Ruby trunk - Feature #13820

Add a nill coalescing operator

08/16/2017 10:50 AM - williamn (William Newbery)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

It would be nice if Ruby had an operator that only considered nil as false, like the null coalescing operators or "Logical Defined-Or operator" (Perl) found in some other languages. Ive seen things like // and //=m ?? and ??=, or ?: used for this.

This would work like || and ||= for short circuiting etc. except that only nil is considered a false condition.

While Ruby considers only "false" and "nil" as false, with everything else true ("", [], {}, etc.) I still find occasionally people trip up when using logical or, || and ||= when the value may be false.

```
a = 0     || 55
# = 0 Ruby already considers 0, "", etc. as true (oter languages do differ a lot here)
a = 0     ?? 55 # = 0 So no change here
a = nil   || 55 # = 55, nil is false so right side is evaulated.
a = nil   ?? 55 # = 55, again no change
a = false || 55 # = 55, however false is false for logical or
a = false ?? 55 # = false, but its still a non-nil value
```

For example when doing things like:

```
def lazy
  @lazy ||= compute_this
end


def fetch(id, **opts)
  host  = opts[:host] || default_host
  https = opts[:https] || true
  port  = opts[:port] || (https ? 443 : 80)
  ...
```

Normally the intention is to use a default value or compute an action if no value is provided, which if the value may be false then requires special handling, or sometimes is missed and results in a bug.

History

#1 - 08/16/2017 12:44 PM - shevegen (Robert A. Heiler)

I am not sure that using a special-purpose operator would make a lot of sense.

I myself use nil primarily as means to indicate a default, "non-set" value. The
moment it is set to a boolean, be it false or true, is for me an indication
that it has been "upgraded" (or set by a user on the commandline etc...)

I do also tend to explicitely query for .nil? on some objects.

By the way, did you actually propose an actual syntax? The two '?'?

I do not think that ?? has any realistic chance for implementation due to
? already being used in ruby - in method definitions or ternary
operator for example. People may wonder why there are so many ? coming out
of nowhere. (For the record, I also consider || to be not pretty ... I
strangely end up using a more verbose but explicit way to set or ensure
defaults in ruby code. I would never write a line such as "
port = opts[:port] || (https ? 443 : 80)" simply because it takes my
brain too long to process what is going on there; my code always ends
up being so simple that I do not have to think about it much at all).

#2 - 08/16/2017 09:56 PM - phluid61 (Matthew Kerwin)

In perl I find $x // $y useful vs $x || $y because sometimes you want to accept "" and 0 as values.

In ruby the only 'defined' falsey value is false, so I'm not sure how useful this feature is here.

If you're pulling options from a hash, for example, it's probably better to use a signal like h.fetch 'x', y to show that you accept falsey values from the hash, and/or x.nil? ? y : x to show that you explicitly only don't want nil

#3 - 08/16/2017 11:45 PM - williamn (William Newbery)

shevegen (Robert A. Heiler) wrote:

> By the way, did you actually propose an actual syntax? The two '?'?

Not really personally set on any given syntax, just ?? and // are familiar to me from other programming. Although actually for ?? specifically, I guess the fact Ruby uses it in both methods and ternary causes a conflict rather than just one or the other (x.nil?? "was nil" : "not nil"). I wouldn't know if the parser can figure that out or not.

But more the concept that any specific syntax.

> I myself use nil primarily as means to indicate a default, "non-set" value. The
> moment it is set to a boolean, be it false or true, is for me an indication
> that it has been "upgraded" (or set by a user on the commandline etc...)

Hmm, maybe I didn't explain clearly. That is pretty much the pattern I come across repeatedly in Ruby code, and it fails for the false value because false is not "upgraded" when people do "x || my_default".

```
"a truthy value" || foo("something else")
# The operator also short circuits so the method `foo` will never even get called
false || foo("something else") # Left side is falsy, so evaluate the right side, but is often not the intent
nil || foo("something else") # Nil is also falsy, so evaluate the right side

"a truthy value" ?? foo("something else") # String is true and not nil, so nothing changes here
false ?? foo("something else") # This changes. Left side is not nil, so the right side is never evaluated
nil ?? foo("something else") # Like with `||`, left side is nil so evaluate the right side

# so this "does the right thing", as far as my maybe not great example goes
https = opts[:https] ?? true
```

phluid61 (Matthew Kerwin) wrote:

> In perl I find $x // $y useful vs $x || $y because sometimes you want to accept "" and 0 as values.
> But not false?

While hash.fetch is nice, I still see || used a lot, in places that maybe wont convert so nice. Also it wont short circuit, in the event the default is not trivial (e.g. with say active record stuff, its easy to have something that goes to the DB without really thinking about it).

```
opts[:foo] || @foo_config || App.config.foo # Occasionally I see 3 or more chained together
hash[:foo] ||= fetch_foo
# fetch doesn't assign the value like this does, and it wont short circuit `fetch_foo`
@lazy ||= calc_lazy # Sometimes used with non-hashes
```

But yes your right, they can all be done other ways, and maybe the better answer is to discourage || in the first place, but I struggled finding ones as tidy to suggest instead.

```
opts.fetch(:foo, !@foo_config.nil? @foo_config : App.config.foo)
hash[:foo] = fetch_foo if !hash.has_key?(:foo)

@lazy = calc_lazy if @lazy.nil?
@lazy # needed because get nil if the if condition is false

if @lazy.nil?
  @lazy = calc_lazy
end
@lazy
```

#4 - 08/17/2017 12:01 AM - phluid61 (Matthew Kerwin)

williamn (William Newbery) wrote:

> phluid61 (Matthew Kerwin) wrote:

> In perl I find $x // $y useful vs $x || $y because sometimes you want to accept "" and 0 as values.

> But not false?

Not in perl ;)

> While hash.fetch is nice, I still see || used a lot, in places that maybe wont convert so nice. Also it wont short circuit, in the event the default is not trivial (e.g. with say active record stuff, its easy to have something that goes to the DB without really thinking about it).

Yes, short-circuit is handy. It's why I was a proponent of &.. Maybe it's okay to add // even if it's only used sometimes.

#5 - 08/18/2017 12:16 AM - nobu (Nobuyoshi Nakada)

williamn (William Newbery) wrote:

> shevegen (Robert A. Heiler) wrote:
>
> > By the way, did you actually propose an actual syntax? The two '?'?
>
> Not really personally set on any given syntax, just ?? and // are familiar to me from other programming. Although actually for ?? specifically, I guess the fact Ruby uses it in both methods and ternary causes a conflict rather than just one or the other (x.nil?? "was nil" : "not nil"). I wouldn't know if the parser can figure that out or not.

?? is a string literal, and // is a regexp literal.

```
def fetch(id, **opts)
  host  = opts[:host] || default_host
  https = opts[:https] || true
  port  = opts[:port] || (https ? 443 : 80)
```

Why not keyword arguments?