

`['a', 'b', 'c'].map(&:upcase)` is  
何？

初学者が調べるRubyのしくみ





# 目次

1. 自己紹介
2. 概要説明
3. ``['a', 'b', 'c'].map(&:upcase)`` のコードの説明
4. ちょっと遊んでみた
5. 感想

# 自己紹介

FJORD BOOT CAMPで学習しています。 外部のLT会は初参加です！よろしくお願いします☺

- 名前：haruguchi-yuma

-  HaruguchiYuma  @haruguchiyuma  Blog 

- 趣味：バンド、数学、読書、モブプロ

- 住んでいるところ：滋賀県

- 前職：公立中学校の教員



# 概要説明

# 何を発表するのか？

## 1. どのような仕組みで動いているのか

```
1  [ 'a' , 'b' , 'c' ].map(&:upcase)
2  ⇒ [ 'A' , 'B' , 'C' ]
```

- 引数に&をつけるのはなぜ？
- ただのシンボル` :upcase `がなぜメソッドっぽい振る舞いをするのか？

## 2. この仕組みを使って遊んでみた

```
1  [ 'a' , 'b' , 'c' ].map(&'upcase' )
2  ⇒ [ 'A' , 'B' , 'C' ]
```

- 文字列を渡しても動くようにしてみた！

**['a', 'b', 'c'].map(&:upcase)の説明**

# 理解するために必要な知識

- ブロック付きメソッド
- Procオブジェクト
- Symbol#to\_procメソッドの挙動

# ブロック付きのメソッド

## 定義方法と呼び出し

```
1  #
2  def foo(&block)
3      p block # => #<Proc:0x0000000104a79338>
4      p block.class # => Proc
5      # call
6      block.call('Alice') #
7  end
8
9  #
10 foo do |name|
11     puts "#{name}"      ! "
12 end
13 => Alice                !
```

## ☆Point

- (&…)とすることでブロックはProcオブジェクトに変換される
- ブロック (Procオブジェクト)の処理は`call`メソッドで実行



ブロック→Procオブジェクト→.callで実行

メソッドに直接Procオブジェクトを渡すこともできるのでは？🤔

=> できた！！！！

# Procオブジェクトをメソッドに渡す

```
1  def foo(&block)
2    block.call('Alice')
3  end
4
5  # Proc
6  greeting_proc = Proc.new { |name| puts "#{name}          !" }
7
8  foo(&greeting_proc) #&
9  ⇒ Alice          !
```

## ☆Point

- ブロックの代わりに最初からProcオブジェクトを引数として渡すことも可能
- メソッド定義で(&…)としているときはメソッド呼び出しも引数の先頭に&をつけて呼び出す

もっとよく調べてみると、、、

# 実はprocオブジェクト以外にも渡せる

```
1 foo(&greeting_proc)
2 ⇒ Alice !
```

引数の先頭に&をつけて呼び出すとき、オブジェクトに対して暗黙的に`to\_proc`メソッドを呼び出している！！(るりま参照)

=> `to\_proc`メソッドを定義しているオブジェクトであれば引数として渡すことができる！！

---

## to\_procメソッドを定義しているクラス

- Proc
- Method
- **Symbol**

なのでこのようなコードが動くということになります。

```
1 ['a', 'b', 'c'].map(&:upcase)
```

# Symbol#to\_procの挙動

callするとselfをメソッドとして呼び出す

```
1 :upcase.to_proc.call  
2 ⇒ no receiver given (ArgumentError)
```

```
1 # 'a'.upcase  
2 :upcase.to_proc.call('a')  
3 ⇒ "A"
```

## ☆Point

- :symbol.to\_procを実行 (call)するとselfをメソッドとして呼び出す
- ただし、`call`メソッドの第一引数でメソッドのレシーバを指定する

# まとめ

```
1  ['a', 'b', 'c'].map(&:upcase)
2  ⇒ ['A', 'B', 'C']
3
4  #
5      :upcase.to_proc.call('a') ⇒ 'A'
6      :upcase.to_proc.call('b') ⇒ 'B'
7      :upcase.to_proc.call('c') ⇒ 'C'
```

1. ただのシンボル:upcaseは&によってProcオブジェクト化する
2. mapメソッドは#イメージのように配列の要素を一つずつ渡していくのでそれぞれ大文字に変換する

ちょっと遊んでみた🎤

# 文字列を渡しても動くようにしたい！

```
1 class String
2   def to_proc
3     Proc.new { |obj| eval "obj.#{self}" }
4   end
5 end
6
7 ['a', 'b', 'c'].map(&'uppercase') #      uppercase
8 ⇒ ['A', 'B', 'C']
```

動きました！！

# 感想：技術ネタを初めて発表してみても

適当なこととは言えないので、ドキュメントをくまなくみる癖がついた

- 意外と調査するのに時間がかかる
- この表現は正しいのか？伝わるのか？
- 改めてRubyの柔軟性に気がついた
- Rubyで色々試すのが楽しかった



# 参考資料

- るりま - ブロック付きメソッド呼び出し
- るりま - class Proc
- るりま - class Symbol#to\_proc

ご清聴  
ありがとうございました

