Ruby - Feature #17685

Marshal format for out of band buffer objects

03/10/2021 05:43 PM - dsisnero (Dominic Sisneros)

Status:	Feedback	
Priority:	Normal	
Assignee:		
Target version:		

Description

Allow the use of the marshal protocol to transmit large data (objects) from one process or ractor to another, on same machine or multiple machines without extra memory copies of the data.

See Python PEP 574 - https://www.python.org/dev/peps/pep-0574/ Pickle protocol with out of band data.

When marshalling memoryview objects, it would be nice to be able to use zero copy loads of the memoryviews. That way when loading the file we can use that memoryview without copying it also if desired.

Add a Marshal::Buffer type in new version of Marshal to represent something that indicates a serializable no-copy buffer view.

The marshal_dump must be able to represent references to a Marshal::Buffer to indicate that the loader might get the actual buffer out of band

The marshal_load must be able to provide the Marshal::Buffer for deserialization

Marshal load and dump should work normally if not used out of band.

```
class Apache::Arrow

  def marshal_dump(*)
     if marshal.version > '0.4'
         Marshal::Buffer.new(self)
     else
        #normal dump
     end
  end
end
```

History

#1 - 03/11/2021 02:52 AM - mrkn (Kenta Murata)

Do you want the way to load and dump the memory view metadata of any objects that support exporting their memory view?

Could you please tell me the example use cases you've assumed?

#2 - 03/11/2021 02:52 AM - mrkn (Kenta Murata)

- Description updated

#3 - 03/11/2021 09:38 PM - dsisnero (Dominic Sisneros)

On the consumer side, we can Marshal those objects the usual way, which when unserialized will give us a copy of the original object:

b = ZeroCopyByteArray.new("abc".bytes)
data = Marshal.dump(b)
new_b = Marshal.load(data)
puts b == new_b # True
puts b.equal? new_b # False: a copy was made
But if we pass a buffer_callback and then give back the accumulated buffers when unserializing, we are able to get back the original object:

b = ZeroCopyByteArrayi.new("abc".bytes)
buffers = []
data = Marshal.dump(b, buffer_callback: buffers.method('append')
new_b = Marshal.load(data, buffer: buffers)
puts b == new_b # True
puts b.equal? new_b # True: no copy was made

```
class ZeroCopyByteArray < Arrow::Buffer

def _dump()
if Marshal.protocol >= 5
return self.class._reconstruct(MarshalBuffer.new(self), nil
else
# PickleBuffer is forbidden with Marshal protocols <= 4.
return type(self)._reconstruct, (bytearray(self),)
end

def self._load( obj)
m = MemoryView.new(obj)
obj = m.obj
if obj.class == self.class
return obj
else
return new(obj)
end
end

end
```

#4 - 03/16/2021 07:49 AM - mrkn (Kenta Murata)

You cannot get the original object from Marshal.load. This is Marshal.load's nature.
Marshal.load always creates a new object (the different object from the original one).
Object#equal? compares object identities, so b.equal? new_b is always false.

#5 - 03/16/2021 11:48 PM - dsisnero (Dominic Sisneros)

that is the case now. I am proposing changing Marshal to allow Marshal to load into an existing object for object identities. This is one of the things python's latest pickle format allows. They use it to marshal large numpy arrays to a distributed object store. See my original link.
https://www.python.org/dev/peps/pep-0574/

#6 - 03/17/2021 12:00 AM - mrkn (Kenta Murata)

The object identity in Ruby is defined by the value of object_id. Object#equal? just compares the value of object_id.
No more than one object has the same value of object_id.

Marshal cannot generate an object whose equal? returns true for the other object because no more than one objects have the same value of object_id.

What is the reason why you stick to equal? method and Marshal combination? Doesn't == work well for your purpose?

#7 - 03/24/2021 07:39 AM - mrkn (Kenta Murata)

- Status changed from Open to Feedback