Ruby本 第8章
Moduleを理解する

- P282 ~ P304 -

WEF 2105
Momo Hayashi

Moduleとは？？

classと同じようにmodule内に関数の定義ができること、
複数のメソッドや定数を**一箇所にまとめて整理**しておくことができる

Module定義例：

```
module Greeter
  def ‘Hello’
    ’Hello’
  end
end
```

⚠️モジュール名は必ず英数字の大文字で始める！

クラスの定義と同じ・・・
クラスとModuleは何がちがう？？

Moduleとは？？

クラス定義との違いは？

⚠️Moduleからインスタンスを作成することはできない

```
greeter = Greeter.new
 #=> NoMethodError: undefined method ‘new’ for Greeter:Module
```

⚠️ほかのModuleやクラスを継承することはできない

```
module AwesomeGreeter < Greeter
  end
 #=> SyntaxError: systax error, unexpected ‘<’
```

Moduleの役割とは？？

Moduleは、4つの目的によく使われます。

- インスタンスメソッドとして取り込む（Mix-in -include- ）
- Moduleをオブジェクトに取り込む（Mix-in -extend- ）

- 名前空間の提供
- Module関数

ModuleのMixinとは？？

そもそもRubyでは、**単一継承** が採用されている
（単一継承：クラスを一つしか継承できないという設計）

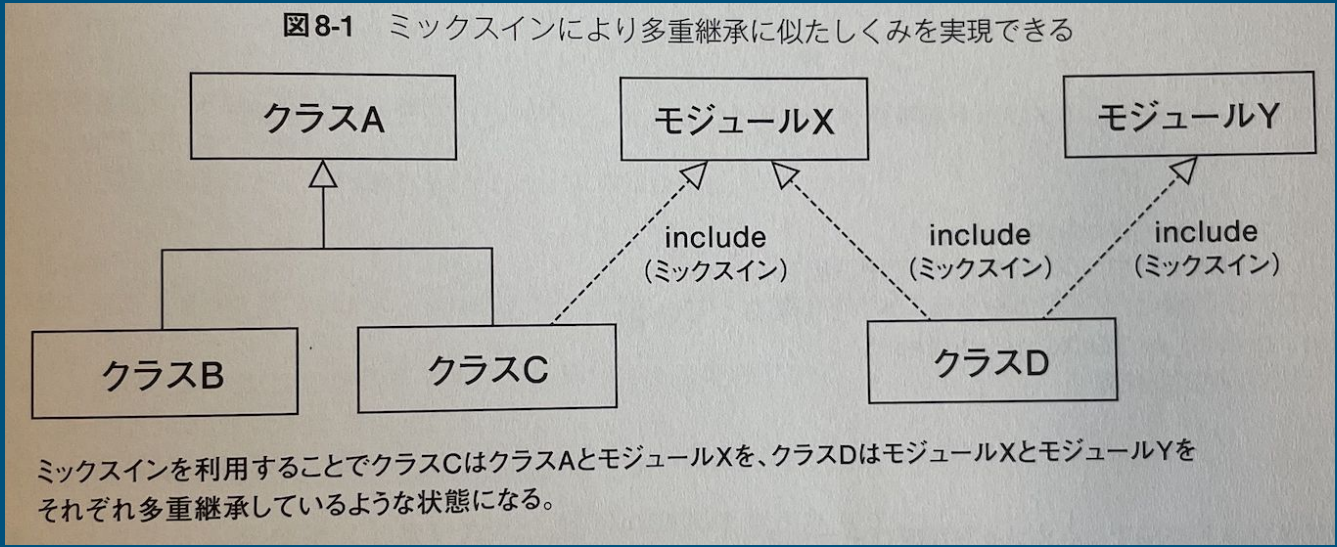
しかし、**ModuleのMixin**を使うことで
擬似的に**多重継承**を行っているようにコードを書くことができる！

・Mixinとは？？
Moduleをクラスにinclude, extend して**機能を追加**すること。
似たような機能を持っているが違う種類のクラスに対して
共有のメソッドを提供したい場合などに使用。

⇨重複を避けた、**DRYに則ったコード**を記述できる

ModuleのMixinとは？？

多重継承を可能にするMixinでは、
- Mixin先のクラスはどんなクラスでもOK！
- 1つのクラスに複数のModuleをMixinすることもOK！



ModuleのMixinとは？？

ModuleのMixinには、includeとextendの２種類がある

Include
・対象のクラスに「include モジュール名」と記載する
・モジュールのメソッドは、**クラスのインスタンスメソッド**になる

Extend
・対象のクラスに「extend モジュール名」と記載する
・モジュールのメソッドは、**クラスメソッド**として組み込まれる

ModuleのMixin　- include -

プログラム

```
module Greeting
  def morning
    p "おはよう"
  end

  def night
    p "おやすみ"
  end
end

class User
  include Greeting
end

user = User.new
user.morning
```

実行結果

```
"おはよう"
```

⇦「User」クラスに「Greeting」モジュールを読み込む例

Includeを利用すると、
モジュールのメソッドは、**インスタンスメソッド**になるので

user = User.new
user.morning

のように、Userクラスから作成したインスタンスから、
Greetingモジュール内の morningメソッド を呼び出す。

ModuleのMixin　- extend -

プログラム

```
module Greeting
  def morning
    p "おはよう"
  end

  def night
    p "おやすみ"
  end
end

class User
  extend Greeting
end

User.morning
```

実行結果

```
"おはよう"
```

⇦「User」クラスに「Greeting」モジュールを読み込む例

Extendを利用すると、
モジュールのメソッドは、**クラスメソッド**になるので

User.morning

のように、Userクラスから
Greetingモジュール内の morningメソッド を呼び出す
（インスタンスを作成する必要がない）

ModuleのMixin　- extend -

```
module Greet
  def say
    puts "Hello"
  end
end

obj = Object.new
obj.extend Greet
obj.say
```

⇦このような使い方もできる

空のオブジェクトを生成（obj）し、
extendを使ってGreetモジュールのメソッドを
特異メソッドとしてミックスインしています。

⇨includeでの拡張は静的であるのに対し、
extendは動的に対応できるので
プログラムの柔軟性が増すらしい

ここからはModuleのもう一歩踏み込んだ使い方！

IncludeされたModuleの有無を確認する方法

・特定のModuleが含まれているか確認する方法:
Class名.include?(モジュール名)　#=> true/ falseで返される

・includeされているModule全てを確認する方法:
Class名.included_modules　#=> [Loggable, Karnel] 配列で返される

・Module & Superclass を全て確認する方法:
Class名.ancestors
#=> [Product, Loggable, Karnel, Object, BasicObject] 配列で返される

Enumerable Moduleとは・・

Enumerable Moduleとは？

・Ruby Install時に自動的にシステムに組み込まれているModuleの一つ
・Array, Hash, Range, Enumerator等のクラスにincludeされている
・繰り返し処理を行うためのModuleで、
　Module内のメソッドは全て each を使って定義されている
・自分で定義したクラスにincludeして使うことも可能。
　（その際eachメソッドを必ず定義する）

・代表的メソッド：map, select, find, count, all?, include?

```
irb(main):010:0> ary = [3, 4, 2, 8]
=> [3, 4, 2, 8]
irb(main):011:0> ary.class.included_modules
=> [ActiveSupport::ToJsonWithActiveSupportEncoder, JSON::Ext::Generator::GeneratorMethods::Array, MessagePack::CoreExt, Enumerable, ActiveSupport::ToJsonWithAct
iveSupportEncoder, PP::ObjectMixin, ActiveSupport::Dependencies::Loadable, JSON::Ext::Generator::GeneratorMethods::Object, ActiveSupport::Tryable, Kernel]
irb(main):012:0>
irb(main):013:0>
irb(main):014:0>
irb(main):015:0> ha = { 'this' => 'is', hash: 'object' }
=> {"this"=>"is", :hash=>"object"}
irb(main):016:0> ha.class.included_modules
=> [ActiveSupport::ToJsonWithActiveSupportEncoder, JSON::Ext::Generator::GeneratorMethods::Hash, MessagePack::CoreExt, Enumerable, ActiveSupport::ToJsonWithActi
veSupportEncoder, PP::ObjectMixin, ActiveSupport::Dependencies::Loadable, JSON::Ext::Generator::GeneratorMethods::Object, ActiveSupport::Tryable, Kernel]
```

Enumerable Moduleとは・・

Enumerable Moduleのメソッドを利用することで、
左のコードを右のコードのように書くことができる

```
for value in [1, 2, 3, 4, 5]
  puts value
end
1
2
3
4
5
```

```
strings = []
for value in [1, 2, 3, 4, 5]
  strings.push(value.to_s)
end
strings.inspect # => ["1", "2", "3", "4", "5"]
```

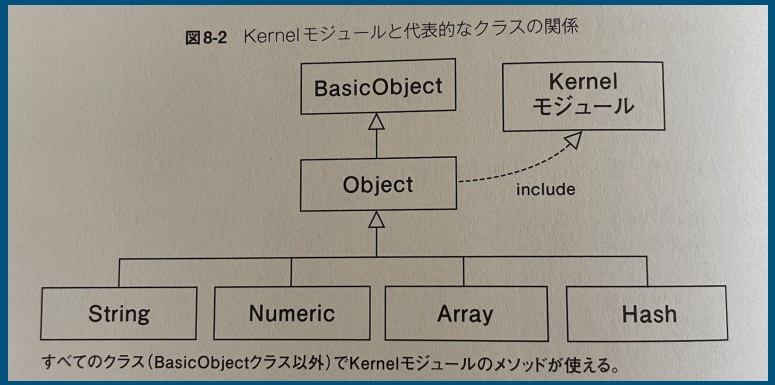
```
# Array#each
[1, 2, 3, 4, 5].each do |value|
  puts value
end
1
2
3
4
5

# Array#map
[1, 2, 3, 4, 5].map{ |value| value.to_s } => ["1", "2", "3", "4", "5"]

# mapをもっと簡単に書くと
[1, 2, 3, 4, 5].map(&:to_s) => ["1", "2", "3", "4", "5"]
```

Karnel Moduleとは・・

Karnel Moduleとは？

・Object クラスは Karnel Moduleをincludeしている

・Rubyのプログラミングはobjectクラスを継承したクラスを使うことが大半なので、結果としてKarnel Moduleのメソッドはいつでもどこでも使える（いわばグローバル関数）になっている



・代表的メソッド：puts, p, print, require, loop

<<番外編>>
Rails の HelpersってModuleってかいてあるよね？？

RailsにおけるModule - Helper -

Ruby on Railsで私たちが利用しているHelperの実体はModule。

```ruby
module SessionsHelper
  def current_user
    @current_user ||= User.find_by(id: session[:user_id])
  end
  def logged_in?
    current_user.present?
  end
end
```

```ruby
module BlogsHelper
  def choose_new_or_edit
    if action_name == 'new' || action_name == 'create'
      confirm_blogs_path
    elsif action_name == 'edit'
      blog_path
    end
  end
end
```

Helperに定義したメソッドは、Controller（クラス）、Viewで使用が可能

```ruby
class BlogsController < ApplicationController
  before_action :set_blog, only:[:show, :edit, :update, :de

  def create
    @blog = current_user.blogs.build(blog_params)
    if params[:back]
      render :new
    else
```

```erb
    <% if logged_in? %>
      <%= link_to "Profile", user_path(current_user.id) %>
      <%= link_to "Logout", session_path(current_user.id), me
    <% else %>
      <%= link_to "Sign up", new_user_path %>
      <%= link_to "Login", new_session_path %>
```

しかし、ModuleのIncludeが見つからない・・・

RailsにおけるModule - Helper -

・rails5ではデフォルトで、全ファイルincludeされる設定。

```
By default, each controller
  # will include all helpers. These helpers are only accessible on the controller throu
  #
  # In previous versions of \Rails the controller will include a helper which |
  # matches the name of the controller, e.g., <tt>MyController</tt> will automatically
  # include <tt>MyHelper</tt>. To return old behavior set +config.action_controller.inc
```

・Controllerに対応するものしか読み込ませたくない時は、
config/application.rbで設定可能！

（helperを全てincludeしていると意図していない同名のメソッドが呼ばれる可能性があるため）

```
module xxx
  class Application < Rails::Application
    config.action_controller.include_all_helpers = false #これをセット!!
  end
end
```

References;

・【Ruby入門】モジュール（Module）の使い方まとめ【include, extend, Mixin】
https://26gram.com/ruby-module

・【Ruby入門】Rubyのモジュールの使い方
https://uxmilk.jp/23190

・RubyのEnumerableを使ってみる
https://sandmark.hatenadiary.org/entry/20120306/1331052501

・【Rubyの基礎】配列の検索で使われるfindメソッドの使い方
https://style.potepan.com/articles/26676.html

・Ruby の Enumerable モジュールの使い方の覚書
https://blog.emattsan.org/entry/2020/04/30/193036

・rails helper 基本
https://qiita.com/yukiyoshimura/items/f0763e187008aca46fb4