2010年夏号

あんどきゅめんてっどでびあん

東京エリアDebian勉強会
関西エリアDebian勉強会著

Deb専

日本唯一のDebian専門誌

2010 年 8 月 14 日 初版発行

東京エリア Debian 勉強会

関西 Debian 勉強会

デビアン勉強会

目次

1 Introduction

東京エリア Debian 勉強会/関西 Debian 勉強会

Debian 勉強会へようこそ。これから Debian の世界に足を踏み入れるという方も、すでにどっぷりとつかっているという方も、月に一回 Debian について語りませんか?

Debian 勉強会の目的は下記です。

- Debian Developer (開発者) の育成。
- 日本語での「開発に関する情報」を整理してまとめ、アップデートする。
- 場の提供。
  - 普段ばらばらな場所にいる人々が face-to-face で出会える場を提供する。
  - Debian のためになることを語る場を提供する。
  - Debian について語る場を提供する。

Debian の勉強会ということで究極的には参加者全員が Debian Package をがりがりと作るスーパーハッカーになった姿を妄想しています。情報の共有・活用を通して Debian の今後の能動的な展開への土台として、「場」としての空間を提供するのが目的です。

Debian 勉強会は Debian GNU/Linux のさまざまなトピック (新しいパッケージ、 Debian 特有の機能の仕組、Debian 界隈で起こった出来事、などなど) について話し合う会です。

目的として次の三つを考えています。

- ML や掲示板ではなく、直接顔を合わせる事での情報交換の促進
- 定期的に集まれる場所
- 資料の作成

それでは、楽しい一時をお楽しみ下さい。

2 Debian の紹介

やまねひでき

2.1 Debian とは何か

「 Debian とは一体何ですか? [*1]」には以下のように書かれています。

> Debian Project は、フリーなオペレーティングシステムを作成するために連携した個人の集団です。 我々が作成したこのオペレーティングシステムは Debian GNU/Linux もしくはもっと短かく簡単に Debian と呼ばれています。

2.2 Debian の特徴

今だと Windows/MacOSX 以外の「 いわゆるフリーな OS」はいくつかあります。では、他の OS / ディストリビューションと Debian の違い、その特徴を語るキーワードとは何でしょうか? 私は「 Universal OS」「 フリー」「 ボランティア」の三つを挙げます。順を追って説明します。

2.2.1 Universal OS

これが Debian が目指すものです。その意味するところは「 あらゆるマシンで動くフリーなソフトウェアによる誰もが使える OS」です。単に PC で動くだけではなく、最近は廃れてきましたが UNIX ワークステーションや汎用機、組み込み用機器、モバイル端末、ゲーム機…あらゆるマシンで動作することを目指しています。そのため多数の CPU アーキテクチャをサポートしているのが特徴です。サポートする／した／しようとしているアーキテクチャは以下があります。

- i386 (通常の PC)
- amd64 (最近の 64bit CPU)
- ia64 (流行らない Intel の 64bit CPU. Itanium など)
- mips/mipsel (プレイステーション 2 など)
- arm/armel(シャープの Netwalker やモバイル端末がこれ)
- alpha (DEC)
- hppa (HP のワークステーション)
- sparc (Sun)
- powerpc (old mac など)
- m68k (昔の Macintosh や Amiga など)
- s390 (汎用機です)
- sh (セガサターンやドリームキャストに搭載された)
- avr32 (Atmel 社がデザインした組込向け CPU)

また、その動作の核となるカーネルも Linux だけではなく他のカーネルに取り替えても動作することを目指しています。この移植版としては

- Hurd (永遠の開発版?)

[*1] http://www.debian.org/intro/about

- kfreeBSD (i386, amd64)[*2]

があります。[*3]

単に動作する機器／カーネルが多いだけではなく、その上のユーザランドのソフトも豊富で、パッケージ化されており導入が容易になっています。現在リリースされている Debian 5.0 コードネーム「Lenny」ではその数は 25,000 パッケージを越え、その数はさらに増えつづけています。Linux で使えるソフトウェアを探す場合、大抵は既に Debian のパッケージとして提供されているので気軽に試すことができるでしょう。

それから Debian で利用可能な言語は多種に渡ります。それは自然言語（英語、日本語など）でもあり、計算機言語という意味でもあります[*4]。巷ではマイナーと呼ばれるような言語であっても「Universal OS」を目指す Debian は積極的に取り込んでいます。例えば、ブータン公用語「ゾンカ語」をサポートする DzongkhaLinux は Debian をベースに開発され、その成果は Debian に取り込まれています[*5]。

2.2.2 フリー

Debian の考える「フリー」は単に無料に止まらず、Debian フリーソフトウェアガイドライン (DFSG) という形でまとまっており、これが元になって「オープンソース」が生まれました。この点が担保される、この考えを皆が共有することでさらに豊かなソフトウェア／コンテンツ／社会が生まれています。このフリーというのは考えてみると中々奥深いものがありますので、ぜひ DFSG には一度目を通した上で Debian の考えるフリーという意味について Debian Developer の方などと話をしてみてください。

2.2.3 ボランティア

最後のキーワードです。Debian はその開発や財政基盤を会社や財団に持たない極めて稀有な開発集団です。大抵の有名ディストリビューションが企業をバックに開発をしていたり財団を持ってそのいたりする[*6]のですが、Debian 自体は財団や企業を持ちません[*7]。ボランティアが世界中でインターネットを介して開発するという状態が 10 年以上も続けられており、その規模は 1000 人を優に越えています。

2.3 誰が Debian を使っているの?

では、実際に誰が Debian を使っているのでしょうか?「仕事で使うなら Red Hat Enterprise Linux かそのクローンの CentOS が普通だよね〜」などと言い切っている人はいませんか? 実は、世の中に Debian で実際のビジネスを回している企業は山のようにあります。その中にはあなたが知っている企業もあるはずです。また、開発に愛用しているという方も少なくありません。あなたが使っているソフト／サービスは実は Debian が動いている／ベースになっている…かも知れませんよ。

2.4 最後に

簡単ではありますが、Debian の紹介をさせて頂きました。これも何かの縁だし Debian を使ってみてもいいかな、と多少でも思っていただければ幸いです。

[*2] NetBSD, OpenBSD は途中で作業する人の気力が尽きているようです。
[*3] 残念ながら Plan9 はありませんが、その上で動くツール類は移植されています。
[*4] 計算機言語の話は後で別の方が滔々としてくれるでしょう :-)
[*5] これは商用 OS では「採算にあわない」のでサポートが遅れがちになる少数言語／民族にとっての希望の現れと言えるでしょう
[*6] Fedora ← Red Hat, openSUSE ← Novell, Ubuntu ← Canonical, OpenOffice.org ← Oracle (Sun), Firefox ← Mozilla Foundation/Corporation など
[*7] 寄付などのために Software Public Interest という別法人がいますが、これは Debian だけではなく PostreSQL なども支援しています

2.5 Debian フリーソフトウェアガイドライン

Debian フリーソフトウェアガイドライン全文[*8]を掲載します。

1.「 自由な再配布」…Debian システムを構成するソフトウェアのライセンスは、そのソフトウェアを、複数の異なる提供元から配布されているプログラムを集めたソフトウェア ディストリビューションの一部として、誰かが販売したり無料配布したりすることを 制限してはいけません。また、ライセンスはそのような販売に対して 使用料やその他の手数料を要求してはいけません。
2.「 ソースコード」…プログラムにはソースコードが含まれていなければならず、 かつ実行形式での配布に加えてソースコードでの配布をも 許可していなければなりません。
3.「 派生ソフトウェア」…ライセンスは、ソフトウェアの修正や派生ソフトウェアの作成、並びにそれら をオリジナルソフトウェアのライセンスと同じ条件の下で配布することを認め ていなければいけません。
4.「 原作者によるソースコードの整合性維持」…ライセンスは、プログラムを構築時に変更する目的でパッチファイル をソースコードとともに配布することを容認している場合に限り、 ソースコードを修正済の形式で配布することを制限することができます。 この場合、そのライセンスは修正済のソースコードから構築されたソフトウェアの 配布を明示的に許可していなければなりません。 またライセンスは派生ソフトウェアにオリジナルソフトウェアと異なる名前 を付けること、あるいは異なるバージョン番号を付けることを要求できます (これは妥協案です。 Debian グループは全ての作者に、ファイル、 ソース、バイナリについての変更を制限しないよう奨めています)。
5.「 すべての個人、団体の平等」…ライセンスは、すべての個人や団体を差別してはなりません。
6.「 目標分野の平等」…ライセンスは、人々が特定の目標分野でプログラムを利用することを 制限してはいけません。たとえば、商用利用や、遺伝学の研究での プログラムの使用を制限していてはいけません。
7.「 ライセンスの配布」…プログラムに付随する権利は、プログラムが再配布された すべての人々 に対して、追加ライセンスの履行を必要とすることなく、 適用されなければなりません。
8.「 ライセンスは Debian に限定されない」…プログラムに付随する権利は、プログラムが Debian システムの 一部であるかどうかに左右されてはいけません。 プログラムが Debian から取り出され Debian とは別に使用 または配布されるとしても、その他の点でそのプログラムの ライセンス条項を満たしているならば、プログラムが再配布された すべての当事者は Debian システムにおいて付与されたのと 同じ権利を与えられなければなりません。
9.「 ライセンスは他のソフトウェアを侵害しない」…ライセンスは、そのソフトウェアとともに配布される他のソフトウェア に制約を加えてはなりません。たとえば、同じ媒体で配布される 他のソフトウェアがすべてフリーソフトウェアでなければならないと 要求してはいけません。
10.「 フリーなライセンスの例」…GPL、 BSD、および Artistic ライセンスは私たちがフリーと判断しているライセンスの例です。

[*8] http://www.debian.org/social_contract#guidelines

3 みんなの Debian デスクトップ環境を見てみよう

佐々木洋平、のがたじゅん

3.1 Debian リファレンスを読もう

Debian を使い始める前に、青木 修さんが書かれた「Debian リファレンス」(http://www.debian.org/doc/manuals/reference/) を読んでみましょう。デスクトップ環境については、第 7 章の「X Window システム」に多くのヒントがあると思うので、きっと役に立つと思います。

3.2 Debian デスクトップ環境インストール Tips

Debian の標準デスクトップ環境は GNOME と思われていますが、Debian Installer(d-i) に

```
desktop=gnome|kde|xfce|lxde
```

と、オプションをつけて tasksel に「デスクトップ環境」を指定すると、それぞれのデスクトップ環境がインストールされます。

3.3 統合デスクトップ環境とウィンドウマネージャの違い

Debian(を含めた unix 環境) に初めて触れる方にとって「統合デスクトップ環境とウィンドウマネージャの違い」と言われても「なに?」と思われる方もいらっしゃるでしょう。

統合デスクトップ環境とは、GNOME や KDE などのようにアイコンやタスクバーがあり、ファイルマネージャなどを使ってグラフィカルにファイル操作などができる環境をいいます。

もう一つのウィンドウマネージャについてですが、こちらは統合デスクトップ環境からグッと範囲が狭くなり、X.org などのウィンドウシステムで、ウィンドウの配置や外観、そのウィンドウへの入力 (フォーカス) を管理するソフトです。乱暴ですが「ウィンドウの枠」と言えばわかりやすいかもしれません。

ウィンドウマネージャには、ウィンドウを重ね合わせて表示するスタックな形のウィンドウマネージャのほかに、画面全体を使いウィンドウをタイルのように敷き詰めて利用する、タイル型ウィンドウマネージャもあります。[*9]

3.4 統合デスクトップ環境あれこれ

3.4.1 Gnome - GNU Network Object Model Environment

[*9] 日本タイル型ウィンドウマネージャ推進委員会 Wiki -SourceForge.JP: http://sourceforge.jp/projects/tilingwm/wiki/FrontPage

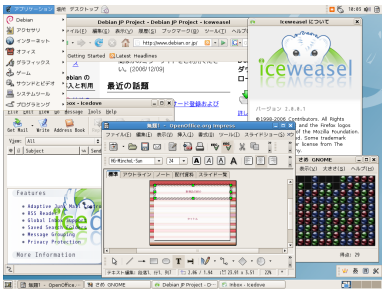


図 1　lenny での Gnome の画面

GNOME は GUI ツールキットに GTK+ を使用した統合デスクトップ環境です。 Lenny に収録されているのは 2.22、現在 squeeze に収録されているのは 2.30 です。 (2010 年 7 月現在)

Linux において「 統合デスクトップ環境」という単語が目立ち始めた時から KDE(後述) と双璧をなして発展してきました。現在では「 統合デスクトップ環境」と言えば、この GNOME か KDE(後述) と言って良いくらい流行っています。 Debian ではインストーラで GUI インストールを選択すると GNOME のデスクトップ環境が導入されます。また、 Ubuntu でも標準で採用されているため馴染のある人も多いかもしれません。

GNOME をインストールするには、タスクから「 Gnome デスクトップ環境」を選ぶか、導入したいパッケージの量に合わせて

gnome　GNOME 環境全て (GNOME プロジェクトが配布していない物も含める)
gnome-desktop-enviornment　GNOME プロジェクトの公式配布物としての GNOME 関連のパッケージ全て
gnome-accessibility　必要最小限のパッケージにスクリーンリーダなどの小物を加えた環境
gnome-core　必要最小限の環境。アプリケーションは別途導入する必要がある

等のメタパッケージをインストールします。

その他、 Debian での GNOME については、 Debian GNOME Packaging に情報が集まっているので参考にするとよいでしょう。

- Debian GNOME Packaging
  `http://pkg-gnome.alioth.debian.org/`

3.4.2　KDE – the K Desktop Environment

KDE は、 GUI ツールキットに Qt(キュート) を利用した統合デスクトップ環境で、 Lenny ではリリースのタイミングから KDE 3.5、 squeeze/sid では KDE 4.4 が収録されています。 (2010 年 7 月現在)

KDE 3 系と KDE 4 系は、ツールキットが Qt3 と Qt4 が違うほか、機能やデスクトップ自体の考え方まで変わっているので KDE 3 系が好きだった人はとまどうかもしれません。

KDE をインストールするには、タスクから「 KDE デスクトップ環境」を選ぶか、導入したいパッケージの量に合わせて

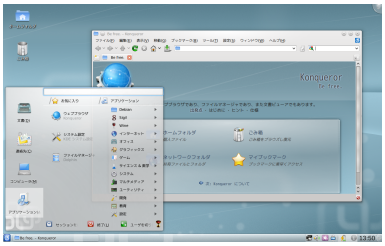


図 2　KDE 4.4 の画面

kde　KDE 環境全て (KDE プロジェクトが配布していない物も含める)
kde-core　必要最小限の環境。アプリケーションは別途導入する必要がある

等のメタパッケージをインストールします。

その他、 Debian での KDE については、 Debian KDE Maintainers のサイトに情報が集まっているので参考にするとよいでしょう。

- The Debian KDE maintainers website:
  `http://pkg-kde.alioth.debian.org/`

3.4.3　Xfce4

GNOME や KDE はそれなりにメモリを必要としますし、それなりに重いです。そこで X で利用できる軽量なデスクトップ環境の構築を目標として作成されたのが Xfce です。名前の由来は *XForms Common Environment* です。バージョン 3 までは GUI ツールキットとして XForms を使用し、商用 UNIX の CDE(Common Desktop Environment) を模していましたが、バージョン 4 以降は GUI ツールキットとして GTK+2 を使用し、それまでと雰囲気ががらりと変わりました (そんな訳で バージョン 4 以降を強調するために Xfce**4** と呼ぶ事も多いです) 。

図 3　Xfce4 の画面

同じ GTK+ を使用している GNOME と比較して (見た目も綺麗な割に) 非常に軽量に動作するのが特徴です。また、プロジェクトの公式配布物ではないものの、 Xfce Goodies と呼ばれるプラグインが続々と開発されており、非常に使い易い環境となっています。

Xfce4 をインストールするには、タスクから「 Xfce デスクトップ環境」を選ぶか、 `xfce4` パッケージおよび `xfce4-goodies` パッケージをインストールします。

その他、 Debian における Xfce に関する情報は Debian Xfce Group のサイトに情報が集まっているので参考にするとよいでしょう。

- Debian Xfce Group:
  `http://pkg-xfce.alioth.debian.org/`

3.4.4　LXDE

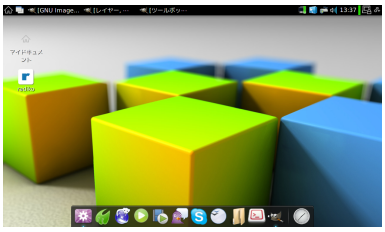


図 4　LXDE を eeePC で利用している画面 (ランチャーは GNOME Do)

LXDE は初期起動時のメモリ使用量が 100MB ほどの軽量なデスクトップ環境です。

LXDE は GNOME や KDE、 Xfce4 などの他のデスクトップ環境と比較すると、統合デスクトップ環境としての共通ライブラリなどがなく、ウィンドウマネージャに OpenBox、ファイルマネージャに PCManFM、パネルに lxpanel など軽量のアプリケーションを組み合わせ、ゆるやかな形として統合デスクトップ環境を実現しています。

Debian での LXDE パッケージは Andrew Lee さんが管理しています。

インストールについては KDE と同様、 lxde というメタパッケージが用意されているので、 aptitude で容易にインストールできます。

```
$ sudo aptitude install lxde
```

LXDE は最新版が squeeze に取り込まれる予定です。

4 次期リリースの squeeze を見てみよう

のがたじゅん

4.1 はじめに

Debian な人なら知ってることも改めて解説しながら、次期リリース予定の squeeze について **2010 年 5 月時点** の状況を解説します。

4.2 squeeze って何?

squeeze とは次期リリース予定の Debian 安定版の開発コードネームです。バージョン番号は 6.0 です。 Debian の開発コードネームは、映画 Toy Story のキャラクター名から取られ、 squeeze は 3 つ目のエイリアンです。

4.3 いつリリースなの?

2009 年 7 月に開催された DebConf 9 でリリースをタイムベース (時間で区切って) のリリースに移行することが発表されました。[*10] それによると 12 月フリーズ (新規機能などの追加停止)、翌年 2010 年 3 月にリリースの予定でした。

しかし現安定版の Lenny は、 2009 年 2 月にリリースされ、まだ一年も経っていない状況では早すぎるとの判断から、改めて 2010 年 3 月にフリーズに入る予定でした。[*11]

が、また一転。 3 月にネットワーク障害が起こったため、また仕切り直しになりフリーズは 5 月末から 6 月上旬に延期され[*12] リリースは未定になっています。

4.4 squeeze のリリースゴール

squeeze のリリース目標ですが、現在のところ 2009 年に発表されたリリースゴールから変わっていません。以下、Debian ニュースの「 Debian GNU/Linux 6.0 ”squeeze” リリースの目標」からの引用です。[*13]

- 多数のアーキテクチャサポートによる、 64 ビットマシンへの 32 ビットパッケージのインストール事情の改善
- kFreeBSD サポート、 Debian 初の non-linux アーキテクチャの導入
- dash を新しいデフォルトシェルとしてブート性能を改善し、 依存ベースのブートシステムによるブートプロセスのクリーンアップと 並行処理による性能向上を図る
- 品質保証 (QA) プロセスをさらに拡張してパッケージの品質向上につなげる。その内容:

[*10] http://lists.debian.org/debian-announce/2009/msg00009.html
[*11] http://lists.debian.org/debian-devel-announce/2009/10/msg00002.html
[*12] http://lists.debian.org/debian-devel-announce/2010/04/msg00001.html
[*13] http://www.debian.org/News/2009/20090730

  - 全パッケージについてクリーンインストール、アップグレード及び削除
  - 基礎的な品質チェックに通らなかったパッケージの自動拒否
  - ダブルコンパイルのサポート
- 新しいパッケージ形式を策定して、 将来の開発の能率化と圧縮アルゴリズムの改善を図る
- 旧式のライブラリを削除してセキュリティを改善
- ipv6 の完全サポート
- ラージファイルのサポート
- アーカイブ全体の debug パッケージの自動生成。 Google Summer of Code プロジェクトのインフラへの統合を保留しています。
- パッケージの長い説明を翻訳済みパッケージリストに分離して翻訳を促進し、 また、組み込みシステム向けに小さくしたフットプリントを提供します。 小さくなった Packages ファイルに感謝します。
- パッケージに複数の属性をタグ付けするシステム、 debtags の統合の改善によってパッケージ選択をもっと簡単に
- メンテナによりアップロードされたバイナリパッケージを破棄、再ビルドし、 制御下の環境でビルドされたパッケージだけを残す

4.5 今、やることは?

まだ (2010 年 5 月現在) フリーズになってません。[*14] 変更点があってもまだ間に合います。 (といっても大幅な変更は「 5 月 21 日までに連絡を」と言っていたので難しいかもしれませんが、相談することが重要だと思います。) それまでに Release Critical Bug(リリースに障害となるバグ) を潰すこと、翻訳を進めるなど、いろいろあります。

4.6 日本語関連で注意しなければいけないこと

まず Defoma を使わなくなったことによる影響があげられます。

Debian には Defoma(Debian Font Manager) というフォントを独自で管理する仕組みがありますが、現在メンテナンスされておらず外されることが決定しました。

デスクトップ環境では Fontconfig によるフォント管理があるので Defoma が外されても影響はありませんが、 TeX 環境、特に日本語 TeX 環境については Defoma に機能を依存していたこともあり影響が出ることが確認されています。

日本語 TeX 環境での影響は、 GhostScript で日本語 PostScript が扱えない、 dvipsk-ja がビルドできない/使えないなどあります。

TeX 以外では日本語 man では man の整形をする roff(groff) が日本語の文字に中途半端な対応のため、きちんと整形して表示されないなどの問題が確認されています。

これらの問題について、先日、対応するテストパッケージと Debian JP のアナウンスが出たので、ご協力いただける方はアナウンスを読んで、 Debian JP Devel ML で反応してください。[*15]

また、問題ではありませんが、武藤さんより uim から IBus への変更の提案が Debian JP Devel ML 出されていて、意見を求めているので、関心のある方はご意見などをお寄せください。[*16]

(2010 年 7 月追記): TeX 周りは佐々木さん、武藤さんらによって、ほぼ解消されました。 IBus の変更については uim が継続となり、議論は squeeze リリース後に持ち越しになりました。

[*14] 2010 年 6 月に 2010 年 8 月下旬フリーズ予定と発表されました。 http://lists.debian.org/debian-devel-announce/2010/06/msg00002.html

[*15] http://lists.debian.or.jp/debian-announce/201005/msg00003.html

[*16] http://lists.debian.or.jp/debian-devel/201005/msg00006.html

5　ブート方法が変わるよ

まえだこうへい

5.1　squeeze からブート方法が変わる

Debian のブートの仕組みには、 System-V 系 Unix では伝統的な init (sysvinit) が使われています。 Debian の次期安定版 6.0 (コードネーム Squeeze) から、これが upstart に変わる予定です。今年の 3 月にフリーズ予定、 8 月にリリース予定の Squeeze での予習の意味を込めて[*17]、今回はこの upstart に変わることになった背景、 init との比較を含めた upstart の仕組み、実際に切り替え方法について説明します。

5.1.1　背景

upstart は、 Debian をベースとしたディストリビューションである Ubuntu で、独自に sysvinit の後継の仕組みとして開発されました。元々 sysvinit は伝統的で安定した仕組みではありますが、現在使われているハードウェアで使うには機能面、性能面から問題が出てきています。そこで、 sysvinit の後継の仕組みとして、 upstart 以外にも複数のブートプロセスの仕組みが開発されています。しかし、 Ubuntu 以外にも Fedora 、また Google の Chrome OS, Chromium OS でも upstart が採用されています。 Debian でも squeeze から採用される予定ということもあり、 sysvinit の後継は upstart に落ち着く可能性が高いのではないでしょうか。

5.1.2　そもそも init って何よ?

upstart の話をする前に、 init って何? という人向けにまず説明しましょう。 init は今更説明しなくても知っとるがな、という読者は、先に読み進めてください。

init は Unix/Linux システムにおいて、カーネルがブートした後、ユーザプログラムが起動するための仕組みです。例えば、あなたの使っているノート PC やサーバに入っている Debian システムでもこの仕組みが必ず入っています。マシンに電源を入れてから、ログイン画面が表示されるまでの流れは大まかには図 5(ブートの流れ) のようになります。

[*17] 今回も予定通り遅れています。遅れた理由は 4.3「 次期リリースの Squeeze を見てみよう」の「 いつリリースなの? 」を参照。

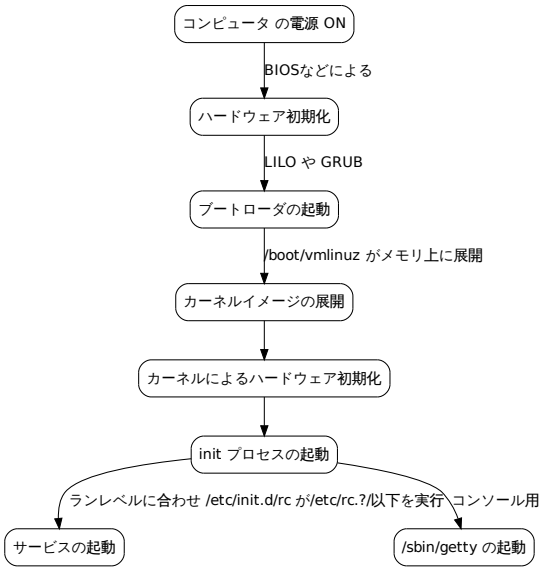


図 5　ブートの流れ

init プロセスが起動すると、 init は /etc/inittab の内容に従って、プロセスのを生成や停止を行います。 inittab の書式は次のようになります。

```
id:runlevels:action:process
```

Debian システムのデフォルトランレベルは 2 なので、プロセスの生成に関わる基本的なエントリは次の 4 行です。

```
id:2:initdefault:                       ←デフォルトランレベルは 2
si::sysinit:/etc/init.d/rcS             ←起動時は必ず実行
l2:2:wait:/etc/init.d/rc 2              ←ランレベル 2 で実行。
1:2345:respawn:/sbin/getty 38400 tty1 ← getty を常駐
```

一行目 (id が si) にランレベルの指定が無いのは、 action に sysinit が指定されているためです。これはステムブート中に実行され、他のブート用の action よりも優先して実行されます。/etc/init.d/rcS の中では

```
exec /etc/init.d/rc S
```

だけが実行されます。このワンライナーは /etc/init.d/rc でのブート用の変数を設定します。この後、上記の rc スクリプトに対し、起動時に指定するランレベルを引数として実行されますが、 Debian でのデフォルトランレベルは 2 です。

```
id:2:initdefault:
```

ですので、実際には下記が実行されます。

```
l2:2:wait:/etc/init.d/rc 2
```

とだけが実行されます。これはランレベル S のシングルユーザモードのときのものです。上記 3 行目でランレベル 2 のときに実行されるエントリがあることからも分かるとおり、

1. ランレベル S のプロセスが実行
2. ランレベル 2 のプロセスが実行

の順で起動プロセスが実行されます。ランレベル S 用の起動処理が終わってから、ランレベル 2 用の起動処理が実行されるのですから、/etc/rcS.d/ 以下と/etc/rc2.d/ 以下を比較しても分かるとおり、これが逐次実行されるのはかなり時間がかかるでしょう。ただし、同じレベルのスクリプトは並行して実行されるように改善はされています。

```
# Now run the START scripts for this runlevel.
# Run all scripts with the same level in parallel
CURLEVEL=""
for s in /etc/rc$runlevel.d/S*
do
        # Extract order value from symlink
        level=${s#/etc/rc$runlevel.d/S}
        level=${level%%[a-zA-Z]*}
        if [ "$level" = "$CURLEVEL" ]
        then
                continue
        fi
        CURLEVEL=$level
        SCRIPTS=""
        for i in /etc/rc$runlevel.d/S$level*
        do
                [ ! -f $i ] && continue

                suffix=${i#/etc/rc$runlevel.d/S[0-9][0-9]}
(snip)
                SCRIPTS="$SCRIPTS $i"
                if is_splash_stop_scripts "$suffix" ; then
                        $debug splash_stop || true
                fi
        done
        startup $ACTION $SCRIPTS
done
```

起動スクリプトの起動以外には、ランレベル 2 から 5 または、 2 か 3 の時にはコンソールから getty が実行されます。 action が `respawn` となっていますが、これは getty プログラムが終了したら、 init が再起動させるための指示です。あるユーザがコンソールからログインしたセッションを、ログアウトすると getty は終了しますが、 init により再びログイン画面で待ち受けることができる、というわけです。

```
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
3:23:respawn:/sbin/getty 38400 tty3
4:23:respawn:/sbin/getty 38400 tty4
5:23:respawn:/sbin/getty 38400 tty5
6:23:respawn:/sbin/getty 38400 tty6
```

init の他の役割としては、システム停止時のプロセスの停止にも関わっています。

5.2 upstart とは

それでは、 init についての予備知識を得たところで、本題の upstart に入りましょう。 upstart は sysvinit をイベントベースに置き換えたもので、サービスの開始と停止はイベントの通信にもとづきます。 upstart の主な特徴は次の 6 つです。

- イベントドリブンでタスクやサービスを起動・停止する。
- タスクやサービスが起動・停止することでイベントが発生する。
- イベントはシステム上の他のプロセスから受け取ることができる。
- サービスが予期せず突然終了しても再起動することができる。
- デーモンの監視と再起動は親プロセスから分離できる。
- D-Bus を通じて init デーモンと通信できる。

upstart は、 sysvinit と同様な機能を提供しますが、**非同期イベントに応じて自律的に動作する点**がもっとも異なります。そのため、 sysvinit に対する upstart のメリットには、

- 利用可能なハードウェアだけでブートするため、 runlevel が必要ない。これは存在しないハードウェアを必要とするジョブをトリガーとしないため。
- ホットプラグデバイスに対応

といったことが挙げられます。

例えば、システムがブート後、 NIC を挿すと

1. network-interface-add イベント生成

2. DHCP ジョブがネットワークカードを構成
3. network-interface-up イベントが生成
4. デフォルトルートが新しいインタフェースに割り当て
5. default-route-up イベントが生成
6. NIC を必要とするジョブ (各種サーバ) が自動的に開始される

という動きをします。逆にネットワークカードがなくなった場合は自動的に停止されます。

ただし、現在、 squeeze/sid で採用されている upstart は、 sysvinit の互換モードのものです。 upstart の最終目標は、イベントドリブンのブートプロセスに完全に移行することですが、互換モードでは、 sysvinit の動作を模倣しています。[*18]

upstart の状態遷移は図 6 のようになります。[*19]

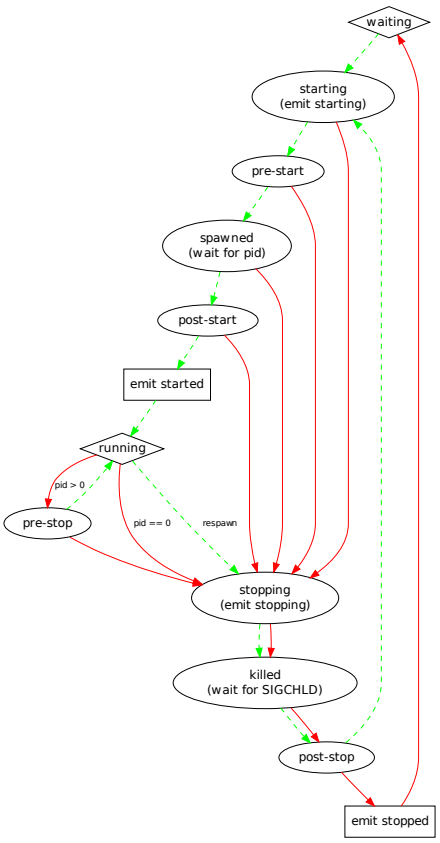


図 6　upstart 状態遷移

5.2.1　sysvinit と変わらない点

Debian の upstart は、前述のとおり、 Debian システムの起動･停止という重要な部分の置き換えを行うため互換モードのものが採用されています、下記は sysvinit と仕様上の変更がない部分です。[*20]

- initscript がインストールされる場所。パスは /etc/init.d/。
- 起動･停止用の initscript 。/etc/init.d/以下から /etc/rc?.d/ 以下に symlink が張られる。

[*18] なお、 Ubuntu 9.10 以降では ネイティブモードに移行しているようです。
[*19] upstart のドキュメントに付属のものを掲載。
[*20] なお、 5.2.1「 sysvinit と変わらない点」と 5.2.2「 sysvinit と違う点」は upstart パッケージの README.Debian.gz に記載されている FAQ をまとめ直したものです。

- 起動・停止用 initscript の順序。 SNNname, KNNname として symlink を張る。 NN は 00 から 99。 K スクリプトが最初に序数順に実行され、 S スクリプトがそのあと実行される。
- 現在および一つ前のランレベルを確認する方法。 `runlevel` コマンドを使う。
- ランレベルの変更方法。 `telinit` コマンドか `init` コマンドを実行する。
- デフォルトのランレベルの変更方法。/etc/inittab ファイルの `id:N:initdefault:` の N を書き換える。
- シャットダウンの方法。 upstart パッケージで提供される、 `shutdown` コマンドや `reboot`, `halt`, `poweroff` といったショートカットを使う。コンソールで Control-Alt-Delete を押してリブート出来る点も同じ。
- シングルユーザモードにする方法。 GRUB から **(recoveryode)** オプションを選択か、カーネルのコマンドラインで、`-s`, `S`, `single` などの引数を指定する。稼働中のマシンでは、 `telinit 1` か `shutdown now` コマンドを実行する。

5.2.2 sysvinit と違う点

互換モードでも全てが sysvinit と動作が同じというわけではなく、 upstart の固有の部分もあります。主に getty 関連の設定が変わる点が大きな違いです。

- Control-Alt-Delete による挙動の変更方法。/etc/init/control-alt-delete.conf の **exec** で始まる行を変更する。キーを押しても何も実行されないようにするには、ファイルを削除するだけ。

```
(snip)
start on control-alt-delete

task
exec shutdown -r now "Control-Alt-Delete pressed"
```

- getty を常駐させる数を減らす方法。/etc/init/ttyN.conf ファイルを変更する[*21]。必要なければファイル自体を削除する。
- getty の設定変更の反映方法。ファイルを変更したり削除してもすぐには反映されない。
  停止は `stop ttyN` コマンドを、起動は `start ttyN` コマンドを実行する。
- getty のパラメータの変更方法。/etc/init/ttyN.conf の **respawn** で始まる行を変更する。

```
# tty1 - getty
#
# This service maintains a getty on tty1 from the point the system is
# started until it is shut down again.

description     "Start getty on tty1"
author          "Scott James Remnant <scott@netsplit.com>"

start on stopped rc RUNLEVEL=[2345]
stop on runlevel [!2345]

respawn
exec /sbin/getty 38400 tty1
```

- getty を実行するランレベルの変更方法。/etc/init/ttyN.conf の次の 2 行を変更する。
  stop の**!** は否定で、 start と stop の設定は、**!**以外は基本同じにする。
- getty の数を増やす方法。/etc/init/ttyN.conf を、 ttyS0 などの名前でコピーする。 **respawn** の行に必要な設定をを記述する。
- シリアルコンソールを追加する場合は、上記の”getty の数を増やす方法” と同じ。
- upstart が動かない場合のデバッグ方法。カーネルコマンドラインに`--debug` オプションをつけ、 `quiet` と `splash` オプションがある場合はそれらを削除する。 upstart が実行されるとデバッグメッセージが出力される。[*22]。
- upstart が動かない場合のシステム復旧手順。

[*21] **N** は 1 から 6 の数字
[*22] initramfs-tools ではなく initramfs 生成ツールを使っている場合にはこのオプションを使うと既知のバグもあるので気をつけましょう。

1. カーネルコマンドラインから、 quite と splash があれば削除し、 init=/bin/bash を引数として起動すると、 root shell が起動される。
2. /etc/init.d/rcS を実行して、ハードウェアやネットワークの基本設定を行う。
3. upstart がちゃんとインストールされているか確認する。/etc/init ディレクトリに全てのファイルがインストールされているかチェックし、正常にインストールされてない場合は upstart パッケージを再インストールする。
4. /etc/init にファイルが今度はちゃんとあるか確認する。
5. sync と reboot -f コマンドを実行し、マシンを再起動する。

- upstart ジョブリストをクエリする方法。 initctl list コマンドでジョブとステータスを表示する。

```
$ sudo initctl list
tty4 start/running, process 25474
rc stop/waiting
tty5 start/running, process 25478
control-alt-delete stop/waiting
rcS stop/waiting
rc-sysinit stop/waiting
dbus-reconnect stop/waiting
tty2 start/running, process 25473
tty3 start/running, process 25475
tty1 start/running, process 25477
tty6 start/running, process 25476
```

- ジョブの起動・停止方法。 start JOB, stop JOB コマンドを実行する。
- ジョブのステータス表示方法。 status JOB コマンド。

```
$ sudo status tty1
tty1 start/running, process 25477
```

- 手動でイベントを発行する方法。 initctl emit EVENT コマンドで名前付きイベントを発行し、待機中のジョブが状況に応じて起動 or 停止する。

5.3 upstart への切り替え

それでは、早速 sysvinit から upstart へ切り替えてみましょう。 squeeze/sid と Lenny との場合を見てみます。

5.3.1 squeeze/sid での場合

squeeze/sid での upstart への切り替えには、 upstart パッケージをインストールします。通常のパッケージのインストールとは異なり、続行する場合は、 **Yes, do as I say** と入力しなさい、というメッセージが表示されます。これは入れ替えは非常にリスクが高いためです。

```
$ sudo apt-get install upstart
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています
状態情報を読み取っています... 完了
以下の特別パッケージがインストールされます:
  dbus libdbus-1-3 libexpat1
提案パッケージ:
  dbus-x11
以下のパッケージは「削除」されます:
  sysvinit
以下のパッケージが新たにインストールされます:
  dbus libdbus-1-3 libexpat1 upstart
警告: 以下の不可欠パッケージが削除されます。
何をしようとしているか本当にわかっていない場合は、実行してはいけません!
  sysvinit
アップグレード: 0 個、新規インストール: 4 個、削除: 1 個、保留: 9 個。
1,005kB のアーカイブを取得する必要があります。
この操作後に追加で 2,105kB のディスク容量が消費されます。
重大な問題を引き起こす可能性のあることをしようとしています。
続行するには、'Yes, do as I say!' というフレーズをタイプしてください。
 ?] Yes, do as I say!
```

lxc の環境で試してみましたが、 getty がうまく動かず、起動しては突然死して、再起動されて、また突然死、というの

を繰り返してしまうので、コンソールからのログインは出来ない状態でしたが[*23]、 6.1「 upstart 再入門」で KVM の環境下で試した際には正常に起動することを確認済みです。

```
$ sudo lxc-start -n bootsid
cat: /proc/cmdline: No such file or directory
Setting the system clock.
Cannot access the Hardware Clock via any known method.
Use the --debug option to see the details of our search for an access method.
Unable to set System Clock to: Tue Feb 9 14:16:26 UTC 2010 ... (warning).
Activating swap...done.
mount: you must specify the filesystem type
Cannot check root file system because it is not mounted read-only. ... failed!
Setting the system clock.
Cannot access the Hardware Clock via any known method.
Use the --debug option to see the details of our search for an access method.
Unable to set System Clock to: Tue Feb 9 14:16:27 UTC 2010 ... (warning).
Cleaning up ifupdown....
Checking file systems...fsck from util-linux-ng 2.16.2
done.
Setting up networking....
Mounting local filesystems...done.
Activating swapfile swap...done.
Cleaning up temporary files....
Configuring network interfaces...done.
Setting kernel variables ...done.
Cleaning up temporary files....
Starting system message bus: dbus.
Starting OpenBSD Secure Shell server: sshd.
init: tty4 main process (239) terminated with status 1
init: tty4 main process ended, respawning
init: tty5 main process (241) terminated with status 1
init: tty5 main process ended, respawning
init: tty2 main process (242) terminated with status 1
init: tty2 main process ended, respawning
init: tty3 main process (244) terminated with status 1
init: tty3 main process ended, respawning
init: tty6 main process (245) terminated with status 1
init: tty6 main process ended, respawning
init: tty1 main process (306) terminated with status 1
init: tty1 main process ended, respawning
init: tty4 main process (307) terminated with status 1
init: tty4 main process ended, respawning
(snip)
```

この時も、 ssh 経由のターミナルログインは問題なくできました。

```
$ ssh bootsid
Enter passphrase for key '/home/user/.ssh/id_rsa':
Linux bootsid 2.6.32 #1 SMP Mon Dec 7 05:27:50 UTC 2009 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Feb  9 14:18:38 2010 from 192.168.189.114
user@bootsid:~$
```

5.3.2 Lenny での場合

squeeze/sid でもうまく行っていない状況ですので、 squeeze が stable としてリリースされるときに、検討しましょう[*24]。あるいは、 sid にアップグレードすることを検討しても良いでしょう。

5.3.3 まとめ

現時点では、実際に切り替わった際にはオペレーション上も多少変更があります。 Ubuntu 9.10 で採用されている ネイティブモードでは更にオペレーションも変わるようですので、今回はじめて upstart を知ったという方は、次章も参考の上、早めに使い方や仕組みを予習し、バグ出しに協力されると Debian の品質も向上することでしょう。

[*23] 2010 年 2 月 9 日現在

[*24] upstart が予定どおり squeeze に含まれていることが前提ですが。

6 upstart 再入門

まえだこうへい

6.1 はじめに

今年の 2 月の Debian 勉強会(Debian 温泉) [*25]で一度扱ったテーマです。今回は起動速度の観点から変化があるかを見てみましょう。[*26]

6.1.1 事前準備

前章とは異なり、検証のための環境は KVM で用意し、起動速度の比較には bootchart を用いました。用意するインストールイメージと、パッケージは下記の通りです。

- debian-testing-amd64-businesscard.iso
- qemu-kvm パッケージ
- qcow2 フォーマットのディスクイメージ
- bootchart パッケージ

KVM/QEMU ゲストには、 sysvinit と upstart の 2 種類を用意しました。

- Debian GNU/Linux sid の最小構成をインストール
- インストール後、ディスクイメージからコピー作成
- コピー後、 upstart パッケージをインストール

6.1.2 upstart のインストール

前章でも触れた部分ですが、再掲します。切り替えには upstart パッケージをインストールします。通常のパッケージのインストールとは異なり、続行する場合は、 **Yes, do as I say** と入力しなさい、というメッセージが表示されます。これはうまくいかない場合は Debian システムが起動できなくなるなどのリスクがあるためです。

[*25] 本書では前章にあたります。

[*26] 4 月時点では、書き下ろすつもりでしたが、個人的な都合で余裕がなく、今回の事前配布資料は基本的に、 2010 年 2 月の Debian 勉強会の資料「 ブート方法が変わるよ」を再編しただけだったので、起動速度の検証について当日発表のみ行いました。

```
$ sudo apt-get install upstart
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています
状態情報を読み取っています... 完了
以下の特別パッケージがインストールされます:
  dbus libdbus-1-3 libexpat1
提案パッケージ:
  dbus-x11
以下のパッケージは「削除」されます:
  sysvinit
以下のパッケージが新たにインストールされます:
  dbus libdbus-1-3 libexpat1 upstart
警告: 以下の不可欠パッケージが削除されます。
何をしようとしているか本当にわかっていない場合は、実行してはいけません!
  sysvinit
アップグレード: 0 個、新規インストール: 4 個、削除: 1 個、保留: 9 個。
1,005kB のアーカイブを取得する必要があります。
この操作後に追加で 2,105kB のディスク容量が消費されます。
重大な問題を引き起こす可能性のあることをしようとしています。
続行するには、'Yes, do as I say!' というフレーズをタイプしてください。
 ?] Yes, do as I say!
```

2 月の Debian 勉強会時点[*27]では、 lxc の環境で試した際[*28]は、 getty がうまく動かず、起動しては突然死して、再起動されて、また突然死、というのを繰り返してしまい、コンソールからのログインは出来ない状態でしたが、今回の資料作成時点[*29]では、 KVM 環境で問題なく起動します。バージョンが変わってませんが、環境が異なるので原因は後者にありそうではあります。

6.1.3 起動速度を比べてみる

プロセスを並行処理させることで処理速度を向上させるので、プロセスが多いほど、効果は高いはずではないか、とか考えられます。

比較のために用意した環境は以下の通りです。

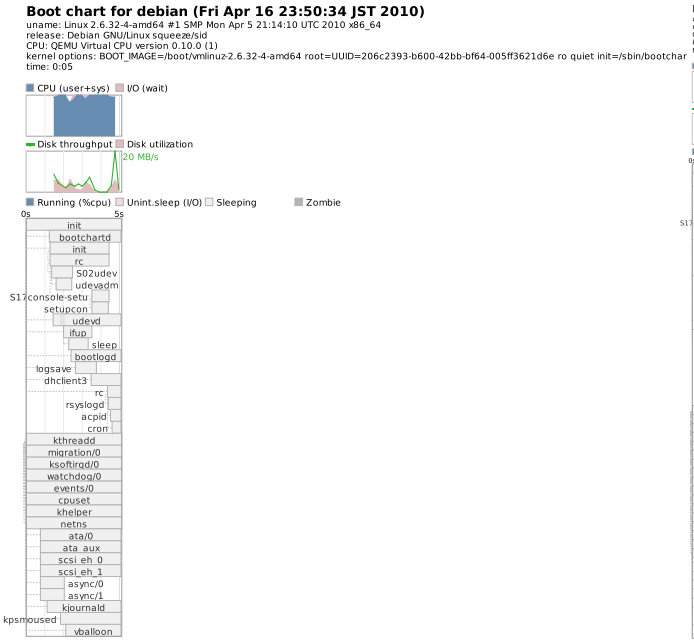
- ブートの種類
  - sysvinit と upstart
- 構成
  - 最小構成 base に bootchart をインストール
  - CouchDB 最小構成に couchdb をインストール
  - LAMP CouchDB の構成に apache2,rails, mysql-server をインストール
  - GNOME 最小構成に xserver-xorg, gnome をインストール

[*27] 本書では前章

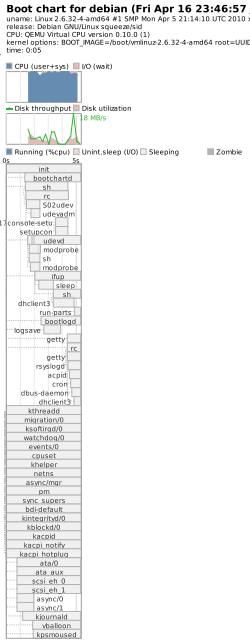
[*28] 2010 年 2 月 9 日現在

[*29] 2010 年 4 月 11 日に、 Squeeze Official Snapshot amd64 BC Binary-1 20100322-03:30 の ISO イメージを使用した最小構成。

6.1.4 bootchart の結果

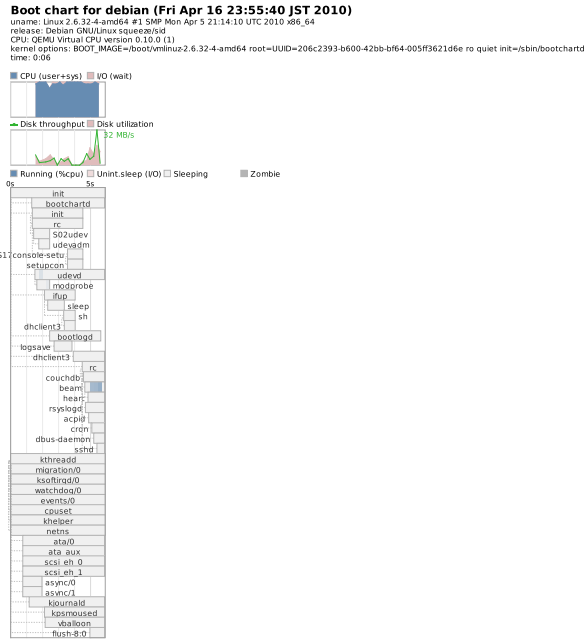


(a) sysvinit

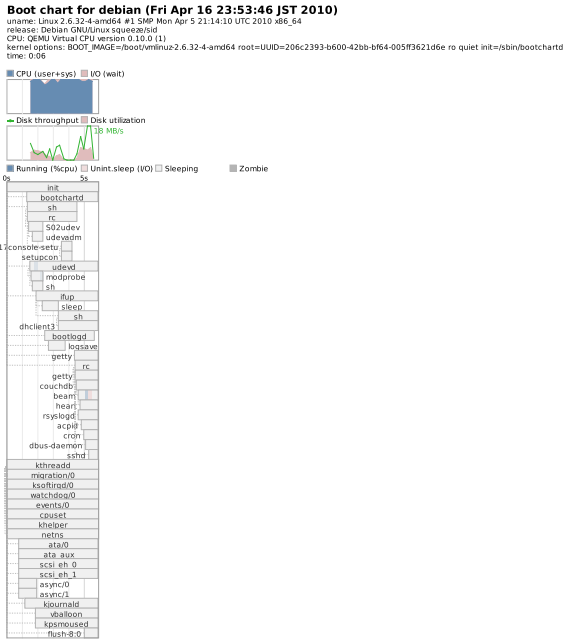


(b) upstart

図 7　最小構成

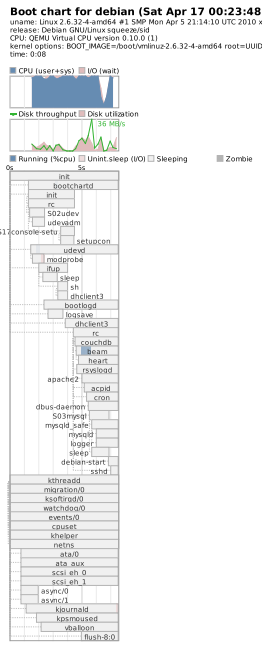


(a) sysvinit

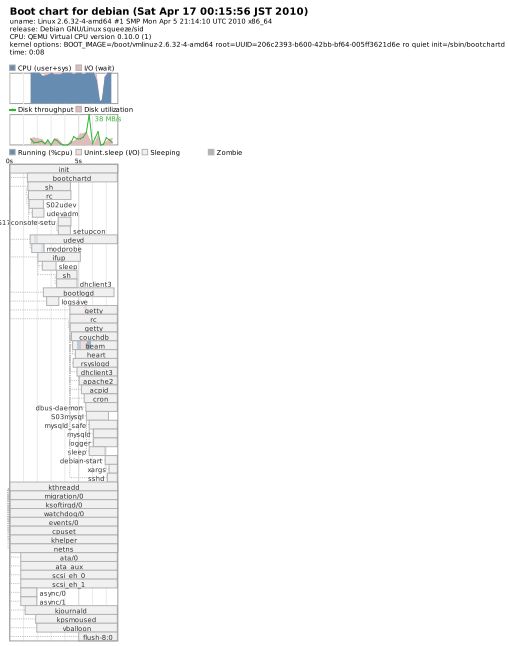


(b) upstart

図 8　CouchDB

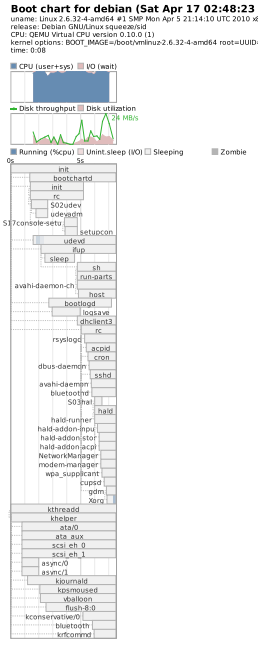


(a) sysvinit

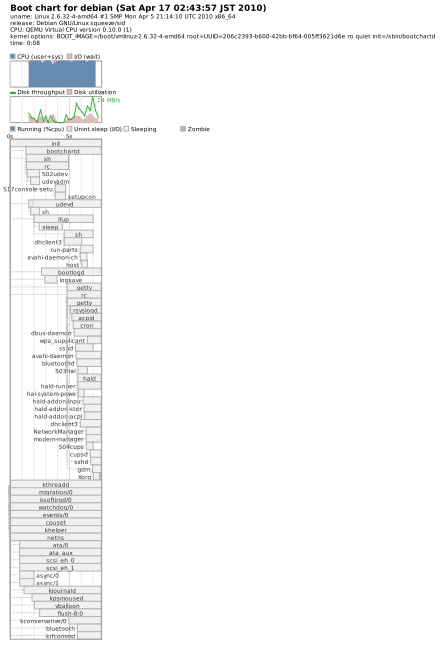


(b) upstart

図 9　LAMP



(a) sysvinit



(b) upstart

図 10　GNOME

6.1.5 結果

起動速度調査結果を表 1 に示します。

表 1　起動速度調査結果

init の種類	最小構成	CouchDB	LAMP	GNOME
sysvinit	5 sec	6 sec	8 sec	8 sec
upstart	5 sec	6 sec	8 sec	8 sec

起動時間にはまったく違いがありませんでした。が、本書を読んでいる読者の皆さんは既にお気づきのはず。 Debian の upstart は互換モードなので、 sysvinit の動作を模倣しているのでした。これでは比較になりませんね。

6.2 Ubuntu 9.10 を試してみた。

そこで、 Debian とは若干構成が変わりますが、ネイティブモードの upstart になっている Ubuntu 9.10 で試してみることにしました (図 11)。違いとしては、以下のようになりました。

- init の処理終了時で bootchart のログ収集を止めるわけではないようだが、実質的には 25 秒程度で起動が完了。
- でも起動プロセスが並行処理されていること Debian での結果を比べても分かります。

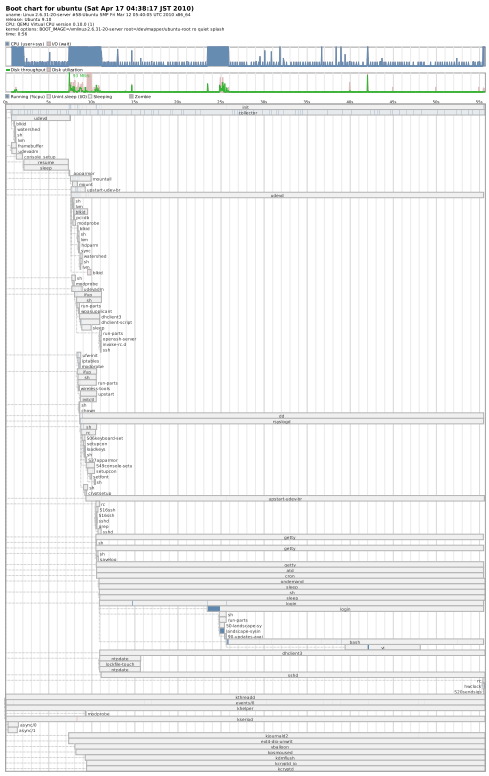


図 11　Ubuntu 9.10 の upstart

6.3 まとめ

ネイティブモードじゃないと upstart は本来のメリットが活かせない感じということが分かりました。とはいえ、いきなりネイティブモードの upstart に切り替えるのはリスクあるので、一時的に互換モードを使うのは止むを得ないのかも

しれません。ただ、 Squeeze でずっと互換モードなのもどうかと思います。 SqueezeAndAHalf リリース時 (リリースされるなら、ですが) にネイティブモードに切り替えられるとうれしいかもしれません。

結論としては、「 みんなで試してバグ出ししましょう」というところです。

6.4 参考資料

http://www.ibm.com/developerworks/jp/linux/library/l-boot-faster/index.html

7 Xen で作る自宅サーバ

川江 浩

7.1 はじめに

近年、 CPU の性能がパワフルになるにつれて、高価なハードウェアを前提とした仮想化技術がパーソナルベースでも使えるようになってきました。

そこで、脚光を浴びてきた仮想化技術の代表格である Xen を使って、インタネット関連のサーバ群を構築しましたので、 Debian ベースで Xen の仮想サーバを構築する時に注意することや、上記サーバ群を構築する際に思ったことをレポートします。

また、以下の仕様はパーソナルベースでの運用を前提に構築したものです。仕様を試そうとするときは、必ずデータ等のバックアップをとって自己責任で行ってください。より詳しく知りたい方は専門書を参照してください。

7.2 Xen とは

Xen は、仮想マシンソフトウェアの一つで、 OS より 1 つの下の階層でハイパーバイザというプログラムを動かすものです。このタイプはハイパーバイザ型と呼ばれ、「 VMware Infrastrucure」などがあります。

他方、仮想マシンソフトウェアにはアプリケーションタイプと呼ばれるものがあり、「 VMware Workstation」「 VirtualBox」「 QEMU」が有名です。

7.3 Xen の特徴

Xen は仮想化するためのモデルとして、準仮想化と完全仮想化の 2 つを提供しています。

- 準仮想化 (ParaVirtualization)
  Xen での準仮想化はハードウェアをエミュレートする代わりに、仮想マシン用のハードウェアを使用します。このハードウェアは操作をするためにハイパーバイザコールを呼び出します。ハイパーバイザコールは仮想マシン環境に対応し、 OS は Xen 仮想ハードウェア用に修正する必要があります。
- 完全仮想化 (FullVirtualization)
  Xen は完全仮想化機能も提供しています。この機能を利用すると、デフォルトの OS をそのまま Xen 上で動作させることができます。

7.4 Xen の形態

Xen は Linux をベースに作られていますので、 Xen 用にコンパイルされたカーネルを利用します。このカーネルは起動時にハイパーバイザをロードし、その上にカーネルをロードします。イメージ的にはハイパーバイザ上を管理 OS の

DomainO が動き、その OS に管理される形でゲスト OS と呼ばれる DomainU が動きます。

- Domain-O（ 管理 OS）以下 DomO
  Xen を起動した OS。ハードウェアを管理し、ハイパーバイザ上で動作するゲスト OS の管理を行う。
- Domain-U（ DomU-準仮想化）以下 DomU
  Domain-O によって起動、管理されるゲスト OS。特に、準仮想化で動作する。
- HVM Domain（ HVM-完全仮想化）以下 HVM
  Domain-O によって起動、管理されるゲスト OS であるが、完全仮想化である HVM(Hardware Virtual Machine) で動作する。

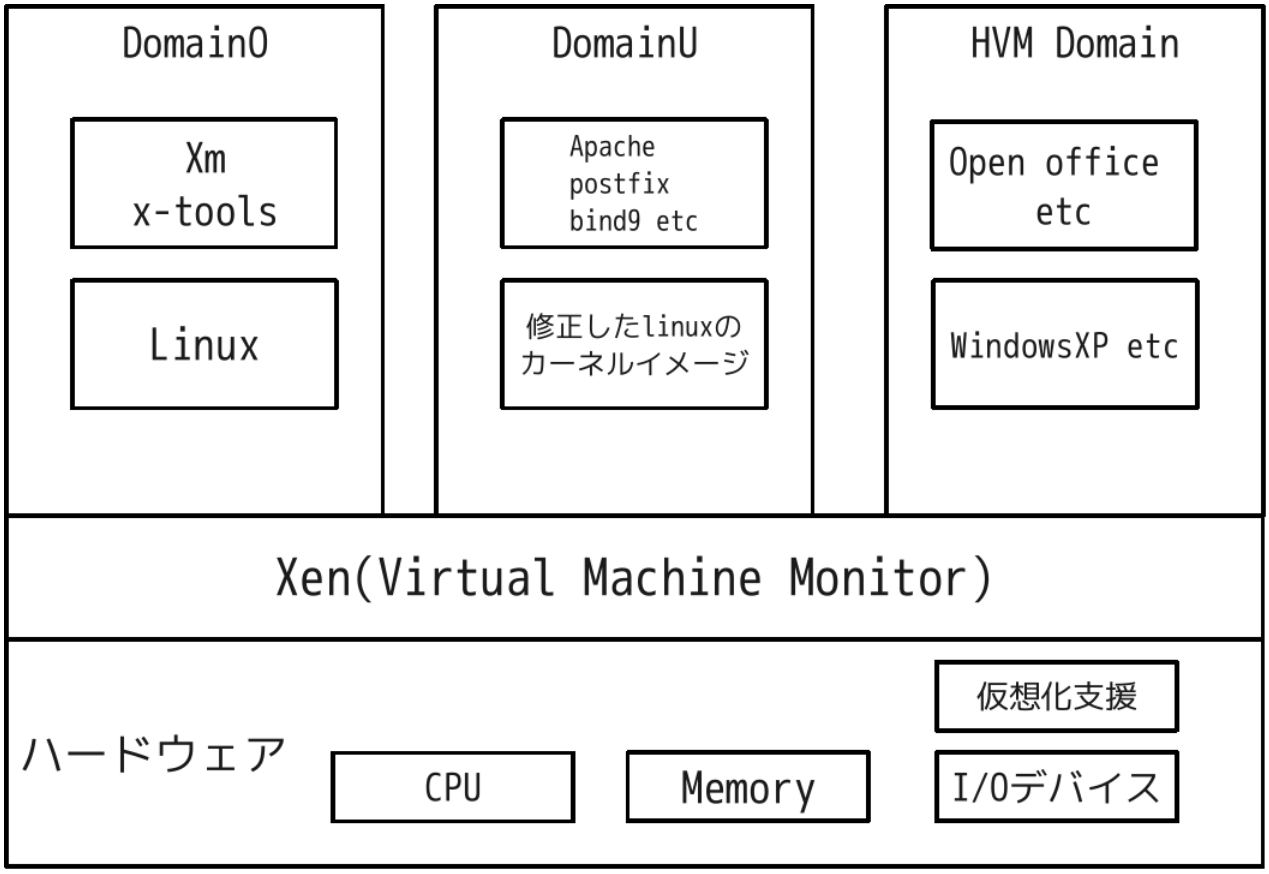


図 12　Xen のハイパーバイザモデルのイメージ

7.5 Xen の導入

次に, Xen をインストールします. インストールは各アーキテクチャによって異なります. [*30] ここでは, インテルをベースに Lenny の Xen カーネルイメージ 2.6.26 を以下の様にインストールします.

```
# aptitude install xen-linux-system-2.6.26-2-xen-686
```

また, Debian には DomU を作るツールも用意してありますので, これもインストールします.

```
# aptitude install xen-tools
```

各インストールが済んだら, /xen/xen/xend-config.sxp ファイルの以下の箇所を変えます.

```
(network-script 'network-bridge netdev=eth1')
(network-script 'network-bridge netdev=eth0')
```

再起動し, DomO の起動を確認します.

```
# xm list
Name                ID   Mem  VCPUs   State   Time (s)
Domain-0            0   1478     1   r-----    217.6
```

7.6 DomU の設定

Xen のツールを使ってゲスト OS を以下の手順で入れます. (etch や Ubuntu, CentOS も可能).

1. 設定ファイルの編集
2. xen-create-image の実行
3. DomU の起動

7.6.1 設定ファイルの編集

設定ファイルは, /etc/xen-tools/xen-tools.conf です. 以下, DomU に Lenny をインストールものとして編集します.

[*30] 詳しくは, 仮想化技術 Xen -概念と内部構造などを参照してください. 完全仮想化を目的にするのであれば Intel-VT や AMV-V が, CPU で仮想化支援機能を持っています.

```
##
#  /etc/xen-tools/xen-tools.conf
##            (中略)
#  Output directory for storing loopback images.
#
#  If you choose to use loopback images, which are simple to manage but
# slower than LVM partitions, then specify a directory here and uncomment
# the line.
#
#  New instances will be stored in subdirectories named after their
# hostnames.
#
##
dir = /home/xen (イメージファイルの保管場所です)
#             (中略)

##
#  Disk and Sizing options.
##
size   = 4Gb      # Disk image size.
#memory = 128Mb    # Memory size
memory = 384Mb    # Memory size
#swap   = 128Mb    # Swap size
swap   = 512Mb    # Swap size
# noswap = 1      # Don't use swap at all for the new system.
fs     = ext3     # use the EXT3 filesystem for the disk image.
#dist   = etch     # Default distribution to install.
dist   = lenny     # Default distribution to install.

#  Currently supported and tested distributions include:
#
# via Debootstrap:
#
#  Debian:
#   sid, sarge, etch, lenny.(他のディストリビューションの選択も可能)
#
#  Ubuntu:
#   edgy, feisty, dapper.
#
# via Rinse:
#   centos-4, centos-5.
#   fedora-core-4, fedora-core-5, fedora-core-6, fedora-core-7

##
# Networking setup values.
##
#
## Uncomment and adjust these network settings if you wish to give your
# new instances static IP addresses.
#
# gateway   = 192.168.1.1
gateway   = 192.168.0.1
# netmask   = 255.255.255.0
netmask   = 255.255.255.0
# broadcast = 192.168.1.255
broadcast = 192.168.0.255
#(ネットワークはご自由に)

#(以下はデフォルトにしました)
# Default kernel and ramdisk to use for the virtual servers
#
kernel      = /boot/vmlinuz-`uname -r`
initrd      = /boot/initrd.img-`uname -r`

#  The architecture to use when using debootstrap, rinse, or rpmstrap.
#
#  This is most useful on 64 bit host machines, for other systems it
# doesn't need to be used.
#
# arch=[i386|amd64]
#

# The default mirror for debootstrap to install Debian-derived distributions
#
mirror = http://ftp.jp.debian.org/debian/

#  If you're using the lenny or later version of the Xen guest kernel you will
# need to make sure that you use 'hvc0' for the guest serial device,
# and 'xvdX' instead of 'sdX' for serial devices.
#
#  You may specify the things to use here:
#
serial_device = hvc0 #default
# serial_device = tty1
#
disk_device = xvda #default
# disk_device = sda

#  Here we specify the output directory which the Xen configuration
# files will be written to, and the suffix to give them.
#
#  Historically xen-tools have created configuration files in /etc/xen,
# and given each file the name $hostname.cfg.  If you want to change
# that behaviour you may do so here.
#
# output    = /etc/xen
# extension = .cfg
#
```

7.6.2 xen-create-image の実行

次に, DomU のイメージを作ります. 同時に DomU に割り当てる IP アドレスをオプションで指定します. 例えば, アドレスを 192.168.0.2, ホストネームを dns とするなら以下のようにします.

```
# xen-create-image --ip 192.168.0.2 --hostname dns
```

DomU の制作には, イメージディスクの大きさやネットワークの状況によって異なりますが, 自分の環境では 10G のイメージで 30 分ぐらいでした.

7.6.3 DomU の起動

無事に, インストールができたら起動して, 稼働状況を見てみましょう. [*31]

```
# xen create -c dns.cfg
# xm list
 Name                                        ID   Mem VCPUs      State   Time (s)
 Domain-0                                     0  1478     4     r-----    429.8
 dns                                          5   384     1     -b----     10.5
 mail                                         2   384     1     -b----    123.6
 www                                          4  1792     1     -b----     23.1
```

起動してくる画面は, 全くの初期状態でログインプロンプトしか出ません. root でログインしてパスワードとユーザを作成します.

```
# passwd
# adduser ipv6waterstar
```

7.6.4 バックアップ, 他

また, DomU をデフォルトでインストールした場合, DomO の/home/xen/domain に各 DomU のドメインごとにメージファイルが置かれます. また, 設定ファイルは/etc/xen に ".cfg" ファイルとして保存されます.

従って, 例えば何らかの設定ミスをして DomU が起動不能になっても, 上記のイメージと設定ファイルのバックアップがあれば, 各ディレクトリと設定ファイルをそのままコピーし直すだけで, 同じ環境の DomU を復元できます.

また, 管理用の DomO はセキュリティの関係からプロセス数が少ない方がいいのですが, 後述のように設定ファイルを多数, 作成する場合のことも考えると, GUI で操作ができるなどの利点から, gnome などをインストールすることを勧めます.

7.7 Xen のネットワークの概要

Xen は仮想インターフェイスをベースにしたネットワーク機能を持っています.

具体的には, DomU の各ホストを直接外部ネットワークに接続するブリッジ経由の接続. 仮想インターフェイスを通して, DomO のルーティングし, ネットワークインターフェイスに出力するブリッジを経由しない接続 (NAT 接続) の二つの形態があります.

ブリッジ経由の接続のイメージはハードウェア上に, DomO と複数の DomU の仮想 PC があって, それぞれが対等にネットワークハブで繋がっているような状態です.

今回は, 各 DomU サーバをインターネットサーバとして運用したいので, 仮想インターフェイスを使って直接, セグメントが異なる外部ネットワークに接続できるブリッジ経由の接続でネットワークを構成します.

DomU は仮想マシンを作成するときに, IP アドレスを割り当てたので特別な設定は必要ありません. 同時に, Mac アドレスも各仮想インターフェイスごとに自動的に割り当てられます.

また, IP アドレスを後から変更したいときなどは/etx/xen 以下の".cfg" ファイルを書き換えます.

例 dns.cfg

[*31] Xen には管理用ツールとして「 xm 」などがあります. 詳しくは, Xen 徹底入門などを参照してください.

```
#
# Configuration file for the Xen instance www, created
# by xen-tools 3.9 on Tue Nov 17 10:35:03 2009.
#

#
#  Kernel + memory size
#
kernel      = '/boot/vmlinuz-2.6.26-2-xen-686'
ramdisk     = '/boot/initrd.img-2.6.26-2-xen-686'
memory      = '384'(メモリーの容量の変更も可能)

#
#  Disk device (s).
#
root        = '/dev/xvda2 ro'
disk        = [
                  'file:/home/xen/domains/www/swap.img,xvda1,w',
                  'file:/home/xen/domains/www/disk.img,xvda2,w',
              ]


#
#  Hostname
#
name        = 'dns'(ホスト名)

#
#  Networking
#
vif         = [ 'ip=192.168.0.2,mac=00:12:34:56:78:9A' ]
(アドレスの変更も可能ですが, DomU の interfaces を書き換えているときはそちらも書き換えてください)

#
#  Behaviour
#
on_poweroff = 'destroy'
on_reboot   = 'restart'
on_crash    = 'restart'
```

7.7.1 各インターネットサーバの設定する前の注意事項

次に, DNS, Mail サーバ, Web サーバのパッケージを, 各 DomU にインストールします.

インストールは, 仮想マシンでもノーマルのインストールと変わりません. ただ, Xen のネットワーク構成は独特の「癖」のようなものがあります. 以下, いくつか例を挙げます.

1. NTP を使った時間の設定
2. SSH でのログイン
3. その他

7.7.2 NTP を使った時間の設定

DomU は Dom0 からのみ時刻を更新できるという Xen の仕様なので, Dom0 で時刻を合わせていれば, DomU も正確な時刻を得ることができます. ただ, DomU で ntpd や ntpdate を実行するのであれば, 時刻同期できないことがあります.

普通に DomU で時計を合わせるのであれば, Aisa/Tokyo に合わせることで日本時間にできます (デフォルトでは UTC).

```
# dpkg-reconfigure tzdata
# date
  Tue Jan 24 15:00:00 JST 2010
```

また, DomU で NTP 等を使うのであれば, DomU の/etc/sysctl.conf に `xen.independent_wallclock=1` を加えて再起動してください.

7.7.3 SSH でのログイン

Xen で SSH を使って, ネットワーク越しにログインする場合も通常と同じでが, インストールしたばかりの DomU は何も入っていないので, そのままではエラーになります.

具体的には, まず DomU に SSH をインストールします (ポート番号の変更などはお好みで).

```
# aptitude install ssh
```

次にローカルから SSH を使ってログインしようとしても, 以下のエラーがでます.

```
$ ssh mail.kinsen.gr.jp
  ipv6waterstar@dns.kinsen.gr.jp's password:
  PTY allocation request failed on channel 0
```

原因は, xen-tools を使った DomU の構築で「 udev をインストールしないため」に起こります. 従って, DomU に udev をインストールすることで解決します.

```
# aptitude install udev
```

7.7.4 その他

その他, Xen でサーバを運用するときにいくつか気づいたものを挙げます.

まず, DomO と DomU がブリッジ経由で接続している場合, iptables で FORWARD を使うと DomU からネットワークに繋がらない現象が起こります (理由は不明).

また, 完全仮想化でゲスト OS を動かそうとするのであれば, CD をゲスト OS からマウントし, VNC などを立ち上げてインストールする必要があります (検証はしてませんので, 可能かどうかは不明).

詳しくは, 以下のドキュメント等を参照してください.

- http://wiki.debian.org/Xen

7.8 各インターネットサーバの設定

次に, 各インターネットサーバの設定の手順を説明します. ここでの環境は, グローバル IP アドレスが 1 つで, インターネットには電話会社から提供されているルータで外部ネットワークに, 繋がっていることを前提とします.

7.8.1 DNS の概要

DNS サーバはホスト名と IP アドレスを対応させたゾーンと呼ばれるファイルを持ち, ホスト名の検索に対して対応する IP アドレスを返します. また, ゾーンファイルの内容を他のサーバに転送します.

具体的に www.kinsen.gr.jp というホスト名に対応する IP アドレスを (再帰) 検索する場合, まず, jp ドメインの IP アドレスを問い合わせる必要があります. そのために, ルートサーバで jp ドメインの IP アドレスを検索し, jp サーバに接続します. 同様に, jp に属する gr ドメインの IP アドレスを検索し, さらには gr ドメインに属する kinsen そして www と順次, 各 IP アドレスを検索し, 各サーバに接続していきます. そして最終的に, www.kinsen.gr.jp の IP アドレスが 203.141.158.41 である事がわかります.

Debian には DNS サーバとして bind9 のパッケージが用意されています. 以下, インストールと設定を行います.

7.8.2 DNS の設定

bind9, bind9utils パッケージをインストールします.

```
# aptitude install bind9
# aptitude install bind9utils
```

今回はパーソナルユースなので 1 個のグローバル IP アドレスを使うことを前提とし, ネームサーバの設定には view や match-client を使います (例として, 内部ゾーンで, 192.168.0.0/24 ネットワーク内に各サーバを, グローバルアドレスとして 203.141.158.41 を使うものとします).

view は指定の IP アドレスやネットワークごとに, 個別のオプションとゾーンデータを提供します.

具体的には, acl で IP アドレスとネットワークを指定し, match-clients で acl にマッチするクライアントと, その他のクライアントごとに別々ゾーンを提供します. これを利用し, 1 台のサーバで内部用 (acl にマッチする) と外部用 (acl に

マッチしない) の DNS を構築します.

なお, グローバル IP アドレスが複数の場合は, 各 DomU にグローバル IP アドレスを設定してください. [*32]

7.8.3 view の設定

通常, BIND で view を設定する場合, named.conf に view や acl, match-clients の設定を書き加えます. だだ, Debian では, named.conf.options や named.conf.local などのファイルがありますので, それらを利用します.

まず, named.conf に以下のように書き換え, include を使って設定ファイルを挿入するようにします.

```
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind9/README.Debian.gz for information on the
// structure of BIND configuration files in Debian, *BEFORE* you customize
// this configuration file.
//
// If you are just adding zones, please do that in /etc/bind/named.conf.local
include "/etc/bind/named.conf.acl";  //view を使うアドレスの範囲を設定する
include "/etc/bind/named.conf.options";  //BIND の調整ためのオプション
view "internal"{  // 内部用ゾーンの設定
        match-clients { localnet; };  // 内部ゾーンの適用範囲の設定
        recursion yes;
include "/etc/bind/named.conf.conf";  // 内部用ゾーンの初期設定
include "/etc/bind/named.conf.local";  // 内部用ゾーンのローカル設定
};
view "external" {  // 外部用ゾーンの設定
        match-clients { any; };  // 外部ゾーンの適用範囲の設定
        recursion no;
include "/etc/bind/named.conf.view"; // 外部用ゾーンのローカル設定
};
```

7.8.4 named.conf.acl の設定

acl ステートメントでは, アドレスマッチリストを設定します.

named.conf.acl の内容は以下の通り

```
acl localnet{
        192.168.0.0/24;
        127.0.0.1;
};
```

7.8.5 named.conf.options の設定

options ステートメントでは, BIND の動作に関わるもろもろのオプションを設定します.

named.conf.options の内容は以下の通り

```
options {
        directory "/var/cache/bind";

        // If there is a firewall between you and nameservers you want
        // to talk to, you may need to fix the firewall to allow multiple
        // ports to talk.  See http://www.kb.cert.org/vuls/id/800113

        // If your ISP provided one or more IP addresses for stable
        // nameservers, you probably want to use them as forwarders.
        // Uncomment the following block, and insert the addresses replacing
        // the all-0's placeholder.

        forwarders {
                123.456.789.001; 123.456.789.002;
            //ISP で指定された DNS に代理問い合わせを要求するアドレスを記入します.
         };

        allow-query { any; };  // 問い合わせ可能なホストを無制限に
        allow-transfer { localnet; };  // 転送先を限定
        version "no version";  // バージョン表示を無効に
        auth-nxdomain no;    # conform to RFC1035
        listen-on-v6 { any; };

};
```

7.8.6 named.conf.conf の設定

このファイルはもとの named.conf (初期設定) ファイルです.

[*32] 詳しくは, DNS BIND 第 5 版などを参照してください.

named.conf.conf の内容は以下の通り.

```
// prime the server with knowledge of the root servers
zone "." {
        type hint;
        file "/etc/bind/db.root";
};

// be authoritative for the localhost forward and reverse zones, and for
// broadcast zones as per RFC 1912

zone "localhost" {
        type master;
        file "/etc/bind/db.local";
};

zone "127.in-addr.arpa" {
        type master;
        file "/etc/bind/db.127";
};

zone "0.in-addr.arpa" {
        type master;
        file "/etc/bind/db.0";
};

zone "255.in-addr.arpa" {
        type master;
        file "/etc/bind/db.255";
};
```

7.8.7 named.conf.local の設定

ローカルネット用の初期設定ファイル.

named.conf.local の内容は以下の通り

```
//
// Do any local configuration here
//
zone "kinsen.gr.jp" {
        type master;
        file "/etc/bind/db.in-kinsen.gr.jp"; // 内部順引きゾーンテーブル
};
zone "0.168.192.in-addr.arpa" {
        type master;
        file "/etc/bind/db.192.168.0"; // 内部逆引きゾーンテーブル
};
// Consider adding the 1918 zones here, if they are not used in your
// organization
//include "/etc/bind/zones.rfc1918";
```

以下は各ゾーンファイルの設定例です.

1. 順引きゾーンテーブル

```
;
; BIND data file for kinsen.gr.jp
;
$TTL        86400
@       IN      SOA     dns.kinsen.gr.jp. root.dns.kinsen.gr.jp. (
                        1               ; Serial
                     1800               ; Refresh
                      900               ; Retry
                   604800               ; Expire
                     1200 )         ; Negative Cache TTL

                IN      NS      dns

; localhost
localhost       IN      A       127.0.0.1
localhost       IN      AAAA    ::1

; Mail exchange
                IN      MX   0  mail.kinsen.gr.jp.
;
; Host entry
;
noren           IN      A       192.168.0.1
;               IN      AAAA    2001:
dns             IN      A       192.168.0.2
;               IN      AAAA    2001:
mail            IN      A       192.168.0.3
;               IN      AAAA    2001:
www             IN      A       192.168.0.4
;               IN      AAAA    2001:
;
; Alias
;
;www            IN      CNAME   noren
;
; Domain
@               IN      A       192.168.0.2
                IN      MX 0    mail
```

2. 逆引きゾーンテーブル

```
;
;BIND data file for 219.117.222 network
;
$TTL    86400
@       IN      SOA     dns.kinsen.gr.jp. root.dns.kinsen.gr.jp. (
                              1         ; Serial
                           1800         ; Refresh
                            900         ; Retry
                         604800         ; Expire
                           1200 )       ; Negative Cache TTL

                IN      NS      dns
;
; Host entry
;
1                IN      PTR     noren.kinsen.gr.jp.
2                IN      PTR     dns.kinsen.gr.jp.
3                IN      PTR     mail.kinsen.gr.jp.
4                IN      PTR     www.kinsen.gr.jp.
```

7.8.8 named.conf.view の設定

グローバルネット用の初期設定ファイルです. named.conf.view の設定は以下の通り

```
zone "." {
        type hint;
        file "/etc/bind/db.root";
};
zone "kinsen.gr.jp"{
        type master;
        file "/etc/bind/db.out-kinsen.gr.jp"; // 外部順引きゾーンテーブル
        allow-transfer{
                localnet;
                123.456.789.001;
                123.456.789.002;
        };
};
zone "158.141.203.in-addr.arpa"{
        type master;
        file "/etc/bind/db.203.141.158"; // 外部逆引きゾーンテーブル
        allow-transfer{
                localnet;
                123.456.789.001;
                123.456.789.002;
        };
};
```

以下は各ゾーンテーブルの設定例です.

1. 順引きゾーンテーブル

```
;
; BIND data file for kinsen.gr.jp
;
$TTL        86400
@        IN        SOA        dns.kinsen.gr.jp. root.dns.kinsen.gr.jp. (
                                1               ; Serial
                             1800               ; Refresh
                              900               ; Retry
                           604800               ; Expire
                             1200 )        ; Negative Cache TTL

                IN        NS        dns

; localhost
localhost       IN      A       127.0.0.1
localhost       IN      AAAA    ::1

; Mail exchange
                IN      MX   0 mail.kinsen.gr.jp.
;
; Host entry
;
noren           IN      A      203.141.158.41
;               IN      AAAA   2001:
dns             IN      A      203.141.158.41
;               IN      AAAA   2001:
mail            IN      A      203.141.158.41
;               IN      AAAA   2001:
www             IN      A      203.141.158.41
;               IN      AAAA   2001:
;
; Domain
@               IN      A       203.141.158.41
                IN      MX 0    mail
```

2. 逆引きゾーンテーブル

```
;
;BIND data file for 203.141.158 network
;
$TTL    86400
@       IN      SOA     dns.kinsen.gr.jp. root.dns.kinsen.gr.jp. (
                                1         ; Serial
                             1800         ; Refresh
                              900         ; Retry
                           604800         ; Expire
                             1200 )       ; Negative Cache TTL

                IN      NS      dns
;
; Host entry
;
41                IN      PTR     noren.kinsen.gr.jp.
41                IN      PTR     dns.kinsen.gr.jp.
41                IN      PTR     mail.kinsen.gr.jp.
41                IN      PTR     www.kinsen.gr.jp.
```

以上の設定が終わったら, BIND を再読み込み, 再起動します.

```
# /etc/init.d/bind9 reload      注一必ず再読み込みからしてください.
# /etc/init.d/bind9 restart  再読み込みでエラーがでたら必ず修正してください.
```

正常に読み込みが終わったら設定内容を dig や nslookup を使って確かめます.

```
$ dig @localhost dns.kinsen.gr.jp (ホスト名はいろいろ試してください)
; <<>> DiG 9.5.1-P3 <<>> @localhost dns.kinsen.gr.jp
; (2 servers found)
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 55957
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0

;; QUESTION SECTION:
;dns.kinsen.gr.jp.              IN      A

;; ANSWER SECTION:
dns.kinsen.gr.jp.      86400     IN      A       192.168.0.2

;; AUTHORITY SECTION:
kinsen.gr.jp.          86400     IN      NS      dns.kinsen.gr.jp.

;; Query time: 0 msec
;; SERVER: 127.0.0.1#53 (127.0.0.1)
;; WHEN: Thu Jan 21 07:26:57 2010
;; MSG SIZE  rcvd: 64

$ dig @localhost kinsen.gr.jp MX (メールについても確認します)
; <<>> DiG 9.5.1-P3 <<>> @localhost kinsen.gr.jp MX
; (2 servers found)
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 640
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; QUESTION SECTION:
;kinsen.gr.jp.                  IN      MX

;; ANSWER SECTION:
kinsen.gr.jp.           86400   IN      MX      0 mail.kinsen.gr.jp.

;; AUTHORITY SECTION:
kinsen.gr.jp.           86400   IN      NS      dns.kinsen.gr.jp.

;; ADDITIONAL SECTION:
mail.kinsen.gr.jp.      86400   IN      A       192.168.0.3
dns.kinsen.gr.jp.       86400   IN      A       192.168.0.2

;; Query time: 0 msec
;; SERVER: 127.0.0.1#53 (127.0.0.1)
;; WHEN: Thu Jan 21 07:25:15 2010
;; MSG SIZE  rcvd: 101

$ nslookup -type=mx kinsen.gr.jp (このコマンドでもいいです)
Server:         192.168.0.2
Address:        192.168.0.2#53

kinsen.gr.jp        mail exchanger = 0 mail.kinsen.gr.jp.
```

7.9 Mail サーバの設定

Mail サーバはメールの送受信を行います. 具体的には 2 つの機能に分別できます.

- MTA (Mail Transfer Agent メール転送エージェント)
  ユーザが送信したメールを受け取って, 他のサーバとバケツリレー式に目的地まで配送したり, 届いたメールを保管する機能
- MDA (Mail Delivery Agent メール配送エージェント)
  振り分けられたメールをサーバ内のユーザや別のサーバへ配送する機能

Debian では Exim がデフォルトのメールサーバになっていますが, インストールしたばかりの DomU にはインストールされていません. そこで, 今回は設定が容易で, 柔軟性も高く, ドキュメントも豊富な MTA の postfix をインストールします. [*33]

また, Mail サーバからメールを取り出すために MDA は postfix と相性のよい Dovecot にし, IMAP プロトコルを使用します.

[*33] より詳しくは, Postfix 詳解 MTA の理解とメールサーバの構築, 運用を参照してください.

7.9.1 postfix の設定

次に, postfix をインストールし, 大まかな設定を dpkg-reconfitgure postfix で行い, 詳細な設定は後述の/etc/postfix/ の main.cf で直接, 書き換えるようにします.

```
# aptitude install postfix
# dpkg-reconfitgure postfix
```

以下, コンソールに設定画面が表示されます. 最初にサイトのタイプ, 次に各完全修飾ドメイン名, ルートやポストマスターとしてメールを受け取るユーザ, メールを受け取ることのできるその他のドメイン, メールの同期, 送信可能なネットワークの範囲, メールボックスのサイズ, 使用するプロトコルの種類などを環境に合わせて設定します.

```
1.General type of mail configuration: Internet Site

2.System mail name: mail.kinsen.ge.jp

3.Root and postmaster mail recipient: ipv6waterstar

4.Other destinations for mail: mail.kinsen.gr.jp, kinsen.gr.jp, localhost

5.Force synchronous updates on mail queue?: No

6.Local networks: 127.0.0.0/8

7.Mialbox size limit (bytes): 0

8.Local address extension character: +

9.Internet protocols to use: all
```

7.9.2 Maildir の設定

また, 受信したメールを Mail ディレクトリで扱うように設定します. Maildir は保管する受信メールの取扱いが容易で, dovecot でも同様に扱うことができます.

```
# postconf -e "home_mailbox = Maildir/"
# postconf -e "mailbox_command ="
```

7.9.3 再起動とテスト

大まかな設定が終わったら, 設定を読み込んでテストをしてみましょう.

```
# /etc/init.d/postfix reload
# /etc/init.d/postfix restart
```

テストは telnet を使います. ただし, DomU には Telnet はインストールされていませんので, インストールしておいてください.

```
# telnet localhost 25
  Trying 127.0.0.1...
  Connected to localhost.
  Escape character is '^]'.
  220 mail.kinsen.gr.jp ESMTP Postfix (Debian/GNU)
```

以上のようにでれば正常です. 引き続き自分宛にメールを送ってみましょう.

```
  mail from: ipv6waterstar@kinsen.gr.jp
  rcpt to: ipv6waterstar@gmail.com
  data
  To: ipv6waterstar@gmail.com
  From: ipv6waterstar@kinsen.gr.jp
  Subject: Test
  This is my frist email on debian. Is it successed to send your mail?
       (本文ができたら)
  .    (を入力し)
  quit (で終了しましょう)
```

7.9.4 main.cf の設定

次に, メールサーバを立ち上げた場合, 気になるのは spam などの迷惑メール対策です. postfix は main.cf で詳細な設定が可能で, spam についてもアンチ spam 用の設定があります.

以下, main.cf の該当箇所を書き換えます.

```
smtpd_recipient_restrictions = reject_invalid_hostname,
        reject_unknown_recipient_domain,
        reject_unauth_destination,
        reject_rbl_client sbl.spamhaus.org,
        permit

smtpd_helo_restrictions = reject_invalid_helo_hostname,
        reject_non_fqdn_helo_hostname,
        reject_unknown_helo_hostname（この設定を入れると，proxy 経由のメールが受け取れなくなります）
```

また, RBL (spam のブラックリスト) を使うこともできます.

```
smtpd_client_restrictions = reject_rbl_client dnsbl.sorbs.net
```

7.9.5 その他オプション (SMPT-AUTH, Sasl, TLS, サブミッション) の設定

また, Postfix はオプションで様々なセキュリティの設定が行えますので, 代表的なものをいくつか設定します.

- SMPT-AUTH (メール送信に使うプロトコルである SMTP にユーザ認証機能を追加した仕様)
  SMTP がもともと認証を持たない仕様であったため, spam や不正中継などが横行し, 対策として, メール送信の際に SMTP サーバとユーザとの間で認証を行い, 認証された場合のみメールの送信を許可するようにしたも. 認証方式としては PLAIN, LOGIN, DIGEST-MD5, CRAM-MD5 などがある.
- Sasl (Simple Authentication and Security Layer)
  プロトコルから認証機構を分離して, SASL でサポートする任意の認証機構を任意のプロトコルで使うことができる.
- TLS (Transport Layer Security SSL から名称変更)
  インターネットで情報を暗号化し, 送受信するプロトコル. 通常は, TCP をラッピングする形で利用する. HTTP での利用を意識して設計 (ただし, 特定のプロトコルを前提とはしない).
- サブミッションポート (認証機能付きポート 587 番)
  迷惑メール対策として, ISP がポート番号の 25 を自身のサーバのメールの送信にのみ開放し事により, 一般のユーザが外部のサーバからメールの送信することができなくなったため, 認証機能付きポートを開放する事でメールの送信を可能にしたもの.

以上のオプションを使用するために, Sasl の認証機構を使ってユーザの認証 (SMPT-AUTH) を行い, ユーザ認証で用いるパスワード (平文) を TLS で暗号化します. そして, メール送信のために認証機能のついたサブミッションポートを使用するための設定をします.

7.9.6 Sasl, TLS, サブミッションポートのインストールと設定

sasl2-bin, libsasl2-modules, postfix-tls をインストールします.

```
# aptitude install sasl2-bin
# aptitude install libsasl2-bin
# aptitude install postfix-tls
```

/etc/postfix/sasl/smtpd.conf を書き加えます.

```
pwcheck_method: saslauthd
mech_list: PLAIN LOGIN
```

/etc/default/saslauthd で sasl デーモンを許可し, /etc/init.d/saslauthd start で起動します.

```
START=yes
```

/etc/postfix/main.cf の以下を書き換え, smtpd_recipient_restrictions へ新たな設定を加えます.

```
smtpd_sasl_local_domain = $myhostname
smtpd_sasl_auth_enable = yes
broken_sasl_auth_clients = yes
```

```
smtpd_recipient_restrictions = reject_invalid_hostname,
        reject_unknown_recipient_domain,
        reject_unauth_destination,
        reject_rbl_client sbl.spamhaus.org,

        permit_sasl_authenticated, //
        permit_mynetworks,         // 以上を加えます.
        reject_unauth_destination, //

        permit
```

Postfix ユーザに sasl グループを加えます.

```
# adduser postfix sasl
```

bind を使い, saslauthd に名前をつける.

```
# /var/run/saslauthd /var/spool/postfix/var/run/saslauthd bind bind 0 0
```

fstab に加えます.

```
# cd /var/spool/postfix
# mkdir -p var/run/saslauthd
# mount /var/spool/postfix/var/run/saslauthd
```

設定が終わったら, sasl と postfix を再起動します.

```
# /etc/init.d/saslauthd restart
# /etc/init.d/postfix reload  // エラーがでたら，必ず設定を確認して下さい.
# /etc/init.d/postfix restart
```

再起動ができたら, テストをしてみましょう

```
# telnet localhost 25
  Trying 127.0.0.1...
  Connected to localhost.
  Escape character is '^]'.
  220 mail.kinsen.gr.jp ESMTP Postfix (Debian/GNU)

  ehlo local (ローカルで調べます)
  250-mail.kinsen.gr.jp
  250-PIPELINING
  250-SIZE 10240000
  250-VRFY
  250-ETRN
  250-STARTTLS (TLS の確認)
  250-AUTH PLAIN LOGIN (SMTP-AUTH の確認)
  250-ENHANCEDSTATUSCODES
  250-8BITMIME
  250 DSN

  quit (で終了します)
  221 2.0.0 Bye
  Connection closed by foreign host.
```

次に, 現在の設定ではパスワードが平文でネットワーク上を流れる事になりますので, パスワードを TLS で暗号化するために, main.cf を書き換えます. また, 証明書や鍵については以下を参照して下さい.

- http://yocum.org/faqs/postfix-tls-sasl.html

```
smtp_use_tls = yes
smtpd_use_tls = yes
smtp_tls_note_starttls_offer = yes
smtpd_tls_key_file = /etc/ssl/certs/cacert.pem   //
smtpd_tls_cert_file = /etc/ssl/certs/cacert.pem  // この部分は個々の鍵の作成の内容ごとに変わります.
smtpd_tls_["CAfile"] = /etc/ssl/certs/cacert.pem //
smtpd_tls_loglevel = 1
smtpd_tls_received_header = yes
```

サブミッションポートについては以下の通り設定します. /etc/postfix/master.cf ファイルを以下の用に書き換えます

```
submission inet n       -       -       -       -       smtpd
        -o smtpd_etrn_restrictions=reject
        -o smtpd_enforce_tls=yes
        -o smtpd_sasl_auth_enable=yes
```

設定ができたら, Postfix を再起動し, 設定を確認しましょう.

```
$ netstat -at
  Active Internet connections (servers and established)
  Proto Recv-Q Send-Q Local Address       Foreign Address         State
  tcp        0      0 *:imaps             *:*                     LISTEN
  tcp        0      0 *:submission        *:*                     LISTEN
  tcp        0      0 *:imap2             *:*                     LISTEN
  tcp        0      0 *:smtp              *:*                     LISTEN
  tcp6       0      0 [::]:submission     [::]:*                  LISTEN
  tcp6       0      0 [::]:smtp           [::]:*                  LISTEN
```

以上で, Postfix の設定は以上ですがより詳しく知りたい方は

- `http://wiki.debian.org/Postfix` を参照して下さい.

なお, 今回は触れませんでしたが, 上記以外の spam 対策として, spamassassin やウィルス対策として Clamav などがあります.

7.9.7 dovecot の設定

Dovecot は Linux のような UNIX ライクな OS で動作する, POP3/IMAP に対応した MDA です.

今回は, IMAP プロトコルを使います. IMAP (InternetMessageAccessProtocol) は, メールサーバのメールにアクセスし操作するためのプロトコルで, オフラインとオンラインの双方で利用できます. オフラインではローカルでメールを扱い, オンラインでメールの保管されてメールディレクトリと同期してメールを扱います. これにより, 複数のクライアントでもディレクトリによって一元管理ができます.

因みに, Dovecot を Debian で動かすための設定を書いた適当なドキュメントがありません. そこで, 下記の Ubuntu の設定をそのまま使います. また, より詳しい情報は Dovecot の wiki などを参照して下さい.

- `https://help.ubuntu.com/community/Dovecot`
- `http://wiki.dovecot.org/`

7.10 Dovecot のインストールと設定

今回は, プロトコルを imap に限定したので, Dovecot の imap のパッケージのみをインストールします.

```
# aptitude install dovecot-imapd
```

次に, /etc/dovecot/dovecot.conf の設定を以下のように書き換えます.

```
protocols = imap imaps              // プロトコルの設定
    (中略)
listen = *                          //IP アドレスの Ver
    (中略)
mail_location = maildir:~/Maildir // メールディレクトリの設定
```

7.11 Dovecot でのユーザ認証と SSL の設定

また, Dovecot でもユーザの認証と SSL でパスワードの暗号化をします. 以下その設定です.

```
disable_plaintext_auth = no // 認証の許可
    （中略）
ssl_disable = no  //ssl の許可と認証鍵の設定
ssl_cert_file = /etc/ssl/certs/ssl-cert-snakeoil.pem
ssl_key_file = /etc/ssl/private/ssl-cert-snakeoil.key
```

以上で, Dovecot の設定ができましたので, 再起動してテストしてみましょう.

```
# /etc/init.d/dovecot restart
# telnet localhost 143
  Trying localhost...
  Connected to localhost.
  Escape character is '^]'.
  +OK dovecot ready.
```

以上のようにでたら OK です. netstat などでポートも確かめておいて下さい.

7.12 まとめ

最後は, Web サーバの設定なのですが, 今の時点で Web サーバの構築ができていない状況です. 仕様としては Red5[*34] をインストールし, 動画サイトの運営を目標としたい思っています. 構築できましたら, また発表します.

以上が, Xen でサーバを作るときの基本的な設定です. Xen を使うとソフトベースであるため比較的楽にサーバの構築ができます. ですが, 1 台の PC で, 1 年, 24 時間, サーバを動かすことを考えると, 改善する点も多数あると考えています. あと, サーバ群の PC が 1 台に集中できるので, 省スペースで省エネルギー (電気代は月 700 円ぐらい) です.

今後は, 各サーバを IPv6 に対応させる事や, 仮想化の「 主流」になるとだろう「 KVM 」と比較していきたいと思います. 設定方法も, もっといい方法を見つけていきたいと思います. また, 何かありましたら, メールを送っていただけるとありがたいです.

- ipv6waterstar@kinsen.gr.jp

[*34] http://osflash.org/red5

8 lxc コンテナを使ってみる

まえだこうへい

Debian 勉強会に参加されている面々は、そろそろ Xen や KVM 以外に何かないのか、新たなネタを求めている向きが多いかと思います。そこで、最近、 Linux Kernel に新たにマージされた機能を使って実現している、 lxc という仮想化技術について紹介します。

8.1 はじめに

8.1.1 lxc の概要

lxc[*35] は、正式名称を Linux Containers と言い、コンテナ自体が稼働するためのカーネルの機能と、コンテナを管理するためのとユーザツールから構成されます。 lxc で使用している Kernel の機能 (Control Group, 以下 Cgroup と省略) は、 kernel 2.6.29 で完全にマージされ、カーネルにパッチを当ててビルドする必要がなくなってなっています。kernel 2.6.29 より前のカーネルでは、 kernel 2.6.27 以降であればパッチを当てれば使うことができます。

lxc は GPL2 ライセンスで公開されていて、開発およびメンテナンスは、 Daniel Lezcano 氏が実質一人で行っています。

Debian では、 squeeze/sid からパッケージ化されており、最新版[*36]がパッケージとなっています。[*37]

8.1.2 他のコンテナ型仮想化技術との比較

コンテナ型の仮想化技術というと有名なのは、 Solaris Containers や FreeBSD jail がありますが、 Linux では Linux-VServer, OpenVZ[*38]などがあります。いずれも既に使ったことがある方が多いのではないのでしょうか。

lxc で提供されるサービスは、大きく分類してシステムコンテナと、アプリケーションコンテナの 2 つがあります。前者は、いわゆる OS まるごとの仮想化です。 init から起動して、仮想 OS の空間を提供します。後者は、 chroot によるアプリケーションの分離に近いです。単一アプリケーションを分離するだけなので、とても軽くシンプルなのが特徴です。

lxc は現状、一人で開発・メンテナンスされており、今後プロジェクトがどうなるのか先行き見えないところではあります。現在、メーリングリストを見ている限りでは開発は続いているようです。

8.2 導入してみる

8.2.1 ソースコードを取得する

ユーザスペースのツールのソースコードは、 SourceForge にあります。ソースコードは git で管理されており、最新版は git リポジトリから取得できます。

[*35] `http://lxc.sourceforge.net/`

[*36] 2010 年 7 月 16 日現在、 0.7.1。 2010 年 6 月に Web サイト `http://lxc.sourceforge.net/` もリニューアルされました。

[*37] 元々の内容は 2009 年 11 月 27 日時点での一つ前の最新版だった 0.6.3 を元に記述していましたが、本書に掲載するにあたり最新版の 0.7.1 で内容をアップデートしています。

[*38] Parallels Virtuozzo Containers の OSS 版。

```
$ git clone git://lxc.git.sourceforge.net/gitroot/lxc/lxc
```

Debian では前述のとおり、 squeeze/sid でパッケージになっている[*39] ため、最新版である必要がなければ、ソースコードは特に必要ありません。

8.2.2 カーネルオプションを有効にする

lxc の機能をフルに活用するには以下のカーネルオプションが有効になっている必要があります。

```
* General setup
  * Control Group support
    -> Namespace cgroup subsystem
    -> Freezer cgroup subsystem
    -> Cpuset support
    -> Simple CPU accounting cgroup subsystem
    -> Resource counters
      -> Memory resource controllers for Control Groups
  * Group CPU scheduler
    -> Basis for grouping tasks (Control Groups)
  * Namespaces support
    -> UTS namespace
    -> IPC namespace
    -> User namespace
    -> Pid namespace
    -> Network namespace
* Device Drivers
  * Character devices
    -> Support multiple instances of devpts
  * Network device support
    -> MAC-VLAN support
    -> Virtual ethernet pair device
* Networking
  * Networking options
    -> 802.1d Ethernet Bridging
* Security options
  -> File POSIX Capabilities
```

これらが有効になっているかを確認するには、 lxc のソースツリーに含まれている、 src/lxc/lxc-checkconfig.in というシェルスクリプトを実行すれば、現在起動中のカーネルでどのカーネルオプションが無効になっているかをチェックできます。

また、 Debian パッケージでは、/usr/bin/lxc-checkconfig としてインストールされています。 squeeze/sid で Debian のカーネルパッケージ[*40]を使っている環境で確認すると以下の結果になります。 Namespaces と Cgroup memory controller が無効になっているようです。

```
$ lxc-checkconfig
--- Namespaces ---
Namespaces: required
Utsname namespace: enabled
Ipc namespace: enabled
Pid namespace: enabled
User namespace: enabled
Network namespace: enabled
Multiple /dev/pts instances: enabled

--- Control groups ---
Cgroup: enabled
Cgroup namespace: enabled
Cgroup device: enabled
Cgroup sched: enabled
Cgroup cpu account: enabled
Cgroup memory controller: missing

--- Misc ---
Veth pair device: enabled
Macvlan: enabled
Vlan: enabled
File capabilities: enabled

Note : Before booting a new kernel, you can check its configuration
usage : CONFIG=/path/to/config /usr/bin/lxc-checkconfig
```

[*39] http://packages.debian.org/search?keywords=lxc&searchon=names&exact=1&suite=all&section=all

[*40] 2010 年 7 月 16 日現在、 linux-image-2.6.32-3-amd64

8.2.3 Debian でのインストール

ここから先は、 squeeze/sid でパッケージを使うことを前提として話を進めますが、このままでは、 cgroup でのメモリ管理は無効になっていますので、カーネルオプション CONFIG_CGROUP_MEM_RES_CTLR を有効にしてリビルドしてください。それ以外で実際に Debian で lxc を使うために必要なパッケージは何かというと lxc だけです。

```
$ sudo apt-get install lxc
```

8.2.4 cgroup ファイルシステムのマウント

インストールが完了したら、/etc/fstab に以下を追加します。

```
cgroup  /var/local/cgroup  cgroup  defaults  0  0
```

マウントポイントの/var/local/cgroup は任意の場所で構いません。今回のパスはもともと存在しないのでディレクトリを作成します。作成したら、マウントします。

```
$ sudo mkdir /var/local/cgroup
$ sudo mount cgroup
```

8.2.5 動作確認

これでアプリケーションコンテナを試すことができます。開発元のドキュメント[*41]にも載っている手順ですが、次のコマンドを実行すると、即席のコンテナを起動できます。

```
$ uname -a
Linux silicon 2.6.34 #1 SMP Mon May 17 22:08:26 JST 2010 x86_64 GNU/Linux
$ sudo lxc-execute -n foo -f /usr/share/doc/lxc/examples/lxc-macvlan.conf /bin/bash
root@alpha:~# id
uid=0(root) gid=0(root) 所属グループ=0(root)
```

他のコンソールからコンテナが起動しているか確認してみます。

```
$ sudo lxc-info -n foo
'foo' is RUNNING
$ sudo lxc-ps -n foo
CONTAINER    PID TTY          TIME CMD
          20751 pts/1    00:00:00 lxc-ps
          20752 pts/1    00:00:00 ps
```

ちゃんと確認できましたね。今回は、これで以上です、と言いたいところですが、この環境はコンテナを起動させただけでしかなく、はっきり言って役に立ちません。 bash を sudo で起動しているだけで、ホスト OS のファイルシステムにもアクセスできてしまいます。

コンテナだけを起動させて満足、はい、終了とするのであれば良いかもしれませんが、実際に lxc を活用しようと考えているなら次に挙げる他のパッケージをインストールし、さらにコンテナ用のクローズ環境を作る必要があります。

- iproute : コンテナのネットワーク設定を行うため。
- debootstrap : コンテナのイメージを作成するため。

今回は、クローズな Debian 環境を簡単に作るための方法を紹介します。[*42]

8.2.6 ネットワークの設定

ブリッジの設定

[*41] http://lxc.sourceforge.net/man/lxc.html

[*42] 厳密に言うと、コンテナからホスト OS で稼働しているプロセスや、カーネルメッセージが見えてしまうのでクローズにはまだなっているとは言えませんが、まだ開発中でもあるのでそのうち改善されるでしょう。

必要なパッケージをインストールしたら、まずはブリッジの設定を行う必要があります。/etc/network/interfaces で直接ブリッジの設定をすれば良いと思いますが。シェルスクリプトを用意して、それを post-up で実行させれば良いでしょう。

```
#!/bin/sh

brctl addbr br0                              <- ブリッジデバイスの追加
brctl setfd br0 0                            <- ブリッジデバイス br0 の設定
ifconfig br0 192.168.0.1 promisc up
brctl addif br0 eth0
ifconfig eth0 0.0.0.0 up
route add -net default gw 192.168.0.254 br0 <- ホスト OS のゲートウェイ
```

interfaces は以下のように設定します。

```
$ cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
allow-hotplug eth0
iface eth0 inet static
        address 192.168.0.101
        netmask 255.255.255.0
        broadcast 192.168.0.255
        pre-up  /etc/init.d/iptables start
        post-up /etc/network/if-up.d/brctl.sh
```

以上のあと、ブリッジの設定がきちんとされているか確認してみると、以下のようになります。

```
$ /usr/sbin/brctl show
bridge name        bridge id               STP enabled        interfaces
br0                8000.00wwwwyyyyxxno                eth0
                                                        veth0_14820
                                                        veth0_15932
                                                        veth0_17164
```

ちなみに “veth0_” の後ろの数字は、コンテナで起動した init プロセスのプロセス ID です。

IP フォワードと NAT の設定

ホスト OS とコンテナ、コンテナとホスト OS の外部のネットワーク、コンテナ間での通信は、上記の設定だけでなく、IP フォワードや NAT の設定をする必要があります。例えば、コンテナ起動後、ホスト OS から ssh でログインするのにも、IP フォワードが必要となります。

```
$ sudo bash -c ‘‘echo 1 > /proc/sys/net/ipv4/ip_forward’’
```

IP フォワードを設定するととで、ホスト OS の外部のネットワークへの通信や、コンテナ同士の通信も行えるようにはなります。[*43]

一方、ホスト OS の外部ネットワークからはこのままではアクセスできません。アクセスを許可するには、ホスト OS でポートフォワーディングや、宛先 NAT などを行う必要があります。ポートフォワーディングを行うのであれば、以下のようなルールを設定します。

```
iptables -t nat -A PREROUTING -d 192.168.0.1 -p tcp --dport 80 -i br0 -j DNAT --to 192.168.0.101
iptables -t nat -A PREROUTING -d 192.168.0.1 -p tcp --dport 5984 -i br0 -j DNAT --to 192.168.0.102
```

ここでの設定は、デフォルトポリシーが全て ACCEPT であることを前提にしていますが、実際には当然 DROP にすると思いますので、FORWARD のルールなども定義する必要があります。Netfilter のルールと IP フォワーディングの許可はスクリプトにして、/etc/network/interface で pre-up として設定しておくと良いでしょう。

なお、FORWARD チェインのデフォルトポリシーが ACCEPT である場合は問題ありませんが、DROP や

[*43] 同じブロードキャストドメインの場合。

REJECT に設定してある場合は、コンテナ同士での通信はできません。FORWARD チェインでの ACCEPT ルールが必要になります。

8.2.7 システムコンテナの作成

では、システムコンテナのイメージを作成してみます。lxc のパッケージに含まれる、/usr/lib/lxc/lxc/templates/lxc-debian を使って、Debian のシステムコンテナを作成します。

このスクリプトでは debootstrap を使ってイメージが作成されますが、ディストリビューションは lenny になっています。前述した通り本環境は squeeze/sid ですので、コンテナも squeeze/sid の方が良ければ、コンテナイメージを作成する前に予めスクリプトを書き換えておく必要があります。

パッケージは、デフォルトでは以下のものがインストールされます。

- ifupdown
- locales
- libui-dialog-perl
- dialog
- dhcp-client
- netbase
- net-tools
- iproute
- openssh-server
- iputils-ping

sudo や vi, dig コマンドが無かったりするので、そのままインストールすると不便であったりするので、予めインストールするパッケージの指定を変更しておく必要があります。lxc-debian スクリプト内の download_debian() 関数の `packages` 変数でパッケージの指定は変更できます。

```
$ diff -u /usr/lib/lxc/lxc/templates/lxc-debian lxc-debian
--- /usr/lib/lxc/lxc/templates/lxc-debian       2010-06-28 00:18:36.000000000 -1000
+++ lxc-debian  2010-07-16 02:38:25.000000000 -1000
@@ -94,7 +94,10 @@
 netbase,\
 net-tools,\
 iproute,\
-openssh-server
+openssh-server,\
+vim-tiny,\
+sudo,\
+dnsutils

     cache=$1
     arch=$2
```

それでは、コンテナを作成します。コンテナを作成するディレクトリは予め作成しておき、lxc-debian スクリプトの-p または–path オプションでそのパスを指定します。指定したディレクトリの下に、lxc の設定ファイル config と、rootfs イメージディレクトリが作成されます。

```
$ sudo mkdir /var/cache/lxc/hoge
$ sudo ./lxc-debian -p /var/cache/lxc/hoge
debootstrap is /usr/sbin/debootstrap
Checking cache download in /var/cache/lxc/debian/rootfs-amd64 ...
Downloading debian minimal ...
I: Retrieving Release
I: Retrieving Packages
I: Validating Packages
I: Resolving dependencies of required packages...
I: Resolving dependencies of base packages...
(snip)
I: Base system installed successfully.
Download complete.
debootstrap is /usr/sbin/debootstrap
Checking cache download in /var/cache/lxc/debian/rootfs-amd64 ...
Copying rootfs to /var/cache/lxc/hoge/rootfs...Generating locales (this might take a while)...
Generation complete.
 Removing any system startup links for /etc/init.d/umountfs ...
   /etc/rc0.d/S40umountfs
   /etc/rc6.d/S40umountfs
 Removing any system startup links for /etc/init.d/hwclock.sh ...
   /etc/rc0.d/K25hwclock.sh
   /etc/rc6.d/K25hwclock.sh
   /etc/rcS.d/S11hwclock.sh
 Removing any system startup links for /etc/init.d/hwclockfirst.sh ...
   /etc/rcS.d/S08hwclockfirst.sh
Root password is 'root', please change !
```

これで、/var/cache/lxc/hoge/rootfs という名前でコンテナイメージのディレクトリが作成され、ここに、 debootstrap による Debian イメージのコピーが作成されます。初めて `lxc-debian` スクリプト を実行すると、 debootstrap でダウンロードされるキャッシュイメージが、/var/cache/lxc/debian/rootfs-arch として作成されます。[*44]

コンテナ自体のメタ情報は、コンテナを起動させた後に、 cgroup ファイルシステムのマウントポイントの下に、/var/local/lxc/コンテナ名 に格納されます。

```
$ ls  /var/local/cgroup/hoge/
cgroup.event_control             debug.taskcount
cgroup.procs                     devices.allow
cpu.shares                       devices.deny
cpuacct.stat                     devices.list
cpuacct.usage                    freezer.state
cpuacct.usage_percpu             memory.failcnt
cpuset.cpu_exclusive             memory.force_empty
cpuset.cpus                      memory.limit_in_bytes
cpuset.mem_exclusive             memory.max_usage_in_bytes
cpuset.mem_hardwall              memory.memsw.failcnt
cpuset.memory_migrate            memory.memsw.limit_in_bytes
cpuset.memory_pressure           memory.memsw.max_usage_in_bytes
cpuset.memory_spread_page        memory.memsw.usage_in_bytes
cpuset.memory_spread_slab        memory.move_charge_at_immigrate
cpuset.mems                      memory.soft_limit_in_bytes
cpuset.sched_load_balance        memory.stat
cpuset.sched_relax_domain_level  memory.swappiness
debug.cgroup_css_links           memory.usage_in_bytes
debug.cgroup_refcount            memory.use_hierarchy
debug.current_css_set            net_cls.classid
debug.current_css_set_cg_links   notify_on_release
debug.current_css_set_refcount   tasks
debug.releasable
```

8.2.8 システムコンテナの設定

システムコンテナの起動の前後にやることがあります。設定しない場合のデメリットを記載しておきました。

- /etc/hostname の設定 設定しない場合、ホスト OS と同じホスト名になる。
- /etc/hosts の設定 設定しないと localhost やコンテナ自身のホスト名を解決できない。
- /etc/networik/interface の設定 通常、 eth0 がデフォルトの NIC なので、ホスト OS 側で WiFi を使って eth0 がリンクアップしていないと、 DHCP で IP アドレスが割り当てられず起動に時間がかかる。
- ログイン用のユーザの作成 ログイン可能なユーザが root ユーザだけ[*45]。コンテナ起動後でもできるので、後で必ず行うこと。

2009 年 12 月のときに紹介した、バージョン 0.6.3 とは違い、コンテナの OS に対する初期設定は何もされず、debootstrap で作成した状態のままです。ですので、コンテナ起動前後での OS の基本的な設定が必要になってきます。

[*44] 今回は amd64 を選択しているので、/var/cache/lxc/debian/rootfs-amd64 となります。
[*45] 初期パスワードは root なので変更すること。

8.2.9 システムコンテナの起動

上記の設定を終えたら、システムコンテナを起動します。システムコンテナの起動は、 `lxc-start` コマンドを使います。一つの環境で複数のコンテナを起動できるので、コンテナ名の指定が必ず必要です。コンテナ名の指定には、オプション`-n` を使います。また、`-f` オプションでの config の指定も必要です。

```
$ sudo lxc-start -n hoge -f /var/cache/lxc/hoge/config
INIT: version 2.86 booting
Activating swap...done.
Cleaning up ifupdown....
Checking file systems...fsck 1.41.3 (12-Oct-2008)
done.
Setting kernel variables (/etc/sysctl.conf)...done.
Mounting local filesystems...done.
Activating swapfile swap...done.
Setting up networking....
Configuring network interfaces...SIOCADDRT: No such process
Failed to bring up eth0:1.
done.
INIT: Entering runlevel: 3
Starting OpenBSD Secure Shell server: sshd.

Debian GNU/Linux 5.0 hoge console

hoge login: kohei
Password:
Last login: Fri Jul 16 05:06:28 HST 2010 on console
Linux hoge 2.6.34 #1 SMP Mon May 17 22:08:26 JST 2010 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
kouhei@hoge:~$
```

このまま起動させると、現在のシェルでそのままコンテナのコンソールが表示されます。ログインするにはそのままコンソールログインすれば良いでしょう。バックグラウンドで起動させるには、`-d` オプションをつけます。

```
$ sudo lxc-start -n hoge -d -f /var/cache/lxc/hoge/config
```

KVM や Xen などのようにカーネルから起動させる訳ではなく、 init プロセスから起動させるので、起動完了までに要する時間はわずかです。

8.3 lxc の仕組みを見てみる

8.3.1 コンテナのライフサイクル

Linux コンテナのライフサイクルは他の仮想化技術と大きく異なるところはなく、図 13 の様になります。

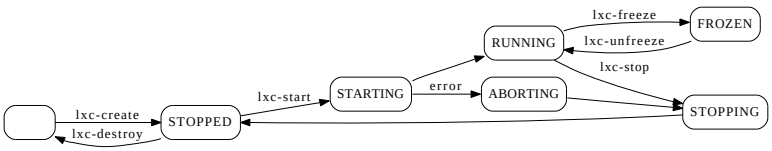


図 13　Linux コンテナのライフサイクル

この図 13 での状態を遷移させるためのコマンド、つまりコンテナを管理するためのコマンドは以下の通りです。

status	コマンド	実行後の状態	備考
コンテナの起動	lxc-start	STARTING, RUNNING	システムコンテナ
	lxc-execute		アプリケーショ ンコンテナ
コンテナの一時停止	lxc-freeze	FROZEN	
コンテナの再開	lxc-unfreeze	RUNNING	
コンテナの停止	lxc-stop	STOPPING, STOPPED	
コンテナの再起動	lxc-restart	STOPPING, STOPPED	
		STARTING, RUNNING	
コンテナの作成	lxc-create	STOPPED	lxc-debian は内部で lxc-create を実行。
			lxc-fedora, lxc-sshd なども同様。
コンテナの破棄	lxc-destroy		

8.3.2 リソースを制御する

lxc では、コンテナのリソースを制御するために、 cgroup というカーネルの機能を使っています。 cgroup とは、Control group の略です。 Linux Kernel は通常プロセス単位でのリソース制御を行っていました。 cgroup を使うと、同じ cgroup に所属しているプロセス間でのリソースの共有ができます。 lxc では、 `lxc-cgroup` コマンドを用い、コンテナのリソースの設定を表示したり、値をセットすることができます。

設定を変更する。

cgroup の変数を表示する場合は、 “`lxc-cgroup -n` **コンテナ名 変数名**” で、変数の値を変更するには、 “`sudo lxc-cgroup -n` **コンテナ名変数名 設定する値**” とします。

例えば、 CPU リソースの割り当てには、 cpu.shares というパラメータを用いますが、あるコンテナに割り当てられている比率を算出するには、**任意のコンテナの** `cpu.shares` **/ 各コンテナの** `cpu.share` **の総和**を計算する必要があります。具体的には、現在の各コンテナの cpu リソースの割り当て設定を確認すると、

```
$ lxc-ls
couchdb  git  hoge  octave  w3m  web
$ for i in `lxc-ls`; do echo -ne $i"\t"; lxc-cgroup -n $i cpu.shares; done
couchdb         2048
git         1024
hoge         1024
octave         1024
w3m         1024
web         1024
```

総和は 7168 で、各コンテナは、 couchdb は約 28.6%、他のコンテナは約 14.3% ごとの比率で CPU リソースをシェアするということ分かります。

コンテナ全体のリソースは?

上記の通り、 `lxc-cgroup` コマンドは、コンテナを明示的に指定する必要があるので、 cgroup 管理下全体での使用率などを把握するには不便です。 cgroup で制御されるリソースは 8.2.7 節でマウントした、 cgroup ファイルシステムからアクセスすることもできます。今回はマウントポイントを/var/local/cgroup にしているのでこのディレクトリの下を見てみると以下の様になっています。[*46]

[*46] lxc-cgroup コマンド自体は、各コンテナの値を/var/lib/lxc/コンテナ名/nsgroup/以下から取得しますが、これは、/var/local/cgroup/コンテナ名 への symlink になっています。

```
$ cd /var/local/cgroup
$ ls -F
cgroup.event_control             debug.taskcount
cgroup.procs                     devices.allow
cpu.shares                       devices.deny
cpuacct.stat                     devices.list
cpuacct.usage                    hoge/
cpuacct.usage_percpu             memory.failcnt
cpuset.cpu_exclusive             memory.force_empty
cpuset.cpus                      memory.limit_in_bytes
cpuset.mem_exclusive             memory.max_usage_in_bytes
cpuset.mem_hardwall              memory.memsw.failcnt
cpuset.memory_migrate            memory.memsw.limit_in_bytes
cpuset.memory_pressure           memory.memsw.max_usage_in_bytes
cpuset.memory_pressure_enabled   memory.memsw.usage_in_bytes
cpuset.memory_spread_page        memory.move_charge_at_immigrate
cpuset.memory_spread_slab        memory.soft_limit_in_bytes
cpuset.mems                      memory.stat
cpuset.sched_load_balance        memory.swappiness
cpuset.sched_relax_domain_level  memory.usage_in_bytes
debug.cgroup_css_links           memory.use_hierarchy
debug.cgroup_refcount            net_cls.classid
debug.current_css_set            notify_on_release
debug.current_css_set_cg_links   release_agent
debug.current_css_set_refcount   tasks
debug.releasable
```

/var/local/cgroup ディレクトリ以下にある、“xxx.yyyy” の形式をとっているのが、 cgroup で管理するリソース項目で、このディレクトリ直下はシステム全体のリソースです。一方、 hoge/などのディレクトリがありますが、これは各コンテナに割り当てられているリソース項目です。ですので、このディレクトリを使って、 lxc 全体と各コンテナのリソースの管理を行うことができます。

8.4 まとめ

Linux Kernel の標準機能だけを使ったコンテナである、 Linux Containers について説明しました。まだまだ機能的には不十分なところもあり、開発体制も不安なところはありますが、特別なカーネルパッチを適用せずに試せる点では非常に気軽に使えるのではないかと思います。

また、 lxc は libvirt のサポート対象にもなっています。本資料を作成前に libvirt で扱えるか検証してみたところ、定義ファイルを作るところまではできたものの、 virsh でコネクトするとうまくリソースにアクセスできないといった問題もありますが、やることが多いのでハックするには良いネタになるのではないでしょうか。

8.5 参考文献

- LXC
  http://lxc.sourceforge.net/lxc.html
- LXC: Linux コンテナツール ; IBM developer Works Japan
  http://www.ibm.com/developerworks/jp/linux/library/l-lxc-containers/[*47]
- Cgroup And Memory Resource Controller
  http://www.linux-foundation.jp/uploads/seminar20081119/CgroupMemcgMaster.pdf

[*47] 2009 年 2 月の記事ですが内容が若干古く、現在存在しないコマンドもあるので要注意。

9 Debian ユーザのための Ubuntu 入門

あわしろいくや

9.1 自己紹介

- Ubuntu Japanese Team: 日本語入力担当 (らしい)
- OpenOffice.org 日本ユーザ会
- Debian では scim-anthy とかのメンテナ (最近は任せきり)
- Anthy もリリースしてた (今はやる気がなくて困り中)
- 雑誌とかに原稿書いたり
- 本業も微妙に Ubuntu

9.2 Ubuntu はやわかり

- Linux ディストリビューション

- Debian はユニバーサルオペレーティングシステム
- アーキテクチャは i386/AMD64/ARM(非公式には IA64 とか PowerPC とか)
- 半年に 1 度リリース, 2 年に 1 度 LTS(Long Term Support) がリリース
- GNOME 採用, 格好いいアートワーク...
- 運営は Canonical を主体としたコミュニティ. DD もたくさん
- CD1 枚に収まるサイズでライブ CD(DVD もあるけどね)
- たくさんの派生版: Kubuntu とか Netbook Editon とか (非公式なものもたくさん)
- サーバ版もある: 簡単にクラウド環境をセットアップ
- リポジトリ
  - main: Canonical を含む Core Developpers によってメンテナンスされる
  - universe: コミュニティによるメンテナンス
  - multiverse: Debian でいうところの non-free
  - restricted: プロプライエタリなドライバ
- Alternate CD: CUI インストーラ. アップグレードにも使える
- Ubuntu Netbook Editon: 専用 UI

9.3 Launchpad ってなに?

- やたら高性能な CMS
  → BTS, 翻訳, アイディアの集積, Q&A 開発プロジェクトのホスティング, Personal Package Archive
- これのアカウントを取ることによって、 Ubuntu One とかも使用可能になる

9.4 Ubuntu One ってなに?

- Dropbox みたいなファイル同期サービス
  2GB まで無料。 50GB まで月 10 ドル. Tomboy とか Firefox のブックマークも同期. Ubuntu One Music Store で買った曲も Ubuntu One に転送される

9.5 LTS ってなに?

- Long Term Support: 通常はリリース後 18 ヶ月サポート
- LTS はデスクトップ版で 3 年、サーバ版で 5 年サポート

- LTS はポイントリリースあり: Ubuntu 10.04.1 とか
- LTS → LTS のアップグレードもサポート.

9.6 コードネームとバージョニング

- 動詞 + 動物. 頭韻を踏む: Lucid Lynx とか、Maverick Meerkat とか. LL → MM. https://wiki.ubuntu.com/DevelopmentCodeNames に予想とか書いてあって笑える
- バージョンはリリース年 + 月.

9.7 デスクトップ版

- Ubuntu 10.04,
- Ubuntu Netbook Editon 10.04,
- Kubuntu 10.04 LTS,
- Kubuntu Netbook Remix,
- Xubuntu 10.04,
- Lubuntu 10.04 (非公式)

9.8 サーバ版

- Ubuntu Server: 専用のインストーラが用意. X なし
- 通常のサーバと、Ubuntu Enterprise Cloud(Eucalyptus) のセットアップ
- LTS → サポート期間が長い
- Byobu: screen 用のプロファイル
- インストールの最初で、Ubuntu Server か UEC を選択する→ Ubuntu Server を選択すると、具体的に何をインストールするか聞かれる

9.9 Ubuntu 10.04 LTS の概要

- 3 番目の LTS
- Ubuntu One Music Store(MP3 の音楽が買える)
- 起動時間の短縮 (5 秒切るとか)
- Noueveau（ぬーぼー）: NVIDIA 用のオープンソースなドライバ
- GIMP サヨウナラ: F-Spot で簡単なレタッチならできるという理屈
- PiTiVi コンニチハ: 動画編集ソフト
- GWibber コンニチハ: Twitter クライアント
- ウィンドウ・マネージャのボタンの位置: 左に移った
- テーマの変更
- フォントの変更: Takao フォント
- ARM の話
  - Ubuntu 9.04 までは ARMv5 でビルド
  - 9.10 ではカーネルのみ ARMv7 でビルド: Sheevaplug で動かない
  - 10.04 ではユーザランドも ARM7 でビルドとか → Debian の出番だ!

9.10 Ubuntu の活用事例

- NetWalker
- SheevaPlug
- Dell Inspiron Mini10 → あろうことか LPIA は 10.04 で廃止。プリインストールの Ubuntu は IPIA の 8.04

9.11 Ubuntu の歴史

- 2004 年、Mark Shuttleworth とオープンソース開発者が集まって準備開始
  - タイムベースリリース
  - Debian ベース, GNOME
  - 自由へのコミットメント
- 最初のリリースは Ubuntu 4.10, 最初に日本語サポートが入ったのは 6.06

9.12 Debian との関係

- まあまあ良好? メンテナが同じパッケージも多い (IBus とか)
- universe は unstable あるいは testing のパッケージからインポート
- 最近は、まず Debian のリポジトリに入ることも多いらしい。
- Ubuntu の人たちが Debian のバグを潰したり
- もともと 10.04 は squeeze と同時期リリースの予定だった
  - 10.04 では testing からのインポートで、bug squash の手伝いになるというもくろみもあった

[*48] https://wiki.ubuntu.com/UbuntuDevelopment/PatchTaggingGuidelines

らしい

- Debian 向けのパッチも公開. パッチの共用化[*48]

9.13 Debian との違い

- 言語サポート: 各言語ごとに言語パックを準備
- 日本語入力: IBus
- ハードウェア・ドライバ: ビデオカードとか無線 LAN のプロプライエタリなドライバを簡単にインストール
- Firefox は、 Mozilla から正式にライセンスされている(らしい)

9.14 開発スタイル

- まずスケジュールを立てる
- みんなでがんばる
- 開発版と称してスナップショットをリリース
- スケジュールは、前のバージョンのリリース前に、コードネームと一緒に発表
- リリース後に UDS というミーティングを開いて、何をするか決める
- Launchpad の Blueprints に、概要や重要性や進捗を書く

  `https://blueprints.launchpad.net/ubuntu/maverick`

9.15 Debian の方が優れているところ

- head のおっかけ: sid とか experimental とか
- Ubuntu だと半年に一度変更しないとダメ.
- 最小構成でのインストール
- いちおう minimal はあるが, 基本的にはメタパッケージベース

9.16 さらなる情報

- Software Design 6 月号 (書いたのが結構前なので、情報がやや古い)
- ASCII.technologies 7 月号
- アスキー・メディアワークス (福原遥ちゃんの表紙, 売り切れ必須!)
- うぶんちゅ!(CC-BY-SA) で公開
- Ubuntu Monthly Report (Software Design で執筆中.
- 行っとけ! Ubuntu 道場! (ASCII.jp で公開中)

まとめ: 今後も持ちつ持たれつ共存共栄でいきましょう! というか、みんな好きなディストリビューション使えばいいよね

10 debtags 入門

やまねひでき

10.1 概要

今日ここでは、「 debtags 便利だぜ! 」というのと「 もっと便利にするためにオラに元気を分けてくれ! 」というのを説明します。

10.1.1 探し物は何ですか?

Debian には大量のパッケージが用意されていますが、最初から全てがインストールされているわけではないので、必要に応じて導入を行います。で、必要なパッケージがわかっているのならば良いですがそうでない場合も度々です。その場合にはパッケージの検索を行ってパッケージを見つけてインストールという流れになります。

欲しいファイル・機能を思いつく→適当なキーワードで検索→見つけたパッケージをインストール

これが多くの Debian ユーザが一般的に行っている流れではないかと思います。

10.1.2 見つけにくいモノですか?

通常の apt-cache/aptitude/synaptic の検索 (search オプション) では、単純なキーワードマッチ検索を行います。しかし、キーワードにマッチするパッケージが少ない場合はともかく、大量にマッチしてしまう一般的なキーワードしか思いつかない場合は、ノイズ混じりの検索結果になって一苦労します。

例えば、ウェブブラウザを普段使っているものから変更したいと思ってパッケージを探してみましょう。

```
$ apt-cache search web browser| wc -l
295
```

いくら Debian にはパッケージが多いといってもこれは多すぎですね。その中身をちょっと見てみると…?

```
<snip>
tor - anonymizing overlay network for TCP
torrentflux - web based, feature-rich BitTorrent download manager
trac-graphviz - Graphs printing plugin for Trac
unhtml - Remove the markup tags from an HTML file
uzbl - Lightweight Webkit browser following the UNIX philosophy
vdetelweb - Telnet and Web interface for VDE 2.x
iceweasel-vimperator - Iceweasel extension to make it have vim look and feel.
<snip>
wwwoffle - World Wide Web OFFline Explorer
xdg-utils - desktop integration utilities from freedesktop.org
xemacs21-bin - highly customizable text editor -- support binaries
xemacs21-gnome-mule-canna-wnn - highly customizable text editor -- transitional package
xemacs21-gnome-mule - highly customizable text editor -- transitional package
xemacs21-gnome-nomule - highly customizable text editor -- transitional package
xemacs21-mule-canna-wnn - highly customizable text editor -- Mule binary compiled with Canna and Wnn
xemacs21-mule - highly customizable text editor -- Mule binary
xemacs21-nomule - highly customizable text editor -- Non-mule binary
xemacs21-support - highly customizable text editor -- architecture independent support files
xemacs21-supportel - highly customizable text editor -- non-required library files
xemacs21 - highly customizable text editor
<snip>
```

ネット接続の匿名化ツールである tor やブラウザのプラグインである iceweasel-vimperator、さらには xemacs21 のパッケージ群までもが検索一覧に含まれてしまっています。これは期待している「ウェブブラウザ」の検索結果ではありません。

ではどうすれば効率的にできるのか? そこで debtags ですよ。

10.2 debtags を導入してつかってみよう

では早速 debtags を導入してみましょう。

```
$ sudo aptitude install debtags
```

これだけです。簡単ですね。では実際の検索を。

```
$ sudo debtags update
$ debtags search web::browser
arora - simple cross platform web browser
chimera2 - Web browser for X
conkeror - keyboard focused web browser with Emacs look and feel
dillo - Small and fast web browser
edbrowse - A /bin/ed-alike webbrowser written in C
elinks - advanced text-mode WWW browser
elinks-lite - advanced text-mode WWW browser - lightweight version
epiphany-browser - Intuitive GNOME web browser
epiphany-extensions - Extensions for Epiphany web browser
epiphany-gecko - Dummy, transitional package
epiphany-webkit - Dummy, transitional package
ezmlm-browse - Web browser for ezmlm-idx archives
galeon - GNOME web browser for advanced users
galeon-common - data for the galeon web browser
iceape-browser - Iceape Navigator (Internet browser) and Composer
iceweasel - Web browser based on Firefox
(snip)
w3m - WWW browsable pager with excellent tables/frames support
w3m-el - simple Emacs interface of w3m
w3m-el-snapshot - simple Emacs interface of w3m (development version)
w3m-img - inline image extension support utilities for w3m
wapua - Web browser for WAP WML pages
```

先ほどよりはずっとまともな情報が検索できるようになっています。キーワードの組み合わせ方が思いつかないという方も、auto complete があるので安心です。

```
$ debtags search web(タブキーで補完)
$ debtags search web\:\:(さらにタブキーで補完)
web::TODO           web::browser        web::forum          web::server
web::application    web::cgi            web::portal         web::wiki
web::appserver      web::cms            web::scripting
web::blog           web::commerce       web::search-engine
```

さらに複数のキーワードを組み合わせての検索もお手の物です。” でくくって && で組み合わせるだけです。

```
$ debtags search 'web::browser && x11::application'
arora - simple cross platform web browser
chimera2 - Web browser for X
conkeror - keyboard focused web browser with Emacs look and feel
dillo - Small and fast web browser
epiphany-browser - Intuitive GNOME web browser
epiphany-extensions - Extensions for Epiphany web browser
epiphany-gecko - Dummy, transitional package
epiphany-webkit - Dummy, transitional package
galeon - GNOME web browser for advanced users
galeon-common - data for the galeon web browser
iceape-browser - Iceape Navigator (Internet browser) and Composer
iceweasel - Web browser based on Firefox
junior-internet - Debian Jr. Internet tools
kazehakase - GTK+-based web browser that allows pluggable rendering engines
konq-plugins - plugins for Konqueror, the KDE file/web/doc browser
konqueror - KDE 4's advanced file manager, web browser and document viewer
links2 - Web browser running in both graphics and text mode
midori - fast, lightweight graphical web browser
netsurf-gtk - Small portable web browser with CSS and Unicode support - GTK version
w3m-el-snapshot - simple Emacs interface of w3m (development version)
wapua - Web browser for WAP WML pages
```

これで「X 上のウェブブラウザっていったらどんなのがあるの? 」が検索できました。他に役立ちそうな例としては「ruby の deb パッケージ開発環境を整えたい」などはいかがでしょう?

```
$ debtags search 'devel::lang:ruby && devel::packaging'
dpkg-ruby - ruby interface for dpkg
ruby-pkg-tools - Tools for building Debian Ruby packages
rubygems1.8 - package management framework for Ruby libraries/applications
```

便利さがいまいちわからない場合は、同じ検索を apt-cache/aptitude のキーワード検索で行ってみてください。debtags rocks!

ちなみに、

```
aptitude ~Gdevel::lang:ruby
```

などとして aptitude をインターフェイスとしてタグ検索が可能になっています(ので、そちらが主に使われるようです)。さらについ先日から、新しいインターフェイスとして axi-cache search が使えるようになりました。

```
$ axi-cache search implemented-in::ruby devel::packaging
182 results found.
Results 1-20:
100% libcairo-ruby1.8 - Cairo bindings for the Ruby language
100% tictactoe - tic-tac-toe game written in Ruby
100% libgnomecanvas2-ruby1.8 - GNOME Canvas 2 bindings for the Ruby language
100% flvtool2 - a manipulation tool for flash video files
100% dpkg-ruby - ruby interface for dpkg
100% libfilesystem-ruby1.8 - Ruby1.8 extension for file-system information
100% libexif-ruby1.9.1 - EXIF tag parsing Library for ruby1.9.1
100% xmms2-scrobbler - Audioscrobbler/Last.FM client for XMMS2
100% ri1.9 - Ruby Interactive reference (for Ruby 1.9)
100% libdbm-ruby - DBM interface for Ruby
100% schleuder - GnuPG enabled mailing list manager with remailer-capabilities
100% libactiveldap-ruby1.8 - an object-oriented interface to LDAP for Ruby
100% libhttp-access2-ruby1.8 - HTTP accessing library for ruby (transitional package)
100% quickml - Very-easy-to-use mailing list system
100% libreadline-ruby1.9.1 - Readline interface for Ruby 1.9.1
100% docdiff - Compares two files word by word / char by char
100% zonecheck-cgi - DNS configuration checker (web interface)
100% libactiveldap-ruby - an object-oriented interface to LDAP for Ruby
100% ohai - Detects data about your operating system and reports it in JSON
100% ditz - distributed issue tracker
More terms: ruby ruby1.8 dependency libactiveldap activeldap interface oriented
More tags: devel::lang:ruby role::program mail::list role::shared-lib devel::library works-with::db interface::commandline
'axi-cache more' will give more results
```

おまけ

```
$ debtags search 'devel::lang:lisp && devel::packaging'
common-lisp-controller - Common Lisp source and compiler manager
dh-lisp - Debhelper to support Common Lisp related packages
$
$ debtags search 'devel::lang:haskell && devel::packaging'
haskell-devscripts - Tools to help Debian developers build Haskell packages
libhugs-cabal-bundled - A framework for packaging Haskell software
libhugs-quickcheck-bundled - Automatic testing of Haskell programs
$
$ debtags search 'devel::lang:ocaml && devel::packaging'
$
```

10.3 で、debtags というのは何ぞや?

Enrico Zini さんによって開発された「パッケージごとに予めリストアップしてある『カテゴリタグ』をつけておき、後々検索しやすくしよう」という取り組みです。ウェブサイトを見ると、どうやらランガナータンの図書コロン分類法からヒントを得て作ったようです&ポイントは複数のタグが付けられるところ。

- 任意のパッケージに対してタグをつけまくる
- タグデータが Alioth に保存される。
  `http://debtags.alioth.debian.org/tags/tags-current.gz` から現在のデータを取得できる(この時点でタグデータは未レビュー)。
- 定期的に Enrico さんと David Paleino さんが登録されたものをすべてレビューし、コミットする。
  コミット先は `http://svn.debian.org/wsvn/debtags/tagdb/tags` で確認できる。使っているスクリプトなどは `http://svn.debian.org/wsvn/debtags/tagdb/sessions/#_tagdb_sessions_`
- コミットされたデータを override ファイルに変換する。

(override ファイルについては https://wiki.debian.org/FtpMaster/Override 参照)

- 変換したデータをアップロードする。
  自動処理スクリプトによってミラーされる。
- ミラーしたデータが apt-get/aptitude update 時に取得され、/var/lib/debtags/package-tags に保存される。
  (apt-xapian-index がインストールされてる場合は、インデックスが作成され /var/lib/apt-xapian-index/ に保存される。インデックス作成時には大体 40MB ほどメモリを食う模様。)
- debtags パッケージのインストール debtags/axi-cache search で検索
- ウマー (AA 略

となります。すばらしいですね。

しかし、最初にタグをつけまくらないと検索しても引っかかりません。ということで、これからタグ付けのお時間です。現在、タグ付けされているパッケージとそうでないパッケージはどの程度あるのでしょうか?

```
$ debtags stats
Total count of packages: 38476
Total count of packages (according to APT): 38476
Total count of packages (according to Debtags): 22816
Number of facets: 31
Number of tags: 583
Number of packages with tags, but no special::not-yet-tagged tags: 22816 (100.0%)
Number of packages with special::not-yet-tagged tags: 0 (0.0%)
Number of packages with only special::not-yet-tagged tags: 0 (0.0%)
Number of packages with no tags: 0 (0.0%)
```

やりがいがありますね!

10.4 どうやってタグを付けるの?

タグをつけるには 3 種類のやり方があります

- CLI (debtags tag)
- GUI (debtags-edit)
- web インターフェイス

このうち、一番取っ付き易いと思われるのは web インターフェイスなので、こちらの説明を中心に行います。

図 14 debtags タグ付け web インターフェイスの様子

- パッケージメンテナの人向け

  まず、 `http://debtags.alioth.debian.org/todo.html?maint=<your_mail_address>` にアクセスしてください。メンテナンスしているパッケージとつけられているタグの一覧が表示されます。自分のパッケージをいい状態にメンテナンスする作業の一環ですよ！ 忘れないで。
- ユーザの方向け

  debtags のサイト (`http://debtags.alioth.debian.org/todo.html`) にアクセス、 Current View を full text search にして自分が良く使っているパッケージの名前を入れます。特に debtags grep で検索してみて「このパッケージが何でこのキーワードで引っかからないんだ！」というのがあれば、それは要改善点なわけなので入力してみるのが良いでしょう。

10.5 実際のタグ付け

適当なパッケージを選んだらタグ付けに入りましょう。サイトの画面は大きく４つに分けられます (図 15)。

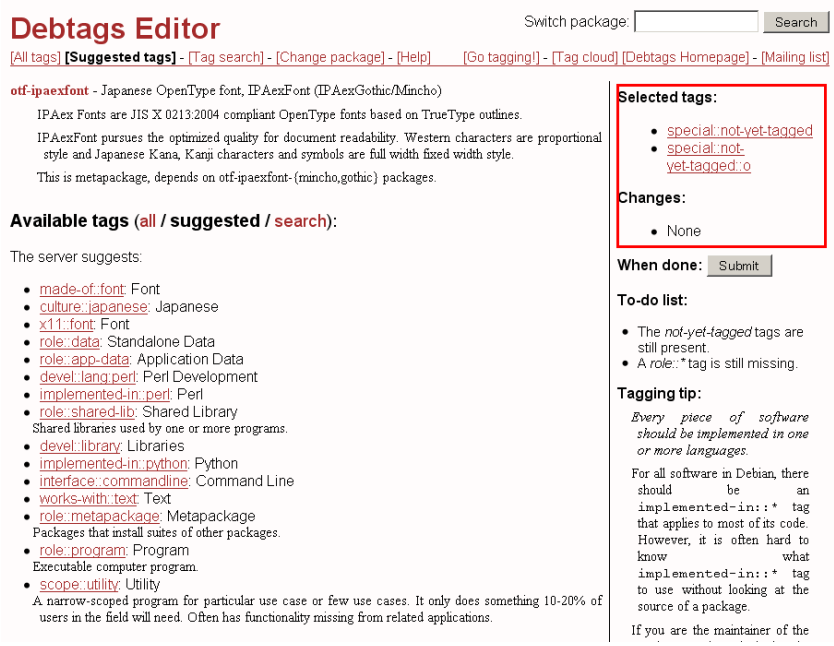


図 15　debtags タグ付け web インターフェイスの様子

- 選択したパッケージの説明(画面左上)
- 利用可能なタグの区別:　all (すべて) / suggested (おすすめ) / search (検索)
- タグの一覧(画面左下)
- 既に付けられた・付けられるタグ(画面右)

10.6 修正が必要なタグ

まだキチンとタグ付けがされていないパッケージには、赤い×印と共に以下のような注意が表示されます。ここからまず直していくことを考えましょう。

それぞれ以下のような意味合いです。

表示される注意	意味合い
The not-yet-tagged tags are still present.	このパッケージはまだタグ付けが終わってないよ、のタグが残ってます。
An implemented-in::* tag seems to be missing.	このソフトはほげほげ言語で実装されています、ってタグ付けしようね
A role::* tag is still missing.	このソフトの役割をタグ付けせよ(role は必須です)
A devel::lang:* tag seems to be missing.	ほげほげ言語開発用のタグを付けましょう

表 2　タグ付けがされていないパッケージの注意書き

これを踏まえて以下のような作業をします。

- まだパッケージを指定していないなら、パッケージ名をクリック
- 提示されるタグや検索したタグをクリックして追加
- 必要ないタグが付けられている場合はクリックして削除
- 画面右側の「 Selected tags:」と「 Changes:」を見て、問題がないことを確認図 16
- 最後に submit

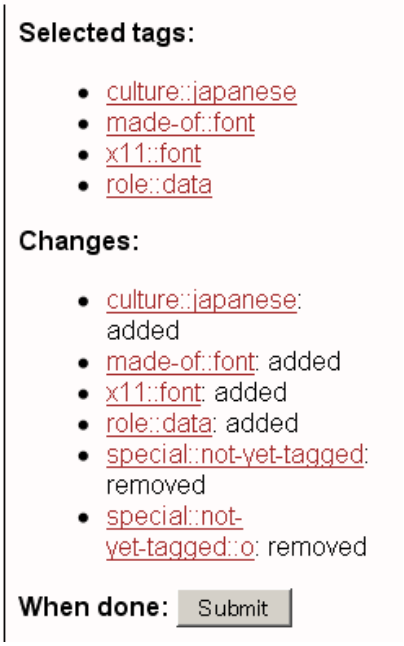


図 16　画面右側の「 Selected tags:」と「 Changes:」を確認!

これだけで作業は終わりです。簡単ですね!

10.7 どんなタグがあるの?

タグ付け自体は簡単なものなので、パッケージに対して適切なタグを付けることが肝要なのですが、すべてのタグを覚えることは大変すぎるのであきらめて、サイトが適宜提示してくれるものの中から選択しましょう。あと一応ガイドラインも

あります (http://wiki.debian.org/DebTaggingGuidelines)
で、最初に「推奨」タグが表示されています。適当なものがあればここから選ぶのもいいですが、お勧めは

- 他の似たパッケージを見てみる → 同じタグ使う
- all を選んで、検索窓からキーワードでタグを検索する

です。

10.8 その他疑問点

- 悪意のあるコミットについては? spammer などは大丈夫か
  コミットされたタグについては一応ブラウザのクッキーが紐付けられていて、レビュー時に重宝している模様です。
- コミットは誰でもできる? レビューは?
  レビューするには Alioth の 'debtags' グループに参加する必要があるそうです。また少なくとも Debian を使っていないと、レビューの分類をする際、細かな点でうまく判断できないだろうということでした。

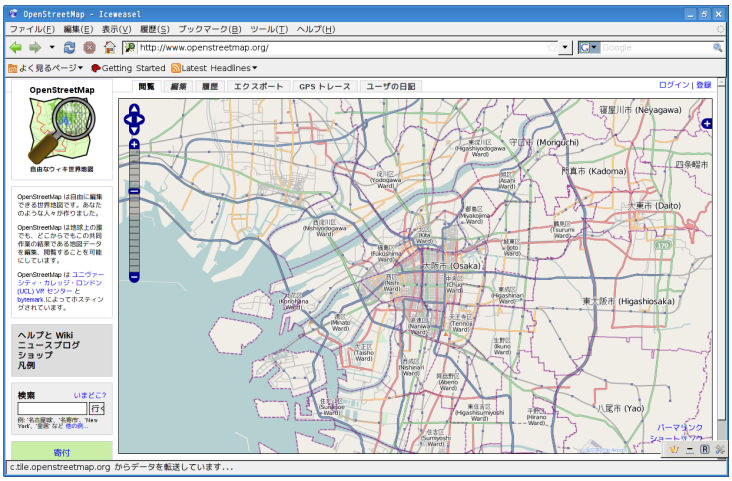
10.9 最後に

Happy tagging!

11 Debian を使って愉しむ Open Street Map 入門

たなかとしひさ

11.1 Open Street Map をご存知ですか?



2009 年度 12 月の Debian 勉強会のお知らせで、 `http://www.openstreetmap.org/` にアクセスして、お住まい近辺の地図を見て頂けたらと案内しました。

ご覧になられた皆さん、感想はいかがでしょうか。

「 すげぇ、素晴らしい! 」

「 Google Map の廉価版? 」

「 Debian と何の関係があるのさ? 」

等など、色々な感想があると思います。

今回の勉強会では、 Debian の応用として、 Debian を使って、この OpenStreetMap(以降、 OSM と表記) について、一緒に勉強していきましょう。

11.2 免責

- このテキストは、たなかとしひさ (tosihisa@netfort.gr.jp) が書いたものです。
- このテキストには、間違いがあるかもしれません。
- 記載内容は、できるだけ最新 (2009 年 12 月) の事情にあわせて記載したつもりですが、時間の経過で内容が変化する場合があります。

11.3 OpenStreetMap(OSM) って何ですか?

http://wiki.openstreetmap.org/wiki/Ja:Main_Page からの引用です。

> OpenStreetMap は道路地図などの地理情報データを誰でも利用できるよう、フリーの地理情報データを作成することを目的としたプロジェクトです。自由に使えると思っている地図の多くが実は法的・技術的に問題があり、人々がクリエイティブに、生産的に、あるいは今まで予期しなかった方法でそれを 利用する事を妨げているため、このプロジェクトは開始されました。

11.4 Debian と OSM

今回の関西 Debian 勉強会は、従来のテーマとは異なり、異色でもあります。「 OSM って、 Debian と何か関係あるの? 」と言われると、確かに強い関係…はありません。

しかし、 Debian と OSM を組み合わせる事で、

「 自由度の高い」地図ソフト環境

が実現できます。これは、画期的な事だと筆者は考えています。

Debian で、どれだけ素晴らしい地図ソフトが .deb になって apt で得られるとしても、地図データが無ければ魅力を欠きます。

Debian を始めたとした Linux ディストリビューションは、「 自由に使う事が出来るコンピュータソフトウェア環境」を実現できるものですが、それと OSM とを組み合わせる事で、自由に使う事が出来るコンピュータソフトウェア環境の中に、「 地図の閲覧」を含める事が出来ます。

Debian もそうであるように、 OSM もまた、「 一部の特権階級」のものではありません。望めば、誰でもが自由に使う事が出来ます。 OSM は、使うには一苦労かかる事もしばしばありますし、地図自身、日本国内では十分に揃っているとは言えない状況です。

しかし、誰でもが、自由に使える地図データを提供できる事は、 Debian が目指すゴールと重なります。

また、「 単に Debian を使う」だけでなく、 Debian の応用事例を増やす事は、 Debian コミュニティに取っても有益と考えています。 Debian は、サーバ、医療、組込みなど、様々な分野で使われています。その中に「 自由に使える地図環境」を加える事が出来ます。

11.5 OSM のライセンス

OSM の地図データは、 Creative Commons Attribution-ShareAlike 2.0、簡略系で書くと CC BY-SA (表示-継承) です。[*49]

なお、 OSM の地図データのライセンスは、 2009 年 12 月現在 Open Database License (ODbL) への移行が検討されています。[*50]

12 月の関西 Debian 勉強会実施頃には、もしかするとライセンスが変更されているかも知れません。

＊急募＊

日本の OSM コミュニティは、 ODbL に関する情報を必要としています。 ODbL に詳しい (出来れば) 日本語の情報がありましたら、筆者までお知らせ下さいますと助かります。

ODbL は、日本ではまだ認知が低いためか、日本語の情報が少なく、どの様な情報でも構いませんので、筆者までお知らせ下さいますと助かります。

[*49] http://wiki.openstreetmap.org/wiki/Ja:OpenStreetMap_License
[*50] http://wiki.openstreetmap.org/wiki/Ja:Open_Database_License

11.6 OSM の特徴

11.6.1 OSM の地図データは、ビットマップではなくベクトルデータ

OSM の地図データはベクトルデータです。 http://www.openstreetmap.org/ や、 http://osm.jp/ から参照できる地図画像は、そのベクトルデータをビットマップデータへレンダリングしたものです。

地図データがベクトルデータなので、地図の表現能力自身は、ビットマップに比べると劣りますが、ベクトルデータの場合、拡大・縮小が自由に行える事と、ルート探索が可能になります。

OpenStreetMap の地図データを用いたルート探索サービスを提供しているサイトの一つに、 CloudMade があります。 CloudMade の地図サイト (http://maps.cloudmade.com/) にアクセスして、大阪 (梅田) から、関西 Debian 勉強会までのルート探索結果を図 17 に示します。

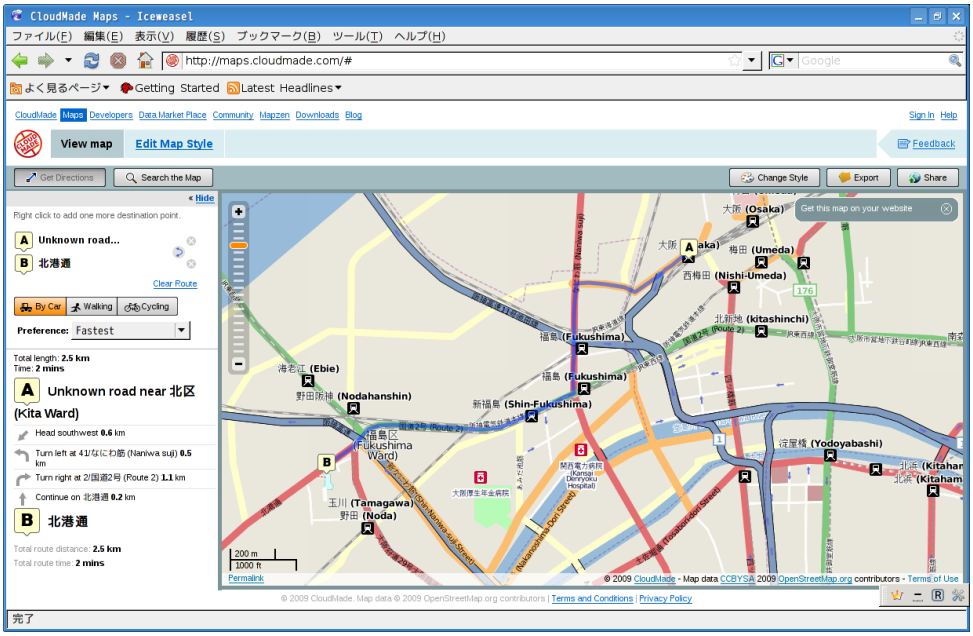


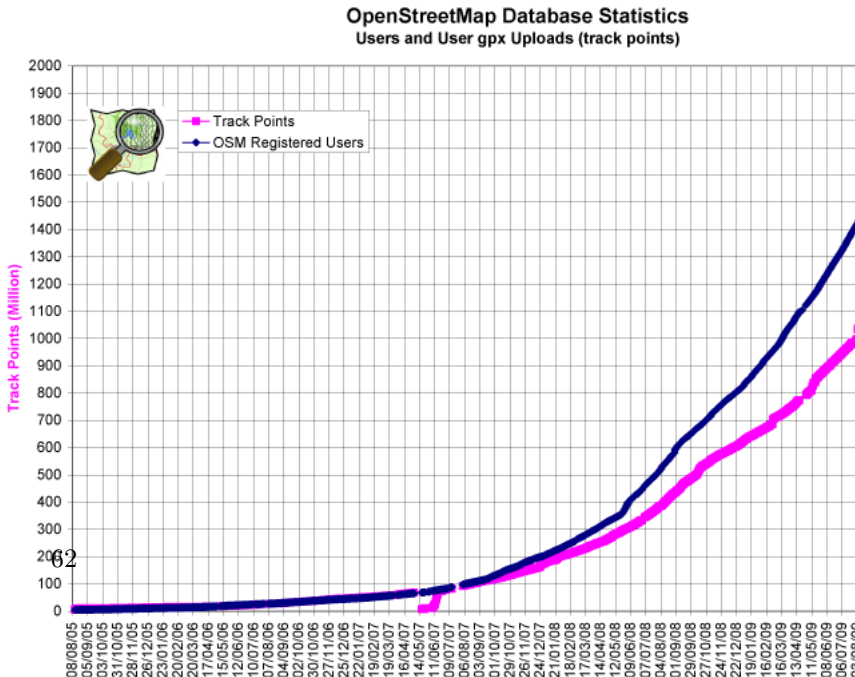
図 17 CloudMade による関西 Debian 勉強会へのルート探索図

OSM の地図データ自身がまだまだ不足している事と、地図データの精度が足りないので、期待したルート探索結果にはまだならないかも知れませんが、この様なルート探索が行える可能性を OpenStreetMap は持っています。

11.6.2 地図データの作成は、アカウント登録さえすれば誰にでも可能

OSM の地図データの作成は、 OSM へのアカウントを登録すれば誰にでも作成、編集が可能です。

2009 年 5 月に、 OSM ユーザ数は 100,000 を越えました。右に、その登録者数と作図データの推移グラフを示します。(出典: http://www.opengeodata.org/2009/03/17/osm-passes-100000-users/)



11.6.3 Wiki の考え方をベースにしている

地図データの作成は、 Wiki の考え方が根本にあります。 A さんが作図したデータを B さんが修正する事が可能です。 Wiki の悪い面 いわゆる「 荒らし」的な事も、やろうと思えば出来てしまいますし、地域によっては「 編集合戦」があるのも事実です。

しかしながら、 Wiki ベースであるので、「 より良い方向に向かって修正する」事が可能であり、例えば一方通行の方向が逆である事を見つけた場合、アカウントを持っていれば修正する事が出来ます。

11.6.4 地図データはオフラインでも利用できる

GoogleMap は実に便利ですが、基本はオンライン、要するにインターネットに繋がっている環境下である事が前提にあります。また、 GoogleMap が提供する地図は、事前に Google の書面に同意を得る事無しに独自の技術によるアクセスや地図の複製は出来ません。

出典:

- `http://www.google.co.jp/intl/ja_jp/help/terms_maps.html`
- `http://www.google.com/intl/ja_ALL/help/terms_local.html`

これは、オフライン対応地図閲覧ソフトは、 Google Map の地図データを書面による同意無しに使えない事を意味します。オフラインでも地図を閲覧できるソフトに Mobile GMaps (`http://www.mgmaps.com/`) があります。これはオンラインでもオフラインでも使用できる PDA や SmartPhone 向け地図ソフトですが、上記の理由から、このソフトは Google Map の地図に対応していません。

誤解の無い様に付け加えると、筆者は GoogleMap のあり方は問題視していません。 GoogleMap が提供するサービスは、これらを補うに十分と考えています。

大事な事は、インターネットがどこでも安価に使える様になったとは言え、それは「 全世界から見ればごく一部」でしか無いと言う事です。日本でも、山間の地域に行くと、携帯も圏外になる場合があります。

この様な場合でも、地図データをオフラインで持っておけば、圏外でも地図を閲覧できますし、圏内でもパケット課金を気にする必要はありません。

OSM の地図データをオフラインで見られるソフトの一つに、 navit (`http://wiki.navit-project.org/index.php/OpenStreetMaps`) があります。 navit は後の章で紹介します。

11.7 (今だけの愉しみですが) 走行ログになります

筆者はバイクに乗り、アチコチに移動するのが趣味で、 GPS を持って出かけてログを取って後で眺めたりします。その時、「 単にログを見る」だけではなく、「 そのログを元に地図を作図する」事が出来れば、さらにより良いと考えています。

筆者が OSM に地図データをコミットし始めた頃は、大阪は殆ど何もありませんでした。記憶ですが、名神か阪神高速の一部があった程度です。どなたか、大阪を通過された方が作図されたのだと思います。

筆者はまず、大阪のシンボル御堂筋を作図しました。続けて、四ツ橋筋を作図しました。当時は「 広大な白地図」でしたので、どこを走っても作図できましたが、最近の大阪の OSM 地図の発展は目覚しく、少し考えてログを取らないと、誰かが既に作図済みの所を走っているだけになります。

近畿地方で最も OSM の作図が進んでいるのは、滋賀県長浜市の OSM 地図と考えています。滋賀県長浜市の OSM 地図発展は目覚しいもので、「 よくぞここまで作図したものだなぁ・・・ 」と感嘆する事しきりです。

筆者は、「 自由に使える地図を【 使いたい】 」と言う理由で OSM に注目し始めました。が、ミイラ取りがミイラと言う訳ではありませんが、目的が「 自由に使える地図を【 作ること】 」に変ってきているのも事実です。

11.8 OSM へのコミットは、地域社会への貢献にもなりえます。

筆者は常々、事 OpenSource の成果は、地元社会にもつながればと考えています。

プログラミング言語 Ruby は、島根県の IT ならびに OpenSource 促進に良い影響を与えたと考えています。 OSM は、それと同じ効果を持ちえると考えています。

「 地図」と言うのは、 OpenSource コミュニティに限らず、誰にでも一応の興味があります。私の母はコンピュータ環境と無縁な生活を送りつづけていますが、母は山歩きが趣味なので地図帳は持っています。

また、日本は地震が多いので、避難場所への地図を載せた看板を目にすると思います。筆者は、 OSM が地震等の災害発生時に役に立つ日が来ればと考えています。地震等の災害で、道路が分断された場合、どこからどこまでが通行不可なのかどうかを、迅速に反映できる仕組みを、 OSM は持っていると考えます。

また、視力が弱く、地図を見る事が出来ない場合にも、 OSM の地図をベースに「 触地図」を作ってみたケースがあります。

現在の OSM の地図データは、日本国内で見れば、まだまだ足りないのが現状ですが、「 地図」は殆どの人には、少なからず関係があるものですので、

この様に、様々な形での応用を OSM は持っています。

11.9 OSM への参加

OSM への参加は、 `http://wiki.openstreetmap.org/wiki/Ja:Beginners_Guide` を参考にすると良いでしょう。基本的には下記の事をしていきます。

11.9.1 OSM アカウントの作成 (初めの 1 回だけ)

`http://www.openstreetmap.org/create-account.html` にアクセスして、 OSM アカウントを作成します。同時に、 `http://wiki.openstreetmap.org/index.php?title=Special:UserLogin&type=signup&uselang=ja` にアクセスして、 OSM の Wiki アカウントを作成しておくと良いでしょう。

OSM は、地図データのアップロードは「 OSM アカウント」を用いますが、それ以外にも OSM の Wiki ページ (`http://wiki.openstreetmap.org/`) がありますので、 OSM に関する情報公開に用いると良いでしょう。

アカウントの作成は初めの 1 回だけですが、後々の GPS ログのアップロードや作図では OSM アカウント情報が必要です。

OSM 作図の流れを図 19 に示します。

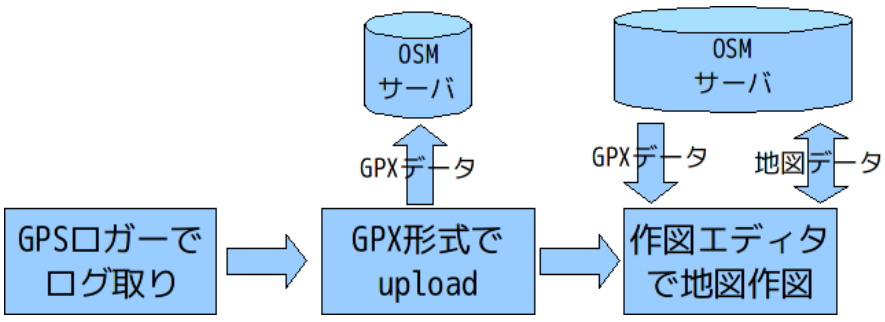


図 19　OSM 作図の流れ

11.9.2 GPS ロガーでログ取り

GPS ロガーを持って、まだ地図データの無い白地図の所に行き、 GPS データをログしていきます。

「 マッピングパーティ」と言う催しがあります。これは、 OSM 同好の集まりで GPS 等を持ってログを取る催しです。この「 マッピングパーティー」に参加するのも良いでしょう。

11.9.3 GPS ログデータをアップロードする

GPS ロガーのデータを GPX 形式にして、 OSM サーバにアップロードします。

技術的には、 GPS ログデータをアップロードしなくても作図そのものは可能ですが、「 GPS を元にした道である事」への根拠として、 GPS ログデータはアップロードしておいた方が良いでしょう。

11.9.4 地図データを作成、編集する

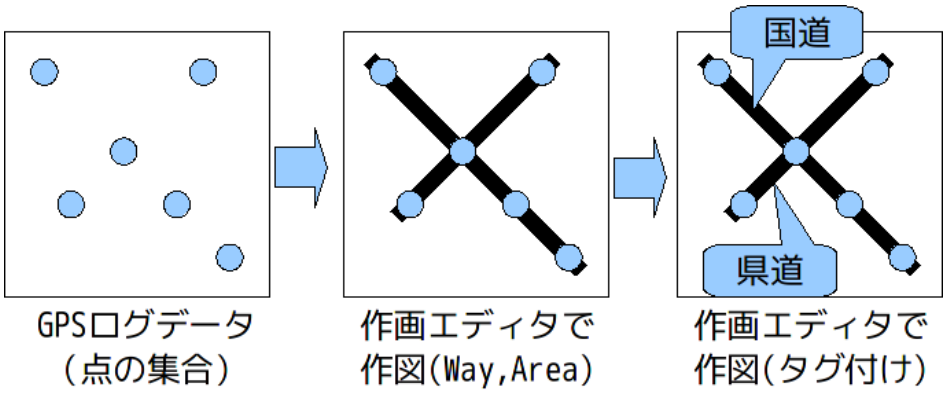


図 20 作図イメージ

GPS ログデータを元に、 OSM 作画エディタを使って、実際に作図していきます。

GPS ログデータは、点データの集合ですので、それを OSM 作図エディタで繋いで行き、線にしていきます。

次に、その線データが国道なのか、県道なのか、一方通行かどうかの情報を付加していきます。これを、「 タグ付け」と言います。作図のイメージを図 20 に示します。

11.9.5 マップを描画する!

出来上がった地図を見てみましょう! 地図のレンダリングには少し時間がかかりますが、早ければ 1 時間位でレンダリングされます。

11.10 OSM へ参加するには、 GPS ロガーは不可欠なの?

GPS ロガーが無いからと言って、 OSM に参加できない事はありません。 GPS ロガーが無くても、下記の形で OSM へコミットできます。

- 使ってみる。
  OSM は地図データですので、実際にそれが現実とあっているかが重要です。とにもかくにも、 OSM を使ってみて下さい。 Debian と同様に、「 それを使う」だけでも、貢献として充分なのです。
- 間違いを修正する。
  OSM の作図は、できるだけ正しくなるように作図が進んでいますが、例えば一方通行の方向が逆だったり、国道／県道の番号が間違っている場合もあります。この様な場合、作画エディタで OSM データをダウンロードできますので、間違いのある部分をダウンロードして修正してアップロードすれば、間違いが修正できます。

11.11 Debian で使える OSM (GIS) 関連ソフト

Debian は、メンテナの尽力により豊富なバイナリパッケージを使う事ができますが、 OSM 関連で利用できるソフトウェアを下記に示します。もちろん、これだけではありません。 私見ですが、 GPS を扱うソフトは、 Windows よりも Debian の方が多岐に渡っていると感じています。

11.11.1 gpsd

```
# apt-get install gpsd
```

gpsd は、 OSM では必須では無いのですが、 Debian や Linux で GPS データを受信する際にほぼ標準として使われているので紹介します。このソフトは、 GPS 受信機と PC を接続し、 NMEA-0183 センテンスまたは GPS の独自プロトコルと通信し、現在の緯度経度、 UTC 時間を処理します。

Debian で動作する地図関連のソフトは、 GPS 受信機と直接通信せず、 gpsd を経由して緯度経度の情報を得るものが多いです。 gpsd を経由させる事で、 GPS 受信機一つに対し、複数の (GPS を必要とする) ソフトが使えるようになります (図 21)。

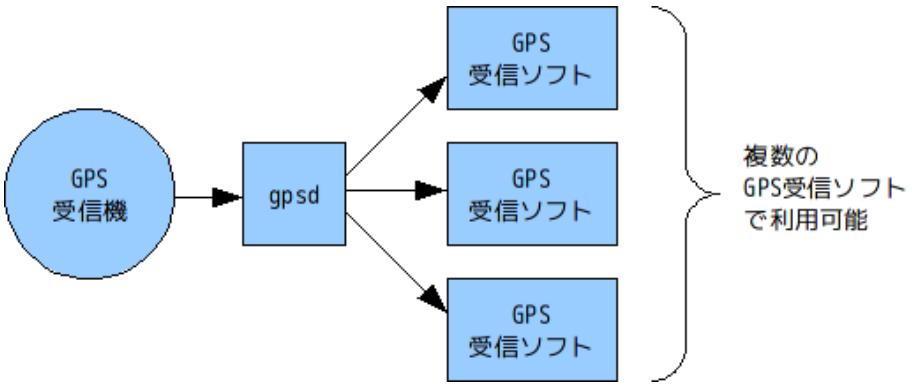


図 21　gpsd の動作イメージ

また、 gpsd は、 NTP サーバ向けの時刻源としても機能します。 NTP(ntpd) とは、共有メモリドライバを介して行います。ただし、 gpsd を NTP サーバ向けの時刻源にしても、 NTP 階層の Stratum 1 の精度になるわけではありません。これは、 gpsd が時刻情報を受信する時、共有メモリに書き込むときに「 ゆらぎ」が生じるからです。

Debian と OSM からは少し外れますが、もし、 GPS を時刻源として Stratum 1 相当の NTP サーバを作るならば、”1PPS (one pulse per second)” 出力つき GPS が必要になります。

1PPS とは、「 正確に 1 秒のパルス」を発生するもので、このパルスを Linux カーネルで捕まえる事で、時刻のゆらぎを少なくします。

11.11.2 gpsbabel

```
# apt-get install gpsbabel
```

gpsbabel は、様々な GPS データの形式を変換するソフトです。

GPS データの「 基本的な仕様」としては、 NMEA-0183 センテンスがその基本なのですが、 GPS 受信機ベンダーは、独自フォーマットでデータを保存する場合があります。それは様々な GPS ベンダーから、様々なデータ形式があります。 Google Earth の“ .kml ” 形式も、その GPS ログの保存形式の一つです。

gpsbabel は、その GPS データ形式を変換するソフトです。

OSM が採用している GPS ログ形式は、¡time¿タグ付き GPX 形式のみですので、もし GPS 受信機、あるいは付属ソフトが¡time¿タグ付き GPX 形式に対応していない場合、 gpsbabel で変換しなければならない場合があります。

参考に、 NMEA-0183 センテンスの GPS ログデータを GPX に変換するには、下記の様に実行します。

```
$ gpsbabel -w -r -t -i nmea -f {nmea-log-file} -o gpx -F {GPX_DATA}.gpx
```

11.11.3 Merkaartor (発音は" メルカトル" です)

「deb http://www.backports.org/debian lenny-backports main contrib non-free」を、/etc/apt/sources.list に追加します。

```
# apt-get update
# apt-get -t lenny-backports install merkaartor
```

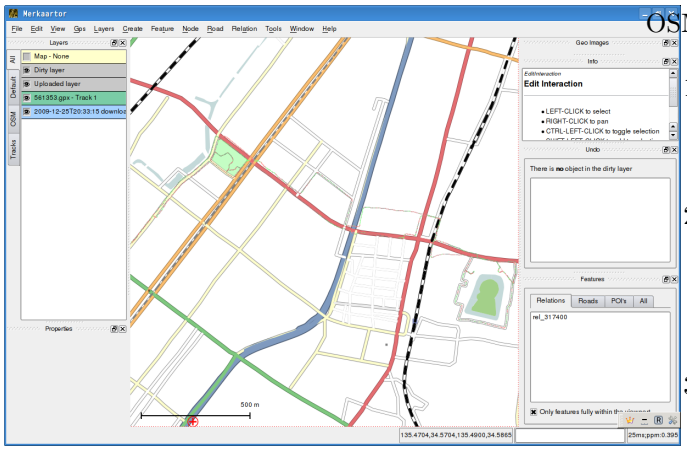


図 22　Merkaartor イメージ

Merkaartor(図 22 は、 OSM の作図ソフトです。 OSM 向けの作図ソフトは、大きく 3 つあります。

1. Potlatch
   Potlatch は、フラッシュベースの作図ソフトで、 Web ブラウザがあれば使う事が出来ます。
2. JOSM
   JOSM は、 Java で書かれた作図ソフトで多機能です。 Java 版なので、 Debian でも Windows でも動きます。
3. Merkaartor Merkaartor は、 Qt ライブラリを使って書かれた作図ソフトです。 JOSM と同じく Debian でも Windows でも動きます。

どれがお奨めか、これは 3 つとも自由に使えるソフトですので、 3 つ試して一番自分に合うものを選んで下さい。一概にどれが良い・悪いと言うのはありません。ただ、私見ですが、 Merkaartor は多機能で無い分、初心者にとってはかえって分かりやすいと考えています。

Debian lenny に入っている Merkaartor は少しバージョンが古く、 Debian Backports サイト (http://www.backports.org/) から、出来るだけ新しい Merkaartor をインストールすると良いでしょう。

11.11.4 navit

「deb http://navit.latouche.info/debian lenny main」を、/etc/apt/sources.list に追加します。

```
# gpg --recv-keys CB229096
# gpg --export -a CB229096 | apt-key add -
# apt-get update
# apt-get install navit
```

navit(図 23) は、 OSM 地図データに対応した地図表示ソフトです。 gpsd が必要で、 gpsd から現在の緯度経度を取得し、 OSM 地図データと重ね合わせ表示する事が出来ます。 navit は OSM 地図データを事前に取り込む方法ですので、ネットが使えないオフライン環境下でも地図を見る事が出来ます。

筆者は Debian Lenny をインストールした EeePC に、 gpsd と navit をインストールし、大阪から新潟まで使ってみたことがあります。実際の走行は、殆どが高速道路ですので、ナビと言うほどのものはありませんでしたが、私が念願としていた、「自由な OS で、自由なソフトで、自由な地図データでナビをする」と言う事が達成できた瞬間でした。

筆者はまだまだ、 Debian は初心者の域を出ません。もし Debian にあるパッケージで、良いものがありましたら紹介下さい。

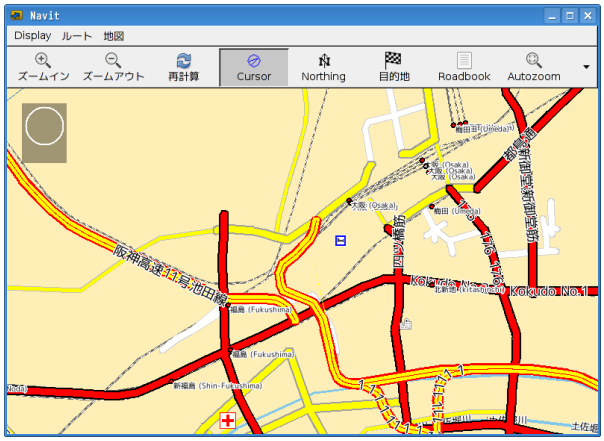


図 23　navit イメージ

11.12 筆者の GPS ログ機器

筆者は、主に 2 台の GPS 受信機を使ってログを取っています。

11.12.1 GT-31

GT-31 は、小型防水の GPS ロガーで、バイクに取り付けることが出来ます。

GT-31 の良い所は、 GPS ログを SD カードに保存する事が出来るので、記録容量が SD カードの容量次第で大きく出来る点です。また、 IPX7 相当の防水性能があります。 IPX7 相当の防水機能とは、 1m の水中に、 30 分間沈んでいても内部に水が入らない構造を指します。

筆者は GT-31 の SD カードに、 NMEA-0183 の形式で保存しています。

11.12.2 HI-406BT

GT-31 と併用して、 HI-406BT も使っています。

HI-406BT は、純粋な GPS 受信機であり、この機器自身にログ機能はありません。 HI-406BT は Bluetooth インターフェースなので、筆者は Bluetooth 付きの PDA や PC に、これも NMEA-0183 の形式で保存するようにしています。

11.12.3 HOLUX m-241

HOLUX m-241 は、単 3 乾電池一つで動作する小型 GPS ロガーです。

小型でありながら、ロガーとしての記録容量が大きく、 130,000 点のログが可能です。加えて Bluetooth に対応しています。筆者が OSM に作図し始めた頃は、この m-241 でログを取っていました。残念な事に、筆者の m-241 は壊れてしまい、今は GT-31 と HI-406BT を併用しています。

GPS は、余裕があれば 2 台欲しいです。白地図を作図するときは、やはり GPS ログから作図しますので、ログ取り忘れはかなり寂しい事になります。

GT-31 は、 GPS ログしない。と言う選択肢が無く、常にログします。 m-241 は、ボタンのトグルでログする・しないを選択できますが、ついうっかり忘れてしまう事がありますので、 GPS ログを取る・取らないが選択できる GPS 受信機を使う場合、 GPS ログ状態がすぐに確認できるものを選ぶと良いでしょう。

11.13 GPS ロガー選び方ノウハウ

11.13.1 DOP (Dilution of Precision - 精度低下率) が分かるものを選ぼう

GPS レシーバは、「 GPS 衛星から現在位置をもらう」のではなく、 GPS 衛星から正確な時刻と GPS 衛星の移動情報を元に、「 GPS レシーバが自力で現在地を計算する」仕組みです。

そのため、計算結果から、精度がどれくらい低下しているかを判断する事が出来ます。これは GPS 衛星一つから現在地を知るのではなく、最低 3 つの GPS 衛星から現在地を計算するためです。

DOP は、小さければ小さいほど精度が良い事を示します。この DOP は、受信した (計算に選択した)GPS 衛星のばらけ具合によります。 DOP には、水平方向の HDOP、垂直方向の VDOP 、位置を示す PDOP があります。

下記のページに記載がありますが、 OSM は、 PDOP の値は 4 より下、 2 より下ならかなり良く固定されているとあります。

`http://wiki.openstreetmap.org/wiki/Ja:Recording_GPS_tracks`

筆者自身、作図のために GPS ログを取る時は、 DOP 値には相当気を使っています。

GPS ロガー ”GT-31” は、 DOP 値を表示することが出来ますので、筆者が GPS ログを取る時は、現在の緯度経度よりも、 DOP 値を表示させるようにしています。 DOP をログする GPS ロガーを使用した場合、 GPX ログにも DOP を残す事が出来ますので、 JOSM や Merkaartor でも視覚的に確認できます。

NMEA-0183 センテンスの場合、 GSA センテンスがこれにあたります。

11.13.2 外部アンテナが付けられるのと付けられないのでは大違い

GPS 受信機には、外部アンテナを取り付ける事が出来るものがあります。筆者は ”HI-406BT” と言う Bluetooth GPS レシーバも使うのですが、これには外部アンテナを取り付けることが出来ます。

外部アンテナを取り付けることで、 GPS 信号の感度が上がり、精度が向上します。また、外部アンテナは概ね防水型ですので、外部アンテナを車の屋根に取り付け、天候に左右されずにログを取る事が出来ます。

11.13.3 測位精度の誤差

最近の GPS 受信機は精度が向上し、概ね「 10m 2drms」の範囲です。「 2drms」と言うのは、これを半径とする円内に、およそ 95% の測位点が入る事を表します。 10m 2drms ならば、半径 10m の円内に 95% の測位点が入る事を示します。

半径 10m だと、地図として使うのに誤差としては大きいかも知れませんが、 DGPS だと 5m 2drms にまで誤差が少なくなるものもあります。 GPS 受信機を購入する時は、この 2drms の値は確認しておくと良いでしょう。

11.13.4 バッテリーの持ちと形状

１時間程度のログであればあまり問題はありませんが、バイクツーリング等でほぼ１日乗りっぱなしで GPS ログを取る場合、 GPS 受信機のバッテリーも気になります。

GPS 受信機のバッテリーの持ちもそうですが、バッテリーの形式、例えば乾電池か専用電池かも、利用形態に応じて考える必要があります。

最近の個人用 GPS 受信機は、 USB で充電できるものがあります。これと携帯の充電器にあるような、乾電池の電力を USB に変換して出す充電器を併用する事で、専用電池でもバッテリーを気にせずに使う事が出来ます。

せっかくの GPS 受信機も、バッテリーが干上がると、 mapper にしてみるととても寂しい事になりますので、バッテリーの持ちや、形状は確認しておくと良いでしょう。

11.14 おわりに

OpenStreetMap は、日本ではまだまだ認知の低いプロジェクトですが、 Debian System とも十分親和性の高いプロジェクトですので、興味を持って頂けたならうれしいです。

12 ハンドメイド GPS ロガーの構築

まさ

12.1 はじめに

これまで Debian のインストール、環境整備、ウェブアプリケーションのシステム構築をするなどシステムを触ることはある程度経験したものの、お小遣い帳サーバや、 GPS ロガーなど、何らかの実用になるものはなかなかつくれませんでした。

そこでプログラムのスキルを得るための教材として、 GPS ロガーを作成したので、報告します。

これから Debian を目指す人の興味を惹き、手がかりとなって少しでもお役に立てれば幸いです。

12.2 GPS とは

“Global Positioning System” は数十個の衛星からの電波を受信して現在位置を割り出すシステムです。

原理等はいろんな書籍やインターネットに掲載されているようですので、ここでは説明を割愛します。

12.3 GPS ロガーのハード構成

必要なハード構成は以下の通りです。

今回実際に使用したハードの概略仕様は Appendix を参照してください。

ネットブック ⟸ PS2-USB 変換 ⟸ GPS レシーバ

1. GPS レシーバ
   GPS アンテナ、受信回路、信号送出回路から成り、 PS2 コネクタからシリアルにキャラクタ文字で GPS 情報が送出されます。
2. PS2-USB 変換
   GPS レシーバから信号送出用のシリアルインタフェースを USB インタフェースに変換するためのものです。変換ケーブルになっています。
3. ネットブック
   USB インタフェースで受信した GPS 情報を受け取り、ファイルに溜め込みます。また、 GPS 情報をいろいろ加工して楽しめます。

12.4 ネットブックセットアップ概要

ネットブック購入直後から GPS ロガーを構築する場合について概要を説明します。

1. Debian Base system のインストール
   Debian-netinst-iso を使って Base system をインストールします。
2. X Window System のインストール
3. Gnome のインストール
4. 日本語入力システムのインストール
   プログラム作成時必要です。今回は、“uim-anthy” をインストールしました。
5. NTP のインストール
   システム時刻自動調整用のプログラムです。今回は GPS を扱うため、特にシステム時刻が大切になると思われます。
6. プログラム言語のインストール
   GPS ロガーアプリケーション作成に必要です。今回は、とっつきやすく、 GUI ができ、ライブラリが豊富と言われる “Python”（ バイトン or パイソン? ）を選択しました。私にとって、確かにとっつきやすいですが、それなりに苦労しました。 python のせいではなく、プログラミングというもの自体、思ったものがそう簡単にできるわけではないということだと思われます。
7. Python library のインストール

（ a） PySerial
   シリアルポート入出力用のプログラムです。

（ b） Python Tkinter
   GUI プログラミング用のひとつです。

（ c） datetime
   時刻を扱うとき使います。

（ d） os, os.path
   ファイルの開閉、読み書きするとき使います。

（ e） sys(場合によって使用)
   コマンドパラメータを取得するとき使います。

12.5 GPS データ出力形式

今回使用した GPS レシーバは NMEA0183 というプロトコルです。 NMEA0183 では以下のような形式などでデータが出力されますが、

GLL Geographic Position, Latitude and Longitude
GSA GNSS DOP and Active Satellites
RMC Recommended Minimum Specific GNSS Data

今回は

GGA Global Positioning System Fix Data

という形式のみ利用しました。

GGA 形式の仕様は次の通りです。

```
GGA,123519.00,4807.038247,N,01131.324523,E,1,08,0.9,545.42,M,46.93,M,5.0,1012*42
123519.00          =  測位時刻( UTC)    12:35:19.00
4807.038247,N        =  緯度   48 度 07.038247 分( 北緯)
01131.324523,E       =  経度   11 度 31.324523 分( 東経)
1          =  GPS のクオリティ;  0 = 受信不能、 1 = 単独測位、 2 = DGPS
08         =  受信衛星数
0.9        =  HDOP
545.42, M        =  平均海水面からのアンテナ高度( m)
46.93, M         =  WGS-84 楕円体から平均海水面の高度差( m)
5.0        =  DGPS データのエイジ( 秒)
1012       =  DGPS 基準局の ID
*42        =  チェックサム
```

ここから、時刻、緯度、経度などのデータを利用しました。

12.6 アプリケーション作成

いよいよアプリケーションの作成です。上の GPS GGA データを加工して、表示したり、ログをとります。

12.7 実行

アプリケーションを走らせます。

12.8 今後の抱負

今回のアプリケーションを基にいろんな展開が考えられます。 Sunday programmer として今後も少しづつであっても、根気よく前進します。

1. Open Street Map への貢献
2. Open Street Map の利用

地図上への現在位置の表示や、ログデータからトレース表示したりと地図情報は重要です。この場合、種々の縮尺の地図が必要ですが、 Open Street Map はベクトルデータであることからこの点が有利です。また、ナビのように指定した位置の地図を得る場合、これが可能なインタフェースが OSM にあれば(あるのかも知れませんが) 、より柔軟に利用できるのではないかと期待しています。

Appendix

[A] ネットブック仕様概略

```
PC:       Asus EeePC 901
CPU:      Intel Atom (1.6GHz)
Memory:   1GB
HDD(SSD): 4GB + 8GB
Monitor:  8.9 型、 1024 x 600
I/F:      USB 2.0
```

[B] ネットブックシステム仕様

```
OS:      Debian Lenny
Kernel:  2.6.26-2-686
言語:    Python 2.5.2
Library: pySerial, tkinter 等
```

[C] GPS レシーバ仕様

```
型式:       GPS-S103(Wonde proud 社製)
位置 精度:  5 - 25m CEP without SA
時刻精度:   1 μ s synchronized to GPS time
プロトコル: NMEA0183
動作温度:   -40 ° C ~ +85 ° C
動作湿度:   5% to 90% ( 結露なき事)
電源:       DC 3.7 ~ 6V, typical 5V ( 今回は USB より供給)
I/F:        TTL serial(PS2 コネクタ). USB option
```

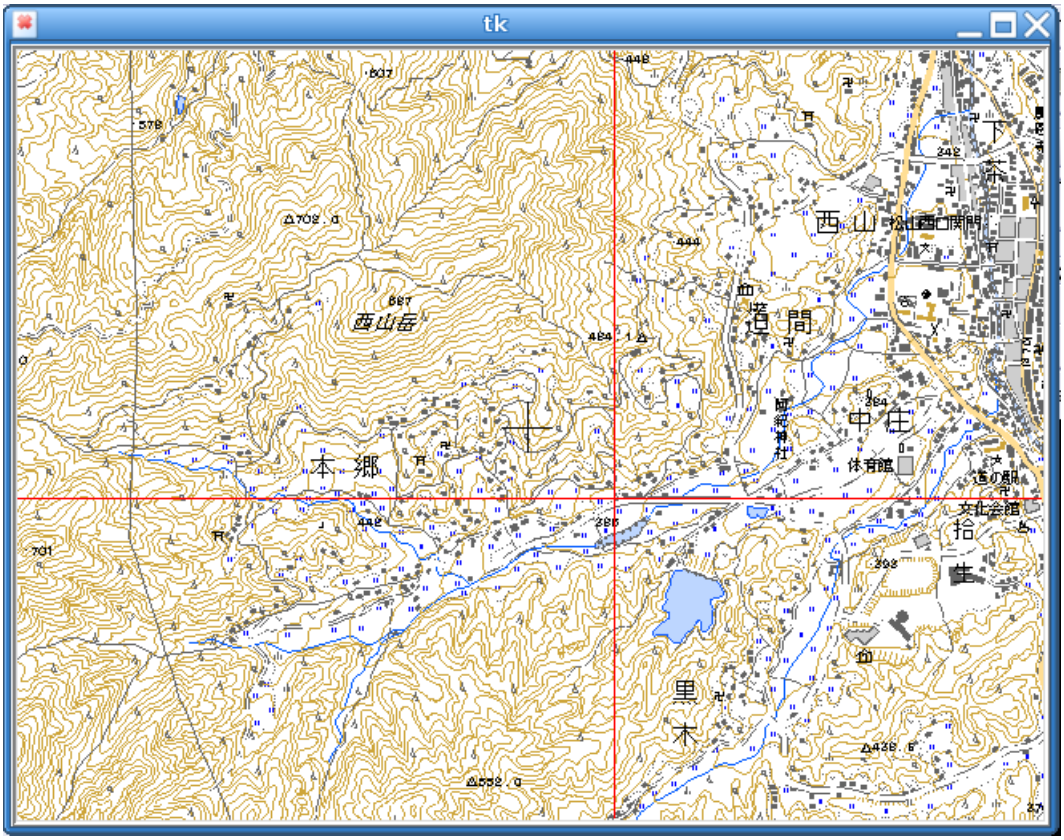
[D] GPS ロガー実行例

1. ログファイル

```
$GPVTG,113.96,T,,,0.00,N,0.00,K,A*7C
$GPGGA,054832.084,3450.2455,N,13615.2118,E,1,09,01.0,253.6,M,37.3,M,,*6B
$GPRMC,054832.084,A,3450.2455,N,13615.2118,E,0.00,113.96,260909,,,A*6C
$GPVTG,113.96,T,,,0.00,N,0.00,K,A*7C
$GPGGA,054833.083,3450.2455,N,13615.2118,E,1,08,01.1,253.5,M,37.3,M,,*6E
$GPRMC,054833.083,A,3450.2455,N,13615.2118,E,0.00,113.96,260909,,,A*6A
$GPVTG,113.96,T,,,0.00,N,0.00,K,A*7C
$GPGGA,054834.083,3450.2455,N,13615.2118,E,1,09,01.0,253.4,M,37.3,M,,*68
$GPGSA,A,3,03,06,07,08,11,16,19,22,25,,,,2.2,1.0,1.9*39
$GPGSV,3,1,09,3,44,055,37,6,33,060,33,7,50,266,39,8,28,311,37*7A
```

2. 現在位置データ表示例

3. 現在位置地図表示例


tk
西山
本郷
中庄
黒木
体育館
文化会館

4. プログラム例1：データのログ

```
#!        /usr/bin/python2.5
# -*- coding: utf8 -*-

#*****************************************
#
#        Created                        .... 2010-01-03
#        updated                        .... 2010-07-11
#
#        object:                        logging data from gps  reeiver
#
#        input device:        gps receiver (N103)
#        output-1:               file
#        output-2:               terminal
#
#        data format:        NMEA-0183.
#
# *****************************************

import        serial

# open read/write file.
d_fname = "logdata01"
fw = open(d_fname, 'w')
fw.close()
fw = open(d_fname, 'a')

# open serial line.
ser = serial.Serial('/dev/ttyUSB0', 38400, timeout = 3)
serstr = ser.portstr + "\n"
print ser.portstr

for  i in range(10):
        line = (ser.readline() ).strip()         # get 1 line 'GPGGA' sentence.
        print line
        line += "\n"
        fw.write(line)                                                  # write new data to log file.

fw.close()
ser.close()
```

5. プログラム例1：データのログプログラム例2：　現在位置（ 緯度）の表示

```
#!       /usr/bin/python2.5
# -*- coding: utf8 -*-

#*****************************************
#
#        Created                        .... 2009-02-07 ...
#        updated                        .... 2010-07-11
#
#        object:                        display latitude on the screen
#
#        input device:        gps receiver (N103)
#        output-1:               file
#        output-2:               screen
#
#        data format:        NMEA-0183.
#
#*****************************************

import        serial
import Tkinter as Tk
import        string
# import os.path


root = Tk.Tk()

buff_dLa = Tk.StringVar()
buff_mLa = Tk.StringVar()
buff_sLa = Tk.StringVar()
buff_dLa.set('')
buff_mLa.set('')
buff_sLa.set('')

# define function
def cmd_quit():
        root.destroy()

# display frame

# define display font
dfont = 'Monospace '
dfont12 = dfont + '12'
dfont18 = dfont + '18'
dfont24 = dfont + '24'
dfont28 = dfont + '28'
dfont48 = dfont + '48'

# display latitude
f1 = Tk.Frame(master=None, relief= 'ridge', bd=3)
b10 = Tk.Label(f1, text="北緯", font=dfont24, relief = 'ridge')
b11 = Tk.Label(f1, textvariable = buff_dLa, font=dfont48, relief = 'ridge')
b12 = Tk.Label(f1, text="度", relief = 'ridge')
b13 = Tk.Label(f1, textvariable = buff_mLa, font=dfont48, relief = 'ridge')
b14 = Tk.Label(f1, text="分", relief = 'ridge')
b15 = Tk.Label(f1, textvariable = buff_sLa, font=dfont48, relief = 'ridge')
b16 = Tk.Label(f1, text="秒", relief = 'ridge')

for e in [b10, b11, b12, b13, b14, b15, b16 ]:
        e.pack(side='left', padx=5)

# display Exit Button
b99 = Tk.Button(master=None, text="Bye", command=cmd_quit)

# display Tate ni naraberu.
for e in [ f1, b99]:
# for e in [ f0, f1, f2, f3, f4, b99]:
        e.pack(side='top', padx=5, pady=5)

def showtime():
        # get 'GPGGA' sentence from gps receiver.
        ser = serial.Serial('/dev/ttyUSB0', 38400, timeout = 3)
        # ser = serial.Serial('/dev/ttyUSB0', 4800, timeout = 3)
        print ser.portstr
        line = ser.readline()
        words = line.split(',')
        while( not words[0] == '$GPGGA'):
                line = ser.readline()
                words = line.split(',')
        print line
        ser.close()

        # latitude
        lati = words[2]
        deg_lati = " " + lati[:2]
        min_lati = lati[2:4]
        sec_lati = str( int(float(lati[4:]) * 60 ) ).zfill(2)
        gpslati = 'latitude = ' + deg_lati + ':' + min_lati + ':' + sec_lati + ' ' + words[3] + '  '

        buff_dLa.set(deg_lati)
        buff_mLa.set(min_lati)
        buff_sLa.set(sec_lati)

        root.after(1, showtime)

showtime()
root.mainloop()
```

13 ハッカーに一歩近づく Tips ～正規表現編～

山下康成@京都府向日市

13.1 正規表現とは

検索や置換などテキスト処理で使用する効率的なパターンマッチングの表現方法です。ファイル名に使う *, ? と概念は同じです。今回、説明するのは

```
^$.[]*\(){}
```

のたった 11 文字＋数字です。

13.2 事前課題

/etc/passwd を解析してユーザ bin のログインシェルを表示しなさい

1. 思考に近い解答例

```
% grep '^bin:' /etc/passwd|cut -d: -f7
```

2. 空気を読んで正規表現を使った解答

```
% sed -n -e 's/^bin:.*:\([^:]*\)$/\1/p' /etc/passwd
```

13.3 正規表現で使う文字

13.3.1 ^ と $

^ 行頭に一致

$ 行末に一致

行頭 (^), 行末 ($) にマッチさせる場合の例:

```
% grep '^bin:' /etc/passwd
% grep '/bash$' /etc/passwd
```

grep 以外でも、 emacs や vi でも同じ様に使える。もちろん, perl や awk でも...

```
emacs   ->  RE search: ^bin
vi      ->  /bash$
```

13.3.2 . ピリオド

何か一文字に一致する。文字数だけが一致すれば良いときや何か文字があることが重要なときに使う。

13.3.3 列挙

列挙したどれかに一致させたいとき。例: bin や root など b, r で始まるユーザを表示

```
% grep '^[br]' /etc/passwd
```

ほかには

```
[! " \# ]   !, ", \# に一致
[0-9]   半角数字１文字に一致
[A-Za-z]   英字に一致
```

13.3.4 [^ 列挙]

列挙したどれにも一致させたくないとき。例: daemon や sys など b, r 以外で始まるユーザを表示

```
% grep '^[^br] /etc/passwd
```

※行頭を表す ^ と [ ] 内の ^ とは、同じ文字だが意味が違う

13.4 *(アスタリスク)

すぐ前の文字／正規表現 0 回以上に一致。例:

```
何がいくつあっても一致
  .*
ホワイトスペースがいくつあっても一致
  [<space><tab>]*
```

13.4.1 {n,m}

すぐ前の文字／正規表現 n 回以上 m 回以下に一致。 m は省略可能

```
ホワイトスペースが 1 個以上あれば一致
[<space><tab>]\{1,\}
数字が 1 文字以上５文字以下( 例えばざっと short の範囲) あれば一致
[0-9]\{1,5\}
```

13.4.2 ( ) と \n (n=1..9)

( ) で囲んだところを順に \1 ,\2 ... として使える。置換に使うと置換元の一部を使える。 vi の ex モードや、 emacs の replace regexp でも使える。例:

```
# aaa bbb 等、行頭に同じ文字が３つある時に一致
% echo 'aaa' | grep '^\(.\)\1\1'
# abab 1212 等、行頭から２文字の繰り返しに一致
% echo 'abab' | grep '^\(.\)\(.\)\1\2'
% echo 'abcd' | sed - e 's/^\(.\)\(.\).*$/1 文字目は \1、 2 文字目は \2 です/'
```

13.5 事前課題の空気を読んだ解答をレビュー

```
sed -n -e 's/^bin:.*:\([^:]*\)$/\1/p' /etc/passwd
```

これは、

{

行頭が bin: で

何か文字が 0 文字以上あって
: があって
: 以外が 0 以上あるところを 1 番目のバッファにいれて
行末
}

というパターンがあれば、1 番目のバッファを表示する。

13.6 実例

13.6.1 ssh アタッカーをブロックする

/var/log/daemon.log に

```
Apr 11 04:46:07 ns sshd[31776]: Invalid user oracle from 211.233.73.66
Apr 11 09:17:05 ns sshd[5607]: Did not receive identification string from 211.155.227.20
```

のような行があれば、その IP アドレスを /etc/hosts.deny に登録する。

```
#! /bin/sh
# Apr 11 04:46:07 ns sshd[31776]: Invalid user oracle from 211.233.73.66
# Apr 11 09:17:05 ns sshd[5607]: Did not receive identification string from 211.155.227.20

export LANG=C
HOSTSDENY=/etc/hosts.deny

sed -n -e 's/^.* sshd.*: Invalid user \(.*\) from \([0-9][0-9\.]*\)/\1 \2/p' \
       -e 's/^.* sshd.*: Did not receive identification string from \([0-9][0-9\.]*\)/\1/p' \
        /var/log/daemon.log | sort -u |
while read NAME IP; do
        # /etc/hosts.deny
        L=`grep '^ALL : '$IP'$' $HOSTSDENY`
        if [ "$L" = "" ]; then
                echo "ALL : $IP" >> $HOSTSDENY
        fi
done
```

13.6.2 ロードアベレージが 3 以上なら、日時と ps ax の結果を /tmp/highload に残す

```
# LANG=C uptime
11:57PM  up 165 days,  2:26,  1 user,  load average: 0.00, 0.06, 0.07
```

上記のロードアベレージの一つ目を切り出す

```
#! /bin/sh
LOAD=`LANG=C /usr/bin/uptime | sed -e 's/^.*: \([0-9]*\)\.[0-9]*,.*$/\1/'
if [ "$LOAD" -ge 3 ]
then
      echo >> /tmp/highload
      date >> /tmp/highload
      ps ax >> /tmp/highload
fi
```

13.6.3 キモ

行頭もしくは行末を起点にパターンマッチをすると良い場合が多い。→ ^ と $ とを活用
スペース区切りは TAB とスペースがいくつあるかわからない。→ * や {n,m} を活用

```
[<SP.><TAB>][<SP><TAB>]*
[<SP.><TAB>]\{1,\}
```

13.7 おわりに

正規表現をうまく使いこなせれば操作が少なくなり、もしくはスクリプトが小さくできて効率的。正規表現をマスタして、ハッカーに一歩近づこう!

14 ニューラルネットワークで画像認識してみた

本庄弘典

14.1 はじめに

ドキュメントスキャナで本をスキャンした際、画像のサイズが大きすぎるため保存に適しません。この画像を 2 値画像とグレースケール、カラー画像それぞれの処理を加えることでファイルサイズを縮小し、ニューラルネットを用いることによりある程度自動化できないかと考えました。今回はニューラルネットとして一般的な三層パーセプトロンを用いた画像判別の一例を解説します。

14.2 三層パーセプトロンとバックプロパゲーション

14.2.1 三層パーセプトロン

三層パーセプトロンは入力層、中間層、出力層と別れた三層の各ニューロンが重みと呼ばれる係数で結ばれたモデルとなります (図 24)。

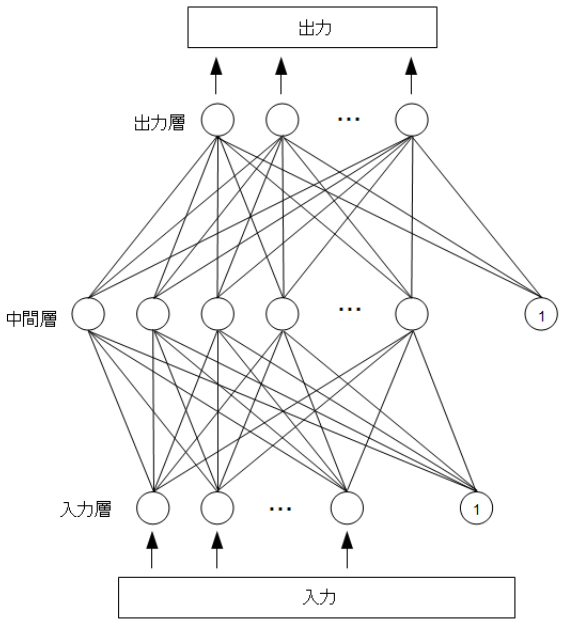


図 24 三層パーセプトロン

それぞれの重みは実数で表され、パーセプトロンが機能するためにはこの重みが適切に設定されている必要があります。

ある入力が与えられた際、入力値に重みを掛け合わせ、それぞれの合計に次のようなシグモイド関数を適用した数値を中間層の持つ値とします。

$$\varsigma_1(x) = \frac{1}{1+e^{-x}}$$

図 25　シグモイド関数の式

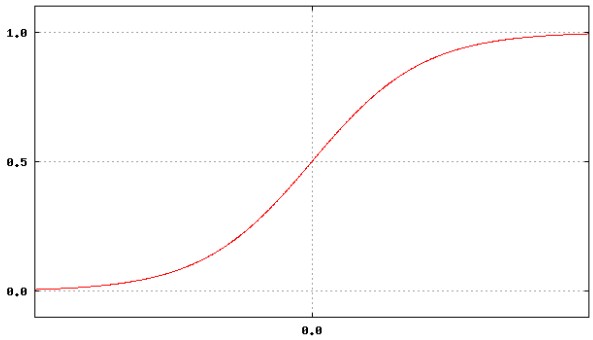


図 26　シグモイド関数のグラフ

各出力層も同様の計算がなされ、パーセプトロンの出力が行われます。

14.2.2　バックプロパゲーション

多層パーセプトロンで適切な出力を行うための学習方法として一般的なものにバックプロパゲーションがあります。バックプロパゲーションではまず入力に対する正しい出力 (教師信号) を多数用意し、各重みをランダムに設定します。用意された入力に対してランダムな重みからパーセプトロンの出力はでたらめな値となりますが、この出力と教師信号との比較から出力層と中間層の間の重みを修正し、次いで中間層と入力層の重みを修正することで適切な重みを探し出します。

14.3　足し算と引き算を学習してみる

作成したパーセプトロンとバックプロパゲーションが正常に動作するかを確かめます。次のような入力を用意しました。

```
# 学習用教師信号ペア
0.40,0.20       0.60,0.20
0.30,0.20       0.50,0.10
0.80,0.10       0.90,0.70
0.20,0.10       0.30,0.10
0.50,0.50       1.00,0.00
0.60,0.20       0.80,0.40
# 評価用入力値
*0.50,0.10
*0.50,0.40
*0.10,0.40
```

入力値と教師信号のペアはタブ区切りの左が入力、右が入力に対する教師信号です。ここでは足し算と引き算の教師信号を与えました。

実行します。

```
$ ./backprop.exe sample.txt 10000
       0 0.87640153
     100 0.26410368
     200 0.10289131
     300 0.03820243
     400 0.02475167

...(中略)...

    9600 0.00077714
    9700 0.00077174
    9800 0.00076646
    9900 0.00076128
0.4000, 0.2000  0.60, 0.18      0.60, 0.20
0.3000, 0.2000  0.50, 0.11      0.50, 0.10
0.8000, 0.1000  0.90, 0.70      0.90, 0.70
0.2000, 0.1000  0.30, 0.11      0.30, 0.10
0.5000, 0.5000  0.98, 0.02      1.00, 0.00
0.6000, 0.2000  0.80, 0.41      0.80, 0.40
0.5000, 0.1000  0.63, 0.35
0.5000, 0.4000  0.93, 0.06
0.1000, 0.4000  0.87, 0.00
Ratio=0.00075626
Count=10000
Sample=6
Input=2
Middle=4
Output=2
InputHidden0=-2.57936471,-2.20525001,-1.50656422,4.05055823,-0.66468037
InputHidden1=-1.29032439,8.71632107,-1.24344376,-0.85214732,-0.66468037
InputHidden2=2.04901840,-2.94096519,1.04866634,-1.98825291,0.29698485
HiddenOutput0=-2.91458436,-1.16992032
HiddenOutput1=5.84673832,-6.31188860
HiddenOutput2=-1.80018561,-0.42470539
HiddenOutput3=3.60356071,3.84028669
HiddenOutput4=1.40998866,-1.22885398
```

頼りないながらもそれなりの演算結果が出力されています。評価として最後の数値は減算結果が負になるはずなのですが、シグモイド関数を通すことで出力が 0.0〜1.0 となるため正常な結果が得られません。

14.4 画像を分類するための入力値を考える

画像判別の入力値として次の値を使用しました。

- 補正した画像の RGB の差
- 微分した画像の RGB の差
- 画像の複雑さ。
- 使われている色の数
- 平均彩度
- FFT 処理した画像の明るいピクセルを利用する
- HSV に変換し、色素の平均を利用する
- 色素の分散を利用する

この中から文章と絵の判別として画像の FFT を、カラー画像の判別として HSV への変換を解説します。

14.4.1 モノクロ画像の処理・文字と絵を分類してみる

縦書きの文章は横方向に一定の周波数を持っていると見なすことが出来ます。これにより、文章の画像を微分し FFT 処理を行った結果から振幅を描画するとで、明るく光る点が現れることがわかりました。

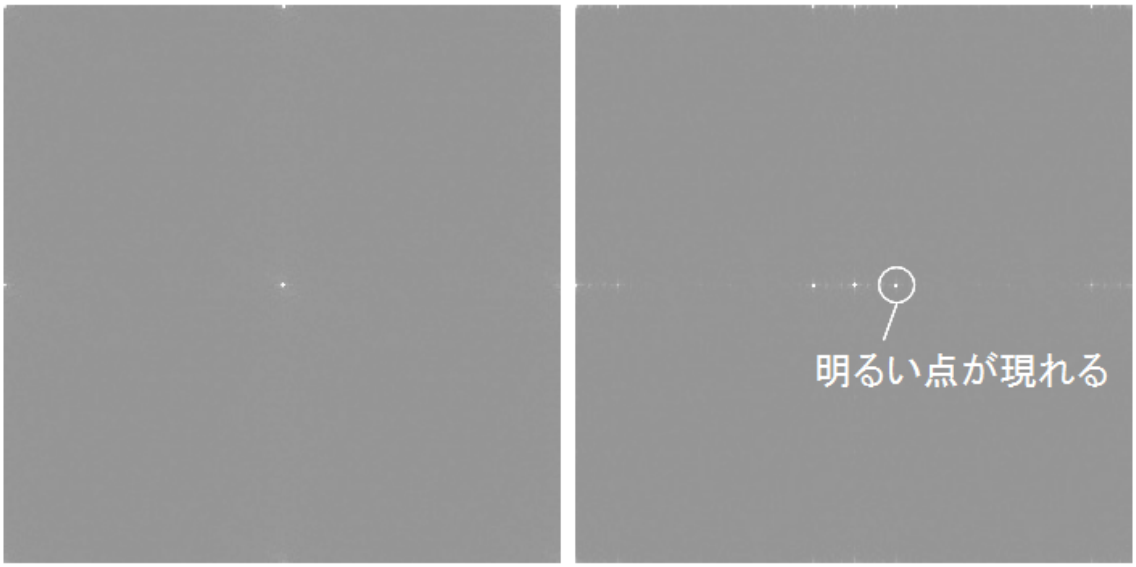


図 27　イラストと文章を微分した画像の FFT 結果

この点の明るさを入力値とすることで、文章とイラストの判別が行えると期待できます。

14.4.2　カラー画像とそうでない画像を分類してみる

カラー画像とモノクロ画像は画像の RGB を HSV に変換し、色相から判別を行っています。

RGB のうちから最大のものを MAX、最小のものを MIN とすると色相は図 28 の式となります。

$$
\begin{aligned}
H = & 60\frac{G-B}{MAX-MIN}+0, && if MAX = R \\
& 60\frac{B-R}{MAX-MIN}+120, && if MAX = G \\
& 60\frac{R-G}{MAX-MIN}+240, && if MAX = B
\end{aligned}
$$

図 28　色相の計算式

モノクロ画像は色相を持たないため、 RGB のうち青の成分を減らすことで黄色いフィルタをかけました。こうすることでモノクロ画像の色相の平均は黄色となり、カラーとモノクロを判別するための入力値として期待できます。

14.5　学習の条件

ニューラルネットの学習は次の条件で行いました。

- 入力層 8 個
- 中間層 24 個
- 出力層 2 個
- サンプルとして使用した本 23 冊 (漫画 2 冊/文庫 20 冊/技術書 1 冊)
- ページ数 6742 ページ
- 学習回数 50 万回

実際にこの条件で学習を行った際、 Core i7 950 で 7 時間弱の学習時間となりました。

14.6 判別の精度

作成されたツールで実際に判別を行い、その精度を調べました。評価に使用した本は学習に使われていないものを選びました。

サンプル 1. **ライトノベル最新巻**

240 ページ中、人間の判別と食い違うページが 2 ページ。内訳はカラー 12 ページ、グレースケール 15 ページ、文章 213 ページ。そのうちイラストが文章と判別されたのが 1 ページ、文章がイラストと判別されたのが 1 ページ。

サンプル 2. SF **長編シリーズの上巻**

568 ページ中、人間の判別と食い違うページはなし。内訳はカラー 5 ページ、グレースケール 3 ページ、文章 560 ページ。

サンプル 3. **ローマ人のシリーズ** 1 **巻**

216 ページ中、人間の判別と食い違うページは 5 ページ。内訳はカラー 6 ページ、グレースケール 9 ページ、文章 201 ページ。そのうちカラー 4 ページの判別に失敗していますが、原因はカバーページがベージュで文章として判別されました。地図やイラストと文章が混じったページも学習通り文章として判別されています。

サンプル 4. 50 **年前に発行された芥川賞受賞作**

40 年近く前に出版された本で、 280 ページ中、人間の判別と食い違うページが 16 ページ。内訳はカラー 6 ページ、文章 274 ページ。判別の失敗が多い理由は変色と推測され、カラーだけではなくイラストとも多く間違って判別されました。

15 Weka を使ってみる

まえだこうへい

15.1 概要

ニューラルネットワークをはじめとするデータマイニングを一般人が使うには、前章の本庄さんのネタのように理論を理解した上で自分でプログラムを作らないといけないとすると、非常にハードルが高いと思います。学生のころに学んだり、研究していたか、仕事として普段から扱っているような人でなければ、単語としては耳にしたことがあっても、なんだかよう分からん、という人がほとんどではないでしょうか。[*51]

そこで、バックグラウンドとしてニューラルネットワークだけでなく、データマイニングの基礎知識を持っていない私と同じような立場の人でも、 Debian なら気軽に試してみる環境を整えて、取り合えず使ってみることができるよ、という趣旨で、 Weka というツールを紹介します。

15.2 Weka とは

Weka とは、 “Waikato Environment for Knowledge Analysis” の略で、ニュージーランドの国立ワイカト大学[*52]で GPL のもとオープンソースで開発されているデータマイニングツールです。[*53] Java で書かれています。

15.3 インストール

Debian ではパッケージが用意されています。

```
$ sudo apt-get install weka
```

15.4 Weka の使い方

コマンドラインで `weka` スクリプトを実行します。

```
$ weka &
```

すると Weka のウィンドウが起動します。そこから、 “Applications” の “Explorer” を実行すると、 Weka Explorer が起動します。

[*51] Debian 勉強会の常連はむしろ知っている人の方が多いのかもしれませんが、少なくとも私は単語を耳にしたことがあるレベルです。
[*52] `http://www.waikato.ac.nz/`
[*53] `http://www.cs.waikato.ac.nz/~ml/weka/index.html`

図 29　Weka 起動メニュー

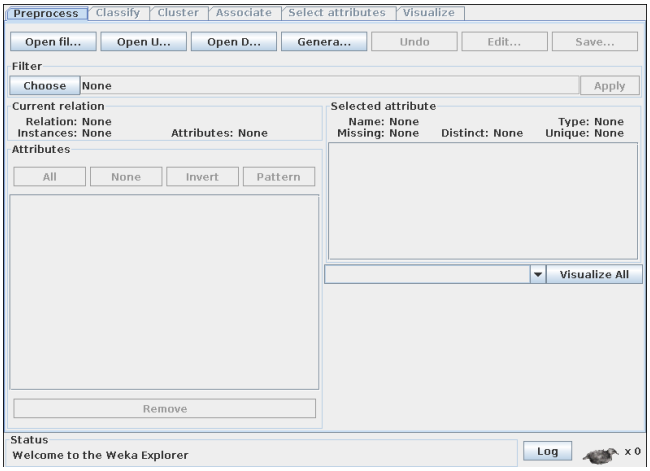


図 30　Weka Explorer

15.4.1　Weka で扱うデータフォーマット

Weka では、 ARFF(Attribute-Relation File Format) というフォーマットのテキストファイルを入力データとして扱います。

データフォーマットは次のようになります。

```
@relation 名前 ←データ全体の名前を指定
@attribute 属性名 属性の型 ←データの属性。 1 つ目の引数は属性名、 2 つめの引数は属性データの型
@attribute 属性名 属性の型
:
:
@data ←データ領域の宣言
データ, データ,…, データ ← CSV 形式でデータを記述
```

- @relation はデータ全体の名前を指定します。
- @attribute はデータの属性を表し、 1 つ目の引数は属性名、 2 つめの引数は属性データの型を表します。
- @attribute のデータ型には、 numeric, real, integer, string, date 型を使えます。
  - numeric は real か integer を指定できます。
  - real は実数を指定できます。
  - integer は整数を指定できます。
  - string は文字列を指定できます。
  - date 型は日時で、デフォルトは”yyyy-MM-dd’T’HH:mm:ss” という書式です。
- @data データ領域の宣言です。
  - その次の行から CSV 形式でデータを記述します。
  - @attribute 行で上から設定した順に CSV の一行での左から右への各データとなります。

昨年の 11 月の勉強会で GNU R で扱った光熱費のデータと、気象庁が公開している気温、降水量などの気象データ[*54]を使ってみます。

まず、 CSV で以下のように記述したとします。

```
"年","月","降水量合計 (mm)","平均日平均気温 (℃)","平均日最高気温 (℃)","平均日最低気
温 (℃)","平均風速 (m/s)","最大風速 (m/s)","日照時間 (h)","電気使用量 (kWh)","電気使用
量 (kWh)/日","料金 (円)/日","合計料金","ガス使用量 (m3)","ガス使用量 (m3)/日","料金 (
円)/日","合計料金"
2007,1,50,6,10.8,1.1,1,5,188.9,234,6.88235294117647,161.235294117647,5482,9,0.333333333333333,80.962962962963,2186
2007,2,44,7.3,12.6,1.9,1.3,6,198.1,198,7.07142857142857,168.071428571429,4706,9,0.321428571428571,87.7142857142857,2456
(snip)
```

*54 http://www.data.jma.go.jp/obd/stats/etrn/index.php

これは、 ARFF フォーマットでは以下のようになります。

```
@relation 降水量・気温（府中市）と電気代、ガス代の関係について

@attribute 年 real
@attribute 月 real
@attribute 降水量合計（mm） real
@attribute 平均日平均気温（℃） real
@attribute 平均日最高気温（℃） real
@attribute 平均日最低気温（℃） real
@attribute 平均風速（m/s） real
@attribute 最大風速（m/s） real
@attribute 日照時間（h） real
@attribute 電気使用量（kWh） real
@attribute 電気使用量（kWh)/日 real
@attribute 料金（円)/日 real
@attribute 合計料金 real
@attribute ガス使用量（m3） real
@attribute ガス使用量（m3)/日 real
@attribute 料金（円)/日 real
@attribute 合計料金 real

@data
2007,1,50,6,10.8,1.1,1,5,188.9,234,6.88235294117647,161.235294117647,5482,9,0.333333333333333,80.962962962963,2186
2007,2,44,7.3,12.6,1.9,1.3,6,198.1,198,7.07142857142857,168.071428571429,4706,9,0.321428571428571,87.7142857142857,2456
(snip)
```

15.4.2 ARFF をロードする

先ほど用意した kohnetsu.arff をロードしてみましょう。 Preprocess タブの Open File ボタンを押します。ダイアログが表示されるので、 kohnetsu.arff を指定します。 (図 31)

ARFF ファイルを読み込むと図 32 のようになります。 UTF-8 エンコードであれば、ご覧のとおり日本語も正常に読み込めます

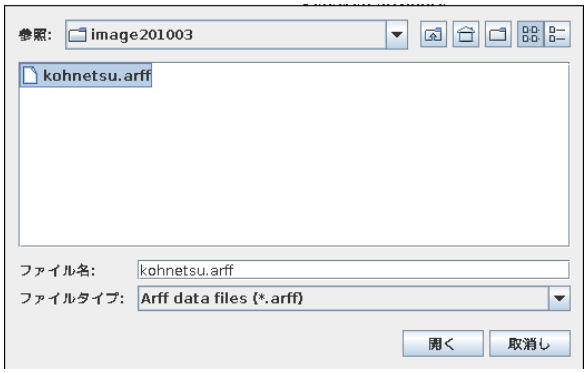


図 31　ARFF ファイルをロードする

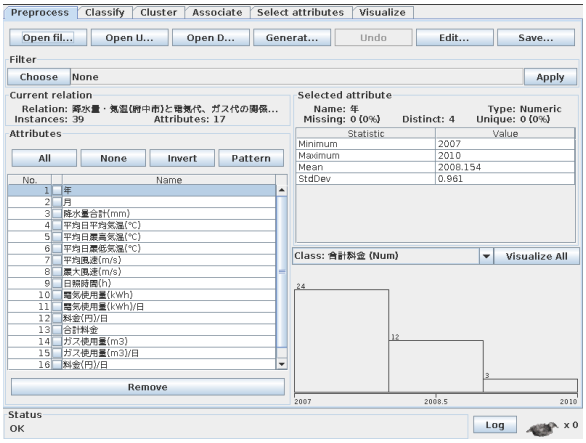


図 32　ARFF からデータを読み込んだ結果

15.4.3 可視化してみる

それでは読み込んだデータを可視化してみましょう。 Visualize タブをクリックすると図 33 のようなマトリックスが表示されます。

適当に開いてみます。 Y 軸に一日の平均気温の月平均 (℃) と、 X 軸に一日あたりの電気使用量 (kWh) を取って見てみると図 34 のようになります。

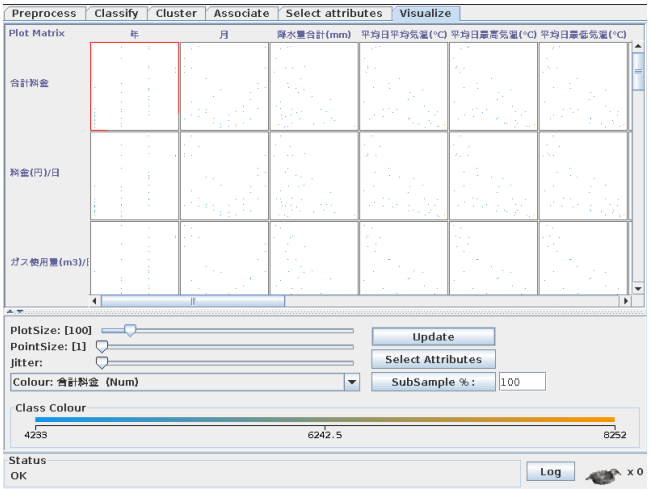


図 33　Visualize 画面

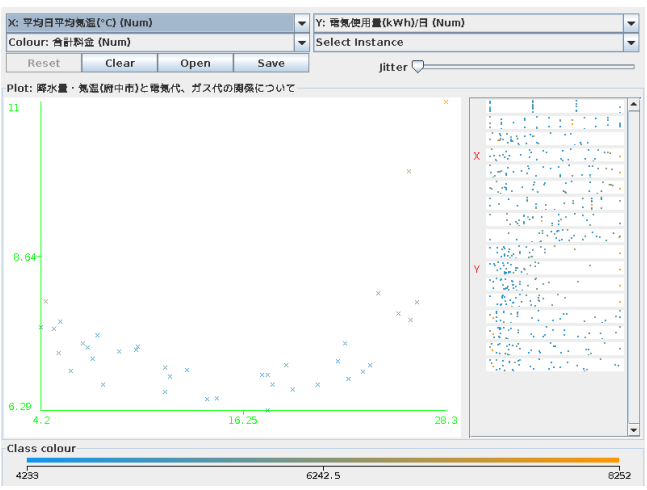


図 34　Visualize 詳細画面

15.4.4　分類してみる

次に分類してみます。 Classify タブ → Choose ボタンを押し、表示されたツリーから Multilayer Perceptron (ニューラルネットワークによる分類) を選択します。次に、 (Num) 平均日平均気温 (℃) を目的関数として選択します。

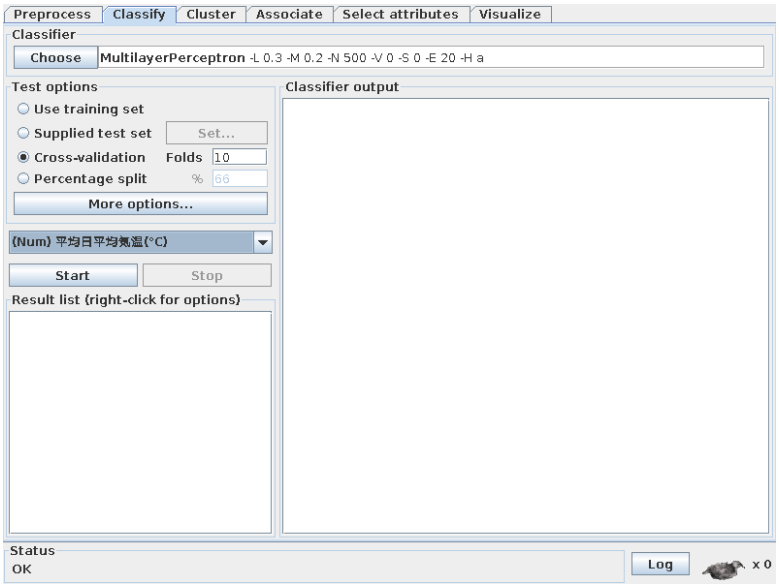


図 35　Classify 画面

Start ボタンをクリックすると分類が実行され、結果が表示されます。

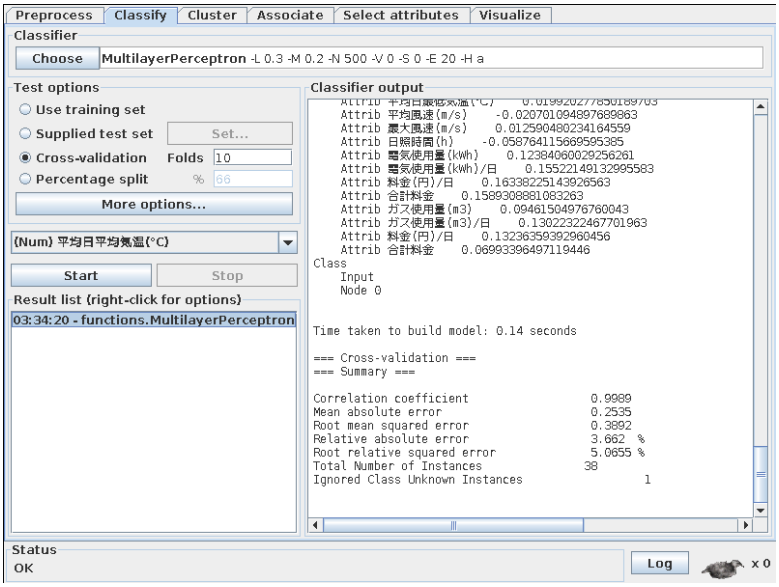


図 36　Classify 結果

15.4.5　予測してみる

実は、 kohnetsu.arff の最後の行 (2010 年 3 月のデータ) は、ほとんどの項目を'?' を入力しています。

```
2010,3,?,?,?,?,?,?,?,?,?,?,?,?,?,?
```

これはまだ今月のデータが出ていないからです。それでは、これらを予測してみます。先ほどの Classify タブの画面で More options をクリックし、 Output predictions のチェックを入れ、 OK を押します。

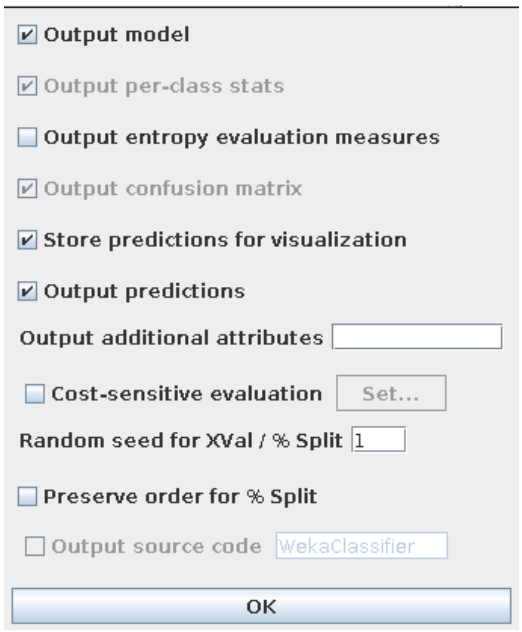


図 37　Classify の More Options ダイアログ

Start を実行すると、予測結果が表示されます。 actual が実データで predicted が予測結果です。今回見たいのは、 3 月の平均日平均気温です。 actual が ? になっている行の、 predicted の値を見ると、 15.436 となっています。過去読み込ませたデータからニューラルネットワークでの分析して予測した結果、おそらく一日あたりの平均気温は 15.4 ℃ になるという予測です。来月、気象庁の統計データが更新されたら確認してみましょう。

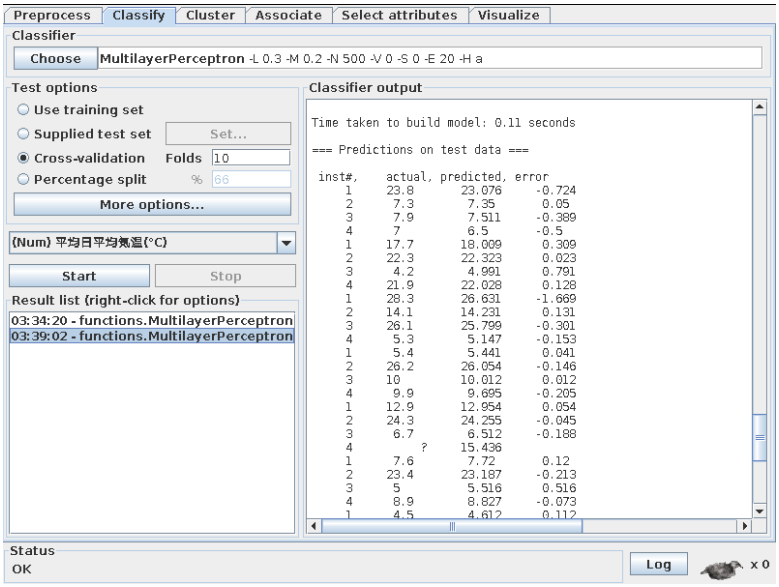


図 38　予測結果

15.5　まとめ

なんとなく使えそうな気がしたでしょうか。私はこれで毎月のガス代、電気代、さらには水道代の予測をして、給与日前日の予算計画に利用していこうと思います。仕事でも、企画上の裏付けデータの分析、予測などにも使えそうですね。

15.6　参考文献

- http://www.ilibrary.jp/MOTtextBooks/text/weka.pdf
- http://web.sfc.keio.ac.jp/~soh/dm03/
- http://www1.doshisha.ac.jp/~mjin/R/23.pdf
- http://weka.wikispaces.com/ARFF

16 Debian で libfftw を使ってみる

上川純一

16.1 はじめに

Debian で FFT を取り扱う C のアプリケーションを書いてみたいと思うことはありませんか?今日は音声データを分析してみましょう。

wav ファイルを入力として受け取り、 FFT を実行してその結果を表示するアプリケーションを作成してみます。

16.2 インストール

libfftw3 をインストールします。あと、音声ファイルをロードするために sndfile1 を利用します。

```
$ apt-get install libfftw3-dev libsndfile1-dev
```

16.3 実験対象の準備:簡単な sine 波を作成する

まず、テスト用に FFT の結果が予想できるデータを作成してみます。ここでは 440Hz のきれいなサイン波を作成しています。

$$data(x) = sin(\frac{2\pi 440x}{44100})^{*55}$$

*55 sin は radian

```
/*BINFMTC: -lsndfile -lm

  Create a sine wave at 44.1kHz for 1 second called sine.wav
 */
#include <stdlib.h>
#include <stdio.h>
#include <sndfile.h>
#include <math.h>

int create_sine(const char* filename, int size, double frequency)
{
  SF_INFO sfinfo = {
    .frames = size,
    .samplerate = 44100,
    .channels = 1,
    .format = SF_FORMAT_WAV | SF_FORMAT_PCM_16,
    .sections = 0,
    .seekable = 0
  };
  SNDFILE* s = sf_open(filename, SFM_WRITE, &sfinfo);
  double* data = malloc(sizeof(double) * size);
  int i;

  for (i=0; i < size; ++i)
    {
      data[i] = sin(frequency * 2.0 * M_PI * i / 44100.0);
    }

  sf_writef_double(s, data, size);
  sf_close(s);
  return 0;
}

int main()
{
  return create_sine("sine.wav", 44100, 440.0);
}
```

16.4 実験対象の準備:複雑な入力値例の準備

テスト用の入力値として、適当な wav ファイルを用意しましょう。

今回は手元で、 aeolus というオルガンシミュレータを起動し、 jack で接続させ、 ecasound を jack 入力に対して待機させ、 qjackctl で接続させて収録しました。

それなりに長い時間録音したデータから 16-bit mono の PCM データ 1 秒分を切り出して実験用データを作成しました。

```
$ qjackctl &
$ aeolus &
$ vkeybd &
$ ecasound -i jack -o test.wav
ctrl-C で中断
$ sweep test.wav # 適当に編集
$ file ra-mono.wav  # 切り出した結果を確認
ra-mono.wav: RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, mono 44100 Hz
```

16.5 FFTW を使って wav ファイルを処理してみる

sndfile と fftw3 を使ってフーリエ変換して出力をダンプしてみましょう。サンプルコードは sndfile を使い double の配列に wav ファイルの中身を展開して、その内容を fftw に渡して処理しています。 double の値は各 1/44100 秒の瞬間における空気の圧力を表しているようです。

```
/*BINFMTC: -lsndfile -lfftw3 -lm
 */

#include <stdlib.h>
#include <stdio.h>
#include <sndfile.h>
#include <math.h>
#include <complex.h>
#include <fftw3.h>

/*
  process with FFTW
 */
void study_sound(double* data, int size)
{
  fftw_complex* spectrum;
  fftw_plan p;
  int i;

  spectrum = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * (size / 2 + 1));
  p = fftw_plan_dft_r2c_1d(size, data, spectrum, FFTW_ESTIMATE);

  /* process with FFTW */
  fftw_execute(p);

  /* dump output in CSV format */
  printf("i,abs,arg\n");
  for (i=0; i<(size/2+1); ++i) {
    printf("%i,%f,%f\n", i,
           cabs(spectrum[i]),
           carg(spectrum[i]) / 2.0 / M_PI * 360.0);
  }
  fftw_destroy_plan(p);
  fftw_free(spectrum);
}

/*
  Process wav file.

  @return 1 on failure, 0 on success.
*/
int process_wav_file(const char* filename, int size)
{
  SF_INFO sfinfo = {0, 0, 0, 0, 0, 0};
  SNDFILE* s = sf_open(filename, SFM_READ, &sfinfo);
  double* data = malloc(sizeof(double) * size);

  if (!s || !data)
    {
      fprintf(stderr,
              "Something went wrong opening the file or allocating memory\n");
      return 1;
    }
  if (sfinfo.channels != 1)
    {
      fprintf(stderr,
              "Please give me monaural audio data\n");
      return 1;
    }

  /* Read wav file into an array of double */
  sf_readf_double(s, data, size / sfinfo.channels);
  study_sound(data, size / sfinfo.channels);
  sf_close(s);
  return 0;
}

int main(int argc, char** argv)
{
  process_wav_file(argv[1], atoi(argv[2]));
  return 0;
}
```

実行してみます。

```
$ ./sndfile-fftw.c sine.wav 44100 > sine.csv
$ ./sndfile-fftw.c ra-1sec.wav 44100 > ra.csv
```

16.6 出力を確認してみる

CSV ファイル形式でデータが出力されました。簡単にグラフを作成するためのツールとしてここでは R を使ってみます。

```
$ R
> sine <- read.csv("sine.csv")
> ra <- read.csv("ra.csv")
> postscript("sine.eps", horizontal=FALSE, height=3, width=3)
> plot(sine$i, sine$abs, xlim=c(400,500), ylim=c(0,22000), type="l")
> dev.off()
> postscript("ra.eps", horizontal=FALSE, height=3, width=3)
> plot(ra$i, ra$abs, xlim=c(0,2000), ylim=c(0,100), type="l")
> dev.off()
```

図 16.6 の 440Hz のサイン波を処理した結果を見てみると、 440 Hz あたりにグラフの突起があるのが見て取れます。

しかし、実際にオルガン音を処理した結果の図 16.6 を見てみると、グラフに突起が多数あって、結構複雑です。そのまま簡単に処理させてくれはしなさそうです。

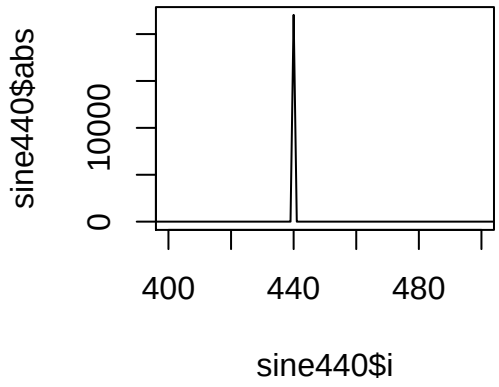


図 39　440Hz のサイン波

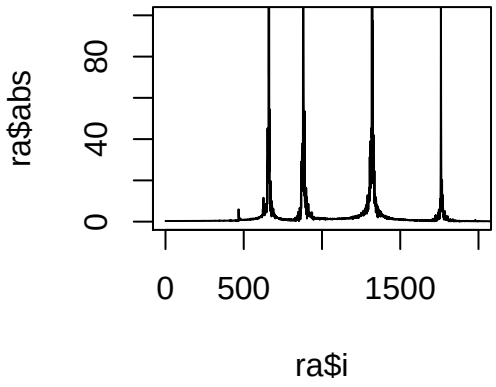


図 40　ラを aeolus で適当に演奏した音

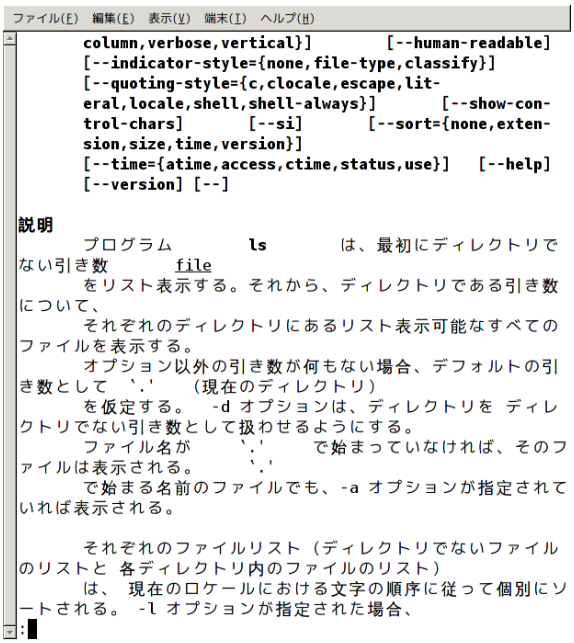
17 man-db を深追いした

日比野 啓

17.1 日本語の man が変

こんな感じ

日本語の文字の表示幅がアルファベットと同じだと判定されてしまっているので、結果的に日本語で書かれた行の長さが倍ぐらいになってしまっています。



17.2 解は groff にあり

17.2.1 そもそも roff ってどういうもの?

簡単に言えば、タイプライターのようなものです。いろいろなデバイスに対して印字を行なうことができるシステムになっていますが、 man では文字端末に文字の幅と高さを考慮しながら印字をしていきます。

17.2.2 roff 文字幅の指定

以前のバージョンの groff では日本語の文字の表示幅がアルファベットの倍であることを unicode の code point の範囲で設定してありました。 groff にはもともと code point の範囲で文字の幅を指定する機能が無いため、その機能の追加と日本語文字の表示幅を倍にするパッチがあててありました。

更新されたバージョンにおいては日本語対応のパッチが当たらなくなってしまっていますが、やはり同様の対処を行なう必要がありそうです。

18 Debian の OCaml 環境で開発する関数型言語インタプリタ

日比野 啓

18.1 OCaml はどんな言語

静的型の型推論言語というのが今の私の認識です。関数型言語らしい機能が注目されますが、それ以外のスタイルも良く利用されます。

ここではこの記事を読みすすめるにあたって必要そうな OCaml の機能について簡単に紹介します。

18.1.1 値

以下、対話環境の入出力を前提に話をすすめます。

ocaml-interp パッケージの /usr/bin/ocaml が対話環境のプログラムです。

は対話環境のプロンプトです。ユーザーの入力の終わりを対話環境に認識してもらうために ;; を入力します。なので対話環境の# プロンプトの後ろから;; までがユーザーの入力です。- :から始まる内容は対話環境の出力です。

```
# 1;;
- : int = 1
# "abc";;
- : string = "abc"
# (1, "abc");;
- : int * string = (1, "abc")
# (1, ("abc", 2), 3);;
- : int * (string * int) * int = (1, ("abc", 2), 3)
# [ "a"; "b"; "c"];;
- : string list = ["a"; "b"; "c"]
# "a" :: "b" :: "c" :: [];;
- : string list = ["a"; "b"; "c"]
```

値を表す式を入力すると対話環境は型の名前と値を出力しています。 1 は int 型、"abc"は string 型となっています。対話環境に式を入力したので評価が行なわれ、その結果としての出力です。

(1, ("abc", 2), 3) のようにコンマで区切って括弧でくくった値はタプルです。値と値の組を表現できます。型の名前は *で連ねます。任意の型の値を組にできる強力な機能です。

[ ] でリストを表現することができます。また、::は lisp でいうところの cons です。リストの要素は全て同じ型である必要があります。

18.1.2 レコードとバリアント

次は値を表現する前にあらかじめ型の定義が必要であるような値です。

```
# type vec_2d = { x : int; y : int };;
type vec_2d = { x : int; y : int; }
# { x = 1; y = 2 };;
- : vec_2d = {x = 1; y = 2}
# type int_or_string_or_none = I of int | S of string | N;;
type int_or_string_or_none = I of int | S of string | N
# I 3;;
- : int_or_string_or_none = I 3
# S "hello";;
- : int_or_string_or_none = S "hello"
# N;;
- : int_or_string_or_none = N
```

一つ目の例はレコードです。C の構造体のようなもので、フィールド名とフィールドの型を定義に記述します。`vec_2d` というレコードの型を定義して、`{ x = 1; y = 2 }`という値を評価させました。

二つ目はバリアントあるいは代数データ型と呼ばれている種類の型です。`int_or_string_or_none` という型を定義していて、その値は I というタグの付いた int か S というタグの付いた string か N というタグ のどれかということです。このタグのことを**コンストラクタ**と呼びます。

バリアントは再帰的な定義も可能で、この機能で簡単に木構造を表現できます。

```
# type str_tree = Leaf of string | Tree of (str_tree * str_tree);;
type str_tree = Leaf of string | Tree of (str_tree * str_tree)
# Leaf "abc";;
- : str_tree = Leaf "abc"
# Tree (Leaf "a", Tree (Leaf "b", Leaf "c"));;
- : str_tree = Tree (Leaf "a", Tree (Leaf "b", Leaf "c"))
```

18.1.3 関数

次は関数を表現する値です。

```
# (fun x -> x + 1);;
- : int -> int = <fun>
# (fun x -> x);;
- : 'a -> 'a = <fun>
```

`(fun x -> x + 1)` は関数です。型が `int -> int` と出力されていますが、これは int を受けとって int を返す関数という意味です。`+` の引数は int と int なので結果として `x` は `int`、`fun x -> x + 1` は `int -> int` というように型の推論が行なわれています。

二番目の `(fun x -> x)` も関数です。型が`'a -> 'a` と出力されていますが、これは何かある型の値を受けとって、同じ型の値を返す関数という意味です。この `'a -> 'a` の関数は任意の型に対して適用が可能なので、全ての型についてこの関数が定義されている (`int -> int` も `string -> string` も ... etc) のと同様の効果があります。このような型を多相型と呼びます。

18.1.4 束縛

`let` を使って値を変数に束縛します。以下は `v` にたいして整数を `f` に対して関数を束縛しています。

```
# let v = 123;;
val v : int = 123
# v;;
- : int = 123
# let f x y = (x + y) * (x - y);;
val f : int -> int -> int = <fun>
# f 5 3;;
- : int = 16
```

18.1.5 パターン照合

OCaml は値の構造を分解しつつ変数を束縛することができます。次の例ではタプルを分解して束縛を行なっています。

```
# let (x, y) = (2, 3 + 4);;
val x : int = 2
val y : int = 7
# let (a, (b, c)) = (1, (2, 3));;
val a : int = 1
val b : int = 2
val c : int = 3
```

match ... with を使用すると、構造分解を試みつつ条件分岐することができます。

関数 len はリストが空かどうかを判定しながら再帰しています。束縛の定義時に自身の定義を使用するときには let ではなく let rec を使います。

関数 what は先程定義したバリアント int_or_string_or_none の値の種類によって返す文字列を変えています。

関数 get_int は int_or_string_or_none が int のときだけ Some int を返し、そうでないときは None を返します。'a option は組み込みの型で、ある型の値が有るかあるいは無いかを表現できるバリアントです。

```
# let rec len ls = match ls with [] -> 0 | x :: rest -> 1 + len rest;;
val len : 'a list -> int = <fun>
# len [1; 2; 3];;
- : int = 3
# let what v = match v with I _ -> "int" | S _ -> "string" | N -> "none";;
val what : int_or_string_or_none -> string = <fun>
# what (S "abc");;
- : string = "string"
# what N;;
- : string = "none"
# let get_int v = match v with I i -> Some i | _ -> None ;;
val get_int : int_or_string_or_none -> int option = <fun>
# get_int N;;
- : int option = None
# get_int (I 10);;
- : int option = Some 10
```

18.1.6 モジュール

プログラムの規模がおおきくなってくると名前空間が重要になってきます。 OCaml には module の機能があり名前空間を分けることができます。あるコンパイル単位が xyz.ml というファイル名だった場合、その中の定義は Xyz という module の中に配置されます。モジュール内にたとえば abc という名前があった場合、公開されていれば他のコンパイル単位のファイルからも Xyz.abc という名前でアクセスすることができます。

```
(* xyz.ml *)
let abc = "abc"
...


(* 他のファイル *)
... Xyz.abc ...
```

module の中にさらに module を定義することもできます。 xyz.ml の中に作った module SubXyz は、 Xyz.SubXyz という名前の module です。

定義済みのモジュールを使って module を定義することもできます。組み込みの module である List を使って Xyz.L を定義しています。

```
(* xyz.ml *)
module SubXyz =
  struct
  ...
  end

module L = List
```

18.2 関数型言語のインタプリタの簡単な作りかた

関数型言語とは何でしょうか。ここでは関数を値としてあつかえる言語ということにして、そのような言語のインタプリタを作る話をします。

18.2.1 環境を渡すナイーブなインタプリタ実装

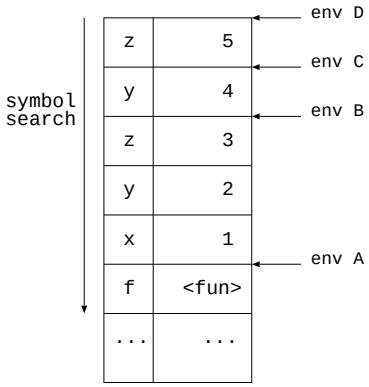
■let の環境

例えば以下のようなプログラムを考えます。

```
;; scheme function
(define (f)
  ;; env A
  (let ((x 1) (y 2) (z 3))
    ;; env B
    (let ((y 4))
      ;; env C
      (let ((z 5))
        ;; env D
        (+ x y z)))))
```

```
(* OCaml function *)
let f () =
  (* env A *)
  let (x, y, z) = (1, 2, 3) in
    (* env B *)
  let y = 4 in
    (* env C *)
  let z = 5 in
    (* env D *)
    x + y + z
```

変数の値の解決のためのテーブルを環境 (environment) と呼ぶことにして、次の図ような構造になっていると考えてみます。変数の検索は上から下に行なわれるとすると、それぞれのコメントの位置の環境は矢印の位置を参照していると考えてよいはずです。



■関数呼び出しにおける環境

さらに以下のようなプログラムを考えます。

```
(define (f x y)
  ;; env f
  (* x y 2))

;;env X

(let ((x 1) (y 2))
  (define (g)
    ;; env g
    (f 3 4))
  (g))

;;; may be another call of (f x y)
```

```
let rec f x y =
  (* env f *)
  x * y * 2

(* env X *)

let (x, y) = (1, 2)
let rec g () =
  (* env g *)
  f 3 4

let v = g ()

(* may be another call of f x y *)
```

するとそれぞれのコメントの位置の環境は図 41 のようになっているはずです。

ここで環境 X は同じ環境なので、共有させることにすると次のような構造を考えることができます。ここで注意する必

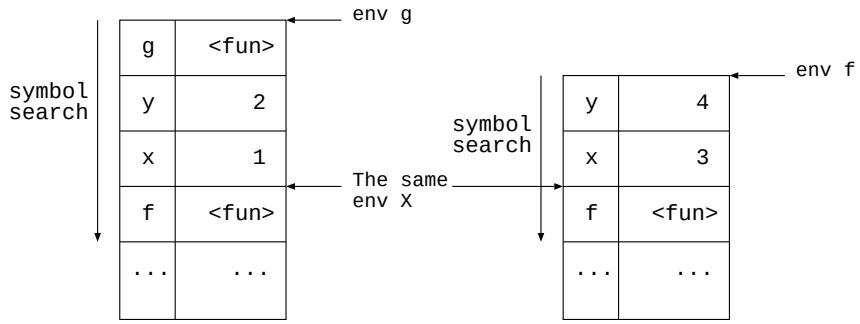
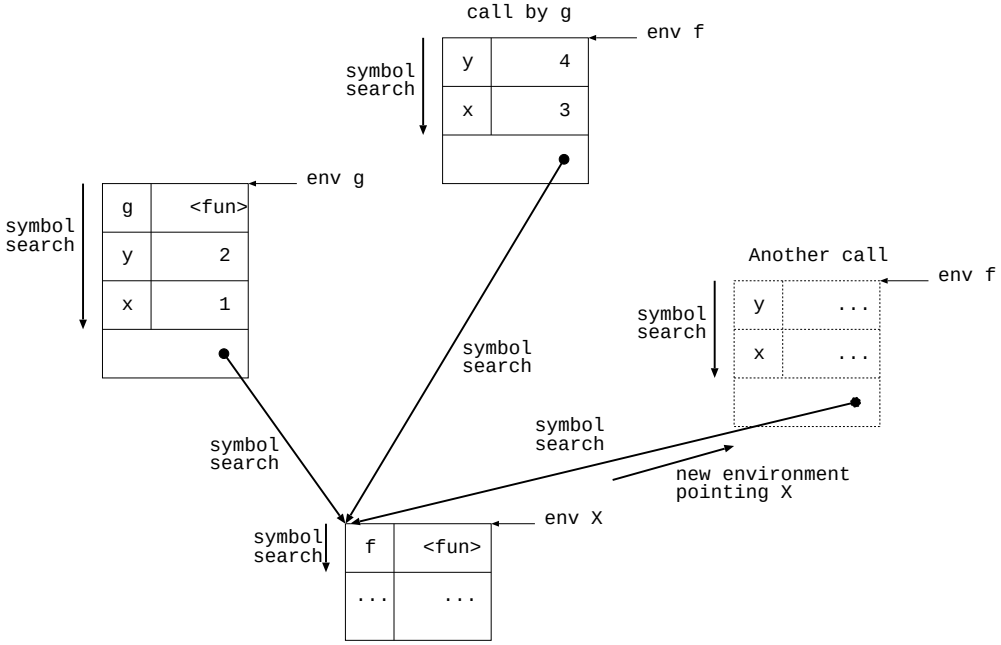


図 41　コメントの位置の環境

要があるのは、環境 f は関数 f が呼び出される度に異なるということです。 f の定義位置である環境 X は共通ですが、環境 f は関数 f が呼び出される度に環境 X を指す環境を伸長する必要があります。



■クロージャの環境

以下のようなプログラムを考えます。

```
(define h
  (let ((x 1) (y 2))
    ;; env c
    (let ((f (lambda (z)
               ;; env l
               (+ x y z))))
      f)))

;; env X

(h 3)
;;; may be another call of (h x)
```

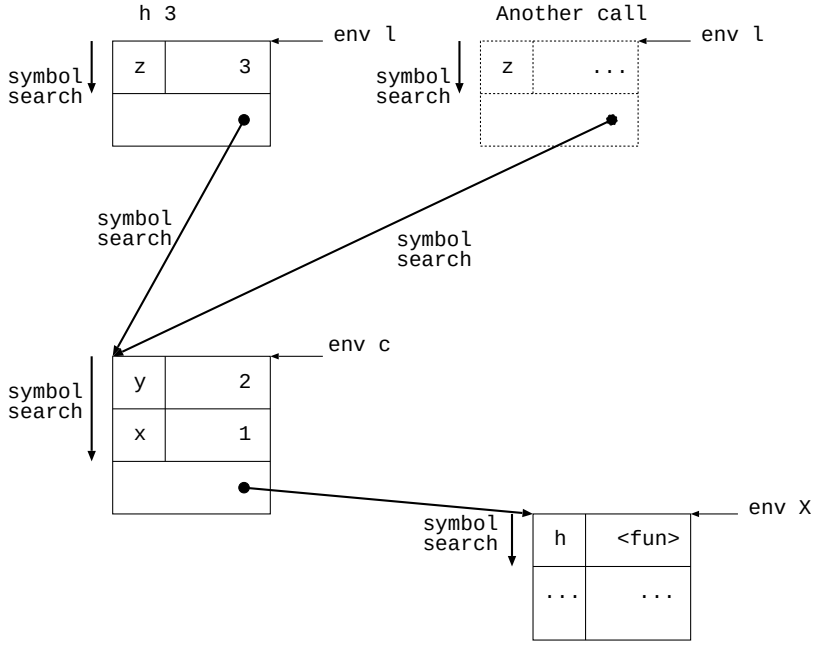
```
let rec h =
  let (x, y) = (1, 2) in
  (* env c *)
  let f z =
    (* env l *)
    x + y + z
  in f

(* env X *)

let v = h 3
(* may be another call of h x *)
```

関数が変数 h に束縛されています。しかも h は x, y を束縛している let の外側にあるにもかかわらず、呼び出した際に

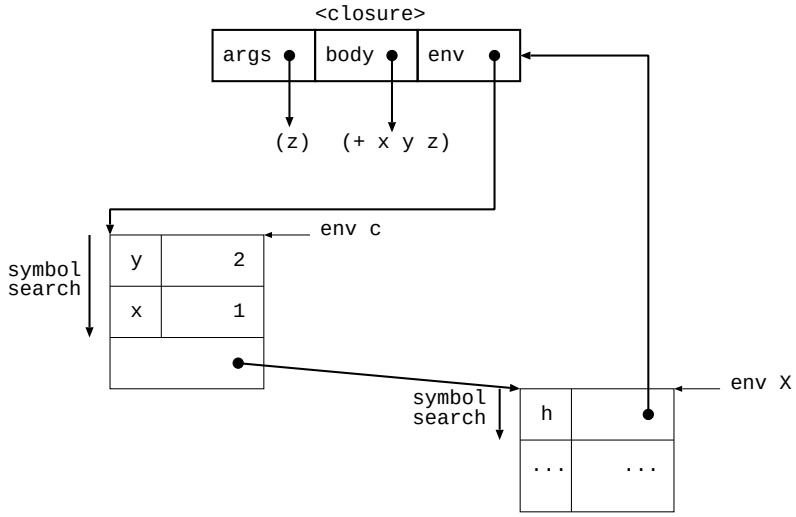
は x, y の値を評価します。

この let のような機能をレキシカルスコープと呼び、またこのような関数をレキシカルクロージャ (lexical closure) あるいは単にクロージャ (closure) と呼びます。

上の例での環境を考えてみると次のような構造になっているはずです。



h に束縛されている closure はあきらかに環境 c を知っている必要があります。なぜなら呼び出し時には環境 c を指す環境を生成しなければならないからです。したがって closure は以下のような構造になっていると考えることができます。env は closure の定義位置の環境で args は仮引数リストそして body は関数本体の式です。



式の評価に必要な環境を評価器内で渡しながら、 let や関数呼び出しの際には環境を伸長する方針を取ると、比較的簡単にインタプリタを実装することができます。また、環境を保持する構造を考えれば closure を実現することもできます。

参考資料: `http://www.sato.kuis.kyoto-u.ac.jp/~igarashi/class/isle4-05w/text/eopl003.html`

18.3 OCaml での lexing と parsing - ocamllex と ocamlyacc

18.3.1 ocamllex

ocamllex は OCaml に付属している字句解析関数生成器 (lexer generator) で、 OCaml から呼び出せる lexer を生成してくれます。以下のように C 言語でもおなじみの lex, flex と似たような使用感になっています。

```
{
  (* header *)
  (* Lexing のルール部分で参照したい内容を OCaml で書く *)
}

(* 文字列パターンの定義 *)

(* 字句解析 (lexing) のルール記述 *)

{
  (* trailer *)
  (* ルール部分で生成された関数を参照する内容を OCaml で書く *)
}
```

18.3.2 ocamlyacc

同様に、 ocamlyacc は OCaml に付属している構文解析関数生成器 (parser generator) で、 OCaml から呼び出せる parser を生成してくれます。こちらもやはり、以下のように yacc, bison と似たような使用感になっています。

```
%{
  (* header *)
  (* 構文木生成処理で参照したい内容を OCaml で書く *)
%}
  /* declarations */
  /* 終端記号の型や構文木の root の宣言 */
%%
  /* rules */
  /* 文脈自由文法と構文木生成処理を記述 */
%%
  (* trailer *)
  (* 生成された parser の関数を参照する内容を OCaml で書く *)
```

18.3.3 ocamllex と ocamlyacc の使用例

ocamllex と ocamlyacc を併用する場合には、 ocamlyacc で生成させた parser の終端記号の定義を lexer から参照させるようにするのがもっとも単純な使い方です。

以下が S 式 parser を生成させる例です。 sParser.mly を ocamlyacc で処理すると sParser.ml が生成され、sLexer.mll を ocamllex で処理すると sLexer.ml が生成されます。

■sParser.mly 型ごとに終端記号を %token で宣言し、 %start と %type で構文木の root とその型を指定します。 root の名前が構文解析関数の名前になります。

yacc と同じ要領で文脈自由文法を記述していきます。 expr は真偽値、数値、文字列であるか、あるいは、 expr ドット対または expr を並べたもの を括弧で括ったものという定義になっています。

```
%{
  (* sParser.mly *)

  module C = SCons

%}

/* File sparser.mly */
%token LPAREN RPAREN DOT_SYMBOL EOL BOOL_TRUE BOOL_FALSE
%token <SCons.s_int> INT
%token <float> FLOAT
%token <string> SYMBOL
%token <string> STRING
%start expr
%type <SCons.s_expr> expr
%%

expr_list:
  { C.Null }
| expr DOT_SYMBOL expr { C.Cons($1, $3) }
| expr expr_list { C.Cons($1, $2) }

expr:
| BOOL_TRUE               { C.Bool(true) }
| BOOL_FALSE              { C.Bool(false) }
| INT                     { C.Int($1) }
| FLOAT                   { C.Float($1) }
| SYMBOL                  { C.Symbol($1) }
| STRING                  { C.String($1) }
| LPAREN expr_list RPAREN { $2 }
```

■sLexer.mll　文字列パターンの定義では、正規表現の要領で文字集合や繰り返しの表現を利用して定義を作り、 let で名前を付けていきます。文字を ” でくくる以外は正規表現と同様です。定義した文字列パターンをさらに別の文字列パターンに再利用することができます。

ルール記述の部分では、 token の文字列パターンと token 生成式を lex の要領で記述していきます。キーワード rule の後の文字列が lexer の関数名になります。なのでここではその関数の名前は token です。

字句解析 (Lexing) の過程における入力ファイル内の位置は、 lexbuf の lex_start_p に記号 (token) の開始位置が、lex_curr_p に token の終了の次の位置が保持されています。 ocamllex デフォルトの動作では pos_cnum フィールドが更新されるのみなので、ファイル先頭からのバイト数しかわかりません。陽に行数の認識やタブによるカラム数の補正を行なう場合には、 lex_start_p と token をもとに lex_curr_p を修正してやる必要があります。[*56]

```
{
  (* sLexer.mll*)

  module LX = Lexing
  module P = SParser
  ... (* 中略 *)
  let fix_position lexbuf =
    let newline pos = {
      pos with
        LX.pos_lnum = pos.LX.pos_lnum + 1;
        LX.pos_cnum = pos.LX.pos_cnum + 1;
        LX.pos_bol = pos.LX.pos_cnum + 1;
    } in

    let tab pos = {
      pos with
        LX.pos_cnum = pos.LX.pos_cnum + 8 - (pos.LX.pos_cnum - pos.LX.pos_bol) mod 8
    } in

    let other pos = {
      pos with
        LX.pos_cnum = pos.LX.pos_cnum + 1
    } in

    let rec fix_pos_rec pos str =
      let len = (String.length str) in
        match (if len > 0 then (Some (str.[0]), String.sub str 1 (len - 1))
               else (None, "")) with
            (None, _) -> pos
          | (Some '\n', rest) -> fix_pos_rec (newline pos) rest
          | (Some '\t', rest) -> fix_pos_rec (tab pos) rest
          | (Some _, rest) -> fix_pos_rec (other pos) rest
    in
    let _ = lexbuf.LX.lex_curr_p <- fix_pos_rec (LX.lexeme_start_p lexbuf) (LX.lexeme lexbuf) in
      ()
}

/* 文字列パターンの定義 */
let str_esc = '\\'
let double_quote = '"'
let str_escaped_char = str_esc _
let str_char = [^ '\\' '"']
let str = double_quote (str_char | str_escaped_char)* double_quote

let left_paren = '('
let right_paren = ')'
let space = [' ' '\t' '\n' '\r']+
let dot_symbol = '.'
let bool_true  = '#' 't'
let bool_false = '#' 'f'

let int = '-'? ['0' - '9']+
let float = '-'? ['0' - '9']+ '.' ['0' - '9']* | '-'? ['0' - '9']* '.' ['0' - '9']+
let symbol = [^ '"' '(' ')' ' ' '\t' '\n' '\r']+

/* lexing のルール記述 */
rule token = parse
  | left_paren      { P.LPAREN }
  | right_paren     { P.RPAREN }
  | space      { fix_position lexbuf; token lexbuf }
  | dot_symbol      { P.DOT_SYMBOL }
  | bool_true       { P.BOOL_TRUE }
  | bool_false      { P.BOOL_FALSE }
  | int      { expr_integer (LX.lexeme lexbuf) }
  | float    { P.FLOAT(Pervasives.float_of_string(LX.lexeme lexbuf)) }
  | symbol   { P.SYMBOL(LX.lexeme lexbuf) }
  | str      { fix_position lexbuf; P.STRING(expr_string(LX.lexeme lexbuf)) }
  | eof      { raise Eof }
```

[*56] 次の lex_start_p は現在の lex_curr_p から引き継がれるので、 lex_curr_p を修正すれば十分です。

18.4 Haskell の Lexing

Haskell にはブロックの開始や終了の token や式の区切りの token を省略することができる layout rule という機能があります。そのため省略された token を lexing の過程で補ってやる必要があります。

まず、通常と同様に lexing を行なって token 列を生成し、その token 列に規則に従って token を補うというように、2 段階の工程を行ないます。

18.4.1 layout なしの Lexing

■lexer0.mll header 部分

まずは.mll の header 部分です。

後から layout rule において必要となるカラム数を数えあげる処理ために位置情報の修正を行なう関数 (fix_position) を定義しています。また、 Haskell の文字および文字列リテラルはリテラル内のルールが複雑度の高い仕様なので、別の lexer(後述) を呼び出しつつ実際の文字列表現を構成する関数 (decode_char, decode_string) を準備しています。

■fix_position 位置情報の修正

```
{
  (* lexer0.mll header 部分 *)
  module LX = Lexing
  module P = Parser
  ... (* 中略 *)
  let fix_position lexbuf =
    let newline pos =
      { pos with
          LX.pos_lnum = pos.LX.pos_lnum + 1;
          LX.pos_cnum = pos.LX.pos_cnum + 1;
          LX.pos_bol = pos.LX.pos_cnum + 1;
      } in

    let tab pos =
      { pos with
          LX.pos_cnum = pos.LX.pos_cnum + 8 - (pos.LX.pos_cnum - pos.LX.pos_bol) mod 8
      } in

    let other pos =
      { pos with
          LX.pos_cnum = pos.LX.pos_cnum + 1
      } in

    let rec fix_pos_rec pos str =
      let len = (String.length str) in
        match (if len > 0 then (Some (str.[0]), String.sub str 1 (len - 1))
               else (None, "")) with
            (None, _) -> pos
          | (Some '\n', rest) -> fix_pos_rec (newline pos) rest
          | (Some '\t', rest) -> fix_pos_rec (tab pos) rest
          | (Some _, rest) -> fix_pos_rec (other pos) rest
    in
    let _ = lexbuf.LX.lex_curr_p <- fix_pos_rec (LX.lexeme_start_p lexbuf) (LX.lexeme lexbuf) in
      ()
  ... (* 中略 *)
```

■decode_char, decode_string **文字および文字列デコーダー**

```
  ... (* 中略 *)
  let decode_cexpr cexpr =
    let fchar = String.get cexpr 0 in
    let escexp = String.sub cexpr 1 ((String.length cexpr) - 1) in
    let fmatch exp str = Str.string_match (Str.regexp exp) str 0 in
      if fchar = '\\' then
        match escexp with
            "NUL"   -> Some '\x00'
          | "SOH" | "^A"   -> Some '\x01'
          | "STX" | "^B"   -> Some '\x02'
        ... (* 中略 *)
          | "RS"  | "^^"   -> Some '\x1e'
          | "US"  | "^_"   -> Some '\x1f'
          | "SP"           -> Some ' '

          | "\\"           -> Some '\\'
          | "\""           -> Some '"'
          | "'"            -> Some '\''

          | "DEL"          -> Some '\x7f'

          | _ when fmatch "^[0-9]+$" escexp
              -> Some (Char.chr (int_of_string escexp))
          | _ when fmatch "^[xX][0-9a-zA-Z]+$" escexp
              -> Some (Char.chr (int_of_string ("0" ^ escexp)))
          | _ when fmatch "^[oO][0-7]+$" escexp
              -> Some (Char.chr (int_of_string ("0" ^ escexp)))

          | _ -> None

      else Some fchar

  let decode_char lexbuf =
    let cstr = LX.lexeme lexbuf in
    let len = String.length cstr in
      match decode_cexpr (String.sub cstr 1 (len - 2)) with
          Some c -> c
        | None   -> failwith (F.sprintf "Unkown char expression %s" cstr)

  let decode_string lexbuf =
    let sexpr = LX.lexeme lexbuf in
    let len = String.length sexpr in
    let strlbuf = Lexing.from_string (String.sub sexpr 1 (len - 2)) in
    let rec decode result =
      match HsStr.char strlbuf with
          HsStr.Eos -> result
        | HsStr.Char cstr ->
            if cstr = "\\&" then decode (result ^ "&")
            else decode (result ^
                           match (decode_cexpr cstr) with
                               None -> failwith (F.sprintf "Unkown char expression '%s' in literal string" cstr)
                             | Some c -> (String.make 1 c))
        | HsStr.Gap g -> decode result
    in decode ""
}
```

■lexer0.mll **文字列パターン定義部分**

次に文字列パターンの定義です。

行数はちょっと多いですが、特に難しいところはありません。問題のリテラル文字列ですが、リテラル文字列部分の文字列パターン自体は問題なく表現できています。しかし、リテラルで表現される文字列自体を復元するのが複雑なので前記および後述のような準備が必要になります。

```
/* lexer0.mll 文字列パターン定義部分 */
let special = ['(' ')' ',' ';' '[' ']' '`' '{' '}']

let space = ' '
let newline = ("\r\n"|['\n' '\r'])
let tab = '\t'

let dashes = '-' '-' '-'*

let ascSmall = ['a'-'z']
let small = ascSmall | '_'
let ascLarge = ['A'-'Z']
let large = ascLarge

let plus = '+'
let minus = '-'
let exclamation = '!'
let ascSymbol_nbs = [ '!' '#' '$' '%' '&' '*' '+' '.' '/' '<' '=' '>' '?' '@' '^' '|' '-' '~' ]
let ascSymbol = ascSymbol_nbs | '\\'
let symbol = ascSymbol

let ascDigit = ['0'-'9']
let digit = ascDigit

let octit = ['0'-'7']
let hexit = ascDigit | ['a'-'z' 'A'-'Z']

let decimal = (digit)+
let octal = (octit)+
let hexadecimal = (hexit)+

let exponent = ['e' 'E'] ['+' '-']? decimal
let float = decimal '.' decimal exponent? | decimal exponent

let graphic = small | large | symbol | digit | special | [':' '"' '\'']
let any = graphic | space | tab

let comment = dashes ((space | tab | small | large | symbol | digit | special | [':' '"' '\'']) (any)*)? newline

let whitechar = newline | space | tab
let whitestuff = whitechar | comment
let whitespace = (whitestuff)+

(*
let lwhitechar = space | tab
let lwhitestuff = lwhitechar | comment
let lwhitespace = (lwhitestuff)+
*)

let char_gr = small | large | ascSymbol_nbs | digit | special | [':' '"']
let str_gr  = small | large | ascSymbol_nbs | digit | special | [':' '\'']

let charesc = ['a' 'b' 'f' 'n' 'r' 't' 'v' '\\' '"' '\'']
let str_charesc = charesc | '&'
let cntrl = ascLarge | ['@' '[' '\\' ']' '^' '_']
let gap = '\\' (whitechar)+ '\\'
(* let gap = '\\' (lwhitechar | newline)+ '\\' *)

let ascii = ('^' cntrl) | "NUL" | "SOH" | "STX" | "ETX" | "EOT" | "ENQ" | "ACK"
  | "BEL" | "BS" | "HT" | "LF" | "VT" | "FF" | "CR" | "SO" | "SI" | "DLE"
  | "DC1" | "DC2" | "DC3" | "DC4" | "NAK" | "SYN" | "ETB" | "CAN"
  | "EM" | "SUB" | "ESC" | "FS" | "GS" | "RS" | "US" | "SP" | "DEL"

let escape = '\\' ( charesc | ascii | decimal | 'o' octal | 'x' hexadecimal )
let str_escape = '\\' ( str_charesc | ascii | decimal | 'o' octal | 'x' hexadecimal )

let char = '\'' (char_gr | space | escape) '\''
let string = '"' (str_gr | space | str_escape | gap)* '"'

let varid = small (small | large | digit | '\'')*
let conid = large (small | large | digit | '\'')*

let varsym = symbol (symbol | ':')*
let consym = ':' (symbol | ':')*

let modid = conid
```

■lexer0.mll **ルール記述部分**

最後にルール記述です。

スペース、タブ、改行などを含んでいる whitespace や string のところで fix_position を呼んで位置情報を補正しています。また char や string のリテラルから文字や文字列を構成するために decode_char, decode_string を呼び出しています。

```
(* lexer0.mll ルール記述部分 *)
rule token = parse
  | '('  { P.SP_LEFT_PAREN(loc lexbuf) }
  | ')'  { P.SP_RIGHT_PAREN(loc lexbuf) }
  | ','  { P.SP_COMMA(loc lexbuf) }
  | ';'  { P.SP_SEMI(loc lexbuf) }
  | '['  { P.SP_LEFT_BRACKET(loc lexbuf) }
  | ']'  { P.SP_RIGHT_BRACKET(loc lexbuf) }
  | '`'  { P.SP_B_QUOTE(loc lexbuf) }
  | '{'  { P.SP_LEFT_BRACE(loc lexbuf) }
  | '}'  { P.SP_RIGHT_BRACE(loc lexbuf) }
     (** special tokens *)

  | "case"     { P.K_CASE(loc lexbuf) }
  | "class"    { P.K_CLASS(loc lexbuf) }
  | "data"     { P.K_DATA(loc lexbuf) }
  | "default"  { P.K_DEFAULT(loc lexbuf) }
  | "deriving" { P.K_DERIVING(loc lexbuf) }
  | "do"       { P.K_DO(loc lexbuf) }
  | "else"     { P.K_ELSE(loc lexbuf) }
  | "if"       { P.K_IF(loc lexbuf) }
  | "import"   { P.K_IMPORT(loc lexbuf) }
  | "in"       { P.K_IN(loc lexbuf) }
  | "infix"    { P.K_INFIX(loc lexbuf) }
  | "infixl"   { P.K_INFIXL(loc lexbuf) }
  | "infixr"   { P.K_INFIXR(loc lexbuf) }
  | "instance" { P.K_INSTANCE(loc lexbuf) }
  | "let"      { P.K_LET(loc lexbuf) }
  | "module"   { P.K_MODULE(loc lexbuf) }
  | "newtype"  { P.K_NEWTYPE(loc lexbuf) }
  | "of"       { P.K_OF(loc lexbuf) }
  | "then"     { P.K_THEN(loc lexbuf) }
  | "type"     { P.K_TYPE(loc lexbuf) }
  | "where"    { P.K_WHERE(loc lexbuf) }
  | "_"        { P.K_WILDCARD(loc lexbuf) }
     (** reservedid *)

  | ".."       { P.KS_DOTDOT(loc lexbuf) }
  | ":"        { P.KS_COLON(loc lexbuf) }
  | "::"       { P.KS_2_COLON(loc lexbuf) }
  | "="        { P.KS_EQ(loc lexbuf) }
  | "\\"       { P.KS_B_SLASH(loc lexbuf) }
  | "|"        { P.KS_BAR(loc lexbuf) }
  | "<-"       { P.KS_L_ARROW(loc lexbuf) }
  | "->"       { P.KS_R_ARROW(loc lexbuf) }
  | "@"        { P.KS_AT(loc lexbuf) }
  | "~"        { P.KS_TILDE(loc lexbuf) }
  | "=>"       { P.KS_R_W_ARROW(loc lexbuf) }
     (** reservedop *)

  | "as"              { P.K_AS(loc lexbuf) }  (** maybe varid *)
  | "qualified"       { P.K_QUALIFIED(loc lexbuf) }  (** maybe varid *)
  | "hiding"          { P.K_HIDING(loc lexbuf) }  (** maybe varid *)
  | varid      { P.T_VARID(LX.lexeme lexbuf, loc lexbuf) }
  | conid      { P.T_CONID(LX.lexeme lexbuf, loc lexbuf) }
     (** identifiers or may be qualified ones *)

  | whitespace  { fix_position lexbuf; P.WS_WHITE(loc lexbuf) }  (** comment begining with dashes is not varsym *)
     (** white spaces *)

  | plus       { P.KS_PLUS(loc lexbuf) }  (** maybe varsym *)
  | minus      { P.KS_MINUS(loc lexbuf) } (** maybe varsym *)
  | exclamation  { P.KS_EXCLAM(loc lexbuf) } (** maybe varsym *)
  | varsym     { P.T_VARSYM(LX.lexeme lexbuf, loc lexbuf) }
  | consym     { P.T_CONSYM(LX.lexeme lexbuf, loc lexbuf) }
     (** symbols or may be qualified ones *)

  | modid '.' varid   { P.T_MOD_VARID(decode_with_mod lexbuf, loc lexbuf) }
  | modid '.' conid   { P.T_MOD_CONID(decode_with_mod lexbuf, loc lexbuf) }
  | modid '.' varsym  { P.T_MOD_VARSYM(decode_with_mod lexbuf, loc lexbuf) }
  | modid '.' consym  { P.T_MOD_CONSYM(decode_with_mod lexbuf, loc lexbuf) }
     (** qualified xx *)

  | char      { P.L_CHAR(decode_char lexbuf, loc lexbuf) }
  | string    { fix_position lexbuf; P.L_STRING(decode_string lexbuf, loc lexbuf) }

  | decimal | ('0' ['o' 'O'] octal) | ('0' ['x' 'X'] hexadecimal)
       { P.L_INTEGER(Int64.of_string(LX.lexeme lexbuf), loc lexbuf) }

  | float      { P.L_FLOAT(float_of_string(LX.lexeme lexbuf), loc lexbuf) }

  | eof        { P.EOF(loc lexbuf) }
  ... /* 以下略 */
```

■hsStr.mll

文字列の lexer です。

ここでの token は文字列リテラル内の 1 文字の表現あるいはギャップ (gap) です。 Haskell では 1 つの文字列リテラルを中断して、間に空白や改行やコメントを記述した後に、再開することができます。この空白や改行やコメントの部分が gap です。

ここで定義された char 関数を利用して decode_string 関数は文字列を構成していくようになっています。

```
{
  (* hsStr.mll *)
  module LX = Lexing

  type ct =
      Char of string
    | Gap of string
    | Eos
}

let special = ['(' ')' ',' ';' '[' ']' '`' '{' '}']

let space = ' '
let newline = ("\r\n"|['\n' '\r'])
let tab = '\t'

let ascSmall = ['a'-'z']
let small = ascSmall
let ascLarge = ['A'-'Z']
let large = ascLarge

let ascSymbol_nbs = [ '!' '#' '$' '%' '&' '*' '+' '.' '/' '<' '=' '>' '?' '@' '^' '|' '-' '~' ]

let ascDigit = ['0'-'9']
let digit = ascDigit

let octit = ['0'-'7']
let hexit = ascDigit | ['a'-'z' 'A'-'Z']

let decimal = (digit)+
let octal = (octit)+
let hexadecimal = (hexit)+

let lwhitechar = space | tab

let str_gr  = small | large | ascSymbol_nbs | digit | special | [':' '\'']

let charesc = ['a' 'b' 'f' 'n' 'r' 't' 'v' '\\' '"' '\'']
let str_charesc = charesc | '&'
let cntrl = ascLarge | ['@' '[' '\\' ']' '^' '_']
let gap = '\\' (lwhitechar | newline)+ '\\'

let ascii = ('^' cntrl) | "NUL" | "SOH" | "STX" | "ETX" | "EOT" | "ENQ" | "ACK"
  | "BEL" | "BS" | "HT" | "LF" | "VT" | "FF" | "CR" | "SO" | "SI" | "DLE"
  | "DC1" | "DC2" | "DC3" | "DC4" | "NAK" | "SYN" | "ETB" | "CAN"
  | "EM" | "SUB" | "ESC" | "FS" | "GS" | "RS" | "US" | "SP" | "DEL"

let str_escape = '\\' ( str_charesc | ascii | decimal | 'o' octal | 'x' hexadecimal )

rule char = parse
  | str_gr | space | str_escape  { Char(LX.lexeme lexbuf) }
  | gap                          { Gap(LX.lexeme lexbuf) }
  | eof                          { Eos }
```

参考資料: `http://www.sampou.org/haskell/report-revised-j/lexemes.html`

18.4.2 Haskell の layout rule

この節の始めにも書いたように layout rule は、 token 列にさらに token を補ってやる処理です。まずは補うルールを確認してみましょう。以下に、 Haskell 98 Language Report の改訂版の和訳[*57]から引用してみます。

..... 引用ここから .....

レイアウトの影響は、この節では、レイアウトを用いているプログラムに、どのようにして、ブレースとセミコロンを追加するかを記述することによって指定する。この仕様は、変換を行う関数 L の形をとる。 L への入力は

- この Haskell レポートの字句構文で指定されたような字句の並びで、以下のような追加トークンがついているもの。
  - キーワード let、 where、 do あるいは of のあとに字句 { が続かない場合、トークン $\{n\}$ をキーワードの後に挿入する。ここで n は、もし次の字句があればそれのインデント、または、ファイルの終端に到達していれ

[*57] http://www.sampou.org/haskell/report-revised-j/syntax-iso.html#layout

ば 0 である。

– モジュールの最初の字句が { あるいは module では ないとき、その字句の前に $\{n\}$ を置く。ここで、 n はその字句のインデントである。

– 同一行で、字句の開始の前には白空白しかないとき、この字句の前に $< n >$ を置く。ここで n はこの字句のインデントで、 前の二つの規則の結果、その前には $\{n\}$ が置かれていない。 (注意: 文字列リテラルは複数行にまたがることがある – 2.6 節。したがって、

```
f = ("Hello \
       \Bill", "Jake")
```

では、\Bill の前に $< n >$ は挿入されることはない。なぜなら、完全な字句の開始場所ではないからだ。また、, の前にも $< n >$ は置かれることはない。なぜなら、その前に 白空白以外のものがあるからだ。)

- 「 レイアウト文脈」のスタックのそれぞれの要素は以下のどれかである。
  – ゼロ、これは文脈を明示的に囲うこと (たとえばプログラマが開ブレース を用意した場合) を示す。もし最も内側の文脈が 0 なら、囲まれた文脈が 終了するか、新しい文脈がプッシュされるまで、レイアウトトークンは挿入されない。
  – 正の整数、これは囲まれたレイアウト文脈のインデントカラム数

字句の「 インデント」は字句の最初の文字のカラム数である。ひとつの行のインデントとは最も左にある字句のインデントを表す。このカラム数を決定するために以下のような規約をもつ固定幅のフォントを仮定する。

- 改行、リターン、ラインフィードおよびフォームフィード文字はすべて新しい行を開始する
- 最初のコラムは 0 ではなく 1 である
- タブストップは 8 文字文ずつの位置にある
- タブ文字は現在位置から次のタブストップ位置までそろえるのに必要なだけの空白を挿入する。

レイアウトルールにあわせるために、ソースプログラム中の Unicode 文字は ASCII 文字と同じ幅の固定幅であると看倣す。しかしながら、見た目との混 乱を避けるためプログラマは暗黙のレイアウトの意味が非空白文字の幅に依 存するようなプログラムを書かないようにすべきである。

■適用

L tokens [ ] は、 tokens のレイアウトに関知しない変換をもたらす。ここで、 tokens はモジュールの字句解析および上述のようにカラム数表示子を追加した結果である。 L の定義は以下のとおり、ここでは「 :」をストリーム構築操作子として使い、「 [ ]」は空のストリームである。

```
L (<n>:ts) (m:ms)   = ; : (L ts (m:ms))  if m = n
                    = } : (L (<n>:ts) ms) if n < m
L (<n>:ts) ms       = L ts ms
L ({n}:ts) (m:ms)   = { : (L ts (n:m:ms)) if n > m   (Note 1)
L ({n}:ts) []       = { : (L ts [n]) if n > 0        (Note 1)
L ({n}:ts) ms       = { : } : (L (<n>:ts) ms)        (Note 2)
L (}:ts) (0:ms)     = } : (L ts ms)                  (Note 3)
L (}:ts) ms         = parse-error                    (Note 3)
L ({:ts) ms         = { : (L ts (0:ms))              (Note 4)
L (t:ts) (m:ms)     = } : (L (t:ts) ms) if m /= 0 and parse-error(t)  (Note 5)
L (t:ts) ms         = t : (L ts ms)
L [] []             = []
L [] (m:ms)         = } : L [] ms if m /=0           (Note 6)
```

.....引用ここまで 以下略 .....

だいぶ長くなってしまったので Note は省略です。

わかりにくいですが、ここでも内部的には 2 段階になっています。

まず元の token 列に一つ目の操作を適用します。以下のような規則だと考えるとわかりやすいかもしれません。

- let, where, do, of の後に { が無い場合には代わりにブロックの開始をあらわす token $\{n\}$ を挿入。
- ファイルの先頭もモジュール宣言が省略されていて { が無い場合には token $\{n\}$ でブロック開始。
- インデントのレベルで後からブロックを認識するために token $< n >$ を挿入しておく。

つぎに二つ目の操作である関数 L を適用します。

L はもとの token 列とインデントレベルのスタックを引数にとり、もとの token 列に token を挿入したものを返す関数です。やはり以下のような規則だと考えるとわかりやすいかもしれません。

- インデントレベル $< n >$ が同じレベルのブロック内であれば (スタック参照) ; を挿入して式を終了、ブロックを継続
- インデントレベル $< n >$ の方が浅ければ } を挿入してブロックを終了
- インデントレベル $< n >$ があって上のどちらでもなければ式を継続
- あるブロック内で (スタック参照) よりインデントレベルの深いブロック開始 $\{n\}$ があったら { を挿入しブロックを開始。ブロック開始をあらわすインデントレベル n をスタックに積む
- ブロック開始 $\{n\}$ が最も外側でも同様にブロック開始。 { を挿入し、 n をスタックに積む
- ブロック開始 $\{n\}$ があって上のどちらでもなければ { および } を挿入し空のブロックを作る。実はブロック開始ではなかったということがここでわかるので代わりに $< n >$ を挿入する。
- } があったらスタックから 0 を取り出す
- } があって 0 を降ろせなかったら parse error
- { があったらスタックに 0 を積む
- 上のどれでもなく、またブロック内であり、ブロックを継続すると parse error になるときは } を挿入してブロックを閉じる。
- 上のどれでもなく、またブロック内であるときは parse error にならない限りブロックを継続。
- トークンもスタックも空ならおわり
- トークンが空でスタックに 0 でない値が残っているなら } を挿入してスタックから取り出す。(0 があったらエラー)

参考資料: `http://www.sampou.org/haskell/report-revised-j/syntax-iso.html#layout`

■OCaml での実装

「 parse error にならない限りブロックを継続」のルールはかなり実装がやっかいでした。今回利用した ocamlyacc は yacc や bison と同じように LALR(1) の parser generator となっており、基本的には token 列を途中まで読み込んだ位置とその token および一つ先の token によって構文解析中の次の動作を決定しています。ここで出てきたような parse error になるまでブロックが閉じるかわからないような仕様とはあまり相性が良くありません。 backtrack を行なえるような parser ならこのような仕様に対してもより簡単に対応できると考えられます。今回は token 列の token ごとに parse error のフラグを付加してやりなおしを行なうことで対応しました。実行効率はよくありませんが問題になるほど token 数が多くならなければ大丈夫そうです。将来的には Packrat parsing のような方法で parser を書き直してみたいところです。

OCaml で上 layout rule を実装した関数が次のような感じです。 token 列への一つ目の操作と二つ目の操作を順番に載せています。 P.BLK_OPEN が $\{n\}$ で P.BLK_LEVEL が $< n >$ にあたるものです。もとの定義と似たよう記述で表現できているのがわかるでしょうか。

```
let all_token_rev_list lexbuf =
  let unget_s = S.create () in
  let get_token () = L0.token lexbuf in
  let blk_level_pair tk =
    let loc = L0.get_location tk in (loc.T.start_p.T.col + 1, loc) in
  let eof_token_p = (function P.EOF(_) -> true | _ -> false) in

  let rec scan_start () =
    match get_token () with
        (P.SP_LEFT_BRACE _ | P.K_MODULE _) as start -> start
      | P.WS_WHITE _ -> scan_start ()
      | other ->
          let _ = S.push other unget_s in
            P.BLK_OPEN (blk_level_pair other)
  in

  let scan_next prev =
    let rec scan_next_rec () =
      let cur =
        if (S.is_empty unget_s) then (get_token ())
        else (S.pop unget_s) in

        match (prev, cur) with
            (_, (P.EOF(_) as eoft)) -> eoft
          | (_, P.WS_WHITE(_)) -> (scan_next_rec ())
          | ((P.K_LET(_) | P.K_WHERE(_) | P.K_DO(_) | P.K_OF(_)), (P.SP_LEFT_BRACE(_) as lbr)) -> lbr
          | ((P.K_LET(_) | P.K_WHERE(_) | P.K_DO(_) | P.K_OF(_)), tk) ->
              let (_, (level, loc)) = (S.push tk unget_s, blk_level_pair tk) in
                P.BLK_OPEN((if (eof_token_p tk) then 0 else level), loc)
          | (_, tk) ->
              let (_, loc) as p = blk_level_pair tk in
                if (loc.T.start_p.T.line
                    - (L0.get_location prev).T.end_p.T.line) > 0 then
                  let _ = S.push tk unget_s in P.BLK_LEVEL p
                else tk
    in (scan_next_rec ())
  in
    (LST.fold_left
       (fun r a -> ((a, new_err_flag ()) :: r))
       []
       (LST.create_stream (scan_start ()) scan_next eof_token_p))
```

```
let rec layout istream levels =
  let push_new_token tok lform =
    LST.Cons ((tok, new_err_flag ()), lform)
  in

  let (tok, err) =
    match LST.peek istream with
        None -> raise Parsing.Parse_error
      | Some x -> x
  in
    match (tok, levels) with
        ((P.BLK_LEVEL (n, loc)), (m :: mstl as ms)) when m = n ->
          let addtk = P.SP_SEMI(loc) in
            push_new_token addtk (lazy (layout (LST.tl istream) ms))
      | ((P.BLK_LEVEL (n, loc)), m :: ms) when n < m  ->
          push_new_token (P.SP_RIGHT_BRACE(loc)) (lazy (layout istream ms))
      | ((P.BLK_LEVEL (n, _)), ms)                    -> layout (LST.tl istream) ms
      | ((P.BLK_OPEN (n, loc)), (m :: ms as levels)) when n > m  ->
          push_new_token (P.SP_LEFT_BRACE(loc)) (lazy (layout (LST.tl istream) (n :: levels))) (* Note 1 *)
      | ((P.BLK_OPEN (n, loc)), []) when n > 0        ->
          push_new_token (P.SP_LEFT_BRACE(loc)) (lazy (layout (LST.tl istream) [n])) (* Note 1 *)
      | ((P.BLK_OPEN (n, loc)), ms)                   ->
          push_new_token
            (P.SP_LEFT_BRACE(loc))
            (lazy (push_new_token
                     (P.SP_RIGHT_BRACE(loc))
                     (lazy (layout (push_new_token
                                      (P.BLK_LEVEL(n, loc))
                                      (lazy (LST.tl istream))) ms)))) (* Note 2 *)
      | ((P.SP_RIGHT_BRACE _ as rbr), 0 :: ms)        ->
          LST.Cons ((rbr, err), lazy (layout (LST.tl istream) ms)) (* Note 3 *)
      | ((P.SP_RIGHT_BRACE _), ms)                    -> raise Parsing.Parse_error (* Note 3 *)
      | ((P.SP_LEFT_BRACE _ as lbr), ms)              ->
          LST.Cons ((lbr, err), lazy (layout (LST.tl istream) (0 :: ms))) (* Note 4 *)

      | ((P.EOF loc as eoft), [])                     -> LST.Cons ((eoft, err), lazy (LST.Nil))
      | ((P.EOF loc), m :: ms) when m <> 0            ->
          push_new_token (P.SP_RIGHT_BRACE(loc)) (lazy (layout istream ms)) (* Note 6 *)

      | (t, (m :: mstl)) when m <> 0 && (!err)        ->
          err := false;
          push_new_token (P.SP_RIGHT_BRACE(L0.get_location t)) (lazy (layout istream mstl))
          (* parse-error(t) Note 5 case *)
      | (t, ((m :: mstl) as ms))                    ->
          LST.Cons ((t, err),
                    lazy (layout (LST.tl istream) ms))
      | (t, ms)                                       ->
          LST.Cons ((t, err),
                    lazy (layout (LST.tl istream) ms))
```

18.5 Haskell の Parsing

文脈自由文法の定義を全部載せてしまうと長すぎて大変なので、ここでは単項の expression の部分と、二項演算および二項演算パターンの定義を見ていくことにします。

18.5.1 Haskell の Expression

■単項の expression

単項の表現 aexp としてここで定義されているのは、変数、コンストラクタ、リテラル、 括弧でくくられた expression、 タプル、 リスト、 リスト内包表記、 括弧でくくられた left secion、括弧でくくられた right secion (section は二項演算式でどちらか充足しているもの)、 レコードの生成、 レコードをコピーして更新です。単項の表現は二項演算の引数、関数、関数の引数となり得ます。

```
/* parser.mly 単項 expression */
/*
 aexp    ->      qvar    (variable)
        |       gcon    (general constructor)
        |       literal
        |       ( exp )         (parenthesized expression)
        |       ( exp1 , ... , expk )   (tuple, k>=2)
        |       [ exp1 , ... , expk ]   (list, k>=1)
        |       [ exp1 [, exp2] .. [exp3] ]     (arithmetic sequence)
        |       [ exp | qual1 , ... , qualn ]   (list comprehension, n>=1)
        |       ( expi+1 qop(a,i) )      (left section)
        |       ( lexpi qop(l,i) )       (left section)
        |       ( qop(a,i)<-> expi+1 )   (right section)
        |       ( qop(r,i)<-> rexpi )    (right section)
        |       qcon { fbind1 , ... , fbindn }  (labeled construction, n>=0)
        |       aexp<qcon> { fbind1 , ... , fbindn }    (labeled update, n >= 1)
*/

aexp:
  qvar  { E.VarE $1 }   /*(variable)*/
| gcon  { E.ConsE $1 }  /*(general constructor)*/
| literal  { E.LiteralE $1 }
| SP_LEFT_PAREN exp SP_RIGHT_PAREN  { E.ParenE $2 }     /*(parenthesized expression)*/
| SP_LEFT_PAREN exp SP_COMMA exp_list SP_RIGHT_PAREN  { E.TupleE ($2 :: $4) }   /*(tuple, k>=2)*/
| SP_LEFT_BRACKET exp_list SP_RIGHT_BRACKET  { E.ListE ($2) }   /*(list, k>=1)*/
| SP_LEFT_BRACKET exp KS_DOTDOT SP_RIGHT_BRACKET  { E.ASeqE($2, None, None) }   /*(arithmetic sequence)*/
| SP_LEFT_BRACKET exp SP_COMMA exp KS_DOTDOT SP_RIGHT_BRACKET  { E.ASeqE($2, Some $4, None) }   /*(arithmetic sequence)*/
| SP_LEFT_BRACKET exp KS_DOTDOT exp SP_RIGHT_BRACKET  { E.ASeqE($2, None, Some $4) }    /*(arithmetic sequence)*/
| SP_LEFT_BRACKET exp SP_COMMA exp KS_DOTDOT exp SP_RIGHT_BRACKET  { E.ASeqE($2, Some $4, Some $6) }
    /*(arithmetic sequence)*/
| SP_LEFT_BRACKET exp KS_BAR qual_list SP_RIGHT_BRACKET  { E.LCompE ($2, $4) }  /*(list comprehension, n>=1)*/

| SP_LEFT_PAREN op2_left_section SP_RIGHT_PAREN  { E.MayLeftSecE ($2) }         /*(left section)*/
| SP_LEFT_PAREN op2_right_section SP_RIGHT_PAREN  { E.MayRightSecE ($2) }       /*(right section)*/

| qcon SP_LEFT_BRACE fbind_list SP_RIGHT_BRACE  { E.LabelConsE ($1, OH.of_list $3) }    /*(labeled construction, n>=1)*/
| qcon SP_LEFT_BRACE SP_RIGHT_BRACE  { E.LabelConsE ($1, OH.create 0) }         /*(labeled construction, n=0)*/
| aexp SP_LEFT_BRACE fbind_list SP_RIGHT_BRACE  { E.LabelUpdE ($1, OH.of_list $3) }     /*(labeled update, n >= 1)*/
;

exp_list:
  exp SP_COMMA exp_list  { $1 :: $3 }
| exp  { [$1] }
;

qual_list:
  qual SP_COMMA qual_list  { $1 :: $3 }
| qual  { [$1] }
;

fbind_list:
  fbind SP_COMMA fbind_list  { $1 :: $3 }
| fbind  { [$1] }
;

qual:
  pat KS_L_ARROW exp  { LC.Gen($1, $3) }        /*(generator)*/
| K_LET decl_list  { LC.Let $2 }        /*(local declaration)*/
| exp  { LC.Guard $1 }  /*(guard)*/
;
```

18.5.2 二項演算子の優先順位

■二項演算 expression **および二項演算パターンの構文解析**

Haskell の二項演算子には level 0 から level 9 までの優先順位 (おおきい方が先に結合) があり、その優先順位と結合規

則 (右結合、左結合、なし) を**演算子を定義しているソースコード内で指定することができます**。しかも通常の関数を二項演算子として扱う[*58]こともでき、括弧なしで非常に複雑な式が記述可能です。パターンについても、関数にもなっているコンストラクタを二項演算子として使って、二項演算式の形式でパターンを記述していけます。もちろん二項演算子として使った場合のコンストラクタにも優先順位と結合規則があり、指定の方法は同様となっています。

ここで問題となるのは**構文解析時に優先順位が決定していない**ということです。ここでは expression、パターン共に、引数と演算子が交互に連なっているリストとして構文解析を行なっています。

```
/* parser.mly 二項演算 expression */
... /* 略 */
/* expression */
exp:
  exp0  { E.Top ($1, None) }
| exp0 KS_2_COLON context KS_R_W_ARROW typ  { E.Top ($1, Some ($5, Some $3)) }  /*(expression type signature)*/
| exp0 KS_2_COLON typ  { E.Top ($1, Some ($3, None)) }  /*(expression type signature)*/

/*
lexp6:
  - exp7
;
*/

/*
expi    ->      expi+1 [qop(n,i) expi+1]
        |       lexpi
        |       rexpi
lexpi   ->      (lexpi | expi+1) qop(l,i) expi+1
rexpi   ->      expi+1 qop(r,i) (rexpi | expi+1)
*/

/*
exp0:   ->      [-] exp10 {qop [-] exp10}
*/

exp0:
  op2_expn_list  { E.Exp0 $1 }

op2_expn_list:
  ks_minus exp10 op2_right_section  { E.ExpF (E.Minus $2, $3) }
| exp10 op2_right_section  { E.ExpF ($1, $2) }
| ks_minus exp10  { E.ExpF (E.Minus $2, E.Op2End) }
| exp10  { E.ExpF ($1, E.Op2End) }

op2_right_section:
  qop op2_expn_list { E.Op2F ($1, $2) }

op2_left_section:
  ks_minus exp10 qop op2_left_section  { E.ExpF (E.Minus $2, E.Op2F ($3, $4)) }
| exp10 qop op2_left_section  { E.ExpF ($1, E.Op2F($2, $3)) }
| ks_minus exp10 qop  { E.ExpF (E.Minus $2, E.Op2F ($3, E.Op2NoArg)) }
| exp10 qop  { E.ExpF ($1, E.Op2F ($2, E.Op2NoArg)) }

exp10:
  KS_B_SLASH apat_list KS_R_ARROW exp  { E.LambdaE ($2, $4) }   /*(lambda abstraction, n>=1)*/
| K_LET decl_list K_IN exp  { E.LetE ($2, $4) }         /*(let expression)*/
| K_IF exp K_THEN exp K_ELSE exp  { E.IfE ($2, $4, $6) }        /*(conditional)*/
| K_CASE exp K_OF SP_LEFT_BRACE alt_list SP_RIGHT_BRACE  { E.CaseE ($2, $5) }   /*(case expression)*/
| K_DO SP_LEFT_BRACE stmt_list_exp SP_RIGHT_BRACE  { E.DoE $3 }         /*(do expression)*/
| fexp  { E.FexpE $1 }

/*
 fexp    ->      [fexp] aexp     (function application)
*/

fexp:
  aexp_list  { E.make_fexp $1 }
;

aexp_list:
  aexp aexp_list  { fun fexp -> $2 (E.FappE (fexp, $1)) }
| aexp  { fun fexp -> E.FappE (fexp, $1) }
;
/* fexp -- FfunE (fexp) */
/* fexp ae1 -- FappE (FfunE (fexp), ae1) */
/* fexp ae1 ae2 -- FappE (FappE (FfunE (fexp), ae1), ae2) */
```

[*58] バッククオート (‘) でくくる

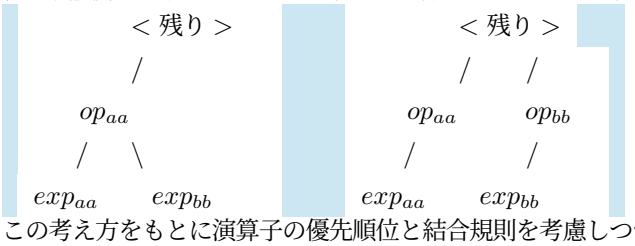
```
/* parser.mly パターンおよび 二項演算パターン */
.../* 略 */
pat:
  var ks_plus integer   /*(successor pattern)*/
      { match $3 with (S.Int (i), loc) -> P.PlusP($1, i, loc) | _ -> failwith "plus integer pattern syntax error." }
| pat0  { $1 }

/*
pati     ->      pati+1 [qconop(n,i) pati+1]
         |       lpati
         |       rpati
*/

/*
lpati   ->      (lpati | pati+1) qconop(l,i) pati+1
*/

/*
lpat6:
  ks_minus integer      (negative literal)
      { match $2 with
          (S.Int (v), l) -> S.P.MIntP (v, l)
        | _ -> failwith "negative integer literal pattern syntax error." }
| ks_minus float        (negative literal)
      { match $2 with
          (S.Float (v), l) -> S.P.MFloatP (v, l)
        | _ -> failwith "negative integer literal pattern syntax error." }
;
*/

/*
rpati   ->      pati+1 qconop(r,i) (rpati | pati+1)
*/

pat0:
  op2_patn_list  { P.Pat0 $1 }

op2_patn_list:
  ks_minus integer op2_patn_right
    { let p = match $2 with
        (S.Int (x), loc) -> P.MIntP (x, loc)
      | _ -> failwith "negative integer literal pattern syntax error."
      in P.PatF (p, $3)
    }
| ks_minus float op2_patn_right
    { let p = match $2 with
        (S.Float (x), loc) -> P.MFloatP (x, loc)
      | _ -> failwith "negative integer literal pattern syntax error."
      in P.PatF (p, $3)
    }
| pat10 op2_patn_right  { P.PatF ($1, $2) }

op2_patn_right:
  qconop op2_patn_list  { P.Op2F ($1, $2) }
|   { P.Op2End }

pat10:
  apat  { $1 }
| gcon apat_list        /*(arity gcon = k, k>=1)*/
      { P.ConP($1, $2) }

apat_list:
  apat apat_list { $1::$2 }
| apat           { [$1] }

apat:
  var
      { P.VarP $1 }
| var KS_AT apat        /*(as pattern)*/
      { P.AsP($1, $3) }
| gcon  /*(arity gcon = 0)*/
      { P.ConP($1, []) }
| qcon SP_LEFT_BRACE fpat_list SP_RIGHT_BRACE   /*(labeled pattern, k>=0)*/ /* may be error pattern */
      { P.LabelP($1, $3) }
| literal
      { P.LiteralP($1) }
| K_WILDCARD    /*(wildcard)*/
      { P.WCardP }
| SP_LEFT_PAREN pat SP_RIGHT_PAREN      /*(parenthesized pattern)*/
      { $2 }
| SP_LEFT_PAREN tuple_pat SP_RIGHT_PAREN        /*(tuple pattern, k>=2)*/
      { P.TupleP $2 }
| SP_LEFT_BRACKET list_pat SP_RIGHT_BRACKET     /*(list pattern, k>=1)*/
      { P.ListP $2 }
| KS_TILDE apat         /*(irrefutable pattern)*/
      { P.Irref $2 }
```

■二項演算子の優先順位の解決

一度、構文解析が完了した後に二項演算子の優先順位の解決を行ないます。構文木を再帰的に辿り、二項演算式の交互のリストを木構造に置き換えます。

具体的には、二項演算式の一番左側の 2 つの単項表現 exp_{aa}, exp_{bb} および 2 つの演算子 op_{aa}, op_{bb} について着目する

と、二項演算式は次の二通りのいずれかの形になっているはずです。



この考え方をもとに演算子の優先順位と結合規則を考慮しつつ再帰呼び出しの関数を実装すると以下のようになります。

```
  type 'exp op2list_opf =
      Op2F of (ID.idwl * 'exp op2list_expf)
    | Op2End
  and 'exp op2list_expf =
      ExpF of ('exp * 'exp op2list_opf)
(*     | UniOpF of (ID.idwl * 'exp * 'exp op2list_opf) *)
    | Op2NoArg
  ... (* 中略 *)
  let rec explist2term func list =
    let exp10_fun = SYA.maptree_exp10 func in

    let rec fold_leafs list =
      let scanned_op2exp op expAA expBB =
        E.VarOp2E (op,
                   exp10_fun expAA,
                   exp10_fun expBB) in
        match list with
          | E.ExpF (exp, E.Op2End) -> (* list *)
              E.uni_exp (exp10_fun exp)
          | E.ExpF (expAA, E.Op2F (op_aa,
                                   (E.ExpF (expBB, E.Op2End)))) ->
              E.uni_exp (scanned_op2exp op_aa expAA expBB)
          | E.ExpF (expAA, E.Op2F ((op_aa, _) as op_aa_wl,
                                   ((E.ExpF (expBB, E.Op2F ((op_bb, _) as op_bb_wl, rest))) as cdr))) ->
              begin
                let (aa_fixity, _) = eval_op2_fixity modbuf op_aa in
                let (bb_fixity, _) = eval_op2_fixity modbuf op_bb in
                  match (aa_fixity, bb_fixity) with
                    | ((_, aa_i), (_, bb_i)) when aa_i > bb_i ->
                        fold_leafs (E.expf_cons (scanned_op2exp op_aa_wl expAA expBB) op_bb_wl rest)
                    | ((SYN.InfixLeft, aa_i), (SYN.InfixLeft, bb_i)) when aa_i = bb_i ->
                        fold_leafs (E.expf_cons (scanned_op2exp op_aa_wl expAA expBB) op_bb_wl rest)
                    | ((_, aa_i), (_, bb_i)) when aa_i < bb_i ->
                        E.expf_cons expAA op_aa_wl (fold_leafs cdr)
                    | ((SYN.InfixRight, aa_i), (SYN.InfixRight, bb_i)) when aa_i = bb_i ->
                        E.expf_cons expAA op_aa_wl (fold_leafs cdr)
                    | _ ->
                        failwith (F.sprintf "Syntax error for operator priority. left fixity %s, right fixity %s"
                                    (SYN.fixity_str aa_fixity)
                                    (SYN.fixity_str bb_fixity))
              end
          | _ -> failwith "Arity 2 operator expression syntax error."
    in
      match fold_leafs list with
        | E.ExpF (exp, E.Op2End) -> exp
        | E.ExpF (exp, E.Op2F (_, E.Op2NoArg)) -> failwith "explist2term: section not implemented."
        | folded -> explist2term func folded
```

```
type 'pat op2list_opf =
    Op2F of (ID.idwl * 'pat op2list_patf)
  | Op2End
and 'pat op2list_patf =
    PatF of ('pat * 'pat op2list_opf)
  | Op2NoArg
... (* 中略 *)
let rec patlist2term min_i func list =
  let pat_fun = SYA.maptree_pat func in

  let rec fold_leafs list =
    let scanned_op2pat op patAA patBB =
      P.ConOp2P (op,
                 pat_fun patAA,
                 pat_fun patBB) in

      match list with
        | P.PatF (pat, P.Op2End) ->
            P.uni_pat (pat_fun pat)
        | P.PatF (patAA, P.Op2F (op_aa_wl, (P.PatF (patBB, P.Op2End)))) ->
            P.uni_pat (scanned_op2pat op_aa_wl patAA patBB)
        | P.PatF (patAA, P.Op2F ((op_aa, _) as op_aa_wl, ((P.PatF (patBB, P.Op2F ((op_bb, _) as op_bb_wl, rest))) as cdr))) ->
            begin
              let (aa_fixity, _) = eval_op2_fixity modbuf op_aa in
              let (bb_fixity, _) = eval_op2_fixity modbuf op_bb in
                match (aa_fixity, bb_fixity) with
                    ((_, aa_i), _) when aa_i < min_i ->
                      failwith (F.sprintf "Pat%d cannot involve fixity %s operator." min_i (SYN.fixity_str aa_fixity))
                  | (_, (_, bb_i)) when bb_i < min_i ->
                      failwith (F.sprintf "Pat%d cannot involve fixity %s operator." min_i (SYN.fixity_str bb_fixity))
                  | ((_, aa_i), (_, bb_i)) when aa_i > bb_i ->
                      fold_leafs (P.patf_cons (scanned_op2pat op_aa_wl patAA patBB) op_bb_wl rest)
                  | ((SYN.InfixLeft, aa_i), (SYN.InfixLeft, bb_i)) when aa_i = bb_i ->
                      fold_leafs (P.patf_cons (scanned_op2pat op_aa_wl patAA patBB) op_bb_wl rest)
                  | ((_, aa_i), (_, bb_i)) when aa_i < bb_i ->
                      P.patf_cons patAA op_aa_wl (fold_leafs cdr)
                  | ((SYN.InfixRight, aa_i), (SYN.InfixRight, bb_i)) when aa_i = bb_i ->
                      P.patf_cons patAA op_aa_wl (fold_leafs cdr)
                  | _ ->
                      failwith (F.sprintf "Syntax error for operation priority. left fixity %s, right fixity %s"
                                  (SYN.fixity_str aa_fixity)
                                  (SYN.fixity_str bb_fixity))
            end
        | _ -> failwith "Arity 2 operator pattern syntax error."
  in
    match fold_leafs list with
      | P.PatF (pat, P.Op2End) -> pat
      | P.PatF (pat, P.Op2F (_, P.Op2NoArg)) -> failwith "patlist2term: section not implemented."
      | folded -> patlist2term min_i func folded
```

18.6 Haskell の評価器

18.2.1 節でも述べた、環境を渡してゆく方法で Haskell の評価器を実装するために以下の型の環境 (env_t)、単純な closure(lambda_t)、関数定義のための closure(closure_t) を定義しました。

先の議論での closure は仮引数リスト、 body の expression、および環境の 3 つを持っているデータ型でした。こちらで同じ役割を果たす lambda_t では、 Haskell の関数の仮引数がパターン照合 (pattern match) を行なうので仮引数リストの代わりに pattern のリストを持っているのと、 Haskell の関数束縛宣言およびパターン照合による束縛宣言における where 節の環境を構築するための関数を保持するのフィールドを増やしています。

また、 Haskell の関数定義の pattern match によって複数の expression を書き分ける機能を、単純な closure に置き換えるのは困難なため、複数の closure を持つことのできる型 closure_t を導入しました。

```
type lambda_t = {
  arg_pat_list : P.pat list;
  body : E.t;
  lambda_env : env_t;
  apply_where : (env_t -> env_t);
}

and closure_t =
  | SPat of (lambda_t)
  | MPat of (lambda_t list)
  | Prim of (thunk_t list -> value_t)

and value_t =
  | Bottom
  | IO
  | Literal of SYN.literal
  | Cons of (ID.id * (thunk_t list))
  | LabelCons of (ID.id * (ID.id, thunk_t) OH.t )
  | Tuple of (thunk_t list)
  | List of (thunk_t list)
  | Closure of (closure_t * int * E.aexp list)

and thunk_t = unit -> value_t

and pre_value_t =
    Thunk of (unit -> value_t)
  | Thawed of value_t

and scope_t = (S.t, thunk_t) H.t

(* あるスコープでの環境 *)
and env_t = {
  symtabs : (scope_t) list;
  top_scope : scope_t;
}
```

18.6.1 遅延評価

Haskell の評価戦略はデフォルトで遅延評価 (lazy evaluation) です。次のようなプログラムを定義して g（f 1 2）2 を評価することを考えてみます。

```
f x y = x + y
g x y = x * y
```

話を簡単にするために +, * はプリミティブであるということにすると、まず g を評価すると（f 1 2）* 2 となります。つぎに、*はそれ以上評価しても意味がないプリミティブなので f が評価され、（1 + 2）* 2 となり、以下 3 * 2, 6 となって評価が終了します。

他の多数のプログラミング言語は eager evaluation を採用しているものが多く、その場合は関数の引数が完全に評価されたあとに関数が評価されます。書きくだしてみると、g（f 1 2）2 − > g（1 + 2）2 − > g 3 2 − > g 3 2 − > 3 * 2 − > 6 のような感じになるはずです。

■遅延評価の実装

lazy evaluation を実装するには、関数の引数を評価するときに最後まで評価するのではなく、引数の計算を行なうような closure を生成することで対応することができます。この小さな closure をここでは thunk と呼んでいます。環境 env_t の持っている値を thunk_t にしているのはそのためです。

make_thunk で thunk ごとに評価前の関数 (Thunk) または評価後の値 (Thawed) を保持する pre_value_t 型の構造を作っています。 thunk を初めて呼び出したときに評価が行なわれて値が保持され、以降の thunk 呼び出しでは単に Thawed の保持する値が返るようになります。

```
and thunk_t = unit -> value_t

and pre_value_t =
    Thunk of (unit -> value_t)
  | Thawed of value_t

and scope_t = (S.t, thunk_t) H.t

(* あるスコープでの環境 *)
and env_t = {
  symtabs : (scope_t) list;
  top_scope : scope_t;
}
...(* 中略 *)
let thunk_value thunk =
  match thunk with
      Thunk (f) -> f ()
    | Thawed (v) -> v

let expand_thunk thunk_ref =
  match !thunk_ref with
      Thunk (_)  ->
        let v = thunk_value (!thunk_ref) in
        let _ = thunk_ref := Thawed v in
          v
    | Thawed (v) -> v

let make_thawed value =
  (fun () -> value)

let make_thunk eval_fun env evalee =
  let delay_fun = fun () -> (eval_fun env evalee) in
  let thunk_ref = ref (Thunk delay_fun) in
    fun () -> expand_thunk thunk_ref
```

18.6.2 遅延パターン照合

Haskell のパターン照合 (pattern match) は lazy evaluation と組み合わさるようにして動作します。実際の評価に必要な部分しか pattern match に対応する expression を評価しないように動きます。

次のプログラムを考えます。

```
main = let { (p, (q, r)) = (print 1, (print 2, print 3)) } in
       q
```

このプログラムを実行すると 2 のみが出力されます。`(p, (q, r))` と `(print 1, (print 2, print 3))` の pattern match が行なわれますが実際に必要になる `q` のみが最後まで評価されます。

■遅延パターン照合の実装

pattern match の際に pattern に従って構造分解を行ない、その構造分解に対応した thunk から thunk への分解を行なっていきます。末端に変数の pattern があれば環境に thunk を書き込みます。 pattern match の失敗をたとえば case 式で認識するために真理値を返しています。

たとえばタプルの場合は、まずタプルに対応するはずの thunk を評価し、結果をタプルの要素に分解します。タプルのそれぞれの要素はやはりまた thunk になっているので、タプルの pattern のそれぞれの要素と pattern match を行なうように再帰します。それぞれの要素の pattern match が全て成功すれば、タプルの pattern match も成功です。

```
(* Lazy pattern match against thunk *)
and bind_pat_with_thunk pat =
  let sub_patterns_match env pat_list thunk_list =
    L.fold_left2
      (fun (matchp_sum, tlist_sum) pat thunk ->
         let (matchp, tlist) = bind_pat_with_thunk pat env thunk in
           (matchp_sum & matchp, L.append tlist_sum tlist))
      (true, [])
      pat_list
      thunk_list
  in
    match pat with
... (* 中略 *)
      | P.VarP (id, _) ->
          (fun env thunk ->
             let _ = bind_thunk_to_env env id thunk in (true, [thunk]))

      | P.AsP ((id, _), pat) ->
          (fun env thunk ->
             let (_, (matchp, tlist)) = (bind_thunk_to_env env id thunk,
                                         bind_pat_with_thunk pat env thunk)
             in (matchp, thunk :: tlist))
... (* 中略 *)
      | P.WCardP ->
          (fun _ thunk -> (true, [thunk]))

      | P.TupleP pat_list ->
          (fun env thunk ->
             let value = thunk () in
               match value with
                   Tuple (args) when (L.length args) = (L.length pat_list)
                     -> sub_patterns_match env pat_list args
                 | _ -> (false, [thunk]))

      | P.ListP pat_list ->
          (fun env thunk ->
             let value = thunk () in
               match value with
                   List (args) when (L.length args) = (L.length pat_list)
                     -> sub_patterns_match env pat_list args
                 | _ -> (false, [thunk]))
... (* 略 *)
```

18.7 まとめと今後の課題

結構な分量になってしまいましたが、それでもかなり端折って関数型言語のプログラミングと Haskell ライクなインタプリタを実装するにあたって苦心したトピックを紹介してみました。

今後の課題としては未実装の部分を実装していくことと、 ocamlyacc による parsing を Packrat parsing に置き換えることでより仕様に沿った構文解析をエレガントに行なえるようにすることです。

19 Debian での Linux カーネルとの付き合い方

岩松 信洋

19.1 はじめに

ここ数年で Debian の Linux カーネル開発体制や Linux カーネルに関するツールの使い方が変わってきました。Debian JP の ML でもたまに Linux カーネルに関してハマっている方おられるようです。昔と違って最近のユーザは Debian から提供されているカーネルを使う人が多いようで、情報が古くなっており情報がまとまっていません。今回は、Debian Linux カーネル開発の背景と、今時のカーネルコンパイル方法について簡単にまとめました。

19.2 Debian Linux カーネルの開発状況

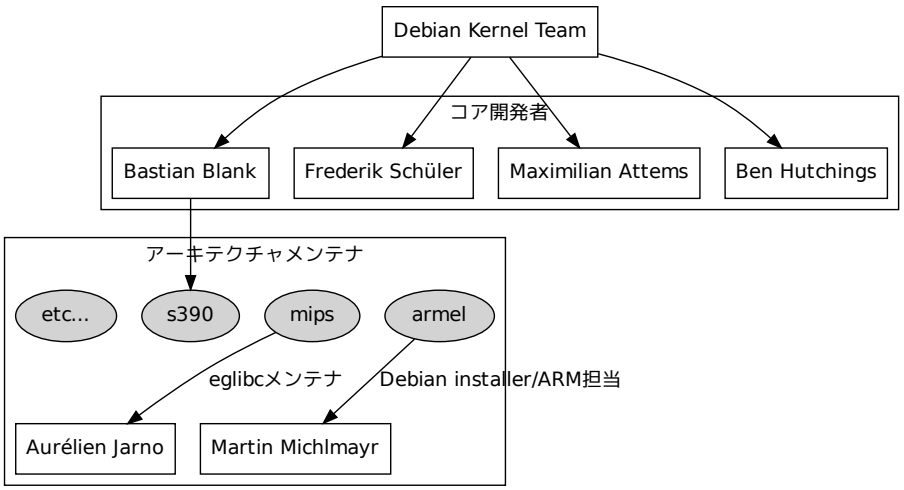


図 42 Debian Linux カーネルチーム構成図

Debian の Linux カーネルは Debian Kernel Team によって開発およびメンテナンスされています。もちろん Debian では Linux カーネルも Debian パッケージとして提供されており、容易に利用可能です。チームコアメンバは Bastian Blank, Frederik Schul, Maximilian Attems, Ben Hutchings の 4 名で、彼らがが中心になってメンテナンスしています。各々はもちろんカーネル開発者です。彼らだけでは全アーキテクチャの面倒を見きれないので、各アーキテクチャメンテナとともにカーネルパッケージをメンテナンスしています。アーキテクチャメンテナは Buildd メンテナや、

Debian-instller チーム、 eglibc メンテナなど、カーネルに関係するパッケージをメンテナンスしている約 20 名の開発者によって構成されます。

19.3 Debian での Linux カーネル開発プロセス

去年の 6 月頃までは Debian でベースとする Linux カーネルバージョンは決まっていましたが、パッケージのリリースサイクルはあまり決まっていませんでした。去年の Debconf では Linux カーネルの stable リリースに合わせて開発を行うことが決まり、パッケージのバージョニングと開発/メンテナンススタイルもリリースサイクルに合わせて変更されました。開発プロセスについてみてみましょう。

19.3.1 Debian カーネル用語

Debian カーネルについて説明するにあたり、用語を簡単に説明します。ディストリビューションではカーネルという言葉はいろんな意味を持ちます。よく使われる用語をまとめてみました。

- Debian カーネル
  Debian からパッケージとしてリリースされているパッケージ。
- LTS
  Long-term Support の略。現在は 2.6.32 が対象。以前は 2.6.27。
- stable カーネル
  `The Linux Kernel Archives` からダウンロードできるカーネル。現在（ 2010 年 5 月）、 2.6.33.3、2.6.32.12、 2.6.31.13、 2.6.30.10、 2.6.27.46 の 5 つが存在します。
- Linus/HEAD
  Linus 氏がメンテナンスする git リポジトリの HEAD。 HEAD はその時の最新を意味します。

19.3.2 カーネルパッケージのバージョン関係

開発体制プロセスの変更により、カーネルパッケージのバージョンやアップロードのタイミングが変わりまました。2010 年 5 月現在、 Linux の LTS サポートカーネルバージョンは 2.6.32、最新版は 2.6.32.12 です。このカーネルをベースに Debian パッケージにした場合、パッケージバージョンは linux-2.6_2.6.32-12 になります。 stable リリースバージョンを Debian バージョンに置き換えており、 Debian バージョン = Linux カーネルの stable リリースバージョンになります。これにより新しい stable リリースが出ない限り、 Debian パッケージもアップロードされません。

19.3.3 パッチのバックポート

Linus カーネルからのバックポート（ 例えば、 linux-2.6.34-rc7 で取り込まれたパッチを linux-2.6.32 に取り入れてもらうなど）は、 Debian パッケージに直接取り込まれることはなく、 stable カーネルからのみ受け付けます。 stable カーネルで採用されない限りは Debian でも使えないということです。取り込んでもらうには `stable@kernel.org` にメールし、 stable カーネルに取り込んでもらうように交渉する必要があります。

19.3.4 バグレポートとパッチ

バグがあった場合には、 Debian の BTS を利用できます。パッチが用意できる場合には添付しましょう。メンテナが Upstream(stabel カーネル、場合によっては Linus/HEAD) に転送してくれます。

Debian Kernel Team で作成されたパッチは積極的に Linus カーネルに取り込まれるように働きかけています。これらが、 Liuns/HEAD または stable カーネルに取り込まれない場合、 Debian specific パッチとして管理されます。例えば、ドライバの non-free ファームウェアのパッチなどはまだ全て取り込まれておらず、一部は Debian specifc パッチとして残っています。

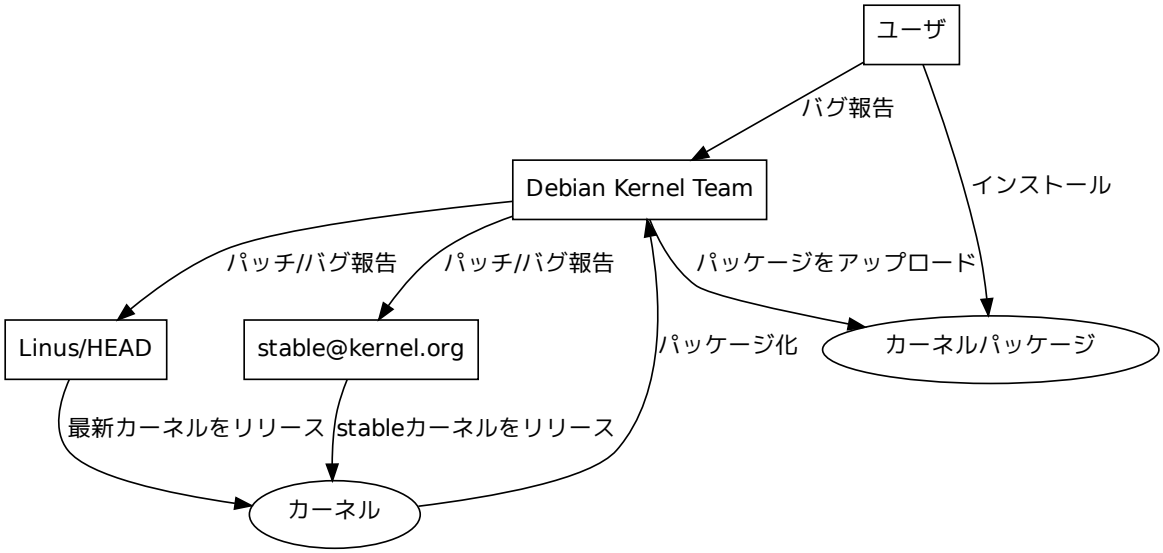


図 43　Debian Linux カーネル開発プロセス

19.4　Debian Kernel Team によってメンテナンスされている主要なパッケージ

Debian Kernel Team では Linux カーネルに関するいくつかのパッケージをメンテナンスしています。ここでは、主要なパッケージと関係について説明します。

19.4.1　linux-2.6 パッケージ

カーネルコンフィグ

linux-2.6 ソースパッケージで持っているカーネルコンフィグは基本 config , アーキテクチャ用 config , flavour 用 config 3 つに分けられており、 debian/config ファイルに格納されています。これらのファイルはバイナリパッケージビルド時に一つにまとめられ、まとめられた config を使ってカーネルコンフィグが行われます。ファイルで設定されているコンフィグは基本的に重複しません。しかし、組み込みで使われるボードでは、ドライバモジュールを組み込みにしないと動作しないものもあるため、 flavour 用の config ファイルによってオーバライドされます。よって、各ファイルの優先順位としては基本 config <アーキテクチャ用 config <flavour 用 config となります。また、コンフィグを各ファイルに自動的で分割するプログラムは存在せず、アーキテクチャメンテナがちまちまファイルを修正し、アップデートします。

19.4.2　linux-latest-2.6 パッケージ

`linux-latest-2.6` パッケージは Debian カーネルの最新 ABI を追従するためのメタパッケージを提供します。例えば、 amd64 アーキテクチャ向けの基本 Linux カーネルイメージパッケージは `linux-image-2.6.32-5-amd64` になりますが、 `linux-latest-2.6` ソースパッケージからビルドされる `linux-image-2.6-amd64` は `linux-image-2.6.32-5-amd64` に依存します。パッケージ名に ABI のバージョン(上の例だと 5）を含めてるので、 ABI が変更された場合にカーネルアップデートがされません。新規にパッケージをインストールする必要があるわけです。例えば、 `linux-image-2.6.32-4-amd64` と `linux-image-2.6.32-5-amd64` では ABI が異なるので別パッケージ扱いになります。このときに、 `linux-image-2.6-amd64` をインストールしておくと、 ABI が 4 から 5 にアップーデートされた場合に、 `linux-image-2.6-amd64` もアップデートされ、 `linux-image-2.6.32-5-amd64` が自動的にインストール

Debian Kernel Team がメンテナンスしているパッケージの一つに linux-2.6 があります。これは Linux カーネルの主要なパッケージであり、このパッケージから各アーキテクチャ向けのカーネル、ヘッダファイル、 libc 向けヘッダファイル、ドキュメント等のパッケージが生成されます。

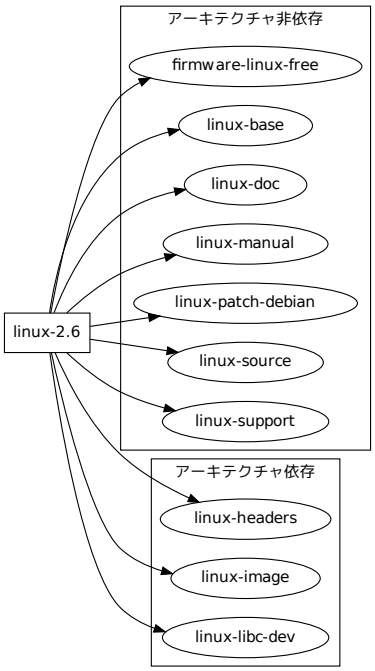


図 44　linux-2.6 パッケージから生成されるパッケージ

されます。

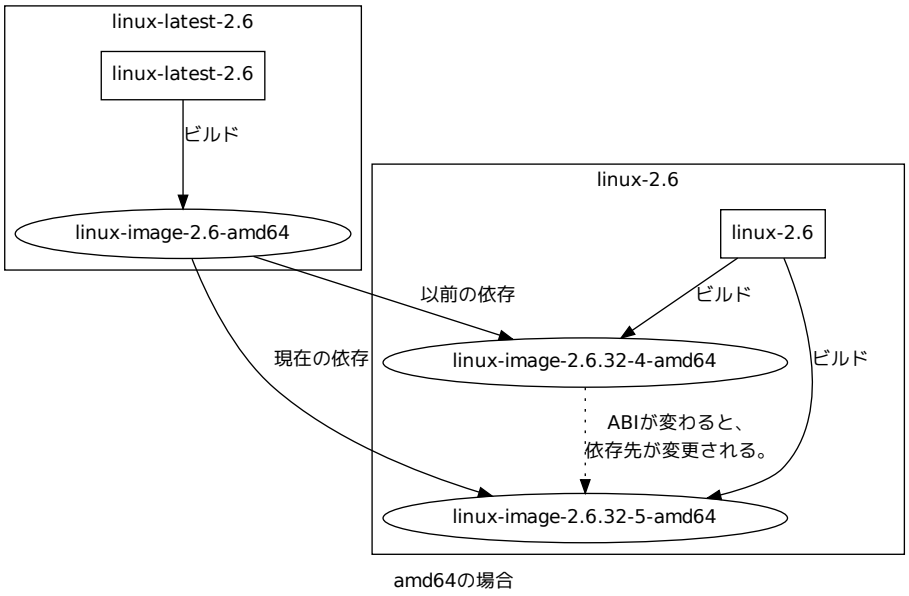


図 45　linux-2.6 と linux-latest-2.6 の関係

ABI のチェック

ABI のチェックは linux-2.6 パッケージ内にある `debian/bin/buildcheck.py` を使って、カーネルパッケージビルド時に実行されます。カーネルパッケージメンテナは手元でビルドしたときに、ABI のアップデートをチェックし、ABI を更新して Debian にアップロードします。といっても、大幅な変更、例えば sycall が追加/変更されるなどの変更がない場合には ABI を変更しない場合もあるようです。

以下に ABI のチェックを行なった例を示します。新しい symbol **dev_attr_usbip_debug が追加されたことがわかります。**

```
--省略--
make[3]: Leaving directory `/home/mattems/src/linux-2.6-2.6.32/debian/build/build_amd64_none_amd64'
python debian/bin/buildcheck.py debian/build/build_amd64_none_amd64 amd64 none amd64
ABI has changed!  Refusing to continue.

Added symbols:
dev_attr_usbip_debug    module: drivers/staging/usbip/usbip_common_mod, version: 0x79bd9084, export: EXPORT_SYMBOL_GPL
getboottime             module: vmlinux, version: 0x0619ca8a, export: EXPORT_SYMBOL_GPL
monotonic_to_bootbased  module: vmlinux, version: 0xdb274e52, export: EXPORT_SYMBOL_GPL
--省略--
```

19.4.3 linux-kbuild-2.6

`linux-kbuild-2.6` **はカーネルドライバ構築をサポートするためのスクリプトを持っています。よって、linux-headers パッケージに依存しています。このパッケージのソースコードは linux-2.6 パッケージから作られず、別途の stable カーネルのソースコードから kbuild を行うために必要な部分を抽出して作られます。これは、kbuild システムが stable リリース毎に更新する必要がないためです。**

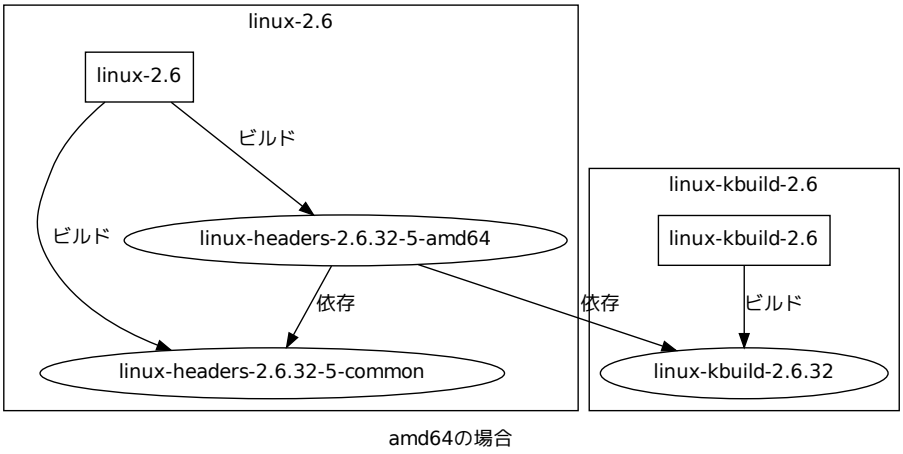


図 46　linux-2.6 と linux-kbuild-2.6 の関係

19.4.4 まとめ

- **Debian のカーネルメンテナンスはチーム制。**
- **カーネルに関係するパッケージメンテナやアーキテクチャメンテナによってメンテナンスされている。**
- **パッケージのアップデートは stabel リリースベース。**
- **更新を用意にするためにメタパッケージを使っている。**
- **ABI のチェックなどもしてけっこう真面目。**

19.5 今時の Debian カーネルのビルド方法

大抵のユーザは Debian で提供されているバイナリパッケージを使います。しかし、たまにビルドしたい人がいるわけです。理由としては以下のものが考えられます。

- 何か不具合修正するためのパッチを適用したい。
- オレオレパッチを適用したカーネルを使いたい。
- プリエンプションモデルが気に入らない。
  CONFIG_PREEMPT_XXX の変更
- Timer frequency を変更したい。
  CONFIG_HZ_XXX の変更
- 毎朝、自分のマシンで使うカーネルをビルドしないと気が済まない。

このような事から日頃からカーネルのビルドを行って置くことが重要です。しかし、Debian ではいくつかのカーネル構築方法があります。これらを一つずつみてみましょう。

19.5.1 Debian オフィシャルカーネルをリビルドする

まずは基本の Debian カーネルのリビルド方法を説明します。ソースパッケージをダウンロードし、debuild を実行すれば linux カーネルパッケージがビルドされますが、この方法では自分の必要のない flavour までビルドします。ここでは、指定した flavour のみをビルドする方法を説明します。

1. Linux-2.6 ソースコードをダウンロードする。
   まず、linux-2.6 ソースパッケージをダウンロードします。ダウンロードできたら、展開されたディレクトリに移動します。

```
$ apt-get source linux-2.6
$ cd linux-2.6-2.6.32
```

2. linux-2.6 パッケージのビルドに必要なパッケージをインストールする。
   パッケージのビルドに必要なパッケージをインストールするには build-dep オプションを使います。

```
$ sudo apt-get build-dep linux-2.6
```

3. Debian カーネル向けのパッチを適用する。

```
$ make -f debian/rules clean
$ make -f debian/rules source-all
```

   debian/rules source-all では、全てのアーキテクチャ向けにパッチを適用してしまうので、特定のアーキテクチャのパッチを適用したい場合には以下のように実行します。

```
$ make -f debian/rules.gen source_amd64
```

4. 利用したい flavour で初期化する。
   amd64 アーキテクチャの amd64 flavour で初期化したい場合には以下のように実行します。

```
$ fakeroot make -f debian/rules.gen setup_amd64_none_amd64
```

5. カーネルコンフィグを変更する。
   カーネルコンフィグを変更したい場合には、debian/build/build_amd64_none_amd64 ディレクトリ移動して、カーネルコンフィグを行います。コンフィグ終了後は元のディレクトリに戻る必要があります。

```
$ cd debian/build/build_amd64_none_amd64
$ make menuconfig
$ cd ../../..
```

6. パッケージをビルドする。

   debuild / dpkg-buildpackage コマンドは利用せず、 debian/rules のターゲットを指定してパッケージをビルドします。

```
$ fakeroot make -f debian/rules.gen binary-arch_amd64_none_amd64
```

19.5.2 Debian カーネルにパッチを適用して利用する。

よく行うと思われるのが、 Debian カーネルをベースに自分が作ったパッチを当てて管理するというものです。これを行うには、 Debian のカーネルパッチ機構を知る必要があります。

Debian カーネルパッチ機構

基本的に通常のパッケージのパッチシステムと変わりません。 debian/patches にパッチが格納されています。四つのディレクトリがあり、さらにアーキテクチャ毎にパッチが分かれています。

- bugfix
  重要なバグ修正用パッチを格納します。
- debian
  Debian 専用パッチを格納します。
- features
  まだ upstream にマージされていないパッチを格納します。
- series
  パッチを管理するファイルを格納しているディレクトリ。 Debian バージョン毎にファイルがあります。

自分のパッチを適用したカーネルをビルドする方法

1. カーネルソースコードを取得する。
   展開後に、ディレクトリに移動します。

```
$ apt-get source linux-2.6
$ cd linux-2.6-2.6.32
```

2. チェンジログを更新する。
   dch コマンドを使って、新しい Debian バージョンで Changelog を作成します。このときに、-D オプションを使って、ディストリビューション名に UNRELEASED を指定しない場合、カーネルパッケージビルドのチェックに引っかかります。以下の例では、サフィックスにローカルバージョンとして +text を指定し、Changelog ファイルを更新しています。この場合、 Liux カーネルパッケージのバージョンが 2.6.32-12 の場合、 2.6.32-12+test1 というバージョンになります。

```
$ dch --local +test -D UNRELEASED
```

3. パッチをディレクトリにコピーする。

   パッチを debian/patches ディレクトリ以下にコピーします。

```
$ cp ~/oreore.patch debian/patches/bugfix/
```

4. コピーしたパッチを有効にする。
   コピーしたパッチを有効にするには、 debian/patches/series/ディレクトリにパッチを適用したい Debian バージョンのファイルを作成し、パッチのパスを指定します。

```
$ echo ‘‘+ bugfix/oreore.patch’’ >> debian/patches/series/12+test1
```

5. ./debian/bin/gencontrol.py を実行する。./debian/bin/gencontrol.py を実行し、ビルド用のスクリプトや設定ファイルを新しい Debian バージョン向けに更新します。

```
$ ./debian/bin/gencontrol.py
```

6. 一度初期化し、パッチを適用する。

```
$ make -f debian/rules clean
$ make -f debian/rules.gen source_amd64
```

7. パッケージをビルドする。
   パッチが適用できたら以下のコマンドを実行し、パッケージをビルドします。

```
$ fakeroot make -f debian/rules.gen binary-arch_amd64_none_amd64
```

   エラーがなければ、パッチが有効になったカーネルパッケージがビルドされます。

19.5.3 Debian カーネルの Linux カーネルソースコード (linux-source-2.6.XX パッケージ) から再ビルドする。

Debian カーネルを再ビルドする方法はもうひとつあり、 linux-source-2.6.XX パッケージを利用する方法です。このパッケージにはアップストリームのソースコードのみを提供しています。このパッケージからカーネルパッケージをビルドする方法を説明します。

1. Debian が提供しているカーネルのビルドに必要なパッケージをインストールする。

```
$ sudo apt-get build-dep linux-source-2.6.32
```

2. Debian のカーネルソースをインストールする。

```
$ sudo apt-get install linux-source-2.6.32
```

3. make-kpkg コマンドを使ってカーネルパッケージをビルドする。

```
$ fakeroot make-kpkg --revision=test00debian kernel_image kernel_headers
```

しかし、これでは Debian パッケージにはパッチが適用されていない状態です。 linux-patch-debian-XXXX パッケージをインストールし、パッチを適用する必要があります。-a でアーキテクチャ、-f で flavour を指定します。

```
$ sudo apt-get install linux-patch-debian-2.6.32
$ /usr/src/kernel-patches/all/2.6.32/apply/debian -a x86_64 -f xen
```

19.5.4 リリースされたカーネルを debian パッケージにする

Linus や stable チームによってリリースされた Linux カーネルを Debian パッケージを作成するには kernel-package パッケージを使うのが Debian 流です。

1. kernel-package パッケージと fakeroot パッケージをインストールします。

```
$ sudo apt-get install kernel-package fakeroot
```

2. ソースをダウンロードし、展開します。
3. カーネルコンフィグを実行します。

```
$ make menuconfig
```

4. make-kpkg コマンドを使ってカーネルパッケージを構築する。
   make-kpkg コマンドにはいくつかのオプションがありますが、よく利用するオプションについて説明します。
   - kernel_image
     カーネルイメージパッケージビルドを指定します。
   - kernel_headers
     カーネルヘッダビルドを指定します。
   - –revision
     リビジョンを指定します。これは Debian バージョンに付加されます。
   - –append_to_version
     カーネルバージョンを追加します。これはパッケージ名に付加されます。
   - –added_modules
     Debian パッケージになっているカーネルモジュールをビルドします。
   - –added_patches
     Debian パッケージなっているカーネルパッチを有効にしてビルドします。
   - –initrd
     initrd イメージをビルドする際に必要です。 initrd イメージはパッケージインストール時に作成するように仕様が変わっています。

   例えば、リビジョンを test12345、バージョンに append67890 指定し、カーネルパッケージとカーネルヘッダパッケージをビルドする場合には以下のように実行します。リビジョンの前に.(ピリオド) をつけているのは、2.6.33.3 の場合には 2.6.33.3.append67890 となるようにするためです。

```
$ fakeroot make-kpkg --revision=.test12345 --append-to-version=append67890 kernel_image
```

   作成されるパッケージ名は以下のようになり、--append-to-version と--revision は以下のように配置されます。

```
linux-image-(kernel-version)(--append-to-version)_(--revision)_(architecture).deb
```

5. ビルドが終わるとパッケージがビルドされているので、インストールします。

```
$ sudo dpkg -i ../linux-image-2.6.33.3.append67890_testrev12345_amd64.deb
```

19.6 git/HEAD を Debian パッケージにする

今時はカーネルがリリースされる度にカーネルのソースコードをダウンロードするのではなく、常に git リポジトリを更新し、 git リポジトリからビルドするのが通でしょう。たぶん。 kernel-package パッケージでは、 git リポジトリからビルドできる機能があるのでこれを利用すると容易にカーネルパッケージを作成することができます。

1. Linux git リポジトリをコピーする。
   Linux カーネルの git リポジトリがない場合には git clone コマンドで取得します。

```
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git
```

   linux-2.6 ディレクトリができるので移動します。

```
cd linux-2.6
```

2. リポジトリのアップデートを行う。
   普段から git リポジトリを使っていじっている人はアップデートしましょう。

```
$ git pull
```

3. make-kpkg clean を実行する。
   make-kpkg clean を実行し、一度初期化をします。

```
$ make-kpkg clean
```

4. カーネルパッケージをビルドする。
   make-kpkg を使ってカーネルパッケージをビルドします。
   ビルドすると、 Makefile からバージョンを抽出し、パッケージバージョンをつけてくれます。 git log --pretty=format:%h -1 はチェックアウトしている HEAD の短縮されたハッシュ値を取得し、--revision オプションに渡しています。これにより、どのコミットから作成したカーネルイメージなのかわかるようになります。

```
$ fakeroot make-kpkg --revision=1+‘git log --pretty=format:\%h -1‘ --initrd kernel_image
-- 省略 --
$ ls ../
linux-2.6  linux-image-2.6.34-rc7_1+be83567_amd64.deb
```

19.7 ドライバモジュールの取扱い

Debian カーネルパッケージでは、多くのカーネルドライバモジュールが用意されています。最近では module-init-tools の仕様変更があり、設定ファイルのサフィックスが変更されています。ドライバモジュールの取り扱いについて一度簡単におさらいしてみましょう。

19.7.1 ロードしたいカーネルを指定する。

起動時に自動的にモジュールをロードする設定は、/etc/modprobe.d/に新しいファイルを作成して、そこに記述するします。

たとえば、 eth0 として eepro100 をロードさせたい場合、/etc/modprobe.d/local.conf ファイルを用意して、設定を記述します。

```
echo ’alias eth0 e100’ >> /etc/modprobe.d/local.conf
```

とします。

19.7.2 カーネルモジュールパラメータの設定

パラメータの設定も/etc/modprobe.d/local.conf に記述します。カーネルモジュールのパラメータを調べるには、modinfo コマンドを利用します。

```
$ modinfo e100
filename:       /lib/modules/2.6.31-1-686/kernel/drivers/net/e100.ko
firmware:       e100/d102e_ucode.bin
firmware:       e100/d101s_ucode.bin
firmware:       e100/d101m_ucode.bin
version:        3.5.24-k2-NAPI
--省略--
vermagic:       2.6.31-1-686 SMP mod_unload modversions 686
parm:           debug:Debug level (0=none,...,16=all) (int)
parm:           eeprom_bad_csum_allow:Allow bad eeprom checksums (int)
parm:           use_io:Force use of i/o access mode (int)
```

例えば、 e100 ドライバの debug レベルを設定する場合には以下のように記述します。

```
$ echo 'options e100 debug=16' >> /etc/modprobe.d/local.conf
```

設定したら、 OS の再起動を行うか、カーネルモジュールを再読込します。

19.8 よくある質問

19.8.1 最新カーネル向けパッケージを lenny 上で作れません

kernel-package パッケージが古いので lenny ではビルドできません。 testing/unstable にある kenrel-package パッケージを lenny にインストールすることによって対応できます。 kernel-package に依存しているパッケージも lenny 内から持ってくることができるので、特にシステムが壊れるということはないと思われます。

19.8.2 initrd が作られません。

grub のメニューを変更して、 initrd を使わないようにしましょう。というのは半分冗談で、 kenrel-pakcage 12.012 以降から initrd を作らない仕様に変更されました。 make-kpkg コマンドを使って initrd を含めたカーネルイメージを作成するには、以下を実行する必要があります。

```
$ sudo mkdir -p /etc/kernel/postinst.d/
$ sudo cp
 /usr/share/doc/kernel-package/examples/etc/kernel/postinst.d/initramfs \
 /etc/kernel/postinst.d/
$ fakeroot make-kpkg --revision=1 --initrd kernel_image
```

実行した後に再度カーネルパッケージを作ると、インストール時に initrd イメージを構築します。

19.8.3 -j オプションを使ってカーネルパッケージをビルドしたいのですが

make-kpkg コマンドの DEBIAN_KERNEL_JOBS 変数を使いましょう。例えば、-j8 相当は以下のように実行します。

```
$ make-kpkg --revision=test00 kernel_image kernel_headers DEBIAN_KERNEL_JOBS=8
```

19.8.4 最新カーネル向けの linux-kbuild-2.6 を作りたいのですが

最新リリースカーネルの Debian パッケージは experimental ディストリビューションにアップロードされます。2010 年 05 月の時点では、バージョン 2.6.33 で、 linux-2.6_2.6.33-1 experimental.5 としてアップロードされています。カーネルはアップロードされているのですが、 linux-kbuild-2.6 パッケージが最新カーネル向けにアップロードされない事があるので、使いたいユーザは自分で用意する必要がある場合があります。作り方は http://wiki.debian.org/HowToRebuildAnOfficialDebianKernelPackage#Thestoryoflinux-kbuild-2.6 を参照してください。ちなみに 2.6.33 以降のパッケージを作成する場合には、 バグ (#573176) があります。パッチを当てて、ビルドしましょう。

19.8.5 最新のカーネルを使いたいのだけど、パッケージにするのがめんどいです。

http://kernel-archive.buildserver.net で提供されていますが、現在サーバダウン中です。

20 dpkg ソース形式 “3.0 (quilt)”

吉野与志仁

20.1 はじめに

普段 Debian を使っている皆さんはご存知とは思いますが、近々デフォルトになる予定で squeeze の release goal である Debian ソースパッケージのフォーマット “3.0 (quilt)” の復習をしたいと思います。

20.2 “1.0”

まずは “3.0 (quilt)” の前に、いままで一般に使われてきたフォーマット (“1.0”) を簡単にまとめます。
1.0 では、ソースパッケージは以下の 3 ファイルで構成されます。

- *packagename-upstreamversion*.orig.tar.gz
- *packagename-debianversion*.diff.gz
- *packagename-debianversion*.dsc

なお、正確には 1.0 は 2 種類あって、上の通常のパッケージのほかに “Debian native な” パッケージがあります。Debian native パッケージは次の 2 ファイルで構成されます。

- *packagename-version*.tar.gz
- *packagename-version*.dsc

ここで、*.orig.tar.gz には、通常上流の元のソースツリーが含まれます。*.diff.gz には、ソースパッケージからパッケージなどをビルドするのに必要なスクリプトなどが入った debian/ ディレクトリや、上流のソースに対するパッケージメンテナの変更が含まれます。

20.3 しかし

と説明してきましたが、このファイル構成には

1. アーカイブの圧縮形式に gzip しか使えない
2. 複数のアーカイブで構成される上流のソースがそのまま扱えない
3. メンテナが当てたソースへのパッチが全部つながってしまっている
4. debian/ 以下にバイナリファイルが直接置けない

などの問題点があります。
そこで、さまざまな方法が検討されました。

1 は、これにより上流が bz2 で配布していても gz に圧縮し直さなければならなかったりしていました。*.orig.tar.gz の中身が上流のアーカイブの実体である、といった方法なども(ちょっと無駄ですが...) 使われてきました。この方法はビルド時にその tarball を展開して作業します。 cdbs にはこの方法へのサポートもあります。

2 は 1 と同様の方法で複数の tarball が入った*.orig.tar.gz を用意したりしていました。

3 は、当たっているパッチのそれぞれがどんな意図で行われたのかがわからない、ということ、また、 debian/以下のファイルも上流ソースへのパッチも一緒くたになってしまっていること、が問題でした。そこで、まとまった意味のある単位に分割されたパッチをまず用意しておき、それらを debian/patches/ 下に配置し、その細かいパッチをビルド時に当てる/外すフレームワーク (patch system) が利用されています。これには dpatch や quilt などがあります。なお、この細かいパッチのそれぞれには、先頭にパッチの意図を説明する文章を記述することが推奨されています (http://dep.debian.net/deps/dep3/)。

4 は、バイナリファイルの diff を取ろうとしても普通の patch ではできないことが原因なので、 uuencode などでテキストに落として patch を取る、といった手法が用いられてきました。

20.4 そこで

このような問題を解決するために新たなソースパッケージのフォーマットも検討されました。それが “3.0 (quilt)” フォーマット(と “3.0 (native)” フォーマット) です。

3.0 (quilt) は次の 3 つ以上のファイルで構成されます。

- *packagename-upstreamversion*.orig.tar.*ext*
- *packagename-upstreamversion*.orig-*component*.tar.*ext* (任意)
- *packagename-debianversion*.debian.tar.*ext*
- *packagename-debianversion*.dsc

なお、 1.0 にあった Debian native パッケージに相当する 3.0 (native) は次の 2 ファイルで構成されます。

- *packagename-version*.tar.*ext*
- *packagename-version*.dsc

ここで、まず、 tar の拡張子部分 *ext* に gz のほか、 bz2, lzma, xz が利用できるようになりました。これにより問題 1 が解決されました(3.0 (native) における主な変更点はこれです) 。

また、 *component* の部分を適当に変えることにより複数の tarball をきちんと扱えるようになりました。これが問題 2 を解決します。

次に、 debian/下のファイルはすべて *.debian.tar.gz に入れることになりました。これですべてが混ざった状態はなくなりました。さらに、 debian/patches/ 下のパッチが、パッチシステム quilt と基本的に同じ方法で dpkg-source(1) によって “ソースパッケージの展開時に” 自動的に当たるようになりました。これにより、ビルド時にパッチを当てるように debian/rules ファイルを記述する必要はなくなりましたし、 debian/control ファイルに Build-Depends: quilt などと書く必要もなくなりました。これらによって問題 3 は解決されました。

問題 4 については、 debian/下のファイルを diff として保持することはもはやなくなったので解決し、*.debian.tar.*ext* に直にバイナリファイルを配置できます。

20.5 最近の動向

この 3.0 (quilt) フォーマットは、 Debian のアーカイブが巨大化していっているため、 gzip より lzma (lenny 当時で、現在なら xz) を使って圧縮すればサイズを抑えられるといった理由で、 lenny 以降でより推進されるようになりました。 Debian にあるソースパッケージすべてが 3.0 (quilt) 化可能になったら(潰すべき minor/wishlist バグが http://bugs.debian.org/cgi-bin/pkgreport.cgi?users=hertzog@debian.org;tag=3.0-quilt-by-default にあります) 、 dpkg はデフォルトで 3.0 (quilt) フォーマットでソースパッケージをビ

ルドするように変更される、とのことです。というわけでこれから作るパッケージは 3.0 化しましょう。

20.6 例 1： gzip パッケージ

具体例として、まず gzip ソースパッケージ (1.3.12-9) を 3.0 (quilt) 化してみました。このソースパッケージは直に patch を当てた形式を取っていました。 dpkg-source(1) による展開時のメッセージで分かります:

```
$ apt-get source gzip
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています
状態情報を読み取っています... 完了
479kB のソースアーカイブを取得する必要があります。
取得:1 http://ftp.jp.debian.org testing/main gzip 1.3.12-9 (dsc) [1,647B]
取得:2 http://ftp.jp.debian.org testing/main gzip 1.3.12-9 (tar) [462kB]
取得:3 http://ftp.jp.debian.org testing/main gzip 1.3.12-9 (diff) [15.7kB]
479kB を 1s で取得しました (250kB/s)
dpkg-source: info: extracting gzip in gzip-1.3.12
dpkg-source: info: unpacking gzip_1.3.12.orig.tar.gz
dpkg-source: info: applying gzip_1.3.12-9.diff.gz
dpkg-source: info: upstream files that have been modified:
 gzip-1.3.12/.gbp.conf
 gzip-1.3.12/deflate.c
 gzip-1.3.12/gzip.1

(snip)
```

20.6.1 バージョンの切り換え、ビルド

現在のデフォルトは 1.0 なので、 3.0 (quilt) と明示してからソースパッケージを作ります。

```
$ cd gzip-1.3.12/
$ mkdir -p debian/source
$ echo '3.0 (quilt)' > debian/source/format
$ debuild -S -us -uc
 dpkg-buildpackage -rfakeroot -d -us -uc -S
dpkg-buildpackage: set CFLAGS to default value: -g -O2

(snip)

 dpkg-source -b gzip-1.3.12
dpkg-source: info: using source format `3.0 (quilt)'
dpkg-source: info: building gzip using existing ./gzip_1.3.12.orig.tar.gz
dpkg-source: info: local changes stored in gzip-1.3.12/debian/patches/debian-changes-1.3.12-9, the modified files are:
 gzip-1.3.12/.gbp.conf
 gzip-1.3.12/deflate.c

(snip)

dpkg-source: info: building gzip in gzip_1.3.12-9.debian.tar.gz
dpkg-source: info: building gzip in gzip_1.3.12-9.dsc
 dpkg-genchanges -S >../gzip_1.3.12-9_source.changes

(snip)
```

と、とりあえず昔の diff.gz (の上流ソースへのパッチ部分) に相当する 1 つのパッチを自動で出力してくれます。この大きなパッチそのままではこのフォーマットにした意味がほとんどないので分けましょう。また、パッチの説明文テンプレートも debian/changelog を基にして付けてくれるので、修正して正しい説明にしましょう。(今回は共に省略)

バイナリパッケージをビルドしてみます。

```
$ debuild -b -us -uc
 dpkg-buildpackage -rfakeroot -D -us -uc -b
dpkg-buildpackage: set CFLAGS to default value: -g -O2
dpkg-buildpackage: set CPPFLAGS to default value:
dpkg-buildpackage: set LDFLAGS to default value:
dpkg-buildpackage: set FFLAGS to default value: -g -O2
dpkg-buildpackage: set CXXFLAGS to default value: -g -O2
dpkg-buildpackage: source package gzip
dpkg-buildpackage: source version 1.3.12-9
dpkg-buildpackage: source changed by Bdale Garbee <bdale@gag.com>
dpkg-buildpackage: host architecture amd64
dpkg-checkbuilddeps: Unmet build dependencies: mingw32
dpkg-buildpackage: warning: Build dependencies/conflicts unsatisfied; aborting.
dpkg-buildpackage: warning: (Use -d flag to override.)
debuild: fatal error at line 1330:
dpkg-buildpackage -rfakeroot -D -us -uc -b failed
```

Build-Depends を満たして、ビルドします。

```
$ mk-build-deps
dh_testdir

(snip)

dpkg-deb: ‘../gzip-build-deps_1.0_all.deb’ にパッケージ ‘gzip-build-deps’ を構築しています。

The package has been created.
Attention, the package has been created in the current directory,
not in ".." as indicated by the message above!
$ sudo dpkg -i gzip-build-deps_1.0_all.deb
未選択パッケージ gzip-build-deps を選択しています。
(データベースを読み込んでいます ... 現在 203017 個のファイルとディレクトリがインストールされています。)
(gzip-build-deps_1.0_all.deb から) gzip-build-deps を展開しています...
dpkg: 依存関係の問題により gzip-build-deps の設定ができません:
 gzip-build-deps は以下に依存 (depends) します: mingw32 ... しかし:
  パッケージ mingw32 はまだインストールされていません。
dpkg: gzip-build-deps の処理中にエラーが発生しました (--install):
 依存関係の問題 - 設定を見送ります
以下のパッケージの処理中にエラーが発生しました:
 gzip-build-deps
$ sudo aptitude install gzip-build-deps
パッケージリストを読み込んでいます... 完了

(snip)

タスクの記述を読み込んでいます... 完了

現在の状態: 依存関係破損が 0 個 [-1]。
$ debuild -b -us -uc
 dpkg-buildpackage -rfakeroot -D -us -uc -b
dpkg-buildpackage: set CFLAGS to default value: -g -O2

(snip)

dpkg-deb: ‘../gzip_1.3.12-9_amd64.deb’ にパッケージ ‘gzip’ を構築しています。
 dpkg-genchanges -b >../gzip_1.3.12-9_amd64.changes
dpkg-genchanges: binary-only upload - not including any source code
dpkg-buildpackage: binary only upload (no source included)
Now running lintian...
W: gzip: missing-dependency-on-install-info
Finished running lintian.
```

20.7 例 2： bash-completion パッケージ

次に、パッチシステムとして quilt を使っていた bash-completion パッケージを 3.0 (quilt) 化してみました。

20.7.1 まずビルド

パッチシステムでは、ビルド前はパッチは当たっていないので、自動でツリーにパッチが当てられてソースパッケージがビルドされます。

```
$ apt-get source bash-completion
パッケージリストを読み込んでいます... 完了

(snip)

$ cd bash-completion-1.1/
$ mkdir -p debian/source
$ echo ’3.0 (quilt)’ > debian/source/format
$ debuild -S -us -uc
 dpkg-buildpackage -rfakeroot -d -us -uc -S
dpkg-buildpackage: set CFLAGS to default value: -g -O2

(snip)

 fakeroot debian/rules clean
dh --with quilt clean
   dh_testdir
   dh_auto_clean
   dh_quilt_unpatch
適用されているパッチはありません
   dh_clean
 dpkg-source -b bash-completion-1.1
dpkg-source: info: using source format ‘3.0 (quilt)’
dpkg-source: warning: patches have not been applied, applying them now (use --no-preparation to override)
dpkg-source: info: applying 01-fix_550943.patch
dpkg-source: info: applying 02-fix_552109.patch
dpkg-source: info: applying 03-fix_552631.patch
dpkg-source: info: building bash-completion using existing ./bash-completion_1.1.orig.tar.gz
dpkg-source: info: building bash-completion in bash-completion_1.1-3.debian.tar.gz
dpkg-source: info: building bash-completion in bash-completion_1.1-3.dsc
 dpkg-genchanges -S >../bash-completion_1.1-3_source.changes

(snip)
```

今回は問題ありませんでしたが、 3.0 (quilt) は quilt とは微妙に異なり、すべて patch -p1 として扱われるので path を適当に調整する必要があるかもしれません。

20.7.2 quilt 周りの変更

パッチは展開時に当たるので、 debian/rules 内でパッチを当てている部分はもう要りません。このソースパッケージは dh(1) を使用していたので簡単でした。

```
--- debian/rules        2010-03-18 10:50:30.000000000 +0900
+++ debian/rules.new        2010-03-18 10:54:53.000000000 +0900
@@ -21,11 +21,11 @@

 build: build-stamp
 build-stamp:
-        dh --with quilt build
+        dh build
         touch $@

 clean:
-        dh --with quilt $@
+        dh $@

 install: install-stamp
 install-stamp: build
```

quilt には通常は Build-Depends しません。

```
--- debian/control        2010-03-18 10:50:30.000000000 +0900
+++ debian/control.new        2010-03-18 10:57:00.000000000 +0900
@@ -3,7 +3,7 @@
 Priority: standard
 Maintainer: Bash Completion Maintainers <bash-completion-devel@lists.alioth.debian.org>
 Uploaders: David Paleino <dapal@debian.org>
-Build-Depends: debhelper (>= 7.0.50), quilt (>= 0.46-7~)
+Build-Depends: debhelper (>= 7.0.50)
 Build-Depends-Indep: perl
 Standards-Version: 3.8.3
 Vcs-Git: git://git.debian.org/git/bash-completion/debian.git
```

20.7.3 もう一回

今度は初めから 3.0 (quilt) です。

```
$ debuild -S -us -uc
 dpkg-buildpackage -rfakeroot -d -us -uc -S

(snip)

 fakeroot debian/rules clean
dh clean
   dh_testdir
   dh_auto_clean
   dh_clean
 dpkg-source -b bash-completion-1.1
dpkg-source: info: using source format `3.0 (quilt)'
dpkg-source: info: building bash-completion using existing ./bash-completion_1.1.orig.tar.gz
dpkg-source: info: building bash-completion in bash-completion_1.1-3.debian.tar.gz
dpkg-source: info: building bash-completion in bash-completion_1.1-3.dsc
 dpkg-genchanges -S >../bash-completion_1.1-3_source.changes
dpkg-genchanges: not including original source code in upload
dpkg-buildpackage: binary and diff upload (original source NOT included)
Now running lintian...
Finished running lintian.
```

バイナリパッケージをビルドします。

```
$ debuild -b -us -uc
 dpkg-buildpackage -rfakeroot -D -us -uc -b
dpkg-buildpackage: set CFLAGS to default value: -g -O2

(snip)

dpkg-deb: `../bash-completion_1.1-3_all.deb' にパッケージ `bash-completion' を構築しています。
 dpkg-genchanges -b >../bash-completion_1.1-3_amd64.changes
dpkg-genchanges: binary-only upload - not including any source code
dpkg-buildpackage: binary only upload (no source included)
Now running lintian...
Finished running lintian.
```

20.8 例 3： ptex-bin パッケージ

次に、実際には複数のソースアーカイブを使用している ptex-bin ソースパッケージ (3.1.11+0.04b-0.1) を 3.0 (quilt) 化してみました。

ptex-bin ソースパッケージは、実際には ptex-src-*.tar.gz と jmpost-*.tar.gz の 2 つの上流 tarball から構成されます。さらに Build-Depends: ptex-buildsupport となっていますが、この ptex-buildsupport パッケージは tetex-src-*-stripped.tar.gz のみが含まれる、ほぼビルド専用のパッケージです。すなわち、3 つの上流 tarball が使われています。その上、このソースパッケージには debian/patches/ディレクトリがありますが、quilt や dpatch といった近代的なものではなくビルド時に patch を debian/rules 内で直接呼ぶ構成になっていました。

20.8.1 ソースの用意、名前の変更

```
$ apt-get source ptex-bin
$ apt-get source ptex-buildsupport
```

適当に名前を変えます。ここでは

```
$ mv ptex-bin-3.1.11+0.04b/ptex-src-3.1.11.tar.gz ptex-bin_3.1.11+0.04b+3.0.orig.tar.gz
$ mv ptex-bin-3.1.11+0.04b/jmpost-0.04b.tar.gz ptex-bin_3.1.11+0.04b+3.0.orig-jmpost.tar.gz
$ mv ptex-buildsupport-3.0/tetex-src-3.0-stripped.tar.gz ptex-bin_3.1.11+0.04b+3.0.orig-tetex-stripped.tar.gz
```

としました。3.0 (quilt) では、*.orig.tar.*ext* がまずメインで展開され、そのソースツリーに *component* ディレクトリが掘られてその下に各*.orig-*component*.tar.*ext* が展開されます。また、*component* には英数字とハイフンのみが使えます。

20.8.2 とりあえずビルド

```
$ mkdir ptex-bin-3.1.11+0.04b+3.0
$ cd ptex-bin-3.1.11+0.04b+3.0/
$ cp -a ../ptex-bin-3.1.11+0.04b/debian .
$ mkdir -p debian/source
$ echo '3.0 (quilt)' > debian/source/format
$ dch -v 3.1.11+0.04b+3.0-0.1
( 適当に changelog 追加)
$ debuild -S -us -uc
 dpkg-buildpackage -rfakeroot -d -us -uc -S
dpkg-buildpackage: set CFLAGS to default value: -g -O2
dpkg-buildpackage: set CPPFLAGS to default value:
dpkg-buildpackage: set LDFLAGS to default value:
dpkg-buildpackage: set FFLAGS to default value: -g -O2
dpkg-buildpackage: set CXXFLAGS to default value: -g -O2
dpkg-buildpackage: source package ptex-bin
dpkg-buildpackage: source version 3.1.11+0.04b+3.0-0.1
dpkg-buildpackage: source changed by YOSHINO Yoshihito <yy.y.ja.jp@gmail.com>
 fakeroot debian/rules clean
dh_testdir
dh_testroot
rm -f build-stamp configure-stamp
# Add here commands to clean up after the build process.
# Remove teTeX source directory.
rm -rf tetex-src-3.0
dh_clean
dh_clean: Compatibility levels before 5 are deprecated.
 dpkg-source -b ptex-bin-3.1.11+0.04b+3.0
dpkg-source: info: using source format `3.0 (quilt)'
dpkg-source: info: building ptex-bin using existing ./ptex-bin_3.1.11+0.04b+3.0.
orig-jmpost.tar.gz ./ptex-bin_3.1.11+0.04b+3.0.orig-tetex-stripped.tar.gz ./ptex
-bin_3.1.11+0.04b+3.0.orig.tar.gz
dpkg-source: warning: ignoring deletion of file kanji.defines
dpkg-source: warning: ignoring deletion of file pconvert
dpkg-source: warning: ignoring deletion of file tftopl.ch

(snip)

dpkg-source: info: building ptex-bin in ptex-bin_3.1.11+0.04b+3.0-0.1.debian.tar
.gz
dpkg-source: info: building ptex-bin in ptex-bin_3.1.11+0.04b+3.0-0.1.dsc
 dpkg-genchanges -S >../ptex-bin_3.1.11+0.04b+3.0-0.1_source.changes
```

とりあえず空のソースツリーで作ったのでソースがないという warning はスルーします。

```
$ cd ..
$ dpkg-source -x ptex-bin_3.1.11+0.04b+3.0-0.1.dsc
dpkg-source: warning: extracting unsigned source package (ptex-bin_3.1.11+0.04b+3.0-0.1.dsc)
dpkg-source: info: extracting ptex-bin in ptex-bin-3.1.11+0.04b+3.0
dpkg-source: info: unpacking ptex-bin_3.1.11+0.04b+3.0.orig.tar.gz
dpkg-source: info: unpacking ptex-bin_3.1.11+0.04b+3.0.orig-jmpost.tar.gz
dpkg-source: info: unpacking ptex-bin_3.1.11+0.04b+3.0.orig-tetex-stripped.tar.gz
dpkg-source: info: unpacking ptex-bin_3.1.11+0.04b+3.0-0.1.debian.tar.gz
$ cd -
$ ls -F
COPYRIGHT      README.txt   jbibtex.ch       mkconf         ptexextra.h
COPYRIGHT.jis  configure*   jbibtex.defines  pconvert*      ptexhelp.h
Changes.txt    debian/      jmpost/          pdvitype.ch    tetex-stripped/
Files          jbibd.sed    kanji.c          pltotf.ch      tftopl.ch
INSTALL.txt    jbibextra.c  kanji.defines    ptex-base.ch   usage.c
Makefile.in    jbibextra.h  kanji.h.in       ptexextra.c    version.c
```

メインのソースのトップディレクトリでそのままビルドできるようなものはいいですが、ptex-src-*.tar.gz はそうではないので微妙ですね… ともかく、今回はビルド用ディレクトリを掘ってビルドできるように debian/rules の変更が必要でした。

20.8.3 quilt 化

普段パッチシステム quilt を使っていればほとんど変更の必要はありませんが、このパッケージはパッチは分かれているものの自力で色々やっているのでまず quilt 化が必要でした。

全パッチを-p1 化した上で、debian/rules に書かれている通りの順番でパッチを quilt import && quilt push しました。quilt では同じパッチ名はダメのようですね。

```
$ cd debian/
$ mv patches patches.old
$ sed -i 's@^\(---\|+++\) @&ptex-bin-3.1.11+0.04b+3.0/tetex-stripped/@' patches.old/teTeX/*.patch
$ sed -i 's@^\(---\|+++\) @&ptex-bin-3.1.11+0.04b+3.0/@' patches.old/*.patch
$ sed -i 's@^\(---\|+++\) @&ptex-bin-3.1.11+0.04b+3.0/jmpost/@' patches.old/jmpost/*.patch
$ mv patches.old/Makefile.in.patch patches.old/pTeX_Makefile.in.patch
$ mv patches.old/jmpost/Makefile.in.patch patches.old/jmpost/jmpost_Makefile.in.patch
$ quilt import patches.old/teTeX/*.patch patches.old/*.patch patches.old/jmpost/*.patch
パッチ patches.old/teTeX/Makefile.in.patch を取り込んでいます (Makefile.in.patch として保存されます)
パッチ patches.old/teTeX/common.mk.patch を取り込んでいます (common.mk.patch として保存されます)
パッチ patches.old/teTeX/config.h.patch を取り込んでいます (config.h.patch として保存されます)
パッチ patches.old/teTeX/depend.mk.patch を取り込んでいます (depend.mk.patch として保存されます)
パッチ patches.old/teTeX/splitup.c.patch を取り込んでいます (splitup.c.patch として保存されます)
パッチ patches.old/teTeX/web2c.depend.mk.patch を取り込んでいます (web2c.depend.mk.patch として保存されます)
パッチ patches.old/pTeX_Makefile.in.patch を取り込んでいます (pTeX_Makefile.in.patch として保存されます)
パッチ patches.old/jmpost/jmpost_Makefile.in.patch を取り込んでいます (jmpost_Makefile.in.patch として保存されます)
$ cd ..
$ env QUILT_PATCHES=debian/patches quilt push -a
パッチ Makefile.in.patch を適用しています
patching file tetex-stripped/texk/web2c/web2c/Makefile.in

パッチ common.mk.patch を適用しています
patching file tetex-stripped/texk/make/common.mk

パッチ config.h.patch を適用しています
patching file tetex-stripped/texk/web2c/config.h

パッチ depend.mk.patch を適用しています
patching file tetex-stripped/texk/web2c/lib/depend.mk

パッチ splitup.c.patch を適用しています
patching file tetex-stripped/texk/web2c/web2c/splitup.c

パッチ web2c.depend.mk.patch を適用しています
patching file tetex-stripped/texk/web2c/web2c/depend.mk

パッチ pTeX_Makefile.in.patch を適用しています
patching file Makefile.in

パッチ jmpost_Makefile.in.patch を適用しています
patching file jmpost/Makefile.in

現在位置はパッチ jmpost_Makefile.in.patch です
$ rm -r debian/patches.old/
```

20.8.4 debian/rules, debian/control の変更

tarball をもう展開する必要はありません。ビルド用ディレクトリにコピーすることにします。パッチを当てる必要もないのでその辺も削ります。

```
--- debian/rules        2010-03-18 12:01:15.000000000 +0900
+++ debian/rules.new        2010-03-18 13:19:09.000000000 +0900
@@ -6,29 +6,14 @@
 # Uncomment this to turn on verbose mode.
 #export DH_VERBOSE=1

-####################
-# Things you should change when new upstream version is available.
-
-# Minimal teTeX source tarball.
-# You should install ptex-buildsupport package beforehand.
-TETEX_SRC_TARBALL=/usr/src/tetex-src-3.0-stripped.tar.gz
-
 # teTeX source directory.
-TETEX_SRC_DIR=tetex-src-3.0
-
-# pTeX source tarball.
-PTEX_SRC_TARBALL=ptex-src-3.1.11.tar.gz
+TETEX_SRC_DIR=tetex-src

 # pTeX source directory.
-PTEX_SRC_DIR=ptex-src-3.1.11
-
-# Japanized MetaPost tarball.
-JMPOST_SRC_TARBALL=jmpost-0.04b.tar.gz
+PTEX_SRC_DIR=ptex-src

 # Japanized MetaPost source directory.
-JMPOST_SRC_DIR=jmpost-0.04b
-
-####################
+JMPOST_SRC_DIR=jmpost-src

 DEB_HOST_GNU_TYPE        := $(shell dpkg-architecture -qDEB_HOST_GNU_TYPE)

@@ -42,31 +27,14 @@
 configure: configure-stamp
 configure-stamp:
         dh_testdir
-        # Unpack tarballs.
-        tar xfz $(TETEX_SRC_TARBALL)
+        cp -al tetex-stripped $(TETEX_SRC_DIR)
         mkdir $(TETEX_SRC_DIR)/texk/kpathsea_tetex
         mv $(TETEX_SRC_DIR)/texk/kpathsea/c-proto.h $(TETEX_SRC_DIR)/texk/kpathsea_tetex/
         rm -rf $(TETEX_SRC_DIR)/texk/kpathsea
         ln -s /usr/include/kpathsea $(TETEX_SRC_DIR)/texk/kpathsea
-        tar xfz $(PTEX_SRC_TARBALL) -C $(TETEX_SRC_DIR)/texk/web2c
-        tar xfz $(JMPOST_SRC_TARBALL) -C $(TETEX_SRC_DIR)/texk/web2c/$(PTEX_SRC_DIR)
-
-        # Apply patches to teTeX source
-        for f in debian/patches/teTeX/*.patch; do \
-                patch -p0 -d $(TETEX_SRC_DIR) < $$f; \
-        done
-
-        # Apply patches to pTeX source (should be named as *.patch)
-        # Put patches in debian/patches.
-        (for f in debian/patches/*.patch ; do \
-          patch -p0 -d $(TETEX_SRC_DIR)/texk/web2c/$(PTEX_SRC_DIR) < $$f ; \
-        done)
-
-        # Apply patches to Japanized MetaPost source (should be named as *.patch)
-        # Put patches in debian/patches/jmpost.
-        (for f in debian/patches/jmpost/*.patch ; do \
-          patch -p0 -d $(TETEX_SRC_DIR)/texk/web2c/$(PTEX_SRC_DIR)/$(JMPOST_SRC_DIR) < $$f ; \
-        done)
+        mkdir $(TETEX_SRC_DIR)/texk/web2c/$(PTEX_SRC_DIR)
+        for f in `find . -maxdepth 1 -type f`; do cp -al $$f $(TETEX_SRC_DIR)/texk/web2c/$(PTEX_SRC_DIR); done
+        cp -al jmpost $(TETEX_SRC_DIR)/texk/web2c/$(PTEX_SRC_DIR)/$(JMPOST_SRC_DIR)

         # Copy texmf.cnf from your system.
         cp /usr/share/texmf/web2c/texmf.cnf \
```

ptex-buildsupport は不要になりました。

```
--- debian/control        2010-03-18 12:01:15.000000000 +0900
+++ debian/control.new        2010-03-18 13:23:09.000000000 +0900
@@ -2,7 +2,7 @@
 Section: tex
 Priority: optional
 Maintainer: Masayuki Hatta (mhatta) <mhatta@debian.org>
-Build-Depends: debhelper (>> 4.0.0), texlive-binaries,
+Build-Depends: debhelper (>> 4.0.0), texlive-binaries,
 texlive-extra-utils, texlive-metapost, flex, bison, libkpathsea-dev, \
ptex-base (>= 2.4), ptex-buildsupport (>= 3.0), libtool
 Standards-Version: 3.7.3

 Package: ptex-bin
```

20.8.5 ビルド

```
$ debuild -S -us -uc
 dpkg-buildpackage -rfakeroot -d -us -uc -S
dpkg-buildpackage: set CFLAGS to default value: -g -O2

(snip)
```

試しに `dpkg-source -x` **してみます。**

```
$ dpkg-source -x ptex-bin_3.1.11+0.04b+3.0-0.1.dsc
dpkg-source: warning: extracting unsigned source package (ptex-bin_3.1.11+0.04b+3.0-0.1.dsc)
dpkg-source: info: extracting ptex-bin in ptex-bin-3.1.11+0.04b+3.0
dpkg-source: info: unpacking ptex-bin_3.1.11+0.04b+3.0.orig.tar.gz
dpkg-source: info: unpacking ptex-bin_3.1.11+0.04b+3.0.orig-jmpost.tar.gz
dpkg-source: info: unpacking ptex-bin_3.1.11+0.04b+3.0.orig-tetex-stripped.tar.gz
dpkg-source: info: unpacking ptex-bin_3.1.11+0.04b+3.0-0.1.debian.tar.gz
dpkg-source: info: applying Makefile.in.patch
dpkg-source: info: applying common.mk.patch
dpkg-source: info: applying config.h.patch
dpkg-source: info: applying depend.mk.patch
dpkg-source: info: applying splitup.c.patch
dpkg-source: info: applying web2c.depend.mk.patch
dpkg-source: info: applying pTeX_Makefile.in.patch
dpkg-source: info: applying jmpost_Makefile.in.patch
```

無事当たりました。バイナリパッケージをビルドします。

```
$ debuild -b -us -uc
 dpkg-buildpackage -rfakeroot -D -us -uc -b
dpkg-buildpackage: set CFLAGS to default value: -g -O2

(snip)

dpkg-buildpackage: warning: Build dependencies/conflicts unsatisfied; aborting.
dpkg-buildpackage: warning: (Use -d flag to override.)
debuild: fatal error at line 1330:
dpkg-buildpackage -rfakeroot -D -us -uc -b failed
$ mk-build-deps

(snip)

$ sudo dpkg -i ptex-bin-build-deps_1.0_all.deb

(snip)

$ sudo aptitude install ptex-bin-build-deps

(snip)

$ debuild -b -us -uc
 dpkg-buildpackage -rfakeroot -D -us -uc -b
dpkg-buildpackage: set CFLAGS to default value: -g -O2

(snip)

dh_builddeb
dh_builddeb: Compatibility levels before 5 are deprecated.
dpkg-deb: '../ptex-bin_3.1.11+0.04b+3.0-0.1_amd64.deb' にパッケージ 'ptex-bin' を構築しています。
dpkg-deb: '../jbibtex-bin_3.1.11+0.04b+3.0-0.1_amd64.deb' にパッケージ 'jbibtex-bin' を構築しています。
dpkg-deb: '../jmpost_3.1.11+0.04b+3.0-0.1_amd64.deb' にパッケージ 'jmpost' を構築しています。
 dpkg-genchanges -b >../ptex-bin_3.1.11+0.04b+3.0-0.1_amd64.changes
dpkg-genchanges: binary-only upload - not including any source code
dpkg-buildpackage: binary only upload (no source included)
Now running lintian...
W: jmpost: binary-without-manpage usr/bin/pmakempx
W: jbibtex-bin: copyright-without-copyright-notice
Finished running lintian.
```

無事できました。

20.9 References

- **第 41 回東京エリア Debian 勉強会資料（ 2008 年 6 月）。パッチシステムなどの使い方の詳細はこちらにまとまっています。**
- `dpkg-source(1)`（ `man 1 dpkg-source`）。`3.0 (quilt)` **などのソースフォーマットの詳細。**
- `http://wiki.debian.org/ReleaseGoals/NewDebFormats`
- `http://release.debian.org/squeeze/goals.txt`。 **squeeze release goals.**

21 piuparts の使い方

岩松 信洋

21.1 はじめに

piuparts は 次期リリース squeeze の目標の一つに挙げられている Package clean install/uninstall を達成するためのサポートツールです。既に構築された Debian パッケージのインストール、アンインストール、アップグレードのチェックを行います。通常、パッケージ構築時の依存関係チェックや実際の構築には pbuilder/cowbuilder を使います。パッケージのインストール、動作確認までは行いますが、アンインストールまでの確認を行っているパッケージメンテナは少ないようで(実際にそのまま使う人が多いためと考えられる) 、アンインストールできない事が稀にありました。最悪の場合、パッケージを作ってテストせずにアップーロードしてしまう事もあるようです。また、 stable からのアップグレードチェックもパッケージメンテナはあまりやってないのではないでしょうか。このような問題をチェックするためのツールとして piuparts は作られました。では、 piuparts はどのように動き、どのように使うのか見ていきましょう。

21.2 piuparts の使い方

piuparts を使ってパッケージのチェックを行う場合には、 piuparts コマンドにチェックしたいパッケージを指定します。例えば、 libcv4_2.0.0-4_i386.deb パッケージをチェックしてその結果を/tmp/libcv4_2.0.0-4_i386.piuparts-log に保存する場合には以下のように実行します。実行するとログが標準出力にも出力されますが、-l オプションで指定したログ指定先にも保存されます。出力されるログからパッケージのインストール、アンインストール、アップグレードを確認することができます。

```
$ sudo piuparts libcv4_2.0.0-4_i386.deb -l /tmp/libcv4_2.0.0-4_i386.piuparts-log
0m0.0s INFO: --------------------------------------------------------------------------------
0m0.0s INFO: To quickly glance what went wrong, scroll down to the bottom of this logfile.
0m0.0s INFO: FAQ available at http://wiki.debian.org/piuparts/FAQ
0m0.0s INFO: --------------------------------------------------------------------------------
0m0.0s INFO: piuparts version 0.38 starting up.
0m0.0s INFO: Command line arguments: /usr/sbin/piuparts libcv4_2.0.0-4_i386.deb -l /tmp/libcv4_2.0.0-4_i386.piuparts-log
0m0.0s INFO: Running on: Linux chimagu 2.6.31-1-686 #1
 SMP Sun Nov 15 20:39:33 UTC 2009 i686
0m0.0s DEBUG: Starting command: ['dpkg', '--info', 'libcv4_2.0.0-4_i386.deb']
0m0.2s DUMP:
.... 省略 .....
6m32.6s DEBUG: Starting command: ['chroot',
 '/tmp/tmplunhrZ', 'umount', '/proc']
6m32.6s DEBUG: Command ok: ['chroot',
 '/tmp/tmplunhrZ', 'umount', '/proc']
6m33.0s DEBUG: Removed directory tree at /tmp/tmplunhrZ
6m33.0s INFO: PASS: All tests.
6m33.0s INFO: piuparts run ends.
```

piuparts の使い方は大きくわけて 2 つあります。一つはローカル PC にあるパッケージをチェックする場合、もう一つは既に Debian にインストールされているパッケージをチェックする場合に使います。前者はパッケージメンテナがよく使う方法です。パッケージをアップロードする前にパッケージを指定してテストします。後者の場合はあまりメンテナはあまり使う機会はないと思いますが、後で説明する piuparts.debian.org で使われています。

21.3 piuparts の動作

では、 piuparts の動作を見てみましょう。状態遷移図を図 47 に示します。非常にシンプルなチェック方法になっていることがわかります。

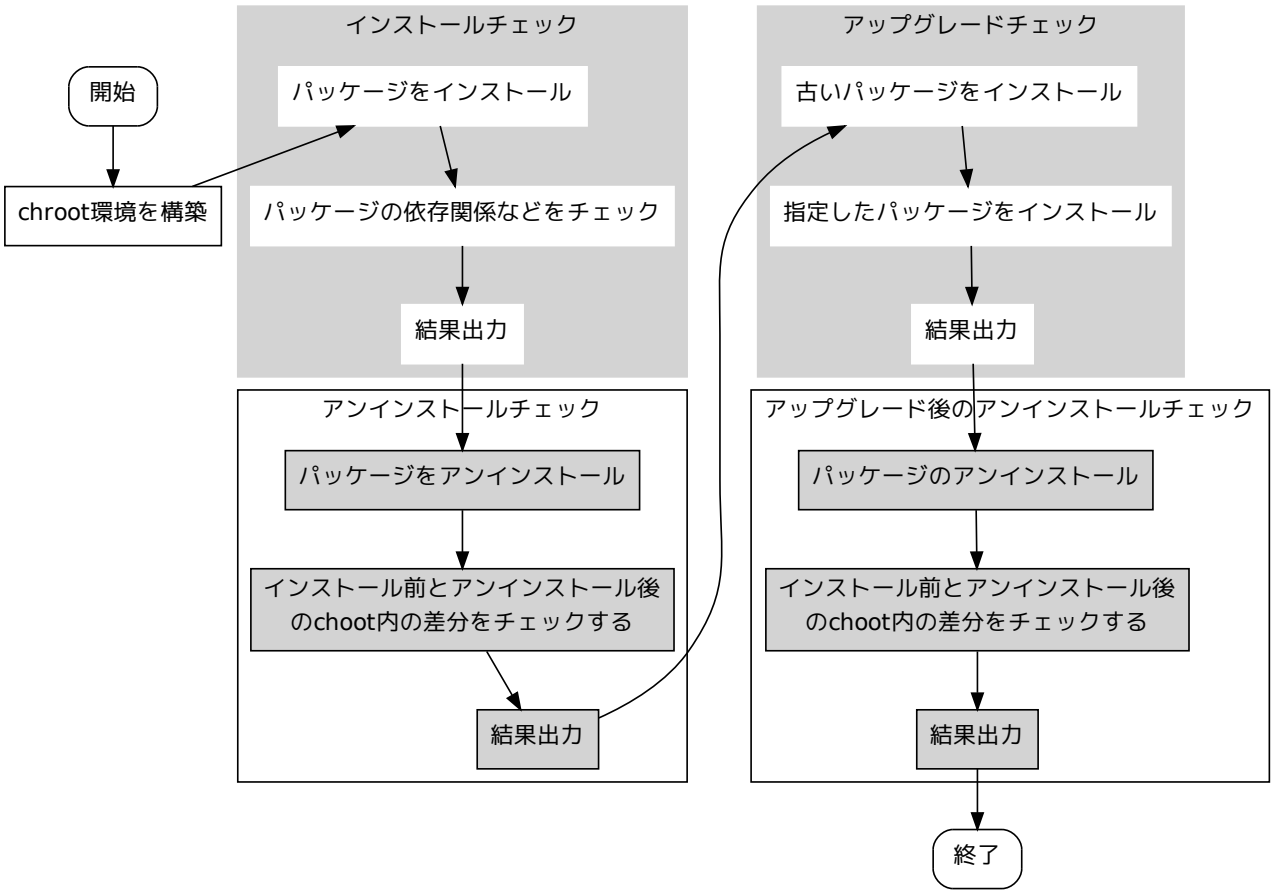


図 47　piuparts の動作

大まかな動きは以上になりますが、内部では dpkg、 apt の動きを利用したものになっています。例えば、パッケージのインストールチェックは以下のような動きになっています。

1. dpkg -i で 指定されたパッケージをインストールする。
   依存するパッケージがあるばあい、これは失敗する。
2. apt-get -yf --no-remove でパッケージの依存関係を apt で回避してインストールする。

これは、パッケージ依存関係をパッケージ情報から抽出して指定する方法を取らず、 apt のチェック機構を用いて依存関係を回避しようとしています。

21.4 ログの見方

piuparts はログが多いので正しい動きをしているのか非常にわかりずらいです。どのように見たらよいのか簡単に説明します。

piuparts のログは、基本的に以下の順で出力されます。

1. コマンド実行 (DEBUG: Starting command:)
2. 実行開始タグ (DUMP:)
3. 実行時のログ
4. コマンド結果 (ERROR: or DEBUG: Command ok)

そして、最後にテスト結果が出力されます。次にテストが正常に終了した場合と、問題がある場合を見てみます。

21.4.1 テストが正常に終了した場合

テストに問題がない場合には、以下のように出力されます。インストール → アンインストール、インストール → アップグレード → アンインストール のチェックが正常に終了していることがわかります。

```
..... 省略.....
6m13.7s INFO: PASS: Installation and purging test.
..... 省略.....
6m32.6s INFO: PASS: Installation, upgrade and purging tests.
..... 省略.....
6m33.0s INFO: PASS: All tests.
6m33.0s INFO: piuparts run ends.
```

21.4.2 エラーがある場合

エラーがある場合には、 ERROR:の次にエラー内容がされます。以下に、実際のエラー内容を示します。これは、 upstart のチェック結果ですが、 essential パッケージである、 sysvinit をアンインストールしようとして、エラーになっています。

```
0m6.0s DEBUG: Starting command: ['chroot', '/org/piuparts.debian.org/tmp/tmpZ-SX9D', 'apt-get', '-y', 'install', 'upstart']
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^: コマンド実行
0m6.3s DUMP:
^^^^^^^^^^^^: コマンド実行時の情報
  Reading package lists...
  Building dependency tree...
  The following extra packages will be installed:
    libdbus-1-3
  Recommended packages:
    dbus
  The following packages will be REMOVED:
    sysvinit
  The following NEW packages will be installed:
    libdbus-1-3 upstart
  WARNING: The following essential packages will be removed.
  This should NOT be done unless you know exactly what you are doing!
    sysvinit
  0 upgraded, 2 newly installed, 1 to remove and 0 not upgraded.
  Need to get 636kB of archives.
  After this operation, 1196kB of additional disk space will be used.
  E: There are problems and -y was used without --force-yes
0m6.3s ERROR: Command failed (status=100): ['chroot', '/org/piuparts.debian.org/tmp/tmpZ-SX9D', 'apt-get', '-y', 'install', 'upstart']
^^^^^^^^^^^^^: コマンド結果
```

21.5 piuparts のオプション

普通の使い方では使いづらいので、 piuparts で提供されているオプションを自分が使っている開発環境に合わせて使うのが普通のようです。以下ではよく使うオプションを紹介します。

21.5.1 pbuilder の base.tgz を piuparts で利用する

piuparts はテストする度に base イメージを構成するパッケージ群をミラーサーバから取得し、 base イメージを構築します。キャッシュする機構はいまのところ存在せず、毎回取得する仕様になっています。これ

ではサーバに負荷がかかります。そこで、-p オプションを使います。 このオプションは pbuilder の base イメージ/var/cache/pbuilder/base.tgz を使ってテストを行うオプションです。通常、パッケージメンテナは pbuilder/cowbuilder を使ってパッケージビルドチェックを行うので、便利なオプションの一つです。また、pbuilder を使ってないが、 base.tgz を独自のスクリプトで保持している人もいるでしょう。この場合には--basetgz オプションで tgz ファイルを指定することによって、利用できます。

21.5.2 ディストリビューションの指定

piuparts は unstable(sid) だけでなく、現在サポートされている Debian のディストリビューションと Ubuntu のディストリビューションをサポートしています。ディストリビューションの指定には-d オプションを指定します。これは複数指定することができ、指定した順にテストが実行されます。以下の例では、 sid 環境を構築し、テストした後、squeeze の環境を構築し、テストします。

```
$ sudo piuparts -d sid -d squeeze libcv4_2.0.0-4_i386.deb
.....
```

21.6 debian にインストールされているパッケージのテスト

既に debian にインストールされているパッケージのテストを行う場合には、--apt オプションを使います。特定のディストリビューションにあるパッケージのテストを行いたい場合には、-d オプションでディストリビューションを指定する必要があります。

```
$ sudo piuparts --apt -d squeeze libcv4
```

21.6.1 ミラーサーバの指定

デフォルトでは、 piuparts が使う Debian リポジトリは、/etc/apt/sources.list からパーサして利用します。一家に一台 Debian ミラーの時代です。グローバルなミラーサーバを使用せずにローカルにあるミラーサーバを指定する場合には、--mirror オプションを使って指定します。

```
$ sudo piuparts -p libcv4_2.0.0-4_i386.deb --mirror http://debmirror.example.org/debian
```

21.6.2 依存するパッケージの回避方法

通常、ライブラリソースパッケージは libfoo0、 libfoo-dev など複数のバイナリパッケージを生成します。この場合、 libfoo-dev パッケージは libfoo0 パッケージに依存しているので、 libfoo-dev パッケージだけでテストしても、libfoo0 がないのでテストでエラーになります。この場合には、パッケージを 2 つ (libfoo0, libfoo-dev) 指定することで回避できます。

```
$ sudo piuparts -p libcv-dev_2.0.0-4_i386.deb libcv4_2.0.0-4_i386.deb
```

また、ソースパッケージから作成されたバイナリパッケージのテストを行う場合には、*.changes ファイルを指定します。

```
$ sudo piuparts -p opencv_2.0.0-4_i386.changes
```

21.7 piuparts.debian.org

piuparts がパッケージとして用意されたとしても、まだ利用しているパッケージメンテナは少ないようです。また、パッケージ作成時には問題がなかったが、依存関係があるパッケージが更新および削除され、正常にインストール等ができない状態になる場合があります。そこで、 QA チームは既に Debian にインストールされたパッケージをチェックするためのサービス piuparts.debian.org を立ち上げました。これでチェックされた内容は、 qa.debian.org で提供さ

れている情報の一部として、表示されています。

21.7.1 現在チェックエラーになっているパッケージ

piuparts でチェックでエラーになっているパッケージはタグとユーザダグで検索できます。

- タグ： piuparts
- ユーザタグ： debian-qa@lists.debian.org

BTS では以下の URL で参照できます。 http://bugs.debian.org/cgi-bin/pkgreport.cgi?tag=piuparts;users=debian-qa@lists.debian.org

21.8 現在の問題点

現在、 piuparts にはいくつかの問題点があります。 http://packages.qa.debian.org で表示されるチェック結果が誤解を受けやすいという点です。依存しているパッケージが問題を持っているのに、それが自分のパッケージにエラーとなって表示されます。情報を追えばどのパッケージでエラーになっているのか分かりますが、情報を追うのがめんどうです。

細かいところでは、-B の使い方がわかりません。

```
$ piuparts --help
.... 中略 ....
 -B FILE, --end-meta=FILE
                     XXX
```

ちなみに #560050 で 報告されていますが、開発者本人が報告しているので修正する気がないのかもしれません。

21.9 まとめ

今回は基本的な使い方と、パッケージメンテナから利用する場合に使うオプションを説明しました。パッケージメンテナの方は自分でもっているパッケージテストに組み込んでみてはいかがでしょうか。パッケージの基本的な部分の問題が減るのでよいと思います。

また、現在 piuparts v2 を開発中です。開発は bzr 上で行われています。 http://code.liw.fi/piuparts2/bzr/trunk/ ソースコードも多くなく、やっていることも単純です。興味のある方は参加してみてはいかがでしょうか。

22 qemubuilder 2009 年アップデート

上川純一

22.1 qemubuilder の基本コンセプト

qemubuilder は Debian パッケージをビルドするためのツールです。debootstrap のクロスブートストラップ機能を利用してベースイメージを作成した後、qemu を利用して各アーキテクチャ用の仮想マシンを実行し、その中でパッケージをビルドします。ネイティブビルドと変わらない使用感でパッケージのビルドができるので面倒なクロスビルドの設定が必要ありません。特に Debian パッケージは buildd でネイティブビルドされ、クロスビルドされない前提なので、buildd でビルドできるようなパッケージの作成・デバッグに便利です。

22.2 qemubuilder の使い方

利用したいアーキテクチャ向けのカーネルと設定ファイルを用意します。手元の設定例です:

```
KERNEL_IMAGE=vmlinuz-2.6.24-1-versatile-armel
ARCH=armel
BASEPATH=/home/dancer/tmp/base-armel.qemu
INITRD=
```

create でイメージをまず作成、BASEPATH に指定したファイル名に qemu の RAW ディスクイメージが作成されます。update でディスクイメージをアップデートできます。そして、パッケージをビルドするには build で dsc ファイルを指定します。

```
# qemubuilder --configfile arm.config --create
# qemubuilder --configfile arm.config --update
# qemubuilder --configfile arm.config --build xxx.dsc
```

22.3 qemubuilder の課題

カーネルと設定ファイルの取得が今一番面倒なところです。Debian の標準のカーネルと initrd でできた時期もあったのですが、現在そういうようにはなっていません。

アーキテクチャの組み合わせがあまりにも多いため、動く組み合わせを同定することや、デバッグが困難です。スクリプトで自動化すること、またテストの自動化が必要ではないかと考えています。

また、qemu の-append コマンドでカーネルにブートパラメータを指定できることを想定していますが、実際にはそれができないアーキテクチャ (ppc など) があり、そのままでは動きません。[*59]

最近は kFreeBSD アーキテクチャなども登場してきていますが、現状の設計では手が回らなさそうです。

[*59] いまでもそうかはわかりません

23 東京エリア Debian 勉強会予約システムの構想

上川 純一

23.1 背景

東京エリア Debian 勉強会では「 えんかい君」を予約システムとして利用していました。えんかい君はシンプルなユーザインタフェースで認証もなく、全員のメールアドレスと名前が閲覧でき、他人の登録を誰でも削除できるなど、利用者を信頼したモデルになっていました。後で立ち上がった関西では cotocoto を利用していました。 cotocoto は DFSG の観点では non-free なサービスです。

「 えんかい君」は YLUG などでも利用されていましたが、不便でした。東京では、「 えんかい君」の制限を回避するため、課題の提出をメール経由でやっていました。当初はフリーフォーマットのメールを LaTeX 形式に上川がバッチで変換する形式をとっており、のちに LaTeX のソースコードをメールで git format-patch で送るという運用になっていました。ただ、 Git で課題提出をしていても、マージが面倒という問題点がありました。

2009 年 12 月の勉強会登録には実験的に atnd を利用しました。 atnd は DFSG non-free なサービスですが、最近流行している勉強会等の予約システムです。

DFSG 準拠のアプリケーションのほうが望ましいが、「 えんかい君」ではうまく運用できないということと、アプリケーション自体はシンプルな問題であることが予想されたため、自前で勉強会予約システムを準備してみることにしました。

23.2 実装目標

Debian 勉強会の予約システムでは何が必要でしょうか。

- イベントの主催者が簡便に登録情報を設定することができること。
- イベントの主催者が事前課題を設定し、回答を簡単に収集することができること。
- イベントの主催者が簡単に参加人数を確認することができること。
- イベントの主催者が新規参加者の情報を迅速に確認できること。
- イベントの主催者が参加者に直接連絡がとれる手段があること。
- 参加者が簡単に事前課題もあわせて登録できること。
- 参加者がイベント参加をキャンセルする方法があること。
- 参加者が参加しているイベントを把握する方法があること。

他にもいろいろあるかもしれませんが、とりあえずこういうものを目標にしてやってみました。
そして、 DFSG Free であることが望ましいです。

23.3 開発環境の準備

23.3.1 App Engine Python SDK の準備

今回はウェブアプリケーションのフレームワークとして、Python 版の Google App Engine を利用しました。開発環境を Debian GNU/Linux sid 上で準備する方法を紹介します。

まず、Debian GNU/Linux sid の環境を用意します。

次に、Google App Engine の Python 版の開発環境をダウンロードします。Google App Engine のサイト[*60]にいって最新の SDK をダウンロードしてきます。

「Linux/その他のプラットフォーム」向けの google_appengine_1.3.1.zip をダウンロードしてきました。

```
# apt-get install unzip python python-openssl python-webtest python-yaml
$ wget http://googleappengine.googlecode.com/files/google_appengine_1.3.1.zip
$ unzip google_appengine_1.3.1.zip
```

これでインストールは完了です。Google App Engine のインストールディレクトリを ./google_appengine, App Engine アプリケーションのソースコードのおいている場所を ./utils/gae とします。utils/gae ディレクトリにから dev_appserver.py を実行すれば、開発用のウェブサーバが起動します。

```
hoge@core2duo:appengine/utils/gae$ ../../google_appengine/dev_appserver.py .
INFO     2010-02-16 15:28:08,816 appengine_rpc.py:159] Server: appengine.google.com
Allow dev_appserver to check for updates on startup? (Y/n): n
dev_appserver will not check for updates on startup.  To change this setting, edit
/home/hoge/.appcfg_nag
WARNING  2010-02-16 15:28:13,792 datastore_file_stub.py:623] Could not read datasto
re data from /tmp/dev_appserver.datastore
WARNING  2010-02-16 15:28:13,906 dev_appserver.py:3581] Could not initialize images
 API; you are likely missing the Python "PIL" module. ImportError: No module named
 _imaging
INFO     2010-02-16 15:28:13,914 dev_appserver_main.py:399] Running application deb
ianmeeting on port 8080: http://localhost:8080
```

23.3.2 テストの実行方法

Django の通常のアプリケーションはテスト用の仕組みがあるようなのですが、appengine にはないようです。ここでは、WebTest モジュールを利用して自動テストコードを実装しています。

```
$ PYTHONPATH=../../google_appengine:../../google_appengine/lib/django/ \
 python testSystem.py
```

23.4 実装

[*60] http://code.google.com/intl/ja/appengine/

23.4.1 認証の仕組み

このアプリケーションでは Google App Engine を利用しています。ユーザ認証は Google App Engine で標準で提供される Google の認証を流用しています。パスワードの管理やユーザのメールアドレスの管理などをフレームワークに一任することで管理を簡単にしています。

23.4.2 データベースの構造

バックエンドのデータベースには、 AppEngine の Datastore を利用しています。 Event と、 Attendance と UserRealName というのを定義しています。

Event は主催者がイベントについて登録した情報を保持しています。イベント毎に存在しています。

Attendance はユーザがイベントに登録したという情報を保持しています。イベントに対して登録したユーザの数だけ存在します。

UserRealname はユーザの表示名前の情報を保持しています。各ユーザ毎に存在します。

```
class Event(db.Model):
    eventid = db.StringProperty()
    owner = db.UserProperty() # the creator is the owner
    owners_email = db.StringListProperty() # allow owner emails to be added if possible
    title = db.StringProperty()
    location = db.StringProperty(multiline=True)
    content = db.StringProperty(multiline=True)
    content_url = db.StringProperty()
    prework = db.StringProperty(multiline=True)
    event_date = db.StringProperty()
    timestamp = db.DateTimeProperty(auto_now_add=True)
    capacity = db.IntegerProperty() # the number of possible people attending the meeting

class Attendance(db.Model):
    eventid = db.StringProperty()
    user = db.UserProperty()
    user_realname = db.StringProperty() # keep a cache of last realname entry.
    prework = db.StringProperty(multiline=True) # obsolete, but used in initial version
    prework_text = db.TextProperty() # Used everywhere, populate from prework if available.
    attend = db.BooleanProperty()
    enkai_attend = db.BooleanProperty()
    timestamp = db.DateTimeProperty(auto_now_add=True)

class UserRealname(db.Model):
    """Backup of user realname configuration so that user doesn't have to reenter that information."""
    user = db.UserProperty()
    realname = db.StringProperty()
    timestamp = db.DateTimeProperty(auto_now_add=True)
```

23.4.3 ソースコードの構造

ソースコードは現在下記の構成です。

- debianmeeting.py: どのページがどのコードを呼び出すのかという部分を管理しているコードです。あと、どこに入れるのか迷ったコードもここにあるかも。
- admin_event.py: 主催者のイベントの管理関連のコードです。
- user_registration.py: ユーザの登録関連のコードです。
- webapp_generic.py: とりあえず共通のロジックを定義しています。 POST と GET を同じように扱うためのコードなどが入っています。
- schema.py: データストアのスキーマが定義されています。
- send_notification.py: メール送信と XMPP 送信ロジックが記述されています。
- testSystem.py: ユニットテストです。

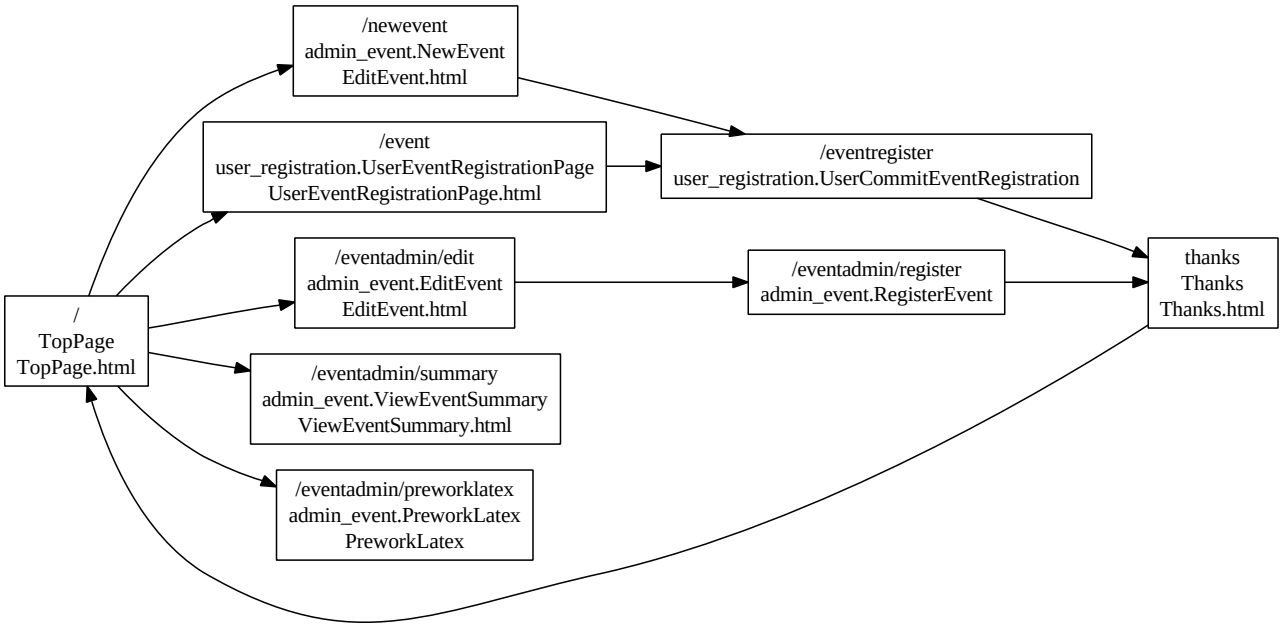
ソース内部からテンプレートファイルが参照されています。

- EditEvent.html
- PreworkLatex.txt
- RegisterEvent.txt

- Thanks.html
- TopPage.html
- UserCommitEventRegistration.txt
- UserEventRegistrationPage.html
- UserEventRegistrationPage_Simple.html
- ViewEventSummary.html

23.4.4 ウェブページの遷移

ウェブページの遷移とソースコードの対応をみてみます。



23.5 今後の展望

とりあえずは動いています。今後、何が変わるべきか。今後どういう点が実装されるべきか。パッチウェルカム。

24 Python も Google App Engine も知らない人が「Debian 勉強会予約管理システム」のソースを見てみたよ

のがたじゅん

24.1 はじめに

結論をいうと当初の目標「OpenID よるログインの実装」はできませんでした。

24.2 とりあえずやったことを書いてみる

それはさておき、まったく何もしなかったわけではないので、この一ヶ月やってみたことを書きます。

24.2.1 ソースはどこ

「見る」と言ったからからには、「Debian 勉強会予約管理システム」のソースはどこかと探してみると、勉強会リポジトリの utils/gae/ディレクトリに置いてあります。

自分は関西 Debian の資料を作るので手元にリポジトリはありましたが、見てみたいなと思った人は勉強会リポジトリを git clone してください。

```
$ git clone git://git.debian.org/git/tokyodebian/monthly-report.git
```

24.2.2 Google App Engine とは何者?

Google App Engine が何者かわからないので公式ページを見てみた。

- http://code.google.com/intl/ja/appengine/

ええっと。ざっくり言うと Web アプリに必要なサーバーからデータベース、開発環境までまるっと一式無償で使わせてくれる Google のサービスという認識でおｋ?

フレームワークには Java と Python を使ったものがあって、勉強会予約システムは Python を使ってる。

公式ページの頭のところに開発を始める人のためのクイックガイドが書いてあるので、それに沿って始めればいいのか。

24.2.3 クイックスタートの順番にしてみた

まずクイックスタート 1 番目には、「App Engine カウントを登録します」と書いてあるけど、今のところ公開するつもりはないので飛ばして、「Google App Engine SDK for Python」の「Linux/その他のプラットフォーム」の SDK をダウンロード。

- http://googleappengine.googlecode.com/files/google_appengine_1.3.1.zip

アーカイブを展開して、クイックスタートを見ると「スタートガイドを参照します」だそうなので読んでみた。

- http://code.google.com/intl/ja/appengine/docs/python/gettingstarted/

「概要」を見るとスタートガイドを読むと一通り作れるように説明してあるので、まず読んでみる。

「開発環境」の説明を読むと、SDK にある dev_appserver.py を使うとローカルで動かすこともできるのか。Python は Debian だと頼まずとも入っているので、特にインストールする必要はなし。ライブラリ関係も書いてないので、飛ばしてもいいかな。

Python といえばインデントなので、エディタ必須だけど開発に使う環境はどうしたらいいだろう。試しに emacs で python-mode.el、pymacs、ipython の環境を作ろうとしたけど、どうもしっくりこない。

いい機会なので、geany、gedit、kate、scribes などエディタをとっかえひっかえ試してみたけど、gedit の python コンソール、コードスニペット、外部のツール、埋め込み式の端末プラグインを有効にすると、タブ補完も効くし、ターミナルや Python コンソールもあるので試しながら使えるので、なかなかいい感じ。灯台下暗しとはこのことか。

24.2.4 世界のみなさんこんにちは!

スタートガイドの「Hello,world!」からコードが出てきた。

まず最初は CGI を作るとき一番最初に説明するような HTTP ヘッダとメッセージを出力する helloworld.py ともう一つ。Google App Engine ならではの設定ファイルを YAML で書いて置かないと、使えないそうな。

アプリケーションのテストはディレクトリを指定して SDK についてた dev_appserver.py を実行。

```
$ google_appengine/dev_appserver.py helloworld/
```

ブラウザで http://localhost:8080/ を開くとアプリケーションが使えると。

Web サーバーを起動したままコードを書き換えられるのは、Ruby の Sinatra みたい。

24.2.5 webapp フレームワークがわからない

> App Engine には、シンプルな独自の Web アプリケーション フレームワークが用意されています。これが webapp です。
>
> webapp フレームワークの使用 - Google App Engine - Google Code: http://code.google.com/intl/ja/appengine/docs/python/gettingstarted/usingwebapp.html より。

ええ!シンプルというけれど、なんの説明もなくいきなり!

すいません。ここでわからなくなり、みんなの Python を買って読んだりしていたら時間切れになりました。

予約システムの debianmeeting.py なども見たところ、チュートリアルから派生した感じのようなのですが、似たような事をしているということはチュートリアルの意味がわからなければ、予約システムもわからないということで…。すいません。

24.3 これから

24.3.1 予約システムで解決したいところ

- OpenID でログインしたい

  今回、解決したい目標でもあったことですが、勉強会に参加するために、わざわざ Google アカウントを取らなければいけないのは参加者の利便性をそこなうし、Google のプラットフォーム依存は自由ではないので早めに解決したいですね。

  幸い、Google App Engine のアプリケーションギャラリーに OpenID のサンプル[*61] があるので、それを

[*61] http://appgallery.appspot.com/results?q=openid

見ながら、なんとか実装できるようになりたいと思ってます。

- トップページから予約ページに飛びたい
  予約システムのトップページで、参加者が EventID を入力するようになっていますが、 EventID が長すぎるので、おそらく誰も使っていないと思われます。 EventID を入力しなくとも、トップページに直近のイベントのリンクがあれば EventID を入力しなくて済むので、これも解決したいですね。

- テンプレートの HTML をきれいにしたい
  今は div タグでくるんだりして、あらっぽいので直したいですね。

事前課題でも書いていた人がいましたが、シンプルであるのはいいところだと思うので、シンプルさをうしなわないように、よりよくしたいですね。

25 東京エリア Debian 勉強会 2009 年度各種イベント開催実績と総括

上川 純一

今月で 5 年目の Debian 勉強会が終了しました。 Debian Developer になった人がいたり変化もみられ、私生活の面でも結婚したメンバーが多数いたり、転職したメンバーがいたり、当時と所属が変わっていないメンバーのほうがめずらしくなってきました。

25.1 Debian 勉強会の野望の進捗具合

今年は Debian 勉強会にとって大きなできごとがありました。当初の目標であった Debian Developer の育成という目標がすこしづつ実現してきたのです。東京エリア Debian 勉強会常連の岩松さん、関西 Debian 勉強会立ち上げ時期に尽力した矢吹さんが Debian Developer になりました。苦節 5 年。おめでとうございます。

2009 年までは基本的な Debian 開発者になるまでの基本的な教養の共有を目的にやってきました。これからは少し方向性を変えて開発に必要な実践的な内容にシフトしていってよいかと考えています。 2010 年はスポンサー、 NMU、 BSP の行い方をより実践的に行える方向を模索したいと思います。

25.2 運営方法

2009 年の勉強会には上川は半分くらいしか出席しておらず、岩松・前田が中心として運営にあたりました。

事前課題は latex のソースコードに対する git format-atch の出力を ML に投稿するという手順で、宴会君と atnd を補助的に利用しました。勉強会会場の予約はとりあえず 30 人くらいが入れる場所を確保しましたが、宴会会場の予約は人数の確定する開催二日前、でした。

勉強会の会費は 500 円を維持しています。過去荻窪の「 あんさんぶる荻窪」という公民館を利用していたので、費用計算もそこを基準におこなってきていました。 2009 年はいろいろな会場を試しており、公営ではない施設も試しており、会場費用の高騰にともない赤字決算になった回もあります。公営の施設を利用していると会場が非常に安かったので印刷費用がまかなえていました。

幹事は昨年までは上川が集中的にしていたので運営についてのノウハウが十分伝わっていなかった苦労もあったみたいです。

25.3 基本的な数値

Debian 勉強会は毎回事前課題事後課題を設定しており、予習復習を必要だとうたっている勉強会です。実際にどれくらいの人が出席しているのか、またその人たちがどれくらい事前課題・事後課題を提出しているのか、確認してみましょう。図 48 です。値は一年の移動平均です。

毎回の参加者の人数と、その際のトピックを見てみます。今年の場合は、平日の夜に行った GPG キーサインパーティーの参加者の多さが目立っています。

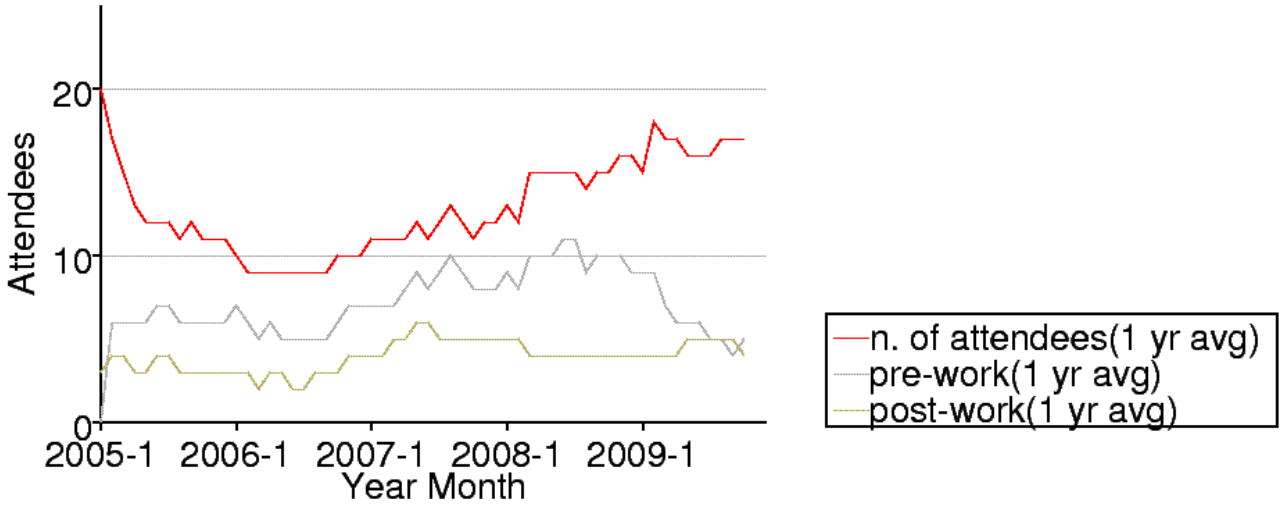


図 48　東京エリア Debian 勉強会事前課題・事後課題提出実績 (12ヶ月移動平均)

表 3　東京エリア Debian 勉強会参加人数 (2009 年)

	参加人数	内容
2009 年 1 月	12	一年を企画する
2009 年 2 月	30	OSC パッケージハンズオン
2009 年 3 月	23	Common Lisp, パッケージ作成
2009 年 4 月	15	Java Policy, ocaml, 開発ワークフロー
2009 年 5 月	13	MC-MPI パッケージ化、 Erlang、 Android アプリ、 DDTP
2009 年 6 月	14	DDTP・ DDTSS、 bsdstats パッケージ、 Debian kFreeBSD
2009 年 7 月	?	スペインにて Debconf 9
2009 年 8 月	14	スペイン Debconf 9 参加報告
2009 年 9 月	26	GPG キーサインパーティー
2009 年 10 月	?	OSC Tokyo Fall
2009 年 11 月	12	Octave, R, gnuplot, auto-builder
2009 年 12 月	?	忘年会

表 4　東京エリア Debian 勉強会参加人数 (2005-2006 年)

	参加人数	内容
2005 年 1 月	21	秘密
2005 年 2 月	10	debhelper 1
2005 年 3 月	8	(早朝) debhelper 2、social contract
2005 年 4 月	6	debhelper 3
2005 年 5 月	8	DFSG、 dpkg-cross、lintian/linda
2005 年 6 月	12	alternatives、 d-i
2005 年 7 月	12	toolchain、 dpatch
2005 年 8 月	7	Debconf 参加報告、ITP からアップロードまで
2005 年 9 月	14	debconf
2005 年 10 月	9	apt-listbugs、 バグレポート、 debconf 翻訳、debbugs
2005 年 11 月	8	DWN 翻訳フロー、 stateoverride
2005 年 12 月	8	忘年会
2006 年 1 月	8	policy、 Debian 勉強会でやりたいこと
2006 年 2 月	7	policy、 multimedia
2006 年 3 月	30	OSC: debian 勉強会、sid
2006 年 4 月	15	policy、 LaTeX
2006 年 5 月	6	mexico
2006 年 6 月	16	debconf、 cowdancer
2006 年 7 月	40	OSC-Do: MacBook Debian
2006 年 8 月	17	13 執念
2006 年 9 月	12	翻訳、 Debian-specific、 oprofile
2006 年 10 月	23	network、 i18n 会議、Flash、 apt
2006 年 11 月	20	関西開催: bug、 sid、packaging
2006 年 12 月	14	忘年会

表 5　東京エリア Debian 勉強会参加人数 (2007-2008 年)

	参加人数	内容
2007 年 1 月	15	一年を企画する
2007 年 2 月	13	dbs, dpatch
2007 年 3 月	80	OSC 仮想化
2007 年 4 月	19	quilt, darcs, git
2007 年 5 月	23	etch, pbuilder, superh
2007 年 6 月	4	エジンバラ開催: Debconf7 実況中継
2007 年 7 月	18	Debconf7 参加報告
2007 年 8 月	25	cdn.debian.or.jp
2007 年 9 月	14	exim
2007 年 10 月	30	OSC Tokyo/Fall(CUPS)
2007 年 11 月	19	live-helper, tomoyo linux kernel patch, server
2007 年 12 月	11	忘年会
2008 年 1 月	23	一年を企画する
2008 年 2 月 29+3 月 1 日	36	OSC
2008 年 3 月	37	データだけのパッケージ、ライセンス
2008 年 4 月	17	バイナリパッケージ
2008 年 5 月	20	複数のバイナリパッケージ
2008 年 6 月	10	debhelper
2008 年 7 月	17	Linux kernel patch / module パッケージ
2008 年 8 月	10	Debconf IRC 会議と Debian 温泉
2008 年 9 月	17	po4a,「Debian メンテナのお仕事」
2008 年 10 月	11?	OSC Tokyo/Fall
2008 年 11 月	17	「その場で勉強会資料を作成しちゃえ」 Debian を使った LaTeX 原稿作成合宿
2008 年 12 月	12	忘年会

26 関西 Debian 勉強会 2009 年度各種イベント開催実績と総括

倉敷・佐々木・野方

26.1 運営状況

関西は運営に関わっている人に学生が多いので、いろいろ無理をお願いする場面も多かったような気がします。

26.1.1 勉強会全体

今年度途中 (7 月) より、運営担当が山下尊也から倉敷・佐々木・野方の三名体制に交代しました。これは山下の身辺が多忙になり身動きがとれないという理由からです。幸い、以前より分担に向け運営の見直しを進めていたこともあり、大きな混乱もなく継続することができました。

年度当初、ライブ中継に若干盛り上がりを見せましたが、その後、うまく継続できませんでした。問題としては IP アンリーチャブルな会場をメインにしていることと、中継の実作業を担っていた人が運営側にシフトし、余力を回せなくなっていることが原因と思われます。

5 月には神戸市を中心とした関西地域の新型インフルエンザ流行により、勉強会を中止する出来事がありました。社会的な要因により勉強会開催の判断を迫られる状況は初めてでしたが、こういう事は二度とあって欲しくないですね。

9 月は京都リサーチパークにお邪魔して勉強会初の京都で開催しました。会場を変えると、いつもとは違う参加者も増えるので、たまに場所を変えるのもよいのではと思いました。

講師については現状、固定化している中、継続して常連参加者への講師依頼をするほかに、 DMC を取り入れたり、LT 発表も可能な参加者自己紹介の常設などをおこないました。

LT 発表可能な参加者自己紹介は、話題にバリエーションが加わったなど興味深いこともあった反面、年度後半は関西の勉強会参加者も参加している Open Street Map にトピックを持っていかれてしまった感もあり、うまくバランスを取る必要がありそうです。

また、今年度は佐々木、山下の 2 名が Package Maintainer として Debian の New queue に新しくパッケージを送り込みました。来年もこの流れを維持できればと思います。

26.1.2 扱ったテーマ

勉強会の内容としては、パッケージ開発自体に加えて、 Debian の体制にまつわる話 (gpg や mentors など) や、周辺ツールの利用 (bash や reportbug や gdb など) をとりあげました。来年度のテーマについては、年末年始に相談をする予定をしています。

翻訳関連では、東京での流れに乗り DDTSS のハンズオン実習をしましたが、予想外に反応がありました。もともと需要があったのか、実習したことで身近になったのか、はよくわかりませんが...。

26.1.3 イベント関連

例年通り、夏のオープンソースカンファレンス Kansai@Kyoto(OSC) と、秋の関西オープンフォーラム (KOF) に出展しました。

セッションでは、 OSC では大浦さんによる Debian GNU/kFreeBSD について、 KOF では矢吹さんに DD になるまでの軌跡をお話してもらいました。矢吹さんは、関西 Debian 勉強会立ち上げの立役者なので、できれば勉強会にも来て欲しいところですが、最近は、なかなかご多忙で難しいとのことです。

また、四国ではじまったオープンフォース勉強会 [*62]と、岡山でのオープンセミナー@岡山 [*63] に、野方が参加して Debian Live やノウハウの紹介などを行いました。

26.2 開催実績

関西 Debian 勉強会の出席状況を確認してみましょう。グラフで見ると図 49 になります。表で見ると表 8 です。

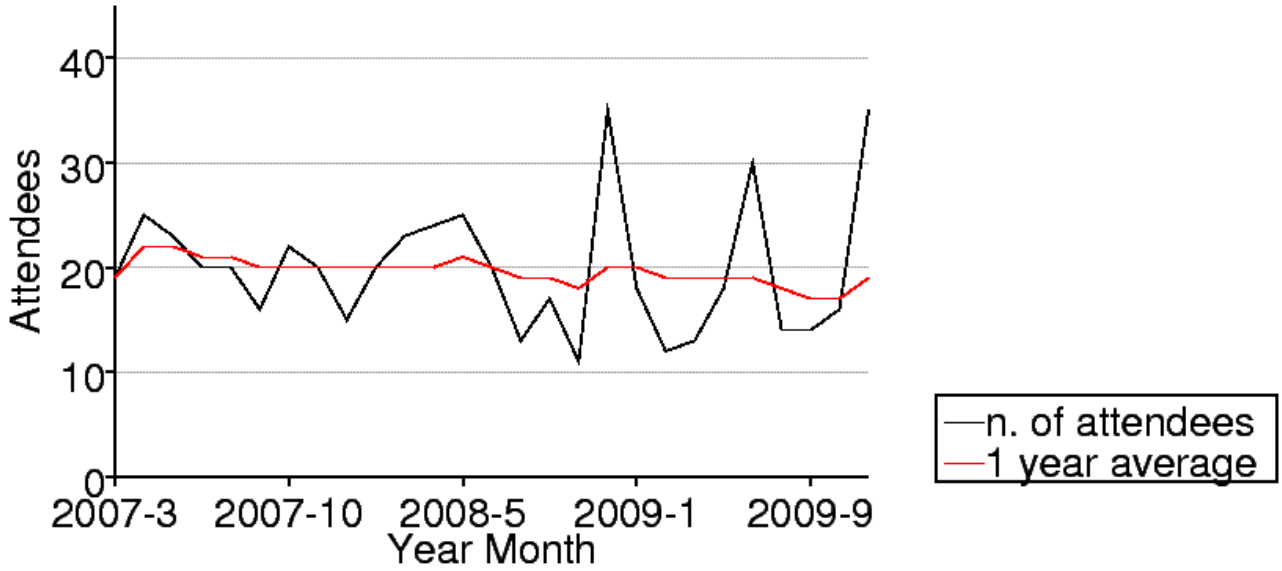


図 49　関西の参加人数推移

[*62] http://openforce.project2108.com/
[*63] http://openseminar.okaya.ma/

表 6　関西 Debian 勉強会参加人数 (2007 年)

	参加人数	内容
2007 年 3 月	19	開催にあたり
2007 年 4 月	25	goodbye、youtube、プロジェクトトラッカー
2007 年 6 月	23	社会契約、テーマ、debian/rules、bugreport
2007 年 7 月	20 前後	OSC-Kansai
2007 年 8 月	20	Inkscape、patch、dpatch
2007 年 9 月	16	ライブラリ、翻訳、debtorrent
2007 年 10 月	22	日本語入力、SPAM フィルタ
2007 年 11 月	20 前後	KOF
2007 年 12 月	15	忘年会、iPod touch

表 7　関西 Debian 勉強会参加人数 (2008 年)

	参加人数	内容
2008 年 2 月	20	PC Cluster, GIS, TEX
2008 年 3 月	23	bug report, developer corner, GPG
2008 年 4 月	24	coLinux, Debian GNU/kFreeBSD, sid
2008 年 5 月	25	ipv6, emacs, ustream.tv
2008 年 6 月	20	pbuilder, hotplug, ssl
2008 年 8 月	13	coLinux
2008 年 9 月	17	debian mentors, ubiquity, DFSG
2008 年 10 月	11	cdbs,cdn.debian.or.jp
2008 年 11 月	35	KOF
2008 年 12 月	?	TeX 資料作成ハンズオン

表 8　関西 Debian 勉強会参加人数 (2009 年)

	参加人数	内容
2009 年 1 月	18	DMCK, LT
2009 年 3 月	12	Git
2009 年 4 月	13	Installing sid, Mancoosi, keysign
2009 年 6 月	18	Debian Live, bash
2009 年 7 月	30?	OSC2009Kansai
2009 年 8 月	14	DDTSS, lintian
2009 年 9 月	14	reportbug, debian mentors
2009 年 10 月	16	gdb, packaging
2009 年 11 月	35	KOF2009
2009 年 12 月	??	GPS program, OpenStreetMap

27 2009 年を振り返ってみる

上川 純一

27.1 最近のトレンドと今後の推移

最近どんなことがあって、これからどういうことがあるでしょうか。みんなで予想してみましょう。

2007	2008	2009	2010	2011
VT・AMD-V(仮想化技術)が普及(ML115!)、玄箱(ARM)、OpenBlocks(PPC?iPhone 登場、HSDPA 月額 5000 円くらいに、google mobile、VISTA リリース、Leopard リリース、GPL3.0、メモリ 2G がコモディティーに、SparcT2 がオープン、ニコニコ動画、	python 3.0 ruby 1.9 wine 1.0, wine64 登場 RoR 2.0 登場で普及に 4 コア・64bit の CPU がデスクトップに普及、Core2Quad 値下げ。 ニコニコ動画 1000 万ユーザ突破、初音ミクブームに 地デジ関連の PC 製品の普及 勉強会の普及 (楽天とか) 公衆無線 LAN (wireless gate) 携帯電話の売上が落ちる、iPhone, Android 登場、emobile 100 円 PC 抱きあわせ (eeePC, Dell mini9) Zaurus 販売終了。 Chumby 発売。 サーバの仮想化 ESXi・シンクライアント MacBook Air 発売、無線 802.11n が実機に SystemZ10 発表 世界経済の崩壊 (IT 投資緊縮財政、職を失う人が増加) FreeBSD 7 (malloc, ZFS ?) Debian 次世代育成計画始動 Debian Maintainer 制度始動 セキュリティー関連 (OpenSSL 事件、DNS 事件) クラウド関連が流行? Nintendo DSi	政権交代, スパコン事業仕分け, 円高 Windows7,Snow Leopard 発売 Netwalker 発売 MacBook から IEEE1394 が消えた。 メモリが DDR3 に移行中, メモリ高騰 マジコン販売取り締まり ラブプラス,OSS を使ったエロゲー登場 (OpenCV),AR, セカイカメラ JLS で Linus 来て大騒ぎ DD,2 世誕生 デジタルサイネージ Google Voice,Wave,Chrome,Chrome OS,Go, 日本語入力, 徒歩ナビ CouchDB Twitter,*なうブーム Eye-Fi,Kindle2,DS LL,PSP-GO,POKEN Cell 終了のお知らせ tile window manager boom ? Lenny リリース Debian 結婚ブーム デスクトップ、4 コア、8GB ノートパソコン、2 コア、4GB Linux が標準インストールの PC。(Dell) SSD の値段と容量がこなれる (まあまあ) HDD がなくなる?高くなる?(ならず) SSD 特化した FS が出てきた ipv6 使えるようになってる (来年) DL 禁止法? torrent に逆風?	Debian OAuth サービス開始のお知らせ SolidICE 使った Debian VDI サービス開始のお知らせ Chrome OS,Android 統合のお知らせ Willcom 終了のお知らせ Netbook, クラウド,Ameba なう終了のお知らせ Debian Cloud リリースのお知らせ 10GbE,SSD 普及 新 iPhone リリース 自民復活 SIM ロックフリー延期のお知らせ 新 Android 端末 (日本以外) 次世代用 FS:btrfs,NILFS Lenny and Half リリース squeeze リリース遅延,kFreeBSD,SH4 オフィシャルアーキテクチャに ToyStory3 リリース 消費税上昇に伴う繁忙期 クラウドにより、単純なホスティング業者がつづかない? 一部は自社でもつようになる? USB 3.0 搭載、wireless USB vs Bluetooth ? 組み込み CPU は Atom に統一?Arm は残ってる? ruby 2.0 リリース?Perl6 リリース?	Windows ドライバシグニチャチェックがなくなる IEEE1394 終了のお知らせ LTE が徐々に普及 地デジ延期のお知らせ Debian 11 日本開催 Google に統合 (オフィスソフト、グループウェア、メール、ファイルサーバ止めようぜ運動) Scala のエンタープライズ利用 Windows11 リリース C/S を意識せずにアプリ開発できる環境 (コンパイラが自動判断)

28 Debian Trivia Quiz

上川 純一

ところで、みなさん Debian 関連の話題においついていますか? Debian 関連の話題はメーリングリストをよんでいると追跡できます。ただよんでいるだけでははりあいがないので、理解度のテストをします。特に一人だけでは意味がわからないところもあるかも知れません。みんなで一緒に読んでみましょう。

28.1 第 63 回勉強会

第 63 回勉強会の出題範囲は debian-devel-announce@lists.debian.org に投稿された内容と Debian Project News からです。

問題 1. DPL 2010 に立候補しているのは誰?

- ☐ A Yasuhiro Araki
- ☐ B Charles Plessy
- ☐ C Kurt Roeckx

問題 2. Debian policy 3.8.4.0 で追加された項目は?

- ☐ A /sys と /selinux の FHS に対する例外ポリシー
- ☐ B kFreeBSD と Linux を共存するポリシー
- ☐ C スーパ牛さんパワーに関するポリシー

問題 3. 最近ハードウェアトラブルがあったサーバは?

- ☐ A rie.debian.org
- ☐ B ries.debian.org
- ☐ C rise.debian.org

問題 4. buildd.debian.org のあるサーバが移動しました。どこに移動したでしょう。

- ☐ A peri.debian.org
- ☐ B cimarosa.debian.org
- ☐ C grieg.debian.org

問題 5. squeeze のインストーラで追加された機能は

- ☐ A Recommends をインストールするようにします
- ☐ B インストーラ上でパッケージがビルドできます
- ☐ C クロスアーキテクチャインストール機能を追加しました

問題 6. 新しく mips 用 porterbox が追加されました。 CPU コア数はいくつでしょうか。

□ **A 64**

□ **B 32**

□ **C 16**

28.2 第 64 回勉強会

第 64 回勉強会の出題範囲は `debian-devel-announce@lists.debian.org` **に投稿された内容と Debian Project News からです。**

問題 7. DPL 2010 になったのはだれでしょうか?

□ **A Yasuhiro Araki**

□ **B Stefano Zacchiroli**

□ **C Steve McIntyre**

問題 8. DMUP が 1.1.2 にアップデートされました。何が追加されたでしょうか。

□ **A Yasuhiro Araki**

□ **B Stefano Zacchiroli**

□ **C Steve McIntyre**

29 Debian Trivia Quiz 問題回答

上川 純一

Debian Trivia Quiz の問題回答です。あなたは何問わかりましたか?

1. B
2. A
3. B
4. C
5. A
6. C
7. B
8. B

『あんどきゅめんてっど でびあん』について

本書は、東京および関西周辺で毎月行なわれている『東京エリア Debian 勉強会』および『関西エリア Debian 勉強会』で使用された資料・小ネタ・必殺技などを一冊にまとめたものです。収録範囲は東京エリアは勉強会第 59 回から第 64 回、関西エリアは第 30 回から第 35 回まで。内容は無保証、つっこみなどがあれば勉強会にて。

あんどきゅめんてっど でびあん 2010 年夏号

2010 年 8 月 14 日　　初版第 1 刷発行

東京エリア Debian 勉強会/関西エリア Debian 勉強会（ 編集・印刷・発行）