Ruby master - Feature #7299

Ruby should not completely ignore blocks.

11/07/2012 01:06 PM - marcandre (Marc-Andre Lafortune)

Status:	Rejected	
Priority:	Normal	
Assignee:		
Target version:		

Description

Ruby should not completely ignore blocks.

const_set :Example, Class.new do
p "Hello, world"
end
Doesn't print anything, generate any warning nor error.

To minimize any impact, Ruby should issue a warning, and in future version could even raise an error.

Even unused variables provide warnings in verbose mode, and they have their use.

I can't think of a case where passing a block to a builtin method that doesn't accept a block is not a programming error though.

If this is approved, I volunteer to implement this.

History

#1 - 11/07/2012 01:42 PM - matz (Yukihiro Matsumoto)

- Status changed from Open to Feedback

I considered this issue before, and had problem with how to detect non block calling block.
Things go easier if & block argument is mandatory for block taking methods, but I am not doing so in near future.
Do you have any good idea?

Matz.

#2 - 11/07/2012 02:23 PM - marcandre (Marc-Andre Lafortune)

- Status changed from Feedback to Open

matz (Yukihiro Matsumoto) wrote:

> I ... had problem with how to detect non block calling block.

Sorry, I am not sure I understand completely.

To avoid case like the example I gave, we could modify "rb_mod_const_set" by adding "WARN_IF_BLOCK_GIVEN", for example, or create a more advanced version of "rb_check_arity" like "rb_check_arity_and_block" that accepts a parameter to warn if there is a block given.

What I would *really* like to do is create an improved "rb_define_method" with arguments to specify:

- mininum arity
- maximum arity
- if block is accepted
- and ideally parameter names, at least for methods with simple interfaces

This way:

- easier to warn if block is accepted
- improved result for "".method(:gsub).arity
- much improved result for "".method(:gsub).parameters
- possible to implement arity_max or arity_range with good result if ever it is accepted
- improved behavior when using curry

Also, this would simplify many method that would not have to call rb_check_arity like we do now.

#3 - 11/07/2012 09:48 PM - matz (Yukihiro Matsumoto)

- Status changed from Open to Rejected

So you think of changing introducing new functions. I see.
In that case, it's better to submit a new issue for the idea, with API proposal.

Matz.

#4 - 11/07/2012 10:23 PM - Anonymous

On Wed, Nov 07, 2012 at 01:06:34PM +0900, marcandre (Marc-Andre Lafortune) wrote:

> Issue #7299 has been reported by marcandre (Marc-Andre Lafortune).
>
> ---
>
> Feature #7299: Ruby should not completely ignore blocks.
> https://bugs.ruby-lang.org/issues/7299
>
> Author: marcandre (Marc-Andre Lafortune)
> Status: Open
> Priority: Normal
> Assignee:
> Category: core
> Target version:
>
> Ruby should not completely ignore blocks.
>
> const_set :Example, Class.new do
> p "Hello, world"
> end
> # Doesn't print anything, generate any warning nor error.
>
> To minimize any impact, Ruby should issue a warning, and in future version could even raise an error.
>
> Even unused variables provide warnings in verbose mode, and they have their use.
>
> I can't think of a case where passing a block to a builtin method that doesn't accept a block is not a programming error though.

This happens with normal ruby code:

ruby -w -e'def foo; 10; end; p foo { raise };'

Why would "builtin" methods be special?

--
Aaron Patterson
http://tenderlovemaking.com/

#5 - 11/07/2012 10:23 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Em 07-11-2012 11:00, Aaron Patterson escreveu:

> On Wed, Nov 07, 2012 at 01:06:34PM +0900, marcandre (Marc-Andre Lafortune) wrote:
>
>> Issue #7299 has been reported by marcandre (Marc-Andre Lafortune).
>>
>> ---
>>
>> Feature #7299: Ruby should not completely ignore blocks.
>> https://bugs.ruby-lang.org/issues/7299
>>
>> Author: marcandre (Marc-Andre Lafortune)
>> Status: Open
>> Priority: Normal
>> Assignee:
>> Category: core
>> Target version:
>>
>> Ruby should not completely ignore blocks.
>>
>> const_set :Example, Class.new do
>> p "Hello, world"
>> end
>> # Doesn't print anything, generate any warning nor error.

> > To minimize any impact, Ruby should issue a warning, and in future version could even raise an error.
> >
> > Even unused variables provide warnings in verbose mode, and they have their use.
> >
> > I can't think of a case where passing a block to a builtin method that doesn't accept a block is not a programming error though.
> > This happens with normal ruby code:
>
> ruby -w -e'def foo; 10; end; p foo { raise };'
>
> Why would "builtin" methods be special?

I agree. I'd prefer to just raise an exception when a block is passed
and no block_given? or yield is called for the method.

I was bitten some times by bugs hard to detect because of this behavior
where I thought the block was being given to a method while it was being
given to another one that didn't even expect any block.

#6 - 11/08/2012 12:10 AM - marcandre (Marc-Andre Lafortune)

matz (Yukihiro Matsumoto) wrote:

> So you think of changing introducing new functions. I see.
> In that case, it's better to submit a new issue for the idea, with API proposal.

Of course, but first I wanted to validate you were positive with the idea of warning for unused block

tenderlove wrote:

> This happens with normal ruby code:
> Why would "builtin" methods be special?

Agreed, it would be best if user methods also warned. I was just lacking ambition by suggesting it only for builtin methods :-)

Rodrigo's suggestion of flagging block_given?, yield, (as well as Proc.new, super and &capture_block) would work.

#7 - 11/08/2012 05:48 AM - matz (Yukihiro Matsumoto)

I am positive as long as there's rational way to declare methods as 'non-block taking'.
Last time I tried, I couldn't think any good idea to do so (without adding new API).
New API (alternative version of rb_define_method, I suppose) is a good idea.
The remaining problem should be how to declare Ruby-define methods to be 'non-block taking'.
Under the current language spec, absence of '& argument' may or may not mean the method would take a block.

Matz.

#8 - 11/08/2012 08:29 AM - ko1 (Koichi Sasada)

(2012/11/08 5:48), matz (Yukihiro Matsumoto) wrote:

> Under the current language spec, absence of '& argument' may or may not mean the method would take a block.

I agree that such checking is very useful.

example code
p 'str'.gsub('x') do

end

One idea:

If compiled method does not contain

- `yield' statement
- super statement
- block argument then the method is marked as "block is not needed" method.

This approach introduce incompatibility because we can call block in `eval'.

example

def foo str
eval str
end

foo('yield') do
...
end

And maybe there are other issues.

--
// SASADA Koichi at atdot dot net

#9 - 11/08/2012 08:53 AM - shyouhei (Shyouhei Urabe)

On 11/07/2012 03:23 PM, SASADA Koichi wrote:

> This approach introduce incompatibility because we can call block in `eval'.

So you are proposing to deprecate eval? :p

#10 - 11/09/2012 01:53 AM - marcandre (Marc-Andre Lafortune)

ko1 (Koichi Sasada) wrote:

> If compiled method does not contain
>
> - `yield' statement
> - super statement
> - block argument then the method is marked as "block is not needed" method.
>
> This approach introduce incompatibility because we can call block in `eval'.
>
> And maybe there are other issues.

There is also Proc.new...

```
def foo
  Proc.new.call
end

foo{ p 42 } # => prints 42
```

Both eval and Proc.new are strange cornercases though. The only valid uses I can think of for Proc.new also imply use of super. If we only issue a warning, the incompatibility would be very minimal.

So I'll make a proposal for an expanded API for rb_define_method. OTOH, marking ruby methods as {non-}block taking would seriously challenge my cruby skills

#11 - 11/11/2012 02:23 PM - nobu (Nobuyoshi Nakada)

(12/11/09 1:53), marcandre (Marc-Andre Lafortune) wrote:

> ko1 (Koichi Sasada) wrote:
>
>> If compiled method does not contain
>>
>> - `yield' statement
>> - super statement
>> - block argument then the method is marked as "block is not needed" method.
>
> There is also Proc.new...

Also proc, lambda, and defined?(yield).

--
Nobu Nakada

#12 - 11/25/2012 02:03 AM - headius (Charles Nutter)

Perhaps methods that want to ensure nobody accidentally passes in a block should just check for it? fail if block_given? for example?

An option for a syntactic check in Ruby code: def foo(&nil) => raise error on call if a block is given.

I don't think magic checks should be put in place for Ruby code that doesn't have an explicit block parameter. There are a lot of edge cases where the block is used and we don't know about it until later. eval/binding has been brought up, Proc.new (which should be deprecated) has been brought up, and so on.