Ruby trunk - Feature #13606

Enumerator equality and comparison

05/27/2017 11:22 PM - glebm (Gleb Mazovetskiy)

Status:	Rejected	
Priority:	Normal	
Assignee:		
Target version:		

Description

In Ruby, most objects are compared by value. What do you think about Enumerators following the same pattern? I think this would greatly increase the expressiveness of Ruby.

Proposal:

Two Enumerators should be considered equal (==) if they yield the same number of elements and these elements are equal (==).
If both of the Enumerators are infinite, the equality operator never terminates.
<=> should be handled similarly.

History

#1 - 05/28/2017 08:29 AM - duerst (Martin Dürst)

- Status changed from Open to Feedback

Sounds interesting in theory, but do you have actual use cases? And do you think that the potential inefficiency is worth it?

#2 - 05/28/2017 02:01 PM - shevegen (Robert A. Heiler)

I am not even sure that I understand the proposal.

If I understood it correctly then two enumerable objects (did I get this part right)
should return true if they behave/return the same? I think I can see it being
related to duck typing... but they are not entirely the same are they? Different
object id for most objects for example. But it also may be that I did not fully
understand the proposal yet.

What would the speed penalty be if one exists? I guess the latter one could be
handled by some "behavioural switch" for people who need the behaviour
desscribed in the proposal, so a use-case example would be helpful.

#3 - 05/28/2017 04:50 PM - glebm (Gleb Mazovetskiy)

shevegen (Robert A. Heiler) wrote:

> [...] but they are not entirely the same are they? Different
> object id for most objects for example. But it also may be that I did not fully
> understand the proposal yet.

Most objects in Ruby are compared semantically if the object IDs are different, including Array and Hash.

> What would the speed penalty be if one exists?

If the Enumerators have different object_ids, previous the equality check was O(1) but is O(n) with this proposal (just like for Array).

> I guess the latter one could be
> handled by some "behavioural switch" for people who need the behaviour
> desscribed in the proposal, so a use-case example would be helpful.

I am not proposing a behavioural switch. This should be a backwards-incompatible change.
Use cases are the same as for arrays, except when the "arrays" are "lazy".

Example:

```
[1, 2, 3].reverse == [3, 2, 1]
#=> true
```

```
[1, 2, 3].reverse_each == [3, 2, 1]
#=> false

[1, 2, 3].reverse_each == [1, 2, 3].reverse_each
#=> false
```

With this proposal, the last two are also true.

#4 - 05/28/2017 04:55 PM - glebm (Gleb Mazovetskiy)

duerst (Martin Dürst) wrote:

> Sounds interesting in theory, but do you have actual use cases? And do you think that the potential inefficiency is worth it?

The use cases are the same as for comparing Arrays.
The potential inefficiency is not a problem because if you need to compare Enumerators by object_id you can do it using equal?. If the object_ids are the same, both the current and the proposed comparisons take constant time.

#5 - 05/28/2017 06:21 PM - MSP-Greg (Greg L)

Could be helpful, but some Enumerators are not ordered. So how would == work for 'hash like' objects (assuming they're not based on a hash, which has an == operator)?

I suppose it could be considered 'restricted' to ordered collections...

#6 - 05/28/2017 07:48 PM - glebm (Gleb Mazovetskiy)

MSP-Greg (Greg L) wrote:

> Could be helpful, but some Enumerators are not ordered. So how would == work for 'hash like' objects (assuming they're not based on a hash, which has an == operator)?
>
> I suppose it could be considered 'restricted' to ordered collections...

Equal objects should produce equal results when used.
From this perspective on equality, enumerators that yield in different order should not be considered equal, even if they internally yield elements from an unordered collection.

If the enumerator yields in a different order every time it's called, then the comparison is not guaranteed to return the same result every time. This is a rare edge case though.

#7 - 05/29/2017 07:13 AM - duerst (Martin Dürst)

MSP-Greg (Greg L) wrote:

> Could be helpful, but some Enumerators are not ordered.

All Enumerators are ordered. The order is defined by the each method, or alternatively by the to_a method.

Actually, I wonder if there's any difference between what the OP wants and

```
enumerator_A.to_a == enumerator_B.to_a
```

Although direct comparison for equality would be shorter, the above makes it explicit that efficiency may be low.

#8 - 05/29/2017 02:50 PM - glebm (Gleb Mazovetskiy)

duerst (Martin Dürst) wrote:

> Actually, I wonder if there's any difference between what the OP wants and
>
> ```
> enumerator_A.to_a == enumerator_B.to_a
> ```

The above allocates Arrays, Enumerator equality would not.

#9 - 08/31/2017 07:36 AM - knu (Akinori MUSHA)

- Status changed from Feedback to Rejected

Without any actual use case, there would be no effective definition of equality for enumerators.

FWIW, the initial design policy is, Enumerator is an abstract entity that only guarantees it responds to each for enumeration, and it's not your problem as to what's behind a given enumerator. You shouldn't have to care about the equality in the first place.