所有的程序都必须和计算机内存打交道，如何从内存中申请空间来存放程序的运行内容，如何在不需要的时候释放这些空间，成了重中之重，也是所有编程语言设计的难点之一。在计算机语言不断演变过程中，出现了多种流派：

- 垃圾回收机制(GC)，在程序运行时不断寻找不再使用的内存，典型代表：Java、Go
- 手动管理内存的分配和释放, 在程序中，通过函数调用的方式来申请和释放内存，典型代表：C++
- 通过所有权来管理内存，编译器在编译时会根据一系列规则进行检查

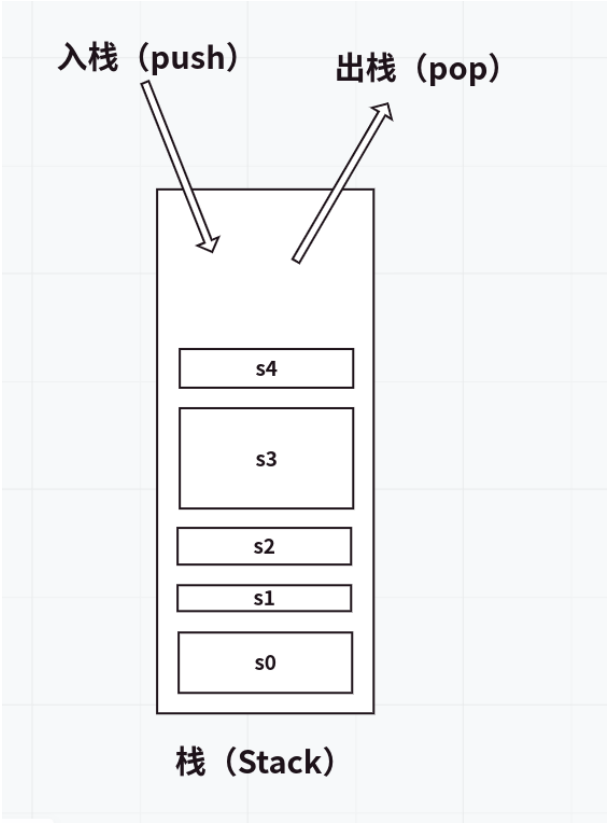
其中 Rust 选择了第三种，最妙的是，这种检查只发生在编译期，因此对于程序运行期，不会有任何性能上的损失。

1. 栈(Stack)与堆(Heap)

1.1. 栈

栈按照顺序存储值并以相反顺序取出值，这也被称作后进先出。

栈中的所有数据都必须占用已知且固定大小的内存空间。



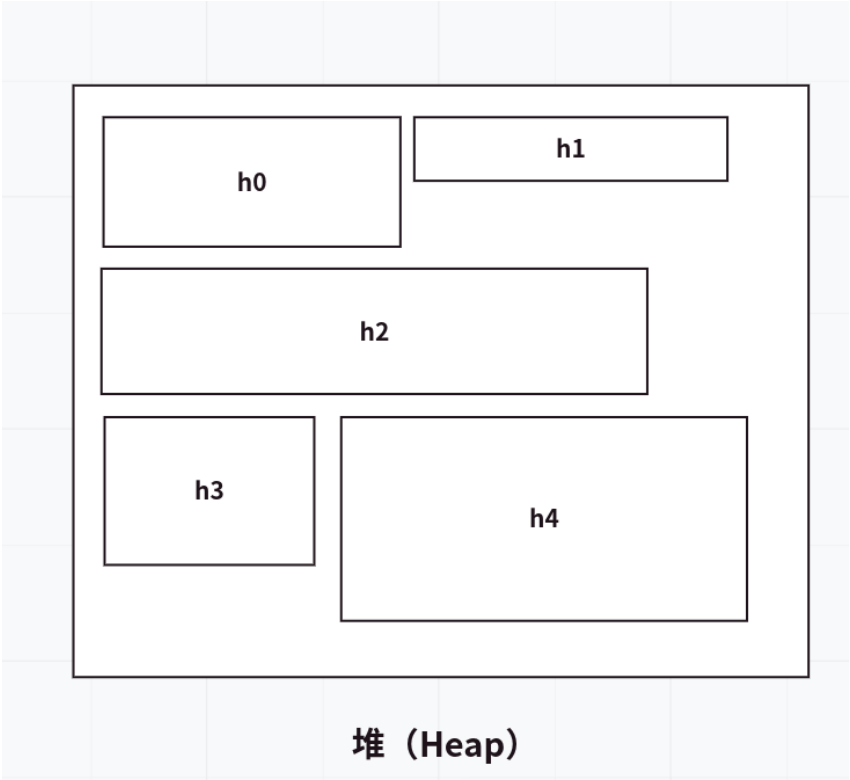
1.2. 堆

与栈不同，对于大小未知或者可能变化的数据，我们需要将它存储在堆上。

当向堆上放入数据时，需要请求一定大小的内存空间。操作系统在堆的某处找到一块足够大的空位，把它标记为已使用，并返回一个表示该位置地址的指针, 该过程被称为在堆上分配内存，有时简称为“分配”(allocating)。

接着，该指针会被推入栈中，因为指针的大小是已知且固定的，在后续使用过程中，你将通过栈中的指针，来获取数据在堆上的实际内存位置，进而访问该数据。

由上可知，堆是一种缺乏组织的数据结构。



1.3. 性能区别

写入方面：入栈比在堆上分配内存要快，因为入栈时操作系统无需分配新的空间，只需要将新数据放入栈顶即可。相比之下，在堆上分配内存则需要更多的工作，这是因为操作系统必须首先找到一块足够存放数据的内存空间，接着做一些记录为下一次分配做准备。

读取方面：得益于 CPU 高速缓存，使得处理器可以减少对内存的访问，高速缓存和内存的访问速度差异在 10 倍以上！栈数据往往可以直接存储在 CPU 高速缓存中，而堆数据只能存储在内存中。访问堆上的数据比访问栈上的数据慢，因为必须先访问栈再通过栈上的指针来访问内存。

因此，处理器处理分配在栈上数据会比在堆上的数据更加高效。

1.4. 所有权与堆栈

当你的代码调用一个函数时，传递给函数的参数（包括可能指向堆上数据的指针和函数的局部变量）依次被压入栈中，当函数调用结束时，这些值将被从栈中按照相反的顺序依次移除。

因为堆上的数据缺乏组织，因此跟踪这些数据何时分配和释放是非常重要的，否则堆上的数据将产生内存泄漏 —— 这些数据将永远无法被回收。这就是 Rust 所有权系统为我们提供的强大保障。

对于其他很多编程语言，你确实无需理解堆栈的原理，但是在 Rust 中，明白堆栈的原理，对于我们理解所有权的工作原理会有很大的帮助。

2. 所有权原则

1. Rust 中每一个值都被一个变量所拥有，该变量被称为值的所有者
2. 一个值同时只能被一个变量所拥有，或者说一个值只能拥有一个所有者
3. 当所有者(变量)离开作用域范围时，这个值将被丢弃(drop)