Fiber

RubySG - April 2020

Who am I?

Software Developer working at Shopify Singapore.

Twitter ↠ @taykangsheng

GitHub ↠ @taykangsheng

What is Fiber?

Fiber is a control flow **primitive** introduced in Ruby-1.9. [6]

They are a means of creating code blocks that can be paused and resumed. [5]

They provide lightweight concurrency (Coroutines) in Ruby. [4]

The documentation for Fiber can be found at
https://ruby-doc.org/core-2.7.1/Fiber.html

Basic usage of Fiber

3 main methods.

1. .new
2. .yield
3. #resume

```
a = Fiber.new do
  puts "checkpoint 1"
  Fiber.yield
  puts "checkpoint 2"
end

>> a.resume
"checkpoint 1"
=> nil
>> a.resume
"checkpoint 2"
=> nil
```

Basic usage of Fiber

.new

The standard operator to create new Ruby objects!

To create a Fiber, pass in a block while calling **.new**.

```
a = Fiber.new do
  puts "checkpoint 1"
  Fiber.yield
  puts "checkpoint 2"
end

>> a.resume
"checkpoint 1"
=> nil
>> a.resume
"checkpoint 2"
=> nil
```

Basic usage of Fiber

.yield(args, ...)

.yield pauses the code execution. It is like the “pause button”.

When the code execution hits **.yield**, it will pause the code execution at that point. Code execution will start from this point again the next time.

```
a = Fiber.new do
  puts "checkpoint 1"
  Fiber.yield
  puts "checkpoint 2"
end

>> a.resume
"checkpoint 1"
=> nil
>> a.resume
"checkpoint 2"
=> nil
```

Basic usage of Fiber

#**resume(args, ...)**

Call #**resume** to execute the code inside the given block.

If #**resume** is not called before, then it will just start executing the code.

If #**resume** is called before, then it will continue where it left off.

```
a = Fiber.new do
  puts "checkpoint 1"
  Fiber.yield
  puts "checkpoint 2"
end

>> a.resume
"checkpoint 1"
=> nil
>> a.resume
"checkpoint 2"
=> nil
```

Basic usage of Fiber

```
a = Fiber.new do
  puts "checkpoint 1"
  Fiber.yield
  puts "checkpoint 2"
end

>> a.resume
"checkpoint 1"
=> nil
>> a.resume
"checkpoint 2"
=> nil
```

What is the difference between Threads and Fiber?

1. Threads run in the “background” while Fiber runs as the main program when it is executing. [2]
2. Fibers are never preempted and that the scheduling must be done by the programmer and not the VM. [5]

Libraries that uses Fiber?

#1 use of fibers in Ruby is to implement Enumerators [2].

Notable RubyGems that uses Fiber:

1. Falcon, https://github.com/socketry/falcon
2. Async, https://github.com/socketry/async

Implementing infinite Fibonacci sequence with Fiber

```
fib = Fiber.new do
  x, y = 0, 1

  loop do
    Fiber.yield y
    x, y = y, x + y
  end
end
```

```
>> fib.resume
=> 1
>> fib.resume
=> 1
>> fib.resume
=> 2
>> fib.resume
=> 3
>> fib.resume
=> 5
>> fib.resume
=> 8
>> fib.resume
=> 13
....
```

Conclusion

Fiber implicitly preserve state without having the programmer to worry about it. [3]

Fiber force the programmer to do explicit scheduling which can certainly add to the complexity of the program, but offer us the full flexibility of determining how our CPU resources are used and also help us avoid the need for locks in mutexes in our code! [3]

References

[1] https://stackoverflow.com/a/9194052

[2] https://www.rubyguides.com/2019/11/what-are-fibers-in-ruby/

[3] https://www.igvita.com/2009/05/13/fibers-cooperative-scheduling-in-ruby/

[4] https://www.infoq.com/news/2007/08/ruby-1-9-fibers/

[5] https://ruby-doc.org/core-2.7.1/Fiber.html

[6] http://www.rubyinside.com/ruby-fibers-8-useful-reads-on-rubys-new-concurrency-feature-1769.html

That’s all folks