Ruby master - Bug #5261
Symbol#to_proc memory leak in 1.9.x
09/02/2011 12:47 AM - ninkendo (Ken Simon)

<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>
<tr>
<td>Backport:</td>
<td></td>
</tr>
<tr>
<td>ruby -v:</td>
<td>ruby 1.9.2p290 (2011-07-09 revision 32553) [x86_64-linux]</td>
</tr>
</tbody>
</table>

**Description**

It appears that running an array through `.map(&:foo)` leaks the array's contents, and they don't get picked up by the Garbage Collector.

Given a simple class:

```ruby
class C
  def foo
    "foo"
  end
end
```

The following appears to leave references around (1.9.3-preview1 irb session shown, ruby -v gives ruby -v ruby 1.9.3dev (2011-07-31 revision 32789) [x86_64-darwin11.1.0]):

```ruby
ruby-1.9.3-preview1 :001 > a = 10.times.map{C.new}
=> [... snip ...]
ruby-1.9.3-preview1 :002 > b = a.map(&:foo)
ruby-1.9.3-preview1 :003 > a = b = nil
=> nil
ruby-1.9.3-preview1 :004 > GC.start
=> nil
ruby-1.9.3-preview1 :005 > ObjectSpace.each_object(C){}
=> 10
```

If I instead run a through the block form of map, the GC collects the objects as expected:

```ruby
ruby-1.9.3-preview1 :001 > a = 10.times.map{C.new}
=> [... snip ...]
ruby-1.9.3-preview1 :002 > b = a.map{|x| x.foo}
ruby-1.9.3-preview1 :003 > a = b = nil
=> nil
ruby-1.9.3-preview1 :004 > GC.start
=> nil
ruby-1.9.3-preview1 :005 > ObjectSpace.each_object(C){}
=> 0
```

The same issue happens in 1.9.2-p180 and 1.9.2-p290, Linux and Darwin, but not in any 1.8 releases I've tried.

Also, as Niklas reported in the StackOverflow post I made about this (http://stackoverflow.com/questions/7263268/ruby-symbol#to-proc-leaks-references-in-1-9-2-p180), replacing Symbol#to_proc with a pure-ruby equivalent solves the issue just fine:

```ruby
class Symbol
  def to_proc
    lambda { |x| x.send(self) }
  end
end
```
The above has no memory leaks with a.map(&:foo). Also, as Niklas said, calling a.map(&:foo.to_proc) explicitly doesn't involve a leak either. The issue seems to me to be with ruby's sym_proc_cache global in string.c... when that code path is avoided, nothing seems to leak.

What I would expect is for a.map(&:foo) and a.map{|x| x.foo} to work identically, but the (&:foo) form seems to leak memory.

This issue is important to me because we had a very high-memory using codebase on our production servers and the items in my array are each a few hundred megs in size, and such memory leaks ran our servers out of memory fairly quickly. (The explicit block way of using map works fine for now, but I want to make sure others don't hit this issue.)

History

#1 - 09/02/2011 04:53 AM - normalperson (Eric Wong)
Ken Simon ninkendo@gmail.com wrote:

It appears that running an array through .map(&:foo) leaks the array's contents, and they don't get picked up by the Garbage Collector.

Given a simple class:

```ruby
class C
  def foo
    "foo"
  end
end
```

The following appears to leave references around (1.9.3-preview1 irb session shown, ruby -v gives ruby -v ruby 1.9.3dev (2011-07-31 revision 32789) [x86_64-darwin11.1.0]):

```ruby
ruby-1.9.3-preview1 :001 > a = 10.times.map{C.new}
=> [... snip ...]
ruby-1.9.3-preview1 :002 > b = a.map(&:foo)
ruby-1.9.3-preview1 :003 > a = b = nil
=> nil
ruby-1.9.3-preview1 :004 > GC.start
=> nil
ruby-1.9.3-preview1 :005 > ObjectSpace.each_object(C){}
=> 10
```

GC never guarantees objects will be freed at any determined point in time.

The only way to prove a leak in the GC is to have an infinite loop and watch for unbounded memory growth (I watch the process in "top"):

```ruby
loop do
  a = 10.times.map{C.new}
  b = a.map(&:foo)
end
```

#2 - 09/03/2011 03:06 AM - ninkendo (Ken Simon)

Indeed you're right, the references certainly take longer to get collected but don't increase considerably over time. It certainly caused some issues with our production code, but we need to accept a high degree of memory usage with the problem we're solving. This ticket can be closed, thank you for your time.

#3 - 09/03/2011 03:11 AM - luislavena (Luis Lavena)
- Status changed from Open to Rejected

Closed on OP request. No memory leak found.