概述

static关键字在c语言中比较常用，使用恰当能够大大提高程序的模块化特性，有利于扩展和维护。

但是对于c语言初学者，static由于使用灵活，并不容易掌握。本文就static在c语言中的应用进行总结，供参考使用。错漏之处，请不吝指正。

最后一节加入了c++面向对象中static的使用特性，当作拓展阅读。

在程序中使用static

变量

1. 局部变量

普通局部变量是再熟悉不过的变量了，在任何一个函数内部定义的变量（不加static修饰符）都属于这个范畴。编译器一般不对普通局部变量进行初始化，也就是说它的值在初始时是不确定的，除非对其显式赋值。

> 普通局部变量存储于进程栈空间，使用完毕会立即释放。

静态局部变量使用static修饰符定义，即使在声明时未赋初值，编译器也会把它初始化为0。且静态局部变量存储于进程的全局数据区，即使函数返回，它的值也会保持不变。

> 变量在全局数据区分配内存空间；编译器自动对其初始化
> 其作用域为局部作用域，当定义它的函数结束时，其作用域随之结束

小程序体会一下静态局部变量的威力：

```c
#include <stdio.h>

void fn(void)
{
    int n = 10;

    printf("n=%d\n", n);
    n++;
    printf("n++=%d\n", n);
}

void fn_static(void)
{
    static int n = 10;
```

```
16      printf("static n=%d\n", n);
17      n++;
18      printf("n++=%d\n", n);
19  }
20
21  int main(void)
22  {
23      fn();
24      printf("--------------------\n");
25      fn_static();
26      printf("--------------------\n");
27      fn();
28      printf("--------------------\n");
29      fn_static();
30
31      return 0;
32  }
```

运行结果如下：

```
1   -> % ./a.out
2   n=10
3   n++=11
4   --------------------
5   static n=10
6   n++=11
7   --------------------
8   n=10
9   n++=11
10  --------------------
11  static n=11
12  n++=12
```

可见，静态局部变量的效果跟全局变量有一拼，但是位于函数体内部，就极有利于程序的模块化了。

2. 全局变量

全局变量定义在函数体外部，在全局数据区分配存储空间，且编译器会自动对其初始化。

普通全局变量对整个工程可见，其他文件可以使用extern外部声明后直接使用。也就是说其他文件不能再定义一个与其相同名字的变量了（否则编译器会认为它们是同一个变量）。

静态全局变量仅对当前文件可见，其他文件不可访问，其他文件可以定义与其同名的变量，两者互不影响。

> 在定义不需要与其他文件共享的全局变量时，加上static关键字能够有效地降低程序模块之间的耦合，避免不同文件同名变量的冲突，且不会误使用。

函数

函数的使用方式与全局变量类似，在函数的返回类型前加上static，就是静态函数。其特性如下：

- 静态函数只能在声明它的文件中可见，其他文件不能引用该函数
- 不同的文件可以使用相同名字的静态函数，互不影响

> 非静态函数可以在另一个文件中直接引用，甚至不必使用extern声明

下面两个文件的例子说明使用static声明的函数不能被另一个文件引用：

```c
/* file1.c */
#include <stdio.h>

static void fun(void)
{
    printf("hello from fun.\n");
}

int main(void)
{
    fun();
    fun1();

    return 0;
}

/* file2.c */
#include <stdio.h>

static void fun1(void)
{
    printf("hello from static fun1.\n");
}
```

使用 `gcc file1.c file2.c` 编译时，错误报告如下：

```
/tmp/cc2VMzGR.o：在函数‘main’中：
static_fun.c:(.text+0x20)：对‘fun1’未定义的引用
collect2: error: ld returned 1 exit status
```

修改文件，不使用static修饰符，可在另一文件中引用该函数：

```c
/* file1.c */
#include <stdio.h>

void fun(void)
{
    printf("hello from fun.\n");
}

/* file2.c */
int main(void)
{
    fun();

    return 0;
}
```

同样使用 `gcc file1.c file2.c` 编译，编译通过，运行结果如下：

同样使用 `gcc file1.c file2.c` 编译，编译通过，运行结果如下：

```
-> % ./a.out
hello from fun.
```

面向对象

静态数据成员

在类内数据成员的声明前加上static关键字，该数据成员就是类内的静态数据成员。其特点如下：

- 静态数据成员存储在全局数据区，静态数据成员在定义时分配存储空间，所以不能在类声明中定义
- 静态数据成员是类的成员，无论定义了多少个类的对象，静态数据成员的拷贝只有一个，且对该类的所有对象可见。也就是说任一对象都可以对静态数据成员进行操作。而对于非静态数据成员，每个对象都有自己的一份拷贝。
- 由于上面的原因，静态数据成员不属于任何对象，在没有类的实例时其作用域就可见，在没有任何对象时，就可以进行操作
- 和普通数据成员一样，静态数据成员也遵从 `public, protected, private` 访问规则
- 静态数据成员的初始化格式：`<数据类型><类名>::<静态数据成员名>=<值>`
- 类的静态数据成员有两种访问方式：`<类对象名>.<静态数据成员名>` 或 `<类类型名>::<静态数据成员名>`

同全局变量相比，使用静态数据成员有两个优势：

- 静态数据成员没有进入程序的全局名字空间，因此不存在与程序中其它全局名字冲突的可能性
- 可以实现信息隐藏。静态数据成员可以是private成员，而全局变量不能

静态成员函数

与静态数据成员类似，静态成员函数属于整个类，而不是某一个对象，其特性如下：

- 静态成员函数没有this指针，它无法访问属于类对象的非静态数据成员，也无法访问非静态成员函数，它只能调用其余的静态成员函数
- 出现在类体外的函数定义不能指定关键字static
- 非静态成员函数可以任意地访问静态成员函数和静态数据成员

总结

static是一个很有用的关键字，使用得当可以使程序锦上添花。当然，有的公司编码规范明确规定只用于本文件的函数要全部使用static关键字声明，这是一个良好的编码风格。

无论如何，要在实际编码时注意自己的编码习惯，尽量体现出语言本身的优雅和编码者的编码素质。