第八章　外部資料輸出入處理

OOP with Ruby

本章內容

- Ruby 的輸出入設計
- 什麼是檔案？
- 開啟與關閉檔案
- 讀寫檔案
- 資料狀態的永續性 (Marshalling)
- 物件狀態的永續性 (Object persistence)

Ruby 的輸出入設計

- Ruby provides what at first sight looks like two separate sets of I/O routines
  - The first is the simple interface:

```
print "Enter your name: "
name = gets
```

  - The second way, which gives you a lot more control, is to use IO objects

IO 物件 (The IO Object)

- Ruby defines a single base class, *IO*, to handle input and output
- This base class is subclassed by classes such as *File* and *BasicSocket* to provide more specialized behavior
  - File is for file-based operation
  - BasicSocket is for network socket-based operation
- An IO object is a bidirectional channel between a Ruby program and some external resource

什麼是檔案？

- Files that are designed to be read by human beings, and that can be read or written with an editor are called text files
  - An advantage of text files is that the are usually the same on all computers, so that they can move from one computer to another
- Files that are designed to be read by programs and that consist of a sequence of binary digits are called binary files
  - Binary files are designed to be read on the same type of computer and with the same programming language as the computer that created the file
  - An advantage of binary files is that they are more efficient to process than text files

開啟與關閉檔案

- Create a new file using File.new

```
file = File.new("testfile", "r")
# ... process the file
file.close
```

  - Use “r” for read, “w” for write, “rw” for both read-write

- Open a file using File.open

```
File.open("testfile", "r") do |file|
  # ... process the file
end
```

  - When the block exits, the file is automatically closed

- Close the file, of course, File.close
  - File should be closed to ensure that all buffered data is written and that all related resources are freed

讀取檔案

- Explicitly open a file and read from it

```
File.open("testfile") do |file|
  while line = file.gets
    puts line
  end
end
```

  - Example output

```
This is line one
This is line two
This is line three
And so on...
```

讀取檔案 (2)

- Use various iterators to process input data
  - *IO#each_byte* invokes a block with the next 8-bit byte from the IO object

```
File.open("testfile") do |file|
  file.each_byte {|ch| putc ch; print "." }
end
```

  - Example output

```
T.h.i.s. .i.s. .l.i.n.e. .o.n.e.
.T.h.i.s. .i.s. .l.i.n.e. .t.w.o.
.T.h.i.s. .i.s. .l.i.n.e. .t.h.r.e.e.
.A.n.d. .s.o. .o.n.......
.
```

讀取檔案 (3)

- Print each line in a file
  - *IO#each_line* calls the block with each line from the file
  - The following example make the original newlines visible using String#dump

```
File.open("testfile") do |file|
  file.each_line {|line| puts "Got #{line.dump}" }
end
```

  - The example output

```
Got "This is line one\n"
Got "This is line two\n"
Got "This is line three\n"
Got "And so on...\n"
```

寫入 / 更新檔案

- Use puts and print to do it!
  - With a couple of exceptions, every object you pass to puts and print is converted to a string by calling that object' to_s method

```
logfile = File.open("captains_log", "a")

# Add a line at the end, then close.

logfile.puts "Stardate 47824.1: Our show has been canceled."

logfile.close
```

更多檔案存取模式整理

- Like C-style file operation modes

```
f1 = File.new("file1", "r+")
# Read/write, starting at beginning of file.

f2 = File.new("file2", "w+")
# Read/write; truncate existing file or create a new one.
f3 = File.new("file3", "a+")
# Read/write; start at end of existing file or create a new one.
```

二進位檔案處理

```
# Input file contains a single line: Line 1.
file = File.open("data")
line = file.readline            # "Line 1.\n"
puts "#{ line.size}  characters."  # 8 characters
file.close

file = File.open("data","rb")
line = file.readline            # "Line 1.\r\n"
puts "#{ line.size}  characters."  # 9 characters
file.close
```

二進位檔案處理 (2)

- The *binmode* method can switch a stream to binary mode
  - Once switched, it cannot be switched back

```
file = File.open("data")
file.binmode
line = file.readline            # "Line 1.\r\n"
puts "#{ line.size}  characters."  # 9 characters
file.close
```

二進位檔案處理 (3)

- For low-level input/output operations, you can use the sysread and syswrite methods
  - sysread method takes a number of bytes as a parameter
    - sysread raises EOFError at end-of-file
  - syswrite method takes a string and returns the actual number of bytes written
    - Sysread and syswrite raise SystemCallError when an error occurs

```
input = File.new("infile")

output = File.new("outfile")

instr = input.sysread(10);

ytes = output.syswrite("This is a test.")
```

檔案的時間戳記 (timestamp)

- The three timestamps that Ruby understands are the modification time, the access time, and the change time
  - Modification time: the last time the file contents were changed
  - Access time: the last time the file was read
  - Change time: the last time the file's directory information was changed

```
# use class methods

t1 = File.mtime("somefile") # Thu Jan 04 09:03:10 GMT-6:00 2001

t2 = File.atime("somefile") # Tue Jan 09 10:03:34 GMT-6:00 2001

t3 = File.ctime("somefile") # Sun Nov 26 23:48:32 GMT-6:00 2000
```

檔案的時間戳記 (2)

- The instance method can also be used

```
myfile = File.new("somefile")
t1 = myfile.mtime
t2 = myfile.atime
t3 = myfile.ctime
```

變更檔案時間

- File access and modification times may be changed using the *utime* method
  - The times may be given either as Time objects or as a number of seconds since the epoch

```
today = Time.now

yesterday = today - 86400

File.utime(today, today, "alpha")

File.utime(today, yesterday, "beta", "gamma")
```

資料狀態的永續性 (Marshalling)

- Java features the ability to serialize objects, letting you store their states somewhere and reconstitute them when needed
- Ruby calls this kind of serialization *marshaling*
  - Saving an object and some or all of its components is done using the method *Marshal.dump*
  - Typically, you will dump an entire object tree starting with some given object. Later, you can reconstitute the object using *Marshal.load*

物件狀態的永續性 (Object Persistence)

```
class Note
  def to_s
    value.to_s
  end
end
```

```
class Chord
  def initialize(arr)
    @arr = arr
  end
  def play
    @arr.join('-')
  end
end
```

```
c = Chord.new( [ Note.new("G"),Note.new("Bb"),Note.new("Db"),Note.new("E") ] )
File.open("posterity", "w+") do |f|
  Marshal.dump(c, f)
end
```

物件狀態的永續性 (2)

```
File.open("posterity") do |f|
   chord = Marshal.load(f)
end
chord.play          # "G-Bb-Db-E"
```

自訂永續儲存格式

```
class Special
  def initialize(valuable, volatile, precious)
    @valuable = valuable
    @volatile = volatile
    @precious = precious
  end
  def marshal_dump
    [ @valuable, @precious ]
  end
  def marshal_load(variables)
    @valuable = variables[0]
    @precious = variables[1]
    @volatile = "unknown"
  end
  def to_s
    "#@valuable #@volatile #@precious"
  end
end
obj = Special.new("Hello", "there", "World")
puts "Before: obj = #{obj}"
data = Marshal.dump(obj)
obj = Marshal.load(data)
puts "After: obj = #{obj}"
```

```
Before: obj = Hello there World
After: obj = Hello unknown World
```

本章回顧