【B6】Rubyセッション

DEVELOPER CAMP

Rubyコーディング作法 - きっと役に立つコーディング規約

株式会社ネットワーク応用通信研究所
小倉正充

アジェンダ

DEVELOPER CAMP

- ネットワーク応用通信研究所について
- コーディングルールについて
- コーディングルールの解説

EMBARCADERO TECHNOLOGIES

DEVELOPER CAMP

ネットワーク応用通信研究所について

ネットワーク応用通信研究所について

EMBARCADERO TECHNOLOGIES

DEVELOPER CAMP

- ネットワーク応用通信研究所(NaCl)
  - NaClについて
  - どんなことをしている会社？
  - Ruby開発とのかかわり
  - 3rdRailの利用

会社紹介

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- NaClについて
  - 株式会社ネットワーク応用通信研究所
    - Network Applied Communication Laboratory Ltd.
      → 略称はNaCl（えぬえーしーえる）
    - 本社　島根県松江市
    - 支社　東京都千代田区外神田
  - Ruby作者のまつもとゆきひろが在籍

会社紹介

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- どんなことをしている会社？
  - オープンソースソフトウェアを活用したソリューションの提供
  - 最近の事例
    - 日本医師会、日医標準レセプトソフトの開発
      - 稼動状況6,370施設(8/15現在)
    - Ruby on Railsを利用したWebサイトの開発と支援
      - 島根県CMS
      - ニフティ株式会社 @nifty Timeline β など
      - 松江医療費高額合算システム

会社紹介

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- Ruby開発とのかかわり
  - まつもとゆきひろはRuby開発専任
  - ruby-lang.orgやrubyist.netによるサービス提供
    - リソースの提供（場所、回線など）
  - Rubyの開発方針はコミュニティ主導
    - NaClは口を出さない

Ruby / Rails による Webアプリケーションの実例（1/2）

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 島根県CMS
  - オープンソースライセンスで公開

Ruby / Rails による Webアプリケーションの実例（2/2）

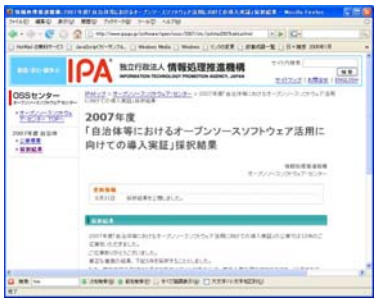
EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- ニフティ株式会社
  - 「@nifty TimeLine β」

Ruby / Railsによる 自治体基幹業務開発

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 松江市医療費高額合算システム
- IPAの公募事業として実施
  - 松江市
  - （株）テクノプロジェクト
  - 伊藤忠テクノソリューションズ（株）
  - （株）ネットワーク応用通信研究所
- 平成20年度から運用開始
- 新規開発は全てRubyで開発
- 成果物は全て公開中
- http://www.tpj.co.jp/service/system/ipa2007/
- http://www.ipa.go.jp/software/open/ossc/2007/stc/report/matsue.html



3rdRailの利用

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 3rdRail
  - Ruby / Ruby on Rails統合開発環境
- 日本語版テスタ
  - 開発者が利用しやすい製品の開発に協力
- 教育事業
  - RubyやRuby on Railsの教育に最適

Ruby / Railsの教育プログラム

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- Rubyの教育プログラム
  - 2006年から開発・提供
- 教育プログラム開催実績
  - 2008年5月時点

コース	提供開始時期	開催数
Ruby入門	2007年7月～	16回
Ruby on Rails入門	2007年7月～	11回
実践 Ruby on Rails アジャイル開発	2006年5月～	16回
Ruby on Rails運用	2007年12月～	1回

  - 2007年7月以降（株）CTCテクノロジーにて定期開催
    - http://www.school.ctc-g.co.jp/
    - 現在3rdRailにて展開中

EMBARCADERO TECHNOLOGIES

DEVELOPER CAMP

コーディングルールについて

コーディングルールについて

EMBARCADERO TECHNOLOGIES

DEVELOPER CAMP

- コーディングルールってなに？
- コーディングルールの目的
- Rubyとコーディングルール

コーディングルールってなに？

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- コーディングルール
  - ソースコードの書き方の規則の集合
  - 規約の種類
    - 命名規約
    - 整形に関する規約
    - 構文に関する規約
    - etc.

コーディングルールの目的

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 共有、保守をしやすくする
  - ソースコードを読みやすくする
- 品質を上げる
  - 開発者間で手法を統一できる
  - レビューしやすくなる
- 方向性を与える
  - 変数名のつけ方などの指針を得られる

Rubyとコーディングルール

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 変数に型がない
- 同じ処理をする違う書き方が多い
  - 文字列の例: `"hello", 'hello', %|hello|, %q|hello|`
- 省略表記が多い
  - メソッド呼び出しの実引数リスト括弧
  - return
  - if/unless式のthen
  - etc.

→ 変数名に型がないから命名規約が重要
→ 同じ処理でも異なるやり方があるのでどちらを使うか統一する

今回説明する コーディングルールについて

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 「Rubyコーディング規約」で紹介されているコーディングルールを元にしています
  - http://shugo.net/ruby-codeconv/codeconv.html

コーディングルールの解説

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 命名規約
- 整形に関する規約
- 構文に関する規約

命名規約

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 定数名
- メソッド名
- 真偽値を返すメソッド名
- 破壊的なメソッド名
- ファイル名

定数名(1/2)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 定数・変数名の制約
  - 先頭の文字を見て変数の種類が決定される

変数の種類	先頭文字	例
ローカル変数	小文字または「_」	`local_variable`
インスタンス変数	@	`@instance_variable`
クラス変数	@@	`@@class_variable`
グローバル変数	$	`$global_variable`
定数	大文字	`CONSTANT`

定数名(2/2)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 値を代入する定数名の規約
  - 「_」を単語の区切りとする
  - すべて大文字とする
- クラス・モジュール名としての定数名の規約
  - 「_」などの区切り文字を使用しない
  - 単語の先頭以外は小文字にする

- 例

```
EXAMPLE_CONSTANT = 10
class ExampleClass
…
end
```


メソッド名

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- メソッド名の制約
  - 先頭が小文字または「_」でなければならない

- メソッド名の規約
  - すべて小文字とする
  - 単語の区切りに「_」を用いる
  - メソッド名に動詞を使う場合は動詞の原形を使用する

- 例

```
def add_something
…
end
```

真偽値を返すメソッド名

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 真偽値を返すようなメソッドの名前の最後には「?」を付ける
- Rubyで定義されている真偽値を返すメソッドの例

```
"foo".empty? #=> false
"".empty?    #=> true
```

  - 「empty?」は例のように空文字列のとき真を、そうでないときは偽を返す

破壊的なメソッド名

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 破壊的なメソッドの名前の最後には「!」を付ける
  - 破壊的なメソッドというのは、レシーバー自身を変更してしまうメソッドのこと
  - Rubyに定義されている破壊的メソッドの例

```
array = [1, 3, 2]

sorted_array = array.sort
puts sorted_array  #=> [1, 2, 3]
puts array  #=> [1, 3, 2]

array.sort!
puts array  #=> [1, 2, 3]
```

ファイル名（1/3）

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- クラス・モジュール名からファイル名を導出する
- 一つのファイル中に複数のクラスを含む場合はモジュールを名前空間として使用し、ディレクトリで階層構造を表現する
  - ディレクトリ名はモジュール名から導出します
- 名前の導出ルール
  - すべて小文字にする
  - 単語を「_」で区切る

- 例
  - foo_bar.rb # FooBarクラスを定義

```
class FooBar
  …
end
```


ファイル名（2/3）

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 例
  - foo.rb # Fooモジュールを定義

```
module Foo
  …
end
```

  - foo/bar_baz.rb # Foo::BarBazクラスを定義

```
module Foo
  class BarBaz
    …
  end
end
```

  - foo/quu.rb # Foo::Quuクラスを定義

```
class Foo::Quu
  …
end
```

ファイル名（3/3）

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 見通しがよければ無理にファイルを分けなくてもよい

```
class Foo
  class Bar
    …
  end

  class Baz
    …
  end
end
```


整形に関する規約

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- インデント
- 一行の桁数
- コメント

インデント

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- インデントはスペースのみを使用し、幅を2とする
- タブは環境により表示のされ方が異なる
  - patch
  - diff
  - 印刷
  - コピー・アンド・ペースト
- チームでインデントの幅を揃えないと共有しにくい

一行の桁数

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 一行の桁数に制限を設ける
  - だいたい80文字程度
- 式の途中で改行する場合は「¥」(バックスラッシュ)を挿入する
- 一行が長すぎると印刷をしたときに折り返しが入って読みにくい

コメント

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- クラス・モジュールやメソッドの仕様をRDocスタイルで記述する
  - RDocとはRubyのソースを解析し、クラス、モジュール、メソッドの定義とそれらに記述されたコメントからドキュメントを生成するアプリケーション
- 例

```
# Fooクラスの仕様
class Foo
  # barメソッドの仕様
  def bar
    ...
  end
end
```


構文に関する規約

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- メソッドの定義
- クラスメソッドの定義
- メソッド呼び出し
- ブロック
- 条件分岐
- 論理演算子
- 同じ処理を異なる書き方で記述できる場合

メソッドの定義

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- メソッド定義の仮引数リストには括弧を付ける
- 例

```
# 悪い例
def foo bar, baz
  …
end
```

```
# 良い例
def foo(bar, baz)
  …
end
```

  - Rubyは仮引数リストの括弧を省略できる
  - 仮引数リストに括弧を付けないと見にくい

クラスメソッドの定義（1/3）

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- クラス名に依存しない形でクラスメソッドを定義する
  - クラスメソッドはクラスから直接呼び出すことのできるメソッド
    - C++やJavaでいうstaticメソッド的に使われる
  - クラスメソッドの例

```
class Foo
  def self.bar
    puts "class method bar"
  end

  def Foo.baz
    puts "class method baz"
  end
end

Foo.bar  #=> class method bar
Foo.baz  #=> class method baz
```

    - barもbazも定義の仕方は違うが同じクラスメソッド

クラスメソッドの定義(2/3)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- クラス名をFooからQuuに変更

```
class Quu
  def self.bar
    puts "class method bar"
  end

  def Quu.baz  # クラス名が変わったので変更
    puts "class method baz"
  end
end
```

  - bazのようにクラスメソッドを定義するとクラス名を変更したときに定義を書き換える必要がある
    - 変更のし忘れでエラーとなりやすい

→ クラス名に依存しない形でクラスメソッドを定義する

クラスメソッドの定義(3/3)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 複数のクラスメソッドを定義する場合以下のようにクラスメソッドを定義するのが簡単

```
class Foo
  class << self
    def bar
      …
    end

    def baz
      …
    end
  end
end
```

メソッド呼び出し

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 引数を持つメソッドの呼び出しの括弧は省略しない
- 例

```
# 悪い例
def foo(x, y)
  …
end

def bar(z)
  …
end

foo 1, bar 2 #=>SyntaxError
foo bar 2, 1 #=>ArgumentError
```

```
# 良い例
def foo(x, y)
  …
end

def bar(z)
  …
end

foo(1, bar(2))
foo(bar(2), 1)
```

  - 括弧を付けてメソッドの引数を明示しないとエラーが起きる

ブロック

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- ブロックには以下のように2種類の書き方がある

```
array.collect do |i|
  …
end

array.collect{|i|
  …
}
```

- ブロックの使用例

```
[1, 2, 3, 4].collect do |i|
  i*2
end  #=> [2, 4, 6]

[1, 2, 3, 4].select{|i| i%2 == 0 }  #=> [2, 4]
```

  - ブロックの引数iに配列の要素が1つずつ渡されながらブロックの処理が実行されます

ブロック

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 一行のときは「{}」を使用し、複数行のときは「do…end」を使う
- 「do…end」を使うときはメソッドチェインしない

- 例

```
array.collect do |i|
  i.to_s
end

array.collect{|i| i.to_s }.join(",")
```


条件分岐(1/4)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- if/unless式

```
if cond
  do_something
end

unless cond
  do_something
end
```

  - if式は条件式condが真のときdo_somethingを実行する
  - unless式は条件式condが偽のときdo_somethingを実行する

→「if !x」のような場合は「unless x」を使うなど、条件式がより簡潔に書ける方を使用する

条件分岐(2/4)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- if/unless修飾子

```
do_something if cond
do_something unless cond
```

  - if修飾子は条件式condが真である場合do_somethingを実行する
  - unless修飾子は条件式condが偽である場合do_somethingを実行する

条件分岐(3/4)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 複数行の式にはif/unless修飾子を使用しない
- 複雑な条件式の場合if/unless修飾子を使用しない

- 例

```
array.collect do |i|
  …
end if x

array.collect{|i| … } if (foo && bar) || ¥
  (baz && quu)
```

条件分岐(4/4)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- case式を使用した方が簡潔に記述できる場合は、case式を使用する

- 例

```
# 悪い例
if x == 0
  …
elsif x == 1
  …
elsif x == 2 || x == 3
  …
end
```

```
# 良い例
case x
when 0
  …
when 1
  …
when 2, 3
  …
end
```


論理演算子

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 論理演算子の優先順位はそれぞれ異なるため括弧を明示する
  - 論理演算子: `&&, ||, !, and, or ,not`

- 例

```
!(true && false) #=> true
! true && false  #=> false
```

同じ処理を異なる書き方で記述できる場合

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 書き方を統一する
- 例

```
for i in [1, 2, 3, 4]
  puts i
end

[1, 2, 3, 4].each do |i|
  puts i
end
```

  - 書き方は異なりますが、それぞれ「1、2、3、4」と出力します

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

おわり

おわり

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

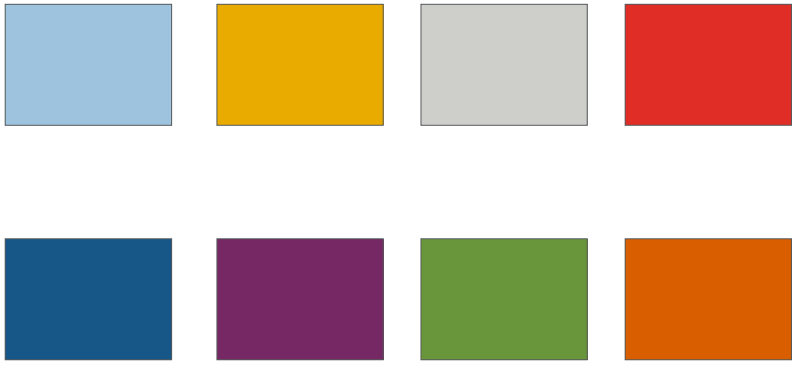
コーディングルールを決めることで書き方に適度な縛りがあると
開発しやすくなると思います

続きはWebで

- http://shugo.net/ruby-codeconv/codeconv.html

カラーパレット

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP



本文書の一部または全部の転載を禁止します。本文書の著作権は、著作者に帰属します。