Ruby master - Feature #7739

Define Hash#| as Hash#reverse_merge in Rails

01/24/2013 09:57 PM - alexeymuranov (Alexey Muranov)

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

Description

=begin
I suggest for to define (({Hash#|})) as (({Hash#reverse_merge})) in ((*Rails*)), in my opinion this would correspond nicely to (({Set#|})), to the logical (({#||})) and to the bitwise (({#|})):

{ :a => 1, :b => 2 } | { :b => 1, :c => 2 } # => { :a => 1, :b => 1, :c => 2 }
=end

History

#1 - 01/25/2013 03:53 AM - nathan.f77 (Nathan Broadbent)

I would personally love a more concise way to merge/reverse_merge hashes. Would you also propose Hash#& as merge?

P.S. in your example, a reverse_merge should result in :b => 2

On Friday, 25 January 2013 at 1:57 AM, alexeymuranov (Alexey Muranov) wrote:

> Issue #7739 has been reported by alexeymuranov (Alexey Muranov).
>
> ---
>
> Feature #7739: Define Hash#| as Hash#reverse_merge in Rails
> https://bugs.ruby-lang.org/issues/7739
>
> Author: alexeymuranov (Alexey Muranov)
> Status: Open
> Priority: Normal
> Assignee:
> Category: core
> Target version:
>
> =begin
> I suggest for to define (({Hash#|})) as (({Hash#reverse_merge})) in ((*Rails*)), in my opinion this would correspond nicely to (({Set#|})), to the logical (({#||})) and to the bitwise (({#|})):
>
> { :a => 1, :b => 2 } | { :b => 1, :c => 2 } # => { :a => 1, :b => 1, :c => 2 }
> =end
>
> --
> http://bugs.ruby-lang.org/

#2 - 01/25/2013 07:22 AM - drbrain (Eric Hodel)

- *Status changed from Open to Assigned*
- *Assignee set to matz (Yukihiro Matsumoto)*
- *Target version set to 2.6*

We don't have any operations that use mathematical or logical operators for Hash.

It seems confusing to introduce an operator (|) when a name (reverse_merge) exists.

Hash and Set are different data structures and serve different purposes. The design of Hash should not be based on the design of Set.

#3 - 01/25/2013 06:32 PM - alexeymuranov (Alexey Muranov)

=begin
nathan.f77 (Nathan Broadbent) wrote:

> I would personally love a more concise way to merge/reverse_merge hashes. Would you also propose Hash#& as merge?

(({Hash#&})) would be the intersection of two hashes, i think:

{ :a => 1, :b => 2 } & { :b => 1, :c => 2 } # => { :b => 1 }

But this does not look very useful because the values of the first hash are ignored.

> P.S. in your example, a reverse_merge should result in :b => 2

You are right, sorry about the confusion, it should be

{ :a => 1, :b => 2 } | { :b => 1, :c => 2 } # => { :a => 1, :b => 2, :c => 2 }
=end

#4 - 01/25/2013 06:42 PM - alexeymuranov (Alexey Muranov)

drbrain (Eric Hodel), my idea was to propose a concise syntax to merge hashes (one or the other direction), and #| looked like a natural candidate.

Update: Also, #reverse_merge! can be replaced with #|=.
Update 2013-08-19: I have realized that "#|=" cannot be a name for "#reverse_merge!" because "#|=" cannot modify the receiver in place.

#5 - 02/08/2013 01:07 PM - PikachuEXE (Pikachu Leung)

I prefer using the name #reverse_merge
Since we don't have operator for #merge

Using an explicit name makes people try to understand what they are doing

#6 - 08/14/2013 03:48 AM - david_macmahon (David MacMahon)

=begin
alexeymuranov (Alexey Muranov) wrote:

> Update: Also, #reverse_merge! can be replaced with #|=.

This would make the "default values for options" idiom (shown in the example in the reverse_merge! documentation) both cleaner and clearer, IMHO.

Plain Ruby:

def setup(options = {})
options = { :size => 25, :velocity => 10 }.merge(options)
end

With reverse_merge!:

def setup(options = {})
options.reverse_merge! :size => 25, :velocity => 10
end

With |=:

def setup(options = {})
options |= { :size => 25, :velocity => 10 }
end

Because Hash#reverse_merge is an ActiveSupport feature, I don't think Hash#| should be an alias for it. Rather it should be the other way around: implement Hash#| as:

class Hash
def |(o)
o.merge(self)
end
end

and then let ActiveSupport make Hash#reverse_merge be an alias to Hash#|.
=end

#7 - 08/16/2013 12:08 AM - alexeymuranov (Alexey Muranov)

I have just realized that "#|=" cannot be a name for "#reverse_merge!" because "#|=" cannot modify the receiver in place.

#8 - 08/18/2013 03:27 AM - david_macmahon (David MacMahon)

=begin
While it's true that "#|=" cannot be a name for "#reverse_merge!", "#|" still can (and should, IMHO) be a name for "#reverse_merge", so using "hash |= other_hash" would still be useful (IMHO).

FWIW, I don't think one can even define an "#=" method for Ruby since the "LHS = RHS" is treated as "LHS = LHS RHS". Attempting to define a "#=" method from within Ruby results in a syntax error. I think attempting to do so from a C extension just creates a method that will never be called.

```
>> class Hash; def |=(o); self.reverse_merge!(o); end; end
SyntaxError: (irb):1: syntax error, unexpected tOP_ASGN
class Hash; def |=(o); self.reverse_merge!(o); end; end
                  ^
(irb):1: syntax error, unexpected keyword_end, expecting end-of-input
        from /Users/davidm/local/bin/irb20:12:in `<main>'

>> class Hash; def |(o); o.merge(self); end; end
=> nil
```

=end

#9 - 08/18/2013 09:44 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

As I stated in #8772, I believe #| being implemented as #reverse_merge instead of #merge is confusing. I believe the original example in the description makes sense though.

#10 - 08/18/2013 04:59 PM - david_macmahon (David MacMahon)

On Aug 17, 2013, at 5:44 PM, rosenfeld (Rodrigo Rosenfeld Rosas) wrote:

> As I stated in #8772, I believe #| being implemented as #reverse_merge instead of #merge is confusing. I believe the original example in the description makes sense though.

Why do you think that...

```
 {a:1, b:1} | {b:2, c:2) #=> {a:1, b:2, c:2} # merge-like
```

...is less confusing than...

```
 {a:1, b:1} | {b:2, c:2) #=> {a:1, b:1, c:2} # reverse_merge-like
```

...? For conflicting keys, the reverse_merge-like behavior gives precedence to the values of the left hand side, which I find reminiscent of the short-circuit behavior of || and &&. The merge-like behavior gives precedence to the values of the right hand side, which seems unusual IMHO.

In addition to the short-circuit behavior of || and &&, Array#| also already gives precedence to the left hand side by preserving the order of the original array:

```
 [ "a", "b", "c" ] | [ "c", "d", "a" ] #=> [ "a", "b", "c", "d" ]
```

> Those alias would already be useful in cases people are using reverse_merge:
>
> options.reverse_merge! a: 1
>
> options = {a: 1} | options # not exactly the same, but usually has the same effect on most well written code

But wouldn't "options |= {a: 1}" be even more concise and similar in form to the "foo ||= 1" idiom? By having Hash#| be an alias for reverse_merge, this also wouldn't be exactly the same as reverse_merge!, but I suspect it would be effectively the same for a majority of uses. For "options" being passed in as a method parameter, I think it would be better than reverse_merge! because it doesn't modify/pollute the caller's Hash. There is a performance trade off (create new object and not pollute caller's Hash vs no new object but pollute caller's Hash) that needs to be considered, but for performance sensitive cases one could use reverse_merge! explicitly.

> I'd even be fine with #>> as an alias for reverse_merge!
>
> options >>= {a: 1} # options.reverse_merge! a: 1

I'm not opposed to #>>= being reverse_merge!. I'm not thrilled with it either, but maybe it's just my C++ extraction operator days haunting me! :-)

Dave

#11 - 08/18/2013 06:19 PM - alexeymuranov (Alexey Muranov)

=begin
rosenfeld (Rodrigo Rosenfeld Rosas) wrote:

As I stated in #8772, I believe #| being implemented as #reverse_merge instead of #merge is confusing. I believe the original example in the description makes sense though.

Rodrigo, compare this (with the originally proposed behavior):

1 || 2 # => 1
{ :a => 1 } | { :a => 2 } # => { :a => 1 }

((*Update.*)) I agree that there would be no perfect consistency in any case:

nil || 2 # => 2
{ :a => nil } | { :a => 2 } # => { :a => nil }
=end

#12 - 08/18/2013 09:25 PM - trans (Thomas Sawyer)

To condense the discussion down, would we agree to:

```
class Hash
  def reverse_merge(h)
    h.merge(self)
  end

  def reverse_merge!(h)
    h.merge!(self)
  end

  def <<(pairs)
    merge!(Hash === pairs ? pairs : Hash[*pairs])
  end

  def >>(pairs)
    reverse_merge!(Hash === pairs ? pairs : Hash[*pairs])
  end

  alias | reverse_merge
end
```

That still leaves out an operator alias for #merge itself though. Do we need it, or is #| enough?

#13 - 08/18/2013 10:14 PM - trans (Thomas Sawyer)

Oops, fixed. Thanks. I've been writing Elixir code lately :-].

#14 - 08/19/2013 03:45 AM - david_macmahon (David MacMahon)

I certainly agree with everything in (the modified) [ruby-core:56720]#12 posting, except for the implementation of #reverse_merge!. As (currently) written, it modifies the operand in place rather than the receiver, but I suspect that I agree with its original intent.

As for an operator alias for #merge, dare I suggest Hash#+? It would mean "take everything in the first (i.e. LHS) hash and add in everything from the second (i.e. RHS) hash". For conflicting keys, the left-to-right evaluation of "+" would imply that values on the right hand side would take precedence over values on the left hash side.

#15 - 08/19/2013 04:34 AM - fuadksd (Fuad Saud)

=begin
Matz rejected (({+})) for merging, as it is different than addition. reverse_merge would probably be closer to the concept od addition, as it only adds data to the receiver. Also, give this, isn't the reverse_merge less appropriate name than add, or something else that suggest this behaviour. I'm ok with reverse_merge, but maybe this is food for thought.

Any reason why shouldn't the polymorphic behaviour be incorporated into the merge/reverse_merge methods? Maybe they could try to convert the arg to a hash before - I'm not sure about this though.

I'd stick with

- ((<<)): merge!
- ((>>)): reverse_merge!
- ((+)): reverse_merge
- ((&)) or ((/)): merge

Also, just a comment: thinking about this, it would be nice too have the except method "imported" from ActiveSupport as well and have ((-)) aliased to it.

=end

#16 - 08/19/2013 05:23 AM - david_macmahon (David MacMahon)

On Aug 18, 2013, at 12:34 PM, fuadksd (Fuad Saud) wrote:

> Matz rejected (({+})) for merging, as it is different than addition.

My understanding was that Matz rejected #+ for merging because of a concern about how to resolve key conflicts. Maybe his concern can be addressed by providing two Hash operators (e.g. #| for reverse_merge-like behavior and #+ for merge-like behavior) each with distinct and well defined resolutions for conflicting keys.

Dave

#17 - 08/19/2013 05:47 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

david_macmahon (David MacMahon) wrote:

> On Aug 17, 2013, at 5:44 PM, rosenfeld (Rodrigo Rosenfeld Rosas) wrote:
>
> > As I stated in #8772, I believe #| being implemented as #reverse_merge instead of #merge is confusing. I believe the original example in the description makes sense though.
>
> Why do you think that...
>
> ```
> {a:1, b:1} | {b:2, c:2) #=> {a:1, b:2, c:2} # merge-like
> ```
>
> ...is less confusing than...
>
> ```
> {a:1, b:1} | {b:2, c:2) #=> {a:1, b:1, c:2} # reverse_merge-like
> ```
>
> ...?

I'm not sure to be honest. I believe that's because in my head when I perform set1 |= set2 I read it as "discard any duplicate values in set2 when merging the values to set1" or "merge all values from set2 that are not present in set1". I never think of it as "remove any values in set1 that also happen to be present in set 2 then add all values of set2 to set1".

#18 - 08/19/2013 05:50 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Thomas, with regards to your example, I'd agree with all of it but aliasing #| to #reverse_merge instead of #merge. Having said that, in case the options are aliasing #| to #reverse_merge or not aliasing it at all, I'd prefer the former.

#19 - 08/19/2013 06:24 AM - alexeymuranov (Alexey Muranov)

rosenfeld (Rodrigo Rosenfeld Rosas) wrote:

> I'm not sure to be honest. I believe that's because in my head when I perform set1 |= set2 I read it as "discard any duplicate values in set2 when merging the values to set1" or "merge all values from set2 that are not present in set1". I never think of it as "remove any values in set1 that also happen to be present in set 2 then add all values of set2 to set1".

How it this different from the proposed behavior for hashes? hash1 |= hash2 discards any duplicate values in hash2 and adds the rest to hash1. ("Duplicate" here means corresponding to the same key.)

#20 - 08/19/2013 07:07 AM - phluid61 (Matthew Kerwin)

=begin
I still believe the original proposal is the most useful:

```
class Hash
  def | other_hash
    other_hash.merge self
  end
end
```

Or, if taking the code from active support:

```
class Hash
  def reverse_merge(other_hash) other_hash.merge(self); end
  alias :| :reverse_merge
end
```

Use-case:

```
def foo options={}
```

```
  options |= {a:1, b:2}
end
```

Note: this is different from kwargs if you're going to operate on the hash as a whole:

```
def foo options={}
  options |= {a: 1, b: 2}
  do_something_with options
end

def bar a: 1, b: 2, **options
  #do_something_with( {a: a, b: b} + options ) ??
end
```

=end

#21 - 08/19/2013 08:59 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Yes, you're right, I'm convinced now. +1 for it aliasing #| to
#reverse_merge.
Em 18/08/2013 18:25, "alexeymuranov (Alexey Muranov)" redmine@ruby-lang.org
escreveu:

> Issue #7739 has been updated by alexeymuranov (Alexey Muranov).
>
> rosenfeld (Rodrigo Rosenfeld Rosas) wrote:
>
>> I'm not sure to be honest. I believe that's because in my head when I
>> perform set1 |= set2 I read it as "discard any duplicate values in set2
>> when merging the values to set1" or "merge all values from set2 that are
>> not present in set1". I never think of it as "remove any values in set1
>> that also happen to be present in set 2 then add all values of set2 to
>> set1".
>
> How it this different from the proposed behavior for hashes? hash1 |=
>
> # hash2 discards any duplicate values in hash2 and adds the rest to hash1.
>
> Feature #7739: Define Hash#| as Hash#reverse_merge in Rails
> https://bugs.ruby-lang.org/issues/7739#change-41261
>
> Author: alexeymuranov (Alexey Muranov)
> Status: Assigned
> Priority: Normal
> Assignee: matz (Yukihiro Matsumoto)
> Category: core
> Target version: next minor
>
> =begin
> I suggest for to define (({Hash#|})) as (({Hash#reverse_merge})) in
> ((*Rails*)), in my opinion this would correspond nicely to (({Set#|})), to
> the logical (({#||})) and to the bitwise (({#|})):
>
> { :a => 1, :b => 2 } | { :b => 1, :c => 2 } # => { :a => 1, :b => 1, :c
> => 2 }
> =end
>
> --
> http://bugs.ruby-lang.org/

#22 - 08/19/2013 10:23 AM - fuadksd (Fuad Saud)

Sorry, I swapped | for + in the last message.
On Aug 18, 2013 5:01 PM, "David MacMahon" davidm@astro.berkeley.edu wrote:

> On Aug 18, 2013, at 12:34 PM, fuadksd (Fuad Saud) wrote:
>
>> Matz rejected (({+})) for merging, as it is different than addition.
>
> My understanding was that Matz rejected #+ for merging because of a
> concern about how to resolve key conflicts. Maybe his concern can be

addressed by providing two Hash operators (e.g. #| for reverse_merge-like
behavior and #+ for merge-like behavior) each with distinct and well
defined resolutions for conflicting keys.

Dave

#23 - 12/25/2017 06:15 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)