Ввод и вывод в Ruby

последнее изменение 18 октября 2023 г.

В этой части руководства по Ruby мы поговорим об операциях ввода-вывода в Ruby. Входные данные - это любые данные, которые считываются программой либо с клавиатуры, либо из файла, либо из других программ. Выходные данные - это данные, создаваемые программой. Выходные данные могут выводиться на экран, в файл или в другую программу.

Ввод и вывод - это большая тема. Мы приводим несколько примеров, чтобы дать вам общее представление о предмете. Несколько классов в Ruby имеют методы для выполнения операций ввода-вывода. Например, Kernel, IO, Dir или File.

stones.txt

```
Гранат
Топаз
Опал
Аметист
Рубин
Яшма
Пирит
Малахит
Кварц
```

Some of the examples use this file.

Ruby writing to console

В Ruby есть несколько методов для вывода выходных данных на консоль. Эти методы являются частью Kernel модуля. Методы Kernel доступны для всех объектов в Ruby.

printing.rb

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

```
выведите "Apple"
выведите "Apple \ n"

вставляет "Orange"
вставляет "Оранжевый"
```

Методы print и puts выдают текстовый вывод на консоль. Разница между ними заключается в том, что последний добавляет новый символ строки.

```
вывести "Apple"
вывести "Apple \ n"
```

Метод print выводит на терминал две последовательные строки "Apple". Если мы хотим создать новую строку, мы должны явно включить символ перевода строки. Символом перевода строки является \n. За кулисами print метод фактически вызывает to_s метод печатаемого объекта.

```
ставит "Оранжевый"
ставит "Оранжевый"
```

putsМетод выводит на консоль две строки. Каждая строка находится в отдельной строке. Метод автоматически включает символ перевода строки.

```
$ ./printing.rb
Apple Яблоко
Оранжевый
Оранжевый
```

Это выходные данные файла сценария printing.rb.

Согласно документации Ruby, print метод является эквивалентом $stdout.print. $stdout это глобальная переменная, которая содержит стандартный поток вывода.

printing2.rb

```
#!/usr/bin/ruby

$stdout.выводит "язык Ruby\n"
$stdout.выводит "язык Python"
```

Мы печатаем две строки, используя $stdout переменную.

В Ruby есть еще три метода вывода на печать.

printing3.rb

```
#!/usr/bin/ruby

p "Lemon"
p "Lemon"

printf "Есть %d яблок \ n", 3

putc 'K'
putc 0xA
```

В примере мы представляем методы p, printf и putc.

```
p "Lemon"
```

The p calls the inspect method upon the object being printed. The method is useful for debugging.

```
printf "There are %d apples\n", 3
```

printfМетод хорошо известен из языка программирования C. Он позволяет форматировать строки.

```
putc 'K'
putc 0xA
```

putcМетод выводит на консоль один символ. Во второй строке выводится новая строка. OxA это шестнадцатеричный код для символа новой строки.

```
$ ./printing3.rb
"Lemon"
"Лимон"
Там 3 яблока
K
```

Печать данных на консоль с использованием методов ядра - это короткий путь: удобный способ печати данных. В следующем примере показан более формальный способ печати данных на терминале.

```
ios = IO.new STDOUT.fileno
ios.написать "ZetCode \ n"
ios.закрыть
```

В приведенном примере мы открываем стандартный поток вывода и записываем в него строку.

```
ios = IO.new STDOUT.fileno
```

newМетод возвращает поток, в который мы можем записывать данные. Метод принимает числовой файловый дескриптор. STDOUT.fileno выдает нам файловый дескриптор для стандартного выходного потока. Мы также могли бы просто написать 2.

```
ios.напишите "ZetCode\n"
```

Мы записываем строку в открытый поток.

```
ios.закрыть
```

Входной поток закрыт.

В системах Unix стандартный вывод терминала подключен к специальному файлу с именем /dev/tty. Открывая его и записывая в него, мы выполняем запись в консоль.

dev_tty.rb

```
#!/usr/bin/ruby

fd = IO.sysopen "/dev/tty", "w"
ios = IO.new(fd, "w")
ios.помещает "ZetCode"
ios.закрыть
```

Небольшой пример, в котором мы записываем данные в /dev/tty файл. Это работает только в Unix.

```
fd = IO.sysopen "/dev/tty", "w"
```

sysopenМетод открывает указанный путь, возвращая номер базового файлового дескриптора.

```
ios = IO.new(fd, "w")
```

The file descriptor number is used to open a stream.

```
ios.puts "ZetCode"
ios.close
```

Мы записываем строку в поток и закрываем его.

Ruby считывает входные данные с консоли

В этом разделе мы создадим несколько примеров кода, которые касаются чтения с консоли.

$stdin - это глобальная переменная, которая содержит поток для стандартного ввода. Ее можно использовать для чтения входных данных с консоли.

чтение.rb

```
#!/usr/bin/ruby

inp = $stdin.read
помещает inp
```

В приведенном выше коде мы используем read метод для считывания входных данных с консоли.

```
inp = $stdin.read
```

readМетод считывает данные из стандартного ввода до тех пор, пока они не дойдут до конца файла. EOF создается нажатием Ctrl+D в Unix и Ctrl+Z в Windows.

```
$ ./reading.rb
Язык Ruby
Язык Ruby
```

Когда мы запускаем программу без параметра, скрипт считывает данные от пользователя. Он считывает до тех пор, пока мы не нажмем Ctrl+D или Ctrl+Z.

```
$ echo "ZetCode" | ./reading.rb
ZetCode

$ ./input.rb < stones.txt
Гранат
Топаз
Опал
Аметист
Рубин
Яшма
Пирит
Малахит
Кварц
```

Скрипт может считывать данные из другой программы или файла, если мы выполним некоторые перенаправления.

Распространенным способом считывания данных с консоли является использование gets метода.

read_input.rb

```
#!/usr/bin/ruby

выведите "Введите свое имя":
name = возвращает
помещает "Hello #{name}"
```

Мы используем gets метод для чтения строки от пользователя.

```
name = получает
```

getsМетод считывает строку из стандартного ввода. Данные присваиваются переменной name.

```
вводит "Hello #{name}"
```

Прочитанные нами данные выводятся на консоль. Мы используем интерполяцию для включения переменной в строку.

```
$ ./read_input.rb
Введите свое имя: Ян
Привет, Ян
```

Это пример вывода.

В следующих двух сценариях мы обсудим chomp метод. Это строковый метод, который удаляет пробелы в конце строки. Он полезен при выполнении операций ввода. Название метода и его использование взяты из языка Perl.

no_chomp.rb

```
#!/usr/bin/ruby

выведите "Введите строку":
inp = получает

помещает "Строка содержит # символов{inp.size}"
```

Мы считываем строку от пользователя и вычисляем длину входной строки.

```
$ ./no_chomp.rb
Введите строку: Ruby
Строка состоит из 5 символов
```

В сообщении говорится, что строка состоит из 5 символов. Это потому, что в ней также учитывается перевод строки.

Чтобы получить правильный ответ, нам нужно удалить символ новой строки. Это задание для chomp метода.

chomp.rb

```
#!/usr/bin/ruby

выведите "Введите строку":
inp = получает.chomp

вводит "Строка содержит #символов{inp.size}"
```

На этот раз мы используем метод "мы вырезаем символ новой строки" с помощью chomp метода.

```
$ ./chomp.rb
Введите строку: Ruby
Строка состоит из 4 символов
```

Строка Ruby действительно содержит 4 символа.

Файлы Ruby

Из официальной документации Ruby мы узнаем, что IO класс является основой для всего ввода-вывода в Ruby. File Класс является единственным подклассом IO класса. Эти два класса тесно связаны.

simple_write.rb

```
#!/usr/bin/ruby

f = File.open('output.txt', 'w')
f.помещает "Руководство по Ruby"
f.закрыть
```

В первом примере мы открываем файл и записываем в него некоторые данные.

```
f = File.open('output.txt', 'w')
```

Мы открываем файл 'output.txt' в режиме записи. open Метод возвращает поток ввода-вывода.

```
f.помещает "Руководство по Ruby"
```

Мы использовали приведенный выше открытый поток для записи некоторых данных. putsМетод также можно использовать для записи данных в файл.

```
f.закрыть
```

В конце поток закрывается.

```
$ ./simple_write.rb
$ cat output.txt
Руководство по Ruby
```

Мы выполняем скрипт и показываем содержимое output.txt файла.

У нас есть аналогичный пример, который показывает дополнительные методы в действии.

simple_write2.rb

```
#!/usr/bin/ruby

File.open('langs.txt', 'w') делаем |f|

 f.помещаем "Ruby"
 f.пишем "Java\n"
 f << "Python \n"

завершение
```

Если после open метода есть блок, Ruby передает открытый поток в этот блок. В конце блока файл автоматически закрывается.

```
f.помещает "Ruby"
f.пишет "Java\n"
f << "Python\n"
```

Мы используем три различных метода для записи в файл.

```
$ ./simple_write2.rb
$ cat langs.txt
Ruby
Java
Python
```

Мы выполняем скрипт и проверяем содержимое langs файла.

Во втором примере мы покажем несколько методов File класса.

testfile.rb

```
#!/usr/bin/ruby

помещает File.exist? 'временный файл'

f = File.new 'tempfile', 'w'
помещает File.mtime 'tempfile'
помещает f.size

Файл.переименовать 'tempfile', 'tempfile2'

f.закрыть
```

В примере создается новый файл с именем tempfile и вызываются некоторые методы.

```
помещает файл.exist? 'временный файл'
```

exist?Метод проверяет, существует ли файл с заданным именем. Строка возвращает false, поскольку мы еще не создали файл.

```
f = File.new 'временный файл', 'w'
```

Файл создан.

```
помещает File.mtime во 'временный файл'
```

mtimeМетод дает нам время последнего изменения файла.

```
помещает f.size
```

Определяем размер файла. Метод возвращает 0, поскольку мы не записывали данные в файл.

```
Файл.переименуйте 'tempfile', 'tempfile2'
```

Наконец, мы переименовываем файл, используя rename метод.

```
$ ./testfile.rb
false
2020-09-14 15:54:13 +0200
0
```

Это пример вывода.

Далее мы считываем данные из файла на диске.

read_file.rb

```
#!/usr/bin/ruby

f = File.open("stones.txt")

в то время как line = f.получает do
 помещает строку
end

f.закрыть
```

В примере открывается файл с именем stones.txt и построчно выводится его содержимое на терминал.

```
f = File.open("stones.txt")
```

Мы открываем stones файл. Режим по умолчанию - режим чтения. stones Файл содержит девять названий ценных камней, каждое в отдельной строке.

```
while line = f.возвращает do
 помещает строку
end
```

getsМетод считывает строку из потока ввода-вывода. Блок while заканчивается, когда мы доходим до конца файла.

```
$ ./read_file.rb
Гранат
Топаз
Опал
Аметист
Рубин
Яшма
Пирит
Малахит
Кварц
```

Следующий пример считывает данные из файла.

all_lines.rb

```
#!/usr/Бен/Рубин

alllines имени ='.РБ'
```

```
Файл.readlines(имени).каждый сделать |строки|
 линейку приставляют
конец
```

Этот скрипт показывает другой способ чтения содержимого файла. Пример кода выводит свой собственный код на терминал.

```
Файл.readlines(имени).каждый сделать |строки|
 линейку приставляют
конец
```

readlinesСчитывает все строки из указанного файла и возвращает их в виде массива. Проходим по массиву с помощью each метода и выводим строки на терминал.

```
$ ./all_lines.rb
#!/usr/bin/ruby

fname = 'alllines.rb'

File.readlines(fname).каждое действие | строка|
 линейку приставляют
конец
```

Каталоги Ruby

В этом разделе мы работаем с каталогами. У нас есть Dir класс для работы с каталогами в Ruby.

dirs.rb

```
#!/usr/bin /ruby

Dir.mkdir "tmp"
помещает Dir.exist? "tmp"

помещает Dir.pwd
Dir.chdir "tmp"
помещает Dir.pwd

Dir.chdir '..'
помещает Dir.pwd
Dir.rmdir "tmp"
помещает Dir.exist? "tmp"
```

В скрипте мы используем четыре метода Dir класса.

```
Dir.mkdir "tmp"
```

mkdirМетод создает новый вызываемый каталог tmp.

```
помещает Dir.exist? "tmp"
```

С помощью exist? метода мы проверяем, существует ли в файловой системе каталог с заданным именем.

```
помещает Dir.pwd
```

pwdМетод выводит текущий рабочий каталог. Это каталог, из которого мы запустили скрипт.

```
Dir.chdir '..'
```

chdirМетод переходит в другой каталог. .. Каталог является родительским каталогом текущего рабочего каталога.

```
Dir.rmdir "tmp"
помещает Dir.exist? "tmp"
```

Наконец, мы удаляем каталог с помощью rmdir метода. На этот раз exist? метод возвращает false.

Во втором примере мы извлекаем все записи каталога, включая его файлы и подкаталоги.

all_files.rb

```
#!/usr/bin/ruby

fls = Dir.entries '.'
помещает fls.inspect
```

entriesМетод возвращает все записи заданного каталога.

```
fls = Dir.entries '.'
помещает fls.inspect
```

Мы получаем массив файлов и каталогов текущего каталога. В данном контексте . символ обозначает текущий рабочий каталог. inspect Метод дает нам более читаемое представление массива.

Третий пример работает с домашним каталогом. Каждому пользователю компьютера присвоен уникальный каталог. Он называется домашним каталогом. Это место, где он может размещать свои файлы и создавать свою собственную иерархию каталогов.

home_dir.rb

```
#!/usr/bin/ruby

помещает Dir.home
помещает Dir.home 'root'
```

Скрипт печатает два домашних каталога.

```
помещает Dir.home
```

Если мы не указываем имя пользователя, то возвращается домашний каталог текущего пользователя. Текущий пользователь является владельцем файла скрипта. Тот, кто запустил скрипт.

```
помещает Dir.home в 'root'
```

Здесь мы печатаем домашний каталог конкретного пользователя: в нашем случае суперпользователя.

```
$ ./homedir.rb
/главная страница/janbodnar
/root
```

Это пример вывода.

Ruby, выполняющий внешние программы

В Ruby есть несколько способов выполнения внешних программ. Мы рассмотрим некоторые из них. В наших примерах мы используем хорошо известные команды Linux. Читатели с Windows или Mac могут использовать команды, специфичные для их систем.

system.rb

```
#!/usr/bin/ruby

данные = системный 'ls'
помещает данные
```

Мы вызываем ls команду, которая выводит список содержимого каталога.

```
данные = system 'ls'
```

systemКоманда выполняет внешнюю программу в подоболочке. Метод принадлежит Kernel модулю Ruby.

Мы покажем два других способа запуска внешних программ в Ruby.

system2.rb

```
#!/usr/bin/ruby

out = `pwd`
выводит

выводит = %x [время безотказной работы]
выводит

выводит = %x [ls | grep 'readline']
выводит
```

Для запуска внешних программ мы можем использовать обратные метки `` или %x[] символы.

```
out = `pwd`
```

Здесь мы выполняем pwd команду с помощью кнопок возврата. Команда возвращает текущий рабочий каталог.

```
out = %x [время безотказной работы]
```

Здесь мы получаем вывод uptime команды, которая сообщает, как долго работает система.

```
out = %x[ls | grep 'строка чтения']
```

Мы также можем использовать комбинацию команд.

Мы можем выполнить команду с помощью open метода. Метод принадлежит Kernel модулю. Он создает объект ввода-вывода, подключенный к данному потоку, файлу или подпроцессу. Если мы хотим подключиться к подпроцессу, мы начинаем путь к нему open с символа канала |.

system3.rb

```
#!/usr/bin/ruby

f = открыть ("|ls -l |head -3")
out = f.прочитать
выводит
f.закрыть

выводит $?.успех?
```

В приведенном примере мы выводим результат выполнения ls -l | head -3 команд. Комбинация этих двух команд возвращает первые три строки ls -l команды. Мы также проверяем состояние дочернего подпроцесса.

```
f = открыть("|ls -l |head -3")
```

Мы подключаемся к подпроцессу, созданному этими двумя командами.

```
out = f.читать
выводит
```

Мы считываем и печатаем данные из подпроцесса.

```
f.закрыть
```

Мы закрываем обработчик файлов.

```
ставит $?.успех?
```

$? это специальная переменная Ruby, которая устанавливается в статус последнего выполненного дочернего процесса. success? Метод возвращает true, если дочерний процесс завершился нормально.

Перенаправление стандартного вывода Ruby

Ruby имеет предопределенные глобальные переменные для стандартного ввода, стандартного вывода и стандартного вывода ошибок. $stdout - это имя переменной для стандартного вывода.

перенаправление.rb

```
#!/usr/bin/ruby

$stdout = File.открыть "output.log", "a"
```

```
помещает "Ruby"
помещает "Java"

$stdout.close
$ stdout = СТАНДАРТНЫЙ вывод

ставит "Python"
```

В приведенном выше примере мы перенаправляем стандартный вывод в output.файл журнала.

```
$stdout = Файл.открыть "output.log", "a"
```

Эта строка создает новый стандартный вывод. Теперь стандартный вывод будет поступать в ouput.log файл. Файл открывается в режиме добавления. Если файл еще не существует, он создается. В противном случае он открывается и данные записываются в конец файла.

```
вводит "Ruby"
вводит "Java"
```

Мы печатаем две строки. Строки не будут отображаться в терминале, как обычно. Скорее, они будут добавлены в output.log файл.

```
$stdout.close
```

Обработчик закрыт.

```
$stdout = СТАНДАРТНЫЙ вывод

ставит "Python"
```

Мы используем предопределенную стандартную константу STDOUT для воссоздания обычного стандартного вывода. Строка "Python" выводится на консоль.

В этой части руководства по Ruby мы работали с операциями ввода-вывода в Ruby.

Содержание Предыдущая страница

Главная Twitter Github Подписка Конфиденциальность О нас