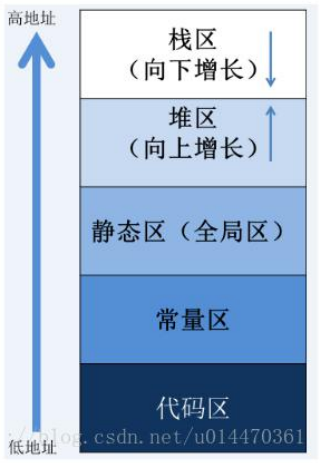

代码区：存放程序的代码，即CPU执行的机器指令，并且是只读的。
常量区：存放常量(程序在运行的期间不能够被改变的量，例如: 10，字符串常量"abcde"， 数组的名字等)
静态区（全局区）：静态变量和全局变量的存储区域是一起的，一旦静态区的内存被分配, 静态区的内存直到程序全部结束之后才会被释放
堆区：由程序员调用malloc()函数来主动申请的，需使用free()函数来释放内存，若申请了堆区内存，之后忘记释放内存，很容易造成内存泄漏
栈区：存放函数内的局部变量，形参和函数返回值。栈区之中的数据的作用范围过了之后，系统就会回收自动管理栈区的内存(分配内存，回收内存),不需要开发人员来手动管理。栈区就像是一家客栈，里面有很多房间，客人来了之后自动分配房间，房间里的客人可以变动，是一种动态的数据变动。

5.2，栈、堆和静态区

对于程序员，一般来说，我们可以简单的理解为内存分为三个部分：静态区，栈，堆。很多书没有把把堆和栈解释清楚，导致初学者总是分不清楚。其实堆栈就是栈，而不是堆。堆的英文是 heap；栈的英文是 stack，也翻译为堆栈。堆和栈都有自己的特性，这里先不做讨论。再打个比方：一层教学楼，可能有外语教室，允许外语系学生和老师进入；还可能有数学教师，允许数学系学生和老师进入；还可能有校长办公室，允许校长进入。同样，内存也是这样，内存的三个部分，不是所有的东西都能存进去的。

静态区：保存自动全局变量和 static 变量（包括 static 全局和局部变量）。静态区的内容在总个程序的生命周期内都存在，由编译器在编译的时候分配。

栈：保存局部变量。栈上的内容只在函数的范围内存在，当函数运行结束，这些内容也会自动被销毁。其特点是效率高，但空间大小有限。

堆：由 malloc 系列函数或 new 操作符分配的内存。其生命周期由 free 或 delete 决定。在没有释放之前一直存在，直到程序结束。其特点是使用灵活，空间比较大，但容易出错。

1、栈区：

```c
#include<stdio.h>

int main(void) {
    int x = 2;                  //在栈上面申请一个int类型长度的空间
    int y[] = {1,2,3};          //在栈上面申请一个int类型数组长度为3的一段空间
    char s[] = {"132423"};      //在栈上面申请一个char类型数组长度为6的一段空间
    printf("x_address = %p\n", &x);
    printf("y_address = %p\n", &y);
    printf("s_address = %p\n", s);
    return 0;
}
```

可以通过输出地址看一下上述几个变量的地址变化。

```
x_address = 0028FF3C
y_address = 0028FF30
s_address = 0028FF29
```


比如上面代码中变量的定义都是在栈上面建立的，在栈上面申请变量由编译器自动分配释放，存放函数的参数值、局部变量的值等，定义的变量出了作用范围之后会由编译器自动释放，栈的大小一般是2M或者1M，一个int是4个字节，2M的大小就是2M=2*1024K = 2 * 1024*1024 B,这么多个字节。栈里面的地址是向低地址扩展的，可以想象成从栈底向栈顶消耗栈空间。在栈里面分配空间的优点是：分配速度快，不用担心用完了回收的事情。缺点就是：整个栈的空间大小太小了，拿2M的大小来算，就能放int类型的数据2 * 1024*1024 / 4 = 524288个，所以栈的地方很适合函数局部变量，不必担心回收的问题，不适合申请很大的空间。

2、堆区

```c
#include<stdio.h>
#include<malloc.h>

int main(void) {
    int *array1  = NULL;
    int count = 5;
    int i;

    array1 = (int *)calloc(sizeof(int), count);   //在堆上面申请一段长度为sizeof(int)*count空间给array1指针
    for(i = 0; i < count; i++) {
        printf("%p\n", &array1[i]);
    }

    free(array1);

    return 0;
}
```

```
00961828
0096182C
00961830
00961834
00961838
```


堆区的申请是动态的，可通过用户传入的参数申请指定长度的内存空间，堆的内存也是很大，可以看成当前后台空闲的内存大小，可以通过new，malloc，calloc，realloc申请，回收的方式是delete或者free。堆区的地址是从低向高扩展的。

每次需要申请一段内存空间的时候，会在当前的内存里面找一段连续的内存合适大小分配给这个指针，至于分配的算法有几种：最佳适应、最差适应、首次适应。

最佳适应算法和最差适应算法是反过来的，这两种算法对当前空间的内存块都是按照大小排序，最差适应是分配了最大的内存，最佳适应是分配了刚好合适的内存块。而首次适应不用对空闲区排序，只要找到一个满足需求大小的空闲分区，就分配。

通常动态在堆上面分配的内存，比如一个长度为10的int类型数组，应该占用字节数是40个，但实际上是申请了40 的下一个2的次方数，也就是64，它会圆整为下一个大于申请数量的2的整数次方，申请的长度是64字节长度。16个int的长度。所以一旦发生内存泄漏，忘记手动释放申请的空间，造成泄漏的内存要比忘记释放的那个的数据结构更大。

在堆上申请空间的优点：堆的空间大、只要有足够的空闲空间可申请任意大小的空间。

缺点：动态申请的空间必须记得写释放函数操作，否则就会造成严重而且很难察觉的内存泄漏！！野指针！！

```c
array1 = (int *)calloc(sizeof(int), 10);
```

3、全局/静态区

static修饰或者全局的变量放在这里，全局静态区的空间大小和堆的大小差不多。

其中初始化的变量和未初始化变量存放的不是一个位置。未初始化的变量放在未初始化数据区（BBS），这个区域用来存放程序中未初始化的全局/静态变量的一块内存区域。

对于这一部分的理解，可以把静态变量和全局变量归为一类理解，不过在c里面的语法要求，static可以用来修饰一个局部变量，java里面的static修饰变量必须是在全局下的，是一个类的成员，函数里面static修饰变量一定报错。

```c
#include<stdio.h>

int x = 2;      //全局变量初始化过的
int y;          //全局变量未初始化


int main(void) {
    static int z = 2;           //静态变量初始化过的
    static int z2;              //静态变量未初始化的

    printf("x: %p\n", &x);
    printf("y: %p\n", &y);
    printf("z: %p\n", &z);
    printf("z2: %p\n", &z2);

    return 0;
}
```

```
x: 00404004
y: 00407074
z: 00404008
z2: 00407020
```


4、字符常量区

```c
char *s1 = "1234";  //常量 不可更改
```

这种方式申请的是一个字符常量，不可更改，相同的内容在内存中只占一份空间，上述就是一个字符常量，字符常量区在全局静态区的一部分，但没有和全局静态变量挨在一起。

下面的代码中虽然有多个字符串，s1,s2,s3，但是这三个字符串在内存中的地址绝对是相同的，并且字符串是不可更改的，这个常量是字符常量，和const修饰的变量还不一样，就算const int a = 2这样子这个a还是在栈上而不是在字符常量区。

```c
#include<stdio.h>

char *s3 = "1234";   //存放字符常量区 相同内容只占一份内存

int main(void) {
    char *s1 = "1234";
```

```c
    char *s2 = "1234";
```

printf("s1 = %p\n", s1);printf("s2 = %p\n", s2);printf("s3 = %p\n", s3);return 0;}

```
s1 = 00405064
s2 = 00405064
s3 = 00405064
```


5、程序代码区：程序代码区，用于存放程序的二进制代码的空间，这一部分不是很理解。