Ruby trunk - Feature #14097

Add union and difference to Array

11/10/2017 06:51 AM - ana06 (Ana Maria Martinez Gomez)

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

Description

Currently there is a concat method in ruby which does the same as +, but modifying the object. We could introduce a union and difference methods, which would be the equivalent for the | and - operators. This operators are normally less used due to lack of readability and expressivity. You end seeing thinks like:

```
array.concat(array2).uniq!
```

just because it is more readable. When it could be written like:

```
array |= array2
```

But, as this is not clear for some people, the new method will solve this problem:

```
array.union(array2)
```

And now this is clean and readable, as everybody expect from Ruby, the language focused on simplicity and productivity. ;)

Can I send a PR? :)

Related issues:

Related to Ruby trunk - Feature #14105: Introduce xor as alias for Set#^	**Feedback**

Associated revisions

Revision 744e816f - 09/20/2018 03:18 AM - nobu (Nobuyoshi Nakada)

Add union method to Array

I introduce a union method equivalent to the | operator, but which
accept more than array as argument. This improved readability, and it
is also coherent with the + operator, which has a similar concat
method. The method doesn't modify the original object and return a new
object instead. It is plan to introduce a union! method as well.

Tests and documentation are included.

It solves partially https://bugs.ruby-lang.org/issues/14097

[Fix GH-1747] [Feature #14097]

From: Ana María Martínez Gómez ammartinez@suse.de

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64787 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64787 - 09/20/2018 03:18 AM - nobu (Nobuyoshi Nakada)

Add union method to Array

I introduce a union method equivalent to the | operator, but which
accept more than array as argument. This improved readability, and it
is also coherent with the + operator, which has a similar concat
method. The method doesn't modify the original object and return a new
object instead. It is plan to introduce a union! method as well.

Tests and documentation are included.

It solves partially https://bugs.ruby-lang.org/issues/14097

[Fix GH-1747] [Feature #14097]

From: Ana María Martínez Gómez ammartinez@suse.de

Revision 64787 - 09/20/2018 03:18 AM - nobu (Nobuyoshi Nakada)

Add union method to Array

I introduce a union method equivalent to the | operator, but which
accept more than array as argument. This improved readability, and it
is also coherent with the + operator, which has a similar concat
method. The method doesn't modify the original object and return a new
object instead. It is plan to introduce a union! method as well.

Tests and documentation are included.

It solves partially https://bugs.ruby-lang.org/issues/14097

[Fix GH-1747] [Feature #14097]

From: Ana María Martínez Gómez ammartinez@suse.de

Revision d0f9184f - 09/20/2018 03:18 AM - nobu (Nobuyoshi Nakada)

Introduce rb_ary_union method in Array

Avoid repeating code and improve readability in rb_ary_or and
rb_ary_union_multi.

[Fix GH-1747] [Feature #14097]

From: Ana María Martínez Gómez ammartinez@suse.de

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64788 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64788 - 09/20/2018 03:18 AM - nobu (Nobuyoshi Nakada)

Introduce rb_ary_union method in Array

Avoid repeating code and improve readability in rb_ary_or and
rb_ary_union_multi.

[Fix GH-1747] [Feature #14097]

From: Ana María Martínez Gómez ammartinez@suse.de

Revision 64788 - 09/20/2018 03:18 AM - nobu (Nobuyoshi Nakada)

Introduce rb_ary_union method in Array

Avoid repeating code and improve readability in rb_ary_or and
rb_ary_union_multi.

[Fix GH-1747] [Feature #14097]

From: Ana María Martínez Gómez ammartinez@suse.de

Revision 4c082239 - 09/20/2018 03:18 AM - nobu (Nobuyoshi Nakada)

Link Array#union from | method

Array#uniq is not really related with Array#|, so I replaced it by
Array#union.

[Fix GH-1747] [Feature #14097]

From: Ana María Martínez Gómez ammartinez@suse.de

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64789 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision ce079f16 - 09/20/2018 03:18 AM - nobu (Nobuyoshi Nakada)

Introduce rb_ary_union_hash method in Array

Avoid repeating code and improve readability in rb_ary_or and
rb_ary_union_multi. Similaty as done with rb_ary_union.

[Fix GH-1747] [Feature #14097]

From: Ana María Martínez Gómez ammartinez@suse.de

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64790 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64789 - 09/20/2018 03:18 AM - nobu (Nobuyoshi Nakada)

Link Array#union from | method

Array#uniq is not really related with Array#|, so I replaced it by
Array#union.

[Fix GH-1747] [Feature #14097]

From: Ana María Martínez Gómez ammartinez@suse.de

Revision 64789 - 09/20/2018 03:18 AM - nobu (Nobuyoshi Nakada)

Link Array#union from | method

Array#uniq is not really related with Array#|, so I replaced it by
Array#union.

[Fix GH-1747] [Feature #14097]

From: Ana María Martínez Gómez ammartinez@suse.de

Revision 64790 - 09/20/2018 03:18 AM - nobu (Nobuyoshi Nakada)

Introduce rb_ary_union_hash method in Array

Avoid repeating code and improve readability in rb_ary_or and
rb_ary_union_multi. Similaty as done with rb_ary_union.

[Fix GH-1747] [Feature #14097]

From: Ana María Martínez Gómez ammartinez@suse.de

Revision 64790 - 09/20/2018 03:18 AM - nobu (Nobuyoshi Nakada)

Introduce rb_ary_union_hash method in Array

Avoid repeating code and improve readability in rb_ary_or and
rb_ary_union_multi. Similaty as done with rb_ary_union.

[Fix GH-1747] [Feature #14097]

From: Ana María Martínez Gómez ammartinez@suse.de

Revision 64921 - 10/05/2018 06:23 AM - nobu (Nobuyoshi Nakada)

Add difference method to Array

I introduce a difference method equivalent to the - operator, but
which accept more than array as argument. This improved readability, and
it is also coherent with the + operator, which has a similar concat
method. The method doesn't modify the original object and return a new
object instead. I plan to introduce a difference! method as well.

Tests and documentation are included.

It solves partially https://bugs.ruby-lang.org/issues/14097

From: Ana María Martínez Gómez ammartinez@suse.de

History

#1 - 11/10/2017 07:29 AM - ana06 (Ana Maria Martinez Gomez)

This will also allow to add multiple arguments to the union, which is currently not possible:

```
array.union(array1, array2)
```

#2 - 11/10/2017 09:20 AM - mame (Yusuke Endoh)

I'm neutral to your proposal itself. My two cents: Array#union should return a new array instead of modifying self, and Array#union! should be its modifying version.

#3 - 11/10/2017 12:05 PM - ana06 (Ana Maria Martinez Gomez)

I think it is a great idea. I do not understand why concat modify the array, as most of the method of the Array class has a ! method for that. Should I also introduced a concat! method?

#4 - 11/10/2017 02:46 PM - k0kubun (Takashi Kokubun)

> I think it is a great idea. I do not understand why concat modify the array, as most of the method of the Array class has a ! method for that. Should I also introduced a concat! method?

Probably changing #concat to be non-destructive is too breaking for backward compatibility. We can use #+ for that purpose. And having #concat and #concat! in the same behavior would be just confusing.

#5 - 11/11/2017 09:11 PM - ana06 (Ana Maria Martinez Gomez)

What about introducing concat! with the same behaviour as concat and deprecating concat. Then we could in the feature give concat the behaviour it deserves. It is confusing as well that this method modify the object and I think we should fix this.

#6 - 11/11/2017 10:57 PM - jeremyevans0 (Jeremy Evans)

ana06 (Ana Maria Martinez Gomez) wrote:

> What about introducing concat! with the same behaviour as concat and deprecating concat. Then we could in the feature give concat the behaviour it deserves. It is confusing as well that this method modify the object and I think we should fix this.

Regarding concat!, there seems to be a misunderstanding that methods should end with ! to be mutating. That is not the convention in the core classes. The core classes have many methods that are mutating but do not end in !. The convention regarding ! is if there is both a method with ! and a method without, the version with ! mutates and the version without returns a potentially modified copy.

The following Array methods that do not end in ! are mutating, so if you want to change concat for "consistency", you would have to change all of them:

- clear
- concat
- delete
- delete_at
- delete_if
- fill
- insert
- keep_if
- pop
- push
- replace
- shift
- unshift

You'd also probably have to change String and Hash similarly if you wanted this "consistency" in regards to !.

In regards to union, not all arrays are sets, and I'm not in favor of introducing additional set-specific methods to Array. Set#union is already implemented.

#7 - 11/13/2017 01:40 PM - ana06 (Ana Maria Martinez Gomez)

I would that the difference is that there are some method where is not expected that the Array is modified, and some others where you expect it. So, for example, with pop and push I don't think we should have two methods, one which modify the object and another one which does it, but at least I would keep the Array operators methods consistent.

jeremyevans0 (Jeremy Evans) do you find it coherent having you union method and only one concat? Won't be that confusing? you will always need to check the documentation as you won't know when the object is modified and when not.

Also, union is an operation with Arrays when you when to use then as set for any reason. The main difference is that in the Array the order of the elements matter, so that is unrelated to the Set class.

#8 - 11/13/2017 02:53 PM - jeremyevans0 (Jeremy Evans)

ana06 (Ana Maria Martinez Gomez) wrote:

> jeremyevans0 (Jeremy Evans) do you find it coherent having you union method and only one concat? Won't be that confusing? you will always

> need to check the documentation as you won't know when the object is modified and when not.

I don't think renaming concat to concat! makes things more coherent. We already have + for a concat that returns a new array. Yes, if you are unfamiliar with the methods you will probably need to read the documentation.

The array class already has a union operator (|) which returns a new array, and in combination with replace you can easily build union. union doesn't seem a common enough need to warrant adding as a separate core method.

#9 - 11/13/2017 03:22 PM - ana06 (Ana Maria Martinez Gomez)

jeremyevans0 (Jeremy Evans)

> The array class already has a union operator (|) which returns a new array, and in combination with replace you can easily build union. union doesn't seem a common enough need to warrant adding as a separate core method.

Yes, a operator that is not clear for many people. I think Ruby deserve something more readable and elegant. Moreover, union would allow to make the union of more than 2 arrays at the same time in a much more efficient way than applying | several times. So it is not only an "stetic" change, it is also a performance improvement.

I am not sure what to mean what replace to build a union. Can you please elaborate?

#10 - 11/13/2017 04:04 PM - jeremyevans0 (Jeremy Evans)

ana06 (Ana Maria Martinez Gomez) wrote:

> jeremyevans0 (Jeremy Evans)
>
> > The array class already has a union operator (|) which returns a new array, and in combination with replace you can easily build union. union doesn't seem a common enough need to warrant adding as a separate core method.
>
> Yes, a operator that is not clear for many people. I think Ruby deserve something more readable and elegant.

The argument against the | operator could potentially apply to any operator. Most things are unclear until they are learned. Someone with no knowledge of English might find the | operator more clear than the union method.

> Moreover, union would allow to make the union of more than 2 arrays at the same time in a much more efficient way than applying | several times. So it is not only an "stetic" change, it is also a performance improvement.

You could build union without a nested application of |. No doubt you could get the maximum performance by implementing it in C, but I don't believe the cost of maintaining such code is worth it, considering how often it is used.

> I am not sure what to mean what replace to build a union. Can you please elaborate?

```
class Array
  def union(*other)
    ret = self
    other.each{|a| ret |= a}
    replace(ret)
  end
  # or
  def union(*other)
    tmp = other.unshift(self)
    tmp.flatten!(1)
    tmp.uniq!
    replace(tmp)
  end
end
```

In similar cases in the past, the recommendation has often been to build the functionality as a gem, and if the gem gets popular and widely used, then it can be considered for inclusion in core.

#11 - 11/15/2017 03:10 PM - ana06 (Ana Maria Martinez Gomez)

> The argument against the | operator could potentially apply to any operator. Most things are unclear until they are learned. Someone with no knowledge of English might find the | operator more clear than the union method.

Ruby is the language, which claims to have an elegant syntax that is natural to read and easy to write. And that's why it have readable method for operators. And that's what people love about Ruby. But I am not saying that we should remove the operator, so you can keep using it. ;)

> You could build union without a nested application of |. No doubt you could get the maximum performance by implementing it in C, but I don't believe the cost of maintaining such code is worth it, considering how often it is used.

It is some really simple code. And I will implement it, why do you care about the effort to do it? It is someone else effort, who is really happy to do it.

> In similar cases in the past, the recommendation has often been to build the functionality as a gem, and if the gem gets popular and widely used, then it can be considered for inclusion in core.

What I want is to provide an efficient union of several arrays, that need to be implemented in C. Implementing this in a Ruby gem makes no point at all.

#12 - 11/15/2017 03:54 PM - jeremyevans0 (Jeremy Evans)

ana06 (Ana Maria Martinez Gomez) wrote:

> > You could build union without a nested application of |. No doubt you could get the maximum performance by implementing it in C, but I don't believe the cost of maintaining such code is worth it, considering how often it is used.
>
> It is some really simple code. And I will implement it, why do you care about the effort to do it? It is someone else effort, who is really happy to do it.

I care because once it is added, it is impossible to remove without breaking backwards compatibility. This is not a one time cost of initial implementation, it's a perpetual maintenance cost.

I'm not in favor of adding methods to the core classes simply because they are useful in certain cases. If we added every method to the core classes that was useful in specific cases, we'd eventually have thousands of methods in each core class.

> > In similar cases in the past, the recommendation has often been to build the functionality as a gem, and if the gem gets popular and widely used, then it can be considered for inclusion in core.
>
> What I want is to provide an efficient union of several arrays, that need to be implemented in C. Implementing this in a Ruby gem makes no point at all.

It appears you may not be aware that plenty of ruby gems are implemented in C using ruby's C-API and have same performance as if they were part of ruby core.

#13 - 12/13/2017 09:13 AM - ana06 (Ana Maria Martinez Gomez)

> I'm not in favor of adding methods to the core classes simply because they are useful in certain cases. If we added every method to the core classes that was useful in specific cases, we'd eventually have thousands of methods in each core class.

It is

- useful
- more efficient
- consistent with the concat method in the same class
- readable
- elegant
- easy to use and to read
- follow Ruby principle of allowing to do thing in several ways
- avoid that people use inefficient methods to make the code understandable

> It appears you may not be aware that plenty of ruby gems are implemented in C using ruby's C-API and have same performance as if they were part of ruby core.

Your example was in Ruby... But it makes no sense a Ruby gem just to add two methods in a class where there is already a similar method for another operator. For something bigger I would agree with you... but this is just a method which makes sense to add to the class...

#14 - 05/11/2018 10:00 AM - ana06 (Ana Maria Martinez Gomez)

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

- Tracker changed from Bug to Feature

#15 - 05/17/2018 08:13 AM - matz (Yukihiro Matsumoto)

- Related to Feature #14105: Introduce xor as alias for Set#^ added

#16 - 05/17/2018 01:07 PM - matz (Yukihiro Matsumoto)

Thank you for the proposal.

I am not sure your real intention. Do you want mutating variation of or-operator?
Or just more readable alias of or-operator?

Matz.

#17 - 05/18/2018 10:45 AM - duerst (Martin Dürst)

matz (Yukihiro Matsumoto) wrote:

> I am not sure your real intention. Do you want mutating variation of or-operator?
> Or just more readable alias of or-operator?

mame (Yusuke Endoh) wrote:

> I'm neutral to your proposal itself. My two cents: Array#union should return a new array instead of modifying self, and Array#union! should be its modifying version.

I would definitely prefer Yusuke's version to a version where Array#union is not modifying. While the modifying version will occasionally be useful, in general, we should gently push people towards using non-modifying code.

#18 - 05/24/2018 12:06 AM - ana06 (Ana Maria Martinez Gomez)

matz (Yukihiro Matsumoto)

> I am not sure your real intention. Do you want mutating variation of or-operator?
> Or just more readable alias of or-operator?

Thanks for taking a look at the issue. What I am proposing is a new union method that it is an alias for | in the case of two arrays but that it is also more efficient in the case of more than two arrays. Exactly as it happens with + and concat. concat, apart from modifying the first array (which maybe shouldn't be the case) is more readable, but in the case on more than two arrays is more efficient as well.

I also send a PR with a possible implementation: https://github.com/ruby/ruby/pull/1747

#19 - 05/24/2018 12:08 AM - ana06 (Ana Maria Martinez Gomez)

and this is not necessarily related to Feature #14105. I would say that they are two different topics even if both of them aim for readability. In the case of set is only an alias and there is not a similar case as it happens here with concat

#20 - 05/24/2018 07:33 PM - Student (Nathan Zook)

I cannot say that I am a fan of this proposal. To be fair, I'm not a fan of #|.

Arrays are not sets. Trying to treat them as if they are is an error, and will create subtle problems.

What should be the result of the following operations?
[1, 1] | [1]
[1] | [1, 1]

Of course, there are more interesting examples. These two are to get you started.

I don't care what the results currently are. I don't care what you think they should be. I can present extremely strong arguments for various answers. For this reason, I believe that #| is an ill-defined concept.

Generalizing an ill-defined concept is a world of pain.

If you insist on treating objects of one class as if they were members of a different class, there should be bumps in the road to at least warn you that maybe this is a bad idea.

I'm not going to argue that we should remove or deprecate #|. I don't think of myself as a fanatic. But encouraging this sort of abuse of the type system just creates problems.

#21 - 08/25/2018 03:13 PM - mame (Yusuke Endoh)

Ana, I watched the video of your EuRuKo talk. I could understand the background of your proposal.

However, I'm now getting confused. I thought you agreed that Array#union should be non-destructive, but in your talk you explained it was a destructive union operator. What are you driving at?

#22 - 08/26/2018 08:52 AM - ana06 (Ana Maria Martinez Gomez)

mame (Yusuke Endoh)

> However, I'm now getting confused. I thought you agreed that Array#union should be non-destructive, but in your talk you explained it was a destructive union operator. What are you driving at?

In the talk I wanted to focus on the necessity of having an union method without entering in the discussion of if it should be non-destructive or not. It didn't seem that important to me if it is needed to add a ! or not. But you are right, having union! in the voting would have been better. matz (Yukihiro Matsumoto) gave me the same feedback after the talk. ;)

#23 - 08/26/2018 09:08 AM - mame (Yusuke Endoh)

- Assignee set to matz (Yukihiro Matsumoto)

- Status changed from Open to Assigned

Thank you for the answer. I'm assigning this ticket to matz.

#24 - 08/28/2018 08:22 AM - ana06 (Ana Maria Martinez Gomez)

@Student as I just answered in the PR about the same concern, I'll answer here to:

> I think there are some uses of case. First, it could be that you have an array in your application, would it be worthwhile to convert it to set just to be able to do a union? Second, what happen if you want to add the elements of an array to another array only if they were not already there but keeping the order? you can not do this with a set. Third, what if after the union you want to apply a method in the Array class that is not in the Set class (like for example sort_by)?

#25 - 08/30/2018 08:51 AM - ana06 (Ana Maria Martinez Gomez)

As mame (Yusuke Endoh) already said, I gave a talk at Euruko about this issue:

https://youtu.be/jUc8InwoA-E?t=2m54s

I did an standing up voting. As in the video you can not see the audience, here is the picture I took during the talk with the result of the voting:

https://user-images.githubusercontent.com/16052290/44840875-5cd3bf00-ac42-11e8-84e9-815bc6473719.jpg

The quality of the picture is not really good, but I have pixelized the faces of the recognizable people wearing a red cord (as they don't want to appear on pictures). I hope is fine.

There are more picture from the conference organization in better quality, which I will add as soon as they send them to me.

#26 - 08/30/2018 12:46 PM - mame (Yusuke Endoh)

I'm now for the addition. By the way, don't you need Array#intersection?

#27 - 09/13/2018 07:09 AM - matz (Yukihiro Matsumoto)

The final proposal seems reasonable. Accepted.

Matz.

#28 - 09/13/2018 08:35 PM - shevegen (Robert A. Heiler)

Funny idea with the picture. :D

#29 - 09/19/2018 10:11 PM - ana06 (Ana Maria Martinez Gomez)

matz (Yukihiro Matsumoto)

> The final proposal seems reasonable. Accepted.

Great to read that!

I have just rebased in the PR, looking forward to see the Array#union method in Ruby!

shevegen (Robert A. Heiler)

> Funny idea with the picture. :D

Thanks ;)

#30 - 09/20/2018 03:18 AM - nobu (Nobuyoshi Nakada)

- Status changed from Assigned to Closed

Applied in changeset trunk|r64787.

Add union method to Array

I introduce a union method equivalent to the | operator, but which
accept more than array as argument. This improved readability, and it
is also coherent with the + operator, which has a similar concat
method. The method doesn't modify the original object and return a new
object instead. It is plan to introduce a union! method as well.

Tests and documentation are included.

It solves partially https://bugs.ruby-lang.org/issues/14097

[Fix GH-1747] [Feature #14097]

From: Ana María Martínez Gómez ammartinez@suse.de

#31 - 09/30/2018 08:04 PM - ana06 (Ana Maria Martinez Gomez)

nobu (Nobuyoshi Nakada) this is not closed yet as https://github.com/ruby/ruby/pull/1758 is not merged yet. I have just rebased. ;)

#32 - 10/05/2018 11:49 AM - ana06 (Ana Maria Martinez Gomez)

Now it is closed! ;)