Functions and Methods

Functions and Methods
函数和方法

☺ *Agent Zhang（章亦春）* ☺

2006.10

☆ *Consistent* interface, *Simple* interface, *rich* interface

一致的接口，简单的接口，丰富的接口

C# seems to have *hundreds* of *different* collection classes, used **inconsistently** in the .NET libraries.

-- Ned Batchelder

C# 似乎拥有成百上千个不同的集合类，它们在 .NET 类库中的使用是如此的不一致。

Perl 6's *array*

➨ *Simple* interface with *rich* functionalities

Perl 6 数组：简单的接口，丰富的功能。

```
# use it as an ordinary array:
```

```
# use it as an ordinary array:
@item = 'dog', 'cat', 'mouse', 'tiger';
```

```
# use it as an ordinary array:
@item  =  'dog', 'cat', 'mouse', 'tiger';
@item[0].say;        # prints 'dog'
```

```
# use it as an ordinary array:
@item  = 'dog', 'cat', 'mouse', 'tiger';
@item[0].say;          # prints 'dog'
 say ~@item[1..-1];   # prints 'cat mouse tiger'
```

```
# use it as an ordinary array:
@item  = 'dog', 'cat', 'mouse', 'tiger';
@item[0].say;          # prints 'dog'
 say ~@item[1..-1];   # prints 'cat mouse tiger'
@item[2] = 'human';
```

```
# use it as an ordinary array:
@item  = 'dog', 'cat', 'mouse', 'tiger';
@item[0].say;         # prints 'dog'
 say ~@item[1..-1];   # prints 'cat mouse tiger'
@item[2] = 'human';
@item.push('camel', 'moose');  # append elements
```

```
# use it as an ordinary array:
@item  = 'dog', 'cat', 'mouse', 'tiger';
@item[0].say;       # prints 'dog'
 say ~@item[1..-1];   # prints 'cat mouse tiger'
@item[2] = 'human';
@item.push('camel', 'moose');  # append elements
@item  = (@item, 'camel', 'moose');   # just the same
```

```
# use it as a stack (FILO):
```

```
# use it as a stack (FILO):
@item.push('moose');
```

```
# use it as a stack (FILO):
@item .push( 'moose' );
 push @item , 'elk' ;  # just another way
```

```
# use it as a stack (FILO):
@item .push('moose');
 push @item , 'elk';   # just another way
$top = @item[-1];
```

```
# use it as a stack (FILO):
@item .push( 'moose' );
 push @item , 'elk' ;   # just another way
$top = @item [-1];
$top = @item .pop();
```

```
# use it as a stack (FILO):
@item.push('moose');
 push @item, 'elk';   # just another way
$top = @item[-1];
$top = @item.pop();
$top = @item.pop;   # ditto
```

```
# use it as a queue (FIFO):
@item.unshift(256);
```

```
# use it as a queue (FIFO):
@item.unshift(256);
 unshift @item, 128, 'hello';
```

```
# use it as a queue (FIFO):
@item.unshift(256);
 unshift @item, 128, 'hello';
$elem = @item.shift();
```

```
# use it as a queue (FIFO):
@item.unshift(256);
 unshift @item, 128, 'hello';
$elem = @item.shift();
$elem = @item.shift;
```

```
# use it as a queue (FIFO):
@item.unshift(256);
 unshift @item, 128, 'hello';
$elem = @item.shift();
$elem = @item.shift;
$elem = shift @item;
```

```
# use it as a queue (FIFO):
@item .unshift( 256 );
 unshift @item , 128 , 'hello' ;
$elem = @item .shift();
$elem = @item .shift;
$elem = shift @item ;
 say "length of the queue: ", @elem .elems;
```

Make simple things *easy* and hard things *possible*.

-- Larry Wall

让简单的事情很容易办到，
让困难的事情有可能办到。

Writing a string to a file should be a *simple* task, however...

向一个文件写入一个字符串本该是一件很简单的任务，可是……

```
# The Java way:
import  java.io.*;
class  WriteFile {
     public static void  main(String args[]) {
         FileOutputStream foStream;
         PrintStream pStream;
         try  {
             foStream =  new  FileOutputStream( "somefile.txt" );
             pStream  =  new  PrintStream( foStream );
             pStream.println ( "This is written to a file" );
             pStream.close();
         }
          catch  (Exception e) {
             System.err.println ( "Error writing to file "  + e);
         }
     }
}
```

```
 /* The C way: */
 #include <stdio.h>
 #include <stdlib.h>

 int main() {
    FILE* fh;
     if ((fh = fopen("somefile.txt", "w") ) == NULL) {
        fprintf(stderr, "Can't open file: %s",
            strerror(errno));
        return 1;
    }
    fprintf(fh, "This is written to a file\n");
    fclose(fh);
    return 0;
}
```

```
 # The Perl 5 way:
open my $fh, "> somefile.txt"
    or die "Can't open file: $!";
print $fh "This is written to a file\n"
    or die "Can't print to file: $!";
close $fh;
```

☆ *output arguments* <=> *return values*

输出参数 <=> 返回值

```
/* pass results via return values */
int  add( int  a,  int  b) {
    return  a + b;
}
...
if  (add(1, 2) == 3) {
    printf("ok");
}
printf("%d", add(5, 6));
```

```
/* pass results via output arguments */
void add( int a, int b, int& c) {
    c = a + b;
}
...
int temp;
add(1, 2, temp);
if (temp == 3) {
    printf("ok");
 }
add(5, 6, temp);
printf("%d", temp);
```

☆ *Multiple* return values

多重返回值

```
# The Perl 6 way:
sub div ( $a , $b ) {
    return ( $a / $b , $a % $b );
 }
 ...
my ( $quo , $rem ) = div(5, 2);
```

```
 /* The C way: */
 void  div ( int  a,  int  b,  int * ptr_quo,  int * ptr_rem) {
    *ptr_quo = a / b, *ptr_rem = a % b;
}
...
 int  quo, rem;
div(5, 2, &quo, &rem);
```

```
// The C++ way:
void div (int a, int b, int & quo, int & rem) {
    quo = a / b; rem = a % b;
}
...
int quo, rem;
div(5, 2, quo, rem);
```

☆ The *power* of notation

记法的威力

Inventing a language doesn't necessarily mean building the successor to Java; often a thorny problem can be *cleared* up by a change of notation...

-- "The Practice of Programming"

发明一种语言并不一定意味着创建 Java 的继承者；通常一个很棘手的问题可以通过记法上的改变而获得澄清。

-- 《程序设计实践》

Case #1: Formated output

案例 #1: 格式化输出

```
// The C++ style:
cout << "{ list[" << i << "][" << j
     << "] = " << elems[i][j] << " }\n";
```

```
// The C++ style:
cout << "{ list[" << i << "][" << j
     << "] = " << elems[i][j] << " }\n";
```

☹ This is *ugly*.

```
// The C style:
printf("{ list[%d][%d] = %f }\n",
          i, j, elems[i][j]);
```

```
// The C style:
printf("{ list[%d][%d] = %f }\n",
        i, j, elems[i][j]);
```

☺ This is *much better*.

The problem is that many developers choose the solution *before* defining the problem. It's not the case that any programming language is "one size fits all".

-- Ovid

问题就在于许多开发人员在定义问题之前就选定了解决方案。并不存在一种编程语言能做到“一劳永逸”。

Case #2 : A simple-minded CSV file *parser* in Perl 6

案例 #2: 一个简单的逗号分隔(CSV)文件的解析器（使用 Perl 6）

```
my  $csv_src  = slurp $csv_file;
my  @lines  = split /[\n\s]+/, $csv_src;
for  @lines  -> $line {
    my  @fields  = split /\s*,\s*/, $line;
    # process the fields here...
 }
```

☺ Job *done*!

♨

In large applications covering problem domains suitable for *both* Perl and Java, the Java programmer *can't* hold a candle to me.

-- Ovid

对于 Perl 和 Java 都适合的大型应用，
Java 程序员根本无法赶上我的编程效率。

☆ *Higher* order functions (λ calculus)

高阶函数（λ 演算）

C# 3.0 (“C# Orcas”) introduces several language extensions that build on C# 2.0 to support the creation and use of *higher order*, functional style class libraries. The extensions enable construction of *compositional* APIs that have equal *expressive power* of query languages in domains such as relational databases and XML.

-- C# Version 3.0 Specification

Closures

➥ A piece of code *manipulable* in *arbitrary* contexts

```
my $foo = { say 1 + 2 };
my $bar = $foo;
$bar.();   # prints 3
$foo.();   # ditto
```

☺ Closures can *remember* the context in which it was created.

```
sub factory( $seed ) {
    return { say $seed };
}
my $foo = factory(7);
$foo.();   # prints 7
my $bar = factory(100);
$bar.();   # prints 100
$foo.();   # still prints 7
```

Higher order functions

➥ o *loops*

高阶函数

➥ o 循环

Case #1 : We want to print out *all* the elements contained in an array of arrays .

案例 #1： 我们想打印出一个
数组的数组里的所有元素。

```
 // The C# way (using loops):
 int [][] elems =  new int [][] {
     new int [] {1,2},  new int [] {3,4,5}
};
 for  ( int  i = 0; i < elems.length; i++)
     for  ( int  j = 0; j < elems[i].length; j++)
         Console.WriteLine(elems[i][j]);
 # The Perl 6 way:
 my  @elems  = [1,2], [3,4,5];
map { map { .say }, $_  }, @elems ;
```

```
# Anyway, we can still use loops in Perl 6:
my @elems = [1,2], [3,4,5];
for @elems { for @$_ { .say } }
```

Case #2 : We want to *filter* out customers with income higher than $ 5000.

案例 #2: 我们想到筛选出所有收入在 5000 美元以上的客户

```
# Traditional way in Perl 6 (using loops):
my  @res ;
for  @customers  -> $customer  {
    if  $customer .income() > 5000 {
        push @res , $customer ;
    }
}
```

```
# As before, but more concise:
my @res;
for @customers {
    if .income > 5000 {
        push @res, $_;
    }
}
```

```
# A functional-style solution:
my  @res  =
     grep { .income > 5000 }, @customers ;
```

Higher order functions
➥ As *expressive* as SQL

高阶函数
➥ 拥有和 SQL 一样的表达力

Case #1 : Customer filtering and sorting

案例 #1: 客户筛选与排序

```sql
select *
from customers
where income > 5000 and gender = 'female'
order by name
```

```
my  @res  =
   sort {  $^a .name  cmp  $^b .name },
      grep { .income > 5000  and  .gender  eq  'female' },
            @customers ;
```

Case #2 : Boy student *statistics*

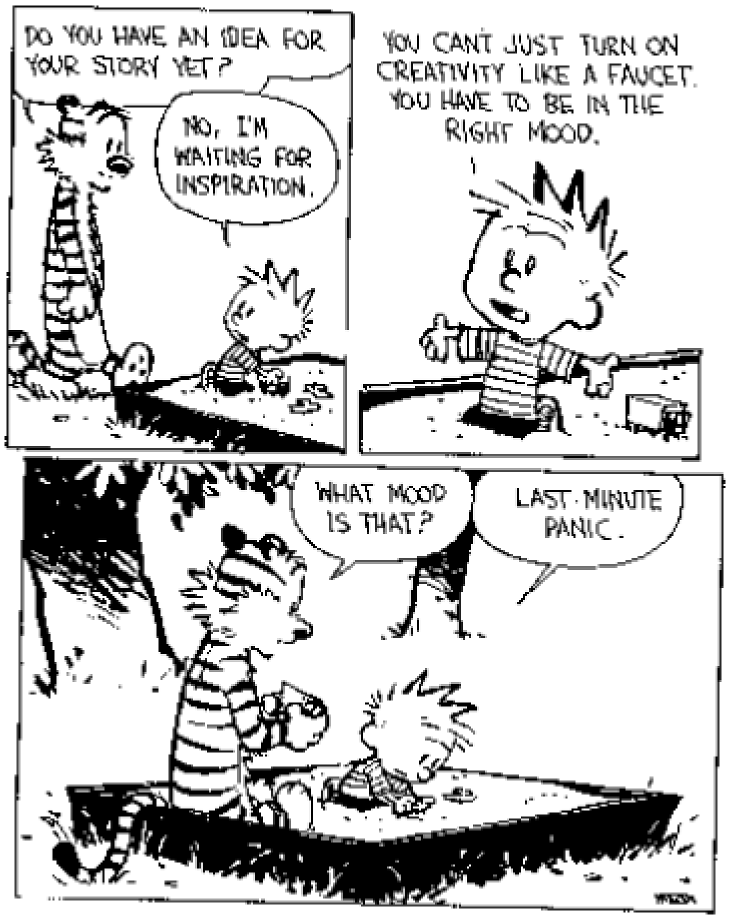
案例 #2: 男生统计信息

```sql
select class_id, count(*) as count
from students
where gender = 'male'
group by class_id
order by count desc
```

```
my  %class ;
 map {  %class {.class_id}++ },
     grep { .gender  eq  'male'  }, @student ;
my  @res  =
     reverse sort {  $^a .value  <=>  $^b .value },
                    %class .pairs;
```

JIT slide making...

即时幻灯片制做……


DO YOU HAVE AN IDEA FOR YOUR STORY YET?
NO, I'M WAITING FOR INSPIRATION.
YOU CAN'T JUST TURN ON CREATIVITY LIKE A FAUCET. YOU HAVE TO BE IN THE RIGHT MOOD.
WHAT MOOD IS THAT?
LAST-MINUTE PANIC.

Thank you!

☺