Ruby master - Bug #10708

In a function call, double splat of an empty hash still calls the function with an argument

01/07/2015 03:49 PM - Gondolin (Damien Robert)

Status:	Closed		
Priority:	Normal		
Assignee:	matz (Yukihiro Matsumoto)		
Target version:			
ruby -v:	ruby 2.2.0p0 (2014-12-25 revision 49005) [x86_64-linux]	**Backport:**	2.0.0: UNKNOWN, 2.1: UNKNOWN, 2.2: UNKNOWN

Description

Consider this:

```
def foo; end
foo(*[]) #Splatting an empty list is ok
foo(**{}) #Double splatting an empty hash is like calling foo({}) which gives an error
```

This is annoying in a function that is a wrapper around another function and just process some keywords:

```
def wrapper(*args, keyword: true, **others)
  puts keyword
  wrappee(*args,**others) #here this code will fail if others is empty
end
```

Related issues:

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

History

#1 - 01/07/2015 08:43 PM - sawa (Tsuyoshi Sawada)

Do you mean **others?

#2 - 01/07/2015 09:07 PM - Gondolin (Damien Robert)

Tsuyoshi Sawada wrote:

> Do you mean **others?

Yes, sorry for the typo

#3 - 01/07/2015 09:09 PM - Gondolin (Damien Robert)

I think I got bitten by markdown's syntax actually, all my * * got replaced, so s/others/**others/g

#4 - 01/08/2015 08:21 PM - dunric (David Unric)

By my subjective opinion I don't find this a bug but a feature.

> Consider this:
>
> ```
> def foo; end
> foo(*[]) #Splatting an empty list is ok
> foo(**{}) #Double splatting an empty hash is like calling foo({}) which gives an error
> ```

Here you define a method without a (keyword) argument placeholder so it does not expect a Hash argument, which is effectively used to pass keyword arguments at method call, even when keyword arguments are empty. Note calling foo(kwarg: 'value') is just a syntactic sugar to foo({kwarg: 'value'}).
Methods unlike procs do check arguments count, so ArgumentError exception is the correct behavior.

This is an expected difference to expanding an Array, which results in *plain arguments list* which can result in *"nothing"*, ie. an empty list. foo(*[]) is interpreted as foo() so you won't get an error.

You probably assume expanding a Hash, ie. applying double-splat operator on empty Hash instance at method call would behave the same way. However implementing such a special corner-case behavior would only introduce inconsistency in the language or would require implementing a *quite*

new different type of plain list like *"plain keyword arguments list"*.

#5 - 01/08/2015 09:11 PM - kkube (Kolja Kube)

Just to inform everyone, this issue stems from this post on Stack Overflow.

Also, I have now idea how the ruby parser works, so if increased parsing complexity is the reason for the discussed behavior, I'm happy to concede my point.

Why I think the current behavior is weird, take these forwarders:

```
def call_args(method, *args); send(method, *args); end
def call_kwargs(method, **opts); send(method, **opts); end
```

This obviously works:

```
def one_arg(arg); end
def one_kwarg(opt:); end

call_args(:one_arg, 0) # fine
call_kwargs(:one_kwargs, opt: 0) # also fine
```

But here the behavior differs:

```
def zero_args(); end
def zero_kwargs(); end

call_args(:zero_args) # fine
call_kwargs(:zero_kwargs) # error
```

Now, this is not a problem per se, since the single-splat syntax also permits fowarding keyword arguments. But this simply tripped me up, and since one of Ruby's principles is the principle of least surprise, I look forward to hearing your opinions.

#6 - 01/09/2015 12:57 AM - dunric (David Unric)

If I am not mistaken, even latest Ruby 2.2 selects keyword arguments as the last method's argument and of Hash type.

Let's imagine an example where both simple and keyword optional arguments are used:

```
def call_multiargs(method, *aopts, **kwopts); send(method, *aopts, **kwopts); end
# kwopts can be passed to send method without double-splat operator as it does _nothing_ here

def args_and_kwargs(*args, **kwargs); p args; p kwargs; end

call_multiargs(:args_and_kwargs, **{a: 1, b:2})
# How should Ruby expand the hash ?
#   - as (:a, 1, :b, 2) list so kwopts would be empty {} ?
#   - as (:a, 1) and {b: 2} so kwopts would be {b: 2} ?
#   - as {a: 1, b:2} so aopts would be empty [] ?
#   - as ([:a, 1], [:b, 2]) list and kwopts would be empty {} ?
#   - as ([:a, 1]) and {:b, 2} ?
#   etc
```

Because Ruby has no special keyword list type like Python has and for keyword arguments a single Hash instance is used, it is fundamentally not possible to do an expansion of **{…} into a list.
Again, in Ruby there does not exist a list of type :a => 1, :b => 2. What you see in a method call is a syntactic sugar for {:a => 1, :b => 2}, ie. optional braces.
To keep consistency there can't exist an exception to this rule for empty hashes.

To sum it up, use of double-splat operator for hash expansion is wrong and makes no sense.

p.s. As far as I know, there are only two cases and only as a parser syntax helpers for Hash and Array constructors, quite unrelated to some list expansion:
{**{:a => 1, :b => 2}} - enclosed hash items used for implicit form
[*{:a => 1, :b => 2}] - enclosed hash converted with Hash#to_a and used for implicit form

#7 - 01/09/2015 08:03 AM - nobu (Nobuyoshi Nakada)

- Description updated

- Category set to syntax

- Status changed from Open to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

Although *args includes and passes keywords too, but seems you want to add/remove/change some of keyword arguments.
It sounds reasonable to me.

#8 - 01/12/2015 12:44 PM - Gondolin (Damien Robert)

@Kolja: I wasn't aware of your post on stackoverflow when I posted this bug report, but this is indeed a nice coincidence!
For context I sometime want to apply some methods from a module without including the module, so I have a function 'apply' that takes an unbound method, bind it to an object and send the arguments to this method. Since the object to bind to and the method itself are passed as keywords to 'apply', I can't use

```
method.call(*args,&block)
```

I need to call

```
method.call(*args,**opts,&block)
```

where I stumbled upon the above bug when opts is empty.

david (david he): more precisely ruby select keywords as symbols keys of the last argument when it is of Hash type.
When you call a function, using keyword like arguments is a syntaxtic sugar for passing a hash of symbols as the last argument.
Now consider this:

```
amethod({keyword: true})
```

is indeed the same as

```
amethod(keyword: true)
```

But

```
amethod({keyword1: true}, keyword2: true) #one argument and one keyword
```

is not the same as

```
amethod(keyword1: true, keyword2: true) #two keywords
```

And

```
amethod(**{keyword1: true}, keyword2: true) #two keywords
```

do indeed gives only keywords in ruby already.

So that's why **{} should not be the same as {} but instead expand into 'nothing'.

@Nobuyoshi: thanks for the consideration!

#9 - 03/08/2016 02:26 AM - ozydingo (Andrew Schwartz)

Adding to this, the current behavior results in the following inconsistent behavior: I can call an argless method using a double-splatted empty Hash directly, but this cannot be done via a delegating or overriding method. I'm encountering this an an issue with subclasses that override argless parent methods with its own definition that accepts kwargs, and I would argue that the difference in behavior when calling bar directly vs calling it via foo is very surprising.

```
def foo(*args, **kwargs)
  p args
  p kwargs
  bar(*args, **kwargs)
end

def bar
  puts "yay"
end

2.2.2 > bar(*[], **{})
yay
2.2.2 > foo
[]
{}
ArgumentError: wrong number of arguments (1 for 0)
    from (irb):13:in `bar'
    from (irb):23:in `foo'
    from (irb):25
```

#10 - 03/08/2016 04:32 AM - sawa (Tsuyoshi Sawada)

The inconsistency is even more serious. See #11860.

#11 - 03/08/2016 04:59 AM - justcolin (Colin Fulton)

#12022 has some further exploration of this bug.

#12 - 05/18/2016 01:07 AM - shyouhei (Shyouhei Urabe)

Matz is positive about #12157 (removal of optional hash parameters).

If that request is to be accepted, this double-splat problem should be solved beforehand. The bug here sources from hash / kwargs confusion so a clear distinction between them is mandatory.

#13 - 06/27/2019 10:12 PM - jeremyevans0 (Jeremy Evans)

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

#14 - 09/02/2019 04:18 AM - jeremyevans0 (Jeremy Evans)

- Status changed from Assigned to Closed

With the acceptance of #14183, double splatting an empty hash when calling a method no longer passes an empty positional hash to the method.