Ворожцов Артём Викторович
Кандидат физико-математических наук, преподаватель кафедры информатики Московского физико-технического института (МФТИ), тренер сборной команды МФТИ по программированию.

Как с помощью рубина получить «волшебные кубики»?

После знакомства с языком Ruby начинаешь верить в чудодейственную силу камней. В старину рубины слыли как «камни жизни и любви». Считалось, что владение таким камнем придаёт хозяину больше власти, смелости и достоинства. Рубинами надеялись защититься от чёрта и от чумы. Принадлежащим рубину знаком зодиака является Козерог, а месяцем – июль.

Программисты на языке Ruby действительно очень смелы и полны достоинства. Они гордятся своим языком и не устают делиться своими эмоциями относительно того, насколько большие возможности предоставляет им этот язык при разработке и насколько удобно и легко осуществлять контроль над развитием программных систем, написанных на Ruby.

Что касается защиты от чёрта и чумы, то здесь также можно проводить аналогии. Современная чума и чёрт программистов – это постоянный поток ошибок и рефакторинг (глобальная переделка кода для его развития), переписывание одного и того же функционала по несколько раз. Рубисты активно продвигают концепцию разработки, основанной на тестировании (Test Driven Development), когда, перед тем как разрабатывать код, программист создаёт абстрактный интерфейс частей системы и систему тестов, которые в течение разработки регулярно запускаются. Результаты тестов на каждый момент отображают процент проделанной работы.

Многие программисты Ruby признают, что и этот язык не идеален. Впрочем, это касается любого языка программирования. Любой язык программирования – это материя, которая как всякая материя, несовершенна. Строитель может работать с песком, глиной, бетоном, гипсом, кирпичом. Каждый из этих материалов имеет свои особенности и не является универсальным строительным материалом. Также и в мире языков программирования. Есть различные семейства, виды и подвиды языков, каждый из них занимает свою «экологическую нишу».

Экологическая ниша языка Ruby – это программирование с максимально высоким уровнем абстракции, нацеленное на модульность и reuse (повторное использование). В каждом приложении на Ruby чувствуется стремление к развитию (стремление заложить максимальный потенциал эволюции системы), простоте и надёжности. Один из лозунгов Ruby следующий: «код должен отражать

образ мышления человека, код должен быть близким к естественному языку». Близость к естественному языку провозглашают многие языки. Но, пожалуй, ближе всего к этому подошёл Ruby (вместе с такими языками как Lua, Python, Haskell и др.)

За модульность и высокий уровень абстракции необходимо платить. Язык Ruby – один из самых медленных интерпретируемых языков. Хотя, в свете выхода следующей версии Ruby 2.0, ситуация может измениться. Официальный интерпретатор языка в этой версии будет целиком заменён на интерпретатор Ruby YARV, который разрабатывался параллельно и нацелен был на повышении производительности языка. По различным оценкам новая версия будет в 3 раза быстрее предыдущей.

В языке Руби широко используется концепция итераторов и генераторов – интересная и полезная штука. В разных языках программирования методам можно в качестве аргументов передавать не только числа, строки и др. объекты, но и методы. Методу, который генерирует какие-то объекты, передаётся в качестве аргумента метод, которому нужно «подсовывать» генерируемые объекты. В Руби методы можно создавать по ходу дела с помощью конструкции

```
do |àðãóìåíò1, àðãóìåíò1,…| … end
```

Эта конструкция называется *блоком*. По сути блок – это обычный метод с аргументами (которых может и не быть), у которого просто нет имени. Такие блоки как бы создаются для временных нужд, нет необходимости давать им какое-то имя. Простейший вариант реализации этой идеи – это итераторы для контейнеров: по элементам массива (класс **Array**), по элементам множества (класс **Set**), по парам (ключ, значение), хранящимся в хэшах (класс **Hash**) и др. Давайте ограничимся классом **Array** и рассмотрим несколько примеров.

```
# ñêîíñòðóèðóåì ìàññèâ ïåðâûõ 100 íàòóðàëüíûõ ÷èñåë
# (1..100) - ýêçåìïëÿð êëàññà Range (ïðîìåæóòîê)
# ñ ïîìîùüþ ìåòîäà to_a ïðåîáðàçóåòñÿ â ìàññèâ à
a = (1..100).to_a

# ïðîáðàçîâàòü ìàññèâ à, âîçâåäÿ êàæäûé ýëåìåíò
# â êâàäðàò; çàòåì îáúåäèíèòü ýëåìåíòû â îäíó ñòðîêó,
# ðàçäåëÿÿ èõ çàïÿòîé
puts a.map {|x| x*x}.join(",")


# âûáðàòü èç ìàññèâà a ýëåìåíòû,
# êîòîðûå èìåþò îñòàòîê 1 ïðè äåëåíèè íà 5
b = a.select {|x| x%5==1}

# êàæäûé ýëåìåíò ìàññèâà b íàïå÷àòàòü
b.each {|x| puts x}

# âû÷èñëèòü ïðîèçâåäåíèå âñåõ ýëåìåíòîâ ìàññèâà b
puts b.inject(1) {|f,x| f*x}
```

Во всех этих примерах используются методы, которые «в качестве аргумента» получают блок. Правильнее говорить, что эти методы имеют *ассоциированный блок*. Блок всегда является последним аргументом и при объявлении метода объявляется как **&block**. Вызов этого блока с аргументами **x**, **y**, ... из тела метода осуществляется командой

block[x,y,…].

Давайте решим задачу «Волшебные игральные кубики» на Ruby (условие на с. 67). Это задача решается методом перебора. По сути, нам нужно перебрать различные сочетания элементов из промежутка (1..*P*) с возможными повторениями, и для получившегося множества сочетаний перебрать всевозможные тройки (уже без повторения). Естественно для этого написать методы для экземпляров класса **Array** – массивов.

```
class Array
  # ïåðåáîðû âñå íàáîðû ñ âîçìîæíûìè ïîâòîðåíèÿìè
  def tuples (size, &block)
      tuples0(size,[],0,block)
  end

  private
  # ïåðåáîðû âñå íàáîðû ýëåìåíòîâ ñ âîçìîæíûìè
  # ïîâòîðåíèÿìè, ïðè óñëîâèè, ÷òî óæå íàáðàí
  # ÷àñòè÷íî íàáîð ary, è â íåãî íóæíî äîáðàòü
  # ýëåìåíòû èç õâîñòà ìàññèâà self[i..].
  def tuples0(size,ary,i,block)
      if ary.size == size
          # ïîëó÷åí î÷åðåäíîé íàáîð
          # îòäàäèì åãî àññîöèèðîâàííîìó áëîêó
          block[ary]
      elsif i < self.size
          # âîçüì¸ì self[i]
          ary.push self[i]
          tuples0(size,ary,i,block)
          ary.pop

          # èëè ïðîïóñòèì self[i]
          tuples0(size,ary,i+1,block)
       end
   end
end

# ñãåíåðèðóåì ñî÷åòàíèÿ 3-õ ýëåìåíòîâ èç 5
# ñ âîçìîæíûìè ïîâòîðåíèÿìè
(1..5).to_a.tuples(3) do |tuple|
  puts tuple.inspect
end
```

```
Âûâîä ïðîãðàììû:
[1, 1, 1]
[1, 1, 2]
[1, 1, 3]
[1, 1, 4]
[1, 1, 5]
[1, 2, 2]
[1, 2, 3]
[1, 2, 4]
[1, 2, 5]
[1, 3, 3]
[1, 3, 4]
[1, 3, 5]
[1, 4, 4]
[1, 4, 5]
[1, 5, 5]
[2, 2, 2]
[2, 2, 3]
[2, 2, 4]
[2, 2, 5]
[2, 3, 3]
[2, 3, 4]
[2, 3, 5]
[2, 4, 4]
[2, 4, 5]
[2, 5, 5]
[3, 3, 3]
[3, 3, 4]
[3, 3, 5]
[3, 4, 4]
[3, 4, 5]
[3, 5, 5]
[4, 4, 4]
[4, 4, 5]
[4, 5, 5]
[5, 5, 5]
```

Предъявленный метод **tuples** у экземпляров класса **Array** генерирует различные сочетания с возможными повторениями из элементов массива. Этот метод вызывается как метод конкретного массива, который внутри определений имеет имя **self** (сам объект). Алгоритм генерации основан на методе рекурсии. Введём в задачу дополнительные параметры – **ary** и **i**:

Подзадача. Пусть у нас уже взято несколько элементов, и они собраны в массиве **ary**. Необходимо доложить в этот набор элементы так, чтобы он имел размер **size**, используя при этом только элементы хвоста массива: **self[i], self[i+1], …**

Эту подзадачу решает функция **tuples0(size,ary,i,block)**, при этом эта подзадача легко сводится сама к себе и её можно решить, используя рекурсию.

Задача. Загрузите и установите Ruby (http://www.ruby-lang.org/en//downloads). Используя редактор SciTe, который поставляется вместе с дистрибутивом, наберите данный текст программы и запустите её, нажав на F5.

Создайте новые функции **subsets** и **subsets0**, которые генерируют наборы без повторений элементов. Для этого нужно взять за основу функции **tuples** и **tuples0**, а именно, сначала нужно точно повторить их (скопировать), заменить везде **tuples** на **subsets**, а также заменить строку

```
tuples0(size,ary,i,block)
```

на строку

```
subsets0(size,ary,i+1,block)
```

Решите эту задачу, не поленитесь. Проверить работу можно с помощью строки

```
(1..5).to_a.subsets(3) { |subset|  puts subset.inspect }
```

Эта строка должна вывести 10 различных сочетаний 3 элементов из 5 первых натуральных чисел без повторений.

Для решения задачи о «волшебных тройках» осталось сделать совсем немного – определить метод сравнения двух наборов. Для этого давайте переопределим оператор <=> так, чтобы он возвращал -1, 0 или 1, в зависимости от того, какой из двух наборов чисел выигрывает:

1 – первый выигрывает;
0 – ничья;
-1 – второй выигрывает.

Этот оператор именно таким образом определён для обыкновенных чисел. Вот определение этого метода сравнения двух наборов («абстрактных игральных кубиков»):

```
class Array
    def <=>(ary)
        res = 0
        self.each do |x|
            ary.each do |y|
                res += (x<=>y)
            end
        end
    end
end
```

Заметьте, что в Ruby можно свободно «дописывать» существующие классы, добавляя в них новые методы и переопределяя существующие.

Функции **subsets** и **tuples** слишком сильно повторяют друг друга. Это не соответствует основной концепции Ruby – не повторяйся (принцип DRY – don't repeat yourself). Этих повторений можно избежать, введя дополнительный аргумент у функции **tuples0**. Кроме того, хотелось бы, чтобы функции **tuples** и **subsets**, если их вызывать без ассоциированного блока, просто возвращали массив наборов. Это сделано в приведённом ниже полном коде программы. Решение задачи выглядит следующим образом:

Начало кода

```
class Array
    def <=>(ary)
        res = 0
        self.each do |x|
            ary.each do |y|
                res += (x<=>y)
            end
        end
        res <=> 0
    end

    # ïîÿñíåíèå ýòîé ôóíêöèè íàì ïðèä¸òñÿ îòëîæèòü
    # äî ëó÷øèõ âðåì¸í
    def set_optional_block(block)
        if block.nil?
            [result=[], lambda {|x| result << x }]
 else
```

Продолжение кода на следующей странице

Продолжение кода

```
[nil, block]
        end
    end

    # ïåðåáðàòü âñå íàáîðû ñ âîçìîæíûìè ïîâòîðåíèÿìè
    def tuples(size, &block)
        result,block = set_optional_block(block)
        tuples0(size,[],0,0,block)
        result
    end

    # ïåðåáðàòü âñå íàáîðû áåç ïîâòîðåíèé
    def subsets(size, &block)
        result,block = set_optional_block(block)
        tuples0(size,[],0,1,block)
        result
    end

    private
    # ïåðåáðàòü âñå íàáîðû ýëåìåíòîâ ïðè óñëîâèè, ÷òî óæå
    # íàáðàíà ÷àñòü íàáîðà ary è íóæíî äîáðàòü ýëåìåíòîâ
    # èç õâîñòà ìàññèâà self[i..].
    # Ïîâòîðåíèÿ âîçìîæíû, åñëè norepeat = 0.
    def tuples0(size,ary,i,norepeat,block)
        if ary.size == size
            block[ary.dup]
        elsif i < self.size
            # âîçüìåì self[i]
            ary.push self[i]
            tuples0(size,ary,i+norepeat,norepeat,block)
            ary.pop

            # èëè ïðîïóñòèì self[i]
            tuples0(size,ary,i+1,norepeat,block)
        end
    end

end

puts "Âîëøåáíûå òðîéêè:"
puts (1..4).to_a.tuples(6).subsets(3).select { |t|
    ((t[0]<=>t[1]) + (t[1]<=>t[2]) + (t[2]<=>t[0])).abs == 3
}.inspect
```

Данный код не намного короче кода на языке C++.

Но заметьте, что приведённое здесь решение содержит определение функции **tuples** и **subsets**, которые могут быть использованы при решении многих других алгоритмических задач. Причём эти функции

можно использовать с ассоциированным блоком или без, в зависимости от того, хотите вы проитерировать наборы («пробежаться» по ним), или хотите получить массив наборов. Задача была разбита на отдельные методы, так что само решение, по сути, состоит из 4-х последних строчек кода.

Данный код можно существенно ускорить, если сделать кэширование значений сравнения:

```
def <=>(ary)
    @cache = Hash.new if @cache.nil?
    return @cache[ary] if @cache.has_key?(ary)
    res = 0
    self.each do |x|
        ary.each do |y|
           res += (x<=>y)
        end
    end
    @cache[ary]=(res <=> 0)
end
```

Это кэширование позволяет уменьшить время поиска с 11,3 сек. до 2,7 сек. (на ноутбуке Pentium M, 1.7MHz). Для получения информации об использовании памяти и процессорного времени удобно использовать модуль **benchmark**:

```
require 'benchmark'
puts Benchmark.measure {
   puts [1,2,3,4].to_a.tuples(6).subsets(3).select { |t|
        ((t[0]<=>t[1]) + (t[1]<=>t[2]) + (t[2]<=>t[0])).abs == 3
   }.size
}
```

Данный код напечатает число найденных волшебных троек (43) и время, потраченное на вычисления: процессорное время, потраченное на пользовательский код, процессорное время, потраченное на выполнение системных вызовов, и реальное время, которое прошло, пока вычислялся заданный блок.

Документацию по различным классам и их методам можно получить, используя графическое приложение fxri (см. рис. 1), включенное в дистрибутив Ruby под Windows.

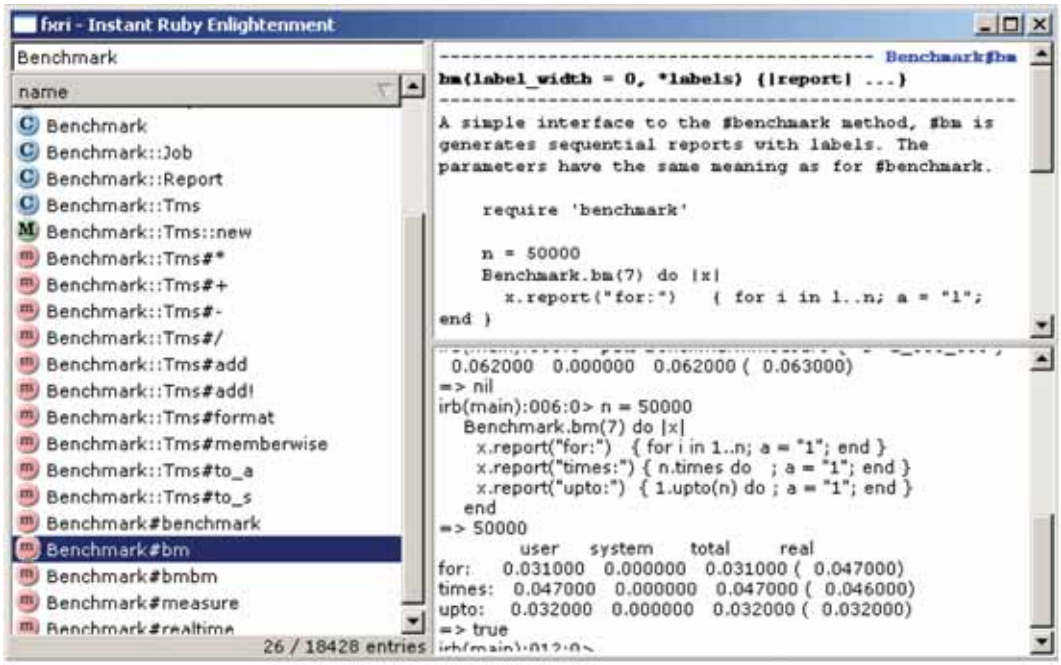

Рис. 1. Скриншот программы, предоставляющей доступ к документации языка Ruby.

См. также

- http://ru.wikibooks.org/wiki/Ruby – книжка про Ruby на русском языке.
- http://acm.mipt.ru/bin/view/Ruby – учебные материалы на сайте МФТИ.
- Programming Ruby, Dave Thomas with Chad Fowler and Andy Hunt (Программирование на Ruby, Дейв Томас, совместно с Чадом Фоулером и Энди Хантом).
- Agile Web Development with Rails – Pragmatic Programmers Guide, Dave Thomas (Гибкое программирование под Web на Rails, Дейв Томас).

Юмор Юмор Юмор Юмор Юмор Юмор

- Студентка на экзамене убеждает преподавателя, что $\ln 0 = e$. Преподаватель, конечно, с этим не согласен. Студентка: «Да что Вы всё время меня запутать пытаетесь?» Достает калькулятор, вычиляет $\ln 0$, и в самом деле, на экране высвечивается E.

- Если вы хотите избавиться от корней сорняков, просто возведите огород в квадрат!