

他言語ライブラリの利用

立石 孝彰

本資料はなるべく偽りがないように努めていますが、内容の正確さ・実現性については十分な推敲ができておりません。2次利用の際には各自の責任で行ってください。

他言語ライブラリとは？

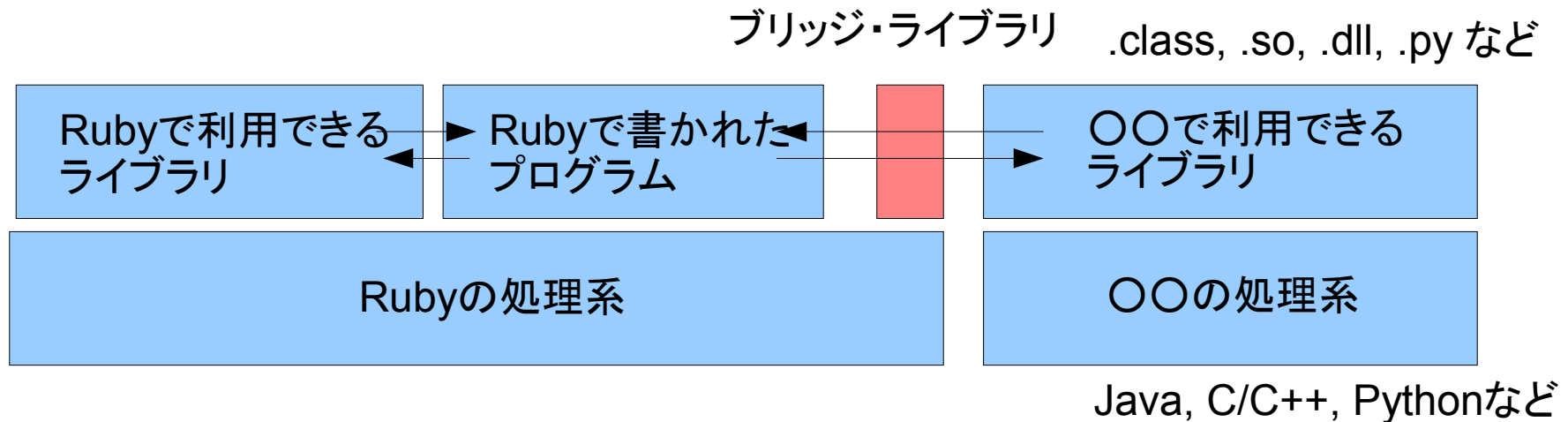
- ◆ Ruby以外のプログラム言語のライブラリ
 - ◆ Javaのクラスファイル、jarファイル
 - ◆ WindowsのDLL
 - ◆ Linux, *BSD などの共有ライブラリ (.soファイル)
 - ◆ Perl, Python などのライブラリ (.pl, .pyファイル)

Rubyにおける外部ライブラリの利用方法

- ◆ 拡張ライブラリを作成 (RubyInline含む)
 - ◆ ビルド環境が必要
 - ◆ 基本的にはC/C++ライブラリが対象
- ◆ 他言語のライブラリへアクセスするライブラリを利用
 - ◆ ビルド環境が必要ない
 - ◆ データ構造が変更されるとRubyのスクリプトも変更しなければいけないことがある。
 - ◆ コンパイラが行っている一部をプログラマが行わなければならないかもしれない。
 - ◆ C/C++以外のライブラリへもアクセスできる
- ◆ XML-RPCやSOAPを利用
 - ◆ 今回の対象外…

FFIとブリッジ・ライブラリ

- ◆ FFI (Foreign Function Interface)
 - ◆ 他言語の関数を呼び出す & 他言語から呼び出されるための機構
 - ◆ スタブの生成を伴う。
 - ◆ スタブ: 関数呼び出しの対応 + データ変換
- ◆ ブリッジ・ライブラリ
 - ◆ 他言語のライブラリへアクセスするためのライブラリ
 - ◆ スタブの生成をなるべく動的に行う。
 - ◆ 他言語へアクセスするためにビルド環境が必要ない。



Rubyにおける他言語へのブリッジ・ライブラリ

- ◆ Ruby/DL C/C++
 - ◆ <http://rubyforge.net/projects/ruby-dl>
- ◆ Rjb Java
 - ◆ <http://rjb.rubyforge.org/>
- ◆ Ruby/JS JavaScript
 - ◆ <http://www.rubyist.net/~tamura/ruby/ruby-js/>
- ◆ Ruby/Python Python
 - ◆ <http://www.goto.info.waseda.ac.jp/~fukusima/ruby/python-e.html>
- ◆ Ruby/Tk Tcl/Tk

Ruby/DL (Ruby/DL2)

- ◆ 対象言語: C/C++
- ◆ 交換可能なデータ:
 - ◆ 基本データ(整数、文字列、配列など)を扱えます。
 - ◆ 構造体やコールバックも可能です。
 - ◆ イテレータを使ってコールバックを指定できます。
- ◆ その他:
 - ◆ 最近開発が滞り気味です。
 - ◆ 呼び出す関数の型(シグネチャ)を明示的に指定する必要があります。

Ruby/DL2: サンプル

```
require 'dl/import'

module LIBC
  extend DL::Importer          # ブリッジのためのメソッドを用意
  dlload LIBC_SO              # libc.so をロード

  extern "int qsort(void*, int, int, void*)" # qsort用のメソッドを定義

  # qsort用のコールバック
  QsortCallback = bind("void *qsort_callback(void*, void*)", :temp)
end

buff = "9341"
LIBC.qsort(buff, buff.size, 1,
  LIBC::QsortCallback){|ptr1,ptr2| ptr1[0] <=> ptr2[0]} # qsortを実行
p buff # "1349" を出力
```

Rjb: Ruby-Java Bridge

- ◆ 対象言語: Java
- ◆ 交換可能なデータ:
 - ◆ 基本データやJavaオブジェクトを扱えます。
 - ◆ コールバックも可能です。
 - ◆ インターフェイスを実装したオブジェクトをRuby側で作れます。
- ◆ その他:
 - ◆ オーバーロードされたメソッドではシグネチャの指定が明示的に必要な場合があります。

Ruby/JS

- ◆ 対象言語: JavaScript (ECMAScript)
- ◆ 交換可能なデータ:
 - ◆ 基本データだけです。
- ◆ その他:
 - ◆ JavaScriptをRubyの中で動かすことができるライブラリです。

サンプル:

```
require 'js'
```

```
a = JS::evaluate("1")
```

```
p [a.class, a] # [Fixnum, 1]
```

```
a = JS::evaluate("'a'")
```

```
p [a.class, a] # [String, "a"]
```

```
a = JS::evaluate("new Object()") # エラー
```

```
p [a.class, a]
```

Ruby以外の言語では？

- ▶ Perl C::DynaLib モジュール
 - ▶ Perl DynaLoader モジュール
- ▶ Python dl モジュール
- ▶ Gauche c-wrapper
- ▶ Perl Win32::API
- ▶ Perl FFI

Perl C::DynaLib

- ◆ 対象言語: C/C++
- ◆ 交換可能なデータ:
 - ◆ 基本データ、コールバック、構造体を扱えます。

サンプル:

```
use C::DynaLib;
$libc = new C::DynaLib("cygwin1.dll");

sub compare_lengths {
    length(unpack("p", $_[0])) <=> length(unpack("p", $_[1]));
}
$callback = new C::DynaLib::Callback("compare_lengths", "i", "p$ptr_len", "p$ptr_len");
$qsort = $libc->DeclareSub("qsort", "", "P", "I", "I", PTR_TYPE);

@list = qw(A bunch of elements with unique lengths);
$array = pack("p*", @list);
&$qsort($array, scalar(@list), length($array) / @list, $callback->Ptr());
@result = unpack("p*", $array);
print join(", ", @result), "\n";
```

Python dlモジュール

- ◆ 対象言語: C/C++
- ◆ 交換可能なデータ:
 - ◆ 整数と文字列のみ扱えます。
 - ◆ それ以外はpackを使います。
 - ◆ 関数の返り値はlongでなければならない。
- ◆ その他:
 - ◆ (私が)良いサンプルを見つけることができなかった。

サンプル:

```
import dl, time
h =
dl.open('/usr/bin/cygwin1.dll')
if h.sym('time'):
    print(h.call('time'))
print(time.time())
```

出力:

```
1181289585
1181289585.77
```

Gauche FFI

- ◆ 対象言語: C/C++, ObjectiveC
- ◆ 交換可能なデータ:
 - ◆ 基本データ、コールバック、構造体を扱えます。
- ◆ その他:
 - ◆ Cヘッダファイルをパースして自動的にシグネチャを判断します。
 - ◆ libffiを利用しています。
 - ◆ 現状では最も進んだCへのブリッジ・ライブラリではないだろうか？

サンプル:

```
(use c-wrapper)  
(c-load "stdio.h" :libs "-lc")  
(printf "Hello, world\n")
```

共通機能

- ◆ ライブラリ/ファイルのロード
 - ◆ 必ず必要です。
- ◆ クラス、モジュールの定義
 - ◆ 対象言語に依存します。
- ◆ シグネチャの指定
 - ◆ Rjbでもメソッド・オーバーロードへの対応に必要です。
 - ◆ シグネチャを全く指定できないことは逆に不自由です。
 - ◆ シグネチャは、関数呼び出しやデータの変換の方法の指定方法として利用されます。
 - ◆ ブリッジ・ライブラリ毎で期待とは異なる変換を行うかもしれない。
- ◆ コールバックのためのスタブ
 - ◆ 対象言語上に、Rubyへアクセスするためのオブジェクトやスケルトンコードを静的/動的に用意する必要があります。

共通機能の実現方法

▶ ライブラリ/ファイルのロード

```
Rjb::load(); import "org.example.foo.Foo"  
dload "foo.dll"
```

▶ クラス、モジュールの定義（対象言語に依存）

```
Foo = import("org.example.foo.Foo") # 定数を利用  
module Foo                          # モジュールを利用  
  dload "foo.dll"  
end
```

▶ シグネチャの指定

```
extern "void foo(void*, char*)", :foo_str
```

▶ コールバック

```
# DL内に用意されたRubyへアクセスする関数と関連付ける  
bind "void my_cb(void*, void*)"  
# JVMに用意されたプロキシを介してobjのメソッドが呼ばれる  
Rjb::bind(obj, "java.util.Comparator")
```

こんなことができる嬉しい？（その1）

Javaの場合：

```
require 'java.util.HashMap' of Java
m = HashMap.new()
```

C/++の場合：

```
require 'libc' of C # Javaの場合とほぼ同様
import void qsort(:void*, :size_t, :size_t, :void*) # シグネチャの指定
buff = "9341"
qsort(buff, buff.size(), 1, proc{|ptr1, ptr2| ptr1[0] <=> ptr2[0]})
```

C/C++の場合(その2)：

```
module Libc
  require 'libc' of C
  # 配列用と文字列用に別々の名前で宣言 + デフォルト値/イテレータも指定
  import void qsort(:char*, :size_t=$1.size(), :size_t=1, :_) as qsort_str
  import void qsort(:char*[], :size_t=$1.size(), :size_t=4, :_) as qsort_ary
end
buff = "9341"
Libc.qsort(buff){|ptr1, ptr2| ptr1[0] <=> ptr2[0] }
```

こんなことができる嬉しい？（その2）

- ◆ Javaのメソッドをコールバック関数としてCの関数を呼び出す。
 - ◆ 現状でも技術的には可能
 - ◆ しかし、Java/C 以外も考えると規約や共通プラットフォームが必要

サンプルコード:

```
module MixedLib
  import 'libc' of C # Cライブラリをインポート
  import 'org.example.MyComparator' of Java # Javaのクラスをインポート
end
```

Javaの Comparator を使って C の qsort を実行

```
MixedLib.qsort(["a","b","c"], method(MixedLib::MyComparator.compare))
```

今後の期待

- ◆ ブリッジ・ライブラリのための共通フレームワーク/ライブラリ
 - ◆ 複数の他言語のデータの取り扱いを統一します。
 - ◆ 利用者はほとんど同じマナーで他言語ライブラリを利用できます。
- ◆ ブリッジ・ライブラリ用の文法 or DSL
 - ◆ スタブ生成用の言語内DSL

まとめ

- ◆ ブリッジ・ライブラリ: 他言語ライブラリのためのライブラリです。
- ◆ 拡張ライブラリとは補完的な関係です。
 - ◆ 面倒なデータ変換を考えるくらいなら拡張ライブラリを作る。
 - ◆ 他言語ライブラリのためのRubyライブラリをブリッジ・ライブラリを使って作るというのは現実的ではない(と思う)。
 - ◆ 限定された簡単なことを行うならブリッジ・ライブラリを使う。
- ◆ ブリッジ・ライブラリ共通の仕組みがあると良いかもしれない。
 - ◆ ユーザにとっては、ブリッジ・ライブラリ毎に新しいマナーを覚える必要はない。
 - ◆ 開発者にとっては、Rubyらしく扱えるように実装する部分が面倒だと思うので、共通化できないだろうか？