08-调用栈：为什么JavaScript代码会出现栈溢出？

在上篇文章中，我们讲到了，当一段代码被执行时，JavaScript引擎先会对其进行编译，并创建执行上下文。但是并没有明确说明到底什么样的代码才算符合规范。

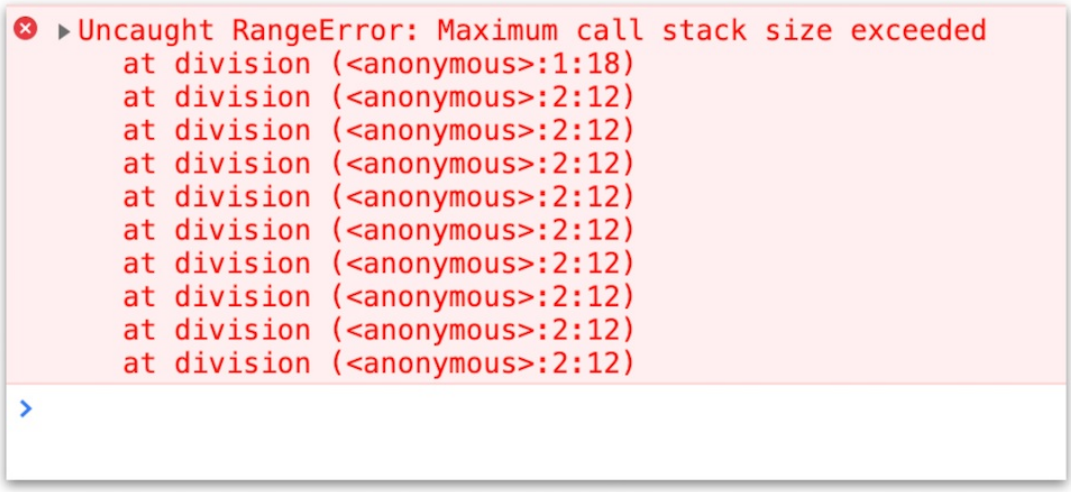
那么接下来我们就来明确下，哪些情况下代码才算是“一段”代码，才会在执行之前就进行编译并创建执行上下文。一般说来，有这么三种情况：

1. 当JavaScript执行全局代码的时候，会编译全局代码并创建全局执行上下文，而且在整个页面的生存周期内，全局执行上下文只有一份。
2. 当调用一个函数的时候，函数体内的代码会被编译，并创建函数执行上下文，一般情况下，函数执行结束之后，创建的函数执行上下文会被销毁。
3. 当使用eval函数的时候，eval的代码也会被编译，并创建执行上下文。

好了，又进一步理解了执行上下文，那本节我们就在这基础之上继续深入，一起聊聊**调用栈**。学习调用栈至少有以下三点好处：

1. 可以帮助你了解JavaScript引擎背后的工作原理；
2. 让你有调试JavaScript代码的能力；
3. 帮助你搞定面试，因为面试过程中，调用栈也是出境率非常高的题目。

比如你在写JavaScript代码的时候，有时候可能会遇到栈溢出的错误，如下图所示：



栈溢出的错误

那为什么会出现这种错误呢？这就涉及到了**调用栈**的内容。你应该知道JavaScript中有很多函数，经常会出现在一个函数中调用另外一个函数的情况，**调用栈就是用来管理函数调用关系的一种数据结构**。因此要讲清楚调用栈，你还要先弄明白**函数调用**和**栈结构**。

什么是函数调用

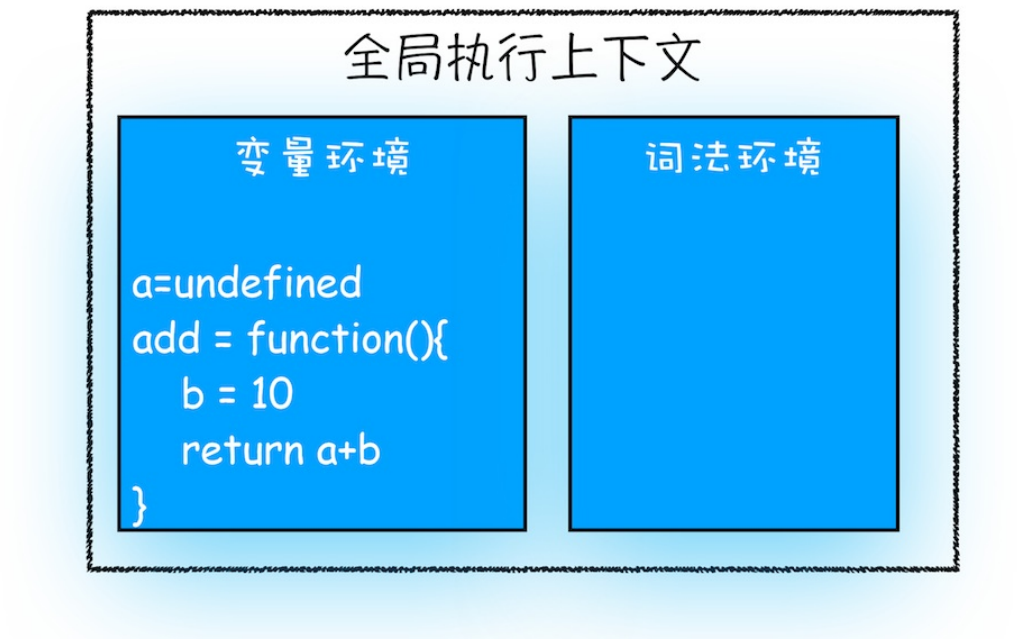
函数调用就是运行一个函数，具体使用方式是使用函数名称跟着一对小括号。下面我们看个简单的示例代码：

```
var a = 2
function add(){
var b = 10
return  a+b
}
add()
```

这段代码很简单，先是创建了一个add函数，接着在代码的最下面又调用了该函数。

那么下面我们就利用这段简单的代码来解释下函数调用的过程。

在执行到函数add()之前，JavaScript引擎会为上面这段代码创建全局执行上下文，包含了声明的函数和变量，你可以参考下图：



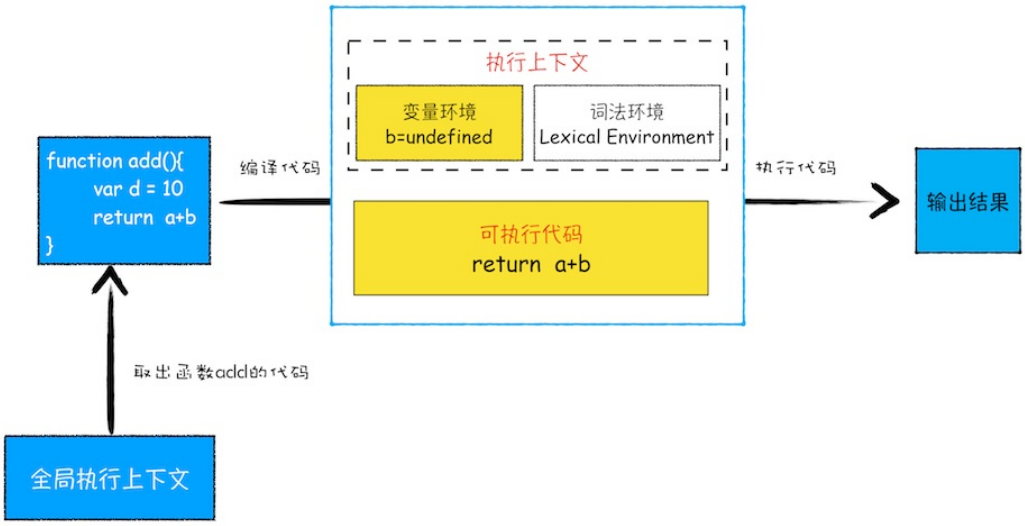
全局执行上下文

从图中可以看出，代码中全局变量和函数都保存在全局上下文的变量环境中。

执行上下文准备好之后，便开始执行全局代码，当执行到add这儿时，JavaScript判断这是一个函数调用，那么将执行以下操作：

- 首先，从**全局执行上下文**中，取出add函数代码。
- 其次，对add函数的这段代码进行编译，并创建**该函数的执行上下文**和**可执行代码**。
- 最后，执行代码，输出结果。

完整流程你可以参考下图：



函数调用过程

就这样，当执行到add函数的时候，我们就有了两个执行上下文了——全局执行上下文和add函数的执行上下文。

也就是说在执行JavaScript时，可能会存在多个执行上下文，那么JavaScript引擎是如何管理这些执行上下文的呢？

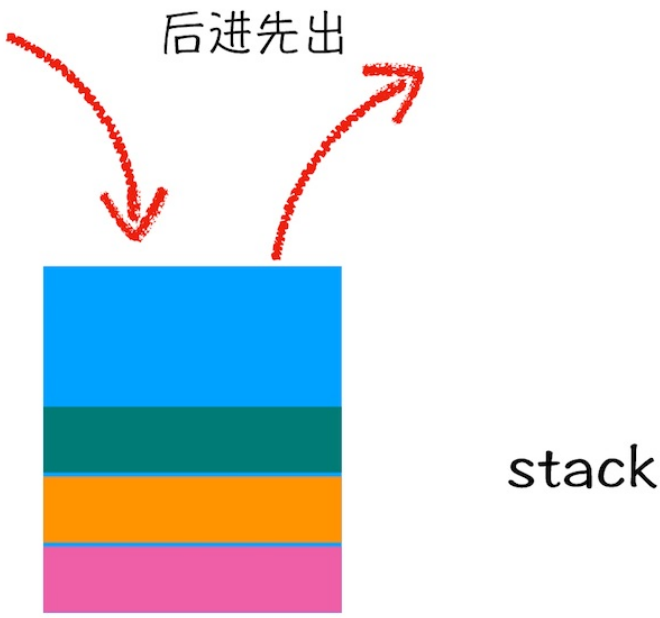
答案是**通过一种叫栈的数据结构来管理的**。那什么是栈呢？它又是如何管理这些执行上下文呢？

什么是栈

关于栈，你可以结合这么一个贴切的例子来理解，一条单车道的单行线，一端被堵住了，而另一端入口处没有任何提示信息，堵住之后就只能后进去的车子先出来，这时这个堵住的单行线就可以被看作是一个**栈容器**，车子开进单行线的操作叫做**入栈**，车子倒出去的操作叫做**出栈**。

在车流量较大的场景中，就会发生反复的入栈、栈满、出栈、空栈和再次入栈，一直循环。

所以，栈就是类似于一端被堵住的单行线，车子类似于栈中的元素，栈中的元素满足**后进先出**的特点。你可以参看下图：



栈示意图

什么是JavaScript的调用栈

JavaScript引擎正是利用栈的这种结构来管理执行上下文的。在执行上下文创建好后，JavaScript引擎会将执行上下文压入栈中，通常把这种用来管理执行上下文的栈称为**执行上下文栈**，又称**调用栈**。

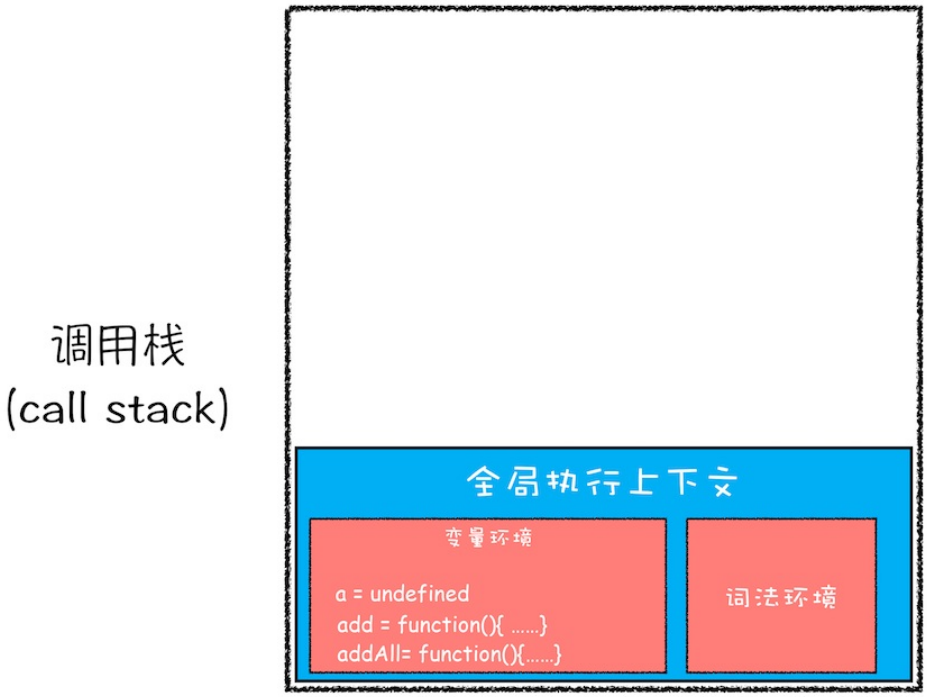
为便于你更好地理解调用栈，下面我们再来看段稍微复杂点的示例代码：

```
var a = 2
function add(b,c){
  return b+c
}
function addAll(b,c){
var d = 10
result = add(b,c)
return  a+result+d
}
addAll(3,6)
```

在上面这段代码中，你可以看到它是在addAll函数中调用了add函数，那在整个代码的执行过程中，调用栈是怎么变化的呢？

下面我们就一步步地分析在代码的执行过程中，调用栈的状态变化情况。

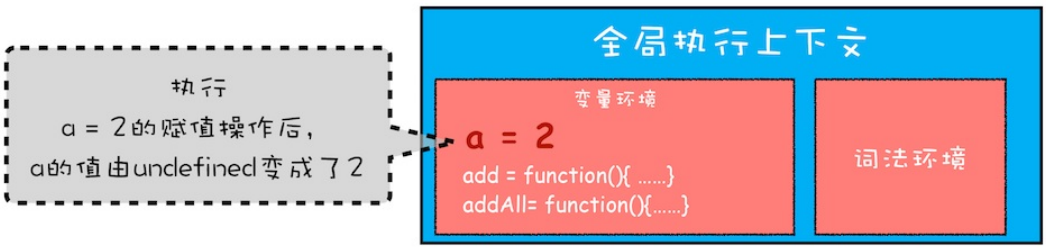
第一步，创建全局上下文，并将其压入栈底。如下图所示：



全局执行上下文压栈

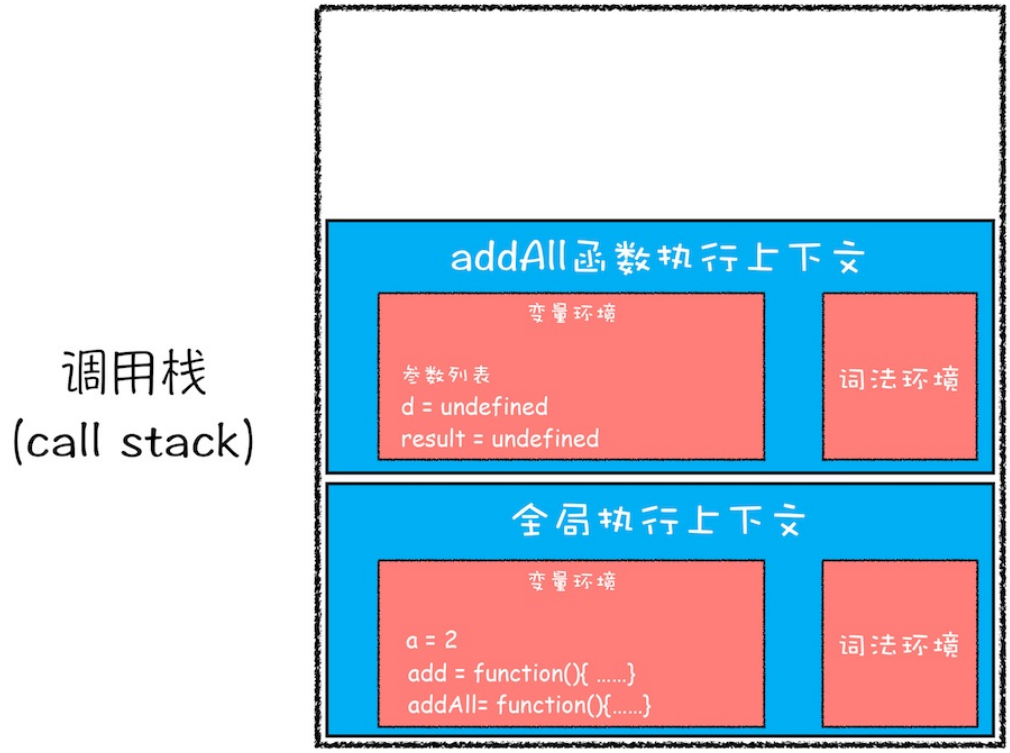
从图中你也可以看出，变量a、函数add和addAll都保存到了全局上下文的变量环境对象中。

全局执行上下文压入到调用栈后，JavaScript引擎便开始执行全局代码了。首先会执行a=2的赋值操作，执行该语句会将全局上下文变量环境中a的值设置为2。设置后的全局上下文的状态如下图所示：



赋值操作改变执行上下文中的值

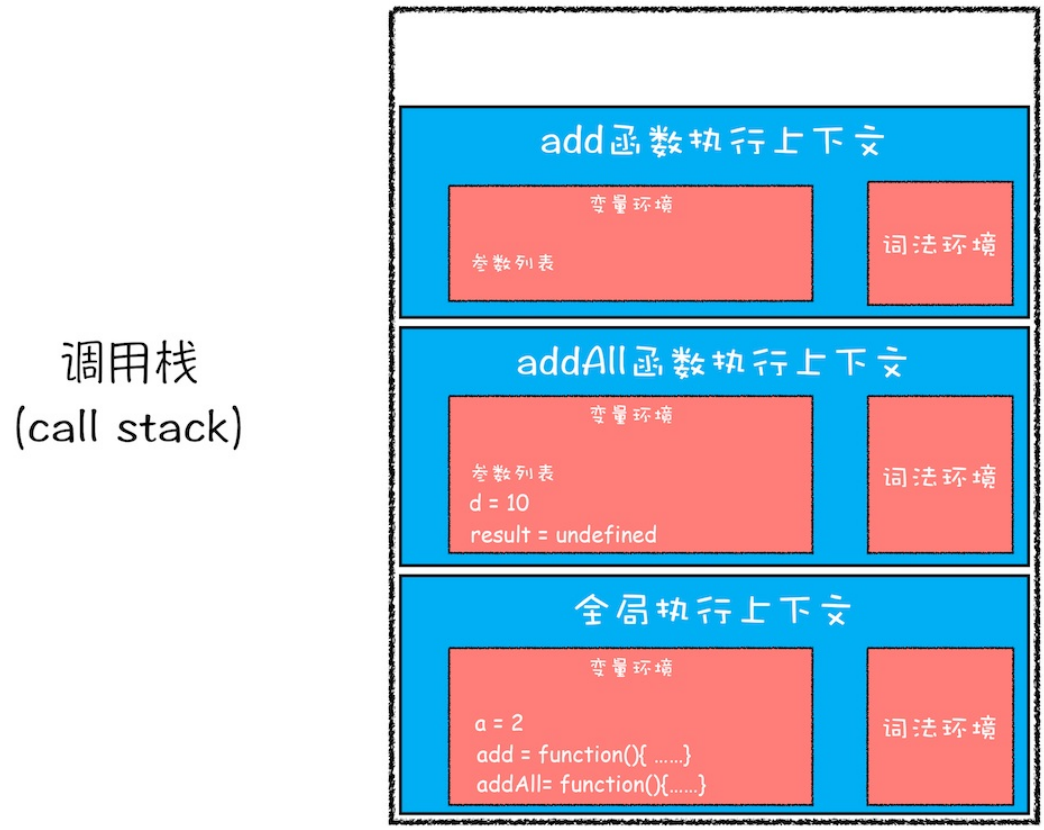
接下来，**第二步是调用addAll函数**。当调用该函数时，JavaScript引擎会编译该函数，并为其创建一个执行上下文，最后还将该函数的执行上下文压入栈中，如下图所示：



执行addAll函数时的调用栈

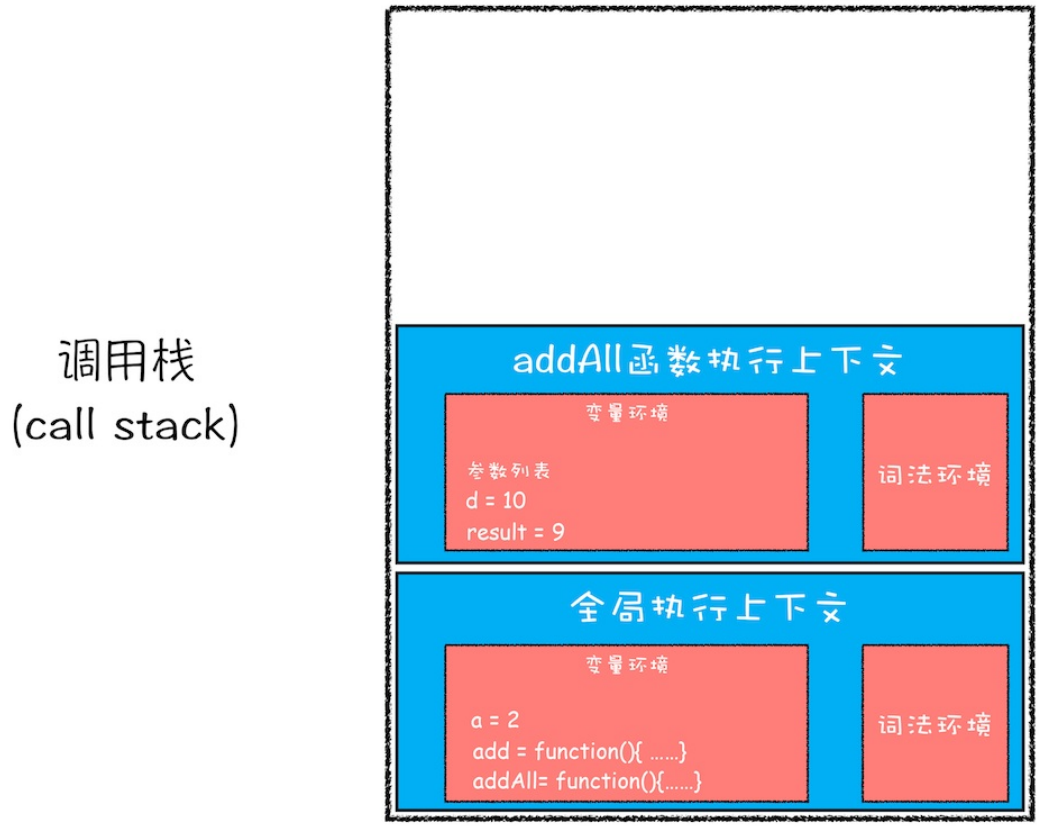
addAll函数的执行上下文创建好之后，便进入了函数代码的执行阶段了，这里先执行的是d=10的赋值操作，执行语句会将addAll函数执行上下文中的d由undefined变成了10。

然后接着往下执行，**第三步，当执行到add函数**调用语句时，同样会为其创建执行上下文，并将其压入调用栈，如下图所示：



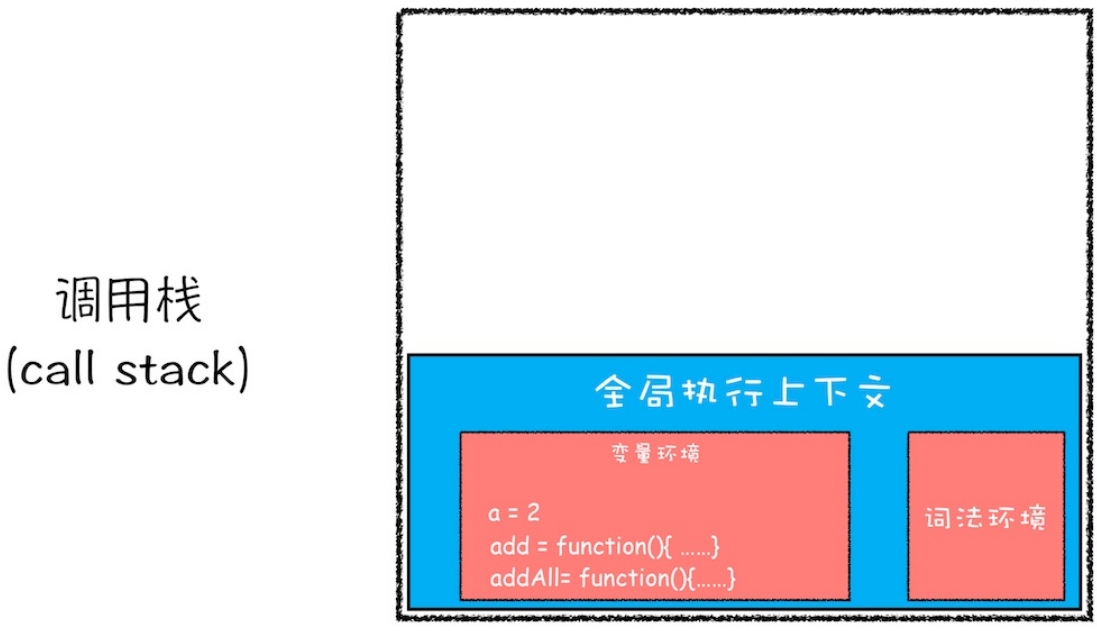
执行add函数时的调用栈

当add函数返回时，该函数的执行上下文就会从栈顶弹出，并将result的值设置为add函数的返回值，也就是9。如下图所示：



add函数执行结束时的调用栈

紧接着addAll执行最后一个相加操作后并返回，addAll的执行上下文也会从栈顶部弹出，此时调用栈中就只剩下全局上下文了。最终如下图所示：



addAll函数执行结束时的调用栈

至此，整个JavaScript流程执行结束了。

好了，现在你应该知道了**调用栈是JavaScript引擎追踪函数执行的一个机制**，当一次有多个函数被调用时，通过调用栈就能够追踪到哪个函数正在被执行以及各函数之间的调用关系。

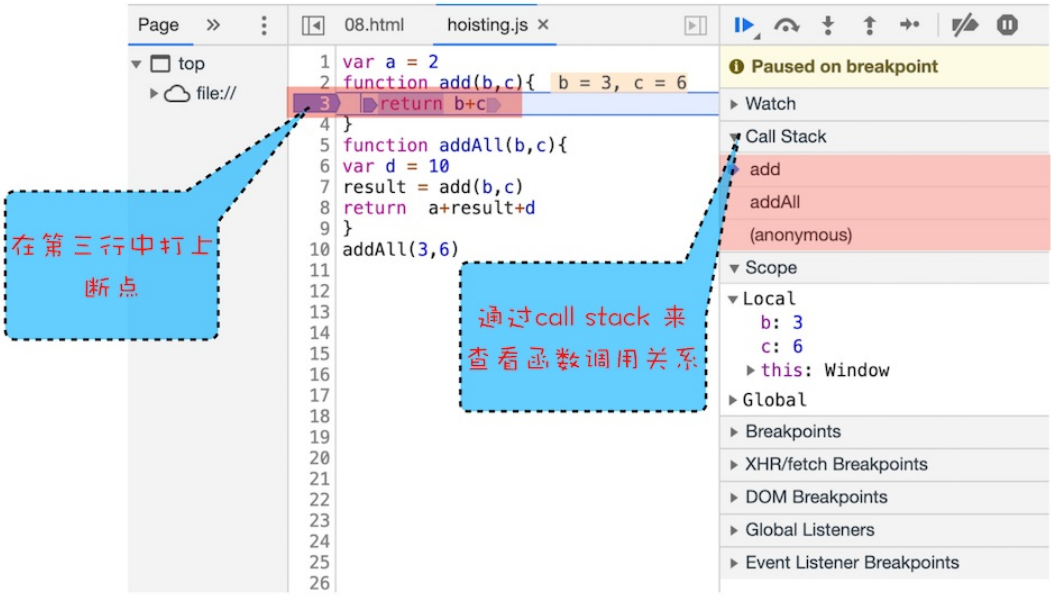
在开发中，如何利用好调用栈

鉴于调用栈的重要性和实用性，那么接下来我们就一起来看看在实际工作中，应该如何查看和利用好调用栈。

1. 如何利用浏览器查看调用栈的信息

当你执行一段复杂的代码时，你可能很难从代码文件中分析其调用关系，这时候你可以在你想要查看的函数中加入断点，然后当执行到该函数时，就可以查看该函数的调用栈了。

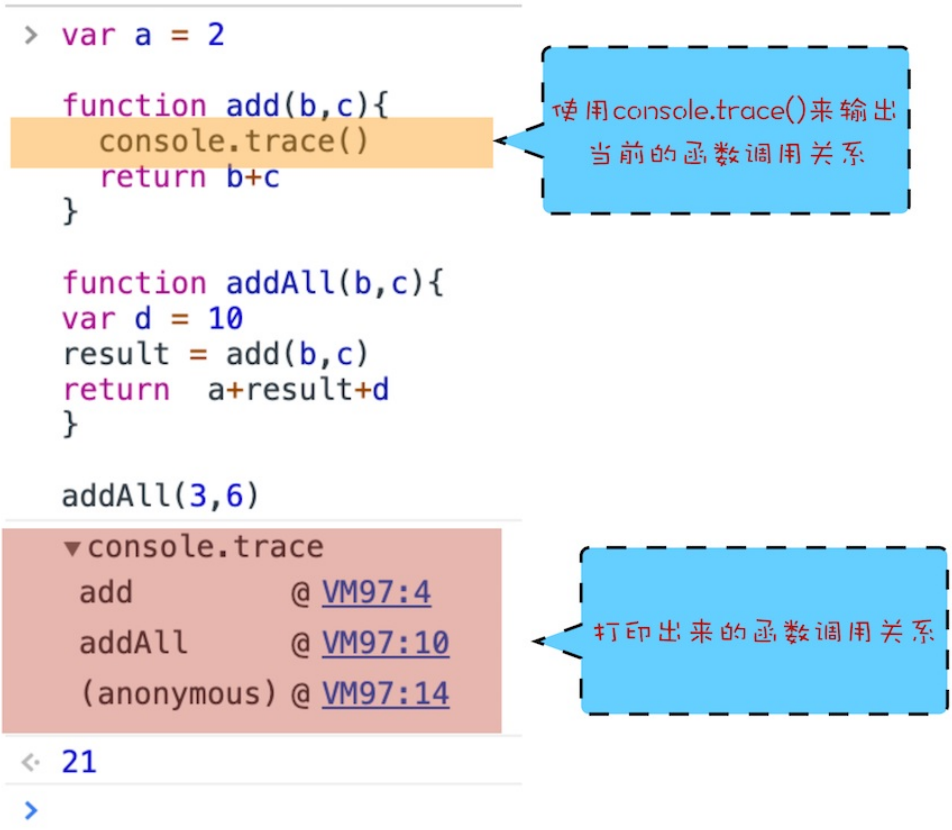
这么说可能有点抽象，这里我们拿上面的那段代码做个演示，你可以打开“开发者工具”，点击“Source”标签，选择JavaScript代码的页面，然后在第3行加上断点，并刷新页面。你可以看到执行到add函数时，执行流程就暂停了，这时可以通过右边“call stack”来查看当前的调用栈的情况，如下图：



查看函数调用关系

从图中可以看出，右边的“call stack”下面显示出来了函数的调用关系：栈的最底部是anonymous，也就是全局的函数入口；中间是addAll函数；顶部是add函数。这就清晰地反映了函数的调用关系，所以**在分析复杂结构代码，或者检查Bug时，调用栈都是非常有用的**。

除了通过断点来查看调用栈，你还可以使用console.trace()来输出当前的函数调用关系，比如在示例代码中的add函数里面加上了console.trace()，你就可以看到控制台输出的结果，如下图：



使用trace函数输出当前调用栈信息

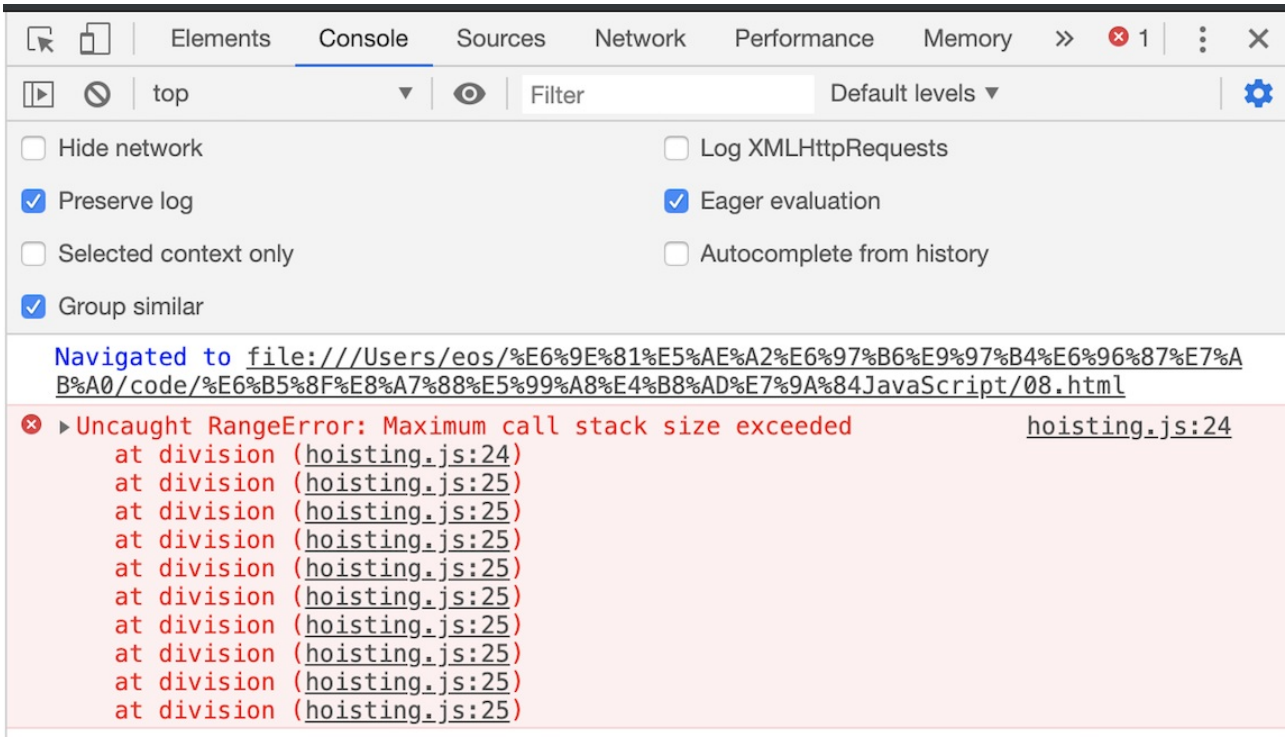
2. 栈溢出（Stack Overflow）

现在你知道了调用栈是一种用来管理执行上下文的数据结构，符合后进先出的规则。不过还有一点你要注意，**调用栈是有大小的**，当入栈的执行上下文超过一定数目，JavaScript引擎就会报错，我们把这种错误叫做**栈溢出**。

特别是在你写递归代码的时候，就很容易出现栈溢出的情况。比如下面这段代码：

```
function division(a,b){
    return division(a,b)
}
console.log(division(1,2))
```

当执行时，就会抛出栈溢出错误，如下图：



栈溢出错误

从上图你可以看到，抛出的错误信息为：超过了最大栈调用大小（Maximum call stack size exceeded）。

那为什么会出现这个问题呢？这是因为当JavaScript引擎开始执行这段代码时，它首先调用函数division，并创建执行上下文，压入栈中；然而，这个函数是**递归的，并且没有任何终止条件**，所以它会一直创建新的函数执行上下文，并反复将其压入栈中，但栈是有容量限制的，超过最大数量后就会出现栈溢出的错误。

理解了栈溢出原因后，你就可以使用一些方法来避免或者解决栈溢出的问题，比如把递归调用的形式改造成其他形式，或者使用加入定时器的方法来把当前任务拆分为其他很多小任务。

总结

好了，今天的内容就讲到这里，下面来总结下今天的内容。

- 每调用一个函数，JavaScript引擎会为其创建执行上下文，并把该执行上下文压入调用栈，然后JavaScript引擎开始执行函数代码。
- 如果在一个函数A中调用了另外一个函数B，那么JavaScript引擎会为B函数创建执行上下文，并将B函数的执行上下文压入栈顶。
- 当前函数执行完毕后，JavaScript引擎会将该函数的执行上下文弹出栈。
- 当分配的调用栈空间被占满时，会引发“堆栈溢出”问题。

栈是一种非常重要的数据结构，不光应用在JavaScript语言中，其他的编程语言，如C/C++、Java、Python等语言，在执行过程中也都使用了栈来管理函数之间的调用关系。所以栈是非常基础且重要的知识点，你必须得掌握。

思考时间

最后，我给你留个思考题，你可以看下面这段代码：

```
function runStack (n) {
  if (n === 0) return 100;
  return runStack( n- 2);
}
runStack(50000)
```

这是一段递归代码，可以通过传入参数n，让代码递归执行n次，也就意味着调用栈的深度能达到n，当输入一个较大的数时，比如50000，就会出现栈溢出的问题，那么你能优化下这段代码，以解决栈溢出的问题吗？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

精选留言：

- ytd 2019-08-22 08:04:33
  改成循环不会栈溢出了，不过就有可能陷入死循环：
  // 优化
  function runStack(n) {
  while (true) {
  if (n === 0) {
  return 100;
  }

  if (n === 1) { // 防止陷入死循环
  return 200;
  }

  n = n - 2;
  }

}

console.log(runStack(50000)); [2赞]

- mfist 2019-08-22 07:15:47
1. 改成尾递归调用（需要在严格模式下面生效）
function runStack (n, result=100) {
if (n === 0) return result;
return runStack(n- 2, result);
}
runStack(50000, 100)
2. 改成循环调用，不使用递归函数，就不存在堆栈溢出 [2赞]

- Marvin 2019-08-22 08:29:30
这个代码运行情况依赖入参，有三种情况：1、n=0，返回100；2、n为正偶数，递归n/2次之后返回100；3、n为非上述情况，栈溢出。优化方案：判断参数n，1、2两种情况返回100，3的情况抛错。 [1赞]

- Hurry 2019-08-22 05:24:29
将递归，改成循环:
```
function runStack(n) {
if (n === 0)
return 100;
}

function run(n) {
while (n > 0) {
runStack(n)
n = n - 2;
}

return runStack(n)
}

run(50000)
``` [1赞]

- 许童童 2019-08-22 11:41:49
老师你好，如果函数中有闭包，那执行上下文就不会被弹出了，这是一种什么情况?
栈的大小具体是多大，哪里可以看?
老师的图画得很好，用的是什么软件?

  作者回复2019-08-22 12:39:03
  图用mac自带的keynote画的，我画的的比较原始，编辑MM帮整理过的。

  关于闭包，执行结束后执行上下文依然会从调用栈中弹出来，但是相关内容不会销毁的。

  第十节就要来详细讨论闭包了!

- 许童童 2019-08-22 11:40:22

将递归改成迭代就好了，还可以使用尾递归优化。感觉老师这道题改成斐波那契数列会更好。

```
function runStack (n) {
while (n > 0) {
n -= 2
}
return 100
}
runStack(50000)
```

- Chao 2019-08-22 11:08:26
递归是比较好理解的一种方式。

runstack 目的是否最后能被减为0 return 100。
或者直接改循环 递减

- 梦飞 2019-08-22 10:40:34
```
function callStack(n){
if(n<1) return 1090;
return callStack(n/2)
},
callStack(5000000)
```

- Jim 2019-08-22 09:43:34
老师，您的执行上下文图里都会有一个变量环境和词法环境，可是为什么词法环境没有东西呢？请问变量环境和词法环境的区别是什么呢？

作者回复2019-08-22 09:53:48
词法环境下节就开始介绍了

- 徐承银 2019-08-22 09:26:40
不进栈，就不会栈溢出了。function runStack (n) {
if (n === 0) return 100;
return setTimeout(function(){runStack(n- 2)},0);
}
runStack(50000)

- William 2019-08-22 00:37:43
没太明白老师说的优化是什么意思，改造了一下，去掉了递归的使用。

```
function runStack (n) {
if (n > 0 && (n&1)=== 0){
return 100
} else {
throw new Error("illegal input!")
}
}
```