Ruby master - Feature #15955

UnboundMethod#apply

06/24/2019 05:49 PM - nelhage (Nelson Elhage)

Status:	Closed	
Priority:	Normal	
Assignee:		
Target version:		

Description

I'd love a way to apply an UnboundMethod to a receiver and list of args without having to first bind it. I've ended up using UnboundMethods in some hot paths in my application due to our metaprogramming idioms, and the allocation from .bind is comparatively expensive.

I'd love unbound_method.apply(obj, args…) to be equivalent to unbound_method.bind(obj).call(args…) but without allocating the intermediate Method

Associated revisions

Revision 83c6a1ef - 08/30/2019 02:13 AM - mame (Yusuke Endoh)

proc.c: Add UnboundMethod#bind_call

umethod.bind_call(obj, ...) is semantically equivalent to
umethod.bind(obj).call(...). This idiom is used in some libraries to
call a method that is overridden. The added method does the same
without allocation of intermediate Method object. [Feature #15955]

```
class Foo
  def add_1(x)
    x + 1
  end
end
class Bar < Foo
  def add_1(x) # override
    x + 2
  end
end

obj = Bar.new
p obj.add_1(1) #=> 3
p Foo.instance_method(:add_1).bind(obj).call(1) #=> 2
p Foo.instance_method(:add_1).bind_call(obj, 1) #=> 2
```

Revision 09c940b1 - 08/30/2019 02:13 AM - mame (Yusuke Endoh)

spec/ruby/core/unboundmethod/bind_call_spec.rb: Added

For UnboundMethod#bind_call [Feature #15955] introduced in
002e592e0d67bb0271d16314a32380ad947c9ae9.

Revision c9fc8298 - 08/30/2019 02:13 AM - mame (Yusuke Endoh)

lib/pp.rb: Use UnboundMethod#bind_call instead of .bind(obj).call(...)

Related to [Feature #15955].

History

#1 - 06/28/2019 10:25 AM - Eregon (Benoit Daloze)

Escape analysis might be able to remove the Method allocation of unbound.bind(recv).call(*args).
In fact, TruffleRuby does it for such a pattern.

So the interesting question for me is whether this should be fixed by the JIT or by a new method.

Could you share a benchmark representing your usage?

#2 - 07/25/2019 09:01 PM - nelhage (Nelson Elhage)

Whoops, sorry for the belated response -- Redmine email seems to not be working for me. We have a replace_method helper that is a shorthand for doing something like:

```
orig_require = Kernel.instance_method(:require)
Kernel.define_method(:require) do |*args|
  # … do some pre-processing
  orig_require.bind(self).call(*args)
end
```

We use it in a number of places, including to replace require as part of our custom autoloader (c.f. this talk: https://www.youtube.com/watch?v=IKMOETQAdzs)

I'm not quite sure how to get a representative microbenchmark; I am sure I could construct ones where the overhead is anywhere from ~0% to arbitrarily high. Profiling shows that as much as ~6% of the allocation on app startup comes from UnboundMethod#bind calls.

There are also other places I'd like to use this idiom. We had an incident the other day related to Sorbet's use of is_a? (a BasicObject subtype that was being passed around had a surprising is_a? implementation) in runtime checking.

I'd love to be able to grab Object.instance_method(:is_a?) and then use that in our typechecking to make sure that we can test true subtyping no matter what monkey-patches are in place, but we know from past work that adding even a single allocation to the common case of runtime typechecking is too expensive.

#3 - 07/26/2019 01:48 PM - byroot (Jean Boussier)

Zeitwerk had the exact same use case recently:
https://github.com/fxn/zeitwerk/blob/ba7ff65d40a4309701981f9443249ac7e0e8c65f/lib/zeitwerk/real_mod_name.rb

i.e. get the true Module name even if the method was redefined.

#4 - 07/26/2019 06:47 PM - Eregon (Benoit Daloze)

I think this makes sense for convenience and better performance on MRI or during interpretation (vs in compiled code).

Using an UnboundMethod for getting a copy of a method at a given time is indeed a good usage, we use it in TruffleRuby quite a bit.

For the specific case of is_a?, may I recommend using Module#===?
That's much less often overridden, and it works for BasicObject (#is_a? is only defined in Kernel, so not for BasicObject).

#5 - 07/30/2019 08:13 AM - ko1 (Koichi Sasada)

> In fact, TruffleRuby does it for such a pattern.

I wonder that people use this pattern! (I'd never used it except test).

#6 - 08/13/2019 09:27 PM - darkdimius (Dmitry Petrashko)

> I wonder that people use this pattern! (I'd never used it except test).

This pattern currently represents substantial fraction of allocations that Sorbet runtime does, so building a way to not allocate in this pattern might have a sizeable impact in reducing the overhead of runtime type checking.

#7 - 08/23/2019 02:40 AM - mame (Yusuke Endoh)

- File umethod_apply.patch added

Hi nelhage (Nelson Elhage) and darkdimius (Dmitry Petrashko) :-)

I'm attaching a patch for UnboundMethod#apply(obj, *args, &blk) as a shortcut to .bind(obj).call(*args, &blk) without allocation of a Method object.

I have heard the same situation as Sorbet for pp. pp calls method method against its arguments. The intention is to call Object#method, but sometimes it is overridden with completely different behavior, e.g., Net::HTTPGenericRequest#method. So pp is using the same hack:
https://github.com/ruby/ruby/blob/9ffb0548bf95e1113f5657453c64477e792d1230/lib/pp.rb#L92

So I agree with the proposal. I'm unsure if the name apply is good or not. I'd like to ask matz at the next dev-meeting.

#8 - 08/23/2019 03:28 AM - mame (Yusuke Endoh)

Here is a benchmark:

```
class Foo
  def foo
  end
end
meth = Foo.instance_method(:foo)
obj = Foo.new

10000000.times { meth.bind(obj).call } # 1.84 sec
10000000.times { meth.apply(obj)     } # 1.04 sec
```

#9 - 08/30/2019 12:05 AM - mame (Yusuke Endoh)

Matz approved the feature. The name "apply" was arguable in some terms:

- We may want to use the name "apply" for other purpose in future.
- This API will be used not so frequently. Only some fundamental libraries (like pp, sorbet-runtime, zeitwerk, etc.) will use it.

I proposed UnboundMethod#bind_call at the developers' meeting, and matz liked it. I'll commit it soon.

#10 - 08/30/2019 03:04 AM - mame (Yusuke Endoh)

- Status changed from Open to Closed

Applied in changeset git|83c6a1ef454c51ad1c0ca58e8a95fd67a033f710.

---

proc.c: Add UnboundMethod#bind_call

umethod.bind_call(obj, ...) is semantically equivalent to
umethod.bind(obj).call(...). This idiom is used in some libraries to
call a method that is overridden. The added method does the same
without allocation of intermediate Method object. [Feature #15955]

```
class Foo
  def add_1(x)
    x + 1
  end
end
class Bar < Foo
  def add_1(x) # override
    x + 2
  end
end

obj = Bar.new
p obj.add_1(1) #=> 3
p Foo.instance_method(:add_1).bind(obj).call(1) #=> 2
p Foo.instance_method(:add_1).bind_call(obj, 1) #=> 2
```

Files

umethod_apply.patch	5.01 KB	08/23/2019	mame (Yusuke Endoh)