Ruby - Feature #8961

Synchronizable module to easily wrap methods in a mutex

09/27/2013 08:23 PM - tobiassvn (Tobias Svensson)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

=begin
I propose a Synchronizable mixin to easily wrap methods in a mutex which works together with Ruby 2.1's method name symbols returned from '(({def}))'.

The Mixin adds a new '(({synchronized}))' class method which would alias the referenced method and redefines the original method wrapped in a '(({synchronize do .. end}))' block.

This is probably somewhat related and an alternative to #8556.

Proof of concept (I've used Monitor here so potential users won't have to worry about reentrancy):

require 'monitor'

module Synchronizable
module ClassMethods
def synchronized(method)
aliased = :"#{method}_without_synchronization"
alias_method aliased, method

```
    define_method method do |*args, &block|
      monitor.synchronize do
        __send__(aliased, *args, &block)
      end
    end
  end
end

def monitor
  @monitor ||= Monitor.new
end

def self.included(base)
  base.extend(ClassMethods)
end
```

end

class Foo
include Synchronizable

```
synchronized def bar
  # ...
end
```

end
=end

Related issues:

Related to Ruby - Feature #8556: MutexedDelegator as a trivial way to make an...	**Rejected**

History

#1 - 09/27/2013 08:40 PM - headius (Charles Nutter)

I would like to see this in 2.1, as a standard Module method. The fact that "def" returns the method name now makes this really easy.

I think this would need to be implemented natively to work, however. The prototype above has a key flaw: there's no guarantee that only one Monitor will be created, so two threads could execute the same method at the same time, synchronizing against different monitors. Putting the synchronized wrapper into C code would prevent a potential context switch when first creating the Monitor instance (or it could simply use some other mechanism, such as normal Object monitor synchronization in JRuby).

This feature is similar to an extension in JRuby called JRuby::Synchronized that causes all method lookups to return synchronized equivalents.

Combined with https://bugs.ruby-lang.org/issues/8556 this could go a very long way toward giving Ruby users better tools to write thread-safe code.

#2 - 09/27/2013 09:45 PM - tobiassvn (Tobias Svensson)

Having this as a method on Module directly would of course be ideal. However, I believe the mutex/monitor used should still be exposed as a private method so it can be used without the 'synchronized' method.

#3 - 09/28/2013 03:45 AM - headius (Charles Nutter)

tobiassvn (Tobias Svensson) wrote:

> Having this as a method on Module directly would of course be ideal. However, I believe the mutex/monitor used should still be exposed as a private method so it can be used without the 'synchronized' method.

Maybe. I don't like the idea of exposing this mutex/monitor, since it could be modified or locked and never released. I would be more in favor of a "tap" form that synchronizes against the same internal monitor, similar to Java's "synchronized" keyword.

obj.synchronized { thread-sensitive code here }

That would also open up the possibility of using a lighter-weight internal mutex/monitor rather than the rather heavy-weight Ruby-land version.

#4 - 09/28/2013 09:23 AM - nobu (Nobuyoshi Nakada)

headius (Charles Nutter) wrote:

> Maybe. I don't like the idea of exposing this mutex/monitor, since it could be modified or locked and never released. I would be more in favor of a "tap" form that synchronizes against the same internal monitor, similar to Java's "synchronized" keyword.
>
> obj.synchronized { thread-sensitive code here }

Use MonitorMixin.

#5 - 10/01/2013 02:14 PM - nobu (Nobuyoshi Nakada)

- Description updated

#6 - 10/02/2013 02:13 AM - headius (Charles Nutter)

nobu (Nobuyoshi Nakada) wrote:

> headius (Charles Nutter) wrote:
>
> > Maybe. I don't like the idea of exposing this mutex/monitor, since it could be modified or locked and never released. I would be more in favor of a "tap" form that synchronizes against the same internal monitor, similar to Java's "synchronized" keyword.
> >
> > obj.synchronized { thread-sensitive code here }
>
> Use MonitorMixin.

Yeah, that's not a bad option from a pure-Ruby perspective. We could add "synchronized" to classes that include MonitorMixin, perhaps?

- added to monitor.rb:

```
module MonitorMixin
module ClassMethods
def synchronized(method)
aliased = :"#{method}_without_synchronization"
alias_method aliased, method

  define_method method do |*args, &block|
    mon_enter
    begin
```

```
      __send__(aliased, *args, &block)
    ensure
      mon_exit
    end
  end
end
```

end

def self.included(base)
base.extend(ClassMethods)
end
end

class Foo
include MonitorMixin

synchronized def bar
...
end
end

...

My suggestion to have it be native on Module opened up the possibility of implementing it in a faster, native way. MonitorMixin has a very large perf hit on all impls right now, but especially MRI. See my benchmarks in https://github.com/ruby/ruby/pull/405#issuecomment-25417666

#7 - 10/02/2013 10:15 PM - tobiassvn (Tobias Svensson)

I suppose if this is being added to MonitorMixin it should probably be in Mutex_m as well?

#8 - 10/03/2013 06:52 AM - headius (Charles Nutter)

tobiassvn (Tobias Svensson) wrote:

> I suppose if this is being added to MonitorMixin it should probably be in Mutex_m as well?

I don't think so, since a Mutex is not reentrant and what we want is monitor semantics for #synchronized.

#9 - 12/23/2021 11:43 PM - hsbt (Hiroshi SHIBATA)

- Project changed from 14 to Ruby