Ruby学习

数据类型：

Number:1

String:"jim"

Bool:true,false

Array:[1,2,3]

Hash: { :name => "jim", :age => 20 }

赋值：

Ruby中的变量，不需要做类型声明。直接就用：

```
name=“jim”
# => jim
```

命名规则

常量：全都是大写字母.ANDROID_SYSTEM='android'

变量：如果不算@，@@，$的话，是小写字母开头.下划线拼接。例如:color,age,is_created

class,module: 首字母大写，骆驼表达法：Apple，Human

方法名： 小写字母开头，下划线拼接。可以以问号 ？ 或者等号= 结尾，例如：name，created? ，color=

Class的写法

作为面向对象语言，class毫无疑问是最重要的。Ruby中的任何变量都是object（不想java，int是基本数据类型，integer才是class）

具体的写法：

1. 名字首字母大写
2. class开头，end结尾
3. 文件名字与class名称一样，只是改为：下划线+小写

例如：

```
Class Apple
    #这个方法就是在Apple.new 时自动调用的方法
    def initialize
        #instance variable,实例变量
        @color
    end

    # getter 方法
    def color
        return @color
    end

    # setter 方法
    def color=color
        @color=color
    end

    #private
    def i_am_private
    end
end

red_apple=Apple.new
red_apple.color='red'
puts "red_apple.color:#{red_apple.color}"
```

例2：

```
ruby apple.rb
# => "red_apple.color:red"

class Apple
    #自动声明了@color, getter, setter
    attr_accessor 'color'
end
```

各种变量

类变量：class variable,例如：@@name，作用域：在这个类中，所有的多个instance会共享这个变量，用的很少。

实例变量：instance variable，例如：@color，作用域仅在instance之内，在rails中被大量使用（比如小明的name属性只是他自己的，小红的name属性是她自己的，不同的实例，name属性不同）

普通变量：local variable，例如：age=20，作用域仅在某个方法内，大量使用

全局变量：global variable,例如：$name="jim",作用域在全局。用的更少。

例如：

```
class Apple
    @@from = 'china'

    def color= color
        @color = color
    end

    def color
        return @color
    end

    def get_from
        @@from
    end

    def set_from= from
        @@from = from
```

```
    end
end
```

```
red_one = Apple.new
red_one.color = 'red'
puts red_one.color

red_one.set_from('Japan')
puts red_one.get_from
```

方法：类方法与实例方法

类方法:

跟实例无关，可以由class直接调用的方法

实例方法:

由某个class的实例调用的方法

例如:

```
class Apple
    #类方法
    def Apple.name
        'apple'
    end

    #实例方法
    def color
        'red'
    end
end

Apple.new.color

Apple.name
```

字符串

```
single_line="我是一个字符串"
multiple_line=%Q{
    今天天气不错
    出去走走
    }
```

Symbol

不变的字符串

```
#内容永远不变，等同于一个常量
#特别适合做hash的key
#大量被用到

class Apple
    attr_accessor :color
end

#该 :color 就是symbol.不会变化的字符串,
#:name等同与"name".to_symbol
```

判断数据类型

```
'abc'.class
:abc.class
'abc'.to_sym
```

字符串插值

```ruby
puts 'hi,#{name}!'
puts 'hi,jim'

#报错
a=1
puts "a is:" + a

puts "a is: #{a}"
```

数组

```
包含同一数据类型的数组:
numbers = [1,2,3]
包含多种数据类型的数组:
[1,'two',:three,{ :name => 4 }]
```

Hash: key/value

```
{
  :name => 'jim',
  :age => 18
}

hash,也叫dictionary
```

同一个hash的三种写法

```ruby
#任何情况下都生效的语法:  =>
jim = {
     :name => 'jim',
     :age => 20
     }

#Ruby1.9之后产生的语法: 更加简洁
jim={
     name:'jim',
     age:20
     }

```

```
#也可以写成:
jim={}
jim[:name]='jim'
jim[:age]=20
```

hash的key：symbol与string不同

但是，symbol与string，是不同的key，例如：

```
a = {:name =>'jim','name'=>'hi'}
a[:name] #=> 'jim'
a['name'] #=> 'hi'
```

条件语句 if-else

if else end 是最常见的

```
a=1
if a == 1
    puts "a is 1"
elsif a == 2
    puts "a is 2"
else
    puts "a is not in [1,2]"
end
```

case 分支语句

例如：

```
a=1
case a
    when 1 then puts "a is 1"
    when 2 then puts "a is 2"
    when 3,4,5 then puts "a is in [3,4,5]"
    else puts "a is not in [1,2,3,4,5]"
end
```

三元表达式

```
a=1
puts a == 1 ? 'one':'not one'
#=> one
```

循环：for,each,loop,while

for,each (前者是关键字，后者是普通方法)

```
#for 与 each 几乎一样，例如：
    [1,2,3].each {
        |e|
        puts e
        }
#等同于下方
for e in [1,2,3]
    puts e
end
```

eachd 与for的区别

for与each都可以做循环，一般使用each

区别在于：for是关键字，each是方法。for后面的变量，是全局变量，不仅仅存在与for...end这个作用域之内

循环：while与loop

loop与while是几乎一样的

```ruby
loop do
    #your code
    break if <condition>
end

while <condition>
   #code
end
```

例如:

```ruby
a=[2,1,0,-1,-2]
loop do
    current_element =a.pop
    puts current_element
    break if current_element <0
end

count = 1

while count < 10
 puts count
 count = count + 1
end
```

Heredoc表示法

```ruby
a = 'in here'

b = <<METHOD_DESCRIPTION
This is a  test_sting.
just fun with ruby #{a}
METHOD_DESCRIPTION

print b
```

Bool

```
true    true
false   false
Object  true
0       true
1       true
-1      true
nil     false
""      true
[]      true
{}      true
```

内置方法

```
数组:
    1. 增加
        a. months  <<  "August"                往数组最后的位置加入新值
        b. months.push("September")
        c. months.insert(2,"October")  往索引位置插入新值
    2. 删除
        a. months.pop 删除数组中最后一项，如果有参数，则删除相应个数的item
        b. months.delete_at(2) 删除指定索引位置的item
    3. 改
        a. months[索引] = 新的值  改变原数组的值
    4. 查
        a. months.Include?(检验的值)   检查参数的值是不是数组中的元素
    5. 内置方法
        a. months.sort  对数组进行排序
        b. months.flatten 将嵌套数组合并成一维数组
        c. months.each { |item| puts item } 数组的迭代器 将数组中的每个item运行一次block中定义的代码
        d. months.map { |item| item**2}  对数组每个元素调用块内的代码一次，返回包含新值的数组

哈希:
    1. 创建:
        a. person = { "key" => "value"}
        b. person = Hash.new
```

```
    2. 访问:
        a. person["key"]
    3. 增加:
        a. person["key"] = 'value'
    4. 删除:
        a. person.delete("key")
    5. 内置方法:
        a. Person.each do |key,value|
          Puts "#{key} is #{value}"
        End
        b. person.has_key?("key")  检查hash中有没有特定的键
        c. person.select{ |key,value| key == "name" }  根据块中的条件，检索符合的键值对
        d. person.fetch("name") 返回指定键的值
集合
    1. 创建:
        require 'set'
        my_set = Set.new([5, 2, 9, 3, 1])
    2. 增加:
        a. my_set << 5 向数组尾部加入新值
        b.  my_set.add 1
Range
    1. …  三点[1,10) 1到9
    2. ..  二点[1,10]  1到10
a  = Range.new(1,10)  和二点一样
```