

卒業論文

コマンドラインツール作成ライブラリ Thor による hikiutils の書き換え

関西学院大学 理工学部 情報科学科

27013554 山根亮太

2017 年 3 月

指導教員 西谷 滋人 教授

目次

1	概要	3
2	序論	4
2.1	目的	4
2.2	既存システムの背景	4
2.3	目的	6
2.4	既存システムの説明	6
3	結果	8
3.1	コマンド名の検討	8
3.2	thor と optparse のコードの比較	12
4	CLI のコード	23
4.1	CLI の解説	23
5	Thor の初期化	25
5.1	コード	26
6	コマンド表示と処理	27
6.1	Thor	27
6.2	optparse	28
6.3	コード	29
7	CLI の実行	31
7.1	Thor	31
7.2	optparse	33
7.3	コード	33

コマンドラインツール作成ライブラリ Thor による hikiutils の書き換え
関西学院大学 理工学部情報科学科 27013554 山根亮太

目次

1 概要

2 序論

2.1 目的

本研究では hiki の編集作業をより容易にするためのツールの開発を行った。hiki は通常 web 上で編集を行っているが、GUI と CUI が混在しており、操作に不便な点がある。そこで、編集操作が CUI で完結するために開発をされたのが hikiutils である。しかし、そのユーザインタフェースにはコマンドが直感的でないという問題点がある。そこで、Thor というコマンドラインツール作成ライブラリを用いる。optparse というコマンドライン解析ライブラリを使用している hikiutils を新たなコマンドライン解析ライブラリを使用することコマンドを書き換え、より直感的なコマンドにすることが可能である。

2.2 既存システムの背景

2.2.1 hiki

hiki とはプログラミング言語 Ruby を用いられることで作られた wiki クローンの 1 つである。hiki の主な特徴として

- オリジナル wiki に似たシンプルな書式
- プラグインによる機能拡張
- 出力する HTML を柔軟に変更可能
- ページにカテゴリ付けできる
- CSS を使ったテーマ機能
- 携帯端末可能
- InterWiki のサポート
- HikiFarm に対応
- ページの追加、編集がしやすい

等がある [1]。

2.2.2 hikiutils

hikiutils は hiki の編集作業を容易に行うことができるようにするツール群であり、プログラミング言語 Ruby のライブラリである gem フォーマットに従って提供されている [2]。hikiutils は CLI(Command Line Interface) で操作するため、オプション解析をお

こなう必要がある. gem には, この用途 に適合したライブラリがいくつも提供されている [3]. この中 で, あまり利用頻度は高くないが古くから開発され, 使用例 が広く紹介されている optparse を利用している.

2.3 目的

本研究では hiki の編集
常 web 上で編集を行っ
こで, 編集操作が CUI
のユーザインタフェース
というコマンドラインツ
解析ライブラリを使用し
することコマンドを書き

2.4 既存システム

図に従って hiki syste

図 1 hiki web system と hiki system の対応関係.

hiki は, hiki 記法を用いた wiki clone です. wiki はワード・カニングガムが作った wikiwikiweb を源流とする home page 制作を容易にするシステムで, hiki も wiki の基本
要求仕様を満足するシステムを提供しています. wiki の特徴である web 上で編集する機
能を提供しています. これを便宜上 hiki web system と呼びます. 図にある通り, 一般的

な表示画面の他に、編集画面が提供されており、ユーザーはこの編集画面からコンテンツを編集することが可能です。リンクやヘッダー、リスト、引用、表、図の表示などの基本テキストフォーマットが用意されています。

hiki web system の実際の基本動作は、hiki.cgi プログラムを介して行われています。こちらを便宜上 hiki system と呼びます。hiki system は、data/text に置かれた書かれたプレーンテキストを html へ変換します。この変換は hikidoc[1-1] という hiki フォーマット converter を使っています。また、添付書類は cache/attach に、一度フォーマットした html は parser に置かれており、それらを参照して html を表示する画面を hiki.cgi は作っています。さらに hiki system では検索機能、自動リンク作成などが提供されています。

研究室内の内部文書、あるいは外部への宣伝資料、さらに wikipedia のように重要な研究成果の発信などに西谷研ではこの hiki system を利用しています。初心者にも覚えやすい直感的な操作です。しかし、慣れてくるとテキスト編集や画面更新にいちいち web 画面へ移行せねばならず、編集の思考が停止します。そこで、テキスト編集に優れた editor との連携や、terminal 上の shell command と連携しやすいように hikiutils という cli(command line interface) を作成して運用しています。この hikiutils のコマンドオプションの実装をしておいて、より使いやすくすることが本研究の目的です。

3 結果

3.1 コマンド名の検討

3.1.1 コマンドの命名原則

機能ごとの動作はコマンドのオプションによって指定されます。このオプションにどのような名前をつけるかは、どれだけコマンドを覚えやすいかという意味で重要です。コマンドの振る舞いを的確に表す名称をつける必要があります。

この振る舞いとしてもっとも受け入れやすいのが shell で用意されているコマンドです。pwd, ls, rm, touch, open などのもっとも直感的に動作がわかるコマンドです。hikiutils の振る舞いを予測できるシェルコマンドと同じ名前でオプションを提供するようにします。

3.1.2 hikiutils があらかじめ想定している利用形態

ここで hikiutils があらかじめ想定している利用形態を解説しておきます。

`{{attach_view(hikiutils_yamane_09_copy.002.jpg)}}` hikiutils は、

- local PC と global server とが用意されており、
- それらのデータを rsync で同期する

ことで動作することを想定しています。これは、ネットに繋がっていないオフラインの状況でもテキストなどの編集が可能で、さらに不用意な書き換えを防ぐための機構です。さらに、どちらもが何かあった時のバックアップともなって、ミスによる手戻りを防いでいます。

これらの設定は、`/.hikirc` に yaml 形式で記述・保存されています。

```
1 bob% cat ~/.hikirc
2 :srcs:
3 - :nick_name: new_ist
4   :local_dir: "/Users/bob/Sites/new_ist_data/ist_data"
5   :local_uri: http://localhost/ist
6   :global_dir: nishitani@ist.ksc.kwansei.ac.jp:/home/
                 nishitani/new_ist_data/ist_data
7   :global_uri: http://ist.ksc.kwansei.ac.jp/~nishitani/
8 - :nick_name: dmz0
```



```

9   :local_dir: "/Users/bob/Sites/nishitani0/Internal/data"
10  :local_uri: http://localhost/~bob/nishitani0/Internal
11  :global_dir: bob@dmz0:/Users/bob/Sites/nishitani0/
      Internal/data
12  :global_uri: http://nishitani0.kwansei.ac.jp/~bob/
      nishitani0/Internal

```

また、一般的に一人のユーザがいくつものまとまりとしての local-global ペアを保持して管理することが普通です。それぞれに `nicke_name` をつけて管理しています。

```

1  bob% hiki -s
2  hikiutils: provide utilities for helping hiki editing.
3  "open -a mi"
4  target_no:1
5  editor_command:open -a mi
6  id | name          | local directory
                                     | global uri
7  -----
8  0 | new_ist         | /Users/bob/Sites/new_ist_data/ist_data
                                     | http://ist.ksc.k
9  *1 | dmz0            | /Users/bob/Sites/nishitani0/Internal/
      data          | http://nishitani
10  2 | ist              | /Users/bob/Sites/hiki-data/data
                                     | http://ist.ksc.k
11  3 | new_maple        | /Users/bob/Sites/new_ist_data/
      maple_hiki_data | http://ist.ksc.k

```

とすると、それらの一覧と、いま `target` にしている `nick_name` ディレクリが表示されます。

3.1.3 コメンド名と振る舞いの詳細

開発の結果コマンドを以下のように書き換えました。上部に記した、特によく使うコマンドに関しては、`shell` でよく使われるコマンド名と一致するようにしました。

それぞれの意図を動作の解説として記述しています。

■**open FILE** ファイルを編集のために `editor` で `open`. Editor は `/.hikirc` に

表 1 hikiutils があらかじめ想定している利用形態

変更前	変更後	動作の解説
edit FILE	open	open file
list [FILE]	ls	list files
rsync	rsync	rsync files
update FILE	touch	update file
show	pwd	show nick_names
target VAL	cd	target を変える, change directory とのメタファ
move [FILE]	mv	move file
remove [FILE]	rm	remove files
add		add sources info
checkdb		check database file
datebase FILE		read datebase file
display FILE		display converted hikifile
euc FILE		translate file to euc
help [COMMAND]		Describe available commands or one specific command
version		show program version

```
:editor_command: open -a mi
```

として保存されている. open -a mi を emacs などに適宜変更して使用.

■ls [FILE] local_dir にあるファイル名を [FILE*] として表示. 例えば, hikiutils_yamane 以下の拡張子がついたファイルを表示. hiki システムでは text ディレクトリーは階層構造を取ることができない. 西谷研では directory の代わりにスネーク表記で階層構造を表している.

■rsync local_dir の内容を global_dir に rsync する. 逆方向は同期に誤差が生じたり, permission がおかしくなるので, 現在のところ一方の同期のみとしている. したがって, 作業手順としてはテキストの変更は local_dir で読み行うようにしている.

■touch FILE local_dir で書き換えた FILE の内容を local_uri に反映させ，ブラウザで表示．シェルコマンドの touch によって，変更時間を現在に変え，最新状態とするのに似せてコマンド名を touch としている．

■pwd nick_name の一覧と target を表示，current target を current dir とみなして，コマンド名を pwd とした．

■cd VAL target を変える，change directory とのメタファ．ただし，いまのところ nick_name では対応しておらず，nick_name の番号を VAL 入力することで変更する．

3.2 thor と optparse のコードの比較

hikiutils のコマンドライン解析ツールを optparse から thor に換えることでコマンドの書き換えを行うことができた。また、thor で書かれた hikiutils は optparse で書かれたものよりもコードが短くなり、コマンドの解析も簡単に行えることができた。ここでは thor と optparse のコードを比較し thor の良さを確認する。

3.2.1 Thor と optparse とは

■**Thor とは** Thor とは、コマンドラインツールの作成を支援するライブラリのことである。git や bundler のようにサブコマンドを含むコマンドラインツールを簡単に作成することができる [4]。

Thor の基本的な流れとしては

1. Thor を継承したクラスのパブリックメソッドがコマンドになる
2. クラス.start(ARGV) でコマンドラインの処理をスタートする

という流れである [4]。

start に渡す引数が空の場合、Thor はクラスのヘルプリストを出力する。また、Thor はサブコマンドやサブサブコマンドも作ることができる。

■**optparse とは** optparse モジュールとは、getopt よりも簡便で、柔軟性に富み、かつ強力なコマンドライン解析ライブラリである。optparse では、より宣言的なスタイルのコマンドライン解析手法、すなわち OptionParser のインスタンスでコマンドラインを解析するという手法をとっている。これを使うと、GNU/POSIX 構文でオプションを指定できるだけでなく、使用法やヘルプメッセージの生成も行える [5]。

利用頻度はあまり高くないが古くから開発され、使用例が広く紹介されている。

optparse の基本的な流れとしては

1. OptionParser オブジェクト opt を生成する
2. オプションを取り扱うブロックを opt.on に登録する
3. opt.parse(ARGV) でコマンドラインを実際に parse する

という流れである。

OptionParser はコマンドラインのオプション取り扱うためのクラスであるためオブ

ジェクト `opt` を生成され `opt.on` にコマンドを登録することができる。しかし、`OptionParser#on` にはコマンドが登録されているだけであるため、`OptionParser#parse` が呼ばれた時、コマンドラインにオプションが指定されていれば実行される。`optparse` にはデフォルトとして `-help` と `-version` オプションを認識する [6]。

3.2.2 コードの解説

■Thor の定義 `{{attach_view(hikiutils_yamane.003.jpg)}}` Thor の `initialize` の定義のされ方

1. `Hikithor::CLI.start(ARGV)` が呼ばれる
2. `initialize` メソッドが呼ばれる
3. これでは Thor の `initialize` メソッドが呼ばれない
4. `super` を書くことで Thor の `initialize` メソッドが呼ばれる

`optparse` では `require` で `optparse` を呼ばれ `optparse` の `initialize` を定義する必要はないが、Thor は `initialize` を定義する必要がある。Thor の定義方法は `require` で Thor を呼び CLI クラスで継承し、`initialize` メソッドに `super` を書くことで Thor の `initialize` が呼ばれる。`initialize` メソッド内では Thor の初期設定がされていないため、スーパークラスのメソッドを読み出してくれる `super` を書き加えることで図のように `initialize` メソッドで Thor の `initialize` メソッドが呼ばれて定義される。

■実際のコード

```
1 # -*- coding: utf-8 -*-
2 require 'thor'
3 require 'kconv'
4 require 'hikidoc'
5 require 'erb'
6 require "hikiutils/version"
7 require "hikiutils/tmarshal"
8 require "hikiutils/infodb"
9 require 'systemu'
10 require 'fileutils'
11 require 'yaml'
12 require 'pp'
13
14 module Hikithor
15
16   DATA_FILE=File.join(ENV['HOME'], '.hikirc')
17   attr_accessor :src, :target, :editor_command, :browser, :
     data_name, :l_dir
18
```

```

19  class CLI <
20    def initia
21      super
22      @data_n
23      glot
24      data_pa
25      DataFil
26      file =
27      @src =
28      file.cl
29      @target
30      @l_dir=
31      browser
32      @browse:
33      p edito
34      @editor
35      : €
36  end

```

■hikiutils の実行

- Thor

1. hiki.thor の Hiki

1. hikiutils.thor.rb

- optparse

図 3

1. Hiki の HikiUtils::Command.run(ARGV) で hikiutils.rb の run メソッドを呼ぶ
2. new(argv).execute で execute メソッドが実行される

このように optparse では実行を行うためのメソッドが必要であるが、Thor ではクラスのメソッドを順に実行していくため run メソッドと execute メソッドは必要ない。

■実際のコード

- Thor

```
1 #!/usr/bin/env ruby
2
3 require "hikiutils_thor"
4
5 HikiThor::CLI.start(ARGV)
```

```

1  # -*- coding: utf-8 -*-
2  require 'thor'
3  require 'kconv'
4  require 'hikidoc'
5  require 'erb'
6  require "hikiutils/version"
7  require "hikiutils/tmarshal"
8  require "hikiutils/infodb"
9  require 'systemu'
10 require 'fileutils'
11 require 'yaml'
12 require 'pp'
13
14 module Hikithor
15
16   DATA_FILE=File.join(ENV['HOME'], '.hikirc')
17   attr_accessor :src, :target, :editor_command, :browser, :
     data_name, :l_dir
18
19   class CLI < Thor
20     def initialize(*args)
21       super
22       @data_name=['nick_name', 'local_dir', 'local_uri', '
         global_dir', 'global_uri']
23       data_path = File.join(ENV['HOME'], '.hikirc')
24       DataFiles.prepare(data_path)
25
26       file = File.open(DATA_FILE, 'r')
27       @src = YAML.load(file.read)
28       file.close
29       @target = @src[:target]
30       @l_dir=@src[:srcs][@target][:local_dir]
31       browser = @src[:browser]
32       @browser = (browser==nil) ? 'firefox' : browser

```

- optparse

```

1  #!/usr/bin/env ruby
2
3  require "hikiutils"
4
5  HikiUtils::Command.run(ARGV)

```

```

1  def self.run(argv=[])

```

```

2      print "hikiutils: provide utilities for helping hiki
          editing.\n"
3      new(argv).execute
4  end
5
6  def execute
7      @argv << '--help' if @argv.size==0
8      command_parser = OptionParser.new do |opt|
9          opt.on('-v', '--version', 'show_program_Version.') {
              |v|
10             opt.version = HikiUtils::VERSION
11             puts opt.ver
12         }
13         opt.on('-s', '--show', 'show_sources') {show_sources}
14         opt.on('-a', '--add', 'add_sources_info') {
              add_sources }
15         opt.on('-t', '--target_VAL', 'set_target_id') {|val|
              set_target(val) }
16         opt.on('-e', '--edit_FILE', 'open_file') {|file|
              edit_file(file) }
17         opt.on('-l', '--list_FILE', 'list_files') {|file|
              list_files(file) }
18         opt.on('-u', '--update_FILE', 'update_file') {|file|
              update_file(file) }
19         opt.on('-r', '--rsync', 'rsync_files') {rsync_files}
20         opt.on('--database_FILE', 'read_database_file') {|
              file| db_file(file)}
21         opt.on('--display_FILE', 'display_converted_hikifile'
              ) {|file| display(f\
22         ile)}}
23         opt.on('-c', '--checkdb', 'check_database_file') {
              check_db}
24         opt.on('--remove_FILE', 'remove_file') {|file|
              remove_file(file)}
25         opt.on('--move_FILES', 'move_file1,file2',Array) {|
              files| move_file(file\
26         s)}}
27         opt.on('--euc_FILE', 'translate_file_to_euc') {|file|
              euc_file(file) }
28         opt.on('--initialize', 'initialize_source_directory')
              {dir_init() }
29     end
30     begin
31         command_parser.parse!(@argv)
32     rescue=> eval
33     p eval
34     end

```

```
35         dump_so
36         exit
37     end
```

■コマンドの表示と処理

- Thor

図 4

1. コマンド名, コマンドの説明を一覧に表示させる
2. パブリックメソッドのコマンドを別のコマンド名でも実行できるようにする
3. コマンドの命令の実行コード

Thor では desc で一覧を表示させるコマンド名, コマンドの説明を登録する. しかし, ここで記述したコマンドは一覧で表示させるものであり, 実行されることはないので実際のコマンドと対応させる必要がある. Thor では処理実行を行うメソッドがコマンドとなる. しかし, それではコマンド名は 1 つしか使うことができない. ここで用いるものが map である. map を使うことで別のコマンドを指定することができる.

- optparse

図 5

1. OptionParser オブジェクト opt を生成
2. opt にコマンドを登録
3. 入力されたコマンドの処理のメソッドへ移動

optparse では OptionParser オブジェクト opt の生成を行い、コマンドを opt に登録することでコマンドを作成することができる。しかし、これはコマンドを登録しているだけでコマンドの一覧ではこれを表示することができるがコマンドの実行を行うためには実行を行うためのメソッドを作成する必要がある。optparse でのコマンドの実行は opt で登録されたコマンドが入力されることでそれぞれのコマンドの処理を行うメソッドに移動し処理を行う。しかし、このコマンド登録は-を付けたコマンドした登録ができなく、-なしのコマンド登録はまた別のやり方でやらなくてはならない。以上より、Thor ではコマンドの指定と処理には desc,map, 処理メソッドだけで済むが、optparse ではコマンドを登録するためのメソッドと処理メソッドが必要になってくる。また、コマンドは Thor では処理メソッドがコマンド名になるが、optparse ではコマンドを登録するための処理も必要となってくる。

■実際のコード

- Thor

```

1 desc 'show,--show', 'show_sources'
2 map "--show" => "show"
3 def show
4   printf("target_no:%i\n",@src[:target])
5   printf("editor_command:%s\n",@src[:editor_command])
6   @i_size,@n_size,@l_size,@g_size=3,5,30,15 #i,g_size
7   are fixed
8   n_l,l_l=0,0
9   @src[:srcs].each_with_index{|src,i|
10     n_l =(n_l= src[:nick_name].length)>@n_size? n_l:
11       @n_size
12     l_l =(l_l= src[:local_dir].length)>@l_size? l_l:
13       @l_size
14   }
15   @n_size,@l_size=n_l,l_l
16   command = Command.new
17   header = command.display_format('id','name','local_
18     directory','global_uri',@i_size,@n_size,@l_size,
19     @g_size)
20
21   puts header
22   puts '-' * header.size
23
24   @src[:srcs].each_with_index{|src,i|
25     target = i==@src[:target] ? '*':'_'
26     id = target+i.to_s
27     name=src[:nick_name]
28     local=src[:local_dir]
29     global=src[:global_uri]
30     puts command.display_format(id,name,local,global,
31       @i_size,@n_size,@l_size,@g_size)
32   }
33 end

```

• optparse

```

1 def execute
2   @argv << '--help' if @argv.size==0
3   command_parser = OptionParser.new do |opt|
4     opt.on('-v', '--version', 'show_program_Version.') {
5       |v|
6       opt.version = HikiUtils::VERSION
7       puts opt.ver
8     }
9     opt.on('-s', '--show', 'show_sources') {show_sources}
10   end

```

```

9      opt.on('-a', '--add', 'add_sources_info') {
10          add_sources }
11      opt.on('-t', '--target_VAL', 'set_target_id') {|val|
12          set_target(val)}
13      opt.on('-e', '--edit_FILE', 'open_file') {|file|
14          edit_file(file) }
15      opt.on('-l', '--list_FILE', 'list_files') {|file|
16          list_files(file)}
17      opt.on('-u', '--update_FILE', 'update_file') {|file|
18          update_file(file) }
19      opt.on('-r', '--rsync', 'rsync_files') {rsync_files}
20      opt.on('--database_FILE', 'read_database_file') {|
21          file| db_file(file)}
22      opt.on('--display_FILE', 'display_converted_hikifile'
23          ) {|file| display(file)}
24      opt.on('-c', '--checkdb', 'check_database_file') {
25          check_db}
26      opt.on('--remove_FILE', 'remove_file') {|file|
27          remove_file(file)}
28      opt.on('--move_FILES', 'move_file1,file2', Array) {|
29          files| move_file(files)}
30      opt.on('--euc_FILE', 'translate_file_to_euc') {|file|
31          euc_file(file)}
32      opt.on('--initialize', 'initialize_source_directory')
33          {dir_init() }
34
35      end
36      begin
37          command_parser.parse!(@argv)
38      rescue=> eval
39      p eval
40      end
41      dump_sources
42      exit
43
44      def show_sources()
45          printf("target_no:%i\n",@src[:target])
46          printf("editor_command:%s\n",@src[:editor_command])
47          check_display_size()
48          header = display_format('id','name','local_directory',
49                                  'global_uri')
50
51          puts header
52          puts '-' * header.size
53
54          @src[:srcs].each_with_index{|src,i|
55              target = i==@src[:target] ? '*':'_'}

```

```

43         id = target+i.to_s
44         name=src[:nick_name]
45         local=src[:local_dir]
46         global=src[:global_uri]
47         puts display_format(id,name,local,global)
48     }
49 end
50
51 def add_sources
52     cont = {}
53     @data_name.each{|name|
54         printf("%s□?□", name)
55         tmp = gets.chomp
56         cont[name.to_sym] = tmp
57     }
58     @src[:srcs] << cont
59     show_sources
60 end

```

コードからも Thor のほうが短くなっていることが分かる。よって、Thor と optparse でのコードの違いは以上の部分になるが全体的にも Thor のほうがコードが短くなり、コマンドの定義も簡単に行うことができる。また、コマンドを簡単に定義できることで書き換えもすぐ行うことができるので、より直感的なコマンドに実装することも可能となった。

4 CLI のコード

hikiutils のコマンドライン解析ツールを optparse から thor に換えることでコマンドの書き換えを行うことができた。また、thor で書かれた hikiutils は optparse で書かれたものよりもコードが短くなり、コマンドの解析も簡単に行えることができた。ここでは thor と optparse のコードを比較し thor の良さを確認する。

4.1 CLI の解説

4.1.1 Thor

Thor とは、コマンドラインツールの作成を支援するライブラリのことである。git や bundler のようにサブコマンドを含むコマンドラインツールを簡単に作成することができる [1-2]。

Thor の基本的な流れとしては

1. Thor を継承したクラスのパブリックメソッドがコマンドになる
2. クラス.start(ARGV) でコマンドラインの処理をスタートする

という流れである [1-2]。

start に渡す引数が空の場合、Thor はクラスのヘルプリストを出力する。また、Thor はサブコマンドやサブサブコマンドも作ることができる。

4.1.2 optparse

optparse とは、getopt よりも簡便で、柔軟性に富み、かつ強力なコマンドライン解析ライブラリである。optparse では、より宣言的なスタイルのコマンドライン解析手法、すなわち OptionParser のインスタンスでコマンドラインを解析するという手法をとっている。これを使うと、GNU/POSIX 構文でオプションを指定できるだけでなく、使用法やヘルプメッセージの生成も行える [1-3]。

利用頻度はあまり高くないが古くから開発され、使用例が広く紹介されている。

optparse の基本的な流れとしては

1. OptionParser オブジェクト opt を生成する
2. オプションを取り扱うブロックを opt.on に登録する
3. opt.parse(ARGV) でコマンドラインを実際に parse する

という流れである.

OptionParser はコマンドラインのオプション取り扱うためのクラスであるためオブジェクト `opt` を生成され `opt.on` にコマンドを登録することができる. しかし, `OptionParser#on` にはコマンドが登録されているだけであるため, `OptionParser#parse` が呼ばれた時, コマンドラインにオプションが指定されていれば実行される. `optparse` にはデフォルトとして `-help` と `-version` オプションを認識する [1-4].

5 Thor の初期化

図 6

- Thor の initialize でのコード
1. Hikithor::CLI.start(ARGV) が呼ばれる
 2. initialize メソッドが呼ばれる
 3. これでは Thor の initialize メソッドが呼ばれない
 4. super を書くことで Thor の initialize メソッドが呼ばれる

optparse では require で optparse を呼び optparse の initialize を定義する必要はないが、Thor は initialize を定義する必要がある。Thor の定義方法は require で Thor を呼び CLI クラスで継承し、initialize メソッドに super を書くことで Thor の initialize が呼ばれる。initialize メソッド内では Thor の初期設定がされていないため、スーパークラスのメソッドを読み出してくれる super を書き加えることで図のように initialize メソッド内で Thor の initialize メソッドが呼ばれ定義される。

5.1 コード

```
1 # -*- coding: utf-8 -*-
2 require 'thor'
3 require 'kconv'
4 require 'hikidoc'
5 require 'erb'
6 require "hikiutils/version"
7 require "hikiutils/tmarshal"
8 require "hikiutils/infodb"
9 require 'systemu'
10 require 'fileutils'
11 require 'yaml'
12 require 'pp'
13
14 module Hikithor
15
16   DATA_FILE=File.join(ENV['HOME'], '.hikirc')
17   attr_accessor :src, :target, :editor_command, :browser, :
     data_name, :l_dir
18
19   class CLI < Thor
20     def initialize(*args)
21       super
22       @data_name=['nick_name','local_dir','local_uri','
         global_dir','global_uri']
23       data_path = File.join(ENV['HOME'], '.hikirc')
24       DataFiles.prepare(data_path)
25
26       file = File.open(DATA_FILE, 'r')
27       @src = YAML.load(file.read)
28       file.close
29       @target = @src[:target]
30       @l_dir=@src[:srcs][@target][:local_dir]
31       browser = @src[:browser]
32       @browser = (browser==nil) ? 'firefox' : browser
33       p editor_command = @src[:editor_command]
34       @editor_command = (editor_command==nil) ? 'open -a mi'
         : editor_command
35     end
```

6 コマンド表示

6.1 Thor

図 7

1. コマンド名, コマンドの説明を一覧に表示させる
2. パブリックメソッドのコマンドを別のコマンド名でも実行できるようにする
3. コマンドの命令の実行コード

Thor では desc で一覧を表示させるコマンド名, コマンドの説明を登録する. しかし, ここで記述したコマンドは一覧で表示させるものであり, 実行されることはないので実際のコマンドと対応させる必要がある. Thor では処理実行を行うメソッドがコマンドとなる. しかし, それではコマンド名は 1 つしか使うことができない. ここで用いるものが map である.

```
map A => B
```

map とは B でしか読めないものを A でも読めるようにしてくれるものである. よって, これを使うことで別のコマンドも指定することができる.

6.2 optparse

図 8

1. OptionParser オブジェクト opt を生成
2. opt にコマンドを登録
3. 入力されたコマンドの処理のメソッドへ移動

optparse では OptionParser オブジェクト opt の生成を行い、コマンドを opt に登録することでコマンドを作成することができる。しかし、これはコマンドを登録しているだけでコマンドの一覧ではこれを表示することができるが、コマンドの実行を行うためには実行を行うためのメソッドを作成する必要がある。optparse でのコマンドの実行は opt で登録されたコマンドが入力されることでそれぞれのコマンドの処理を行うメソッドに移動し処理を行う。しかし、このコマンド登録はハイフンを付けたコマンドしか登録ができず、ハイフンなしのコマンド登録はまた別の手段でやらなくてはならない。以上より、Thor ではコマンドの指定と処理には desc,map, 処理メソッドだけで済むが、optparse ではコマンドを登録するためのメソッドと処理メソッドが必要になってくる。また、コマンドは Thor では処理メソッドがコマンド名になるが、optparse ではコマンドを登録するための処理も必要となってくる。

6.3 コード

- Thor

```
1 desc 'show,--show', 'show_sources'
2 map "--show" => "show"
3 def show
4   printf("target_no:%i\n",@src[:target])
5   printf("editor_command:%s\n",@src[:editor_command])
6   @i_size,@n_size,@l_size,@g_size=3,5,30,15 #i,g_size
7   are fixed
8   n_l,l_l=0,0
9   @src[:srcs].each_with_index{|src,i|
10     n_l=(n_l= src[:nick_name].length)>@n_size? n_l:
11       @n_size
12     l_l=(l_l= src[:local_dir].length)>@l_size? l_l:
13       @l_size
14   }
15   @n_size,@l_size=n_l,l_l
16   command = Command.new
17   header = command.display_format('id','name','local_
18     directory','global_uri',@i_size,@n_size,@l_size,
19     @g_size)
20
21   puts header
22   puts '-' * header.size
23
24   @src[:srcs].each_with_index{|src,i|
25     target = i==@src[:target] ? '*':'_
26     id = target+i.to_s
27     name=src[:nick_name]
28     local=src[:local_dir]
29     global=src[:global_uri]
30     puts command.display_format(id,name,local,global,
31       @i_size,@n_size,@l_size,@g_size)
32   }
33 end
```

- optparse

```
1 def execute
2   @argv << '--help' if @argv.size==0
3   command_parser = OptionParser.new do |opt|
```

```

4      opt.on('-v', '--version', 'show_program_Version.') {
5          |v|
6          opt.version = HikiUtils::VERSION
7          puts opt.ver
8      }
9      opt.on('-s', '--show', 'show_sources') {show_sources}
10     opt.on('-a', '--add', 'add_sources_info') {
11         add_sources }
12     opt.on('-t', '--target_VAL', 'set_target_id') {|val|
13         set_target(val)}
14     opt.on('-e', '--edit_FILE', 'open_file') {|file|
15         edit_file(file) }
16     opt.on('-l', '--list_FILE', 'list_files') {|file|
17         list_files(file)}
18     opt.on('-u', '--update_FILE', 'update_file') {|file|
19         update_file(file) }
20     opt.on('-r', '--rsync', 'rsync_files') {rsync_files}
21     opt.on('--database_FILE', 'read_database_file') {|
22         file| db_file(file)}
23     opt.on('--display_FILE', 'display_converted_hikifile'
24         ) {|file| display(file)}
25     opt.on('-c', '--checkdb', 'check_database_file') {
26         check_db}
27     opt.on('--remove_FILE', 'remove_file') {|file|
28         remove_file(file)}
29     opt.on('--move_FILES', 'move_file1,file2',Array) {|
30         files| move_file(files)}
31     opt.on('--euc_FILE', 'translate_file_to_euc') {|file|
32         euc_file(file)}
33     opt.on('--initialize', 'initialize_source_directory')
34         {dir_init() }
35
36     end
37     begin
38         command_parser.parse!(@argv)
39         rescue=> eval
40         p eval
41     end
42     dump_sources
43     exit
44 end
45
46 def show_sources()
47     printf("target_no:%i\n",@src[:target])
48     printf("editor_command:%s\n",@src[:editor_command])
49     check_display_size()
50     header = display_format('id','name','local_directory',
51         'global_uri')

```

```

37
38     puts header
39     puts '-' * header.size
40
41     @src[:srcs].each_with_index{|src,i|
42         target = i==@src[:target] ? '*' : ' '
43         id = target+i.to_s
44         name=src[:nick_name]
45         local=src[:local_dir]
46         global=src[:global_uri]
47         puts display_format(id,name,local,global)
48     }
49 end
50
51 def add_sources
52     cont = {}
53     @data_name.each{|name|
54         printf("%s?", name)
55         tmp = gets.chomp
56         cont[name.to_sym] = tmp
57     }
58     @src[:srcs] << cont
59     show_sources
60 end

```

7 CLIの実行

7.1 Thor

- 実行手順

1. hiki.thor の Hikithor::CLI.start(ARGV) で hikiutils.thor.rb の CLI クラスを呼ぶ
2. hikiutils.thor.rb の CLI クラスのメソッドを順に実行していく

Thor では start(ARGV) を呼び出すことで CLI を開始する。Hikithor::CLI.start(ARGV) を実行されることにより require で呼ばれている hikiutils.thor.rb の CLI コマンドを順に実行する。そして、入力されたコマンドと一致するメソッドを探し、そのコマンドの処理が実行される。

图 10

7.2 optparse

- 実行手順

1. Hiki の `HikiUtils::Command.run(ARGV)` で `hikiutils.rb` の `run` メソッドを呼ぶ
2. `new(argv).execute` で `execute` メソッドが実行される

一方, `optparse` では `Hikiutils::Command.run(ARGV)` を実行される. `require` で呼び出された `hikiutils.rb` で `run` メソッドが実行される. そこでコマンドを登録している `execute` メソッドへ移動し入力したコマンドと対応させる. そして, 対応したコマンドの処理が行われるメソッドに移動することで実行される. このように `optparse` では実行を行うためのメソッドが必要であるが, `Thor` ではクラスのメソッドを順に実行していくため `run` メソッドと `execute` メソッドは必要ない. また, `optparse` での実行手順はメソッドの移動回数が多く複雑であるが, `Thor` は単純で分かりやすいものとなっている.

7.3 コード

- Thor

```
1  #!/usr/bin/env ruby
2
3  require "hikiutils_thor"
4
5  Hikithor::CLI.start(ARGV)
```

```
1  # -*- coding: utf-8 -*-
2  require 'thor'
3  require 'kconv'
4  require 'hikidoc'
5  require 'erb'
6  require "hikiutils/version"
7  require "hikiutils/tmarshal"
8  require "hikiutils/infodb"
9  require 'systemu'
10 require 'fileutils'
11 require 'yaml'
12 require 'pp'
13
14 module Hikithor
15
```

```

16 DATA_FILE=File.join(ENV['HOME'], '.hikirc')
17 attr_accessor :src, :target, :editor_command, :browser, :
    data_name, :l_dir
18
19 class CLI < Thor
20   def initialize(*args)
21     super
22     @data_name=['nick_name', 'local_dir', 'local_uri', '
        global_dir', 'global_uri']
23     data_path = File.join(ENV['HOME'], '.hikirc')
24     DataFiles.prepare(data_path)
25
26     file = File.open(DATA_FILE, 'r')
27     @src = YAML.load(file.read)
28     file.close
29     @target = @src[:target]
30     @l_dir=@src[:srcs][@target][:local_dir]
31     browser = @src[:browser]
32     @browser = (browser==nil) ? 'firefox' : browser

```

- optparse

```

1 #!/usr/bin/env ruby
2
3 require "hikiutils"
4
5 HikiUtils::Command.run(ARGV)

```

```

1   def self.run(argv=[])
2     print "hikiutils: provide utilities for helping hiki
        editing.\n"
3     new(argv).execute
4   end
5
6   def execute
7     @argv << '--help' if @argv.size==0
8     command_parser = OptionParser.new do |opt|
9       opt.on('-v', '--version', 'show program Version.') {
        |v|
10         opt.version = HikiUtils::VERSION
11         puts opt.ver
12       }
13     opt.on('-s', '--show', 'show sources') {show_sources}
14     opt.on('-a', '--add', 'add sources info') {
        add_sources }

```

```

15     opt.on('-t', '--target_VAL', 'set_target_id') {|val|
16         set_target(val) }
17     opt.on('-e', '--edit_FILE', 'open_file') {|file|
18         edit_file(file) }
19     opt.on('-l', '--list_FILE', 'list_files') {|file|
20         list_files(file) }
21     opt.on('-u', '--update_FILE', 'update_file') {|file|
22         update_file(file) }
23     opt.on('-r', '--rsync', 'rsync_files') {rsync_files}
24     opt.on('--database_FILE', 'read_database_file') {|
25         file| db_file(file)}
26     opt.on('--display_FILE', 'display_converted_hikifile'
27         ) {|file| display(f\
28         ile)}
29     opt.on('-c', '--checkdb', 'check_database_file') {
30         check_db}
31     opt.on('--remove_FILE', 'remove_file') {|file|
32         remove_file(file)}
33     opt.on('--move_FILES', 'move_file1,file2',Array) {|
34         files| move_file(file\
35         s)}
36     opt.on('--euc_FILE', 'translate_file_to_euc') {|file|
37         euc_file(file) }
38     opt.on('--initialize', 'initialize_source_directory')
39         {dir_init() }
40
41     end
42     begin
43         command_parser.parse!(@argv)
44     rescue=> eval
45     p eval
46     end
47     dump_sources
48     exit
49 end

```

コードからも Thor のほうが短くなっていることが分かる。よって、Thor と optparse でのコードの違いは以上の部分になるが全体的にも Thor のほうがコードが短くなり、コマンドの定義も簡単に行うことができる。また、実行手順も分かりやすくコードが読みやすいため書き換えもすぐ行うことができるので、より直感的なコマンドを実装することも可能となった。

[1-1] hikidoc, <https://rubygems.org/gems/hikidoc/versions/0.1.0>, <https://github.com/hikidoc>,
アクセス.

[1-2] 「Thor の使い方まとめ」, <http://qiita.com/succi0303/items/32560103190436c9435b>,
2017/1/30 アクセス.

[1-3] 「15.5. optparse - コマンドラインオプション解析器」, [http://docs.python.jp/2/library/opt](http://docs.python.jp/2/library/optparse.html),
2017/1/30 アクセス.

[1-4] 「library optparse」, <https://docs.ruby-lang.org/ja/latest/library/optparse.html>,
2017/1/30 アクセス.