**内存区域**

一个由C/C++编译的程序占用的内存分为以下几个部分

1、栈区（stack）： 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。其操作方式类似于数据结构中的栈。

2、堆区（heap） ： 一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收 。注意它与数据结构中的堆是两回事，分配方式倒是类似于链表。

3、全局区（静态区）（static）：全局变量和静态变量的存储都是在一块的，初始化的全局变量和静态变量在一块区域， 未初始化的全局变量和未初始化的静态变量在相邻的另一块区域， 程序结束后有系统释放 。

4、常量区：常量、字符串常量池就是放在这里的， 程序结束后由系统释放 。

5、程序代码区：存放函数体的二进制代码。

示例代码

#include<iostream>

int a = 0;//全局初始化区

char \*p1;//全局未初始化区

int main()

{

int b;//栈

char s[] = "abc";//栈

char \*p2;//栈

char \*p3 = "123456";//123456在常量区，p3在栈上

static int c = 0;//全局（静态）初始化区

//分配得来得10和20字节的区域就在堆区

p1 = (char \*)malloc(10);

p2 = (char \*)malloc(20);

//123456字符串放在常量区，编译器可能会将它指向p3所指向的"123456"的内存地址。

//字符串常量池

strcpy(p1, "123456");

return 0;

}

栈溢出

栈溢出指的是栈的使用超出了栈的大小，一个原因有两个：

1、 局部数组变量空间太大

        局部变量是存储在栈中的，因此这个很好理解

  解决这类问题的办法有两个，

      一是增大栈空间,二是改用动态分配，使用堆（heap）而不是栈（stack）。

2、 函数出现无限递归调用，函数调用的时候会将一些运行相关的信息压栈，这个也就好理解了。

# C#--字符串常量池

在介绍字符串常量池之前，先看一段简单的代码：

using System;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

string a = "123";

string b = "123";

if (ReferenceEquals(a, b))

{

Console.WriteLine("相同");//输出相同

}

Console.ReadLine();

}

}

}

CLR是公共语言运行库(Common Language Runtime)和Java虚拟机一样也是一个运行时环境，它负责资源管理（内存分配和垃圾收集等）。

CLR为了减少字符串对象的重复创建，其维护了一个特殊的内存，这段内存被成为字符串常量池。字符串常量池不在堆中也不在栈中,是独立的内存空间管理，在内存的常量区。

基本原理：

当我们定义了a和b的字符串，然后CLR内部机制去字符串常量池中找，如果存在相同内容的字符串对象的引用，则将这个引用返回。否则新的字符串对象被创建，然后将这个引用放入字符串常量池，并返回该引用。

因此，上面的代码就会输出引用相同。

以下三种情况会查询暂存池（若查询不到就将其存入暂存池），使用ReferenceEquals() 判断，string a，string b，string c引用相同：

利用字面量值创建string对象，例如string a = "123";

利用string.Intern()创建string对象，例如string b = string.Intern("123");

字面量值+字面量值拼接创建string对象， 例如string c = "12" + "3";

注意：不是所有的字符串都放在暂存池中，运行时期动态创建的字符串不会被加入到驻留池中。

关于字符串常量池的更深理解：

1. 驻留池由CLR来维护，其中的所有字符串对象的值都不相同。

2. 只有编译阶段的文本字符常量会被自动添加到驻留池。

3.运行时期动态创建的字符串不会被加入到驻留池中。

4.string.Intern()可以把动态创建的字符串加入到驻留池中。