Ruby - Feature #16855

Add a tracepoint for warnings

05/13/2020 05:40 PM - tenderlovemaking (Aaron Patterson)

Status:	Rejected	
Priority:	Normal	
Assignee:		
Target version:		

Description

I would like to add a tracepoint for warnings. I want to do this so that DidYouMean can suggest fixes for instance variables. I noticed did you mean has experimental support, but it looks very complicated. I think if we added a tracepoint for such warnings, DidYouMean can provide more helpful warnings.

I made a pull request here

History

#1 - 05/13/2020 06:57 PM - Eregon (Benoit Daloze)

I'm afraid I don't see the point of a TracePoint for this if warnings can already be hooked via Warning.prepend SomeModuleWithWarn.

Invoking the TracePoint regardless of $VERBOSE would be a large performance cost, and would prevent optimizations like not building the warning message or finding the file:line when $VERBOSE is false/nil.
(e.g., https://github.com/oracle/truffleruby/commit/86af0e5e224680e81e3c0a287f82ca8263ca079e)

One could still speculate on the TracePoint not being used, but if DidYouMean uses it then it's pointless to speculate, it would always be used.

I think people should rather learn to use -w/-d/$VERBOSE, or they'll miss on lots of existing help to find such issues.

#2 - 05/13/2020 06:58 PM - tenderlovemaking (Aaron Patterson)

- Status changed from Open to Rejected

Eregon (Benoit Daloze) wrote in #note-1:

> I'm afraid I don't see the point of a TracePoint for this if warnings can already be hooked via Warning.prepend SomeModuleWithWarn.
>
> Invoking the TracePoint regardless of $VERBOSE would be a large performance cost, and would prevent optimizations like not building the warning message or finding the file:line when $VERBOSE is false/nil.
> (e.g., https://github.com/oracle/truffleruby/commit/86af0e5e224680e81e3c0a287f82ca8263ca079e)
>
> One could still speculate on the TracePoint not being used, but if DidYouMean uses it then it's pointless to speculate, it would always be used.
>
> I think people should rather learn to use -w/-d/$VERBOSE, or they'll miss on lots of existing help to find such issues.

Yep, I agree. I'm closing this :)

#3 - 05/13/2020 07:04 PM - Eregon (Benoit Daloze)

Maybe another angle for this would be to make the instance variable @foobar not initialized shown even with the default $VERBOSE being false (i.e. rb_warn instead of rb_warning).
Then people would likely be a lot more eager to fix it.

It can be advantageous for performance to have ivars always initialized when reading them, otherwise there is some kind of polymorphism introduced (i.e., executed when the ivar doesn't exist and executed when the ivar exists).

#4 - 05/13/2020 07:41 PM - jeremyevans0 (Jeremy Evans)

Eregon (Benoit Daloze) wrote in #note-3:

> Maybe another angle for this would be to make the instance variable @foobar not initialized shown even with the default $VERBOSE being false (i.e. rb_warn instead of rb_warning).
> Then people would likely be a lot more eager to fix it.

This would make optimized code slower. Explicitly initializing instance variables to nil can have a significant negative effect on performance. If you didn't want to explicitly initialize instance variables to nil, you would need to protect all access to them with something like defined?(@ivar) or by using

an attr_reader method (which doesn't warn), both of which would also be slower.

> It can be advantageous for performance to have ivars always initialized when reading them, otherwise there is some kind of polymorphism introduced (i.e., executed when the ivar doesn't exist and executed when the ivar exists).

This is only true if the ivars are accessed directly many times. It isn't until around 50 accesses before it makes sense from a performance standpoint to explicitly initialize ivars to nil if you take into account the time taken to explicitly initialize them.

This is not an insignificant performance issue. Assuming you have 4 instance variables, if you aren't accessing the instance variables, not explicitly initializing them to nil is over 60% faster. If you are only accessing them once, it's over 50% faster. Even if you are accessing them 10 times, not initializing them is about 20% faster.

For optimal performance, you should only explicitly initialize instance variables to nil for long-lived objects.

Benchmark code:

```
require 'benchmark/ips'

class IV
  eval "def check1; #{"@a || @b || @c || @d ||" * 1} nil end"
  eval "def check10; #{"@a || @b || @c || @d ||" * 10} nil end"
  eval "def check50; #{"@a || @b || @c || @d ||" * 50} nil end"
  eval "def check100; #{"@a || @b || @c || @d ||" * 100} nil end"
end

class DefIV < IV
  def initialize; @a = @b = @c = @d = nil end
end

Benchmark.ips do |x|
 x.report("No initialization - No Check"){IV.new}
 x.report("No initialization - Check 1 time"){IV.new.check1}
 x.report("No initialization - Check 10 times"){IV.new.check10}
 x.report("No initialization - Check 50 times"){IV.new.check50}
 x.report("No initialization - Check 100 times"){IV.new.check100}
 x.report("Initialization - No Check"){DefIV.new}
 x.report("Initialization - Check 1 time"){DefIV.new.check1}
 x.report("Initialization - Check 10 times"){DefIV.new.check10}
 x.report("Initialization - Check 50 times"){DefIV.new.check50}
 x.report("Initialization - Check 100 times"){DefIV.new.check100}
end
```

Results with 2.7.1:

```
Calculating -------------------------------------
No initialization - No Check
                          2.255M (_ 0.3%) i/s -     11.295M in   5.007986s
No initialization - Check 1 time
                          1.813M (_ 0.6%) i/s -      9.229M in   5.090357s
No initialization - Check 10 times
                        847.470k (_ 0.3%) i/s -      4.319M in   5.096086s
No initialization - Check 50 times
                        253.392k (_ 0.5%) i/s -      1.291M in   5.094078s
No initialization - Check 100 times
                        135.259k (_ 0.3%) i/s -    689.724k in   5.099341s
Initialization - No Check
                          1.367M (_ 0.3%) i/s -      6.965M in   5.096278s
Initialization - Check 1 time
                          1.203M (_ 0.5%) i/s -      6.134M in   5.097248s
Initialization - Check 10 times
                        711.272k (_ 0.4%) i/s -      3.626M in   5.097540s
Initialization - Check 50 times
                        254.220k (_ 0.3%) i/s -      1.273M in   5.007728s
Initialization - Check 100 times
                        140.121k (_ 0.6%) i/s -    703.400k in   5.020148s
```

#5 - 05/14/2020 12:35 AM - ko1 (Koichi Sasada)

please write a specification of your proposal. there is only a code, test and motivation.