注意点

$\_:

在while(<>)中，$\_用来获取输入或文件中的每一行

在foreach(@array)中，$\_指的是@array中的每一个元素

如果有这种结构

while(<>)

{

print “$\_”; # 指某一行

foreach(split) #split不带参数，则以空白切分

{

print “$\_”; #指split切出来的单词列表中的单词

}

}

命令行

-w:打开警告

-p:-p会让perl自动生成一段一下类似的小程序

while(<>)

{

print;

}

-n:同-p，只不过会把print去掉

-i: 假设-i.bak，则将$^I 设为.bak

-e: 告知后面跟着的是程序代码，即要执行的程序，如果加了-p，即while里要做的事，如果加多个-e，就会有多段程序，但是只有最后一段程序的末尾分号可以省略。

自动变量

$\_: 在foreach(){}的花括号内，若不指定，在默认变量是$\_

@\_: 在子程序调用中，默认传过来的参数是@\_，若使用标量，则用@\_[0]的方法调用

$&: 在正则表达式中，字符串里实际匹配模式的部分会被自动存进这个变量中

$`: ……………………………………………………………………….之前的部分被存入这个变量

$’:……………………………………………………………………………之后的部分…………………………

$| = 1清除缓存 例如你写入文件一行 print {$file\_handler} "something"

这个时候你打开文件去看，something是不会在里面的。估计得close $file\_handler才会写入。

要是$|=1的话，文件会立马写入。

$!: 内置的错误报告

$$:当前运行的进程代号

$^I:配合钻石操作符，若将其赋值为某个字符串，则钻石操作符会将原文件附上这个字符串为扩展名，并复制原文件，具体见 p156

split不带参数的默认行为：它会以空白来切分$\_

日期

my $date = `date`; # date is: 当前日期: 2018/02/10 周六，还可以接受输入的日期。

my $date1 = localtime; # date1 is: Sat Feb 10 12:05:43 2018

perl正则表达式

1.若模式匹配的对象是$\_的内容，只要把模式卸载一对正斜线(/)中就可以了

例：

$\_ = “yabba dabba doo”;

if (/abba/)

{

print “Matched\n”;

}

2.所有在双引号圈引的字符串中能使用的技巧（尤其是反斜线转义），都可以在模式串中使用。因此，/cake\tsprite/这个模式会匹配cake、一个制表符和sprite这11个字符。

3.元字符

点号(.)是任何单字符的通配符（换行符除外），因此/bet.y/可以匹配betsy,betay,bet=y,bet,y等等，但是中间必须是一个字符，点号只能用来匹配一个字符。

如果想匹配点号本身，要加上反斜线，例如/3\.1415926/

3.量词

\*是用来匹配前面的内容零次或多次的

/fred\t\*barney/能匹配fred和barney中间有任意多个制表符的字符串，包括0次，但是不能出现其他字符，只能是前面的字符。

如果中间除了制表符还有其他字符，可以用

.\*，.\*会匹配任意字符无限次，。

+ 也是一个量词，加号会匹配前一个条目一次以上：/fred +barney/只会匹配在fred和barney之间用空格隔开且只用空格隔开的字符串，中间的空格至少是一个。

第三个量词是问号(?)，表示前一个条目是可有可无的。也就是说，它的前一个条目可以出现一次或者不出现，它前面的内容只能出现一次或者不出现。

# 通用量词

# 除了可以使用\*、+、?作为特定量词外，还可以在花括号中指定重复次数的范围

# 例 /a{5,15}/表示a重复出现5-15次

# /a{3,}/ 表示a重复三次以上

# /a{8}/表示a重复8次

4.（.）\1

表示需要匹配连续出现的两个同样的字符。

其中\1表示反向引用，\1是组的编号

例/(.)(.)/则第一个括号内的模式是\1，第二个是\2

实际上用\g{N}的方法可以去除二义性

这种引用方式叫做绝对引用，还有相对引用，使用负数

如\g{-1}

5.择一匹配（|）

读成或，意思是左边匹配或右边匹配都行，如果坐标匹配失败了，还可以试试右边。

因此/fred|barney|betty/能匹配任何含有三者之一或以上的字符串。

则/fred( |\t)+barney/可以匹配fred和barney之间空格、制表符或者两者组合出现一次以上的字符串。

6.字符集

字符集是指一串可能出现的字符集合，通过写在方括号[]内来表示，它只能匹配单个字符，但可以是字符集里列出的任何一个。

例如，使用[abcwxyz]来匹配7个字符中的任何一个。同时，为方便起见，还可以用连字符-，这样就可以写成[a-cw-z]。则[a-zA-Z]可以匹配大小字母中的任何一个。

可以在字符集内部开头的地方加上脱字符^来表示除字符集外的任何字符

7.字符集简写

有些字符集十分常用，因此有自己的简写

如[0-9]简写成\d，

\w表示[A-Za-z0-9\_]，数字，字母，下划线

它们都只会匹配一个字符

但是加上修饰符，\w就就可以匹配单词，\d匹配一串数字

例：/fred \w+ barney/

\s表示[\f\t\n\r]，也就是说\s擅长处理空白

同样它一般不单独使用，而是使用\s\*来匹配任何数目的空白

或者用\s+匹配一个以上的空白字符

perl 5.10后增加了更多字符集描述空白

\h能匹配横向空白，即包含制表符和空格的字符集[\t ]

\v能匹配纵向，即[\f\n\r]

\R能匹配任何类型的断行，这给跨操作系统断行匹配带来了便利

8.反义简写

以上几种的简写。

如[^\d]、[^\w]、[^\s]来表示非数字，非词或非空白字符，

其实它们的大写版本就是用来完成这种任务的。

即\D、\W、\S。这些能匹配相应小写范围以外的字符

这些简写既可以作为模式里独立的字符集，也可以作为方括号字符集的一部分。

例如，/[\dA-Fa-f]+/可以用来匹配十六进制数字。

另一个符合字符集是[\d\D]，表示任何数字或非数字。也就是说它会匹配任何字符（包括换行符），此外还有完全无用的[^\d\D]，它什么都不匹配。

## 可选修饰符

可选修饰符

这是一些可有可无的修饰字符，有时候称为开关。他们可以组成附加在某个正则表达式结尾的定界符的右边，并改变正则表达式的默认行为。

# 1.用/i来进行大小写无关的匹配

print "Would like to play a game?\n";

chomp($\_ = <STDIN>);

if (/yes/i) # 大小写无关的匹配

{

print "Ok!Let's go!\n";

}

# 2.用/s来匹配任意字符，它会将模式中的每个点号

# 按任意字符来处理

$\_ = "I saw Barney\ndown at the bowling alley\nwith fred\nlast night.\n";

if (/Barney.\*fred/s)

{

print "That string mentions fred after Barney!\n";

}

else

{

print "Not good!\n";

}

# 3.用/x加入空白

# /x能在模式里面随意加空白，目的是使它更容易阅读、理解

# 例 /-?\d+\.?\d\*/ 都挤在一起，容易混淆

# / -? \d+ \.? \d\* /x 加入空白，稍微清晰一些

# 由于加上/x后模式里可以任意插入空白，所以原始的空白与制表符就失去了

# 意义，他们会被忽略掉，如果还想匹配空白与制表符，就得在前面补上一个反斜线字符

# 不过，\s \s\* \s+还是比较常见的匹配空白字符的写法

$\_ = "What a amazing things\n";

if(/What \s .\* amazing /x)

{

print "It works!\n";

}

# 4.组合选项修饰符

# 如果一个模式中使用多个修饰符，可将它们连在一起使用

# 它们之间的先后顺序并不影响匹配的结果

$\_ = "Barney is\n the best freind of Fred\n";

if(/barney .\* fred/isx)

{

print "It also works!\n";

}

## 锚位

默认情况下，模式匹配过程开始于待匹配字符串的开头，如果不相符就一直往字符串后面浮动，看其他位置能否匹配。但是加入一些锚位，可以让模式直接匹配字符串的某处。

脱字符（^）是一个锚位，用来表示字符的开头

美元符号（$）也是一个锚位，用来表示字符串的结尾

因此，/^fred/只匹配位于字符串最前端的fred，如果是manfred mann这个字符串，则不能匹配。而/rock$/只能匹配字符串的最后。

有时候，这两个锚位会一起使用，以确保匹配整个字符串。

例/^\s\*$/，这个模式可以匹配空白行，这个空白行可能包含若干空格或制表符。