



# C++语言基础

## 迂者 - 贺利坚

http://blog.csdn.net/sxhelijian/

http://edu.csdn.net



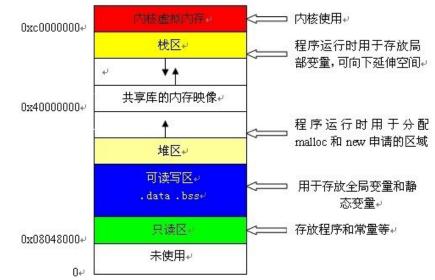
本节主题:

内存中的堆区和栈区



## 一般菜鸟不懂的内幕(C/C++程序运行中的堆区和栈区)

```
#include <iostream>
using namespace std;
int a = 0; //全局初始化区
char *p1; //全局未初始化区
int main()
 int b; //b在栈区
 char s[] = "abc"; //s在栈区
 char *p2; //在栈区
 char *p3 = "123"; //123\0在常量区, p3在栈区
 static int c =0; //全局(静态)初始化区
 p1 = new char(65); //在堆区
 p2 = new char[10]; //在堆区
 return 0;
```



- □ 栈区:栈区是分配局部变量的空间,处于相对较高的 地址,栈地址向下增长,
- □ 堆区: 用于分配程序员申请的内存空间, 是向上增长的
- 🗅 静态数据区: 分配静态变量、全局变量的空间
- □ 只读区: 分配常量和程序代码



## 堆区和栈区的差异

#### 0.申请方式和回收方式不同

栈(stack):栈上的空间是自动分配自动回收的, 生存周期只在函数的运行过程中,运行后就释放。

堆(heap):程序员根据需要自己申请的空间, 只要程序员不释放空间,就一直可以访问到(一 旦忘记释放会造成内存泄露)。

#### 1.申请后系统的响应

栈:只要栈的剩余空间大于所申请空间,系统将为程序提供内存,否则将报异常提示栈溢出。

堆:系统收到程序的申请时,会遍历空闲内存地址的链表,寻找第一个空间大于所申请空间的堆结点,然后.....(各种事),使其效率相对较低。

#### 5.存取效率——栈:快;堆:慢

#### 2.申请效率的比较

栈:由系统自动分配,速度较快;程序员无法控制

堆:由new分配,一般速度较慢,而且容易产生内

存碎片;不过用起来最方便。

#### 3.申请大小的限制

栈:在Windows下,栈的大小是2M,如果申请的空间超过栈的剩余空间时,将提示overflow。

堆:堆的大小受限于计算机系统中有效的虚拟内

存。堆获得的空间比较灵活,也比较大。

#### 4. 堆和栈中的存储内容

栈: 函数调用语句的下一条可执行语句的地址、

函数的各个参数、函数中的局部变量

堆:具体内容由程序员安排。



## 这样安排是合理的

```
#include <iostream>
using namespace std;
                                                  内核虚拟内容。
                                                                      内核使用↩
                                    0xc0000000+
int a = 0; //全局初始化区
                                                     栈区₽
                                                                      程序运行时用于存放局
char *p1; //全局未初始化区
                                                                      部变量,可向下延伸空间₽
int main()
                                                共享库的内存映像↓
                                    0x400000000
 int b: //b在栈区
                                                                      程序运行时用于分配
                                                                      malloc 和 new 申请的区域
 char s[] = "abc"; //s在栈区
                                                     堆区₽
 char *p2; //在栈区
                                                    可读写区↩
                                                                       用于存放全局变量和静
 char *p3 = "123"; //123\0在常量区, p3在栈区
                                                   .data .bss+
                                                                            杰变量↓
 static int c =0; //全局(静态)初始化区
                                                    只读区。
                                                                      存放程序和常量等₹
 p1 = new char(65); //在堆区
                                    0x080480004
 p2 = new char[10]; //在堆区
                                                    未使用↩
                                          04
 return 0;
```



### 再看深复制

#### 当有指针数据成员,必须用深复制,使其使用堆区的空间!

```
class Test
private:
  int a;
  char *str;
public:
  Test(int b, char *s)
    a=b;
    int n=strlen(s)+1;
    str=new char[n];
  Test(const Test& C)
    a=C.a;
    int n=strlen(C.str)+1;
    str=new char[n];
    strcpy(str,C.str);
```

```
~Test()
    delete str;
  void show ()
    cout<<a<<","<<str<<endl;
int main()
  Test *a;
  a=new Test(100,"Hello");
  Test b(*a);
  b.show();
  delete a;
  return 0;
```

```
内核使用₹
0xc00000004
               桟区₽
                               程序运行时用于存放局
                               部变量,可向下延伸空间₽
           共享库的内存映像₽
0x400000004
                               程序运行时用于分配
                               malloc 和 new 申请的区域
               堆区₽
              可读写区↩
                                用于存放全局变量和静
             .data .bss⊬
                                    态变量↓
               只读区。
                               存放程序和常量等₽
0x080480004
              未使用↵
  //浅复制
  Test(const Test& C)
    a=C.a;
    str=C.str;
```





## THANKS

本课程由 迂者-贺利坚 提供

CSDN网站:www.csdn.net 企业服务:http://ems.csdn.net/ 人才服务:http://job.csdn.net/ CTO俱乐部:http://cto.csdn.net/ 高校俱乐部:http://student.csdn.net/ 程序员杂志:http://programmer.csdn.net/

CODE平台: https://code.csdn.net/ 项目外包: http://www.csto.com/ CSDN博客: http://blog.csdn.net/ CSDN论坛: http://bbs.csdn.net/ CSDN下载: http://download.csdn.net/