非同期

- Tatsuhiro Ujihisa
- http://ujihisa.blogspot.com/

一般的な話

- 非同期とは?
- スレッド、プロセス
- プロセス生成
  - system, IO.popen, Open3.popen3, fork

非同期とは?

- "非"非同期とはすなわち逐次処理。上から下へ一本の処理が進む
- 非同期の場合、複数の処理が同時に進む

スレッドとプロセス

- Rubyを立ち上げると、OSが管理するプロセスが一つ立ち上がる
- Rubyでスレッドを作ると、Rubyプロセスの中で仮想的に処理の流れが増える
- Rubyでプロセスを作ると、OSが管理するプロセス自体が増える

スレッドの作り方

```
Thread.start do
  something
end
```

プロセスの作り方

- systemで`&`をつける
- IO.popenする
- forkする

それぞれ詳しく解説すると、

systemで&

```
system 'ls &'
```

- 簡単に外部コマンドを非同期実行
- 出力をRubyで受け取れない
- プロセスID取得不可能
  - $?は"sh -c 'ls &'"のプロセスID
- Windowsで動かない

IO.popen

```
IO.popen('ls') {|io|
  io.gets
```

- 出力をRubyで受け取れる!
- プロセスID取得可能! (`io.pid`)
- 標準エラー出力はそのままターミナルに出てしまう

Open3.popen3

```
Open3.popen3('ls') {|i, o, e
```

- 標準エラー出力も取得可能!
- でもプロセスIDが取得できない...
  - 深い事情がある

fork

```
fork { something }
```

- 起動中のrubyプロセス自体をコピーして、ブロックを非同期に実行
- プロセスIDはブロックの返り値
- ブロック内は別世界。便利!
- Windowsで動かない

Threadとforkの違い

```
Thread.start { $a = 1 }
fork { $a = 1 }
```

- スレッドの場合元の`$a`が変化する。
- forkの場合変化しない。

exec = system + exit

```
exec 'ls'
p 'this message will never show'
```

- 実行中のプロセス自体がexecの引数のコマンドになる
- forkと組み合わせると...!!

具体的な話

具体的な問題と、その解決方法を考えよう!

例題1: Sinatraを動かして、終了したい

解法1: Threadとsystem

```
t = Thread.start do
  system 'ruby aaa.rb'
end
# something
t.kill!
```

ダメ

- t.kill!でSinatraが終了しない
- 元のRubyが終了してもSinatraが生き残る。psしてkill

解法2: Threadとsystem, ただしシェル経由しない

```
t = Thread.start do
  system 'ruby', 'aaa.rb'
end
# something
t.kill!
```

ダメ

- Thread死すともsystem死なず

解法3: IO.popen

```
io = IO.popen('ruby aaa.rb')
# something
Process.kill 'KILL', io.pid
```

惜しい

- きちんとSinatraを終了できる!
- でも大量なアクセスログが

解法4: Open3.popen3

```
stdin, stdout, stderr = Open3.popen3('ruby', 'aaa.rb')
# something
Process.kill 'KILL', io.pid...?
```

もっとダメ

- アクセスログは隠せる
- でもプロセスIDが分からず、終了できない...

解法5: forkとsystem

```
pid = fork { system 'ruby', 'aaa.rb' }
# something
Process.kill 'KILL', pid
```

ぜんぜんダメ

- 終了できない "fork死すともsystem死なず"
- 出力隠せない

解法6: forkとexec

```
pid = fork do
  exec 'ruby, 'aaa.rb'
end
# something
Process.kill 'KILL', pid
```

惜しい

- 終了できる!
- 出力隠せない

解法7: 出力先を変えてからexec

```
pid = fork do
  STDERR.reopen File.open('/dev/null', 'w')
  exec 'ruby, 'aaa.rb'
end
# something
Process.kill 'KILL', pid
```

できた!

- ただ、forkはWindowsで動かない

解法8: spawnを使う (new!)

```
pid = spawn 'ruby', 'aaa.rb', :out => '/dev/null'
# something
Process.kill 'KILL', pid
```

- ruby 1.9から利用可能
- 便利。簡潔。

整理

- ruby 1.9 + UNIX: 最強。ラクなspawnか、面倒なfork+exec
- ruby 1.8 + UNIX: ちょっと面倒だけどfork+exec
- ruby 1.9 + Windows: spawnがあるから問題なし
- ruby 1.8 + Windows: あ...

(win32-open4という拡張ライブラリで一応解決)

実際のところ

- Macならば最初から1.8が入っているのでそれを使いたい

- Winはそもそも Rubyが入っていないので1.9を入れさせればいい
- 一つのコードをMacとWinでコンパチブルにしたければ?
  - fork+execとspawnの両方で書いてプラットフォームで分岐しないといけない
- バグの温床

解決!

Pure Rubyなspawn実装

```
gem install sfl
```

- Spawn for Legacy
- 作者: ujihisa

使い方

```
require 'rubygems'
require 'sfl'
```

- これでspawnが定義される
- ほとんどruby 1.9のspawnとコンパチブル

使用例

```
pid = spawn(
  {"GIT_EXEC_PATH" => "/usr/bin"},
  'git', 'pull', 'origin', 'master',
  [:out, :err] => ['/tmp/log.txt', 'a'],
  :chdir => '~/git/aaa')
sleep 10
Process.kill 'KILL', pid
```

- 環境変数指定、コマンド、リダイレクト、カレントディレクトリ
- 全てruby 1.9標準添付メソッドspawn準拠

まとめ

- Thread.start, system, exec, fork, IO.popen, Open3.popen3, spawn, ...
- spawnが一番便利
- ruby 1.8でもspawn利用可能

おわり

参考文献

- http://ujihisa.blogspot.com/2010/03/all-about-spawn.html
- http://ujihisa.blogspot.com/2010/03/how-to-run-external-command.html
- TokyoRubyKaigiのakrさんのspawn, open3に関するプレゼン (!)

時間的な都合で削ったもの

- ruby 1.9のopen3は大幅書き直しですごい便利
  - spawnを使っている
  - pid取得可能。実質open4
- sflでopen3を動かしたい
  - そのままでは動かないが、open3を少し書き換えれば動くはず
  - 開発中
- gemのopen4という選択肢も
  - 互換性なし
  - windowsで動かないがopen4拡張ならOK
- 実はopen系ライブラリは乱発している
  - `gem search open -r`
- 本当に必要なには外側のopenではなく基盤のspawn
  - pure ruby実装のspawnがあると思ったが探しても見つからなかったので自作
  - みんなspawn知らない?
- pure rubyであることのうまみ
  - forkとexecがあればどこでも動く。JRuby, MacRuby, Rubinius!
  - マルチプラットフォーム対応しやすい (まさに必要だった)
- forkしてexecしたプロセスの出力は事前にSTDOUTをreopenする以外に、DRbを使う方法も
  - gemのforky。Enumerable#mapのようなことをマルチプロセスで行なうライブラリ。