void 类型和 void 指针

注：在实际使用中，只有在修饰函数时才会用 void，而 void *指针通常也只是在内存管理的时候才会用得到。总之，都不常用。了解一下，有利于后期进阶。

void 的含义

void 即“无类型”，void *则为“无类型指针”，可以指向任何数据类型。

void 指针使用规范

①void 指针可以指向任意类型的数据，亦即可用任意数据类型的指针对 void 指针赋值。例如：

```
int * pint;
void *pvoid;
pvoid = pint; /* 不过不能 pint= pvoid; */
```

如果要将 pvoid 赋给其他类型指针，则需要强制类型转换如：pint= (int *)pvoid;

②在 ANSIC 标准中，不允许对 void 指针进行算术运算如 pvoid++或 pvoid+=1 等，而在 GNU 中则允许，因为在缺省情况下，GNU 认为 void *与 char *一样。sizeof(*pvoid)== sizeof(char).

void 的作用

①对函数返回的限定。

②对函数参数的限定。

当函数不需要返回值时，必须使用 void 限定。例如： void func(int, int);

当函数不允许接受参数时，必须使用 void 限定。例如： int func(void)。

由于 void 指针可以指向任意类型的数据，亦即可用任意数据类型的指针对 void 指针赋值，因此还可以用 void 指针来作为函数形参，这样函数就可以接受任意数据类型的指针作为参数。例如：

```
void * memcpy( void *dest, const void *src, size_t len );
void * memset( void * buffer, int c, size_t num);
```

---

1. 综述

许多初学者对 C/C++语言中的 void 及 void 指针类型不甚理解，因此在使用上出现了一些错误。本文将对 void 关键字的深刻含义进行解说，并详述 void 及 void 指针类型的使用方法与技巧。

2.void 的含义

void 的字面意思是“无类型”，void*则为“无类型指针”，void*可以指向任何类型的数据。void 几乎只有“注释”和限制程序的作用，因为从来没有人会定义一个 void 变量，让我们试着来定义：

void a;

这行语句编译时会出错，提示“illegal use of type’ void’ ”。不过，即使 voida 的编译不会出错，它也没有任何实际意义。

void 真正发挥的作用在于：

（1）对函数返回的限定；

（2）对函数参数的限定。

众所周知，如果指针 p1 和 p2 的类型相同，那么我们可以直接在 p1 和 p2 间互相赋值；如果 p1 和 p2 指向不同的数据类型，则必须使用强制类型转换运算符把赋值运算符右边的指针类型转换为左边指针的类型。

例如：

```
float * p1;
in t* p2;
p1 = p2;
```

其中 p1 = p2 语句会编译出错，提示“’ =’ :cannotconvertfrom’ int *’ to’ float *’ ”，必须改为：

```
p1=(float*)p2;
```

而 void*则不同，任何类型的指针都可以直接赋值给它，无需进行强制类型转换：

```
void * p1;
int * p2;
p1 = p2;
```

但这并不意味着，void*也可以无需强制类型转换地赋给其它类型的指针。因为“无类型”可以包容“有类型”，而“有类型”则不能包容“无类型”。道理很简单，我们可以说“男人和女人都是人”，但不能说“人是男人”或者“人是女人”。下面的语句编译出错：

```
void * p1;
int * p2;
p2 = p1;
```

提示“’ =’ :cannotconvertfrom’ void*’ to’ int*’ ”。

3.void 的使用

下面给出 void 关键字的使用规则：

规则一如果函数没有返回值，那么应声明为 void 类型

在 C 语言中，凡不加返回值类型限定的函数，就会被编译器作为返回整型值处理。但是许多程序员却误以为其为 void 类型。例如：

```
add(inta,intb)
{
return a+b;
}
int main(int argc,char * argv[])
{
printf(/"2+3=%d/",add(2,3));
}
```

程序运行的结果为输出：

2+3=5

这说明不加返回值说明的函数的确为 int 函数。

林锐博士《高质量 C/C++编程》中提到：“C++语言有很严格的类型安全检查，不允许上述情况（指函数不加类型声明）发生”。可是编译器并不一定这么认定，譬如在 VisualC++6.0 中上述 add 函数的编译无错也无警告且运行正确，所以不能寄希望于编译器会做严格的类型检查。

因此，为了避免混乱，我们在编写 C/C++程序时，对于任何函数都必须一个不漏地指定其类型。如果函数没有返回值，一定要声明为 void 类型。这既是程序良好可读性的需要，也是编程规范性的要求。另外，加上 void 类型声明后，也可以发挥代码的“自注释”作用。代码的“自注释”即代码能自己注释自己。[Page]

规则二如果函数无参数，那么应声明其参数为 void

在 C++语言中声明一个这样的函数：

```
int function(void)
{
return1;
}
```

则进行下面的调用是不合法的：

```
function(2);
```

因为在 C++中，函数参数为 void 的意思是这个函数不接受任何参数。

我们在 TurboC2.0 中编译：

```
#include"stdio.h"
fun()
{
return1;
}
main()
{
printf(/"%d/",fun(2));
getchar();
}
```

编译正确且输出 1，这说明，在 C 语言中，可以给无参数的函数传送任意类型的参数，但是在 C++编译器中编译同样的代码则会出错。在 C++中，不能向无参数的函数传送任何参数，出错提示“'fun':functiondoesnottake1parameters”。

所以，无论在 C 还是 C++中，若函数不接受任何参数，一定要指明参数为 void。

规则三小心使用 void 指针类型

按照 ANSI(AmericanNationalStandardsInstitute)标准，不能对 void 指针进行算法操作，即下列操作都是不合法的：

```
void * pvoid;
pvoid ++;//ANSI：错误
pvoid += 1;//ANSI：错误
```

ANSI 标准之所以这样认定，是因为它坚持：进行算法操作的指针必须是确定知道其指向数据类型大小的。例如：

```
int * pint;
pint ++;//ANSI：正确
```

pint++的结果是使其增大 sizeof(int)。

但是大名鼎鼎的 GNU(GNU'sNotUnix 的缩写)则不这么认定，它指定 void * 的算法操作与 char * 一致。因此下列语句在 GNU 编译器中皆正确：

```
pvoid ++;//GNU：正确
pvoid += 1;//GNU：正确
```

pvoid++的执行结果是其增大了 1。

在实际的程序设计中，为迎合 ANSI 标准，并提高程序的可移植性，我们可以这样编写实现同样功能的代码：

```
void * pvoid;
(char*)pvoid ++;//ANSI：正确；GNU：正确
(char*)pvoid += 1;//ANSI：错误；GNU：正确
```

GNU 和 ANSI 还有一些区别，总体而言，GNU 较 ANSI 更“开放”，提供了对更多语法的支持。但是我们在真实设计时，还是应该尽可能地迎合 ANSI 标准。

规则四如果函数的参数可以是任意类型指针，那么应声明其参数为 void*

典型的如内存操作函数 memcpy 和 memset 的函数原型分别为：

```
void * memcpy(void*dest,constvoid*src,size_tlen);
void * memset(void*buffer,intc,size_tnum);
```

这样，任何类型的指针都可以传入 memcpy 和 memset 中，这也真实地体现了内存操作函数的意义，因为它操作的对象仅仅是一片内存，而不论这片内存是什么类型。如果 memcpy 和 memset 的参数类型不是 void*，而是 char*，那才叫真的奇怪了！这样的 memcpy 和 memset 明显不是一个“纯粹的，脱离低级趣味的”函数！

下面的代码执行正确：

```
//示例：memset 接受任意类型指针
int intarray[100];[Page]
memset(intarray,0,100*sizeof(int));//将 intarray 清 0
```

```
//示例：memcpy 接受任意类型指针
int intarray1[100],intarray2[100];
memcpy(intarray1,intarray2,100*sizeof(int));//将 intarray2 拷贝给 intarray1
```

有趣的是，memcpy 和 memset 函数返回的也是 void*类型，标准库函数的编写者是多么地富有学问啊！

规则五 void 不能代表一个真实的变量

下面代码都企图让 void 代表一个真实的变量，因此都是错误的代码：

```
void a;//错误
function(void a);//错误
```

void 体现了一种抽象，这个世界上的变量都是“有类型”的，譬如一个人不是男人就是女人（还有人妖？）。

void 的出现只是为了一种抽象的需要，如果你正确地理解了面向对象中“抽象基类”的概念，也很容易理解 void 数据类型。正如不能给抽象基类定义一个实例，我们也不能定义一个 void（让我们类比的称 void 为“抽象数据类型”）变量。

4.总结

小小的 void 蕴藏着很丰富的设计哲学，作为一名程序设计人员，对问题进行深一个层次的思考必然使我们受益匪浅。