dRuby and Rinda.

その実装と応用を(再び)関西で

念のため

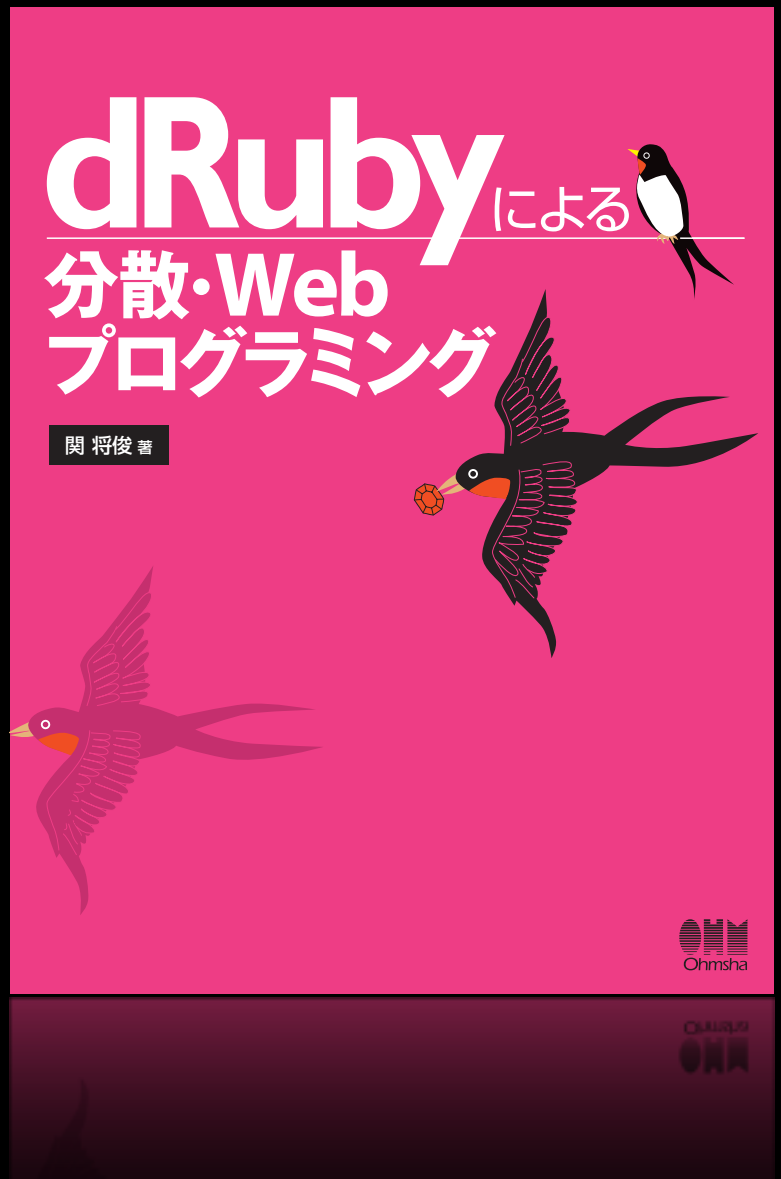
- 先月の札幌Ruby会議01を見ちゃった人いますか?

重要なことを先に

- 先に

￥3360

- 初刷まだ買えます



Who is translating it?

- 英語版はまだ買えません

http://dx.doi.org/10.1007/s10766-008-0086-1

$32.00

- International Journal of PARALLEL PROGRAMING
- コレクターズアイテム
- 余剰資金のある研究室は Must buy!

Int J Parallel Prog
DOI 10.1007/s10766-008-0086-1

dRuby and Rinda: Implementation and Application of Distributed Ruby and its Parallel Coordination Mechanism

Masatoshi Seki

Received: 20 March 2008 / Accepted: 13 August 2008

Abstract The object-oriented scripting language Ruby is admired by many programmers for being easy to write in, and for its flexible, dynamic nature. In the last few years, the Ruby on Rails web application framework, popular for its productivity benefits, has brought about a renewed attention to Ruby for enterprise use. As the focus of Ruby has broadened from small tools and scripts, to large applications, the demands on Ruby's distributed object environment have also increased, as has the need for information about its usage, performance and examples of common practices. dRuby and Rinda were developed by the author as the distributed object environment and shared tuplespace implementation for the Ruby language, and are included as part of Ruby's standard library. dRuby extends method calls across the network while retaining the benefits of Ruby. Rinda builds on dRuby to bring the functionality of Linda, the glue language for distributed co-ordination systems, to Ruby. This article discusses the design policy and implementation points of these two systems, and demonstrates their simplicity with sample code and examples of their usage in actual applications. In addition to dRuby and Rinda's appropriateness for prototyping distributed systems, this article will also demonstrate that dRuby and Rinda are building a reputation for being suitable infrastructure components for real-world applications.

Keywords Ruby · Linda · dRuby · Rinda · TupleSpace · RMI

M. Seki (✉)
Tochigi, Japan
e-mail: m_seki@mva.biglobe.ne.jp
URL: www.druby.org

Springer

http://dx.doi.org/10.1007/s10766-008-0086-1

私について

- toRubyからきました
  - 池澤さんのLTで一躍有名に

会場


南郷屋
五軒町
TSUTAYA
東北本線
東北新幹線
日産
ビジネスホテル
アジサイ
317
オグレ商事
西栄町
東町
西那須野
幼稚園
西原町
西大和
太夫塚
永田町
東小
扇町
西那須野
西朝日町
下永田
市役所支所
南町
あたご町
ATACO
西那須野中

会場


A
福島
新潟県
福島県
石川県
富山
金沢
長野
栃木県
宇都宮
群馬県
前橋
水戸
茨城県
福井
長野県
埼玉県
さいたま
福井県
岐阜県
甲府
千葉
東京都
東京
山梨県
神奈川県
千葉県
横浜
京都府
滋賀県
岐阜
京都
名古屋
愛知県
静岡県
静岡
神戸
大阪
津
奈良
大阪府
三重県
和歌山
奈良県

開催日

- 毎月第一水曜

dRubyによる分散オブジェクト環境

Linux Conference 2000 fall

Perl/Ruby Conference

2000.11.29-12/1

国立京都国際会館

dRubyお披露目

- この京都のイベントがはじまり
  - 以来、進歩のないネタをずっと‥
- いつかRuby関西へ!

Hello, Again.

- ついに関西Ruby会議01
  - 実は京都でやるもんだと勘違いしてた..

Agenda

- Rubyの紹介
- dRubyの紹介
- 入門
- 応用

Rubyは奇妙だ

- Rubyっぽく書いているとOOPな自分になれたように勘違いさせてくれる超かっこいい言語。

Rubyの特徴の一部

- かっこいいクラスライブラリ
- 型のない変数
- 豪華なリフレクション
- ユーザレベルスレッド

ブは分散のブ

- D is for Distributed
- dRubyは分散なRubyである

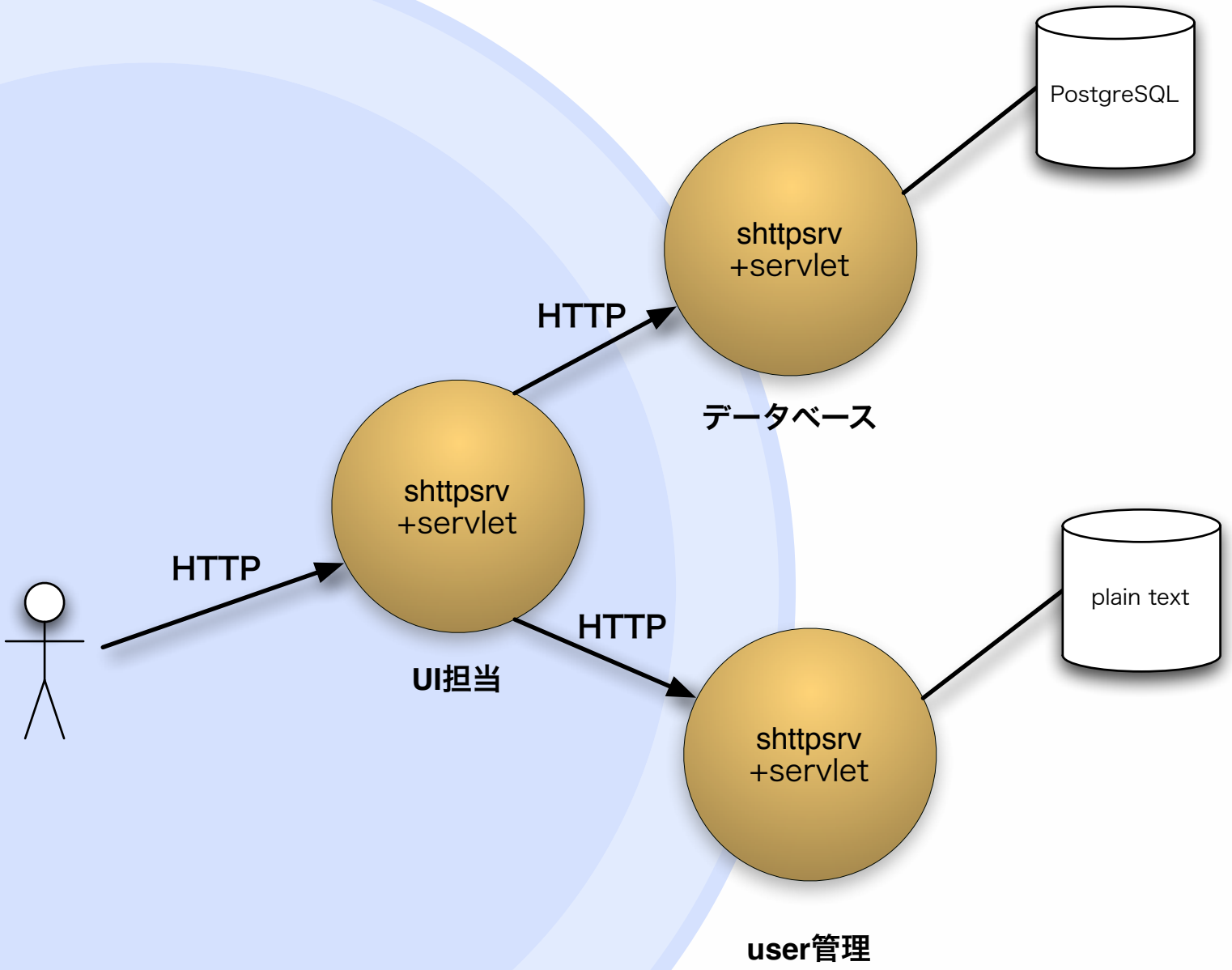
Remote Method Invocation

- 別のプロセス／マシンのオブジェクトにメッセージを送り、メソッドを起動する仕組み
- Remote Procedure Callと紙一重

例えばWeb Service

- まあ似てなくもない
  - XML-RPCとかいうぐらいだし
    - そういえば..

原型はこんなのだった


PostgreSQL
shttpsrv
+servlet
HTTP
データベース
shttpsrv
+servlet
HTTP
HTTP
plain text
UI担当
shttpsrv
+servlet
user管理

発見

- あらゆるプロセスはサーバでありクライアントである
  - まるでOOPのようだ

dRubyの特徴

- たくさんあるRMIライブラリの一つ
- でも既存のRMI, RPCのブリッジではなく
  他のシステムからの移植版でもない

dRubyの特徴

- dRubyはRubyのRMIである
  - 数多あるRMIのRuby版ではない

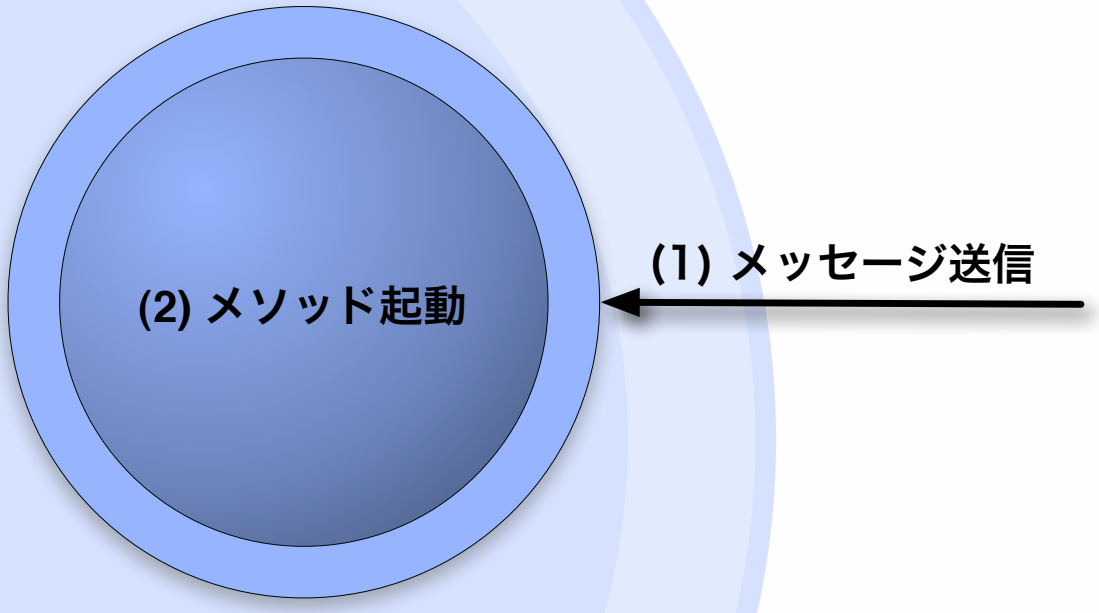
dRubyの特徴

- Rubyのメソッド呼び出しを拡張
  - インタプリタを越えて
    - プロセスを
    - マシンを
- そこだけを守備範囲にしてる

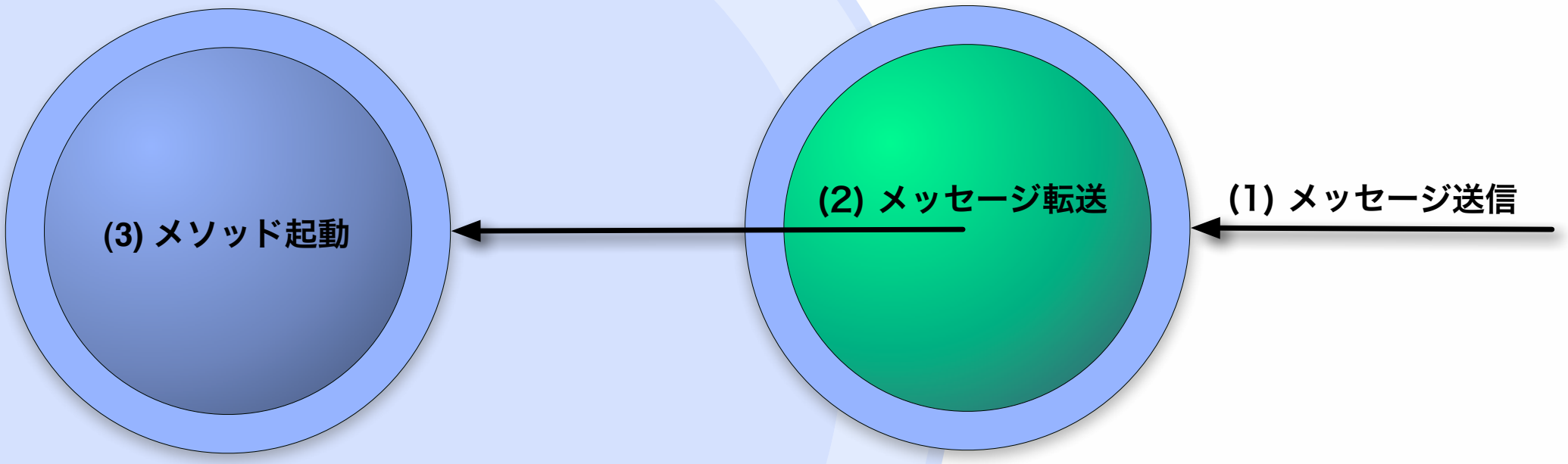
メッセージング

- オブジェクトにメッセージを送信
- メソッドを起動
- ちょっと嘘かも

Ruby



dRuby



Ruby風

- メソッド呼出しに特化したおかげで徹底したRuby風な振る舞いができたよ
  - インターフェイスの定義が不要
  - イテレータ
  - スレッド同期メカニズム

オブジェクトの交換

- それでもRuby風になりきれない部分
- 交換するのはコピーか参照か
  - 引数、戻り値、例外
- Rubyでは参照のやりとりしかないよ

オブジェクトの交換

```
def hoge(a1, a2, a3, &block)
  ...
  raise FooError unless some_cond
  ...
  return value
end
```

受け取ったものは何か

- 意識したくない
  - Rubyだし
- リモートのオブジェクトならdRubyがRMIしてくれる
- 複製注意
  - でもOOPなら複製しないよな‥

送るものは何か

- 意識したくない
  - Rubyだし
- すべてを参照にすべきか?
  - Rubyだからね
- でもいつか値が必要になる
- 値を送るか、参照を送るか

値を送るべきか、それとも参照か

- dRubyの戦略
- オブジェクトの交換方法を勝手に選択
  - Marshal可能かどうかが条件

こんなの

Marshal	
dumpable Number, String, Boolean...	pass by value
can't dump Proc, Thread, File ...	pass by reference

本当にこの選択でよいのか

- わかんないけどわりと実用的
- 気に入らなければ明示的に参照を選べる
- 手間だけど値渡しを選ぶこともできる
  - カスタマイズしたdumpが必要

残念!

- セキュリティ
  - Rubyの可視性を模倣している程度
  - 悪意は防げない
- 分散GC

実験

- memcachedが流行ってるらしい
- Hashを一つ公開する実験
- 二つの端末でirbを動かしてみるよ

はい準備です

- 今すぐTerminal.appを開いて!
  - 早く早く!!

irbを二つ用意してね

```
% irb --simple-prompt --noreadlin
```

OSXの
readlineでスレッドがス
イッチしない人向け

```
% irb --simple-prompt --noreadline
```

DRb.start_service

参加者はまず
DRb.start_service

```
>> require ‘drb/drb’
>> DRb.start_service
=> ....
```

front

frontは入り口になる
オブジェクト

URIと結びつける

```
>> require ‘drb/drb’
>> front = {}
>> DRb.start_service('druby://localhost:12345', front)
```

DRbObject

URIを与えて明示的に
参照を作る

```
>> ro = DRbObject.new_with_uri('druby://localhost:12345')
=> #<DRb::DRbObject:0x.....>
>> ro.keys
=> []
>> ro.class
=> DRb::DRbObject
>> ENV.each {|k, v| ro[k] = v}
=> {....}
>> ro.keys
=> ["MANPATH", "CVS_RSH", "USER", ....]
```

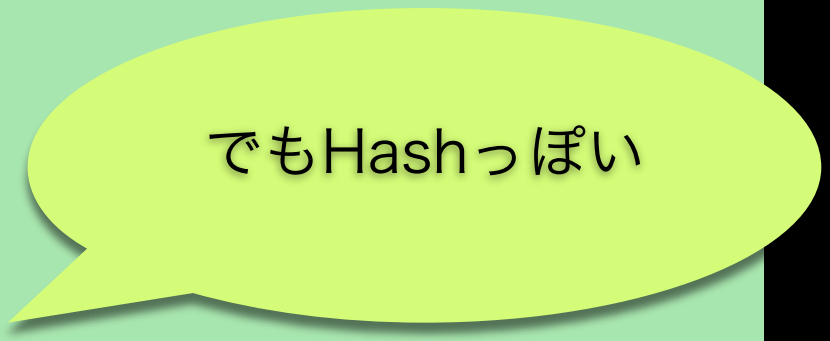
DRbObject

```
>> ro = DRbObject.new_wi                                  12345')
=> #<DRb::DRbObject:0x.
>> ro.keys
=> []
>> ro.class
=> DRb::DRbObject
>> ENV.each {|k, v| ro[k] = v}
=> {....}
>> ro.keys
=> ["MANPATH", "CVS_RSH", "USER", ....]
```

classはDRbObject

DRbObject

```
>> ro = DRbObject.new_with_uri('druby://localhost:12345')
=> #<DRb::DRbObject:0x.....>
>> ro.keys
=> []
>> ro.class
=> DRb::DRbObject
>> ENV.each {|k, v| ro[k] = v}
=> {....}
>> ro.keys
=> ["MANPATH", "CVS_RSH", "USER", ....]
```



ほんとう？

こっちのHashに入ってる!

```
>> front.keys
=> ["MANPATH", "CVS_RSH", "USER", ...]
```

暗黙的な参照

IOを入れてみるよ

```
>> front['stdout'] = $stdout
=> #<IO:0x130d80>
```

```
>> ro['stdout']
=> #<DRb::DRbObject:... @uri="druby://localhost:12345">
>> ro['stdout'].puts('Hello, World.')
```

```
>> Hello, World.
```

暗黙的な参照

```
>> front['stdout'] = $stdout
=> #<IO:0x130d80>
```

リモートからは
DRbObjectに見える

```
>> ro['stdout']
=> #<DRb::DRbObject:... @uri="druby://localhost:12345">
>> ro['stdout'].puts('Hello, World.')
```

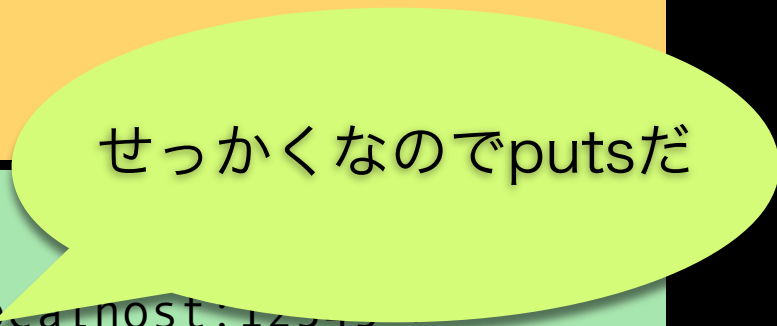
```
>> Hello, World.
```

暗黙的な参照

```
>> front['stdout'] = $stdout
=> #<IO:0x130d80>
```

```
>> ro['stdout']
=> #<DRb::DRbObject:... @uri="druby://localhost:12345
>> ro['stdout'].puts('Hello, World.')
```

```
>> Hello, World.
```



暗黙的な参照

```
>> front['stdout'] = $stdout
=> #<IO:0x130d80>
```

```
>> ro['stdout']
=> #<DRb::DRbObject                                    lhost:12345">
>> ro['stdout'].pu
```

拍手ね。

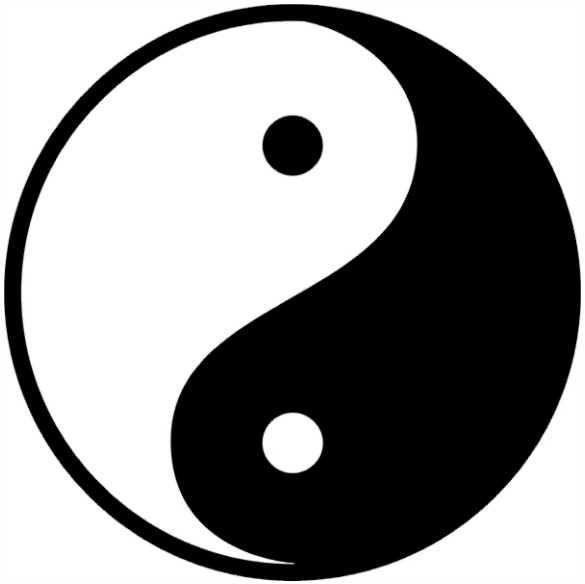
```
>> Hello, World.
```

混沌

- オブジェクトがプロセスの境界を越え混じり合う
- でもやっぱり別プロセス

混沌

- かなり強引だけどこんな感じ



次はスクリプトで

- 典型的な生産者消費者問題をSizedQueueで解く
- スレッド同期メカニズムもそのまま動く
  - という例なので眺めてて

SizedQueue server

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

queue = SizedQueue.new(10)
DRb.start_service('druby://localhost:12345', queue)
sleep
```

Producer

```
require 'drb/drb'

DRb.start_service
queue = DRbObject.new_with_uri('druby://localhost:12345')

100.times do |n|
  sleep(rand)
  queue.push(n)
end
```

Consumer

```
require 'drb/drb'

DRb.start_service
queue = DRbObject.new_with_uri('druby://localhost:12345')

100.times do
  sleep(rand)
  puts queue.pop
end
```

地味だ

- あんまりおもしろくないですね
- dRubyのスクリプトはあんなもん

実装

- 初版を読もう
  - ruby-list:15406
  - 200lines
  - いろいろバグってるが細かいことは気にしない

DRbObject

URIとobject_idで
できてる

```
class DRbObject
 def initialize(obj, uri=nil)
   @uri = uri || DRb.uri
   @ref = obj.id if obj
 end

 def method_missing(msg_id, *a)
   succ, result = DRbConn.new(@uri).send_message(self, msg_id, *a)
   raise result if ! succ
   result
 end

 attr :ref
end
```

DRbObject

- URIはインタプリタを特定し
- refはオブジェクトを特定する

method_missing

```
class DRbObject
 def initialize(obj, uri=nil)
   @uri = uri || DRb.uri
   @ref = obj.id if obj
 end

 def method_missing(msg_id, *a)
   succ, result = DRbConn.new(@uri).send_message(self, msg_id, *a)
   raise result if ! succ
   result
 end

 attr :ref
end
```

serverに
Marshalしたメッセージと
引数を転送する

DRbServer

```
def proc
  ns = @soc.accept
  Thread.start do
    begin
      s = ns
      begin
        ro, msg, argv = recv_request(s)
        if ro and ro.ref
          obj = ObjectSpace._id2ref(ro.ref)
        else
          obj = DRb.front
        end
        result = obj.__send__(msg.intern, *argv)
        succ = true
      rescue
        result = $!
        succ = false
      end
      send_reply(s, succ, result)
    ensure
      close s if s
    end
  end
end
```

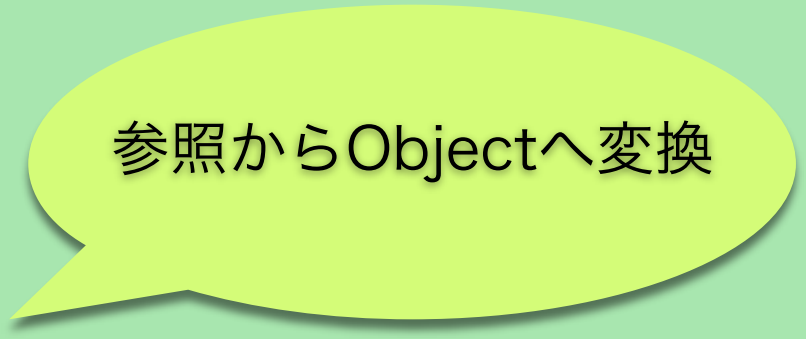
DRb server

acceptしたら処理は
別スレッドで。

```
def proc
  ns = @soc.accept
  Thread.start do
    begin
      s = ns
      begin
        ro, msg, argv = recv_request(s)
        if ro and ro.ref
          obj = ObjectSpace._id2ref(ro.ref)
        else
          obj = DRb.front
        end
        result = obj.__send__(msg.intern, *argv)
        succ = true
      rescue
        result = $!
        succ = false
      end
      send_reply(s, succ, result)
    ensure
      close s if s
    end
  end
end
```

DRbServer

```
def proc
  ns = @soc.accept
  Thread.start do
    begin
      s = ns
      begin
        ro, msg, argv = recv_request(s)
        if ro and ro.ref
          obj = ObjectSpace._id2ref(ro.ref)
        else
          obj = DRb.front
        end
        result = obj.__send__(msg.intern, *argv)
        succ = true
      rescue
        result = $!
        succ = false
      end
      send_reply(s, succ, result)
    ensure
      close s if s
    end
  end
end
```



DRbServer

```
def proc
  ns = @soc.accept
  Thread.start do
    begin
      s = ns
      begin
        ro, msg, argv = recv_request(s)
        if ro and ro.ref
          obj = ObjectSpace._id2ref(ro.ref)
        else
          obj = DRb.front
        end
        result = obj.__send__(msg.intern, *argv)
        succ = true
      rescue
        result = $!
        succ = false
      end
      send_reply(s, succ, result)
    ensure
      close s if s
    end
  end
end
```

オブジェクトのメソッドを起動する

DRbServer

```
def proc
  ns = @soc.accept
  Thread.start do
    begin
      s = ns
      begin
        ro, msg, argv = recv_request(s)
        if ro and ro.ref
          obj = ObjectSpace._id2ref(ro.ref)
        else
          obj = DRb.front
        end
        result = obj.__send__(msg.intern, *argv)
        succ = true
      rescue
        result = $!
        succ = false
      end
      send_reply(s, succ, result)
    ensure
      close s if s
    end
  end
end
```

結果を返信

DRbServer

```
def proc
  ns = @soc.accept
  Thread.start do
    begin
      s = ns
      begin
        ro, msg, argv = recv_request(s)
        if ro and ro.ref
          obj = ObjectSpace._id2ref(ro.ref)
        else
          obj = DRb.front
        end
        result = obj.__send__(msg.intern, *argv)
        succ = true
      rescue
        result = $!
        succ = false
      end
      send_reply(s, su      result)
    ensure
      close s if s
    end
  end
end
```

あ。
間違い..

オブジェクトの交換

```
def dump(obj, soc)
  begin
    str = Marshal::dump(obj)
  rescue
    ro = DRbObject.new(obj)
    str = Marshal::dump(ro)
  end
  soc.write(str) if soc
  return str
end
```

dumpに失敗したら
DRbObjectをdumpする

dRuby作成入門でした

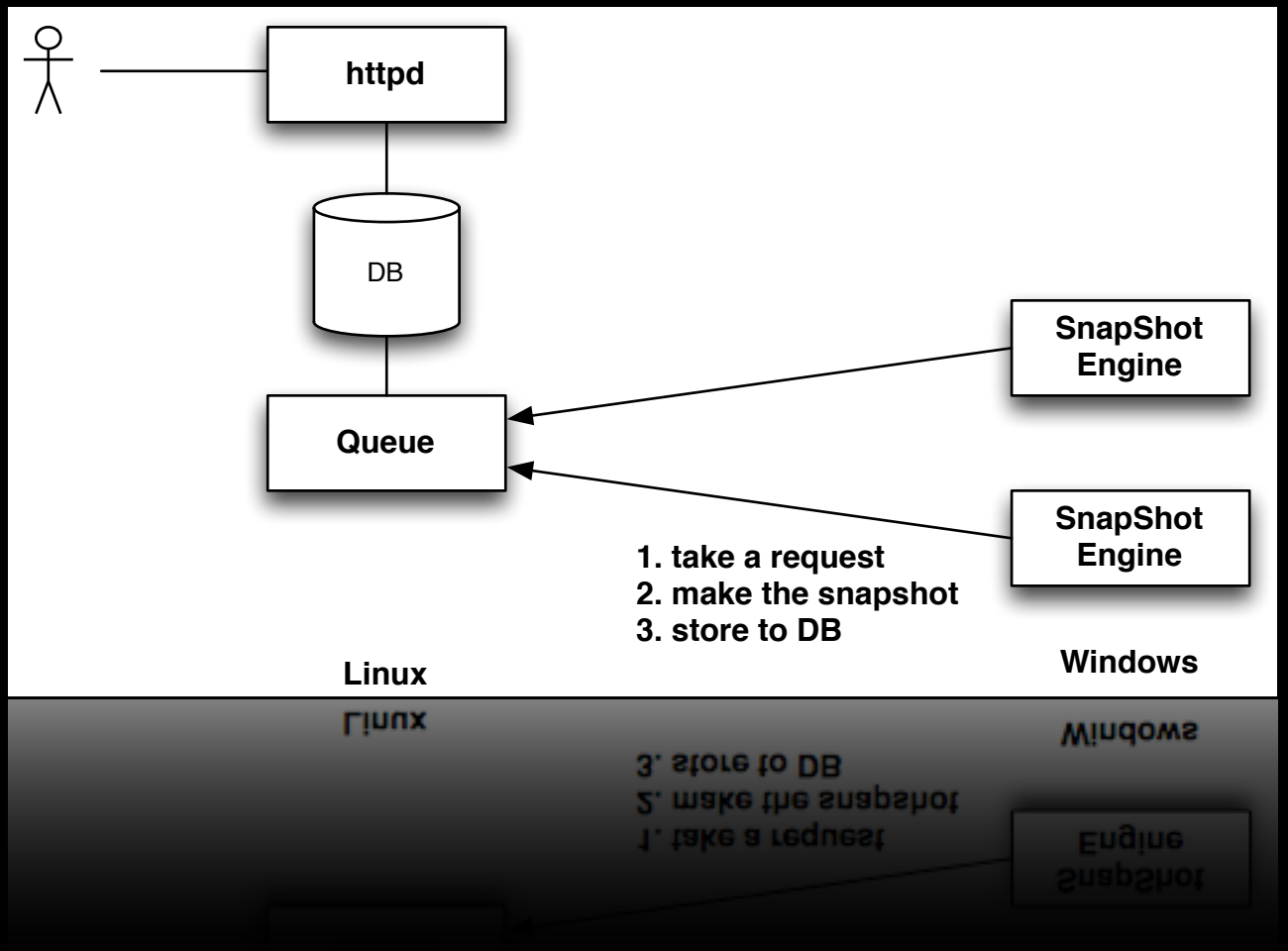
- dRubyを書けるようになりましたね

応用

- たぶんたくさんありそう
- IJPPの論文で引用したものから

hatena screenshot

id:secondlife @ RK06



RWiki

- In-Memory Database with logging
  - すべてメモリの中にあるWiki
    - Objectが潜んでいる
- 25000ページくらいのWikiは実績あるよ
- dRubyやERBなどのサンプル

WEBrick::CGI

- みんな使ってるよね
- WEBrickなサーブレットと同じように書けるCGI
- 環境に依って書き分ける必要がない
  - 抽象化いらないじゃん

同じ値しか返さないカウンタ

```
require 'webrick/cgi'
require 'thread'

class SimpleCountCGI < WEBrick::CGI
  def initialize
    super
    @count = Queue.new
    @count.push(1)
  end

  def count
    value = @count.pop
  ensure
    @count.push(value + 1)
  end

  def do_GET(req, res)
    res['content-type'] = 'text/plain'
    res.body = count.to_s
  end
end

SimpleCountCGI.new.start
```

WEBrick::CGI

- WEBrick::CGIをfrontにしたサービス
- FastCGIもどき
  - お手軽

長生きCGI

```
require 'webrick/cgi'
require 'drb/drb'
require 'thread'

class SimpleCountCGI < WEBrick::CGI
  def initialize
    super
    @count = Queue.new
    @count.push(1)
  end

  def count
    value = @count.pop
  ensure
    @count.push(value + 1)
  end

  def do_GET(req, res)
    res['content-type'] = 'text/plain'
    res.body = count.to_s
  end
end

DRb.start_service('druby://localhost:12321', SimpleCountCGI.new)
sleep
```

cgi.start()の実装は

```
module WEBrick
  class CGI
    ....
    def start(env=ENV, stdin=$stdin, stdout=$stdout)
    ....
```

4 lines CGI

```
#!/usr/local/bin/ruby

require 'drb/'drb'

DRb.start_service('druby://localhost:0')
ro = DRbObject.new_with_uri('druby://localhost:12321')
ro.start(ENV.to_hash, $stdin, $stdout)
```

さよなら遺言モデル

- 状態の永続化とか排他制御とか
- 遺言とか相続問題とか
- 死ななきゃ良いのにね

RindaとLinda

- Lindaは並列処理糊言語
- タプルとタプルスペース
- パターンマッチング
- Lindaを作った人は

author of Linda

- Reviewer#1の人
- 鋭い指摘でずいぶん詳しいと思ったら作者だった
- 余剰資金のある研究室は Must buy!

Int J Parallel Prog
DOI 10.1007/s10766-008-0086-1

dRuby and Rinda: Implementation and Application of Distributed Ruby and its Parallel Coordination Mechanism

Masatoshi Seki

Received: 20 March 2008 / Accepted: 13 August 2008

Abstract The object-oriented scripting language Ruby is admired by many programmers for being easy to write in, and for its flexible, dynamic nature. In the last few years, the Ruby on Rails web application framework, popular for its productivity benefits, has brought about a renewed attention to Ruby for enterprise use. As the focus of Ruby has broadened from small tools and scripts, to large applications, the demands on Ruby's distributed object environment have also increased, as has the need for information about its usage, performance and examples of common practices. dRuby and Rinda were developed by the author as the distributed object environment and shared tuplespace implementation for the Ruby language, and are included as part of Ruby's standard library. dRuby extends method calls across the network while retaining the benefits of Ruby. Rinda builds on dRuby to bring the functionality of Linda, the glue language for distributed co-ordination systems, to Ruby. This article discusses the design policy and implementation points of these two systems, and demonstrates their simplicity with sample code and examples of their usage in actual applications. In addition to dRuby and Rinda's appropriateness for prototyping distributed systems, this article will also demonstrate that dRuby and Rinda are building a reputation for being suitable infrastructure components for real-world applications.

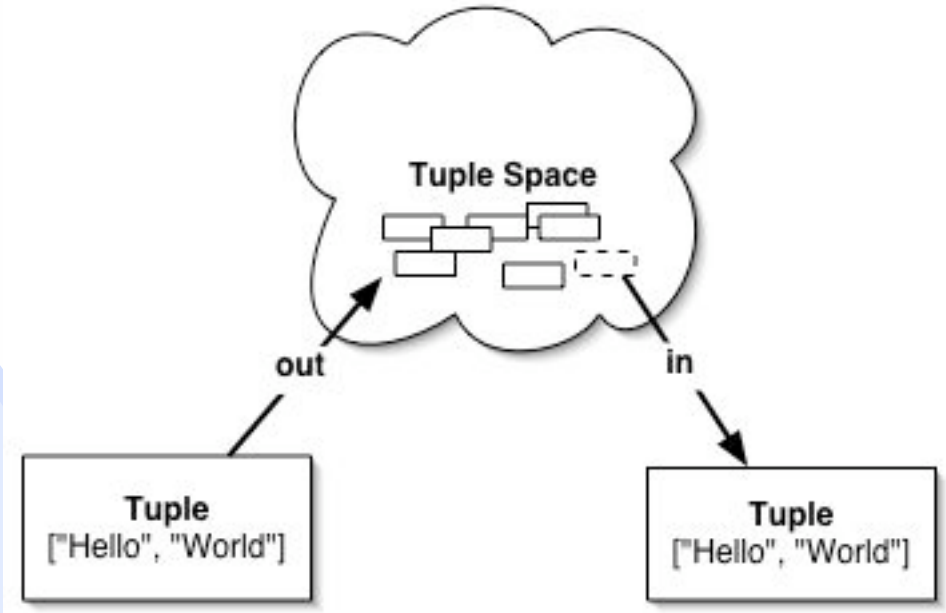
Keywords Ruby · Linda · dRuby · Rinda · TupleSpace · RMI

M. Seki (✉)
Tochigi, Japan
e-mail: m_seki@mva.biglobe.ne.jp
URL: www.druby.org

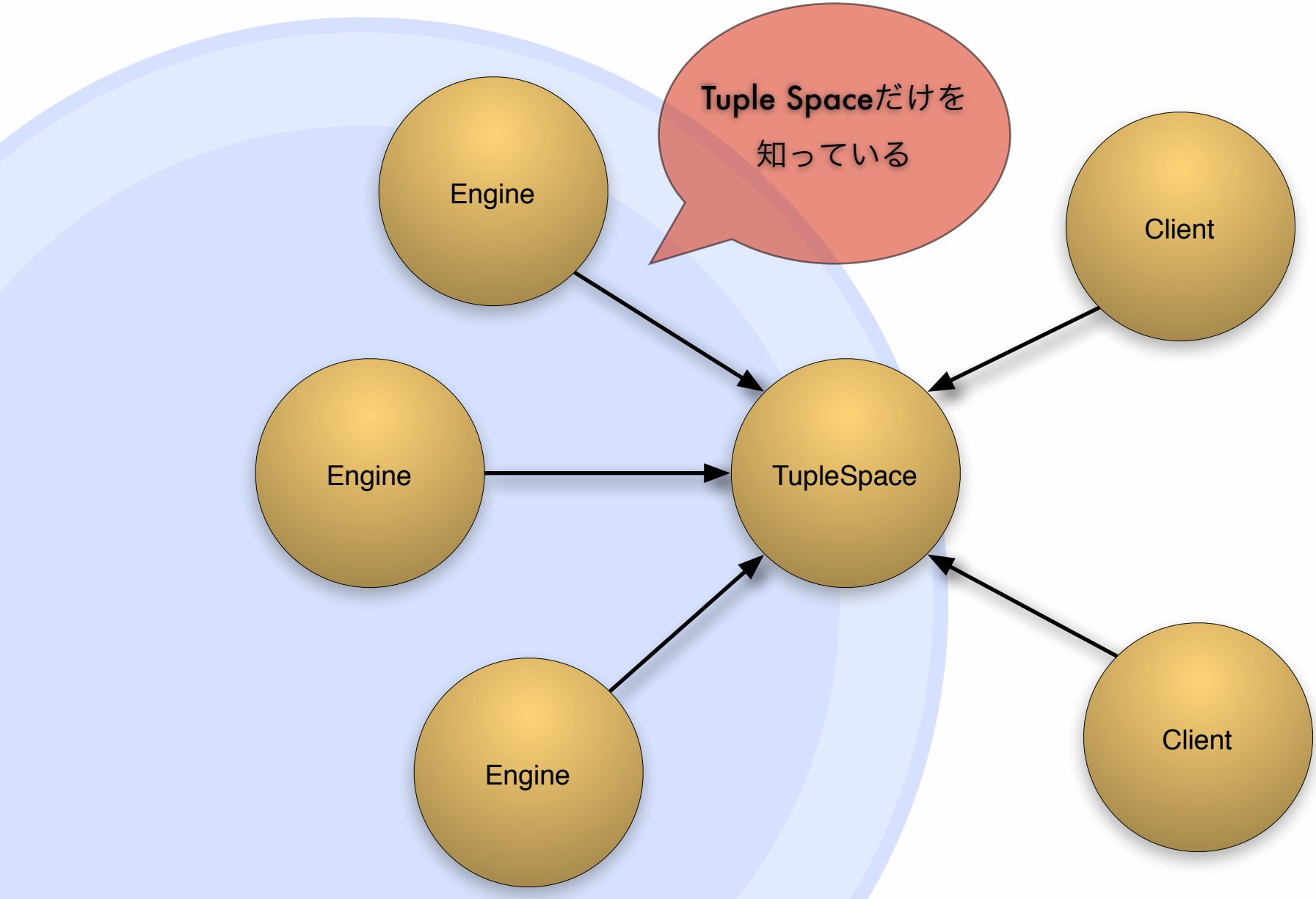
Springer

Lindaの協調

- out
- inとrd
  - パターンで指定
  - ブロックできる



かっこいい並列処理


Tuple Spaceだけを
知っている
Engine
Engine
Engine
TupleSpace
Client
Client

Rinda

- RubyによるLindaの実装
- dRubyを意識してくれる
- TupleはArray|Hashで

Tuple

- Arrayで表現する
  - Hashはやめたほうがいい
- 要素ごとにMarshalする
  - Marshalできない要素も混ぜられる
  - dRubyを意識

Tuple

```
[:chopstick, 2]
[:room_ticket]
[‘abc’, 2, 5]
[:matrix, 1.6, 3.14]
[‘family’, ‘is-sister’, ’Carolyn’, ’Elinor’]
```

Pattern

- Tupleの要素と===で比較する
  - case equalsなのでパターンっぽく動く
  - Regexp, Range, Classとか
- nilはワイルドカード

Pattern

```
[/^A/, nil, nil]
[:matrix, Numeric, Numeric]
['family', 'is-sister', 'Carolyn', nil]
[nil, 'age', (0..18)]
```

['seki', 'age', 20]

```
>> require 'rinda/tuplespace'
>> ts = Rinda::TupleSpace.new
>> ts.write(['seki', 'age', 20])
>> ts.write(['sougo', 'age', 18])
>> ts.write(['leonard', 'age', 18])
>> ts.read_all([nil, 'age', 0..19])
=> [["sougo", "age", 18], ["leonard", "age", 18]]
>> ts.read_all([/^s/, 'age', Numeric])
=> [["seki", "age", 20], ["sougo", "age", 18]]
```

TupleSpace

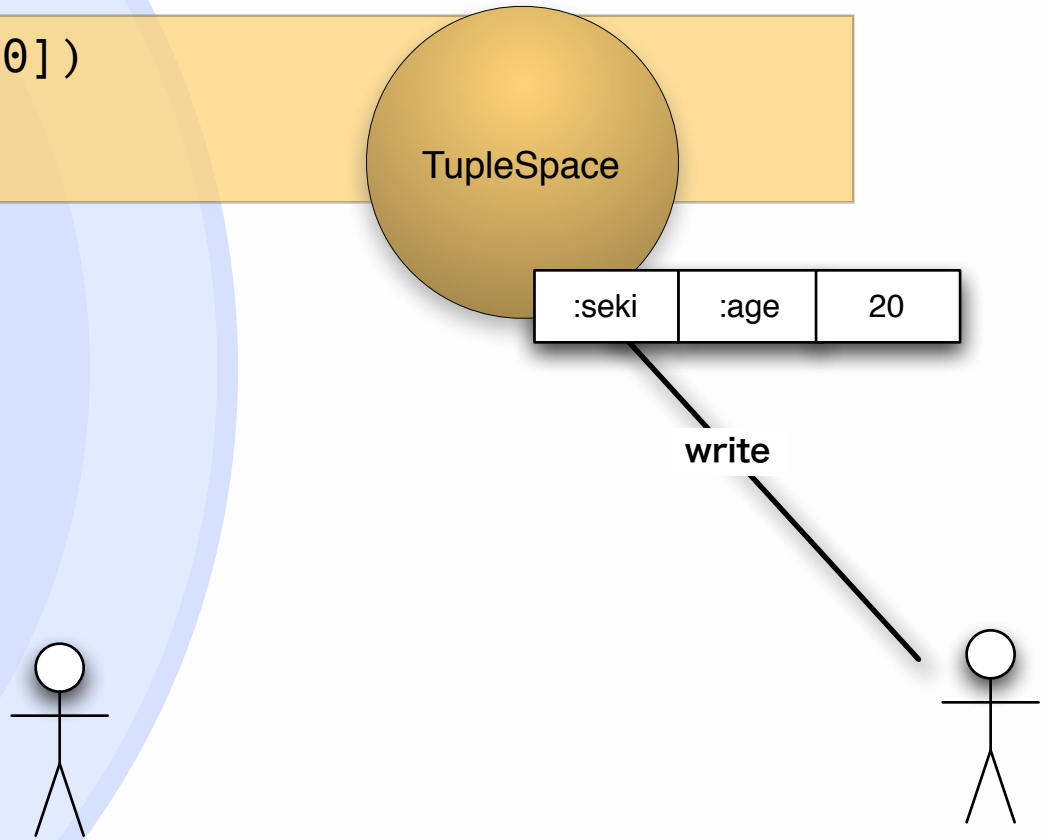
```
>> ts = Rinda::TupleSpace.new
=> #<Rinda::TupleSpace:...>
>> DRb.front['ts'] = ts
=> #<Rinda::TupleSpace:...>
>>
```

```
>> it = DRbObject.new_with_uri('druby://localhost:12345?
ts')
=> #<DRb::DRbObject:...>
>> ts = Rinda::TupleSpaceProxy.new(it)
=> #<Rinda::TupleSpaceProxy:...>
>>
```

TupleSpaceProxyは
dRuby下で安全に動作
させるためのもの

writeとtake

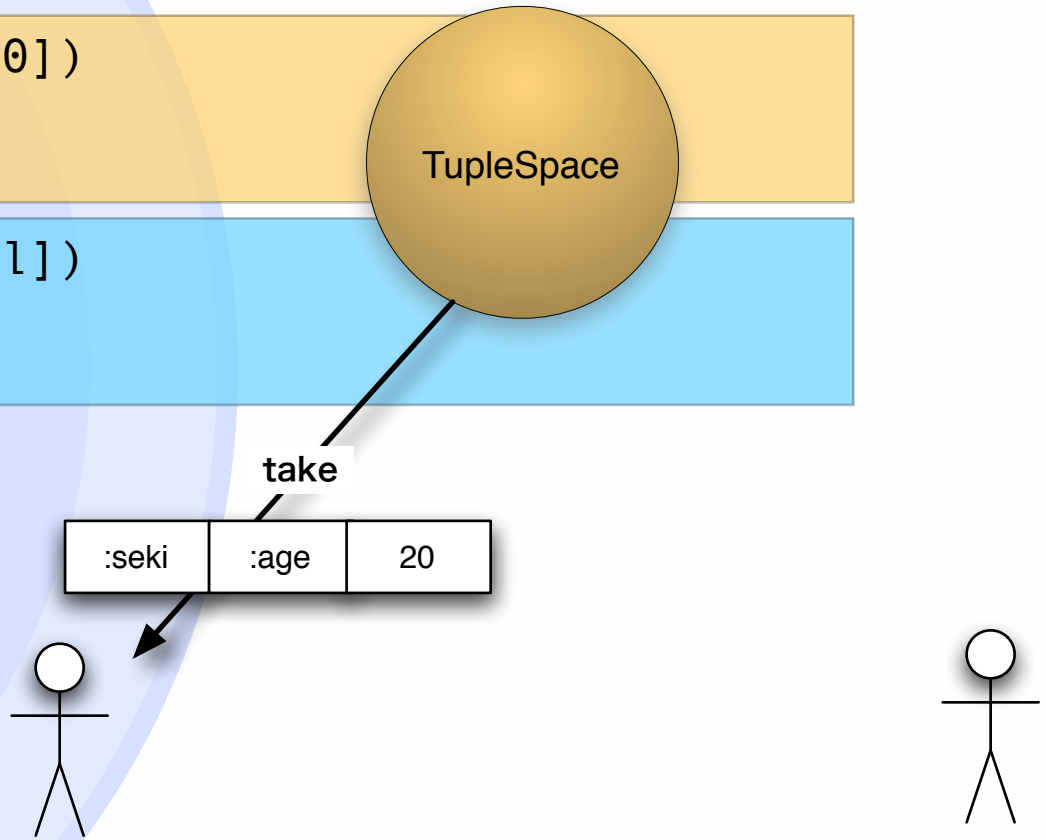
```
>> ts.write([:seki, :age, 20])
=> #<Rinda::TupleEntry:...>
>>
```



writeとtake

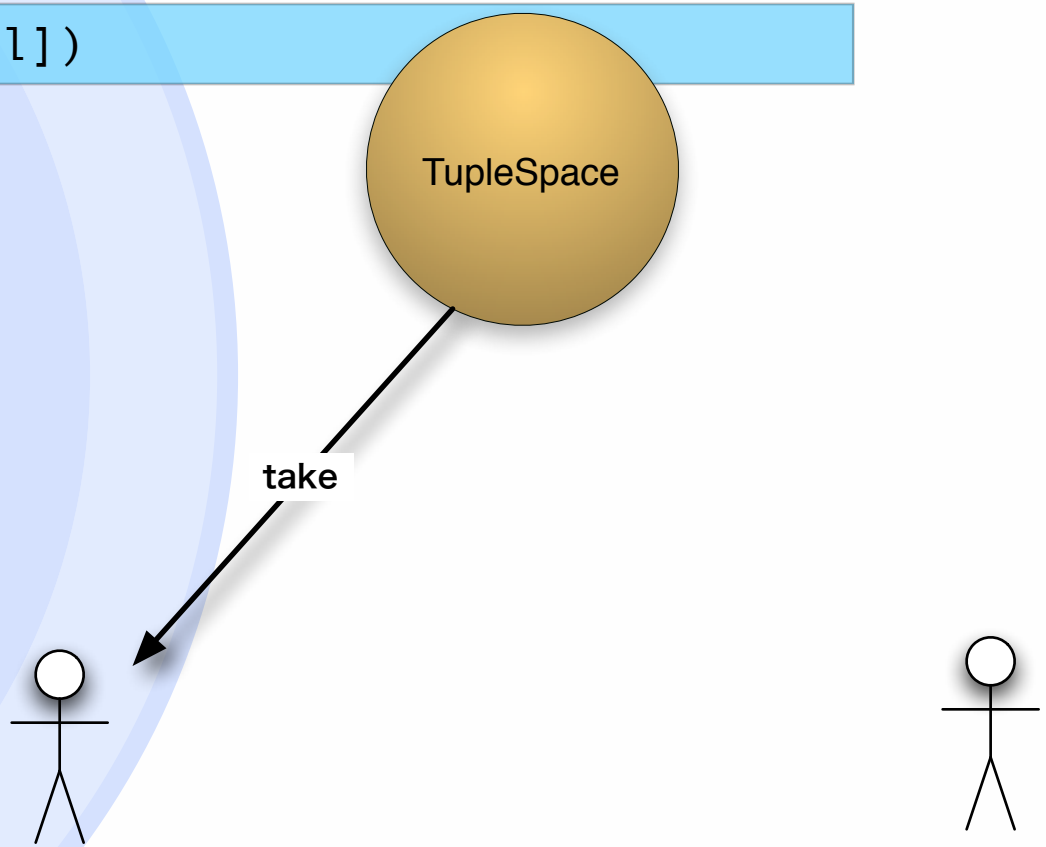
```
>> ts.write([:seki, :age, 20])
=> #<Rinda::TupleEntry:...>
>>
```

```
>> ts.take([:seki, :age, nil])
=> [:seki, :age, 20]
>>
```



待ち合わせ

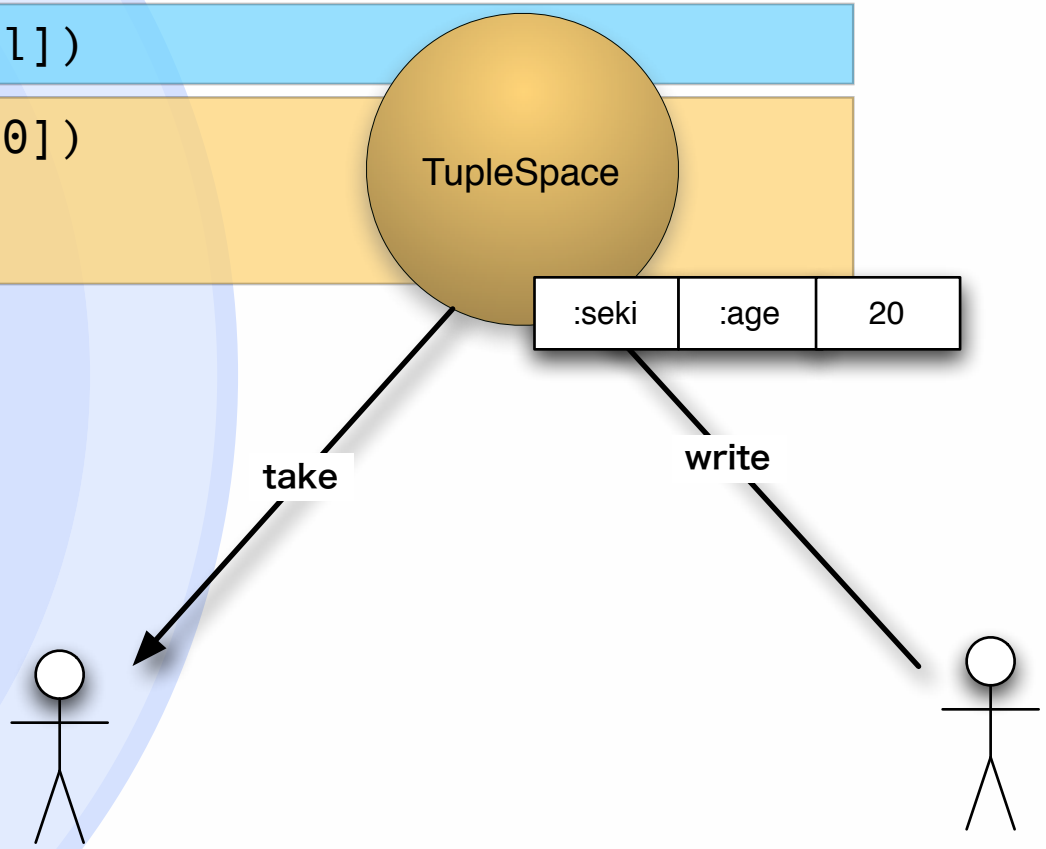
```
>> ts.take([:seki, :age, nil])
```



待ち合わせ

```
>> ts.take([:seki, :age, nil])
```

```
>> ts.write([:seki, :age, 20])
=> #<Rinda::TupleEntry:...>
>>
```

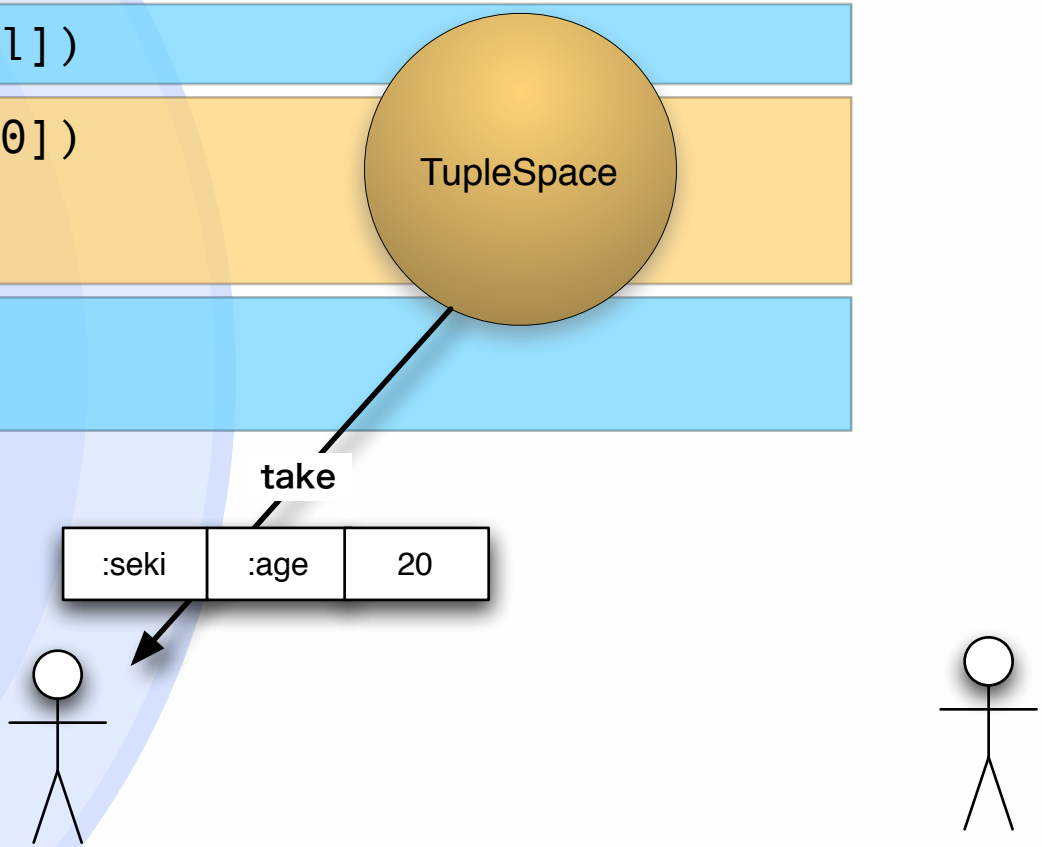


待ち合わせ

```
>> ts.take([:seki, :age, nil])
```

```
>> ts.write([:seki, :age, 20])
=> #<Rinda::TupleEntry:...>
>>
```

```
=> [:seki, :age, 20]
>>
```



Lindaはかっこいい

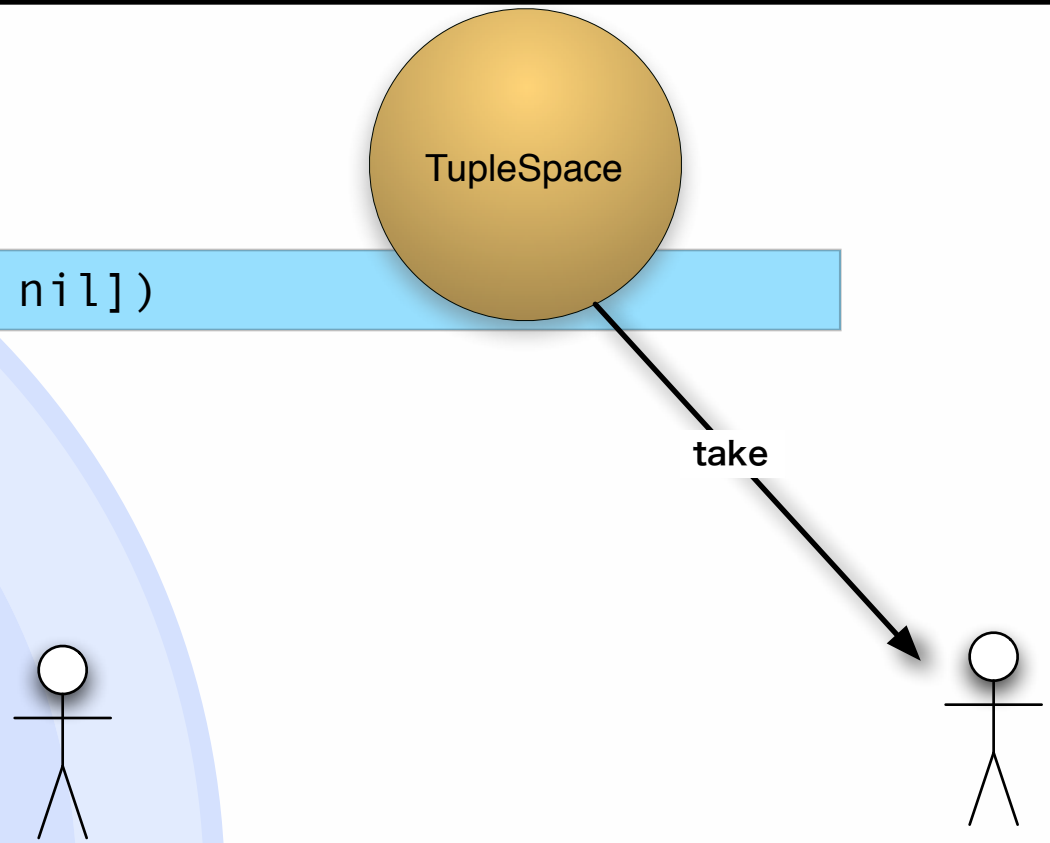
- 限られた操作で並列処理のさまざまな協調を記述できる
  - 一般的な同期メカニズムを記述することもできる
- コンセプトのスケッチにぴったり

安全なカウンタ

- 値のタプルを資源と考える
- 同時にただ一つのプロセスだけが値を更新できる

安全なカウンタ

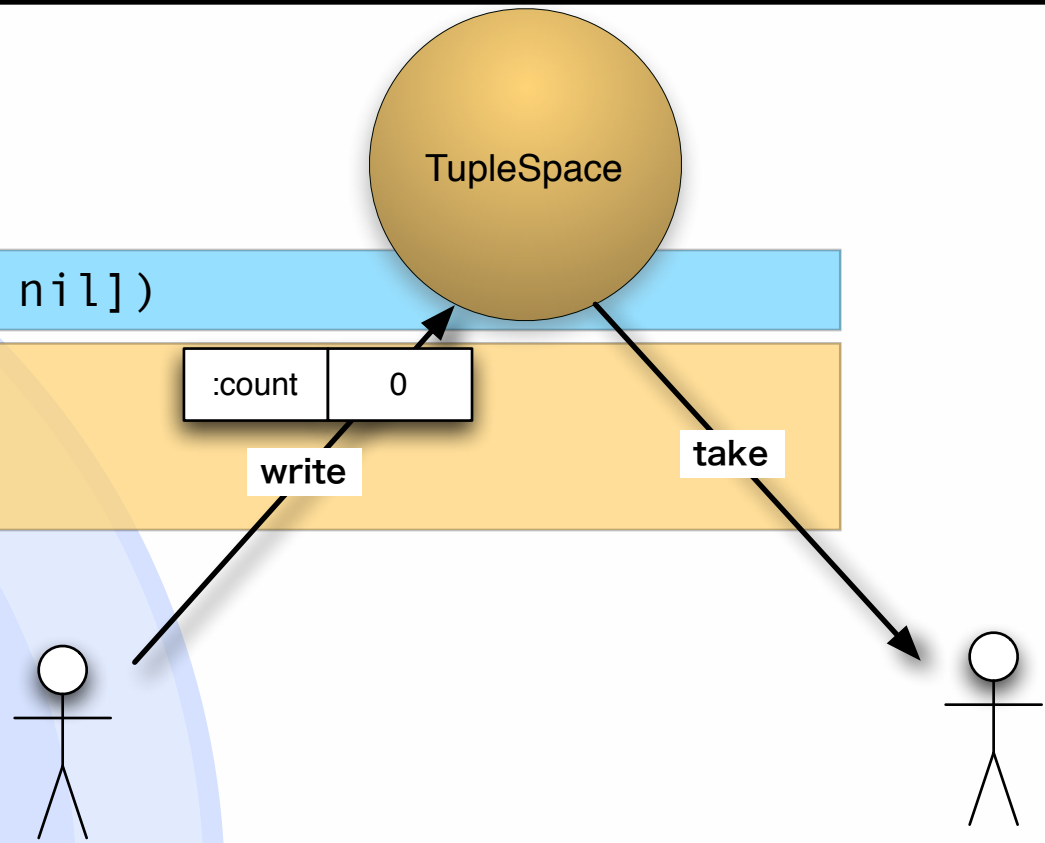
```
>> tuple = ts.take([:count, nil])
```



安全なカウンタ

```
>> tuple = ts.take([:count, nil])
```

```
>> ts.write([:count, 0])
=> #<Rinda::TupleEntry:...>
>>
```

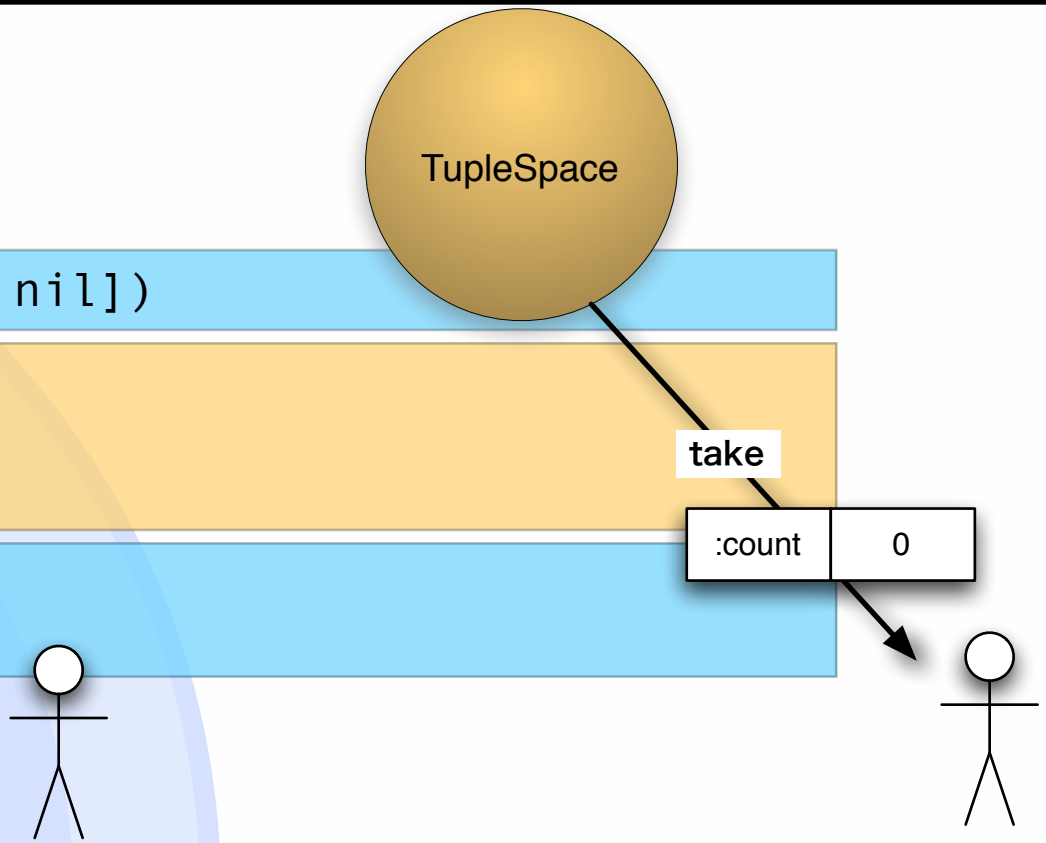


安全なカウンタ

```
>> tuple = ts.take([:count, nil])
```

```
>> ts.write([:count, 0])
=> #<Rinda::TupleEntry:...>
>>
```

```
=> [:count, 0]
>>
```



安全なカウンタ

```
>> tuple = ts.take([:count, nil])
```

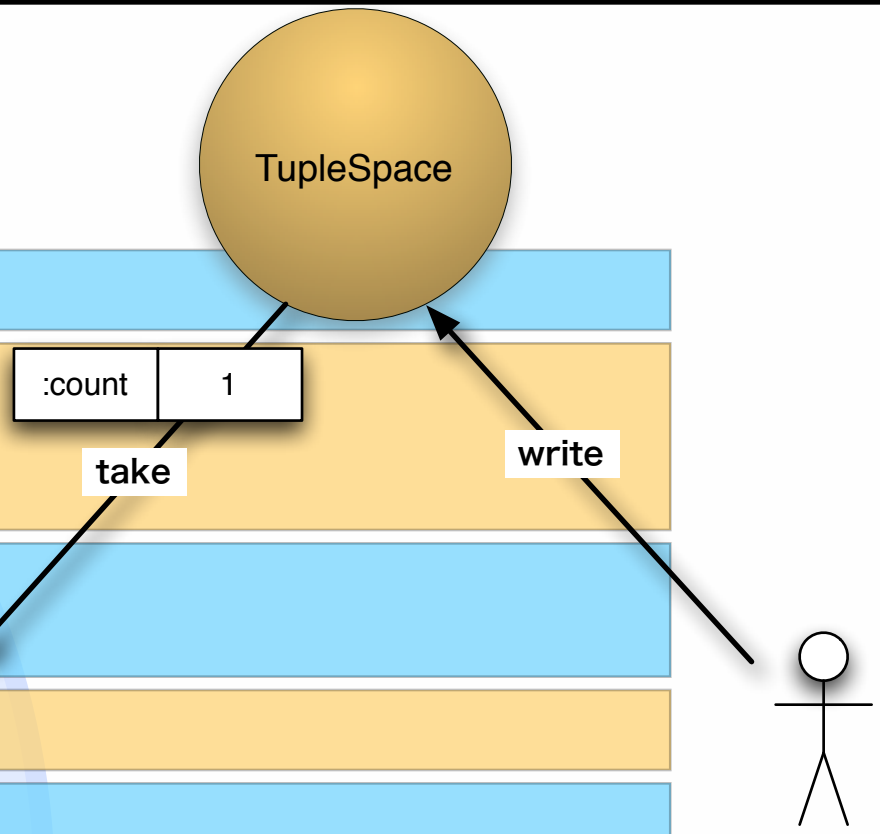
```
>> ts.write([:count, 0])
=> #<Rinda::TupleEntry:...>
>>
```

```
=> [:count, 0]
>>
```

```
>> tuple = ts.take([:count, nil])
```

```
>> tuple[1] += 1
=> 1
>> ts.write(tuple)
=> #<DRb::DRbObject:...>
>>
```

```
=> [:count, 1]
>>
```



不公平な最適化

- 先頭の要素がSymbolである場合に検索が速く動作するようになってるよ!
  - そんなにたくさんのTupleを投入してるとしたらなにか間違ってると思うけどね

間違いの例

- MapReduceのストレージに使う!

MapReduce

- [名前, 値]のタプルを扱う小さなプログラムで大量の情報を処理するかっこいい作戦
- あとは自分で調べてね
  - WEB+DB #47に良い記事があったよ

元ネタを発生させる

```
bin = []
while line = gets
  line.scan(/\w+/) do |w|
    bin << [w, 1]
  end
end
```

キー順に並べる

```
bin.sort_by {|pair| pair[0]}
```

キーごとにまとめる

```
word = nil
sum = 0
bin.sort_by {|pair| pair[0]}.each do |key, value|
  if word && word != key
    p [word, sum]
    sum = 0
  end
  word = key
  sum += value
end
p [word, sum]
```

Hashでさぼることも

小さい空間に向く

```
bin = Hash.new {|h, k| h[k] = []}
while line = gets
  line.scan(/\w+/) do |w|
    bin[w] << 1
  end
end

bin.keys.sort.each do |k|
  p [k, bin[k].reduce(:+)]
end
```

TupleSpaceはHash?

- TupleSpaceはHashみたいなもん
- 知ってるキーへのアクセスが得意
  - つまり
- 知らないキーが苦手
- 知らないかもしれないキーも苦手

TupleSpaceの憂鬱

- 知らないキーへのアクセスが面倒
- 新規のキーを特別扱いするのも辛い

Tiny MapReduce by TupleSpace

```
bin = Rinda::TupleSpace.new
while line = gets
  line.scan(/\w+/) do |w|
    bin.write([w, 1])
  end
end
```

小さいキーから順番に欲しい

全部読むの!?

```
while true
  key ,= bin.read_all([String, nil]).min_by {|k, v| k}
  break unless key
  sum = 0
  while true
    begin
      key, value = bin.take([key, nil], 0)
      sum += value
    rescue
      break
    end
  end
  p [key, sum]
end
```

そのキーのデータを全部くれ

```
while true
  key ,= bin.read_all([String, nil]).min
  break unless key
  sum = 0
  while true
    begin
      key, value = bin.take([key, nil], 0)
      sum += value
    rescue
      break
    end
  end
  p [key, sum]
end
```

takeが失敗
するまで
またtake...

そういえば

- よく考えたらMapReduceはデータのやりとりのところは待ち合わせしなくていいじゃん
- タスク(フェーズ)遷移の待ち合わせに向いてる

TokyoCabinet 一族かっこいい

- いつも並んでいる集合をくれる、QDBMとかTokyoCabinetとかBigTableとか
- 広大な空間をシーケンシャルに扱う
- どんなデータ構造もたどるときは線形だ

OODB Koya

- ここからさらに脱線しますよ
- 小さなOODBの実装
- 任意の時刻の状態に戻ることができる

Koyaの変遷

- PRb - PostgreSQLをメモリ空間としたOODB
- 世界はオブジェクトIDの並びである
- めちゃくちゃ遅い

Koya
with SQLite2

- SQLiteは文字列だけを扱うawkみたいなSQL風ストレージでUNIXの匂いがする
- 型の呪縛からの解放
- インスタンス変数を知るには検索になる

Koya
with QDBM

- Hashではない
- いつも並んでいる集合を提供
- あるデータのとなりのデータを触れる
- キーを工夫するとオブジェクトと属性をまとめて扱える

世界はHashだと思ってた

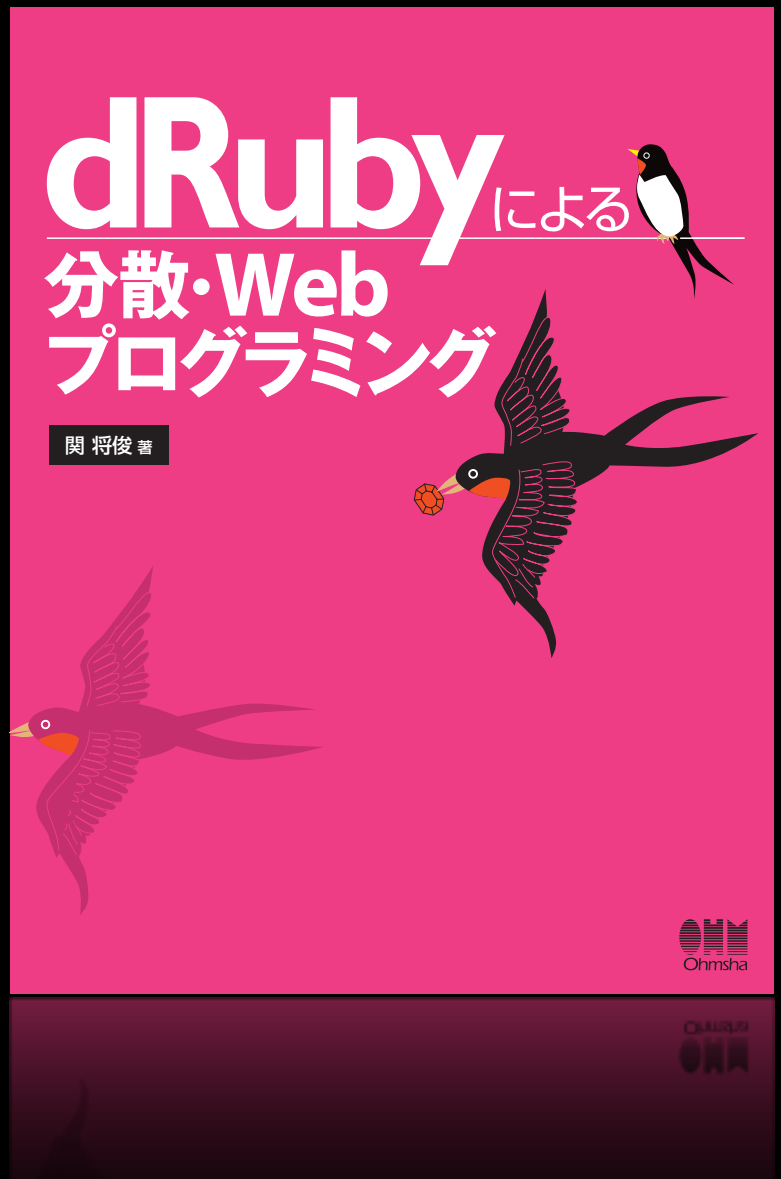
- 実はBigTableだったことに気付いた
  - ちょっと嘘
- 世界はシーケンシャルにアクセス

重要なことをもう一度

- もう一度

￥3360

- 初刷まだ買えます



Who is translating it?

- 英語版はまだ買えません

$32.00

- International Journal of PARALLEL PROGRAMING
- コレクターズアイテム
- 余剰資金のある研究室は Must buy!

Int J Parallel Prog
DOI 10.1007/s10766-008-0086-1

dRuby and Rinda: Implementation and Application of Distributed Ruby and its Parallel Coordination Mechanism

Masatoshi Seki

Received: 20 March 2008 / Accepted: 13 August 2008

Abstract The object-oriented scripting language Ruby is admired by many programmers for being easy to write in, and for its flexible, dynamic nature. In the last few years, the Ruby on Rails web application framework, popular for its productivity benefits, has brought about a renewed attention to Ruby for enterprise use. As the focus of Ruby has broadened from small tools and scripts, to large applications, the demands on Ruby's distributed object environment have also increased, as has the need for information about its usage, performance and examples of common practices. dRuby and Rinda were developed by the author as the distributed object environment and shared tuplespace implementation for the Ruby language, and are included as part of Ruby's standard library. dRuby extends method calls across the network while retaining the benefits of Ruby. Rinda builds on dRuby to bring the functionality of Linda, the glue language for distributed co-ordination systems, to Ruby. This article discusses the design policy and implementation points of these two systems, and demonstrates their simplicity with sample code and examples of their usage in actual applications. In addition to dRuby and Rinda's appropriateness for prototyping distributed systems, this article will also demonstrate that dRuby and Rinda are building a reputation for being suitable infrastructure components for real-world applications.

Keywords Ruby · Linda · dRuby · Rinda · TupleSpace · RMI

M. Seki (✉)
Tochigi, Japan
e-mail: m_seki@mva.biglobe.ne.jp
URL: www.druby.org

Springer

http://dx.doi.org/10.1007/s10766-008-0086-1

まとめ

- まだ初刷買えます。
- 余剰資金のある研究室はMust buy!
- dRubyの作り方入門