# Ruby基础教程（第4版）练习题参考答案

## 第12章

（1）的参考答案

使用问题中提供的公式直接定义方法。

```
def cels2fahr(cels)
  return cels * 9.0 / 5.0 + 32.0
end
```

虽然问题中的公式为“华氏 = 摄氏 × 9 ÷ 5 + 32”，但为了用浮点小数表示计算结果，9、5分别转换为9.0、5.0。如果不做这样的转换，当cels为整数时无法得到正确的结果（除以5的结果为Integer）。

（2）的参考答案

定义（1）的逆运算。请读者注意，由于“+”与“*”优先级不一样，因此这里需要加上括号。

```
def fahr2cels(fahr)
  return (fahr.to_f - 32) * 5.0 / 9.0
end
```

接下来，使用upto方法实现华氏温度从1~100的递增。

```
1.upto(100) do |i|
  print i, " ", fahr2cels(i), "\n"
end
```

（3）的参考答案

使用10.6节中介绍的rand方法。由于执行“rand(6)”时，所得到的返回值只是0到5，因此我们还需要将得到的结果加1。

```
def dice
  return rand(6) + 1
end
```

（4）的参考答案

可能有读者想直接使用dice+dice+……这样的写法，但在这里我们应该使用循环来解答问题。

```
def dice10
  ret = 0
  10.times do
    ret += dice
  end
  ret
end
```

（5）的参考答案

首先，比2小的数不可能为素数。对于2以上的数，我们将该数与从2开始到该数平方根之间的整数逐个进行除法运算，确认所有运算结果的余数都不为0。

```
def prime?(num)
  return false if num < 2
  2.upto(Math.sqrt(num)) do |i|
    if num % i == 0
      return false
    end
  end
  return true
end
```

# 第13章

（1）的参考答案

在这里我们列举两种方法。使用像“ary = [1, 2, 3, ...]”这样一般用于定义小数组的方法，以及使用字面量直接定义数组的方法。

```
# 创建空数组，然后把1到100的值逐个放到数组中
a = []
100.times{|i| a[i] = i + 1 }

# 使用范围对象的to_a方法
a = (1..100).to_a
```

（2）的参考答案

一般使用Array#collect方法时会创建新的数组。在原数组的基础上扩大100倍时，我们使用带“!”的Array#collect!方法。

```
# 创建数组
a = (1..100).to_a

# 利用全部数组元素扩大100倍后的值创建新的数组
a2 = a.collect{|i| i * 100 }
p a2

# 全部数组元素扩大100倍
a.collect!{|i| i * 100 }
p a
```

（3）的参考答案

使用Array#reject方法去除符合条件的元素。

```
# 创建数组
a = (1..100).to_a

# 取出ary中为3的倍数的元素
a3 = a.reject{|i| i % 3 != 0 }
p a3

# 另外，还有只返回符合条件的元素，不需要带!的select方法
a4 = a.select{|i| i % 3 == 0 }
p a4

# 删除ary中3的倍数以外的元素
a.reject!{|i| i % 3 != 0 }
p a
```

（4）的参考答案

在Array#sort以及Array#sort_by方法中，若将块结果乘以-1，则会对结果进行倒排序。

```
# 创建数组
a = (1..100).to_a

# (a) 使用Array#reverse方法
a2 = a.reverse
p a2

# (b) 使用Array#sort方法
a2 = a.sort{|n1, n2| -(n1 <=> n2) }
p a2

# (c) 使用Array#sort_by方法
a2 = a.sort_by{|i| -i }
p a2
```

(5) 的参考答案

使用Array#inject方法是本题的另外一种解法。使用Array#each方法时，我们需要定义变量（本例中的result）用于保存累加的值，而使用Array#inject方法则不需要。

```
# 创建数组
a = (1..100).to_a

# (a) 使用Array#each方法求和
result = 0
a.each{|i| result += i }
p result

# (b) 使用Array#inject方法求和
p a.inject(0){|memo, i| memo += i }
```

(6) 的参考答案

指定首个元素索引以及所获取的元素个数。

```
# 创建数组
ary = (1..100).to_a
result = Array.new
10.times do |i|
  result << ary[i*10, 10]
end
p result
```

(7) 的参考答案

本题的解答重点在于，使用Array#each遍历ary1各元素的同时，使用相应的索引访问ary2的各元素。另外一种解法是使用Array#zip方法，该方法可以同时方法配对的两个数组中的各元素。

```
def sum_array(ary1, ary2)
  result = Array.new
  i = 0
  ary1.each do |elem1|
    result << elem1 + ary2[i]
    i+=1
  end
  return result
end

# 使用Array#zip方法的另外一种解法
def sum_array_zip(ary1, ary2)
  result = Array.new
  ary1.zip(ary2){|a, b| result << a + b }
  return result
end

p sum_array([1, 2, 3], [4, 6, 8])
```

# 第14章

（1）的参考答案

直接使用split方法即可。

```
str = "Ruby is an object oriented programming language"
ary = str.split
p ary
```

（2）的参考答案

对只由英文字母字符串，不带参数调用Array.sort即可以进行排序。这里我们顺便复习一下数组的用法。

```
str = "Ruby is an object oriented programming language"
ary = str.split
p ary.sort
```

（3）的参考答案

在这里，我们使用Array#sort_by方法对参数进行比较。此时，我们先使用String#downcase方法强制将字符

串转换为小写后再进行比较，由此实现不区分大小写进行排序。

```
str = "Ruby is an object oriented programming language"
ary = str.split
p ary.sort_by{|s| s.downcase }
```

（4）的参考答案

使用String#capitalize方法将字符串的首字母转换为大写。配合Array#collect方法，对数组中各元素进行转换。

```
str = "Ruby is an object oriented programming language"
ary = str.split
cap_ary = ary.collect{|word| word.capitalize }

str = ""
cap_ary.each do |s|
  str << s+" "
end
p str

## 另一种解法
p cap_ary.join(" ")
```

另外，使用另外一种解法时，Array#join方法使字符串连接变得简单。这个方法可以连接数组中的字符串，还可以通过参数指定各元素间的连接符。

```
p ["a", "b", "c"].join        #=> "abc"
p ["a", "b", "c"].join("-")   #=> "a-b-c"
```

（5）的参考答案

创建散列，将字符作为键，该字符出现的次数作为值进行记录。最后对字符进行排序，将出现次数转换为星号后输出。

```
str = "Ruby is an object oriented programming language"
count = Hash.new
str.each_char do |c|
  count[c] = 0 unless count[c]
  count[c] += 1
end
count.keys.sort.each do |c|
  printf("'%s': %s\n", c, "*" * count[c])
end
```

在初始化散列时，可以像下面那样将0作为散列的默认值返回。

```
str = "Ruby is an object oriented programming language"
count = Hash.new(0)
str.each_char do |c|
  count[c] += 1
end
count.keys.sort.each do |c|
  printf("'%s': %s\n", c, "*" * count[c])
end
```

（6）的参考答案

本题是一个颇复杂的实际问题。

```
def han2num(string)
  digit4 = digit3 = digit2 = digit1 = "0"

  nstring = string.dup
  nstring.gsub!(/一/, "1")
  nstring.gsub!(/二/, "2")
  nstring.gsub!(/三/, "3")
  nstring.gsub!(/四/, "4")
  nstring.gsub!(/五/, "5")
  nstring.gsub!(/六/, "6")
  nstring.gsub!(/七/, "7")
  nstring.gsub!(/八/, "8")
  nstring.gsub!(/九/, "9")

  if nstring =~ /((\d)?千)?((\d)?百)?((\d)?十)?(\d)?$/
    if $1
      digit4 = $2 || "1"
    end
    if $3
      digit3 = $4 || "1"
    end
    if $5
      digit2 = $6 || "1"
    end
    digit1 = $7 || "0"
  end

  return (digit4+digit3+digit2+digit1).to_i
end

p han2num("七千八百二十三")
p han2num("千八百二十三")
p han2num("八百二十三")
p han2num("百二十三")
p han2num("百三")
p han2num("二十三")
p han2num("十三")
p han2num("三")
```

# 第15章

（1）的参考答案

虽然可以逐个分别定义，但这次我们试试合在一起定义。

```
wday = {
  "sunday"    => "星期天",
  "monday"    => "星期一",
  "tuesday"   => "星期二",
  "wedensday" => "星期三",
  "thursday"  => "星期四",
  "friday"    => "星期五",
  "saturday"  => "星期六",
}
```

(2) 的参考答案

直接使用Hash#size方法即可。

```
wday = {
  "sunday"    => "星期天",
  "monday"    => "星期一",
  "tuesday"   => "星期二",
  "wedensday" => "星期三",
  "thursday"  => "星期四",
  "friday"    => "星期五",
  "saturday"  => "星期六",
}

p wday.size  #=> 7
```

(3) 的参考答案

一般可用数组作为键，此次为了简化程序使用了%w。

```
wday = {
  "sunday"    => "星期天",
  "monday"    => "星期一",
  "tuesday"   => "星期二",
  "wedensday" => "星期三",
  "thursday"  => "星期四",
  "friday"    => "星期五",
  "saturday"  => "星期六",
}

%w(sunday monday tuesday wedensday thursday friday saturday).each do |day|
  puts "“#{day}”是#{wday[day]}。"
end
```

(4) 的参考答案

使用String#split分割字符串后，再使用Array#shift方法将元素逐个取出，创建散列。

```
def str2hash(str)
  hash = Hash.new()
  array = str.split(/\s+/)
  while key = array.shift
    value = array.shift
    hash[key] = value
  end
  return hash
end

p str2hash("bule 蓝 white 白\nred 红")
```

# 第16章

（1）的参考答案

判断电子邮件格式规则颇为复杂，甚至有些正在使用的电子邮件格式本身就不符合规则，因此解析电子邮件不是一件容易的事情，这里我们用简化后的规则进行解析。

```
def get_local_and_domain(str)
  str =~ /^([^@]+)@(.*)$/
  localpart = $1
  domain = $2
  return [localpart, domain]
end

p get_local_and_domain("info@example.com")
```

（2）的参考答案

很难只置换一次就达到题目要求，因此我们分两次置换。若先用“简单”置换“难”，则“难懂”部分则会变为“简单懂”，因此我们首先用“易懂”置换“难懂”。

```
s = "正则表达式真难啊，怎么这么难懂！"
puts s.gsub(/难懂/, "易懂").gsub(/难/, "简单")
```

（3）的参考答案

思路基本上与第14章的问题（4）一样，在正则表达式中使用“-”时需要经过转义。在下面的参考答案中，我们尝试使用方法链（method chain）将代码都写在一行中。

```
def word_capitalize(str)
  return str.split(/\-/).collect{|w| w.capitalize}.join('-')
end

p word_capitalize("in-reply-to") #=> "In-Reply-To"
p word_capitalize("X-MAILER")    #=> "X-Mailer"
```

# 第17章

（1）的参考答案

定义wc方法，用于统计文本行数、单词数和字符数。有一点需要读者注意，本例中使用了String#split方法分割单词，当行首有空白字符时，String#split方法的执行结果中会产生空白字符串，因此我们会删除该空白字符串。

```
def wc(file)
  nline = nword = nchar = 0
  File.open(file){|io|
    io.each{|line|
      words = line.split(/\s+/).reject{|w| w.empty? }
      nline += 1
      nword += words.length
      nchar += line.length
    }
  }
  puts "lines=#{nline} words=#{nword} chars=#{nchar}"
end

wc(__FILE__)
```

（2）的参考答案

根据条件分别定义不同的脚本。首先是文件逆序排列。用IO#readlines方法逐行读取文件后，再用IO#rewind方法返回文件开头，用IO#truncate方法清空内容，最后用Array#reverse方法将逆序后的行写入文件。

```
def reverse(input)
  open(input, "r+") do |f|
    lines = f.readlines
    f.rewind
    f.truncate(0)
    f.write lines.reverse.join()
  end
end

reverse(ARGV[0])
```

接下来是输出第1行的内容，实际上只需稍微修改一下上面的程序即可实现。

```
def reverse(input)
  open(input, "r+") do |f|
    lines = f.readlines
    f.rewind
    f.truncate(0)
    f.write lines[0]
  end
end

reverse(ARGV[0])
```

最后是输出最后一行的内容，这里稍微组合了一下（a）（b）的程序。

```
def reverse(input)
  open(input, "r+") do |f|
    lines = f.readlines
    f.rewind
    f.truncate(0)
    f.write lines.reverse[0]
  end
end

reverse(ARGV[0])
```

（3）的参考答案

本题的关键在于将读取的行临时保存在变量queue中。

```
def tail(lines, file)
  queue = Array.new
  open(file) do |io|
    while line = io.gets
      queue.push(line)
      if queue.size > lines
        queue.shift
      end
    end
  end
  queue.each{|line| print line }
end

puts "==="
tail(10, __FILE__)

puts "==="
tail(3, __FILE__)
```

# 第18章

（1）的参考答案

使用FileTest.directory?方法排除目录以外的对象后，再使用Dir.open方法检查目录内的文件名。

```
def print_libraries
  $:.each do |path|
    next unless FileTest.directory?(path)
    Dir.open(path) do |dir|
      dir.each do |name|
        if name =~ /\.rb$/i
          puts name
        end
      end
    end
  end
end

print_libraries
```

另外，虽然在正文并没有详细说明，Ruby中引用的库除了用Ruby实现的之外，还有用C等其他语言实现的。扩展库的文件后缀不是“.rb”，而是像“.dll”“.so”等这样平台依赖的文件。
利用rbconfig库取得后缀名，支持扩展库的版本如下所示。

```
require "rbconfig"

def print_libraries
  $:.each do |path|
    next unless FileTest.directory?(path)
    dlext = RbConfig::CONFIG["DLEXT"]
    Dir.open(path) do |dir|
      dir.each do |name|
        if name =~ /\.rb$/i || name =~ /\.#{dlext}$/i
          puts name
        end
      end
    end
  end
end

print_libraries
```

（2）的参考答案

利用了find库。

```
require "find"

def du(path)
  result = 0
  Find.find(path){|f|
    if File.file?(f)
      result += File.size(f)
    end
  }
  printf("%d %s\n", result, path)
  return result
end

du(ARGV[0] || ".")
```

# 第19章

（1）的参考答案

```
# encoding: utf-8

def to_utf8(str_gbk, str_gb2312)
  ## 使用encode方法将字符串分别转换为UTF-8编码后再连接
  str_gbk.encode("UTF-8") + str_gb2312.encode("UTF-8")
end

## 像下面这样执行
str_gbk  = "你好".encode("GBK")
str_gb2312 = "再见".encode("GB2312")

puts to_utf8(str_gbk, str_gb2312)
```

（2）的参考答案

按照练习题的处理顺序实现了要求。不过确认创建后的文件编码可能不是一件容易的事。

```
# encoding: utf-8

## 用GBK编码创建gbk.txt
File.open("gbk.txt","w:GBK") do |f|
  f.write("你好")
end

## 打开gbk.txt，按UTF-8编码方式输出
File.open("gbk.txt","r:GBK") do |f|
  str = f.read
  ## str的编码为GBK，因此在使用puts输出时需将其转换为UTF-8编码
  puts str.encode("UTF-8")
end
```

（3）的参考答案

GB 18030，全称：国家标准GB 18030-2005《信息技术 中文编码字符集》，是中华人民共和国现时最新的内码字集，是GB 18030-2000《信息技术 信息交换用汉字编码字符集 基本集的扩充》的修订版。与GB 2312-1980完全兼容，与GBK基本兼容，支持GB 13000及Unicode的全部汉字，共收录汉字70244个。
GB 18030主要有以下特点：

- 代码页 54936；
- 与 UTF-8 相同，采用多字节编码，每个字可以由1个、2个或4个字节组成；
- 编码空间庞大，最多可定义161万个字符；
- 支持中国国内少数民族的文字，不需要动用造字区；
- 汉字收录范围包含繁体汉字以及日韩汉字。

GBK即汉字内码扩展规范，K为汉语拼音 Kuo Zhan（扩展）中“扩”字的声母。英文全称Chinese Internal

Code Specification。
由于GB 2312-80只收录了6763个汉字，部分在GB 2312-80推出以后才简化的汉字（如“啰”），部分人名用字（如中国前总理朱镕基的“镕”字），台湾及香港使用的繁体字，日语及朝鲜语汉字等，并未收录在内。于是微软利用GB 2312-80未使用的编码空间，收录GB 13000.1-93全部字符制定了GBK编码。

本参考答案中，将UTF-8字符串“禧”分别转换为GB1830编码以及GBK编码。由于GBK编码并没有收录该汉字，因此在转换时会产生错误。

```
# encoding: utf-8

str = '禧'

encoding = [Encoding::GB18030, Encoding::GBK]
encoding.each do |enc|
  begin
    print "将str转换为#{enc}。=> "
    puts "结果: %p" % [str.encode(enc)]
  rescue => ex
    p ex
  end
  puts
end
```

（4）的参考答案

两者都不是UTF-8编码，因此我们首先都转换为UTF-8编码后再比较。

```
# encoding: utf-8

Dir.glob("*.txt") do |filename|
  ## 将UTF8-MAC编码的文件名转换为UTF-8编码后，就可以与UTF-8编码的字符串“ルビー.txt”进行比较
  if filename.encode("UTF8-MAC").encode("UTF-8") == "ルビー.txt"
    puts "found."; exit
  end
end
puts "not found."
```

# 第20章

（1）的参考答案

通过正则表达式把日期与时间转换为中文字符串。当前时间用Time.now方法获取，用以填补传进来的字符串缺少的项目。最后，利用Time.mktime方法生成时间。

```
def cparsedate(str)
  now = Time.now
  year = now.year
  month = now.month
  day = now.day
  hour = now.hour
  min = now.min
  sec = now.sec
  str.scan(/(上午|下午)?(\d+)(年|月|日|点|分|秒)/) do
    case $3
    when "年"
      year = $2.to_i
    when "月"
      month = $2.to_i
    when "日"
      day = $2.to_i
    when "点"
      hour = $2.to_i
      hour += 12 if $1 == "下午"
    when "分"
      min = $2.to_i
    when "秒"
      sec = $2.to_i
    end
  end
  return Time.mktime(year, month, day, hour, min, sec)
end

p cparsedate("2010年12月23日下午8点17分50秒")
p cparsedate("12月23日下午8点17分50秒")
p cparsedate("上午8时17分50秒")
p cparsedate("8点17分50秒")
```

（2）的参考答案

先删除以“.”开始的文件，再利用File.mtime方法取得时间后排序。最后输出文件名以及日期。

```
def ls_t(path)
  entries = Dir.entries(path)                 # 获取入口
  entries.reject!{|name| /^\./ =~ name }      # 删除文件名以"."开始的文件

  mtimes = Hash.new                           # 边收集mtime边排序
  entries = entries.sort_by do |name|
    mtimes[name] = File.mtime(File.join(path, name))
  end

  entries.each do |name|
    printf("%-40s %s\n", name, mtimes[name]) # 输入文件名以及mtime
  end
rescue => ex
  puts ex.message
end

ls_t(ARGV[0] || ".")
```

（3）的参考答案

下面是利用文具万年历的原理整理的日历。2月30日等不存在的日期，与月末的日期进行比较后就会弹起来（跳到下一个月）。另外，在表格中不存在的日期，为了让它们遇到相同的条件时也能弹起来，我们将它们的初始值设为99。

```
require "date"

module Calendar
  WEEK_TABLE = [
    [99, 99, 99, 99, 99, 99,  1,  2,  3,  4,  5,  6,  7],
    [ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14],
    [ 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21],
    [16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28],
    [23, 24, 25, 26, 27, 28, 29, 30, 31, 99, 99, 99, 99],
    [30, 31, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99],
  ]

  module_function

  def cal(year, month)
    first = Date.new(year, month, 1)       # 被指定的月的1号
    end_of_month = ((first >> 1) - 1).day  # 次月的1号的前1天
    start = 6 - first.wday                 # 表示在表格的哪个位置

    puts first.strftime("%B %Y").center(21)
    puts " Su Mo Tu We Th Fr St"
    WEEK_TABLE.each do |week|
      buf = ""
      week[start, 7].each do |day|
        if day > end_of_month
          buf << "   "
        else
          buf << sprintf("%3d", day)
        end
      end
      puts buf
    end
  end
end

t = Date.today
Calendar.cal(t.year, t.month)
```

# 第21章

（1）的参考答案

通过each方法从obj中逐个获取元素并用块执行，然后再把执行后的结果保存到数组。

```
def my_collect(obj, &block)
  buf = []
  obj.each do |elem|
    buf << block.call(elem)
  end
  buf
end

ary = my_collect([1,2,3,4,5]) do |i|
  i * 2
end

p ary  #=> [2, 4, 6, 8, 10]
```

(2) 的参考答案

会得到下面的结果。

```
to_class = :class.to_proc
p to_class.call("test")    #=> String
p to_class.call(123)       #=> Fixnum
p to_class.call(2 ** 100)  #=> Bignum
```

(3) 的参考答案

累加在方法中定义的局部变量值。

```
def accumlator
  total = 0
  Proc.new do |x|
    total += x
  end
end

acc = accumlator
p acc.call(1)    #=> 1
p acc.call(2)    #=> 3
p acc.call(3)    #=> 6
p acc.call(4)    #=> 10
```