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)
sleep
```

➡ 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."
```

➡ 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)
sleep
```

➡ 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(3)
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。

課題4 Rindaを使った同期

➡ list 4-1 notify.rb

```
require 'rinda/tuplespace'
require 'drb/drb'

class Notifier
  def initialize
    @ts = Rinda::TupleSpace.new
    @ts.write([:tail, 0])
  end

  def notify(it)
    tuple = @ts.take([:tail, nil])
    tail = tuple[1] + 1
    @ts.write([tail, it])
    @ts.write([:tail, tail])
    tail
  end

  def receive(key)
    @ts.read([key, nil])
  end
end

DRb.start_service('druby://localhost:54321', Notifier.new)
sleep
```

➡ list 4-2 receive.rb

```
require 'drb/drb'

DRb.start_service
ro = DRbObject.new_with_uri('druby://localhost:54321')

key = 0
while true
  key, obj = ro.receive(key + 1)
  p [key, obj]
  sleep(3)
end
```

実験

演習3のQueueと似ている実験です。

まず、端末1でnotify.rbを起動。

次に、端末2でirbを用いてNotifierに任意のタイミングでオブジェクトを通知します。

➡ terminal 1

```
$ ruby notify.rb
(終了しない)
```

➡ terminal 2

```
$ irb
```

```
irb(main):001:0> require 'drb/drb'
=> true
irb(main):002:0> DRb.start_service
=> #<DRb::DRbServer:...>
irb(main):003:0> ro = DRbObject.new_with_uri('druby://localhost:54321')
=> #<DRb::DRbObject:0...>
irb(main):004:0> ro.notify('tweet')
=> 1
irb(main):005:0> ro.notify('Hello, World')
=> 2
```

端末3でreceive.rbを起動すると、それまでに通知されたオブジェクトが順に表示され、尽きると休眠します。

➡ terminal 3

```
$ ruby receive.rb
```

その後、端末2から新しいオブジェクトを通知すると、端末3の休眠が解け、通知されたオブジェクトが表示されます。

さらに別の端末（端末4）でreceive.rbを起動すると、それまでに通知されたオブジェクトが順に表示されます。

Q. 古いデータを削除する機能を入れるにはどのような仕組みが要るでしょう。