19 encoding编码

作者 bluetea
网站:https://github.com/bluetea

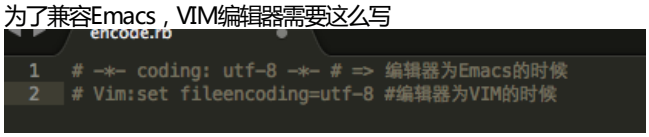
19.1
字符串的编码有多种，即使在同一个程序中，有时候输入和输出也可能不同
Ruby的每个字符串对象都包含“字符串数据本身” 以及 该“数据的字符编码”两个信息，其中，关于字符编码的信息即我们讲的编码

创建字符串一般2种方式
1.在脚本中直接以字面量的形式定义
2.从程序的外部获得（文件，控制台，网络等），数据的获取方式，决定了他的编码方式。 截取字符串的某部分，或者连接多个字符串生成的新字符串的时候，编码会继承原有的编码

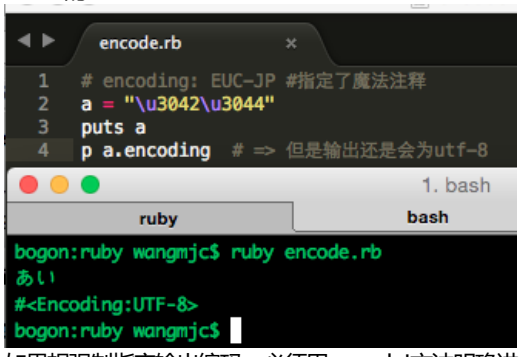
程序向外输出的时候，必须指定适当的编码，ruby会按照以下顺序决定字符串的编码。
1.脚本编码—决定字面量字符串对象编码的信息， 与脚本的字符编码一致
2.内部编码与外部编码—内部编码是从外部获取的数据在程序中如何处理的信息，与之相反，外部编码指程序向外部输出时与编码相关的信息， 两者都与IO对象有关联，
程序内部编码指的是中文字符在程序运行时在内存中的编码形式
程序外部编码的例子就是将中文存储到硬盘文件中选择的编码

19.2 脚本编码和魔法注释
脚本（源码）自身的编码成为 脚本编码（script encoding），脚本中的字符串，正则表达式的字面量都会根据脚本编码进行解释，脚本编码为UTF-8时，，那么字符串和正则表达也为UTF-8

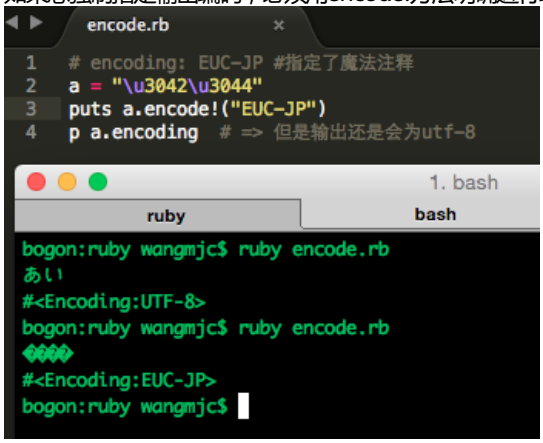
为了兼容Emacs，VIM编辑器需要这么写



在ruby 2.0后，如果没有魔法注释，默认为UTF-8的编码
特殊情况，有时候仅仅使用魔法注释是不够的，例如如果用\u特殊字符创建字符串后，及时脚本编码不是UTF-8,其生成的字符串也一定是UTF-8的



如果想强制指定输出编码，必须用encode!方法明确进行编码转换



这样才是你想要的编码类型呢

19.3 encoding类
String#encoding 实例方法来调查字符串的编码， 返回Econding对象

```
[9] pry(main)> "小红果".encoding
=> #<Encoding:UTF-8>
```

使用String#encode方法可以转换字符串对象的编码

```
1  str = "中国人"
2  p str.encoding
3  str2 = str.encode("GB2312")
4  p str2.encoding
```

```
ruby
bogon:ruby wangmjc$ ruby encode.rb
#<Encoding:UTF-8>
#<Encoding:GB2312>
bogon:ruby wangmjc$
```

当你想要连接一个字符串的时候，ruby会自动检查，如果编码格式不同，是会报错的

```
encode.rb
1  str1 = "中国人".encode("UTF-8")
2
3  str2 = "外国人".encode("GBK")
4
5  str3 = str1 + str2
```

```
1. bash
ruby  bash  bash
bogon:ruby wangmjc$ ruby encode.rb
encode.rb:5:in `<main>': incompatible character encodings: UTF-8 and GBK (Encodi
ng::CompatibilityError)
bogon:ruby wangmjc$
```

所以为了防止错误，在连接字符之前，必须用encode方法等把2者转换为相同的编码。
还有在进行比较是，即使表面一样，如果编码不一样，也会认为不一样

```
1  str1 = "中国人".encode("UTF-8")
2
3  str2 = "中国人".encode("GBK")
4  p "不一样" unless str1 == str2
```

```
1. bash
ruby  bash
bogon:ruby wangmjc$ ruby encode.rb
"不一样"
bogon:ruby wangmjc$
```

Encoding类的方法
Encoding.compatible?(str1, str2) 检查两个字符的兼容性，是否可以连接。
1.可兼容则返回连接后的字符的编码
2.不可兼容则返回nil

```
[17] pry(main)> Encoding.compatible?("china".encode("utf-8"), "自强".encode("GBK
"))
=> #<Encoding:GBK>
[18] pry(main)> Encoding.compatible?("中国人".encode("utf-8"), "自强".encode("GB
[18] pry(main)> Encoding.compatible?("中国人".encode("utf-8"), "自强".encode("GB
=> nil
```

Encoding.default_extrenal
返回默认的外部编码，这个值会影响IO类的外部编码
Encoding.default_internal
返回默认的内部编码，这个值会影响IO类的内部编码
Encoding.find(name)
返回编码名name对应的Encoding对象， 预定义的编码名由不含空格的英文字母，数字与符号构成，查找编码是不去人name的大小写

```
[23] pry(main)> Encoding.default_external
=> #<Encoding:UTF-8>
[24] pry(main)> Encoding.default_internal
=> nil
[25] pry(main)> Encoding.find("gbk")
=> #<Encoding:GBK>
[26] pry(main)> Encoding.find("gb")
ArgumentError: unknown encoding name - gb
from (pry):16:in `find'
[27] pry(main)> Encoding.find("gb2312")
=> #<Encoding:GB2312>
[28] pry(main)> Encoding.find("utf-16")
=> #<Encoding:UTF-16 (dummy)>
[29] pry(main)> Encoding.find("utf-8")
=> #<Encoding:UTF-8>
```

特殊的编码名：

1.locale 根据本地信息决定的编码
2.external 默认的外部编码
3.internal 默认的内部编码
4.filesystem 文件系统的编码

```
[31] pry(main)> Encoding.find("external")
=> #<Encoding:UTF-8>
[32] pry(main)> Encoding.find("internal")
=> nil
[33] pry(main)> Encoding.find("local")
ArgumentError: unknown encoding name - local
from (pry):23:in `find'
[34] pry(main)> Encoding.find("locale")
=> #<Encoding:UTF-8>
[35] pry(main)> Encoding.find("filesystem")
=> #<Encoding:UTF-8>
```

Encoding.list 返回的是Encoding对象一览表

```
[36] pry(main)> Encoding.list
=> [#<Encoding:ASCII-8BIT>,
 #<Encoding:UTF-8>,
 #<Encoding:US-ASCII>,
 #<Encoding:Big5>,
 #<Encoding:Big5-HKSCS>,
 #<Encoding:Big5-UAO>,
 #<Encoding:CP949>,
 #<Encoding:Emacs-Mule>,
 #<Encoding:EUC-JP>,
 #<Encoding:EUC-KR>,
```

Encoding.name_list 返回的是表示编码名的字符串 一览表

```
[37] pry(main)> Encoding.name_list
=> ["ASCII-8BIT",
 "UTF-8",
 "US-ASCII",
 "Big5",
 "Big5-HKSCS",
 "Big5-UAO",
```

返回ruby支持的编码一览表
enc.name
返回Encoding对象enc的编码名

```
[2] pry(main)> en = "中国人".encoding
=> #<Encoding:UTF-8>
[3] pry(main)> en.name
=> "UTF-8"
```

enc.names 有些编码有多个名称，这个方法返回包含Encoding对象名称的一览表，为数组，只要是这个里面返回的，都可以通过Encoding.find方法检索使用

```
[4] pry(main)> en.names
=> ["UTF-8", "CP65001", "locale", "external", "filesystem"]
```

注意：
ASCII-8bit与字节串
ASCII-8bit 是一个特殊的编码，被用于二进制非文本以及字节串，所以也成为BINARY
此外，把字符串对象用字节串的形式保存的时候，也会用到这个编码， Array#pack方法将二进制数据生成字符串的时候，或者Marsha1.dump方法将对象序列化后的数据生成字符串的时候，都会使用该编码
Array#pack方法，把IP地址的4个数值转换为4个字节的字节串

```
[39] pry(main)> str = [127,0,0,1]
=> [127, 0, 0, 1]
[40] pry(main)> a =str.pack("c4")
=> "\x7F\x00\x00\x01"
[41] pry(main)> a.encoding
=> #<Encoding:ASCII-8BIT>
```

pack方法的参数为字节串化时使用的模式，c4表示4个8位的不带符号的整数，执行结果是4个字节的字节串，编码为ASCII-8BIT

unpack方法的用途是将读取的字串根据制定的格式拆开，例如：

```
[1] pry(main)> a = ["a", "b", "c"]
=> ["a", "b", "c"]
[2] pry(main)> b = a.pack("")
=> ""
[3] pry(main)> b = a.pack("a3a4a5")
=> "a\x00\x00b\x00\x00\x00c\x00\x00\x00\x00"
[4] pry(main)> b
=> "a\x00\x00b\x00\x00\x00c\x00\x00\x00\x00"
[5] pry(main)> b.unpack("A3")
=> ["a"]
[6] pry(main)> b.unpack("A3A4")
=> ["a", "b"]
[7] pry(main)> b.unpack("A3A4A5")
=> ["a", "b", "c"]
[8] pry(main)> b.unpack("A3A4Z")
=> ["a", "b", "c"]
[9] pry(main)> b.unpack("A3A4")
=> ["a", "b"]
[10] pry(main)> b.unpack("A3A4Z")
=> ["a", "b", "c"]
```

string 的unpack方法

unpack(format) → anArray

Decodes *str* (which may contain binary data) according to the format string, returning an array of each value extracted. The format string consists of a sequence of single-character directives, summarized in the table at the end of this entry. Each directive may be followed by a number, indicating the number of times to repeat with this directive. An asterisk (“`*`”) will use up all remaining elements. The directives `sSiIlL`may each be followed by an underscore (“`_`”) or exclamation mark (“`!`”) to use the underlying platform’s native size for the specified type; otherwise, it uses a platform-independent consistent size. Spaces are ignored in the format string. See also`Array#pack`.

```
"abc \0\0abc \0\0".unpack('A6Z6')   #=> ["abc", "abc "]
"abc \0\0".unpack('a3a3')           #=> ["abc", " \000\000"]
"abc \0abc \0".unpack('Z*Z*')       #=> ["abc ", "abc "]
"aa".unpack('b8B8')                 #=> ["10000110", "01100001"]
"aaa".unpack('h2H2c')               #=> ["16", "61", 97]
"\xfe\xff\xfe\xff".unpack('sS')     #=> [-2, 65534]
"now=20is".unpack('M*')             #=> ["now is"]
"whole".unpack('xax2aX2aX1aX2a')    #=> ["h", "e", "l", "l", "o"]
```

This table summarizes the various formats and the Ruby classes returned by each.

```
Integer      |         |
Directive    | Returns | Meaning
-----------------------------------------------------------------
   C         | Integer | 8-bit unsigned (unsigned char)
   S         | Integer | 16-bit unsigned, native endian (uint16_t)
   L         | Integer | 32-bit unsigned, native endian (uint32_t)
   Q         | Integer | 64-bit unsigned, native endian (uint64_t)
             |         |
   c         | Integer | 8-bit signed (signed char)
   s         | Integer | 16-bit signed, native endian (int16_t)
   l         | Integer | 32-bit signed, native endian (int32_t)
   q         | Integer | 64-bit signed, native endian (int64_t)
             |         |
   S_, S!    | Integer | unsigned short, native endian
   I, I_, I! | Integer | unsigned int, native endian
   L_, L!    | Integer | unsigned long, native endian
   Q_, Q!    | Integer | unsigned long long, native endian (ArgumentError
             |         | if the platform has no long long type.)
             |         | (Q_ and Q! is available since Ruby 2.1.)
             |         |
   s_, s!    | Integer | signed short, native endian
   i, i_, i! | Integer | signed int, native endian
   l_, l!    | Integer | signed long, native endian
   q_, q!    | Integer | signed long long, native endian (ArgumentError
```

```
   q_/ q!   | Integer | signed long long, native endian (requires the compiler
            |         | if the platform has no long long type.)
            |         | (q_ and q! is available since Ruby 2.1.)
            |         |
   S> L> Q> | Integer | same as the directives without ">" except
   s> l> q> |         | big endian
   S!> I!>  |         | (available since Ruby 1.9.3)
   L!> Q!>  |         | "S>" is same as "n"
   s!> i!>  |         | "L>" is same as "N"
   l!> q!>  |         |
            |         |
   S< L< Q< | Integer | same as the directives without "<" except
   s< l< q< |         | little endian
   S!< I!<  |         | (available since Ruby 1.9.3)
   L!< Q!<  |         | "S<" is same as "v"
   s!< i!<  |         | "L<" is same as "V"
   l!< q!<  |         |
            |         |
   n        | Integer | 16-bit unsigned, network (big-endian) byte order
   N        | Integer | 32-bit unsigned, network (big-endian) byte order
   v        | Integer | 16-bit unsigned, VAX (little-endian) byte order
   V        | Integer | 32-bit unsigned, VAX (little-endian) byte order
            |         |
   U        | Integer | UTF-8 character
   w        | Integer | BER-compressed integer (see Array.pack)

Float       |         |
Directive   | Returns | Meaning
-----------------------------------------------------------------
   D, d     | Float   | double-precision, native format
   F, f     | Float   | single-precision, native format
   E        | Float   | double-precision, little-endian byte order
   e        | Float   | single-precision, little-endian byte order
   G        | Float   | double-precision, network (big-endian) byte order
   g        | Float   | single-precision, network (big-endian) byte order

String      |         |
Directive   | Returns | Meaning
-----------------------------------------------------------------
   A        | String  | arbitrary binary string (remove trailing nulls and ASCII spaces)
   a        | String  | arbitrary binary string
   Z        | String  | null-terminated string
   B        | String  | bit string (MSB first)
   b        | String  | bit string (LSB first)
   H        | String  | hex string (high nibble first)
   h        | String  | hex string (low nibble first)
   u        | String  | UU-encoded string
   M        | String  | quoted-printable, MIME encoding (see RFC2045)
   m        | String  | base64 encoded string (RFC 2045) (default)
            |         | base64 encoded string (RFC 4648) if followed by 0
   P        | String  | pointer to a structure (fixed-length string)
   p        | String  | pointer to a null-terminated string

Misc.       |         |
Directive   | Returns | Meaning
-----------------------------------------------------------------
   @        | ---     | skip to the offset given by the length argument
```

```
   x         | ---     | skip backward one byte
   x         | ---     | skip forward one byte
```

arry的pack方法：

pack (aTemplateString) → aBinaryString

click to toggle source

Packs the contents of *arr* into a binary sequence according to the directives in*aTemplateString* (see the table below) Directives “A,” “a,” and “Z” may be followed by a count, which gives the width of the resulting field. The remaining directives also may take a count, indicating the number of array elements to convert. If the count is an asterisk (“`*`”), all remaining array elements will be converted. Any of the directives “`sSiIlL`” may be followed by an underscore (“`_`”) or exclamation mark (“`!`”) to use the underlying platform’s native size for the specified type; otherwise, they use a platform-independent size. Spaces are ignored in the template string. See also `String#unpack`.

```
a = [ "a", "b", "c" ]
n = [ 65, 66, 67 ]
a.pack("A3A3A3")   #=> "a  b  c  "
a.pack("a3a3a3")   #=> "a\000\000b\000\000c\000\000"
n.pack("ccc")      #=> "ABC"
```

Directives for `pack`.

```
Integer      | Array   |
Directive    | Element | Meaning
---------------------------------------------------------------------------
   C         | Integer | 8-bit unsigned (unsigned char)
   S         | Integer | 16-bit unsigned, native endian (uint16_t)
   L         | Integer | 32-bit unsigned, native endian (uint32_t)
   Q         | Integer | 64-bit unsigned, native endian (uint64_t)
             |         |
   c         | Integer | 8-bit signed (signed char)
   s         | Integer | 16-bit signed, native endian (int16_t)
   l         | Integer | 32-bit signed, native endian (int32_t)
   q         | Integer | 64-bit signed, native endian (int64_t)
             |         |
   S_, S!    | Integer | unsigned short, native endian
   I, I_, I! | Integer | unsigned int, native endian
   L_, L!    | Integer | unsigned long, native endian
   Q_, Q!    | Integer | unsigned long long, native endian (ArgumentError
             |         | if the platform has no long long type.)
             |         | (Q_ and Q! is available since Ruby 2.1.)
             |         |
   s_, s!    | Integer | signed short, native endian
   i, i_, i! | Integer | signed int, native endian
   l_, l!    | Integer | signed long, native endian
   q_, q!    | Integer | signed long long, native endian (ArgumentError
             |         | if the platform has no long long type.)
             |         | (q_ and q! is available since Ruby 2.1.)
             |         |
   S> L> Q>  | Integer | same as the directives without ">" except
   s> l> q>  |         | big endian
   S!> I!>   |         | (available since Ruby 1.9.3)
   L!> Q!>   |         | "S>" is same as "n"
   s!> i!>   |         | "L>" is same as "N"
   l!> q!>   |         |
             |         |
   S< L< Q<  | Integer | same as the directives without "<" except
   s< l< q<  |         | little endian
   S!< I!<   |         | (available since Ruby 1.9.3)
   L!< Q!<   |         | "S<" is same as "v"
```

```
   s!< i!<   |         | "L<" is same as "V"
   l!< q!<   |         |
             |         |
   n         | Integer | 16-bit unsigned, network (big-endian) byte order
   N         | Integer | 32-bit unsigned, network (big-endian) byte order
   v         | Integer | 16-bit unsigned, VAX (little-endian) byte order
   V         | Integer | 32-bit unsigned, VAX (little-endian) byte order
             |         |
   U         | Integer | UTF-8 character
   w         | Integer | BER-compressed integer

Float        |         |
Directive    |         | Meaning
-----------------------------------------------------------------------------
   D, d      | Float   | double-precision, native format
   F, f      | Float   | single-precision, native format
   E         | Float   | double-precision, little-endian byte order
   e         | Float   | single-precision, little-endian byte order
   G         | Float   | double-precision, network (big-endian) byte order
   g         | Float   | single-precision, network (big-endian) byte order

String       |         |
Directive    |         | Meaning
-----------------------------------------------------------------------------
   A         | String  | arbitrary binary string (space padded, count is width)
   a         | String  | arbitrary binary string (null padded, count is width)
   Z         | String  | same as ``a'', except that null is added with *
   B         | String  | bit string (MSB first)
   b         | String  | bit string (LSB first)
   H         | String  | hex string (high nibble first)
   h         | String  | hex string (low nibble first)
   u         | String  | UU-encoded string
   M         | String  | quoted printable, MIME encoding (see RFC2045)
   m         | String  | base64 encoded string (see RFC 2045, count is width)
             |         | (if count is 0, no line feed are added, see RFC 4648)
   P         | String  | pointer to a structure (fixed-length string)
   p         | String  | pointer to a null-terminated string

Misc.        |         |
Directive    |         | Meaning
-----------------------------------------------------------------------------
   @         | ---     | moves to absolute position
   X         | ---     | back up a byte
   x         | ---     | null byte
```

在處理各種格式的檔案或是封包的時候，常常需要對bit level的東西做運算，這時候如果還利用character或是string慢慢在腦中轉換的話，真的很痛苦。剛好最近要做存取BMP格式圖片的東西…就順便看了一下String.unpack跟Array.pack這兩個method。

1. unpack

unpack這個method的用途是將讀入的字串依據指定的格式（Format）拆開，實際的用法有點複雜，先看這個例子：

```
 "ABC".unpack('CCC' ) # [65, 66, 67]
```

在這裡作為參數的Format是'CCC'。根據說明文件，'C'代表的是「取出一個字元，並視為unsigned integer」（extract a character as an unsigned integer），回傳的是Fixnum。所以三個'C'就代表要將字串拆成三個字元，並且用無號整數來表示。更實際的用法像是，Bitmap的File Header是最前面的14個byte。那我們就可以這樣讀取：

```
source = File.open(File_name, "rb" )
bitmap_header = source.read(14)
bitmap_header.unpack("a2LSSL" )
#['BM',61254,0,0,54]
```

在這裡的format中，a代表一個character，後面接著2代表兩次，也就是要讀出兩個字元。接著L代表「將連續四個character視為一個unsigned long integer。」S則是「將兩個連續的character是唯一個unsigned short integer。」因此，我們只要利用這個format做unpack之後，就可以正確的將這14個byte整理成我們要的資料了！

2. pack

pack其實就是unpack的相反動作，unpack是將字串依據format拆成陣列，pack則是將陣列(array)依據format組合成一個字串：

```
[65,66,67].pack('CCC' ) # "ABC"
```

詳細的format格式請參考API文件：unpack與pack。總之，當需要對byte層級，甚至是bit層級做處理的時候，不用自己慢慢算了，直接利用這兩個好用的method吧！

下面使用Array的pack方法，把IP地址的4个数值转换为4个字节的字节串，得到一个string的对像

```
[18] pry(main)> str = [127, 0, 0, 1]
=> [127, 0, 0, 1]
[19] pry(main)> str.pack("C4")
=> "\x7F\x00\x00\x01"
```

然后再转变回来

```
[23] pry(main)> a.unpack("C4")
=> [127, 0, 0, 1]
[24] pry(main)> a.unpack("C3")
=> [127, 0, 0]
[25] pry(main)> a.unpack("C*")
=> [127, 0, 0, 1]
```

此外在使用open-uri库等工具时，通过网络获取文件是，有时候并不知道字符串编码是什么，那么这个时候的编码也默认使用ASCII-8BIT

```
2.0.0-p598 :001 > require "open-uri"
 => true
2.0.0-p598 :002 > str = open("http://www.example.jp").read
 => "\n<html>\n<script language=javascript type=\"text/javascript\">\n    window
.location.replace(\"http://211.98.71.195:8080?HOST=\" \n        + location.hostn
ame + \"&R=\"\n        + location.pathname + \"&\" + location.search.substr(loca
tion.search.indexOf(\"\\?\")+1));\n</script>\n \n<noscript>\n<meta   http-equiv=
\"refresh\"    content=\"0;URL=http://211.98.71.195:8080\">\n</noscript>\n<head>\
n<title>Redirect</title>\n</head>\n<body bgcolor=\"white\" text=\"black\">\n</bo
dy>\n</html>\n"
2.0.0-p598 :003 > str.encoding
 => #<Encoding:ASCII-8BIT>
```

即使是编码ASCII-8BIT的字符串， 是加上也还是正常的字符串， 只要知道字符编码， 就可以使用force_encoding方法，这个方法并不会改变字符串的默认值（二进制数据）的值，而只是改变编码信息

```
[7] pry(main)> str = open("http://www.cnbetea.com").read(200)
=> "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\" \"http://ww
w.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">\r\n<html xmlns=\"http://www.w3
.org/1999/xhtml\">\r\n<head>\r\n<meta http-equiv=\"Conten"
[8] pry(main)> str.encoding
=> #<Encoding:ASCII-8BIT>
[9] pry(main)> str.force_encoding("UTF-8")
=> "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\" \"http://ww
w.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">\r\n<html xmlns=\"http://www.w3
.org/1999/xhtml\">\r\n<head>\r\n<meta http-equiv=\"Conten"
[10] pry(main)> str.encoding
=> #<Encoding:UTF-8>
```

使用force_encoding方法时，即使指定了不正确的编码，也不会马上产生错误，而是在对该字符串进行操作时才会产生错误，valid_encoding方法可以检查编码是否正确

```
[17] pry(main)> str = "中国人"
=> "中国人"
[18] pry(main)> str.encoding
=> #<Encoding:UTF-8>
[19] pry(main)> str.force_encoding("ASCII")
=> "\xE4\xB8\xAD\xE5\x9B\xBD\xE4\xBA\xBA"
[20] pry(main)> str.valid_encoding?
=> false
```

19.4正则表达式与编码

正则表达式要用正确的编码来匹配字符串，不然就会报错

```
[29] pry(main)> str = "中国人"
=> "中国人"
[30] pry(main)> str.force_encoding("GBK")
=> "\x{E4B8}\x{ADE5}\x{9BBD}\x{E4BA}\xBA"
[31] pry(main)> /国/ =~ str
ArgumentError: invalid byte sequence in GBK
from (pry):31:in `__pry__'
[32] pry(main)> /国/ =~ "中国人"
=> 1
```

通常情况下，正则表达式字面量的编码与代码的编码是一样的，指定其他编码时，可以是用Regexp类的new方法，在这个方法中，表示模式第1个参数的字符串编码，就是该正则表达式的编码

```
[43] pry(main)> r = Regexp.new("国".force_encoding("UTF-8"))
=> /国/
[44] pry(main)> r =~ "中国人"
=> 1
[45] pry(main)> r =~ "中国人".encode("GBK")
Encoding::CompatibilityError: incompatible encoding regexp match (UTF-8 regexp w
ith GBK string)
from (pry):45:in `=~'
```

19.5 IO类的编码

19.5.1 外部编码与内部编码

外部编码指的是作为输入，输出对象的文件， 控制台等的编码，

内部编码指的是Ruby脚本的编码，IO对象的编码相关方法如下：

IO#external_encoding 返回IO的外部编码

IO#internal_encoding 返回IO的内部编码

IO#set_encoding 设定IO的编码

```
encode.rb
1  p Encoding.default_external
2  p Encoding.default_internal
3    File.open("line1.txt") do |file|
4      p file.external_encoding
5      p file.internal_encoding
6    end
```

```
1. bash
ruby | bash
bogon:ruby wangmjc$
```

没有明确指定编码时，IO对象的外部编码和内部编码个子使用默认的Encoding.default_external和Encoding.default_internal, 默认情况下，外部编码会基于各个系统的本地信息设定

19.5.2 编码的设定

上面的例子打开了line1.txt文件， 但是IO对象的编码与文件内容没有关系的，编码对文本以外的文件没有多少作用

IO#seek 和 IO#read(size)等这些方法都不受编码印象，对任何数都可以进行读写操作， IO#read(size)方法读取的字符串的编码为表示二进制数据的 ASCII-8BIT

io.set_encoding(external_encoding: internal_encoding) 来设定编码

```
[66] pry(main)> $stdin.set_encoding("GBK:UTF-8")
=> #<IO:<STDIN>>
[67] pry(main)> $stdin.external_encoding
=> #<Encoding:GBK>
[68] pry(main)> $stdin.internal_encoding
=> #<Encoding:UTF-8>
```

File.open(file, "mode:encoding")

```
encode.rb
1  #指定外部编码为UTF-8
2  File.open("line1.txt", "a+:UTF-8") do |f|
3    f.write("hello")
4  end
```

```
1. bash
ruby | bash
bogon:ruby wangmjc$ ruby encode.rb
bogon:ruby wangmjc$ more line1.txt
line 1 line
    line 2 line line
            line 3 line line line
hellohello
bogon:ruby wangmjc$
```

指定外部编码为GBK，内部编码为UTF-8

```ruby
#指定外部编码为UTF-8
File.open("line2.txt", "r:GBK:UTF-8") do |f|
  p f.external_encoding
  p f.internal_encoding
  f.each do |i|
    puts i
  end
end
```

```
1. bash
ruby    bash
bogon:ruby wangmjc$ ruby encode.rb
#<Encoding:GBK>
#<Encoding:UTF-8>
line 1 line
    line 2 line line
            line 3 line line line
hellohello
bogon:ruby wangmjc$
```

19.5.3 编码的作用

输出编码的作用：

外部编码影响IO的输入输出， 在输出的时候， 会基于每个字符串的原有编码和IO对象的外部编码进行转换，（因此不需要指定内部编码）

如果没有设置外部编码，或者字符串的编码与外部编码一直，则不会进行编码的转换

注意，如果输出的原有字符串的编码不正确（比如是实际是日文字符串，但是编码却用中文）， 或者是无法互相转换的编码组合（例如用于日语与中文的编码），程序会抛出异常的

输入编码的作用：

IO读取（输入），

1.首先，如果外部编码没有设置，则会使用Encoding.default_external的值作为外部编码。

```ruby
#指定外部编码为UTF-8
File.open("line2.txt", "r") do |f|
  p Encoding.default_external
  p Encoding.default_internal
  p f.external_encoding
  p f.internal_encoding
  f.each do |i|
    puts i
  end
end
```

```
1. bash
ruby    bash
bogon:ruby wangmjc$ ruby encode.rb
#<Encoding:UTF-8>
nil
#<Encoding:UTF-8>
nil
line 1 line
    line 2 line line
            line 3 line line line
hellohello
bogon:ruby wangmjc$
```

2.如果设定了外部编码，内部编码没有设定时，则会将读取的字符串的编码设置为IO对象的外部编码（这样可以直接存在内部，当什么时候需要的时候在说），这种情况下不会进行编码的转换， 而且是将文件，控制台输入的数据原封不动的保存为string对象。

3.最后，外部编码和内部编码都设定的时候， 则会执行又外部编码转换为内部编码的处理，输入与输出的情况一样， 在编码过程中，如果数据格式或者编码组合不正确，程序都会抛出异常。

```ruby
#指定外部编码为UTF-8
File.open("line2.txt", "r:GBK:EUC-JP") do |f|
  p Encoding.default_external
  p Encoding.default_internal
  p f.external_encoding
  p f.internal_encoding
  f.each do |i|
    puts i
  end
end
```

```
1. bash
ruby | bash

bogon:ruby wangmjc$ ruby encode.rb
#<Encoding:UTF-8>
nil
#<Encoding:GBK>
#<Encoding:EUC-JP>
line 1 line
    line 2 line line
            line 3 line line line
hellohello
bogon:ruby wangmjc$
```