Ruby - Bug #21396

Set#initialize should call Set#add on items passed in

06/04/2025 07:31 PM - tenderlovemaking (Aaron Patterson)

Status:	Open		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:		**Backport:**	3.2: UNKNOWN, 3.3: UNKNOWN, 3.4: UNKNOWN

Description

```
class Foo < Set
  def add(item) = super(item.bytesize)
end

x = Foo.new(["foo"])
p x
p x.include?(3)
```

On Ruby 3.4 the output is this:

```
> ruby -v test.rb
ruby 3.4.1 (2024-12-25 revision 48d4efcb85) +PRISM [arm64-darwin24]
#<Foo: {3}>
true
```

On Ruby master the output is this:

```
> make run
./miniruby -I./lib -I. -I.ext/common  -r./arm64-darwin24-fake  ./test.rb
#<Set: {"foo"}>
false
```

The bug is that initialize is not calling add for the elements passed in, so the subclass doesn't get a chance to change them.

I've sent a PR here: https://github.com/ruby/ruby/pull/13518

Related issues:

Related to Ruby - Bug #21375: Set[] does not call #initialize	**Open**

History

#1 - 06/04/2025 07:41 PM - jeremyevans0 (Jeremy Evans)

- Related to Bug #21375: Set[] does not call #initialize added

#2 - 06/04/2025 07:43 PM - jeremyevans0 (Jeremy Evans)

This is not a bug, IMO. Using underlying functions instead of calling methods was one of the deliberate design decisions for core Set (see #21216), and how other core collection classes work. Array.new(1, true) does not call Array#[]=, it calls rb_ary_store.

If we want to do this for Set#initialize and Set.[] for backwards compatibility, we should be consistent and call methods instead of underlying functions for every case where the methods were called in stdlib set. That will make it slower, though.

FWIW, in the code path you are using in stdlib Set, Set#add is not called directly, you are relying on Set#merge calling it. So this is at least a request to have Set#initialize call Set#merge and to have Set#merge call Set#add for every element.

#3 - 06/04/2025 07:48 PM - tenderlovemaking (Aaron Patterson)

jeremyevans0 (Jeremy Evans) wrote in #note-2:

> This is not a bug, IMO. Using underlying functions instead of calling methods was one of the deliberate design decisions for core Set (see #21216), and how other core collection classes work. Array.new(1, true) does not call Array#[]=, it calls rb_ary_store.
>
> If we want to do this for Set#initialize and Set.[] for backwards compatibility, we should be consistent and call methods instead of underlying functions for every case where the methods were called in stdlib set. That will make it slower, though.

> FWIW, in the code path you are using in stdlib Set, Set#add is not called directly, you are relying on Set#merge calling it. So this is at least a request to have Set#initialize call Set#merge and to have Set#merge call Set#add for every element.

I don't have an opinion, really. But this change in behavior is breaking our existing code, and I suspect we're not the only ones.

#4 - 06/05/2025 01:08 AM - ko1 (Koichi Sasada)

How about to redfine initialize on subclass of Set to call #add?

#5 - 06/05/2025 08:34 AM - Eregon (Benoit Daloze)

Is there any public code in some gem or so that relies on this? (the example is rather synthetic)

#6 - 06/05/2025 01:43 PM - knu (Akinori MUSHA)

I've always created custom variants of Set too, and I don't think it's rare to find these in in-house codebases.

#7 - 06/05/2025 05:05 PM - tenderlovemaking (Aaron Patterson)

ko1 (Koichi Sasada) wrote in #note-4:

> How about to redfine initialize on subclass of Set to call #add?

I think we could, but that means people have to change their code when upgrading.

Eregon (Benoit Daloze) wrote in #note-5:

> Is there any public code in some gem or so that relies on this? (the example is rather synthetic)

It's kind of hard to search for this on GitHub, but I was able to find two similar-ish examples:

This one would not raise an exception in the same way, and this one could potentially be missing entries from its @hash_lookup instance variable.

Maybe we could publish a gem that just has a copy of the current set.rb file, but with Set renamed to RbSet or something. Then if people run in to issues they can use the gem and change the superclass to RbSet. I think it would fix our case at work and probably the cases referenced above. I agree with @jeremyevans0's points, but I'm sure this is going to cause friction for people upgrading and it would be nice if we can make it as easy as possible to upgrade.