Introducing code injection framework. Different from set_trace_func(), this framework injects codes only specified points.

Note that this proposal is not implemented and well designed (only rough idea) but I dumped it to discuss about this topic. It has (huge) possibility to miss 2.0 spec deadline (should be 3.0 spec?).

Background

To trace, debug, profile and any analysis ruby code, Ruby provides `set_trace_func()` method. set_trace_func() is enough powerful to do them. However set_trace_func() injects codes every tracing points. It cause huge performance impact if you have interest restricted places.

Another problem is that set_trace_func() can not affect program behavior. For example, we can not insert type checking code for specific method invocation.

Related works with introducing codes are described below. Please point out if you know another related works.

Bytecode instrumentation

JVM has JVMTI interface [http://docs.oracle.com/javase/6/docs/platform/jvmti/jvmti.html#bci](http://docs.oracle.com/javase/6/docs/platform/jvmti/jvmti.html#bci) to inject any code with bytecode instrumentation. It can be done because JVM bytecode is well defined and become concrete specification. However, Ruby doesn't have any well-defined common bytecode and difficult to make such common bytecode (at least Ruby 2.0 spec deadline, this August).

Manipulate bytecode directly has other problems:

- Needs more knowledge about bytecode
- Difficult to make `well-formed' bytecode sequence

AOP (Aspect Oriented Programming)

Aspect-Oriented programming frameworks provides join points' which we can insert codes [http://www.eclipse.org/aspectj/doc/released/progguide/language.html]. Such join points' design is well abstracted comparing with bytecode instrumentation.

In fact, AOT compilers such as Aspect-J use bytecode instrumentation.

Module#prepend

We already have Module#prepend that enable to insert any program before/after method invocation.

Example:

```ruby
module EachTracer # call tracing method before/after each method
def each(*args)
  begin
    super # call original each
  ensure
    after_each
  end
```
class Array
  prepend EachTracer

  def before_each
    p:before_each
  end

  def after_each
    p:after_each
  end
end

%w(a b c).each{|c|}
 #=> outputs :before_each and :after_each

However, Module#prepend only works for method invocation.

Proposal

Introduce code injection framework. It should provide two features: (1) "where should insert codes?" and (2) "what code should be
insert?".

RubyVM::InstructionSequence#each_point(point_name) is temporal API for (1). each_point invoke block with CodePoint object.
CodePint#set_proc (or something) is for (2).

Example (it is rough API idea):

def m1
  m2(1)
  m2(1, 2, 3)
  m3()
  m4()
end

# insert proc before m2 method invocation
method(:m1).iseq.each_point(:before_call){|point|
  # point is CodePoint object.
  if point.selector == :m2
    point.set_proc{|*args| # before call m2 with #{args.inspect}"
  end
}

# another idea
method(:m1).iseq.each_point(:invoke_method){|point|
  if point.selector == :m2
    point.insert_proc_before{|*args| # before call m2 with #{args.inspect}"
  else
    point.insert_proc_after{|retval| # after call m2 with return value #{retval}"
  end
  else
    point.replace_proc{|*args| # cancel invoking m4 and call this proc instead"
  end
}

Injection points are categorized into 3 types:

- (1) before/after invoke something
- method call (before method call)
- method call (after method call)
- block invocation (before)
- block invocation (after)
- super invocation (before)
- super invocation (after)

- (2) enter/leave (not needed?)
  - method (enter) (set_trace_func/call)
  - method (leave) (set_trace_func/return)
  - class/module definition (enter) (set_trace_func/class)
  - class/module definition (leave) (set_trace_func/end)
  - block (enter)
  - block (leave)
  - rescue (enter)
  - rescue (leave)
  - ensure (enter)
  - ensure (leave)

- (3) misc
  - read variable ($gv, @iv, @@cv)
  - write variable ($gv, @iv, @@cv)
  - read constant (Const)
  - define constant (Const)
  - method definition
  - newline (set_trace_func/line)

This proposal can introduce (limited) code manipulation without any bytecode knowledge.

Usecase

- inserting specific break points for debugger
- inserting specific analysis points for profiler
- inserting type checking code generated by rdoc
- making Aspect-J like tool (note that Module#prepend is enough if you only want to replace method invocation behavior)

Any other idea?

Limitation

It is impossible to inject any code into methods implemented by C.

I'm afraid that this proposed API makes magical (unreadable) codes for script kiddies :P

I repeat it again: Note that this proposal is not implemented and well designed (only rough idea) but I dumped it to discuss about this topic. It has (huge) possibility to miss 2.0 spec deadline (should be 3.0 spec?).

Thanks,
Koichi

History

#1 - 07/14/2012 06:42 PM - mame (Yusuke Endoh)
- Status changed from Open to Assigned

#2 - 07/25/2012 06:52 PM - ko1 (Koichi Sasada)
Nobody has interest about it. I'll implement and show if I can make it before the deadline.

Thanks,
Koichi
ko1 (Koichi Sasada) wrote:

Nobody has interest about it.

I don't think so, I think it's just not clear what can offer this in practice yet.

The use-cases you listed are certainly interesting.

Would this, for example, significantly improve lib/debug.rb and lib/profiler.rb speed?
(My question is more about the API being able to replace set_trace_func in these scenarios)

However set_trace_func() injects codes every tracing points. It cause huge performance impact if you have interest restricted places.

Do you think this would be significantly faster for method tracing as in your example?
It avoids other type of events (and does not generate binding and such), but it still is invoked at every method call. Maybe it would be worth to have an API which can give the method name, like each_point(:invoke_method, :m1).

I think it would be interesting to have the ability to reuse the original call(iseq?) in replace_proc, to wrap it inside other code, which can not easily be written with insert_before/after.

This is interesting. Basically you propose to get rid of the overhead of set_trace_func by injecting code into "code points" only where actually used. If so, that would be very cool, b/c it could be used for event-based AOP and it would be efficient enough to actually be usable!

For better API, I wonder if we can wrangle a clean approach out of ideas of https://bugs.ruby-lang.org/issues/6649?

Btw, might want to refer to "code point" as "join point" to save confusion with encoding terminology.

- Target version changed from 2.0.0 to 2.6

Sorry, maybe I don't have enough time to make it until 2.0 release.

I will introduce C level (helper?) API to implement this feature if I can.

#6 - 03/04/2014 05:29 AM - nobu (Nobuyoshi Nakada)
- Description updated

- Status changed from Assigned to Closed

I hope someone try this idea :p