Ruby master - Feature #7429

Provide options for core collections to customize behavior

11/25/2012 01:49 AM - headius (Charles Nutter)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td></td>
</tr>
<tr>
<td>Target version</td>
<td>2.0.0</td>
</tr>
</tbody>
</table>

Description

Many folks know that Matz is a fan of having a few classes that handle a wide range of behavior. For this reason, I think we're unlikely to get a set of parallelism-safe collections added to Ruby. I propose an alternative that would allow parallel-executing implementations to offer concurrency-friendly versions without impact to any code.

I would like to see a way to pass in an options hash to the `.new` construction of at least Hash and Array. For example:

**Create a hash that is concurrent-safe, resizes when density is > 3 keys per bucket,**

and sets initial bucket count to 60

```ruby
hsh = Hash.new(concurrent: true, density: 3, size: 60)
```

Similar for Array:

```ruby
ary = Array.new(concurrent: true, size: 100)
```

Options like density and size map directly to current implementation details of Hash and Array that are sometimes not accessible (you can't change the load factor for Hash, for example). Options like concurrent could be noops on MRI but would allow other implementations to provide safe versions of the collections behind the scenes.

I know it may be too late for 2.0.0, but we implementers of parallel-executing Rubies feel the pain of thread-unsafe core structures every day. I think the ability to get thread-safe collections in Ruby core would go a long way toward dispelling the myth that Ruby is bad for concurrent application programming.

History

#1 - 11/25/2012 09:55 AM - mame (Yusuke Endoh)
- Tracker changed from Bug to Feature

Too late completely, as you said.

--
Yusuke Endoh mame@tsg.ne.jp

#2 - 11/25/2012 09:54 PM - matz (Yukihiro Matsumoto)
- Status changed from Open to Rejected

Even though I prefer smaller set of built-in fundamental classes, I don't think 'concurrent' option is sufficient, since it requires totally different implementation. I think it's rather better for you to propose 'ParallelHash' etc.

Matz.

#3 - 11/25/2012 11:19 PM - trans (Thomas Sawyer)
=begin
I wonder if concurrency behavior can be designed as a mixin. As long as the underlying class conforms to its interface then "ParallelWhatever" becomes possible. ParallelHash would then not be needed as a built-in class b/c it would be simple to define.

```ruby
class MyParallelHash < Hash
  include Concurrency
end
=end
matz (Yukihiro Matsumoto) wrote:

Even though I prefer smaller set of built-in fundamental classes, I don't think 'concurrent' option is sufficient, since it requires totally different implementation. I think it's rather better for you to propose 'ParallelHash' etc.

That was indeed my intention: concurrent: true would allow using a completely different implementation under the covers, allowing for an efficient concurrent hash/array (rather than one that simply locks on all accesses).

I will repropose as new collection types. Is there any chance of getting them into 2.0.0?

trans (Thomas Sawyer) wrote:

I wonder if concurrency behavior can be designed as a mixin. As long as the underlying class conforms to its interface then "ParallelWhatever" becomes possible. ParallelHash would then not be needed as a built-in class b/c it would be simple to define.

```
class MyParallelHash < Hash
  include Concurrency
end
```

Actually, JRuby already provides this as JRuby::Synchronized:

```
class MyParallelHash < Hash
  include JRuby::Synchronized
end
```

It's rather hacky though for a few reasons:

- In JRuby, the JRuby::Synchronized module inserts itself into the method lookup process, returning all methods wrapped with synchronization against the target object.
- The synchronization is very coarse-grained and much slower than other implementations of a concurrent data structure that would use fewer (or no) locks.
- It is not common to have the inclusion of a module change the nature of every method in the host class.

It would be better if we had some fast, always-concurrent data structures built in.

That is interesting. I suspect part of the problem is the design of the Hash class itself --something I addressed before in #6442. If Hash had a core set of non-reducible methods which all the others depended, then those should be the only methods that would need to be targeted. Also, #prepend might be handy here, rather than "inserts itself into the method lookup process".

Speed isn't everything. I think good design is at least, if not more, important.

Speed isn't everything...until it becomes everything. Ruby should not be wasteful unnecessarily. There are also immutable truths about data structure implementation, like the fact that adding thread-safety to most data structures comes at a cost...which is why in JRuby we decided not to make Hash and Array thread-safe by default.

There are also a limited set of data structures that make unsynchronized reads concurrent with writes safe. You usually have to have a data structure that's designed to do both safely or they both have to be synchronized.

For Hash at least, I think something like a CTrie (concurrent hash array mapped trie) would be a nice choice. It's not O(1) but it's pretty good.

I am afraid it's not going to 2.0, for some reasons:

- it's too late to add new classes to 2.0
- and it will take more time to prepare parallel collection implementation for other Ruby (in this case mainly CRuby).
- the future CRuby will lean toward non threading way, e.g, Actors.

Matz.

matz (Yukihiro Matsumoto) wrote:
headius (Charles Nutter) I am afraid it's not going to 2.0, for some reasons:

- it's too late to add new classes to 2.0
- and it will take more time to prepare parallel collection implementation for other Ruby (in this case mainly CRuby).

In CRuby, with the GIL, you can pretend the existing ones are parallel-safe, can't you?

- the future CRuby will lean toward non threading way, e.g. Actors.

If those actors are going to run in parallel, you'll still need to have parallel-safe collections to pass between them...or an explicit mechanism for preventing users from passing mutable state between actors.

#10 - 09/27/2013 01:02 AM - dsisnero (Dominic Sisneros)

Maybe combine it with https://bugs.ruby-lang.org/issues/8909

options = {klass:Hamster}

{ bug_number: 7429, status: maybe}.f(options)

node_option = {f:deep , size: 2}

[:add, [:left, :right]].f(f:node_option)