

## Ruby master - Bug #12022

### Inconsistent behavior with splatted named arguments

01/26/2016 06:29 AM - justcolin (Colin Fulton)

<b>Status:</b> Closed	
<b>Priority:</b> Normal	
<b>Assignee:</b>	
<b>Target version:</b>	
<b>ruby -v:</b> 2.2.2 and 2.3.0	<b>Backport:</b> 2.0.0: UNKNOWN, 2.1: UNKNOWN, 2.2: UNKNOWN, 2.3: UNKNOWN

#### Description

### The Bug

When an empty hash is splatted (using \*\*) into a call of a method with no parameters, the empty hash is passed in as a positional argument *instead* of doing nothing. This causes an `ArgumentError`, which is confusing because when you splat an empty array into a method that doesn't accept any arguments the method is called without raising an error.

Similarly, if you splat a hash into a method that only has positional arguments, the method is called with the hash added as the last argument. This either causes an `ArgumentError` or unexpected bugs.

### Examples

(tested in MRI 2.2.2 and 2.3.0)

```
def without_parameters
  # some code
end

def with_parameters(*args)
  args
end

def with_one_parameter(arg)
  arg
end

empty_hash = {}
filled_hash = { example: "value" }
array = []

without_parameters(*array)
# calls the method without an error because `array' is empty

without_parameters(**empty_hash)
# unexpectedly raises an ArgumentError despite `empty_hash' being empty

with_parameters(**empty_hash)
# unexpectedly returns [{}] instead of []

with_parameters(**filled_hash)
# unexpectedly returns [{ example: "value" }] instead of raising an ArgumentError

with_one_parameter(**empty_hash)
with_one_parameter(**filled_hash)
# both unexpectedly do not raise an ArgumentError
```

### Further Information

This behavior makes it more difficult to do things like write specialized decorator classes using `#method_missing`. The following example does not work if the method being called does not have any named parameters. The variable `named_args` gets passed in

as a positional argument, causing `ArgumentErrors` or unexpected bugs:

```
class TrivialDecoratorExample
  def initialize(value)
    @value = value
  end

  def method_missing(name, *args, **named_args, &block)
    @value.send(name, *args, **named_args, &block)
  end
end
```

Instead one has to write something really ugly like:

```
def method_missing(name, *args, **named_args, &block)
  if @value.method(name)
    .parameters
    .any? { |type, _| [:keyreq, :key].include?(type) }

    @value.send(name, *args, **named_args, &block)
  elsif named_args.empty?
    @value.send(name, *args, &block)
  else
    raise ArgumentError.new
  end
end
```

#### Related issues:

Related to Ruby master - Feature #14183: "Real" keyword argument

Closed

#### History

##### #1 - 01/26/2016 02:39 PM - justcolin (Colin Fulton)

- `ruby -v` set to 2.2.2 and 2.3.0

##### #2 - 02/13/2016 05:13 PM - justcolin (Colin Fulton)

More details about why errors are not raised in some cases, and wrong errors are raised in other cases:

```
def example(required_param, **optional_named)
  p required_param
end

# Works as expected.
example(42) # => 42

# Should raise an error because no positional argument is being passed in,
# instead it passes in an option hash to required_param.
example(named_arg: 43)

def example2(required_param, second_required_param, **optional_named)
  # Some code...
end

# Should say it is missing two arguments, instead says it is missing one:
example2(named: 42)
```

I understand that all of this is probably related to the option hash syntax leftover from when Ruby didn't have named arguments, but this all leads to very confusing code. Named arguments are much more powerful, easier to deal with, and more "Ruby-like" than the option hash syntax. As such I feel that Ruby should assume that if you use the `**` operator in your parameters you never want that method to use an options hash.

In the distance future I even think that option hashes should be deprecated then removed, since named arguments do everything option hashes do and more. What opinions do you all have?

##### #3 - 02/14/2016 03:55 AM - sawa (Tsuyoshi Sawada)

This is a duplicate, or is at least related to [#11860](#).

##### #4 - 02/14/2016 04:46 PM - justcolin (Colin Fulton)

Sorry, I didn't see that issue. The cause is probably the same, but the difference is that [#11860](#) deals with using the double splat in the **argument** list

of a method call, whereas this issue deals using with the double splat in the **parameter** list of a method definition. That being said, these two issues can probably be merged.

Thanks for catching that!

#### #5 - 03/10/2016 12:07 AM - justcolin (Colin Fulton)

**NOTE:** I did find a "cleaner" way to do the decorator mentioned in the **Further Information** section, but it uses Kernel#eval which never feels like a good idea:

```
def method_missing *ordered, **named, &block
  args = ordered.map.with_index { |_, index| "ordered[#{index}]" } +
    named.map { |key, _| "#{key}: named[#{key}]" }

  eval "@value.#{name}(#{args.join(', ')})"
end
```

#### #6 - 06/27/2019 10:12 PM - jeremyevans0 (Jeremy Evans)

- Related to Feature #14183: "Real" keyword argument added

#### #7 - 09/02/2019 04:12 AM - jeremyevans0 (Jeremy Evans)

- Status changed from Open to Closed

With the master branch, you now get:

```
without_parameters(*array)
# => nil

without_parameters(**empty_hash)
# => nil

with_parameters(**empty_hash)
# => []

# This doesn't raise ArgumentError, because the method does not accept keyword
# arguments, so a keyword splat is passed as a positional hash.
with_parameters(**filled_hash)
# => [{:example=>"value"}]

with_one_parameter(**empty_hash)
# ArgumentError (wrong number of arguments (given 0, expected 1))

# This also doesn't raise ArgumentError, for the same reason as with_parameters
with_one_parameter(**filled_hash)
# => {:example=>"value"}
```

There are no warnings in this code, and the behavior should be the same in Ruby 3.