Ruby - Feature #19712

IO#reopen removes singleton class

06/05/2023 03:07 PM - itarato (Peter Arato)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

The documentation states:

> This may dynamically change the actual class of this stream.

As well #reopen removes the singleton class, even when the logical class is the same. This can be surprising at times.

An example:

```
io = File.open(__FILE__)
io.define_singleton_method(:foo) {}
io.reopen(File.open(__FILE__))
io.foo # `<main>': undefined method `foo' for #<File:/test.rb> (NoMethodError)
```

An example where this was an issue: https://github.com/oracle/truffleruby/pull/3088 Tl;dr: a popular gem was enhancing the singleton class of STDOUT, while Rails/ActiveSupport was executing an IO#reopen - invalidating the changes by the gem.

While it's hard to trivially solve this with keeping the intended functionality, could it be considered to make edge cases more visible?

Examples:

- IO#reopen issues a warning message when the receiver's singleton class has been updated (but keep removing it as of current state)
- IO#reopen is limited on instances with a default singleton class, and issues an error when called on an instance with an updated singleton class
- make IO#reopen carry over the singleton class *only* if the recipient and argument's class are the same, and yield an error otherwise (different classes)

These are just ideas. I'm more looking for feedback from the core devs at this point. Thanks in advance!

History

#1 - 06/05/2023 03:08 PM - itarato (Peter Arato)

- Description updated

#2 - 06/07/2023 09:50 AM - byroot (Jean Boussier)

For what it's worth, I think:

- It would seem unnatural to me to not clear the singleton_class, as in my mental model at least, the singleton_class is the actual class of an object.
- I think it would be even more error prone if it was sometimes cleared sometimes not.

This however raise the question of whether the singleton_class should be copied in such case:

```
f = File.open('/tmp/debug.txt', "w+")
def f.foo = 42
STDERR.reopen(f)
STDERR.foo # 42 or NoMethodError ?
```

#3 - 06/07/2023 02:56 PM - Eregon (Benoit Daloze)

@byroot (Jean Boussier) It's not possible to reuse the singleton class for that example. A singleton class has a given superclass, and that must never change.
But I suppose it might be possible to *copy* the singleton class, although that might be surprising as now 2 classes have that singleton method and they are independent.

It's existing behavior though, as Kernel#clone already does this.

I think these changing-class semantics of IO#reopen have no place in Ruby, it's not worth breaking the object model so badly for a single method.

So maybe IO#reopen should never change the class.
That would mean probably failing then for e.g. some_File.reopen(some_Socket), but some_IO.reopen(some_Socket) could be OK.
STDOUT, STDERR and STDIN are all IO initially, so reopen would still work on them but they would stay IO instances, no matter the argument's class.

Of course it's possible to assign $stdout/$stderr/$stdin (or even STDOUT/STDERR/STDIN if one wishes), instead of using IO#reopen.
So then maybe we could deprecate the whole method. But that's likely more difficult for compatibility.

#4 - 06/07/2023 03:01 PM - byroot (Jean Boussier)

> It's not possible to reuse the singleton class for that example.

Yes, that's why I wrote: "whether the singleton_class should be copied", same semantic than #clone

#5 - 06/08/2023 08:05 AM - matz (Yukihiro Matsumoto)

The #reopen has special semantics that re-initialized the IO object. As a result, singleton methods will be wiped out.
So I propose to update the document to describe regarding singleton classes (and it may change the class of the receiver).

Matz.

#6 - 06/08/2023 09:01 AM - Eregon (Benoit Daloze)

@matz (Yukihiro Matsumoto) (Yukihiro Matsumoto) wrote in #note-5:

> The #reopen has special semantics that re-initialized the IO object. As a result, singleton methods will be wiped out.

Yes, that's the current semantics.
I think it's both confusing and breaking the general guarantee that a Ruby object's class/singleton class (RBASIC_CLASS(obj) to be precise) does not change.

I think there is no need for IO#reopen to change the class, but we need a migration path.
I think fundamentally IO#reopen is a way to change the file descriptor used for that IO, that doesn't need to change the class.
And suppose STDOUT.reopen(some_Socket), it's very unlikely someone would use Socket-specific methods on STDOUT, so I think we also don't need the methods of the argument's class in most cases.
So I think we could change the semantics of IO#reopen to not change the class, and I think it would still just work in most cases (and we could adapt the rare cases that do rely on changing the class).

#7 - 06/08/2023 09:42 AM - matz (Yukihiro Matsumoto)

We still have concerns about compatibility.

One is keeping the receiver's class, and the other is mixing of file descriptors and sockets (especially on Windows).
Both can be the cause of issues that cannot be solved easily.

We have discussed some possible solutions, for example:

- use dup2 instead of copying the IO object (may not work with sockets)
- prohibit #reopen if the replacement's class does not have direct inheritance relationship (may prohibit current working programs)

But none of them are free from compatibility issues.

I don't think it's worth the incompatibility and additional complexity.

Matz.

#8 - 06/08/2023 11:55 AM - Eregon (Benoit Daloze)

@matz (Yukihiro Matsumoto) What if I made a PR to not change the class for IO#reopen, and we commit it experimentally, and if it causes issues in multiple gems/apps, we revert it?
Would it be OK to merge that experimentally?

#9 - 06/08/2023 11:59 AM - Eregon (Benoit Daloze)

To clarify, my main motivation to solve this issue is to have a nicer object model (the Kernel#class of an object never changes, only a singleton class can be added to an object, never removed) and to avoid surprises when defining singleton methods on IO objects which are then IO#reopen'ed later.

#10 - 06/09/2023 09:47 AM - akr (Akira Tanaka)

I proposed adding a keyword argument as io1.reopen(io2, retain_class:true) at the meeting.
The keyword argument retain_class:true disables changing the class.
That means io1.reopen(io2, retain_class:true) just do dup2(io2.fileno, io1.fileno).
This extension is compatible, as far as retain_class:false is the default.
If we don't need to use io2-specific methods for io1, we don't need to change the class of io1, and retain_class:true should work fine.
(Singleton methods of io1 will retain.)

matz said it is not worth enough over his concern about possible incompatibility.
As far as I understand his concern, he is afraid that IO methods may not work for file descriptors for sockets on Windows.
Although I'm not a Windows expert, I guess IO methods work for sockets.
At least read and write methods are inherited to TCPSocket and they work fine now.

```
% ruby -rsocket -e 'p TCPSocket.method(:read), TCPSocket.method(:write)'
#<Method: TCPSocket(IO).read(*)>
#<Method: TCPSocket(IO).write(*)>
```