Ruby master - Feature #7657

Array#& doesn't accept Enumerables

01/06/2013 09:00 AM - Nevir (Ian MacLeod)

Status:	Open
Priority:	Normal
Assignee:	matz (Yukihiro Matsumoto)
Target version:	

Description

This seems similar to http://bugs.ruby-lang.org/issues/6923

Example:

```
irb(main):001:0> class Thing
irb(main):002:1>   include Enumerable
irb(main):003:1>   def each(*args, &block)
irb(main):004:2>     [1,2,3,4,5].each(*args, &block)
irb(main):005:2>   end
irb(main):006:1> end
=> nil

irb(main):007:0> Array(Thing.new) & [1,3,5]
=> [1, 3, 5]
irb(main):008:0> [1,3,5] & Thing.new
TypeError: can't convert Thing into Array

irb(main):009:0> Thing.class_eval do
irb(main):010:1*   alias_method :to_ary, :to_a
irb(main):011:1> end
=> Thing

irb(main):012:0> [1,3,5] & Thing.new
=> [1, 3, 5]
```

Would it make sense for Enumerable to implement to_ary as well? Or is this purely a bug in Array#&?

History

#1 - 01/06/2013 09:02 AM - Nevir (Ian MacLeod)

Err, I mean it looks like it's similar to http://bugs.ruby-lang.org/issues/1028

#2 - 01/06/2013 08:35 PM - Eregon (Benoit Daloze)

- Assignee set to matz (Yukihiro Matsumoto)

> Would it make sense for Enumerable to implement to_ary as well?

Matz is not agreeing as you see in #1893:

" Implicit conversion methods such as #to_ary and #to_str should be
defined only when the object provides (almost) equivalent behavior
(i.e. method set). Enumerable and Array are not the case. "

> Or is this purely a bug in Array#&?

Array#& (as well as #|, #-, #uniq and #uniq!) compare elements using #hash and #eql? as they build an Hash for performance reasons (O(m+n) instead of O(m*n), m=size of self, n=size of other array (or self in case of #uniq{,!})).
This is still not documented however for some of these methods, I'll try to fix this.

I think the reason this is done is semantically it makes less sense to have an intersection of an Array and an Enumerable. You would not have the reverse, as Enumerable does not have set-like methods (Array is already stealing a bit of Set in this regard because it is very practical).

As an illustration, [1,2,3] + enumerable raises an error as intended (unless enumerable implements #to_ary). Too loose conversion is dangerous (for example IO#each will only yield once the lines).

It could be done though at the expense of using #each instead of directly iterating on the Array structure, or converting with #to_a. I am not sure the gain of having the implicit #to_a would be clear, so I propose to use [1,3,5] & Thing.new.to_a in your case.

About #1028, it was just a bug in the way of checking whether an argument is an Array or not, the code handling Enumerable was already there (that code is actually almost duplicated with enum_take()).

#3 - 01/07/2013 10:25 AM - Nevir (Ian MacLeod)

Let me see if I understand the implicit conversion cases:

- If an object implements #to_ary, #to_str, etc - it is asserting that it is close enough to the desired type in behavior that we don't need to perform an explicit coercion?
- I.e. if I implement #to_ary, I'm asserting that I behave close enough to an Array that you can just use me directly.

In that case, why is it an exception if the result of #to_ary is not an instance (or subclass) of Array?

```
irb(main):001:0> class Thing < BasicObject
irb(main):002:1>   def initialize
irb(main):003:2>     @data = [1,2,3,4,5]
irb(main):004:2>   end
irb(main):005:1>   def method_missing(sym, *args, &block)
irb(main):006:2>     @data.send(sym, *args, &block)
irb(main):007:2>   end
irb(main):008:1>   def to_ary
irb(main):009:2>     self
irb(main):010:2>   end
irb(main):011:1> end

irb(main):012:0> Thing.new.to_a
=> [1, 2, 3, 4, 5]

irb(main):013:0> Array(Thing.new)
TypeError: can't convert Thing to Array (Thing#to_ary gives Thing)

irb(main):014:0> [1,3,5] & Thing.new
TypeError: can't convert Thing to Array (Thing#to_ary gives Thing)
```

Re: building a Hash under the covers, the use of #hash and #eql? is primarily for the individual elements of the container? From looking at the C source, it looks like #& is just doing a traditional for loop over the array instead of using #each. https://github.com/ruby/ruby/blob/trunk/array.c#L3856

It doesn't seem like that code needs to assert that the input is an array at all except to avoid message dispatch overhead - ary_add_hash is also just doing a standard for loop, from what I can tell: https://github.com/ruby/ruby/blob/trunk/array.c#L3736

Maybe it makes more sense to build the hash via #each (and use the existing functions if the input is an Array, to preserve performance)?

#4 - 01/09/2013 02:19 AM - nobu (Nobuyoshi Nakada)

- Tracker changed from Bug to Feature

#5 - 01/25/2013 12:17 PM - ko1 (Koichi Sasada)

- Target version set to 2.6

#6 - 01/25/2013 03:18 PM - marcandre (Marc-Andre Lafortune)

- Category set to core

Nevir (Ian MacLeod) wrote:

> Let me see if I understand the implicit conversion cases:
>
> - If an object implements #to_ary, #to_str, etc - it is asserting that it is close enough to the desired type in behavior that we don't need to perform an explicit coercion?
> - I.e. if I implement #to_ary, I'm asserting that I behave close enough to an Array that you can just use me directly.
>
> In that case, why is it an exception if the result of #to_ary is not an instance (or subclass) of Array?

By implementing to_ary, your class states must return an array because after MRI asks for implicit conversion with to_ary, it will then access the data directly. No call toeach,[]orslice` is made.

> Re: building a Hash under the covers, the use of #hash and #eql? is primarily for the individual elements of the container? From looking at the C

source, it looks like #& is just doing a traditional for loop over the array instead of using #each

Right.

So your request was changed from a bug report to a feature request, i.e. to accept Enumerables as argument to &, |, etc.

Here's another argument in your favor:

```
require 'set'
Set[1,2,3] | (3..4) #  => #<Set: {1, 2, 3, 4}>
   [1,2,3] | (3..4) # =>TypeError: can't convert Range into Array
```

I feel it would be best to be consistent and allow Enumerable arguments too.

#7 - 12/25/2017 06:15 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)