Expose Pattern Matching interfaces in core classes

Pattern matching is an exceptionally powerful feature in modern versions of Ruby, but it has one critical weakness that we should discuss:

It is only as powerful as the number of classes which implement its interfaces.

The more common these interfaces become, the more powerful pattern matching will become for everyday use in any scenario.

Areas of Attention

That said, what are some classes in core Ruby where it may make sense to implement pattern matching interfaces, and what do we gain from them? I will provide an abbreviated list, but can look to qualify a larger list of potentials if this is of interest.

Set

Currently Set does not implement deconstruct. Especially Enumerable-like and Array like entities make sense here:

```ruby
# Hypothetical implementation
class Set
  alias_method :deconstruct, :to_a
end
Set[1, 2, 3] in [1, 2, *]
  => true
```

Matrix

Speaking of Array-like structures, Matrix may make sense as well:

```ruby
class Matrix
  alias_method :deconstruct, :to_a
end
  #=> :deconstruct
Matrix[[25, 93], [-1, 66]] in [[20..30, _], [..0, _]]
  => true
```

CSV

In the case of headers especially this can become very powerful with deconstruct_keys:

```ruby
require "csv"
require "net/http"
require "json"

# Hypothetical implementation
class CSV::Row
  def deconstruct_keys(keys)
    # Symbol/String is contentious, yes, I will address in a moment
    self.to_h.transform_keys(&:to_sym)
  end
end
```
# Creating some sample data for example:
json_data = URI(\"https://jsonplaceholder.typicode.com/todos\")
  .then { Net::HTTP.get(_1) }
  .then { JSON.parse(_1, symbolize_names: true) }

headers = json_data.first.keys
rows = json_data.map(&:values)

# Yes yes, hacky
csv_data = CSV.generate do |csv|
  csv << headers
  rows.each { csv << _1 }
end.then { CSV.parse(_1, headers: true) }

# But can provide very interesting results:
csv_data.select { _1 in userId: "1", completed: "true" }.size
# => 11

Though this one does raise the broader question of the conflation of Symbol and String keys for our convenience. Given that Ruby has a habit of coercing between the two in other cases I do not find this to be against the spirit of Ruby.

### RegExp MatchData

In a similar line of thinking to the CSV I believe this would present interesting opportunities, though does raise the question of what to do with nil types (perhaps return [] and {} respectively? May be hacky though)

class MatchData
  alias_method :deconstruct, :to_a

  def deconstruct_keys(keys)
    namedcaptures.transform_keys(&:to_sym).slice(*keys)
  end
end

IP_REGEX = /
  (?<first_octet>\d{1,3})\.
  (?<second_octet>\d{1,3})\.
  (?<third_octet>\d{1,3})\.
  (?<fourth_octet>\d{1,3})\n/x

'192.168.1.1'.match(IP_REGEX) in {
  first_octet: '198',
  fourth_octet: '1'
}
# => true

As with before though, we do risk setting a precedent on the conflation of Symbol and String keys when it is convenient to us, so may be worth proceeding with caution there.

### OpenStruct

Much like Struct I believe there's a good case to make here:

class OpenStruct
  def deconstruct_keys(keys) = keys ? to_h.slice(*keys) : to_h
end

me = OpenStruct.new(name: 'Brandon', age: 31)
me in { name: /^B/ }
# => true

### Other Thoughts
I believe there is great potential in the core of Ruby to spread the pattern matching interface. The more common it becomes the more useful it will be to users.

Especially if this were to be adopted into places like Rack, Net::HTTP, JSON, and other areas where frequently more imperative deconstructions and querying are already commonly used.

I bring this up, rather than opening PRs, as I would like to see whether or not the core Ruby team is interested in these types of PRs and work of finding where else these interfaces may make sense.

If you would like my more complete thoughts on this, and considerations for pattern matching interfaces in Ruby, I had written Pattern Matching Interfaces in Ruby some time ago to note concerns, potential guidelines, and other considerations.

History

#1 - 06/08/2022 07:28 AM - znz (Kazuhiro NISHIYAMA)
I think it's a good idea to create a pull request in each repository.

https://github.com/ruby/set
https://github.com/ruby/matrix
https://github.com/ruby/csv
https://github.com/ruby/ruby (MatchData is built-in class)
https://github.com/ruby/ostruct

#2 - 06/09/2022 03:30 AM - matz (Yukihiro Matsumoto)
I agree with part of the proposal. Set and OpenStruct do not have order of elements, so it's not suitable for converting to arrays (ordered collections), where others are OK.

Matz.

#3 - 06/09/2022 03:41 AM - baweaver (Brandon Weaver)
matz (Yukihiro Matsumoto) wrote in #note-2:

I agree with part of the proposal. Set and OpenStruct do not have order of elements, so it's not suitable for converting to arrays (ordered collections), where others are OK.

Matz.

Understandable on Array-like representation of those. If I have more areas where I think it might make sense what would be the most effective way to communicate those? I can open them as PRs if you would like, rather than mention on the bug tracker, as each are fairly isolated changes.

#4 - 06/09/2022 04:58 AM - baweaver (Brandon Weaver)
I have opened one PR against CSV: https://github.com/ruby/ruby/pull/5994

I have noted the potential precedent this may set in the PR, and the larger later discussion on whether or not we treat pattern matching keys as Symbols (literal interpretation) or as keywords that we can reasonably infer on String-like structures. That, of course, is a much larger discussion.

The CSV change I believe does not cross the line into that larger discussion.

#5 - 06/09/2022 07:24 AM - baweaver (Brandon Weaver)
I'll write a bit more on this later in longer form, but I do believe we have a very interesting and amusingly very Ruby idea hiding in here:

The keys passed to pattern matching's deconstruct_keys are representations before they are Symbols.

For literal Hash matching yes, they represent literal Symbol keys. For objects of any other type they represent method calls, keys, variables, or several other items.

Taking this viewpoint opens up a lot of possibility, but it also opens a very interesting one that I do believe there is already precedent for:

String key coercions for CSV and other similar types.

The precedent exists that Ruby will do the reasonable thing as long as it can reasonably guess what a user meant. It's the reason method takes either, and it's one of many.

If we consider keys passed to deconstruct_keys as representations of query params rather than as literal Symbols we open up a lot of expressiveness.

This may seem off topic, but a few of the above are technically String keyed like CSV::Row, MatchData, and probably others that I haven't found yet.
I agree with MatchData. And I think deconstruct for the class should be an alias to captures instead of to_a.

Matz.