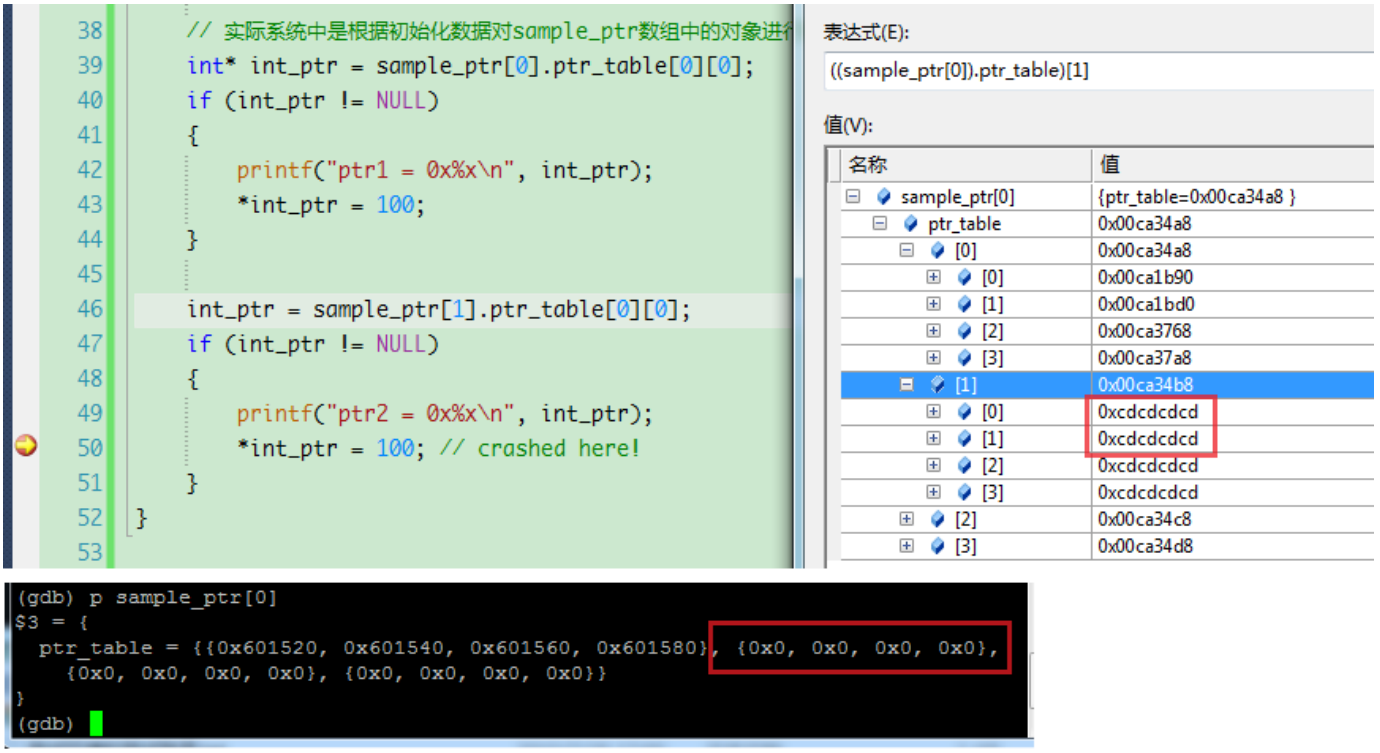
由一次程序崩溃引起的对 new 表达式的再次学习 - 文章 - 伯乐在线

原文出处： tryflying

1. 起因

某天，一个同事跟我反馈说在windows上调试公司产品的一个交易核心时出现了使用未初始化的指针导致后台服务崩溃的情况。示例代码如下所示：

```
C++
struct sample
{
        int* ptr_table[4][4];
        //... other members
};
void test()
{
        sample* sample_ptr = new sample[10];
        for (int i = 0; i < 4; i++)
                sample_ptr[0].ptr_table[0][i] = new int(i);
        // 实际系统中是根据初始化数据对sample_ptr数组中的对象进行赋值，但不是所有的对象都有初始化数据;
        int* int_ptr = sample_ptr[0].ptr_table[0][0];
        if (int_ptr != NULL)
        {
                printf("ptr1 = 0x%x\n", int_ptr);
                *int_ptr = 100;
        }
        int_ptr = sample_ptr[1].ptr_table[0][0];
        if (int_ptr != NULL)
        {
                printf("ptr2 = 0x%x\n", int_ptr);
                *int_ptr = 100; // crashed here!
        }
}
```



将sample* sample_ptr = new sample[10]这行改为sample* sample_ptr = new sample[10]()后两个系统执行的结果变一样了，都是被初始化的内存。

那么问题来了：

(1) 为什么相同的代码(new sample[10])在两个系统下表现形式不一样呢？是两个系统的内存分配机制的原因还是类库的原因？

(2) new sample[10]和new sample[10]()的区别到底是什么？

2. 研究

在c++中，一般都是以new/delete来申请和释放内存。对于以下几种new的用法，各自的区别是什么呢？

```
C++
int* p1 = new int;
int* p2 = new int();
int* p3 = new int(1);
// define class A;
A* p4     = new A;
A* p5     = new A();
A* p6     = new A[10];
A* p7     = new A[10]();
```

在平时写代码的时候，对于一块新申请的内存我都会要对它进行初始化后才会去用它，一般是例如p3的直接初始化或者memset，因此对于new A和new A()两种用法的结果还真是不了解之间的区别。

根据《C++ Primer, Fourth Edition》中5.11节[1] The new and delete Expressions中关于new的描述，new A属于Default Initializing of Dynamically Allocated Objects（动态创建对象的默认初始化），而new A()则属于Value Initializing of Dynamically Allocated Objects（动态创建对象的值

初始化）。

当为默认初始化操作时，若被创建的对象没有显式定义默认构造函数，则按照2.3.4节[2]Variable Initialization Rules的规则进行初始化：

- 对象为内置类型时，任何在函数体外定义的变量都会被初始化0（全局变量或者静态变量），在函数体内定义的变量都不会进行初始化
- 对象为类类型时，调用对象的默认构造函数

当为值初始化操作时，若被创建的对象没有显式定义默认构造函数，则认为对该对象进行初始化操作。

不同new的用法对应的初始化的逻辑总结如下：

	new A	new A()	new A(parameters)
A为内置类型	无初始化动作	进行值初始化 例A为int类型，则初始化为0	进行值初始化，A被初始化为parameters
A为calss/struct	调用默认构造函数，A中成员是否 初始化依赖于默认构造函数的实现	若自定义了默认构造函数，则调用自定义的默认构造函数。 否则调用系统默认构造函数，并对A中的成员进行值初始化	调用A的自定义构造函数

因此上面代码的执行结果分别为：

- p1指向了一个未被初始化的int空间
- p2指向了一个被初始化的int空间，其值为0
- p3指向了一个被初始化的int空间，其值为1
- p4指向了一个调用了默认构造函数的实例A，除非A的默认构造函数对A的成员进行初始化，否则A的成员全为未初始化变量
- p5指向了一个调用了默认构造函数的实例A，若A自定义了默认构造函数，A成员变量的初始化依赖于自定义的默认构造函数，反之A的成员变量全为初始化后的变量
- p6、p7指向了调用了默认构造函数的实例A的数组，其执行结果同p4、p5

回到之前的问题，结构体sample是没有自定义默认构造函数的，按照c++标准，new sample的执行结果是sample中的ptr_table是不会被初始化的。这个在windows上是一致的，可是在linux上为什么被初始化成了0了呢？在stackoverflow上找到了一个类似的问题[3]，其答案为：“Memory coming from the OS will be zeroed for security reasons….the C standard says nothing about this. This is strictly an OS behavior. So this zeroing may or may not be present on systems where security is not a concern”。操作系统的安全机制会在用户程序申请内存时对分配的内存进行初始化，以防止别有用心的人从新申请到的内存中读取到敏感数据，例如密码等，正是由于这个linux的特性使程序在首次申请内存时总是能得到一块被初始化的内存。

关于这个安全机制我在网上并没有搜到比较官方的说明文档，也许是我的搜索方式有误或者没有找对关键词，而且发现在程序运行过程中申请的内存也总并不是被初始化了的内存，例如下面的代码：

C++