Ruby master - Bug #18780
Incorrect binding receiver for C API rb_eval_string()
05/13/2022 11:00 PM - daveola (David Stellar)

Status: Closed
Priority: Normal
Assignee: 
Target version: 
ruby -v: ruby 2.7.0p0 (2019-12-25 revision 647ee6f091) [x86_64-linux] 
Backport: 2.7: UNKNOWN, 3.0: UNKNOWN, 3.1: UNKNOWN

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
</table>
| % ruby -v 
ruby 2.7.0p0 (2019-12-25 revision 647ee6f091) [x86_64-linux] 
(Though looking at the source code this problem still exists in ruby 3.0) 
The rb_eval_string() is seemingly capable of everything that eval is capable of, with one slight deviation. The binding is oddly setup to be correct except for the receiver/self. 
This means that while these both act the same: 
ruby: eval("puts someLocalVariable") 
C API: rb_eval_string("puts someLocalVariable") 
These do not: 
ruby: eval("puts @someInstanceVar") 
C API: rb_eval_string("puts @someInstanceVar")    # nil 
And this is because these do not work: 
ruby: eval("puts self")                          # self of calling context 
ruby: eval("puts binding().receiver")            # self of calling context 
C API: rb_eval_string("puts self")                  # main 
C API: rb_eval_string("puts binding().receiver")  # main 
We can see the problem in the MRI source in ruby_eval_string_from_file() which has:

```c
return eval_string_with_cref(rb_vm_top_self(), rb_str_new2(str), NULL, file, 1); 
```
We've passed in rb_vm_top_self instead of the correct 'self'

Although possibly the issue is in the API itself with the fact that there's no way to plumb through the receiver that you are given in the C extension method function call, i.e.:

```c
// My C extension that calls eval and knows what it's 'self' is. 
VALUE method_myCMethod(int argc, VALUE *argv, VALUE self) { 
  rb_eval_string("..."); // <- no way to be given self? 
}
```

Having said that, rb_eval_string is able to determine the vast majority of the binding context, since it correctly builds the binding() object except that the receiver is set to main, so perhaps this is something that can be determined. It is something that the builtin eval is able to do, after all. So possibly this is just a failure with MRI. I don't have other rubies to test. (I'm on ruby 2.7.0 but the source relevant to this hasn't changed.)

I would argue this is a bug, because we are essentially given a corrupted result from binding() - one where we may have access to the local variables of an instance method, but one where the self itself is not properly set. That's not an actual legit binding state in the ruby code.

<table>
<thead>
<tr>
<th>Related issues:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related to Ruby master - Bug #18487: Kernel#binding behaves differently depen...</td>
</tr>
<tr>
<td>Related to Ruby master - Feature #15778: Expose an API to pry-open the stack ...</td>
</tr>
</tbody>
</table>
rb_eval_string() is used inside C code/C extensions, there is no "caller Binding" to use, C extension frames have no Binding. If you want to eval with a specific Binding, capture it (e.g., with Kernel#binding in Ruby code), then pass it to the C extension and call Binding#eval(code) (e.g. with rb_funcall).

Or simply call Binding#eval from Ruby code and avoid C altogether.

That's actually not really true, there is a caller binding, as evidenced by the fact that you can get the binding inside of the eval.

So it sets up the binding() properly, except that binding().receiver is incorrect and it effectively gives you a state that does not actually exist (the local variables are accessible, but the instance variables are not.).

The entire point of this is to come up with a C api that acts like Kernel::eval but with some modifications, so the answer of "just supply binding() to the method" is not really working.

It's the same as TOLEVEL_BINDING, isn't it?
It doesn't use any information from the caller, which is a C extension, which has binding, hence "no caller binding".

so the answer of "just supply binding() to the method" is not really working.

It is, use e.g. rb_funcall(my_binding, rb_intern("eval"), rb_str_new(code)).
And my_binding comes from an argument to that method defined in the C extension.

No, it's not. It's the same as the caller's binding, except that the receiver/self is set to TOLEVEL.

In other words:

RUBY CODE

class Foo
  def bar
    @instanceVar = 42
    localVar = 42
    <call my C method here>
  end
end

@instanceVar = "main"
localVar = "main"
Foo.new().bar() # Create a foo and call bar which calls my C api

C CODE

  // inside the C api method:
  VALUE bind = evalReq("binding()");

This binding will be pointing to localVar = 42 but @instanceVar = "main"

How you can justify that would be interesting.
OK, finally the issue is clear.

Could you try with latest master?
With #18487 I believe you should get a RuntimeError if you call binding from a C frame/method.

That also makes it clear one should not try to get a Binding from a C frame, it must be captured from a Ruby frame and then everything is consistent.

---

#7 - 05/17/2022 05:35 PM - Eregon (Benoit Daloze)

https://gist.github.com/eregon/31259907c132ab61e9f136d684b16c39 is an easy way to have a single-file repro with both C code and Ruby code, it'd be nice if you can provide a complete repro as well.

---

#8 - 05/17/2022 05:47 PM - daveola (David Stellar)

Eregon (Benoit Daloze) wrote in #note-6:

OK, finally the issue is clear.

Could you try with latest master?
With #18487 I believe you should get a RuntimeError if you call binding from a C frame/method.

That also makes it clear one should not try to get a Binding from a C frame, it must be captured from a Ruby frame and then everything is consistent.

This is 100% not what I was hoping for.

1. That seems like it will break a number of things, if your binding is no longer correct. What if you try to eval some code that ends up needing to use the binding?
2. This is inconsistent with Kernel::eval, which, while written in C, is able to get the binding of the caller.

Let's say I need to write a myEval method which is like eval, but first prints out the string that is eval'd. Or possibly checks it for certain information. You have just made this literally impossible to write in ruby.

---

#9 - 05/17/2022 05:50 PM - daveola (David Stellar)

You have just made this literally impossible to write in ruby.

And before you say "just have the caller pass in the binding" - keep in mind that Kernel::eval does not require this. So you would not, for example, be able to make code that would wrap or replace eval, but still have proper bindings and be able to access local variables, instance variables, scope info, etc.

```ruby
foobar = 42
eval("foobar += 1")  # Works
myEval("foobar += 1") # Impossible
```

---

#10 - 06/11/2022 12:56 PM - Eregon (Benoit Daloze)

- Status changed from Open to Rejected

I'm reluctant to reply since you seem to ignore my comments as a Ruby implementer.

Anyway, you are asking for something to get the caller's binding, that is very much on purpose not provided by Ruby. Such a functionality breaks encapsulation in a very bad manner.
The fact calling Kernel#binding from a C extension did something like that was accidental, and has been fixed (#18487).

There have been discussions in the past and such functionality to get the caller binding should only be made available for debugging situations, in which you can use the debug_inspector C API or https://github.com/banister/debug_inspector.
That is of course slow and should only be used for debugging situations, but maybe it makes sense to use it conditionally in your case based on some configuration/option (and otherwise just alias myEval eval).

It's a fair point this makes it impossible to wrap eval and still get the same binding (without the debug_inspector C API), but Ruby will not add access to the caller binding just for that.
Eregon (Benoit Daloze) wrote in #note-10:

I'm reluctant to reply since you seem to ignore my comments as a Ruby implementer.

I'm sorry you feel that way, and I regret posting to the truffleruby discussion by mistake, I lost track of the many threads that are talking about this.

Anyway, you are asking for something to get the caller's binding, that is very much on purpose not provided by Ruby.

This is being discussed I now see on:
https://bugs.ruby-lang.org/issues/15778

You, in fact, said that you thought it was a good idea (3 years ago) and I hope that is still the case, because this mechanism would solve my problem.

I understand that one almost never wants to break encapsulation (except in debug), but there are cases, and eval() itself does break encapsulation and is part of ruby.

That is of course slow and should only be used for debugging situations, but maybe it makes sense to use it conditionally in your case

I hear the suggestion and I will just have to switch to using a debugger for my code to work, it's not a conditional, I need the ability to write a different version of eval() for my code, and I fought hard at my company to use ruby, so it's a shame that it has to be done with a debug hack and all of it's performance penalties, but I hear that you are not willing to add this type of a pathway.

Hopefully 15778 will get resolved some day and then none of this will be an issue anymore for me. :)

#12 - 06/12/2022 11:26 AM - Eregon (Benoit Daloze)
- Related to Feature #15778: Expose an API to pry-open the stack frames in Ruby added

#13 - 06/12/2022 11:35 AM - Eregon (Benoit Daloze)
daveola (David Stellar) wrote in #note-11:

I hear the suggestion and I will just have to switch to using a debugger for my code to work, it's not a conditional, I need the ability to write a different version of eval() for my code, and I fought hard at my company to use ruby, so it's a shame that it has to be done with a debug hack and all of it's performance penalties, but I hear that you are not willing to add this type of a pathway.

Note that no debugger is needed for the debug_inspector gem, it's a regular gem/dependency.
That C API that the gem uses is meant to only be used for debugging situations, but there is no actual check for it, it relies on people using it only for valid situations.
So you should be able to use it for your use-case without issues.

Hopefully 15778 will get resolved some day and then none of this will be an issue anymore for me. :)

My understanding of that issue's discussion so far is that is unlikely.
I would personally be happy to have a Ruby API for this as proposed there, but I understand concerns of other CRuby committers to not make it too easy to use such a dangerous/breaking-encapsulation API, and so I guess it will remain a C API only (+ the gem) for a while.

#14 - 06/14/2022 09:12 PM - alanwu (Alan Wu)
Thanks for the bug report. The "binding with mixed information from two contexts" situation from https://bugs.ruby-lang.org/issues/18780#note-4 definitely looks wrong. I wrote a reproducer and luckily, it's no longer an issue on the master branch (ab10f111c3).

Let's say I need to write a myEval method which is like eval, but first prints out the string that is eval'd. ... You have just made this literally impossible to write in ruby.

This is in fact possible with the rb_binding_new() API on both 2.7.6 and on the master branch. Here is a demo script:

```ruby
puts RUBY_DESCRIPTION
require 'tmpdir'
require 'rbconfig'
```

# From https://gist.github.com/eregon/31259907c132ab61e9f136d684b1fc39
# courtesy of Benoit.

08/31/2022
def inline_c_extension(c_code)
    Dir.mktmpdir('inline_c_extension') do |dir|
        File.write("#{dir}/cext.c", c_code)
        File.write("#{dir}/extconf.rb", <<RUBY
            require 'mkmf'
            create_makefile('cext')
        RUBY
        out = IO.popen([RbConfig.ruby, 'extconf.rb'], chdir: dir, &read)
        raise "ruby extconf.rb failed: ${?inspect}\n#{out}" unless $??.success?
        out = IO.popen(['make'], chdir: dir, &read)
        raise "make failed: ${?inspect}\n#{out}" unless $??.success?
    end
end

inline_c_extension <<-C
#include "ruby.h"

// A wrapped version of Kernel#eval that prints the program before evaluating.
// Uses the caller's Ruby context just like Kernel#eval.
static VALUE
my_eval(VALUE self, VALUE program)
{
    ID p_id = rb_intern("p");
    ID eval_id = rb_intern("eval");
    VALUE binding;
    if (1 /* do what I understand OP wants */) {
        binding = rb_binding_new();
    } else {
        // Flip the if to check the "impossible binding" issue from [ruby-core:108597]
        binding = rb_eval_string("p [rbeval_string, @foo, foo]; binding");
    }
    rb_funcall(self, p_id, 1, program);
    return rb_funcall(binding, eval_id, 1, program);
}

void
Init_cext(void)
{
    rb_define_global_function("my_eval", my_eval, 1);
}

C

@foo = :main
foo = :main1
my_eval("p [:in_main, self, @foo, foo]"

class Foo
    def foo
        @foo = :inclass
        foo = :inclass1
        my_eval("p [:inside_foo, self, @foo, foo]"
    end

    new.foo
end

$ ruby demo.rb
ruby 3.2.0dev (2022-06-14T16:06:06Z master ab10f111c3) [x86_64-darwin21]
"p [:in_main, self, @foo, foo]"
[<in_main, main, main, main], [:main]
"p [:inside_foo, self, @foo, foo]"
[<inside_foo, Foo:0x000000010fcc1360, :inclass, :inclass1]

In terms of [#18487], my_eval in the demo is not any more powerful than Kernel#binding and Kernel#eval since it's only getting the binding of the Ruby frame immediately below it.

By the way, thanks to some awesome work by @shyouhei (Shyouhei Urabe), newer versions of public C headers

08/31/2022
I have API documentations. I would recommend reading those docs instead of the implementation. For example for rb_eval_string():

```c
/**
 * Evaluates the given string in an isolated binding.
 *
 * Here "isolated" means that the binding does not inherit any other bindings. This behaves same as the binding for required libraries.
 */
```

I wonder if we host the HTML version of the C API docs anywhere...

---

#15 - 06/14/2022 09:19 PM - alanwu (Alan Wu)

> Status changed from Rejected to Open

I wrote a reproducer and luckily, it's no longer an issue on the master branch (ab10f111c3).

I spoke too soon! Flipping the if in demo.rb:

```ruby
ruby 3.2.0dev (2022-06-14T16:06:06Z master ab10f111c3) [x86_64-darwin21]
[:rbeval_string, :main, :mainl]
p [:in_main, self, @foo, foo]
[:in_main, main, :main, :mainl]
[:rbeval_string, :main, :inclassl] <<<< mixed context!
p [:inside_foo, self, @foo, foo]
[:inside_foo, main, :main, :inclassl]
```

I'll reopen this and look into it later.

---

#16 - 06/14/2022 10:52 PM - daveola (David Stellar)

alanwu (Alan Wu) wrote in #note-15:

> I spoke too soon! Flipping the if in demo.rb:
>
> I'll reopen this and look into it later.

Doesn't the change in [https://bugs.ruby-lang.org/issues/18487](https://bugs.ruby-lang.org/issues/18487) (raising a kernel error) imply that this doesn't need to be fixed?

Also, thanks for the pointer to rb_binding_new(), I'll take a look at that, but issue 18487 implies that it will likely be viewed as incorrect to return the caller's binding.

---

#17 - 06/15/2022 12:14 AM - alanwu (Alan Wu)

The issue with [18487](https://bugs.ruby-lang.org/issues/18487) was that Kernel#binding used to find the first Ruby frame on the stack and that this iterating behavior is brittle as it basically acts as an assertion that the frames that it skips over remain as non-Ruby frames in future versions. We changed it so that it doesn't iterate anymore and only checks the direct caller. It's easier to reason about since it's a weaker assertion and there is only one frame involved.

So now binding related APIs should only consider the direct caller. The weird rb_eval_string() scenario somewhat betrays this new semantics because when the frame for Kernel#binding is active the stack looks like this:

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Kernel#binding</td>
</tr>
<tr>
<td>Ruby</td>
<td>&lt;eval code frame&gt;</td>
</tr>
<tr>
<td>C</td>
<td>my_eval</td>
</tr>
<tr>
<td>Ruby</td>
<td>Foo#foo</td>
</tr>
</tbody>
</table>

It returns the local from Foo#foo, so not only does it skip over C frame, it also skips over a Ruby frame! Now, maybe we could understand Kernel#binding's behavior here as not skipping frames but rather returning a binding previously created at the time when we call rb_eval_string(), when the stack looked like:
But the way it idiosyncratically creates a binding by mixing contexts is still surprising. The behavior doesn't match what the docs for rb_eval_string() would suggest either since it's not an isolated binding.

Calling rb_binding_new() from a C method doesn't create a new frame, and it's distinct from calling Kernel#binding from a C method because of that. Using rb_binding_new() is somewhat like assuming the role of Kernel#binding. One can only call rb_binding_new() from a C method so making it raise in that context would render it completely useless -- not desirable!

#18 - 07/21/2022 12:09 PM - mame (Yusuke Endoh)
We discussed this issue at the dev meeting.

@matz (Yukihiro Matsumoto) said self should be changed. rb_eval_string should evaluate the argument under the current context. The self should be picked from the context instead of main. If there is no caller binding (which is a main use case for rb_eval_string, as @Eregon (Benoit Daloze) said), self should be main.

We will revisit the decision if compatibility issues are discovered.

#19 - 07/22/2022 12:24 AM - shyouhei (Shyouhei Urabe)
Updating the header document.

From b968f277386649b7531a8999be54eacf3c599c7d Mon Sep 17 00:00:00 2001
From: =?UTF-8?q?=E5=8D=9C=E9=83=A8=E6=98=8C=E5=B9=B3?=<shyouhei@ruby-lang.org>
Date: Thu, 21 Jul 2022 18:00:04 +0900
Subject: [PATCH] [DOC] update rb_eval_string() document.
---
include/ruby/internal/eval.h | 33 +++++++++++++++++++++++---
template/Doxyfile.tmpl | 1 +
2 files changed, 31 insertions(+), 3 deletions(-)
diff --git a/include/ruby/internal/eval.h b/include/ruby/internal/eval.h
index 34a53849da..b7ab66515f 100644
--- a/include/ruby/internal/eval.h
+++ b/include/ruby/internal/eval.h
@@ -28,10 +28,31 @@
 RBIMPL_SYMBOL_EXPORT_BEGIN()
 RBIMPL_ATTR_NONNULL(()
 /*
- * Evaluates the given string in an isolated binding.
- * Evaluates the given string.
- *
- * Here "isolated" means that the binding does not inherit any other
- * bindings. This behaves same as the binding for required libraries.
- * In case it is called from within a C-backended method, the evaluation is
- * done under the current binding. However there can be no method. On such
- * situation this function evaluates in an isolated binding, like `require`
- * runs in a separate one.
- *
- * `__FILE__` will be "(eval)"; and `__LINE__` starts from 1 in the
- * evaluation.
- *
@@ -39,6 +51,31 @@
 * @param[in] str Ruby code to evaluate.
 * @exception rb_eException Raises an exception on error.
 * @return The evaluated result.
+ *
+ * @shyouhei's old tale about the birth and growth of this function:
+ *
+ * At the beginning, there was no rb_eval_string(). @shyouhei heard that
+ * @shugo, author of Apache httpd's mod_ruby module, requested @matz for this
+ * API. He wanted a way so that mod_ruby can evaluate ruby scripts one by one,
+ * separately, in each different contexts. So this function was made. It was
+ * designed to be a global interpreter entry point like ruby_run_node().
+ *
The way it is implemented however allows extension libraries (not just programs like Apache httpd) to call this function. Because its name says nothing about the initial design, people started to think of it as an orthodox way to call ruby level `eval` method from their extension libraries. Even our `extension.rdoc` has had a description of this function basically according to this understanding.

The old (mod_ruby like) usage still works. But over time, usages of this function from extension libraries got popular, while mod_ruby faded out; is no longer maintained now. Devs decided to actively support both. This function now auto-detects how it is called, and switches how it works depending on it.

@see https://bugs.ruby-lang.org/issues/18780

```c
VALUE rb_eval_string(const char *str);
```

---

```
+ ALIASES += "shugo=@shugo"
```

# Set the OPTIMIZE_OUTPUT_FOR_C tag to YES if your project consists of C sources only. Doxygen will then generate output that is more tailored for C. For

--

2.17.1

---

#20 - 07/22/2022 10:00 AM - Eregon (Benoit Daloze)

Nice story :)

Probably a typo: s/old table/old tale/

#21 - 07/22/2022 12:10 PM - shyouhei (Shyouhei Urabe)

Yes, it is a typo. Thank you.

#22 - 08/24/2022 04:43 PM - jeremyevans0 (Jeremy Evans)

- Status changed from Open to Closed

Fixed by 5bbba76489628f4509495ebf4ba0a7aad4c0b560