

The Art of Naming

命名的艺术

☺ *Agent Zhang* (章亦春) ☺

2006.9

Why naming *matters*?

为什么命名很重要？

Good hackers

➡ poets and wordsmiths

好的黑客

➡ 诗人和词汇大师

A *poetry* in Pugs' **Main.hs** \Rightarrow

Pugs 的 Main.hs 中的英文诗歌 \Rightarrow

A *ship* then new they built for *him*
Of *mithril* and of *elven-glass*
With shining prow; no shaven oar
Nor sail *she* bore on silver mast;
The Silmaril as *lantern light*
And banner bright with *living flame*
To gleam thereon by Elbereth
Herself was set, who thither came...

众人为彼造新舟，
铸以秘银精灵璃。
船首闪耀何需桨，
银桅未有风帆系。

无双宝钻作灯炬，
旗帜辉煌展生焰。
映照燃星雅碧绿，
神祇乘梭下九天。

-- 唐风

Perl 6 *translation* \Rightarrow

翻译到 Perl 6 编程语言 \Rightarrow

```
use v6 ;  
for $*Larry {  
  our Ship $pugs .= new (:of<mithril elven-glass> );  
  given $pugs {  
    $.prow does Shine;  
    Silver $.mast but none (Oar::Shaven, Sail);  
    Light $.lantern := $*Silmaril ;  
    Bright $.banner := Flame.bless:{};  
    when $*Elbereth .gleam {  
      .sail(...);  
    }  
  }  
}
```


Top programmers take **naming** issues
very seriously

顶级的程序员对命名问题
非常认真。

as seen on the *#perl6* channel
of **irc.freenode.net**...

正如在 irc.freenode.net 上的 #perl6 通道里
所看见的那样.....



Larry Wall

<agentzh> TimToady++ # JIT syn fixes

<TimToady> thanks

<TimToady> I looked at *several hundred* words
before picking START.

<章亦春> (Larry Wall)++ # 即时 Perl 6 文档修改

<Larry Wall> 谢谢

<Larry Wall> 我在挑选出 START 这个名字之前查看了
*几百个*单词。



Audrey Tang

<gaal> defer? slothlike

<audreyt> I like "defer"

<audreyt> gaal++

* audreyt deletes Data::Thunk and uploads Data::Defer

<audreyt> ...and it's now Scalar::Defer

<audreyt> *naming* takes *more* time than tests+doc+code

<高浪> defer 怎么样? 有些惰性的味道

<唐凤> 我喜欢“defer”

<唐凤> 高浪++

* 唐凤删除了 Data::Thunk 并上传了 Data::Defer

<唐凤>现在它叫做 Scalar::Defer 了

<唐凤> *命名*所花费的时间比测试 + 文档 + 实现代码 *还要多*

☆ Follow the **naming convention** of the language *you* are using and try to be consistent.

遵循你所使用的语言的命名约定，
并努力保持一致性。

The problem with being **consistent** is that there are **lots** of ways to be consistent, and they're all **inconsistent** with each other.

-- *Larry Wall*

保持一致性的困难在于存在许许多多不同方式来实现一致性，而且它们彼此之间都是不一致的。

-- *Larry Wall*


```
// C++ STL naming style  
vector<string> list;  
list.push_back("hello");  
while (!list.empty()) {...}
```

```
// Java naming style  
List list = new ArrayList();  
list.add("hello");  
while (!list.isEmpty()) {...}
```

// C# naming style

ArrayList myAL = new **ArrayList**();

myAL.*Add*("hello");

Console.*WriteLine*(myAL.*ToString*());

☆ Use *meaningful* names

使用 *有意义* 的名字

```
? Button button1 = new Button("New");  
?  
? Button button2 = new Button("Open");  
?  
? Button button3 = new Button("Save");
```

? Button **button1** = new Button("New");
? Button **button2** = new Button("Open");
? Button **button3** = new Button("Save");

☹ This is *bad*.

? Button **button_New** = new Button("New");

? Button **button_Open** = new Button("Open");

? Button **button_Save** = new Button("Save");

```
? Button button_New    = new Button("New");  
?  
? Button button_Open  = new Button("Open");  
?  
? Button button_Save  = new Button("Save");
```

☹ This is *ugly*.


```
Button btnNew = new Button("New");  
Button btnOpen = new Button("Open");  
Button btnSave = new Button("Save");
```

```
Button btnNew = new Button("New");  
Button btnOpen = new Button("Open");  
Button btnSave = new Button("Save");
```

😊 This is *good*.

The Ugly, the Bad, and the Good

丑的，坏的，好的

☆ Choose *nouns* for
your **class** names

从名词中为你的类取名



```
class Evaluator { ... }  
class Solver { ... }  
class Withdrawal { ... }
```

```
class Evaluator { ... }  
class Solver { ... }  
class Withdrawal { ... }
```

😊 These are *good*.

? class Evaluate { ... }

? class Solve { ... }

? class Withdraw { ... }

? class Evaluate { ... }

? class Solve { ... }

? class Withdraw { ... }

☹ These are *bad*.

The following class/module names are
also good:

package CGI::Simple;

package XML::Smart;

package Class::DBI::Sweet;

package Test::Easy;

☆ Choose active *verbs* for
your **method** names

从主动动词中为你的方法取名



```
$dbh = DBI. connect ( 'dbi:odbc:qqbase' );  
$dbh. commit ();  
$account. update (balance => 100);
```

```
$dbh = DBI.connect ('dbi:odbc:qqbase');  
$dbh.commit ();  
$account.update (balance => 100);
```

☺ These are *good*.

```
? $dbh = DBI.<b>connection</b>('dbi:odbc:qqbase');  
?  
? $dbh.<b>committing</b>();  
? $account.<b>updated</b>(balance => 100);
```

```
? $dbh = DBI.connection ('dbi:odbc:qqbase');  
?  
? $dbh.committing ();  
? $account.updated (balance => 100);
```

☹ These are *bad*.

☆ Choose *nouns* or *adjectives* for
your **property** names

从名词或者形容词中为你的属性取名

```
@list.length  
$persion.name  
if $dbh.available { ... }  
if $set.is_empty;
```

```
@list.length  
$persion.name  
if $dbh.available { ... }  
if $set.is_empty;
```

☺ These are *good*.

☆ Use **namespaces** to split your *verbose* class names.

使用命名空间来分割你冗长的类名。

```
? class Makefile_Parser_AST_Element {  
?  
?     ...  
?  
? }  
?  
? Makefile_Parser_AST_Element elem();
```

```
? class Makefile_Parser_AST_Element {  
?  
?     ...  
?  
? }  
?  
? Makefile_Parser_AST_Element elem();
```

☹ This is *ugly*.

```
namespace Makefile::Parser::AST {  
    class Element { ... }  
}  
Makefile::Parser::AST::Element elem();
```

```
namespace Makefile::Parser::AST {  
    class Element { ... }  
}  
Makefile::Parser::AST::Element elem();
```

😊 This is *good*.

☆ Use **short** names for *common* things.

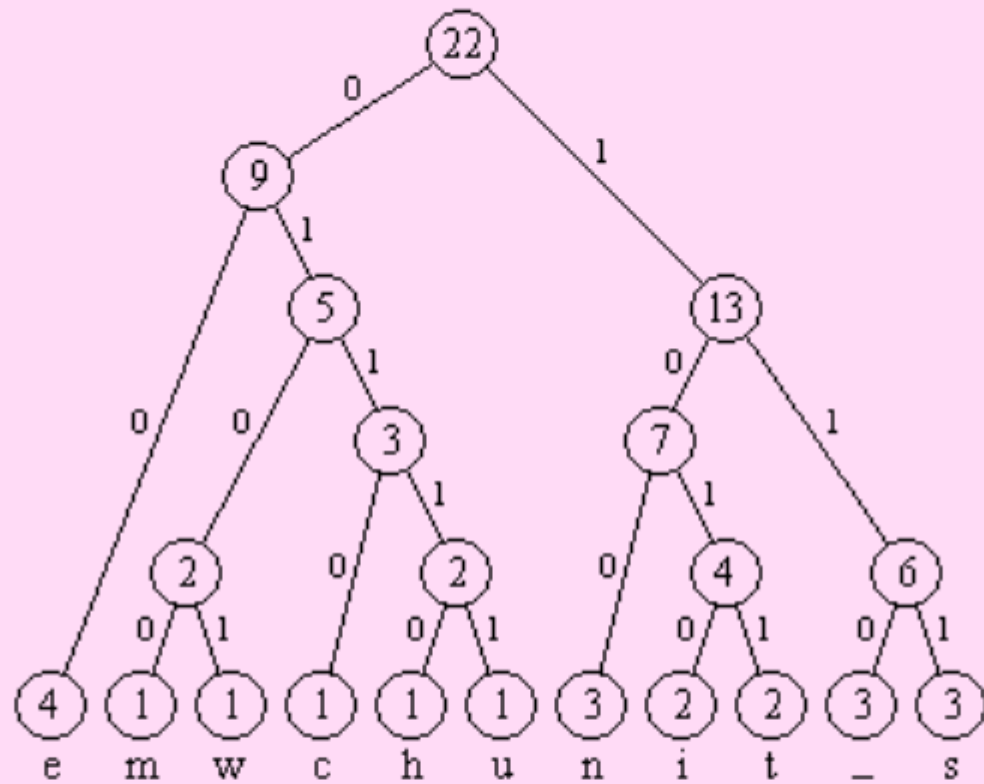
为常用的东西取短名字。

Common operations should be "**Huffman coded**".
That is, frequently used operators should be
shorter than infrequently used ones.

-- *Larry Wall* (Perl 6 Apocalypse 3)

对常见的操作应该进行“哈夫曼编码”。也就是说，频繁使用的运算符的名称应该比那些不经常使用的要短。

-- *Larry Wall* (Perl 6 启示录 3)



Huffman Coding

// The Java way:

```
System.out.println("Computing the sum...");  
for (int i = 1; i <= n; i++) {  
    sum += i;  
    System.out.println(i);  
}  
System.out.println("Sum: " + sum);
```

// The Java way:

```
System.out.println("Computing the sum...");  
for (int i = 1; i <= n; i++) {  
    sum += i;  
    System.out.println(i);  
}  
System.out.println("Sum: " + sum);
```

☹ This is *ugly*.

The Perl 6 way:

```
say "Computing the sum...";
```

```
for 1..$n -> $i {
```

```
    $sum += $i;
```

```
    say $i;
```

```
}
```

```
say "Sum: $sum";
```

The Perl 6 way:

```
say "Computing the sum...";  
for 1..$n -> $i {  
    $sum += $i;  
    say $i;  
}  
say "Sum: $sum";
```

😊 This is *good*.

☆ Use **short** names for *locals*.

为局部变量取短名字。


```
int Len = list.length;  
for (int i = 0; i < Len; i++)  
    System.out.println(list[i]);
```

```
int Len = list.length;  
for (int i = 0; i < Len; i++)  
    System.out.println(list[i]);
```

😊 This is *good*.

```
int Length = list.length;  
for ( int index = 0; index < Length; index++)  
    System.out.println(list[index]);
```

```
int Length = list.length;  
for ( int index = 0; index < Length; index++)  
    System.out.println(list[ index ]);
```

☺ This is *ugly*.

☆ Use **descriptive** names for *globals*.
(long names are okay.)

为全局结构取描述性的名字。
(长名字也可以。)

```
? # Globals in smartlinks.pl:  
? my ($count, $broken_count);
```

```
? # Globals in smartlinks.pl:  
? my ($count, $broken_count);
```

☹ This is *bad*.

```
# Globals in smartlinks.pl:
```

```
my ($link_count, $broken_link_count);
```



```
# Globals in smartlinks.pl:
```

```
my ($link_count, $broken_link_count);
```

😊 This is *better*.

☆ Don't *repeat* yourself.
(The **DRY** principle)

不要重复你自己
(DRY 原则)

? // The Java way:

? **double** value = **Double**.parse**Double**("3.14");

? // The Java way:

? **double** value = **Double**.parse**Double**("3.14");

☹ This is *ugly*.

// The C# way:

```
double value = double.Parse("3.14");
```

// The C# way:

```
double value = double.Parse("3.14");
```

☹ Just a bit better.

// The Perl 6 way:

```
my Double $value = "3.14";
```

// The Perl 6 way:

```
my Double $value = "3.14";
```

☺ This is *good*!

// The C/C++ way:

```
double value = atof("3.14");
```

// The C/C++ way:

```
double value = atof("3.14");
```

😊 This is *good* too,
though a bit fuzzy.

```
? class UserQueue {  
?     int noOfItemsIn Q;  
?     int frontOfTheQueue;  
?     int queueCapacity;  
?     public int noOfUsersInQueue () { ... }  
? }
```

```
? class UserQueue {  
?     int noOfItemsIn Q;  
?     int frontOfTheQueue;  
?     int queueCapacity;  
?     public int noOfUsersInQueue () { ... }  
? }
```

☹ This is *ugly*.

```
class UserQueue {  
    int nitems;  
    int front;  
    int capacity;  
    public int nusers() { ... }  
}
```

```
class UserQueue {  
    int nitems;  
    int front;  
    int capacity;  
    public int nusers() { ... }  
}
```

☺ This is *good*!

☆ Tell me *more*!
(Don't be **handwaving**.)

多告诉我一些！
(不要遮遮掩掩。)

? `if (check_even(num)) { ... }`

? `if (check_even(num)) { ... }`

☹ Check if it *is* an even number?

? `if (check_even(num)) { ... }`

- ☹ Check if it *is* an even number?
- ☹ Check if it *is not* an even number?

```
if (is_even(num)) { ... }
```

```
if (is_even(num)) { ... }
```

😊 Now it is *good*!

☆ Improper **abbreviations** can be
very confusing.

不恰当的缩写名可能会让人
非常迷糊。

<Sal> what is "JQL"?

<clkao> Jabberwocky Query Language?

<TimToady> Just Quack Loudly?

<audreyt> Junctional Quantum Library?

<仲伟祥> 什么是“JQL”？

<高嘉良> 废话查询语言？

<Larry Wall> 不过是高声吹嘘？

<唐凤> 联结性量子库？

★ *Widely-used* abbreviations
are recommended.

提倡选择
广泛使用的缩写。

var ← variable
val ← value
init ← initialize / initialization
elem ← element
id ← identifier
len ← length
eval ← evaluate / evaluation
func ← function
sub ← subroutine
AST ← Abstract Syntax Tree

... and many more

☆ Chinese **Pinyin abbreviations** can be
extremely hateful.

汉语拼音缩写会非常令人讨厌。

http://.../ **xs**_main.aspx? **xh**=3030602110#

http://.../ **xscj_gc**.aspx? **xh**=3030602110& **xm**=章亦春

http://.../ **xscxbm**.aspx? **xh**=3030602110& **xm**=章亦春

http://.../ **xs**_main.aspx? **xh**=3030602110#

http://.../ **xscj_gc**.aspx? **xh**=3030602110& **xm**=章亦春

http://.../ **xscxbm**.aspx? **xh**=3030602110& **xm**=章亦春

xs == 学生 (student) ?

http://.../ **xs**_main.aspx? **xh**=3030602110#

http://.../ **xscj_gc**.aspx? **xh**=3030602110&**xm**=章亦春

http://.../ **xscxbm**.aspx? **xh**=3030602110&**xm**=章亦春

xs == 学生 (student) ?

xh == 学号 (student id) ?

http://.../ **xs**_main.aspx? **xh**=3030602110#

http://.../ **xscj_gc**.aspx? **xh**=3030602110& **xm**=章亦春

http://.../ **xscxbm**.aspx? **xh**=3030602110& **xm**=章亦春

xs == 学生 (student) ?

xh == 学号 (student id) ?

xm == 姓名 (name) ?

http://.../ **xs**_main.aspx? **xh**=3030602110#

http://.../ **xscj**_gc.aspx? **xh**=3030602110& **xm**=章亦春

http://.../ **xscxbm**.aspx? **xh**=3030602110& **xm**=章亦春

xs == 学生 (student) ?

xh == 学号 (student id) ?

xm == 姓名 (name) ?

xscj == 学生成绩 (student grades) ?

http://.../ **xs**_main.aspx? **xh**=3030602110#

http://.../ **xscj_gc**.aspx? **xh**=3030602110& **xm**=章亦春

http://.../ **xscxbm**.aspx? **xh**=3030602110& **xm**=章亦春

xs == 学生 (student) ?

xh == 学号 (student id) ?

xm == 姓名 (name) ?

xscj == 学生成绩 (student grades) ?

xscxbm == 学生重修报名 (XXX) ?

http://.../ **xs**_main.aspx? **xh**=3030602110#

http://.../ **xscj_gc**.aspx? **xh**=3030602110& **xm**=章亦春

http://.../ **xscxbm**.aspx? **xh**=3030602110& **xm**=章亦春

xs == 学生 (student) ?

xh == 学号 (student id) ?

xm == 姓名 (name) ?

xscj == 学生成绩 (student grades) ?

xscxbm == 学生重修报名 (XXX) ?

gc == ???

I'm a **Chinese**,
but I still find it *hard* to understand.

我是中国人，
但我发现理解它们仍然很困难。

☆ Avoid potential *ambiguity* and *confusion*.

避免可能的歧义和混淆。

To get the number of elements in an array, use the `.elems` method. You can also ask for the total string length of an array's elements, in bytes, codepoints or graphemes, using these methods `.bytes`, `.codes` or `.graphs` respectively on the array. The same methods apply to strings as well.

There is no `.length` method for either arrays or strings, because length does *not* specify a **unit**.

-- *Larry Wall* (The Perl 6 Synopsis 2)

想得到数组中的元素数目，可以使用 *.elems* 方法。
你也可以利用 *.bytes*、*.codes* 或者 *.graphs* 分别
以字节、编码点、或者字形为单位来查询数组元素
的总长度。字符串也拥有这些方法。

对数组和字符串而言，都不再有 *.length* 方法了，
因为 `length` 并*没有*指明*长度单位*。

-- *Larry Wall* (Perl 6 纲要 2)

☆ DWIM? Do what *you* mean!

言行一致！

```
? class Array {  
?     // print out the contents:  
?     public void init() {  
?         for (int i = 0; i < this.length; i++)  
?             System.out.println(this.get(i));  
?     }  
? }
```

```
? class Array {  
?     // print out the contents:  
?     public void init() {  
?         for (int i = 0; i < this.length; i++)  
?             System.out.println(this.get(i));  
?     }  
? }
```

init? **emit?**

Get the **slides** today!



<http://perlcabal.org/agent/slides/naming/naming.xul>

<http://perlcabal.org/agent/slides/naming/naming.ppt>

<http://perlcabal.org/agent/slides/naming/naming.pdf>

You need **Firefox** to access the .xul link above.

Contact me on the *web*!



agentzh@gmail.com

These slides are *powered* by
Sporx and 😁*Takahashi++*

Thank you!

