

# Ruby 快速参考

对象、块、迭代器、正则、文件 I/O 要点

## 基础

### Hello World

```
puts "Hello, World!"
print "no newline"
p [1, 2, 3] # inspect output: [1, 2, 3]
```

### 运行 Ruby

```
ruby script.rb # run a file
ruby -e 'puts "hi"' # run inline
irb # interactive REPL
```

## 变量

<b>name</b>	局部变量
<b>@name</b>	实例变量
<b>@@count</b>	类变量
<b>\$debug</b>	全局变量
<b>MAX_SIZE</b>	常量 (约定全大写)

## 类型

42.class	# Integer
3.14.class	# Float
"hello".class	# String
true.class	# TrueClass
nil.class	# NilClass
:symbol.class	# Symbol

## 字符串

### 字符串基础

```
name = "World"
puts "Hello, #{name}!" # interpolation (double quotes)
puts 'No #{interpolation}' # literal (single quotes)
multi = <<-HEREDOC
  indented heredoc
HEREDOC
```

### 字符串方法

<b>.length / .size</b>	字符数量
<b>.upcase / .downcase</b>	大小写转换
<b>.strip</b>	去除首尾空白
<b>.split(' ,')</b>	分割为数组
<b>.gsub(/pat/, 'rep')</b>	全局替换
<b>.include?('sub')</b>	是否包含子串
<b>.start_with?('pre')</b>	是否以某前缀开头
<b>.chars / .bytes</b>	字符/字节数组
<b>.to_i / .to_f</b>	转换为整数/浮点数
<b>.freeze</b>	使字符串不可变

## 数组与 Hash

### 数组

```
arr = [1, "two", :three]
arr << 4 # push (append)
arr[0] # 1
arr[-1] # 4 (last element)
arr[1..2] # ["two", :three] (slice)
```

## 数组方法

<b>.push / .pop</b>	从末尾添加/删除
<b>.shift / .unshift</b>	从开头删除/添加
<b>.flatten</b>	展平嵌套数组
<b>.compact</b>	删除 nil 值
<b>.uniq</b>	去除重复元素
<b>.sort / .reverse</b>	排序/反转
<b>.map {  x  x * 2 }</b>	对每个元素进行变换
<b>.select {  x  x &gt; 0 }</b>	过滤元素
<b>.reduce(0) {  sum, x  sum + x }</b>	归约为单一值

## Hash

```
user = { name: "Alice", age: 30 } # symbol keys
old = { "key" => "value" } # string keys
user[:name] # "Alice"
user[:email] = "a@b.com" # add pair
user.fetch(:name, "default") # with default
```

## Hash 方法

<b>.keys / .values</b>	键/值数组
<b>.each {  k, v  }</b>	遍历键值对
<b>.merge(other)</b>	合并两个 Hash
<b>.key?(k) / .value?(v)</b>	检查是否存在
<b>.select {  k, v  }</b>	过滤键值对
<b>.transform_values {  v  }</b>	变换所有值

## 控制流

### 条件语句

```
if score >= 90 then "A"
elsif score >= 80 then "B"
else "C"
end
puts "adult" if age >= 18 # inline if
puts "minor" unless age >= 18 # inline unless
```

### Case / When

```
case status
when :ok then puts "success"
when :error then puts "failed"
when 400..499 then puts "client error"
else puts "unknown"
end
```

## 循环

```
5.times { |i| puts i }
(1..10).each { |n| puts n }
while condition do end
until condition do end
loop { break if done }
```

## 三目运算符与逻辑

```
status = age >= 18 ? "adult" : "minor"
name = input || "default" # or-assign
name ||= "fallback" # same effect
```

## 方法

### 定义方法

```
def greet(name, greeting = "Hello")
  "#{greeting}, #{name}!"
end
greet("Alice") # "Hello, Alice!"
greet("Bob", "Hi") # "Hi, Bob!"
```

## 返回值

```
def add(a, b)
  a + b # last expression is implicit return
end
def divide(a, b)
  return nil if b == 0
  a.to_f / b
end
```

## 关键字参数与 Splat

```
def connect(host:, port: 80, **opts)
  puts "#{host}:#{port} #{opts}"
end
def log(*messages)
  messages.each { |m| puts m }
end
```

## 方法命名约定

<b>method?</b>	返回布尔值 (谓词方法)
<b>method!</b>	修改接收者 (bang 方法)
<b>self.method</b>	类方法定义

## 类

### 类定义

```
class User
  attr_accessor :name, :email
  def initialize(name, email)
    @name = name
    @email = email
  end
end
```

## 继承

```
class Admin < User
  def initialize(name, email, level)
    super(name, email)
    @level = level
  end
end
```

## 访问控制

<b>public</b>	默认; 任何地方均可访问
<b>private</b>	仅本类内部可访问
<b>protected</b>	本类及子类可访问
<b>attr_reader</b>	生成 getter 方法
<b>attr_writer</b>	生成 setter 方法
<b>attr_accessor</b>	生成 getter 和 setter

## 模块

### Mixin

```
module Greetable
  def greet
    "Hello, I'm #{name}"
  end
end
class User; include Greetable; end
```

## 命名空间

```
module Payment
  class Processor
    def charge(amount) end
  end
end
p = Payment::Processor.new
```

# Ruby 快速参考

## include vs extend

<b>include ModName</b>	作为实例方法引入
<b>extend ModName</b>	作为类方法引入
<b>prepend ModName</b>	在方法查找链中插到类前面

## 块与迭代器

### 块语法

```
[1, 2, 3].each { |n| puts n }      # single-line block
[1, 2, 3].each do |n|
  puts n                          # multi-line block
end
```

### yield

```
def with_logging
  puts "start"
  result = yield
  puts "end"
  result
end
with_logging { expensive_operation }
```

## Proc 与 Lambda

```
square = Proc.new { |x| x ** 2 }
square.call(5)          # 25
double = ->(x) { x * 2 } # lambda
double.call(3)         # 6
[1, 2, 3].map(&square)  # [1, 4, 9]
```

## 常用迭代器

<b>.each</b>	遍历元素
<b>.map / .collect</b>	对每个元素进行变换
<b>.select / .filter</b>	保留匹配元素
<b>.reject</b>	删除匹配元素
<b>.reduce / .inject</b>	归约为单一值
<b>.each_with_index</b>	带索引遍历
<b>.flat_map</b>	map 后展开一层
<b>.any? / .all? / .none?</b>	集合的布尔检查

## 正则表达式

### 匹配

```
"hello 42" =~ /\d+/      # 6 (match position)
"hello" =~ /\d+/        # nil (no match)
"hello".match?(/ell/)   # true
md = "age: 30".match(/(\d+)/)
md[1]                   # "30"
```

### 常用模式

<b>/^start/</b>	匹配开头
<b>/end\$/</b>	匹配结尾
<b>/\d+/</b>	一个或多个数字
<b>/\w+/</b>	单词字符
<b>/\s+/</b>	空白字符
<b>/[a-z]+/i</b>	忽略大小写
<b>/(group)/</b>	捕获组

### 替换

```
"hello world".sub(/world/, "Ruby") # first match
"aabba".gsub(/a/, "x")            # all matches: "xxbbx"
"foo bar".gsub(/(\w+)/) { $1.upcase } # "FOO BAR"
```

## 文件 I/O

### 读写文件

```
content = File.read("data.txt")
lines = File.readlines("data.txt", chomp: true)
File.write("out.txt", "hello\n")
File.open("log.txt", "a") { |f| f.puts "entry" }
```

### 文件操作

<b>File.exist?(path)</b>	检查文件是否存在
<b>File.directory?(path)</b>	检查路径是否为目录
<b>File.basename(path)</b>	不含目录的文件名
<b>File.extname(path)</b>	文件扩展名
<b>File.size(path)</b>	文件大小 (字节)
<b>File.delete(path)</b>	删除文件
<b>Dir.glob('*.*rb')</b>	按模式查找文件
<b>FileUtils.mkdir_p(path)</b>	递归创建目录

## CSV 与 JSON

```
require "json"
data = JSON.parse(File.read("data.json"))
File.write("out.json", JSON.pretty_generate(data))
require "csv"
CSV.foreach("data.csv", headers: true) { |row| puts row["name"] }
```