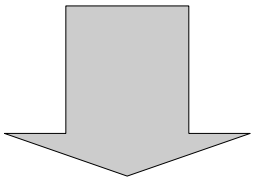
# matzを説得する方法
# How to persuade matz

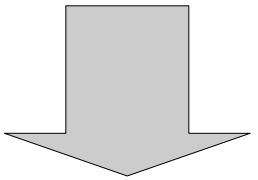
田中 哲
akr@fsij.org
産業技術総合研究所 / FSIJ
2008-06-22

# Ruby の変化
# Ruby development

誰かが何かを提案する
Someone propose something



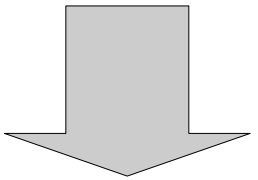
matz が受け入れる
matz accepts



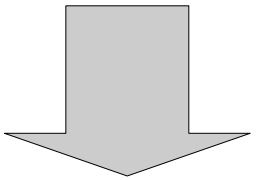
提案が採用される
It is merged into Ruby

# YARV

ko1 が YARV を提案する
ko1 propose YARV



matz が受け入れる
matz accepts



YARV が採用される
YARV is merged into Ruby

# 仮定
# Assumption

あなたは Ruby に不満がある

You have requests for Ruby

# 問題
# Problem

不満を解決するため
Ruby を変えるには
matz を説得する
必要がある
You must persuade
matz.

# YARV

- Ruby は遅い
  Ruby is slow
- YARV を採用すると速くなる
  YARV makes Ruby fast
- Ruby が速いと幸せ
  It is happy if Ruby is fast

速度ならわかりやすいが
速度以外なら?
Speed is easy to understand.
But how others?

# バグレポートは通りやすい
# Bug reports are easy

- 明らかに変なこと示す
  - SEGV
  - 実装とドキュメントの差異
- 再現可能な報告をする
  - SEGV するコード
  - 実行例をコピペ
- 余計な要求をしない
  - 憶測に基づいた修正の指示などはしない

客観的にできる

- Some issue is clearly a bug
  - SEGV
  - Difference between implementation and documents
- Reproducible bug report
  - code for SEGV
  - example of run
- Don't overrequest
  - fix by possibly wrong assumption

Be objective

# 新機能・機能変更
# New feature / Change behavior

- バグレポートよりは難しい
  Difficult than bug reports
- 変更を行う動機は何か?
  What is the motivation?
- 提案された解決策は適切か?
  Is the solution is appropriate?
- その他の要素
  Other issues
  - 一貫性　　consistency
  - 名前問題　naming problem

# いろんな問題例
# Various problems

- stacktrace の途中が省略されて情報が足りない
  skip in stacktrace
- ファイルから相対パスで require したい
  require a file in relative path
- ポータブルにリダイレクトできない
  redirects portably
- ノンブロッキング I/O を簡単に使いたい
  non-blocking I/O
- ブロックのラッパーを書きたい
  block wrapper

# 例: stacktrace の途中が省略される
# skip in stacktrace

```
% ruby-1.8.6 -e 'def x() y end; def y() x end; x'
-e:1:in `x': stack level too deep (SystemStackError)
        from -e:1:in `y'
        from -e:1:in `x'
        from -e:1:in `y'
        from -e:1:in `x'
        from -e:1:in `y'
        from -e:1:in `x'
        from -e:1:in `y'
        from -e:1:in `x'
         ... 7211 levels...
        from -e:1:in `x'
        from -e:1:in `y'
        from -e:1:in `x'
        from -e:1
```

途中の情報が出ない
skipped

省略させたくないことがある
some don't like the skip

# 数多くの挑戦
# Many Challenges

- 2002-11-17 [ruby-talk:56054] Tom Clarke
- 2003-07-03 [ruby-talk:75008] Nigel Gilbert
- 2003-11-06 [ruby-talk:84751] Dmitry Borodaenko
- 2003-11-19 [ruby-list:38810] usa
- 2003-11-26 [ruby-talk:86343] Tobias Peters
- 2004-10-10 [ruby-talk:115962] Alexey Verkhovsky
- 2005-10-06 [ruby-talk:159209] Eric Mahurin
- 2005-11-05 [ruby-core:6572] Hugh Sasse
- 2006-04-05 [ruby-talk:187642] Sylvain Joyeux
- 2007-06-17 [ruby-dev:31014] akr (変更成功 succeed)

# 典型的な返事 (1)
# Typical Response (1)

[ruby-list:38811]

|というわけで、このバックトレースの表示行数を、固定値ではなく、
|スクリプトの実行時に指定できるインターフェースが欲しいのですが、
|いかがでしょう?

```
begin
  foo
rescue => e
  cb = e.backtrace
  print cb.shift, ":", e.message, "¥n"
  cb.each{|c| print "¥tfrom ", c, "¥n"}
  exit 1
end
```

とでもすればいくらでも表示できると思うのですが、それではだめ
ということ?

# 典型的な返事 (2)
# Typical Response (2)

[ruby-talk:86352]

|Is there a way to tell ruby that it must never skip levels in the
|backtrace like this:

```
begin
  foo
rescue => e
  cb = e.backtrace
  print cb.shift, ":", e.message, "¥n"
  cb.each{|c| print "¥tfrom ", c, "¥n"}
  exit 1
end
```

matz.

# 提案されたいろいろな解決策 (1)
# Various Solutions Proposed (1)

- 自分で表示するコードを書く (matz)
  Show it yourself (matz)
- TRACE_HEAD,TRACE_TAIL を変えてリコンパイル
  Change TRACE_HEAD and TRACE_TAIL then recompile
- CFLAGS で -DTRACE_TAIL=40 -DTRACE_HEAD=20 とできるようにする
  Make them specifiable in CFLAGS
- Exceptionのクラスメソッドで行数を設定
  New class method of Exception to set them

# 提案されたいろいろな解決策 (2)
# Various Solutions Proposed (2)

- 全て表示する新規コマンドラインオプションを作る
  New command line option
- -d, -w がついていたら全て表示
  Don't skip if -d or -w
- ~/.rubyrc を導入してそこで設定
  Introduce ~/.rubyrc
- SystemStackError以外では全て表示 (akr)
  Skip only if SystemStackError (akr)

# SystemStackError 以外では全て表示
# Skip only if SystemStackError

受け入れられた理由 Why it is accepted

- 省略の意図を尊重している
  It respects the intent of the skip
  SystemStackError ならたしかに長すぎる
  stacktrace is too long if SystemStackError
- 設定がない No configuration is good configuration
  人間の嬉しいことを精度良く推測する
  It estimates programmer's expects
- 繰り返し繰り返し繰り返し要求が出ている
  The request is repeated
  ML で話題が出た 8回を示した
  I showed the 8.

# 他の案
# Other proposals

受け入れられなかった理由

Why they are not accepted

- 設定がある
  Configuration
  - クラスメソッド　Class method
  - コマンドラインオプション　Command line option
- 話が大きすぎる　Too big issue
  - ~/.rubyrc　悪影響がありすぎる可能性が高い
    It may have many problems

# 受け入れられやすい提案 (内容)
# Good Proposals

- 必要性が納得できる necessity
  - 具体的に問題があることを示す real problem
  - いろんなひとが困っている many peple faced
  - 計算量が必要以上に悪い complexity problem
- 解決策が妥当 appropriate solution
  - 人間の期待をうまく推測する estimate expection
  - 副作用が少ない less side effects
  - 変更が小規模 small change
  - POSIX や C言語が採用している POSIX, C
  - Perl が採用している Perl

# 受け入れられにくい提案 (内容)
# Bad proposals

- 必要性が納得できない unnecessary
  - なにが問題なのか分からない unclear problem
  - 本人以外が困っているのか疑わしい only one man
  - 効果がはっきりしない高速化 unclear speedup
- 解決策が疑わしい suspicious solution
  - 人間の期待に背く unexpected behavior
  - 新たな問題を発生させる many side effects
  - 変更が大規模 (~/.rubyrc, Indexer) big change
  - 独特な (C や Perl とは異なる) 解決法 unique

# 受け入れられやすい提案 (内容以外)
# Good Proposals

- 開発版に対する提案　　development version
  - 1.9
- リリースの直前でない　not just before a release
- 余計な要求が入っていない　　minimal request

# 受け入れられにくい提案 (内容以外)

- 安定版に対する提案　　stable version
  - 1.8
- リリースの直前　　just before a release
  - プレビューが出たとき　　when preview release
- 余計な要求が入っている　　not minimal
  - Random と srand　(抱き合わせ)　　tie-in sale

# 受け入れられにくい問題提起の例
# Example of Bad Proposal

```
% ruby-1.8.6 -e 'def x() y end; def y() x end; x'
-e:1:in `x': stack level too deep (SystemStackError)
        from -e:1:in `y'
        from -e:1:in `x'
        from -e:1:in `y'
        from -e:1:in `x'
        from -e:1:in `y'
        from -e:1:in `x'
        from -e:1:in `y'
        from -e:1:in `x'
         ... 7211 levels...
        from -e:1:in `x'
        from -e:1:in `y'
        from -e:1:in `x'
        from -e:1
```

途中の情報が出ない
skipped

省略させたくないことがある
some don't like the skip

# もっと現実的な例を使う
# Use practical example

```
...
test_condvar_wait_not_owner(TC_Thread): .
test_local_barrier(TC_Thread): ./lib/timeout.rb:54:in `sysread':
execution expired (Timeout::Error)
        from ./lib/net/protocol.rb:133:in `rbuf_fill'
        from ./lib/timeout.rb:56:in `timeout'
        from ./lib/timeout.rb:76:in `timeout'
        from ./lib/net/protocol.rb:132:in `rbuf_fill'
        from ./lib/net/protocol.rb:116:in `readuntil'
        from ./lib/net/protocol.rb:126:in `readline'
        from ./lib/net/http.rb:2020:in `read_status_line'
        from ./lib/net/http.rb:2009:in `read_new'
          ... 26 levels...
        from ./lib/test/unit/ui/testrunnerutilities.rb:29:in `run'
        from ./lib/test/unit/autorunner.rb:216:in `run'
        from ./lib/test/unit/autorunner.rb:12:in `run'
        from ./test/runner.rb:7
```

# その他の要素
# Other issues

- 一貫性　　consistecy
- 名前問題　　naming problem
- 互換性　　compatibility
- 対面で言う　　face-to-face request

# 一貫性
# Consistency

- 一貫性は Ruby の目標ではない
  Consistency is not a goal of Ruby
  - 説得材料になることもある　　may be good reason
  - 説得材料にならないこともある may be bad reason
- 実際に役に立つなら説得材料　good if useful
  - 多態性　polymorphism
    0x3fffffff.div(2.0) と 0x40000001.div(2.0)
- 一貫性よりも便利であることが重要
  Usefulness is important than consistency
  - obj.m がメソッドオブジェクトを返すようになるか?
    obj.m doesn't return an method object.

# 名前問題
# Naming Problem

- 機能には問題ないのに名前が決まらないだけが理由で受け入れられない
  Rejected just because no good name
  - readpartial
  - read_nonblock
- 話題がでるたびに「名前さえ決まれば入るんだけど」と言う
  "good name is the last problem" for each time when some requests
- ruby-core, ruby-talk で行えば英語 native のひとのコメントが得られる
  English native speakers will comments if ruby-core and ruby-talk.

# 互換性
# Compatibility

- 完全な互換性は求めない
  Perfect compatibility is not required
- 単純で長期的にうまくいくほうが良い
  Simple solution is better
- 短期的な痛みを和らげることも必要
  Short term compatibility problem may needs some workarounds.

# 対面で言う
# Face to face requests

- Ruby会議とかで matz に会って尋ねる
  Ask matz at RubyConf, etc.
- なんらかの反応は得られる
  You can get some response
- 難しい問題はやっぱり難しい
  Difficult problems are still difficult

# strftime で小数点以下
# strftime and fraction

Timeオブジェクトはusecを持っていますが、
今のstrftimeの書式指定文字列ではusecや
ミリ秒が取り出せません。

Although Time objects has usec,
current strftime format cannot extract usec
or milli seconds.

[ruby-dev:34978]

# 提案の問題点
# Problems of the proposal

- どういう用途で欲しいのか書いてない　No usage
  - どういう出力形式がいいのか?　Output format
  - 3桁? 6桁? 桁数指定?　3-digits, 6-digits or ...
  - 下位桁の 0 はどうする?　lower zeros?
- 1.8 を対象としている　targets 1.8
  1.9 なら usec (マイクロ秒)ではなく nsec (ナノ秒)
- C の strftime は秒単位しか扱わない　difference
  扱うようにすれば C と違ってしまう　to C
  独自に %X を決めると将来的に他と衝突するかも
  It may conflicts if original %X is introduced
- 実装で C の strftime を使えるか?　impl?

# まずすべきこと:サーベイ
# 他ではどう扱っているか

- C 言語
  - strftime は time_t を受け取るので小数点以下は扱えない　　don't use fraction
- Perl
  - POSIX モジュールの strftime は小数点以下は受け取らない　　don't use fraciton
- Python
  - time モジュールの strftime は小数点以下は受け取らない　　don't use fraction

Cのstrftime影響下のは参考にならない
C like lang. don't use fraction

ここで IRC で情報提供がありました
ありがとうございます

takesakoさん曰く:
一応、PerlでもDateTimeモジュールで%N記法使えます
http://search.cpan.org/dist/DateTime/lib/DateTime.pm#strftime_Patterns

# サーベイ (2)

- GNU date
  - %N で ナノ秒 (000000000..999999999)
  - % date +%N
    298688729
- Java (java.util.Formatter)
  - %L でミリ秒 (000 - 999)
  - %N でナノ秒 (000000000 - 999999999)
- C# にもある(カスタム DateTime 書式指定文字列)
  - strftime とは書式指定が違う?
  - fffff とか指定した f の数だけ

# より良い提案

- 用途を書く　　Describe usage
  - %N が適切な用途だととても良い (なんとか見つける)
    Usage for %N is desired (Find it)
- %N を提案する　　Propose %N
  - GNU date でも Java でも使っている
    GNU date and Java use it
- 現実的に実装できることを示す　　practical impl.
  - missing/strftime.c を改造?
  - % でぶつ切りにしてフォーマットしてから連結?
  - 話を小さくするなら後者
  - locale と strftime という話も有るのであまり変えない
    [ruby-talk:216305] [ruby-talk:286696]

# いつも完璧な提案はできない
# Not always perfect

- 大規模 Big change
  - spawn でリダイレクトなど spawn and redirects
  - stdio 排除 stdio-less IO
  - YARV YARV
- 独特な解決法 unique solution
  - readpartial
  - read_nonblock
  - Regexp#to_s
  - NotImplementedError なメソッドと respond_to?
- それでも可能な限り良い要素を増やすようにする
  Try better proposal anyway

# まとめ

- 用途を書く
  Describe usage
- どう使うかを書く
- なんで欲しいのか書く
- どう役に立つのか書く
- なんで必要なのか書く
- どう問題を解決できるのか書く
- なんで嬉しいのか書く
- どう幸せなのか書く
- 解決策を書くならなるべく妥当なものにする
  Describe appropriate solution if a solution is described