**一、typedef用途**

**1、用途1**：定义一种类型别名，而不只是简单的宏替换。可以用作同时声明指针型的多个对象。

例：typedef char\* PCHAR;

PCHAR pa,pb;//同时声明了两个指向字符变量的指针

**2、用途2**：用在旧的C的代码中（具体多旧没有查），帮助struct。以前的代码中，声明struct新对象时，必须要带上struct，即形式为： struct 结构名 对象名，如：

1. **struct** tagPOINT1
2. {
3. **int** x;
4. **int** y;
5. };
6. **struct** tagPOINT1 p1;

而在C++中，则可以直接写：结构名 对象名，即：

tagPOINT1 p1;

也可以这样写：

1. **typedef** **struct** tagPOINT
2. {
3. **int** x;
4. **int** y;
5. }POINT;
6. POINT p1; // 这样就比原来的方式少写了一个struct，比较省事，尤其在大量使用的时候

**3、用途3**、用typedef来定义与平台无关的类型。  
比如定义一个叫 REAL 的浮点类型，在目标平台一上，让它表示最高精度的类型为：  
typedef long double REAL;   
在不支持 long double 的平台二上，改为：  
typedef double REAL;   
在连 double 都不支持的平台三上，改为：  
typedef float REAL;   
也就是说，当跨平台时，只要改下 typedef 本身就行，不用对其他源码做任何修改。  
标准库就广泛使用了这个技巧，比如size\_t。  
另外，因为typedef是定义了一种类型的新别名，不是简单的字符串替换，所以它比宏来得稳健（虽然用宏有时也可以完成以上的用途）。

4**、用途4、**为复杂的声明定义一个新的简单的别名。方法是：在原来的声明里逐步用别名替换一部分复杂声明，如此循环，把带变量名的部分留到最后替换，得到的就是原声明的最简化版。

例：int \*(\*a[5])(int, char\*);

typedef int \*(\*pFun)(int, char\*);

pFun a[5];

理解复杂声明可用的“右左法则”：  
从变量名看起，先往右，再往左，碰到一个圆括号就调转阅读的方向；括号内分析完就跳出括号，还是按先右后左的顺序，如此循环，直到整个声明分析完。举例：  
int (\*func)(int \*p);  
首先找到变量名func，外面有一对圆括号，而且左边是一个\*号，这说明func是一个指针；然后跳出这个圆括号，先看右边，又遇到圆括号，这说明 (\*func)是一个函数，所以func是一个指向这类函数的指针，即函数指针，这类函数具有int\*类型的形参，返回值类型是int。  
int (\*func[5])(int \*);  
func 右边是一个[]运算符，说明func是具有5个元素的数组；func的左边有一个\*，说明func的元素是指针（注意这里的\*不是修饰func，而是修饰 func[5]的，原因是[]运算符优先级比\*高，func先跟[]结合）。跳出这个括号，看右边，又遇到圆括号，说明func数组的元素是函数类型的指 针，它指向的函数具有int\*类型的形参，返回值类型为int。

也可以记住2个模式：  
type (\*)(....)函数指针   
type (\*)[]数组指针

**二、两大陷阱：**

**陷阱一：**

记住，typedef是定义了一种类型的新别名，不同于宏，它不是简单的字符串替换。比如：  
先定义：  
typedef char\* PSTR;  
然后：  
int mystrcmp(const PSTR, const PSTR);

const PSTR实际上相当于const char\*吗？不是的，它实际上相当于char\* const。  
原因在于const给予了整个指针本身以常量性，也就是形成了常量指针char\* const。  
简单来说，记住当const和typedef一起出现时，typedef不会是简单的字符串替换就行。

**陷阱二：**

typedef在语法上是一个存储类的关键字（如auto、extern、mutable、static、register等一样），虽然它并不真正影响对象的存储特性，如：  
typedef static int INT2; //不可行  
编译将失败，会提示“**指定了一个以上的存储类**”。

**三、typedef与define区别**

**案例一：**

通常讲，typedef要比#define要好，特别是在有指针的场合。请看例子：

**[cpp]** [view plain](http://blog.csdn.net/ameyume/article/details/6326278) [copy](http://blog.csdn.net/ameyume/article/details/6326278)

1. typedef char \*pStr1;
2. #define pStr2 char \*;
3. pStr1 s1, s2;
4. pStr2 s3, s4;

在上述的变量定义中，s1、s2、s3都被定义为char \*，而s4则定义成了char，不是我们所预期的指针变量，根本原因就在于**#define只是简单的字符串替换而typedef则是为一个类型起新名字。**

**案例二：**

下面的代码中编译器会报一个错误，你知道是哪个语句错了吗？

1. typedef char \* pStr;
2. char string[4] = "abc";
3. const char \*p1 = string;
4. const pStr p2 = string;
5. p1++;
6. p2++;

是p2++出错了。这个问题再一次提醒我们：typedef和#define不同，它不是简单的文本替换。上述代码中const pStr p2并不等于const char \* p2。const pStr p2和const long x本质上没有区别，都是对变量进行只读限制，只不过此处变量p2的数据类型是我们自己定义的而不是系统固有类型而已。因此，const pStr p2的含义是：限定数据类型为char \*的变量p2为只读，因此p2++错误。