Railsによるメタプログラミング入門

株式会社ネットワーク応用通信研究所
前田 修吾

自己紹介

- 前田修吾(Shugo Maeda)
- Schemeと同い年
- 家族
  - 奥さん
  - 娘
  - 息子
- NaCl勤務(1999～)

リクエストがありましたので...

一応息子も

Rubyとの出会い

- 1997年 JavaHouse-Brewers MLにて
- 恩人　高木浩光先生

Rubyへの貢献

- 標準添付ライブラリ
  - curses
  - readline
  - monitor
  - net/ftp
  - net/imap
- ruby-lang.org管理者
- 細かい仕様提案など

Ruby関係のプロジェクト

- mod_ruby
- eruby
- ximapd

Ruby関係の書籍

- 『Rubyアプリケーションプログラミング』共著
- 『Rubyによるアジャイルwebアプリケーション開発』監訳

本日のテーマ

- Railsを肴にメタプログラミングについてお話し

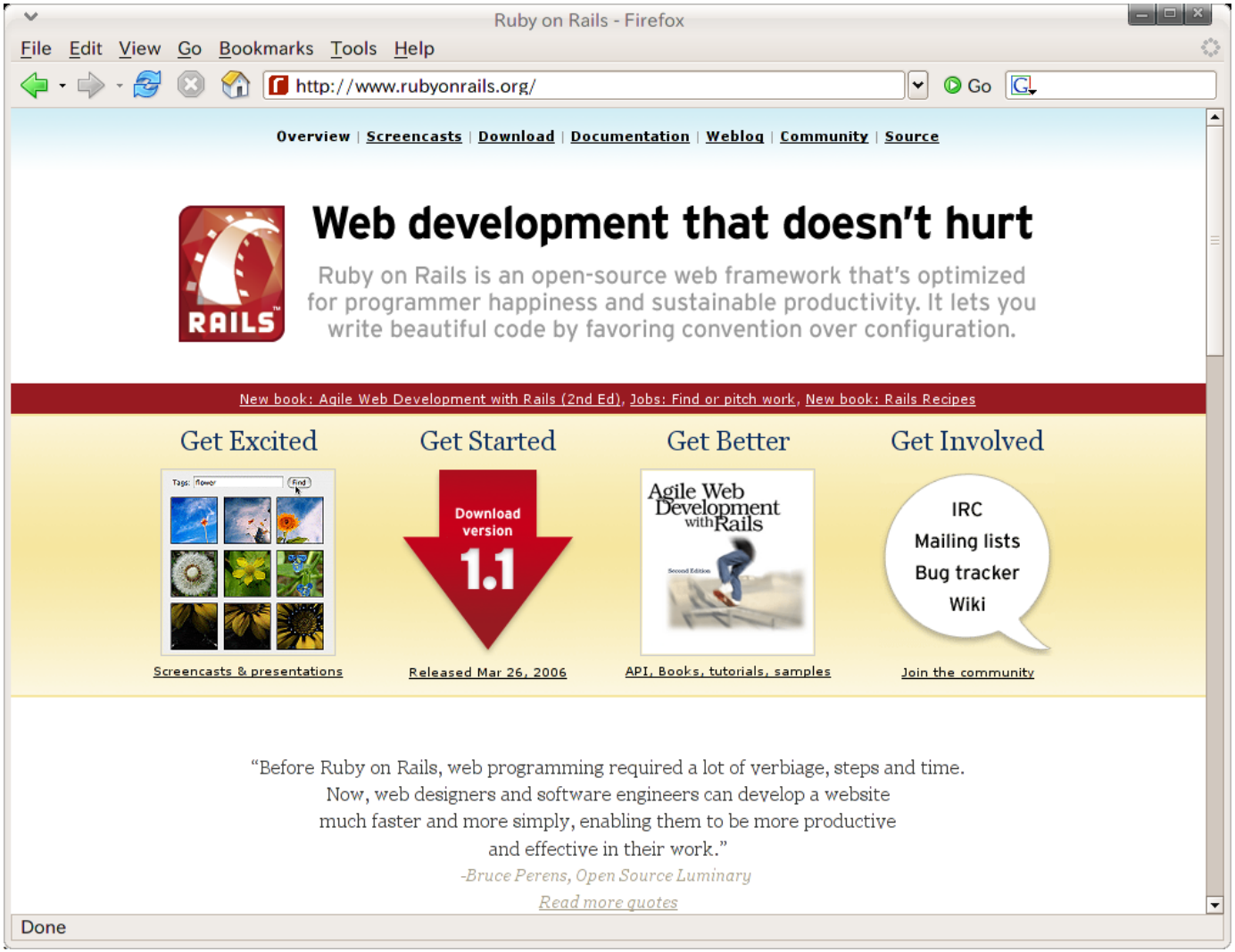
Ruby on Railsとは

- Rubyの
- Rubyによる
- Rubyのための
- Webアプリケーションフレームワーク

Railsの成功の理由

- DRY(Don't Repeat Youself)
- CoC(Convention over Configuration)
- Javaの10倍の生産性
- TODOリストとかブログが10～15分でできるらしい

Railsの成功の理由(2)




RAILS

Before


オブジェクト指向言語Ruby - Firefox
File Edit View Go Bookmarks Tools Help
http://www.ruby-lang.org/ja/
Go
The Object-Oriented Scripting Language
Ruby
オブジェクト指向スクリプト言語
Ruby
Top
Rubyとは
What's Ruby?
ライセンス
ダウンロード
Ruby本体
ドキュメント
ドキュメント
インストールガイド
チュートリアル
リファレンスマニュアル
FAQ
www.ruby-lang.orgの歩き方
その他のドキュメント
コミュニティ
メーリングリスト
日本Rubyの会
開発
CVSリポジトリガイド
セキュリティ問題
«前8件
日本Rubyカンファレンス2006開催
[2006-05-02] by usa
来る2006年6月10日(土)、11日(日)の二日間に渡って、日本での初のRubyオンリーな大規模イベントである日本Rubyカンファレンス2006が開催されることが発表されました([ruby-list:42182])。
Ruby開発者のまつもとさんと、Rails開発者のDavid Heinemeier Hanssonさんの基調講演のほか、多くのスピーカーによる興味深いセッションが多数用意されています。チケットは5月9日よりローソンチケットで発売開始されるということですが、早期に売り切れることも予想されるので、興味のある方はお早めにお買い求めください。
Last update on May 02, 2006 17:42
Ruby 1.8.4 リリース!!
[2005-12-24] by maki
[ruby-list:41728]にて、まつもとさんより、ruby 1.8.4 を公開したとのアナウンスがありました。
1.8.4 のソースコードは以下のURLから入手可能です。
<URL:ftp://ftp.ruby-lang.org/pub/ruby/ruby-1.8.4.tar.gz>
md5sumは bd8c2e593e1fa4b01fd98eaf016329bb です。また、サイズは 4312965 バイトです。
1.8.4での変更点は、以下のページを参考にしてください。
<URL:http://www.ruby-lang.org/ja/man/?cmd=view;name=ruby+1.8.4+feature>
Language
Japanese
English
What's New RDF
ダウンロード
日本Rubyカンファレン..
Ruby 1.8.4 リリース!!
Ruby 1.8.4 preview3..
Zeta OS用Rubyバイナ..
Get Ruby now!!
安定版: 1.8.4
Ads by Goooogle
使えるコードが満載
開発者のための実装系Webマガジン
CodeZineは開発者を支援します。
codezine.jp
Done


RAILS

After


オブジェクト指向スクリプト言語 Ruby - Firefox
File Edit View Go Bookmarks Tools Help
http://new.ruby-lang.org/ja/
Go
Ruby
A Programmer's Best Friend
Search
ダウンロード ドキュメント ライブラリ コミュニティ ニュース セキュリティ Rubyとは
Rubyとは...
オープンソースの動的なプログラミング言語で、シンプルさと高い生産性を備えています。エレガントな文法を持ち、自然に読み書きができます。
もっと読む...
The Greeter class
class Greeter
def initialize(name)
@name = name.capitalize
end
def salute
puts "Hello #{@name}!"
end
end
# Create a new object
g = Greeter.new("world")
# Output "Hello World!"
g.salute
はじめよう!
Try Ruby!
ダウンロード
インストールガイド
チュートリアル
探求しよう!
ドキュメント
ライブラリ
成功事例
コミュニティに参加しよう
メーリングリスト: 世界中のプログラマとRubyについて話しましょう。
日本Rubyの会: Rubyの利用者/開発者の支援を目的としたグループです。
更新順リンク: Ruby関連のサイトのリンクを更新順に並べたものです。
トッププロジェクト
日本Rubyカンファレンス2006開催
来る2006年6月10日(土)、11日(日)の二日間に渡って、日本での初のRubyオンリーな大規模イベントである日本Rubyカンファレンス2006が開催されることが発表されました([ruby-list:42182])。
Ruby開発者のまつもとさんと、Rails開発者のDavid Heinemeier Hanssonさんの基調講演のほか、多くのスピーカーによる興味深いセッションが多数用意されていま
Done

Railsの成功の理由(3)

Railsの成功の理由(4)

Rails is fun!

フレームワークとたのしさ

- フレームワークを使うのはたのしくない
  - Don't call us, we'll call you.
- でもRailsはたのしい
  - なぜ?

Railsの成功の理由(5)

Ruby is fun!

Railsの成功の理由(6)

Metaprogramming is fun!

メタプログラミング

- メタプログラミング (metaprogramming) とはプログラミング技法の一種で、ロジックを直接コーディングするのではなく、あるパターンをもったロジックを生成する高位ロジックによってプログラミングを行う方法、またその高位ロジックを定義する方法のこと。主に対象言語に埋め込まれたマクロ言語によって行われる。

出典: フリー百科事典『ウィキペディア（Wikipedia）』

メタ

- メタ（meta-）とは、「高次な―」「超―」「―間の」等の意味の接頭語。ギリシャ語から。

  出典: フリー百科事典『ウィキペディア（Wikipedia）』

例

- メタ言語
  - 言語を記述する言語
  - 日本語で英語について言及する時→日本語=メタ言語
  - あるいはXMLとか
- メタミステリ
  - ミステリについて言及するミステリ
  - 中井英夫『虚無への供物』
  - 東野圭吾『名探偵の掟』

間違った例

通常のプログラミング

メタプログラミング

- 自己言及的な構造

例

- コード生成(何でも)
- マクロ(Lisp, C)
- テンプレート(C++)

効用

- DRY
  - 繰り返しの排除
  - 似たようなプログラムを自動生成
- 生産性の向上
- たのしい
  - WSDLを手で書きたい?
  - WSDLを生成するプログラムを書きたい?

副作用

- 読めないプログラム
  - Cのマクロ
  - 奇妙なDSL
  - 正直Railsのソースも読みにくい
- 実行効率
  - Rubyは動的なので
  - 効率のために使うことも
    - マクロ
    - テンプレート
- ハイリスクハイリターン

Railsに学ぶテクニック

- コード生成
- 言語内DSL
- ブロック
- クラスメソッド
- eval
- 既存クラスの拡張
- マクロ

コード生成

- scaffold
  - CRUD処理の自動生成
  - 10分でできる～みたいなのはこれのおかげ
- コード生成自体はRubyに特化したものではない
  - あまりおもしろくない
- bladeテクノロジー的アプローチ
  - カラム名にpasswordという文字列が含まれてたら、<input type="password">にしとけ、みたいな

DSL

- Domain Specific Language
- 特定の分野に特化 ⇔ 汎用言語
- 例
  - エディタなどのマクロ言語
  - 設定ファイル

DSLとコード生成

- パワーのない言語(!= 静的型付言語)で使用
- 例
  - XMLからJavaコードを生成

言語内DSL

- 汎用プログラミング言語でDSLを実現
- 例
  - Lispのマクロ
  - C++のテンプレート

ブロック

- Builderテンプレート

```
xml.div do
  xml.h1(@person.name)
  xml.p(@person.bio)
end
```

```
<div>
  <h1>David Heinemeier Hansson</h1>
  <p>A product of Danish Design during the Winter of
  ’79...</p>
</div>
```

クラスメソッド

- 予約語っぽいものも実はメソッド
- クラスがレシーバ

例: アクセサ

- 下のような冗長なコードが

```
class Person
  def name
    return @name
  end

  def age
    return @age
  end
end
```

例: アクセサ(2)

- こんなに簡潔に

```
class Person
  attr_reader :name, :age
end
```

has_many

- テーブル間の関係をクラスメソッドで表現

```
class Category < ActiveRecord::Base
  has_many :products
end

category = Category.find(1)
for product in category.products
  ...
end
```

has_many: 返り値

- 配列に見えるけどちょっと違う

```
products = category.products
p products[0]  #=> #<Product:0xb7367448 ...>
cheap_products =
  products.find(:all,
                :conditions => "price < 100")
```

- Enumerable#findは使えない
- 代りにdetectを使う → collect派が拡大

has_many: 返り値(2)

- やっぱり配列?

```
p category.products.class #=> Array
```

- findは特異メソッド?

```
p category.products.singleton_methods #=> []
```

- 違うようだ

has_many: 返り値(3)

- classが嘘をついてる?

```
p category.products.class #=> Array
kernel_class = Kernel.instnace_method
p kernel_class.bind(category.products).call
      #=> ActiveRecord::Associations::HasManyAssociation
```

- なぜこんなことに?

has_many: 返り値(4)

- HasManyAssociationのスーパークラス

```
module ActiveRecord
  module Associations
    class AssociationProxy #:nodoc:
      alias_method :proxy_respond_to?, :respond_to?
      alias_method :proxy_extend, :extend
      instance_methods.each { |m| undef_method m unless
m =~ /(^__|^nil¥?|^proxy_respond_to¥?|^proxy_extend|
^send)/ }
```

- classまでundef_methodしちゃってるとは

has_many: 返り値(5)

- undefされたメソッドはmethod_missingで処理

```
def method_missing(method, *args, &block)
  load_target
  @target.send(method, *args, &block)
end
```

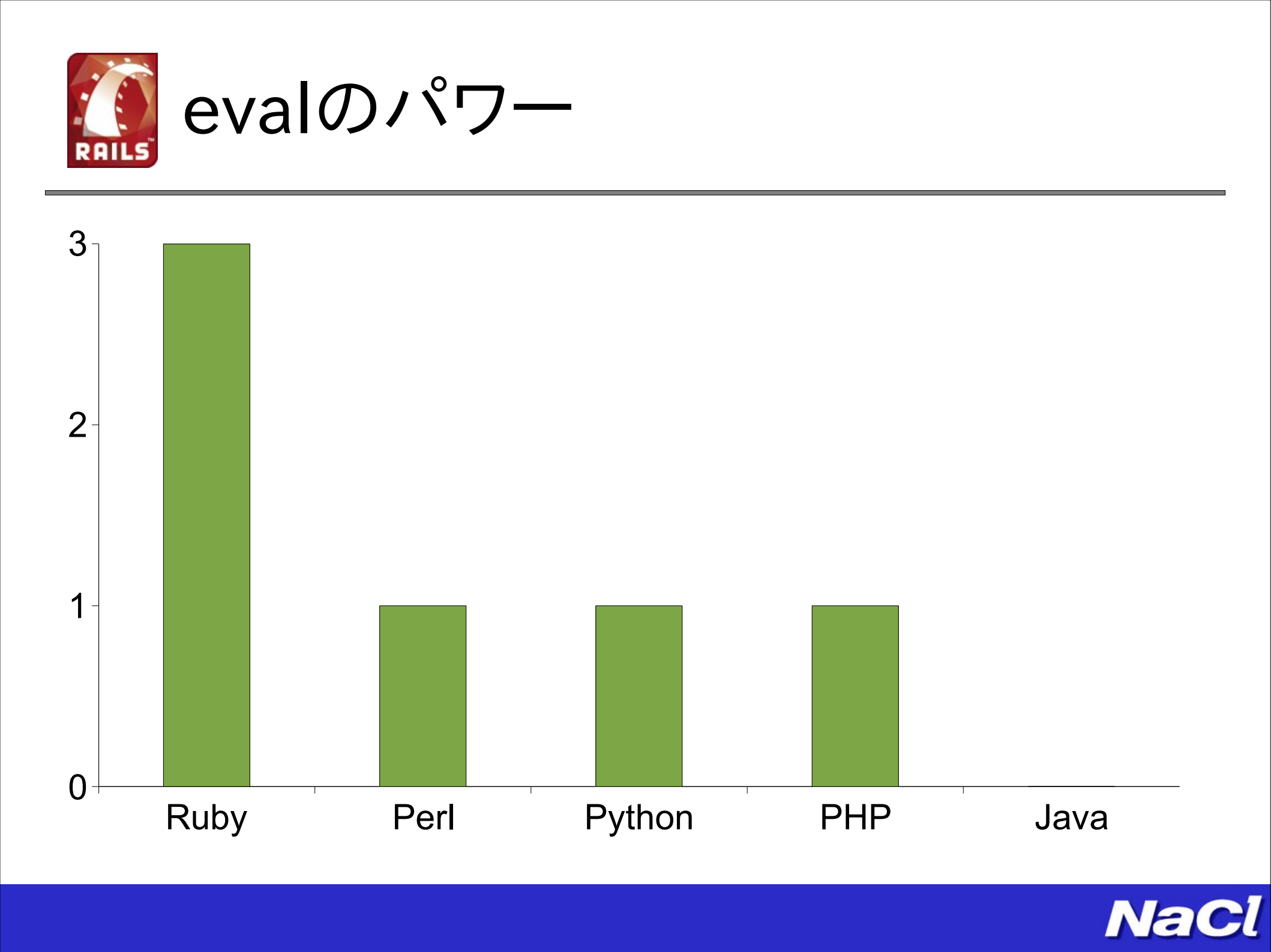
- method_missingはメソッドが見つからなかった時に呼ばれる特殊なメソッド
- load_targetで@targetに委譲先を設定
- @targetに委譲

eval

- Rubyではコードはオブジェクトではない

  ⇒コードそのものは操作できない
- evalは文字列をコードとして実行

  ⇒文字列なら操作できる
- 実行時に動的にコード生成
  - 動的に(って全部動的ですが)メソッドを定義したりとか
  - 使えるならdefine_methodの方がいいかも


evalのパワー
3
2
1
0
Ruby
Perl
Python
PHP
Java

Rubyのevalの種類

- eval
  - 普通のeval
- instance_eval
  - オブジェクト内部のコンテキストでeval
- class_eval/module_eval
  - クラス/モジュール内部のコンテキストでeval

Railsとeval

- Railsはevalを多用

```
$ ls
actionmailer/  actionwebservice/  activesupport/
actionpack/    activerecord/      railties/
$ find . -name '*.rb' | xargs egrep '¥b(|instance_|
  class_|module_)eval¥b'|wc -l
429
```

evalを使うリスク

- セキュリティ
  - ブラウザから受け取ったパラメータをevalとか
  - $SAFE = 4 でもセキュアとはいえない
- 実行速度
  - オーバーヘッドがあるため遅い
  - Ruby 2.0ではもっと遅くなるらしい
    - evalは使うなという教育的配慮

既存クラスの拡張

- Rubyのクラスは再オープン可能
- 定義済みのクラスにメソッドなどを追加

標準クラスの拡張

- active_support/core_ext
- たとえば、Symbol#to_proc

```
class Symbol
  def to_proc
    Proc.new { |obj, *args| obj.send(self, *args) }
  end
end

p [1, 2, 3].collect(&:to_s) #=> ["1", "2", "3"]
```

プラグイン

- Railsのクラスの拡張
- includeでクラスメソッドを継承するため、append_featuresを利用

```
module Foo
  def self.append_features(base)
    super(base)
    base.extend(ClassMethods)
  end
  module ClassMethods
    def foo
      ...
```

既存クラスの拡張の問題

- 驚き最大
- RailsなしでRubyは使えますか?
- jcode.rb問題
  - 既存のメソッドはなるべく再定義しない方がいいような

マクロ

- LispにあってRubyにないもの
- 目的
  - 効率
  - 構文の拡張
    - 評価の遅延
    - 呼び出し元のコンテキストでの評価
  - DSL
- 代用
  - ブロック
  - Binding.of_caller

ブロック

- 評価の遅延が可能
- 呼び出し元のコンテキストで評価可能
- ただし、評価されるコードは呼び出し側で記述

caller binding

- bindingは実行コンテキストを表す
- evalの第2引数に渡すと、そのコンテキストで評価
- caller bindingはメソッドの呼び出し元のbinding
- 何度かリクエストされたものの、採用されず

Binding.of_caller

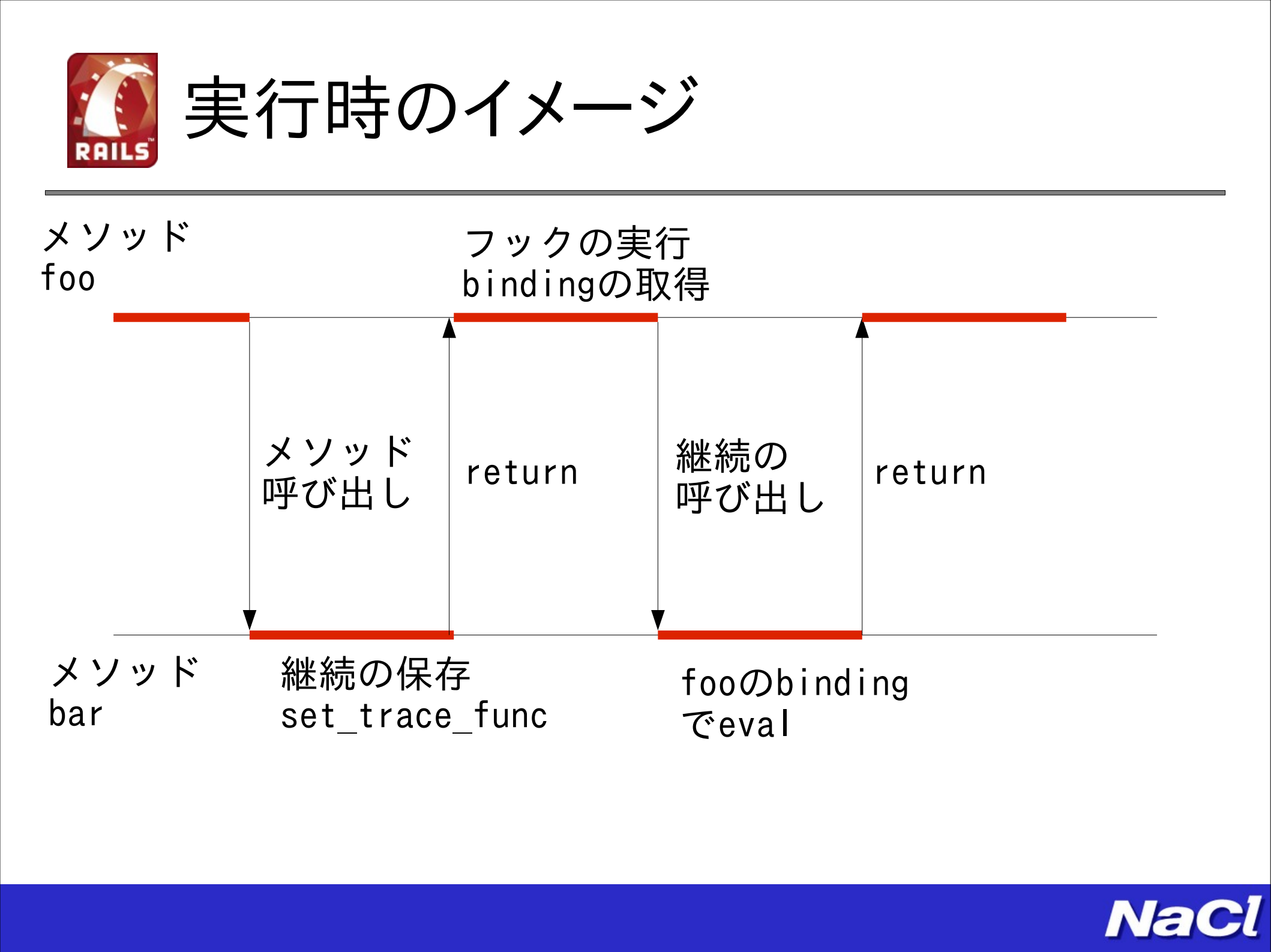
- Railsが提供するメソッド
- caller bindingを取得できる!

```
def print_x
  Binding.of_caller do |binding|
    eval(”puts x”, binding)
  end
end

x = 1
print_x #=> 1
```

Binding.of_callerの実装

- 継続とset_trace_funcを活用
- 継続とは
  - コード中の実行位置を保存し、後でその場所に復帰
  - メソッドをまたげるgotoもどき
- set_trace_func
  - メソッドからのreturnなどの時に呼び出されるフック



実行時のイメージ

メソッド
foo

フックの実行
bindingの取得

メソッド
呼び出し

return

継続の
呼び出し

return

メソッド
bar

継続の保存
set_trace_func

fooのbinding
でeval

まとめ

- メタプログラミングでRubyのパワーを活用
- 用法・用量を守って正しく使いましょう

おわり

ご質問は?