Implementing this feature can boost template engine performance.

Currently Kernel#eval can accept binding argument, so code running with eval will have access to local variables and current instance. This feature used by template languages:

- **ERB:** [https://github.com/ruby/ruby/blob/trunk/lib/erb.rb#L887](https://github.com/ruby/ruby/blob/trunk/lib/erb.rb#L887)
- **Erubis:** Can't find code on github, but it uses instance_eval or Kernel#eval

My proposal is to make RubyVM::InstructionSequence#eval to receive binding argument. So it can be used for caching templates. As I see from ERB and Haml, cached template is stored as ruby code string, every time when we render template that string (ruby code) is evaluated, internally ruby will parse it into RubyVM::InstructionSequence and then evaluate.

Before I try to implement it myself in ruby, but could not. Lack of experience with C [https://github.com/Paxa/ruby/commit/f5b602b6d9eada9675a9c002c9a5a79129df73a6](https://github.com/Paxa/ruby/commit/f5b602b6d9eada9675a9c002c9a5a79129df73a6) (not working)

### History

**#1 - 02/21/2016 04:53 AM - nobu (Nobuyoshi Nakada)**
- Project changed from 14 to Ruby master

**#2 - 02/22/2016 08:06 AM - nobu (Nobuyoshi Nakada)**

Depending on the context, an identifier may be a local variable or a method call. I think that RubyVM::InstructionSequence#compile would need the binding, instead of #eval.

**#3 - 03/03/2016 05:10 AM - shyouhei (Shyouhei Urabe)**

"ISeq#compile's need of binding" means a template engine cannot cache compiled ISeqs for later invocation, right? I doubt the benefit of compile's taking bindings.

**#4 - 03/04/2016 02:44 PM - nobu (Nobuyoshi Nakada)**

Do you mean same template with different contexts, a name is a variable one time, but a method call next time? I doubt that it is a common use case.

**#5 - 09/29/2016 05:41 AM - nobu (Nobuyoshi Nakada)**

I discovered an old patch for this issue. This enables the following code, but doesn't seem useful to me.

```
obj = Struct.new(:a, :b).new(1, 2)
bind = obj.instance_eval { [binding] }
RubyVM::InstructionSequence.compile("a + b").eval_with(bind) #=> 3
```

**#6 - 07/22/2019 04:51 PM - dalehamel (Dale Hamel)**

Howdy,

Sorry to ping a 3 year old issue, I just wanted to add my 2 cents here.

I came across this issue when googling for a way to evaluate an instruction sequence with a particular binding. I'm working on an experimental gem that would inject "breakpoints" in arbitrary lines in ruby methods, with the idea that eBPF / bpftrace can be used to read values from these overridden methods.

Right now I'm using a block to 'handle' the original source code in its original binding, but I have to use ruby's 'eval' method to do this.

I'd ideally like to precompile the original source code sequence, and evaluate this with the original binding.
Here's the current draft of the patch set, which I intend to submit a github pull request for as well.

I've retained Nobu's patch, and built on it.

Nobu recently added it for the next developer meeting (in August; see https://bugs.ruby-lang.org/issues/15996) so stay tuned. :)

Awesome I just saw that - thanks for the update!

The latest patch is now at https://github.com/ruby/ruby/pull/2298 and so that's where the review should go.

I'll stay-tuned and watch for updates from that meeting, thanks Robert!

What the last line should output?

def a; :m_a end
def b; :m_b end
def bind
    a = :l_a
    b = :l_b
    binding
end
eval('p [a, b]', bind())
 #=> [:l_a, :l_b]

RubyVM::InstructionSequence.compile("p [a, b]").eval
 #=> [:m_a, :m_b]

RubyVM::InstructionSequence.compile("p [a, b]").eval(bind())
 #=> ???

I believe we shouldn't introduce binding option to ISeq#eval.

Yes when I test out Koichi's sample, the iseq look like:

disasm: #<ISeq:<compiled>@<compiled>:1 (1,0)-(1,6)> (catch: FALSE)
0000 putself  ( 1)[Li]
0001 opt_send_without_block  <callinfo!mid:a, argc:0, FCALL|VCALL|ARGS_SIMPLE>, <callcache>
0004 putself
0005 opt_send_without_block  <callinfo!mid:b, argc:0, FCALL|VCALL|ARGS_SIMPLE>, <callcache>
0008 newarray  2
0010 leave

So there is no way for the local variables from the binding to be evaluated, as the original instruction sequence expects a method call. I hadn't realized that when compiling the iseq string, methods calls are found in this way.

This indicates that yeah, Nobu's comment above appears correct, you must have the binding available when the iseq is compiled. It appears to do so implicitly based on the current binding.

It looks works with the struct example because the values for a and b have method calls that can receive and respond instead of these local variables, avoiding the problem. This seems inconsistent with Kernel#eval and binding#eval, which is counterproductive.

Ok. I reject this ticket, and pls remake your proposal if you find a good way.
Understood, I've closed the pull request.

#14 - 08/17/2019 11:49 PM - nobu (Nobuyoshi Nakada)
Indeed eval with an arbitrary Binding doesn't make a sense.
How about eval on a given object?
Currently, iseqs eval always on the top-level object without any argument, and I've needed code like:

```
RubyVM::InstructionSequence.compile("proc {...}").eval.call(obj).call(*args)
```

I think it should be simpler.

```
RubyVM::InstructionSequence.compile("proc {...}").bind(obj).call(*args)
```

# or
```
RubyVM::InstructionSequence.compile("proc {...}", receiver: obj).call(*args)
```

@dalehamel (Dale Hamel), does this suffice your use case?

#15 - 08/24/2019 05:10 AM - dalehamel (Dale Hamel)

does this suffice your use case?

Interesting, I'll need to investigate this - it certainly has potential.

My use case is for experimental tracing work, and I basically want to be able to pre-compile original source with added instructions, and execute them within the context they were originally intended to be executed within.

This is why I had a use for being able to execute within arbitrary bindings, but if I can target right receiver / bind to the right object, this could work.

I will try a prototype by seeing which receiving I am presently binding to, and look into modifying the patch to support the prototype you suggest to see if it can fit my use case by passing this receiver rather than the binding.

Thank you for response and feedback Nobu.

Files

<table>
<thead>
<tr>
<th>File Name</th>
<th>Size</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>0002-Update-iseq.eval-to-accept-optional-binding-FIXES-Bu.patch</td>
<td>4.83 KB</td>
<td>07/23/2019</td>
<td>dalehamel (Dale Hamel)</td>
</tr>
<tr>
<td>0001-RubyVM-InstructionSequence-eval_with.patch</td>
<td>2.91 KB</td>
<td>07/23/2019</td>
<td>dalehamel (Dale Hamel)</td>
</tr>
</tbody>
</table>