Provide a public WeakMap that compares by equality rather than by identity

I know ObjectSpace::WeakMap isn't really supposed to be used, and that the blessed interface is WeakRef. However, I'd like to make a case for a better public WeakMap.

Usage

As described in [Feature #16035], WeakMap is useful for deduplicating "value objects". A typical use case is as follows:

```ruby
class Position
  REGISTRY = {}
  private_constant :REGISTRY

  class << self
    def new(*)
      instance = super
      REGISTRY[instance] ||= instance
    end
  end

  attr_reader :x, :y, :z

  def initialize(x, y, z)
    @x = x
    @y = y
    @z = z
    freeze
  end

  def hash
    self.class.hash ^
    x.hash >> 1 ^
    y.hash >> 2 ^
    z.hash >> 3
  end

  def ==(other)
    other.is_a?(Position) &&
    other.x == x &&
    other.y == y &&
    other.z == z
  end

  alias_method :eql?, :==
end
```

```ruby
p Position.new(1, 2, 3).equal?(Position.new(1, 2, 3))
```

That's pretty much the pattern I used in Rails to deduplicate database metadata and save lots of memory.

The big downside here is that these value objects can't be GCed anymore, so this pattern is not viable in many cases.

Why not use WeakRef

A couple of reasons. First, when using this pattern, the goal is to reduce memory usage, so having one extra WeakRef for every single value object is a bit counter productive.
Then it's a bit annoying to work with, as you have to constantly check whether the reference is still alive, and/or rescue \texttt{WeakRef::RefError}.

Often, these two complications make the tradeoff not worth it.

**Ruby 2.7**

Since [Feature #13498](https://bugs.ruby-lang.org/issues/13498) \texttt{WeakMap} is a bit more usable as you can now use an interned string as the unique key, e.g.

```ruby
class Position
  REGISTRY = ObjectSpace::WeakMap.new
  private_constant :REGISTRY

  class << self
    def new(*)
      instance = super
      REGISTRY[instance.unique_id] ||= instance
    end
  end

  attr_reader :x, :y, :z, :unique_id

  def initialize(x, y, z)
    @x = x
    @y = y
    @z = z
    @unique_id = "#{self.class}#{x},#{y},#{z}"
    freeze
  end

  def hash
    self.class.hash ^
    x.hash >> 1 ^
    y.hash >> 2 ^
    y.hash >> 3
  end

  def ==(other)
    other.is_a?(Position) &&
    other.x == x &&
    other.y == y &&
    other.z == z
  end

  alias_method :eql?, :==
end
```

That makes the pattern much easier to work with than dealing with \texttt{WeakRef}, but there is still that an extra instance.

**Proposal**

What would be ideal would be a \texttt{WeakMap} that works by equality, so that the first snippet could simply replace {} by \texttt{WeakMap.new}.

Changing ObjectSpace::WeakMap's behavior would cause issues, and I see two possibilities:

- The best IMO would be to have a new top level ::WeakMap be the equality based map, and have ObjectSpace::WeakMap remain as a semi-private interface for backing up \texttt{WeakRef}.
- Or alternatively, ObjectSpace::WeakMap could have a \texttt{compare_by_equality} method (inverse of \texttt{Hash#compare_by_identity}) to change its behavior post instantiation.

I personally prefer the first one.

**Related issues:**

Related to Ruby master - Feature #16471: Two feature requests for WeakRef: ge... Open

08/06/2021
History

#1 - 08/05/2019 05:20 PM - sawa (Tsuyoshi Sawada)
- Description updated
- Subject changed from Provide a public WeakMap which compare by equality rather than identity to Provide a public WeakMap that compares by equality rather than by identity

#2 - 08/29/2019 06:45 AM - matz (Yukihiro Matsumoto)
I am not sure if the proposal has real-world use-case. Can you elaborate?
Matz.

#3 - 08/29/2019 10:12 AM - byroot (Jean Boussier)

matz (Yukihiro Matsumoto) Of course. What prompted me to open this feature request is a feature I implemented in Rails where I use this exact pattern: https://github.com/rails/rails/pull/35891 (more specifically https://github.com/rails/rails/blob/48ae1ba36132d7deec0bd43ee50e076786011a5e/activerecord/lib/active_record/connection_adapters/deduplicable.rb)

In short Rails ORM keep a data structure in memory that reflect the database schema, and that structure contains quite a lot of duplications, for instance all tables have an id BIGINT PRIMARY KEY column, so if you have 100 tables, you'll end up with 100 identical Column objects in memory.

By using the pattern described in this issue, I was able to get rid of all the deduplications, which resulted in massive memory saving (over 115MB for us, but granted our schema is massive).

But my worry now is that I made all these objects totally non garbage collectable, so if someone is instantiating these dynamically, it might cause a RAM based DOS vulnerability, an attacker could trigger a lot of instantiations until the application runs out of RAM.

That is why I'd like to use a WeakMap instead, but for that I need it to have an equality semantic like a regular hash. If it existed, I believe this pattern could be used in many libraries and applications which can't do it today because it's too risky.

#4 - 01/02/2020 12:54 AM - nobu (Nobuyoshi Nakada)
- Related to Feature #16471: Two feature requests for WeakRef: get original object, callback feature added

#5 - 05/11/2021 09:08 AM - byroot (Jean Boussier)
I'd like this to be reconsidered. I'll add it to the next developer meeting.

#6 - 05/11/2021 03:00 PM - marcandre (Marc-Andre Lafortune)

Seems reasonable to me.

I also wish that WeakMap was officially public, it can be quite useful. I wish it's interface was closer to Hash.

I'd suggest an option be added to the initializer of ObjectSpace::WeakMap.

#7 - 05/12/2021 11:17 AM - Eregon (Benoit Daloze)

marcandre (Marc-Andre Lafortune) wrote in #note-6:

I'd suggest an option be added to the initializer of ObjectSpace::WeakMap.

I think we should have a separate class if this feature is accepted. It's already complicated enough to implement ObjectSpace::WeakMap, I wouldn't want to have to dynamically handle how comparison is done for keys.

For instance on TruffleRuby one would likely need to wrap keys if we compare by hash/eql?. If we have to selectively wrap it gets ugly and error-prone.

#8 - 05/20/2021 06:34 AM - nobu (Nobuyoshi Nakada)

Your example always allocates a new instance first, then deduplicates it. Why not:

```
def new(x, y, z)
  REGISTRY[-"#{self}#{x}#{y}#{z}"] ||= super
end
```

This would work even in 2.7 too.

#9 - 05/20/2021 06:46 AM - byroot (Jean Boussier)
Your example always allocates a new instance first, then deduplicates it.

Yes, for more complex cases that's kind of the only way. What I'm after here is mostly memory retention, not so much allocations.

Why not:

Yes, I used this in some cases when the number of property was slow enough. But:

- The object need to hold that string.
- That just doubled the number of retained objects, just like WeakRef.
- Not everything is well suited to be concatenated as a string like this.

And even for your approach, equality based WeakMap would offer a much cleaner interface:

```ruby
def new(x, y, z)
  REGISTRY[[x, y, z]] ||= super
end
```

---

**#10 - 05/24/2021 05:39 PM - mame (Yusuke Endoh)**

This ticket was discussed in the previous dev meeting. The use case is understandable, but matz (Yukihiro Matsumoto) was not completely convinced that WeakMap is a suitable solution for that. He said he would reply.

In addition, ko1 (Koichi Sasada) said that it would be (possible but) tough to implement the feature. For lookup, it is safe to assume that a key is reachable (otherwise, the key is never looked up). However, the assumption is not safe for compare_by_equality. Also, we need to care a GC during the equality check.

This is just my opinion. I want this kind of feature (non-memory-leakable hash for object dedup), but I'm unsure whether WeakMap is actually usable for that because it is difficult to predict its behavior. It has no configurable size limit, and it may forget almost all contents at every GC (depending upon an application). Instead of exploiting WeakMap, a dedicated cache library makes more sense.

---

**#11 - 05/25/2021 01:39 PM - Eregon (Benoit Daloze)**

mame (Yusuke Endoh) wrote in #note-10:

This is just my opinion. I want this kind of feature (non-memory-leakable hash for object dedup), but I'm unsure whether WeakMap is actually usable for that because it is difficult to predict its behavior. It has no configurable size limit, and it may forget almost all contents at every GC (depending upon an application).

It's the same or similar behavior for Java WeakHashMap.

When used as a cache, then whatever uses it should keep a strong reference to the key (value if it's a weak values map), so the entry stays alive as long as there are users of it, and it's bounded by the number of usages.

Not keeping a reference to the key (value if it's a weak values map) is a bug, then that cache is of no use as it would indeed be empty after every GC.

byroot (Jean Boussier) Actually, something is not clear in this proposal: do you want weak keys or weak values?

For deduplication, weak values often make sense, then one only needs to hold to the value to keep the entry alive.

My understanding is that ObjectSpace::WeakMap is weak values, and compare_by_identity.

---

**#12 - 05/25/2021 01:48 PM - byroot (Jean Boussier)**

My understanding is that ObjectSpace::WeakMap is weak values

I believe it's both weak keys and weak values:

```ruby
>> m = ObjectSpace::WeakMap.new
=> #<ObjectSpace::WeakMap:0x00007fa56bf75ea0>  
>> Object.new.tap { |o| m[o] = o }  
=> #<Object:0x00007fa575c167f0>  
>> m.size  
=> 1  
>> 4.times { GC.start }  
=> 4  
>> m.size  
=> 0
```

And that's essentially the behavior I'm after, as the deduplication code, as defined in the description: REGISTRY[instance] ||= instance. E.g. you create a new instance, and then check for a pre-existing equal one.
Same as String#-@, you first need to build the string, and then can deduplicate it.

#13 - 05/25/2021 01:51 PM - byroot (Jean Boussier)

Instead of exploiting WeakMap, a dedicated cache library makes more sense.

I'm not sure how one could write such library today without a WeakMap (or other Weak\* datastructure) like I'm requesting.

#14 - 05/25/2021 04:37 PM - Eregon (Benoit Daloze)

byroot (Jean Boussier) wrote in #note-12:

My understanding is that ObjectSpace::WeakMap is weak values

I believe it's both weak keys and weak values:

I actually asked this question on the ruby-core Slack a while ago and IIRC I could not get a clear answer. Maybe nobu (Nobuyoshi Nakada) knows?

I'm not sure both weak keys and weak values make sense. For instance, if one of them is no longer referenced (e.g., the key), should the map remove the entry (even if the value is still referenced; which means two equivalent instances could be alive => sounds bad)?

If the logic is to keep the entry as long as either the key or value is alive, that sounds like it needs very deep GC support. At least it doesn't seem possible to implement just with WeakReferences (unless I'm missing something).

TruffleRuby and JRuby currently implement ObjectSpace::WeakMap with weak values (and strong keys). I am concerned it might not be possible to have both weak keys and weak values on the JVM, and probably it's also hard with some GCs (seems to need: an object cannot die while another, unrelated by reference, object is alive).

cc headius (Charles Nutter)

The specific case you illustrate sounds like a "deduplication set" (my words, probably not a well-known term). That specifically might be easier to implement and safer (since the key and value of an entry are the same, so they naturally GC at the same time).

#15 - 05/25/2021 05:12 PM - mame (Yusuke Endoh)

byroot (Jean Boussier) wrote in #note-13:

I'm not sure how one could write such library today without a WeakMap (or other Weak\* datastructure) like I'm requesting.

I guess that it is possible to write such library in C extension by using finalizers, but some APIs might lack.

#16 - 05/25/2021 05:32 PM - byroot (Jean Boussier)

I guess that it is possible to write such library in C extension by using finalizers

Ah I see. Yeah I didn't think about C-exts. That could work, would probably need to pin all references to avoid compaction problems etc.

But it could allow to experiment more with the idea.