autoload の話

田中哲
2019-04-20
RubyKaigi AfterParty

autoload ってご存知ですか？

- autoload :Foo, "foo"
- 定数 Foo を参照したときに
  自動的に require "foo" してくれる機能

プログラムを高速化する最高の方法

- 不要な処理をしないのがいちばん速い
- 不要かどうかの判断コストが無視できる場合

autoload

- 本当に必要になるまでロードしない
- プログラムを高速化する最高の方法のひとつ
- autoload サイコー

autoload は危険？

- matz は multi thread での autoload は根本的に危険と主張している
- そのうち消したいと昔から言っている
- autoload サイテー
- が、Ruby 3.0 で消すのは断念した模様

Rails

- Rails は autoload を使わず、const_missing を使っている
- matz が autoload を消すと言うから
- でも今度のバージョンでは autoload を使うらしい (Zeitwerk)

autoload は危険？

- marcandre: autoload_relative を加えたい [Feature#15330]
- matz: autoload の非決定性は嫌い。#10892, #11384, #12688 とか問題が発生してて直せてないし。互換性で消せないにしても、便利にするのは嫌
- eregon: どれも直ってる感じなんだけど？

なんか、根拠がはっきりしてない感じ？

single thread で autoload

- 定数 Foo を参照したときに require 'foo' してから Foo を参照する
- とくに問題ない
- foo.rb で定義されるブツが、必ず定数 Foo を経由してアクセスされるなら

素朴な動作

main.rb:

```
autoload :Foo, "foo" #  定数 Foo  を  autoload  用に仕掛ける
Foo #  定数参照で仕掛けが発火 :
        #  仕掛けを外して  require  し、Foo  の参照をやり直す
```

foo.rb:

```
class Foo #  定数  Foo  を定義
  def initialize(v)
  end
end
```

multi thread で autoload

- 複数のスレッドで Foo を参照する
- autoload :Foo, "foo"
  t1 = Thread.new { Foo.new(0) }
  t2 = Thread.new { Foo.new(0) }

context switch

- クラス定義の途中で thread が context switch するかもしれない

foo.rb:

```
# context switch
class Foo
  # context switch
  def initialize(v)
  end
  # context switch
end
# context switch
```

危険な context switch

- クラス定義が終わる前に context switch して、不完全な Foo が参照されると危険
- クラス定義が完了してしまえば安全

foo.rb:

```
# dangerous context switch 1
class Foo
  # dangerous context switch 2
  def initialize(v)
  end
end
```

race condition 1

- t1 が Foo を参照して require している途中、Foo の定義前 (context switch 1) で t2 が Foo を参照する
- t2 が参照したとき、Foo の autoload の仕掛けは外されているので Foo は単に存在しない状態であり、uninitialized constant になる

race condition 2

- t1 が Foo を参照して require している途中、Foo の定義後、initialize メソッドの定義前 (context switch 2) に t2 が Foo を参照する
- t2 が参照したとき、Foo は存在する (uninitialized constant にはならない)
- でも、Foo#initialize が存在しないので BasicObject#initialize が呼ばれ、Foo.new(0) は wrong number of arguments になる

どうすればいい？ ロックすればいい

- ロックがなければどうしようもない
- ロックすれば動くだろう

ロックするには

- 定数 Foo が定義されていても、メソッドは足りないかもしれない
- foo.rb 全体を読み終われば Foo は完成していると想定
- Foo の参照で autoload の仕掛けが動き始めてから、foo.rb の実行が終わるまで、他のスレッドからの Foo の参照はブロック
- Foo が定義されても、foo.rb を読み終わるまではダメ
- const_missing で実現するのが無理筋なのがわかる

ちゃんとロックしてますか？

- race condition 1 : Ruby 1.9.1 からは大丈夫ぽい
- race condition 2 : Ruby 2.0.0 からは大丈夫ぽい

autoload と Thread の導入時期

- 1995-12-21 Ruby 0.95: autoload
- 1996-12-24 Ruby 0.99.4-961224: Thread

12 年経過

- 2009-01-30 Ruby 1.9.1p0: race condition 1 解決
- 2013-02-24 Ruby 2.0.0p0: race condition 2 解決

all-ruby サイコー

なんで 12 年もかかったの？

たぶん昔はまともな lock がなかったから
処理系内部でつかえるまともな lock は YARV 以降

- 1995-12-21 Ruby 0.95 autoload
- 1996-12-24 Ruby 0.99.4-961224: Thread
- 2007-12-25 Ruby 1.9.0-0: YARV
- 2009-01-30 Ruby 1.9.1p0 race condition 1 解決
- 2013-02-24 Ruby 2.0.0p0 race condition 2 解決

matz に尋ねてみた

multi thread な autoload が根本的に危険っていうの、
昔はまともなロックがなかったからなんじゃないの？

反応

覚えてない感じ
否定も肯定もされなかった

個人的邪推

ロックがない状況で検討した結果の
根本的に危険という結論を、
検討の内容を忘れて
引きずっているだけなのではないか？

もし matz が autoload に肯定的になれば現状で Happy?

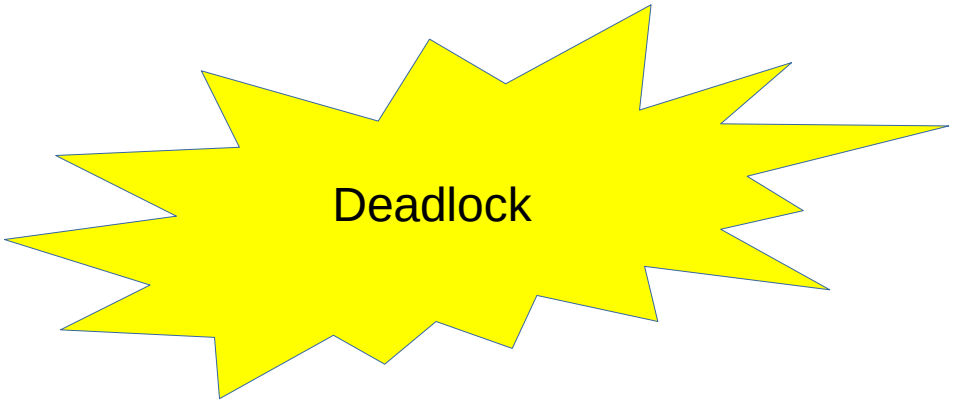
残念ながら No

相互参照による deadlock [Bug #15598]

a.rb:
```
class A
  def a1() end
  p [:a, A, B] # refer B
  def a2() end
end
```

b.rb:
```
class B
  def b1() end
  p [:b, A, B] # refer A
  def b2() end
end
```

base_ab.rb:
```
autoload :A, "./a.rb"
autoload :B, "./b.rb"
t1 = Thread.new { A }
t2 = Thread.new { B }
```

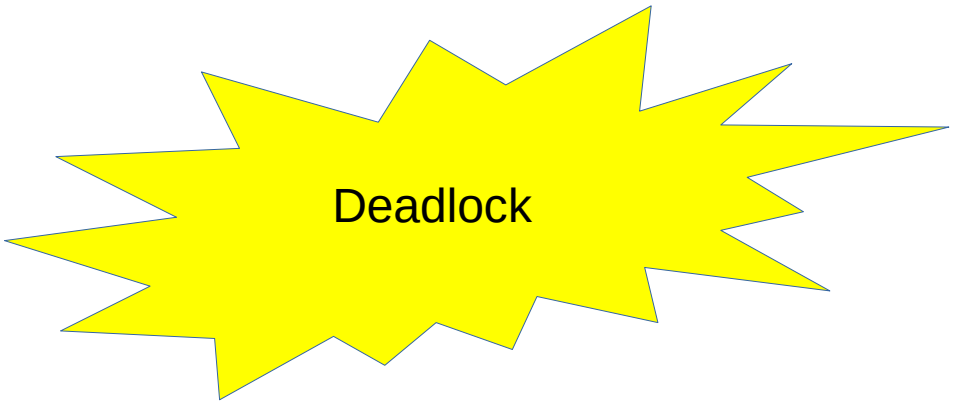


なんで相互参照が deadlock になる？

- ロックが file 単位だから (feature 単位かも?)
- ロック順
  - t1: A → B の順でロック
  - t2: B → A の順でロック
- 参照関係はファイルの中に散在して書かれている
  - ファイルを実際に読まないと参照関係がわからない
  - ファイルを読む前にロックしなければならない
- 適切なロックの順序を決められない

autoload と require の混用で deadlock [Bug #15599]

a.rb:
```
class A
  def a1() end
end
```



base_r.rb:
```
autoload :A, './a'
t1 = Thread.new { A }
t2 = Thread.new { require './a' }
```

なんで混ぜると deadlock?

- 深い理由はなくて単に考えが足りないだけでは？

修正できるか？

たぶん可能

- ロックを粗粒度にすればいい
- file 単位じゃなくて global にする
  つまり global autoload lock
- autoload 対象のライブラリを require するときも global autoload lock をロックする

- なお、すべての require を lock するのはやりすぎ
- require から返ってこないやつがありそう

もっとある？

- global autoload lock を導入したと仮定
- アプリケーションがロックを使っていると？
  - t1: global autoload lock →app lock の順でロック
  - t2: app lock → global autoload lock の順でロック
    deadlock
- まぁ、これはさすがに諦めざるをえないのでは

autoload 関係の機能拡張の可能性

- [Feature #] autoload_relative: 相対パスで autoload
- autoload されているものを即座にすべて require するメソッド
- [Feature #15777] autoload?
- Zeitwerk の発表でも述べられていた気が

まとめ

- const_missing でやれというのは無理筋
- multi thread で autoload が根本的に危険というのは再考の余地あり (控えめな表現)