演習1 Hello, World.

➡ list 1-1 hello_server.rb

```
require 'drb/drb'

class Hello
  def hello
    puts('Hello, World.')
  end
end

DRb.start_service('druby://localhost:54000', Hello.new)
while true
  sleep 1
end
```

➡ list 1-2 hello_client.rb

```
require 'drb/drb'

DRb.start_service
ro = DRbObject.new_with_uri('druby://localhost:54000')
ro.hello
```

実験

はじめに端末1でhello_server.rbを起動する。

➡ terminal 1

```
$ ruby hello_server.rb
```

次に端末2でhello_client.rbを起動する

➡ terminal 2

```
$ ruby hello_client.rb
```

すると端末1に「Hello, World.」が印字される。

Q. hello_server.rbを[Ctrl]+Cなどで停止させてhello_client.rbを起動するとどうなりますか？

演習2 Hash

➡ list 2-1 hash_server.rb

```
require 'drb/drb'
require 'pp'

front = Hash.new
DRb.start_service('druby://localhost:54300', front)
while true
  sleep 10
  pp front
end
```

実験

端末1でhash_server.rbを起動。続いて複数の端末からirbを使ってサーバ上にあるHashを操作します。

➡ terminal 1

```
$ ruby hash_server.rb
```

➡ terminal 2

```
$ irb
irb(main):001:0> require 'drb/drb'
=> true
irb(main):002:0> DRb.start_service
=> #<DRb::DRbServer:0...
irb(main):003:0> ro = DRbObject.new_with_uri('druby://localhost:54300')
=> #<DRb::DRbObject...
irb(main):004:0> ro[1] = 'Hello, World.'
=> "Hello, World."
irb(main):005:0> ro[1]
=> "Hello, World."
irb(main):006:0> ro[1] = nil
=> nil
irb(main):007:0> ro[1]
=> nil
```

➡ terminal 3

```
$ irb
irb(main):001:0> require 'drb/drb'
=> true
irb(main):002:0> DRb.start_service
=> #<DRb::DRbServer:0...
irb(main):003:0> ro = DRbObject.new_with_uri('druby://localhost:54300')
=> #<DRb::DRbObject...
irb(main):004:0> ro[1]
=> "Hello, World."
irb(main):006:0> ro[2] = 'Hello, Again.'
=> "Hello, Again."
```

Q. irbを使っていろんな種類のオブジェクト（String, Integer, Hash, IO）を入れてみてください。

Q. 別のプロセスからも同じものが取れますか？

演習3 Queue

➡ list 3-1 queue_server.rb

```
require 'drb/drb'
require 'thread'

DRb.start_service('druby://localhost:54320', Queue.new)
while true
  sleep 1
end
```

➡ list 3-2 dequeue.rb

```
require 'drb/drb'

DRb.start_service
queue = DRbObject.new_with_uri('druby://localhost:54320')
while true
  p queue.pop
  sleep(rand)
end
```

実験

まず、端末1でqueue_server.rbを起動、端末2でdequeue.rbを起動します。

次に、irbで好きなオブジェクトをいくつもQueueにpushし、端末2の出力を観察してください。

➡ terminal 1

```
$ ruby queue_server.rb
```

➡ terminal 2

```
$ ruby dequeue.rb
```

➡ terminal 3

```
$ irb
irb(main):001:0> require 'drb/drb'
=> true
irb(main):002:0> DRb.start_service
=> #<DRb::DRbServer:0...
irb(main):003:0> ro = DRbObject.new_with_uri('druby://localhost:54320')
=> #<DRb::DRbObject...
irb(main):004:0> ro.push('Hello, World.')
=> ....
```

Q. dequeue.rbを複数起動して動かしてみましょう。

Q. irbの数も増やして試してみましょう。

演習4 Dripの準備

インストールしてある？

実験の準備（非UNIX系）

WindowsなどPOSIXでないひと向けの準備です。MyDripはUNIXドメインソケットとホームディレクトリ、forkを用いるため、これらがない環境では簡単なサーバが必要です。

➡ list 4-1 drip_s.rb

```
require 'drip'
require 'drb'

class Drip
  def quit
    Thread.new do
      synchronize do |key|
        exit(0)
      end
    end
  end
end

drip = Drip.new('drip_dir')
DRb.start_service('druby://localhost:54321', drip)
DRb.thread.join
```

➡ terminal 1

```
$ ruby drip_s.rb
```

演習中、my_drip.rbをrequireする代わりに次のファイルをrequireしてください。

➡ list 4-2 drip_d.rb

```
require 'drb/drb'
MyDrip = DRbObject.new_with_uri('druby://localhost:54321')
```

実験の準備（POSIX系）

OSX、LinuxなどPOSIX系の場合、MyDrip.invokeでDripデーモンを起動します。

➡ terminal 1

```
$ irb
irb(main):001:0> require 'my_drip'
irb(main):002:0> MyDrip.invoke
irb(main):003:0> exit
```

演習4-1 Dripを使った同期

実験

DripをQueueとしてつかってみます。端末2から要素を書き込み、端末3で読みます。

➡ terminal 2

```
$ irb -r my_drip --prompt simple
>> MyDrip.write('Hello')
=> 1312541947966187
>> MyDrip.write('world')
=> 1312541977245158
```

➡ terminal 3

```
$ irb -r my_drip --prompt simple
>> MyDrip.read(0, 1)
=> [[1312541947966187, "Hello"]]
```

続けて読むには次のようにします。イディオムですよ！そして三度目のreadでブロックします。

➡ terminal 3 (continue)

```
>> k = 0
=> 0
>> k, v = MyDrip.read(k, 1)[0]
=> [1312541947966187, "Hello"]
>> k, v = MyDrip.read(k, 1)[0]
=> [1312541977245158, "World"]
>> k, v = MyDrip.read(k, 1)[0]
```

端末2から新しい要素を書き込んで端末3のreadが動き出す様子をみてください。

➡ terminal 2 (continue)

```
>> MyDrip.write('Hello, Again')
=> 1312542657718320
>> k, v = MyDrip.read(k, 1)[0]
=> [1312542657718320, "Hello, Again"]
```

端末4からreadします。読み手を増やすとどうなるでしょうか。

➡ terminal 4

```
$ irb -r my_drip --prompt simple
>> k = 0
=> 0
>> k, v = MyDrip.read(k, 1)[0]
=> [1312541947966187, "Hello"]
>> k, v = MyDrip.read(k, 1)[0]
=> [1312541977245158, "World"]
>> k, v = MyDrip.read(k, 1)[0]
=> [1312542657718320, "Hello, Again"]
```

実験

Dripのサーバを再起動します。

非POSIXでは端末1のプロセス(ruby drip_s.rb)をCtrl-Cなどで終了させて、また起動してください。

POSIXマシンでは次のようにMyDripをquit、invokeさせます。

➡ terminal 2 (continue)

```
>> MyDrip.quit
=> #<Thread:...>
>> MyDrip.invoke
=> 61470
```

三つの要素をreadして内容が復元されていることを確認しましょう。

➡ terminal 2 (continue)

```
>> MyDrip.read(0, 3)
=> [[1312541947966187, "Hello"], [1312541977245158, "World"],
[1312542657718320, "Hello, Again"]]
```

演習4-2 Dripを使ったKVS

実験

Dripを履歴つきHashとしてつかってみます。タグを使ってデータをset/getします。

➡ terminal 2 (continue)

```
>> MyDrip.write(29, 'seki.age')
=> 1313358208178481
>> MyDrip.head(1, 'seki.age')
=> [[1313358208178481, 29, "seki.age"]]
>> k, v = MyDrip.head(1, 'seki.age')[0]
=> [[1313358208178481, 29, "seki.age"]]
>> v
=> 29
```

再設定します。

➡ terminal 2 (continue)

```
>> MyDrip.write(49, 'seki.age')
=> 1313358584380683
>> MyDrip.head(1, 'seki.age')
=> [[1313358584380683, 49, "seki.age"]]
>> MyDrip.head(10, 'seki.age')
=> [[1313358208178481, 29, "seki.age"], [1313358584380683, 49, "seki.age"]]
```

データがなければ、空の配列になります。read_tagで待合せもできます。

➡ terminal 2 (continue)

```
>> MyDrip.head(1, 'sora_h.age')
=> []
>> MyDrip.read_tag(0, 'sora_h.age')
```

➡ terminal 3 (continue)

```
>> MyDrip.write(12, 'sora_h.age')
=> 1313359385886937
```

➡ terminal 2 (continue)

```
=> [[1313359385886937, 12, "sora_h.age"]]
```

演習4-3 タグを使ったブラウズ

実験

タグを使って時系列のデータをブラウズします。

まず実験用のデータを用意します。

➡ terminal 2 (continue)

```
>> MyDrip.write('sentinel', 'test1')
=> 1313573767321912
>> MyDrip.write(:orange, 'test1=orange')
=> 1313573806023712
>> MyDrip.write(:orange, 'test1=orange')
=> 1313573808504784
>> MyDrip.write(:blue, 'test1=blue')
=> 1313573823137557
>> MyDrip.write(:green, 'test1=green')
=> 1313573835145049
>> MyDrip.write(:orange, 'test1=orange')
=> 1313573840760815
>> MyDrip.write(:orange, 'test1=orange')
=> 1313573842988144
>> MyDrip.write(:green, 'test1=green')
=> 1313573844392779
```

まず最初の（一番古い）要素のキーを求めます。

➡ terminal 2 (continue)

```
>> k, = MyDrip.head(1, 'test1')[0]
=> [1313573767321912, "sentinel", "test1"]
>> k
=> 1313573767321912
```

それより後の四つの要素を読みます。

➡ terminal 2 (continue)

```
>> ary = MyDrip.read(k, 4)
=> [[1313573806023712, :orange, "test1=orange"],
[1313573808504784, :orange, "test1=orange"],
[1313573823137557, :blue, "test1=blue"],
[1313573835145049, :green, "test1=green"]]
```

さらに、その四つより後の要素を読みます。キーを更新しながら読むのは前回のイディオムです。三つしかないので三つだけ返ります。さらに読み進めるとブロックします。他の端末から要素を増やしてreadが動き出すことを確認してください。

➡ terminal 2 (continue)

```
>> k = ary[-1][0]
=> 1313573835145049
>> ary = MyDrip.read(k, 4)
=> [[1313573840760815, :orange, "test1=orange"],
[1313573842988144, :orange, "test1=orange"],
```

```
[1313573844392779, :green, "test1=green"]]
>> k = ary[-1][0]
=> 1313573844392779
>> ary = MyDrip.read(k, 4)
```

➡ terminal 3 (continue)

```
>> MyDrip.write('hello')
=> 1313574622814421
```

カーソルを先頭に戻し、read_tagで"test1=orange"というタグを持つ要素を四つ（最小二つ）読みます。繰り返し読むとブロックします。他の端末から二つの要素を追加すると、read_tagが再開します。

➡ terminal 2 (continue)

```
>> k, = MyDrip.head(1, 'test1')[0]
=> [1313573767321912, "sentinel", "test1"]
>> ary = MyDrip.read_tag(k, 'test1=orange', 4, 2)
=> [[1313573806023712, :orange, "test1=orange"],
[1313573808504784, :orange, "test1=orange"],
[1313573840760815, :orange, "test1=orange"],
[1313573842988144, :orange, "test1=orange"]]
>> k = ary[-1][0]
=> 1313573842988144
>> ary = MyDrip.read_tag(k, 'test1=orange', 4, 2)
```

➡ terminal 3 (continue)

```
>> MyDrip.write('more orange', 'test1=orange')
=> 1313575076451864
>> MyDrip.write('more orange', 'test1=orange')
=> 1313575077963911
```