになりました。

では、何が本作をファンタジーであると同時にSFにしているのでしょう？　それが、仮想と現実の境界（インターフェース）。ゲーム内世界は剣と魔法が支配するファンタジーですが、そのゲーム内世界は今のMMORPGと同じサーバ内で動いているプログラムであり、しかし「コンソール」に相当するのは今なお実現されていない神経直結型ヘッドギア。本書がゲームに閉じ込められる数多のデスゲーム型ファンタジーと一線を画しているのは、「あっち側」だけではく「こっち側」もしっかり描写している点。『攻殻機動隊』（士郎正宗）が近未来SFであるように、本作もそうであると私は分類しています。

しかし作者は実際にMMORPGのプレイ体験、つまり実体験を下敷きにしているというところは21世紀的で、そしてそのデビューはさらに21世紀的でした。本作は文庫として出版する前に、Webで公開されていたのです。作者はやはりWebで公開していた『アクセル・ワールド』で「普通に」出版社主催の新人賞を受賞して商業デビューしたのですが、それ以前から公開された本作の方がさらに売れた。今や出版界は、「続きはWebで」ならぬ「続きは書籍で」が一般的なのです。

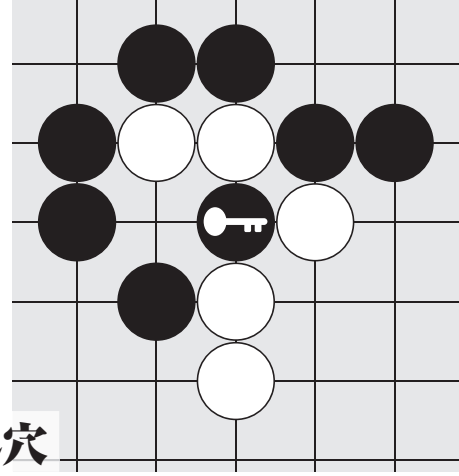
まさに20世紀のSFが実現した21世紀の現実が生み出したSFである本作は、手軽に読めて、それでいて境界というものを考えざるを得ない、SF入門にぴったりの一作。未読の方はぜひ第1巻だけでも。SD



タイトル／イラストby aico

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

【第七回】SSL/TLSデジタル署名の落とし穴

フィッシングサイトによる詐欺の被害が後を絶ちません。最近のフィッシングサイトは巧みに本物のサイトに似せてあり、見破るのも一苦労です。本物のサイトかどうかを見極めるときに頼りになるのが、SSL/TLSのデジタル署名です。しかし、これらも過信するのはよくありません。今回はデジタル署名の落とし穴について解説します。

贅沢(ぜいたく)ふせん

最近、「一万円札の代わりに百万円札が使われた」という事件を知りました。この百万円札は「贅沢ふせん」というおもしろグッズです。「百万円札と数字が書いてある」「福沢諭吉ならぬ贅沢諭吉がニヤリと笑っている」「百万円札が存在しないので偽造通貨には当たらない」などと報道されており、これだけの情報だと、こんなオモチャにだまされることが信じられません。ところが、「贅沢ふせん」の画像を検索して見てみると、全体のレイアウトと色が本物そっくりで、「確かにだまされるのも不思議ではない」と感じました。

でも逆を考えれば、人間が簡単に細部[注1]の違いを判断できるのなら、贅沢ふせんはもちろんのこと、世間で多くの被害が出ているフィッシングサイトも、これだけ大きな問題にはならなかっただろう、と思います。

ちなみにこの事件をうけて、オンラインショップからは「贅沢ふせん」はすべて引き上げられてしまい、現物はすでに入手不可能のようです。

最近では、オンラインショッピングで、有名量販店サイトによく似た作りのサイトがあって、(量販店サイトだと思って)買ってみたはいいが品物が届かない、といったトラブルが出てきているそうです。これまでのフィッシングとは、ちょっと違い、ちょっとした隙に情報を盗むのではなく、大胆に売買の操作までさせてしまう(でも代金だけ取って、品物は送らない)タイプなのだそうです。最初は「そんなに長々と操作しているならわかるものだろう」と思ったのですが、「贅沢ふせん」の事件を知り、こんな事件があるなら「有名量販店のサイトだと信じきっていることも、きっとあるんだろうな」と思うようになりました。

SSL/TLSによる安全性の確保

Webサイトの安全性といえば、SSL/TLSというキーワードが思い浮かぶと思います。この技術はhttpサーバとWebブラウザの通信を暗号技術によって安全に行うことを目的としています。

このことは「暗号を使っているから安心」ということに結び付きますが、技術的な意味で安全かどうかは、当たり前ですが運用にかかっています。そしてまた、技術的な(暗号学的な)意味で十分に安全に運

注1) といっても、贅沢ふせんは、本物のお札とはかなり違っており、デザインして売り出した人は、こんなに問題になるとは考えてはいなかったでしょう。

用しているかどうかといえば、疑問符がつくサイトも多いのは事実です。

SSLの登場

SSL（Secure Socket Layer）はNetscape Communications社が作った独自の暗号通信のためのプロトコルで、必ずしもHTTPプロトコル専用に作られたわけではありません。HTTPに着目したS-HTTPやコンテンツを暗号化してしまうSHENといった提案が同時期にありました。S-HTTPはHTTPを暗号化通信にしてしまうプロトコルで、SHENはコンテンツ自体を暗号化してしまうプロトコルでした。

一方、SSLはHTTPプロトコルではなく、TCP/IPの通信に一皮かぶせる汎用のプロトコルとして設計されました。そのため、SSLはファイル転送プロトコルのFTPでも、メール転送のSMTPでも、もちろんWWWのプロトコルであるHTTPでも、任意のアプリケーションのプロトコルに対し、暗号技術を組み込むことができる考え方になっています。通信では重要な考え方である、通信層を重ねる考え方に沿っています（**図1**）。

SSL 1.0はNetscape Communications社の内部的な仕様として存在していましたが、一般には公開されていません。のちにSSL 2.0が1995年、そしてSSL 3.0が1996年に公開されますが、IETF RFCのような公な文書として存在しているわけではなく、一企業が提案している仕様にしか過ぎませんでした。

その一方で、SSL 3.0はデファクトスタンダードとして多くのWebブラウザに使われ、広まってしまっていました。SSL 3.0の内容はRFC 6101で確認できます。これは、SSL 3.0が公式に参照できるドキュメントが存在しないので、過去にこのような規格があったということを示すためだけに作られたRFCの中でもあまり例をみない文書です。

SSLをもとに作られたTLS

TLS（Transport Layer Security）は、SSLをベースとしてIETFのWGグループで作られた規格で、TLS 1.0は1999年にRFC 2246で発行されました。改訂されたTLS 1.1はRFC 4346（2006年）、TLS 1.2はRFC 5246（2008年）です。アップデートのおもな理由は、すでに安全ではなくなってしまった暗号アルゴリズムの改訂のためです。

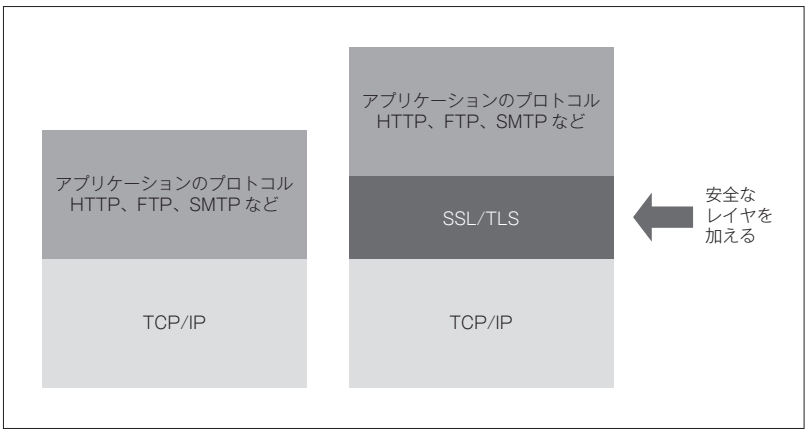
コンピュータの計算能力や理論的な暗号の分析が、日々向上しています。暗号の危殆化（きたいか）[注2]が発生するので、常に暗号技術はアップデートしていかなければなりません。TLS 1.2では、現在の技術レベルでは安全な水準を満たしています。

暗号技術というのは、いろいろな意味で、どんどん古くなっていくので、常に新しい技術を取り入れていかなければならないという宿命があります。ですからTLS 1.2の先も当然出てくるでしょう。

TLS利用における苦肉の策

最新の規格であるTLS 1.2を使っていれば、それなりに安心なのですが、互換性の問題もあってか、理想的な形では運用できていない現実がありま

◆図1　SSL/TLSのレイヤ関係



注2）過去に安全であった暗号アルゴリズムも段々とまるで劣化するように安全ではなくなっていくこと。

す。

Twitter、Facebook、Google Gmailの3つを確認してみました[注3]。いずれもTLS 1.2と表示されるのですが、暗号の組み合わせ(CipherSuite)が、それぞれ微妙に違っていました。

- Twitter：
  TLS_ECDHE_RSA_WITH_RC4_128_SHA
- Facebook：
  TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- Google Gmail：
  TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256

注目すべきは、Twitterの共通鍵暗号がRC4 128を選択していることです。RC4を選択しているのはSSL 3.0/TLS 1.0にはCBCモードの脆弱性(CVE-2011-3389)があります。これを回避するにはTLS 1.1/1.2にあるAES128 GCMを使うか、あるいはRC4を使うかどちらかにする必要があります。そこで互換性を考えて、苦肉の策でRC4を選択したのかもしれません。

RC4はすでに安全な暗号とは言えません。RC4は弱い暗号ですので利用は推奨されず、使うとしても互換性のために条件付きで使う暗号です。これまでにいろいろな攻撃方法が開発されており、2013年3月にもSSL/TLSのRC4に対して新しい攻撃法が開発されました。もうまったなしで安全な共通鍵暗号(AES GCMなど)へ移行すべきだと思います。

互換性の問題があるため、SSL 3.0/TLS 1.0のみしか使えない古いブラウザを使っているユーザがサービスを使えなくなる、というトラブルは一定量発生するでしょう。

Microsoft社が、この問題にブラウザ側で対応したMS12-006を出したのですが、このパッチを適用するとほかのSSL 3.0/TLS 1.0を使っているサーバにアクセスできないというトラブルが発生していました[注4]。

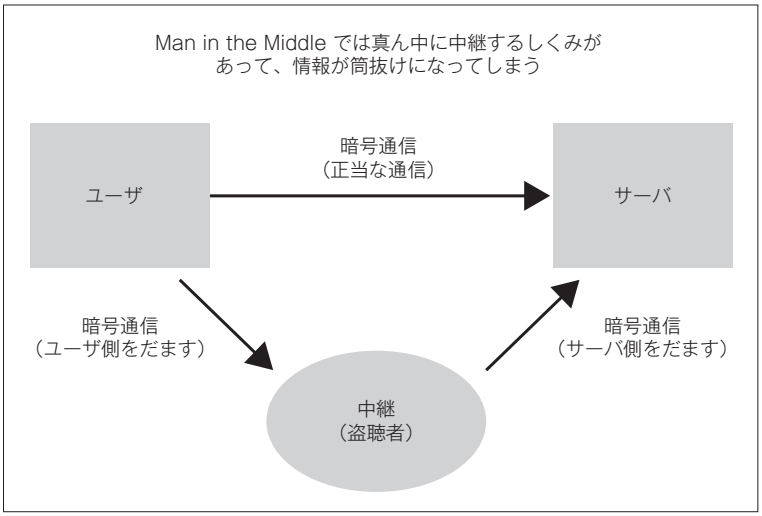
じつは、FacebookとGmailはブラウザ側が対応していなければ、TLS_ECDHE_RSA_WITH_RC4_128_SHAを選択するようにマルチで対応するようなことをしています。アクセスできないユーザがいるので、どうしてもしかたのないことなのかもしれませんが、やはり、あまり好ましいことではありません。

デジタル証明書

だまされること防ぐ手段として、SSL/TLSでデジタル署名を使って接続先が正しいものかどうかを認証します。

先ほどの話は通信路の安全確保のみの話でした

◆図2 MAN-IN-THE-MIDDLE攻撃



注3) 確認したのは2013年11月中旬です。

注4) 「マイクロソフトセキュリティ情報MS12-006 -重要 SSL/TLSの脆弱性により、情報漏えいが起こる(2643584)」http://technet.microsoft.com/ja-jp/security/bulletin/ms12-006

が、通信路が安全でも、相手の確認をしなければ途中で中継されて、そこで中身を全部見られているかもしれません。MAN-IN-THE-MIDDLE攻撃と呼ばれる方法ですが、なんのことはない読んで字のごとく「真ん中に人が挟まっていて、その人にだまされる」ということです（**図2**）。

これを防ぐには、接続先が正しいサーバであるかどうかを確認する必要があります。そのためにデジタル署名の技術を使います。デジタル署名を検証するには、事前に相手の（この場合はサーバの）検証鍵（公開鍵）が必要です。検証鍵を使ってデジタル署名を検証し、それが正しければ、その検証鍵の持ち主が作ったデジタル署名であるとわかります。

Twitter社やFacebook社から受け取った検証鍵が、本当にTwitter社やFacebook社のものであるかどうか、自分で直接確認できれば問題ないのですが、現実的にはほぼ不可能です。証明するオーソリティー（権威者／責任者）が必要になってきます。

公開鍵の認証モデルを定めているのがX.509です。そのX.509を用いているのが、SSL/TLSの認証です。

X.509

X.509といえばデジタル署名、デジタル署名といえばX.509と理解されていますが、これはX.500シリーズに含まれる規格の1つです。

ではX.500とは何でしょうか。ITU-T（国際電気通信連合電気通信標準化部門）がまだCCITTと呼ばれていた1988年に作られた規格で、ディレクトリサービス、つまり、どこに誰がいるのかといったことをディレクトリ構造（ツリー構造）を前提として一意に示すための規格です。そして、このディレクトリサービス自体は、X.400という電子メール交換および名前参照のために作られました。X.400は我々が日々インターネットでやりとりしている電子メール体系とはまったく異なる規格で、一般に普及することはありませんでした。

一方でX.500は、たとえばDAP（Directory Access Protocol）というのがあって、それ自体では普及しなかったのですが、それをベースに再定義したのがLightweight Directory Access Protocolで、その機能を実装しているオープンソースのOpenLDAPはユーザ管理などで有名かと思います。

X.509は「どこの誰かを（ツリー構造により）一意に指し示すための構造」「その人の情報と公開鍵の情報が正しいことを上位の人が保証して、その人の公開鍵で署名することで証明書とする」という役割をもった規格といえます。こうすれば、一番トップの公開鍵さえ確実であれば、ツリー構造に収まっている公開鍵は正しいものになります。

でも、X.509の規格は表現をするフォーマット（方法論）だけを与えていて、実際にどうツリーを作るのかは別の話なのです。

PKI公開鍵基盤

ここでやっとPKI（Public Key Infrastructure）公開鍵基盤の話が現れます。ここではSSL/TLSの話にのみ話題を絞ります。ここでは2つのキーワードが登場します。

- **登録局（RA：Registration Authority）**
  公開鍵の所有者を示す情報が正しいかどうかを確認するところ
- **認証局（CA：Certification Authority）**
  正しければ公開鍵に署名するところ

ここでは2つの違う性格のものが同時に存在しなければなりません。登録局はX.509に含まれている組織名や住所などが正しいかどうかを確認します。事務的な手続きです。また人間が介在する部分です。

「申請された公開鍵情報が正しい」と登録局が保証してくれたことを前提に、認証局はその公開鍵に署名を行います。そして、その署名された公開鍵を相手に渡すことで、その公開鍵が認められたものだとを示すことができます。

デジタル署名における疑問点

ここまで読んでこられた方は、いくつか疑問が出てくると思います。筆者でさえ、次のことが思い浮

かびます。

①X.509証明書が発行されて、それを確認するブラウザに入っている証明書は誰が確認したのか
②ツリー構造になっているが、その中間を署名しているところは信頼できるのか
③申請した内容をちゃんと確認しているのか
④①～③までが妥当だとして、どうしてそれが正しいと認識／確信できるのか

ブラウザに入っている証明書は正しいのか

信頼のおけるメーカー品を購入するような場合、問題は極めて小さいと思いますが、筆者の場合、いろいろなところのファイルサーバからUbuntuやDebianやCentOSをダウンロードしてインストールしています。ディストリビューションが入っているサーバにアクセスする際に、パスワードはありません。こちらは隠すほどの情報もないですし、何よりも暗号化をすると、それだけハードウェアに負荷がかかるので、ほしいとも思いません。

となると、このブラウザに事前に入っているCAの証明書は誰が担保しているのでしょうか。じつは、MAN-IN-THE-MIDDLEで別のファイルサーバなのかもしれません。筆者のシステムのように信頼のチェーンが存在しないものは、定性的には安全性を担保していないシステムとなります。

CA局が乗っ取られた場合はどうなるのか

「こんなことは、そうそうないだろう」と考えるかもしれませんが、すでに発生しています。2011年にオランダで認証局を運用しているDigiNotar社が外部から侵入され、500件を越える偽造証明書が発行されました。2012年には*.google.comへの不正なワイルドカード証明書が発行されていることが、Googleによって発見されました。これは2011年にトルコの認証局が誤って中間CA証明書を発行したことが原因であることが判明しました。

世界中にCA局やRA局を運用している会社はたくさんあります。そこの中の1つでも間違いを起こせば、その影響はインターネット全部に及びます。

申請は正しいのか

SSLサーバ証明書発行の価格を見ると、本当にピンからキリです。その違いは認証プロセスの違いにあると言えます。安ければそれなりに信頼度が低く、高ければそれなりに信頼度が高いと言えるでしょう。

たとえばEV SSL証明書は、業界の認証ガイドラインに沿ってチェックし、発行審査を厳しくするがゆえに、安全であると言えます。その代わり、その分の価格は高いということになっています。

そこでTwitter、Facebook、Google Gmailをチェックしてみました。

- Twitter：　VeriSign社EV SSL証明書
- Facebook：　VeriSign社Class 3証明書
- Google Gmail：　Google社内証明書（上位はGeo Trust Global CA）

TwitterはEV SSL証明書でブラウザのアドレスバーのところに社名が表示されます。FacebookはVeriSignのClass 3証明書ですので、会社の登記簿などを提出して確認するレベルの認証を受けていますが、ブラウザのアドレスバーのところに社名などは表示されません。Gmailは単純に鍵のマークだけが表示されますが、Gmail自体はGoogleが発行する証明書を持っていて、その権限はさらに上位のGeoTrust Global CAから受けています。

各々特徴はあるにはあるのですが、はっきり言って、これでどれだけの違いがあるのか、どう判断していいのか、コンピュータに詳しくないユーザにわかるとは思えません。なぜなら、この証明書はClass 3であるとか、いちいち確認しませんし、確認したとしてもClass 3の意味はこまごまと調べない限りわかりません。

Googleにいたっては、Firefoxで見ると「このWebサイトは認証されています」とありますが、「運営者は不明です」「認証局：Google Inc」と表示されます（**図3**）。知らない人が見ると、認証局はGoogle自身であるうえに運営者が不明と出てくるわけですか

◆図3　Gmailの不安になるような証明書表示



ら、相当不安になるのではないでしょうか。

どうして、それが正しいと認識できるのか

図3の表示ではGmailのGoogle社内で発行しているSSLサーバ証明書しかわかりません。さらに詳しく調べないと、その上位はGeoTrust Global CAで認証しているとは、わかりません。そして、どうしてそのようになっているのか、それで良いのかを説明するのは、一苦労です。ここまで書いてきたことを、筆者は、自分の75才になる母親に説明して納得させる自信がありません。

TLS/SSLは魔法の杖ではない

TLS/SSLはサイトの安全性の保証はしていない

TLS/SSLはサイトの安全性のほんの一部の機能を実現しているにすぎません。当たり前のことですが、TLS/SSLは「そのサイトがクラックされて乗っ取られているか」などの判断には使えません。高価なTLS/SSLのサーバ証明書を購入するときに、監査やあるいは保険がついてきます。それはTLS/SSLの機能を入れたから安全性が高まったのではなく、運用できて始めて安全性が保てるという、じつにあたり前の前提に立っているからです。

悪意のあるサイトでもSSLサーバ証明書を取れる？

悪意のあるサイトも、悪意のある行為をするまでは「悪意のないサイト」であるわけですから、SSLサーバ証明書を獲得した後に悪意の行為を働けば、SSLサーバ証明書を持っている悪意のあるサイトになります。愉快犯ではなく、計画的な犯罪者であれば、それを防ぐことはできません。もちろん、EV SSL証明書のようなものは取れませんが、年会費数千円のようなものはクレジットカード程度の信用度で取れます。

そして、ユーザはいちいちサーバ証明書の中身を確認するとは思えません。今、Amazonにアクセスしてみるとev SSL証明書ではなく、VeriSign Class 3です。ちなみに、AmazonはGoogle Chromeバージョン31.0でアクセスすると、プロトコルはTLS 1.0、共通鍵暗号はRC4 128を選択していました。

贅沢ふせんとTLS/SSL

一番最初の贅沢ふせんの話に戻りますが、大きく100万円と書いてあって福沢諭吉ではなく、贅沢諭吉がニヤリと笑っているにもかかわらず、ひっかかった人は全体の色やバランスでだまされて、1万円札として受け取っています。「贅沢ふせん」は明らかにお札ではないけれど、それっぽいように見えてしまっています。

TLS/SSLで接続すると、Google Chromeは、アドレスバーの鍵のアイコンとhttpsが緑になります。EV SSL証明書と同じ色です。ドメイン名だって、どことなく似ている名前にすることだって可能です。「安くて早いSSLサーバ証明書発行」という宣伝はみなさんも多く見かけると思います。もちろんそれ自体は悪くはありません。このしくみの限界なのです。

TLS/SSLは通信をしている途中のデータを安全にすることはできますが、人間を錯誤から守るのは、この手の技術では限界があります。人間が勘違いをしないような、もっと人間側に近いアシスト機能のようなものがあればいいのですが、まだまだ身近なものはありません。ユーザ中心（ヒューマンセントリック）なセキュリティが登場するには、もう少し時間が必要かもしれません。SD



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9回

インタラクティブ性を持たせよう（その3）卓上時計と写真の保存

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。今回はこれまで作ってきたブックアプリに、卓上時計の機能と写真を保存する機能を追加する方法を解説します。

今回のゴール

今回はブックアプリをただのブックアプリで終わらせないために、おまけ要素を加えたいと思います（App Store審査への対策の意味もあります）。画面上の好きなところを長押しすると、写真を卓上時計にしたり、フォトライブラリに保存できる選択肢が現れ、押したボタンに応じて処理が行われるという形を目指します（**図1**）。

おまけとはいえ、長押しのジェスチャ処理や選択肢を与える「UIActionSheet」はさまざまなアプリで応用が効くので、覚えておいて損はありません。一度にたくさんの新しいことを覚えるように思えますが、1つ1つは割と単純なしくみなのでステップを追って順番に組み込んでいきましょう。

おさらい：画面上の文字、ボタン、画像を定義する

今回組み込む機能の1つである卓上時計を成立させるためには、まず画面上の文字やボタンなど、余計なものを一時的にコードを使って隠す必要があります。最初にすべての文字、ボタン、画像がストーリーボードと関連付けられているか確認しましょう。これまでの連載の流れでは、第7回の「分岐」をテーマにした回で1ページ目に2つのボタンを配置しましたが、コード中で扱えるようには関連付けませんでした。そのときはボタン自体を操作する必要がなかったため、とくに定義しなかったのです。今回は画面から一度「見えなくする」必要性ができたので、ボタンも関連付けなければなりません。

それでは早速「Page1ViewController.h」に

▼図1　卓上時計&フォトライブラリへの保存機能イメージ



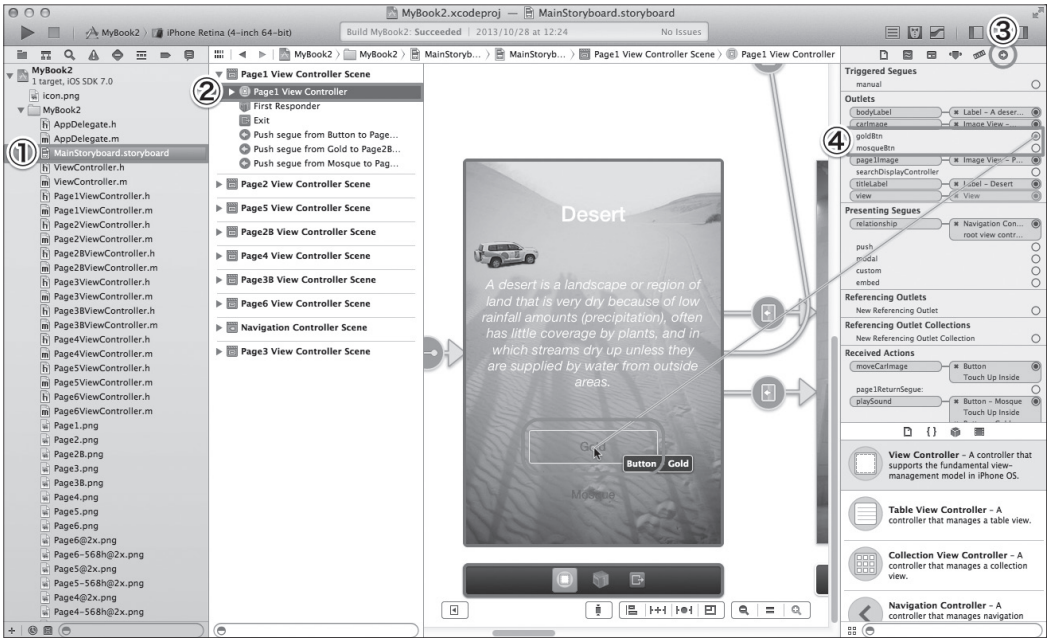
コードを追加しましょう。次の2行を追記してください。

```
IBOutlet UIButton *goldBtn;
IBOutlet UIButton *mosqueBtn;
```

コードを加えたら、次にストーリーボードを開いて（**図2-**①）1ページ目のView Controllerを選択します（**図2-**②）。Xcodeの右上ペインにある「Connections Inspector」のアイコンを押し（**図2-**③）、「Outlets」の中の「goldBtn」と「mosqueBtn」の「○」をドラッグし、それぞれのボタンとつなげましょう**図2-**④）。

ここまでは何度かやってきたことなのでおさらいのようなものですね。それでは、ジェスチャやアクションシートを加えるための準備に取りかかりましょう。

▼図2　ボタンのをストーリーボードに関連付ける



卓上時計と写真保存機能を付けよう！

ステップ1

まず、画面を長押ししたときに出現する「UIActionSheet」を呼び出すための準備をします。ヘッダファイル「Page1ViewController.h」を再び開いてください。ステップ1～3まではこのファイルへの操作です。`@interface Page1ViewController : UIViewController {`の行の「UIViewController」と「{」の間に`<UIActionSheetDelegate>`を追記します。これでアクションシートを呼び出す準備が整いました。

ステップ2

次にステップ1で追記した`@interface Page1ViewController : UIViewController <UIActionSheetDelegate>{ …… }`の中にインスタンス変数の宣言を加えます。`}`の上に**リスト1**の2行を加えてください。

▼リスト1

```
UILabel *clockLabel;
NSTimer *clockTimer;
```

この2つは時刻表示に必要なインスタンスになります。clockLabelは時刻を表示するためのラベル、そしてclockTimerは時刻を更新するために必要なタイマーになります。

ステップ3

ヘッダファイルの一番下にある@endの上にメソッドの宣言を記述します。次の5つをまとめて追加してください。

```
- (void)longPress:(UILongPressGestureRecognizer*)sender;
- (void)actionSheet:(UIActionSheet*)actionSheet clickedButtonAtIndex:(NSInteger)buttonIndex;

- (void)runTimer;
- (void)updateTime;

- (void)savePhoto;
```

1行目がロングプレス(長押し)を加えるためのメソッドです。2行目がアクションシートを呼び出すためのメソッドになります。runTimerとupdateTimerはどちらも時刻表示のために必要なメソッドです。そして最後のsavePhotoは指定した画像をフォトライブラリに保存するためのメソッドになります。

ここまでの操作で、ヘッダファイルは最終的に**図step3**になるはずです。

step3(Page1ViewController.hより)

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>

@interface Page1ViewController : UIViewController <UIActionSheetDelegate>{

    IBOutlet UIImageView *page1Image;

    IBOutlet UILabel *titleLabel;
    IBOutlet UILabel *bodyLabel;

    IBOutlet UIImageView *carImage;

    IBOutlet UIButton *goldBtn;
    IBOutlet UIButton *mosqueBtn;

    AVAudioPlayer *sound;

    UILabel *clockLabel;
    NSTimer *clockTimer;
}

@property (nonatomic, retain) IBOutlet UIImageView *page1Image;
@property (nonatomic, retain) IBOutlet UILabel *titleLabel;
@property (nonatomic, retain) IBOutlet UILabel *bodyLabel;

- (IBAction)playSound;

- (void)longPress:(UILongPressGestureRecognizer*)sender;
- (void)actionSheet:(UIActionSheet*)actionSheet clickedButtonAtIndex:(NSInteger)buttonIndex;

- (void)runTimer;
- (void)updateTime;

- (void)savePhoto;

@end
```

ステップ1で追加

「おさらい」で追加

ステップ2で追加

ステップ3で追加

ステップ4

インスタンス変数とメソッドの宣言ができたら、次は実装ファイルに実際の処理を書いていきましょう。「Page1ViewController.m」を開いてください。ステップ4〜10まではこのファイルへの操作になります。まずは起動と同時に時計を動かすための処理を加えることにします。

- (void)viewDidLoad内の最後に次のコードを追加してください。

```
clockLabel = [[UILabel alloc] initWithFrame:CGRectMake(0, 100, 320, 100)];
clockLabel.textAlignment = NSTextAlignmentCenter;
clockLabel.textColor = [UIColor whiteColor];
[clockLabel setFont:[UIFont fontWithName:@"HelveticaNeue-UltraLight" size:100]];
[self.view addSubview:clockLabel];
clockLabel.hidden = YES;
[self runTimer];
```

ここでは時刻を表示するためのラベル（clockLabel）をコードのみを使って追加しています。これまではラベルをストーリーボード上に配置し、コードと紐付けてきましたが、このようにコードだけでもたったの数行で色や配置位置、フォントまで指定することが可能です。下から2番目のclockLabel.hidden = YES; はラベルを隠すための処理です。起動時にはまだ見えなくしておきたいからです。そして最後の [self runTimer]; は起動と同時に時計を動かすための処理を呼び出しています。runTimerの処理は別に加えなければいけません。

ステップ5

次は時刻を更新する処理です。下までスクロールし、@endの上に次のコードを加えてください。

```
- (void)runTimer{
    clockTimer = [NSTimer scheduledTimerWithTimeInterval: 0.5
                                                  target: self
                                                selector: @selector(updateTime)
                                                userInfo: nil
                                                 repeats: YES];
}

- (void)updateTime{
    NSDateFormatter *formatter = [[NSDateFormatter alloc] init];
    NSDate *date = [NSDate date];
    [formatter setDateFormat:@"HH:mm"];

    [clockLabel setText:[formatter stringFromDate:date]];
}
```

- (void)runTimer{の中では0.5秒おきにupdateTimeという処理が繰り返し呼び出されるように命じられています。- (void)updateTime{の中では処理が呼ばれた時点での時刻を取得し、clockLabelに新しい時刻を表示させる処理が行われます。つまり、0.5秒おきに現在時刻の確認が行われ、実際に時刻に変化があれば最新のものに更新されるということです。

ステップ6

次にプログラムが長押しのジェスチャに反応するためにUIGestureRecognizerを追加します。UIGestureRecognizerは長押し以外にも、タップの回数、ピンチやドラッグなど、さまざまな指の動き（ジェスチャ）を識別できます。それぞれのジェスチャに応じて特別な処理を加えるときに非常に便利です。

もう一度**- (void)viewDidLoad**に戻り、ステップ4で加えたコードの後ろに、さらに次のコードを追加してください。

```
UILongPressGestureRecognizer *longPressGesture =
[[UILongPressGestureRecognizer alloc] initWithTarget:self action:@selector(longPress:)];

[self.view addGestureRecognizer:longPressGesture];
```

これでviewDidLoadの部分はステップ4と6の操作で**図step6**のようになっていればOKです。

self.view、つまりビュー全体に対して長押しのジェスチャを識別するよう命令を与え、長押しが識別されると同時にlongPress:の処理が呼ばれるようになります。つまり次はlongPress:の処理を書かなければいけないということです。

step6（Page1ViewController.mより）

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.

    if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPhone) {

        CGRect frame = [[UIScreen mainScreen] bounds];
        if (frame.size.height==568.0) {
            [page1Image setImage:[UIImage imageNamed:@"Page1-568h.png"]];
        } else {
        }
    }


    // 文字に装飾を加える
    // 影の形を設定
    NSShadow *shadowAttr = [[NSShadow alloc] init];
    [shadowAttr setShadowColor:[UIColor blackColor]];
    [shadowAttr setShadowBlurRadius:5.0];

    // タイトルに影を適用
    NSMutableAttributedString *titleText = [[NSMutableAttributedString alloc] initWithString:titleLabel.text];
    [titleText addAttribute:NSShadowAttributeName value:shadowAttr range:NSMakeRange(0, [titleText length])];
    titleLabel.attributedText = titleText;

    // 本文に影を適用
    NSMutableAttributedString *bodyText = [[NSMutableAttributedString alloc] initWithString:bodyLabel.text];
    [bodyText addAttribute:NSShadowAttributeName value:shadowAttr range:NSMakeRange(0, [bodyText length])];
    bodyLabel.attributedText = bodyText;

    clockLabel = [[UILabel alloc] initWithFrame:CGRectMake(0, 100, 320, 100)];
    clockLabel.textAlignment = NSTextAlignmentCenter;
    clockLabel.textColor = [UIColor whiteColor];
    [clockLabel setFont:[UIFont fontWithName:@"HelveticaNeue-UltraLight" size:100]];
    [self.view addSubview:clockLabel];
    clockLabel.hidden = YES;
    [self runTimer];

    UILongPressGestureRecognizer *longPressGesture =
    [[UILongPressGestureRecognizer alloc] initWithTarget:self action:@selector(longPress:)];

    [self.view addGestureRecognizer:longPressGesture];
}
```

ステップ4で追加

ステップ6で追加

ステップ7

一番下までスクロールし@endの上に次のコードを追記してください。

```
-  (void)longPress:(UILongPressGestureRecognizer*)sender {
    if (sender.state == UIGestureRecognizerStateBegan){

        UIActionSheet *actionSheet = [[UIActionSheet alloc] init];
        actionSheet.delegate = self;
        [actionSheet addButtonWithTitle:@"時計モード"];
        [actionSheet addButtonWithTitle:@"写真をダウンロード"];
        [actionSheet addButtonWithTitle:@"キャンセル"];
        actionSheet.cancelButtonIndex = 2;
        [actionSheet showInView:self.view];
    }
    else if (sender.state == UIGestureRecognizerStateEnded) {

    }

}
```

最初のifから始まる1行はジェスチャの識別がスタートした時点で発動する処理です。長押しは指が画面に触れてから0.4秒以上経過した時点で認識がスタートされるのですが、0.4秒が経過した瞬間にUIGestureRecognizerStateBegan（ジェスチャーがはじまった）ことが認識され、指を離した時点でUIGestureRecognizerStateEnded（ジェスチャーの終了）が認識されます。そのため、BeganとEndedで処理を分けてあげなければ2回同じ動作が繰り返されてしまう問題が起こります。今回はBeganの中にアクションシートを呼び出す処理を書き、Endedでは何も起こらないようにしました。

必要なボタンの数だけ、[actionSheet addButtonWithTitle:@" "]; を加え、それぞれの@" "の中にボタンごとのタイトルを書き込みます。

ステップ8

次はアクションシート内のボタンを押したときの処理です。@endの上にもう1つ別の処理を追記してください（**リスト2**）。

これで、ステップ5（タイマー）→ステップ7（長押し）→ステップ8（アクションシートの処理）までを加えた「Page1ViewController.m」は**図 step8**のようになります。

アクションシートはボタンごとに「0」から順に番号が振られます。そして、押された番号がbuttonIndexに入った状態で- (void)actionSheet:(UIActionSheet*)actionSheet clickedButtonAtIndex:(NSInteger)buttonIndexが呼び出されます。switch(buttonIndex){というのはbuttonIndexの値によって次にする処理

step8（Page1ViewController.mより）

```
- (void)runTimer{
    clockTimer = [NSTimer scheduledTimerWithTimeInterval: 0.5
                                                  target: self
                                                selector: @selector(updateTime)
                                                userInfo: nil
                                                 repeats: YES];
}

- (void)updateTime{
    NSDateFormatter *formatter = [[NSDateFormatter alloc] init];
    NSDate *date = [NSDate date];
    [formatter setDateFormat:@"HH:mm"];

    [clockLabel setText:[formatter stringFromDate:date]];
}

-  (void)longPress:(UILongPressGestureRecognizer*)sender {
    if (sender.state == UIGestureRecognizerStateBegan){

        UIActionSheet *actionSheet = [[UIActionSheet alloc] init];
        actionSheet.delegate = self;
        [actionSheet addButtonWithTitle:@"時計モード"];
        [actionSheet addButtonWithTitle:@"写真をダウンロード"];
        [actionSheet addButtonWithTitle:@"キャンセル"];
        actionSheet.cancelButtonIndex = 2;
        [actionSheet showInView:self.view];
    }
    else if (sender.state == UIGestureRecognizerStateEnded) {

    }

}

- (void)actionSheet:(UIActionSheet*)actionSheet
clickedButtonAtIndex:(NSInteger)buttonIndex {

    switch (buttonIndex) {
        case 0:
            //1番目のボタンが押された時の処理
            clockLabel.hidden = NO;
            titleLabel.hidden = YES;
            bodyLabel.hidden = YES;
            carImage.hidden = YES;
            goldBtn.hidden = YES;
            mosqueBtn.hidden = YES;
            break;
        case 1:
            //2番目のボタンが押された時の処理
            [self savePhoto];
            break;
        case 2:
            //3番目のボタン（キャンセル）が押された時の処理
            break;
    }
}
```

ステップ5で追加

ステップ7で追加

ステップ8で追加

▼リスト2

```
- (void)actionSheet:(UIAction
Sheet*)actionSheet
clickedButtonAtIndex:(NSInteger)
buttonIndex {

    switch (buttonIndex) {
        case 0:
            //1番目のボタンが押された
時の処理
            clockLabel.hidden = NO;
            titleLabel.hidden = YES;
            bodyLabel.hidden = YES;
            carImage.hidden = YES;
            goldBtn.hidden = YES;
            mosqueBtn.hidden = YES;
            break;
        case 1:
            //2番目のボタンが押された
時の処理
            [self savePhoto];
            break;
        case 2:
            //3番目のボタン(キャンセル)
が押された時の処理
            break;
    }
}
```

を選ぶための命令です。buttonIndexが「0」ならcase 0:の部分が「1」ならcase 1:の部分が実行されます。

1番目のボタン(case 0:)を押すと、隠れていた時計(clockLabel)が表示され、残りの文字や画像がすべて隠れます。2番目のボタン(case 1:)を押すと写真をフォトライブラリに保存させるための処理が呼び出されます。3番目(case 2:)のボタンはキャンセルなので、何も書く必要はありません。

時計そのものの処理はすでにできているので、この時点で1番目のボタンが押されると隠れていた時計が表示されます。時刻表示に関してはこれ以上何もする必要がありません。つまり2番目のボタンを押したときのsavePhotoの処理だけが必要というわけです。

ステップ9

次は、画像をフォトライブラリへ保存する処理を加えます。@endの上に、次のコードを追加してください。

```
- (void)savePhoto{
    UIImage *_saveImage = [UIImage imageNamed:@"Page1-568h@2x.png"];
    UIImageWriteToSavedPhotosAlbum(_saveImage, self, @selector(savingImageIsFinished:
didFinishSavingWithError:contextInfo:), nil);
}

- (void)savingImageIsFinished:(UIImage *)_image didFinishSavingWithError:(NSError *)_error
contextInfo:(void *)_contextInfo
{
    //エラーのとき
    if(_error){
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"エラー"
                                                        message:@"壁紙の保存に失敗しました。"
                                                       delegate:nil
                                              cancelButtonTitle:nil
                                              otherButtonTitles:@"OK", nil
                              ];

        [alert show];

    }
    //保存できたとき
    else{
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"完了"
                                                        message:@"壁紙をフォトライブラリに保存
しました。"
                                                       delegate:nil
                                              cancelButtonTitle:nil
                                              otherButtonTitles:@"OK", nil
                              ];

        [alert show];
    }

}
```

- (void)savePhoto{は、指定した画像をフォトライブラリへ保存するための処理です。作例では「@"Page1-568h@2x.png"」となっていますが、この部分はそれぞれのページごとで保存させたい画像の名前に変更する必要があります。

- (void)savingImageIsFinished:〜は- (void)savePhoto{が終わった時点で呼ばれる処理です。ここでは保存のエラー時と成功時にそれぞれ別のメッセージが表示されるようにしています。メッセージを表示させる理由としては、何らかの変化が起こらない限りユーザは実際に画像が保存されたかどうかの区別がつかず、何度もボタンを押してしまうかもしれないからです。こういう部分はアプリを作るうえで結

構重要なポイントになります。

ステップ10

さて、これですべての機能が動作するようになりましたが、最後に1つ加えなければならない処理があります。それは一度時計モードにした状態から、元の状態へ戻す処理です。

(IBAction)page1ReturnSegue:(UIStoryboardSegue *)segue{の中に**リスト3**のコードを加えましょう。

これは1つ先のページから現在のページに戻ってくるときに発動する処理です。つまり、1ページ目を時計モードにした状態で2ページ目へ進んだ場合、もう一度1ページ目に戻ってくるときにclockLabelが隠れ、消えていた文字や画像がまた見える状態に戻ります。時計モードから元の状態に戻す処理はどこに加えてもかまいません。今回の記事で覚えたことをうまく応用して、好きなタイミングで元に戻す動作を加えてみるのも良いかもしれませんね。

▼リスト3

```
titleLabel.hidden = NO;
bodyLabel.hidden = NO;
carImage.hidden = NO;
goldBtn.hidden = NO;
mosqueBtn.hidden = NO;
clockLabel.hidden = YES;
```

実行＆確認

最後にシミュレータを実行して動作確認を行いましょう。画面を長押しするとアクションシートが現れ、時刻を表示したり写真をダウンロードしたりできるはずです。ですが、これらの処理はページごとに加えなければいけないので、それぞれのページのViewControllerに同じようにコードを追記し、必要な部分だけ変更していく必要があります。「クラス継承」を覚えるとこのへんをもう少し楽にすることも可能なので、興味のある方は独自にリサーチしてチャレンジしてみてはいかがでしょう？SD

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「江頭ジャマだカメラ」（無料総合1位獲得）、「i列車の車窓から―そうだ! 京都行こう!―」（バーチャル旅行アプリ）などをリリース。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44回

[アプリ開発2013] 5 ストレージのデータ保存基礎を学ぼう

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！

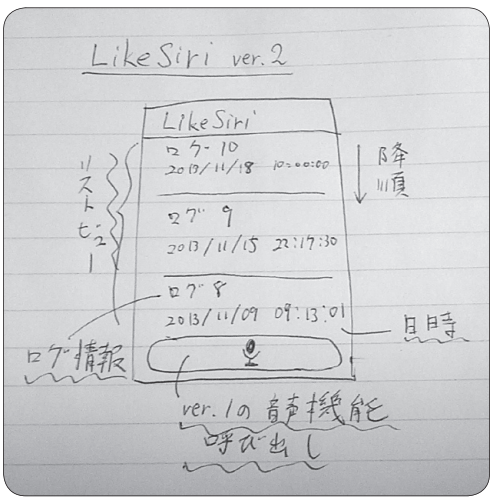
重村 浩二　SHIGEMURA Koji
日本Androidの会 中国支部長
Mail k-shigemura@android-group.jp

アプリをバージョンアップしよう

前回ではLikeSiriというアプリをとおして、アイデアを仕様に落としてから形にするところまでを見ていきました。動くアプリがあれば、家族や友人などの身近な方々に「これ作ってみたんだけど、どう？」と聞いて、フィードバックも得やすいものです。一定の評価を得られそうなら、Google Playストアへの公開を考えていくことにもなるかと思います。

今回は、前回のアプリにストレージの機能追加をして、読者の皆さんが継続してアプリを改善しながら作り上げていくことを、もっとイメージしてもらえるように進めてみたいと思います。

写真1　LikeSiriのバージョンアップ案



ラフ書きでイメージを具現化しよう

アプリのバージョンアップを行うきっかけはさまざまです。ユーザからの不具合報告や小さな機能改善など、ユーザからの要望に応える際、課題解決へのアプローチは状況に応じて変わってくるものだと思います。その中でも、ユーザの操作が変わるような変更の場合には、ユーザにどんなUIを提供するのかを再度見直す必要が出てきます。

筆者の場合、バージョンアップ時には、前回モックを作るために行ったラフ書きを再度実施しています。紙上でラフ書きを行うことで、アプリを変更してユーザにどのような体験を提供したいのかを視覚的に捉えることができるためです。実現したい機能の仕様を箇条書きで書くことも行いますが、ユーザの体験を起点にすることが、素早く必要な機能を実装するための近道ではないかと考えています。

前回作成したLikeSiriのバージョンアップのために、**写真1**をノートに書き出しました。

音声認識のボタンを画面下に移動して、音声認識のログをリストビューで表示するような作りとしています。また、今回の画面バージョンをベータ版の位置づけとして、前回の画面レイアウトと切り替えることができるような仕掛け

▼表1　プリファレンスへのデータ格納のイメージ

key	value	type
key_state	0	int
key_publish	技術評論社	string
key_title	ソフトウェアデザイン	string
key_visible	false	boolean

も盛り込みたいと思います。このしくみを実現するためには、次の機能を導入する必要があります。

- 画面切り替えの実装(プリファレンス)
- ログの保存(SQLite)
- ログの表示(ListView)

次項では、LikeSiriの改修に必要なデータ保存の基礎知識を見ていきたいと思います。サンプルコードを公開[注1]してありますので、そちらも参考にしながら読み進めてください。

▼リスト1　プリファレンスからデータの読み込み

```
// プリファレンスの取得
SharedPreferences prefs = getPreferences(Context.MODE_PRIVATE);
if (!prefs.getBoolean(KEY_UI, false)) {
    // 新UIの設定
    setContentView(R.layout.activity_main2);
} else {
    // 旧UIの設定
    setContentView(R.layout.activity_main);
}
```

▼リスト2　プリファレンスへのデータの書き込み

```
if(item.getItemId() == R.id.action_ui) {
    // UI切り替え
    SharedPreferences prefs = getPreferences(Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = prefs.edit();
    if (!prefs.getBoolean(KEY_UI, false)) {
        editor.putBoolean(KEY_UI, true);
        editor.commit();
    } else {
        editor.putBoolean(KEY_UI, false);
        editor.commit();
    }

    // Activityの再起動
    finish();
    startActivity(new Intent(this, MainActivity.class));

    return true;
}
```

データ保存の基礎知識

Androidで提供されているデータの保存方法として、頻繁に使用されているのが今回紹介する「プリファレンス」と「SQLite」です。ここからは、これらについて、具体的な利用方法を解説します。

プリファレンスによるデータ保存

プリファレンスは、**表1**にあるように、keyとvalueの組み合わせでデータを保存するしくみです。検索性などはそれほど優れていませんが、とにかく容易にデータの保存が可能なため、Androidでは設定画面などでよく利用されています。

LikeSiriの場合のプリファレンスを取得する実装が、**リスト1**となります(次ページのコラム参照)。

リスト1では"key_ui"(プログラム上は定数でKEY_UIと定義しています)というキーの値を読み込み、新旧UIの切り替え処理を行っています。このとき、データ格納時の型に合わせてgetBooleanメソッドを呼び出している点がポイントです。ほかにもboolean、int、float、long、stringといった型に対応したメソッドが標準で用意されていますので、格納したいデータに合わせたメソッドを用いましょう。

逆に、データを書き込む場合の実装が、**リスト2**となります。

書き込みを行う場合にはSharedPreferences.Editorを利用します。putBooleanメソッドで書き込みたいデータ(true or false)をセットし、commitメソッドを呼び出すことでプリファレンスへの書き込みを行うことができます。

リスト2からもわかるとおり、とても簡単に

注1) サンプルは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/2014/201401/support

利用できるプリファレンスですが、欠点は冒頭でも述べた検索性やデータの加工にはSQLほど向いていない点が挙げられるでしょう。シンプルなデータ保存には大変重宝する機能ですので、状況に応じて利用するのがベターです。

SQLiteでのデータ管理

Androidでは、組み込み向けのデータベースとして標準でSQLiteを採用しています。データベースというだけあって、しくみはプリファレンスに比べて高機能です。

SQLiteをAndroidで扱うために、SQLite OpenHelperというヘルパークラスが用意されています(**図1**)。メソッドで簡単にデータの読み書きを実装できる仕掛けとなっていて、大変使いやすいクラスです。使い方はSQLiteOpenHelperを継承したクラスを用意し、データベースを生成する必要があります。とくに重要なメソッドが次になります。

- コンストラクタ
- onCreateメソッド
- onUpgradeメソッド

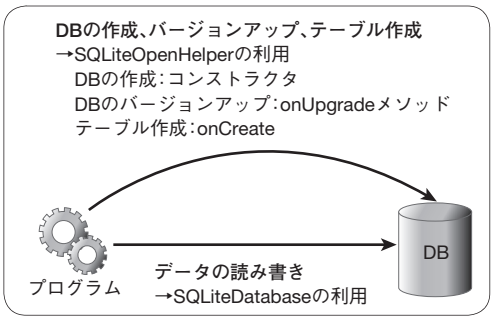
リスト3にあるように、SQLiteOpenHelperを継承したクラスのコンストラクタからsuperメソッドでスーパークラスを呼び出す際、引数としてデータベース名などを指定して渡すことにより、データベースを生成することができます。そして、onCreateメソッド内でexecSQLメソッドを呼び出し、指定したテーブルが存在しなければ、テーブルの作成を行うSQLを実行しています。

ここで出てくるDB_NAMEが作成されるデータベース名となり、DB_VERSIONがデータベースのバージョンとなります。このバージョンが、継続してアプリを更新していく際に重要なキーとなっています。

データベースにテーブル追加などを行いたいタイミングでDB_VERSIONをカウントアップすることで、データベースのバージョンアップを行うと判断され、onUpgradeメソッドが呼び出されます。onUpgradeメソッドの引数にあるoldVersion、newVersionに新旧のバージョン値が格納されて渡されてきますので、こちらを参照してどのようなバージョンアップ処理を行う必要があるかを判断します。

ユーザがアプリの更新ごとに必ずバージョンアップしてくれていればよいですが、中には自動更新を止めて、任意のタイミングに手動でバー

図1　プログラムからDBへのアクセスのしくみ



Column

SharedPreferencesの取得方法に注意しよう

SharedPreferencesの取得には複数のメソッドが用意されており、それぞれで格納される場所とファイル名が異なります。

- **ContextWrapper#getSharedPreferences メソッド**
  /data/data/[パッケージ名]/shared_prefs/[指定したファイル名].xml
- **PreferenceManager#getDefaultSharedPreferencesメソッド**
  /data/data/[パッケージ名]/shared_prefs/[パッケージ名]_preference.xml
- **Activity#getPreferencesメソッド**
  /data/data/[パッケージ名]/shared_prefs/[呼び出し元のActivityクラス名].xml

書き込み時と読み込み時の場所が正しく一致するように注意してください。

ジョンアップを行うユーザもいます。そのため、DBのバージョンも1→2→3というパターン以外に、1からいきなり3にバージョンアップを行うユーザがいることを考慮した作りにしておく必要があるでしょう。

生成したテーブル上にデータを書き込むために、**リスト4**にあるような処理を行います。データベースの書き込みにはContentValuesを使って、書き込むデータをセットする必要があります。**リスト4**ではLikeSiriの実行ログとタイプ（地図呼び出しなのか、電話呼び出しなのか）、処理日時の情報を設定しています。そして、先頭のgetWritableDatabaseメソッドで取得したSQLiteDatabaseのインスタンスを利用して、insertメソッドを呼び出してテーブルへのデータ挿入を実行しています。

▼リスト3　DBの作成・バージョンアップ、テーブルの作成

```
public class LogDBHelper extends SQLiteOpenHelper {
    private static LogDBHelper instance = null;

    private static final String DB_NAME = "likesiri2.db";  // DB名
    private static final int    DB_VERSION = 1;  // バージョン
    public static final String DB_TABLE = "tblog";  // テーブル名
    …省略…
    public LogDBHelper(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        // テーブルの生成
        db.execSQL("create table if not exists "
                + DB_TABLE
                + "(_id integer primary key autoincrement, "
                + "log text, "
                + "type integer, "
                + "date text)");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // テーブルの再生成処理を行うようにしています
        db.execSQL("drop table if exists " + DB_TABLE);
        onCreate(db);
    }
```

▼リスト4　テーブルの書き込み処理成

```
public void insertEventLog(int eventType, String log) {
    // 書き込み可能なデータベースの取得
    SQLiteDatabase db = getWritableDatabase();

    // ContentValuesに書き込むデータをセット
    ContentValues values = new ContentValues();
    values.put("log", log);
    values.put("type", eventType);
    // 日時の情報は、フォーマットを変換後格納
    Date date = new Date();
    SimpleDateFormat format = new SimpleDateFormat("yyyy/MM/dd, kk:mm:ss", Locale.JAPAN);
    values.put("date", format.format(date));

    // テーブルに挿入
    db.insert(DB_TABLE, null, values);
}
```

▼リスト5　テーブルからの読み込み処理

```
@Override
public Cursor loadInBackground() {
    // ログ取得用のCursorインスタンスをSQLiteから取得
    LogDBHelper logHelper = LogDBHelper.getInstance(getContext());
    SQLiteDatabase logDB = logHelper.getReadableDatabase();
    Cursor cursor = logDB.query(LogDBHelper.DB_TABLE, LogDBHelper.LOG_PROJECTION,
            null, null, null, null, "_id desc", null);

    return cursor;
}
```

▼表2　queryメソッド

構文	
Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)	
引数	解説
table	テーブル名
columns	テーブルから取得するカラム
selection	取得条件
selectionArgs	取得条件用の引数
groupBy	集計の条件指定
having	集計の条件指定
orderBy	並び順の指定
limit	項目の取得件数

テーブルからデータを読み込みたい場合には、**リスト5**にあるような処理を行います。

データの読み込み処理には次項で説明するLoaderが深くかかわっていますので、その点については次項で述べたいと思います。ここではgetReadableDatabaseメソッドで取得したSQLiteDatabaseのインスタンスを利用して、queryメソッドでデータの取得を行っています。なお、queryメソッドの引数は**表2**のようになっています。

リスト5ではqueryメソッドにorder byの指定を行い、取得時に_id(データ挿入時に自動でつく番号)を降順に取得できるような指定を行っています。

ListViewとデータベースをつなぐLoader

データベースからのデータ取得の肝となるのが、Loaderというしくみです。Honeycomb(Android 3.0)から追加されたしくみで、データベースからデータを読み込む際に使うことを推奨されています。

今回は処理状況のログを表示するために、ListViewを利用します。ListViewは縦方向に任意のデータをリスト形式で表示できるウィジェットで、Twitterのタイムラインなどをイメージするとわかりやすいかと思います。

ListViewの利用には、XxxxAdapter(Xxxxにはさまざまな名前が入ります)というクラスのインスタンスを用意し、データとListViewとを結び付ける必要があります。データベース内のデータをListViewに結び付けたい場合には、SimpleCursorAdapterを利用するのが便利でしょう。SimpleCursorAdapterに渡すデータを取得するのが、Loaderというしくみです。

Loaderを利用するには、次の実装を行う必要があります。

- Loaderの初期化(**リスト6**)
- Loaderが生成されたときのコールバック実装(**リスト7**)
- Loaderでデータ取得が完了したときのコールバック実装(**リスト7**)
- Loaderがリセットされたときのコールバック実装(**リスト7**)

onCreateメソッド内でinitLoaderメソッドを呼び出してLoaderの初期化を行い、Loaderの生成時・データ取得完了時・リセット時のそれぞれのコールバックメソッド(onCreateLoaderメソッド、onLoadFinishedメソッド、onLoaderResetメソッド)を実装します。

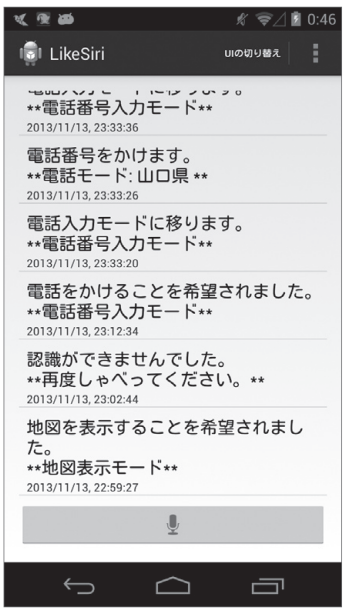
LikeSiriではonCreateLoaderメソッドで渡す必要があるCursorのインスタンスを、SimpleCursorLoaderというオープンソースのコードを用いて取得するようにしています。SimpleCursorLoaderを用いることで、SQLiteからのデータベース読み込みを簡単に実装することができます。

SimpleCursorLoader.java
URL https://gist.github.com/casidiablo/1217628

まとめ

これでLikeSiri ver.2の完成です(図2)。今回はLikeSiriにログ機能を付けていく中で、データ保存の基礎となるプリファレンスとSQLiteについて学びました。データ保存方法にはファイルに保存する方法や、クラウドにデータを保存する方法など、今回の手法以外にもさまざまなやり方があります。しかし、スマートフォンという限られたリソースで動かすことを考えれば、ローカル上での手法は外すことができない必須の知識です。ぜひご自身でアプリを作られる際に、導入してみてください。SD

▼図2 LikeSiri ver.2の画面



▼リスト6 Loaderの初期化

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    …省略…
    // Loaderの初期化
    getSupportLoaderManager().initLoader(0, null, this);
}
```

▼リスト7 Loaderの各種コールバック実装

```
@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    return new LogCursorLoader(this);
}

@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
    // 読み込んだCursorインスタンスを設定
    mAdapter.swapCursor(cursor);
}

@Override
public void onLoaderReset(Loader<Cursor> loader) {
    // リストビューに空を設定
    mAdapter.swapCursor(null);
}
```

重村 浩二(しげむら こうじ) 日本Androidの会 運営委員 中国支部長

日本Androidの会にて運営委員、中国支部長として毎月山口県・広島県を中心に自作アプリの発表やハッカソン、ハンズオンなどを中心とした勉強会を精力的に開催。個人としても、雑誌への寄稿などを中心に活動中。
URL http://buildbox.net/ Twitter @shige0501

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16回 PCIパススルー その2「VT-dの詳細」

浅田 拓也（ASADA Takuya）Twitter @syuu1228

はじめに

前回は、PCIパススルーとIOMMUの概要について解説しました。今回は、VT-dの詳細について解説していきます。

前回のおさらい

PCIデバイスが持つメモリ空間をゲストマシンのメモリ空間にマップすることによりPCIデバイスをパススルー接続できますが、DMAを使用するときに1つ困った問題が生じます。

PCIデバイスはDMA時のアドレス指定にホスト物理アドレスを使用します。ゲストOSは、ゲストOSは自分が持つゲスト物理ページとホスト物理ページの対応情報を持っていないので正しいページ番号をドライバに与えることができません。

そこで、**物理メモリとPCIデバイスの間にMMUのような装置を置きアドレス変換を行う**方法が考え出されました。このような装置を**IOMMU**と呼びます。DMA転送時にアドレス変換を行うことで、パススルーデバイスが正しいページへデータを書き込めるようになります(**図1**)。Intel VT-dは、このような機能を実現するためにチップセットへ搭載されたIOMMUです。

VT-d の提供する機能

VT-dが提供する機能には、次のようなものが挙げられます。

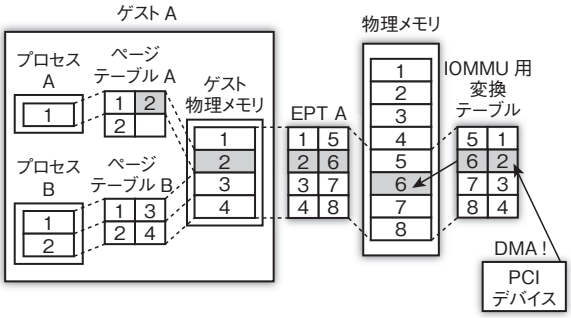
- IO device assignment：特定のデバイスを特定のVMに割り当てるための機能
- DMA remapping：仮想マシン上へDMAするためにアドレス変換を行う機能
- Interrupt remapping：特定のデバイスから特定のVMへ届くように割り込みをルーティングする機能
- Reliability: DMA・割り込みエラーをシステムソフトウェアに記録・レポートできる

今回はVT-dによるアドレス変換の話を解説することを目的としているため、このうち"DMA remapping"機能に絞って解説を進めていきます[注1]。なお、VT-dに関するより詳しい内容は"Intel Virtualization Technology for Directed I/O Architecture Specification"という資料にて解説されているので、こちらをご覧ください[注2]。

注1）より正確には、DMA remapping機能のうち"Requests-without-PASID"であるものについてだけを解説しています。

注2）http://www.intel.co.jp/content/www/jp/ja/intelligent-systems/intel-technology/vt-directed-io-spec.html

▼図1 IOMMUを用いたDMA時のアドレス変換



アドレス変換テーブル

VT-dでは、アドレスリマップ対象のデバイスごとにCPUのMMUと同様の多段ページテーブルを持ちます。デバイスごとのページテーブルを管理するため、PCIデバイスを一意に識別するBus Number・Device Number・Functionの識別子から対応するページテーブルを探すための2段のテーブルを用います。

1段目はRoot Tableと呼ばれ、0から255までのBusナンバーに対応するエントリからなるテーブルです。このテーブルはアドレス変換時にVT-dから参照するため、Root Table Address Registerへセットされます。Root Tableエントリのフォーマットを**表1**に示します。

Root tableエントリはcontext-table pointerフィールドで2段目のテーブルであるContext-tableのアドレスを指します。Context-tableはRoot tableエントリで示されるBus上に存在するDevice 0-31・Function 0-7の各デバイスに対応するページテーブルを管理しています。Context-tableエントリのフォーマットを**表2**に示します。

Context-tableエントリはsecond level page translation pointerでページテーブルのアドレスを指します。ページテーブルの段数はaddress widthフィールドで指定されます。**図2**に4段ページテーブル・4KBページを使用する場合のアドレス変換テーブルの全体図を示します。ここで使用されるページテーブルエントリのフォーマットは通常のページテーブルエントリと若干異なるのですが、ここでは解説は割愛します。

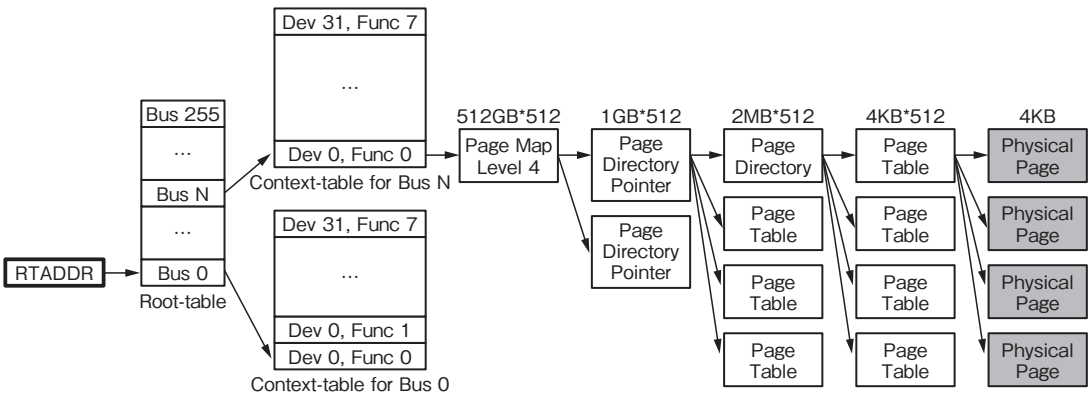
▼表1　Root table entry format

bits	field	description
127:64	reserved	予約フィールド
63:12	context-table pointer	Context-tableのアドレス
11:1	reserved	予約フィールド
0	present	このエントリが有効化どうか

▼表2　Context-table entry format

bits	field	description
127:88	reserved	予約フィールド
87:72	domain identifier	ドメインID　このエントリがどのVMに属するかを示す
71	reserved	予約フィールド
70:67	ignored	無視されるフィールド
66:64	address width	ページテーブルの段数を示す （0x0：2段、0x1：3段、0x2：4段、0x3：5段、0x4：6段）
63:12	second level page translation pointer	アドレス変換に使用するページテーブルエントリのアドレスを指定する
11:4	reserved	予約フィールド
3:2	translation type	アドレス変換時の挙動を設定
1	fault processing disable	0x0：フォールトレコード・レポートを有効化 0x1：フォールトレコード・レポートを無効化
0	present	このエントリが有効化どうか

▼図2　VT-dのアドレス変換テーブル全体図（例）



フォールト

変換対象になるアドレスに対する有効なページ割り当てが存在しない場合、または対象ページへのアクセス権がない場合、VT-dはフォールトを起こします。フォールトが発生した場合、メモリアクセスを行おうとしたPCIデバイスはアクセスエラーを受け取ります。OSへは、MSI割り込みを使用して通知されます。

IOTLB

IOMMUのアドレス変換を高速に行うには、通常のMMUと同じようにアドレス変換結果のキャッシュが必要です。通常のMMUではこのような機構のことをTLBを呼びますが、IOMMUではIOTLBと呼びます。通常のMMUのTLBでは、TLBエントリが古くなったときにinvalidateと呼ばれる操作によりエントリを削除します。このときのinvalidateの粒度は、グローバルなinvalidate・プロセス単位のinvalidate[注3]・ページ単位のinvalidateなどが選べます。VT-dのIOTLBでは、グローバルなinvalidate・デバイス単位のinvalidate・VM単位（ドメインと呼ばれる）のinvalidate・ページ単位のinvalidateが行えるようになっています。

Context-cache

IOTLBに類似していますが、VT-dではContext-table entryもキャッシュされています。これについても場合によってinvalidate操作が必要になります。

DMARによるIOMMUの通知

DMAリマッピング機能がハードウェア上に存在することをOSに伝えるため、ACPIはDMARと呼ばれるテーブルを用意しています。DMARでは、いくつかの異なる種類の情報が列挙されています。IOMMUはDMA Remapping Hardware Unit Definition（DRHD）という名前の構造体で記述されており、他にDMAに用いることのできない予約済みメモリ領域を示すReserved Memory Region Reporting（RMRR）などが存在します。DRHDはIOMMUのレジスタベースアドレスと、IOMMUがDMAリマッピング対象にしているPCIデバイスのリストを持ちます。

VT-dの設定をOS上から簡単に確認することは難しいですが、ACPIテーブルは簡単に見ることができるので、ここでその方法を説明します。例としてUbuntu LinuxでDMARを表示するコマンドを**画面1**に示します。

Hardware Unit Definitionと表示されているのがDRHDで、Reserved Memory Regionと表示されているのがRMRRです。

このテーブルの情報が誤っていると、BIOSとカーネルでVT-dを有効にしてもLinuxカーネルがエラーを起こしてPCIパススルーが正常に動作しない場合があります[注4]。

VT-d のレジスタ

VT-dで使用される主なレジスタを**表3**に示します。VT-dのレジスタはメモリマップドでアクセスでき、ベースアドレスは前述のDMAR上のDRHDで通知されます。

VT-d の有効化

VT-dを有効化し、DMAリマップを行うには次のような手順で設定を行います。

設定完了までウエイトするために、global status registerのtranslation enable statusにビットが立つまでループします。

① メモリ上にルートテーブル、コンテキストテーブルを作成

② root table address registerにroot tableのアドレスを設定し、global command registerにset root table pointerをセット（**表4**）してroot tableのアドレスを設定します。設定完了までウエイトするために、global status registerのroot table pointer statusにビットが立つまで（**表5**）ループします

注3） Tagged TLBの場合。

注4） この場合、ユーザの設定ミスではなくBIOSのバグなので、対策としてはサーバベンダからBIOSアップデートを受け取るか、カーネル側でDMARを無視して強引に初期化するような方法しかありません。

▼画面1　DMAR表示コマンド（Ubuntu Linux）

```
$ sudo apt-get install iasl
$ sudo cp /sys/firmware/acpi/tables/DMAR .
$ sudo iasl -d DMAR

Intel ACPI Component Architecture
AML Disassembler version 20100528 [Dec 19 2012]
Copyright (c) 2000 - 2010 Intel Corporation
Supports ACPI Specification Revision 4.0a

Loading Acpi table from file DMAR
Acpi Data Table [DMAR] decoded, written to "DMAR.dsl"
syuu@hiratake:~$ cat DMAR.dsl
/*
 * Intel ACPI Component Architecture
 * AML Disassembler version 20100528
 *
 * Disassembly of DMAR, Mon Nov 25 09:11:42 2013
 *
 * ACPI Data Table [DMAR]
 *
 * Format: [HexOffset DecimalOffset ByteLength]  FieldName : FieldValue
 */

[000h 0000  4]                    Signature : "DMAR"    /* DMA Remapping table */
[004h 0004  4]                 Table Length : 00000130
[008h 0008  1]                     Revision : 01
[009h 0009  1]                     Checksum : 22
[00Ah 0010  6]                       Oem ID : "AMI"
[010h 0016  8]                 Oem Table ID : "OEMDMAR"
[018h 0024  4]                 Oem Revision : 00000001
[01Ch 0028  4]              Asl Compiler ID : "MSFT"
[020h 0032  4]        Asl Compiler Revision : 00000097

[024h 0036  1]           Host Address Width : 26
[025h 0037  1]                        Flags : 01

[030h 0048  2]                Subtable Type : 0000 <Hardware Unit Definition>
[032h 0050  2]                       Length : 0020
[034h 0052  1]                        Flags : 01
[035h 0053  1]                     Reserved : 00
[036h 0054  2]          PCI Segment Number : 0000
[038h 0056  8]        Register Base Address : 00000000FBFFE000

[040h 0064  1]      Device Scope Entry Type : 03
[041h 0065  1]                 Entry Length : 08
[042h 0066  2]                     Reserved : 0000
[044h 0068  1]               Enumeration ID : 06
[045h 0069  1]               PCI Bus Number : F0
[046h 0070  2]                     PCI Path : [1F, 07]

[048h 0072  1]      Device Scope Entry Type : 03
[049h 0073  1]                 Entry Length : 08
[04Ah 0074  2]                     Reserved : 0000
[04Ch 0076  1]               Enumeration ID : 07
[04Dh 0077  1]               PCI Bus Number : 00
[04Eh 0078  2]                     PCI Path : [13, 00]

[050h 0080  2]                Subtable Type : 0001 <Reserved Memory Region>
[052h 0082  2]                       Length : 0058
[054h 0084  2]                     Reserved : 0000
[056h 0086  2]          PCI Segment Number : 0000
[058h 0088  8]                Base Address : 00000000000EC000
[060h 0096  8]          End Address (limit) : 00000000000EFFFF
…… (省略) ……
```

③IOTLB、Context-cacheをinvalidateします（細かい手順は省略します）
④global command registerにtranslation enableをセットしてDMAリマッピングを有効化します

まとめ

今回は、VT-dの詳細について解説しました。次回からは、よりソフトウェア寄りの視点から仮想化を解説していきたいと思います。SD

▼表3　VT-dの主なレジスタ

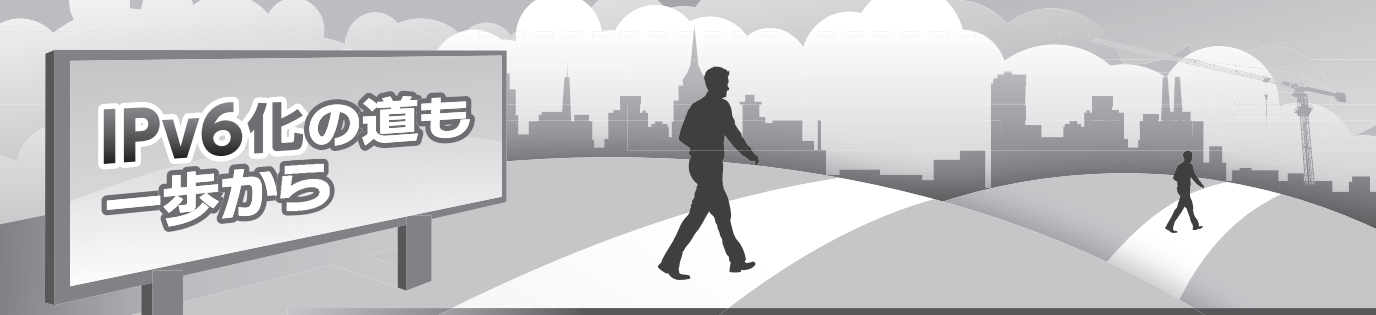
offset	name	description
008h	capability register	VT-d上の機能の有無を示す
010h	extended capability register	追加のcapabilityレジスタ
018h	global command register	VT-dの操作を行う
01ch	global status register	VT-dのステートを通知
020h	root table address register	Root tableのアドレスを設定
028h	context command register	context-cacheを操作
034h	fault status register	フォールトを通知
XXXh	invalidate address register	IOTLB invalidate時のアドレス指定 invalidate address registerのアドレスはextended capability registerで定義
XXXh+008h	IOTLB invalidate register	IOTLB invalidateを実行

▼表4　global command register

bits	field	description
31	translation enable	DMA remapping有効・無効化
30	set root table pointer	root table pointerのセット・アップデート
29	set fault log	fault log pointerのセット・アップデート
28	enable advanced fault logging	advanced fault logging有効・無効化
27	write buffer flush	write bufferをflush
26	queued invalidation enable	queue invalidation有効・無効化
25	interrupt remapping enable	interrupt remaping有効・無効化
24	set interrupt remap table pointer	interrupt remap table pointerのセット・アップデート
23	compatibility format interrupt	compatibility format interrupt有効・無効化
22:00	reserved	予約フィールド

▼表5　global status register

bits	field	description
31	translation enable status	DMA remappingの状態
30	root table pointer status	root table pointerの状態
29	fault log status	fault log pointerの状態
28	advanced fault logging status	advanced fault loggingの状態
27	write buffer flush status	write buffer flushの状態
26	queued invalidation enable status	queue invalidation enableの状態
25	interrupt remapping enable status	interrupt remapingの状態
24	interrupt remap table pointer status	interrupt remap table pointerの状態
23	compatibility format interrupt status	compatibility format interruptの状態
22:00	reserved	予約フィールド



最終回

IPv6で外部接続した場合の動作確認方法

IPv6普及・高度化推進協議会　IPv4/IPv6共存WG　アプリケーションのIPv6対応検討SWG
廣海 緑里 HIROMI Ruri　渡辺 露文 WATANABE Tsuyufumi　新 善文 ATARASHI Yoshifumi　藤崎 智宏 FUJISAKI Tomohiro

外部からのWebサーバの接続確認

前回(本誌2013年10月号)までで、外部に公開されていないテスト環境の構築を題材に、IPv4との違い、Webサイトの構築、セキュリティの考え方など一通り網羅しました。今回は、Webサーバの外部接続でIPv6が有効化され、グローバルな接続性が確保された際の動作確認方法から説明します。

3種類の確認方法を紹介します。1番目はWebブラウザを利用する方法。2番目は外部のUNIX系OS搭載のPCを利用する方法。3番目は外部ネットワークのルータを利用する方法です。ここでは、金沢星稜大学のWebサイトを例に確認します。

Webブラウザによる動作確認

Webサーバへのアクセスの実際の手順は、次のようになります。

(1)URL中のFQDNから名前解決を行う
(2)(1)の結果として接続先のAレコード(IPv4アドレス)とAAAAレコード(IPv6アドレス)のリストを得る
(3)リストの順番に接続を試みる
(4)応答のあった接続先とデータ交換を行う

ここでクライアントOSの標準的な使い方をしている場合、IPv6が優先処理されるようになっているため、AAAAレコードから処理が開始されます。しかし、まだIPv6普及期にある状況を加味して、Aの宛先とAAAAの宛先に同時に接続を試みるブラウザの実装や、通信のデフォルトの優先順位以外にIPv6、IPv4での通信が成り立つかも組み合わせて接続先を決定する実装もあります。そこで正確にIPv6での通信性を確認するには、名前解決処理をしないようにIPv6アドレスを直接入力します。ほかの多くのブラウザでもIPv6アドレスを直接入力することに対応しています。直接指定する場合は、「`http://[2001:db8::1]`」のように、ブラケット(`[]`)でIPv6アドレスを囲みます。

ここではさらに、やりとりするコンテンツごとの詳細な確認ができるように、ブラウザのプラグインを活用した実例を紹介します。利用したブラウザはFirefoxで、プラグインはFirebugです。FirefoxはMac版のバージョン24.0(確認当時の最新版)を用いています。筆者は同じサイトがIPv6対応した直後(およそ1年前)にもアクセスしていますが、当時のバージョンは13.0(Windows版)でした。FireFoxやFirebugのIPv6対応はずっと前から行われていたことがわかります。

プラグインのFirebugは、WebページのCSS、HTML、及びJavaScriptをリアルタイムに編集、デバッグ、モニタするための開発ツールですが、モニタ機能ではネットワークの状況確認もできて便利です。**図1**のキャプチャ画像では、このネットワークのモニタ機能を利用しています。

UNIX系OSによる動作確認

UNIX系OS搭載のPCを利用すると、ブラウザを利用する場合よりももっと詳細な確認ができます。今回は、CentOS 5を用いました。確認したい内容に対応するコマンドを次に記します。

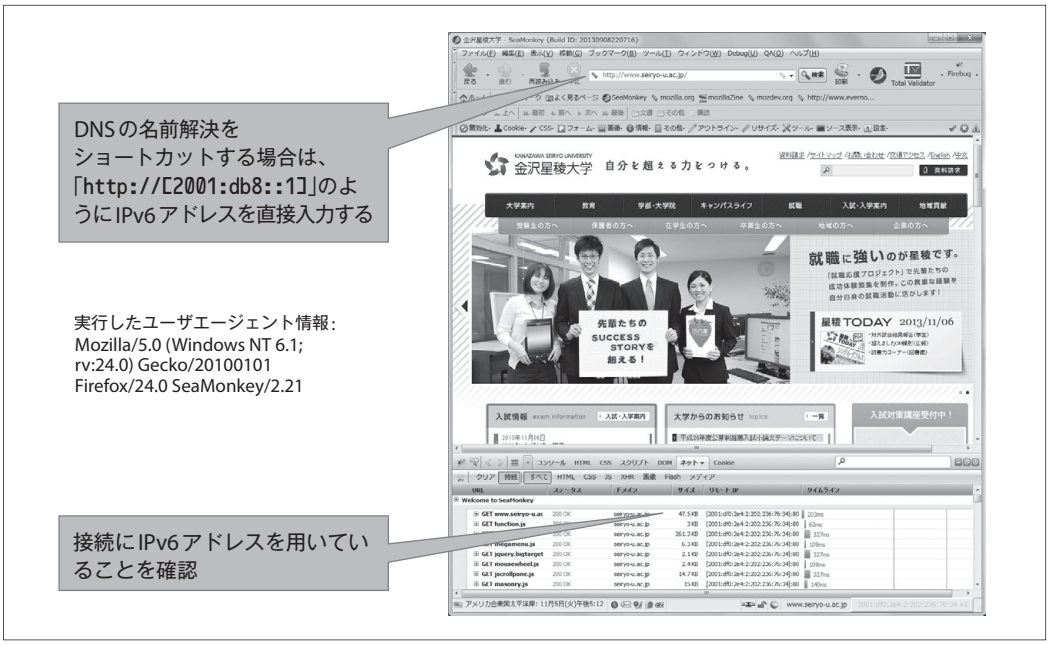
①ネットワーク到達性：ping6、traceroute6、tracepath6（ipコマンドでも同様の結果が得られます）
②サービス到達性：nc
③名前解決：dig、host
④HTTPレスポンス：curl

Webブラウザで確認したときと同じWebサイトに対してコマンドを実行した結果を、いくつか紹介します。

図2はtracepath6コマンドの実行結果です。tracerouteの結果とPath MTUの値も同時に確認できます。**図2**の例ではPath MTUは1500バイトということが確認できます。

図3はhostコマンドの実行結果です。「あるFQDNに対するAレコード（IPv4アドレス）と

▼図1　Webブラウザでの確認方法



▼図2　tracepath6の実行例

```
$ tracepath6 2001:df0:2e4:2:202:236:76:35
 1?: [LOCALHOST]                      pmtu 1500
 1:  uranus-g.inetcore.com                      5. 44ms
 2:  2001:278:0:222e::1                       170.172ms
 3:  2001:278:0:2020::34                        4. 48ms
 4:  2001:278:0:2010::23                        4.870ms
 5:  2001:278:0:222d::2                       asymm  6   2.549ms
 6:  tokyo-dc-rm-ae3-vlan10.s4.sinet.ad.jp    asymm  7  10.598ms
 7:  kanazawa-dc-rm-xe0-0-1-vlan10.s4.sinet.ad.jp asymm  8  47. 39ms
 8:  2001:2f8:ff00:66::2                      asymm 11  21.704ms
 9:  2001:df0:2e4:1:202:236:76:1              asymm 12  22.974ms
10:  no reply
11:  no reply
12:  no reply
```

AAAAレコード(IPv6アドレス)がとにかくわかればいい」という場合は、hostコマンドを実行してみてください。指定したFQDNを管理するネームサーバのAとAAAAレコードの応答結果を表示してくれます。

さらに詳しい状況を知りたい場合には、digコマンドを使ってみてください。**図4**では、上位ネームサーバに登録する前に、立ち上げたDNSサーバが正しく応答を返しているかを確認するという状況で、「@"DNSサーバのIPv6アドレス"」を付与しています。

図5はcurlコマンドの実行結果です。ブラウザを使わずに、Webサーバのコンテンツの取得が可能かどうか確認できます。「-g」オプションを指定して、コンテンツを取得した出力結果です。

同じくcurlコマンドでブラウザを使わずにコンテンツの取得を確認します(**図6**)。こちらの例は、「-g」オプションに「--trace-ascii」を追加して、HTTPのプロトコルをトレースさせた結果です。単純にコンテンツ確認をするだけではなく、プロトコル全体のやりとり状況を確認したい場合に便利です。

サービス到達性の確認コマンドで紹介したncコマンドは、ターゲットホストのポート80を指定することで応答があるかどうか確認することができます。オプションの利用やパイプラインで入力を与えることでサービスに対応したやりとりが可能なため、HTTP以外のポートについての簡単な状況確認もncコマンド1つでできるので便利です。実行結果は省略しますが、次の

▼図3　hostの実行例

```
$ host www01.seiryo-u.ac.jp
www01.seiryo-u.ac.jp has address 202.236.76.34
www01.seiryo-u.ac.jp has IPv6 address 2001:df0:2e4:2:202:236:76:34
```

▼図4　digの実行例

```
$ dig @2001:df0:2e4:2:202:236:76:35 AAAA www01.seiryo-u.ac.jp.

; <<>> DiG 9.3.6-P1-RedHat-9.3.6-16.P1.el5 <<>> @2001:df0:2e4:2:202:236:76:35 AAAA www01.
seiryo-u.ac.jp.
; (1 server found)
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2206
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;www01.seiryo-u.ac.jp.          IN      AAAA

;; ANSWER SECTION:
www01.seiryo-u.ac.jp.   14400   IN      AAAA    2001:df0:2e4:2:202:236:76:34

;; AUTHORITY SECTION:
seiryo-u.ac.jp.         14400   IN      NS      keidai.seiryo-u.ac.jp.
seiryo-u.ac.jp.         14400   IN      NS      star88.seiryo.jp.
seiryo-u.ac.jp.         14400   IN      NS      sigw.sinet.ad.jp.

;; ADDITIONAL SECTION:
keidai.seiryo-u.ac.jp.  14400   IN      A       202.236.76.35
keidai.seiryo-u.ac.jp.  14400   IN      AAAA    2001:df0:2e4:2:202:236:76:35
star88.seiryo.jp.       14400   IN      A       202.236.76.88

;; Query time: 29 msec
;; SERVER: 2001:df0:2e4:2:202:236:76:35#53(2001:df0:2e4:2:202:236:76:35)
;; WHEN: Wed Nov  6 10:19:26 2013
;; MSG SIZE  rcvd: 203
```

▼図5　curlの実行例（その1）

```
$curl -g "http://[2001:df0:2e4:2:202:236:76:34]:80"
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="ja" xml:lang="ja">
<!--メインイメージ-->
<!--ヘッダーリンク-->
<!--経済学部-->
<!--人間科学部-->
（中略）
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Style-Type" content="text/css" />
<meta name="description" content="" />
<meta name="keywords" content="自分を超える　金沢星稜大学　星稜大　星稜　seiryo" />
<style type="text/css" media="all">
        @import url('common.css');
        @import url('top.css');
</style>
<script type="text/javascript" src="function.js" charset="utf-8"></script>
<script type="text/javascript" src="jquery.js" charset="utf-8"></script>
```

▼図6　curlの実行例（その2）

```
$curl --trace-ascii trace.txt -g "http://[2001:df0:2e4:2:202:236:76:34]:80"
== Info: About to connect() to 2001:df0:2e4:2:202:236:76:34 port 80
== Info:   Trying 2001:df0:2e4:2:202:236:76:34...
== Info: connected
== Info: Connected to 2001:df0:2e4:2:202:236:76:34 (2001:df0:2e4:2:202:236:76:34) port 80
=> Send header, 172 bytes (0xac)
0000: GET / HTTP/1.1
0010: User-Agent: curl/7.15.5 (i686-redhat-linux-gnu) libcurl/7.15.5 O
0050: penSSL/0.9.8b zlib/1.2.3 libidn/0.6.50077: Host: [2001:df0:2e4:2:202:236:76:34]009d:
Accept: */*
00aa:
<= Recv header, 17 bytes (0x11)0000: HTTP/1.1 200 OK
<= Recv header, 37 bytes (0x25)0000: Date: Tue, 21 Aug 2012 02:54:09 GMT
<= Recv header, 16 bytes (0x10)0000: Server: Apache
<= Recv header, 46 bytes (0x2e)0000: Last-Modified: Fri, 17 Aug 2012 05:03:12 GMT
<= Recv header, 35 bytes (0x23)0000: ETag: "161fec-646e-4c76f13b43488“
<= Recv header, 22 bytes (0x16)0000: Accept-Ranges: bytes
<= Recv header, 23 bytes (0x17)0000: Content-Length: 25710
<= Recv header, 25 bytes (0x19)0000: Content-Type: text/html
<= Recv header, 22 bytes (0x16)0000: Content-Language: ja
<= Recv data, 1183 bytes (0x49f)0000:
:
:
<br />Copyright &050a: copy; Kanazawa Seiryo University. All Rights Reserved.</p>0546:
</div><!-- /footer -->055e: 0560: 0562: </div><!-- /wrapper -->057b: 057d: <script
language="JavaScript" type="text/javascript" src="//o.ad05bd: vg.jp/ojs?aid=6241&pid=1"
charset="UTF-8">05e9: </script>05f4: <noscript>0600: <iframe src="//o.advg.jp/
oif?aid=6241&pid=1" width="1" height="10640: ">0644: </iframe>064f: </noscript><!-- /
disco_student_targeting -->067d: 067f: </body>0688: </html>
== Info: Connection #0 to host 2001:df0:2e4:2:202:236:76:34 left intact
== Info: Closing connection #0
```

ような形式で利用できます。

```
nc -z 2001:df0:2e4:2:202:236:76:35 80

echo "QUIT" | nc
2001:df0:2e4:2:202:236:76:35 80
```

外部ネットワークからの経路／到達性の確認

外部ネットワークからの経路／到達性を確認するには、Looking Glass[注1]を利用するのが便利です。多くのISPでLooking GlassのWebページが提供され、ルータの情報を公開しています。なお、こうしたLooking Glassは、pingやtracerouteなどを実行できるものがあります。ルータの操作に縁がないと躊躇された人も、pingやtracerouteで到達性が確認できるのは便利なのではないでしょうか。サーバの状況確認に加えて、海外サイトからの到達性まで確認できれば、安心してIPv6対応のWebサーバのカットオーバーを迎えられるでしょう。あとは、IPv6クライアントからのアクセスを待つばかりです。

注1）遠隔から経路情報を確認するために設置される経路情報表示サービス。特定のサーバやPCに対してpingやtracerouteなどのコマンドを実行し、その出力結果をブラウザ上で確認できる。

最近のIPv6トピック

総括として、「もっとIPv6対応を前向きに考えてみよう」と思えるようなトピックスを挙げたいと思います。

今年10月7日に、インターネット関連の10団体(IAB、ICANN、IETF、ISOC、W3Cと5つの地域インターネットレジストリ(AFRINIC、ARIN、APNIC、LACNIC、RIPE NCC))が「今後のインターネット協力体制に関するモンテビデオ声明」を出しました。この声明では、4つの事項に対する取り組みが表明されています。4つの事項とは、(1)グローバルに調和の取れたインターネット運営、(2)インターネットガバ

COLUMN DNSの役割とあり方に注意!

コマンドの実行例で紹介したように、通信先のホスト名からIP(v4/v6)アドレスを知るには名前解決を行います。グローバルに名前解決を行うためにはDNSサーバに設定し、全世界に情報を公開する必要があります。ここで最近、耳にするのは、(1)名前解決時のDNSフォールバックに時間がかかる、(2)IPv6とIPv4を管理しているネームサーバが違うことによる弊害、などの問題です。

(1)は、ネームサーバをIPv6対応させる(NSレコードを参照するとAレコードもAAAAレコードも紐付けられている)と、ネームサーバへのアクセスには従来のIPv4のトランスポートを使う場合に加えて、IPv6トランスポートが使われる場合が出てきます。ネームサーバの設定やサーバのインターフェースには問題がないのに、ネットワーク運用側でIPv6でのアクセスを禁じているケースがありました。その場合、IPv6では応答がないことで、IPv4トランスポートを使うようにIPv6からIPv4へのフォールバックが発生すると名前解決に時間がかかるようになります。サーバとネットワークで担当者が違っていて、IPv6対応についての情報共有がうまくできていないと、このような状況に陥ることがあるようです。また、途中のセキュリティ製品はIPv4前提で、IPv6についてはすべて拒否という設定である場合も同じ状況になります。一気通貫で、確認／対処が必要な事例と言えます。

(2)は、IPv4で構築したネームサーバとは別にIPv6用にネームサーバをたてたものの、ゾーン管理もばらばらになり、どのネームサーバが応答するかで得られる結果が異なり、さらには回答が得られない場合もあり、安定した接続性が確保できなくなっているサイトがあったそうです。

いずれも、既存のIPv4環境を意識したまでは良かったのでしょうが、IPv4と分離した構成にしようと囚われ過ぎていた結果が招いた事故と思われます。IPv6ではIPv4のこの設定はいらない(あるいは必要)という原点のところをふまえた設定を行う必要があります。適切にIPv6を利用すべく、設定／運用には注意したいものです。

ナンス、(3)ICANNとIANA機能のグローバル化、(4)IPv6移行です。

最後に挙がっているIPv6移行は最優先課題とされ、とくにインターネットのコンテンツ事業者に対応を呼びかけています。この声明の日本語訳は、JPNICのWebサイト[注2]で公開されています。

こうした取り組みは地味ながらも確実に浸透していくでしょうから、コンテンツ事業に関係する人は、今後IPv6が仕様化されていくと見ておいたほうが良いでしょう。実際、ITS(Intelligent Transport Systems:高度道路交通システム)、M2M(Machine to Machine)[注3]やIoT(Internet of Things)[注4]といった少し先を見ているような技術分野では、IPv6への期待が大きいです。このようなモノをつなぐ技術が普及期を迎えたときのコンテンツ提供は、従来とは違った新しいビジネス分野の展開もあるかもしれません。「我こそは」と思われた方は今から備えておくのも良いのではないでしょうか。

最近および来年のイベント情報

昨年もお伝えしたInternetWeekですが、今年も実施されました。本号が出版されたころは開催後の資料公開などがされている時期かもしれません。InternetWeekのWebサイトにアクセスして、どのような話題がホットだったのかチェックしてみてください。

また、今年度はインターネット協会が主催する「IPv6サミット」がいくつかの主要都市で開催されています。また、広島県の広島地域IPv6推進委員会では毎年定期的にIPv6に関するセミナーを開催し、最新動向の共有がはかられていますが、今年はハンズオンセミナーも手掛けています。中国地方の方は、過去のトピックスを含めてご覧になられてみてはいかがでしょうか。

InternetWeek

2013年は以下の日程で行われました。

- 【日程】2013年11月26日(火)～11月29日(金)
- 【場所】富士ソフトアキバプラザ(秋葉原)
- 【テーマ】荒ぶるインターネットを乗りこなす ～Ride and break the wild Internet～

IPv6の最新状況(統計情報やサービス動向ほか)、セキュリティ、基礎知識など幅広いトピックが並びました。同時開催イベントとして、「IPv6 Summit 2013 in Tokyo」と「第25回JPNICオープンポリシーミーティング」も開催されました。詳細は、IW2013公式サイト[注5]をご覧ください。

IPv6 Summit

InternetWeekと同時開催されたIPv6 Summitは、京都など全国の主要都市でも開催されています。これから実施されるものは、次のとおり。

- 【イベント名】IPv6 Summit in Sapporo
- 【日程】2014年2月1日(土)
- 【場所】札幌市生涯学習センター「ちえりあ」

詳細は、インターネット協会IPv6デプロイメント委員会の「IPv6サミット」のWebサイト[注6]でご確認ください。

広島地域IPv6推進委員会セミナーとハンズオンセミナー

コンスタントにIPv6に関する情報発信を行っている広島地域IPv6推進委員会では、今年は「IPv6ハンズオンワークショップ ～サーバのIPv6運用におけるノウハウを学ぶ～」と題して、いつものフォーラムスタイルではなく機材を使ったハンズオンセミナーを実施しています。

- 広島地域IPv6推進委員会
  http://www.supercsi.jp/ipv6deploy/

注2) https://www.nic.ad.jp/ja/topics/2013/20131008-01.html
注3) 機械同士が通信ネットワークを使って互いに情報をやり取りすることで、自律的に高度な制御や動作を行うこと。
注4) IT機器以外のさまざまなモノがインターネットに接続され、相互作用するしくみ／概念。
注5) https://internetweek.jp/
注6) http://www.iajapan.org/ipv6/summit/index.html

最終回にあたって

「IPv6化の道も一歩から」と題して、IPv6導入の基礎知識を広く浅く取り上げてきた本連載も今回で最終回となりました。これまで、IPv6を導入していく際に考えておきたいことを提示してきました。まだまだ書き足りないこともたくさんありますが、考えるきっかけ作りとしてお役に立てたなら幸いです。

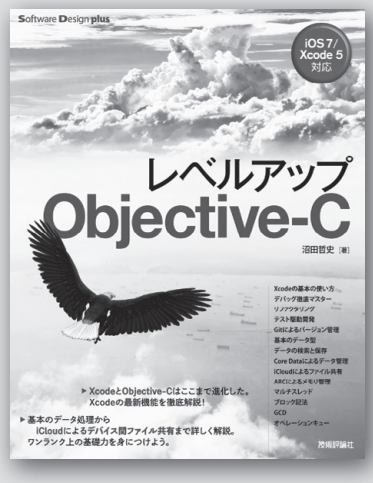
この連載が始まったのは、2012年12月号でした。当初、6回の予定でスタートしましたが、気がつけば10回を数えます。筆者らは、この連載を続ける傍ら、「アプリケーションのIPv6対応ガイド」(IPv6普及・高度化推進協議会著)を公開することができました。

続けてWebアプリケーション編を年内に公開したいところでしたが、気がつくと2013年も終わりを迎えております。Webアプリケーションについては、クライアント端末の進化と変遷が速いうえに、HTML5/WebSocket/認証連携(フェデレーション)……といったアプリケーションの基盤技術も日々進歩しています。その輪の中に、それほど難しくなくIPv6を巻き込んで、より身近なモノ同士を簡単にコミュニケートさせたり、インターネットの新しい使い方が生まれたりすることを期待しています。そのためにも、筆者らはWebアプリケーション編を早くまとめて、パブリックコメントを募集し、国内のWebアプリケーション開発に携わる方のご意見をいただければと思います。

およそ1年に渡り読んでいただき、ありがとうございました。IPv6利用が本格化するのはまだこれからですが、サービスや設備の更改時にはIPv6を意識すべき時期にきています。筆者が所属しているIPv6普及・高度化推進協議会[注7]では、新たに企業ネットワークへの導入に関するWG(ワーキンググループ)を立ち上げ、今後も各種の情報提供やイベント支援を行っていきます。IPv6に関する最新情報をご覧になりたい場合には、「そういう団体があったな」と記憶の片隅に置いといていただき、そっとアクセスしていただけたらと思います。SD

注7) http://www.v6pc.jp/jp/index.phtml

Software Design plus 技術評論社



iOS 7/ Xcode 5 対応

レベルアップ Objective-C

本書は、初心者から中級者へとレベルアップすることを目標としています。Objective-Cを学んでいく上でMac OS XやiOSでの実用的なコード例を示しながらXcode 5を効果的に利用するための技術解説を行います。Xcode 5ではユニットテストやGitなど、現代的な開発環境として必須とされる機能がサポートされています。
本書では、Xcode 5でのデバッグ、リファクタリング、バージョン管理をはじめ、最新のObjective-Cを利用したデータ処理、Core Data、iCloud、メモリ管理、マルチスレッドなどを解説しています。

沼田哲史 著
B5変形判／360ページ
定価3,360円(本体3,200円)
ISBN 978-4-7741-6076-4

大好評発売中!

こんな方におすすめ
・Xcodeを使ってiOSまたはOS X用の簡単なアプリケーションをいくつか書いたことのある、Objective-Cプログラミング中級者以上の方
・最新のiOS 7とXcode 5の魅力的な機能を知りたい方

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第3回 ◇ top(1)その2 ～スレッドの優先順位やI/O回数から見えるもの～

◉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。

前回に引き続き

システムの状態をモニタリングするためのコマンドの1つにtop(1)があります。前回はtop(1)コマンドの出力においてシステムの状態を表しているヘッダ部分の読み方を紹介しました。今回はリスト部分を取り上げます。なかでも普段は注目されることがないものの、動作状態を知るうえで大切になってくる項目をピックアップして紹介します。

カーネルプロセスを調べる

ユーザが実行するソフトウェアが動作できるように、オペレーティングシステムではカーネルと呼ばれるソフトウェアがお膳立てをしています。お膳立てというのは、割り込みの処理、ディスクI/Oの処理、ネットワークインターフェースI/Oの処理、ファイルシステムの処理、スワップの処理、プロセスのスケジューリングなどさまざまな対象におよぶのですが、こうした処理をしているプロセスをユーザが目にすることはほとんどありません。こうした働きをするプロセスは「カーネルプロセス」と呼ばれています。

カーネルプロセスはカーネル空間の中で動作します。こうしたプロセスはtop(1)コマンドではデフォルトでは表示されません。top(1)コマンドに**-S**オプションを指定することで、カーネルプロセスをプロセス一覧に表示するようになります。**top -S -U root -a**のようにプロセスの種類をrootユーザが実行しているものに限定しつつ、**-a**で出力を整理するとよりわかりやすくなります（**図1**）。

プロセス名が"["からはじまっているプロセスがカーネルプロセスです。名前を見ればどのカーネルプロセスがどういった機能を担当しているか推測できます。このようにカーネルの機能もそれぞれがプロセスとして実行されており、こうしたプロセスの相互作用でシステムは動作しているということが見て取れると思います。

カーネルスレッドを調べる

FreeBSD 5以降、動作の主体は「プロセス」ではなく「スレッド」になっています。top(1)コマンドでデフォルトで表示されるのはプロセスですが、これは動作しているスレッドの平均値ということになります。スレッドそのものを表示させるにはtop(1)コマンドに**-H**オプションを指定します。たとえば、**top -S -U root -H -a**のように実行すれば**図2**のような出力が得られます。

オプション**-H**を指定しなかったときには単一のプロセスに見えていたカーネルプロセスも、実際には複数のスレッドが動作していることがわかります。カーネルプロセスのスレッドだけに限定して抜き出すと**図3**のようになります。

この出力からはアイドル（**[idle]**）、スワッパー（**[kernel{swapper}]**）、em0のタスクキュー（**[kernel{em0 taskq}]**）、スレッドのタスクキュー（**[kernel{thread taskq}]**）、割り込み

各コマンドのうしろについている括弧書きの数字は、manコマンドで見ることができるマニュアルに記載されている章番号を表しています。



（[intr]）、GEOMの上がりと下りの処理を担当するカーネルプロセス（[geom]）が複数のスレッドで動作していることを確認できます。アイドルを担当するカーネルプロセスはプロセッサコアの数だけスレッドが生成され、実行されています。



スケジューリングプライオリティって何?

カーネルプロセスを含むすべてのプロセスには「スケジューリングプライオリティ」の値が設定されています。プロセスはこの優先度にしたがって処理されます。この値はマイナス値からプラス値まで設定することができ、小さければ小さいほど（マイナス値）高い優先度を持っています。たとえば**top -S -U root -a -o pri**のようにプライオリティの値で整理した状態でプロセス（カーネルプロセスを含む）の一覧を表示させ、スケジューリングプライオリティの値に関連する部分だけ抜き出すと**図4**のようになります。

リストの上のほうが優先順位が低く、下のほうが

▼**図1　top -S -U root -aの実行例**

```
  PID USERNAME    THR PRI NICE   SIZE    RES STATE   C   TIME    WCPU COMMAND
   11 root          2 155 ki31     0K    32K RUN     1 224.9H  9.91% [idle]
    0 root         11 -52    0     0K   176K -       1  20:28  0.00% [kernel]
   12 root         14 -76    -     0K   224K WAIT    0  17:24  0.00% [intr]
    8 root          1  16    -     0K    16K syncer  1   4:43  0.00% [syncer]
   13 root          3  -8    -     0K    48K -       1   3:58  0.00% [geom]
   14 root          1 -16    -     0K    16K -       0   1:51  0.00% [rand_harvestq
  647 root          1  20    0 25328K   984K select  1   0:24  0.00% /usr/sbin/ntpd
    4 root          1 -16    -     0K    16K psleep  0   0:13  0.00% [pagedaemon]
   16 root          1  20    -     0K    16K sdflus  0   0:10  0.00% [softdepflush]
    7 root          1  20    -     0K    16K psleep  1   0:09  0.00% [bufdaemon]
  723 root          1  20    0 23980K  1564K select  1   0:09  0.00% sendmail: acce
    5 root          1 -16    -     0K    16K psleep  0   0:07  0.00% [vmdaemon]
    9 root          1  20    -     0K    16K vlruwt  0   0:05  0.00% [vnlru]
   15 root          4 -68    -     0K    64K -       0   0:05  0.00% [usb]
 9318 root          1  52    0 39044K 34352K select  1   0:02  0.00% /usr/obj/usr/s
  730 root          1  20    0 16520K     0K nanslp  0   0:02  0.00% /usr/sbin/cron
82945 root          1  20    0  9084K   808K select  0   0:02  0.00% make -j2 build
  565 root          1  20    0 14424K   652K select  1   0:01  0.00% /usr/sbin/sysl
82951 root          1  20    0  6276K  1132K select  1   0:01  0.00% /usr/obj/usr/s
 9330 root          1  20    0  6276K  1052K select  1   0:00  0.00% /usr/obj/usr/s
32625 root          1  72    0 34052K 19608K CPU0    0   0:00  0.00% [cc]
    1 root          1  20    0  9432K   124K wait    0   0:00  0.00% [init]
 9335 root          1  52    0 16988K   608K wait    0   0:00  0.00% sh -ev
```

▼**図2　top -S -U root -H -aの実行例**

```
  PID USERNAME   PRI NICE   SIZE    RES STATE   C   TIME    WCPU COMMAND
   11 root       155 ki31     0K    32K RUN     0 112.3H 28.96% [idle{idle: cpu0}]
   11 root       155 ki31     0K    32K RUN     1 112.6H 23.97% [idle{idle: cpu1}]
   12 root       -88    -     0K   224K WAIT    0   3:38  1.95% [intr{irq14: ata0}]
   13 root        -8    -     0K    48K -       0   2:30  0.98% [geom{g_down}]
   13 root        -8    -     0K    48K RUN     0   1:29  0.98% [geom{g_up}]
    0 root       -16    0     0K   176K swapin  0    ???  0.00% [kernel{swapper}]
    0 root       -92    0     0K   176K -       0  20:27  0.00% [kernel{em0 taskq}]
   12 root       -60    -     0K   224K WAIT    1   7:44  0.00% [intr{swi4: clock}]
   12 root       -92    -     0K   224K WAIT    1   5:28  0.00% [intr{irq11: em0 uh
    8 root        16    -     0K    16K syncer  0   4:43  0.00% [syncer]
   14 root       -16    -     0K    16K -       1   1:51  0.00% [rand_harvestq]
   12 root       -88    -     0K   224K WAIT    1   0:36  0.00% [intr{irq15: ata1}]
  647 root        20    0 25328K   984K select  0   0:24  0.00% /usr/sbin/ntpd -c /
    4 root       -16    -     0K    16K psleep  0   0:13  0.00% [pagedaemon]
   16 root        20    -     0K    16K sdflus  0   0:10  0.00% [softdepflush]
    7 root        20    -     0K    16K psleep  0   0:09  0.00% [bufdaemon]
  723 root        20    0 23980K  1564K select  0   0:09  0.00% sendmail: accepting
    5 root       -16    -     0K    16K psleep  1   0:07  0.00% [vmdaemon]
    9 root        20    -     0K    16K vlruwt  0   0:05  0.00% [vnlru]
   15 root       -68    -     0K    64K -       0   0:05  0.00% [usb{usbus0}]
 9318 root        39    0 43140K 35460K select  0   0:02  0.00% /usr/obj/usr/src/ma
  730 root        20    0 16520K     0K nanslp  1   0:02  0.00% /usr/sbin/cron -s (
82945 root        20    0  9084K   808K select  1   0:02  0.00% make -j2 buildkerne
```

優先順位が高くなります。割り込みに関するカーネルプロセスのスレッドに高い優先度が割り当てられており、これに続く形でスワッパー、ページデーモン、バッファデーモン、仮想マシンデーモンなどのカーネルプロセスのスレッドに高い優先度が割り当てられています。ユーザ空間で動作するプロセスには20から52の間で優先度が割り当てられ、アイドルカーネルプロセスにはもっとも低い優先度が割り当てられています。

つまり、何もやることがない状態になると最終的にもっとも優先順位の低いアイドルプロセスのスレッド（**[idle{idle: cpu0}]**など）に処理が渡ってくることになります。このアイドルプロセスの実行時間を集計すると、**システムが遊んでいる時間**を計測することができます。逆にカーネルとして処理すべきものには高い優先順位が与えられ、ユーザ空

▼図3　top -S -U root -H -aの出力からカーネルプロセスのみを抜き出した例

```
  PID USERNAME   PRI NICE   SIZE    RES STATE   C   TIME    WCPU COMMAND
   11 root       155 ki31     0K    32K RUN     1 112.6H 100.00% [idle{idle: cpu1}]
   11 root       155 ki31     0K    32K CPU0    0 112.3H 100.00% [idle{idle: cpu0}]
    0 root       -16    0     0K   176K swapin  1    ???   0.00% [kernel{swapper}]
    0 root       -92    0     0K   176K -       0  20:27   0.00% [kernel{em0 taskq}]
   12 root       -60    -     0K   224K WAIT    0   7:45   0.00% [intr{swi4: clock}]
   12 root       -92    -     0K   224K WAIT    1   5:28   0.00% [intr{irq11: em0 uhci0}]
    8 root        16    -     0K    16K syncer  0   4:43   0.00% [syncer]
   12 root       -88    -     0K   224K WAIT    0   3:40   0.00% [intr{irq14: ata0}]
   13 root        -8    -     0K    48K -       0   2:31   0.00% [geom{g_down}]
   14 root       -16    -     0K    16K -       0   1:51   0.00% [rand_harvestq]
   13 root        -8    -     0K    48K -       0   1:31   0.00% [geom{g_up}]
   12 root       -88    -     0K   224K WAIT    0   0:36   0.00% [intr{irq15: ata1}]
    4 root       -16    -     0K    16K psleep  1   0:14   0.00% [pagedaemon]
   16 root        20    -     0K    16K sdflus  1   0:10   0.00% [softdepflush]
    7 root        20    -     0K    16K psleep  1   0:09   0.00% [bufdaemon]
    5 root       -16    -     0K    16K psleep  0   0:07   0.00% [vmdaemon]
    9 root        20    -     0K    16K vlruwt  1   0:05   0.00% [vnlru]
   15 root       -68    -     0K    64K -       1   0:05   0.00% [usb{usbus0}]
    0 root         8    0     0K   176K -       0   0:00   0.00% [kernel{thread taskq}]
    3 root       -16    -     0K    16K ccb_sc  0   0:00   0.00% [xpt_thrd]
    1 root        20    0  9432K   124K wait    1   0:00   0.00% [init]
   13 root        -8    -     0K    48K -       0   0:00   0.00% [geom{g_event}]
```

▼図4　プロセス（カーネルプロセスを含む）を優先順位の低い順に整理して関連項目だけを表示させた場合の例

```
  PID PRI  COMMAND
   11 155  [idle{idle: cpu1}]
   11 155  [idle{idle: cpu0}]
    6 155  [pagezero]
  744  52  /usr/libexec/getty Pc ttyv0
  745  52  /usr/libexec/getty Pc ttyv1
  663  52  nginx: master process /usr/local/
  118  52  adjkerntz -i
 1646  21  sshd: uec [priv] (sshd)
  646  20  /usr/sbin/ntpd -c /etc/ntp.conf -
  698  20  sendmail: accepting connections (
   16  20  [softdepflush]
  563  20  /usr/sbin/syslogd -ss
    1  20  [init]
  705  20  /usr/sbin/cron -s
  695  20  /usr/sbin/sshd -oPort=13439 -oCha
    8  16  [syncer]
    0   8  [kernel{thread taskq}]
    0   8  [kernel{acpi_task_0}]
    0   8  [kernel{firmware taskq}]
    0   8  [kernel{acpi_task_1}]
    0   8  [kernel{acpi_task_2}]
    0   8  [kernel{ffs_trim taskq}]
    0   8  [kernel{kqueue taskq}]
   13  -8  [geom{g_up}]
   13  -8  [geom{g_down}]
   13  -8  [geom{g_event}]
    0 -16  [kernel{swapper}]
   14 -16  [rand_harvestq]
    3 -16  [xpt_thrd]
    4 -16  [pagedaemon]
    7 -16  [bufdaemon]
    9 -16  [vnlru]
    2 -16  [sctp_iterator]
   10 -16  [audit]
    5 -16  [vmdaemon]
   12 -52  [intr{swi6: task queue}]
   12 -52  [intr{swi6: Giant task}]
    0 -52  [kernel{mca taskq}]
   12 -56  [intr{swi5: fast taskq}]
   12 -60  [intr{swi4: clock}]
   12 -60  [intr{swi4: clock}]
   12 -64  [intr{swi3: vm}]
   15 -68  [usb{usbus0}]
   12 -68  [intr{swi2: cambio}]
   15 -68  [usb{usbus0}]
   15 -68  [usb{usbus0}]
   12 -72  [intr{swi1: netisr 0}]
   15 -72  [usb{usbus0}]
   12 -76  [intr{swi0: uart}]
   12 -84  [intr{irq1: atkbd0}]
   12 -84  [intr{irq12: psm0}]
   12 -88  [intr{irq15: ata1}]
   12 -88  [intr{irq14: ata0}]
    0 -92  [kernel{em0 taskq}]
   12 -92  [intr{irq11: em0 uhci0}]
    0 -92  [kernel{virtio_balloon}]
```

間で動作するプロセスよりも高い優先順位で動作していることがわかります。このようなしくみでカーネルとして処理すべき内容を実行してお膳立てをしていることがわかります。

top(1)はI/Oモードの表示もできる!

top(1)コマンドはプロセッサの使用率を中心にしてシステムの使用状況を見るために使われることが多いツールですが、FreeBSDのtop(1)コマンドはI/Oやコンテキストスイッチの切り替え回数の面からシステムやプロセスの状況を見ることもできます。top(1)コマンドに`-m io`というオプションを指定するか、top(1)コマンドを実行した後でⓂキーを押して表示モードを切り替えることで表示させることができます(**図5**)。それぞれの列は**表1**のような意味になっています。

オプション`-o total`を指定して実行すれば、I/Oの合計数が多いプロセスから順に表示されるようになります。ディスクI/Oが多いプロセスなどは自発的にコンテキストスイッチを発生させることにもなるので、VCSWの値も増える傾向があります。逆にディスクI/OをともなわずプロセッサバウンダリなプロセスではVCSWの値は小さく、IVCSWの値が増える傾向があります。

ディスク負荷が高いシステムの場合には、このようにtop(1)コマンドを使うことで**どのプロセスがI/Oバウンダリになっているか**などを把握できます。

しくみを学んでコマンドを深く理解する

カーネルの実装は性能を発揮するために日々の開発で徐々に複雑になっています。時折整理や統合が実施されて部分的にすっきりしますが、それでも全体としては徐々に複雑になっているといえます。

しかし、動作の本質部分はUNIXが登場した当初からあまり変わっていません。ページング、仮想メモリ空間、デバイスドライバ、ブロック転送、プロセス、プロセスのfork(2)、ファイル、ファイルシステム、コンテキストスイッチ、ページキャッシュ、バッファキャッシュなどなど。こうしたしくみを調べて本質部分のしくみを把握しておけば、top(1)コマンドの出力する内容がとてもよく理解できるようになります。

連載「チャーリールートからの手紙」ではそういった内部のしくみについても取り上げて解説していこうと思います。SD

▼**表1　top -m ioのそれぞれの項目の意味**

項目	意味
PID	プロセスID。プロセスに付与される識別番号でシステムにおいて重複することがない値
USERNAME	プロセスを実行しているユーザ名
VCSW	単位時間あたりの自発的コンテキストスイッチ回数
IVCSW	単位時間あたりの強制的コンテキストスイッチ回数
READ	単位時間あたりのブロック入力回数
WRITE	単位時間あたりのブロック書き込み回数
FAULT	単位時間あたりのページフォールトの発生回数
TOTAL	READ・WRITE・FAULTの合計値
PERCENT	システム全体の入出力(TOTAL)回数に対するそのプロセスの割合
COMMAND	実行されているコマンド

▼**図5　top -m io -o total -U root -S -aの実行例**

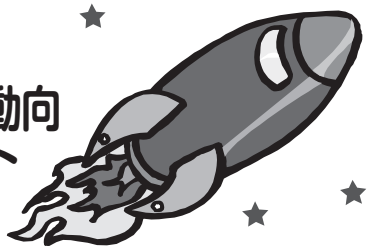
```
  PID USERNAME     VCSW  IVCSW   READ  WRITE  FAULT  TOTAL PERCENT COMMAND
    7 root            7      1      0    108      0    108  93.91% [bufdaemon]
   16 root           11      1      1      2      0      3   2.61% [softdepflush]
78148 root            8      1      2      0      0      2   1.74% /usr/obj/usr/src/make.amd64/bmake install DIRPRFX=ker
78155 root            7      2      1      0      0      1   0.87% /usr/obj/usr/src/make.amd64/bmake install DIRPRFX=ker
78158 root            5      4      1      0      0      1   0.87% /usr/obj/usr/src/make.amd64/bmake install DIRPRFX=ker
68220 root            0      0      0      0      0      0   0.00% /usr/obj/usr/src/make.amd64/bmake -f Makefile.inc1 CO
68219 root            0      0      0      0      0      0   0.00% [<sh>]
68101 root            0      0      0      0      0      0   0.00% /usr/obj/usr/src/make.amd64/bmake -m /usr/src/share/m
68096 root            0      0      0      0      0      0   0.00% make installworld (<make>)
40033 root            0      0      0      0      0      0   0.00% sudo su
85074 root            0      0      0      0      0      0   0.00% nginx: master process /usr/local/sbin/nginx (<nginx>)
  770 root            0      0      0      0      0      0   0.00% /usr/libexec/getty Pc ttyv1
  769 root            0      0      0      0      0      0   0.00% /usr/libexec/getty Pc ttyv0
```

11

Debian Developer　やまねひでき　henrich@debian.org

Debian Hot Topics

Init Systemとデスクトップ環境の動向／MiniDebConf in 台湾レポート



はじめに

先日、台湾で行われたイベントに参加してきたのですが、帰国日の朝3時まで雑談し、4時過ぎに起床して出発というなかなかの強行軍で、たいそう体力を消耗してしまいました（原稿遅延の言いわけ）。いくら往復2万円しない格安チケットとはいえ、次回以降は事前に出発時刻を確認してから予約しようと思います……。

Init Systemの変更提案

Debianでは現状のデフォルトは「insserv」というSUSE由来のSysVinitベースのシステムを利用しています。insservはそれ以前の素のSysVinitよりもシステム起動時間の短縮が得られるようになっていますが、ここ数年で、Init周りにはより高速な起動、柔軟なプロセスの管理を目指したイベントベースの各種システム（OS Xの「launchd」、Ubuntu/RHEL 6の「Upstart」、そしてFedora/openSUSEの「systemd」）が出てきています。次のリリース「Jessie」に向けての開発で、「DebianもデフォルトのInit Systemを、これらのより先進的なシステムに変更してはどうか？」という提案がメーリングリストで出されました[注1]。

「じゃあ、insservからsystemd/Upstartに乗り換えよう」となったかというと、ここで激論が交わされることとなりました。この背景には、systemdの「cgroupsなどのLinux Kernel固有の機能を大前提としている」作りがあります。通常のLinuxディストリビューションでは問題とはならないのですが、Debianの場合はkFreeBSD/Hurdなどの非Linuxカーネルを利用することを目標の1つに置いています。従って、systemdをデフォルトにするということは、kFreeBSD/Hurdを捨てることになり、コンフリクトが生じてしまいます。

systemdの主要開発者であるRed Hat社のLennart Poettering氏は「Debian kFreeBSD is a toy OS」[注2]と発言しており、systemdのUpstreamによる歩み寄りは、さほど期待できません。

Upstartはどうかというと、こちらにも不満が聞こえてきます。プロダクト自体に対してではありません。Upstreamに取り込まれるパッチについてはCanonical社に対して「Contributor License Agreement[注3]にサインしないといけない」ということに対する不満です。通常、開発者は著作権を持っているので任意のライセンスのもとでソースコードを提供できますが、これによるとCanonicalも開発者同等の権利を持ちます。つまり、GPL-2などで提供するコードでも、Canonicalが意図すればプロプライエタリなプロダクトに利用される[注4]ことがあり得るわけ

注1）URL http://wiki.debian.org/Debate/initsystem

注2）URL http://linuxfr.org/nodes/86687/comments/1249943
注3）URL http://www.canonical.com/contributors
注4）たとえばUbuntu Oneの場合、クライアントはGPL-3ですが、サーバ側のコードはプロプライエタリです。

です。forkして開発すればこれからは逃れられますが、Debian単体のリソースで開発を行っていかねばならないという多大な負担が生じます。それもあまり現実的ではありません。最低限、Debian側でそのようなUpstreamに採用されないパッチを管理していく必要が生じます。

どちらについてもUpstreamのRed HatとCanonicalの意向で、方向性が大きく変えられてしまう可能性が高い点も移行に二の足を踏ませます。さらに「Gentooでも採用されているopenRCというのもあるよ！」という提案も追加され、事態は混沌に……。

「技術委員会(ctte)で裁定してもらおう！」[注5]ということで技術委員会にも持ち込まれたものの、本稿執筆時点ではまだ「議論真っ最中」という状態です。

なお、「技術委員会のメンバーの大半はCanonical、もしくは元Canonicalの人だから政治的要因でUpstartが優位だ」などと言う人もいました。しかし、それは穿ち過ぎではないかと思います。Canonical／元Canonicalの人とは、Ian Jackson氏、Steve Langasek(vorlon)氏、Colin Watson(cjwatson)氏の3名で、確かに技術委員会7名のうちの多くを占めています。ですが、彼らはもともと「Debian Developerとして重要な立場にいた」[注6]人たちが技術委員会に参加したのであって、Canonical関係者としてDebianの技術委員会に参加したのではない点は理解しておく必要があるでしょう。

デスクトップ環境の変更提案

Init Systemの変更提案と同時期に、「デスクトップの標準インストールをXfce4に変更しよう」という提案が出されました。おもな理由は「CD1にGNOME3が入りきらないことを心配しなくてもいい」[注7]、「よりシンプルなデスクトップを提供できる」、「Init Systemへの依存を減らせる」の3点です。

サイズに関しては「そもそもCDでのインストールイメージに固執しなくてもいいのでは？DVDで提供すればいいだろう。もう、みんな前世紀にDVDへ移行してるよ」、「USBメモリでのインストールのほうがメジャーになってきてるよ」、「DVDサイズが普通……って、超遅いインターネット接続しかない地域の人には辛い」などの意見が寄せられていますが、決定的なものはありません。

デスクトップがGNOME2からGNOME3で大きく変わり、ほかのデスクトップ環境と大きく使い勝手が変わっている点については「Flashbackモード[注8]があるからそれを使えばいい」という回答が寄せられました。

議論の中でとくに沸騰していたのが、GNOMEをデフォルトのデスクトップ環境とするのに「systemdの存在を前提(GNOMEが依存するlogindはsystemdが必要)としている」という問題です。前述したように、FreeBSDカーネルやHurdカーネルもサポートの範疇として、幅広い選択肢を提供したいDebianからすると、このような「縛り」は好ましくありません[注9]。

では、*BSDシステムではGNOMEは動かないのかというと、実際にOpenBSDの開発者がGNOME 3.10を動かしているビデオ[注10]がありますので「動きはする」ようです。彼によると「systemd依存の部分はdropしている」そうで、

注5) URL http://bugs.debian.org/727708
注6) Ian Jacksonは元Debianプロジェクトリーダー、vorlonは元FTPマスター、cjwatsonはopensshをはじめとする各主要パッケージのメンテ、というように錚々たる面子がUbuntuの立ち上げのときに、Canonicalに雇用されたのです。
注7) 過去、Debian 7 "Wheezy"のリリース前にも一時期、同様の理由からGitリポジトリのほうにはXfce4を標準としてコミットされました。しかし、パッケージをxzで圧縮することで、何とかGNOME3を標準としたという経緯があります。
注8) 過去には「Fallbackモード」という名前でした。Debianではgnome-session-flashbackパッケージで提供されています。詳細はURL https://wiki.gnome.org/GnomeFlashback参照。ちなみに開発者はDebianのGNOMEチームの中心の1人、Josselin Mouetteさんのようです。
注9) 「いや、Xfce4でもGNOME関連コンポーネントを利用するからsystemdからは逃れられない。xfswitch-plugin→gdm3→gnome-settings-daemonという形で引っ張られるぞ」などというコメントも付いています。一応、xfswitch-pluginパッケージ自体はXfce4としては必須ではありませんので、systemd依存からは外れます。
注10) URL https://www.bsdfrog.org/tmp/gnome310.webm

そのため「GNOMEのおもしろさは半分ぐらい損なわれる」ともコメントしています。

そのほかにも「Unityデスクトップはなぜパッケージとしてないのか？」→「いろいろなコンポーネントにパッチが当たりまくっていて難しい」という話が出たり、「俺はIceWMで満足だ」「KDE Plasmaデスクトップをデフォルトにすればいいよ（釣り by KDEメンテナ）」「現在、Linux Mintのデスクトップ環境であるMATEをDebianに入れるべく作業中」[注11]など収拾のつかない議論となりました。

途中、「Xfce4は開発者の数が少ないよね（UpstreamとDebianの両方で）」「Xfce4だとGNOMEが提供しているアクセシビリティ機能がないのは問題だ（Universal OSを目指すDebianとしてはふさわしくないのでは？）」などの意見が出されましたが、いったんtaskselパッケージのGitリポジトリではXfce4への変更がコミットされ、「来年のDebConfで経過を見て話し合って最終的に決めよう」ということになりました。

MiniDebConf in 台湾

台北駅からバスで1時間ほどの烏來（うーらい）で行われたminiDebConf[注12]（**写真1**）に参加してきましたので、そのご報告を。

筆者は初めて台湾を訪れたのですが、11月だというのに30度近い気温に驚きました。現地の人曰く「暖かいでしょ。台湾はクリスマスを祝うことないので、クリスマスシーズンにバカンスに来るといいよ！」とのこと。もっと驚いたのはバスから眺めた日本ブランドが氾濫する台北市内の街並みです（セブンイレブン、ファミリーマート、モスバーガー、吉野家、無印良品、東急ハンズ、JTBなど）。道行く人たちも日本人と似た風貌と服装。違いと言えば日本語

▼写真1　MiniDebConf in 台湾の様子



ではなく中国語で書かれた看板の文字だけ……とこれまで出かけた海外のイメージを打ち壊されました。コンビニ店内も若干地元の品々が並ぶものの、日本製のお菓子やお酒がところ狭しと並べられていて、店舗内の販促キャンペーンもONE PIECEのキャラグッズプレゼント……と台湾にいることを感じさせてくれない状態でした。

このイベントは、Paul Liuさん[注13]やAndrew Leeさん[注14]らが中心となり、烏來から車で曲がりくねった山道を15分ほど走ったところにある「原烏來生態農場」という合宿所のようなところで、40名ほどの参加者を集めて行われたものです。期間は11月9日～11日の3日間で、最初の2日が発表、最後の1日がアンカンファレンスとなっていました。内容をかいつまんで紹介すると、

- NextVODBox（網楽通）というsh4ベースのセットトップボックス（STB）でのDebianの利用
- Mozilla台湾の人によるFirefox OSの概要紹介
- PgREST（PostgreSQLのRestful API）の紹介
- SlateKit Shellを利用したStarTrek風インターフェースの作成
- CloneZillaなどとともに本イベントのメインスポンサーである國家高速網路與計算中心（National Center for High-Performance Computing）で開発されているツール、Ezilla

注11）MATE関連パッケージがNEW queue（URL http://ftp-master.debian.org/new.html）で審査待ちとなっている様子が伺えます。
注12）URL http://wiki.debian.org/DebianTaiwan/MiniDebConf2013
注13）Debianでメンテしているパッケージにnicovideo-dlやonscripterなどがあるところからわかるように、ある方面での日本文化にはやたら詳しい方です。
注14）軽量デスクトップ環境の1つ、LXDEの開発者でもあります。

とPartcloneの紹介
- Apache BigTopの紹介とpreseedを利用したインストーライメージについて
- 各種ARMデバイスの紹介
- 台湾の先住民族の利用言語のサポート活動

と、意外なことにあまりDebian色は強くありません。筆者は日本でのDebian関連の活動について簡単な紹介をさせていただきました。サイトのページには発表者のスライドが掲載されていますので、興味のある方はご覧ください（多くが中国語／英語のチャンポンですが、なんとなく意味が伝わるかと思います）。

そんな中、参加者の1人が教えてくれた台湾のサイト「零時政府」がとても興味深かったので紹介させてください。

オープンデータ「零時政府」

零時政府（http://g0v.tw/）は、台湾政府が公開している情報をさまざまな形で補足／加工して共有する、いわゆる「オープンデータ」の試みです。いくつか目立ったものを取り上げると

- 「中央政府總預算」プロジェクト（http://budget.g0v.tw/budget）では政府の予算についての可視化が行われており、どのような支出が多いのか、ここ数年で増加／減少しているのかなどがわかりやすく把握できる
- 「新聞小幫手」（http://newshelper.g0v.tw/）はインターネットに多く広まる「デマニュース」について、利用者側から訂正を報告してもらい、その拡散を食い止めようという試み。Firefox/Chrome/Safari用ブラウザプラグインも提供されており、サイトにアクセスした際に新聞小幫手サイトにそのURLが登録されている場合は「デマの可能性があるという報告があるよ！」と教えてくれる
- 「政誌」（http://fact.g0v.tw/）は過去にあった大きな事件を時系列でまとめ、わかりやすい形で表示している。大きな事件でも、過去の出来事は日々のニュースで押し流されてしまいがちだが、そんなニュースを「振り返り」できる

ほかにも、「過去の議会で各議員がどのような発言をしたか、どのような議案について賛成／反対の票を投じたのかを時系列で追える」サイトもありました。現状だと選挙前のパフォーマンスだけで判断することが多いでしょうが、これを利用することで過去の事実を確認したうえで投票ができます。

零時政府関連サイト／プラグインなどのコードはGitHubに公開されています（https://github.com/g0v/）ので、興味のある方はpull requestを送ってみてはいかがでしょうか。

パッケージ作成のチュートリアル

話題をDebianに戻しましょう。Debianプロジェクトリーダー Lucas Nussbaumさんによる Debianパッケージ作成指南のためのドキュメント「packaging-tutorial」パッケージ（そのままですね）が倉澤望さん（@nabetaro）の手によって日本語に訳されています。testing/unstableのユーザは、

```
# apt-get install packaging-tutorial
```

するだけで手に入ります。ぜひ見ていただき、これを参考にしてパッケージ作成にチャレンジしてみてください。やがては未来のパッケージメンテナを目指していただければと思います。

Debian 7.3

先日もアップデートの案内を書いたような気もするのですが、Debian 7.3がそろそろお目見えです。スケジュールからすると予定が12月12日ごろですので、そのあとの週末（12月14～15日、12月21～22日あたり）が候補になります。本誌が発売される頃には、すでに出ているかもしれませんが、準備を怠りなく。SD



Red Hat流OpenStack

小島 啓史
KOJIMA Hirofumi
レッドハット㈱　グローバルサービス本部
プラットフォームソリューション統括部
ソリューションアーキテクト

恵比寿からこんにちは

はじめまして。ソリューションアーキテクトの小島啓史です。Red Hatではプラットフォーム系プリセールスエンジニアとして、さまざまな顧客への製品説明や提案活動を行っています。Red Hatの社員は皆さまざまな経歴を持っていますが、実は筆者は新卒エンジニアとして2010年4月に入社しました。今回はRed HatとOpenStackとのかかわりを紹介しますが、Red Hatでも数少ない(というかほぼいない)新卒ならではの話も合わせて展開していきます。

新卒でRed Hat!?

実はRed Hatは、不定期に新卒募集を行っています。ここ数年はマイナビで募集(秋開始と遅い時期ですが)をしています。筆者も先人達と同じく、オープンソースというキーワードに興味を持って入社しました。

Red Hatに入社した当初は、学生時代は情報学専攻なので多少の下地はありましたが、エンタープライズ業界のプリセールスとして求められる知識や経験はまったくありませんでした。そのため、一人前になるために入社してから最初の2年間は、先輩社員のありがたい指導の元、必死にOJTをこなしました。客先同行から始まって、ミーティングの議事録作成、QA対応、セミナー発表、技術記事執筆、研修コースの講師、顧客からのヒアリングおよび提案資料作成／紹介まで多岐にわたりました。その合間を縫って技術レベルを上げるべく、製品の動作確認や該当箇所のソースコードの確認もしました。今はFedoraを使うと、1台のPC上でネストされた仮想化機能(仮想マシンの上でさらに仮想マシンを作成する機能)を使って、検証環境の台数を最低限にすることもできますので、何百回も検証環境を作成してきました。その甲斐あってか、入社2年目で初めてデリバリーした研修コースでお客様から高評価を得られて、やっと質の高いサービスを提供できるようになったなあと思ったことを今でも覚えています。

FOLLOW THE CODE

Red Hatがコミュニティへの投資およびOpenStackを製品としてリリースしていることもあり、2013年11月上旬に香港で開催されたOpenStack Summitの企業ブースに出展するという話があがりました。筆者もブーススタッフとして参加しました。ちなみに、OpenStackとは一言で言うと、Amazon EC2/S3相当のクラウド環境を自分で構築するためのオープンソースソフトウェアです。OpenStackに関する情報は日本語でもまとめられていますので、Webページ(日本OpenStackユーザ会[注1])もご覧ください。この見出しは、その時展示ブースに記載していたメッセージです。これまでの連載で何度か触れているとおり、いくつかの例外はあるものの、原則としてRed Hatの製品はOSSコミュニティで開発されているプロジェクトをベース

注1）http://openstack.jp/

にしています。Red HatのOpenStackもその規則に則っています。このメッセージはそうした意味が込められています。そのため、製品と企業が疎結合しており、お客様が利用する製品を主体的に選択できるメリットがあります。いいことですよね。

ちなみに、通常何らかのイベントに企業出展する際は、Red Hatの製品だけを取り扱った資料やデモをお見せしているのですが、今回はOSSコミュニティのイベントということもあり、OpenStackコミュニティのソフトウェアのデモも製品と合わせて紹介させていただきました。紹介したデモは「RDO + TripleO」です。RDOとは何かというと、RHEL/FedoraでOpenStackを利用するユーザのコミュニティです。OpenStack Foundationにおいて開発されたOpenStackをRHEL/Fedoraで簡単にインストールが実行できるように、packstackというインストーラを提供しています。たとえば、1台のPC上にRDOを導入するオールインワン環境を構築する際は、必要なパッケージをyumでインストールしたあとに、次のコマンドを実行するだけで済みます。

```
$ sudo packstack --allinone
```

もちろんこれだけでは本番環境への導入はとてもできませんが、最低限の機能確認だけならばこれだけで済んでしまいますのでとても便利です。このRDOをベースとして、現在Grizzlyベースの Red Hat Enterprise Linux OpenStack Platform 3.0という製品をリリースしています。

では、TripleOとは何でしょうか。名称の由来は、OpenStack on OpenStackという名前から頭文字の3つのOを取っていることに由来します。TripleOは、OpenStack環境そのものをインストールからアップグレードといったライフサイクル管理の対象にできないか、といったコンセプトの元に生まれたプロジェクトです。イメージを手っ取り早くつかみたい方は、Webページで公開しているビデオ[注2]を確認ください。

OpenStackのWeb UIを提供するコンポーネントにHorizonというものがありますが、そのHorizonのPlug-inの1つとしてTuskarが開発されています。Tuskarを利用してWeb UIを叩き、物理環境にRDOのコントローラノード、インスタンスを実行するコンピュートノードを1台ずつプロビジョニングするというデモをこちらのビデオでご紹介しています。このTripleOもRed Hatは製品として取り込む予定であり、近い将来OpenStack-Mという製品名でリリースする予定ですので、Red Hatの今後の動向にもぜひ注目ください。

展示ブースでは、そのほかにも、ハイブリッドクラウド環境の運用管理製品であるCloudFormsのデモも紹介しました。おかげ様で展示ブースは盛況で、多くの方にご訪問していただきました。ただ、展示ブースの一番人気は製品のパンフレットでもデモでもなく、別途配布していたRed Hatのロゴ入の赤い野球帽だったのはここだけのお話です(笑)。



OpenStackのリリース名

ここからは少し余談となりますが、OpenStackのリリース名はOpenStackのデザインサミット(開発者が集まって実装方針や新機能を議論します)の開催地またはその近辺の地名に由来します。OpenStackの現在の最新リリース名はHavanaですが、次期リリース名はデザインサミットの開催地である香港のIce House Streetから名前を取って、Icehouseと名付けられました。次のOpenStack Summitに参加される方は、リリース名の由来となった観光地を訪問すると、よりOpenStackに愛着が湧くかもしれませんよ。SD

注2) http://openstack.redhat.com/Deploying_RDO_Using_Tuskar_And_TripleO

第45回 Ubuntu Monthly Report

シェルスクリプトでもGUIのダイアログを使いたい！

Ubuntu Japanese Team
あわしろいくや　AWASHIRO Ikuya
Mail ikuya@fruitsbasket.info

今回はUbuntu 13.10日本語Remixに収録されたim-setup-helperを例に、
シェルスクリプトでもGUIダイアログを使用して設定ツールを書くヒントをまとめます。

im-setup-helperというシェルスクリプトで書いたGUI設定ツール

何年も前からUbuntuをお使いの方は、かつて日本語Remixに「日本語環境セットアップヘルパ」が収録されていたことを覚えているのではないかと思います。諸般の事情により収録されなくなりましたが、13.10では微妙に復活することになりました。それがim-setup-helperです。これはパッケージ名で、Mozcセットアップヘルパーと Fcitxセットアップヘルパーというスクリプトが含まれています。「ヘルパー」と最後伸ばすようにしたのはちょっとしたこだわりです。

im-setup-helperとは

本来このようなものは使用されるべきではないのですが、今回どうしても必要と考えて準備することになりました。というのも、13.04の日本語Remixではデフォルトの変換エンジンをMozcにしていました。そのためibus-anthyパッケージに手を加えることによってMozc（ibus-mozc）をデフォルトにできたのです。

ibus-anthyとibus-mozcが同時にインストールされている場合、ハードコードされている優先度（ランク）によってibus-anthyがデフォルトになっていましたが、13.10ではこの優先度を下げたのです。ibus-anthyのパッケージをインストールしないという手もあるのですが、想像以上に大変なので選択しませんでした。しかし13.10ではこの方法を選択することができませんでした。というのもGNOMEと統合することにより、この設定をIBusではなくgnome-settings-daemonで行うようになったためです。13.10のgnome-settings-daemonは、デフォルトの変換エンジンをソースコード内で決め打ちしており、優先度などは考慮しません。最終的にはdconfというGNOMEなどの設定を行うシステムに書き込まれた値で決定するため（これをオーバーライドするパッケージを作成するという手もありますが）とりあえずはユーザが設定するものにしようと思いました。そのときはまだ日本語Remixで採用するとは思っていませんでしたし、とりあえず動くものが欲しかったのです。結果的にはその方向で正しかったのですが。

IBusの起動キー

もうひとつ大きな問題がありました。これまで半角/全角キーないしCtrl＋スペースキーでIBusを起動していましたが、13.10（というかIBus 1.5のデフォルト）ではSuper[注1]＋スペースキーで切り替える[注2]ようになりました。しかし、あろうことかSuper＋スペースキーは動作せず、Shift＋Super＋スペースを使用するというワークアラウンドが提示されました。

注1）デフォルトではWindowsキー。キーボードのレイアウトのオプションで変更可能。

注2）起動というほうがわかりやすいのですが、正確ではないので切り替えとします。

IBus 1.5単体にはこのような設定はないのですが、システム設定(gnome-control-center)ではこのような設定が可能であり、Super＋スペースキーでは次の入力ソースに、Shift＋Super＋スペースキーは前の入力ソースに変更するショートカットが割り当てられており、これを使用するということになったのです。同時に、これはgnome-settings-daemonのバージョンが3.6.4から3.8.5に上がったら修正されると聞いていたのですが、そのようなことにはならなかったのです。これも困りましたが、結局はmozc-setup-helperにCtrl＋スペースキーで切り替えるようにするオプションを追加することにしました。

問題はまだあります。Mozcの設定ツールや単語登録などの諸ツールを起動できません。これはindicator-keyboardでやるべきことですが、そのような機能は実装されていないので、とりあえずUnityから起動してもらうように細工する必要がありました。

ほかには、ライブイメージで起動した場合は質問するまでもなくMozcにするようにしました。起動するたびにいちいち質問されたらうっとうしいためです。もちろんUbuntuだけではなくそれ以外のデスクトップ環境などへの対応も必要です。

zenityの採用

このように13.10のリリースが近づくとどんどん機能が増えていったのですが、当然端末で設定を行ってもらうわけにはいかないわけで、シェルスクリプトでもGUIでダイアログが表示できるzenityを採用しました。「日本語環境セットアップヘルパ」でも採用していましたが、いっさい参考にはしていません。

前述のとおり、Mozcセットアップヘルパー(mozc-setup-helper)だけではなくFcitxセットアップヘルパー(fcitx-setup-helper)も用意しました。これはFcitxの動作に必要なパッケージをインストールしてもらい、かつIBusからFcitxに変更するのに必要な設定を行います。そればかりか、IBusに戻すこともできます。zenityはGNOMEのコンポーネントの1つであり、zenityとほぼ同じことができるKDE版のKdialogもあり、fcitx-setup-helperではこの両方に対応しました。KDE(Kubuntu)ではIBusよりもFcitxを使ってほしいという願望です。

前置きが長くなりましたが、今回はそんなim-setup-helperを書いた際に得た知見をまとめようと思います。

zenityとは

zenityはGNOMEの一部として提供されている、さまざまなダイアログを表示するプログラムです。と、説明するよりも実際に見てもらったほうが早いと思いますので、端末から次のコマンドを実行してください。Ubuntuのほか、Ubuntu GNOMEやXubuntuにもインストールされています。

```
$ zenity --info --title="zenityテスト" --text=↵
"情報です！"
```

を実行すると**図1**のように表示されます。

```
$ zenity --error --title="zenityテスト" ↵
--text="エラーです！"
```

を実行すると**図2**のように表示されます。

これだと単純な情報だけですが、

```
$ zenity --question --title="zenityテスト" ↵
--text="Ubuntuはお好きですか？"
```

のように、[はい]か[いいえ]を問うこともできます(**図3**)。

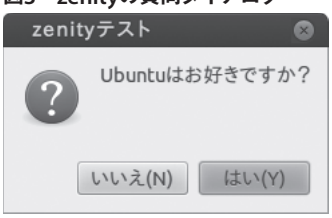
図1　zenityの情報ダイアログ



図2　zenityのエラーダイアログ



図3　zenityの質問ダイアログ



```
$ zenity --entry --title="zenityテスト" ➡
--text="お好きなLinuxディストリビューションを➡
入力してください"
```

を実行すると、ユーザにテキスト入力を促すことができます(**図4**)。

zenityのキモは、ユーザに端末を触ってもらわなくてもインタラクティブに情報を提示することであるのは間違いないですが、その入力された情報を取得することも同じくらい重要です。「--info」や「--error」は単なる情報の提示ですのでとくに問題ありませんが、「--question」の場合はユーザが[はい]か[いいえ]をクリックしています。この情報は戻り値で取得できます。[はい]だと0。[いいえ]だと1です。ユーザが入力ないし選択した情報を取得する場合は変数にしてしまいます。このことから、**リスト1**のようなスクリプトが書けます。

エラー対策はまったくしていませんが、原則としてはシェルスクリプトとしてまったく違和感のないものになっていると思います。

図4　zenityの入力ダイアログ



kdialogとは

kdialogはzenityのKDE版です。正確にはzenityがGNOME版kdialogなのですが。リリースされたのはkdialogのほうがずっと古いですし、機能も多いです。ただ、今回はzenityとkdialogで同じ機能を使う必要があったため、kdialogの独自機能はまったく使っていません。**リスト1**のサンプルスクリプトをkdialogを使用するように書き換えると**リスト2**のようになります。

zenityとkdialog両方に対応させる

まずはサンプルスクリプト(**リスト3**)をご覧ください。

2行目のif文で、デスクトップ環境がKDEかそうでないかでzenityを使うかkdialogを使うか決定しています。実際のスクリプトでは、zenityかkdialogがインストールされているかどうかを確認するべきでしょう。環境変数"DESKTOP_SESSION"は、その名のとおり、どのデスクトップセッションを使用しているかを示しています。実際に想定している環境でechoで出力してみるのが確実でしょう。ここで

リスト1　zenityによる入力テスト

```
#!/bin/sh
zenity --question --title="zenityテスト" --text="Ubuntuはお好きですか？"
RETURN=`echo $?`
if [ "${RETURN}" = "0" ]; then
    zenity --info  --title="zenityテスト" --text="それはよかったです"
elif [ "${RETURN}" = "1" ]; then
    DISTRO=`zenity --entry --title="zenityテスト" --text="お好きなLinuxディストリビューションを入力して➡
ください" `
    zenity --info  --title="zenityテスト" --text="${DISTRO}ですね。"
fi
```

リスト2　kdialogによる入力テスト

```
#!/bin/sh
kdialog --title "kdialogテスト" --yesno "Ubuntuはお好きですか？"
RETURN=`echo $?`
if [ ${RETURN} = "0" ]; then
    kdialog --title "kdialogテスト" --msgbox "それはよかったです"
elif [ ${RETURN} = "1" ] ;then
    DISTRO=`kdialog --title="kdialogテスト" --inputbox "お好きなLinuxディストリビューションを入力して➡
ください" `
    kdialog --title "kdialogテスト" --msgbox "${DISTRO}ですね。"
fi
```

のポイントは、kdialogだとスペースに続いてテキストを入力しますが、zenityだと"="のあとに続けるということです。ですからkdialogの一番後ろには半角スペースが入れてあります。これがないとエラーになります。実際に使用する場面でも半角スペースなど入れず、変数の後ろに続けて表示したいテキストを入力します。

im-setup-helperの勘所

im-setup-helperを改造したいということはあまりないとは思いますが、いちおう勘所を記しておきます。fcitx-seup-helperはとてもシンプルで、おおむねGNOMEとの統合をオフにするために、

```
$ gsettings set org.gnome.settings-daemon.plugins.keyboard active false
```

を実行し、IBusからFcitxに切り替えるために、

```
$ im-config -n fcitx
```

を実行しているだけです、Fcitx診断ツールは何か問題があったときにコピー＆ペーストして報告してもらおうと思って実行していますが、幸いにも今のところは役に立ったことはありません。

mozc-setup-helperはもっと手が込んでいて、まずはsetxkbmapコマンドで英語キーボードか日本語キーボードかをチェックしています。よって、このうちのどちらかでないとうまく動作しません。あとはGNOMEと統合した状態とIBus単体では、異なった設定を使用します（設定自体はdconfで同じなのですが）。

図5の上2行はGNOME、下2行はIBusの設定です。どちらも同じことを行っています。変数GSD_LANG とIBUS_LANG には、英語キーボードか日本語キーボードが入ります。2行目と3行目では、ログイン時に英語ないし日本語キーボードとMozcを設定しています。1行目と4行目では、ログイン時に英語ないし日本語キーボードにセットしています。ただ、これは終了時に使用していたもの（キーボードあるいはMozc）にセットされます。

mozc-setup-helperに関しては、起動時にトリッキーなことを行っています。/etc/xdg/autostart/mozc-setup-helper-autostart.desktopで起動していますので、併せて確認するとより理解が深まるのではないでしょうか。SD

リスト3　zenityとkdialog両方に対応させた入力テスト

```
#!/bin/sh
if [ "${DESKTOP_SESSION}" = "kde-plasma" ]; then
    INFO_DIALOG='kdialog --title "kdialogテスト" --msgbox '
    QUESTION_DIALOG='kdialog --title "kdialogテスト" --yesno '
    ENTRY_DIALOG='kdialog --title "kdialogテスト" --inputbox '
else
    INFO_DIALOG='zenity --title=zenityテスト --info --text='
    QUESTION_DIALOG='zenity --title=zenityテスト --question --text='
    ENTRY_DIALOG='zenity --title=zenityテスト --entry --text='
fi
${QUESTION_DIALOG}"Ubuntuはお好きですか？"
RETURN=`echo $?`
if [ "${RETURN}" = "0" ]; then
    ${INFO_DIALOG}"それはよかったです"
elif [ "${RETURN}" = "1" ]; then
    DISTRO=`${ENTRY_DIALOG}"お好きなLinuxディストリビューションを入力してください"`
    ${INFO_DIALOG}"${DISTRO}ですね。"
fi
```

図5　GNOMEとiBusの設定

```
$ gsettings set org.gnome.desktop.input-sources current 0
$ gsettings set org.gnome.desktop.input-sources sources "[${GSD_LANG}, ('ibus', 'mozc-jp')]"
$ gsettings set org.freedesktop.ibus.general preload-engines "[${IBUS_LANG}, 'mozc-jp']"
$ gsettings set org.freedesktop.ibus.general preload-engine-mode 0
```

Linux カーネル観光ガイド



第22回 ダイナミックトレースツール「ktap」

Text：青田 直大　AOTA Naohiro

Linux 3.12も無事にリリースされ、3.13の開発が始まっています。Linux 3.13にはスクリプトベースの新しいダイナミックトレースツールである「ktap」がマージされかけました。stagingのメンテナであるGregが一度、drivers/stagingディレクトリ下にマージしたものの、patchが細かく分割されていないことや、perfへの統合ができていないことなどがほかのメンテナから問題とされました。その結果、マージするにはまだ時期尚早だとしてコードが削除されてしまいました。

ということで、3.13に入ることはなさそうですが、今回はこのktapについて解説していきます。

ktapとは何か

ktapはカーネルのさまざまなところにフックをしかけ、スクリプトで指定された処理を実行できるようにするツールです。たとえば、システム全体でシステムコールが呼び出されるごとに、どのシステムコールがどのプロセスからどのような引数で呼び出されたのかを出力したり、どのプロセスがどのファイルを開いたのかをリアルタイムに出力するといったことができます。**図1**ではシステムコール呼び出し時のCPU番号、プロセスID、プロセス名、システムコールの関数名と引数、そしてその戻り値が出力されています。

この出力は**リスト1**のような、たったの3行のスクリプトで実現されています。

ktapでは、ftraceやperfとは違ってスクリプトを実行できるので、こういった簡単なことだけではなくもっと複雑な処理を行うことができます。たとえばテトリスを実行するといったことさえもできます(**図2**)。

▼リスト1　システムコール出力用のスクリプト

```
trace syscalls:* {
    print(cpu(), pid(), execname(), argevent)
}
```

▼図1　システムコール呼び出しのトレース

```
0   10724   ktap    sys_set_robust_list(head: 7f9cad4b89e0, len: 18)
0   10724   ktap    sys_set_robust_list -> 0x0
0   10724   ktap    sys_rt_sigprocmask(how: 0, nset: 7f9cad4b7dc0, oset: 0, sigsetsize: 8)
0   10724   ktap    sys_rt_sigprocmask -> 0x0
0   10724   ktap    sys_open(filename: 7f9cad4b7e40, flags: 0, mode: 7f9cad4b7e67)
0   10724   ktap    sys_open -> 0x5
0   10724   ktap    sys_read(fd: 5, buf: 7f9cad497dc0, count: 20000)
0   10724   ktap    sys_read -> 0x188
0   10724   ktap    sys_write(fd: 2, buf: 7f9cad497dc0, count: 188)
0   10723   head    sys_read -> 0x188
```

SystemTap/DTraceとの比較

ktapのようなスクリプト型のトレースツールには、ほかにもSystemTapやDTraceなどが知られています。これら2つとktapとはどのように違っているのでしょうか。

先ほどのktapでのシステムコール表示と同じことが「SystemTap」でも実現できます。SystemTapではスクリプトをCのコードに変換します。このCのコードをgccを使ってカーネルモジュールとしてビルドし、ロードするという実装になっています。この実装では解析するマシン上にgccが必要になっていますので、gccを入れておくことが少ない組み込み用途には向いていません。

一方でktapでは、ktap専用のバイトコードを生成し、それをカーネル内部に組み込まれたバイトコードインタプリタが解釈し実行するという実装をとっています。そのためgcc、その他のビルドのためのツールチェインを必要とせず、組み込みなどのマシンにも移植しやすい作りとなっています。またバイトコードがVMの中で実行されることから、スクリプトがカーネルをクラッシュさせる危険性がより少ないといった特徴もあります。

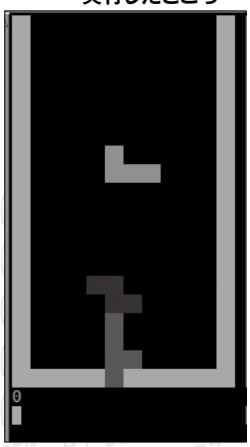
もう1つのスクリプト型のトレースツール「Dtrace」はもともとSolarisを起源にもちます。DTraceもバイトコードを使ってはいますが、DTraceで使われるスクリプト言語は安全性を重視して、ループができないように作られています。また、DTraceはCDDL[注1]のライセンスをとっていてLinuxカーネル本体にマージできません。一方でktapはGPLライセンスであるので、3.13にマージされかけたことからもわかるようにマージされるのは間近であると言えるでしょう。

ktapの使い方

それではktapを使ってみましょう。**図3**はktapのビルドの様子です。❶ktapのgitリポジトリをcloneし、❷makeすることでカーネルモジュールktapvm.koとユーザランドツールktapが生成されます。❸次に“make load”によって生成されたカーネルモジュールを読み込ませます。

注1）Common Development and Distribution License

▼図2　ktapでテトリスを実行したところ



▲https://github.com/ktap/ktap/blob/master/samples/game/tetris.kp

▼図3　ktapのビルドとHello World

```
$ git clone https://github.com/ktap/ktap.git ……❶
Cloning into 'ktap'...
remote: Counting objects: 4791, done.
remote: Compressing objects: 100% (2256/2256), done.
remote: Total 4791 (delta 2743), reused 4340 (delta 2512)
Receiving objects: 100% (4791/4791), 1.84 MiB | 623.00 KiB/s, done.
Resolving deltas: 100% (2743/2743), done.
Checking connectivity... done
$ cd ktap
$ make  ……❷
    CHK libelf
make -C /lib/modules/3.12.0+/build M=/tmp/ktap modules
make[1]: ディレクトリ `/home/naota/linux-2.6' に入ります
…
cc -Wall -O2 -o ktap  userspace/lex.o userspace/parser.o
userspace/code.o userspace/dump.o userspace/main.o userspace/util.o
userspace/ktapio.o userspace/eventdef.o userspace/kp_opcode.o
userspace/kp_tab.o userspace/kp_str.o userspace/kp_obj.o
userspace/symbol.o -lpthread -lelf
$ sudo make load  ……❸
    CHK libelf
insmod ktapvm.ko
$ sudo ./ktap scripts/helloworld.kp ……❹
Hello World! I am ktap
```

これによってktapのインタプリタがカーネル内に読み込まれます。❹あとはktapコマンドでスクリプトを指定するだけで実行できます。

また、“-s”を使って簡単なイベントトレースを行うことができます。カーネル内にあらかじめ登録されているイベントフックがトレースされ、そこで定義されている値が表示されます。たとえば“sched:sched_switch”を使えばスケジューラによるタスクのスイッチの様子を見ることができます(**図4**)。左からCPU番号、プロセスID、プロセス名で残りの部分が各イベントごとの表示部分になります。この場合、prev_commというプロセス名で、プロセスIDがprev_pidであるプロセスから、next_commというプロセス名でプロセスIDがnext_pidであるプロセスへとスイッチしている様子が見てとれます。

また“--”のあとにコマンドと引数を書くことで、指定したコマンドの動きだけをトレースできます(**図5**)。

ktapの全体的な動き

では、ktapの全体的な動きから見てみましょう(**リスト2**)。

ktapがロードされるとktap/interpreter/ktap.cのinit_ktap()が呼び出されます。ここではまず❶ debugfs下(/sys/kernel/debug下)に“ktap”ディレクトリを作成し、❷さらにその下に“ktapvm”という読み込み専用のファイルを作成しています。このファイルへの操作は“ktapvm_fops”によって設定されています。❸そしてプログラムの終了に使われる領域を初期化するコードが続きます。バイトコード実行中にエラーが発生した場合などスクリプトの実行を停止するときには、スク

▼図4　タスクのスイッチをトレース

```
$ sudo ./ktap -s sched:sched_switch |& head
1       4881    ktap    sched_switch: prev_comm=ktap prev_pid=4881 prev_prio=120 prev_state
=S ==> next_comm=swapper/1 next_pid=0 next_prio=120
2       0       swapper/2       sched_switch: prev_comm=swapper/2 prev_pid=0 prev_prio=120
prev_state=R ==> next_comm=urxvt next_pid=5592 next_prio=120
2       5592    urxvt   sched_switch: prev_comm=urxvt prev_pid=5592 prev_prio=120 prev_
state=S ==> next_comm=swapper/2 next_pid=0 next_prio=120
2       0       swapper/2       sched_switch: prev_comm=swapper/2 prev_pid=0 prev_prio=120
prev_state=R ==> next_comm=urxvt next_pid=5592 next_prio=120
1       0       swapper/1       sched_switch: prev_comm=swapper/1 prev_pid=0 prev_prio=120
prev_state=R ==> next_comm=X next_pid=5136 next_prio=120
2       5592    urxvt   sched_switch: prev_comm=urxvt prev_pid=5592 prev_prio=120 prev_
state=S ==> next_comm=swapper/2 next_pid=0 next_prio=120
1       5136    X       sched_switch: prev_comm=X prev_pid=5136 prev_prio=120 prev_state=S
==> next_comm=swapper/1 next_pid=0 next_prio=120
3       0       swapper/3       sched_switch: prev_comm=swapper/3 prev_pid=0 prev_prio=120
prev_state=R ==> next_comm=ktap next_pid=4882 next_prio=120
0       0       swapper/0       sched_switch: prev_comm=swapper/0 prev_pid=0 prev_prio=120
prev_state=R ==> next_comm=head next_pid=4878 next_prio=120
3       4882    ktap    sched_switch: prev_comm=ktap prev_pid=4882 prev_prio=120 prev_state
=S ==> next_comm=swapper/3 next_pid=0 next_prio=120
```

▼図5　指定したコマンドのトレース

```
$ sudo ./ktap -s 'syscalls:*' -- sleep 1 |& head
3       22562   ktap    sys_pause -> 0xfffffffffffffdfe
1       22562   sleep   sys_brk(brk: 0)
1       22562   sleep   sys_brk -> 0x972000
1       22562   sleep   sys_mmap(addr: 0, len: 1000, prot: 3, flags: 22, fd: ffffffff, off: 0)
1       22562   sleep   sys_mmap -> 0x7f9b8b70e000
1       22562   sleep   sys_access(filename: 7f9b8b50bec0, mode: 4)
1       22562   sleep   sys_access -> 0xfffffffffffffffe
1       22562   sleep   sys_open(filename: 7f9b8b509853, flags: 80000, mode: 1)
1       22562   sleep   sys_open -> 0x5
1       22562   sleep   sys_newfstat(fd: 5, statbuf: 7fff4cef9030)
```



リプトの戻り先をこのアドレスに書き換えて強制的に終了させます。

ここで作られた“ktapvm”は、ただioctlができるだけのファイルで、ioctlに対する動作は“ktapvm_ioctl”で見ることができます(**リスト3**)。このファイルはioctlされると❶“ktap_fops”で動作が定義された新しいファイルを返します。

このktapvmから返ってきたファイルに、さらにioctlでKTAP_CMD_IOC_RUNを行うことができ、このioctlによって、ユーザランドで生成されたバイトコードやその他の引数がカーネルに渡されます(**リスト4**、**表1**)。



バイトコードの実行

次にバイトコードの実行部分を見てみましょう。先にバイトコードを見ておきます。ktapに“-b”を使うことでバイトコードのダンプを見ることができます(**図6**)。

もともとのコードは、だいたい**リスト5**のよ

▼リスト2　ktap/interpreter/ktap.c: init_ktap

```
static int __init init_ktap(void)
{
    struct dentry *ktapvm_dentry;

    kp_dir_dentry = debugfs_create_dir("ktap", NULL);  ……❶
    if (!kp_dir_dentry) {
        pr_err("ktap: debugfs_create_dir failed¥n");
        return -1;
    }

    ktapvm_dentry = debugfs_create_file("ktapvm", 0444, kp_dir_dentry, NULL, …❷
                        &ktapvm_fops);

    if (!ktapvm_dentry) {
        pr_err("ktapvm: cannot create ktapvm file¥n");
        debugfs_remove_recursive(kp_dir_dentry);
        return -1;
    }

    kp_init_exit_instruction();  ……❸

    return 0;
}
```

▼リスト3　ktap/interpreter/ktap.c: ktapvm_ioctl

```
static long ktapvm_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
{
    int new_fd, err;
    struct file *new_file;

    new_fd = get_unused_fd();
    if (new_fd < 0)
        return new_fd;

    new_file = anon_inode_getfile("[ktap]", &ktap_fops, NULL, O_RDWR); ……❶
    if (IS_ERR(new_file)) {
        err = PTR_ERR(new_file);
        put_unused_fd(new_fd);
        return err;
    }

    file->private_data = NULL;
    fd_install(new_fd, new_file);
    return new_fd;
}
```

うに変換されています。もとのtraceの中がclosureになり、sched:sched_switchが対応するprobeのIDである267に変換され、これらを引数としてkdebug.probe_by_id関数が呼び出されています。

ではktapの心臓部分とも言えるバイトコード実行部分ktap_mainについて見てみましょう(**リスト6**)。❶まずインタプリタの状態を初期化します。ここで同時にdebugfs下に"ktap/trace_pipe_<pid>"というファイルが作られます。ktapのスクリプト内でprintfした出力は、このパイプを通じてユーザ空間から読み取ることができます。❷kp_loadでbuffに読み込まれているバイトコードを、カーネル内の構造体に読み込みます。バイトコードのフォーマットについては次の項で解説します。❸kp_callを使って"function 0"を呼び出します。この部分は読み込んだバイトコードをただただ解釈していくだけです。❹スクリプト実行中に作ったタイマーの削除などの後片付けを行います。

▼表1　ktapインタプリタに渡されるデータ

型	変数名	説明
char *	trunk	バイトコード
int	trunk_len	バイトコード長
int	argc	スクリプトに渡される引数
char **	argv	引数の数
int	verbose	出力を冗長にする
int	trace_pid	トレース対象プロセスID
int	workload	コマンドを指定したかどうか
int	trace_cpu	トレース対象CPU
int	print_time stamp	タイムスタンプの表示

▼リスト4　ktap/interpreter/ktap.c ktap_ioctl

```
static long ktap_ioctl(struct file *file, unsigned int
cmd, unsigned long arg)
{
    ktap_parm parm;
    int ret;

    switch (cmd) {
    case KTAP_CMD_IOC_VERSION:
        print_version();
        return 0;
    case KTAP_CMD_IOC_RUN:              (1)
        ret = copy_from_user(&parm, (void __user *)arg,
                    sizeof(ktap_parm));
        if (ret < 0)
            return -EFAULT;

        return ktap_main(file, &parm);
    default:
        return -EINVAL;
    };

        return 0;
}
```

バイトコードのフォーマット

バイトコードの冒頭にはヘッダが入っています。ヘッダには次のデータが記載されています。

- シグネチャ("¥033ktap")
- バージョン番号
- フォーマット番号
- エンディアン
- int、size_t、ktap_instruction、ktap_numberのサイズ
- ktap_numberが整数かどうか

このヘッダがスクリプトとカーネル側とで一致しなければktapスクリプトは実行されません。ヘッダのあとには関数のデータが並びます。いわゆるスクリプトのメイン部分も"function 0"として関数になっていることに注意してください。

関数のデータは次のものからなります。ここで「下位の関数」とあるように先ほどのバイトコードダンプの"function 0"のダンプの定数データと上位環境の変数のデータとの間に、下位の関数である"function 1"のデータが入ることになります。

- 関数定義開始行数
- 関数定義終了行数
- 引数の数
- 可変長引数かどうか
- 最大スタックサイズ
- バイトコード



▼図6　バイトコードのダンプ

```
$ cat sched.kp
trace sched:sched_switch {
        print(cpu(),pid(),execname(),argevent)
}
$ sudo ./ktap -b sched.kp

------------------------------------------------
function 0 [level 1]:
linedefined: 0
lastlinedefined: 0
numparams: 0
is_vararg: 1
maxstacksize: 3
source: foo.kp
sizelineinfo: 6         1 1 1 3 1 3
sizek: 3
        TSTRING: kdebug
        TSTRING: probe_by_id
        TSTRING: 267 ,
sizelocvars: 0
sizeupvalues: 1
        name: _ENV instack: 1 idx: 0

sizecode: 6
00400006   GETTABUP   base + 0 <- global{kdebug}
00404007   GETTABLE   base + 0 <- base + 0{probe_by_id}
00008041   LOADK      base + 1 <- 267 ,
000000a9   CLOSURE    base + 2 <- closure(func starts ➪
from line 2)
01804021   CALL       base + 0
00800023   RETURN
sizep: 1

------------------------------------------------
function 1 [level 2]:
linedefined: 1
lastlinedefined: 3
numparams: 0
is_vararg: 0
maxstacksize: 5
source: foo.kp
sizelineinfo: 10        2 2 2 2 2 2 2 2 2 3
sizek: 4
        TSTRING: print
        TSTRING: cpu
        TSTRING: pid
        TSTRING: execname
sizelocvars: 0
sizeupvalues: 1
        name: _ENV instack: 0 idx: 0

sizecode: 10
00400006   GETTABUP   base + 0 <- global{print}
00404046   GETTABUP   base + 1 <- global{cpu}
00808061   CALL       base + 1
00408086   GETTABUP   base + 2 <- global{pid}
008080a1   CALL       base + 2
0040c0c6   GETTABUP   base + 3 <- global{execname}
008080e1   CALL       base + 3
0000012c   EVENT
02804021   CALL       base + 0
00800023   RETURN
sizep: 0
```

▼リスト5　変換されたコード

```
local function f() {
    print(cpu(),pid(),execname(),argevent)
}
kdebug.probe_by_id("267", f)
```

- 定数のデータ
- 下位の関数
- 上位環境の変数
- バイトコードに対応する行数、ファイル名などのデバッグ情報



まとめ

今回は新しいダイナミックトレースツール「ktap」について解説しました。3.13には入ることはなさそうですが、近いうちにまた正式にマージされるかと思います。その日が楽しみですね。SD

▼リスト6　ktap/interpreter/ktap.c: ktap_main

```
/* Ktap Main Entry */
static int ktap_main(struct file *file, ktap_parm *parm)
{
…
    ks = kp_newstate(parm, kp_dir_dentry); ……❶
…
    cl = kp_load(ks, (unsigned char *)buff); ……❷

    vfree(buff);

    if (cl) {
        /* optimize bytecode before excuting */
        kp_optimize_code(ks, 0, cl->p);

        delta_time = gettimeofday_us() - start_time;
        kp_verbose_printf(ks, "booting time: %d (us)¥n"➪
, delta_time);
        kp_call(ks, ks->top - 1, 0); ……❸
    }

    kp_final_exit(ks); ……❹
    return ret;
}
```

January 2014

NO.27

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
波田野 裕一　HATANO Hirokazu　tcsh@tcsh.csh.sh

イノベーションの鍵をにぎる「デザイン思考」とは

今回は10月に行われたインターネットコンファレンスの模様を報告します。

インターネットコンファレンス2013

■インターネットコンファレンス2013
【日時】2013年10月24日（木）～10月25日（金）
【会場】慶応大学三田キャンパス

■カンファレンスの概要

インターネットコンファレンスは1996年からjusを含む複数の組織で共催しているインターネットに関する研究の発表会です。

今年はjusに加え、慶應義塾大学大学院メディアデザイン研究科（KMD）、日本学術振興会産学協力研究委員会インターネット技術第163委員会（ITRC）、日本ソフトウェア科学会インターネットテクノロジー研究会（ITECH）、WIDEプロジェクト（WIDE）で共催しました。参加者は2日間で約70名でした。

本会合の中心となる論文発表は3つのセッションで構成され、8件の研究成果が発表されました。内訳は、ネットワークマネジメントに関するものが3件、オーバーレイに関するものが3件、ショートプレゼンテーションが2件です。招待講演として、慶應義塾大学大学院メディアデザイン研究科の奥出直人教授による「インターネットとデザイン思考」、および独立行政法人情報通信研究機構の河合栄治氏による「JGN-XにおけるOpenFlow/SDN技術の取り組み」の2つの講演が行われました。本稿では、奥出直人教授による招待講演について報告します。

■招待講演「インターネットとデザイン思考」

高度経済成長期は「作る人＝ユーザ」であり、エンジニアリングや技術や品質が競争力の中心でした。しかし、現在はユーザの要求が多様化し、ユーザが本当に求めているものを提供できない企業は生き残れない時代になってきました。

デザイン思考とは、ユーザが何を求めているかを設計段階で考え、比較的安価なコストでイノベーションを起こすために実践される手法のことです。ここでのイノベーションとは必ずしも新しい技術のことではなく、ユーザが欲しがっているものを探すことです。多くの場合は既存のものを組み合わせて生まれます。たとえば、スティーブ・ジョブズがウォークマンに感動し、日本企業の持つ部品などを使い、iPodやiPhoneでイノベーションを起こした事例などが挙げられます。

デザイン思考は、まずビジョンを決めて、ユーザを観察することから始まります。観察においてはユーザに対してあたかも師匠をさがしている弟子となり、信頼関係を築いたうえで気づきを書き取ることに没頭します。そして、ユーザ（師匠）に対する感受性が増していくことによりユーザへの理解が深まり、最終的にはユーザの抱える問題点をあたかも自分の抱える問題点としてとらえるようになっていきます。

次に、自分の会得したユーザ視点での問題点に対して、何を提供すると価値を生むか、おもしろいかを分析し、シナリオを作成します。そしてプロトタ

イプを作成して、実際にユーザに触れてもらいます。このときはプロトタイプをすばやく作って、すぐにサービスレベルに達して、アウトプットを出さないといけません。また、発想し、作り、ユーザに渡すまでを1つのチームでやりきらないといけません。こうして作ったプロダクトについて、使い始めたユーザがまるで自分が開発者のように他人に語り始めたら成功です。今回は実際に医療現場においてこのような実践を行った事例について紹介がありました。

このようなデザイン思考によるものづくりに対して、多くの日本企業は高度経済成長期の成功体験に縛られており、自部署のメンバーを別分野にいかせなかったり、このような活動をするには別の組織を作らないといけなかったりと、デザイン思考の実践がしにくいのが現状です。その結果、技術を持っていながら、普及において他国に出し抜かれる事例が多くなってきています。

また、研究所などの予算や予備資金がなくなって、おもしろいプロダクトが生まれなくなったことと併せて、中間管理職がモノ作りを知らない職場が多くなり、モノ作りに秀でるためのマネジメントが不在になっています。しかし、そんな中にあってもコマツ(㈱小松製作所)は現在もモノ作りに抜きん出ています。このあたりはマネジメントのしかたによって今後の状況を変えられることを示唆していると言えそうです。

その後、「身体の延長」としてのハードウェア、ソフトウェア抽象化の話題となり、会場にいる人たちが関わるインターネットサービスにおいても、そのプロダクトやサービス設計におけるデザイン思考、その実装における「身体の延長線」という意識は今後非常に重要になる、という内容で招待講演を終えられました。

最後に質疑応答が行われ、「デザイン思考に必要な基礎知識は何か」という会場からの質問に対して、奥出先生は「ユースケースが書けること」、「スケッチ力」、「感動する感受性」の3つであるとの回答をされていました。

「デザイン思考」というと、デザイナのような職業における感性的なものというイメージでとらえがちですが、プロダクトやサービスを設計するうえでも、今後は非常に重要になってくるものだとの感想を持ちました。

■受賞論文／発表

会場では例年どおり、ポスター展示とデモンストレーション展示が行われ、ポスター展示が11件、デモンストレーション展示が3件実施されました。さらに今年はそれぞれのポスターやデモンストレーションを紹介するためのセッションも開かれました。

本会では優秀な論文や発表を表彰しており、今年度は次の論文が論文賞を受賞しました。

- 「Implementation and Evaluation of Latency Efficient Overlay Network(LEON)」
  (Yudai Yamagishi, Katsuhiro Horiba, Keisuke Uehara, Jun Murai)
- 「A Measurement Study of Open Resolvers and DNS Server Version」
  (Yuuki Takano, Ruo Ando, Takeshi Takahashi, Satoshi Uda, Tomoya Inoue)

また、優秀な発表をした発表者に送られるプレゼンテーション賞には次の2者が選ばれました。

- 「大規模IaaSシステムにおける予見可能な通信性能サービス」(下國治)
- 「無線LANアクセス環境のAndroid端末間における往復遅延時間を考慮した通信制御ミドルウェアの提案と実装」(早川愛)

発表された論文はインターネットコンファレンスのWebサイト[注1]から参照できます。来年度は広島大学の近堂徹氏を実行委員長に迎えて開催の予定とのことです。SD

注1) URL http://www.internetconference.org/ic2013/

Hack For Japan
エンジニアだからこそできる復興への一歩

●Hack For Japanスタッフ
鎌田 篤慎　KAMATA Shigenori
Twitter @4niruddha
佐伯 幸治　SAEKI koji
Twitter @widesilverz
高橋 憲一　TAKAHASHI Kenichi
Twitter @ken1_taka

第25回 今年の夏もアツかった! ～石巻ハッカソン開催～[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。今回は前回に引き続き、復興支援イベントとして宮城県石巻市で開催された石巻ハッカソンについて紹介します。

前号に引き続き、7月に行われた石巻ハッカソンのレポートをお届けします。石巻ハッカソンは前号でお伝えしたIT Boot Camp部門のほかに、今回紹介するチャレンジング部門とどや部門の3つの部門に分かれて行われました。

事前のアイデアソン開催

チャレンジング部門とどや部門では、本番のハッカソンの前にアイデアソンを行って、どのようなものを開発するかのアイデア出しとチーム分けを事前に行いました。当初はどや部門のみ、東京からの参加者が多いこともあって東京でのアイデアソンを企画していたのですが、それに触発される形でチャレンジング部門でも行おうということになりました。この時点ですでに、百戦錬磨のどや部門の人たちがチャレンジング部門の人たちに良い影響を及ぼすという効果が出ています。

チャレンジング部門

この部門はアプリ開発を始めて間もない初級者から中級者を対象にしたものです。

参加者は昨年のIT Boot Campでアプリ開発に初めて触れ、その後東北TECH道場[注1]の石巻道場でも研鑽を積んできた高校生や大学生が中心となっていることもあり、普段は東北TECH道場で講師を務めている4人(出張純也さん、谷口泰大さん、そしてHack For Japanスタッフでもある小泉と高橋)がサポーターとして参加しました。

アイデアソンのときにできたチームだけでなく、即席でプロのエンジニアと大学生で編成されたチームや地元石巻だけではなく岩手県立大学から参加してくれたチームもありました。

主な開発プラットフォームはJavaでのAndroidアプリ開発でしたが、Corona SDKを使う人、iPhoneアプリを開発する人もいました。また、端末ローカルで動くものだけではなく積極的にWeb APIを活用したり、サーバ側とアプリ側の実装を分担するという本格的な開発を進めるチームもありました。

ノートパソコンだけではなく外付けのディスプレイとキーボードを持ち込み、どや部門の人たちをあっと言わせるようなものを作るぞ!と意気込む参加者もいました(**写真1**)。

どや部門

東京や遠くは大阪から腕に覚えのある皆さんに、3日間で「どや!」言えるようなものを作ってもらうことで石巻の若者を触発するべく参加していただき

◆写真1　チャレンジング部門の部屋



注1　https://sites.google.com/site/tohokudojo/

ました（**写真2**）。エンジニアだけでなくデザイナーの方にも参加いただいており、デザイナーがいると最終成果の出来が違ってくるということを身をもって示してもらえました。

また、凄腕のエンジニアにはガジェット好きも多いためか、Leap Motionや某眼鏡型デバイスなどの普段なかなかお目にかかれない最新ガジェットを持ち込んでくれた参加者もおり、一時はIT Boot Camp部門やチャレンジング部門の参加者が興味津々でどや部門部屋に殺到するという場面もありました。

ハッカソン以外の楽しみも

今回はひたすら開発に打ち込むこと以外の楽しみもありました。開催中に運営サイドの突然の思いつきからカレーを作るプロジェクトが立ち上がり、急遽“カレー部門”を結成。手作りカレーが振る舞われました（**写真3**）。ほかにも、石巻を訪れた方や在住の方が集う「石巻復興バー」で飲みながら親交を深めたり、石巻を一望できる観光スポットの日和山公園まで早朝にランニングをするなど、いろいろな形で石巻訪問を楽しみました。

いよいよ成果発表

最終日に行われた成果発表の結果です（**写真4**）。

IT Boot Camp部門

IT Boot Camp部門では「ビリヤード」、「エアホッケー」、「太鼓をたたくと音が出る（Corona Wikiにも演習として出てくるものを発展させたアプリ）」、「画面上に並んだ数字をボールではじき飛ばす」、「シルエットの木に触ると鳥が飛んだり、時間が経つとイベントが起こる時計アプリ」など、初めてプログラミングを体験した方がほとんどにもかかわらず、多様なアイデアのアプリが発表されました。

またKwik[注2]を使ったチームでは、「オス・メスを仕分けるひよこ仕分けアプリ」、「ピンボール」、「石巻の名産をモチーフにしたパズルゲーム」、「納豆をひたすら混ぜて、回数が増えていくとイベントが発生するアプリ」、「ボーリングゲーム」など、こちらもユニークな発想のアプリが発表されました。

このIT Boot Camp部門で優秀賞として選ばれた

◆写真2　どや部門の部屋。ヘッドホンをして集中する場面も



◆写真3　お昼のカレー支給の様子も



◆写真4　最終日の朝の様子



注2　Corona SDK用のPhotoshopプラグイン。インタラクティブな電子ブックなどを作るのに適している。

のは、高校1年生が制作した「画面上を逃げる顔文字をタップして捕まえる鬼ごっこゲームアプリ」、高校3年生が制作した「先生のキャラクターを海の底から救出するゲームアプリ」、別の高校3年生が制作した「先生のキャラクターをタップすると人形のように触ることができるアプリ」です。これらの作品は「アイデアが良かった」、「デザイン的におもしろい」、「工夫しようという試みがあった」、「発表を見ている方たちが楽しんでいた」といった点で高い評価を得ていました。

高校生の参加者が多かったこの部門では学校の先生をネタにしたアプリがいくつか見られ、参加者の笑いを誘っていました。

チャレンジング部門

チャレンジング部門では「未来へのキオクAPIを使い、どこにいてもご当地キャラが出入りし、タップすると現在の宮城の情報が得られるARアプリ」、「被災地でがんばるママ向けの子育て・食育系アプリ」、「わんこそばをモチーフにしたゲームアプリ」、「石巻のロゴをパズルにした落ちものゲーム」、「Google Play ServicesのActivity Recognition APIを利用して、徒歩、自転車に乗っている、クルマに乗っているなどユーザの行動を表示するアプリ」などが発表されました。

チャレンジング部門の優秀賞は「学スケ」と名付けられた学生用時間割スケジューラアプリです。このアプリは大学1年生とプロのエンジニアのチーム（**写真5**）により制作されたもので、“自分の欲しいアプリを作る”ことを意識したアイデアから生まれました。“石巻工業高校の生徒に捧ぐ”とされたこのアプリは、「普通科の学校用時間割アプリでは機械製図など特殊な教科が入力できず、自分の時間割を作れない」という課題を解消することを目指しました。時間割をみんなで共有でき、さらには宿題を忘れないようにするアラーム機能も付いています。自分の身近なところからの発想である点や完成度が評価されました。

◆写真5　チャレンジング部門の優秀賞受賞チーム



作者の大学1年生は昨年の石巻ハッカソンでIT Boot Campに参加した学生（当時は石巻工業高校3年生）の一人で、Boot Campで触れたCorona SDKをマスターし、さらにJavaでのAndroidアプリ開発を学んでいるとのこと。今後が楽しみです。

どや部門

どや部門では「Corona SDKのテストが行えるアプリ『Corona SDK test runner』」、「きれいにデザインされた背景色が選べるアラーム時計」、「防災に関するツイートを発信するアカウント、特務機関NERV（ネルフ）の情報をまとめた『NERVまとめ』サイト」、「困ったこととその解決策を募る他力本願コラボレーションツール『ポテンシャライザー』」、「国会議員の所属政党履歴をgitで管理」、「ジオキャッシング[注3]において某眼鏡型デバイスを使って宝の場所が通知されるアプリ」などが発表されました。さすがにプロの開発者たちが集まっている部門だけあって、アプリやサービスの目的が明確かつデモのレベルも高く、参加者はとてもいい刺激を受けたようでした。

印象的だったのは「NERVまとめサイト」を作った石森大貴氏のプレゼンでした。石森氏は石巻市出身で実家が被災されたとのことで、「石巻の中高生や大学生にプラスになれば」と今回参加したとのこと。石森氏は「自分で何かをやると世の中が変わるよ」ということを、震災時の停電を防ぐ目的で行われた特務機関NERVのヤシマ作戦[注4]でのエピソー

注3　GPS機能を備えた機器を使って現実世界で行うアウトドア宝探しゲーム。
注4　2011年3月11日に発生した東日本大震災の際、節電を呼びかけるためにTwitterで広まった非公式キャンペーンのこと。アニメ『新世紀エヴァンゲリオン』の作中に登場する同作戦名のエピソードになぞらえたもの。

ドなども交えて話してくださいました。

どや部門における優秀賞は「ボクスケ」[注5]と名付けられたアプリです。3Dプリンタで出力できる3Dモデルを小学生でも作れることを目指したアプリという、とても難易度の高いアプリをハッカソンで形にしてしまう圧倒的なレベルの高さで文句なしの優秀賞でした。このアプリの素晴らしい点は、3D CADで使われるz軸の難しさを解消した点にあります。z軸を2Dの色の濃淡で表現することにより、直感的に3Dモデルが作れるというものです。デモにおいても実際にモデルを作るところから、3Dプリンタで出力できるCADデータに書き出すまでが見られ、まさに“どや!”という完成度ですべての参加者に驚きを与えていました。審査員からの「今すぐ投資を受けられるくらいのレベル」というコメントもその完成度の高さを物語っていました。

最後に

すべての参加者の成果発表が終わった後、このハッカソンの主催者であるイトナブ石巻[注6]代表の古山氏の総括が行われました。

「昨年の第1回目と比べて思ったのはIT Boot Campのレベルの高さ。とてもクオリティが高かったと思います。また、どや部門は本当にどやでした。同じ時間を使って開発して、これだけスゴイものを作っている。上には上がいるのを実感できたのではないでしょうか。また、チャレンジング部門では昨年のIT Boot Campで初めてアプリ開発に触れ、その後も継続して学んでいるメンバーが優勝できたのがうれしかったです。これからももっと開発者を目指す人の輪が広がっていってほしいと思います」

この1年でここまでできるようになったということに、古山氏は熱くこみ上げてくるものがあったようです。同じくこの1年、Androidアプリ開発の講師として彼らをサポートして来た筆者(高橋)にとっても感慨深いものがありました。

こうして3日間にわたる第2回石巻ハッカソンが幕を閉じました。

イトナブ石巻には「震災から10年後の2021年までに石巻で1,000人のIT技術者を育成する」という目標があります。今回のイベント後のアンケートでは、参加者の満足度、次回への参加意志ともに高く、そのきっかけのいくつかを作るお手伝いができたのではないかと思っています。

実は石巻ハッカソンの後、東京で“世界で一番面白い街「東北・石巻」に学ぶコミュニティデザイン「なぜ震災後、石巻には『面白い人』が集まるのか?」”というトークイベントが開催され、イトナブの古山氏に加え、Hack For Japanスタッフの及川、高橋もディスカッションに参加しました。そこで来年の石巻ハッカソンのスケジュール(2014年7月25～27日)が発表されるなど、次への動きがすでに始まっています。

この記事を読んで興味がわいた方は、ぜひ来年の石巻ハッカソンへ参加してみてください。楽しく充実したひとときを過ごせると思いますし、石巻の元気のある若者と触れ合い、ともに開発することは自分自身への新たな刺激にもなると思います。

それでは来年の夏、石巻で皆様をお待ちしております! SD

Column 会場の石巻工業高校の震災発生時の状況

2011年3月11日の震災発生時、今回の会場となった石巻工業高校の付近にも津波は押し寄せてグランドや校舎の中にも浸水し、残っていた生徒や職員だけでなく、避難して来た近隣の方々が孤立した状態が3日間続いていたとのことです。わずかな食べ物と水を分け合うもののすぐに底をつき、周辺道路の水がなかなか引かなかったために外から物資を届けることも難しかったという状況だったそうです。

今では当時のことが想像できないくらい通常どおりきれいな校舎になっていますが、こうして今回のようなイベントを多数の参加者の皆さんと笑顔のもとで開催できたことは、復興はもとより、前よりも良くする未来へ向けた取り組みの1つなのだと思います。

注5 **https://www.rinkak.com/voxke**(その後、実際にサービスが立ち上げられています)
注6 **http://itnav.jp/**

温故知新
IT むかしばなし

第29回

富士通の8ビットパソコン



LINE株式会社　佐野 裕　SANO Yutaka　sanonosa@gmail.com

はじめに

今回は1980年代に発売されていた富士通の8ビットパソコンについてお話したいと思います。

ラインナップと特徴

富士通の8ビットパソコンは、FM-8、FM-7、FM-77、FM77AVなどを含むFM-7シリーズのことを指します。FM-8は1981年、FM-7は1982年に発売開始されました。

FM-8では本体デザインが黒を基調とした固いデザインであることやサウンド機能がないことなどから、おそらくビジネス用途を意識して設計されたと思われますが[注1]、FM-7では一転して黄色を基調としたデザインに変わったことやPSG（Programmable Sound Generator）サウンド機能が搭載されホビーユースでの用途を意識して設計されたように見受けられます。

FM-7シリーズの大きな特徴は、モトローラ社の6809系CPUを2個搭載していたことです。現在であれば複数のCPUを搭載することは一般的に行われますが、当時としてはかなり画期的なことでした。

マルチCPUとは言っても、現在のマルチCPUのようなマルチタスクを前提としたOS上で、複数CPUコアやスレッドを管理して並列処理するような機構ではなく、シングルタスクを前提としたメインCPUとサブCPUが組み合わさって2つのタスクが同時に行えるようになっていました。具体的にはメインCPUでは一般的な処理を行い、サブCPUでは画面処理を担うという役割分担です。このしくみにより、演算やキー入力などを行いながら画面表示を同時に行うことができたのでした。

サブCPU側には画面処理用のコマンドが用意されていて、メインCPU側からサブCPU側にコマンドとデータを引き渡すことで表示や描画などの処理が行われていました。

裏RAMと共有RAM

FM-7では、メインCPUが扱う64KBのメモリ空間とは別に、VRAM（Video RAM）としてサブCPUが扱う48KBのメモリ空間が存在していました。またメインCPUのROM-BASICの約32KBをRAMに切り替えることができ、このメモリ空間は俗称で裏RAMとも呼ばれていました。

メインCPUとサブCPUは128Byteの共有RAMを介してデータのやりとりを行う必要がありました。メインCPU側で実行されるプログラムが共有RAMにコマンドとデータを置き、サブCPUがコマンドを受け取って描画処理を行うしくみでした。

YAMAUCHIコマンド

もともとサブCPU側で用意されていたコマンドは、汎用性が高いものの処理が遅いという欠点がありました。そこでデバッグ用のTESTコマンドとして用意されていた「YAMAUCHIコマンド」を用いることで、サブのメモリ空間にユーザが作成したプログラムを送り、それを直接実行することで描画処理を高速化する試みが一般的

注1）キーボードの上方に磁気バブルメモリ（32KB/128KB）が搭載できるバブルカセットホルダがあり、フタもついていたので、本来の使い方ではなく小物入れにされていました。

に行われるようになりました[注2]。YAMAUCHIコマンドは、富士通のマニュアル上では「$59 $41 $4D $41 $55 $43 $48 $49」と16進数で紹介されていましたが、これを文字列に変換すると「YAMAUCHI」となるため、一般的には「ヤマウチ」コマンドと呼ばれていました。

YAMAUCHIコマンドは、サブCPUがコマンド待ち状態のときにメインCPU側から望むタイミングで送り込んだプログラムを実行できるというもので、画面処理だけでなくプログラムしだいで画面処理以外の演算も可能ですので、メインCPUとサブCPUで同時に別々な処理を行って高速化を図る人もいました。



FM-77

FM-7の時代は外部記憶媒体としてカセットテープが一般的でしたが、FM-77では3.5インチフロッピーディスクドライブが標準搭載されるようになり、プログラムの読み込みと保存が非常に高速化しました。カセットテープだと1つのプログラムを呼び出すのに数分～数十分かかっていた[注3]のがフロッピーディスクだと数秒～数十秒というくらい高速化したのでした。

またFM-77ではグラフィックスの描画速度の向上や、本体とキーボードの分離といった変化も表れました。



総天然色PC FM77AV

その後1985年に総天然色PCという触れ込みのFM77AVという意欲的なモデルが登場しました。FM-7では640×200ドット8色というグラフィック性能だったのに対し、FM77AVでは320×200ドット4,096色、もしくは640×200ドット8色というグラフィック性能に進化しました。またサウンド面でもFM-7ではPSGだったものがFM音源とPSGをサポートすることで、完全にホビーユースを意識したモデルに進化しました。

このころの時代は、ゲームセンターにあるゲームをパソコンに移植することが多く行われていて、FM77AVでも例にもれず、セガのスペースハリアー、ナムコのドラゴンバスターやドルアーガの塔など、アーケードゲームがFM77AVの高い表現力を活かして高いクオリティで移植されたのに驚愕したものです。



FM-7シリーズの終焉

FM-7シリーズは1988年に発売されたFM77AV40SXが最後のモデルとなり、その後Intel 80386を搭載した32ビットパソコンFM TOWNSへのシフトを以ってFM-7シリーズの時代は終わりました。

この時代は世界がDOS/V互換機に向かっていたためか、FM TOWNSではモトローラの68020系CPUではなくDOS/Vで使われているIntel 80386が採用されたものと思われます。

8ビットパソコンの時代はそろそろ終わるという感覚は誰にもあり、32ビットへの移行は避けられない運命というのは誰しも理解していたように思いますが、著者を含むモトローラCPUをこよなく愛していた人たちにとって、非モトローラCPUが搭載されたパソコンに移行する気になれないというのが本音でした。



終わりに

筆者は小学生から高校生まで、富士通の8ビットパソコンを使っていました。学生時代は有り余るほど自由時間があったため、アセンブラで数ヵ月かけてゲームプログラミングしていたのも懐かしい思い出です。あまりに富士通の8ビットパソコンを愛し過ぎていたため、大学院を出た後、富士通に入社してしまったくらいです。

ただ時代は8ビットどころかすでに32ビットが中心になっていたため、社内で8ビットパソコンの話題がまったく出てこず寂しい思いをしたのは懐かしい思い出です。SD



注2) 当時のパソコン雑誌「I/O」(工学社)に「"YAMAUCHI"の秘密」と題した衝撃的な記事が掲載されたのがきっかけです。

注3) 当時はテープレコーダにつないでプログラムを音としてダウンロードしていました。保存したものと同じテープレコーダでないと、相性でエラーが起こるなんていうのも日常茶飯事でした。





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



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

特別企画

ソーシャルゲームのDevOpsを支える技術 後編

～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）

前編のおさらい

こんにちは。ユビキタスエンターテインメント（UEI）の水野[注1]です。Software Design 2013年11月号に掲載した前編に引き続いて今回は、『魔法少女リリカルなのはINNOCENT[注2]』のDevOpsに関して書いていきます。

前編では、ソーシャルゲームに重要な目的と特徴は次のとおりであることと、運営の実際についての事例を書きました。

【目的】

プレイヤーが常に新鮮だと感じるイベントやシステム、成長素材を提供し、安定してゲームを遊んでもらうこと

【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＝「ゲーム内イベント」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への問い合わせが日に数件～数十件必ず発生すること

今回は弊社で行っているDevOpsへの取り組みを、そして、IDCフロンティアの藤城さんからは、これらを支えているクラウドサービスのインフラについてお話しいただきます。

注1）URL https://facebook.com/takuhiro.mizuno
URL https://twitter.com/mizunon
注2）URL http://sp.pf.mbga.jp/12010803/

©NANOHA INNOCENT PROJECT



[Part1：開発サイド] DevOpsを構成する部門要素

本作品でのDevOpsは、「開発(Dev)」「運用(Ops)」「品質保証(QA)」の3部門から成り立っています。前述の目的を達成するには、上記の3部門が互いの立場を理解してコミュニケーションをより多くとることがとても重要です。そのために、どのようなことを行っているかを解説していきたいと思います。



開発するシステムの企画構成をリーンにする

ソーシャルゲームのDevOpsをスムーズに行うためには、まず「企画構成をリーンにする」という意識が重要です。「リーン」とは「引き締まった」という意味で、基本機能をユーザが楽しむために本当に必要な企画のみに絞り、「あるといいんじゃないかな……」や「機能Aと機能Bが連動していると、ちょっと嬉しいよね？」という程度のものは勇気を持って省きます。

たとえば本作では、対CPU戦と対人間戦の機能間は、片方の進行度や勝敗の結果が他方に影響を与えることはありません。報酬も経験値とゲーム内マネー、そしてアイテムに限定されています。しかし企画者としては「対人戦で負けたら、同じようなカード組み合わせが対CPU戦に出現して、勝つまでいろいろと試せる機能があったらおもしろいよね」とか言いたくなります。

が、ここはぐっとガマンです。

もしユーザが本当にそれを求めるのであれば、その内容でイベントを企画・開発すれば良いのです。あまり基本機能を盛りだくさんにしてはいけません。

このことは、ゲーム運営に入ったときに大きなメリットとなって効いてきます。なぜなら、機能的に軽く疎結合であるほど修正や機能追加の範囲を狭くできますから、万が一問題が発生した場合の原因も特定しやすくなるからです。それは目的の1つである「プレイヤーが常に新鮮だと感じるイベントやシステムを安定して提供すること」につながります。



開発チーム構成を考える

企画の次はゲーム運営時のチーム構成を考えます。開発チーム編成が、マイルストーンに対して不足していては新機能を安定して提供していくことはもちろんできません。前述のとおり、本作品では、ひと月の間に最低数回のリリースが発生する以上、1チームでマイルストーンをクリアしていくことは難しく、開発チームのマルチライン化が必須です。私たちは開発を3チームに分けて、このスケジュールを次の役割構成で開発を行っています。

・メインイベントの開発
・スキル追加とUI/UX改善
・ユーザサポート

この3つのチームはそれぞれ、

・企画からの要望による開発
・ゲーム運営からの要望による開発
・ユーザからの要望による開発

に対応しています。これらの開発の中にイベントなどの振り返りで得られた運用(Ops)からの意見も取り込んでいきます。

そして、このチームを数ヵ月ごとにローテーションで役割を変更します。これには次の意味があります。

・役割が固定されないので飽きにくい
・それぞれの役割の相互理解が深まる

©NANOHA INNOCENT PROJECT

とくに、ユーザサポートを全員が体験することには大きな意味があります。ユーザサポートチームでは、ユーザの問い合わせによって内容を調査し結果を返答しますが、その調査は単純なものでなく、MySQL内のレコードからApacheログまでマッチングした結果でないと返答できないことも少なくありません。ですので、幅広い知識を自然に学習することになります。また、ユーザの生の声やUI要望を聞くことで、それぞれの開発の役割を担うときに、ユーザが理解しやすく質の高いコンテンツや、より調査や追跡がしやすいシステム設計にフィードバックしようという意識を持つことができます。



ファイル構成を分離する

サーバ上に配置するファイル構成は、高頻度でデプロイしてもミスを起こさないように設計しておきます。

▼図1　システム動作に必要なファイルのディレクトリ構成

```
project_root
    ├─config      プログラム設定ファイル
    ├─include     プログラムファイル
    ├─lib         pearライブラリ等
    ├─materials   素材ファイル
    ├─standalone  バッチファイル
    ├─templates   HTMLテンプレート
    ├─web         Webルートディレクトリと
                  静的ファイル
```

▼図2　機能別ディレクトリ構成の例

```
project_root
    ├─config          プログラム設定ファイル
    ├─include         プログラムファイル
    │    ├─ battle 対人戦バトル
    │    ├─ binder カードバインダー
    │    ├─ boss   ボスバトル
   …省略…
    │    ├─ trade  トレード
```

■全ファイルを1つのディレクトリ以下にまとめる

システム動作に必要なすべてのファイルを1つのディレクトリ以下に集約します。本作品ではPHPを使用していますが、そのライブラリもシステム側にインストールするのではなく、他のコードファイルと並列に格納しておきます(**図1**)。

これにより、コードと同じようにバージョン管理ができますので、ホスト間でのライブラリバージョン違いなどのわかりにくい問題が少なくなります。

■機能別にディレクトリを分ける

本当に基本的なことですが、チームごとに分離して作業しやすくするために機能別にプログラムのディレクトリを分けます(**図2**)。

■プログラムの設定ファイルをサーバに依存しないようにする

プログラムの設定ファイルは各サーバに依存しない記述にしておきます。本案件では**リスト1**のように、DBのホスト名などはhostsでの解決にまかせ、設定ファイル上にIPアドレスなどのサーバ固有情報を書かないようにします。

個別の設定ファイルにサーバ固有情報などを書かないでおけば、すべてのサーバで同じ設定ファイルが利用でき、デプロイするすべてのファイルがバージョン管理できることになります。

▼リスト1　設定ファイルの一部抜粋

```
;スレーブDB接続の定義
db_slave_host = "masterdb"
db_slave_user = "dbuser"
db_slave_passwd = "dbpassword"
db_slave_name = "dbname_001"

;マスタDB接続の定義
db_master_host = "slavedb"
db_master_user = "dbuser"
db_master_passwd = "dbpassword"
db_master_name = "dbname_001"
```


©NANOHA INNOCENT PROJECT

これにより、設定ファイルに記述追加が必要になったときでもデプロイミスを防げます。



インフラ構成は事前準備が重要

インフラ構築にからむ部分はサービス前の設計と事前検討が重要です。リリース後にインフラ構成の変更を行うのは一大事ですので、その備えがあるとないでは余裕が違います。

■デプロイ環境専用のバージョニングシステムを用意する

開発用のバージョニングシステムだけでなく、デプロイ環境用のバージョニングシステムを用意してデプロイ履歴を管理します。

直接の手動配布やrsyncを行うことは絶対にせず、各本番サーバ経由でデプロイシステムを利用し、バージョニングシステムからファイルを適用することで更新を行うようにします。頻繁にコミットされる開発用とは分離しておくことで、問題発生時の原因絞り込みをわかりやすくし、巻き戻しやミス反映があっても相互に影響を与えることがありません。

■リリース時からログを集約する

後述のユーザサポートや、リリース直後から発生するバグ調査のために、何十台ものWebサーバのアクセス／エラーログが必要になります。Fluentd[注3]などで事前にログ集約をしておくと、いざ必要になったときにとても楽です。

■負荷が想定を超えた場合を事前検討する

ソーシャルゲームは負荷との戦いです。想定外の場合でも、開発と運用で事前検討しておくことで協力して素早い対策をうつことができます。

事が起きてから「こうしておけば、スケールアウトできたのに！」では遅すぎます。リリース前に、現状の構成でスケールアウトできないポイントはどこなのか、そこに負荷がかかったらどうするのかを話し合って開発チームにフィードバックして修正しておくことが重要です。



テスト環境は可能な限り本番環境に近づける

テスト環境と本番環境の違いは、本番デプロイ後のバグを引き起こす原因になります。とくにDB回りの環境差はトラブルになりやすい原因なので、次の点が要注意ポイントです。

■テスト環境でもDB分割を行う

テスト時にはDBの水平や垂直分割を行わず、本番でのみ分割しているのを見ることがあります。本番でのみ実行されるような動作は可能な限り減らすべきですので、きちんとDB分割された環境でテストすることが重要です。

■テスト環境でもマスター／スレーブ構成を用意する

上記と同様、テスト環境でのマスター／スレーブ構成も行っておくと安心です。テストをマスターのみで行っていると、予期せぬ動作やレプリケーション停止を起こす可能性があり、本番反映で慌てることになります。



追跡性を確保する

携帯向けソーシャルゲームでは、デッキ構築やカード合成などの複雑な操作を狭い画面で行うということもあり、根本的にユーザの操作ミスや勘違いが起きやすい環境です。そのため、ユーザサポートには本当のバグと誤操作に基づいた報告が混ざることになりますので、それを判別できるように、すべてのデータを追跡可能にする必要があります。

注3） URL http://fluentd.org/

たとえば本作品では、次のデータがサービスインにさかのぼってすべて追跡可能なログを残しています。

■カードの出入り

ユーザが受け取った1枚1枚のカードにはすべて個別のレコードとシリアルIDが割り当てられています。これを使ってその入手方法と、削除されたのであれば論理削除フラグとその理由、その際に別のカードレコードに変化したのであれば、その変化先のカードシリアルIDが保持されています。

たとえば、「XというユーザがAというカードをイベントで手に入れ、それを合成してBを入手し、その後、Yというユーザにトレードで譲渡した」場合は、**表1**のようなレコードが残ります。これにより、「カードがなくなった」という問い合わせに対しても、すべての操作を追跡することができるのでバグなのかそうでないのかの判断ができます。

論理削除されたレコードは一定期間後にAWSのS3やGlacierなどのストレージにバックアップのうえで実削除され、必要であればDBに戻して検証を行えるようにしています。

■課金ログ

カードと同様に課金ログについても、課金状態(支払処理中なのか、支払い済なのかなど)と購入アイテムの付与状態(付与中か、付与完了したかなど)のフラグを持っていて、これについてもトラッキング可能にしています。

課金は、外部のシステムと連携して行う都合上、予期せぬ異常があることは考えられますので、その場合でも「どの状態まで課金処理が進んだのか」の追跡が行えることが重要です。



開発と品質保証のコミュニケーションを取りやすくする

日々連続的に開発を進めるソーシャルゲームでは、品質保証部門(以下、QA部門)と開発部門との連携も重要です。連携にはバグトラッキングシステム(以下、BTS)を利用しますが、弊社ではこれに「QA部門がレポートを上げやすくする」という目標を設定しています。

弊社では技術的に詳しいテスターが常駐し確認を行っていますが、ゲームとしてのバランスなどを知るのに実際にプレイしている層に近いユーザの意見も取り入れたいため、何人かの学生アルバイトにもデバッグをお願いしています。このため品質保証部門が利用する報告のためのBTSは「開発者が見やすい」よりは「報告を上げやすい」という観点でシステム選定を行いました。

今までの案件から、学生アルバイトのような層から意見を吸い上げやすくするためには「BTSの習得コストと心理的ハードルが低い」ことが重要だというのが経験的にわかってきました。そのため、数年前から次に挙げるBTSを試験的に導入して、レポート数を比較してみました。

- BugZilla
- 影舞
- Redmine
- 2ch式スレッドフロー掲示板
- Mantis

これらの中で、実際にレポート数が一番多く、

▼表1　カードの状態レコードの例(カラムの一部を変更して抜粋)

シリアルID	ユーザID	カードID	入手方法	削除フラグ	削除理由ID	移動先シリアルID	レコード変更時刻
1	X	A	イベント	削除	合成	2	2013/11/28 00:00
2	X	B	合成	削除	トレード	3	2013/11/29 03:30
3	Y	B	トレード	未削除			2013/11/29 03:30

©NANOHA INNOCENT PROJECT

気軽に意見をもらえたのは「2ch式スレッドフロー式掲示板」でしたが、報告すべき必須項目が設定できず、また気軽すぎたのかバグと関係ない内容まで投稿されてしまって、BTSとしての収拾がつかなくなることもありました。

そのため次点であった影舞を採用しました。影舞は報告項目のコントロールもしやすく、進捗管理も明確なのが採用の理由です。レポート項目も上記を反映して、「タイトル」「参考URL（ネットの報告に基づく場合）」「再現方法」「スクリーンショット画像ファイル」に絞っています。

それ以外のものは上記2つに比べてレポート数も少なく、画面の印象が開発ツールっぽくて難しいという意見が多いようでした。私たち開発者にとっては使いやすいツールなのですが、レポートがもらえなくては意味がないので今回の選択からは外しています。

これにより小さな意見でも、学生アルバイトでも気になることがあれば報告しやすくなり、その意見を効果的に開発側に上げることが可能になりました。吸い上げられた意見は、DevOps定例にてチーム精査し、開発タスクとして取り入れるべきか否かを判断しています。

まとめ

ソーシャルゲームに限らずデプロイ頻度の高いプロジェクトでは、開発・運用・QAがそれぞれ目的を忘れず、できるだけ共通認識を持つようにし、立場の違いを超えて歩み寄って対話できるようにすることが重要だと考えています。これからも目的にそって、より質の高いコンテンツを継続的にリリースできるよう努力してまりますので、興味をもっていただけたら、ぜひ『魔法少女リリカルなのはINNOCENT』をプレイしてみてください！

［Part2：クラウド事業者サイド］ソーシャルゲームの恐ろしい負荷

ソーシャルゲームをサーバや運用サイドから見たとき、最も注意し事前に対策を検討しておくべき1つに、**急激な負荷への対策**があげられます。急激な負荷が生じる例として、

・ゲームのサービスリリース当初
・ゲーム内イベントの開始直後と終了間際
・機器故障や不具合での構成縮退時

といったケースが考えられます。このような急激な負荷への対策として、クラウドサービスを利用し、スケーラブルなシステムを構成することが一般的になってきました。フロントのWebサーバやアプリケーションサーバは簡単にスケーラブルな構成を組める一方で、ボトルネックになりやすい、つまりスケールが難しいポイントがおもに2つあげられます。1つ目にデータベースなどのディスクI/O、2つ目にサーバからのトラフィックです。

『魔法少女リリカルなのはINNOCENT』では、IDCフロンティアのクラウドサービスセルフタイプ（以下、IDCFクラウド）を採用いただきましたので、スケールが難しいポイントへの対策をどのようにとられたか、クラウドプラットフォーム側から一部を紹介させていただきます。

数万IOPSをさばくには

クラウドでも速いディスクI/O

最近の各社クラウドサービスでは、IOPSを保証するオプションやフラッシュドライブを用いたディスクを提供する事業者が出てきており、クラウドサービスを利用したとしてもディスクI/Oに対して高いパフォーマンスを出せるよう

©NANOHA INNOCENT PROJECT

になってきました。

IDCFクラウドの仮想マシンでは、自動階層化にてフラッシュドライブ(SSD)をキャッシュとして利用する機能を搭載したストレージを利用しています。つまり、I/O数が多いディスク領域は自動でフラッシュドライブにキャッシュされます。これにより、仮想マシンはより高パフォーマンスなディスクI/Oを発揮することができます。また、I/O数が多いディスク領域がフラッシュドライブに退避されることで、キャッシュされなかったディスクでも安定したIOPSを得られます。これにより数千〜1万IOPS程度は出すことが可能です。データベース用途でも十分な性能を出すことが可能です。



ioDriveはやっぱり速い

本作品でも、構築当初はDBサーバを仮想マシンの構成で進めていました。しかし、サービスリリースを目前にして、ゲームの事前登録の人数が多いことから高負荷によるシステム停止を起こさせないための再検討が行われ、データベースにより高いI/O性能が求められてきました。

そこで、DBサーバを仮想マシンからFusionIO ioDriveを搭載した物理サーバへ変更し、構築し直しました。IDCFクラウドでは、物理サーバを選択できるオプション(プライベートコネクト)を提供しており、これを利用しています。ioDriveを利用するメリットとして、仮想マシンと比較して低レイテンシーかつ大量のIOPSを処理できるため、高速なディスクI/Oを手に入れることができます。その性能は他のフラッシュドライブを圧倒し、数万〜10万IOPS以上の性能が得られますので、DBサーバとしてディスクI/Oがボトルネックになることは回避できることでしょう。

もう1つのメリットに、仮想マシンのようなパブリック環境ではなく物理サーバ占有環境であるため、ほかからの負荷の影響を受けることによるパフォーマンス低下を避けることができるという点もあります。



1Gbpsオーバーのトラフィックをさばくには

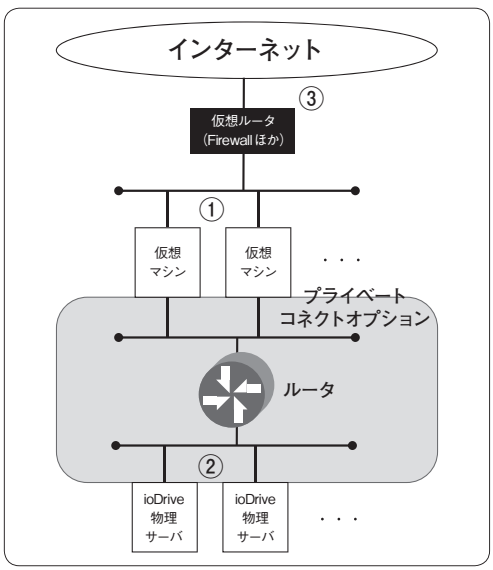
物理サーバと接続したIDCFクラウドでは**図3**のようなネットワーク構成となります。ここからは、図中に①〜③として記したポイントについて紹介します。



高帯域な仮想マシン間通信：①

10Gbps対応機器が普及してきた昨今、各社クラウドサービスでも10Gbpsのネットワークで構成されている環境が出てきています。IDCFクラウドでは、物理環境としては、すべてのネットワークを10Gbpsで構成しています。仮想マシンから見ると、そのスペックタイプに応じて100Mbps、500Mbps、1Gbpsとプランを用意しています。本作品の環境でも使われていますが、オプションで2Gbpsを提供することも可能です。大容量の物理ネットワークと仮

▼図3　IDCFクラウドのネットワーク構成



©NANOHA INNOCENT PROJECT

想マシンのトラフィックを一定の値で制限することにより、高速かつ安定したネットワークを提供しています。

先述のとおり、DBサーバのディスクI/Oボトルネックが解消することで、サーバ間の通信が増えることが想定されます。また、キャッシュサーバのように通信が集中する用途のサーバもありますので、サーバ間の通信においても安定した高帯域が必要になってきます。



物理サーバが1Gbpsで輻輳（ふくそう）した：②

ioDriveによってディスクI/Oのボトルネックが解消されたDBサーバが、ゲーム内のイベント開始直後に1Gbpsで輻輳するという事態がありました(**図4**)。物理サーバは1000BASE-Tにて1Gbps接続されており、2つのNICにてBondingのActive-Backup(mode=1)で冗長化しています。つまり、最大1Gbpsまでしか出せませんでした。

このときの回避策は、BondingのBalance-TLB(mode=5)によって「アクティブ－アクティブ」の構成に設定を変更するというものでした。これによってサーバからの送信については1Gbpsを2本使えるようになり、最大2Gbps出せるようになりました。あわせて、データベース側からはMySQLの通信圧縮も設定され、物理サーバからの通信ボトルネックも解消されました。



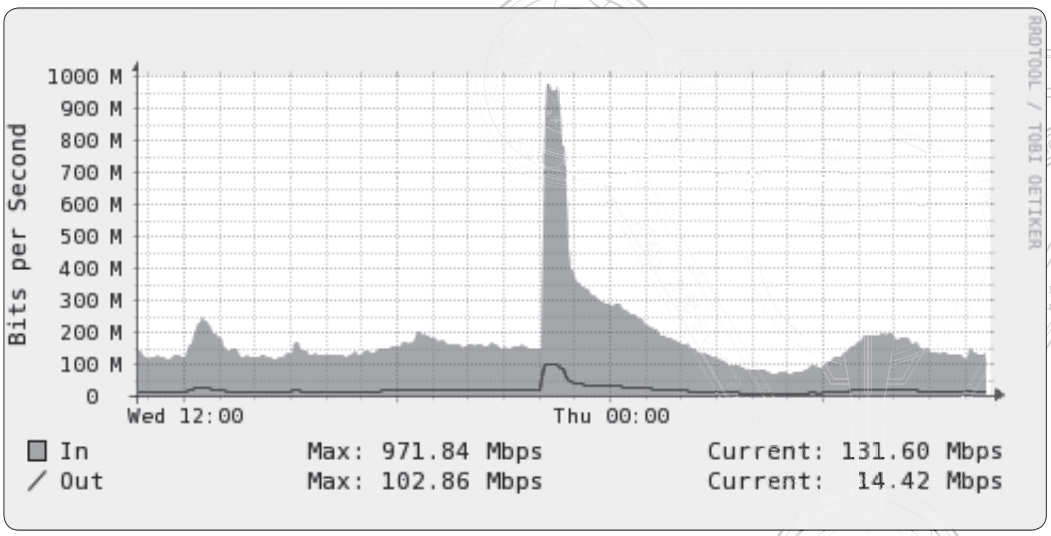
上位も10Gbps：③

インターネット側の帯域はボトルネックになりやすいポイントです。たとえば、共有の回線だったり、100Mbpsの回線だったりするかもしれません。IDCFクラウドでは、10Gbpsの共有回線を提供していますので帯域としては十分に余裕がありますが、帯域の前に上位のファイアウォール、つまり仮想ルータ[注4]の処理性能がボトルネックになる可能性が想定されます。

本作品でもサービスリリース直後から、トラフィックがぐんぐん伸びていきました。**図5**はサービスリリース直後の仮想ルータのリソース状況です。このままのペースでトラフィックが

注4) IDCFクラウドはApache CloudStackをベースとしたCitrix CloudPlatformを利用して構築していますが、この機能の1つとして仮想ルータが提供されています。Debian系のLinuxをベースとした仮想アプライアンスであり、機能としてファイアウォール、ロードバランサ、NATやポートフォワーディング、VPNなどが利用できます。各アカウントに占有する形で提供されます。また、冗長構成をとることも可能で、IDCFクラウドでは冗長構成で提供しています。

▼図4　物理サーバの接続先L2スイッチでみたトラフィック



©NANOHA INNOCENT PROJECT

▼図5　リリース直後の仮想ルータ



増えていくと、アクセスのピークの時間帯である22～23時には仮想ルータがボトルネックになる可能性も出てきました。仮想ルータはパケット数や新規コネクションにより、CPUがボトルネックになることがほとんどです。

このとき、ユーザアクセスが多かったのはもちろんですが、コンテンツがCDNのキャッシュに想定したとおりにのらない問題がありました。すぐに修正されてCDNのキャッシュにのりはじめると、そのとき最大で500Mbps近いトラフィックが約半分まで下がりました。実際に今回のゲームに限らず、サービスリリースやメンテナンス直後に、想定どおりにCDNのキャッシュにのらないということはしばしば見かけますので、とくにサービスリリース時には十分な上位の帯域を確保しておくとよいでしょう。

その後、本作品ではイベント終了間際の大量のアクセスに対応するにあたり、1Gbps以上のトラフィックを出すためにファイアウォールを外して、10Gbpsの回線へ直結するネットワークオプションを利用しています。



おわりに

ここまで、クラウド上でもスケールが難しい、ディスクI/Oとネットワーク帯域について、本作品にて実際に行ったプラットフォーム側からの対応について紹介しました。ボトルネックを解消してもいつかは次のボトルネックが出てきます。そのほかにも、開発期間が短く、十分な構築やチューニングができないこともあるかもしれません。そんなときは、ミドルウェアやアプリケーション側からの対策とあわせ、プラットフォーム側からも対策ができないか検討してみてはいかがでしょうか。SD

©NANOHA INNOCENT PROJECT

Hardware

東和電子、
高音質Bluetoothモバイルスピーカー「Olasonic TW-BT5」を発売

㈱東和電子は12月中旬より、高音質Bluetoothスピーカー「Olasonic TW-BT5」を発売する。

Olasonicは同社が2010年春より発売しているスピーカーのブランド。「音響的に理想とされる卵型の高剛性キャビネットを採用したクリアな音質」「スピーカー前面のパネル裏に取り付けられた大型ディフューザーによる、1台のスピーカーで再生しているとは思えないほどの広い音場空間」などの特徴を持つ。

TW-BT5は従来の特徴に加え、Bluetoothワイヤレス接続の使いやすさを組み合わせた。ワンタッチ接続（NFC）機能を搭載したスマートフォンや音楽プレーヤーを、本機の指定位置にタッチするだけでペアリングからワイヤレス接続が簡単に行える。約10分間Bluetooth接続されていないと自動的に電源オフするオートパワーオフ機能も搭載。

定価はオープンプライス（市場推定税込価格9,980円）。



▲Olasonic TW-BT5

CONTACT ㈱東和電子
URL http://www.olasonic.jp

Hardware

Zabbix社、
Zabbixプロキシ搭載アプライアンス製品を提供開始

Zabbix社は、ぷらっとホーム㈱製ハードウェアを採用し、Zabbixプロキシを搭載したアプライアンス「Zabbix Enterprise Appliance ZP-1200」を11月22日より提供開始した。

本製品を導入することで、サーバの購入、検証、セットアップ、Zabbixプロキシのインストールと初期設定、周辺ソフトウェアの設定といった煩雑な作業を行うことなく、Zabbixプロキシを簡単、安価に導入できるようになる。

本製品はZabbixサーバの子サーバとして動作し、監視処理を仲介させることでリモート拠点やNATやファイアウォール越しに監視を行える。VPN機能の搭載によりインターネット越しの監視でもZabbixサーバに安全にデータを送付できる。

価格は149,000円（税別）、1年間の保守費用は50,000円（税別）。



▲Zabbix Enterprise Appliance ZP-1200

CONTACT Zabbix社
URL http://www.zabbix.com/jp

Hardware

バッファロー、
セキュアなリモートアクセス環境を実現するVPNルータを発売

㈱バッファローは、SMB（Small and Medium Business）・SOHO向けのVPN（仮想専用線ネットワーク）ルータ「VR-S1000」を12月中旬より発売する。

本製品は複数の拠点間で安全なVPNを構築できるVPNルータ。拠点間をIPsec VPNで接続することでサーバなどのネットワーク資産を安全に共有できる。

外出先のPC、タブレット、スマートフォンなどのモバイル端末から、オフィスのサーバや自宅のPCにアクセスできる「L2TP over IPsec」にも対応。LTEなど高速化が著しいモバイル回線を活用した、場所に縛られない自由なワークスタイルにもマッチするという。

IPsec環境で100Mbpsの高速スループットは光回線の帯域をフルに活用でき、スムーズで安定したVPN通信が可能。一度に10本のVPNトンネルを設立でき、同時に10台のクライアントで、ストレスのないVPN通信を行える。価格は31,290円（税込）。



▲VR-S1000

CONTACT ㈱バッファロー
URL http://buffalo.jp

Service

シーズホスティングサービス、添付ファイル自動暗号化「オートクリプト」を提供開始

シーズホスティングサービスはメールの添付ファイルを自動で暗号化する「オートクリプト」の提供を開始した。

オートクリプトは送信メールの添付ファイルを自動で暗号化、送信する機能を、メールサーバに追加するソフトウェアオプションサービス。

重要な情報を添付ファイルとして送信する際、これまでは送信者自身がファイルのZIP暗号化、パスワードの作成、送信などの作業を実施しなければならなかった。しかし、同サービスを用いることで、これらの作業が自動で行え、作業効率化とともに安全な添付ファイルの送信が可能となる。暗号化不要の送信先を設定することもでき、必要に応じたメールセキュリティ対策を行える。

オートクリプトは「専用サーバーサービス・フルマネージドプラン／ライトマネージドプラン」「シーズVPSサービス・フルマネージドプラン」で利用できる。初期費用が3,000円（税別）、月額利用料が3,000円（税別）となっている。

CONTACT シーズホスティングサービス
URL http://www.seeds.ne.jp

Service

ハイパーボックス、iPhone/iPadなどマルチデバイス向け配信のできる動画配信クラウドサービスを提供

㈱ハイパーボックスは、動画配信サービスのビムーブ㈱と提携し、中小企業向けに定額制動画配信のクラウドサービス「HyperCloudライブストリーミング」を、2014年1月より提供する。

同サービスは、可用性の高いクラウド基盤「HyperCloud」上にてSaaS型で提供される動画配信サービス。Apple社が開発したHLS（HTTP Live Streaming）という新しい配信方式を利用することで、iPhone、iPad、iPod TouchなどのAppleのモバイル機器では今まで閲覧不能だった、Flashプレーヤー向けの動画配信が可能になる。

また、このHLS配信方式はHTTPにてコンテンツを配信するため、ファイアウォールを使用して特定のコンテンツへのアクセスをブロックしている場合などにも有効で、企業内ユーザへの配信を考えるうえでも大きな利点がある。たとえば、iPhoneやiPadを持つ社内外の全営業社員に対する社内コミュニケーション用途などにも、安心してライブストリーミング配信ができるという。価格は月額19,800円（税別）。

CONTACT ㈱ハイパーボックス
URL http://www.hyperbox.co.jp

Topic

Skeed、「2013 Red Herring Top 100 Global」を受賞

経済全体を変化させる可能性のある技術トレンドや最新情報を随時紹介し、投資家、経営者、技術者の間で権威ある雑誌／オンラインニュースとして知られる『Red Herring』。それを発行するRed Herring社は、毎年、北米、欧州、アジアで画期的なしくみやテクノロジを持ち、世界を牽引し得る革新的な有望企業上位100社に対して、「Red Herring Top 100 Global」を贈っている。過去には、Google、Yahoo!、Salesforce.comなども受賞しており、ベンチャー企業の権威ある登竜門的位置づけとして注目されている。

㈱Skeedはこのたび、その「2013 Red Herring Top 100 Global」を受賞した。11月18～20日に米ロサンゼルスで開催された「Red Herring Global Forum」の中で、Skeedは各地域から選出された企業とともに最終プレゼンテーションを行い、技術革新度、経営能力、事業戦略性、財務遂行能力などを基準に、成長の可能性と潜在力、革新性が最も高い上位100社として同賞を受賞した。

同社は、栄えある受賞を糧にグローバル市場に向けて技術および事業の革新を一層進めていくという。

CONTACT ㈱Skeed
URL http://skeed.jp

新連載

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke

第1回 OMM killer

新刊『Linuxシステム【実践】入門』が絶好調のくつな先生の連載が読めるのはSDだけ!!



今月から「りな充」(Linuxを充実して使ってる人)になるためのお得情報を提供することになりました。たぶん毒にも薬にもならない1ページになるでしょうが、どうぞご贔屓におねがいします。今回は、Linuxサーバをメンテナンスしていると一度は襲われたことがあるであろうメモリの地上げ屋、プロセスの殺し屋OOM killerです。こんなのが客のシステムに来られたら……怖いですね、恐ろしいですね。この殺し屋から逃れるには**/proc/プロセスID/oom_score_adj**に**-1000**を設定します(Linuxカーネル2.6.35以前はoom_adjに-17を設定します)。デーモン起動した際にプロセスID取得して設定してもいいですし、OpenSSHなど各自調整をしてくれるソフトウェアもあります。改めて自分の管理するサーバのプロセスを見なおしてみましょう。

Letters from Readers



2013年の流行語大賞

先日、2013年の「ユーキャン新語流行語大賞」が発表されましたね。予想どおり「（いつやるの?）今でしょ」が大賞を受賞していました。じつは本誌でも2013年1月号～12月号の記事の中で、2回「今でしょ」が登場しているのです。ご存じでしたか? ただ、「今でしょ」と同時に大賞を受賞していた「倍返し」は本誌の中にはありませんでした。さすがに使えるシチュエーションがないか……。

2013年11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　我が友Emacs

10月号で特集したVimと双璧をなすエディタ「Emacs」を特集しました。Emacsの操作方法から実際の開発に使うまで、さまざまな局面でヒントとなるテクニックを紹介しました。

10月号がVimだったので、余計におもしろく読めた。

大阪府／松井さん

OS XとFreeBSDでEmacsを使用していますが、たいして設定していないので、勉強になりました。outline-modeをよく使うのですが、org-modeも使ってみようかな、と思いました。

神奈川県／吉田さん

UNIXでは、vi→Emacs→Muleです。PCでは、MIFES→VZ Editor→秀丸に移っています。正規表現を使った一括置換など頭を悩ませました。

東京都／はじまりはMIFESさん

高石さんの章のコラムにあった設定ファイルをorgで書くことと、るびきちさんの章のhelmが興味深かったです。個人的には、標準の設定からどのように設定を変更したか、などを簡単に把握する方法があれば知りたいです。機会があれば取り上げてくださるとうれしいです。

大阪府／出玉のタマさん

具体的な内容が多くておもしろかったです。私の周りのEmacserは年代によって使い方が結構違うので、もし次のEmacs特集があるのなら、そのあたりが気になるところです。

東京都／yasunrcさん



Vim特集のとき同様、熱いコメントが多かったです。具体的な要望や感想も多く、次回の企画の際の参考になります。

第2特集　サーバサイドフラッシュ Fusion-io全方位解説

サーバサイドフラッシュの代表格Fusion-ioについて取り上げました。フラッシュメモリの動作原理から新技術の性能検証まで幅広く解説しました。

ioDrive2については、価格性能比がかなり高いようだので使ってみたいですが、現状の業務でioDrive2を活用するような仕事が入らないのが残念です。

岩手県／隼さん

あまり見たことのない情報だったので勉強になりました。

長野県／北村さん

NAND型フラッシュのしくみ、今まで何度も勉強したんですが、この記事がわかりやすかったです！ フラッシュストレージ、懐に余裕があれば、ぜひ試してみたい!

神奈川県／ewiad420さん



ちまたにSSDなどコンシューマ向けのフラッシュストレージの記事は多々あれど、サーバ向けの記事はあまりありません。とくにFusion-ioは最も注目されている製品ですので、今後、業務で扱う機会もあるかもしれません。今のうちに製品の特性など、基本的な情報は把握しておきたいですね。

一般記事　Ubuntu 13.10 “Saucy Salamander”

2013年10月17日にリリースされたUbuntuの新バージョンについて、その新機能と今後の方向性について紹介しました。

X-Windowが置き換えられるとは思いませんでした。びっくりです。

千葉県／Tayuさん

Ubuntuばかりですが、ほかのLinuxにも言及してください。たとえば、Vine、KNOPPIXなど。

熊本県／佐伯さん

今回のリリースは、ディスプレイサーバの変更など比較的大きな変更がありました。スマートフォンやTVなどへの対応など、Ubuntuは今後も大きく進化していきそうです。Vine、KNOPPIXはないものの、RHEL、Debianなどは、連載記事でも随時レポートしていきますので、お見逃しなく。

短期連載　小規模プロジェクト現場から学ぶJenkins活用

全5回に渡るJenkins短期連載の最終回。最後はJenkins環境をバックアップする際のノウハウを紹介しました。

今、Jenkinsを使おうかTravis CIを使おうか考えていたところだったので、参考になりました。

北海道／後藤さん

CIを職場に浸透させるために、日々ネタを探してるので、役に立ちました。

東京都／前田さん

最終回では1台のPCでスレーブを使う方法を紹介していましたが、このように小規模なプロジェクトでもJenkinsを導入できることを伝えるのが、本連載の大きなテーマでした。ご自身の職場でも、導入できそうだと思っていただけたなら幸いです。

特別企画　ソーシャルゲームのDevOpsを支える技術

モバゲーから提供されているゲーム「魔法少女リリカルなのはINNOCENT」の環境は、IDCフロンティアのクラウドサービス上に構築されており、開発や運用はユビキタスエンターテインメントが行っています。今回は前編ということで、インフラ環境や運用方法について解説しました。

実際のシステムが例でよかった。

大阪府／太田さん

ソーシャルゲームの運用について、いろいろ知れておもしろかった。

岐阜県／silverfoxさん

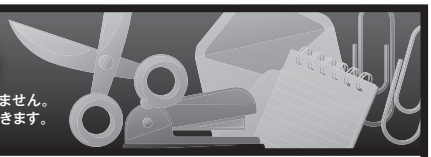
問題解決事例などがあってほしかった。

東京都／佐藤さん

読者の中にも本ゲームをプレイした人がいるかもしれません。毎回のゲームイベントの裏側で、どんな運用が行われているのかが垣間見れたのではないでしょうか。

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。



ウォーク麺

800円（税込）／㈱ナイクウイン　http://www.cupmenholder.com



仕事が忙しいときは食事はカップ麺で簡単に済ます、という人も多いのではないでしょうか。今回紹介するのは、そんなときに便利なカップ麺専用ホルダーです。使い方は簡単。カップ麺を容器ごとホルダーに入れ、お湯を注ぎ、付属のフタを取り付けます。あとは3分たったら食べるだけ。付属のフタには溝が付いていて回転してロックできるようになっている（写真）ので、カップ麺の紙のフタが浮き上がることはありません。カップ麺を持ちながら歩くようなときでも、スープがこぼれる心配もありません。万が一カップ麺を倒しても、そうそうこぼれません。PCや書類で手狭なデスクでも安心してカップ麺を食べられます。カップヌードルの通常サイズなら問題なく使えます。多少メーカーが変わっても大丈夫。ただし、BIGサイズやKINGサイズになると、ホルダーとしては使えますが、フタははめられません。そこは要注意です。（読者プレゼントあります。p.16を参照）



▲写真　フタにある溝のおかげでフタがしっかりロックされる

祝　11月号のプレゼント当選者は、次の皆さまです

①データ引越ソフト添付 USB3.0用ポータブルHDD ..... 東京都　新屋賢一様
②ESET FAMILY SECURITY ..... 東京都　尾崎潤二様

★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

Software Design
February 2014

2014年2月号
定価1,280円　176ページ

1月18日発売

［第1特集］Lisp/Ocaml/Haskell/Python/Erlang/Ruby

関数型プログラミング再入門
λ式からはじめませんか?

CPUのマルチコア・マルチスレッド化が進む今、関数型プログラミングが注目を集めています。LispはEmacsLispを環境にして入門トレーニング、そしてOcaml、Haskellと正統派のプログラミング技術に触れ、スクリプト言語はPythonで動きを知り、Erlangで最先端を学び、Rubyでそもそも関数型とは何か深く探求してみましょう。

［第2特集］目利きによるトレンド予測

2014年IT業界はどうなるのか?

昨年を振り返りながら、今年はどんな技術が受け入れられ、IT業界がどのように変化していくのか、ソフトウェア、ハードウェア、ネットワーク、SIerなど各分野の識者が予測します。今年のトレンドを押さえて、エンジニアのライフプランやビジネスに活かしてください。

［特別企画］京セラコミュニケーションシステム社の実現力

本音も語れるコンパで組織を活性化しませんか!

基本は車座で意見を交わし合う。単なる飲み会ではなく、人と人の語り合いの中から円滑なコミュニケーションと問題を解決できるチームを作り上げてきた同社の「コンパ」を紹介。

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

SD Staff Room

●iPhone5から5sに機種変更完了。その理由は地下鉄での通信の途切れやすさ。使用している路線のせいだけど、某駅で必ず途切れるのだ。あとはクーポンとポイントと下取りで半額以下になるので決断。CPUの性能向上もさることながらカメラ機能もよくなっているし、指紋認証も上々の出来。(本)

●今月の「ITむかしばなし」はFM-7の話だった。自分もサイクルスチール化してスピードアップし、キャリーラボのアセンブラ「BASE09」でサブシステムのGVRAM上で動く機械語を書いていたなぁと思い出す。ゲームの移動キーはテンキーで、方向を押すと動きっぱなしで、Breakキーがシュートボタンだったっけ。(幕)

●路側帯しかない道路では自転車は左側通行のみと明確に法規制されました。嘆かわしいことですが、もっと恐ろしいのは自転車を運転しながらスマホの画面を見ている人がいること。テクノロジーは使う人間によって良いものにも悪いものにもなる。言い古された言葉だけれども、楽しみたいなら忘れてはいけません。(キ)

●JR市ヶ谷駅のすぐそばには、釣り堀があります。もちろん屋外。こんな季節でも日中は結構お客さんがいるのです。こんなに寒くて冷たい風が吹く時期に、水辺で釣竿をにぎってじっとしているなんて、何かの苦行のように思えてなりません。私は温泉にでも行って、ぬくぬく温まりたい。(よし)

●先だって旭山動物園でカバが鮮烈デビュー(?!)した。その愛らしい姿とは裏腹に、カバはかなり獰猛らしい。生息域ではワニに喰われるよりカバに喰われる人の方が多いとのこと。そして、食すとかなり美味しいらしい。カバ料理屋をご存じの方、ぜひ情報提供をお願いします!(ムー○ン谷のひぼたます)

●新生活が始まって一ヶ月弱。何となく生活のリズムはできてきましたが、今まで実家でぬくぬくと育ってきたので、平日は早起きすることと料理の時間を取るので精いっぱい。引っ越しの片付けも終わらず、年末までに終わらせたいけどどうなることやら……。早く効率的に何でもできるようになりたいです。(ま)

本誌に記載の商品名などは、一般に各メーカーの登録商標または商標です。　© 2014 技術評論社

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

[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
2014年1月号

発行日
2014年1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。