Ruby - Feature #12624

!== (other)

07/24/2016 10:02 PM - eike.rb (Eike Dierks)

Status:	Rejected
Priority:	Normal
Assignee:	
Target version:	

Description

I'd like to suggest a new syntactic feature.

There should be an operator !==
which should just return the negation of the === operator

aka:

```
def !==(other)
     ! (self === other)
end
```

Rationale:

The === operator is well established.
The !== operator would just return the negated truth value of ===
That syntax would mimick the duality of == vs !=

Impact:

To my best knowledge, !== is currently rejected by the parser,
so there should be no exsiting code be affected by this change.

Do we really need that?

obviously (! (a === b)) does the job,
while, (a !== b) looks a bit more terse to me.

What's the use case?

I personally got a habit of using === in type checking arguments:

```
raise TypeError() unless (SomeClass === arg)
```

You might argue that I should write instead:

```
raise TypeError() unless arg.kind_of?(SomeClass)
```

(you are obviously right in that)

But the === operator is there for a reason,
and it is actually a strong point of ruby,
that we do not only have identity or equivalence,
but this third kind of object defined equality.

I believe, that in some cases
the intention of a boolean clause
would be easier to understand if we had that !== operator
instead of writing !(a===b)

I agree, syntax ahould not change.
But I believe that would add to the orthogonality.

Please see also:
my request on reserving the UTF operator plane for operators

History

#1 - 07/25/2016 01:18 AM - duerst (Martin Dürst)

Eike Dierks wrote:

> I believe, that in some cases
> the intention of a boolean clause
> would be easier to understand if we had that !== operator
> instead of writing !(a===b)

We usually don't add new features to Ruby just based on 'belief'. If you think there are such use cases, please find them, in actual existing code.

#2 - 07/26/2016 01:55 AM - nobu (Nobuyoshi Nakada)

- Description updated

I'm sometimes wanting it, too.

And can find some lines in standard libraries.

```
ext/psych/lib/psych/visitors/yaml_tree.rb:334:        elsif not String === @ss.tokenize(o) or /\A0[0-7]*[89]/ =~ o
lib/irb.rb:500:                !(SyntaxError === exc)
lib/optparse.rb:1353:      if (!(String === o || Symbol === o)) and o.respond_to?(:match)
lib/rdoc/class_module.rb:777:      !(String === mod) && @store.modules_hash[mod.full_name].nil?
lib/rdoc/class_module.rb:793:      !(String === mod) && @store.modules_hash[mod.full_name].nil?
lib/rdoc/parser/ruby.rb:244:        break if first_comment_tk_class and not first_comment_tk_class === tk
lib/resolv.rb:534:             if reply.tc == 1 and not Requester::TCP === requester
lib/resolv.rb:1028:                  !(Array === ns_port) ||
lib/resolv.rb:1030:                  !(String === ns_port[0]) ||
lib/resolv.rb:1031:                  !(Integer === ns_port[1])
lib/rubygems/security/signer.rb:51:      @key and not OpenSSL::PKey::RSA === @key
test/objspace/test_objspace.rb:76:    assert_empty(arg.select {|k, v| !(Symbol === k && Integer === v)}, bug8014)
test/rinda/test_rinda.rb:212:    assert(!(tmpl === ro))
test/rinda/test_rinda.rb:218:    assert(!(tmpl === ro))
test/rinda/test_rinda.rb:221:    assert(!(tmpl === ro))
test/rinda/test_rinda.rb:230:    assert(!(tmpl === ro))
test/ruby/test_m17n_comb.rb:1131:      if [s, *args].all? {|o| !(String === o) || o.valid_encoding? }
```

https://github.com/ruby/ruby/compare/trunk...nobu:feature/!==

#3 - 07/27/2016 06:04 PM - shevegen (Robert A. Heiler)

I don't have any particular strong pro or con opinion here, but I should like to note that my bad eyes have it not so easy to distinguish between = == != =! !== ==!.

I actually think that !(String === mod) may be easier to read than (String !== mod) - the amount of characters saved is very negligible.

But it is just an opinion, as said, I have neither strong pro or con opinion on it really.

#4 - 08/09/2016 02:33 PM - matz (Yukihiro Matsumoto)

- Status changed from Open to Rejected

The explicit use of === for type checking is against duck typing principle.
I don't accept syntax enhancement proposal to encourage something against duck typing in Ruby.

Matz.

#5 - 02/06/2020 08:00 PM - jonathanhefner (Jonathan Hefner)

Recently, I had a use case for this. I was writing an assertion helper method which accepts a comparison operator (e.g. :==, :!=, :===, etc) to send to the expected value. For my use case, having !== would be nice for a few reasons:

- Can express "assert not expected === actual" without the need for a "refute" method
- If defining a "refute" method, can implement it in terms of "assert" using operator inversion lookup table, i.e. { :== => :!=, :=== => :!==, :< => :>=, ... }

- Error messages can be expressed without special casing, i.e. "Expected: #{expected.inspect} #{op} #{actual.inspect}"