東京エリア デビアン 勉強会

Debian勉強会幹事 上川純一

2009 年 4 月 18 日

1 Introduction

上川 純一

今月の Debian 勉強会へようこそ。これから Debian の世界にあし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。

2009 年の計画は仮です。

1. 新年の企画 (アンサンブル荻窪開催)
2. OSC Tokyo
3. VAIO P インストール記録、カーネル読書会 ディストリビューション大集合 (小林さん)(東京大学?)
4. Git Handson (岩松)(あんさんぶる荻窪?)
5. 家 Debian サーバ vs 職場のネットワーク (千代田区都立図書館?[*1])
6. Asterisk (東京大学?)
7. スペインにて開催
8. Debconf 報告会
9. OSC Fall?
10. udev + HAL(岩松さん)
11. 3D graphics 開発（藤沢さん）
12. Debian サーバ +VMware + 各種 OS、他の仮想化ツール (vserver etc.)、忘年会

会場候補としては下記があります:

- 大学
- 恵比寿 SGI ホール
- Google オフィス
- 公民館 (あんさんぶる荻窪等)
- 都立会議室 (無線 LAN)
- 健保の施設

[*1] http://www.library.chiyoda.tokyo.jp/

デビアン勉強会

目次

2 事前課題

上川 純一

事前課題は:

1. 各自の「私の Debian 開発ワークフロー」を紹介してください。

この課題に対して提出いただいた内容は以下です。

2.1 上川純一

2.1.1 私の Debian ワークフロー

- メールでバグレポートを受け取る
- コードを直す・パッチを git am で適用
- pdebuild-normal スクリプトを実行、cowbuilder –update, cowbuilder –build が実行され、一連のインストール・実行テストスクリプトが実行される。成功したら pending ディレクトリにパッケージが移動される。
- pending ディレクトリを確認、debsign で署名、dput でアップロード

2.1.2 こう改善したい

全アーキテクチャでのビルドとテストを自動化したい。

2.2 まえだこうへい

2.2.1 私の Debian ワークフロー

ganttproject を初めて ITP してから止まったまま。

2.2.2 こう改善したい

家庭と仕事に影響されずにパッケージメンテナンスできるようにしていきたい。

2.3 小川伸一郎

2.3.1 私の作業環境について

会社では Ubuntu 8.04.1 Desktop をインストールしたデスクトップ PC で，家では Ubuntu 8.10 をインストールした Thinkpad X61 を使って，開発や日々の業務などをこなしています．全然 Debian じゃないのですが，Ruby on Rails なので，Ruby の Version があわないので，Ubuntu 使っています．

GW 中に Thinkpad に Lenny 入れる予定です．会社のサーバ群も，Debian にしたいなと，いろいろ模索中です．

2.4 山本浩之

2.4.1 私の Debian ワークフロー

パッケージ化したいソフトウェアを見つけたら、まず自分自身用の野良パッケージを作り、試します。次に大雑把にライセンスを確認し、良さげなら、自分に喝を入れるため ITP します (笑)。それからコードなど、技術的な検討に入ります (ここで挫折したものもいくつあるのやら…)。さらにコピーライトやライセンスの精査をし、debian/copyright を完成させます。次に私にとってとても難関の (笑) 英語のドキュメントをつけて、pbuilder でビルドします。最後に mentors.debian.net へのアップロードと mentors@org ML、および debian-develop@jp ML にメールを投げてスポンサー探しをします。以上。

2.4.2 こう改善したい

みんなが使っている文字コードや locale を UTF-8 に統一したい。

2.5 やまだたくま

2.5.1 私の Debian ワークフロー

1. DDTSS (ja) で Pending review の項目を順番に選びます。
2. doc/(パッケージ名) フォルダを作成し、原文 (英語) と日本語訳のコピーのテキストファイルを作成します。
3. 翻訳ソフトで英日翻訳を実行します。
4. 用語とその日本語訳を確認して、対訳リストを作ります。
5. 文章の内容を確認するため、オンラインマニュアルやパッケージ関連ファイルを参照します。
6. 使用例や用語 (訳語) の使用頻度を調査して、訳語を選びます。
7. 原文を先頭から順番に手動で再翻訳します。
8. Debian JP の文書作成／翻訳ルールを守っているか確認します。
9. debian-doc ML へ査読依頼します。
10. 査読完了後に DDTSS (ja) へ登録します。

- 作業は、複数の場所、複数の PC で行なっています。
- 作業ファイルは、Mercurial で同期管理しています。
- オンラインマニュアルやパッケージのファイルは、VMware 上の Debian (sid) または ssh で Debian サーバに接続して確認作業しています。

2.5.2 こう改善したい

- 対訳表の収録語を増やして、訳語の確認時間を短縮したい。
- Debian JP の文書作成／翻訳ルールの確認作業を自動化したい。

2.6 中尾圭佐

2.6.1 私の Debian ワークフロー

私は Debian に貢献しているわけではないで、Debian 開発の開発工程はもっていませんので、普段の作業工程を記述します。

- まず、必要とされている機能を見付けます。見つけ方は、ボスから指示があったり、手作業でやっていていらついたとき、簡単な作業でも毎日やっていることに気付いたとき等によく見付かります。
- どうやったら、その機能が実現できるかを考えます。個人的にこの段階が一番楽しいです。
- この機能が本当に必要か考えます。
- 必要ならば、本当に実装して良いか考えます。私がいる職場は、放射線が出たり、100kV の高電圧がかかっていたりするので、放射線管理上問題がないか安全上問題がないか検討します。
- 一番楽しい段階が終ったこともあり、本当に私が実装すべきか考えます。
- 私が実装すべきという結論が出た場合、ぶつぶつ文句を言いながら、実装します。
- 時々Debug します。テストファーストとか、自動化はできていませんが、ユニットテストを行います。
- だいたいできたら、職場のみんなに見せびらかします。そうすると色々コメントが出てくるので、それに対応します。運用が大変なら、修正します。

2.6.2 こう改善したい

改善したいことは、私以外の誰かが、書いたコードをメンテナンスできるようにすることです。そのためには:

- バージョン管理システムを導入する
- ユニットテストの自動化を徹底する
- ドキュメント、資料、コメントをちゃんと残す

が必要だと考えています。

私の職場では、コードの所有権のようなものが心理的に存在しています。バージョン管理システムを導入する事で所有者のコードを残しつつ、所有者以外の人がコードを取得でき、また修正する事ができます。これによりコードの所有権の意味を曖昧にすることができあます。

ユニットテストの自動化を実現する事で、コードの所有者以外の人がコードを修正した場合、その修正が他に影響を及ぼさない事を確認する事ができます。このことは他人のコードを修正すことの心理的障壁を下げることにつながります。

さらにドキュメントを残す事で、トラブルの時、修正する時に非常に有用な情報を提供する事ができます。時が進み、開発時何を考えていたか、当時どのような必要

があってこのような実装したか、このような記録は、どこを捨ててどこを残すかという判断に必要な情報になります。

私がいなくなっても、私が書いたコードやシステムがちゃんと維持でき、状況に応じてメンテナンスできるように、以上のことを改善したいと考えています。

2.7 あけど

2.7.1 私の Debian ワークフロー

Debian 上で作業することが少ないなと思っています。せいぜい管理しているサーバのファイアウォールルールを手直しするくらいなので、手元のメインマシンが Mac OS X(10.5.6) ということもあり、Debian なデスクトップ環境を殆ど使ってません。Debian のデスクトップ環境は Debian 勉強会の事前課題に使う程度なので (いろんな環境に慣れるという意味で) もっと使う様にするには Debian 勉強で標準的な環境の emacs を使うのがいいかなと思います。

2.7.2 こう改善したい

上記を踏まえて、勉強がてら emacs を使う様にしてみます。

2.8 藤沢理聡

2.8.1 私の Debian ワークフロー

パッケージをメンテナンスしたり、といった Debian への貢献はまったくできていないのですが、Debian 上でスクリプトを書くことは結構あります。

フローにすると、

1. 仕事とかしてて、こういうのがあったらなあ、と思う
2. 思いついたことを実現する仕組みを考えてみる
3. 実現できそうなら、実際に使用する環境やユーザの範囲を考える
4. 具体的に作るもののイメージができたら、エディタを起動する
5. 適当に書いて、とりあえず動くものを作る
6. 規模が小さければ、ホワイトボックステストをする
7. 自分以外に使う人がいれば、とりあえず試してもらう
8. 動くものを見て、新たに出てきた要望に答える
9. 飽きたら開発終了

Debian である必然性のないワークフローになりました。この情報は一体何の役に立つんだろう、と自問自答。

2.8.2 こう改善したい

たいてい 2 つ目のステップで「やっぱいらないか」と思って終了してしまうのを改善したい。

ワークフローとして改善すべきは、

- 他人と共同で開発することに向いていない
- 誰かが作ったものの改良や、ある程度作ってから誰かに引き継ぎ、に向いていない

ことかな、と自分では思っています。

2.9 日比野

2.9.1 私の Debian ワークフロー

Debian のワークフローかどうかわかりませんが、Debian も利用している私の会社でのワークフローを紹介します。

1. 自動化できそうな機能や Debian 化することでインストールが楽になりそうな機能を見つける - たとえば
   - バッチジョブのログを取りながら経過や結果を監視し、問題があるようならアラームをいろんな方法で投げる perl script
   - 複数のアーキテクチャや Debian のバージョンに対して内製のパッケージを build する cowbuilder や pbuilder の wrapper
   - 社内の Debian で運用するサーバーのインストーラースクリプト
2. 利用者になりうる人や他の開発者の意見を取り入れながら、機能をプログラム化する
3. まとまった規模になったら Debian 化を行なう
4. 適当な区切りで履歴管理システムにタグを打ち、内製の package spool にリリースする

2.9.2 こう改善したい

内製パッケージのビルドや Debian のインストール、ネットワークの設定の自動化をすすめたい。

3 最近の Debian 関連のミーティング報告

上川 純一

3.1 東京エリア Debian 勉強会 50 回目報告

東京エリア Debian 勉強会報告。2009 年 3 月 21 日土曜日に東京エリア Debian 勉強会の第 50 回を東京大学にて開催しました。

今回の参加者は岩松, あけど, 前田, キタハラ, たかはし, キタムラ, じつかた, やまだたくま, 日比野, 藤澤とおる, よしの, 虎, こたに,matsuu, かい, たなか, ささき (uwabami), まとはら, こう,John, よしだ@板橋, 藤澤りそう, 上川 x2 の 24 名でした。

今回事前課題はなかったので参加者の自己紹介をかねて今回に期待することを紹介してもらいました。関西や新潟から参加している人もいてビックリです。(内容については発表資料参照)

最初に久しぶりにクイズをしました。9 問の簡単な常識問題でした。やまだたくまさんが全問正解しました。

最近のイベントの紹介をしました。

藤澤とおるさんが gxp と MC-MPI のパッケージ作成について発表しました。パッケージになれてない人がパッケージの作成をしていろいろひっかかった内容を発表しているのをみながらみんなでにやにや。

日比野さんが common lisp の話をしました。Debian のパッケージでなぜ common lisp のライブラリの取扱いがこうなっているのかということを common lisp のマクロの仕組みから解き明かすセッションでした。素晴らしい。とりあえず emacs ユーザは sbcl と slime のインストールからはじめればよいようです。

最後に今回の期待についておさらいをして、その後に次回の内容を検討しました。次回は John が clozurecl のパッケージ作成したよ、藤澤さんがパッケージを Debian official にするまでの道、やまねさんがワークフローをさらすので「俺の開発効率をあげてください」というセッションの三本立ての予定です。

宴会は東京大学の赤門前の宴会場「鉄板焼き 郷」で。

4 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。

今回の出題範囲は `debian-devel-announce@lists.deban.org` に投稿された内容と Debian Project News からです。

問題 1. 4 月 9 日にアップデートされた etch のバージョンは?

☐ A 4.0r8
☐ B 4.0beta8
☐ C etch-a-sketch

問題 2. 4 月 11 日にアップデートされた lenny のバージョンは?

☐ A 5.0r1
☐ B 5.0.1
☐ C lenny++

問題 3. Debian.org DPL になったのは？

☐ A Stefano Zacchiroli
☐ B Steve McIntyre
☐ C Nobuhiro Iwamatsu

問題 4. Debian JP Leader になったのは?

☐ A Kei Hibino
☐ B Hiroyuki Yamamoto
☐ C Nobuhiro Iwamatsu

問題 5. Debian で新しく追加されたアーキテクチャは？

☐ A GNU/kFreeBSD i386/amd64
☐ B GNU/kMinix-3.0 i386/amd64
☐ C GNU/kOpenDarwin i386/amd64

5 Java ポリシーを読んでみた

まえだこうへい

5.1 Java ポリシーを読むハメになった経緯

初っ端からブチまけると、実は Java ポリシーなんて読むつもりは毛頭なかったのです。なぜなら私は Java なんて大嫌いだからです。長たらしいクラス名やメソッド名なんてみると、身の毛もよだちます。ちゃんと 80 バイトの幅に収めろよ、って感じです。(わら

冗談[*2]はさておき、なぜ読むことになったかと言うと、実は別の目的がありました。典型的な手段が目的になってしまった例です。ことの発端は、友人と別件での活動で進捗管理が必要になったためです。某所の PC は admin 権限が与えられておらず、好き勝手にソフトウェアを入れられません。ですが、プロジェクト管理用に GanttProject[*3]はライセンス使用料無しで導入できるので、これで管理するか、ということにしました。オープンソースソフトウェアなので、きっと Debian にもパッケージがあるに違いない、と。WBS を作り、自宅で Debian に GanttProject を導入しようとしたら、パッケージが無いではありませんか。使えないとせっかく作った WBS も更新できず困ります。そこで仕方ないので、ITP してみることにしました。ちょうど、まだ BTS へ投げたことも、ITP もしたことがなく、ある意味ちょうど良いきっかけなので ITP してみました[*4]。

で、Debian Hack Cafe で、岩松さんから「どうせなら Java Policy 読んで、勉強会のネタにする」と言われた訳です。特に断る理由もなかったので、やることにしてみました。が、ちょうど今回の Debian 勉強会と、KVM の記事[*5]の公開が時期的に重なってしまったため、締切り直前で勉強会資料の作成をやっています。終わるのかな。当日の事前配布資料に、このネタが掲載されていればきっと間に合ったのでしょう。

ちなみに、肝心の当初の目的ですが、完全に置いてけぼりを喰らっています。Java ポリシーの翻訳はもう少しで終わるのですが、ITP した GanttProject の deb パッケージ化は未着手、GanttProject で進捗管理する対象は友人が中心に進めているもの、進捗は遅れ気味になっています。巻き返ししなければ。

5.2 さて、今回の本題。

前置きが長くなりましたが、今回の本題は、Java ポリシーについて調査してみたよ、でした。要約によれば、背景説明、ポリシーの内容、議論すべき問題点、Java パッケージメンテナ向けのアドバイスであり、Java 仮想マシン、Java コンパイラ、Java プログラム、Java ライブラリをターゲットにしています。以下、各章ごとに要点をまとめてみました。

[*2] 嫌いなのは本当なんですが。
[*3] http://www.ganttproject.biz/
[*4] http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=436792
[*5] http://www.atmarkit.co.jp/flinux/rensai/kvm02/kvm02a.html

5.3 1 章 背景説明

要点は 2 つです。

- 特定の議題をより詳細に決める場合、Debian ポリシーにはサブポリシーとして扱われる。
- コメントなどをするには、java-common パッケージチームや、Debian Java メーリングリストに投稿しる。

要は、Java には固有の事情があるので、Java ポリシーというサブポリシーで扱っているよ、ということです。

5.4 2 章 ポリシー

Java パッケージに対するポリシーをまとめています。そのポリシーとは以下のものです。

- コンパイラ、仮想マシン、ランタイム用の仮想パッケージが作られます。
- 全 Java コードは、Java バイトコードとして、全アーキテクチャ向けに配布しなければなりません。
- 開発者向け (ライブラリ) とエンドユーザ向け (プログラム) の 2 つのカテゴリに分類されます。

5.4.1 仮想マシン

Java 仮想マシンをパッケージ化する際のポリシーです。パッケージ名、依存関係、コマンド名を規定してます。

- java-virtual-machine パッケージを提供し、java-common パッケージに依存していなければなりません。
- ランタイム環境を配布することもできます。
- Java2 に準拠したランタイムをパッケージでは、java2-runtime を提供すべきです。[*6]
- Sun Java プログラムと互換性のあるコマンドラインであれば、/etc/alternatives/java という名前にするべきです。
- 必要なランタイム環境を、あらかじめ CLASSPATH に設定しておくべきです。
- JDK のようにコンパイラや仮想マシンのソースコードを提供する場合は、コンパイラパッケージを xxxx-dev と命名しなさい。
- Java クラスの実装は、ネイティブ言語を用いた処理系で、実行時にロードされる動的ライブラリとしてコンパイルされ、配置されています。仮想マシンがネイティブコードをサポートする場合は、動的ライブラリの検索パスに/usr/lib/jni ディレクトリを設定しないといけません。

5.4.2 コンパイラ

Java コンパイラに関するポリシーです。仮想マシンの時と基本的には同じで、3 点挙げられています。

- java2-compiler パッケージを提供し、java-common パッケージに依存していなければなりません。
- Sun JDK の javac と互換性があるなら、/etc/alternatives/javac という名前にすべきです。
- コンパイルに必要な java コアクラスを CLASSPATH に設定しておくべきです。

5.4.3 Java プログラム

Java のプログラムに関するポリシーです。

- /usr/bin 以下に配置し、実行可能でなければなりません。
- プログラムは binfmt_misc を使用した Java クラスか、あるいはラッパーでも構いません。

[*6] Java1.1 の場合は、java1-runtime なのですが、今時 Java1.1 はないと思われるので省略。

- 特定の環境変数[*7]がなくても、実行できなければなりません。
- 実行ファイルに関するポリシールールに従う必要があります。
- プログラム自体が補助クラスの場合は、/usr/share/java ディレクトリ配下に jar ファイルとして配置しなければなりません。
- プログラム自体は仮想マシン (java-virtual-machine) とランタイム環境 (java2-runtime) に依存していなければなりません。
- プログラムの名前規則は無く、ユーザ視点で一般的なプログラムです。

5.4.4 Java ライブラリ

Java ライブラリに関するポリシーです。Java ならではで特徴的なことは、開発者向け[*8]やユーザ向けのバージョンには分かれていないという点です。

- Java では開発者向けとユーザ向けとでライブラリのバージョンを分けません。意味がないからです。
- Java ライブラリパッケージは、libXXX[version]-java という形式の名前でなければなりません。
- version 部分はオプションで、必要な部分だけを含むべきです。
- version 部分は名前の衝突を避けるようにすべきです。
- XXX は実際のパッケージ名です。
- jar アーカイブのクラスは、/usr/share/java 配下に配置しなければなりません。
- 名前は package[-extraname]-fullversion.jar の形式です。
- extraname はオプションで、パッケージで提供されている jar とは別に分けるためにパッケージ内部で使われます。
- fullversion は jar ファイルのバージョンです。パッケージとは同一ではない場合があります。
- Java ライブラリはランタイム環境には依存しなければなりませんが、仮想マシンには依存すべきではありません。

など。

5.4.5 main, contrib, non-free

main, contrib, non-free に分類するためのポリシーです。

- ソースパッケージが non-free なツールでしか (正しく) コンパイルできないなら[*9]、main にすることはできません。パッケージ自体が自由であるなら、contrib に入れなければなりません。
- バイナリパッケージが non-free の仮想マシンだけで動くことができるなら[*10]、main にすることはできません。 パッケージ自体が自由であるなら、contrib にしなければなりません。

5.5 議論すべき問題

現在、8 つほどの問題が取り上げられています。いずれも議論の余地があるとのことです。

- リポジトリの名前と存在について。最新バージョンでは削除されています。
- /usr/share/java におけるシンボリックリンクを C 言語のライブラリのようにスクリプトで置き換えるべきか否か。
- コアクラスの必要性について。

[*7] 例えば、CLASSPATH など
[*8] -dev がパッケージ名につくもの。
[*9] 自由な Java コンパイラは、guavac と、gcj と jikes だけのようです
[*10] GNU-Classpath には、自由なバージョンがリストされてます。

- Sun のコミュニティーソースライセンスは、Debian では使えるのか？使う場合はどうすれば良いのか
  item すべての jar ファイルは適切な CLASSPATH ドキュメントが必要です。

以下、省略。

5.6 Java パッケージ作成者向けのアドバイス

Java パッケージ作成者向けのアドバイス、とはなっているものの、実際のところは、足りないツールを作ってくれだとか、他の言語向けパッケージ作成では自動で行っている部分を手動でやれ、といった話になっています。しかもひどいことに、「これはアドバイスに過ぎず、ポリシーの一部ではない」と言っています。

- debian/control のすべての依存関係を手動で管理しなければなりません。Debian の開発ツールでは Java のプログラムを検出することができません。
- だから、Perl のように dh_java プログラムを作ってくれるボランティアは大歓迎とのこと。
- debian/rules の中の、dh_strip や dh_shlibdeps のような Java には無意味なコマンドを無効にしましょう。

などです。

5.7 まとめ

実は、ポリシーマニュアルをちゃんと読んだことがまだないので、通常のポリシーに比べた場合の、Java ポリシーの特徴がどうなのかがよく分かりません。ただ、自由なコンパイラが 3 つあるとか、main に入れるための制約とか、命名規約やら、依存関係についてのポリシーがあることを知りました。これを元に途中で止まっていた ganttproject の deb パッケージ化を進めます。またプロセスは、本来の目的を達成するために早く進めなきゃです。ついでに、今回訳した文章は、取り合えず debian-doc に投げて査読してもらおうと思います。

5.8 今月のヨメ八苦

最近は、毎週水曜の Hack Cafe は認めてもらえるようになりました。まあ、本人もその時間にジムに行けるからという理由もありますが。しかし、IRC 会議や Debian 勉強会以外で、Debian 絡みの作業したり、外にでかけると、「また Debian？」と怪訝な顔をされます…。なかなか思うとおりにはならんものです。

6 ocaml に入門してみた

上川純一

6.1 ocaml 入門の動機

ocaml はフランス界隈で流行っている言語のようです。advi や unison などのかゆいところに手が届く感じのソフトウェアが ocaml で実装されていて、気にはなっていました。ocaml は素晴らしい計算機理論を活用して最新の何かが実装されているという噂は聞いているのですが、いまいち何がどうすごいのかよくわからないのでものは試しと、試してみました。

6.2 ocaml を使うために

Debian GNU/Linux には ocaml 関連のパッケージが多数入っています。ためしに検索してみると 224 パッケージありました。

```
$ apt-cache search ocaml | wc -l
224
```

ocaml にはインタラクティブに使うためのインタプリタと、バイナリを生成してくれるコンパイラの両方があります。ocaml をとりあえず一通り使うためには ocaml のインタプリタとコンパイラがあればよさそうです。ocaml のネイティブコード用のコンパイラは ocaml-native-compilers パッケージのようです。ocaml のインタープリタは ocaml-interp パッケージのようです。

emacs ユーザとしては、emacs 用のモードも必要です。emacs のモードでそのものずばりの ocaml-mode というものもあるのですが、 tuareg というのがよさそうなので、それをインストールしてみました。

```
# apt-get install tuareg-mode ocaml-native-compilers ocaml-interp ocaml
```

6.3 インタラクティブに使ってみる

インタラクティブに ocaml を使って見ました。まず足し算をしてみます。

emacs から M-x tuareg-run-caml で ocaml のインタラクティブな端末 (ocaml 用語では「トップレベル」という) を起動しました。足し算をするには、 1 + 2 を入力し、文の終了は ';;' で表現します。

```
# 1 + 2 ;;
- : int = 3
```

int 型の 3 がかえってきました。めでたしめでたし。

6.4 関数を定義してみる

では、関数を定義してみます。

```
# let my_func a b = a + b ;;
val my_func : int -> int -> int = <fun>
# my_func 2 3;;
- : int = 5
```

int int をとって int をかえす関数が定義され、それを 2 と 3 で呼び出したら 5 が帰って来ました。

めでたしめでたし。

型をまったく宣言していないですが、勝手に定義されているあたりが型推論機能のようです。どうして型が推論できるかということを疑問に思われるかもしれませんが、「+」は実は int 専用の足し算です、float の場合は「+.」を使うそうです。これを知って失神しそうになりました。

6.5 コンパイルしてみる

とりあえずインタラクティブに使う方法はわかったので、コンパイルして使う方法を見てみましょう。ocaml は中間言語に変換する方法とネイティブコンパイルする方法の二つがあるようです。ocamlc は中間言語、ocamlopt はネイティブコンパイラのようです。ocamlc には二種類のバイナリが提供されており、ocamlc と ocamlc.opt があります。ややこしいですが、ocamlc.opt はネイティブコンパイラーでコンパイルした中間言語コンパイラーのようです。

```
$ time ocamlc fibo.ml -o a.1

real        0m0.038s
user        0m0.032s
sys        0m0.004s
$ time ocamlc.opt fibo.ml -o a.2

real        0m0.014s
user        0m0.012s
sys        0m0.000s
```

種類を表 1 にまとめてみました。

表 1　ocaml コンパイラの種類

	ネイティブコンパイルされた	中間言語にコンパイルされている
ネイティブコンパイル結果を出力	ocamlopt.opt	ocamlopt
中間言語を出力	ocamlc.opt	ocamlc

簡単な fibonacci のコードを書いてみて動作速度の雰囲気を感じてみました。バイトコードのオーバヘッドが感じられる結果になりました。

```
$ time ./byte-code
102334155

real        0m14.544s
user        0m14.541s
sys        0m0.000s

$ time a./native-code
102334155

real        0m2.933s
user        0m2.932s
sys        0m0.000s
$ time ./fibo.c 40
102334155

real        0m2.325s
user        0m2.192s
sys        0m0.040s
```

6.6 ocaml 関連のパッケージ化の課題

Debian パッケージにする際には、コンパイルする回数よりもインストールして実行される回数のほうが多いため、基本としてはネイティブコンパイルを選択します。ただし、ネイティブコンパイルできない CPU アーキテクチャもあるため、どのコンパイラを利用するのかを抽象化するレイヤーが必要です。

ocaml packaging policy によると、ネイティブコンパイラに対応しているのは amd64, i386, kfreebsd-i386, powerpc, sparc だけだそうです。重要なアーキテクチャで動きません。たとえば、arm、superh、mips などには対応してないようです。

ライブラリの扱いは特殊で、/usr/lib/ocaml/3.10.2/以下においてあるようです。ライブラリといっても、バイナリの libxxx.so 共有ライブラリがおいてあるわけではなく、.mli インタフェースファイルが大量においてあります。ocamlc -where で発見できるようです。

```
$ ocamlc -where
/usr/lib/ocaml/3.10.2
```

ひどい点は、年に一回くらいマイナーバージョンがリリースされているのですが、その度にライブラリコードがバイナリ非互換に変更されているっぽく、リビルドするハメになっているようです。

Debian の ocaml メンテナンスチームはあらゆるパッケージを一つの subversion リポジトリに統合して管理しているようですが、その理由の一つがこのバージョン間の非互換性ではないでしょうか。

6.7 参考

- OCaml packaging policy `http://pkg-ocaml-maint.alioth.debian.org/ocaml_packaging_policy.txt`
- dh-ocaml パッケージ

Debian 勉強会資料

2009 年 4 月 18 日　　初版第 1 刷発行
東京エリア Debian 勉強会 （編集・印刷・発行）