**[关于堆和栈的那些事](http://www.codeceo.com/article/stack-and-heap.html" \o "关于堆和栈的那些事 - 码农网)**

2014-09-14    分类：[算法设计](http://www.codeceo.com/article/category/develop/algorithm)、[编程开发](http://www.codeceo.com/article/category/develop)、[首页精华](http://www.codeceo.com/article/category/pick)[2人评论](http://www.codeceo.com/article/stack-and-heap.html#comments)     来源：[独酌逸醉](http://www.perfect-is-shit.com/what-and-where-are-the-stack-and-heap.html)

分享到：更多[7](http://www.codeceo.com/article/stack-and-heap.html)

**欢迎分享开发问题到**[**码农社区**](http://bbs.codeceo.com/)**！将问题帖子私信**[**@程序员俱乐部**](http://weibo.com/u/2832482174)**，将有近60000人为您解答！**

**问题描述**

编程语言书籍中经常解释值类型被创建在栈上，引用类型被创建在堆上，但是并没有本质上解释这堆和栈是什么。我仅有高级语言编程经验，没有看过对此更清晰的解释。我的意思是我理解什么是栈，但是它们到底是什么，在哪儿呢（站在实际的计算机物理内存的角度上看）？

1. 在通常情况下由操作系统（OS）和语言的运行时（runtime）控制吗？
2. 它们的作用范围是什么？
3. 它们的大小由什么决定？
4. 哪个更快？

**答案一**

栈是为执行线程留出的内存空间。当函数被调用的时候，栈顶为局部变量和一些 bookkeeping 数据预留块。当函数执行完毕，块就没有用了，可能在下次的函数调用的时候再被使用。栈通常用后进先出（LIFO）的方式预留空间；因此最近的保留块（reserved block）通常最先被释放。这么做可以使跟踪堆栈变的简单；从栈中释放块（free block）只不过是指针的偏移而已。

堆（heap）是为动态分配预留的内存空间。和栈不一样，从堆上分配和重新分配块没有固定模式；你可以在任何时候分配和释放它。这样使得跟踪哪部分堆已经被分配和被释放变的异常复杂；有许多定制的堆分配策略用来为不同的使用模式下调整堆的性能。

每一个线程都有一个栈，但是每一个应用程序通常都只有一个堆（尽管为不同类型分配内存使用多个堆的情况也是有的）。

直接回答你的问题： 1. 当线程创建的时候，操作系统（OS）为每一个系统级（system-level）的线程分配栈。通常情况下，操作系统通过调用语言的运行时（runtime）去为应用程序分配堆。 2. 栈附属于线程，因此当线程结束时栈被回收。堆通常通过运行时在应用程序启动时被分配，当应用程序（进程）退出时被回收。 3. 当线程被创建的时候，设置栈的大小。在应用程序启动的时候，设置堆的大小，但是可以在需要的时候扩展（分配器向操作系统申请更多的内存）。 4. 栈比堆要快，因为它存取模式使它可以轻松的分配和重新分配内存（指针/整型只是进行简单的递增或者递减运算），然而堆在分配和释放的时候有更多的复杂的 bookkeeping 参与。另外，在栈上的每个字节频繁的被复用也就意味着它可能映射到处理器缓存中，所以很快（译者注：局部性原理）。

**答案二**

**Stack:**

1. 和堆一样存储在计算机 RAM 中。
2. 在栈上创建变量的时候会扩展，并且会自动回收。
3. 相比堆而言在栈上分配要快的多。
4. 用数据结构中的栈实现。
5. 存储局部数据，返回地址，用做参数传递。
6. 当用栈过多时可导致栈溢出（无穷次（大量的）的递归调用，或者大量的内存分配）。
7. 在栈上的数据可以直接访问（不是非要使用指针访问）。
8. 如果你在编译之前精确的知道你需要分配数据的大小并且不是太大的时候，可以使用栈。
9. 当你程序启动时决定栈的容量上限。

**Heap：**

1. 和栈一样存储在计算机RAM。
2. 在堆上的变量必须要手动释放，不存在作用域的问题。数据可用 delete, delete[] 或者 free 来释放。
3. 相比在栈上分配内存要慢。
4. 通过程序按需分配。
5. 大量的分配和释放可造成内存碎片。
6. 在 C++ 中，在堆上创建数的据使用指针访问，用 new 或者 malloc 分配内存。
7. 如果申请的缓冲区过大的话，可能申请失败。
8. 在运行期间你不知道会需要多大的数据或者你需要分配大量的内存的时候，建议你使用堆。
9. 可能造成内存泄露。

举例：

int foo()  
{  
char \*pBuffer; //<–nothing allocated yet (excluding the pointer itself, which is allocated here on the stack).  
bool b = true; // Allocated on the stack.  
if(b)  
{  
//Create 500 bytes on the stack  
char buffer[500];

//Create 500 bytes on the heap  
pBuffer = new char[500];

}//<– buffer is deallocated here, pBuffer is not  
}//<— oops there’s a memory leak, I should have called delete[] pBuffer;

**答案三**

堆和栈是两种内存分配的两个统称。可能有很多种不同的实现方式，但是实现要符合几个基本的概念:

1.对栈而言，栈中的新加数据项放在其他数据的顶部，移除时你也只能移除最顶部的数据（不能越位获取）。



2.对堆而言，数据项位置没有固定的顺序。你可以以任何顺序插入和删除，因为他们没有“顶部”数据这一概念。



上面上个图片很好的描述了堆和栈分配内存的方式。

**在通常情况下由操作系统（OS）和语言的运行时（runtime）控制吗？**

如前所述，堆和栈是一个统称，可以有很多的实现方式。计算机程序通常有一个栈叫做[调用栈](http://en.wikipedia.org/wiki/Call_stack)，用来存储当前函数调用相关的信息（比如：主调函数的地址，局部变量），因为函数调用之后需要返回给主调函数。栈通过扩展和收缩来承载信息。实际上，程序不是由运行时来控制的，它由编程语言、操作系统甚至是系统架构来决定。

堆是在任何内存中动态和随机分配的（内存的）统称；也就是无序的。内存通常由操作系统分配，通过应用程序调用 API 接口去实现分配。在管理动态分配内存上会有一些额外的开销，不过这由操作系统来处理。

**它们的作用范围是什么？**

调用栈是一个低层次的概念，就程序而言，它和“作用范围”没什么关系。如果你反汇编一些代码，你就会看到指针引用堆栈部分。就高级语言而言，语言有它自己的范围规则。一旦函数返回，函数中的局部变量会直接直接释放。你的编程语言就是依据这个工作的。

在堆中，也很难去定义。作用范围是由操作系统限定的，但是你的编程语言可能增加它自己的一些规则，去限定堆在应用程序中的范围。体系架构和操作系统是使用虚拟地址的，然后由处理器翻译到实际的物理地址中，还有页面错误等等。它们记录那个页面属于那个应用程序。不过你不用关心这些，因为你仅仅在你的编程语言中分配和释放内存，和一些错误检查（出现分配失败和释放失败的原因）。

**它们的大小由什么决定？**

依旧，依赖于语言，编译器，操作系统和架构。栈通常提前分配好了，因为栈必须是连续的内存块。语言的编译器或者操作系统决定它的大小。不要在栈上存储大块数据，这样可以保证有足够的空间不会溢出，除非出现了无限递归的情况（额，栈溢出了）或者其它不常见了编程决议。

堆是任何可以动态分配的内存的统称。这要看你怎么看待它了，它的大小是变动的。在现代处理器中和操作系统的工作方式是高度抽象的，因此你在正常情况下不需要担心它实际的大小，除非你必须要使用你还没有分配的内存或者已经释放了的内存。

**哪个更快一些？**

栈更快因为所有的空闲内存都是连续的，因此不需要对空闲内存块通过列表来维护。只是一个简单的指向当前栈顶的指针。编译器通常用一个专门的、快速的寄存器来实现。更重要的一点事是，随后的栈上操作通常集中在一个内存块的附近，这样的话有利于处理器的高速访问（译者注：局部性原理）。

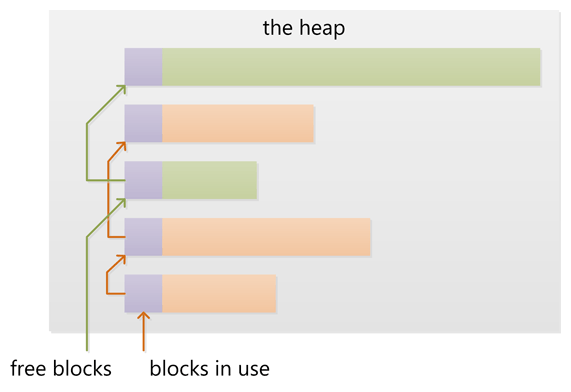
**答案四**

你问题的答案是依赖于实现的，根据不同的编译器和处理器架构而不同。下面简单的解释一下：

1. 栈和堆都是用来从底层操作系统中获取内存的。
2. 在多线程环境下每一个线程都可以有他自己完全的独立的栈，但是他们共享堆。并行存取被堆控制而不是栈。

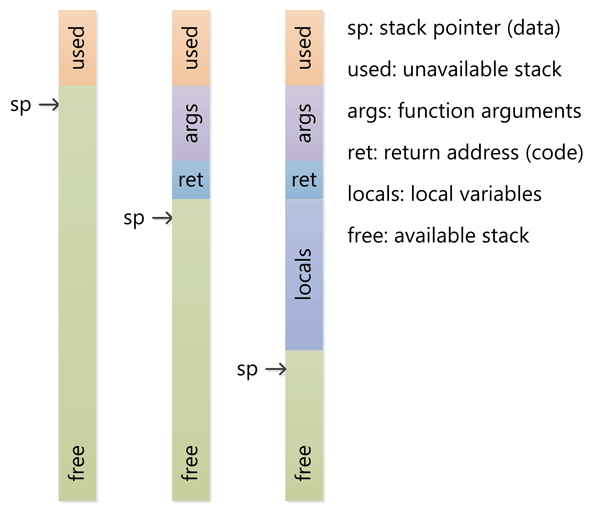
**堆:**

1. 堆包含一个链表来维护已用和空闲的内存块。在堆上新分配（用 new 或者 malloc）内存是从空闲的内存块中找到一些满足要求的合适块。这个操作会更新堆中的块链表。这些元信息也存储在堆上，经常在每个块的头部一个很小区域。
2. 堆的增加新快通常从地地址向高地址扩展。因此你可以认为堆随着内存分配而不断的增加大小。如果申请的内存大小很小的话，通常从底层操作系统中得到比申请大小要多的内存。
3. 申请和释放许多小的块可能会产生如下状态：在已用块之间存在很多小的空闲块。进而申请大块内存失败，虽然空闲块的总和足够，但是空闲的小块是零散的，不能满足申请的大小，。这叫做“堆碎片”。
4. 当旁边有空闲块的已用块被释放时，新的空闲块可能会与相邻的空闲块合并为一个大的空闲块，这样可以有效的减少“堆碎片”的产生。



**栈:**

1. 栈经常与 sp 寄存器（译者注：”stack pointer”，了解汇编的朋友应该都知道）一起工作，最初 sp 指向栈顶（栈的高地址）。
2. CPU 用 push 指令来将数据压栈，用 pop 指令来弹栈。当用 push 压栈时，sp 值减少（向低地址扩展）。当用 pop 弹栈时，sp 值增大。存储和获取数据都是 CPU 寄存器的值。
3. 当函数被调用时，CPU使用特定的指令把当前的 IP （译者注：“instruction pointer”，是一个寄存器，用来记录 CPU 指令的位置）压栈。即执行代码的地址。CPU 接下来将调用函数地址赋给 IP ，进行调用。当函数返回时，旧的 IP 被弹栈，CPU 继续去函数调用之前的代码。
4. 当进入函数时，sp 向下扩展，扩展到确保为函数的局部变量留足够大小的空间。如果函数中有一个 32-bit 的局部变量会在栈中留够四字节的空间。当函数返回时，sp 通过返回原来的位置来释放空间。
5. 如果函数有参数的话，在函数调用之前，会将参数压栈。函数中的代码通过 sp 的当前位置来定位参数并访问它们。
6. 函数嵌套调用和使用魔法一样，每一次新调用的函数都会分配函数参数，返回值地址、局部变量空间、嵌套调用的活动记录都要被压入栈中。函数返回时，按照正确方式的撤销。
7. 栈要受到内存块的限制，不断的函数嵌套/为局部变量分配太多的空间，可能会导致栈溢出。当栈中的内存区域都已经被使用完之后继续向下写（低地址），会触发一个 CPU 异常。这个异常接下会通过语言的运行时转成各种类型的栈溢出异常。（译者注：“不同语言的异常提示不同，因此通过语言运行时来转换”我想他表达的是这个含义）



\*函数的分配可以用堆来代替栈吗？

不可以的，函数的活动记录（即局部或者自动变量）被分配在栈上， 这样做不但存储了这些变量，而且可以用来嵌套函数的追踪。

堆的管理依赖于运行时环境，C 使用 malloc ，C++ 使用 new ，但是很多语言有垃圾回收机制。

栈是更低层次的特性与处理器架构紧密的结合到一起。当堆不够时可以扩展空间，这不难做到，因为可以有库函数可以调用。但是，扩展栈通常来说是不可能的，因为在栈溢出的时候，执行线程就被操作系统关闭了，这已经太晚了。

**译者注**

关于堆栈的这个帖子，对我来说，收获非常多。我之前看过一些资料，自己写代码的时候也常常思考。就这方面，也和祥子（我的大学舍友，现在北京邮电读研，技术牛人）探讨过多次了。但是终究是一个一个的知识点，这个帖子看完之后，豁然开朗，把知识点终于连接成了一个网。这种感觉，经历过的一定懂得，期间的兴奋不言而喻。

这个帖子跟帖者不少，我选了评分最高的四个。这四个之间也有一些是重复的观点。个人钟爱第四个回答者，我看的时候，瞬间高潮了，有木有？不过需要一些汇编语言、操作系统、计算机组成原理的的基础，知道那几个寄存器是干什么的，要知道计算机的流水线指令工作机制，保护/恢复现场等概念。三个回复者都涉及到了操作系统中虚拟内存；在比较速度的时候，大家一定要在脑中对“局部性原理”和计算机高速缓存有一个概念。

如果你把这篇文章看懂了，我相信你收获的不只是堆和栈，你会理解的更多！

兴奋之余，有几点还是要强调的，翻译没有逐字逐词翻译，大部分是通过我个人的知识积累和对回帖者的意图揣测而来的。请大家不要咬文嚼字，逐个推敲，我们的目的在于技术交流，不是么？达到这一目的就够了。

下面是一些不确定点:

1. 我没有听过 bookkeeping data 这种说法，故没有翻译。从上下文理解来看，可以想成是用来寄存器值？函数参数？返回地址？如果有了解具体含义的朋友，烦请告知。
2. 栈和堆栈是一回事，英文表达是 stack，堆是 heap。
3. 调用栈的概念，我是第一次听说，不太熟悉。大家可以去查查资料研究一下。

以上，送给大家，本文结束。