# Ruby master - Feature #15049

## [Request] Easily access all keyword arguments within a method

08/30/2018 07:38 PM - bherms (Bradley Herman)

| | | |
|---|---|---|
| **Status:** | Feedback | |
| **Priority:** | Normal | |
| **Assignee:** | | |
| **Target version:** | | |

**Description**

As a developer, I'd like to leverage the power of keyword arguments (requirements, defaults, etc) and then be able to access a hash of all the arguments supplied.

```
def foo(bar:, baz: 1, qux: 2...)
  post('/', body: kargs)
end
```

This is currently possible by leveraging the RubyVM debug inspector and some meta programming to retrieve the binding and name of the calling method. There is a gem https://github.com/banister/binding_of_caller that abstracts away the logic of crawling through the frame bindings in the debug inspector to find the binding of the caller, but I feel like this functionality would be useful in Ruby.

With the binding_of_caller gem, you can hack together a kargs method like so:

```
def kargs
  method(caller_locations(1,1)[0].label).parameters.map do |(_type, name)|
    [name, binding.of_caller(2).local_variable_get(name)]
  end.to_h
end
```

This gets the name of the calling method, pulls the local variables from the method, retrieves the binding, and then retrieves the variables from the binding. By exposing a simpler API to retrieve the caller binding, a kargs method could be added to Ruby fairly easily I would think.

### History

#### #1 - 08/30/2018 07:44 PM - shevegen (Robert A. Heiler)

I have had a somewhat related idea in that I wanted a way to access all parameters in a simple, short manner. A bit similar to $1 $2 etc... for regexes, just that we could do so programmatically for the arguments passed into a method or a block. I was unable to come up with a short, succinct and elegant way for this though so I gave up on the idea - but I think introspection is great, so +1 for your idea (the idea itself; not sure if kargs is a good name... won't people confuse that with kwargs?).

#### #2 - 08/30/2018 08:11 PM - bherms (Bradley Herman)

shevegen (Robert A. Heiler) wrote:

> I have had a somewhat related idea in that I wanted a way to access all parameters in a simple, short manner. A bit similar to $1 $2 etc... for regexes, just that we could do so programmatically for the arguments passed into a method or a block. I was unable to come up with a short, succinct and elegant way for this though so I gave up on the idea - but I think introspection is great, so +1 for your idea (the idea itself; not sure if kargs is a good name... won't people confuse that with kwargs?).

Tbh I hadn't thought too much about naming... Was just trying to get something mocked up quickly. Either way, glad to see someone else has desired something similar.

#### #3 - 09/02/2018 06:11 AM - duerst (Martin Dürst)

*- Status changed from Open to Feedback*

Maybe I'm totally confused, but what's wrong with using **keyword_arguments?

#### #4 - 09/02/2018 07:10 AM - zverok (Victor Shepelev)

duerst (Martin Dürst)

> Maybe I'm totally confused, but what's wrong with using **keyword_arguments?

I am not the ticket's authors, but I understand the demand and met with the same requirement in some real-life projects. The most expressive example would be something like this:

```
def get(path:, accept: :json, headers: {}, **options)
  _request(method: :get, __all the rest of what have passed to this method___)
end

def post(path:, body:, accept: :json, headers: {}, **options)
  _request(method: :post, __all the rest of what have passed to this method___)
end
# ...and so on
```

Two of currently available options:

1. Accept just **arguments, and make checking what was mandatory, what should have default value and so on manually (also making auto-generated docs less expressive)
2. Accept everything as in my example, and then just do

```
_request(method: :get, path: path, body: body, accept: accept, headers: headers, **options)
```

...that looks not DRY at all.

So...

#### #5 - 10/15/2018 05:36 PM - bherms (Bradley Herman)

duerst (Martin Dürst) wrote:

> Maybe I'm totally confused, but what's wrong with using **keyword_arguments?

I agree with zverok's answer.

The main reason for not just doing **args is that you lose several of the benefits of using keyword arguments in the first place: 1) lowering connascence , 2) enforcing requirements (required kargs), 3) settings defaults.

```
def post(required:, other_required:, defaulted: 'default')
end

def post(**kargs)
  raise ArgumentError, 'missing keyword "required"' unless kargs.has_key?(:required)
  raise ArgumentError, 'missing keyword "other_required"' unless kargs.has_key?(:other_required)
  kargs.reverse_merge!(defaulted: 'default')
end
```

Those two method definitions are *very* different IMO and the first is far superior.  Obviously we could DRY up the required argument checks, but it still wouldn't be ideal.