Rails 1.0 のコードで学ぶ find_by_* と method_missing の仕組み

2025/03/01

TokyoWomen.rb #1

Mai Muta @maimux2x

Mai Muta(maimu)

@maimux2x

フィヨルドブートキャンプ卒業生🎓

甘党🍩


永和システムマネジメント
アジャイル事業部
ホーム
English
お問合せ
esm.co.jp
事業部紹介
サービス
事例
採用情報
ニュース
ブログ
Ruby
×
Agile
永和システムマネジメントアジャイル事業部は、
Rubyとアジャイルに関連する技術力をさらに先鋭化させ、
業界にとって必要不可欠な存在となるため、より専門性を高めた組織です。
アジャイル事業部について
アジャイル事業部に入りたい


しんめ.rb

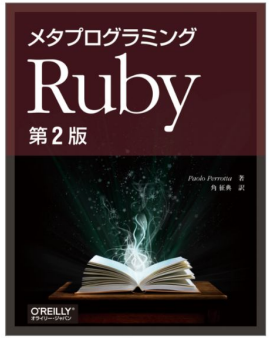
TokyoWomen.rb #1のテーマ

Rubyっておもしろい！

Rubyっておもしろい！~その1~

メタプログラミングRuby 第2版

Paolo Perrotta　著、角 征典　訳



TOPICS:
Programming , Ruby

発行年月日:
2015年10月

PRINT LENGTH:
292

ISBN:
978-4-87311-743-0

原書:
Metaprogramming Ruby 2

FORMAT:
Print PDF EPUB

Ebook
3,300円
Ebookを購入する

Print
3,300円
書籍のご注文はオーム社サイトへ

本書はRubyを使ったメタプログラミングについて解説する書籍です。メタプログラミングとは、プログラミングコードを記述するコードを記述することを意味します。前半では、メタプログラミングの背景にある基本的な考えを紹介しながら、動的ディスパッチ、ゴーストメソッド、フラットスコープといったさまざまな「魔術」を紹介します。後半では、ケーススタディとしてRailsを使ったメタプログラミングの実例を紹介します。今回の改訂では、Ruby 2とRails 4に対応し、ほぼすべての内容を刷新。Rubyを使ったメタプログラミングの魔術をマスターし、自由自在にプログラミングをしたい開発者必携の一冊です。

https://www.oreilly.co.jp/books/9784873117430/

メタプログラミングRuby 第2版

自分が知っているRubyから、

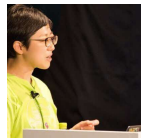
知らない世界が広がるおもしろさ

Rubyっておもしろい！~その2~

Rails 1.0 のコードリーディング

なぜRails 1.0 なのか？

きっかけ

Fukuoka RubyistKaigi 04 &&  しおいさん！

一人で読むのは難しそう・・・

まいむ
@maimux2x

これ、やりたい機運が高まってるんだけど興味ある人いるかな〜？

まいむ @maimux2x · 2024年9月30日

Rails1.0のコードを読みたいのだけど、これはしんめでやったら参加してくれる人はいるだろうか🤔アドバイザーポジションの方もいないと厳しいか。

しんめ.rbでコードリーディングを開催

Rails 1.0 のコードリーディング

RailsがRubyでどのように実装されているか

月日を経て実装がどう変化しているのか

に触れたおもしろさ

特におもしろいと感じた部分

find_by_* と method_missing

おもしろいと感じた理由

一緒に見ていきましょう！！

準備

1. https://github.com/rails/rails をローカルにclone
2. $ cd rails
3. $ git checkout refs/tags/v1.0.0

準備

1. $ curl https://gist.githubusercontent.com/en30/d4fff101aec19c546da6b0b415c6cde6/raw/26c845254a3649b84c101ea09b5a8277ec14cc16/gistfile1.txt | patch -p1
2. $ cd activerecord
3. $ rake rdoc
4. $ open doc/ActiveRecord/Base.html

Rails 1.0 の頃の世界にいざ出発〜！！

Rails頻出メソッドといえば

find, find_by, where

Rails 1.0 の頃は

find, ~~find_by~~, ~~where~~

疑問

find_by や where がなくて

どうやってデータの検索をしていたのか？

RDocを読んでみる

- ::establish_connection
- ::exists?
- ::extract_options_from_args!
- ::find
- ::find_by_sql
- ::increment_counter
- ::inheritance_column

RDocを読んでみる

find(*args) click to toggle source

Find operates with three different retrieval approaches:

- Find by id: This can either be a specific id (1), a list of ids (1, 5, 6), or an array of ids ([5, 6, 10]). If no record can be found for all of the listed ids, then RecordNotFound will be raised.
- Find first: This will return the first record matched by the options used. These options can either be specific conditions or merely an order. If no record can matched, nil is returned.
- Find all: This will return all the records matched by the options used. If no records are found, an empty array is returned.

All approaches accept an option hash as their last parameter. The options are:

- `:conditions`: An SQL fragment like “administrator = 1” or [“user_name = ?”, username]. See conditions in the intro.
- `:order`: An SQL fragment like “created_at DESC, name”.
- `:group`: An attribute name by which the result should be grouped. Uses the GROUP BY SQL-clause.
- `:limit`: An integer determining the limit on the number of rows that should be returned.
- `:offset`: An integer determining the offset from where the rows should be fetched. So at 5, it would skip the first 4 rows.
- `:joins`: An SQL fragment for additional joins like “LEFT JOIN comments ON comments.post_id = id”. (Rarely needed). The records will be returned read-only since they will have attributes that do not correspond to the table’s columns. Pass :readonly => false to override.
- `:include`: Names associations that should be loaded alongside using LEFT OUTER JOINs. The symbols named refer to already defined associations. See eager loading under Associations.
- `:select`: By default, this is * as in SELECT * FROM, but can be changed if you for example want to do a join, but not include the joined columns.
- `:readonly`: Mark the returned records read-only so they cannot be saved or updated.

・findの引数で色々指定できる

・:conditionsがwhere相当？

RDocを読んでみる

Examples for find by id:

```
Person.find(1)       # returns the object for ID = 1
Person.find(1, 2, 6) # returns an array for objects with IDs in (1, 2, 6)
Person.find([7, 17]) # returns an array for objects with IDs in (7, 17)
Person.find([1])     # returns an array for objects the object with ID = 1
Person.find(1, :conditions => "administrator = 1", :order => "created_on DESC")
```

Examples for find first:

```
Person.find(:first) # returns the first object fetched by SELECT * FROM people
Person.find(:first, :conditions => [ "user_name = ?", user_name])
Person.find(:first, :order => "created_on DESC", :offset => 5)
```

Examples for find all:

```
Person.find(:all) # returns an array of objects for all the rows fetched by SELECT * FROM people
Person.find(:all, :conditions => [ "category IN (?)", categories], :limit => 50)
Person.find(:all, :offset => 10, :limit => 10)
Person.find(:all, :include => [ :account, :friends ])
Person.find(:all, :group => "category")
```

疑問

find_by や where がなくて

どうやってデータの検索をしていたのか？

答え

find で処理されていた！

例を見ながら find メソッドの使い方を確認

id	name	email	active
1	Alice	alice@example.com	true
2	Bob	bob@example.com	false
3	Carol	carol@example.com	false

name が Alice のレコードを1件取得したい場合

⬇️Rails 1.0 の時はこのように書いていた

```
User.find(:first, :conditions => "name = ?", "Alice")
```

```
# => #<User id: 1, name: "Alice", email: "alice@example.com", active: true>
```

⬇️今だと find_by で書くことができる

```
User.find_by(name: "Alice")
```

```
# => #<User id: 1, name: "Alice", email: "alice@example.com", active: true>
```

active が false の user を取得したい場合

⬇️Rails 1.0 の時はこのように書いていた

```
User.find(:all, :conditions => "active = ?", false)
# => [#<User id: 2, name: "Bob", email: "bob@example.com", active: false>,
      #<User id: 3, name: "Carol", email: "carol@example.com", active: false>]
```

⬇️今だと where で書くことができる

```
User.where(active: false)
# => [#<User id: 2, name: "Bob", email: "bob@example.com", active: false>,
      #<User id: 3, name: "Carol", email: "carol@example.com", active: false>]
```

findメソッドまとめ

- find メソッドの役割の範囲が今よりも広い
- :first, id, :all, :conditionsなどの引数の指定が可能だった
- :conditions などで指定された条件に応じてSQLを組み立ててデータを検索して返していた
    - Rails 1.0 の find メソッドの実装もおもしろいです！！

ここまでの感想

find メソッドで検索条件を書くのが大変そう・・・

もう一度RDocを読んでみる

Dynamic attribute-based finders¶ ↥

Dynamic attribute-based finders are a cleaner way of getting (and/or creating) objects by simple queries without turning to SQL. They work by appending the name of an attribute to `find_by_` or `find_all_by_`, so you get finders like Person.find_by_user_name, Person.find_all_by_last_name, Payment.find_by_transaction_id. So instead of writing `Person.find(:first, ["user_name = ?", user_name])`, you just do `Person.find_by_user_name(user_name)`. And instead of writing `Person.find(:all, ["last_name = ?", last_name])`, you just do `Person.find_all_by_last_name(last_name)`.

It's also possible to use multiple attributes in the same find by separating them with "*and*", so you get finders like `Person.find_by_user_name_and_password` or even `Payment.find_by_purchaser_and_state_and_country`. So instead of writing `Person.find(:first, ["user_name = ? AND password = ?", user_name, password])`, you just do `Person.find_by_user_name_and_password(user_name, password)`.

Userの名前とメールアドレスを条件に検索したい場合

User.find(:first, [:conditions => “name = ? AND email = ?” ″Alice″, ″alice@example.com″])

は以下のように書くことができる

User.find_by_name_and_email(″Alice″, ″alice@example.com″)
# => #<User id: 1, name: ″Alice″, email: ″alice@example.com″, active: true>

User.find_by_name_and_email_and_acitve･･･

のように属性名は and で繋いでいくことできるらしい

find メソッドで検索条件を書くのが大変そう・・・

find_by_* という別の書き方がある！

疑問

find_by_＊ の ＊ 部分は

メソッド定義されてないのに

どうやって検索結果を取得して返してるのか？

メソッド定義されていないといえば

instance method BasicObject#method_missing

`method_missing(name, *args) -> object` [permalink][rdoc][edit]

呼びだされたメソッドが定義されていなかった時、Rubyインタプリタがこのメソッドを呼び出します。

呼び出しに失敗したメソッドの名前 (**Symbol**) が name にその時の引数が第二引数以降に渡されます。

デフォルトではこのメソッドは例外 **NoMethodError** を発生させます。

[PARAM] name:

未定義メソッドの名前（シンボル）です。

[PARAM] args:

未定義メソッドに渡された引数です。

[RETURN]

ユーザー定義の method_missing メソッドの返り値が未定義メソッドの返り値であるかのように見えます。

https://docs.ruby-lang.org/ja/latest/method/BasicObject/i/method_missing.html

method_missing を探してみる

```
def method_missing(method_id, *arguments)
  if match = /find_(all_by|by)_([_a-zA-Z]\w*)/.match(method_id.to_s)
    finder = determine_finder(match)

    attribute_names = extract_attribute_names_from_match(match)
    super unless all_attributes_exists?(attribute_names)

    conditions = construct_conditions_from_arguments(attribute_names, arguments)

    if arguments[attribute_names.length].is_a?(Hash)
      find(finder, { :conditions => conditions }.update(arguments[attribute_names.length]))
    else
      send("find_#{finder}", conditions, *arguments[attribute_names.length..-1]) # deprecated API
    end
  # …
end
```

BasicObject#method_missingをオーバーライドしている！！

activerecord/lib/active_record/base.rb:970

今回は find_by_* のパターンのみを追っていきます

method_missing の実装を見ていく

```
def method_missing(method_id, *arguments)
  if match = /find_(all_by|by)_([_a-zA-Z]\w*)/.match(method_id.to_s)

    finder = determine_finder(match)
```

(all_by|by) で、find_all_by_ と find_by_ のどちらであるかを判定

([_a-zA-Z]¥w*) で、属性名（例：name、name_and_email）を取得

method_missing の実装を見ていく

```
def method_missing(method_id, *arguments)
  if match = /find_(all_by|by)_([_a-zA-Z]\w*)/.match(method_id.to_s)
  finder = determine_finder(match)
  # …
```

```
def determine_finder(match)
  match.captures.first == 'all_by' ? :all : :first
end
```

変数 match をcaptures した最初の要素をチェックして :all または :firstを返している

activerecord/lib/active_record/base.rb:995

method_missing の実装を見ていく

```
attribute_names = extract_attribute_names_from_match(match)
super unless all_attributes_exists?(attribute_names)
```

```
def extract_attribute_names_from_match(match)
  match.captures.last.split('_and_')
end
```

変数 match を captures した最後の要素に対して split で属性名の配列（例：[“name”] や [“name”, “email”]）を返している
属性名が存在しない場合は super で通常の method_missing に処理を任せている

activerecord/lib/active_record/base.rb:999

method_missing の実装を見ていく

```
conditions = construct_conditions_from_arguments(attribute_names, arguments)
```

```
def construct_conditions_from_arguments(attribute_names, arguments)
  conditions = []
  attribute_names.each_with_index { |name, idx| conditions <<
    "#{table_name}.#{connection.quote_column_name(name)} / # 本来は1行
     #{attribute_condition(arguments[idx])} "                    }

  [ conditions.join(" AND "), *arguments[0...attribute_names.length] ]
end
```

find_by_name(″Alice″) の場合、[″users.name = ?″, ″Alice″] となるように属性名と指定された値を組み立てている

activerecord/lib/active_record/base.rb:1003

method_missing の実装を見ていく

```
if arguments[attribute_names.length].is_a?(Hash)
  find(finder, { :conditions => conditions }.  # 本来は1行
  update(arguments[attribute_names.length]))
else
  # deprecated API
  send("find_#{finder}", conditions, *arguments[attribute_names.length..-1])
end
```

引数の最後に :order や :limit などのオプションが存在する場合は :conditions のハッシュにマージして組み立てた条件を find メソッドに渡している
else の場合は deperecated API とコメントされているが、ここでは send で find_first を実行している

activerecord/lib/active_record/base.rb:979

find_first 定義箇所を見てみる

```
module ActiveRecord
  class Base
    class << self
    # …
      def find_first(conditions = nil, orderings = nil, joins = nil) # :nodoc:
        find(:first, :conditions => conditions, :order => orderings, :joins => joins)
      end
    # …
  end
end
```

method_missing の send() はここを呼び出している
さらに古いRailsのバージョンで使われていたらしい?

activerecord/lib/active_record/deprecated_finders.rb:21

method_missing の実装を見ていく

```
if arguments[attribute_names.length].is_a?(Hash)
  find(finder, { :conditions => conditions }.   # 本来は1行
  update(arguments[attribute_names.length]))
else
  # deprecated API
  send("find_#{finder}", conditions, *arguments[attribute_names.length..-1])
end
```

スライド上で例に使用したサンプルコードでは else 節が実行されてsend が find_first を呼び出して最終的に find メソッドが実行される
deprecated API となっているのは互換性を維持するためだろうと推測

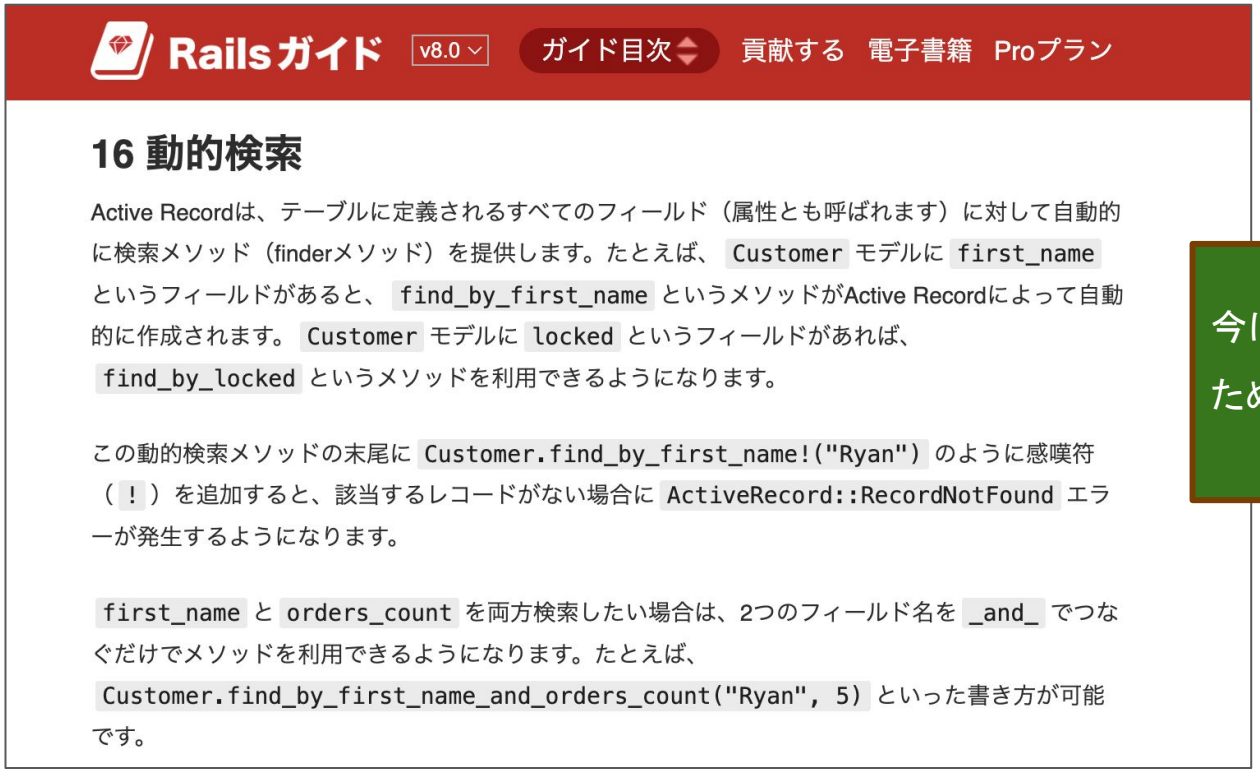
activerecord/lib/active_record/base.rb:979

find_by_* が method_missing で
どのように処理されているかが
見終わりました🙌

find_by_* とmethod_missing まとめ

- find メソッドに:first, :all, options などを指定する代わりに使用する
- find_by_* はメソッドとして個別に定義されているわけではなく method_missing をオーバーライドして動的に実現されている
- method_missing の処理としては最終的に find メソッドが実行される
    - 条件分岐で偽の場合は send メソッド で find_first メソッドを経由していた

実は find_by_* は今もあります



今は find_by や where が使えるため、あえて使う必要はない

https://railsguides.jp/active_record_querying.html#%E5%8B%95%E7%9A%84%E6%A4%9C%E7%B4%A2

Rails 8.0 の find_by_* の実装箇所

```
module ActiveRecord
  module DynamicMatchers # :nodoc:
    private
      def respond_to_missing?(name, _)
        if self == Base
          super
        else
          match = Method.match(self, name)
          match && match.valid? || super
        end
      end

      def method_missing
```

module として切り出されていて、
内容も Rails 1.0 の頃とはかなり違う！

activerecord/lib/active_record/dynamic_matchers.rb

特におもしろいと感じた部分

find_by_* と method_missing

おもしろいと感じた理由

Rails 1.0 のころは find_by や where など

今は当たり前にあるメソッドがなかったという

当時特有の背景

おもしろいと感じた理由

method_missing はお仕事のコードでは使わないので、Rails で使用箇所を見てテンションが上がった！！

コードの内容もわかりやすい！

おもしろいと感じた理由

find_by_* は今も使用できるけれど、

実装が Rails 1.0 の頃とは全然違う！

月日を経た Rails の変遷

Ruby っておもしろい！！