日本Ruby会議2007

Inside Ruby/Tk

永井　秀利

九州工業大学　情報工学部　知能情報工学科

2007/06/10

Inside Ruby/Tk

．．．まず最初に

ごめんなさい!!

- ドキュメントプロジェクトでは足を引っ張ってます
- 現在のRuby1.9 (YARV)上では，まだRuby/Tkはまともに動きません
  - 新しいthread (native thread)絡みのややこしい問題があり，どうしようかと悩みつつ，放置したまま既に半年…
- 協力者募集中(^_^;

Ruby/Tkとは

- Rubyに標準添付のGUIライブラリ
  （...でも，邪魔だと言う人も多数）
- 面倒な前準備の必要なく，非常にお手軽にGUIプログラミングが可能

```
require ‘tk’
TkButton.new(:text=>’Hello, World!!’,
                    :command=>proc{puts ‘Hello!!’}).pack
Tk.mainloop
```

one liner

```
ruby -r tk  -e “TkButton.new(:text=>’Hello, World!!’,
  :command=>proc{puts ‘Hello!!’}).pack”  -e Tk.mainloop
```

Ruby/Tkとは

- Tcl/TkをRubyから使えるようにwrapしたもの

  既存のTcl/Tkスクリプトや拡張ライブラリを含め，Tcl/Tkの全機能を利用可

- 多くのプラットフォームで動く

  Unix系(X11)，Windows，MacOS(9,X)

  （ Ruby/TkORCAを使えば，
  Web等のネット越しでの利用も ）

Tcl/Tkとは

- 手軽にGUIを作成できるスクリプト言語
  (古くから存在するが，現在も開発は進行中)
- 標準の構成は汎用的なものに絞り込んであり，比較的コンパクト
  - 少数種類の多機能/多用途なウィジェットからなるウィジェット集合
  - キャンバスウィジェットの高機能さは有名
- 拡張ライブラリも多数存在

Tcl/Tkの欠点？

- 処理が遅い
  → リアルタイム処理でないなら，
  問題にはなるほどではない

  (現在のRuby/Tkウィジェットデモが
  遅いのは別の理由)

- 不細工
  → 古めかしいのはUnix系(X11)だけ

  (Tcl/Tk8.5a6からはウィジェットテーマ
  を扱えるTile拡張を標準搭載)

Tcl/Tkの欠点？

- 真に使いこなすのは難しいのかも？
  - 入口は広い(最低限の必要知識は少量)が，奥が深い
  - 各種機能を目的に合わせて組み合わせて使いこなす技量が求められる
  - 標準ではオブジェクト指向ではない
    - ライブラリ化が面倒で，苦労して組み上げても使い捨てになりがち
    - 既存のものに少し手を加えての利用も簡単とは言えず，再利用性が悪い

Tcl/Tkの欠点？

- 真に使いこなすのは難しいのかも？
  - 入口は広い(最低限の必要知識は少量)が，奥が深い
  - 各種機能を目的に合わせて組み合わせ

Ruby/Tkによって改善可能な部分

- 標準ではオブジェクト指向ではない
  - ライブラリ化が面倒で，苦労して組み上げても使い捨てになりがち
  - 既存のものに少し手を加えての利用も簡単とは言えず，再利用性が悪い

今日のお題

新しいウィジェットクラスの作成をサポートするための機能を中心に

ウィジェット生成方法のタイプ

- Ruby/Tkから見た際の生成方法に基づくウィジェットのタイプ

(1) Tcl/Tk上の単一のウィジェットクラス

(2) Tcl/Tk上の特定のコマンドで生成するもの(複合ウィジェットを含む)

(3) Ruby/Tk上の既存のウィジェットクラスからの継承

(4) Ruby/Tk上で複数ウィジェットを組み合わせて構築する複合ウィジェット

ウィジェット生成方法のタイプ

- Ruby/Tkから見た際の生成方法に基づくウィジェットのタイプ

(1) Tcl/Tk上の単一のウィジェットクラス

(2) Tcl/Tk上の特定のコマンドで生成するもの(複合ウィジェットを含む)

(3) Ruby/Tk上の既存のウィジェットクラスからの継承

(4) Ruby/Tk上で複数ウィジェットを組み合わせて構築する複合ウィジェット

生成方法タイプ１＆２

- Tcl/Tkのウィジェット生成コマンドの一般形に則っているはず

〈生成コマンド〉〈ウィジェットパス〉〈オプション〉〈値〉…

(例) button .b -text Hello -command {puts “Hello !!”}

- Ruby/Tkのウィジェットクラスの仕組みに従うことで，容易にクラス定義可能

Ruby/Tkのウィジェットクラス

- TkWindowクラスの子孫として定義
- オプションをHashで与えて生成する

〈ウィジェットクラス〉.new(〈親ウィジェット〉,
〈オプション〉=>〈値〉, …)

(例) TkButton.new(Tk.root,
:text=>'Hello',
:command=>proc{puts 'Hello !!'})

TkWindow#initialize

- 通常は再定義の必要なし
- 処理内容
  1. ウィジェットパス指定オプションの処理とウィジェットパスの決定
  2. フォント等特殊処理を要するオプションの抜き出し
  3. create_selfメソッドの呼出
  4. (必要なら)フォント設定処理の実行
  5. (必要なら)メソッド呼出を必要とするオプションの処理

パス指定オプション

- parent
  - 親ウィジェットの指定
  - 通常はnewメソッドの第1引数で指定するが，オプションHash上での指定も可能
- widgetname
  - ウィジェット名を特に指定する必要がある場合や，既に存在するウィジェット用にオブジェクトを生成する場合に指定
- without_creating
  - 値が真の場合はウィジェット生成処理を呼ばない

特殊処理オプション

- フォント指定オプション
  - TkFontオブジェクトによる管理のため
- メソッド呼び出しを必要とするもの
  - ウィジェット属性ではないものを属性と同様に扱うための機構
  - 登録には__methodcall_optkeysメソッドを再定義する
- Tcl/Tkに渡す値に特別な変換処理を要するもの
  - 登録には__ruby2val_optkeysメソッドを再定義する

TkWindow#create_self

- Tcl/Tk上のコマンドを呼び出して，ウィジェットを生成する実体
- 通常は再定義の必要なし
- self.class::TkCommandNames[0]をウィジェット生成コマンドと解釈

ウィジェットクラスの最小定義

例えば…

“tk::hoge” コマンドによって生成されるウィジェット用にTk::Hogeクラスを定義

```
class Tk::Hoge < TkWindow
  TkCommandNames = [‘tk::hoge’.freeze].freeze
end
```

Tk::Hogeウィジェットの機能の呼び出し

```
hoge1 = Tk::Hoge.new(parent, opt=>val, ・・・ )
hoge1.tk_send(〈サブコマンド〉, 〈オプション〉, ・・・ )
Tk.tk_call(hoge1, 〈サブコマンド〉, 〈オプション〉, ・・・ )
```

ウィジェットクラスの最小定義

例えば…

“tk::hoge” コマンドによって生成される

ただし …

この最小定義ではオプションデータベースとの連携が定義されていない

hoge1.tk_send(〈サブコマンド〉, 〈オプション〉, ・・・)

Tk.tk_call(hoge1, 〈サブコマンド〉, 〈オプション〉, ・・・)

オプションデータベース

- オプション(リソース)データベース
  - 各ウィジェットまたはウィジェットクラスのデフォルトの属性値を外部管理/設定可能にするもの
  - X Window Systemのリソースデータベースと同様の記述(というか，そのもの)
  - ウィジェットパス
    == データベース上のウィジェット名
  - ウィジェットクラス
    == データベース上のクラス名
  - ウィジェット属性名/属性クラス
    == データベース上の属性名/属性クラス名

データベースクラス名の保持

- TkOptionDBクラスにより，オプションデータベースの操作が可能
- ウィジェットクラスに対応するデータベースクラス名が存在するなら，定数WidgetClassNameに設定しておくべき
- ウィジェットパスからウィジェットオブジェクトへの自動変換が必要となりうるなら，TkComm::WidgetClassNamesへの登録も必要
  → データベースクラス名からウィジェットクラスを検索するため

ウィジェットパスから オブジェクトへの変換

- Tcl/Tk上ではウィジェットをウィジェットパスで判別
- ウィジェットパスとウィジェットオブジェクトとの対応付けが必要
- Ruby/Tk上で作られたウィジェット
  → 対応をHashテーブルで管理
- Tcl/Tk上で作られたウィジェット
  → ？？？？

対応情報のないウィジェットパスからのオブジェクト自動生成

1. Tcl/Tk上のウィジェットクラス名を確認
2. そのクラス名に対応するウィジェットクラスがRuby/Tk上に存在するかを調べる
   a) 存在する→
      ウィジェット生成なしにオブジェクト生成
   b) 存在しない→
      クラス名に基づいてウィジェットクラスを自動生成し，そのクラスのオブジェクトとして生成

TkComm.window(〈パス〉)

→ 説明した方法でウィジェットパスからオブジェクトを生成してくれるメソッド

ウィジェットクラスの最小定義 (データベースクラス設定あり)

例えば…

先の例(Tk::Hogeクラス)で，生成されるウィジェットのデータベースクラス名が“HogeWidget”であるとき，

```
class Tk::Hoge < TkWindow
   TkCommandNames = [‘tk::hoge’.freeze].freeze
   WidgetClassName = ‘HogeWidget’.freeze
   WidgetClassNames[WidgetClassName] = self
end
```

ウィジェットクラスの最小定義 (データベースクラス設定あり)

この定義で…

```
h = Tk::Hoge.new.pack

h.opt = val
h[:opt] = val

h.font.size *= 2

h.bind('Enter'){|ev| p ev; puts ev.window.path}
h.bind_append('Enter', '%W'){|w| w.focus}
h.bind(['Control-x', 'a'], '%x %y'){|x,y| p [x,y]}
h.bind_remove('Enter')

h.destroy
```

…とか

Rubyオブジェクトの自動変換

- ウィジェット → ウィジェットパス文字列
- 配列 → Tcl/Tkのリスト
- Hash（{k1=>v1, k2=>v2, … }）
  → オプション列（ -k1 v1 -k2 v2 … ）
- 手続きオブジェクト
  → コールバックコマンド文字列
  (手続きオブジェクトはRuby/Tk上で管理)
- その他，適切な形式の文字列に変換

Rubyオブジェクトへの変換

- Tcl/Tkの値(文字列)からRubyオブジェクトへの自動変換を試みる

  TkComm.tk_tcl2ruby(val_str)

- 自動変換では判断ミスする場合，期待する型がわかっているなら
  - TkComm.bool(val_str)
  - TkComm.string(val_str)
  - TkComm.number(val_str)
  - TkComm.list(val_str)

    … など

ウィジェット生成方法のタイプ

- Ruby/Tkから見た際の生成方法に基づくウィジェットのタイプ

(1) Tcl/Tk上の単一のウィジェットクラス

(2) Tcl/Tk上の特定のコマンドで生成するもの(複合ウィジェットを含む)

(3) Ruby/Tk上の既存のウィジェットクラスからの継承

(4) Ruby/Tk上で複数ウィジェットを組み合わせて構築する複合ウィジェット

生成方法タイプ３

- 記述量を減らすための仕組みも存在するが，今回は細かいことは省略
- 一部のクラスを除き，継承では個別のデータベースクラスを与えることはできない
  → 与えたい場合は，タイプ４の生成方法が必要

ウィジェット生成方法のタイプ

- Ruby/Tkから見た際の生成方法に基づくウィジェットのタイプ

(1) Tcl/Tk上の単一のウィジェットクラス

(2) Tcl/Tk上の特定のコマンドで生成するもの(複合ウィジェットを含む)

(3) Ruby/Tk上の既存のウィジェットクラスからの継承

(4) Ruby/Tk上で複数ウィジェットを組み合わせて構築する複合ウィジェット

生成方法タイプ４

- 複数ウィジェットの集合体を一つのウィジェットのように扱うもの
  - 一つの土台(フレームウィジェット)の上に構築
  - 土台に積み上げられたすべてを一つのウィジェットとして操作
- 構築の支援
  → TkCompositeモジュール

TkCompositeモジュール

- 複合ウィジェットの組み立て支援
- initializeメソッドを再定義する
  1. データベースクラス名を推定または決定する
  2. 土台となるフレームウィジェットを生成し，@frameに設定する
  3. @epathと@pathとを共に@frameのウィジェットパスに設定する
  4. initialize_compositeメソッドを呼ぶ

@epathと@path

- @epath
  - ジオメトリマネージャの管理対象となるウィジェットのパス
  - デフォルトでは土台である@frameのパスとなっており，通常は変更不要
- @path
  - 複合ウィジェットの機能上の中心となるウィジェットのパスに設定されるべきもの
  - 設定されたウィジェットがメソッド呼び出し等の主たる適用対象となる

initialize_compositeメソッド

- initializeから呼ばれ，複合ウィジェットの構築や設定を行う
- 構成要素は，土台である@frameが祖先のウィジェットとなるように作成する
- @pathの設定を忘れずに

ウィジェット属性操作の支援

- 複合ウィジェットの属性操作の適用範囲は，一般に属性ごとに異なる
- 同じ属性を持つものを複数グループに分けて操作したい場合もある
- 何らかのメソッドを用いて行う必要がある操作を，一般の属性操作とシームレスにしたい場合もある

→ TkCompositeにはこれらの設定を支援するメソッドが存在

ウィジェット属性設定の委譲

- delegateとdelegate_alias
  属性設定の適用対象ウィジェットの規定
  (delegate_aliasは属性の別名を設定)

```
delegate(〈option〉, 〈widget〉, … )
delegate(“DEFAULT”, 〈widget〉, … )

delegate_alias(〈alias〉, 〈option〉, 〈widget〉, … )
```

- option_methods
  属性として扱えるが，実体はメソッドで
  実装されているものの規定

```
option_methods([〈set〉, 〈get〉, 〈info〉], … )
```

TkComposite使用時のデータベースクラス名の設定

- 土台(@frame)のデータベースクラス名が複合ウィジェットのデータベースクラス名として扱われる
- 必要なら，以下のいずれかで指定
  i. ウィジェット作成時のclassnameオプション
  ii. 定数WidgetClassName

TkCompositeの使用例

- Rubyのソースに含まれるサンプル(ext/tk/sampleの下)を参照

※ 他にも色々なサンプルがあるので,
ぜひチェックしてみてください

まとめ

- Ruby/Tkにおいて，ウィジェットクラス作成をサポートする機能の一部を紹介した
- その他のRuby/Tkに特有の機能(の一部)については，Rubyist Magazine 0003号の「Ruby/Tkの動向」も参照を

おまけ(会場でのQ&A)

- サンプルとかチュートリアルとかは?
  - Rubyのソースに含まれるサンプルを見て欲しい
  - (会議の時には言い忘れたが)サンプルの一つとして含まれているWidgetDemoは，デモのソースを表示して，表示されたものを編集して再実行できる(元のファイルは変更されない)ので，色々と書き換えて試してみると練習になるはず
- 標準添付のメリット/デメリットは?
  - Rubyの内部構造に密に絡んでAPIを利用したり，Ruby自体の変更をお願いしたりできる点は良い．添付されているけど動きませんではまずいが，1.9対応は難しくて困っている．公式の修正や機能追加がRubyのリリースに依存してしまう点も少し問題
- JRubyで動きますか?
  - メリットが見えない．JavaユーザがTkを使いたいと考えるとは思えないので，意味がないと思う