Ruby trunk - Feature #12515

Create "Boolean" superclass of TrueClass / FalseClass

06/23/2016 01:13 AM - lsegal (Loren Segal)

Status:	Rejected
Priority:	Normal
Assignee:	
Target version:	

Description

Since Ruby 2.4 is unifying Bignum/Fixnum into Integer (https://bugs.ruby-lang.org/issues/12005), it seems reasonable to do something similar for TrueClass / FalseClass, and create a proper Boolean hierarchy. The implementation would be fairly straightforward and should be back compat unless someone already has a "Boolean" class at toplevel out there.

Given the compatibility implications of Integer, this Boolean proposal is even less intrusive.

Sample implementation:

```
class Boolean < BasicObject; end
class TrueClass < Boolean; end
class FalseClass < Boolean; end
```

Related issues:

Has duplicate Ruby trunk - Bug #12827: Add Boolean data type.	**Rejected**

History

#1 - 06/23/2016 01:47 AM - shyouhei (Shyouhei Urabe)

Loren Segal wrote:

> should be back compat unless someone already has a "Boolean" class at toplevel out there.

You failed here.

https://github.com/search?l=ruby&q=%22class+Boolean%22&type=Code
https://rubygems.org/search?query=Boolean

I guess it's too late.

#2 - 06/23/2016 02:10 AM - olivierlacan (Olivier Lacan)

Shyouhei Urabe wrote:

> You failed here.

This admittedly could be lost in translation but I don't think this a very nice way to react to Loren's proposal. He didn't fail here. He just proposed something. You either think it's a good idea or a bad idea. I wish you could provide your opinion a bit more kindly.

> https://github.com/search?l=ruby&q=%22class+Boolean%22&type=Code
> https://rubygems.org/search?query=Boolean
>
> I guess it's too late.

I can change the search query to Integer and provide similarly dubious reasons why Integer would be bad idea:

https://github.com/search?l=ruby&q=%22class+Integer%22&type=Code
https://rubygems.org/search?query=Integer

I don't think this is very solid counterpoint.

#3 - 06/23/2016 02:24 AM - lsegal (Loren Segal)

Shyouhei Urabe wrote:

> https://github.com/search?l=ruby&q=%22class+Boolean%22&type=Code

> https://rubygems.org/search?query=Boolean
>
> I guess it's too late.

As Olivier pointed out, isn't this an argument to revert the Integer patch? Why is the Integer change okay but not Boolean?

#4 - 06/23/2016 02:51 AM - lsegal (Loren Segal)

edit: as a sidenote, adding a Boolean class to Ruby-core is unlikely to break any of the linked code, because Ruby is just defining an empty Boolean class. Existing libraries will just re-open the class as normal.

#5 - 06/23/2016 04:24 AM - djberg96 (Daniel Berger)

This was brought up in the past a lot:

http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-talk/96321

http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-talk/186881

There's also this: http://stackoverflow.com/questions/3192978/why-does-ruby-have-trueclass-and-falseclass-instead-of-a-single-boolean-class

#6 - 06/23/2016 04:46 AM - shyouhei (Shyouhei Urabe)

- I'm not a big fan of Integer, too. However,
- Integer has been there since the beginning. It was just not used before. You are always free to reopen this class. Boolean on the other hand, is something new. The situation is different.
- Because most wild Boolean implementation I have seen inherits Object (or depending on project ActiveRecord::Base), I'm sorry I guess Loren's suggested implementation that inherits BasicObject won't interface.

#7 - 06/23/2016 05:25 AM - shugo (Shugo Maeda)

At http://qiita.com/yatemmma/items/3597a0e20556fb846d3f, Matz explained the reason why Ruby doesn't have Boolean.

- true and false are just representative samples of boolean values.
- There is no method to be defined in Boolean (e.g., implementations of TrueClass#& and FalseClass#& are different).
- The use case of Boolean is therefore limited to kind_of? checks, and it prevents duck typing.

However Boolean might be useful to extend true and false in user code (e.g., refining Boolean).
Another use case is type checking system, but it might not be a good idea to consider a class a type.

#8 - 06/23/2016 07:21 AM - lsegal (Loren Segal)

In reality it's very common to treat true/false as a separate thing. The "everything is boolean" argument falls apart when you have something like var args in a method declaration and you want to perform distinct operations based on the actual value.

This actually comes up fairly often:

https://github.com/search?p=2&q=is_a%3F+TrueClass&ref=searchresults&type=Code&utf8=%E2%9C%93

And that's only the checks for is_a? syntax, because GitHub cannot search for "TrueClass === x" style code. But if it's an anti-pattern, it's an extremely common one.

I didn't realize about Integer being around though, that certainly changes things.

#9 - 06/24/2016 07:23 AM - duerst (Martin Dürst)

Loren Segal wrote:

> This actually comes up fairly often:
>
> https://github.com/search?p=2&q=is_a%3F+TrueClass&ref=searchresults&type=Code&utf8=%E2%9C%93
>
> And that's only the checks for is_a? syntax, because GitHub cannot search for "TrueClass === x" style code. But if it's an anti-pattern, it's an extremely common one.

The search results aren't very convincing to me. Where there's a Boolean class or module, it's usually really tiny. And there are checks for TrueClass and FalseClass that I'd write as checks for true and false, which would make the code a lot shorter.

Actually, because both true and false are singletons, TrueClass and FalseClass look like overkill; in Ruby's ducktyping world, most if not all things would work just as well with the necessary methods for true and false being defined directly as singleton methods on true and false.

#10 - 06/24/2016 03:12 PM - nobu (Nobuyoshi Nakada)

Let's leave TrueClass/FalseClass!!

#11 - 07/02/2016 10:14 AM - sawa (Tsuyoshi Sawada)

Martin Dürst wrote:

> Actually, because both true and false are singletons, TrueClass and FalseClass look like overkill; in Ruby's ducktyping world, most if not all things would work just as well with the necessary methods for true and false being defined directly as singleton methods on true and false.

My understanding is that defining a singleton method on an object creates a singleton class for that object (and the method is actually an instance method of the singleton class). And in case of true, false or nil, the singleton classes are in fact the classes in question:

```
true.singleton_class # => TrueClass
false.singleton_class # => FalseClass
nil.singleton_class # => NilClass
```

so there is no way to get rid of these classes. We don't need to worry as

Nobuyoshi Nakada wrote:

> Let's leave TrueClass/FalseClass!!

And by the way for the Boolean class proposed in this thread, it surely is un-useful.

#12 - 07/19/2016 08:29 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Rejected

Rejected for several reasons:

- many gems and libraries had already introduced Boolean class. I don't want to break them.
- true and false are the only representative of true-false values. In Ruby. nil and false are falsy values, and everything else is a true value. There's no meaning for having a superclass of TrueClass and FalseClass as Boolean.

Matz.

#13 - 10/11/2016 02:30 AM - shyouhei (Shyouhei Urabe)

- Has duplicate Bug #12827: Add Boolean data type. added

#14 - 10/30/2016 11:00 PM - ioquatix (Samuel Williams)

The fact that so many gems are introducing "class Boolean" is an argument FOR it not AGAINST it, IMHO. Because when this code is loaded together, it might behave strangely if there is no shared meaning for "class Boolean".

Having a Boolean(String) constructor would be useful.

Having a class named Boolean would make things more readable, for example here: http://sequel.jeremyevans.net/rdoc/files/doc/schema_modification_rdoc.html#label-Column+types - you can see that because there is no Boolean class, they resort to, IMHO quite a strange naming convention, using either TrueClass or FalseClass.

The naming of TrueClass and FalseClass are also inconsistent with other names, e.g. it's not IntegerClass or FloatClass or StringClass. It's a little bit ugly. (EDIT: or ZeroClass, OneClass, etc :)

> There's no meaning for having a superclass of TrueClass and FalseClass as Boolean

I believe you are wrong on this point. There is meaning.

The meaning is that "This variable is of class Boolean".

There is one example I can think of:

```
#!/usr/bin/env ruby

x = true
# x = false

case x
    when TrueClass
        puts "trueclass"
    when FalseClass
        puts "falseclass"
```

```
    # How can we implement this?
    # when Boolean
end
```

Situations were this kind of logic comes up include serialisation and deserialisation libraries, data modelling, etc.

#15 - 10/30/2016 11:35 PM - phluid61 (Matthew Kerwin)

Samuel Williams wrote:

> The fact that so many gems are introducing "class Boolean" is an argument FOR it not AGAINST it, IMHO. Because when this code is loaded together, it might behave strangely if there is no shared meaning for "class Boolean".

But that's how things already are, and the market deals with it just fine. Adding a core 'Boolean' class means every existing implementation, even those only ever used in isolation, will suddenly have a conflict.

> > There's no meaning for having a superclass of TrueClass and FalseClass as Boolean
>
> I believe you are wrong on this point. There is meaning.
>
> The meaning is that "This variable is of class Boolean".
>
> There is one example I can think of:
>
> ```
> #!/usr/bin/env ruby
>
> x = true
> # x = false
>
> case x
>   when TrueClass
>       puts "trueclass"
>   when FalseClass
>       puts "falseclass"
>   # How can we implement this?
>   # when Boolean
> end
> ```
>
> Situations were this kind of logic comes up include serialisation and deserialisation libraries, data modelling, etc.

If that was:

```
case x
when Boolean
  #do something
end
```

What "something" would you do that is identical for both True and False? Or put another way, what method would you define inside class Boolean that is useful to both True and False? This comes back to one of Ruby's basic OO-tenets: that classes have shared *functionality*, not a shared *type*.

#16 - 10/31/2016 01:14 AM - ioquatix (Samuel Williams)

> But that's how things already are, and the market deals with it just fine.

Well, I know you are being metaphorical ("market deals with it"). But this is not really a case of a market, and I prefer DRY for many reasons. There should be one correct implementation and it makes sense for it to be within Ruby.

To be explicit about it: If I include two gems, and they both define "class Boolean", and they collide for some reason, it's a problem for me, and it might not even be an obvious problem.

> What "something" would you do that is identical for both True and False?

Sometimes you need to handle multiple data types this way because the functionality is on core classes that you don't want to extend. It's similar to how type classes work in Haskell.

So, as a basic example, sometimes you want to be declarative about, say, an API:

```
format_response(name: String, age: Integer, happy: Boolean)
```

Another example of this would be libraries like optparse, slop, sequel, etc, where they'd like to declaratively specify that something is a Boolean, e.g.

```
option '--foo', type: Boolean

# or

create_table :foo do |t|
    t.happy type: Boolean
end
```

Another use case would be where you want to do validation, conversion or deal with a type as specified by the user, e.g.

```
case @type
when Integer
    Integer(result)
when Float
    Float(result)
when Boolean
    Boolean(result)
end
```

Unfortunately in Ruby, the type constructors (Kernel.Integer) are free functions, not part of the class, e.g you can't write Integer.parse(input). You need to write (verbosely on purpose) Kernel.Integer(input). It's not very OO.

Another case would be handling the above, e.g. Boolean.parse(input) or Kernel.Boolean(input). It would make sense if they could turn the string representations into boolean values and throw an ArgumentError if not convertable.

#17 - 10/31/2016 03:35 AM - ioquatix (Samuel Williams)

It's also interesting to note, so many people are working around this:

Searching on github for:

"is_a? TrueClass FalseClass" gives 90,000 results.

https://github.com/search?p=2&q=is_a%3F+TrueClass&ref=searchresults&type=Code&utf8=

"module Boolean TrueClass FalseClass" gives 63,000 results.

https://github.com/search?utf8=&q=module+Boolean+TrueClass+FalseClass&type=Code&ref=searchresults

Even I'm surprised by the number of people working around this issue!

#18 - 10/31/2016 03:51 AM - rrroybbbean (RRRoy BBBean)

The terms "truthy" and "falsey" were used by Douglas Crockford in his series of JavaScript lectures at Yahoo about 5 years ago. A value is "truthy" if treated as true by a conditional. A value is "falsey" if treated as false by a conditional.

Ruby has possibly the simplest, cleanest and easiest to use distinction between "truthy" and "falsey" values of any programming language: nil and false are falsey, everything else is truthy.

Compare Ruby's to PHP's or JavaScript's allocation of truthy and falsey, where empty strings and empty arrays and even zero can be falsey. If 0 is falsey, should 0.0 be false? What about rounding errors? How about complex numbers? expressed in rectangular notation? expressed in polar notation? How about multidimensional arrays that contain no values other than arrays? How about empty dictionaries? Should they truthy or falsey? The richer the language and it's data structures, the more vexing these issues become.

Ruby has reduced the confusion to a minimum by keeping the falsey values to a minimum: false and nil. Everything else is truthy. And Ruby accomplishes this without a Boolean class. Adding a Boolean class to Ruby is not necessary, and it would be profoundly inelegant.

I quote the definition of elegance provided by Alex W. White in his 'The Elements of Graphic Design': "Distill the essential from the mass of confusing muchness. Nothing may be missing and nothing may be extraneous. This is the definition of elegance."

There is also a well-know translation of a quotation by the famous French pioneer in aviation Antoine de Saint-Exupéry (as translated by

Lewis Galantière): "... perfection is finally attained not when there is no longer anything to add, but when there is no longer anything to take away ..."

Please don't add a Boolean class to Ruby; it doesn't need a Boolean class.

#19 - 10/31/2016 04:32 AM - phluid61 (Matthew Kerwin)

Samuel Williams wrote:

> > But that's how things already are, and the market deals with it just fine.
>
> Well, I know you are being metaphorical ("market deals with it"). But this is not really a case of a market, and I prefer DRY for many reasons. There should be one correct implementation and it makes sense for it to be within Ruby.

It's not a metaphor, it's the real world. Ruby gems exist in open competition, and developers have the opportunity to choose between (or create alternatives to) any available gems in the ecosystem.

If there should be one "correct" implementation, why do we have gems at all? *Everything* should be in the core. And that's without even starting the big argument about which approach should be considered "correct." (Anyone who says "N" is falsy-y in all cases is going to have some serious arguing ahead of them.)

> To be explicit about it: If I include two gems, and they both define "class Boolean", and they collide for some reason, it's a problem for me, and it might not even be an obvious problem.

Indeed. That goes back to an open marketplace and market forces. You say, "If I do this..." but I say, "People already live with that possibility and seem to be surviving just fine."

> > What "something" would you do that is identical for both True and False?
>
> Sometimes you need to handle multiple data types this way because the functionality is on core classes that you don't want to extend. It's similar to how type classes work in Haskell.
>
> *[snip]*

Everything you've listed there is a *type*. Ruby doesn't do types like that. You're conflating the class hierarchy with types.

> Unfortunately in Ruby, the type constructors (Kernel.Integer) are free functions, not part of the class, e.g you can't write Integer.parse(input). You need to write (verbosely on purpose) Kernel.Integer(input). It's not very OO.

They're not constructors; Integers (well, Fixnums) are singleton and cannot be instantiated. The methods you're talking about are for coercion. (And very specific coercion at that.)

If you want to talk about OO, the casting methods (e.g. #to_i) are defined on the object-to-be-recast, or an intermediate expert. The destination class doesn't *and shouldn't* have responsibility to understand every possible class or type of object, and know how to convert them all to an instance of itself. There are all sorts of anti-patterns in there. (Said another way: a String should know whether it represents a 'true' or 'false' value, or there should be an object that knows just enough about Strings and booleans to be able to manage the conversion.)

> Another case would be handling the above, e.g. Boolean.parse(input) or Kernel.Boolean(input). It would make sense if they could turn the string representations into boolean values and throw an ArgumentError if not convertable.

I posit that it should be something like String#to_bool, since the String is the thing that knows whether and how it represents a boolean value.

There is no value in defining a Boolean superclass of both True and False.

#20 - 10/31/2016 06:24 AM - nobu (Nobuyoshi Nakada)

Samuel Williams wrote:

> Having a class named Boolean would make things more readable, for example here: http://sequel.jeremyevans.net/rdoc/files/doc/schema_modification_rdoc.html#label-Column+types - you can see that because there is no Boolean class, they resort to, IMHO quite a strange naming convention, using either TrueClass or FalseClass.

It's a choice by that library author.
They are not classes but methods, there can be Boolean method in the library.
Not providing such method is the choice by the author.

> The naming of TrueClass and FalseClass are also inconsistent with other names, e.g. it's not IntegerClass or FloatClass or StringClass. It's a little bit ugly. (EDIT: or ZeroClass, OneClass, etc :)

Indeed.
We don't need the names for singleton classes of true, false, and nil, and should remove them all.

#21 - 10/31/2016 02:50 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

I'd just like to add a real situation where I missed a Boolean class recently.

A web request takes an array of values in a JSON encoded string and before processing the data the server-side handler will try to validate the input by comparing the elements type. If Boolean existed, the validation rule would be something like:

```
input = JSON.parse(params['value'])
raise InvalidInput, 'size must be 6' unless input.size == 6
raise InvalidInput, "types don't match" unless input.map(&:class) ==
  [String, String, String, Boolean, String, Boolean]
```

As you can imagine, the validation took more effort without the existence of Boolean. Something like:

```
input.map{|x| x == false || x == true ? :boolean : x.class } ==
  [String, String, String, :boolean, String, :boolean]
```

#22 - 10/31/2016 02:51 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Actually, this isn't a good example as it would return TrueClass or FalseClass rather than Boolean...

#23 - 10/31/2016 02:53 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Maybe something like:

```
input.zip([String, String, String, Boolean, String, Boolean]).all?{|(i, c)| c === i }
```

#24 - 10/31/2016 11:08 PM - ioquatix (Samuel Williams)

> It's a choice by that library author. They are not classes but methods, there can be Boolean method in the library.

Nobu, you are not looking deep enough.

https://github.com/jeremyevans/sequel/blob/c39594dd35b3f8a8975b05be156ba664f1630804/lib/sequel/database/schema_generator.rb#L57-L61

The author makes methods to produce the types. The type name must be the same. My feeling is that if a Boolean type existed, they would prefer it.

> "Distill the essential from the mass of
> confusing muchness. Nothing may be missing and nothing may be
> extraneous. This is the definition of elegance."

Yes, and if you apply this logic, why have two classes when one would be better in every way?

> There is no value in defining a Boolean superclass of both **TrueClass and FalseClass**.

There is value in the check x.is_a? Boolean. The value is in the check itself, finding out if x contains [true, false]. And the value is in the consistency that there is a single class name "Boolean which is polymorphic over the set of classes Integer, Float, Numeric, String, etc, which are used over and over again in code which does formatting, type checking, serialisation, deserialisation, database modelling, etc.

The evidence is overwhelming. There are 150,000 instances of code working around the lack of Boolean on GitHub. At what point do you draw the line and say "Yeah, enough people are doing this, it might make sense to bring this into the core language".

No other language structures booleans around a "TrueClass" and "FalseClass":

- Python has bool data type.
- C has #include <stdbool.h>
- C++ has bool
- Java has bool and class Boolean
- JavaScript has Boolean
- Smalltalk has Boolean
- CLISP has BOOLEAN

> but I say, "People already live with that possibility and seem to be surviving just fine."

Is Ruby a language just to survive or is it a language to provide the tools programmers need to be happy?

I say, that every person who wrote their own Boolean class, wasted an hour of their time trying to figure it out. On GitHub, there are 2,080,658 results for "class Boolean". I know it's an overestimate, but that gives us an upper bound of 228+ years of programmer time. Matz, how can you live with yourself :D (purely for comic effect, honestly we all love you).

#25 - 10/31/2016 11:48 PM - phluid61 (Matthew Kerwin)

Samuel Williams wrote:

> > There is no value in defining a Boolean superclass of both **TrueClass and FalseClass**.
>
> There is value in the check x.is_a? Boolean. The value is in the check itself, finding out if x contains [true, false]. And the value is in the consistency that there is a single class name "Boolean which is polymorphic over the set of classes Integer, Float, Numeric, String, etc, which are used over and over again in code which does formatting, type checking, serialisation, deserialisation, database modelling, etc.

You keep saying things like "polymorphic," but you still haven't answered the question which is fundamental to this whole debate: what method would you define on class Boolean that applies identically to both TRUE and FALSE?

An object's .class is **not the same** as its type.

If they never quack the same, they aren't ducks. A class that doesn't define a #quack method is not a useful class, in the Ruby sense.

> No other language structures booleans around a "TrueClass" and "FalseClass":
>
> - Python has bool data type.
> - C has #include <stdbool.h>
> - C++ has bool
> - Java has bool and class Boolean
> - JavaScript has Boolean
> - Smalltalk has Boolean
> - CLISP has BOOLEAN

As has been mentioned already, TrueClass and FalseClass (and NilClass) are just named eigenclasses of the objects TRUE and FALSE (and NIL). There's no "structure" around them beyond the fact that *every* object (including the three listed above) has one; these three just happen to have a little quirk in their inheritance.

Perhaps it would be better if TRUE.class (etc.) returned Object; being an instance of the eigenclass is rather confusing.

> > but I say, "People already live with that possibility and seem to be surviving just fine."
>
> Is Ruby a language just to survive or is it a language to provide the tools programmers need to be happy?

(Semantics: "surviving just fine" implies satisfaction and comfort (i.e. happiness.) That notwithstanding:)

The latter, if the "programmers" in question are Matz. Ruby isn't designed to make Java or C++ or 8086 programmers happy, it's designed for Ruby programmers. In Ruby class is not type, and type is defined by behaviour. You have to learn this if you want to be a happy Ruby programmer.

#26 - 11/01/2016 12:01 AM - phluid61 (Matthew Kerwin)

Rodrigo Rosenfeld Rosas wrote:

> I'd just like to add a real situation where I missed a Boolean class recently.
>
> A web request takes an array of values in a JSON encoded string and before processing the data the server-side handler will try to validate the input by comparing the elements type. If Boolean existed, the validation rule would be something like:

```
input = JSON.parse(params['value'])
raise InvalidInput, 'size must be 6' unless input.size == 6
raise InvalidInput, "types don't match" unless input.map(&:class) ==
  [String, String, String, Boolean, String, Boolean]
```

> As you can imagine, the validation took more effort without the existence of Boolean. Something like:

```
input.map{|x| x == false || x == true ? :boolean : x.class } ==
  [String, String, String, :boolean, String, :boolean]
```

There's a potential religious argument here about validating against the schema before/after deserialising. Pragmatically I understand that we have a

JSON parser and we don't have a JSON validator, so it's much easier to validate-later. To that end, it's also really easy to monkey-patch your way to happiness:

```
module Boolean; end
class TrueClass; include Boolean; end
class FalseClass; include Boolean; end
[true, false, 1, nil].select {|x| Boolean === x } #=> [true, false]
```

There's also the fact that just because JSON has a boolean type doesn't mean Ruby has to. Ruby has procs, but you can't serialise them into JSON.

#27 - 11/01/2016 12:04 AM - rrroybbbean (RRRoy BBBean)

Joel on Software "Don't Let Architecture Astronauts Scare You" by Joel Spolsky http://www.joelonsoftware.com/articles/fog0000000018.html

#28 - 11/01/2016 01:51 AM - nobu (Nobuyoshi Nakada)

Rodrigo Rosenfeld Rosas wrote:

> Maybe something like:
>
> ```
> Boolean = [true, false]
>
> input.zip([String, String, String, Boolean, String, Boolean]).all?{|(i, c)| c === i }
> ```

#29 - 11/01/2016 02:59 AM - nobu (Nobuyoshi Nakada)

Samuel Williams wrote:

> > It's a choice by that library author. They are not classes but methods, there can be Boolean method in the library.
>
> Nobu, you are not looking deep enough.
>
> https://github.com/jeremyevans/sequel/blob/c39594dd35b3f8a8975b05be156ba664f1630804/lib/sequel/database/schema_generator.rb#L57-L61

It seems specific to the context, and should be defined there.

#30 - 11/01/2016 02:56 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Nobuyoshi Nakada wrote:

> Rodrigo Rosenfeld Rosas wrote:
>
> > Maybe something like:
> >
> > ```
> > Boolean = [true, false]
> >
> > input.zip([String, String, String, Boolean, String, Boolean]).all?{|(i, c)| c === i }
> > ```

This doesn't seem to work as I just tested in irb: [true, false] === true => false

#31 - 11/03/2016 04:23 AM - ioquatix (Samuel Williams)

> It seems specific to the context, and should be defined there.

Yeah but it's not. 2,080,658 instances of code on GitHub disagree with you. It's not my personal opinion, it's tangible evidence. But what evidence do you have to support NOT adding Boolean? I can't think of any good reason why you shouldn't add it.

#32 - 11/03/2016 06:14 AM - nobu (Nobuyoshi Nakada)

Matz has explained the reasons.

#33 - 11/03/2016 06:46 AM - phluid61 (Matthew Kerwin)

Samuel Williams wrote:

> [...] But what evidence do you have to support NOT adding Boolean? I can't think of any good reason why you shouldn't add it.

Ignoring most of the "bad" reasons in this thread, how about reiterating this one:

If Ruby core were to introduce class Boolean, a lot of those 2 million libraries you refer to would break, because they use module Boolean and module can't re-open a class.

- github search

And who knows how many of the class Boolean peoples' code would also break if instance methods on their previously-safe classes were suddenly inherited by TRUE and FALSE?

#34 - 11/04/2016 12:37 PM - ioquatix (Samuel Williams)

Matthew, I ask you to find one of those examples that would break by introducing the following to core Ruby:

```
if $VERBOSE and defined? Boolean
    warn "Top level Boolean is deprecated, please update your code"
end

class Kernel::Boolean
end

class TrueClass < Kernel::Boolean
end

class FalseClass < Kernel::Boolean
end
```

Finally, if this really is the issue, then either 1/ put it inside require 'boolean' or 2/ release with ruby 3.0 - there are plenty of options, one does not need to be so pessimistic.

Personally, I find Matz argument about duck typing superficial. Because, this isn't a case of duck typing (abstraction) but set membership (is this either true or false). We already have useful checks for these things for Integer, Float, Numeric, String, etc. Why not Boolean? It's inconsistent.

#35 - 11/04/2016 07:27 PM - duerst (Martin Dürst)

Just a side question:

Samuel Williams wrote:

> class Kernel::Boolean
> end

Why do you use Kernel::Boolean and not just Boolean?

#36 - 11/04/2016 10:42 PM - phluid61 (Matthew Kerwin)

Samuel Williams wrote:

> Finally, if this really is the issue, then either 1/ put it inside require 'boolean' or 2/ release with ruby 3.0 - there are plenty of options, one does not need to be so pessimistic.

It's just one of the issues, but it's the one backed by the same numbers you seem to think are the be-all and end-all. Anyone can munge code to not conflict with existing implementations. (Kernel::Boolean? Really?)

> Personally, I find Matz argument about duck typing superficial. Because, this isn't a case of duck typing (abstraction) but set membership (is this either true or false). We already have useful checks for these things for Integer, Float, Numeric, String, etc. Why not Boolean? It's inconsistent.

It's because, as has already been stated, Integer#+ and Float#ceil are methods with implementations shared by all instances of those classes. There is no such method to define on Boolean. And if it's all about set membership, why is NIL not an instance of Boolean? (It is false) Why not all other objects? (They are true)

You are free to define whatever arbitrary sets you like, in whatever context it's appropriate. Put "Y" in the truthy set and "N" in the falsey, I don't care. But that doesn't mean you should push those (or any) arbitrary set memberships into the core.

#37 - 11/07/2016 09:41 AM - ioquatix (Samuel Williams)

> Why do you use Kernel::Boolean and not just Boolean?

Some gems define

```
class Boolean
end

class TrueClass; include Boolean; end
class FalseClass; include Boolean; end
```

and some gems define:

```
module Boolean
end

class TrueClass; include Boolean; end
class FalseClass; include Boolean; end
```

If we defined our own class Boolean it would conflict with users (not that many) who are defining module Boolean. So, instead we define Kernel::Boolean. It's just one option. A simple way of understanding that is that Kernel is kind of a top level scope for name resolution so it works just fine.

Personally I don't think it's important in this case, people defining module Boolean in the global scope are categorically doing the wrong thing, but that's just my opinion. However, the sample implantation I gave works, does not collide with any existing implementation and would be perfectly suitable going forward, where in Ruby 3 it could be renamed to class Boolean and all people have received sufficient warning.

> It's just one of the issues, but it's the one backed by the same numbers you seem to think are the be-all and end-all.

The numbers are really just there to support the hypothesis that lots of people are working around the lack of class Boolean in core. The main issue here being a lack of consistency with other data types and specifically x.is_a? Boolean. Lot of people are doing this - there can be no disagreement here. It's an implementation detail whether methods are defined on class Boolean or class TrueClass and class FalseClass so Matz' argument does not resonate with me. I'm not arguing about implementation. I'm simply supporting moving into Ruby core what is already standard practice by 2 million examples.

#38 - 11/08/2016 03:22 AM - nobu (Nobuyoshi Nakada)

Kernel is included by Object, that is the toplevel namespace, they conflict.

```
class Kernel::Boolean; end
module Boolean; end #=> Boolean is not a module (TypeError)
```

#39 - 11/14/2016 03:36 AM - ioquatix (Samuel Williams)

Nobu, thanks for that clarification. I was not aware of that.

If that's the only issue holding back this feature, I'm sure a workaround can be found that works appropriately.

It might be as simple as going with the more common option class Boolean and breaking when people define module Boolean in the global namespace, which in practice is very small percentage, and already represents a problem when people try to include code which does it both ways, and for which the preferred way is most certainly class Boolean.

Another option is to include in the standard library a boolean.rb which defines set semantics for what people are already doing. If you include this yourself, and other code breaks, it's your own fault.

#40 - 11/15/2016 12:44 PM - shyouhei (Shyouhei Urabe)

Samuel Williams wrote:

> If that's the only issue holding back this feature.

Sadly no. The reason why this is not accepted is matz do not like the idea. You have to persuade him.

> Another option is to include in the standard library a boolean.rb which defines set semantics for what people are already doing. If you include this yourself, and other code breaks, it's your own fault.

What's wrong with a gem, in that case? Raise of rubygems made it harder today than past to introduce a new standard library.

#41 - 02/23/2017 04:49 PM - r.smitala (Radovan Smitala)

You have in some cases true.

There should be some circumstances where implemented Boolean class should be problematic. But many of existed gems just reopen class and life goes on. Rails 5 isn't till now compatible with Ruby 2.4 with introduced Integer class.

I think we can looking on potential Boolean as Integer class. For integer doesn't matter which value represents, one condition is it should be whole number. From almost infinite negative to infinite positive number. But still it is Integer. Whole number is representation of type Integer.
Boolean values are also described. They are represented by true and false. Object which can be just true or false is rightful, and is defined by Boolean type.

Still there should be fallback to FalseClass and TrueClass like for Bignum and Fixnum.

Boolean class is very handy. In coercions, in type checking, in validations, in nice looking and more understandable code, any many more examples.

Yukihiro Matsumoto wrote:

> Rejected for several reasons:
>
> - many gems and libraries had already introduced Boolean class. I don't want to break them.
> - true and false are the only representative of true-false values. In Ruby. nil and false are falsy values, and everything else is a true value. There's no meaning for having a superclass of TrueClass and FalseClass as Boolean.
>
> Matz.

#42 - 03/01/2017 03:29 AM - shyouhei (Shyouhei Urabe)

Radovan Smitala wrote:

> There should be some circumstances where implemented Boolean class should be problematic. But many of existed gems just reopen class and life goes on. Rails 5 isn't till now compatible with Ruby 2.4 with introduced Integer class.

Kind of off topic maybe but can you tell us the Rails 5 story? I was not aware of such thing until now. If you want us to follow how Integer was integrated, there must be lessons to be learned.

#43 - 03/01/2017 07:00 AM - r.smitala (Radovan Smitala)

Shyouhei Urabe wrote:

> Radovan Smitala wrote:
>
> > There should be some circumstances where implemented Boolean class should be problematic. But many of existed gems just reopen class and life goes on. Rails 5 isn't till now compatible with Ruby 2.4 with introduced Integer class.
>
> Kind of off topic maybe but can you tell us the Rails 5 story? I was not aware of such thing until now. If you want us to follow how Integer was integrated, there must be lessons to be learned.

Just deprecated warnings for Fixnum and Bignum :) And that is OK. Rails is biggest project/framework/tool in Ruby community. It takes some time to be fully compatible, not only for Integer addition, but all changes. 3 days before was released first public compatible version :)