Ruby master - Feature #12979

Avoid exception for #dup on Integer (and similar cases)

11/25/2016 07:12 AM - duerst (Martin Dürst)

Status:	Closed	
Priority:	Normal	
Assignee:	nobu (Nobuyoshi Nakada)	
Target version:		

Description

This is a proposal resulting from a discussion in Bug #11929. Because this is proposing a different solution from #11929, it has a new number.

#11929 shows that people are confused that e.g. 3.dup throws an exception (but Integer#dup is actually implemented, so Integer.respond_to? :dup => true).

Integer#dup should fail silently, returning the receiver, in the same way as Integer#freeze fails silently. Citing from #11929 (comment by Mike Vastola): "If the object can't be duped/cloned because it's an immediate, dup/clone should return the object itself. (There shouldn't be any harm in doing so since nothing about the object can be changed in the first place.)". Citing some more:

> I literally can't imagine any scenario in which a dev, when, say, coding a class with the line:
>
> return val.dup.freeze
> .. really wants an Exception thrown when val happens to be de-facto un-dup-able. What they really want is:
>
> return val.dup.freeze rescue val

The proposal also has the advantage that it leads to a much more unified, streamlined protocol, avoiding needless exposition of internals. It would do exactly what dup (and clone) are described to do, namely (pretend to) return a shallow copy.

Related issues:

Related to Ruby master - Bug #11929: No programatic way to check ability to d...	**Closed**
Related to Ruby master - Feature #13985: Avoid exception for #dup/#clone on R...	**Closed**

History

#1 - 11/25/2016 07:12 AM - duerst (Martin Dürst)

- Related to Bug #11929: No programatic way to check ability to dup/clone an object added

#2 - 11/25/2016 07:20 AM - matz (Yukihiro Matsumoto)

If we don't care about dup-ability much, the proposal sounds reasonable.
And I agree this dup-ability matters less than confusion caused by exceptions.

Accepted.

Matz.

#3 - 11/25/2016 07:22 AM - duerst (Martin Dürst)

- Assignee changed from matz (Yukihiro Matsumoto) to nobu (Nobuyoshi Nakada)

#4 - 11/25/2016 07:48 AM - nobu (Nobuyoshi Nakada)

I found that a rubygems test ,TestGemSpecification#test_initialize_copy_broken in test/rubygems/test_gem_specification.rb, depends on this exception.
And rubyspec fails as usual.

```
diff --git i/object.c w/object.c
index 05bef4d..7075e13 100644
--- i/object.c
+++ w/object.c
@@ -423,7 +423,7 @@ rb_obj_dup(VALUE obj)
     VALUE dup;

     if (rb_special_const_p(obj)) {
```

```
-        rb_raise(rb_eTypeError, "can't dup %s", rb_obj_classname(obj));
+    return obj;
     }
     dup = rb_obj_alloc(rb_obj_class(obj));
     init_copy(dup, obj);
diff --git i/test/ruby/test_object.rb w/test/ruby/test_object.rb
index 7b3defa..2f80bc6 100644
--- i/test/ruby/test_object.rb
+++ w/test/ruby/test_object.rb
@@ -19,9 +19,9 @@
   end

   def test_dup
-    assert_raise(TypeError) { 1.dup }
-    assert_raise(TypeError) { true.dup }
-    assert_raise(TypeError) { nil.dup }
+    assert_equal 1, 1.dup
+    assert_equal true, true.dup
+    assert_equal nil, nil.dup

     assert_raise(TypeError) do
       Object.new.instance_eval { initialize_copy(1) }
diff --git i/test/rubygems/test_gem_specification.rb w/test/rubygems/test_gem_specification.rb
index 87f0f36..0d51d93 100644
--- i/test/rubygems/test_gem_specification.rb
+++ w/test/rubygems/test_gem_specification.rb
@@ -1260,7 +1260,8 @@
       s.version = '1'
     end

-    spec.instance_variable_set :@licenses, :blah
+    def (broken_license = Object.new).dup; raise TypeError; end
+    spec.instance_variable_set :@licenses, broken_license
     spec.loaded_from = '/path/to/file'

     e = assert_raises Gem::FormatException do
```

#5 - 11/27/2016 12:32 PM - nobu (Nobuyoshi Nakada)

Should Kernel#clone be same?

#6 - 11/29/2016 01:58 PM - nobu (Nobuyoshi Nakada)

I remember clone has freeze option.
What should 1.clone(freeze: false) do, ignore the option, or raise an ArgumentError?

#7 - 11/29/2016 02:00 PM - matz (Yukihiro Matsumoto)

I vote for ArgumentError, because clone(freeze: false) is invalid operation for immediate (thus unfreezable) objects.

Matz.

#8 - 11/29/2016 02:52 PM - znz (Kazuhiro NISHIYAMA)

Integer (Fixnum) can dup now.
But Integer (Bignum) can't dup.

```
% ruby -ve '1.dup'
ruby 2.4.0dev (2016-11-29 trunk 56928) [x86_64-linux]
% ruby -ve '(1<<64).dup'
ruby 2.4.0dev (2016-11-29 trunk 56928) [x86_64-linux]
-e:1:in `dup': allocator undefined for Integer (TypeError)
        from -e:1:in `<main>'
```

#9 - 11/29/2016 08:04 PM - MikeVastola (Mike Vastola)

Nobuyoshi Nakada wrote:

> Should Kernel#clone be same?

The original consensus was that the fix should be for both **#dup** and **#clone** to fail silently on immediate objects. The person who created this issue rendered **#clone** as "and similar cases" in the title, perhaps assuming there could be additional places changes were necessary?

Nobuyoshi Nakada wrote:

> I remember clone has freeze option.
> What should 1.clone(freeze: false) do, ignore the option, or raise an ArgumentError?

This is totally beside the point. Look at it this way:
1.clone(freeze: false) ⇔ **1.clone**
1.clone(freeze: true) ⇔ **1.clone.freeze**

(NB: I'm not familiar with this particular invocation of **#clone**, so it's possible this argument to the method is supposed to do some sort of deep **#freeze**, but since we're dealing only with immediates here, this simplification can suffice.)

In the first line, the hash argument might as well be absent, as not freezing is the default behavior.
In the second line, you're attempting to **#freeze** an immediate. Contrary to Matz's statement, immediates aren't unfreezable, but are -- in fact -- ***always*** frozen. As a result, the **#freeze** instruction here is also silently ignored.

This entire issue is about **#dup**/**#clone**. In fact, if you read the parent issue, a major reason why the current behavior is so unintuitive is precisely because **#freeze** fails silently and returns the object unmodified, but the former two methods do not.

#10 - 11/29/2016 08:39 PM - Eregon (Benoit Daloze)

Mike Vastola wrote:

> Nobuyoshi Nakada wrote:
>
> > I remember clone has freeze option.
> > What should 1.clone(freeze: false) do, ignore the option, or raise an ArgumentError?
>
> This is totally beside the point.

1.clone(freeze: false) does not do what you say.
The keyword :freeze has default value true for #clone, which means if the original object is frozen so will be the clone.
See #12300 for details.

Since freeze: false that can be considered as an explicit request to get a clone/copy, to modify it and re-freeze it later, it seems reasonable to raise in such a strange case. But, no harm could be done to immediate always-frozen objects anyway, so it would also be harmless to ignore the option IMHO.

OTOH I agree 1.clone(freeze: false) is confusing and the behavior not intuitive from the syntax/names.
Maybe 1.clone(freeze_clone: false) or 1.clone(propagate_freeze: false) or so would be clearer. But this should be discussed in #12300 then.

#11 - 11/29/2016 08:59 PM - MikeVastola (Mike Vastola)

Benoit Daloze wrote:

> 1.clone(freeze: false) does not do what you say.
> The keyword :freeze has default value true for #clone, which means if the original object is frozen so will be the clone.
> See #12300 for details.

Oh wow. That's entirely my fault. I totally confused Ruby's dup/cloning with another language (and/or my imagination, haha).

Fortunately my point still stands: the **:freeze** key (and the entire **#freeze** operation) of the **#clone** method is irrelevant vis-à-vis immediates and **#freeze** is already designed to fail silently on **#frozen?** objects. (I think I just got the default behavior wrong.)

#12 - 12/17/2016 02:23 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

#13 - 12/17/2016 09:14 AM - MikeVastola (Mike Vastola)

> Status changed from Open to Closed

So I apologize if this is a stupid question, but does this mean this change has been implemented now?

If so, is there a reference to a particular commit somewhere on this ticket?

#14 - 12/19/2016 02:52 PM - znz (Kazuhiro NISHIYAMA)

Does this feature mention in NEWS file?

#15 - 12/22/2016 07:36 PM - kaspth (Kasper Timm Hansen)

In Rails we've circumvented this with a duplicable? core extension.

We'd need dup support on Method, Complex and Rational too remove it entirely:
https://github.com/rails/rails/blob/fd63aa02289d64e9d14fe56723f1de64bca3bb1f/activesupport/lib/active_support/core_ext/object/duplicable.rb#L115-L143

Would that be possible too? Thanks :)

#16 - 12/22/2016 08:45 PM - MikeVastola (Mike Vastola)

So, looking at this more closely, I'm beginning to think this issue was closed in error.

For starters, while all the "Associated revisions" are to object.c, the most recent one was 23 days ago, after which there have been discussions about the implementation, with no follow-up comments or commits from the assigned coder.

Additionally, though the most recent commit was tagged 2.4.0 RC1, I'm not seeing any mention of this issue in NEWS or ChangeLog for 2.4.0.

Given this, as well as Kasper's comment, I'd like to go ahead and ask that this be reopened (does anyone know who I should ask?), in the hopes of (at the very least) getting clarification as to whether or not this is completed (though ideally to see this fully implemented if it truly is not).

Happy Holidays!

#17 - 12/24/2016 04:13 AM - duerst (Martin Dürst)

Mike Vastola wrote:

> So, looking at this more closely, I'm beginning to think this issue was closed in error.

The original issue has indeed been fixed, and so 'closed' is correct:

```
$ ruby -v
ruby 2.4.0dev (2016-12-24 trunk 57168) [x86_64-cygwin]

$ ruby -e 'puts 1.dup'
1
```

The main fix was at r56906. If you think additional improvements are needed, please open a new issue (and reference this one).

#18 - 12/24/2016 05:25 AM - MikeVastola (Mike Vastola)

Sorry.. I'm not used to Redmine and I was confused because there was no conclusion to the discussion about the **ArgumentError**.

I understand that Casper's request should be a separate issue (I'll leave him to submit that since I trust he can argue it better than I.) but I think the raising of an **ArgumentError** constitutes a bug in this fix (I guess it's technically a regression if we're calling this a fix) and warrants further discussion. Given that it concerns the people active on this issue, should it still be discussed in a separate issue?

If necessary, I'll copy the following into a separate issue, but...

ArgumentError#message is **"can't unfreeze %s"**, but the inline documentation for **Object#clone** doesn't state that setting the **:freeze** key of the option **Hash** to **false** affirmatively causes anything to happen at all. Quite the opposite, in fact. It states that the frozen state is copied if (and only if) the **:freeze** option is either **nil** or **true**.

If **Object#clone** is called with the **:freeze** key of the option **Hash** set to **false**, intuitively, the method should behave identically to **Object#dup**. In other words, there should be no error triggered. The only difference *would* be (as is documented) in the case of descendant classes, but since the classes of these special constants are themselves immutable (and don't have singletons), there is no way to create a descendant class for those.

I understand the theory behind doing this: people who already have this code in place may (depending on usage) expect all objects produced by it to be freely mutable. IMHO, however, this is short-sighted. This prioritizes backwards-compatibility over continuity and intuitiveness. If this is truly a concern, I think printing a warning rather than raising an error might be a sensitive alternative.

We shouldn't look at this simply in terms of the use cases of this code currently in the wild now, however. Once this feature is widely available, it's possible people will still want to use **Object#clone** with **:freeze** set to **false** (due to it's different handling of descendant classes for objects that are thusly altered) while silently passing through immediates.

#19 - 12/24/2016 07:02 AM - duerst (Martin Dürst)

Mike Vastola wrote:

> Sorry.. I'm not used to Redmine and I was confused because there was no conclusion to the discussion about the **ArgumentError**.

There was a conclusion at https://bugs.ruby-lang.org/issues/12979#note-7.

> If necessary, I'll copy the following into a separate issue, but...

Yes, please. Thanks.

#20 - 10/06/2017 08:40 PM - Eregon (Benoit Daloze)

I believe this should make it to the NEWS file on the 2_4 branch, could somebody do it?

#21 - 10/06/2017 09:00 PM - Eregon (Benoit Daloze)

- Related to Feature #13985: Avoid exception for #dup/#clone on Rational and Complex added

#22 - 10/11/2017 03:42 PM - znz (Kazuhiro NISHIYAMA)

I think no one is doing it.
Patches welcome.