卒業論文

コードエラーの日本語表示アプリ i_error の開発

関西学院大学理工学部

情報科学科 西谷研究室

学籍番号 27019582

岩井 未来

2023 年 3 月

概 要

西谷研究室では, プログラミング言語の Ruby を使用している. 研究を進めるにあたって, 卒研生は共通で mac が配布されており, プログラムの実行やサーバーの起動, システム環境設定ではできない高度な設定を, Mac に最初から搭載されているデフォルトアプリであるターミナルで行う. プログラミングを行なっていると, 実行後必ずと言っていいほどエラー文と対峙することになる. エラーが出たとしても, プログラミング初心者にとってそのエラーの読み方や意味を理解することは難しい. そこで, ターミナルで Ruby を使用する際に, エラー読解の助けとなるようなエラーが日本語化される独自の gem である i_error を作成した.

目次

第1章　序論 3

第2章　手法 4

2.1 Webアプリケーション . . . 4

2.2 cygwin . . . 4

2.2.1 gettext . . . 4

2.2.2 locale . . . 5

2.2.3 Rubyでの使用 . . . 5

2.3 translate-shell . . . 6

2.4 Cloud Translation . . . 7

2.5 i_errorの作成 . . . 8

2.6 パイプ . . . 9

第3章　結果と考察 11

3.1 Cloud Translationの動作確認 . . . 11

3.2 API使用版 . . . 12

3.3 ファイル受け渡し版 . . . 14

3.4 パイプ版 . . . 15

3.5 i_errorのインストールと使用法 . . . 16

第4章　まとめ 17

図 目 次

2.1 trans コマンドエラー. 6
2.2 API キー作成手順. 7
3.1 example.rb 実行失敗. 13
3.2 example.rb 実行成功例. 14

第1章　序論

学生の中には，英語学習の継続が困難な学生が少なからず存在する．学生のあいだは，必要最低限の英語読解能力があれば日常生活を送ることができるが, 就職先の職種がプログラマーであると, 卒業後も非常に多くの英語文章・文書の読解が必要になる．特に英語が苦手であるとどのようなエラーが起きているのか, 対処する以前にエラーの内容が理解できていないことが多い.

そのエラー文に頻出するであろう単語を選んで学習することを考え, Web アプリケーションで英単語学習を始めるためのアプリ作成を検討した. Web アプリケーションの動作としては，手動で単語を入力し, 英単語カードを完成させていく方法を考えた．しかし，手動入力した内容は正確性に欠ける上に，自動でデータを収集する手法を検討したが，うまい方法が見当たらなかった.

そうすると, エラー文に頻出する単語の学習を行うのではなく, 出力されるエラーをターミナルアプリ上で直接日本語表示する方法が思いつく．そこで, 実際にエラー文を日本語表示している Cygwin について調べたが，Cygwin における日本語表示方法は，一般のターミナルアプリでは実現できないことが判明した.

そこで，Shell の機能を使って，エラーを翻訳する手法が考えられる．プログラミングの学習や開発を行なっていると，同じようなエラーが発生することがあり, 日本語表示された過去の出力内容は少しずつではあるが記憶に残る．回数を重ねるうちに，日本語表示を行わずともエラーの意味を理解することができると考えられる．そこで本研究では，ターミナル上のエラー表示を日本語訳するフィルター i_error の作成を目的とした.

第2章　手法

2.1　Web アプリケーション

初心者が英単語カードを学習するように, Web アプリを作成することを考えた. Ruby の代表的なフレームワークの一つである Ruby on Rails は, Web サービスの制作時に使われることが多く, Web アプリの制作を行うためのプラットフォームとして Rails の使用を検討した. 過去に Rails での開発経験があったことと, より Rails の知識や技術を増やしたいという思いから, Rails を使用した Web アプリを制作することを試みた. Web システムや Web アプリにおいてユーザー側の目に触れる部分, すなわちデザイン部分であるフロントエンド [2] については構想があったが, サーバー側のシステムやデータベースであるバックエンドについては, エラー文をデータとして自動的にとる方法が分からず, 手動でデータ集めをしなければいけない状況であったため, Rails を使用した Web アプリの制作は断念した. Web アプリとして実現することはできなくとも, 他の形でアプリを作ることができるのではないかと考えたのが, Ruby を使用したファイル形式のエラー文の日本語表示であった.

2.2　cygwin

2.2.1　gettext

Ruby 以外でエラー文を日本語化した実例である cygwin について調べた. cygwin では, gettext 機構を使用して実行時にメッセージの差し替え（翻訳）をおこなっている. gettext[1] はコマンド名でもあり仕組みでもあるが, 国際化や地域化に対応するライブラリで, 多くのコードを打つ手間が省ける仕組みになっている. ここで国際化とは, アプリケーションをさまざまなロケール（言語や地域）で利用できるようにすることを言い, その国際化さ

れたアプリケーションを, 実際に特定の言語や地域で利用できるようにすることを地域化と言う. メッセージ文字列を国際化や地域化するためのフレームワークである gettext は, 以下の手順で変換を行なっている.

1. ソースコードの修正（文字列を_() で囲む, UTF-8 変換）
2. 翻訳対象のテキスト抽出（.pot ファイルの作成）
3. テキストの翻訳（メッセージファイル.po の作成）
4. 翻訳したファイルからリソースファイル（.mo=バイナリ）を作成
5. システムからリソースを使用

以上で利用準備は完了したが, 実行前に環境変数の確認が必要である.

2.2.2 locale

どの言語を選択するかどうかは環境変数の LANG に従っており, インストールしたパソコンの言語によって cygwin 上の言語は初期設定される. 例えば日本語であれば LANG = ja(言語)_JP(文化). エンコード (UTF-8, Shift-JIS など) と書くことでエラーの日本語化が実現される. ここで文化（JP）の違いというのは, 日付の書き方などが挙げられる. 日本では 2023/3/1 という順番だが, アメリカでは 3/1/2023 といったように日付の順序が異なる. また, ロケール (locale) は言語設定を書き留めておく習慣で, その時使用している言語を確認でき,

```
$ locale
```

でコマンド確認できる.

2.2.3 Ruby での使用

ここまで cygwin の日本語表示方法について触れてきたが, なぜ cygwin を使用しないのか. それは, cygwin の環境下で Ruby を使用することはできても, mac 所有者は cygwin を使用できないからだ. cygwin は Windows で UNIX のコマンドを使えるようにするソフト

であるが, macOS は基本が UNIX ベースのため, 標準で入っているターミナルを使用するため, cygwin を使用することはなく, 肝心のエラーの日本語表示はターミナルで行えない. このように全ての環境で Ruby のエラー出力を日本語化するアプリの開発を目指した.

2.3 translate-shell

コマンドラインから機械翻訳サービスを利用することができる translate-shell[3] は, Google 翻訳のほか, Bing 翻訳, Yandex Translate などにも対応しており, 翻訳された文を発音させることもできる. translate-shell を使用するにあたり, 以下コマンドを入力した.

```
$ git clone https://github.com/soimort/translate-shell
$ cd translate-shell/
$ make
$ brew install gawk
$ make
$ [sudo] make install
$ trans :ja "日本語化したい内容"
```

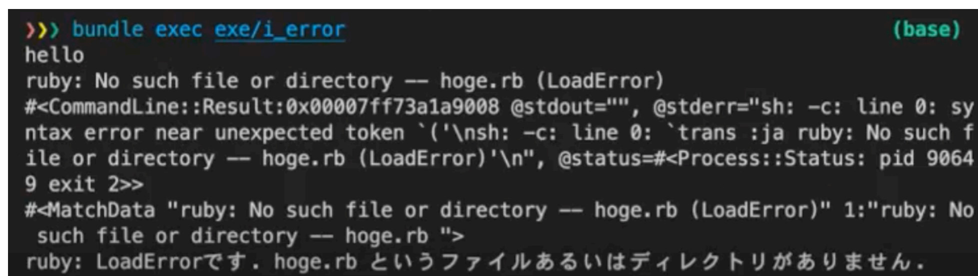

図 2.1: trans コマンドエラー.

trans コマンドの使用により日本語化を試みたが, trans は API で Google の翻訳サイトに移動しているだけで, 実際に Ruby で使用することはできなかった. trans の中身を調べて, 根本にあるエラーを解決するにも, trans を Ruby で使用している情報が見つからなかったため, 日本語表示する別の方法を検討することにした.

2.4 Cloud Translation

Google からは Ruby 言語で Translation サービスにアクセスするための API が提供されている．これは gem としても提供されており, install, coding が容易であるので，この API ライブラリを使って実装することにした．数千もの言語ペアでテキストを動的に翻訳できる Cloud Translation[4] は，ウェブサイトとプログラムを翻訳サービスにプログラムで統合できる．Cloud Translation[5] を使用するにあたり, API キーの発行が必須で, API キー発行の手順は以下の通りである．

図 2.2: API キー作成手順.

第三者に API キーを悪用されることを防ぐために, 他人に見られることがない場所に保管しておく必要がある．Ruby のアプリケーションと連携するために, ターミナル上に API キーを保存する．方法は以下の通りコマンド入力を行う．

```
$ setenv GOOGLE_TRANSLATE_KEY "作成した API キー"
```

あるいは, shell の初期ファイルで設定するようにする．fish の場合は,

```
$ emacs ~/.config/fish/config.fish
```

上記コマンド入力で~/.config/fish/config.fish を開き，先ほど入力したコマンド「setenv GOOGLE_TRANSLATE_KEY ”作成した API キー”」を一番下の行に書き込み，fish がターミナル上で起動されるたびに API キーを参照できるように設定しておく．Google Cloud Translate を Ruby から使用するために, 以下のコマンド [6] で必要な gem がインストールされる．

```
$ gem install google-cloud-translate -v 3.2.2
$ gem install faraday-multipart
```

2.5 i_error の作成

Ruby のライブラリである gem の依存関係とバージョンを管理するためのツールである bundler を使って gem を作成していく．まず bundler が提供している Rubygem 作成のための足場 (scaffoling, スケルトン) を作成する．コマンドを用いて，以下の通り本研究における gem の i_error を作成する．

```
$ bundle gem i_error --exe --git --mit --test=rspec
```

それぞれのオプションは,

1. exe(execute: 実行) ディレクトリ下に実行可能コマンドを自動生成し,
2. git システムをバックアップシステムに用い,
3. オープンソースのライセンス規約として mit を,
4. さらに test として RSpec を使う

ことを意味している．また, 実行権限 (permission) を全員に与えるため, 以下コマンド入力で,

```
$ chmod a+x exe/i_error
```

ファイルやディレクトリのアクセス権を変更する．次に i_error.gemspec を以下のように編集し, 不要な行は削除する．

```
1  # frozen_string_literal: true
2  require_relative "lib/i_error/version"
3  Gem::Specification.new do |spec|
4    spec.name = "i_error"
5    spec.version = IError::VERSION
6    spec.authors = ["名前"]
7    spec.email = ["メールアドレス"]
8    spec.summary = "error translator"
9    spec.description = "error translator"
10   # Specify which files should be added to the gem when it is released.
11   # The ‘git ls-files -z‘ loads the files in the RubyGem
12   # that have been added into git.
```

```
13    spec.files = Dir.chdir(__dir__) do
14      `git ls-files -z`.split("\x0").reject do |f|
15        (f == __FILE__) || f.match(
16         %r{\A(?:(?:bin|test|spec|features)/|\.(?:git|travis|circleci)|appveyor)})
17      end
18    end
19    spec.bindir = "exe"
20    spec.executables = spec.files.grep(%r{\Aexe/}) { |f| File.basename(f) }
21    spec.require_paths = ["lib"]
22    spec.add_dependency "command_line"
23  end
```

今回i_errorというgemを作成したので, gem作成時に生成されるlib/i_error.rbに「puts "hello"」などのテスト内容を書き込み,

```
$ bundle exec exe/i_error
$ rake install:local
```

上記二つのコードを打ち込むことで, i_errorを使える状態にする. 冒頭の「bundle exec」は現在のプロジェクトでインストールされたgemを使用してコマンドを実行することができ, 「bundle exec」がなければシステム共通の場所でコマンドを実行することになる. すなわち, gemfile.lockに基づいた環境下でコマンドを実行するために必要である.

2.6 パイプ

i_errorの動作として, 当初テキストによる受け渡しを考えた. しかし, これではWebアプリに貼り付けるのと手間に大差がない. そこでパイプを利用することを考えた. あるデータに対して複数のコマンドを連続で適用したい時, 1コマンドを実行する度に中間ファイルを出力するが, このような処理を効率良く行うためにフィルタとパイプ[8]という仕組みが存在する. フィルタは標準入力から入力を受け取り, 結果を標準出力へ出力するコマンドやプログラムである. フィルタはあるコマンドの標準出力を, 次のコマンドの標準入力とつなぐことによって, 中間ファイルを介さずに同じ処理ができるため, コマンド間の連携が容易になる. また, この処理を実現するのがパイプの役割で, 複数のコマンドを連携させることで複雑な処理を容易に実現できる. もし日本語化する際にファイル依存を使用すると,

```
$ cd test &> sample.txt
$ i_error sample.txt
```

二度入力を行う必要があり二度手間であるが, 一方でパイプは,

```
$ cd test 2>&1 | i_error
```

1 行の入力で済むため, 今回はより効率の良いパイプを採用することにした. またコード内にある「2>&1」という記号の意味は, そもそもコマンド出力には標準出力と標準エラー出力の 2 種類があり, 標準出力には番号の 1 が, 標準エラー出力には番号の 2 が与えられている. この番号と >（リダイレクト）を使用することでそれぞれの出力を呼び出すことができる. リダイレクトとはコマンドの出力先を変更する記号で, 指定したファイルにコマンドの結果を出力する. 続いて&（アンパサンド）は, 標準出力と標準エラー出力をまとめて出力することを意味する. つまり,「2>&1」を使用することで, 標準出力と標準エラー出力をまとめて i_error に受け渡し, 処理することができる.

第3章　結果と考察

3.1　Cloud Translationの動作確認

まず, Cloud Translation の動作を確認するため, 参考文献 [5] に従って, 以下のような簡単なコードを作った. ここでは, Google が提供している Translation サイトに対して, 直接 HTTP(hyper text transfer protocol) のメッセージ（プロトコル）を送ることで, そこからの反応 (response) を JSON 形式で受け取り, どのように動くかを確かめている.

```
 1  require 'net/http'
 2  require 'uri'
 3  require 'json'
 4
 5  def translate(contents, source, target)
 6   url = URI.parse('https://translation.googleapis.com/language/translate/v2')
 7   params = {
 8     q: contents,
 9     source: source,
10     target: target,
11     format: "text",
12     key: ENV["GOOGLE_TRANSLATE_KEY"]
13   }
14   url.query = URI.encode_www_form(params)
15   p res = Net::HTTP.get_response(url)
16   JSON.parse(res.body)['data']['translations'].first['translatedText']
17  end
18
19  p translate("hello", :en, :es)
```

必要な library を読み込み, translate というメソッドを定義する. このメソッドは contents に書かれた内容を source から target に翻訳することを意図している. 実際の動作を順に追いかけると,

1. URI.parse で Google Translation をパースし,
2. params で必要な情報を渡している.
3. これを URI に www_form でエンコードして,
4. Net::HTTP で反応を受け取る.(get_response)
5. その中身 (body) を JSON によって parse し,
6. 翻訳された内容 (translatedText) を取り出す.

となる. これを実行すると,

```
$ ruby lib/run_check.rb
#<Net::HTTPOK 200 OK readbody=true>
"Hola"
```

と出力される. これは res を直接 'p' しているので, その Class が簡単に出力された後に, translate メソッドらの戻り値を出力していることがわかる.

3.2 API 使用版

Google が提供している API のライブラリを Ruby で直接呼び出して動かすことを試みる. これによって, net/http や uri などの Web サーバーとの通信処理を担う低層のライブラリが不要となり, それを API が自動で処理してくれる. さらに, URL が自動で設定される. これによって, コードを省略することが可能となる. 参考文献 [9] の例をもとに, 以下のような example.rb を作成した.

```
1  # file: example.rb
2  require 'multipart/post'
3  require "google/cloud/translate/v2"
4
5  client = Google::Cloud::Translate::V2.new(
6    key: ENV["GOOGLE_TRANSLATE_KEY"]
7  )
8
```

```
 9  translation = client.translate "Hello world!", to: "la"
10  translation.text #=> "Salve mundi!"
```

これによって, run_check.rb の translate メソッドで行なっていたいくつかの変換メソッドの呼び出しが省略されて translate だけで変換されることが期待できた. しかし, 下記コマンド入力で実行してみると,

```
$ ruby example.rb
```

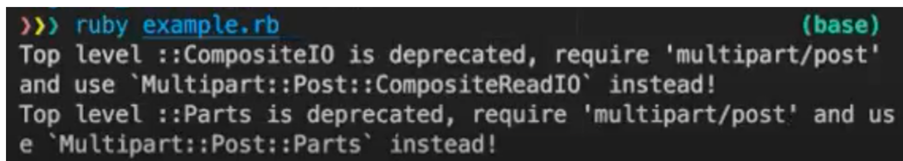

図 3.1: example.rb 実行失敗.

というエラーが出力された. 同じようなエラー内容の報告は見つからず, 読み解けなかった. しかし, エラーの内容から multipart が古い (deprecated) ようなので, そのライブラリを消して, 少し書き換えた.

```
 1  # file: example.rb
 2  require "google/cloud/translate/v2"
 3
 4  client = Google::Cloud::Translate::V2.new(
 5    key: ENV["GOOGLE_TRANSLATE_KEY"]
 6  )
 7  contents = "environment.rb:5:in ’translate’:
 8  wrong number of arguments (given 0, expected 3) (ArgumentError)
 9  from environment.rb:17:in ’<main>’"
10
11  translation = client.translate contents, to: "ja"
12
13  puts translation #=> Salve mundi!
14
15  translation.from #=> "en"
16  translation.origin #=> "Hello world!"
17  translation.to #=> "la"
18  translation.text #=> "Salve mundi!"
```

```
$ ruby example.rb
```

```
>>> ruby example.rb                                                    (base)
environment.rb:5:in &#39;translate&#39;: 引数の数が間違っています (指定
された 0、期待される 3) (ArgumentError) from environment.rb:17:in &#39;
<main> &#39;
```

図 3.2: example.rb 実行成功例.

再度上記コマンドを実行すると, 日本語が返ってきており, contents で記述したエラーの日本語化に成功した. しかし, この方法はエラーが出る度に 7 行目にエラー内容を書き込み, ファイルを編集する必要がある. それではネットで日本語訳を検索することと何ら変わりはなく, むしろネット検索より手間がかかるため, 最善の方法とは言えなかった.

3.3　ファイル受け渡し版

そこで, ファイルとして受け渡す日本語化するコードを作成するため, 以下のように書き換える.

```
1  require "google/cloud/translate/v2"
2
3  def from_file(file)
4    contents = File.read(file)
5
6    client = Google::Cloud::Translate::V2.new(
7       key: ENV["GOOGLE_TRANSLATE_KEY"]
8    )
9
10   translation = client.translate contents, to: "ja"
11
12   puts translation #=> Salve mundi!
13
14   translation.from #=> "en"
15   translation.origin #=> "Hello world!"
16   translation.to #=> "la"
17   translation.text #=> "Salve mundi!"
18 end
19
20 from_file('./tmp.txt')
```

'./tmp.txt' というファイルを読み込んで, その内容を contents に代入して, それを日本

語に翻訳している.

```
$ cat tmp.txt
ruby: No such file or directory -- hoge.rb (LoadError)
```

に対して動作させると,

```
$ ruby from_file.rb
ruby: そのようなファイルやディレクトリはありません -- hoge.rb (LoadError)
```

となる. これでもファイルへの保存などの手間となる.

3.4 パイプ版

そこで, パイプを使用することにした. 当初, エラーの受け取りがうまくいかなかったが, 第2章の手法で述べた通り, パイプのリダイレクトを使って受け取ることが可能となった.

```
 1  lines = STDIN.readlines
 2  client = Google::Cloud::Translate::V2.new(
 3    key: ENV["GOOGLE_TRANSLATE_KEY"]
 4  )
 5
 6  puts lines
 7
 8  lines.each do |line|
 9    translation = client.translate line, to: "ja"
10    puts translation #=> Salve mundi!
11  end
```

これによって Ruby エラーの日本語化が可能となった.

3.5 i_error のインストールと使用法

ここまで様々な方法を検討してきたが, パイプを使用した方法で Ruby のエラー文を日本語表示する方法は, 最も効率が良く最善と言える. エラーが出たその後, 日本語表示するためのコマンドを入力することで, エラー文の日本語化が実現されるため, ページ移動やファイル編集, 別のアプリに移動するといった手間が一切ない. i_error のインストールは以下コマンドをターミナルに入力することで実現する.

```
$ git clone git@github.com:IwaiMIku/i_error.git
$ cd i_error
$ rake install:local
```

例えば, この i_error の階層にはない test という階層への移動を試みると,

```
$ cd test
cd: The directory 'test' does not exist
```

上記のようなエラー内容が返ってくる. ここでパイプを使用し, 下記コマンドを入力すると,

```
$ cd test 2>&1 | i_error
cd: The directory 'test' does not exist
cd: ディレクトリ &#39;test&#39; が存在しません
```

エラー内容とその日本語訳が出力された. 日本語化した際, test の前後に「'」という文字列がついているが, これは'（アポストロフィ）の html 特殊表記で, 変換によって取り除くことが可能である.

第4章　まとめ

今まで Mac や Linux のターミナル上では, Window 上の Cygwin と異なりエラーの日本語表示が行われなかったが, i_error の作成により全ての環境でエラー出力を日本語化することが可能となった. ターミナル上のコマンドで返ってきたエラーの日本語表示を行うことで, エラー文の内容や読み方を理解することができるようになった. また, 当初は API を使用したテキストによる受け渡しを考えたが, パイプを使用した方法でエラー文を日本語表示することで, より手間を省いた方法をとることができた. 学生生活の中で様々なプログラミング言語を学習するが, エラー内容を理解できない学生にとって, 何度も英語のエラー文と遭遇することは恐怖そのものである. しかし, 本研究における i_error の使用により, エラー文の読み方や内容理解に繋がり, 円滑なコード開発が期待される. 今回は自身の将来のために i_error を作成したが, 西谷研究室では数あるプログラミング言語の中でも主に Ruby を使用することが多く, 今後西谷研究室で Ruby を研究するであろう後輩の役に立つことがあれば嬉しく思う.

謝辞

本研究を進めていく中で，毎週のゼミの時間以外にも西谷教授が熱心に指導してくださったこと，心より感謝申し上げます．また，同じ研究室所属の卒研生や院生にはたくさんの助言をいただき，研究に取り組む上で心の励みになったこと、感謝いたします．本当にありがとうございました．

参考文献

[1] i18n してますか？, https://el.jibun.atmarkit.co.jp/kaigaiengineer/2010/08/i18ngettextphp-.html (accessd on 11 Aug 2010).

[2] フロントエンドとバックエンドとは？, https://hnavi.co.jp/knowledge/blog/front-end-back-end/ (accessd on 28 Sep 2022).

[3] translate-shell, https://github.com/soimort/translate-shell (accessd on 1 Feb 2023).

[4] Cloud Translation, https://cloud.google.com/translate/docs?hl=ja (accessd on 3 Feb 2023).

[5] Cloud Translation を Ruby で実行する, https://ts223.hatenablog.com/entry/gcp/translation (accessd on 14 Feb 2021).

[6] google-cloud-translate 3.2.2, https://rubygems.org/gems/google-cloud-translate/versions/3.2.2 (accessd on 12 Jul 2021).

[7] Ruby -gem ライブラリの作成, https://www.mk-mode.com/blog/2016/06/26/ruby-making-gem-by-bundler-tdd/(accessd on 26 Jun 2016).

[8] ファイルの読み書きとコマンドの繋ぎ方, https://sc.megabank.tohoku.ac.jp/ph3-doc/howto/linux/shell-io.html (accessd on 2022).

[9] Cloud Translation V2 API -Module, https://cloud.google.com/ruby/docs/reference/google-cloud-translate-v2/latest/Google-Cloud-Translate-V2 (accessd on 24 Oct 2022).