タイトル：　Ruby Workshop
主催：　日本 UNIX ユーザ会
日程：　1999 年 9 月 4 日(土)
定員：　100 名
会場：　日本オラクル株式会社本社
　　ニューオータニガーデンコート 12F セミナールーム

プログラム：

時刻	講師	テーマ
9:00-9:30	受付 30 分	
9:30-10:45 (講演 45 分 質疑 30 分)	(株)ネットワーク応用通信研究所開発部主任研究員 まつもとゆきひろ	Ruby 作者による Ruby について
10:45-11:30 (講演 25 分 質疑 20 分)	キヤノン株式会社 木村浩一	Windows 環境における Ruby について
11:30-12:15 (講演 25 分 質疑 20 分)	IIJ 前橋孝広	Ruby の GUI 環境について
12:15-13:45	お昼休み 90 分	
13:45-14:30 (講演 25 分 質疑 20 分)	日本オラクル株式会社 Linux 事業推進部 宮原徹	Ruby と oracle について
14:30-14:45	休憩 15 分	
14:45-16:45 (パネリスト講演 20 分*3 +α 討論 60 分)	利用者環境から見た Ruby について ---スクリプト言語に対する誤解を解く	
	アサカ理研工業株式会社 アサカネット事務局 鴫原厚博	SGmail の開発を中心に
	ForUs 松尾尚典	Meeting2000 の開発を中心に
	北海道大学大学院理学研究科 後藤謙太郎	Ruby による地球流体データの 扱い
	キヤノン株式会社 木村浩一	スクリプト言語の簡単講座
16:45-17:00	懇親会準備 15 分	
17:00-18:00	懇親会 60 分	

9:30-10:45 Ruby 作者による Ruby について

JUS Ruby Workshop

オブジェクト指向スクリプト言語 Ruby

(株)ネットワーク応用通信研究所
まつもと ゆきひろ
matz@netlab.co.jp

Rubyとは?

FAQ(Frequent Asked Question)

質問と答えにより明らかにしよう。

なぜ新しい言語を?

- 作りたかったから
  - オブジェクト指向言語を
  - フリーソフトウェアを
- 新しい酒には新しい革袋を
- There's More Than One Way To Do It.

なぜRubyという名前?

- 宝石の名前からから(略語ではない)
- 7月の誕生石
  - 同僚の誕生石から
  - 6月は真珠(Pearl)
  - 「Perl(Pearl)の次」を意識してないわけじゃない
- 同名の言語が既にあったことは知らなかった

なにに向いてるの?

- スピード狂以外ならなんにでも
  - オブジェクト嫌いにも向かないかな
- スクリプト的処理には最適
- 複雑なデータ構造も表現できる
- 拡張ライブラリで機能をいくらでも追加できる
  - GUI
  - データベース
  - 数学演算

Perlとの違いは?

- 若い(良い意味でも、悪い意味でも)
  - 純オブジェクト指向
  - モジュールシステム
- 思想は結構似ている
  - UNIX主義
  - There's More Than One Way To Do It.
  - Easy things shoule be easy, hard things should be possible
  - 非minimalist

Perlとの違いは?(その2)

- 人間の心理の別の面に注目している
  - Iwallは自然言語学
  - matzはプログラミング言語学
- Perlには思想がないがRubyにはある
  - バベル17 ^^;;;

Pythonとの違いは?

- 若い(良い意味でも、悪い意味でも)
- 両方ともオブジェクト指向言語
- 思想は結構似てない
  - 多様性に対する考え
  - 簡潔さに関する考え
  - Minimalismに対する考え
  - インデントに関する考え

Rubyの特長って?

- オブジェクト指向スクリプト言語
- 純オブジェクト指向
- スクリプトプログラミング
- ネットワークプログラミング
- 組み込みスレッド
- 日本語対応(UTF-8にも対応)
- 楽しいプログラミング

Rubyの思想って?

- プログラミングは楽しい
- オブジェクト指向は素晴らしい
- 動的なことは素晴らしい
- 多様性は素晴らしい
- 使いやすさにはバランスが大切
- 「なにを」ではなく「いかに」

楽しいプログラミングって?

- やりたいことがすっきり書ける
  - 簡単なことは簡単に，難しいことも可能に
- お約束が少ない
  - 寛容な文法，良く使うものは簡単に
- 一貫性
- その気でないのに分かりにくいプログラムにならない
- オブジェクト指向プログラミングを簡単に実践できる
- スレッドプログラミングを簡単に実践できる
- 例外でエラー処理がらくちん
- イテレータでループの抽象化

Rubyの設計原理は?

- プログラミング言語の人間工学
- いろいろできればそれで良いと言うわけではない
- 「なにができるか」よりも「いかにできるか」
- 使いやすさ(楽しさ)の追求
- 単純さと複雑さのバランス

どうやって学ぶ?

- 使ってみる
- ソースを読む
- マニュアルやホームページを読む
- メーリングリストで尋ねる
- 雑誌で読む
  - http://www.netlab.co.jp/ruby/jp/press.html
- 本を買う
  - 近日発売予定(こんどこそ本当..のはず)

なぜRubyを使わなきゃ?

- いや..別に..無理には...
- でも、楽しいよ、他の言語より
  - 「楽しいプログラミング」 – Rubyの信条

オブジェクト指向って難しくない?

- 本来オブジェクト指向は人間にとって自然な考え方
- 「難しい」と思う時点で機械よりの立場(かも)
- 別に無理して継承使わなくても

Rubyの将来性は?

- 素材の良さは見る人が見れば分かる(でしょ?)
- 未来は明るい..はず

matzにもしものことがあったら?

- 縁起でもないことを...
- ソースは公開されてるし、Rubyハッカーは他にもいるから。

Rubyの思想に共感した人はなにができる?

- Rubyを使う
- 楽しんでプログラミングする
- バグをみつけたらmatzに伝える(できれば最新版で)

9:30-10:45 Ruby 作者による Ruby について

Windwos環境におけるRuby

その現在・過去・未来

木村浩一

現状

- Cygwin版とmswin32版が存在している
- mswin32版のバイナリ配布は行っていない
- mswin32版の作成にはVisual C++が必要

なぜ移植したのか?

ソースを眺めていたら、途中まで作業されていた形跡があった
gawkやmawkをいじったことがあったのでその延長として興味があった

作業にあたっての苦労

- popen()の実装
- 拡張子の違い(lib、dll等)
- コンパイラーのオプション
- 構築スクリプトの対応
- 使用できるツールが限定されている

popen()

- gawkで一度実装した経験があった
- 完全にうまく動作するものはできなかった
- Visual C++のライブラリのソースを見たらドキュメント化されていないことをしていた
- あきらめて、ライブラリをそのまま使うことに

拡張子の違い

- ライブラリファイル(.a と.lib)
- オブジェクトファイル(.oと.obj)
- 共有ライブラリ(.soと.dll)

コンパイラーのオプション

- UNIX環境のCコンパイラーのオプションとVisual C++とではオプションの種類が異なる

↓

構築スクリプトがそのままでは使えない

構築スクリプト(extmk.rb)の対応

- UNIX一般のものと拡張子が異なる
- Cコンパイラーのオプションが異なる
- Rubyが良くわかってない状態でいじくりまわした
- 本家との一般化を考えてはいたが、なおざりにしている間に小松さんが作業してくれた

使用できるツールの制限

- 実行ファイル作成に際しての手間を軽減するためにコンパイラ以外のツールはできるだけ使わないようにしたかった
- exportする関数の検出ができないため手作業が入る

Cygwin版 Ruby

- わたなべひろふみさんの手によるもの
- Windows用Rubyとして標準的なもの
- バイナリ配布がある
- win32oleにも対応

Cygwin版とmswin32版の違い

- Cygwin版はCygnusによるUNIX環境エミュレートのレイヤーになるDLLが存在し、実行にはこれが必要
- Cygwin版はUNIX環境で動くスクリプトが多分そのまま使える

WindowsでRubyを使う場面

- awk, Perlと同様の(あるいはそれ以上の)テキスト処理言語として
- win32OLE拡張ライブラリを使った「糊」言語として

Win32ole

- 助田さんによる拡張ライブラリ
- OLEによりOfficeソフト(ex. Accessデータベース)等の制御が可能
- Perlの同等の拡張モジュールに比べてよりVBAに近い表記ができる

将来に向けて その１

- 拡張ライブラリのTclTklibがうまく動かない問題の解決
- VC++以外のコンパイラー(ex. Borland C++ Builder, LCC)対応
- Python for Windowsのような環境の構築
- Windows CE対応

将来に向けて その２

- Windows Scripting Host対応
- Internet Explorerとの連携

May the source be with you

Ruby Workshop

Ruby の GUI 環境について

前橋 孝広

内容

GUI とは

- スクリプト言語と GUI の相性
- Ruby の GUI 拡張ライブラリ

Ruby/Tk と Ruby/Gtk

- GUI プログラミング

Ruby のGUI 環境の課題

GUI とは

Graphical な User Interface (⇔ CUI)

- 知らなくてもある程度使えるインターフェース
  - できることを選択肢として見せ、選ばせる

現実世界との類似性
- ボタンを押すと引っ込むとか

- 見た目は重要

GUI の基礎知識

ウィンドウシステム
- 直線、四角、ビットマップの描画
- マウス、キーボードのイベント処理

- ツールキット
  - 頻繁に使うパターンをまとめたもの
  - ウィジェット
    - ボタン、メニュー、スクロールバーなど
  - 例: Tk, Gtk, Motif, Qt など
    - いろいろあるのが良いところでもあり悪いところでもある

イベントドリブン
- イベント処理ループからコールバックルーチンを呼び出す

スクリプト言語と GUI の相性・生産性

GUI プログラミングは、割と決まりきったことをごちゃご
ちゃ書かなければならない

GUI アプリケーションでは、GUI に特化した細かい処理が
7から8割を占める
- 使うは天国、作るは地獄
- 作るも天国でないとやだ

少し修正しては再実行のサイクルが多い

- 結論: スクリプト言語と GUI の相性は「抜群」

スクリプト言語と GUI の相性・生産性

GUI プログラミングは、割と決まりきったことをごちゃご
ちゃ書かなければならない

GUI アプリケーションでは、GUI に特化した細かい処理が
7から8割を占める
- 使うは天国、作るは地獄
- 作るも天国でないとやだ

少し修正しては再実行のサイクルが多い

- 結論: スクリプト言語と GUI の相性は「抜群」

Ruby と GUI の相性

GUI ツールキットそのものがオブジェクト指向的考え方
- 各ウィジェットのクラスからインスタンスを生成
- ウィジェットはそれぞれ属性を持つ
- 同じクラスのウィジェットは同じ振る舞いをする

ニウィジェットはオブジェクトそのもの
- 他のオブジェクトと統一的に扱える

Ruby の使いやすさで GUI を扱えるのは快感

結論: Ruby と GUI の相性は「抜群」

Ruby の GUI 拡張ライブラリ

[ruby-list:9906] より

tk
gtk
ニxtoolkit
ニxview
xforms
ニezwgl
ニxfltk(はまだ作業にかかってない？)
openpgl(はGUIツールキットではないか)

Ruby/Tk

Ruby らしい記述で Tk の GUI を実現

ニTkObject を頂点とするクラス階層

TkObject
- TkWindow
  - TkToplevel
  - TkFrame
  - TkLabel
  - ...
- TkImage
- ...

ニTcl/Tk 自体はオブジェクト指向言語ではない

Ruby/Tk

Tcl/Tk への wrapper
- 昔は内部で wish を起動していた
- Tcl は邪魔:-p
- 将来は pTk(portable Tk)?

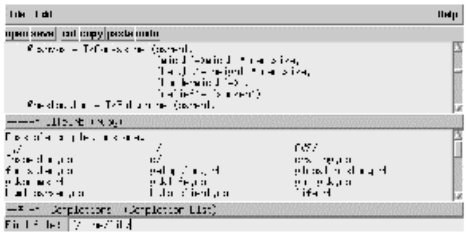
ニコールバックを実現している部分がややトリッキー

ニruby + tk.rb + tcltklib.so + tkutil.so + libtcl + libtk

Ruby/Tk チュートリアル
- http://www.cdrom.co.jp/~hiramatu/#RubyTK

Ruby/Tk の application

SGmail -- IMAP4, POP3 対応のGUI なメーラ
ニmine -- マルチバッファなテキストエディタ
- Ruby mode, Ruby interaction mode
- 簡易クラスブラウザ、インスペクタ、ライフゲーム内蔵
- 無制限 undo
- Mine is not Emacs



Ruby/Gtk

Ruby から Gtk+ ライブラリを使うための拡張ライブラリ

ニGtk+ はGIMP のために開発、GNOME と共に発展中

Gtk+ は最初からオブジェクト指向なツールキット

Gtk+ は最初からインタープリタバインディングを考慮
- Perl, Python, Guile, Pike, ...

ニWindows 環境でも動く

Ruby/Gtk

- ruby + gtk.so + libgtk
- Gtk のクラス階層がそのまま Ruby のクラス階層に対応
- (繰り返さない)イテレータによる signal connect
- APIドキュメント
  - http://www.ueda.info.waseda.ac.jp/~igarashi/ruby/gtk-ja.html
- Ruby/Gtk のチュートリアルページ
  - http://ruby.freak.ne.jp/gtk/

Ruby/Gtk

- Gtk を C から使う場合に比べてより自然なオブジェクト指向的な記述ができる
- C の場合
  - gtk_container_border_width(GTK_CONTAINER (window), 10);
- Ruby の場合
  - window.border_width = 10

Ruby/Tk, Ruby/Gtk の比較

- 余計なもの(tcl)が介在するだけ Ruby/Tk のほうが遅い?
  - しかし実用的には両者とも十分
- Ruby/Tk は Thread との相性が良くない
- Ruby/Tk の方がお手軽なかんじ

Ruby/Tk, Ruby/Gtk の比較

- Ruby/Gtk は、ウィジェットの配置が面倒
- Gtk の方がかっこいい? 流行っている?
- Gtk はドキュメントが少ない

Ruby/Tk プログラム例

```
require "tk"
TkButton.new(nil, 'text'=>'hello',
        'command'=>
        proc{print "hello\n"}).pack('fill'=>'x')
TkButton.new(nil, 'text'=>'quit',
        'command'=>'exit').pack('fill'=>'x')
Tk.mainloop
```

Ruby/Gtk プログラム例

```
require 'gtk'
window = Gtk::Window::new(Gtk::WINDOW_TOPLEVEL)
box = Gtk::VBox::new(FALSE, 0)
window.add(box)
button = Gtk::Button::new("hello")
button.signal_connect("clicked") {print "hello\n"}
box.pack_start(button, TRUE, TRUE, 0)
button.show
button = Gtk::Button::new("quit")
button.signal_connect("clicked") {window.destroy; exit}
box.pack_start(button, TRUE, TRUE, 0)
button.show
box.show
window.show
Gtk::main()
```

11:30-12:15 Ruby の GUI について

GUI プログラムの流れ

ウィジェットの生成

ニウィジェットの各プロパティ設定

コールバックルーチンをイベントにバインド

ウィジェットの配置・表示

ニイベントループに突入

GUI プログラミングの注意点

上から下への直列の実行ではない
- どんな順序でイベントが発生するかわからない

ニコールバックルーチンは短時間で処理を終えなければならない
- そうしないと、他のイベントは待たされる

ニ長い処理時間がかかるものは分割してタイマーで実行
- アニメーションなど

ブロックする I/O を扱う場合は特に注意

ニ→すみやかにイベントループに制御を戻す

部品の組合わせによるプログラミング

大きな一枚岩のようなプログラムではなく、小さな部品の組み合わせによるプログラミング
- 開発が容易
- 再利用可能

部品どうしをつなぐ接着剤(glue)としてスクリプト言語を活用

部品
- ツールキット
  - ウィジェット
- 組み込みクラス
- 他の拡張ライブラリ

オブジェクト指向と GUI プログラミング

ウィジェットの抽象化・カプセル化
- 実装を隠蔽

ニオブジェクト指向の本質は継承ではない
- 下手な継承は上位クラスの実装に依存する

ウィジェットの組合わせで新しいウィジェットを作る
- 継承よりオブジェクトコンポジション

ニクラスどうしの結び付きを弱くする

オブジェクトとして統一的な操作

Ruby の GUI環境: 今後の課題

ドキュメントをもっと充実させる
- Ruby を覚え、Tcl/Tk あるいは Gtk を覚え、C のソースを読む
- 参考になるサンプルも少ない

ニGUI Builder があった方が良い?

キラーアプリケーション

まとめ

スクリプト言語と GUI の相性は抜群

ニRuby と GUI の相性も抜群

ドキュメントの充実を

SGmailとRubyとは

- なぜSGmailを作ったか?
- 候補となった言語と主観的評価
- Ruby/Tkで作ってみて
- 私的Ruby/Tkの評価

なぜSGmailを作ったか?

- 会社でも自宅でも重複せずにメールを読みたい
- 時にはモバイルもしたい
- IMAPを使おう
- X11R6で動くIMAP対応MUAがない!

候補となった言語と主観的評価

- C/C++
- Tcl/Tk
- Perl/Tk
- Ruby/Tk

C/C++

- ○ 速度面では一番
- × 気軽にプログラミングと言うわけにはいかない

Tcl/Tk

- ○ 簡単にwidgetを使える
- ○ 漢字の入力はOK
- × Tclの文法には馴染めなかった
- × 4.2ベースではsocketが扱えない

Perl/Tk

- ○ Perlは良くわかっている
- × Tk3.4(?)ベースで古い
- × 漢字の入力がダメ

Ruby/Tk

- ○ 文法的にはPerlに似ていて馴染みやすそう
- ○ 漢字の入力はOK
- ○ socketも扱える
- ○ OOPで大きめのプログラムも作れそう
- × 新しい言語を学習しなければならない
- × OOPは難しそう

Ruby/Tkで作ってみて

- プログラムの本質以外の部分は記述しなくて良い
- TkとOOPの親和性が良い
- 下手なフレームワークが無い
- デバッグが短時間でできる
- 日本語のサポート
- 最新のTcl/Tkを使う事が可能
- マルチプラットホームに対応できる
- 開発時間が短くて済んだ
- 結構速い

本質以外の部分は記述しなくて良い

- ヘッダ記述が無い
- OOPも最低限の記述で可能
- イテレータ等の便利な機構もある

TkとOOPの親和性が良い

- Tk周りのインターフェースが良くできている
- 継承で専用のクラスを作れる

下手なフレームワークが無い

- 下手なフレームワークは邪魔もの
  - 昔MFCで苦労をしたから(^^;)
- 自分で作ったフレームワークが一番良い
  - 昔DOSのGUIライブラリを作った経験から

デバッグが短時間でできる

- インタプリタはターンアラウンドタイムが短い

日本語のサポート

- 始めから日本語の扱いを考慮している

最新のTcl/Tkを使う事が可能

- Ver 2.33からTcl/Tk8.1をサポート

マルチプラットホームに対応できる

- Ver 2.33からはWindows9{5|8}もサポート
- Ruby/TkさえあればOK

開発時間が短くて済んだ

- プロトタイプ作成に約4ヵ月
  - Ruby,Tk,IMAPを学習しながらなら早い?
- Ver 1.0が約2ヵ月
- Ver 2.0もほとんど作り直しで2ヵ月

結構早い

- Pentium166MHz程度のマシンで結構快適
- WindowsでもPentium500MHzでまあまあ
  - 結構shift-jis→UTF8のオーバーヘッドがある

私的Ruby/Tkの評価

- RubyはPerlと同じレベルの言語ではない
  - CGIだけに使うのではもったい無い
- 結構大きいプログラムもきちんと動く
  - Netscape並にメモリは食いますが(^^;)
- GUIプログラミングに最適
  - Tcl/Tkレベルに一般化して欲しい
  - Tcl/Tkよりプログラミングは楽
  - Gtkだって使えるぞ
- 比較的簡単にマルチプラットホーム化できる
  - Windows版も簡単に作れた
  - Mac版Ruby/Tkに期待
- 敵はjava?
- みんなで使えば普及する?

Ruby と Meeting2000

松尾尚典 @ ForUs

目次

1 Meeting2000について

- 参加希望者の都合を調整し、参加申込を受け付けるWebアプリケーション (便利!)
  - 幹事さんの苦労を超軽減
  - 参加者の手間も削減
- グループウェア
  - 情報の共有、加工と提示
- Rubyの普及にある程度貢献できた感触

Meeting2000について(2)

- 構成
  - 自作HTTPサーバ機能
  - 自作Webアプリフレームワーク (リクエスト、ページ、セッション)
- 規模
  - 3000行、0.5人月 (1人*0.5月)
  - 日曜プログラム程度、中規模というより小規模
  - Webアプリとしては結構複雑 (商用、社内システムを除けば^^)

2 Rubyを選択した理由

- 私
  - 言語屋さんでも言語マニアでもない
  - コーディングが趣味でもない(趣味はゴルフだ)
  - やりたい事がすばやく出来れば良い
  - 以前は Perl4
- Ruby
  - 書きたい事が書け、書きたくない事は書かなくて済む度合が、自分にとって丁度気持ち良い

Rubyのよいところ 概観

- vs システムプログラミング言語
  - 開発効率がよい(C、C++)
  - 型等による制約 が少ない(Java、C、C++)
  - 多少遅いが気にならない
- vs 他のスクリプティング言語
  - OOである。しかも分かりやすいOO
  - 日本製で、日本語のコミュニティが本家

キーとなった機能

- 素直なOO
- gc、thread、socket、正規表現、ハッシュ、例外…
- 変数がオブジェクトのハンドル
- Ruby Home Page を見て(それが本当なら)良いと確信

選択肢

- スクリプティング言語
  - Perl[45]
  - Tcl
    - 以前5000行ぐらい書いたけど、超苦い記憶だから早く忘れたい、ていうか忘れた^^
- システムプログラム言語
  - Java
  - C++
    - 出来るだけかかわりを持たないようにしてます

選択肢 (1) : vs Perl

- 好きだった
- スクリプティング言語の良さを教えてもらった
- どうにでも書ける、というのはちょっと嫌
  - Perl4
    - Nifty4U は沢山の人に使ってもらえた
    - 今更な感、OOでない
  - Perl5
    - 素直なOOでない。Rubyの決定的な勝ち

選択肢 (2) : vs Java

- コンポーネントはC++よりある感じ
- お手軽度が決定的に足らない(ニーズが違うから仕方がない)
  - 例外のcatchを強制される
  - スレッドが面倒
  - 正規表現がない
  - 日本語が下手
  - new Hoge より Hoge.new かな
  - RMI等は、Rubyにも欲しい

3 開発してみて

- やっぱり極めて書きやすい
  - 生産性高い (Perlよりよいぞ。OOだから)
  - 仕事でもプロトタイプ作成に利用
    - 分析結果をすばやく検証できる
- どうでもよいと思ってたけど、実は結構よかった^^機能
  - 1 定義が動的
  - 2 特異メソッド
  - 3 制御構造がオブジェクトを返す

3-1 定義が動的

- 定義も実行文も同じ
- 例えば、既に定義済みのクラスに、メソッドを自由に追加できる
- クラスの機能拡張が容易かつ当該スクリプト中に閉じた変更が可能
  - 標準ライブラリや、人様の作ったクラスに(失礼ながら)機能不足を感じても、気軽にメソッドを追加出来る

3-2 特異メソッド

- インスタンスに固有のメソッドを定義出来る
- ホワイトボックス型フレームワーク
  - サブクラスを定義して、インスタンス化
  - システム中に一つしか要らないインスタンスの為にもサブクラスを定義しなければならない
- 今回使ったWebアプリ用のフレームワーク
  - Pageクラス、Requestクラスのインスタンスは画面、要求毎に一つだけ存在すればよい

特異メソッドを使わなければ

```
class PageX < Page  # サブクラス定義
 def form(session)
  # 固有のメソッドを定義
 end
end # サブクラス定義終了
p = PageX.new # インスタンス化
...
```

特異メソッドを使えば

```
p = Page.new  # インスタンス化
def p.form(session)
 # 固有のメソッドを定義
end
…
```

- サブクラスを定義しなくてよい
- 実際は大した差ではない
  - が、それが気持ちよい^^場合がある
  - が、それが開発にとっては重要

3-3 制御構造がオブジェクトを返す

- ページ生成で利用

```
… + if i.url then
       "<A HREF='#{i.url}'>#{i.name}</A>"
    else
      i.name
    end + …
```

4 今後の展開

- 現在 0.93 のベータ版をRAAにて提供中
- 現在の機能にほぼ満足しているが、機能拡張の要求は多い
  - 時刻も調整しよう、日と場所で表にする等
  - eRuby化、国際化、見栄えをよくしたい
  - しかし、今のMeeting2000は保守性が低く、機能や画面の追加が難しい
- グループウェアサーバ構想
  - その上で次期Meeting2000を構築する予定

グループウェアサーバ

- グループウェアに対するニーズは高い
- でもちまたのCGIスクリプトは物足りない
  - ユーザ登録、掲示板や更新情報通知システム、ランキング物。
  - もっともっと便利なアプリケーションを!
    - ユーザ、グループによる情報の選別、共有
- グループウェア用のアプリケーションサーバが欲しい
  - アプリケーションサーバって???

アプリケーションサーバ

- アプリケーションを開発するためのフレームワークであり、且つアプリケーション群を管理するサーバ
- クライアントがWebブラウザであるアプリケーションが対象 (Webアプリケーション)
- HTTPサーバ、データアクセス、トランザクション、セッション、サーバ分散、セキュリティ等の様々な機能を有する
- OAS(オラクル)、WebSphere(IBM)、NAS(Netscape)、INTERSTAGE(富士通)等

欲しいもの

- 製品版は
  - 高い。買えないって。
  - 機能が多すぎる。そんなにいらない。
- 作りたいものは
  - Webベースのグループウェアが対象のアプリケーションサーバ
  - 日曜プログラマ(僕)が手軽に使える
  - 機能は最小限で、簡単、わかりやすい

現在

- GroupWareServer (仮称) 0.0.2 作成中
  - ユーザ、グループ管理機能
    - ユーザクラス、ログイン、設定変更画面等
  - 排他制御、データのロード、セーブ
  - プレゼンテーション層とビジネスロジック層の分離
  - リクエスト処理オブジェクトとページレイアウトオブジェクトの分離
  - まだ人様に使ってもらえる段階ではない
- 御意見など頂ければ歓迎

5 まとめ

- Ruby は一見して便利、使ってやはり便利
  - 試してみて欲しい
- Meeting2000も便利、でも保守性が悪い
  - グループウェアサーバを作成中
- Rubyについて思うこと
  - 本を! (まつもとさんへ)
  - アプリケーションを! (Rubyユーザへ)

アプリケーションサーバにおける リクエスト処理とページレイアウトの分離

(株)東芝 情報・社会システム社
SI技術開発センター SI技術担当
松尾尚典

TOSHIBA

目次

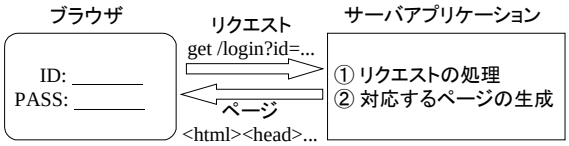
TOSHIBA

1 Webアプリケーション

- よく見かけるCGIなアプリケーション
  - 掲示板、ユーザ登録、情報サービス…
- 画面遷移が少ないものが殆ど
  - 一つの画面で入力、もう一つの画面で出力
- 一般のGUIアプリケーションなら数十、数百の画面を扱っているのに…
- もう少し画面数の多い、役に立つアプリケーションを作りたい

TOSHIBA

サーバアプリケーションの処理

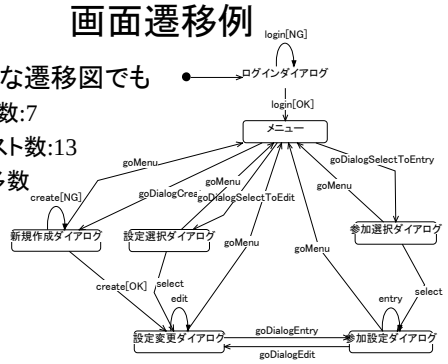


- 多くは①と②を一緒に記述
  - 教本のサンプルコード等殆ど全て
- 単純なアプリケーションではそれで十分
- しかし、リクエストやページの種類、関連数が多くなると、途端に開発が難しくなる

TOSHIBA

画面遷移例

こんな小さな遷移図でも
- ページ数:7
- リクエスト数:13
- 関連:多数



TOSHIBA

リクエストとページ

- リクエストとページは多対多
  - リクエストとページを、1対1として一緒に処理するアプローチでは対応できない
  - 一部のリクエストとページは密接に結びついているので、うまい工夫が出来そう
  - だが全てを奇麗に解決する手段は?
- 多数の画面を扱うのはやっぱり面倒
  - みんないろいろ工夫している

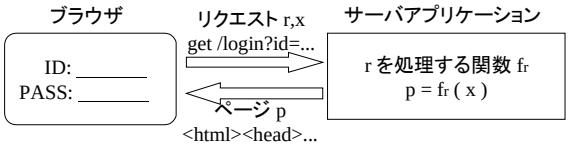
TOSHIBA

アプリケーションサーバでは

- 一つの解として、リクエストとページの分離
  - リクエストの処理とページレイアウトを分離出来る構造、メカニズムを提供
  - もちろんこれで全て解決できるわけではない
  - アプリケーションサーバのメリットがこれだけという訳でもない

TOSHIBA

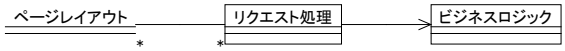
2 リクエストとページの分離



- データ x を持つリクエスト r に対応したページ p を選択する関数 f_r
- 関数 f_r とページ p は明確に分離出来る
  - f_r :リクエスト処理オブジェクト
  - p :ページレイアウトオブジェクト

TOSHIBA

分離することのメリット



- 責任と処理内容の明確化
  - 誰が何をするのか
- 開発効率、分業化、保守性
  - ページレイアウトはデザインの専門家に
  - リクエスト処理はプログラマに
  - ビジネスロジックは業務の専門家に

TOSHIBA

アプリケーションサーバにおけるリクエストとページの分離

- リクエスト処理とページレイアウト
  - 例えば Servlet / JSP、VBScript / ASP、Requestオブジェクト / Pageオブジェクト
- どちらもプログラムだから、やろうと思えばなんでも出来る
- だが、責任範囲以外のことはやるな!
  - わざわざ分離したんだから…

TOSHIBA

フレームワーク利用の心構え

- そもそもフレームワークは制約の塊
  - あれをしてはいけない、これをしてもいけない
  - このフォーマットに従え、範囲を逸脱するな…
- しかしその制約に従って初めて便利
  - 制約があるから迷わずに済む、ありがたい
  - 分業化、パターン化、コンポーネント化
- 開発では、制約に誠実に従うが吉
  - 現実には、構造だけ提供される場合が多い
  - 制約(=ノウハウ)部分は別売

TOSHIBA

3 リクエスト処理オブジェクト

- ① データを受け取り、
- ② 対応するビジネスロジックを実行し、
- ③ ページレイアウトオブジェクトを選択する

TOSHIBA

責任

- 渡されたデータをビジネスロジック用に正規化する
- ビジネスロジックを実行する
- 正常系、準正常系、異常系で分岐する
- ページレイアウトオブジェクト用にデータを正規化する
- ページレイアウトオブジェクトを呼び出す
- その他のことはやっては *いけない*

TOSHIBA

コードのパターン化(Ruby版)

```
begin  # Java なら try - catch
  < データの正規化(ビジネスロジック用) >
  < ビジネスロジックの実行 >
  < データの正規化(ページレイアウトオブジェクト用) >
  < 正常系のページレイアウトオブジェクトの呼び出し >
rescue BusinessLogicException
  case $!.to_s
  when < 例外X >
    < データの正規化(ページレイアウトオブジェクト用) >
    < 例外Xのページレイアウトオブジェクトの呼び出し >
  when
    ...
  else
    raise  # 処理できなかった例外は再発生させる
  end
end
# rescue されなかった例外はスルーされる
```

正常系

準正常系

異常系

TOSHIBA

4 ページレイアウトオブジェクト

- 与えられたデータを用いて、HTML文字列を生成する
  - これだけ
  - デザイナがデザインに集中できる
  - データで表現可能な範囲なら、変更も容易

TOSHIBA

責任

- 渡されたデータをページとして表現する
- リクエストオブジェクト名をリンク先とする
- その他のことはやっては *いけない*

- 前提条件
  - 必要十分なデータが渡される
  - データは表示用に正規化されている
  - 必要なら、データは表示用の構造にパックされている

TOSHIBA

ページのパターン化

- 表示すべきデータは全て与えられているので、プログラミングレス
  → デザイナの手持ちのパターンが使える
- パターンの集積が可能
  → どんどん再利用

TOSHIBA

5 まとめ

- Webアプリケーション開発では、リクエスト処理とページレイアウトを明確に分離するとよい
- アプリケーションサーバは、分離するための構造を提供している
- 開発においては、分離する意義を十分理解して、与えられた構造を利用すべきである

TOSHIBA

Rubyによる地球流体データの扱い

北海道大学大学院
理学研究科数学専攻
後藤謙太郎
gotoken@math.sci.hokudai.ac.jp

謝辞

- 地球流体電脳倶楽部
  - http://www.gfd-dennou.org
- 科学技術振興事業団
  - 計算科学技術活用型特定研究開発推進事業
  - http://www.jst.go.jp
- 日本UNIXユーザ会
  - http://www.jus.or.jp

おことわり

このプロジェクトは始まったばかりで、一部の仮実装はしましたが、まだきちんとしたプロダクトはできてません。
2000年3月をメドに鋭意作成中です。

あらまし

- 地球流体
- したいこと
  - 膨大なデータを共有したい
  - 絵を書きたい
- 背景となる技術
  - 電脳ライブラリ(DCL, gtools)
  - netCDF
- 進行中プロジェクト
  - netCDFの読み書きとラッパ
  - 新たなデータ書式をRubyで
  - DCLラッパとクラスライブラリ化

地球流体

- おもに、温度、風速
- 観測方法もさまざま
  - 固定、船上、ゾンデ(風船)
- モデルによる計算データも扱いたい

膨大なデータを共有したい

- 100Mバイトは当り前、1Gのファイルがあったりする世界
- FORTRAN でも扱えないと困る
- データの説明をデータファイル自体に書いてないと面倒
  - 自然言語だと面倒
  - FORTRANベッタリだと構造を定義しにくい

絵を書きたい

- とにかくプロットしたい
- 地球に依存した事情がある
  - 地図投影
  - 軸の間隔やラベル
- 後者はDCL(後述)で解決されている

netCDF

netCDF (network Common Data Format)

- 配列指向のデータの共有を目指した書式仕様とライブラリ群.
- 汎用なので、制限を付けなければ共有できない.
  - 各種conventionが存在する
- University Corporation for Atmospheric Research 発
  - UNIDATA の一つ <URL:http://www.unidata.ucar.edu/>
  - The NASA CDF に由来
- FORTRAN, C, C++, Perl, Java のインターフェイスがある. (Ruby がないのは悔しい ^^;)

DCL

特徴

- FORTRAN ライブラリである.
  - C 言語全盛の時代にあって, 古式ゆかしい FORTRAN を守っている
  - 大型機ではまだ FORTRAN には勝てない
  - ただし現在Cに書き換え中
- 移植性が高い
  - 機種依存する部分はいくつかのサブパッケージにまとめてあるので, これらが移植できれば OK
- 地球科学特有の事情が考慮されている
  - 欠損値処理や地図投影などの機能がもっとも基本的な部分で サポートされている

どこにRubyを活かすか

- プロトタイピング
  - やっぱりインタプリタだと楽チン
  - 数学的なモデルも容易に実装できる
  - 拡張ライブラリ版と使い分けも可能
- データフォーマット
  - パーザーを書かなくていいので楽そう
  - 構造の定義は自由
  - 名前空間を使って仕様を拡張しやすい
- ラッピングのテスト
  - 電脳ライブラリはあまり包まれた経験がない
  - FORTRANのライブラリのインタフェイスの指針(?)

進行中プロジェクト

- netCDFの読み書きとラッパ
- 新たなデータ書式をRubyで
- DCLラッパとクラスライブラリ化

netCDFの読み書きとラッパ

netCDFデータにはテキスト形式とバック形式がある. テキスト形式はヘッダを見るのには有効だが流通には使われない.

バック形式は読みとりやすい形式なので, 早速書いた.

- このために pack と unpack を拡張させてもらった (e,E,g,G).

問題点

- 遅い. 10M程度のファイルに含まれる数値を全てオブジェクトにするのはキツい.
  - 多次元配列クラスとnetCDFのラッパで対処予定
  - ヘッダに情報があるので, 全部読む必要はない(キャッシュ?)

新たなデータ書式をRubyで

- 観測データとその機能を分けて記述できる
  - 観測データは固有の座標軸を持たない
    位置や時刻も本質的には測定量
    格子データと言っても欠損したデータもありうる
    ゾンデデータの場合格子ですらない
    地球は球ではない
  - 一方，公開する場合はデフォルトのプロット対象は指定すべき
    温度と風速のどちらをプロットするのかとか
- 自由なのはいいけどFORTRANのことも考えないといけない
  - FORTRANでは基本的には配列によるハッシュになる
  - メソッドはサブルーチン渡しで実現(?)
  - 動的な機能(procとか)はどこまで許すかとか

DCLラッパとクラスライブラリ化

- 各種プラットフォームに対応したため名前の制約などがある
  - 現在FORTRAN90とCでラップ中
- 関数へのデータの受渡しが不便なのでこれも改良の余地あり
  - どの引数に影響が及ぶかなど
- 現在，一部関数のみ実装
- 近いうちにRubyでいくつかのクラスに整理
  - 現在は単にFORTRANのサブルーチンを呼んでるだけ

まとめ

感触
- Rubyだとプロトタイピングとアイデアの実装はやっぱり早い
- packを本家Rubyでも拡張してもらえた小気味よさが嬉しい
- 実行は遅いが，モデル計算なら拡張ライブラリで何とかなりそう
- FORTRANからくらべると一気に抽象度があげれるので作戦会議重要

問題点
- スピードをある程度確保しなければならない
  - 文字列で別言語を書いて呼び出すとかしてオブジェクトを生成せずに数値を触る方法は必要
  - 分散処理(?)
- 人手(Rubyで書ける人が少ない)汗

スクリプト言語簡単講座

木村浩一

スクリプティング言語とは?

- These are languages that are primarily interpreted, and on unix sytems, can ususally be invoked directly from a text file using #!.

  (http://www.idiom.com/free-compilers/CATEGORY/scriptin-1.html)

スクリプティング?

- 第四世代のプログラミング
  Scripting - the fourth generation
  http://www.equi4.com/jcw/scripting4.html
- 世代を進むにつれ抽象度が高くなる

Scripting: Higher Level Programming for the 21st Century

http://www.scriptics.com/people/john.ousterhout/scripting.html

パーツを作るシステム言語とそれを組み合わせるためのスクリプティング言語

スクリプティング言語の実例

- Perl
- Python
- Ruby
- Tcl
- Visual BASIC