

火龙果•整理

[uml.org.cn](http://uml.org.cn)



# 良好代码从命名开始

李哲

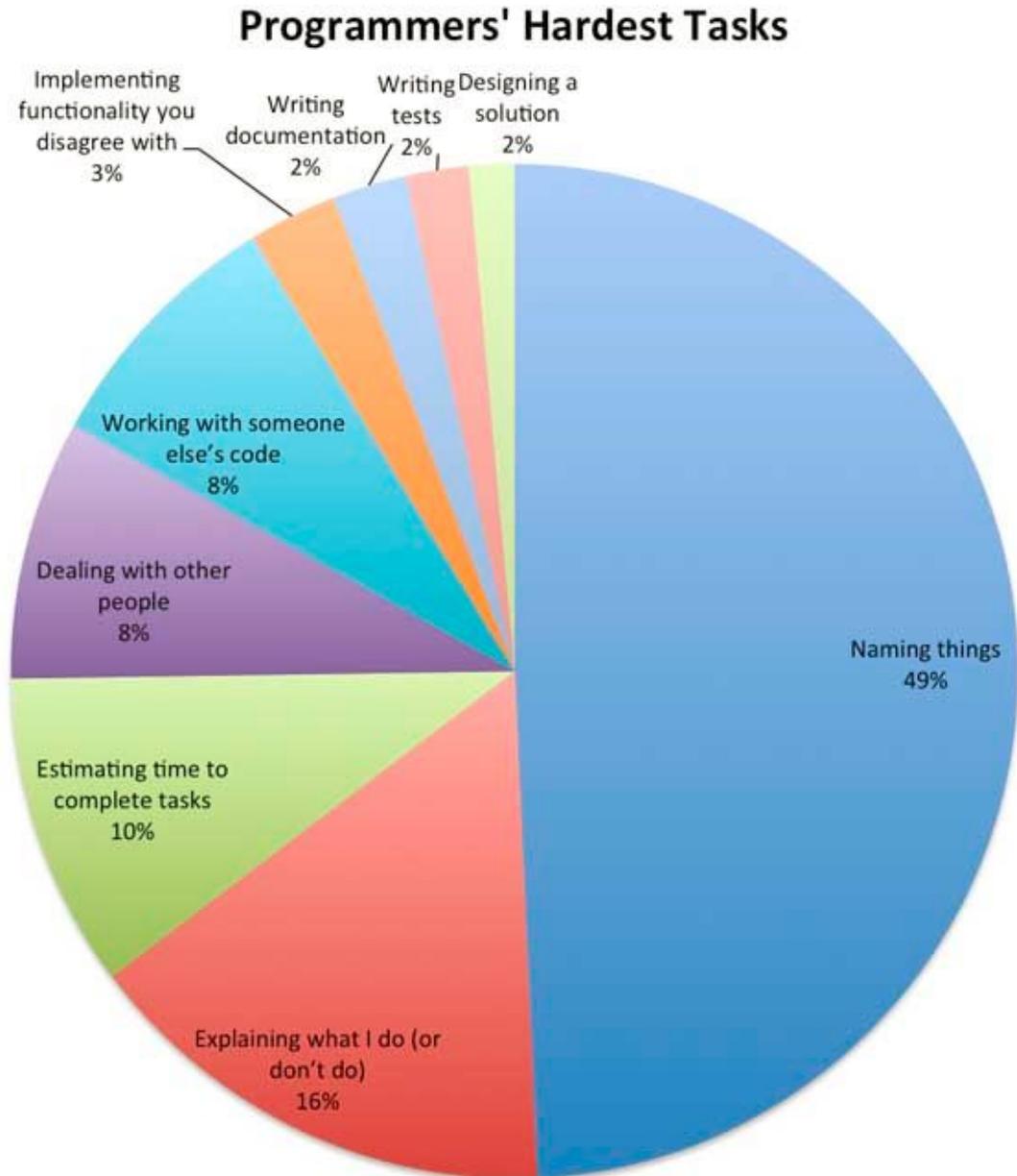


勿忘初心



# 程序员最难的事

- [链接](#)
- [讨论](#)



Data Source: Quora/Ubuntu Forums  
Total Votes: 4,522

# 参考书目



火龙果•整理  
[uml.org.cn](http://uml.org.cn)





## 哪种更容易阅读？

```
public List<int[]> getThem() {  
    List<int[]> list1 = new ArrayList<int[]>();  
    for (int[] x : theList)  
        if (x[0] == 4)  
            list1.add(x);  
    return list1;  
}
```

```
public List<int[]> getFlaggedCells() {  
    List<int[]> flaggedCells = new ArrayList<int[]>();  
    for (int[] cell : gameBoard)  
        if (cell[STATUS_VALUE] == FLAGGED)  
            flaggedCells.add(cell);  
    return flaggedCells;  
}
```

## 选择专业的词



- `def GetPage(url)` VS `FetchPage()`, `DownloadPage()`
- `BinaryTree.size()` VS `Height()`, `NumNodes()`,  
`MemoryBytes()`
- `Thread.stop()` VS `kill()` 或 `pause()`



选择专业的词。

**send** deliver dispatch(派遣) announce(宣布) distribute(分发)  
route (按某路线发送)

**find** search(搜索) extract(提取) locate(找寻位置)

**start** launch create begin open

**make** create set up build generate compose add new

- 不要使用双关语  
**add** (通常指将两个现存的值增加或链接来获得新值)

如果用于添加到某个集合中，最好使用insert或者append



## 避免像**tmp**和**retval**这样泛泛的名字

```
var euclidean_norm = function (v) {  
    var retval = 0.0;  
    for (var i = 0; i < v.length; i += 1)  
        retval += v[i] * v[i];  
    return Math.sqrt(retval);  
};
```

例如，想象如果循环的内部被意外写成：

```
retval += v[i];
```

如果名字换成sum\_squares这个缺陷就会更明显：

```
sum_squares += v[i]; //我们要累加的"square"在哪里？缺陷！
```



## tmp 临时变量

```
if (right < left) {  
    tmp = right;  
    right = left;  
  
    left = tmp;  
}
```

上面特别简单的逻辑求最大值  
某种程度上可以接受 tmp

tmp应当只是名称的一部分

```
tmp_file = tempfile.NamedTemporaryFile()  
...  
SaveData(tmp_file, ...)
```

```
String tmp = user.name();  
tmp += " " + user.phone_number();  
tmp += " " + user.email();  
...  
template.set("user_info", tmp);
```

上面输出个人信息的字符串  
就不该用tmp了

VS

```
SaveData(tmp, ...)
```



## 循环迭代器

下面代码有何问题？

```
for (int i = 0; i < clubs.size(); i++)  
    for (int j = 0; j < clubs[i].members.size(); j++)  
        for (int k = 0; k < users.size(); k++)  
  
            if (clubs[i].members[k] == users[j])  
                cout << "user[" << j << "] is in club[" << i << "]" << endl;
```

- 可以使用clubs\_i, members\_j, users\_k
- 或者简写 ci, mj, uk
- 对于特别简单的循环可以采用i,j,k,如果嵌套层次较多，逻辑比较复杂，则不建议使用i,j,k的命名方式

```
if (clubs[ci].members[ui] == users[mi]) #缺陷! 第一个字母不匹配。
```



## 对于泛泛而谈的名称的裁定

---

### 建议

如果你要使用像tmp、it或者retval这样空泛的名字，那么你要有个好的理由。



## 给布尔值命名（状态值）

通常来讲，加上像is、has、can或should这样的词，可以把布尔值变得更明确。

最后，最好避免使用反义名字。例如，不要用：

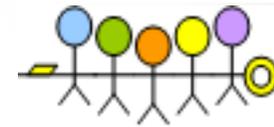
```
bool disable_ssl = false;
```

而更简单易读（而且更紧凑）的表示方式是：

```
bool use_ssl = true;
```

谨记典型的布尔变量名 下面是一些格外有用的布尔变量名。

- *done* 用 `done` 表示某件事情已经完成。这一变量可用于表示循环结束或者一些其他的操作已完成。在事情完成之前把 `done` 设为 `false`，在事情完成之后把它设为 `true`。
- *error* 用 `error` 表示有错误发生。在错误发生之前把变量值设为 `false`，在错误已经发生时把它设为 `true`。
- *found* 用 `found` 来表明某个值已经找到了。在还没有找到该值的时候把 `found` 设为 `false`，一旦找到该值就把 `found` 设为 `true`。在一个数组中查找某个值，在文件中搜寻某员工的 ID，在一沓支票中寻找某张特定金额的支票等等的时候，都可以用 `found`。
- *success* 或 *ok* 用 `success` 或 `ok` 来表明一项操作是否成功。在操作失败的时候把该变量设为 `false`，在操作成功的时候把其设为 `true`。如果可以，请用一个更具体的名字代替 `success`，以便更具体地描述成功的含义。如果完成处理就表示这个程序执行成功，那么或许你应该用 `processingComplete` 来取而代之。如果找到某个值就是程序执行成功，那么你也应该换用 `found`。



## 为状态变量命名

```
if ( flag ) ...  
if ( statusFlag & 0x0F ) ...  
if ( printFlag == 16 ) ...  
if ( computeFlag == 0 ) ...  
  
flag = 0x1;  
statusFlag = 0x80;  
printFlag = 16;  
computeFlag = 0;
```

不好的命名

```
if ( dataReady ) ...  
if ( characterType & PRINTABLE_CHAR ) ...  
if ( reportType == ReportType_Annual ) ...  
if ( recalcNeeded == false ) ...  
  
dataReady = true;  
characterType = CONTROL_CHARACTER;  
reportType = ReportType_Annual;  
recalcNeeded = false;
```

良好的命名



## 为状态变量命名

- 标记应该用枚举类型、具名常量、或用作具名常量的全局变量来赋值

### 在C++中声明状态变量

```
// values for CharacterType
const int LETTER = 0x01;
const int DIGIT = 0x02;
const int PUNCTUATION = 0x04;
const int LINE_DRAW = 0x08;
```

```
const int PRINTABLE_CHAR = ( LETTER | DIGIT | PUNCTUATION | LINE_DRAW );
```

```
const int CONTROL_CHARACTER = 0x80;
```

```
// values for ReportType
enum ReportType {
    ReportType_Daily,
    ReportType_Monthly,
    ReportType_Quarterly,
    ReportType_Annual,
    ReportType_All
};
```



## 枚举类型

Visual Basic示例：为枚举类型采用前缀命名约定

```
Public Enum Color  
    Color_Red  
    Color_Green  
    Color_Blue  
End Enum
```

```
Public Enum Planet  
    Planet_Earth  
    Planet_Mars  
    Planet_Venus  
End Enum
```

```
Public Enum Month  
    Month_January  
    Month_February  
    ...  
    Month_December  
End Enum
```

- 在有些编程语言中，使用枚举类型方式是 `Color.Color_Red` 这种，则没有必要使用前缀，直接使用 `Color.Red`

## 常量

---

- FIVE (too bad)
- CYCLES\_NEEDED





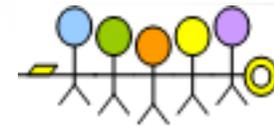
## 用具体的名字代替抽象的名字

- ServerCanStart VS CanListenOnPort

- DISALLOW\_EVIL\_CONSTRUCTORS

```
#define DISALLOW_EVIL_CONSTRUCTORS(ClassName) \  
    ClassName(const ClassName&); \  
    void operator=(const ClassName&);
```

- VS
- DISALLOW\_COPY\_AND\_ASSIGN



## 用具体的名字代替抽象的名字

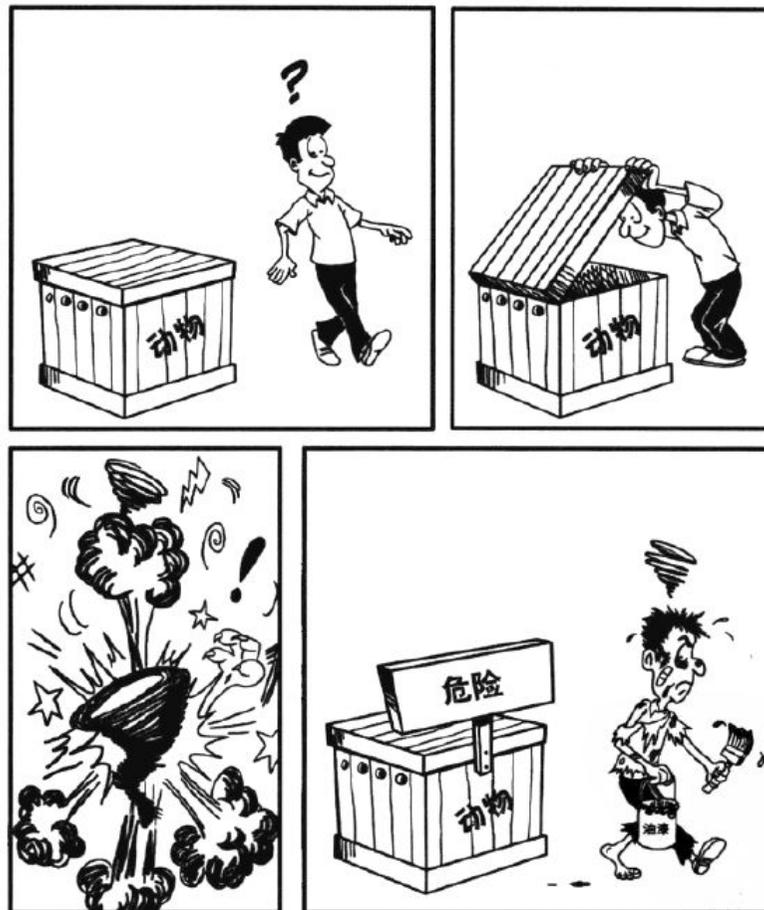
- run-locally
- 场景
  - 这个命令行标志用于让程序输出额外的调试信息（会导致运行变慢一般用于本地机器测试）
- 问题
  - 团队里的新成员不知道它到底是做什么的，可能在本地运行时使用它（想象一下），但不明白为什么需要它。
  - 偶尔，我们在远程运行这个程序时也要输出调试信息。向一个运行在远端的程序传递--run\_locally看上去很滑稽，而且很让人迷惑。
  - 有时我们可能要在本地运行性能测试，这时我们不想让日志把它拖慢，所以我们不会使用--run\_locally。
- 解决方案
- 使用 --extra\_logging这样的名字

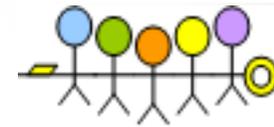


## 用具体的名字代替抽象的名字

- 如果 `run-locally` 的用途比增加日志还有其他用途怎么办?
  - 比如使用一个特殊的本地数据库
  - 建议创建一个新的标志-- `use-local-database`
  - 尽管标志多了，但是这两个标志作用明确，不会混淆这两个正交的含义，可以明确地选择使用

# 为名字附带更多信息





## 为名字附带更多信息

---

```
string id; // Example: "af84ef845cd8"
```

- 建议使用 `hex_id`



## 带单位的值

- 如果该变量是个可以度量的值的话，最好带上单位

```
var start = (new Date()).getTime(); // top of the page
...
var elapsed = (new Date()).getTime() - start; // bottom of the page
document.writeln("Load time was: " + elapsed + " seconds");
```

```
var start_ms = (new Date()).getTime(); // top of the page
...
var elapsed_ms = (new Date()).getTime() - start_ms; // bottom of the page
document.writeln("Load time was: " + elapsed_ms / 1000 + " seconds");
```

函数参数	带单位的参数
Start(int <b>delay</b> )	delay → <b>delay_secs</b>
CreateCache(int <b>size</b> )	size → <b>size_mb</b>
ThrottleDownload(float <b>limit</b> )	limit → <b>max_kbps</b>
Rotate(float <b>angle</b> )	angle → <b>degrees_cw</b>



## 附带其他重要属性

- 比如 `untrustedUrl` `unsafeMessageBody`
- `trustedUrl` `safeMessageBody`

情形	变量名	更好的名字
一个“纯文本”格式的密码，需要加密后才能进一步使用	<code>password</code>	<code>plaintext_password</code>
一条用户提供的注释，需要转义之后才能用于显示	<code>comment</code>	<code>unescaped_comment</code>
已转化为UTF-8格式的html字节	<code>html</code>	<code>html_utf8</code>
以“url方式编码”的输入数据	<code>data</code>	<code>data_urlenc</code>



## 名字的长度

- 名字该有多长  
两个极端

```
newNavigationControllerWrappingViewControllerForDataSourceOfClass
```

d, y, s, e

- 在小的作用域里可以使用短的名字。
- 缩写  
不建议使用项目特有的缩写词  
原则是：团队新成员能否理解该词的含义
- 丢掉没用的词  
函数： ConvertToString DoServerLoop  
ToString ServerLoop



## 缩写

首字母缩写要视意思的表达效果而定. BEManager是另人费解的，而 FormatStr()人们可能会理解它的意思

从某种程度上说，要求使用短变量名是早期计算的遗留物。早期语言，如汇编、一般的 Basic 和 Fortran 都把变量名的长度限制在 2 到 8 个字符，并要求程序员创建简短的名字。早期的计算科学更多的同数学联系在一起，并大量使用求和及其他等式中的  $i$ 、 $j$  和  $k$  等符号。而在现代语言如 C++、Java 和 Visual Basic 里面，实际上你可以创建任何长度的名字；几乎没有任何理由去缩短具有丰富含义的名字。



`firstName`、`lastName`、`street`、`houseNumber`、`city`、`state` 和 `zipcode` 的变量。

`addrFirstName`、`addrLastName`、`addrState` 等，以此提供语境

更好的方案是创建 `Address` 类

- 去掉无用的内容

类 `GSDAccountAddress`

对于实体而言 `accountAddress`，`customerAddress` 都是不错的名称，作为类名就不太好了。`Address` 本身就已经是个好类名了。

如果需要和 `MAC` 地址 端口地址，`Web` 地址相区别，我会使用 `PostalAddress`，`MAC`，`URI`



## 接口和实现（仅供参考）

许多人对于接口采用 大写字母I 开头的方式

`ISharpFactory`

如果必须采用一个的话，宁愿选择`Impl` 作为实现的后缀

`SharpFactoryImpl`



## 成员前缀

- **Clean Code**
  - 不必采用m\_前缀来标明成员变量。
  - 类和函数应该足够小，消除对前缀的需要。
- **Code Complete**
  - 要根据名字识别出变量是类的成员变量。

## 不会误解的名字

- **key:**  
要多问自己几遍：“这个名字会被别人解读成其他的含义吗？”。

- 例子：Filter()  
假设你在写一段操作数据库结果的代码：  
`results = Database.all_objects.filter("year <= 2011")`

结果现在包含哪些信息？

- 年份小于或等于2011的对象？
- 年份不小于或等于2011年的对象？

这里的问题是“Filter”是个二义性单词。我们不清楚它的含义到底是“挑出”还是“减掉”。最好避免使用“filter”这个名字，因为它太容易误解。

- 推荐用 min 和 max 来表示（包含）极限假设你的购物车应用程序最多不能超过10 件物品：

```
CART_TOO_BIG_LIMIT = 10
```

```
if shopping_cart.num_items() >= CART_TOO_BIG_LIMIT :
```

```
    Error( "Too many items in cart." )
```

这段代码有个经典的“大小差一”缺陷。我们可以简单地通过把  $\geq$  变成  $>$  来改正它：

```
if shopping_cart.num_items() > CART_TOO_BIG_LIMIT :
```

或者通过把 `CART_TOO_BIG_LIMIT` 变成11,但问题的根源在于 `CART_TOO_BIG_LIMIT` 是个二义性名字，它的含义到底是“少于”还是“少于且包括”。

## 建议

命名极限最清楚的方式是在要限制的东西前加上 `max_` 或者 `min_`。

- 总结

**不会误解的名字是最好的名字**——阅读你代码的人应该理解你的本意，并且不会有其他的理解。遗憾的是，很多英语单词在用来编程时是多义性的，例如filter、length和limit。你决定使用一个名字以前，要吹毛求疵一点，来想象一下你的名字会被误解成什么。最好的名字是不会误解的。

当要定义一个值的上限或下限时，max\_和min\_是很好的前缀。对于包含的范围，first和last是好的选择。对于包含排除范围，begin和end是最好的选择，因为它们最常用。

当为布尔值命名时，使用is和has这样的词来明确表示它是个布尔值，避免使用反义的词（例如disable\_ssl）。

要小心用户对特定词的期望。例如，用户会期望get()或者size()是轻量的方法。



## 不要使用让人误解和混淆的名字

- l (小写L) I(大写i) 和1
- 0 和O
  
- 2和Z
- 5和S
- G和6



## 不要使用让人误解和混淆的名字

- 

别用 `accountList` 来指称一组账号，除非它真的是 `List` 类型。`List` 一词对程序员有特殊意义。如果包纳账号的容器并非真是个 `List`，就会引起错误的判断<sup>3</sup>。所以，用 `accountGroup` 或 `bunchOfAccounts`，甚至直接用 `accounts` 都会好一些。

提防使用不同之处较小的名称。想区分模块中某处的 `XYZControllerForEfficientHandlingOfStrings` 和另一处的 `XYZControllerForEfficientStorageOfStrings`，会花多长时间呢？这两个词外形实在太相似了。



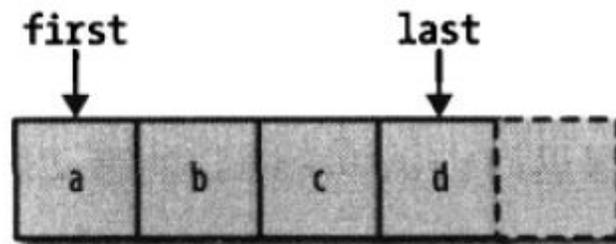
## 做有意义的区分

```
public static void copyChars(char a1[], char a2[]) {  
    for (int i = 0; i < a1.length; i++) {  
        a2[i] = a1[i];  
    }  
}
```

如果参数名改为 `source` 和 `destination`，这个函数就会像样许多。



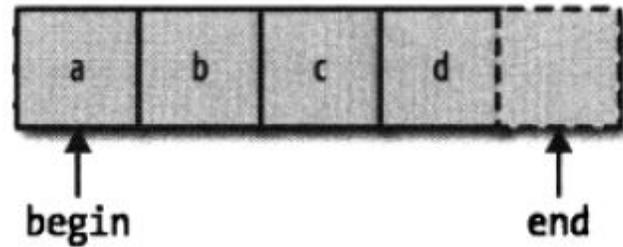
## 推荐使用**first** 和**last**来表示包含的范围



```
print integer_range(start=2, stop=4)
# Does this print [2,3] or [2,3,4] (or something else)?
```

```
set.PrintKeys(first="Bart", last="Maggie")
```

## 推荐使用**begin**和**end**来表示包含/ 排除的范围



- 比如打印发生在10月16号的事件

```
PrintEventsInRange("OCT 16 12:00am", "OCT 17 12:00am")
```

```
PrintEventsInRange("OCT 16 12:00am", "OCT 16 11:59:59.9999pm")
```



## 其他常见对仗词

---

- begin/end
- first/last
- locked/unlocked
- min/max
- next/previous
- old/new
- opened/closed
- visible/invisible
- source/target
- source/destination
- up/down

## 常见对仗词



**add/remove**

**begin/end**

**create/destroy**

**first/last**

**get/put**

**get/set**

**increment/decrement**

**insert/delete**

**lock/unlock**

**min/max**

**next/previous**

**old/new**

**open/close**

**show/hide**

**source/target**

**start/stop**

**up/down**



## 使用解决方案领域名称

- 比如访问者模式 中使用vistor。
- 队列使用Queue

## 利用命名规则

- 类：首字母大写的驼峰式
- 比如函数用首字母大（小）写驼峰式
- 变量用首字母小写的驼峰式或者全小写的蛇形式
- 常量用全大写的蛇形式
- 等等

### 方案1：通过大写字母开头区分类型和变量

```
Widget widget;  
LongerWidget longerWidget;
```

### 方案2：通过全部大写区分类型和变量

```
WIDGET widget;  
LONGERWIDGET longerWidget
```

### 方案3：通过给类型加“t\_”前缀区分类型和变量

```
t_Widget Widget;  
t_LongerWidget LongerWidget;
```

### 方案4：通过给变量加“a”前缀区分类型和变量

```
Widget aWidget;  
LongerWidget aLongerWidget;
```

### 方案5：通过对变量采用更明确的名字区分类型和变量

```
Widget employeeWidget;  
LongerWidget fullEmployeeWidget;
```



## java的规则

### Java 的规则

与 C 和 C++ 不同，Java 语言的风格约定从一开始就创建好了。

- `i` 和 `j` 是整数下标。
- 常量全部大写 (ALL\_CAPS) 并用下画线分隔。
- 类名和接口名中每一个单词的首字母均大写，包括第一个单词——例如，`ClassOrInterfaceName`。
- 变量名和方法名中第一个单词的首字母小写，后续单词的首字母大写——例如，`variableOrRoutineName`。
- 除用于全部大写的名字之外，不使用下画线作为名字中的分隔符。
- 访问器子程序使用 `get` 和 `set` 前缀。