Extracted from:

Programming Ruby

The Pragmatic Programmers' Guide

Second Edition

Library

Observable — The Observer Pattern

The Observer pattern, also known as Publish/Subscribe, provides a simple mechanism for one object (the source) to inform a set of interested third-party objects when its state changes (see *Design Patterns* [GHJV95]). In the Ruby implementation, the notifying class mixes in the module `Observable`, which provides the methods for managing the associated observer objects. The observers must implement the `update` method to receive notifications.

```
require 'observer'
class CheckWaterTemperature # Periodically check the water
  include Observable
  def run
    last_temp = nil
    loop do
      temp = Temperature.fetch   # external class...
      puts "Current temperature: #{temp}"
      if temp != last_temp
        changed                  # notify observers
        notify_observers(Time.now, temp)
        last_temp = temp
      end
    end
  end
end
class Warner
  def initialize(&limit)
    @limit = limit
  end
  def update(time, temp)         # callback for observer
    if @limit.call(temp)
      puts "--- #{time.to_s}: Temperature outside range: #{temp}"
    end
  end
end
checker = CheckWaterTemperature.new
checker.add_observer(Warner.new {|t| t < 80})
checker.add_observer(Warner.new {|t| t > 120})
checker.run
```

produces:

```
Current temperature: 83
Current temperature: 75
--- Thu Aug 26 22:38:59 CDT 2004: Temperature outside range: 75
Current temperature: 90
Current temperature: 134
--- Thu Aug 26 22:38:59 CDT 2004: Temperature outside range: 134
Current temperature: 134
Current temperature: 112
Current temperature: 79
--- Thu Aug 26 22:38:59 CDT 2004: Temperature outside range: 79
```

Library

open-uri — Treat FTP and HTTP Resources as Files

The open-uri library extends Kernel#open, allowing it to accept URIs for FTP and HTTP as well as local filenames. Once opened, these resources can be treated as if they were local files, accessed using conventional IO methods. The URI passed to open is either a string containing an HTTP or FTP URL, or a URI object (described on page 731). When opening an HTTP resource, the method automatically handles redirection and proxies. When using an FTP resource, the method logs in as an anonymous user.

The IO object returned by open in these cases is extended to support methods that return meta-information from the request: content_type, charset, content_encoding, last_modified, status, base_uri, meta.

See also: URI (page 731)

```
require 'open-uri'
require 'pp'
open('http://localhost/index.html') do |f|
  puts "URI: #{f.base_uri}"
  puts "Content-type: #{f.content_type}, charset: #{f.charset}"
  puts "Encoding: #{f.content_encoding}"
  puts "Last modified: #{f.last_modified}"
  puts "Status: #{f.status.inspect}"
  pp f.meta
  puts "----"
  3.times {|i| puts "#{i}: #{f.gets}" }
end
```

produces:

```
URI: http://localhost/index.html
Content-type: text/html, charset: iso-8859-1
Encoding:
Last modified: Wed Jul 18 23:44:21 UTC 2001
Status: ["200", "OK"]
{"vary"=>"negotiate,accept-language,accept-charset",
 "last-modified"=>"Wed, 18 Jul 2001 23:44:21 GMT",
 "content-location"=>"index.html.en",
 "date"=>"Fri, 27 Aug 2004 03:38:59 GMT",
 "etag"=>"\"6657-5b0-3b561f55;411edab5\"",
 "content-type"=>"text/html",
 "content-language"=>"en",
 "server"=>"Apache/1.3.29 (Darwin)",
 "content-length"=>"1456",
 "tcn"=>"choice",
 "accept-ranges"=>"bytes"}
----
0: <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
1:     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2: <html xmlns="http://www.w3.org/1999/xhtml">
```

Library **Open3** Run Subprocess and Connect to All Streams

Runs a command in a subprocess. Data written to *stdin* can be read by the subprocess, and data written to standard output and standard error in the subprocess will be available on the *stdout* and *stderr* streams. The subprocess is actually run as a grandchild, and as a result `Process#waitall` cannot be used to wait for its termination (hence the sleep in the following example).

```
require 'open3'
Open3.popen3('bc') do | stdin, stdout, stderr |
  Thread.new { loop { puts "Err stream:    #{stderr.gets}" } }
  Thread.new { loop { puts "Output stream: #{stdout.gets}" } }
  stdin.puts "3 * 4"
  stdin.puts "1 / 0"
  stdin.puts "2 ^ 5"
  sleep 0.1
end
```

produces:

```
Output stream: 12
Err stream:    Runtime error (func=(main), adr=3): Divide by zero
Output stream: 32
Err stream:
```

Library

OpenSSL — SSL Library

Only if: OpenSSL library available

The Ruby OpenSSL extension wraps the freely available OpenSSL library. It provides Secure Sockets Layer and Transport Layer Security (SSL and TLS) protocols, allowing for secure communications over networks. The library provides functions for certificate creation and management, message signing, and encryption/decryption. It also provides wrappers to simplify access to `https` servers, along with secure FTP. The interface to the library is large (roughly 330 methods), but the average Ruby user will probably only use a small subset of the library's capabilities.

See also: `Net::FTP` (page 677), `Net::HTTP` (page 678), `Socket` (page 714)

- Access a secure Web site using HTTPS. Note that SSL is used to tunnel to the site, but the requested page also requires standard HTTP basic authorization.

```
require 'net/https'
USER = "xxx"
PW   = "yyy"
site = Net::HTTP.new("www.securestuff.com", 443)
site.use_ssl = true
response = site.get2("/cgi-bin/cokerecipe.cgi",
                     'Authorization' => 'Basic ' +
                     ["#{USER}:#{PW}"].pack('m').strip)
```

- Create a socket that uses SSL. This isn't a good example of accessing a Web site. However, it illustrates how a socket can be encrypted.

```
require 'socket'
require 'openssl'
socket = TCPSocket.new("www.secure-stuff.com", 443)
ssl_context = OpenSSL::SSL::SSLContext.new()
unless ssl_context.verify_mode
  warn "warning: peer certificate won't be verified this session."
  ssl_context.verify_mode = OpenSSL::SSL::VERIFY_NONE
end
sslsocket = OpenSSL::SSL::SSLSocket.new(socket, ssl_context)
sslsocket.sync_close = true
sslsocket.connect
sslsocket.puts("GET /secret-info.shtml")
while line =  sslsocket.gets
  p line
end
```

Library

OpenStruct — Open (dynamic) Structure

An open structure is an object whose attributes are created dynamically when first assigned. In other words, if *obj* is an instance of an `OpenStruct`, then the statement `obj.abc=1` will create the attribute *abc* in *obj*, and then assign the value 1 to it.

```
require 'ostruct'

os = OpenStruct.new( "f1" => "one", :f2 => "two" )
os.f3 = "cat"
os.f4 = 99
os.f1   →   "one"
os.f2   →   "two"
os.f3   →   "cat"
os.f4   →   99
```

Library **OptionParser** Option Parsing

`OptionParser` is a flexible and extensible way to parse command-line arguments. It has a particularly rich abstraction of the concept of an option.

- An option can have multiple short names (options preceded by a single hyphen) and multiple long names (options preceded by two hyphens). Thus, an option that displays help may be available as `-h`, `-?`, `--help`, and `--about`. Users may abbreviate long option names to the shortest nonambiguous prefix.
- An option may be specified as having no argument, an optional argument, or a required argument. Arguments can be validated against patterns or lists of valid values.
- Arguments may be returned as objects of any type (not just strings). The argument type system is extensible (we add `Date` handling in the example).
- Arguments can have one or more lines of descriptive text, used when generating usage information.

Options are specified using the on and `def` methods. These methods take a variable number of arguments that cumulatively build a definition of each option. The arguments accepted by these methods are listed in Table 28.2 on the following page.

See also: `GetoptLong` (page 663)

```
require 'optparse'
require 'date'
# Add Dates as a new option type
OptionParser.accept(Date, /(\d+)-(\d+)-(\d+)/) do |d, mon, day, year|
  Date.new(year.to_i, mon.to_i, day.to_i)
end
opts = OptionParser.new
opts.on("-x")                          {|val| puts "-x seen" }
opts.on("-s", "--size VAL", Integer) {|val| puts "-s #{val}" }
opts.on("-a", "--at DATE",  Date)    {|val| puts "-a #{val}" }
my_argv = [ "--size", "1234", "-x", "-a", "12-25-2003", "fred", "wilma" ]
rest = opts.parse(*my_argv)
puts "Remainder = #{rest.join(', ')}"
puts opts.to_s
```

produces:

```
-s 1234
-x seen
-a 2003-12-25
Remainder = fred, wilma
Usage: myprog [options]
    -x
    -s, --size VAL
    -a, --at DATE
```

Table 28.2. Option definition arguments

"-x" "-xARG" "-x=ARG" "-x[OPT]" "-x[=OPT]" "-x PLACE"
: Option has short name x. First form has no argument, next two have mandatory argument, next two have optional argument, last specifies argument follows option. The short names may also be specified as a range (such as "-[a-c]").

"--*switch*" "--*switch*=ARG" "--*switch*=[OPT]" "--*switch* PLACE"
: Option has long name switch. First form has no argument, next has a mandatory argument, the next has an optional argument, and the last specifies the argument follows the switch.

"--no-*switch*"
: Defines a option whose default value is false.

"=ARG" "=[OPT]"
: Argument for this option is mandatory or optional. For example, the following code says there's an option known by the aliases -x, -y, and -z that takes a mandatory argument, shown in the usage as N.

```
opt.on("-x", "-y", "-z", "=N")
```

"description"
: Any string that doesn't start - or = is used as a description for this option in the summary. Multiple descriptions may be given; they'll be shown on additional lines.

/pattern/
: Any argument must match the given pattern.

array
: Argument must be one of the values from array.

proc or method
: Argument type conversion is performed by the given proc or method (rather than using the block associated with the on or def method call).

ClassName
: Argument must match that defined for ClassName, which may be predefined or added using OptionParser.accept. Built-in argument classes are

  Object: Any string. No conversion. This is the default.

  String: Any nonempty string. No conversion.

  Integer: Ruby/C-like integer with optional sign (0ddd is octal, 0bddd binary, 0xddd hexadecimal). Converts to Integer.

  Float: Float number format. Converts to Float.

  Numeric: Generic numeric format. Converts to Integer for integers, Float for floats.

  Array: Argument must be of list of strings separated by a comma.

  OptionParser::DecimalInteger: Decimal integer. Converted to Integer.

  OptionParser::OctalInteger: Ruby/C-like octal/hexadecimal/binary integer.

  OptionParser::DecimalNumeric: Decimal integer/float number. Integers converted to Integer, floats to Float.

  TrueClass, FalseClass: Boolean switch.

Library

ParseDate

Parse a Date String

The `ParseDate` module defines a single method, `ParseDate.parsedate`, which converts a date and/or time string into an array of `Fixnum` values representing the date and/or time's constituents (year, month, day, hour, minute, second, time zone, and weekday). `nil` is returned for fields that cannot be parsed from the string. If the result contains a year that is less than 100 and the *guess* parameter is true, `parsedate` will return a year value equal to *year* plus 2000 if *year* is less than 69, and will return *year* plus 1900 otherwise.

See also: `Date` (page 644)

		ParseDate::parsedate(string, guess)							
string	**guess**	**yy**	**mm**	**dd**	**hh**	**min**	**sec**	**zone**	**wd**
1999-09-05 23:55:21+0900	F	1999	9	5	23	55	21	+0900	–
1983-12-25	F	1983	12	25	–	–	–	–	–
1965-11-10 T13:45	F	1965	11	10	13	45	–	–	–
10/9/75 1:30pm	F	75	10	9	13	30	–	–	–
10/9/75 1:30pm	T	1975	10	9	13	30	–	–	–
Wed Feb 2 17:15:49 CST 2000	F	2000	2	2	17	15	49	CST	3
Tue, 02-Mar-99 11:20:32 GMT	F	99	3	2	11	20	32	GMT	2
Tue, 02-Mar-99 11:20:32 GMT	T	1999	3	2	11	20	32	GMT	2
12-January-1990, 04:00 WET	F	1990	1	12	4	0	–	WET	–
4/3/99	F	99	4	3	–	–	–	–	–
4/3/99	T	1999	4	3	–	–	–	–	–
10th February, 1976	F	1976	2	10	–	–	–	–	–
March 1st, 84	T	1984	3	1	–	–	–	–	–
Friday	F	–	–	–	–	–	–	–	5

Library

Pathname — Representation of File Paths

A `Pathname` represents the absolute or relative name of a file. It has two distinct uses. First, it allows manipulation of the parts of a file path (extracting components, building new paths, and so on). Second (and somewhat confusingly), it acts as a façade for some methods in classes `Dir`, `File`, and module `FileTest`, forwarding on calls for the file named by the `Pathname` object.

See also: `File` (page 444)

- Path name manipulation:

```
require 'pathname'

p1 = Pathname.new("/usr/bin")   →   #<Pathname:/usr/bin>
p2 = Pathname.new("ruby")       →   #<Pathname:ruby>
p3 = p1 + p2                    →   #<Pathname:/usr/bin/ruby>
p4 = p2 + p1                    →   #<Pathname:/usr/bin>
p3.parent                       →   #<Pathname:/usr/bin>
p3.parent.parent                →   #<Pathname:/usr>
p1.absolute?                    →   true
p2.absolute?                    →   false
p3.split                        →   [#<Pathname:/usr/bin>,
                                     #<Pathname:ruby>]

p5 = Pathname.new("testdir")    →   #<Pathname:testdir>

p5.realpath   →   #<Pathname:/Users/dave/Work/rubybook/testdir>
p5.children   →   [#<Pathname:testdir/config.h>,
                   #<Pathname:testdir/main.rb>]
```

- Pathname as proxy for file and directory status requests.

```
require 'pathname'

p1 = Pathname.new("/usr/bin/ruby")
p1.file?                        →   true
p1.directory?                   →   false
p1.executable?                  →   true
p1.size                         →   1913444

p2 = Pathname.new("testfile")   →   #<Pathname:testfile>

p2.read                         →   "This is line one\nThis is
                                     line two\nThis is line
                                     three\nAnd so on...\n"
p2.readlines                    →   ["This is line one\n", "This
                                     is line two\n", "This is line
                                     three\n", "And so on...\n"]
```

Library **PP** Pretty-print Objects

PP uses the PrettyPrint library to format the results of inspecting Ruby objects. As well as the methods in the class, it defines a global function, pp, which works like the existing p method but which formats its output.

PP has a default layout for all Ruby objects. However, you can override the way it handles a class by defining the method pretty_print, which takes a PP object as a parameter. It should use that PP object's methods text, breakable, nest, group, and pp to format its output (see PrettyPrint for details).

See also: PrettyPrint (page 695), YAML (page 737)

- Compare "p" and "pp."

```
require 'pp'
Customer = Struct.new(:name, :sex, :dob, :country)
cust = Customer.new("Walter Wall", "Male", "12/25/1960", "Niue")
puts "Regular print"
p cust
puts "\nPretty print"
pp cust
```

produces:

```
Regular print
#<struct Customer name="Walter Wall", sex="Male", dob="12/25/1960",
 country="Niue">

Pretty print
#<struct Customer
 name="Walter Wall",
 sex="Male",
 dob="12/25/1960",
 country="Niue">
```

- You can tell PP not to display an object if it has already displayed it.

```
require 'pp'
a = "string"
b = [ a ]
c = [ b, b ]
PP.sharing_detection = false
pp c
PP.sharing_detection = true
pp c
```

produces:

```
[["string"], ["string"]]
[["string"], [...]]
```

Library

PrettyPrint — General Pretty Printer

PrettyPrint implements a pretty printer for structured text. It handles details of wrapping, grouping, and indentation. The PP library uses PrettyPrint to generate more legible dumps of Ruby objects.

See also: PP (page 694)

- The following program prints a chart of Ruby's classes, showing subclasses as a bracketed list following the parent. To save some space, we show just the classes in the Numeric branch of the tree.

```
require 'prettyprint'
require 'complex'
require 'rational'
@children = Hash.new { |h,k| h[k] = Array.new }
ObjectSpace.each_object(Class) do |cls|
  @children[cls.superclass] << cls if cls <= Numeric
end
def print_children_of(printer, cls)
  printer.text(cls.name)
  kids = @children[cls].sort_by {|k| k.name}
  unless kids.empty?
    printer.group(0, " [", "]") do
      printer.nest(3) do
        printer.breakable
        kids.each_with_index do |k, i|
          printer.breakable unless i.zero?
          print_children_of(printer, k)
        end
      end
      printer.breakable
    end
  end
end
printer = PrettyPrint.new($stdout, 30)
print_children_of(printer, Object)
printer.flush
```

produces:

```
Object [
   Numeric [
      Complex
      Float
      Integer [
         Bignum
         Fixnum
      ]
      Rational
   ]
]
```

Library

Profile

Profile Execution of a Ruby Program

The `profile` library is a trivial wrapper around the `Profiler` module, making it easy to profile the execution of an entire program. Profiling can be enabled from the command line using the `-rprofile` option or from within a source program by requiring the `profile` module.

See also: `Benchmark` (page 636), `Profiler__` (page 697)

```
require 'profile'
def ackerman(m, n)
  if m == 0 then  n+1
  elsif n == 0 and m > 0 then ackerman(m-1, 1)
  else ackerman(m-1, ackerman(m, n-1))
  end
end
ackerman(3, 3)
```

produces:

```
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 75.14     2.75     2.75     2432     1.13    46.92  Object#ackerman
 13.39     3.24     0.49     3676     0.13     0.13  Fixnum#==
  7.65     3.52     0.28     2431     0.12     0.12  Fixnum#-
  3.83     3.66     0.14     1188     0.12     0.12  Fixnum#+
  0.55     3.68     0.02        1    20.00    20.00  Profiler__.start_profile
  0.00     3.68     0.00        1     0.00     0.00  Kernel.puts
  0.00     3.68     0.00        1     0.00     0.00  Module#method_added
  0.00     3.68     0.00        2     0.00     0.00  IO#write
  0.00     3.68     0.00       57     0.00     0.00  Fixnum#>
  0.00     3.68     0.00        1     0.00  3660.00  #toplevel
```