Ruby: хэши

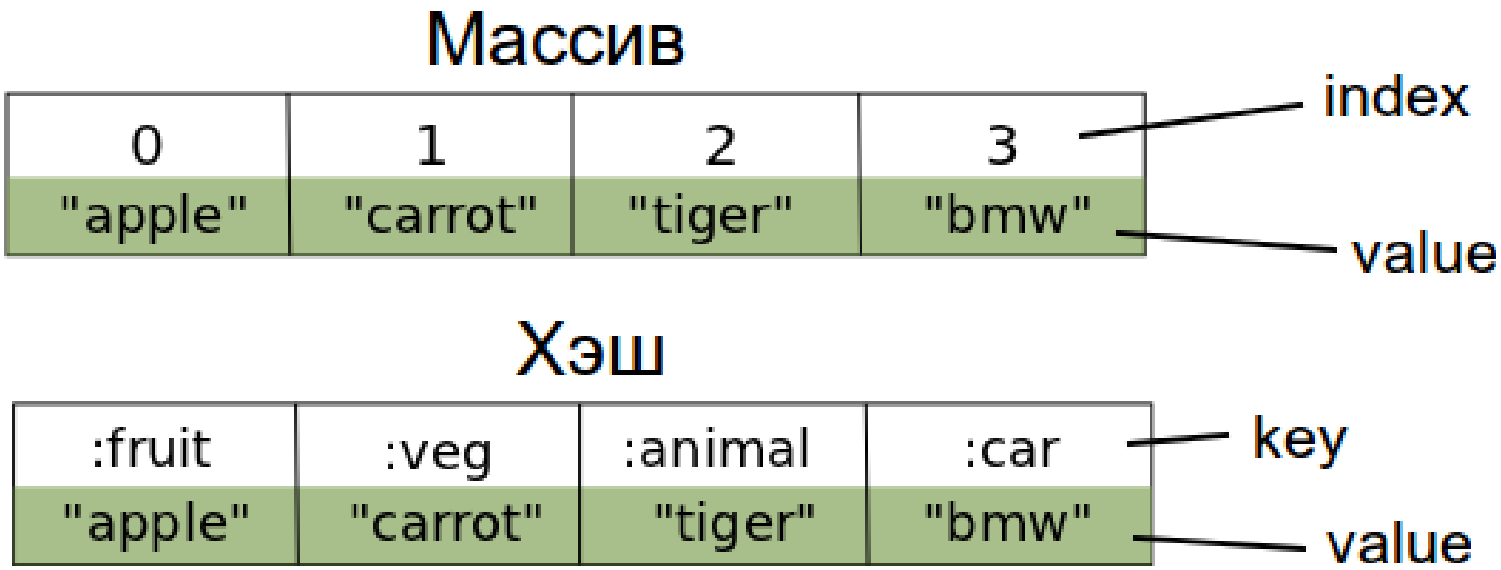
Информатика
10-11 классы

15 февраля 2012 г.

Введение

- Иногда возникает ситуация, когда применение массива является неудачным решением.
- Например, когда индексы расположены неравномерно, с большими пропусками, ведь как мы помним, ключи массива представляют собой последовательные натуральные числа + ноль.
- Эта проблема решается *хэшами*.
- *Хэш* — это массив, ключами которого могут являться строки.

Хэши и массивы



Табличная форма

Ключ	Значение
“kolya”	4
“petya”	5
“sergey”	5
“mamba”	2

- Рассмотрим хэш с оценками.
- В ruby такой хэш записывается следующим образом:

Listing 1: Создание хэша

```
hash = { "kolya"  => 4,
         "petya"  => 5,
         "sergey" => 5,
         "mamba"  => 2 }
```

- где hash — название массива.
- Чтобы вывести на экран, например, оценку Коли, достаточно написать
- **puts hash["kolya"]**.

Создание хэша

- Зададим тестовый хэш телефонных кодов стран.

Listing 2: Способы создания хэша

```
hash = { "Russia" => 7, "USA" => 1, "UK" => 44}

hash = Hash.new (or hash = {})
hash["Russia"] = 7
hash["USA"] = 1
hash["UK"]  = 44
...
```

- 1 способ — обычный, 2 — ручной.

Методы работы с хэшем

Рассмотрим хэш hash = { “vasya” => 5, “kolya” => 4, “petya” => 4 }.

Метод	**Описание**	**Результат**
hash.size	количество пар “ключ-значение”	3
hash.keys	массив ключей*	[“vasya”, “kolya”, “petya”]
hash.values	массив значений*	[5, 4, 4]
hash.invert	поменять ключи и значения местами**	{5 => “vasya”, 4 => “kolya”}
hash.max	поиск максимальной пары	[“vasya”, 5]
hash.min	поиск минимальной пары	[“kolya”, 4]
hash.delete(“vasya”)	удалить элемент по ключу	{ “kolya” => 4, “petya” => 4 }

* хэши в ruby неупорядочены: массивы могут иметь любой порядок элементов.

** при “перевороте” в случае совпадения значений будет выбрано первое.

Методы работы с хэшем - 2

Рассмотрим хэш hash = { “vasya” => 5, “kolya” => 4, “petya” => 4 }.

Метод	**Описание**	**Результат**
hash.empty?	есть ли хоть один элемент	false
hash.key?(“vasya”)	есть ли в хэше элемент с ключом vasya	true
hash.has_key?(“vasya”)	аналогично	true
hash.include?(“vasya”)	аналогично	true
hash.value?(3)	есть ли в хэше элемент со значением 3	false
hash.has_value?(3)	аналогично	false

Сортировка по ключу

- Как и массивы, хэши можно сортировать с помощью метода **sort**.
- Однако на выходе получается не хэш, а двумерный массив пар “ключ-значение”.
- Базово сортировка проводится **по ключам**, а не по значениям, как в массивах.

Listing 3: Сортировка по ключам

```
hash = { "abc" => 10, "def" => 7, "aac" => 25}
hash_sorted = hash.sort
puts hash_sorted.inspect
# [["aac", 25], ["abc", 10], ["def", 7]]
```

Сортировка по значению

- Для сортировки по значению есть специальный метод *sort_by*.
- На выходе опять получается не хэш, а двумерный массив пар “ключ-значение”.

Listing 4: Сортировка по ключам

```
hash = { "abc" => 10, "def" => 7, "aac" => 25}
hash_sorted = hash.sort_by { |key, value| value }
puts hash_sorted.inspect
# [["def", 7], ["abc", 10], ["aac", 25]]
```

Максимальный и минимальный элементы

- Максимальный и минимальный элементы хэша можно найти с помощью методов **max**, **min**, **max_by**, **min_by**.
- На выходе — массив из двух элементов (ключ и значение).
- Первые два метода ищут экстремум по ключу, вторые два — по условию.

Listing 5: Максимальный и минимальный элементы

```
hash = { "abc" => 10, "def" => 7, "aac" => 25}
puts hash.max # ["def", 7]
puts hash.min # ["aac", 25]
puts hash.max_by{ |key, value| value } # ["aac", 25]
arr = hash.min_by{ |key, value| value }
puts arr # ["def", 7]
```

Преобразование хэша в массив

- Иногда в целях удобства (или исходя из технических особенностей, как в методах **sort_by** и пр.) хэши преобразуют в двумерный массив пар “ключ–значение”.
- Для преобразования используется метод **to_a**.

Listing 6: Преобразование в массив

```
hash = { "abc" => 10, "def" => 7, "aac" => 25}
arr  = hash.to_a
```

- В итоге в массив arr будет следующим: [[“abd”, 10], [“def”, 7], [“aac”, 25]]

Итераторы

- Аналогично массивам для хэшей можно использовать методы **map**, **find_all**, **inject**.
- В качестве параметра–"ключа" внутри цикла появляется не просто обычная переменная, а массив из двух элементов — ключа и значения!

Listing 7: Итераторы

```
hash = { "abc" => 10, "def" => 7, "aac" => 25}
res1 = hash.find_all{|array| array[1] < 20}
res2 = hash.map { |array| array[1]*2 }
res3 = hash.inject(0){ |res, arr| res+arr[1] }
```

- Что будет в каждой из переменных?

Результаты работы итераторов

- В переменной **res1** окажется двумерный массив элементов, чьи значения меньше 20: [[“abc”, 10], [“def”, 7]]. Обратите внимание, что результатом вновь окажется именно массив, а не хэш.
- В переменной **res2** окажется одномерный массив значений, умноженных на 2: [20, 50, 14].
- В переменной **res3** будет находиться сумма всех значений хэша: 42.
- **NB**: Не забывайте, что переменная–ключ, используемая в итератора, в случае хэша является массивом из двух элементов — ключа и значения!

Альтернативная запись итераторов

- Если массивы внутри итератора вызывают эстетическое отвращение, можно записать в виде переменных.
- Напишем ту же самую программу, но в другой форме.
- Обратите особое внимание на метод inject. Скобочки там стоят не для красоты!

Listing 8: Альтернативная запись

```
hash = { "abc" => 10, "def" => 7, "aac" => 25}
res1 = hash.find_all{|key, value| value < 20}
res2 = hash.map { |key, value| value*2 }
res3 = hash.inject(0){ |res, (key, value)| res+value }
```

Сумма

- Допустим даны оценки Васи: arr = [3, 5, 5, 4, 5, 2].
- Определим, сколько раз и какую Оценку Вася получил.

Listing 9: Повторяемость

```
arr = [3, 5, 5, 4, 5, 2]
marks = arr.inject(Hash.new{ 0 }){ |res, elem|
  res[elem] += 1
  res
}
puts marks.inspect
```

Разбор программы

- В данном примере мы используем развёрнутую нотацию метода inject.
- Конструкция **Hash.new{ 0 }** означает: создать пустой хэш и сделать любой несуществующий элемент по умолчанию равным нулю.
- **Алгоритм**: заведём хэш res, в котором ключами будут оценки, а значениями — их количество. Изначально каждой оценки — 0 штук.
- Пробегаем по всему массиву **arr**. Берём пробегаемую оценку и увеличиваем количество этих оценок в хэше res на 1.
- Результат выполнения inject записываем в хэш marks.
- Для полного вывода на экран переменной используем специальный метод **inspect**.

Из массива в хэш

- Как преобразовать массив в хэш?
- Пусть дан массив arr. Как сделать чётные элементы ключами хэша, а нечётные — значениями?

Listing 10: Из массива в хэш

```
arr  = [ "abc", 10, "def", 7]
hash = Hash[*arr]
puts hash.inspect
# {"abc"=>10, "def"=>7}
```

Из двумерного массива в хэш

- А если массив двумерный?
- Пусть дан двумерный массив “ключ–значение” arr.
- Сделаем из него хэш. Для этого сначала “сплющим” массив методом **flatten**.

Listing 11: Из массива в хэш

```
arr  = [ ["abc", 10], ["def", 7]]
hash = Hash[*arr.flatten]
puts hash.inspect
# {"abc"=>10, "def"=>7}
```

Задания

- Дана строка s. Посчитать, сколько раз какое слово там встречается?
- Предыдущая задача, только вывести ТОП-10 часто встречающихся слов в порядке убывания.

Задания

- Дан массив строк arr. Каждая строка содержит информацию в следующем формате: дата (число из двух цифр, затем символ ТОЧКА, затем месяц из двух цифр. Примеры: 13.02, 01.01), символ ПРОБЕЛ, температура в виде вещественного числа. Пример правильно оформленной строки (одной из многих в массиве): “02.11 -3.2”. Необходимо сосчитать среднемесячную температуру на каждый месяц, исходя из представленных данных и вывести её на экран по месяцам, начиная с января. Если на какой-либо месяц данные отсутствуют, то напротив этого месяца вывести словосочетание “данные отсутствуют”.

Задания

- Дан двумерный массив, состоящий из двух одномерных массивов равной длины. Составить из этого массива хэш, ключами которого являются элементы первого подмассива, а значениями — второго. Пример: arr = [[1,2,3], [4,5,6]]. Из этого массива должен получиться хэш hash = { 1 => 4, 2 => 5, 3 => 6 }.
- Дан кусок текста в виде строковой переменной s. Посчитайте количество 10 любых союзов (например, “а”, “и”, “а то” и пр.) и выведите их в порядке встречаемости в тексте, начиная с самого часто встречающегося.
- Найти самое употребляемое слово во Вступлении к поэме А.С. Пушкина “Медный всадник”.

References

- При подготовке данного материала использовались сайты: http://ru.wikibooks.org/wiki/Ruby, http://rubydev.ru, http://en.wikipedia.org, http://ruby-lang.org.
- Все презентации доступны на http://school.smirik.ru!
- Вопросы, предложения, д/з: smirik@gmail.com