Ruby 迭代器 Ruby File 类和方法

Ruby 文件的输入与输出

Ruby 提供了一整套 I/O 相关的方法，在内核（Kernel）模块中实现。所有的 I/O 方法派生自 IO 类。

类 *IO* 提供了所有基础的方法，比如 *read*、 *write*、 *gets*、 *puts*、 *readline*、 *getc* 和 *printf*。

本章节将讲解所有 Ruby 中可用的基础的 I/O 函数。如需了解更多的函数，请查看 Ruby 的 *IO* 类。

puts 语句

在前面的章节中，您赋值给变量，然后使用 *puts* 语句打印输出。

puts 语句指示程序显示存储在变量中的值。这将在每行末尾添加一个新行。

实例

```
#!/usr/bin/ruby
val1 = "This is variable one"
val2 = "This is variable two"
puts val1
puts val2
```

以上实例运行输出结果为：

```
This is variable one
This is variable two
```

gets 语句

gets 语句可用于获取来自名为 STDIN 的标准屏幕的用户输入。

实例

下面的代码演示了如何使用 gets 语句。该代码将提示用户输入一个值，该值将被存储在变量 val 中，最后会被打印在 STDOUT 上。

实例

```
#!/usr/bin/ruby
puts "Enter a value :"
val = gets
puts val
```

以上实例运行输出结果为：

```
Enter a value :
This is entered value
This is entered value
```

putc 语句

与 *puts* 语句不同，*puts* 语句输出整个字符串到屏幕上，而 *putc* 语句可用于依次输出一个字符。

实例

下面代码的输出只是字符 H：

实例

```
#!/usr/bin/ruby
str="Hello Ruby!"
putc str
```

以上实例运行输出结果为：

```
H
```

print 语句

print 语句与 *puts* 语句类似。唯一的不同在于 *puts* 语句在输出内容后会跳到下一行，而使用 *print* 语句时，光标定位在同一行。

实例

```
#!/usr/bin/ruby
print "Hello World"
print "Good Morning"
```

以上实例运行输出结果为：

```
Hello WorldGood Morning
```

打开和关闭文件

截至现在，您已经读取并写入标准输入和输出。现在，我们将看看如何操作实际的数据文件。

File.new 方法

您可以使用 *File.new* 方法创建一个 *File* 对象用于读取、写入或者读写，读写权限取决于 mode 参数。最后，您可以使用 *File.close* 方法来关闭该文件。

语法

```
aFile = File.new("filename", "mode")
# ... 处理文件
aFile.close
```

File.open 方法

您可以使用 *File.open* 方法创建一个新的 file 对象，并把该 file 对象赋值给文件。但是，*File.open* 和 *File.new* 方法之间有一点不同。不同点是 *File.open* 方法可与块关联，而 *File.new* 方法不能。

```
File.open("filename", "mode") do |aFile|
# ... process the file
end
```

下表列出了打开文件的不同模式：

模式	描述
r	只读模式。文件指针被放置在文件的开头。这是默认模式。
r+	读写模式。文件指针被放置在文件的开头。
w	只写模式。如果文件存在，则重写文件。如果文件不存在，则创建一个新文件用于写入。
w+	读写模式。如果文件存在，则重写已存在的文件。如果文件不存在，则创建一个新文件用于读写。
a	只写模式。如果文件存在，则文件指针被放置在文件的末尾。也就是说，文件是追加模式。如果文件不存在，则创建一个新文件用于写入。
a+	读写模式。如果文件存在，则文件指针被放置在文件的末尾。也就是说，文件是追加模式。如果文件不存在，则创建一个新文件用于读写。

读取和写入文件

用于简单 I/O 的方法也可用于所有 file 对象。所以，gets 从标准输入读取一行，*aFile.gets* 从文件对象 aFile 读取一行。

但是，I/O 对象提供了访问方法的附加设置，为我们提供了便利。

sysread 方法

您可以使用方法 *sysread* 来读取文件的内容。当使用方法 sysread 时，您可以使用任意一种模式打开文件。例如：

下面是输入文本文件：

```
This is a simple text file for testing purpose.
```

现在让我们尝试读取这个文件：

实例

```
#!/usr/bin/ruby
aFile = File.new("input.txt", "r")
if aFile
content = aFile.sysread(20)
puts content
else
puts "Unable to open file!"
end
```

该语句将输入文件的头 20 个字符。文件指针将被放置在文件中第 21 个字符的位置。

syswrite 方法

您可以使用方法 *syswrite* 来向文件写入内容。当使用方法 syswrite 时，您需要以写入模式打开文件。例如：

实例

```
#!/usr/bin/ruby
aFile = File.new("input.txt", "r+")
if aFile
aFile.syswrite("ABCDEF")
else
puts "Unable to open file!"
end
```

该语句将写入 "ABCDEF" 到文件中。

each_byte 方法

该方法属于类 *File*。方法 *each_byte* 是个可以迭代字符串中每个字符。请看下面的代码实例：

实例

```
#!/usr/bin/ruby
aFile = File.new("input.txt", "r+")
if aFile
aFile.syswrite("ABCDEF")
aFile.rewind
aFile.each_byte {|ch| putc ch; putc ?. }
else
puts "Unable to open file!"
end
```

字符一个接着一个被传到变量 ch，然后显示在屏幕上，如下所示：

```
A.B.C.D.E.F.s. .a. .s.i.m.p.l.e. .t.e.x.t. .f.i.l.e. .f.o.r. .t.e.s.t.i.n.g. .p.u.r.p.o.s.e...
```

IO.readlines 方法

类 *File* 是类 IO 的一个子类。类 IO 也有一些用于操作文件的方法。

IO.readlines 是 IO 类中的一个方法。该方法逐行返回文件的内容。下面的代码显示了方法 *IO.readlines* 的使用：

实例

```
#!/usr/bin/ruby
arr = IO.readlines("input.txt")
puts arr[0]
puts arr[1]
```

在这段代码中，变量 arr 是一个数组。文件 *input.txt* 的每一行将是数组 arr 中的一个元素。因此，arr[0] 将包含第一行，而 arr [1] 将包含文件的第二行。

IO.foreach 方法

该方法也逐行返回输出。方法 *foreach* 与方法 *readlines* 之间不同的是，方法 *foreach* 与块相关联。但是，不像方法 *readlines*，方法 *foreach* 不是返回一个数组。例如：

实例

```
#!/usr/bin/ruby
IO.foreach("input.txt"){|block| puts block}
```

这段代码将把文件 *test* 的内容逐行传给变量 block，然后输出将显示在屏幕上。

重命名和删除文件

您可以通过 *rename* 和 *delete* 方法重命名和删除文件。

下面的实例重命名一个已存在文件 *test1.txt*：

实例

```
#!/usr/bin/ruby
# 重命名文件 test1.txt 为 test2.txt
File.rename( "test1.txt", "test2.txt" )
```

下面的实例删除一个已存在文件 *test2.txt*：

实例

```
#!/usr/bin/ruby
# 删除文件 test2.txt
File.delete("text2.txt")
```

文件模式与所有权

使用带有掩码的 *chmod* 方法来改变文件的模式或权限/访问列表：

下面的实例改变一个已存在文件 *test.txt* 的模式为一个掩码值：

实例

```
#!/usr/bin/ruby
file = File.new( "test.txt", "w" )
file.chmod( 0755 )
```

下表列出了 *chmod* 方法中可使用的不同的掩码：

掩码	描述
0700	rwx 掩码，针对所有者
0400	r ，针对所有者
0200	w ，针对所有者
0100	x ，针对所有者

0070	rwx 掩码，针对所属组
0040	r ，针对所属组
0020	w ，针对所属组
0010	x ，针对所属组
0007	rwx 掩码，针对其他人
0004	r ，针对其他人
0002	w ，针对其他人
0001	x ，针对其他人
4000	执行时设置用户 ID
2000	执行时设置所属组 ID
1000	保存交换文本，甚至在使用后也会保存

文件查询

下面的命令在打开文件前检查文件是否已存在：

实例

```
#!/usr/bin/ruby
File.open("file.rb") if File::exists?( "file.rb" )
```

下面的命令查询文件是否确实是一个文件：

实例

```
#!/usr/bin/ruby
# 返回 true 或false
File.file?( "text.txt" )
```

下面的命令检查给定的文件名是否是一个目录：

实例

```
#!/usr/bin/ruby
# 一个目录
File::directory?( "/usr/local/bin" ) # => true
# 一个文件
File::directory?( "file.rb" ) # => false
```

下面的命令检查文件是否可读、可写、可执行：

实例

```
#!/usr/bin/ruby
File.readable?( "test.txt" ) # => true
File.writable?( "test.txt" ) # => true
File.executable?( "test.txt" ) # => false
```

下面的命令检查文件是否大小为零：

实例

```
#!/usr/bin/ruby
File.zero?( "test.txt" ) # => true
```

下面的命令返回文件的大小：

实例

```
#!/usr/bin/ruby
File.size?( "text.txt" ) # => 1002
```

下面的命令用于检查文件的类型：

实例

```
#!/usr/bin/ruby
File::ftype( "test.txt" ) # => file
```

ftype 方法通过返回下列中的某个值来标识了文件的类型：*file*、 *directory*、 *characterSpecial*、 *blockSpecial*、 *fifo*、 *link*、 *socket* 或 *unknown*。

下面的命令用于检查文件被创建、修改或最后访问的时间：

实例

```
#!/usr/bin/ruby
File::ctime( "test.txt" ) # => Fri May 09 10:06:37 -0700 2008
File::mtime( "text.txt" ) # => Fri May 09 10:44:44 -0700 2008
File::atime( "text.txt" ) # => Fri May 09 10:45:01 -0700 2008
```

Ruby 中的目录

所有的文件都是包含在目录中，Ruby 提供了处理文件和目录的方式。*File* 类用于处理文件，*Dir* 类用于处理目录。

浏览目录

为了在 Ruby 程序中改变目录，请使用 *Dir.chdir*。下面的实例改变当前目录为 */usr/bin*。

```
Dir.chdir("/usr/bin")
```

您可以通过 *Dir.pwd* 查看当前目录：

```
puts Dir.pwd # 返回当前目录，类似 /usr/bin
```

您可以使用 *Dir.entries* 获取指定目录内的文件和目录列表：

```
puts Dir.entries("/usr/bin").join(' ')
```

Dir.entries 返回一个数组，包含指定目录内的所有项。*Dir.foreach* 提供了相同的功能：

```
Dir.foreach("/usr/bin") do |entry|
puts entry
end
```

获取目录列表的一个更简洁的方式是通过使用 Dir 的类数组的方法：

```
Dir["/usr/bin/*"]
```

创建目录

Dir.mkdir 可用于创建目录：

```
Dir.mkdir("mynewdir")
```

您也可以通过 mkdir 在新目录（不是已存在的目录）上设置权限：

注意：掩码 755 设置所有者（owner）、所属组（group）、每个人（world [anyone]）的权限为 rwxr-xr-x，其中 r = read 读取，w = write 写入，x = execute 执行。

```
Dir.mkdir( "mynewdir", 755 )
```

删除目录

Dir.delete 可用于删除目录。*Dir.unlink* 和 *Dir.rmdir* 执行同样的功能，为我们提供了便利。

```
Dir.delete("testdir")
```

创建文件 & 临时目录

临时文件是那些在程序执行过程中被简单地创建，但不会永久性存储的信息。

Dir.tmpdir 提供了当前系统上临时目录的路径，但是该方法默认情况下是不可用的。为了让 *Dir.tmpdir* 可用，使用必需的 'tmpdir' 是必要的。

您可以把 *Dir.tmpdir* 和 *File.join* 一起使用，来创建一个独立于平台的临时文件：

```
require 'tmpdir'
tempfilename = File.join(Dir.tmpdir, "tingtong")
tempfile = File.new(tempfilename, "w")
tempfile.puts "This is a temporary file"
tempfile.close
File.delete(tempfilename)
```

这段代码创建了一个临时文件，并向其中写入数据，然后删除文件。Ruby 的标准库也包含了一个名为 *Tempfile* 的库，该库可用于创建临时文件：

```
require 'tempfile'
f = Tempfile.new('tingtong')
f.puts "Hello"
```

```
puts f.path
f.close
```

内建函数

下面提供了 Ruby 中处理文件和目录的内建函数的完整列表：

- File 类和方法。
- Dir 类和方法。

← Ruby 迭代器 Ruby File 类和方法 →