

Ruby - Feature #2740

Extend const_missing to pass in the nesting

02/12/2010 11:14 AM - wycats (Yehuda Katz)

Status:Closed Priority:Normal Assignee:matsuda (Akira Matsuda) Target version:2.6	
Description =begin At the moment, it is impossible for const_missing to differentiate between these cases: <pre>class Foo::Bar Baz end class Foo class Bar Baz end end</pre> In Rails, we implement a class loading system that loads classes from the file system if they are not found. In the above case, Foo::Baz might be stored in app/models/foo/baz.rb. We would like to be able to use the same Ruby constant lookup logic when looking up classes from the file system. <pre>class Foo::Bar Baz end</pre> Here, we should look in "app/models/foo/bar/baz.rb" and in "app/models/baz.rb" just as Ruby would search for Foo::Bar::Baz and then Object::Baz. <pre>class Foo class Bar Baz end end</pre> Here, we should look in "app/models/foo/bar/baz.rb", then "app/models/foo/baz.rb", and finally "app/models/baz.rb" just as Ruby would search for Foo::Bar::Baz, then Foo::Baz, and then Object::Baz. In order to achieve this, I propose that we extend the const_missing method to take an optional second parameter containing the nesting: <pre>class Foo class Bar def self.const_missing(id, nesting) id == :Baz nesting == [Foo::Bar] # first case nesting == [Foo::Bar, Foo] # second case end end end</pre> This would allow people who wish to do their own custom constant loading (such as Rails) to do so in a way that is consistent with Ruby's own constant lookup. In order to avoid backward-compatibility issues, we can check the arity of the const_missing method, and only pass in the nesting if a second parameter was provided. =end	

History

#1 - 02/12/2010 11:23 AM - naruse (Yui NARUSE)

- Status changed from Open to Assigned
- Assignee set to matz (Yukihiro Matsumoto)
- Priority changed from 6 to Normal

#2 - 02/13/2010 02:14 AM - murphy (Kornelius Kalnbach)

Interesting idea. However, I see two problems with the proposal:

1. It changes the `const_missing` API, breaking code. It would be nice to find a way around this.
2. The feature `Module.nesting` is dependent on the "point of the call". As you described, `const_missing` is not. I'm not sure about the implications; `Module.nesting` is defined in `eval.c`, while `const_missing` and `const_get` are defined in `variable.c`.

There may be another way around this. `method_missing` makes it possible to use Ruby's inheritance chain again by calling `super`. So, what if we just follow the constant lookup nesting chain again when `const_missing` didn't return anything?

You wouldn't even have to write the logic yourself. The implementation could look something like this:

```
class Module
  # classic-style const_missing
  def const_missing(id)
    namespace = "#{name}::" unless self == Object
    puts "Searching for #{namespace}#{id}..."
    nil
  end
end

class Module
  # new-style constant lookup, given nesting
  # This would have to be implemented in C.
  def const_get2 id, nesting
    nesting += [Object] # Don't forget top level.
    # As usual: Try to find existing constant (without const_missing).
    for mod in nesting
      # imitating constant lookup without const_missing
      if mod.constants.map { |c| c.to_sym }.include? id.to_sym
        return mod.const_get(id)
      end
    end
    # New style: Call const_missing on each nesting level.
    for mod in nesting
      value = mod.const_missing id
      return value if value
    end
    # As usual: Raise NameError if nothing was found.
    raise NameError, "uninitialized constant #{name}::#{id}"
  end
end

class Foo
  class Bar
    const_get2(:Baz, Module.nesting)
    # >> Searching for Foo::Bar::Baz...
    # >> Searching for Foo::Baz...
    # >> Searching for Baz...
    #~> uninitialized constant Foo::Bar::Baz (NameError)
  end
end

class Foo::Bar
  const_get2(:Baz, Module.nesting)
  # >> Searching for Foo::Bar::Baz...
  # >> Searching for Baz...
  #~> uninitialized constant Foo::Bar::Baz (NameError)
end
```

This solves 1., but not 2. Calling `const_get` and would still depend on `Module.nesting`.

Also, a constant with a value of nil or false would be interpreted as "not found". Maybe we could define a protocol for this, something like StopIteration?

```
NoConstantError = Class.new NameError
```

```
class Module
  def const_missing(id)
    ...
    raise NoConstantError
  end
end

class Module
  def const_get2 id, nesting
    ...
    for mod in nesting
      begin
        return mod.const_missing(id)
      rescue NoConstantError
        # continue
      end
    end
    ...
  end
end
```

On the other hand, maybe implicit constant lookup through the module nesting *and* automatic loading is a bit too much ;-)

#3 - 02/13/2010 08:53 AM - wycats (Yehuda Katz)

- File `const_missing_nesting.diff` added

Interesting idea. However, I see two problems with the proposal:

1. It changes the `const_missing` API, breaking code. It would be nice to find a way around this.

A simple arity check will get around this issue.

1. The feature `Module.nesting` is dependent on the "point of the call". As you described, `const_missing` is not. I'm not sure about the implications; `Module.nesting` is defined in `eval.c`, while `const_missing` and `const_get` are defined in `variable.c`.

`Module.nesting` is essentially global (stored in a thread-local), so we can get it in `const_missing` via `rb_funcall`. If we made `rb_mod_nesting` not static, we could also access it via `rb_mod_nesting`.

I have attached a patch that makes nesting available in `const_missing` if the `const_missing` method has an arity other than 1.

#4 - 02/13/2010 05:41 PM - nobu (Nobuyoshi Nakada)

Hi,

At Fri, 12 Feb 2010 11:15:14 +0900,
Yehuda Katz wrote in [\[ruby-core:28154\]](#):

```
class Foo::Bar
  Baz
end
```

Here, we should look in "app/models/foo/bar/baz.rb" and in "app/models/baz.rb" just as Ruby would search for `Foo::Bar::Baz` and then `Object::Baz`.

```
class Foo
  class Bar
    Baz
  end
end
```

Here, we should look in "app/models/foo/bar/baz.rb", then "app/models/foo/baz.rb", and finally "app/models/baz.rb" just as Ruby would search for Foo::Bar::Baz, then Foo::Baz, and then Object::Baz.

I don't think you can distinguish the latter from the following.

```
class Foo
  class Bar
    Foo::Bar::Baz
  end
end
```

Is it OK?

--
Nobu Nakada

#5 - 02/13/2010 08:59 PM - mame (Yusuke Endoh)

Hi,

2010/2/13 Yehuda Katz redmine@ruby-lang.org:

1. It changes the const_missing API, breaking code. It would be nice to find a way around this.

A simple arity check will get around this issue.

It's ugly. The core should avoid such a dirty hack when possible.
I also think it would be nice to find another clean API.

I have attached a patch that makes nesting available in const_missing if the const_missing method has an arity other than 1.

It causes SEGV.

```
$ ./miniruby -e '
class C
  def self.const_missing(id, nesting)
    p id, nesting
  end
end
C.enum_for(:const_get, :D).next
'
```

[BUG] Segmentation fault
ruby 1.9.2dev (2010-02-13 trunk 26659) [i686-linux]

```
-- control frame -----
c:0005 p:---- s:0009 b:0009 l:000008 d:000008 CFUNC :nesting
c:0004 p:---- s:0007 b:0007 l:000006 d:000006 CFUNC :const_get
c:0003 p:---- s:0005 b:0005 l:000004 d:000004 CFUNC :each
c:0002 p:---- s:0003 b:0003 l:002168 d:000002 IFUNC
c:0001 p:---- s:0001 b:-001 l:000000 d:000000 -----
-----
-- Ruby level backtrace information -----
-e:0:in `each'
-e:0:in `const_get'
-e:0:in `nesting'
```

```
-- C level backtrace information -----
./miniruby(rb_vm_bugreport+0xbd) [0x81a5c6d]
./miniruby [0x808e69e]
./miniruby(rb_bug+0x28) [0x808e758]
./miniruby [0x813a6b0]
[0xfffffe40c]
./miniruby(rb_vm_cref+0x38) [0x8193d88]
./miniruby [0x8090dc4]
./miniruby [0x819d889]
./miniruby(rb_funcall+0x92) [0x819e142]
./miniruby [0x817d571]
./miniruby [0x80db78c]
```

```

./miniruby [0x8191bfd]
./miniruby [0x819d889]
./miniruby [0x81a1c2e]
./miniruby(rb_iterate+0xac) [0x8191dac]
./miniruby(rb_block_call+0x3f) [0x8191f3f]
./miniruby [0x808c0d9]
./miniruby [0x819d889]
./miniruby [0x81a1c2e]
./miniruby(rb_iterate+0xac) [0x8191dac]
./miniruby(rb_block_call+0x3f) [0x8191f3f]
./miniruby [0x808c162]
./miniruby [0x8195d5b]
./miniruby(rb_vm_invoke_proc+0x76) [0x819c816]
./miniruby(rb_fiber_start+0x1d2) [0x81aed52]
./miniruby [0x8091479]
./miniruby(ruby_run_node+0x32) [0x8092c92]
./miniruby(main+0x5a) [0x805a8da]
/lib/i686/cmov/libc.so.6(__libc_start_main+0xe5) [0xb7d89455]
./miniruby [0x805a7e1]

```

[NOTE]

You may have encountered a bug in the Ruby interpreter or extension libraries.

Bug reports are welcome.

For details: <http://www.ruby-lang.org/bugreport.html>

Aborted

--

Yusuke ENDOH ame@tsg.ne.jp

#6 - 02/15/2010 10:21 PM - wycats (Yehuda Katz)

=begin

Upon further reflection, I really like the idea of raising an exception like NoConstantError from const_missing to tell Ruby to keep searching up the nesting chain.

```

class Module
  def const_missing(name)
    puts "#{self} is missing #{name}"
    raise NoConstantError unless self == Object
  end
end

```

```

module Foo
  module Bar
    Baz
  end
end

```

```

# Output:
# Foo::Bar is missing Baz
# Foo is missing Baz
# Object is missing Baz

```

```

module Foo::Bar
  Baz
end

```

```

# Output
# Foo::Bar is missing Baz
# Object is missing Baz

```

```

module Foo::Bar
  Foo::Bar::Baz
end

```

```

# Output
# Foo::Bar is missing Baz

```

```

module Foo
  Bar::Baz
end

```

```

# Output
# Foo::Bar is missing Baz

```

Thoughts?

=end

#7 - 02/16/2010 09:23 AM - nobu (Nobuyoshi Nakada)

Hi,

At Mon, 15 Feb 2010 22:21:55 +0900,
Yehuda Katz wrote in [\[ruby-core:28177\]](#):

Upon further reflection, I really like the idea of raising an
exception like NoConstantError from const_missing to tell
Ruby to keep searching up the nesting chain.

Maybe reasonable.

Index: error.c

```
=====
--- error.c (revision 26674)
+++ error.c (working copy)
@@ -402,4 +402,6 @@ VALUE rb_eSyntaxError;
     VALUE rb_eLoadError;

+VALUE rb_eNoConstantError;
+
+    VALUE rb_eSystemCallError;
+    VALUE rb_mErrno;
@@ -1143,4 +1145,5 @@ Init_Exception(void)
     rb_define_method(rb_eNoMethodError, "initialize", nometh_err_initialize, -1);
     rb_define_method(rb_eNoMethodError, "args", nometh_err_args, 0);
+    rb_eNoConstantError = rb_define_class("NoConstantError", rb_eNameError);

     rb_eRuntimeError = rb_define_class("RuntimeError", rb_eStandardError);
```

Index: variable.c

```
=====
--- variable.c (revision 26674)
+++ variable.c (working copy)
@@ -1356,7 +1356,29 @@ uninitialized_constant(VALUE klass, ID i
+
+    static VALUE
+const_missing_call(VALUE arg)
+{
+    VALUE *args = (VALUE *)arg;
+    ID const_missing_id;
+    CONST_ID(const_missing_id, "const_missing");
+    return rb_check_funcall(args[0], const_missing_id, 1, &args[1]);
+}
+
+static VALUE
+const_missing_rescue(VALUE arg, VALUE errinfo)
+{
+    return arg;
+}
+
+extern VALUE rb_eNoConstantError;
+
+static VALUE
+const_missing(VALUE klass, ID id)
+{
+    return rb_funcall(klass, rb_intern("const_missing"), 1, ID2SYM(id));
+    VALUE args[2];
+    args[0] = klass;
+    args[1] = ID2SYM(id);
+    return rb_rescue2(const_missing_call, (VALUE)args,
+        const_missing_rescue, (VALUE)Qundef,
+        rb_eNoConstantError, (VALUE)0);
+}

@@ -1598,5 +1620,21 @@ rb_const_get_0(VALUE klass, ID id, int e
+    }

-    value = const_missing(klass, id);
+    if ((value = const_missing(tmp, id)) == Qundef) {
+    NODE *rb_vm_cref(void);
```

```

+ NODE *cref = rb_vm_cref();
+ while (cref && cref->nd_next &&
+       ((cref->flags & NODE_FL_CREF_PUSHED_BY_EVAL) ||
+        NIL_P(tmp = cref->nd_clss) ||
+        (value = const_missing(tmp, id)) == Qundef)) {
+     cref = cref->nd_next;
+ }
+ if (value == Qundef) {
+     if (!exclude && BUILTIN_TYPE(klass) == T_MODULE &&
+         (value = const_missing(rb_cObject, id)) == Qundef) {
+         uninitialized_constant(klass, id);
+     }
+ }
+ }
+
+     rb_vm_inc_const_missing_count();
+     return value;

```

Nobu Nakada

#8 - 02/16/2010 11:53 AM - murphy (Kornelius Kalnbach)

- File *const_missing_murphy.patch* added

Nobu, great to see a patch for this! Thank you.

However, it didn't produce the desired results for me. After some debugging, I got a version that works for me:

```

class Object
  def self.const_missing(id)
    namespace = "#{name}::" unless self == Object
    puts "Searching for #{namespace}#{id}..."
    raise NoConstantError
  end
end

class Foo
  class Bar
    p Module.nesting
    Baz rescue nil
    # [Foo::Bar, Foo]
    # Searching for Foo::Bar::Baz...
    # Searching for Foo::Baz...
    # Searching for Baz...
  end
end

class Foo::Bar
  p Module.nesting
  Baz rescue nil
  # [Foo::Bar]
  # Searching for Foo::Bar::Baz...
  # Searching for Baz...
end

```

It also passes "make test".

Yehuda: Your example works as you described with this patch.

```

Index: error.c
=====
--- error.c (revision 26674)
+++ error.c (working copy)
@@ -402,4 +402,6 @@ VALUE rb_eSyntaxError;
  VALUE rb_eLoadError;

+VALUE rb_eNoConstantError;
+
  VALUE rb_eSystemCallError;
  VALUE rb_mErrno;
@@ -1143,4 +1145,5 @@ Init_Exception(void)
  rb_define_method(rb_eNoMethodError, "initialize", nometh_err_initialize, -1);
  rb_define_method(rb_eNoMethodError, "args", nometh_err_args, 0);
+  rb_eNoConstantError = rb_define_class("NoConstantError", rb_eNameError);

```

```

    rb_eRuntimeError = rb_define_class("RuntimeError", rb_eStandardError);
Index: variable.c
=====
--- variable.c (revision 26674)
+++ variable.c (working copy)
@@ -1355,9 +1355,31 @@
 }

static VALUE
+const_missing_call(VALUE arg)
+{
+    VALUE *args = (VALUE *)arg;
+    ID const_missing_id;
+    CONST_ID(const_missing_id, "const_missing");
+    return rb_check_funcall(args[0], const_missing_id, 1, &args[1]);
+}
+
+static VALUE
+const_missing_rescue(VALUE arg, VALUE errinfo)
+{
+    return arg;
+}
+
+extern VALUE rb_eNoConstantError;
+
+static VALUE
+const_missing(VALUE klass, ID id)
+{
+    return rb_funcall(klass, rb_intern("const_missing"), 1, ID2SYM(id));
+    VALUE args[2];
+    args[0] = klass;
+    args[1] = ID2SYM(id);
+    return rb_rescue2(const_missing_call, (VALUE)args,
+        const_missing_rescue, (VALUE)Qundef,
+        rb_eNoConstantError, (VALUE)0);
+}

@@ -1392,8 +1414,7 @@
VALUE
rb_mod_const_missing(VALUE klass, VALUE name)
{
-    rb_frame_pop(); /* pop frame for "const_missing" */
-    uninitialized_constant(klass, rb_to_id(name));
+    rb_raise(rb_eNoConstantError, "uninitialized constant");
    return Qnil; /* not reached */
}

@@ -1596,8 +1617,26 @@
tmp = rb_cObject;
goto retry;
}

+    tmp = klass;
+    if ((value = const_missing(tmp, id)) == Qundef) {
+NODE *rb_vm_cref(void);
+NODE *cref = rb_vm_cref();
+    cref = cref->nd_next;
+    while (cref && cref->nd_next &&
+        ((cref->flags & NODE_FL_CREF_PUSHED_BY_EVAL) ||
+         NIL_P(tmp = cref->nd_clss) ||
+         (value = const_missing(tmp, id)) == Qundef)) {
+        cref = cref->nd_next;
+    }
+    if (value == Qundef) {
+        if (!exclude && (BUILTIN_TYPE(klass) == T_MODULE || BUILTIN_TYPE(klass) == T_CLASS) &&
+            (value = const_missing(rb_cObject, id)) == Qundef) {
+            uninitialized_constant(klass, id);
+        }
+    }
+}

-    value = const_missing(klass, id);
rb_vm_inc_const_missing_count();

```

```
    return value;
}
```

#9 - 02/16/2010 12:37 PM - mame (Yusuke Endoh)

=begin
Hi,

2010/2/15 Yehuda Katz redmine@ruby-lang.org:

Upon further reflection, I really like the idea of raising an exception like NoConstantError from const_missing to tell Ruby to keep searching up the nesting chain.

Cool! I agree with Yehuda.

Additional idea: NoConstantError may be raised when referring uninitialized constant, to make it consistent with NoMethodError.

--
Yusuke ENDOH mame@tsg.ne.jp

=end

#10 - 03/18/2010 02:00 AM - mame (Yusuke Endoh)

Hi,

Matz, do you accept Yehuda's suggestion?

Current status:

- Three persons (nobu, murphy and I) have approved.
- There is no objection.
- There is a patch written by nobu and murphy, though it has a little bug. I revised a patch below.

2010/2/15 Yehuda Katz redmine@ruby-lang.org:

```
class Module
  def const_missing(name)
    puts "#{self} is missing #{name}"
    raise NoConstantError unless self == Object
  end
end
```

*snip

```
module Foo::Bar
  Foo::Bar::Baz
end
```

Output

Foo::Bar is missing Baz

I'd like to make sure one thing; the above code should raise NoConstantError after the output, ok? This is because scoped constant reference does not check bindings of Object class.

Instead of raising an error, nobu's patch leaks Qundef for the above code, which causes SEGV in RubySpec. It is clearly a problem.

```
diff --git a/error.c b/error.c
index 3fcd184..d0373fd 100644
--- a/error.c
+++ b/error.c
@@ -401,6 +401,8 @@ @@ VALUE rb_eScriptError;
    VALUE rb_eSyntaxError;
    VALUE rb_eLoadError;
```

```

+VALUE rb_eNoConstantError;
+
+  VALUE rb_eSystemCallError;
+  VALUE rb_mErrno;
+  static VALUE rb_eNOERROR;
@@ -701,18 +703,34 @@ exit_success_p(VALUE exc)
+  return Qfalse;
+}

+static VALUE
+new_name_error(VALUE klass, ID id, const char *fmt, va_list args)
+{
+  VALUE argv[2];
+  argv[0] = rb_vsprintf(fmt, args);
+  argv[1] = ID2SYM(id);
+  return rb_class_new_instance(2, argv, klass);
+}
+
+void
+rb_name_error(ID id, const char *fmt, ...)
+{
+  VALUE exc, argv[2];
+  VALUE exc;
+  va_list args;
+
+  va_start(args, fmt);
+  argv[0] = rb_vsprintf(fmt, args);
+  exc = new_name_error(rb_eNameError, id, fmt, args);
+  va_end(args);
+  rb_exc_raise(exc);
+}

+  argv[1] = ID2SYM(id);
+  exc = rb_class_new_instance(2, argv, rb_eNameError);
+void
+rb_name_error_with_class(VALUE klass, ID id, const char *fmt, ...)
+{
+  VALUE exc;
+  va_list args;
+  va_start(args, fmt);
+  exc = new_name_error(klass, id, fmt, args);
+  va_end(args);
+  rb_exc_raise(exc);
+}

@@ -1142,6 +1160,7 @@ Init_Exception(void)
+  rb_eNoMethodError = rb_define_class("NoMethodError", rb_eNameError);
+  rb_define_method(rb_eNoMethodError, "initialize", nometh_err_initialize, -1);
+  rb_define_method(rb_eNoMethodError, "args", nometh_err_args, 0);
+  rb_eNoConstantError = rb_define_class("NoConstantError", rb_eNameError);

+  rb_eRuntimeError = rb_define_class("RuntimeError", rb_eStandardError);
+  rb_eSecurityError = rb_define_class("SecurityError", rb_eException);
diff --git a/test/rake/test_top_level_functions.rb b/test/rake/test_top_level_functions.rb
index 12a8cd1..7315f88 100644
--- a/test/rake/test_top_level_functions.rb
+++ b/test/rake/test_top_level_functions.rb
@@ -86,6 +86,6 @@ class Rake::TestTopLevelFunctions < Test::Unit::TestCase
end

def test_missing_other_constant
-  assert_raise(NameError) do Object.const_missing(:Xyz) end
+  assert_raise(NoConstantError) do Object.const_missing(:Xyz) end
end
end

diff --git a/test/ruby/test_basicinstructions.rb b/test/ruby/test_basicinstructions.rb
index ff14e4a..eb985ef 100644
--- a/test/ruby/test_basicinstructions.rb
+++ b/test/ruby/test_basicinstructions.rb
@@ -271,25 +271,25 @@ class TestBasicInstructions < Test::Unit::TestCase
  Const::A::B._remove_const :C
  assert_equal 'Const::C', Const::C
  assert_equal 'Const::A::C', Const::A::C
-  assert_raise(NameError) { Const::A::B::C }
+  assert_raise(NoConstantError) { Const::A::B::C }
end

```

```

    Const::A._remove_const :C
    assert_equal 'Const::C', Const::C
-   assert_raise(NameError) { Const::A::C }
-   assert_raise(NameError) { Const::A::B::C }
+   assert_raise(NoConstantError) { Const::A::C }
+   assert_raise(NoConstantError) { Const::A::B::C }

    Const._remove_const :C
-   assert_raise(NameError) { Const::C }
-   assert_raise(NameError) { Const::A::C }
-   assert_raise(NameError) { Const::A::B::C }
+   assert_raise(NoConstantError) { Const::C }
+   assert_raise(NoConstantError) { Const::A::C }
+   assert_raise(NoConstantError) { Const::A::B::C }

    Const::A.const_set :C, 'Const::A::C'
-   assert_raise(NameError) { Const::C }
+   assert_raise(NoConstantError) { Const::C }
    assert_equal 'Const::A::C', Const::A::C
-   assert_raise(NameError) { Const::A::B::C }
+   assert_raise(NoConstantError) { Const::A::B::C }

    Const::A::B.const_set :C, 'Const::A::B::C'
-   assert_raise(NameError) { Const::C }
+   assert_raise(NoConstantError) { Const::C }
    assert_equal 'Const::A::C', Const::A::C
    assert_equal 'Const::A::B::C', Const::A::B::C

diff --git a/test/ruby/test_module.rb b/test/ruby/test_module.rb
index f905431..fbeb29b 100644
--- a/test/ruby/test_module.rb
+++ b/test/ruby/test_module.rb
@@ -451,13 +451,13 @@ class TestModule < Test::Unit::TestCase
    assert_equal(:foo, c1::Foo)
    assert_equal(:foo, c2::Foo)
    assert_equal(:foo, c2.const_get(:Foo))
-   assert_raise(NameError) { c2.const_get(:Foo, false) }
+   assert_raise(NoConstantError) { c2.const_get(:Foo, false) }

    eval("c1::Foo = :foo")
-   assert_raise(NameError) { c1::Bar }
-   assert_raise(NameError) { c2::Bar }
-   assert_raise(NameError) { c2.const_get(:Bar) }
-   assert_raise(NameError) { c2.const_get(:Bar, false) }
+   assert_raise(NoConstantError) { c1::Bar }
+   assert_raise(NoConstantError) { c2::Bar }
+   assert_raise(NoConstantError) { c2.const_get(:Bar) }
+   assert_raise(NoConstantError) { c2.const_get(:Bar, false) }

    c1.instance_eval do
      def const_missing(x)
diff --git a/test/ruby/test_object.rb b/test/ruby/test_object.rb
index 0a49422..af91f75 100644
--- a/test/ruby/test_object.rb
+++ b/test/ruby/test_object.rb
@@ -437,7 +437,7 @@ class TestObject < Test::Unit::TestCase
    x = "foo".instance_exec("bar") {|a| self + a }
    assert_equal("foobar", x)

-   assert_raise(NameError) do
+   assert_raise(NoConstantError) do
      InstanceExec.new.instance_exec { INSTANCE_EXEC }
    end
  end

diff --git a/variable.c b/variable.c
index d7b99f1..4be8a8e 100644
--- a/variable.c
+++ b/variable.c
@@ -1341,23 +1341,49 @@ rb_obj_remove_instance_variable(VALUE obj, VALUE name)
    return Qnil; /* not reached */
  }

+extern VALUE rb_eNoConstantError;
+

```

```

+PRINTF_ARGS(NORETURN(void rb_name_error_with_class(VALUE, ID, const char*, ...)), 3, 4);
NORETURN(static void uninitialized_constant(VALUE, ID));
static void
uninitialized_constant(VALUE klass, ID id)
{
-   if (klass && klass != rb_cObject)
-   rb_name_error(id, "uninitialized constant %s::%s",
+   if (klass && klass != rb_cObject){
+   rb_name_error_with_class(rb_eNoConstantError,
+       id, "uninitialized constant %s::%s",
+       rb_class2name(klass),
+       rb_id2name(id));
+   }
-   else {
-   rb_name_error(id, "uninitialized constant %s", rb_id2name(id));
+   rb_name_error_with_class(rb_eNoConstantError,
+       id, "uninitialized constant %s", rb_id2name(id));
+   }
}

static VALUE
+const_missing_call(VALUE arg)
+{
+   VALUE *args = (VALUE *)arg;
+   ID const_missing_id;
+   CONST_ID(const_missing_id, "const_missing");
+   return rb_check_funcall(args[0], const_missing_id, 1, &args[1]);
+}
+
+static VALUE
+const_missing_rescue(VALUE arg, VALUE errinfo)
+{
+   return arg;
+}
+
+static VALUE
const_missing(VALUE klass, ID id)
{
-   return rb_funcall(klass, rb_intern("const_missing"), 1, ID2SYM(id));
+   VALUE args[2];
+   args[0] = klass;
+   args[1] = ID2SYM(id);
+   return rb_rescue2(const_missing_call, (VALUE)args,
+       const_missing_rescue, (VALUE)Qundef,
+       rb_eNoConstantError, (VALUE)0);
}

@@ -1597,7 +1623,27 @@ rb_const_get_0(VALUE klass, ID id, int exclude, int recurse)
    goto retry;
}

-   value = const_missing(klass, id);
+   tmp = klass;
+   if ((value = const_missing(tmp, id)) == Qundef) {
+   NODE *rb_vm_cref(void);
+   NODE *cref = rb_vm_cref();
+   cref = cref->nd_next;
+   while (cref && cref->nd_next &&
+       ((cref->flags & NODE_FL_CREF_PUSHED_BY_EVAL) ||
+       NIL_P(tmp = cref->nd_clss) ||
+       (value = const_missing(tmp, id)) == Qundef)) {
+       cref = cref->nd_next;
+   }
+   if (value == Qundef) {
+       if (!exclude) {
+       value = const_missing(rb_cObject, id);
+       }
+       if (value == Qundef) {
+       uninitialized_constant(klass, id);
+       }
+   }
+   }
+
+   rb_vm_inc_const_missing_count();

```

```
    return value;
  }
@@ -1655,7 +1701,7 @@ rb_const_remove(VALUE mod, ID id)
    rb_name_error(id, "cannot remove %s::%s",
    rb_class2name(mod), rb_id2name(id));
  }
- rb_name_error(id, "constant %s::%s not defined",
+ rb_name_error_with_class(rb_eNoConstantError, id, "constant %s::%s not defined",
    rb_class2name(mod), rb_id2name(id));
}
```

--
Yusuke ENDOH mame@tsq.ne.jp

#11 - 04/04/2010 01:38 AM - znz (Kazuhiro NISHIYAMA)

- Target version changed from 1.9.2 to 2.0.0

=begin

=end

#12 - 12/01/2011 10:36 AM - wycats (Yehuda Katz)

What's the status of this?

#13 - 02/20/2012 05:22 AM - tenderlovmaking (Aaron Patterson)

Bump

#14 - 10/27/2012 05:32 AM - ko1 (Koichi Sasada)

- Assignee changed from matz (Yukihiro Matsumoto) to mame (Yusuke Endoh)

Who's ball?

mame-san, could you give us the response?

#15 - 10/27/2012 10:59 AM - mame (Yusuke Endoh)

- Assignee changed from mame (Yusuke Endoh) to matz (Yukihiro Matsumoto)

It looks we need matz's approval.

Sorry I cannot remember the detailed status immediately.

--
Yusuke Endoh mame@tsq.ne.jp

#16 - 10/27/2012 12:03 PM - mame (Yusuke Endoh)

- Target version changed from 2.0.0 to 2.6

#17 - 02/12/2017 04:22 PM - najamelan (Naja Melan)

This would still be very useful for people implementing auto loading systems, not just Rails

#18 - 04/17/2017 05:58 AM - matz (Yukihiro Matsumoto)

A long time has passed. I am a bit concerned about the situation.

Do we still need this, regarding the fact I am not a big fan of autoloading?

In any case, I don't like the name "NoConstantError" which should be somehow analogous to "StopIteration" but this case, we continue, not stop.

Matz.

#19 - 10/22/2017 01:17 PM - mame (Yusuke Endoh)

- Assignee changed from matz (Yukihiro Matsumoto) to matsuda (Akira Matsuda)

I'm not sure how this feature is related to autoloading... Anyway.

Matsuda-san, do you know if Rails still needs this feature? Or do you have any opinion?

#20 - 11/29/2017 07:43 AM - matz (Yukihiro Matsumoto)
- Status changed from Assigned to Closed

The proposal has been changed from the original. I close this for now. Please re-submit the new proposal (if you want).
Matz.

Files

const_missing_nesting.diff	621 Bytes	02/13/2010	wycats (Yehuda Katz)
const_missing_murphy.patch	2.56 KB	02/16/2010	murphy (Kornelius Kalnbach)