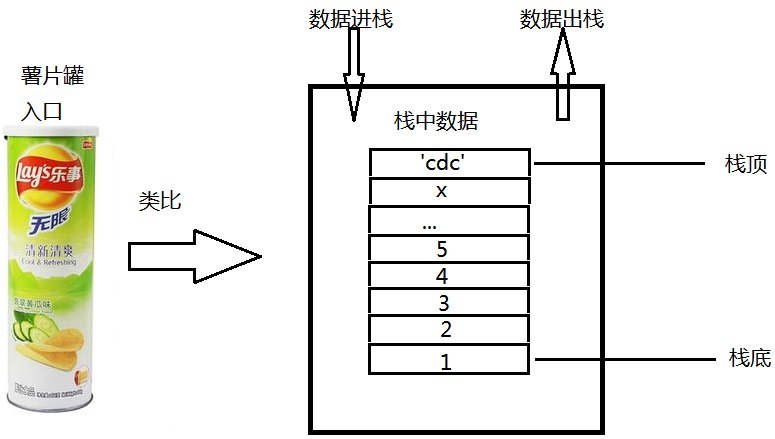
对于前端开发来说，内存空间的概念并没有像其他语言那样被重视，平时涉及到内存相关的情况比较少，通常就是内存溢出之类的。但是并不代表内存空间这块知识并不重要，相反的是，想要对于javascript有一个深刻的认知的话，必须对内存空间有一个清晰的认知！

内存空间中有三种数据结构，分别是堆(heap)，栈(stack)与队列(queue)。

1. 栈数据结构

栈结构是怎样的呢？栈结构具有【先进后出，后进先出】的特点。比如在javascript的执行上下文中就是使用这种栈数据接口，来协调执行顺序(关于执行上下文会在后面文章讲述)。下面我们类比薯片罐加下图描述栈的存取方式：



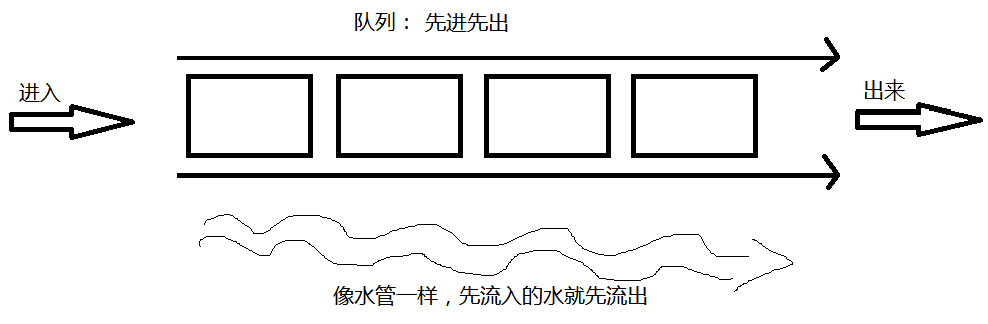
栈数据的存取跟薯片罐类似。放薯片的时候（存数据），最先放进去的薯片位于罐子底部，最后放的薯片位于罐子的顶部。吃薯片的时候（取数据），最先被拿出来的是最后放进去的那一片。如果我们想要拿最后的那片薯片，必须把前面的薯片都先取出来。这就是【先进后出，后进先出】的特点。

1. 堆数据结构

堆数据结构比较类似我们去超市商场用的储物柜。什么意思呢？柜子虽然整整齐齐的摆放在一起，但是我们并不用去按顺序的打开，我们只需要记住自己储物柜的编号，便可以用钥匙打开。类似JSON格式的数据，采用key-val键值对无序的方式。

1. 队列

队列是一种先进先出的数据结构。类似水管流水一样，先进去的水就先流出来，如下图：



经过上面的三个介绍，我们大致了解内存空间的数据结构，接下来，本文将介绍它们在实际中的运用。

**调用堆栈**

由于javascript是一种单线程编程语言，这意味着它只有一个 call stack。Call stack中文意思就是【调用堆栈】。它记录了我们程序中执行的位置。如果我们执行一个函数，则是把它入栈，如果我们从一个函数返回，则是把它弹出堆栈顶部。看下列例子：

function b (x,y) {

Return x+y

}

function a(x) {

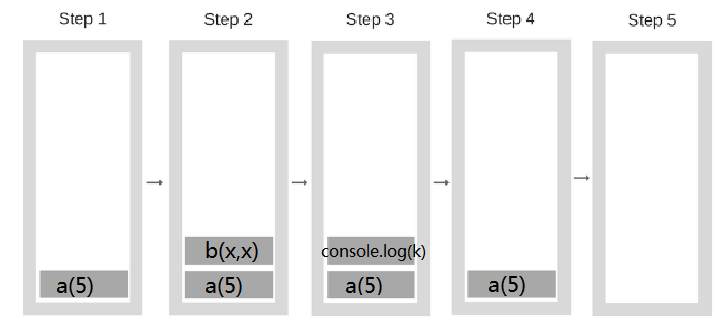
var k = b(x,x)

console.log（b）

}

a(5)

那么当引擎执行该段代码过程如下：



另外，当遇到抛错时，我们也是很容易看到栈结构的顺序的，比如以下代码：

function foo() {

throw new Error('SessionStack will help you resolve crashes :)');

}

function bar() {

foo();

}

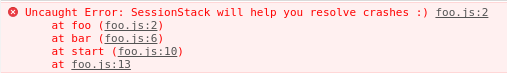
function start() {

bar();

}

start();

假设这段代码在foo.js文件中，并在chrome执行，则会出现以下错误：



**堆栈大小**

还有一种情况是调用堆栈情况达到最大值，比如以下代码：

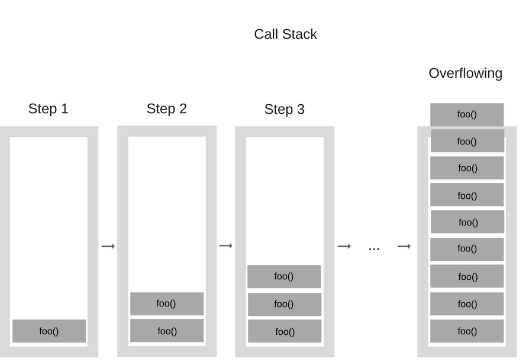
function foo() {

foo();

}

foo();

由于在引擎中执行这个代码时，会先调用foo函数，并且不断的递归调用，没有任何终止调整，所以执行时，同一个函数会不断重复的添加到调用栈中，如下图：



直到最后超出调用堆栈的实际大小，浏览器就会抛出如下错误：

IMG_256

**堆内存与引用数据类型**

在执行上下文中，会生成一个变量对象，拆分理解一下：变量-对象，其实就是一个存储变量的对象。对于javascript中的基础数据类型，如Undefined、Null、Boolean、Number、String，一般是保存在变量对象中，需要访问他们时，都是按值访问，我们可以直接操作保存保存在变量对象中的实际值。

然而对于引用数据类型，如数组Array、对象Object，它们值的大小不是固定的，它们不能按值访问。对于引用数据类型的访问，实际上我们是访问的是变量对象中的一个值，这个值记录着引用数据的在堆内存中的地址。

文字描述总是素然无味，图片插入才比较有画面感，看图与代码：