Ruby trunk - Feature #14575

Switch Range#=== to use cover? instead of include?

03/04/2018 09:00 AM - zverok (Victor Shepelev)

Status:	Closed
Priority:	Normal
Assignee:	nobu (Nobuyoshi Nakada)
Target version:	2.6

Description

This is **a conscious duplicate** of the bug I've created more than a year ago. I believe that the previous one was rejected too easy, mostly due to the fact I haven't provided enough evidence to support my proposal. I also believe that writing the new, better-grounded proposal would be more visible than adding more comments to the rejected ticket.

The problem: Range#=== (used in case and grep) uses include? to check the value against the range, which could be:
a) really ineffective or b) simply unavailable.

Here are real-life and real-life-alike examples of types that suffer from the problem:

- ipaddress IPAddress("172.16.10.1")..IPAddress("172.16.11.255"): it is really readable to describe in some server config "for this range allow this, for that range allow that", yet it could be fascinatingly slow, calculating thousands of IPs inside range just to check with include?;
- Measurement units: (Unitwise(1, 'm')...Unitwise(10, 'm')) === Unitwise(5, 'm') throws "can't iterate from Unitwise::Measurement", which is reasonable: there is no .succ for numeric types; Ruby itself has an ugly workaround of "if this is a numeric type, behave like cover?"
- Dates and times: (Date.today..Date.today + 1) === DateTime.now is false; it is hard to imagine code where it is a desired behavior.

Matz's objections to the previous ticket were:

> I see no real-world use-case for Range#=== with strings. (Because I have provided only string ranges example initially -- VS)

That is addressed, hopefully, with the new set of examples.

> Besides that, using cover? behavior for [string] ranges would introduce incompatibility.

I don't know how to estimate amount of incompatibilities introduced by this behavior change.
Yet it is really hard (for me) to invent some reasonable real-life use case which could be broken by it.

Related issues:

Related to Ruby trunk - Feature #12996: Optimize Range#===	**Open**
Is duplicate of Ruby trunk - Feature #12612: Switch Range#=== to use cover? i...	**Rejected**

Associated revisions

Revision 989e07c0 - 05/17/2018 10:46 AM - nobu (Nobuyoshi Nakada)

range.c: === by cover?

- range.c (range_eqq): switch Range#=== to use cover? instead of include?. [Feature #14575]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@63453 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 63453 - 05/17/2018 10:46 AM - nobu (Nobuyoshi Nakada)

range.c: === by cover?

- range.c (range_eqq): switch Range#=== to use cover? instead of include?. [Feature #14575]

Revision 63453 - 05/17/2018 10:46 AM - nobu (Nobuyoshi Nakada)

range.c: === by cover?

- range.c (range_eqq): switch Range#=== to use cover? instead of include?. [Feature #14575]

History

#1 - 03/05/2018 10:15 AM - shevegen (Robert A. Heiler)

Does anyone really write code such as:

```
IPAddress("172.16.10.1")..IPAddress("172.16.11.255"):
```

?

I don't recall having seen that out in the wild.

To me that is also less readable than method checks such
as .valid_ip? or similarly named methods.

#2 - 03/05/2018 10:26 AM - zverok (Victor Shepelev)

> I don't recall having seen that out in the wild.

I did that once (around the time I've written the first ticket ¯_(ツ)_/¯). Values were not explicitly in code, they've loaded from config, and the final solution looked like...

```
case request.ip
when *developer_ip_ranges
 ...
when *internal_microservices_ip_ranges
 ...
when *vip_client_ip_ranges
 ...
else
 ...
end
```

That can be rewritten several different ways, yet I still believe this is a case that demonstrates the inadequacy of === implementation. I've spent several hours debugging unexpected extra 0.4 seconds in server response time before getting to "range expansion" as a root cause.

#3 - 03/13/2018 07:36 AM - tom_dalling (Tom Dalling)

The Range#=== methods currently works in one of three different ways. The code is here:
https://github.com/ruby/ruby/blob/df1cd0f438bd17a4d3fbe9077e0b308e0b25c4b5/range.c#L1151-L1158

1. If it's a range of "numeric values", it uses #cover?
2. If it's a range of strings, it uses rb_str_include_range_p
3. For everything else, it calls super, which is Enumerable#include?

I can see that changing the behaviour of (3) might lead to incompatibility issues. Maybe it's better to implement this in (1), by allowing objects to declare themselves as "numeric values".

#4 - 03/13/2018 08:09 AM - zverok (Victor Shepelev)

> I can see that changing the behaviour of (3) might lead to incompatibility issues.

Any real case on mind? I can't think of any (well, maybe somebody really relies on the fact how date..date+1 does not match datetimes in between... But I don't believe that it could be some popular gem or any other serious and hard-to-fix incompatibility).

To be bold, I believe that current behavior is "broken" in some sense, and fixing it may be worth the fact somebody (probably) needs to (slightly) update some (weird) code.

> Maybe it's better to implement this in (1), by allowing objects to declare themselves as "numeric values".

Well, besides the fact that I can't think about existing idiomatic way to "declare themselves" (what it could be, implement def numeric? → true?), of my examples neither IPAddress nor Date are really "numeric". So it leans to def yes_for_this_datatype_using_range_cover_is_preferred? → true :)

#5 - 03/13/2018 11:06 AM - nobu (Nobuyoshi Nakada)

tom_dalling (Tom Dalling) wrote:

> 1. If it's a range of strings, it uses rb_str_include_range_p
> 2. For everything else, it calls super, which is Enumerable#include?

> I can see that changing the behaviour of (3) might lead to incompatibility issues. Maybe it's better to implement this in (1), by allowing objects to declare themselves as "numeric values".

I thought handling it by begin.include_range? method, and rb_str_include_range_p was the implementation for String.

#6 - 03/13/2018 03:12 PM - zverok (Victor Shepelev)

nobu (Nobuyoshi Nakada)
Are there imaginable real-life incompatibilities introduced by changing semantics?

I believe that begin.include_range? is more problematic: all older gems and libraries should implement it (or it should be monkeypatched into them by user's code), and a requirement to implement it, or to avoid using ranges without it, would not be very visible and will constantly "shoot in the foot".

#7 - 05/17/2018 07:07 AM - matz (Yukihiro Matsumoto)

- Related to Feature #12612: Switch Range#=== to use cover? instead of include? added

#8 - 05/17/2018 07:07 AM - matz (Yukihiro Matsumoto)

- Related to deleted (Feature #12612: Switch Range#=== to use cover? instead of include?)

#9 - 05/17/2018 07:08 AM - matz (Yukihiro Matsumoto)

- Is duplicate of Feature #12612: Switch Range#=== to use cover? instead of include? added

#10 - 05/17/2018 07:08 AM - matz (Yukihiro Matsumoto)

- Related to Feature #12996: Optimize Range#=== added

#11 - 05/17/2018 07:21 AM - matz (Yukihiro Matsumoto)

Even though we still have compatibility concerns, the performance benefit can be valuable.
Let us try it to see if we have any (serious) issues.

Matz.

#12 - 05/17/2018 07:26 AM - ko1 (Koichi Sasada)

- Target version set to 2.6

- Assignee set to nobu (Nobuyoshi Nakada)

#13 - 05/17/2018 08:20 AM - nobu (Nobuyoshi Nakada)

- Backport deleted (2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN)

- Tracker changed from Bug to Feature

#14 - 05/17/2018 10:46 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

Applied in changeset trunk|r63453.

---

range.c: === by cover?

- range.c (range_eqq): switch Range#=== to use cover? instead of include?. [Feature #14575]