你好，我是李兵。

在前面我们介绍了JavaScript的继承是基于原型链的，原型链将一个个原型对象串起来，从而实现对象属性的查找，今天我们要聊一个和原型链类似的话题，那就是作用域链。

作用域链就是将一个个作用域串起来，实现变量查找的路径。讨论作用域链，实际就是在讨论按照什么路径查找变量的问题。

我们知道，作用域就是存放变量和函数的地方，全局环境有全局作用域，全局作用域中存放了全局变量和全局函数。每个函数也有自己的作用域，函数作用域中存放了函数中定义的变量。

当在函数内部使用一个变量的时候，V8便会去作用域中去查找。我们通过一段在函数内部查找变量的代码来具体看一下：

```
var name = '极客时间'
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

在这段代码中，我们在全局环境中声明了变量name和type，同时还定义了bar函数和foo函数，在bar函数中又再次定义了变量name和type，在foo函数中再次定义了变量name。

函数的调用关系是：在全局环境中调用bar函数，在bar函数中调用foo函数，在foo函数中打印出来变量name和type的值。

当执行到foo函数时，首先需要打印出变量name的值，而我们在三个地方都定义了变量name，那么究竟应该使用哪个变量呢？

在foo函数中使用了变量name，那么V8就应该先使用foo函数内部定义的变量name，最终的结果确实如此，也符合我们的直觉。

接下来，foo函数继续打印变量type，但是在foo函数内部并没有定义变量type，而是在全局环境中和调用foo函数的bar函数中分别定义了变量type，那么这时候的问题来了，你觉得foo函数中打印出来的变量type是bar函数中的，还是全局环境中的呢？

什么是函数作用域和全局作用域？

要解释清楚这个问题，我们需要从作用域的工作原理讲起。

每个函数在执行时都需要查找自己的作用域，我们称为函数作用域，在执行阶段，在执行一个函数时，当该函数需要使用某个变量或者调用了某个函数时，便会优先在该函数作用域中查找相关内容。

我们再来看一段代码：

```
var x = 4
var test
function test_scope() {
    var name = 'foo'
    console.log(name)
    console.log(type)
    console.log(test)
    var type = 'function'
    test = 1
    console.log(x)
}
test_scope()
```

在上面的代码中，我们定义了一个test_scope函数，那么在V8执行test_scope函数的时候，在编译阶段会为test_scope函数创建一个作用域，在test_scope函数中定义的变量和声明的函数都会丢到该作用域中，因为我们在test_scope函数中定了三个变量，那么常见的作用域就包含有这三个变量。

你可以通过Chrome的控制台来直观感受下test_scope函数的作用域，先打开包含这段代码的页面，然后打开开发者工具，接着在test_scope函数中的第二段代码加上断点，然后刷新该页面。当执行到该断点时，V8会暂停整个执行流程，这时候我们就可以通过右边的区域面板来查看当前函数的执行状态。

你可以参考图中右侧的Scope项，然后点击展开该项，这个Local就是当前函数test_scope的作用域。在test_scope函数中定义的变量都包含到了Local中，如变量name、type，另外系统还为我们添加了另外一个隐藏变量this，V8还会默认将隐藏变量this存放到作用域中。

另外你还需要注意下，第一个test1，我并没有采用var等关键字来声明，所以test1并不会出现在test_scope函数的作用域中，而是属于this所指向的对象。（this的工作机制不是本文讨论的重点，不展开介绍。如果你感兴趣，可以在《浏览器工作原理与实践》专栏中《11 | this：从JavaScript执行上下文的视角讲清楚this》这一讲查看。）

那么另一个问题来了，我在test_scope函数使用了变量x，但是在test_scope函数的作用域中，并没有定义变量x，那么V8应该如何获取变量x?

如果在当前函数作用域中没有查找到变量，那么V8会去全局作用域中去查找，这个查找的线路就称为作用域链。

全局作用域和函数作用域类似，也是存放变量和函数的地方，但是它们还是有点不一样：**全局作用域是在V8启动过程中就创建了，且一直保存在内存中不会被销毁的，直至V8退出。 而函数作用域是在执行该函数时创建的，当函数执行结束之后，函数作用域就随之被销毁掉了。**

全局作用域中包含了很多全局变量，比如全局的this值，如果是浏览器，全局作用域中还有window、document、opener等非常多的方法和对象，如果是node环境，那么会有Global、File等内容。

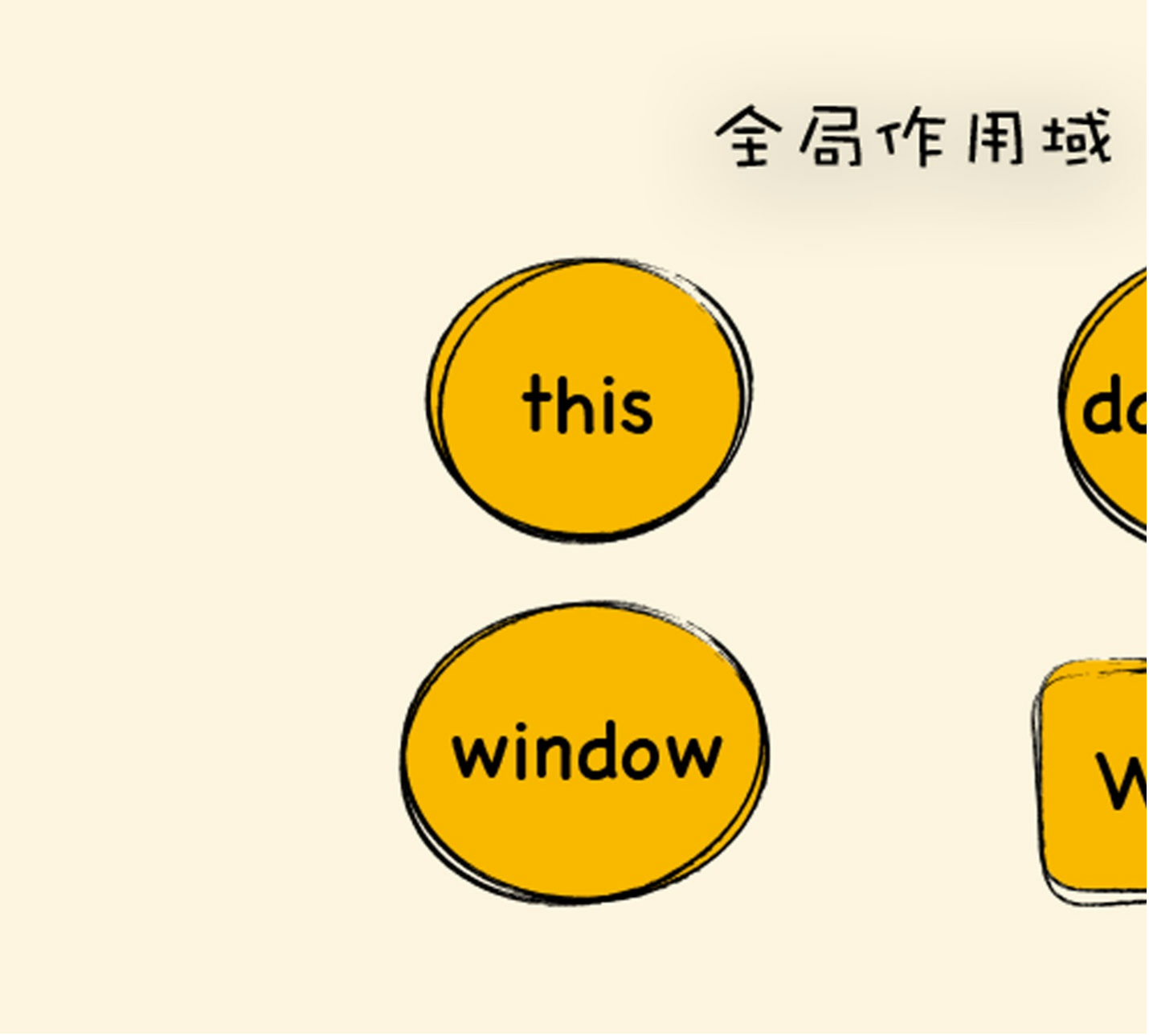
V8启动之后就进入正常的消息循环状态，这时候就可以执行代码了，比如执行到上面那段脚本时，V8会先解析顶层(Top Level)代码，我们可以看到，在顶层代码中定义了变量x，这时候V8就会将变量x添加到全局作用域中。

作用域链是怎么工作的？

理解了作用域和作用域链，我们再回过头来看文章开头的那道思考题：“foo函数中打印出来的变量type是bar函数中的呢，还是全局环境中的呢?”我把这段代码复制到下面：

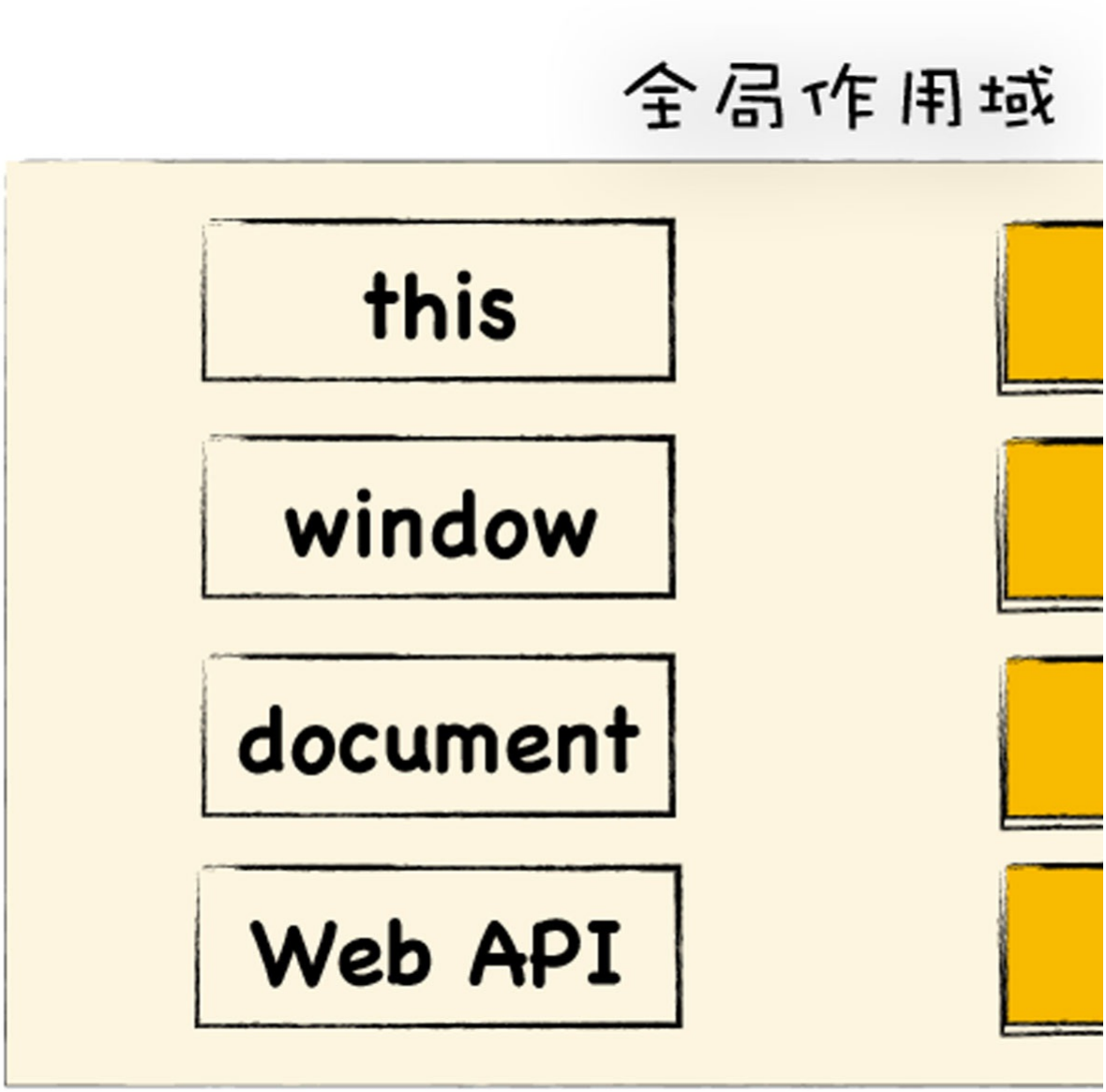
```
var name = '极客时间'
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

现在，我们结合V8执行这段代码的流程来具体分析下。首先当V8启动时，会创建全局作用域，全局作用域中包括了this、window等变量，还有一些全局的Web API接口，创建的作用域如下图所示：



V8启动之后，消息循环系统便开始工作了，这时候，我输入了这段代码，让其执行。

V8会先编译顶层代码，在编译过程中会将顶层定义的变量和声明的函数都添加到全局作用域中，最终的全局作用域如下图所示：

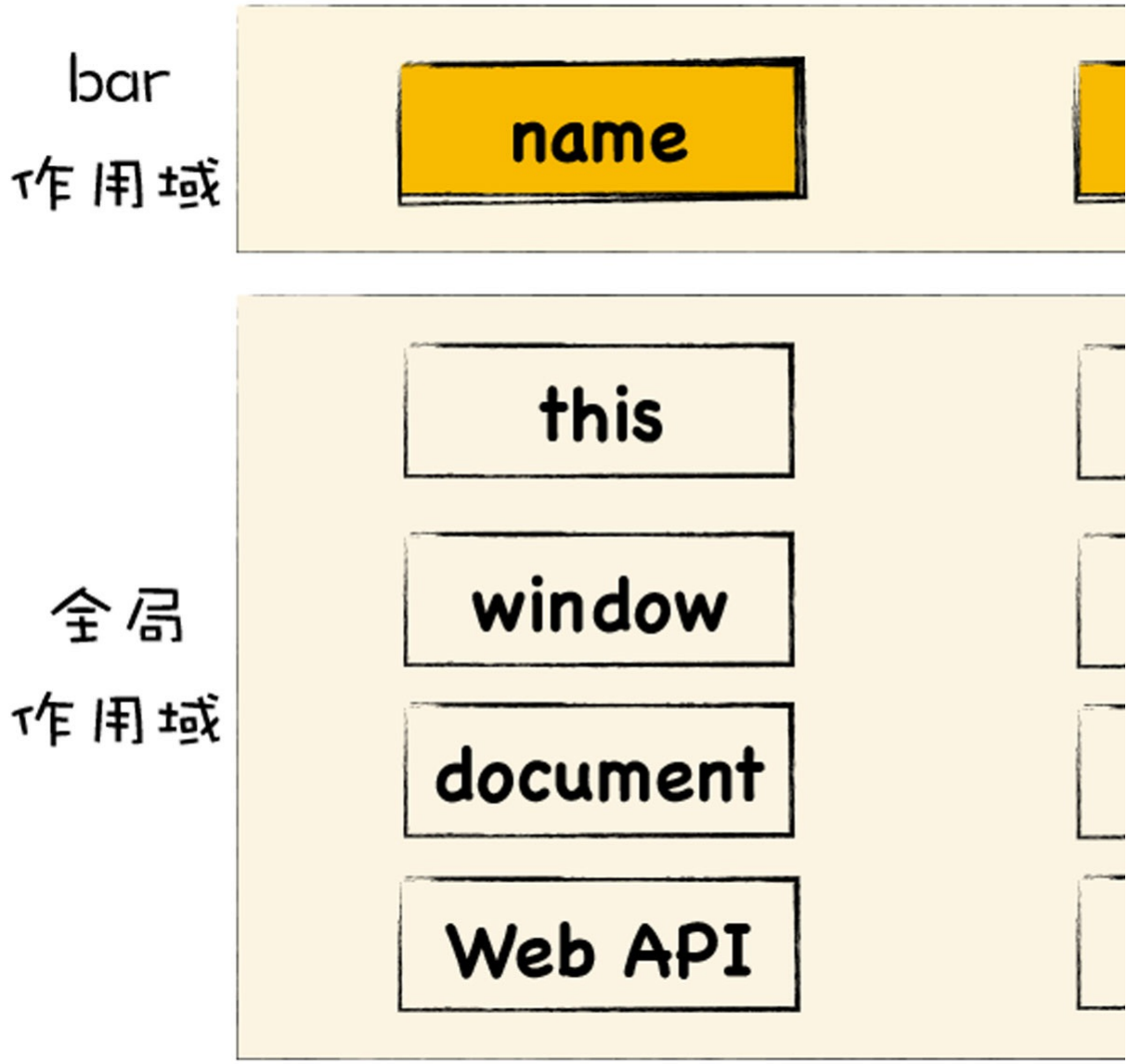


全局作用域创建完成之后，V8便进入了执行状态。前面我们介绍了变量提升，因为变量提升的原因，你可以把上面这段代码分解为如下两个部分：

```
//======解析阶段--实现变量提升=======
var name = undefined
var type = undefined
function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}
function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}



//====执行阶段========
name = '极客时间'
type = 'global'
bar()
```

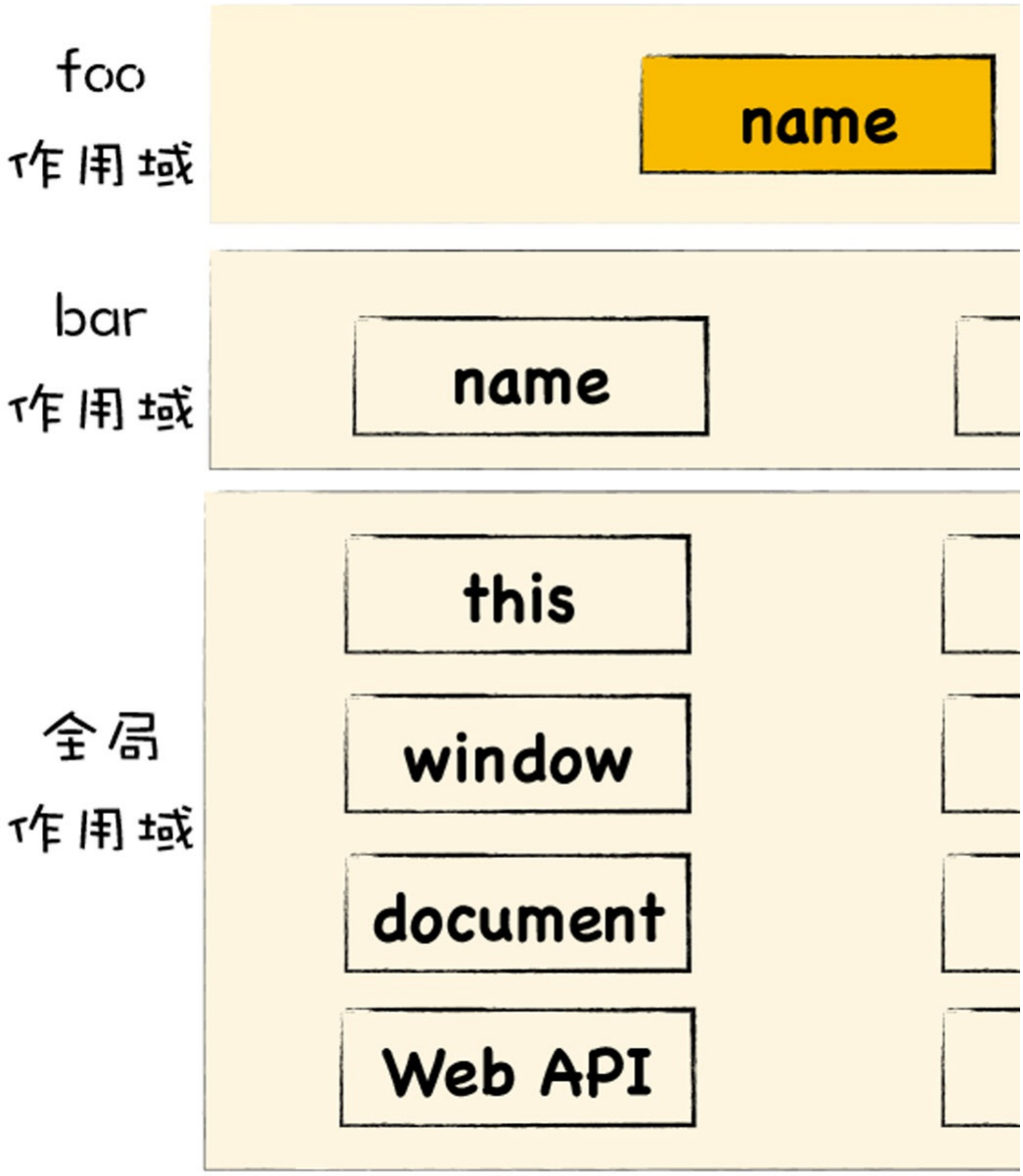
第一部分是在编译过程中完成的，此时全局作用中两个变量的值依然是undefined，然后进入执行阶段；第二部代码就是执行时的顺序，首先全局作用域中的两个变量赋值“极客时间”和“global”，然后就开始执行函数bar的调用了。

当V8执行bar函数的时候，同样需要经历两个阶段：编译和执行。在编译阶段，V8会为bar函数创建函数作用域，最终效果如下所示：



然后进入了bar函数的执行阶段。在bar函数中，只是简单地调用foo函数，因此V8又开始执行foo函数了。

同样，在编译foo函数的过程中，会创建foo函数的作用域，最终创建效果如下图所示：



好了，这时候我们就有了三个作用域了，分别是全局作用域、bar的函数作用域、foo的函数作用域。

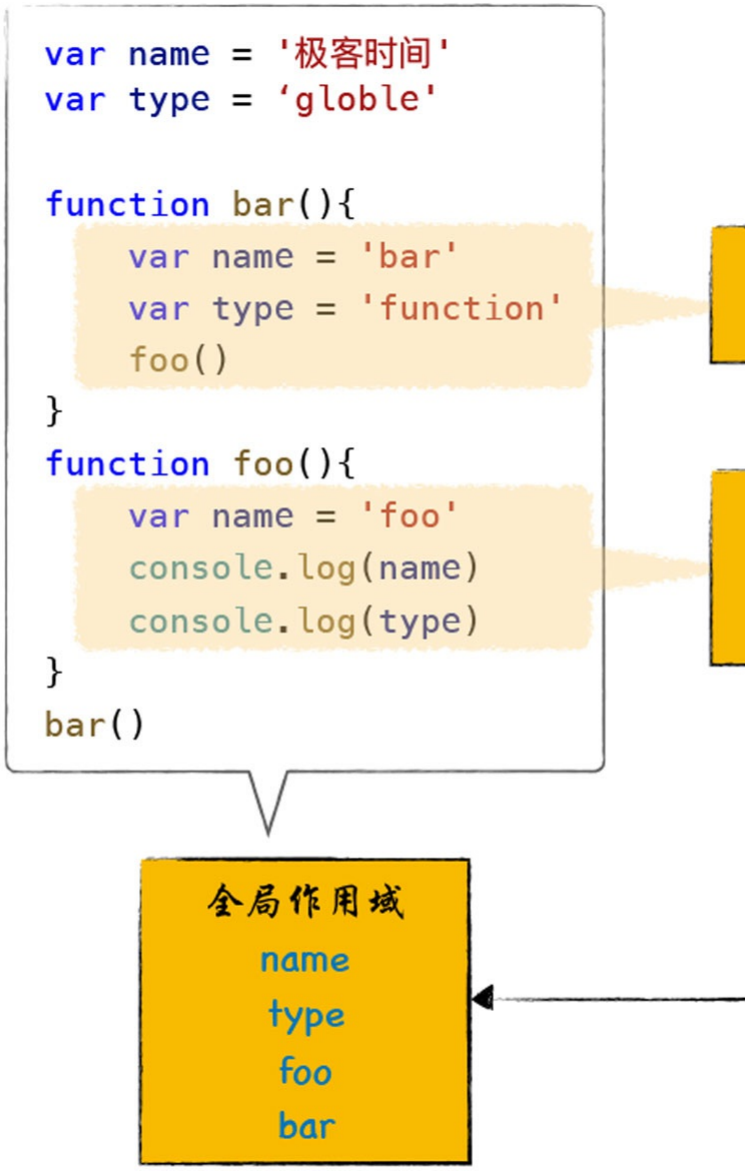
现在我们就可以将刚才提到的问题转换为作用域链的问题了：foo函数查找变量的路径到底是什么？

- 沿着foo函数作用域–>bar函数作用域–>全局作用域;
- 还是，沿着foo函数作用域—>全局作用域？

因为JavaScript是基于词法作用域的，词法作用域就是指，查找作用域的顺序是按照函数定义时的位置来决定的。bar和foo函数的外部代码都是全局代码，所以无论你是在bar函数中查找变量，还是在foo函数中查找变量，其查找顺序都是按照当前函数作用域–>全局作用域这个路径来的。

由于我们代码中的foo函数和bar函数都是在全局下面定义的，所以在foo函数中使用了type，最终打印出来的值就是全局作用域中的type。

你可以参考下面这张图：



另外，我再展开说一些。因为词法作用域是根据函数在代码中的位置来确定的，作用域是在声明函数时就确定好的了，所以我们也将词法作用域称为静态作用域。

和静态作用域相对的是动态作用域，动态作用域并不关心函数和作用域是如何声明以及在何处声明的，只关心它们从**何处调用**。换句话说，作用域链是基于调用栈的，而不是基于函数定义的位置的。（动态作用域不是本文讨论的重点，如果你感兴趣，可以参考《浏览器工作原理与实践》专栏中的《10 | 作用域链和闭包：代码中出现相同的变量，JavaScript引擎是如何选择的？》这一节。）

总结

今天，我们主要解释了一个问题，那就是在一个函数中，如果使用了一个变量，或者调用了另外一个函数，V8将会怎么去查找该变量或者函数。

为了解释清楚这个问题，我们引入了作用域的概念。作用域就是用来存放变量和函数的地方，全局作用域中存放了全局环境中声明的变量和函数，函数作用域中存放了函数中声明的变量和函数。当在某个函数中使用某个变量时，V8就会去这些作用域中查找相关变量。沿着这些作用域查找的路径，我们就称为作用域链。

要了解查找路径，我们需要明白词法作用域，词法作用域是按照代码定义时的位置决定的，而JavaScript所采用的作用域机制就是词法作用域，所以作用域链的路径就是按照词法作用域来实现的。

思考题

我将文章开头那段代码稍微调整了下，foo函数并不是在全局环境中声明的，而是在bar函数中声明的，改造后的代码如下所示：

```
var name = '极客时间'
var type = 'global'
function bar() {
    var type = 'function'
    function foo() {
        console.log(type)
    }
    foo()
}
bar()
```

那么执行这段代码之后，打印出来的内容是什么？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

在前面我们介绍了JavaScript的继承是基于原型链的，原型链将一个个原型对象串起来，从而实现对象属性的查找，今天我们要聊一个和原型链类似的话题，那就是作用域链。

作用域链就是将一个个作用域串起来，实现变量查找的路径。讨论作用域链，实际就是在讨论按照什么路径查找变量的问题。

我们知道，作用域就是存放变量和函数的地方，全局环境有全局作用域，全局作用域中存放了全局变量和全局函数。每个函数也有自己的作用域，函数作用域中存放了函数中定义的变量。

当在函数内部使用一个变量的时候，V8便会去作用域中去查找。我们通过一段在函数内部查找变量的代码来具体看一下：

```
var name = '极客时间'
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

在这段代码中，我们在全局环境中声明了变量name和type，同时还定义了bar函数和foo函数，在bar函数中又再次定义了变量name和type，在foo函数中再次定义了变量name。

函数的调用关系是：在全局环境中调用bar函数，在bar函数中调用foo函数，在foo函数中打印出来变量name和type的值。

当执行到foo函数时，首先需要打印出变量name的值，而我们在三个地方都定义了变量name，那么究竟应该使用哪个变量呢？

在foo函数中使用了变量name，那么V8就应该先使用foo函数内部定义的变量name，最终的结果确实如此，也符合我们的直觉。

接下来，foo函数继续打印变量type，但是在foo函数内部并没有定义变量type，而是在全局环境中和调用foo函数的bar函数中分别定义了变量type，那么这时候的问题来了，你觉得foo函数中打印出来的变量type是bar函数中的，还是全局环境中的呢？

什么是函数作用域和全局作用域？

要解释清楚这个问题，我们需要从作用域的工作原理讲起。

每个函数在执行时都需要查找自己的作用域，我们称为函数作用域，在执行阶段，在执行一个函数时，当该函数需要使用某个变量或者调用了某个函数时，便会优先在该函数作用域中查找相关内容。

我们再来看一段代码：

```
var x = 4
var test
function test_scope() {
    var name = 'foo'
    console.log(name)
    console.log(type)
    console.log(test)
    var type = 'function'
    test = 1
    console.log(x)
}
test_scope()
```

在上面的代码中，我们定义了一个test_scope函数，那么在V8执行test_scope函数的时候，在编译阶段会为test_scope函数创建一个作用域，在test_scope函数中定义的变量和声明的函数都会丢到该作用域中，因为我们在test_scope函数中定了三个变量，那么常见的作用域就包含有这三个变量。

你可以通过Chrome的控制台来直观感受下test_scope函数的作用域，先打开包含这段代码的页面，然后打开开发者工具，接着在test_scope函数中的第二段代码加上断点，然后刷新该页面。当执行到该断点时，V8会暂停整个执行流程，这时候我们就可以通过右边的区域面板来查看当前函数的执行状态。

你可以参考图中右侧的Scope项，然后点击展开该项，这个Local就是当前函数test_scope的作用域。在test_scope函数中定义的变量都包含到了Local中，如变量name、type，另外系统还为我们添加了另外一个隐藏变量this，V8还会默认将隐藏变量this存放到作用域中。

另外你还需要注意下，第一个test1，我并没有采用var等关键字来声明，所以test1并不会出现在test_scope函数的作用域中，而是属于this所指向的对象。（this的工作机制不是本文讨论的重点，不展开介绍。如果你感兴趣，可以在《浏览器工作原理与实践》专栏中《11 | this：从JavaScript执行上下文的视角讲清楚this》这一讲查看。）

那么另一个问题来了，我在test_scope函数使用了变量x，但是在test_scope函数的作用域中，并没有定义变量x，那么V8应该如何获取变量x?

如果在当前函数作用域中没有查找到变量，那么V8会去全局作用域中去查找，这个查找的线路就称为作用域链。

全局作用域和函数作用域类似，也是存放变量和函数的地方，但是它们还是有点不一样：**全局作用域是在V8启动过程中就创建了，且一直保存在内存中不会被销毁的，直至V8退出。 而函数作用域是在执行该函数时创建的，当函数执行结束之后，函数作用域就随之被销毁掉了。**

全局作用域中包含了很多全局变量，比如全局的this值，如果是浏览器，全局作用域中还有window、document、opener等非常多的方法和对象，如果是node环境，那么会有Global、File等内容。

V8启动之后就进入正常的消息循环状态，这时候就可以执行代码了，比如执行到上面那段脚本时，V8会先解析顶层(Top Level)代码，我们可以看到，在顶层代码中定义了变量x，这时候V8就会将变量x添加到全局作用域中。

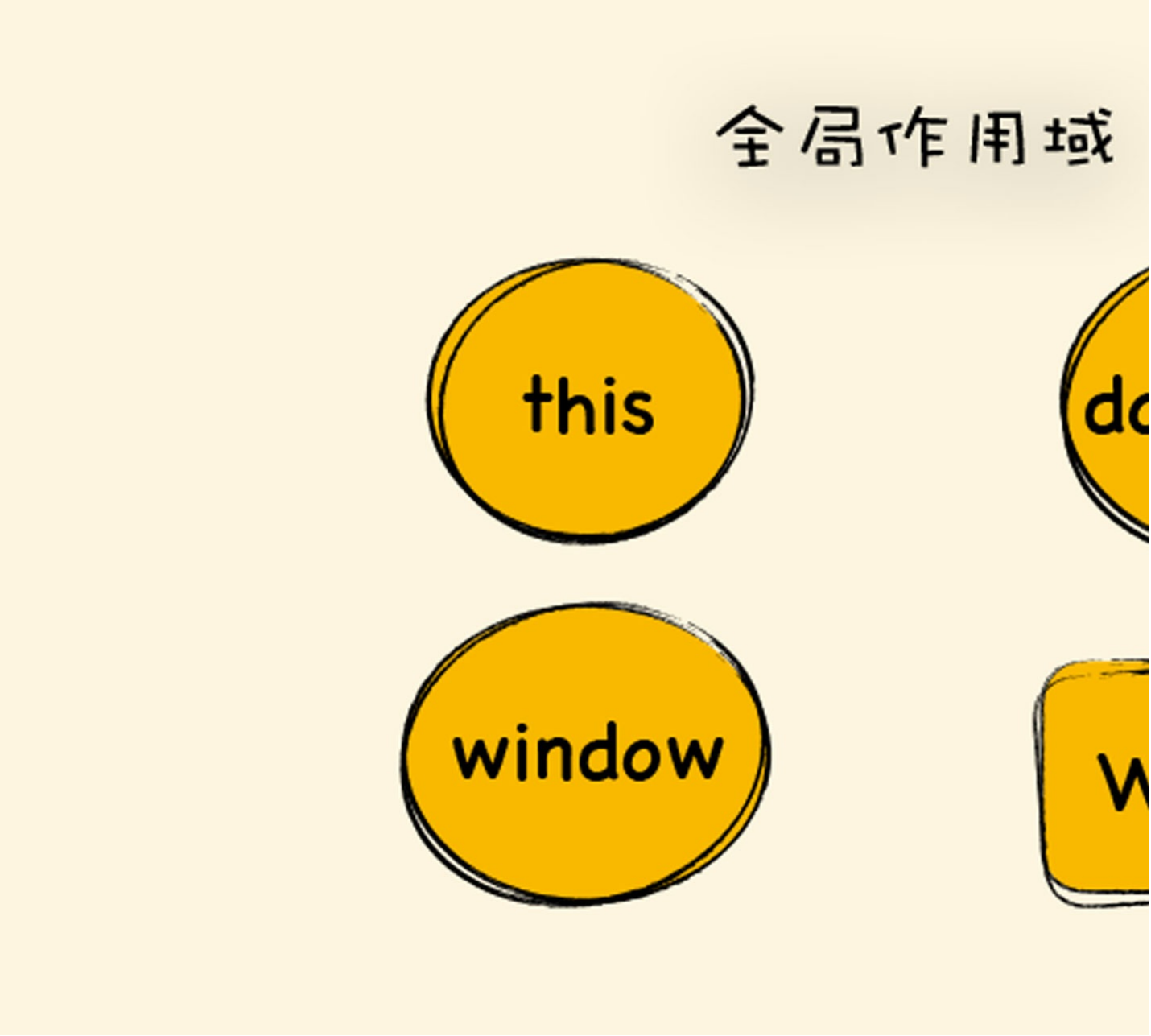
作用域链是怎么工作的？

理解了作用域和作用域链，我们再回过头来看文章开头的那道思考题：“foo函数中打印出来的变量type是bar函数中的呢，还是全局环境中的呢?”我把这段代码复制到下面：

```
var name = '极客时间'
```

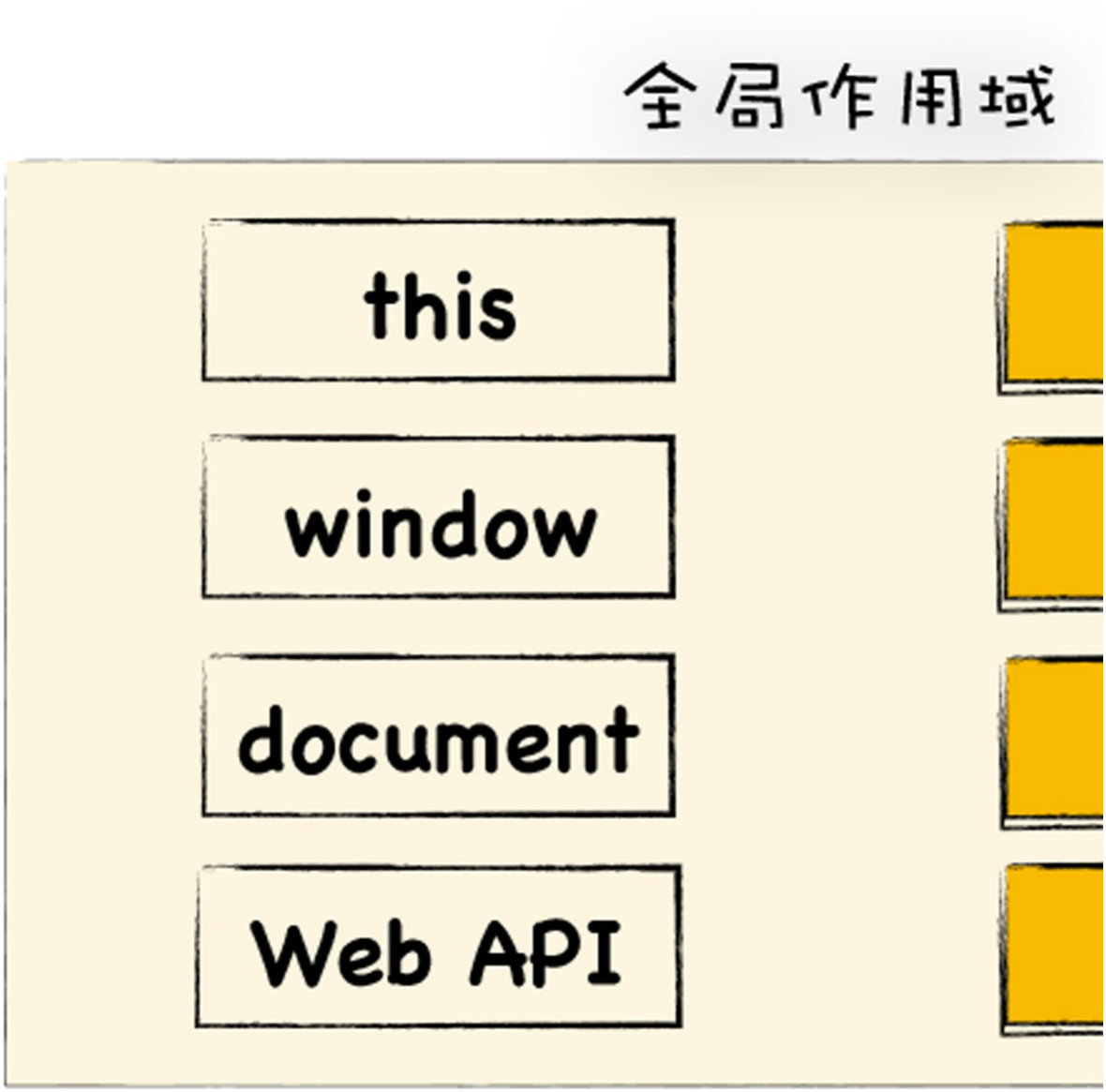
```
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

现在，我们结合V8执行这段代码的流程来具体分析下。首先当V8启动时，会创建全局作用域，全局作用域中包括了this、window等变量，还有一些全局的Web API接口，创建的作用域如下图所示：



V8启动之后，消息循环系统便开始工作了，这时候，我输入了这段代码，让其执行。

V8会先编译顶层代码，在编译过程中会将顶层定义的变量和声明的函数都添加到全局作用域中，最终的全局作用域如下图所示：

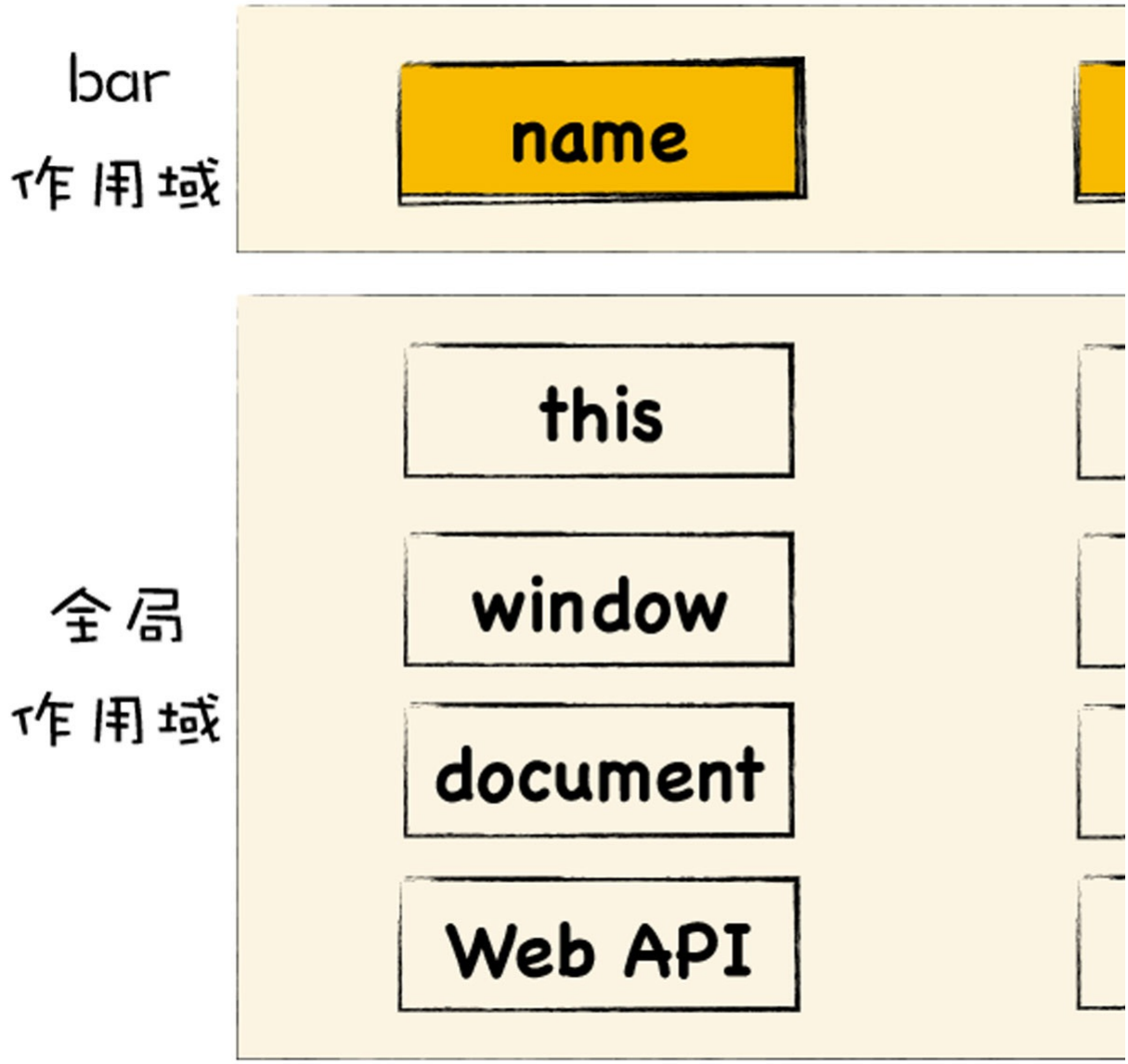


全局作用域创建完成之后，V8便进入了执行状态。前面我们介绍了变量提升，因为变量提升的原因，你可以把上面这段代码分解为如下两个部分：

```
//======解析阶段--实现变量提升=======
var name = undefined
var type = undefined
function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}
function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}



//====执行阶段========
name = '极客时间'
type = 'global'
bar()
```

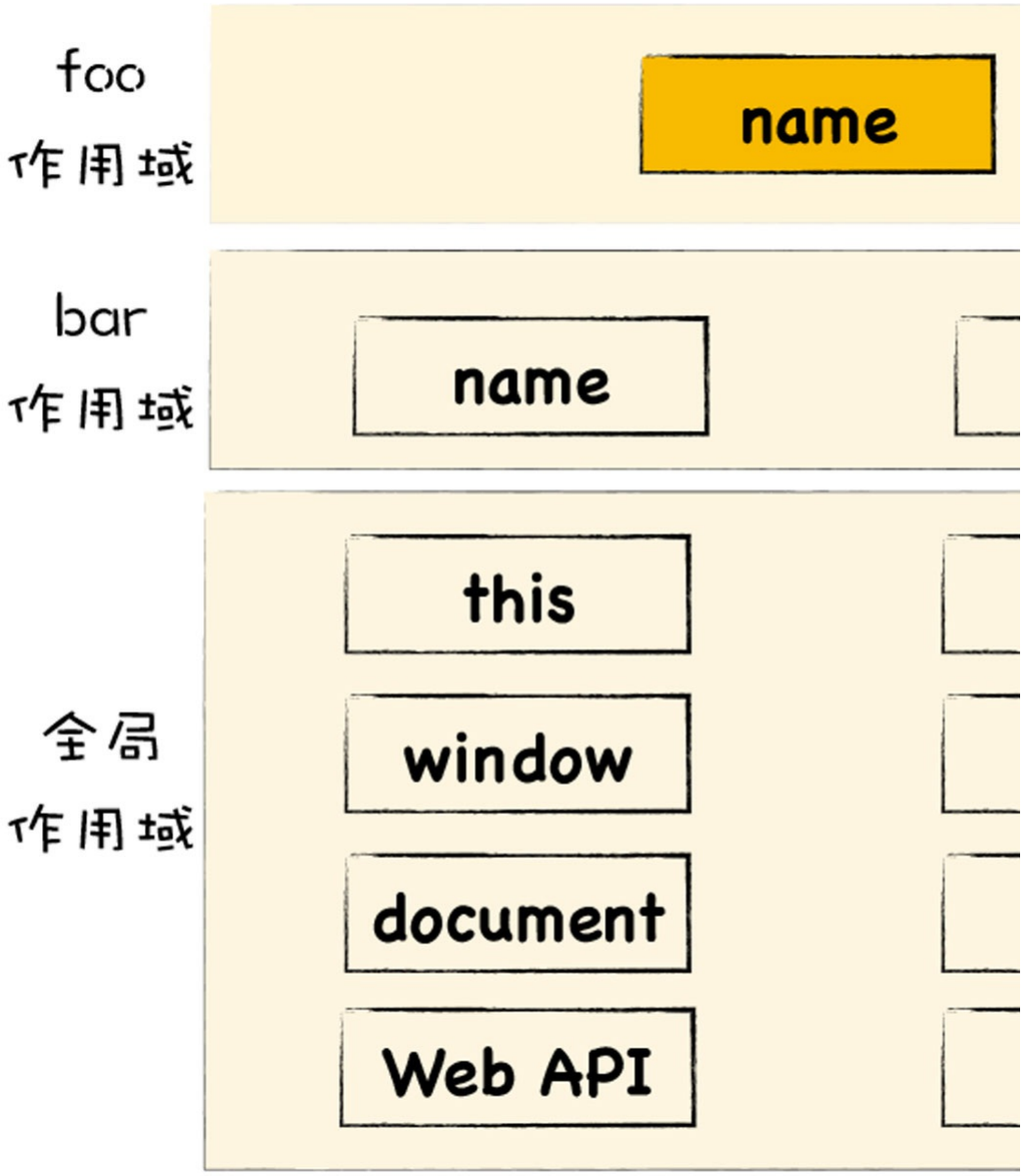
第一部分是在编译过程中完成的，此时全局作用中两个变量的值依然是undefined，然后进入执行阶段；第二部代码就是执行时的顺序，首先全局作用域中的两个变量赋值“极客时间”和“global”，然后就开始执行函数bar的调用了。

当V8执行bar函数的时候，同样需要经历两个阶段：编译和执行。在编译阶段，V8会为bar函数创建函数作用域，最终效果如下所示：



然后进入了bar函数的执行阶段。在bar函数中，只是简单地调用foo函数，因此V8又开始执行foo函数了。

同样，在编译foo函数的过程中，会创建foo函数的作用域，最终创建效果如下图所示：



好了，这时候我们就有了三个作用域了，分别是全局作用域、bar的函数作用域、foo的函数作用域。

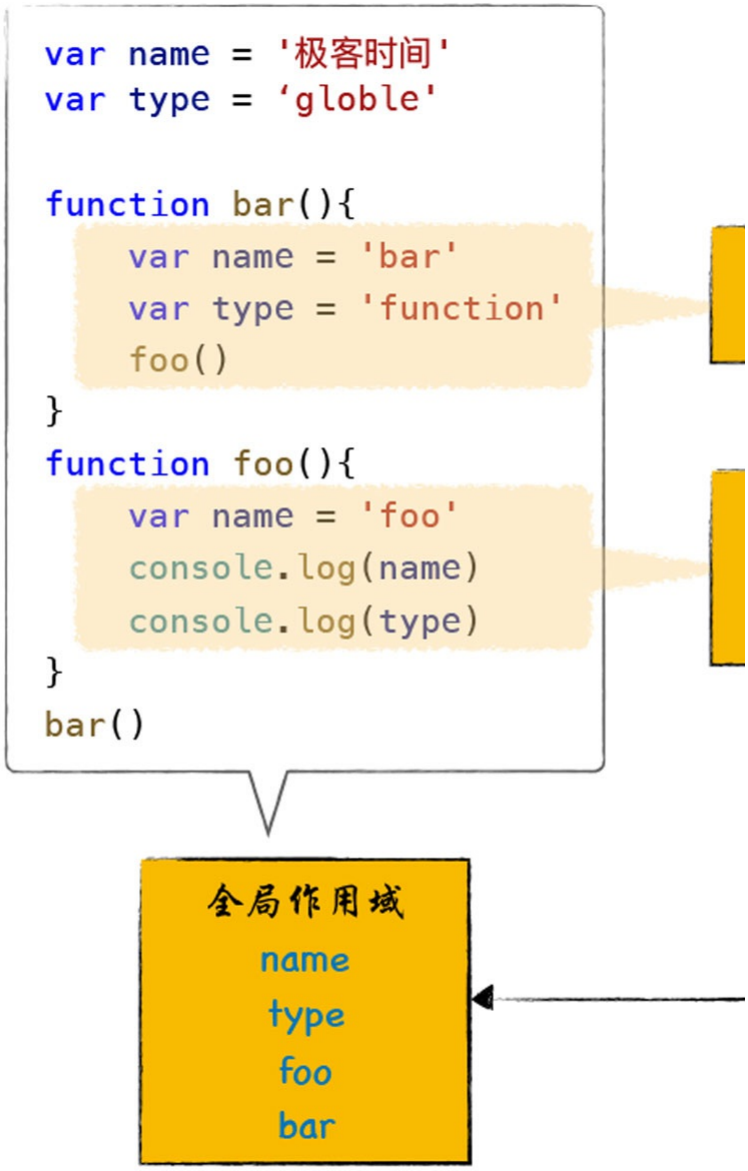
现在我们就可以将刚才提到的问题转换为作用域链的问题了：foo函数查找变量的路径到底是什么？

- 沿着foo函数作用域–>bar函数作用域–>全局作用域;
- 还是，沿着foo函数作用域—>全局作用域？

因为JavaScript是基于词法作用域的，词法作用域就是指，查找作用域的顺序是按照函数定义时的位置来决定的。bar和foo函数的外部代码都是全局代码，所以无论你是在bar函数中查找变量，还是在foo函数中查找变量，其查找顺序都是按照当前函数作用域–>全局作用域这个路径来的。

由于我们代码中的foo函数和bar函数都是在全局下面定义的，所以在foo函数中使用了type，最终打印出来的值就是全局作用域中的type。

你可以参考下面这张图：



另外，我再展开说一些。因为词法作用域是根据函数在代码中的位置来确定的，作用域是在声明函数时就确定好的了，所以我们也将词法作用域称为静态作用域。

和静态作用域相对的是动态作用域，动态作用域并不关心函数和作用域是如何声明以及在何处声明的，只关心它们从**何处调用**。换句话说，作用域链是基于调用栈的，而不是基于函数定义的位置的。（动态作用域不是本文讨论的重点，如果你感兴趣，可以参考《浏览器工作原理与实践》专栏中的《10 | 作用域链和闭包：代码中出现相同的变量，JavaScript引擎是如何选择的？》这一节。）

总结

今天，我们主要解释了一个问题，那就是在一个函数中，如果使用了一个变量，或者调用了另外一个函数，V8将会怎么去查找该变量或者函数。

为了解释清楚这个问题，我们引入了作用域的概念。作用域就是用来存放变量和函数的地方，全局作用域中存放了全局环境中声明的变量和函数，函数作用域中存放了函数中声明的变量和函数。当在某个函数中使用某个变量时，V8就会去这些作用域中查找相关变量。沿着这些作用域查找的路径，我们就称为作用域链。

要了解查找路径，我们需要明白词法作用域，词法作用域是按照代码定义时的位置决定的，而JavaScript所采用的作用域机制就是词法作用域，所以作用域链的路径就是按照词法作用域来实现的。

思考题

我将文章开头那段代码稍微调整了下，foo函数并不是在全局环境中声明的，而是在bar函数中声明的，改造后的代码如下所示：

```
var name = '极客时间'
var type = 'global'
function bar() {
    var type = 'function'
    function foo() {
        console.log(type)
    }
    foo()
}
bar()
```

那么执行这段代码之后，打印出来的内容是什么？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

在前面我们介绍了JavaScript的继承是基于原型链的，原型链将一个个原型对象串起来，从而实现对象属性的查找，今天我们要聊一个和原型链类似的话题，那就是作用域链。

作用域链就是将一个个作用域串起来，实现变量查找的路径。讨论作用域链，实际就是在讨论按照什么路径查找变量的问题。

我们知道，作用域就是存放变量和函数的地方，全局环境有全局作用域，全局作用域中存放了全局变量和全局函数。每个函数也有自己的作用域，函数作用域中存放了函数中定义的变量。

当在函数内部使用一个变量的时候，V8便会去作用域中去查找。我们通过一段在函数内部查找变量的代码来具体看一下：

```
var name = '极客时间'
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

在这段代码中，我们在全局环境中声明了变量name和type，同时还定义了bar函数和foo函数，在bar函数中又再次定义了变量name和type，在foo函数中再次定义了变量name。

函数的调用关系是：在全局环境中调用bar函数，在bar函数中调用foo函数，在foo函数中打印出来变量name和type的值。

当执行到foo函数时，首先需要打印出变量name的值，而我们在三个地方都定义了变量name，那么究竟应该使用哪个变量呢？

在foo函数中使用了变量name，那么V8就应该先使用foo函数内部定义的变量name，最终的结果确实如此，也符合我们的直觉。

接下来，foo函数继续打印变量type，但是在foo函数内部并没有定义变量type，而是在全局环境中和调用foo函数的bar函数中分别定义了变量type，那么这时候的问题来了，你觉得foo函数中打印出来的变量type是bar函数中的，还是全局环境中的呢？

什么是函数作用域和全局作用域？

要解释清楚这个问题，我们需要从作用域的工作原理讲起。

每个函数在执行时都需要查找自己的作用域，我们称为函数作用域，在执行阶段，在执行一个函数时，当该函数需要使用某个变量或者调用了某个函数时，便会优先在该函数作用域中查找相关内容。

我们再来看一段代码：

```
var x = 4
var test
function test_scope() {
    var name = 'foo'
    console.log(name)
    console.log(type)
    console.log(test)
    var type = 'function'
    test = 1
    console.log(x)
}
test_scope()
```

在上面的代码中，我们定义了一个test_scope函数，那么在V8执行test_scope函数的时候，在编译阶段会为test_scope函数创建一个作用域，在test_scope函数中定义的变量和声明的函数都会丢到该作用域中，因为我们在test_scope函数中定了三个变量，那么常见的作用域就包含有这三个变量。

你可以通过Chrome的控制台来直观感受下test_scope函数的作用域，先打开包含这段代码的页面，然后打开开发者工具，接着在test_scope函数中的第二段代码加上断点，然后刷新该页面。当执行到该断点时，V8会暂停整个执行流程，这时候我们就可以通过右边的区域面板来查看当前函数的执行状态。

你可以参考图中右侧的Scope项，然后点击展开该项，这个Local就是当前函数test_scope的作用域。在test_scope函数中定义的变量都包含到了Local中，如变量name、type，另外系统还为我们添加了另外一个隐藏变量this，V8还会默认将隐藏变量this存放到作用域中。

另外你还需要注意下，第一个test1，我并没有采用var等关键字来声明，所以test1并不会出现在test_scope函数的作用域中，而是属于this所指向的对象。（this的工作机制不是本文讨论的重点，不展开介绍。如果你感兴趣，可以在《浏览器工作原理与实践》专栏中《11 | this：从JavaScript执行上下文的视角讲清楚this》这一讲查看。）

那么另一个问题来了，我在test_scope函数使用了变量x，但是在test_scope函数的作用域中，并没有定义变量x，那么V8应该如何获取变量x?

如果在当前函数作用域中没有查找到变量，那么V8会去全局作用域中去查找，这个查找的线路就称为作用域链。

全局作用域和函数作用域类似，也是存放变量和函数的地方，但是它们还是有点不一样：**全局作用域是在V8启动过程中就创建了，且一直保存在内存中不会被销毁的，直至V8退出。 而函数作用域是在执行该函数时创建的，当函数执行结束之后，函数作用域就随之被销毁掉了。**

全局作用域中包含了很多全局变量，比如全局的this值，如果是浏览器，全局作用域中还有window、document、opener等非常多的方法和对象，如果是node环境，那么会有Global、File等内容。

V8启动之后就进入正常的消息循环状态，这时候就可以执行代码了，比如执行到上面那段脚本时，V8会先解析顶层(Top Level)代码，我们可以看到，在顶层代码中定义了变量x，这时候V8就会将变量x添加到全局作用域中。

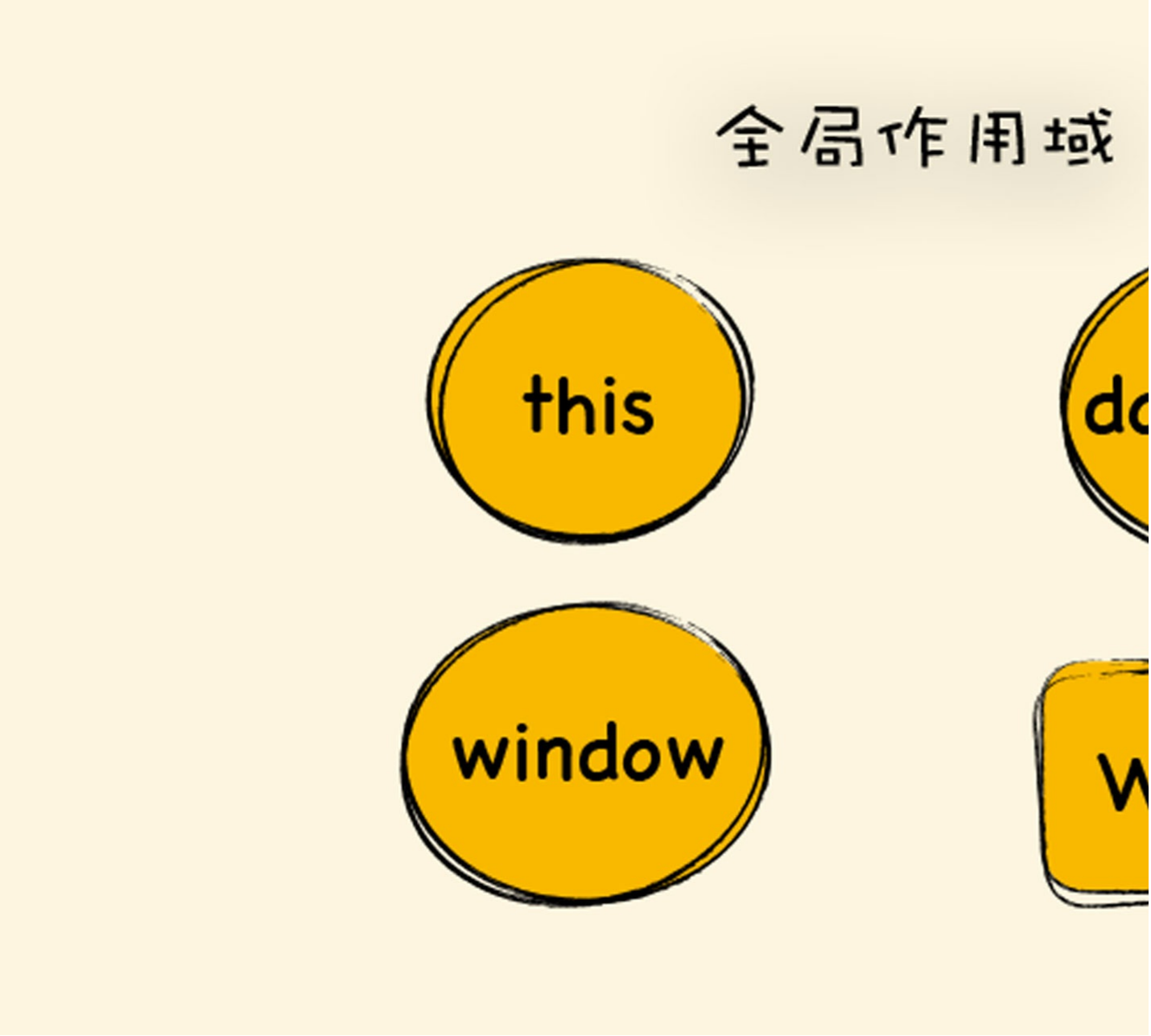
作用域链是怎么工作的？

理解了作用域和作用域链，我们再回过头来看文章开头的那道思考题：“foo函数中打印出来的变量type是bar函数中的呢，还是全局环境中的呢?”我把这段代码复制到下面：

```
var name = '极客时间'
```

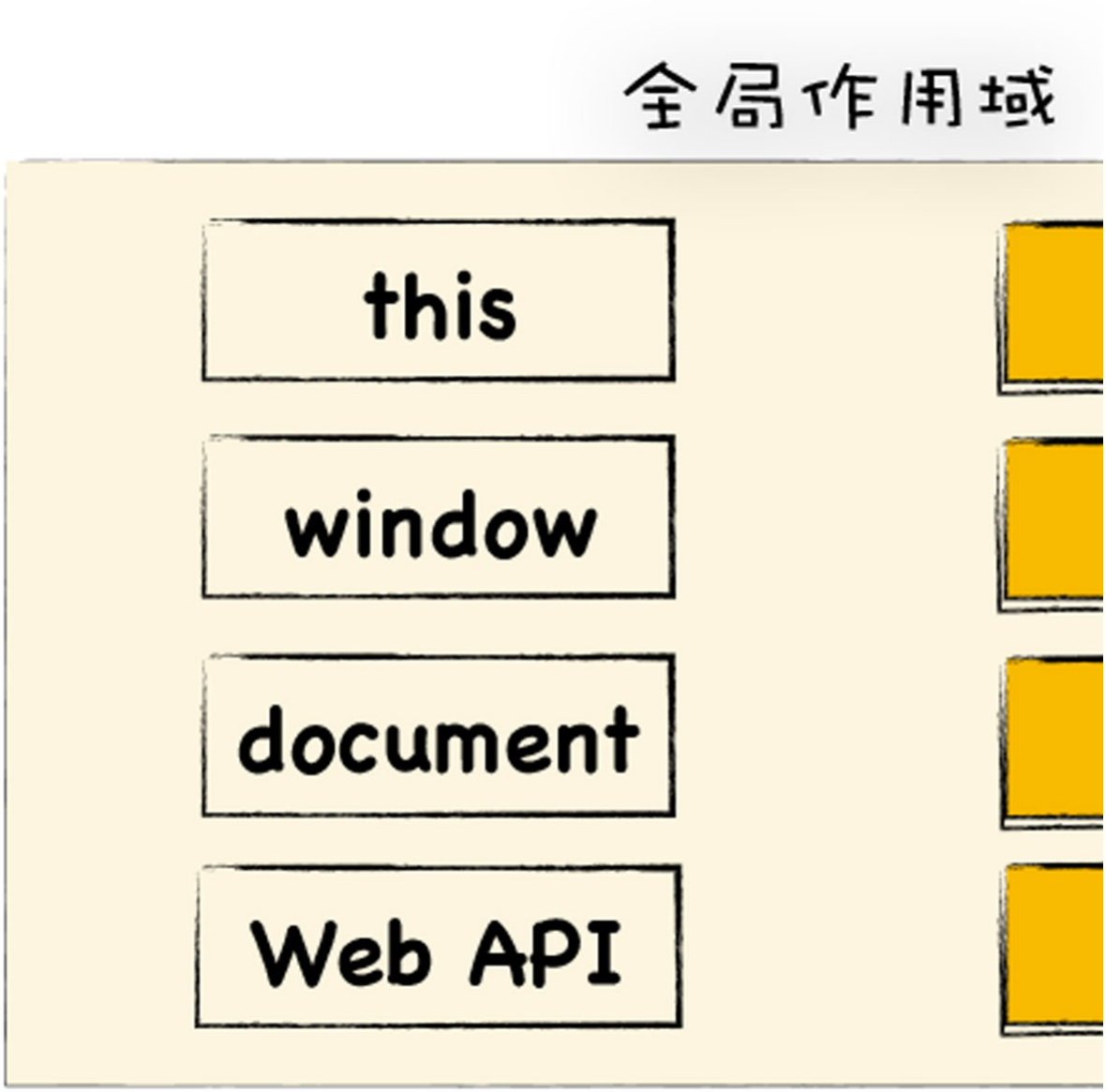
```
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

现在，我们结合V8执行这段代码的流程来具体分析下。首先当V8启动时，会创建全局作用域，全局作用域中包括了this、window等变量，还有一些全局的Web API接口，创建的作用域如下图所示：



V8启动之后，消息循环系统便开始工作了，这时候，我输入了这段代码，让其执行。

V8会先编译顶层代码，在编译过程中会将顶层定义的变量和声明的函数都添加到全局作用域中，最终的全局作用域如下图所示：

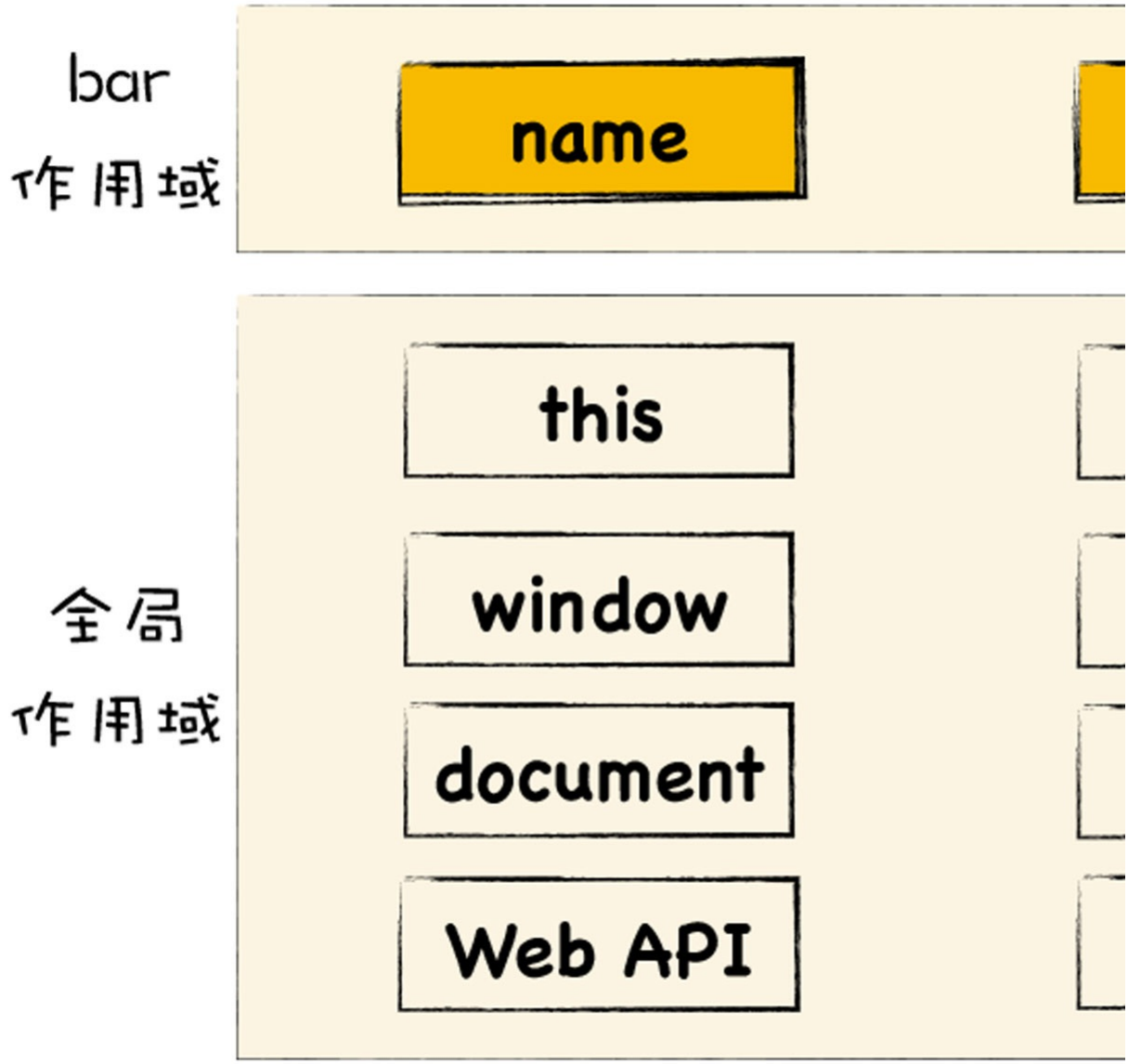


全局作用域创建完成之后，V8便进入了执行状态。前面我们介绍了变量提升，因为变量提升的原因，你可以把上面这段代码分解为如下两个部分：

```
//======解析阶段--实现变量提升=======
var name = undefined
var type = undefined
function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}
function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}



//====执行阶段========
name = '极客时间'
type = 'global'
bar()
```

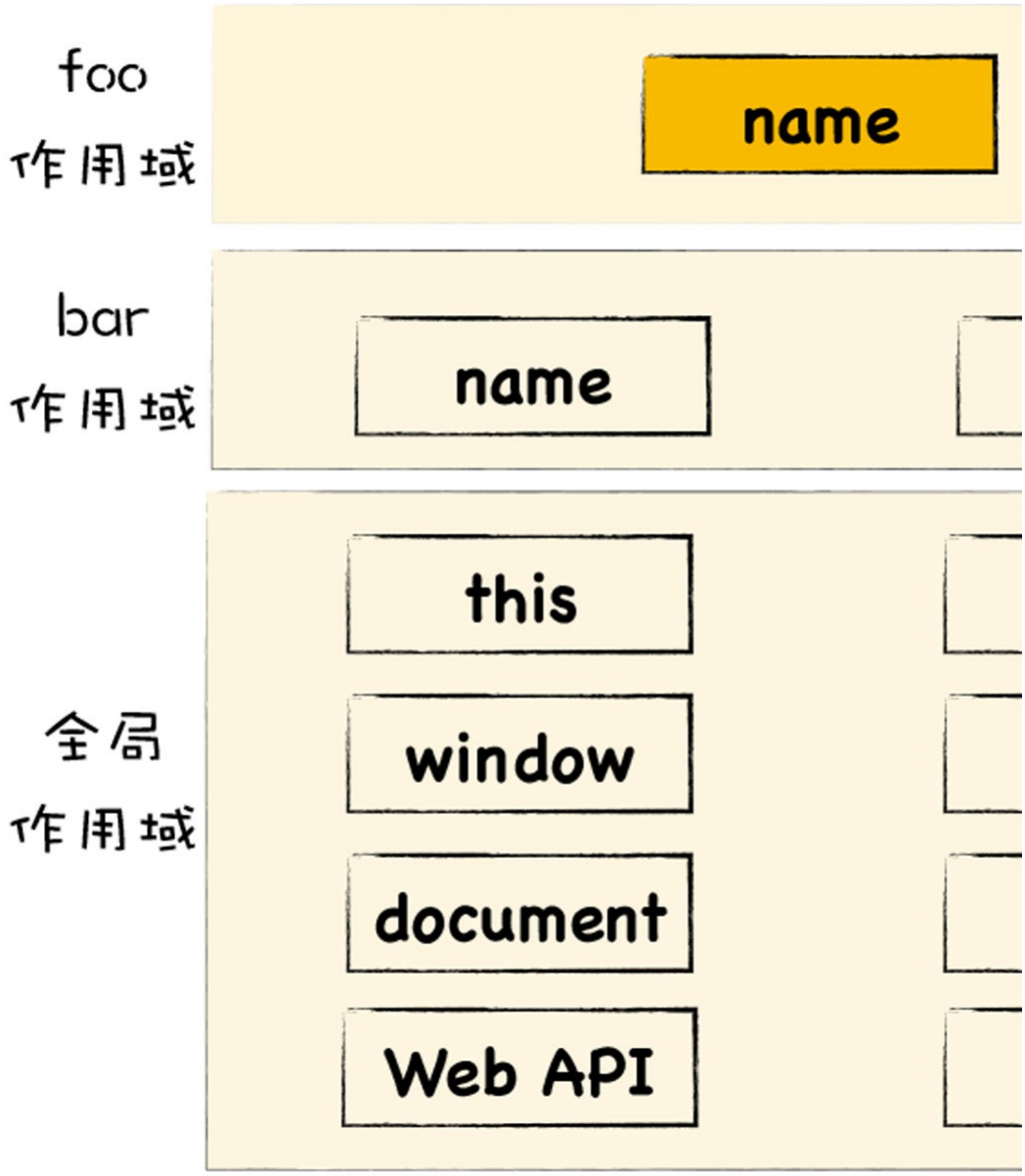
第一部分是在编译过程中完成的，此时全局作用中两个变量的值依然是undefined，然后进入执行阶段；第二部代码就是执行时的顺序，首先全局作用域中的两个变量赋值“极客时间”和“global”，然后就开始执行函数bar的调用了。

当V8执行bar函数的时候，同样需要经历两个阶段：编译和执行。在编译阶段，V8会为bar函数创建函数作用域，最终效果如下所示：



然后进入了bar函数的执行阶段。在bar函数中，只是简单地调用foo函数，因此V8又开始执行foo函数了。

同样，在编译foo函数的过程中，会创建foo函数的作用域，最终创建效果如下图所示：



好了，这时候我们就有了三个作用域了，分别是全局作用域、bar的函数作用域、foo的函数作用域。

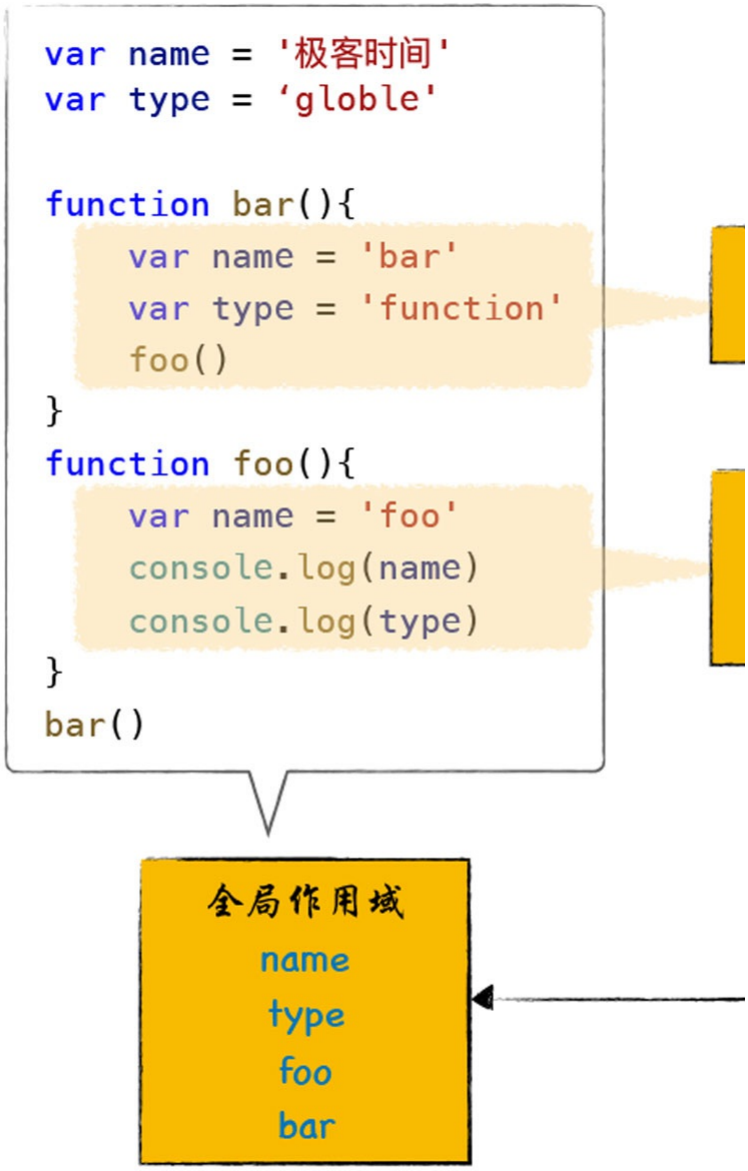
现在我们就可以将刚才提到的问题转换为作用域链的问题了：foo函数查找变量的路径到底是什么？

- 沿着foo函数作用域–>bar函数作用域–>全局作用域;
- 还是，沿着foo函数作用域—>全局作用域？

因为JavaScript是基于词法作用域的，词法作用域就是指，查找作用域的顺序是按照函数定义时的位置来决定的。bar和foo函数的外部代码都是全局代码，所以无论你是在bar函数中查找变量，还是在foo函数中查找变量，其查找顺序都是按照当前函数作用域–>全局作用域这个路径来的。

由于我们代码中的foo函数和bar函数都是在全局下面定义的，所以在foo函数中使用了type，最终打印出来的值就是全局作用域中的type。

你可以参考下面这张图：



另外，我再展开说一些。因为词法作用域是根据函数在代码中的位置来确定的，作用域是在声明函数时就确定好的了，所以我们也将词法作用域称为静态作用域。

和静态作用域相对的是动态作用域，动态作用域并不关心函数和作用域是如何声明以及在何处声明的，只关心它们从**何处调用**。换句话说，作用域链是基于调用栈的，而不是基于函数定义的位置的。（动态作用域不是本文讨论的重点，如果你感兴趣，可以参考《浏览器工作原理与实践》专栏中的《10 | 作用域链和闭包：代码中出现相同的变量，JavaScript引擎是如何选择的？》这一节。）

总结

今天，我们主要解释了一个问题，那就是在一个函数中，如果使用了一个变量，或者调用了另外一个函数，V8将会怎么去查找该变量或者函数。

为了解释清楚这个问题，我们引入了作用域的概念。作用域就是用来存放变量和函数的地方，全局作用域中存放了全局环境中声明的变量和函数，函数作用域中存放了函数中声明的变量和函数。当在某个函数中使用某个变量时，V8就会去这些作用域中查找相关变量。沿着这些作用域查找的路径，我们就称为作用域链。

要了解查找路径，我们需要明白词法作用域，词法作用域是按照代码定义时的位置决定的，而JavaScript所采用的作用域机制就是词法作用域，所以作用域链的路径就是按照词法作用域来实现的。

思考题

我将文章开头那段代码稍微调整了下，foo函数并不是在全局环境中声明的，而是在bar函数中声明的，改造后的代码如下所示：

```
var name = '极客时间'
var type = 'global'
function bar() {
    var type = 'function'
    function foo() {
        console.log(type)
    }
    foo()
}
bar()
```

那么执行这段代码之后，打印出来的内容是什么？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

在前面我们介绍了JavaScript的继承是基于原型链的，原型链将一个个原型对象串起来，从而实现对象属性的查找，今天我们要聊一个和原型链类似的话题，那就是作用域链。

作用域链就是将一个个作用域串起来，实现变量查找的路径。讨论作用域链，实际就是在讨论按照什么路径查找变量的问题。

我们知道，作用域就是存放变量和函数的地方，全局环境有全局作用域，全局作用域中存放了全局变量和全局函数。每个函数也有自己的作用域，函数作用域中存放了函数中定义的变量。

当在函数内部使用一个变量的时候，V8便会去作用域中去查找。我们通过一段在函数内部查找变量的代码来具体看一下：

```
var name = '极客时间'
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

在这段代码中，我们在全局环境中声明了变量name和type，同时还定义了bar函数和foo函数，在bar函数中又再次定义了变量name和type，在foo函数中再次定义了变量name。

函数的调用关系是：在全局环境中调用bar函数，在bar函数中调用foo函数，在foo函数中打印出来变量name和type的值。

当执行到foo函数时，首先需要打印出变量name的值，而我们在三个地方都定义了变量name，那么究竟应该使用哪个变量呢？

在foo函数中使用了变量name，那么V8就应该先使用foo函数内部定义的变量name，最终的结果确实如此，也符合我们的直觉。

接下来，foo函数继续打印变量type，但是在foo函数内部并没有定义变量type，而是在全局环境中和调用foo函数的bar函数中分别定义了变量type，那么这时候的问题来了，你觉得foo函数中打印出来的变量type是bar函数中的，还是全局环境中的呢？

什么是函数作用域和全局作用域？

要解释清楚这个问题，我们需要从作用域的工作原理讲起。

每个函数在执行时都需要查找自己的作用域，我们称为函数作用域，在执行阶段，在执行一个函数时，当该函数需要使用某个变量或者调用了某个函数时，便会优先在该函数作用域中查找相关内容。

我们再来看一段代码：

```
var x = 4
var test
function test_scope() {
    var name = 'foo'
    console.log(name)
    console.log(type)
    console.log(test)
    var type = 'function'
    test = 1
    console.log(x)
}
test_scope()
```

在上面的代码中，我们定义了一个test_scope函数，那么在V8执行test_scope函数的时候，在编译阶段会为test_scope函数创建一个作用域，在test_scope函数中定义的变量和声明的函数都会丢到该作用域中，因为我们在test_scope函数中定了三个变量，那么常见的作用域就包含有这三个变量。

你可以通过Chrome的控制台来直观感受下test_scope函数的作用域，先打开包含这段代码的页面，然后打开开发者工具，接着在test_scope函数中的第二段代码加上断点，然后刷新该页面。当执行到该断点时，V8会暂停整个执行流程，这时候我们就可以通过右边的区域面板来查看当前函数的执行状态。

你可以参考图中右侧的Scope项，然后点击展开该项，这个Local就是当前函数test_scope的作用域。在test_scope函数中定义的变量都包含到了Local中，如变量name、type，另外系统还为我们添加了另外一个隐藏变量this，V8还会默认将隐藏变量this存放到作用域中。

另外你还需要注意下，第一个test1，我并没有采用var等关键字来声明，所以test1并不会出现在test_scope函数的作用域中，而是属于this所指向的对象。（this的工作机制不是本文讨论的重点，不展开介绍。如果你感兴趣，可以在《浏览器工作原理与实践》专栏中《11 | this：从JavaScript执行上下文的视角讲清楚this》这一讲查看。）

那么另一个问题来了，我在test_scope函数使用了变量x，但是在test_scope函数的作用域中，并没有定义变量x，那么V8应该如何获取变量x?

如果在当前函数作用域中没有查找到变量，那么V8会去全局作用域中去查找，这个查找的线路就称为作用域链。

全局作用域和函数作用域类似，也是存放变量和函数的地方，但是它们还是有点不一样：**全局作用域是在V8启动过程中就创建了，且一直保存在内存中不会被销毁的，直至V8退出。 而函数作用域是在执行该函数时创建的，当函数执行结束之后，函数作用域就随之被销毁掉了。**

全局作用域中包含了很多全局变量，比如全局的this值，如果是浏览器，全局作用域中还有window、document、opener等非常多的方法和对象，如果是node环境，那么会有Global、File等内容。

V8启动之后就进入正常的消息循环状态，这时候就可以执行代码了，比如执行到上面那段脚本时，V8会先解析顶层(Top Level)代码，我们可以看到，在顶层代码中定义了变量x，这时候V8就会将变量x添加到全局作用域中。

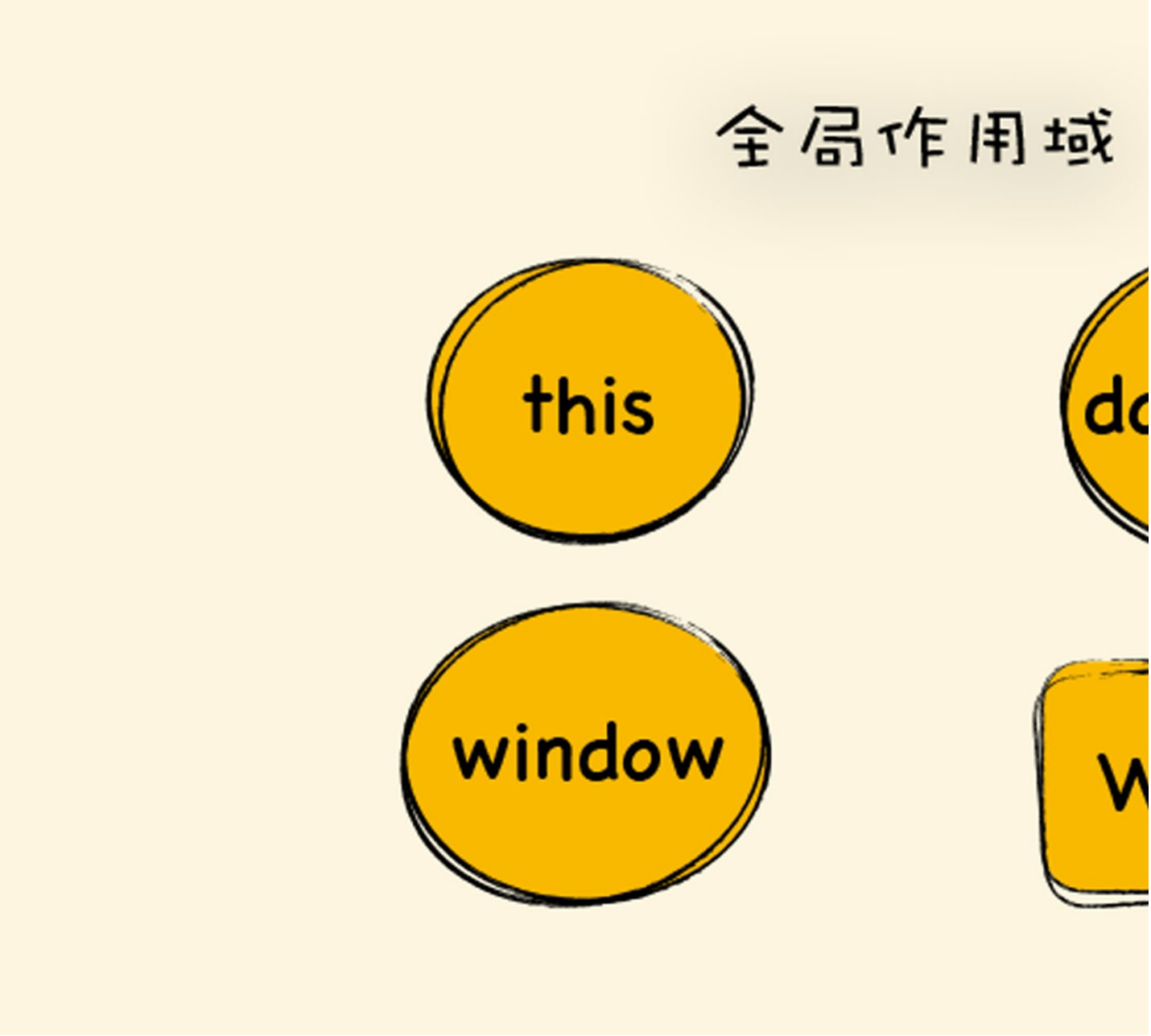
作用域链是怎么工作的？

理解了作用域和作用域链，我们再回过头来看文章开头的那道思考题：“foo函数中打印出来的变量type是bar函数中的呢，还是全局环境中的呢?”我把这段代码复制到下面：

```
var name = '极客时间'
```

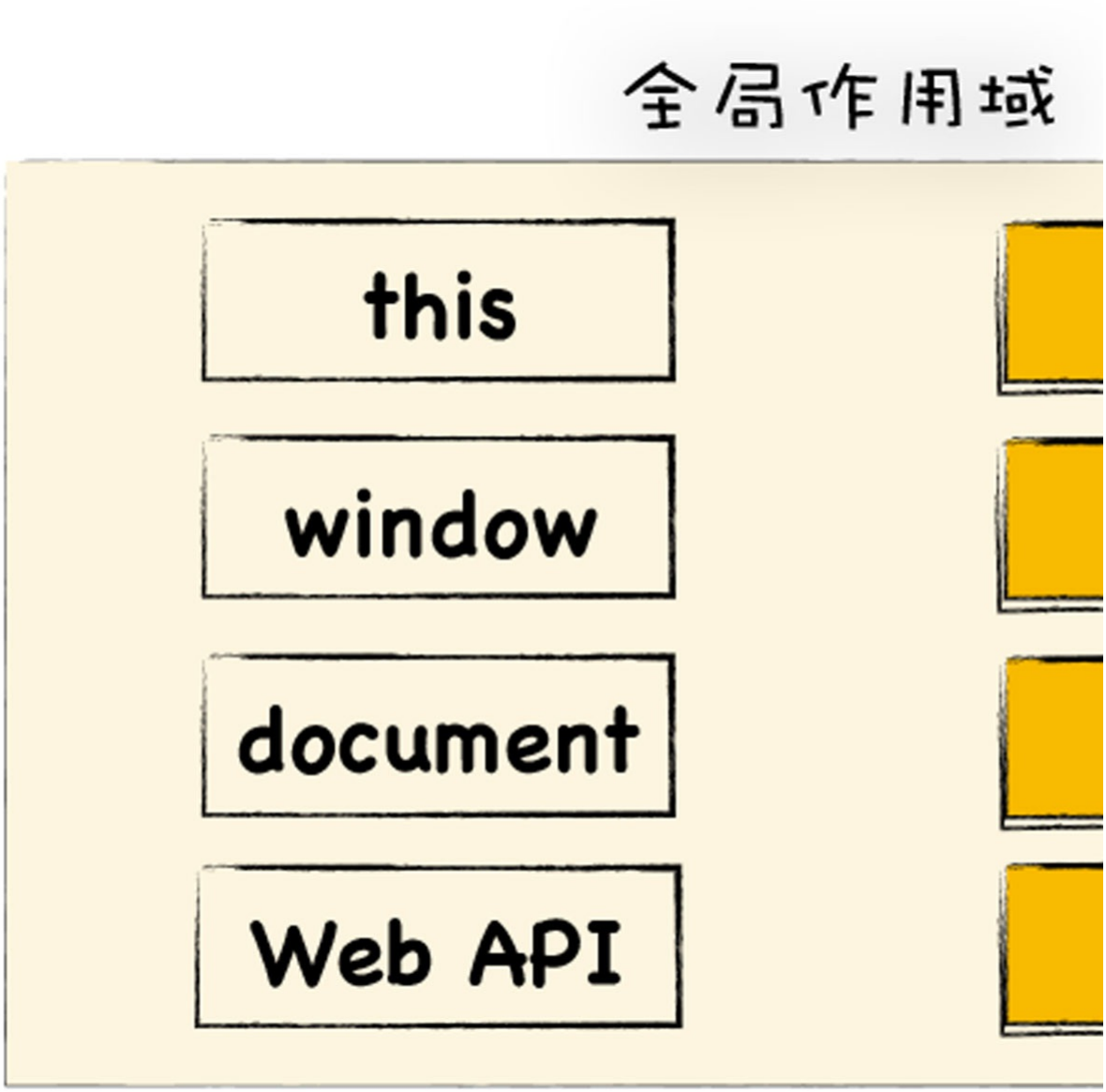
```
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

现在，我们结合V8执行这段代码的流程来具体分析下。首先当V8启动时，会创建全局作用域，全局作用域中包括了this、window等变量，还有一些全局的Web API接口，创建的作用域如下图所示：



V8启动之后，消息循环系统便开始工作了，这时候，我输入了这段代码，让其执行。

V8会先编译顶层代码，在编译过程中会将顶层定义的变量和声明的函数都添加到全局作用域中，最终的全局作用域如下图所示：

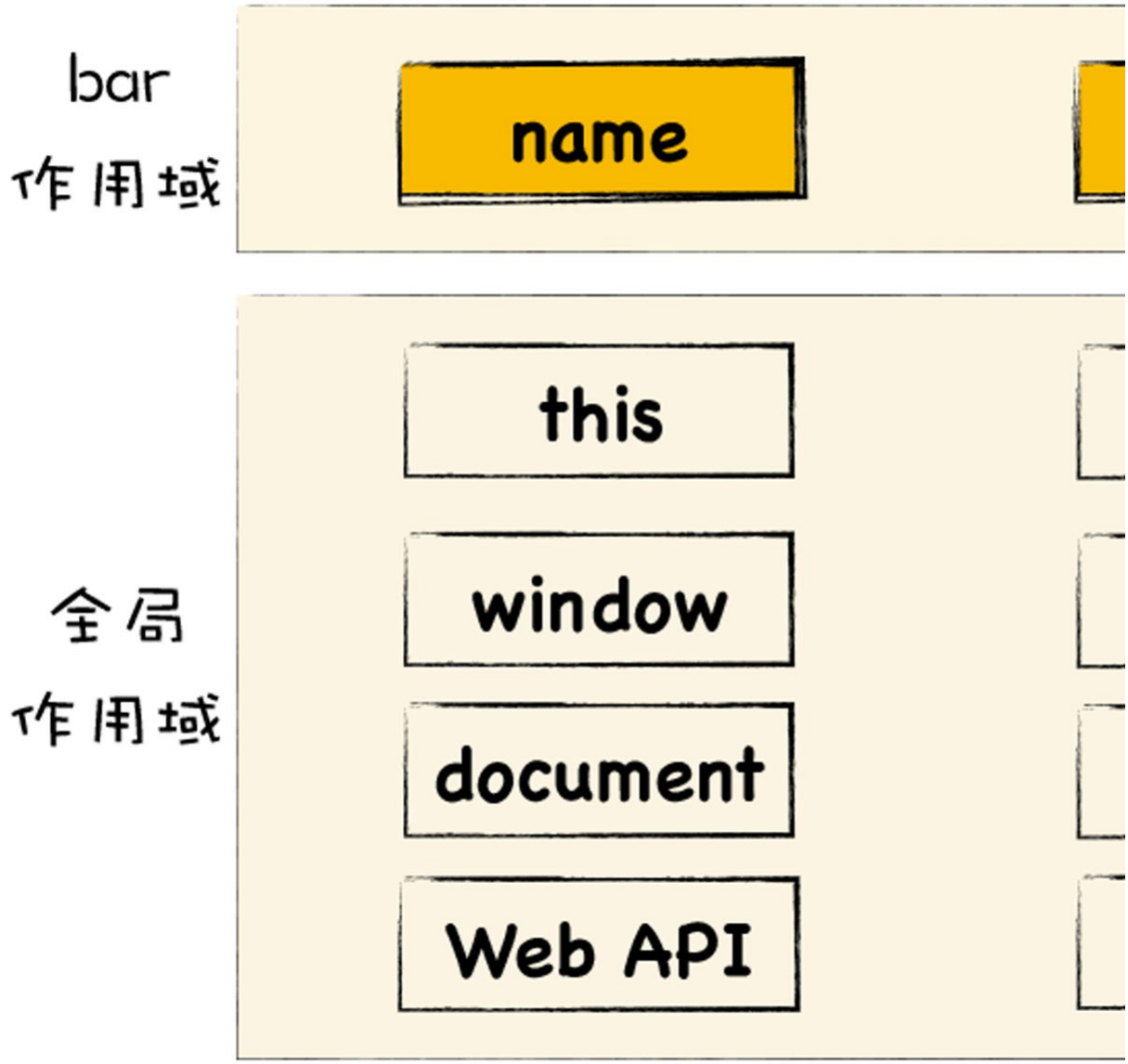


全局作用域创建完成之后，V8便进入了执行状态。前面我们介绍了变量提升，因为变量提升的原因，你可以把上面这段代码分解为如下两个部分：

```
//======解析阶段--实现变量提升=======
var name = undefined
var type = undefined
function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}
function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}



//====执行阶段========
name = '极客时间'
type = 'global'
bar()
```

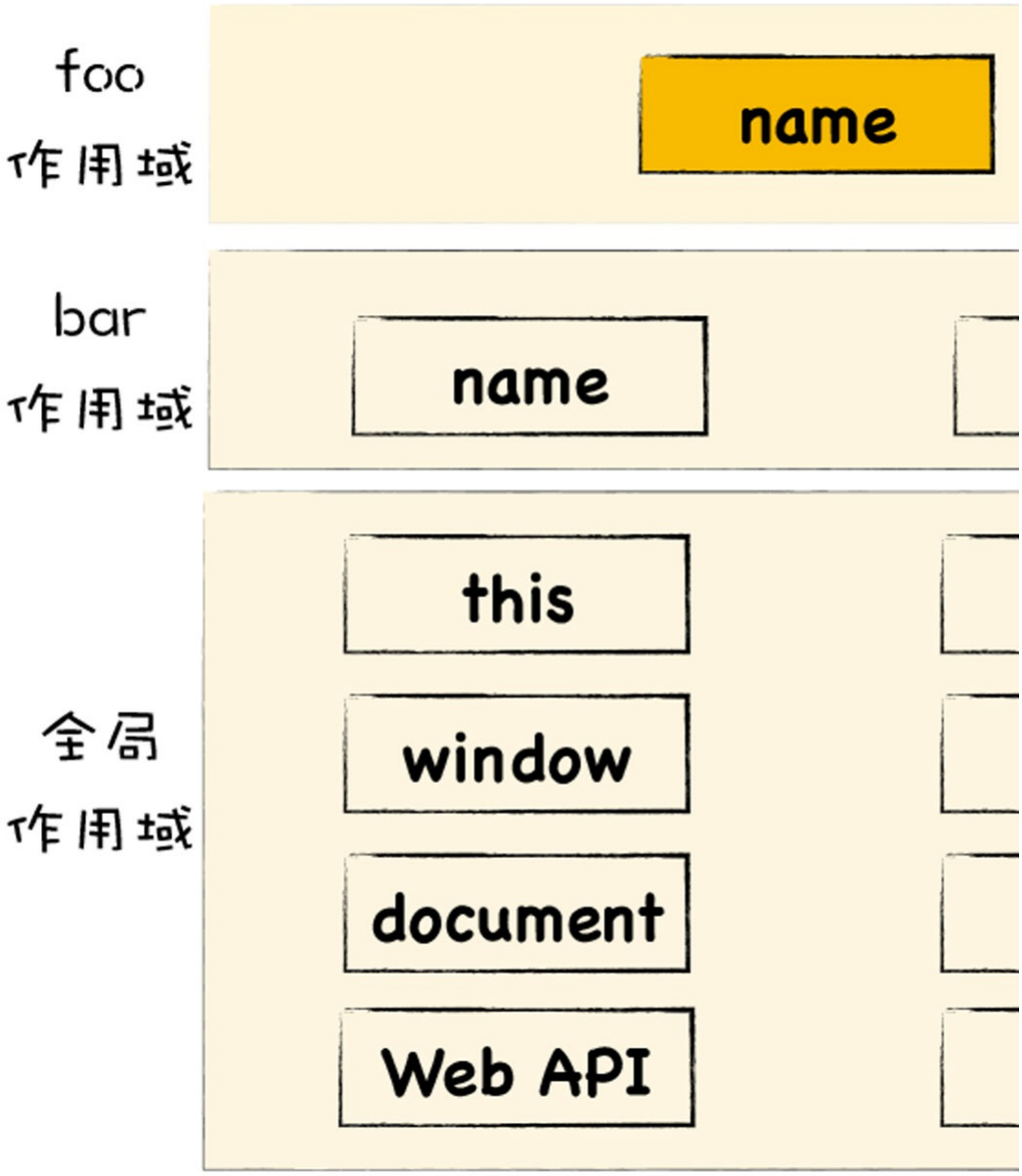
第一部分是在编译过程中完成的，此时全局作用中两个变量的值依然是undefined，然后进入执行阶段；第二部代码就是执行时的顺序，首先全局作用域中的两个变量赋值“极客时间”和“global”，然后就开始执行函数bar的调用了。

当V8执行bar函数的时候，同样需要经历两个阶段：编译和执行。在编译阶段，V8会为bar函数创建函数作用域，最终效果如下所示：



然后进入了bar函数的执行阶段。在bar函数中，只是简单地调用foo函数，因此V8又开始执行foo函数了。

同样，在编译foo函数的过程中，会创建foo函数的作用域，最终创建效果如下图所示：



好了，这时候我们就有了三个作用域了，分别是全局作用域、bar的函数作用域、foo的函数作用域。

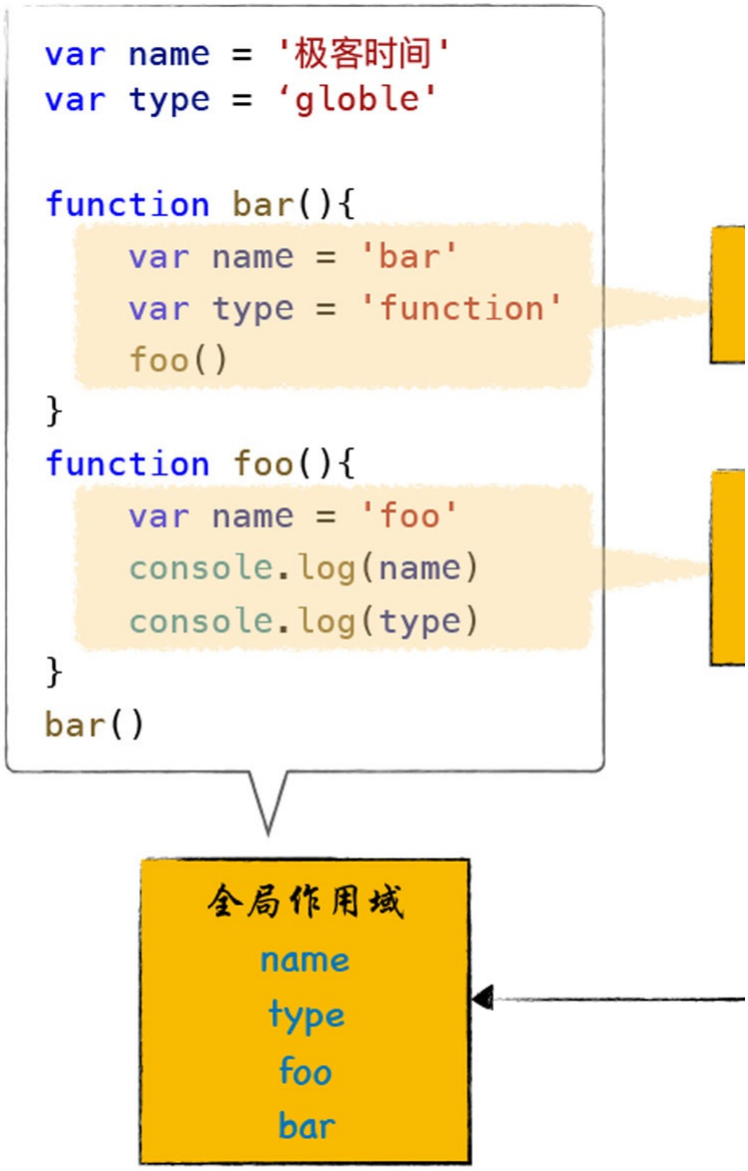
现在我们就可以将刚才提到的问题转换为作用域链的问题了：foo函数查找变量的路径到底是什么？

- 沿着foo函数作用域–>bar函数作用域–>全局作用域;
- 还是，沿着foo函数作用域—>全局作用域？

因为JavaScript是基于词法作用域的，词法作用域就是指，查找作用域的顺序是按照函数定义时的位置来决定的。bar和foo函数的外部代码都是全局代码，所以无论你是在bar函数中查找变量，还是在foo函数中查找变量，其查找顺序都是按照当前函数作用域–>全局作用域这个路径来的。

由于我们代码中的foo函数和bar函数都是在全局下面定义的，所以在foo函数中使用了type，最终打印出来的值就是全局作用域中的type。

你可以参考下面这张图：



另外，我再展开说一些。因为词法作用域是根据函数在代码中的位置来确定的，作用域是在声明函数时就确定好的了，所以我们也将词法作用域称为静态作用域。

和静态作用域相对的是动态作用域，动态作用域并不关心函数和作用域是如何声明以及在何处声明的，只关心它们从**何处调用**。换句话说，作用域链是基于调用栈的，而不是基于函数定义的位置的。（动态作用域不是本文讨论的重点，如果你感兴趣，可以参考《浏览器工作原理与实践》专栏中的《10 | 作用域链和闭包：代码中出现相同的变量，JavaScript引擎是如何选择的？》这一节。）

总结

今天，我们主要解释了一个问题，那就是在一个函数中，如果使用了一个变量，或者调用了另外一个函数，V8将会怎么去查找该变量或者函数。

为了解释清楚这个问题，我们引入了作用域的概念。作用域就是用来存放变量和函数的地方，全局作用域中存放了全局环境中声明的变量和函数，函数作用域中存放了函数中声明的变量和函数。当在某个函数中使用某个变量时，V8就会去这些作用域中查找相关变量。沿着这些作用域查找的路径，我们就称为作用域链。

要了解查找路径，我们需要明白词法作用域，词法作用域是按照代码定义时的位置决定的，而JavaScript所采用的作用域机制就是词法作用域，所以作用域链的路径就是按照词法作用域来实现的。

思考题

我将文章开头那段代码稍微调整了下，foo函数并不是在全局环境中声明的，而是在bar函数中声明的，改造后的代码如下所示：

```
var name = '极客时间'
var type = 'global'
function bar() {
    var type = 'function'
    function foo() {
        console.log(type)
    }
    foo()
}
bar()
```

那么执行这段代码之后，打印出来的内容是什么？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

在前面我们介绍了JavaScript的继承是基于原型链的，原型链将一个个原型对象串起来，从而实现对象属性的查找，今天我们要聊一个和原型链类似的话题，那就是作用域链。

作用域链就是将一个个作用域串起来，实现变量查找的路径。讨论作用域链，实际就是在讨论按照什么路径查找变量的问题。

我们知道，作用域就是存放变量和函数的地方，全局环境有全局作用域，全局作用域中存放了全局变量和全局函数。每个函数也有自己的作用域，函数作用域中存放了函数中定义的变量。

当在函数内部使用一个变量的时候，V8便会去作用域中去查找。我们通过一段在函数内部查找变量的代码来具体看一下：

```
var name = '极客时间'
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

在这段代码中，我们在全局环境中声明了变量name和type，同时还定义了bar函数和foo函数，在bar函数中又再次定义了变量name和type，在foo函数中再次定义了变量name。

函数的调用关系是：在全局环境中调用bar函数，在bar函数中调用foo函数，在foo函数中打印出来变量name和type的值。

当执行到foo函数时，首先需要打印出变量name的值，而我们在三个地方都定义了变量name，那么究竟应该使用哪个变量呢？

在foo函数中使用了变量name，那么V8就应该先使用foo函数内部定义的变量name，最终的结果确实如此，也符合我们的直觉。

接下来，foo函数继续打印变量type，但是在foo函数内部并没有定义变量type，而是在全局环境中和调用foo函数的bar函数中分别定义了变量type，那么这时候的问题来了，你觉得foo函数中打印出来的变量type是bar函数中的，还是全局环境中的呢？

什么是函数作用域和全局作用域？

要解释清楚这个问题，我们需要从作用域的工作原理讲起。

每个函数在执行时都需要查找自己的作用域，我们称为函数作用域，在执行阶段，在执行一个函数时，当该函数需要使用某个变量或者调用了某个函数时，便会优先在该函数作用域中查找相关内容。

我们再来看一段代码：

```
var x = 4
var test
function test_scope() {
    var name = 'foo'
    console.log(name)
    console.log(type)
    console.log(test)
    var type = 'function'
    test = 1
    console.log(x)
}
test_scope()
```

在上面的代码中，我们定义了一个test_scope函数，那么在V8执行test_scope函数的时候，在编译阶段会为test_scope函数创建一个作用域，在test_scope函数中定义的变量和声明的函数都会丢到该作用域中，因为我们在test_scope函数中定了三个变量，那么常见的作用域就包含有这三个变量。

你可以通过Chrome的控制台来直观感受下test_scope函数的作用域，先打开包含这段代码的页面，然后打开开发者工具，接着在test_scope函数中的第二段代码加上断点，然后刷新该页面。当执行到该断点时，V8会暂停整个执行流程，这时候我们就可以通过右边的区域面板来查看当前函数的执行状态。

你可以参考图中右侧的Scope项，然后点击展开该项，这个Local就是当前函数test_scope的作用域。在test_scope函数中定义的变量都包含到了Local中，如变量name、type，另外系统还为我们添加了另外一个隐藏变量this，V8还会默认将隐藏变量this存放到作用域中。

另外你还需要注意下，第一个test1，我并没有采用var等关键字来声明，所以test1并不会出现在test_scope函数的作用域中，而是属于this所指向的对象。（this的工作机制不是本文讨论的重点，不展开介绍。如果你感兴趣，可以在《浏览器工作原理与实践》专栏中《11 | this：从JavaScript执行上下文的视角讲清楚this》这一讲查看。）

那么另一个问题来了，我在test_scope函数使用了变量x，但是在test_scope函数的作用域中，并没有定义变量x，那么V8应该如何获取变量x?

如果在当前函数作用域中没有查找到变量，那么V8会去全局作用域中去查找，这个查找的线路就称为作用域链。

全局作用域和函数作用域类似，也是存放变量和函数的地方，但是它们还是有点不一样：**全局作用域是在V8启动过程中就创建了，且一直保存在内存中不会被销毁的，直至V8退出。 而函数作用域是在执行该函数时创建的，当函数执行结束之后，函数作用域就随之被销毁掉了。**

全局作用域中包含了很多全局变量，比如全局的this值，如果是浏览器，全局作用域中还有window、document、opener等非常多的方法和对象，如果是node环境，那么会有Global、File等内容。

V8启动之后就进入正常的消息循环状态，这时候就可以执行代码了，比如执行到上面那段脚本时，V8会先解析顶层(Top Level)代码，我们可以看到，在顶层代码中定义了变量x，这时候V8就会将变量x添加到全局作用域中。

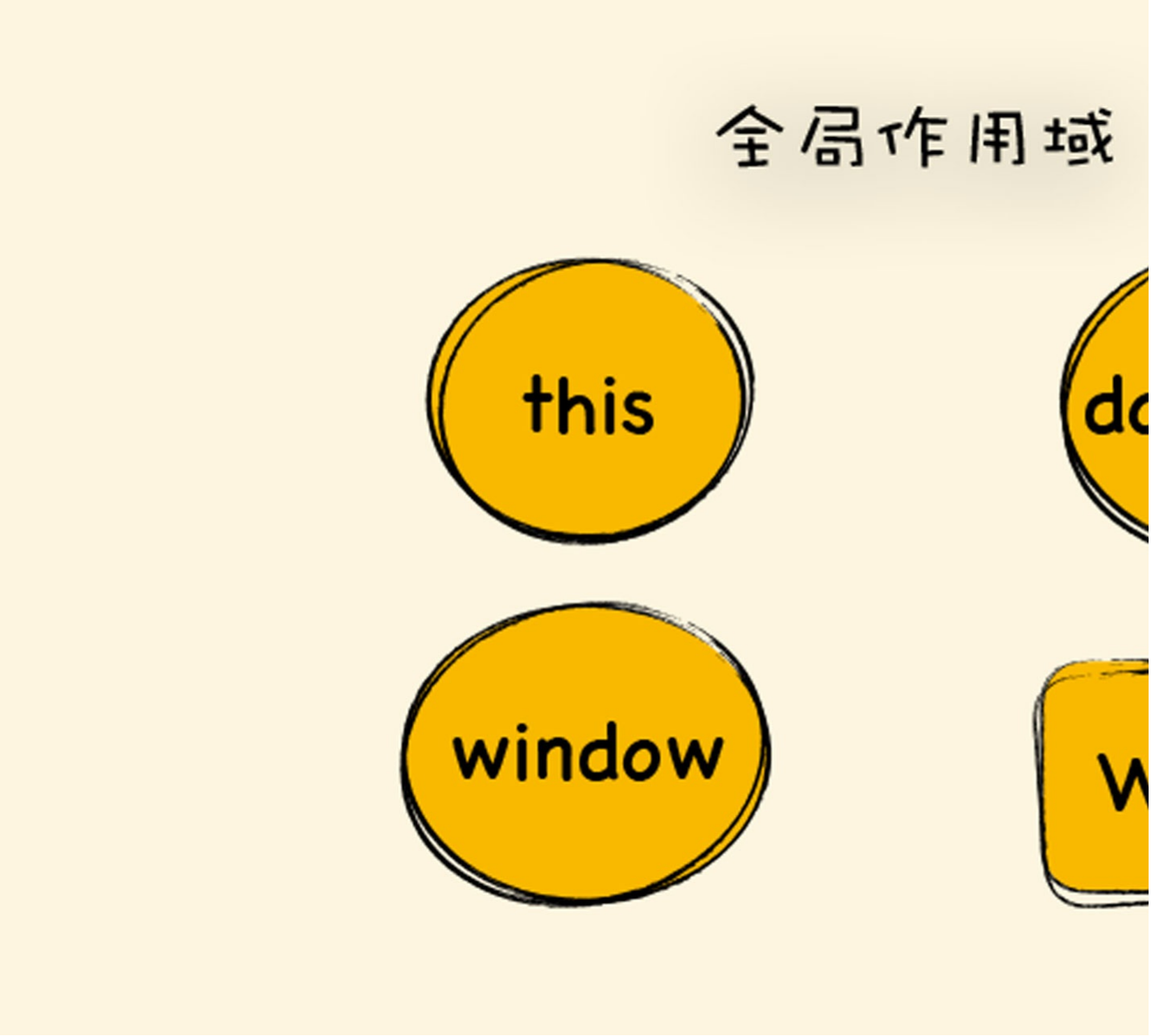
作用域链是怎么工作的？

理解了作用域和作用域链，我们再回过头来看文章开头的那道思考题：“foo函数中打印出来的变量type是bar函数中的呢，还是全局环境中的呢?”我把这段代码复制到下面：

```
var name = '极客时间'
```

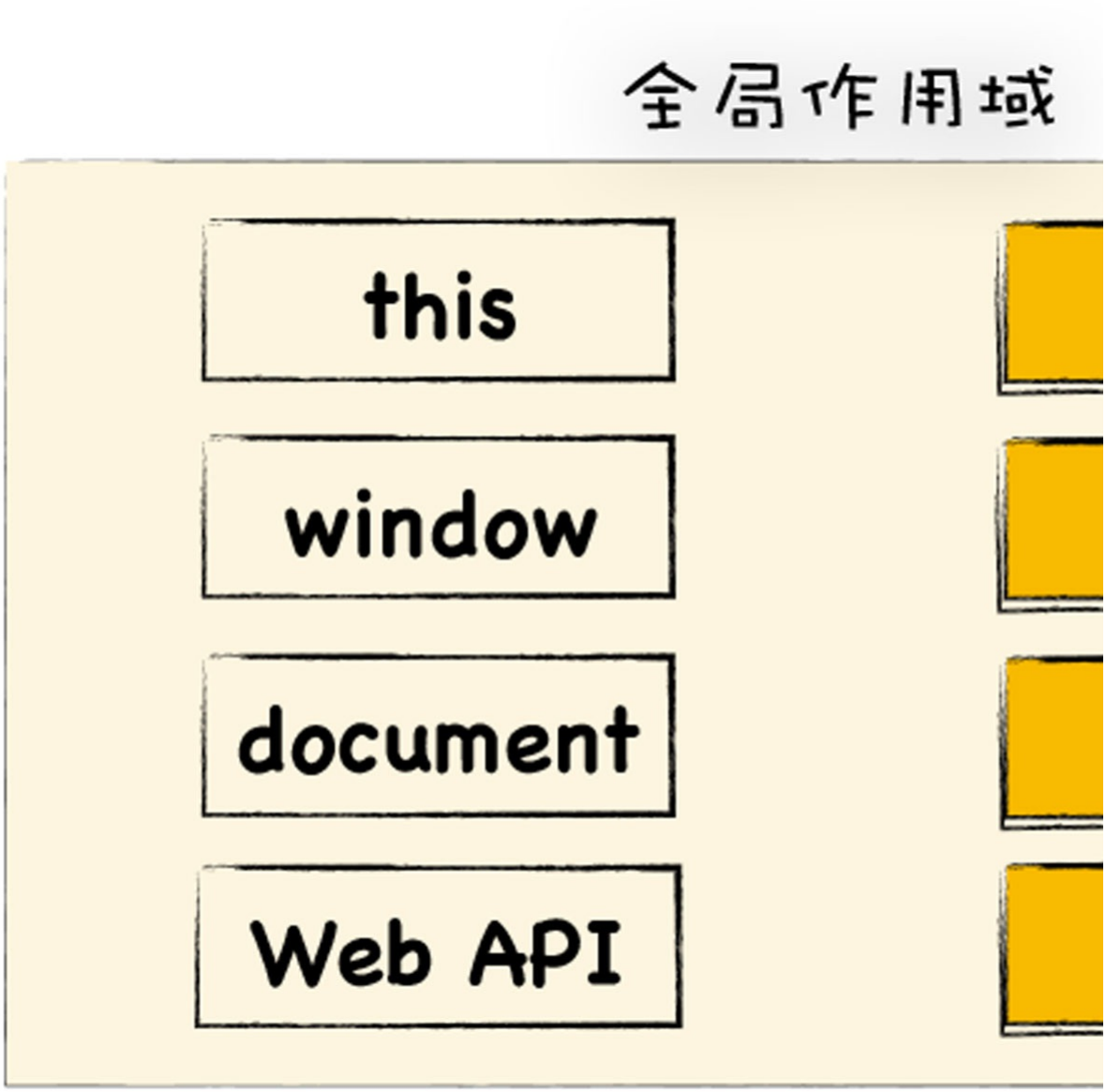
```
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

现在，我们结合V8执行这段代码的流程来具体分析下。首先当V8启动时，会创建全局作用域，全局作用域中包括了this、window等变量，还有一些全局的Web API接口，创建的作用域如下图所示：



V8启动之后，消息循环系统便开始工作了，这时候，我输入了这段代码，让其执行。

V8会先编译顶层代码，在编译过程中会将顶层定义的变量和声明的函数都添加到全局作用域中，最终的全局作用域如下图所示：

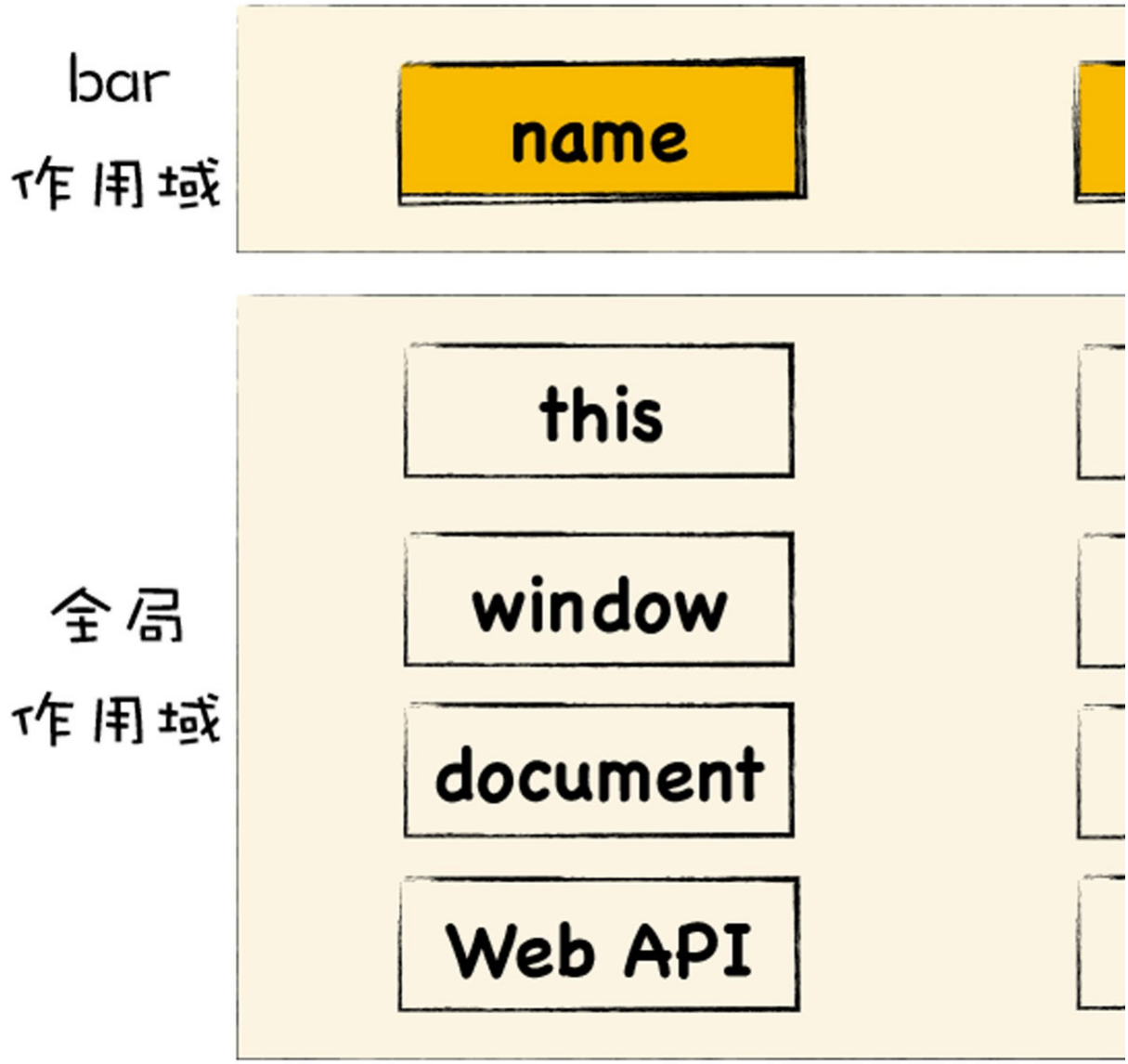


全局作用域创建完成之后，V8便进入了执行状态。前面我们介绍了变量提升，因为变量提升的原因，你可以把上面这段代码分解为如下两个部分：

```
//======解析阶段--实现变量提升=======
var name = undefined
var type = undefined
function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}
function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}



//====执行阶段========
name = '极客时间'
type = 'global'
bar()
```

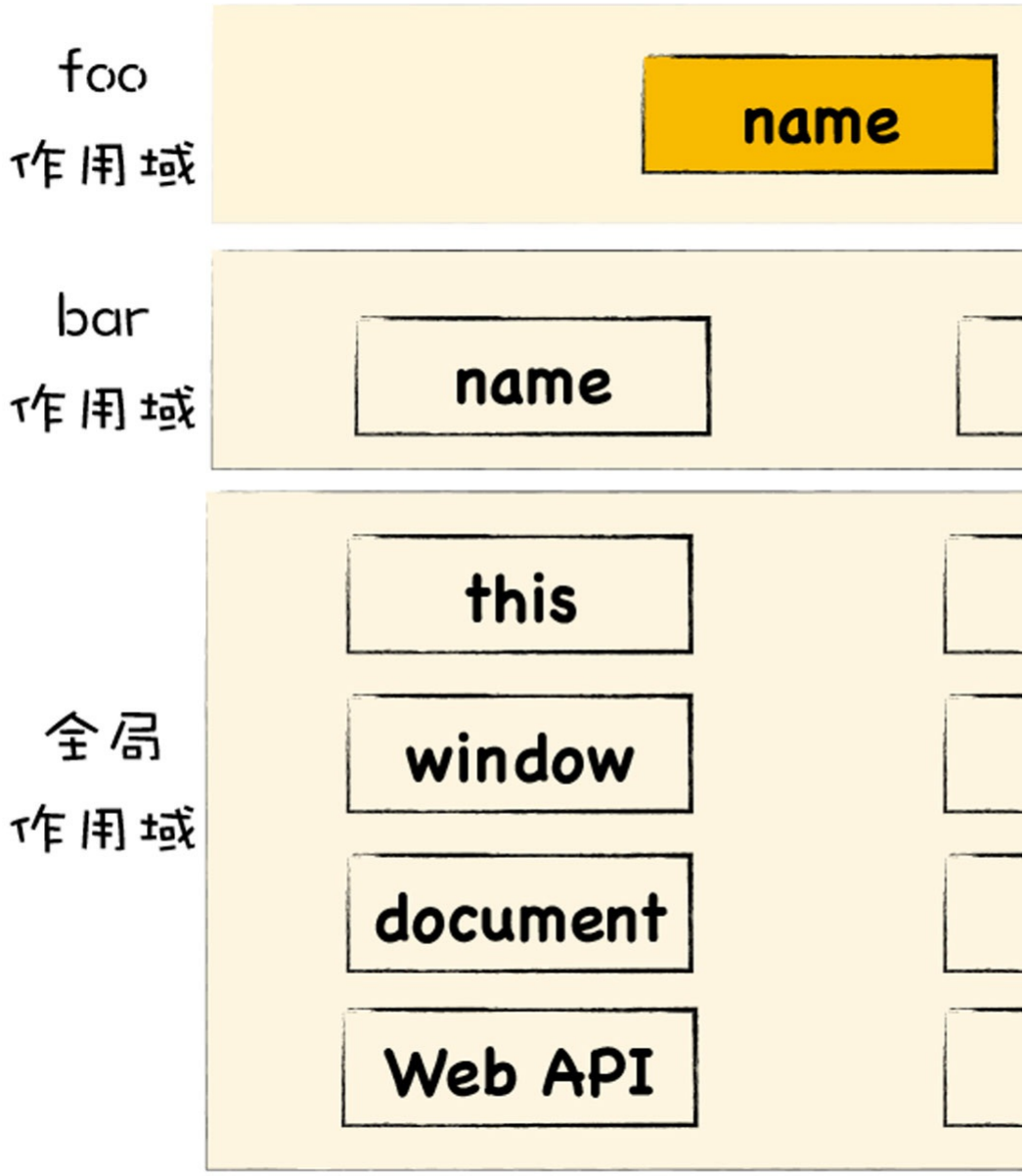
第一部分是在编译过程中完成的，此时全局作用中两个变量的值依然是undefined，然后进入执行阶段；第二部代码就是执行时的顺序，首先全局作用域中的两个变量赋值“极客时间”和“global”，然后就开始执行函数bar的调用了。

当V8执行bar函数的时候，同样需要经历两个阶段：编译和执行。在编译阶段，V8会为bar函数创建函数作用域，最终效果如下所示：



然后进入了bar函数的执行阶段。在bar函数中，只是简单地调用foo函数，因此V8又开始执行foo函数了。

同样，在编译foo函数的过程中，会创建foo函数的作用域，最终创建效果如下图所示：



好了，这时候我们就有了三个作用域了，分别是全局作用域、bar的函数作用域、foo的函数作用域。

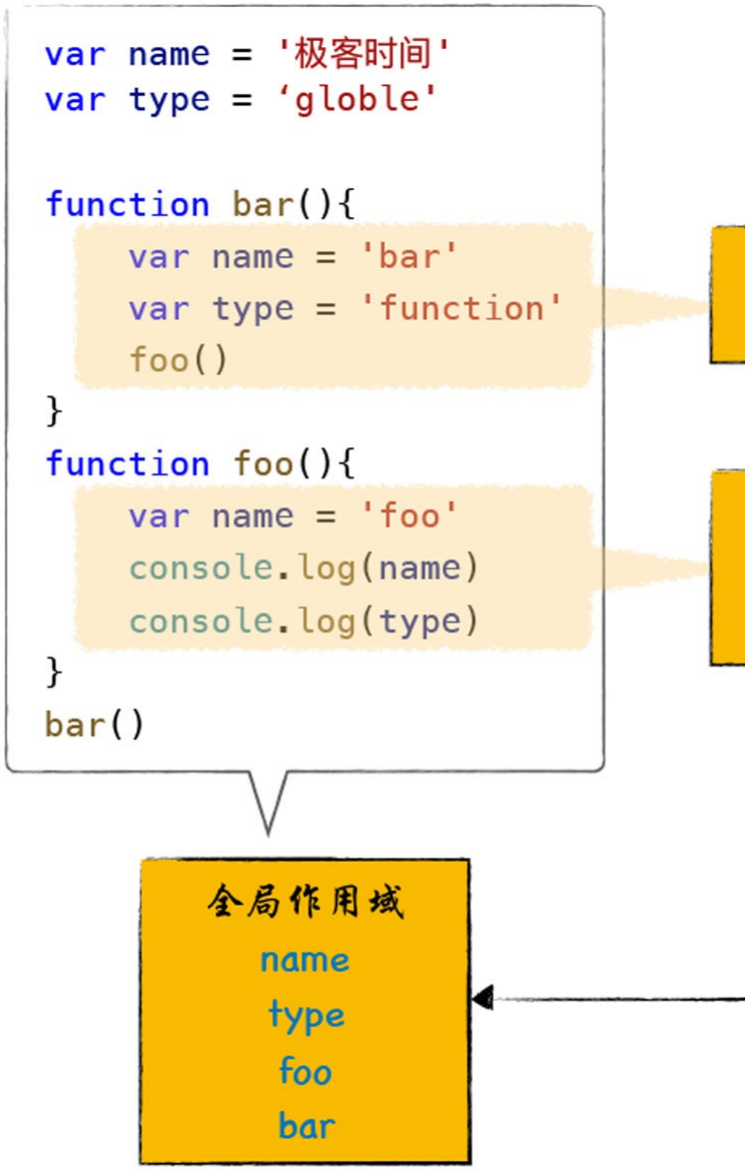
现在我们就可以将刚才提到的问题转换为作用域链的问题了：foo函数查找变量的路径到底是什么？

- 沿着foo函数作用域–>bar函数作用域–>全局作用域;
- 还是，沿着foo函数作用域—>全局作用域？

因为JavaScript是基于词法作用域的，词法作用域就是指，查找作用域的顺序是按照函数定义时的位置来决定的。bar和foo函数的外部代码都是全局代码，所以无论你是在bar函数中查找变量，还是在foo函数中查找变量，其查找顺序都是按照当前函数作用域–>全局作用域这个路径来的。

由于我们代码中的foo函数和bar函数都是在全局下面定义的，所以在foo函数中使用了type，最终打印出来的值就是全局作用域中的type。

你可以参考下面这张图：



另外，我再展开说一些。因为词法作用域是根据函数在代码中的位置来确定的，作用域是在声明函数时就确定好的了，所以我们也将词法作用域称为静态作用域。

和静态作用域相对的是动态作用域，动态作用域并不关心函数和作用域是如何声明以及在何处声明的，只关心它们从**何处调用**。换句话说，作用域链是基于调用栈的，而不是基于函数定义的位置的。（动态作用域不是本文讨论的重点，如果你感兴趣，可以参考《浏览器工作原理与实践》专栏中的《10 | 作用域链和闭包：代码中出现相同的变量，JavaScript引擎是如何选择的？》这一节。）

总结

今天，我们主要解释了一个问题，那就是在一个函数中，如果使用了一个变量，或者调用了另外一个函数，V8将会怎么去查找该变量或者函数。

为了解释清楚这个问题，我们引入了作用域的概念。作用域就是用来存放变量和函数的地方，全局作用域中存放了全局环境中声明的变量和函数，函数作用域中存放了函数中声明的变量和函数。当在某个函数中使用某个变量时，V8就会去这些作用域中查找相关变量。沿着这些作用域查找的路径，我们就称为作用域链。

要了解查找路径，我们需要明白词法作用域，词法作用域是按照代码定义时的位置决定的，而JavaScript所采用的作用域机制就是词法作用域，所以作用域链的路径就是按照词法作用域来实现的。

思考题

我将文章开头那段代码稍微调整了下，foo函数并不是在全局环境中声明的，而是在bar函数中声明的，改造后的代码如下所示：

```
var name = '极客时间'
var type = 'global'
function bar() {
    var type = 'function'
    function foo() {
        console.log(type)
    }
    foo()
}
bar()
```

那么执行这段代码之后，打印出来的内容是什么？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

在前面我们介绍了JavaScript的继承是基于原型链的，原型链将一个个原型对象串起来，从而实现对象属性的查找，今天我们要聊一个和原型链类似的话题，那就是作用域链。

作用域链就是将一个个作用域串起来，实现变量查找的路径。讨论作用域链，实际就是在讨论按照什么路径查找变量的问题。

我们知道，作用域就是存放变量和函数的地方，全局环境有全局作用域，全局作用域中存放了全局变量和全局函数。每个函数也有自己的作用域，函数作用域中存放了函数中定义的变量。

当在函数内部使用一个变量的时候，V8便会去作用域中去查找。我们通过一段在函数内部查找变量的代码来具体看一下：

```
var name = '极客时间'
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

在这段代码中，我们在全局环境中声明了变量name和type，同时还定义了bar函数和foo函数，在bar函数中又再次定义了变量name和type，在foo函数中再次定义了变量name。

函数的调用关系是：在全局环境中调用bar函数，在bar函数中调用foo函数，在foo函数中打印出来变量name和type的值。

当执行到foo函数时，首先需要打印出变量name的值，而我们在三个地方都定义了变量name，那么究竟应该使用哪个变量呢？

在foo函数中使用了变量name，那么V8就应该先使用foo函数内部定义的变量name，最终的结果确实如此，也符合我们的直觉。

接下来，foo函数继续打印变量type，但是在foo函数内部并没有定义变量type，而是在全局环境中和调用foo函数的bar函数中分别定义了变量type，那么这时候的问题来了，你觉得foo函数中打印出来的变量type是bar函数中的，还是全局环境中的呢？

什么是函数作用域和全局作用域？

要解释清楚这个问题，我们需要从作用域的工作原理讲起。

每个函数在执行时都需要查找自己的作用域，我们称为函数作用域，在执行阶段，在执行一个函数时，当该函数需要使用某个变量或者调用了某个函数时，便会优先在该函数作用域中查找相关内容。

我们再来看一段代码：

```
var x = 4
var test
function test_scope() {
    var name = 'foo'
    console.log(name)
    console.log(type)
    console.log(test)
    var type = 'function'
    test = 1
    console.log(x)
}
test_scope()
```

在上面的代码中，我们定义了一个test_scope函数，那么在V8执行test_scope函数的时候，在编译阶段会为test_scope函数创建一个作用域，在test_scope函数中定义的变量和声明的函数都会丢到该作用域中，因为我们在test_scope函数中定了三个变量，那么常见的作用域就包含有这三个变量。

你可以通过Chrome的控制台来直观感受下test_scope函数的作用域，先打开包含这段代码的页面，然后打开开发者工具，接着在test_scope函数中的第二段代码加上断点，然后刷新该页面。当执行到该断点时，V8会暂停整个执行流程，这时候我们就可以通过右边的区域面板来查看当前函数的执行状态。

你可以参考图中右侧的Scope项，然后点击展开该项，这个Local就是当前函数test_scope的作用域。在test_scope函数中定义的变量都包含到了Local中，如变量name、type，另外系统还为我们添加了另外一个隐藏变量this，V8还会默认将隐藏变量this存放到作用域中。

另外你还需要注意下，第一个test1，我并没有采用var等关键字来声明，所以test1并不会出现在test_scope函数的作用域中，而是属于this所指向的对象。（this的工作机制不是本文讨论的重点，不展开介绍。如果你感兴趣，可以在《浏览器工作原理与实践》专栏中《11 | this：从JavaScript执行上下文的视角讲清楚this》这一讲查看。）

那么另一个问题来了，我在test_scope函数使用了变量x，但是在test_scope函数的作用域中，并没有定义变量x，那么V8应该如何获取变量x?

如果在当前函数作用域中没有查找到变量，那么V8会去全局作用域中去查找，这个查找的线路就称为作用域链。

全局作用域和函数作用域类似，也是存放变量和函数的地方，但是它们还是有点不一样：**全局作用域是在V8启动过程中就创建了，且一直保存在内存中不会被销毁的，直至V8退出。 而函数作用域是在执行该函数时创建的，当函数执行结束之后，函数作用域就随之被销毁掉了。**

全局作用域中包含了很多全局变量，比如全局的this值，如果是浏览器，全局作用域中还有window、document、opener等非常多的方法和对象，如果是node环境，那么会有Global、File等内容。

V8启动之后就进入正常的消息循环状态，这时候就可以执行代码了，比如执行到上面那段脚本时，V8会先解析顶层(Top Level)代码，我们可以看到，在顶层代码中定义了变量x，这时候V8就会将变量x添加到全局作用域中。

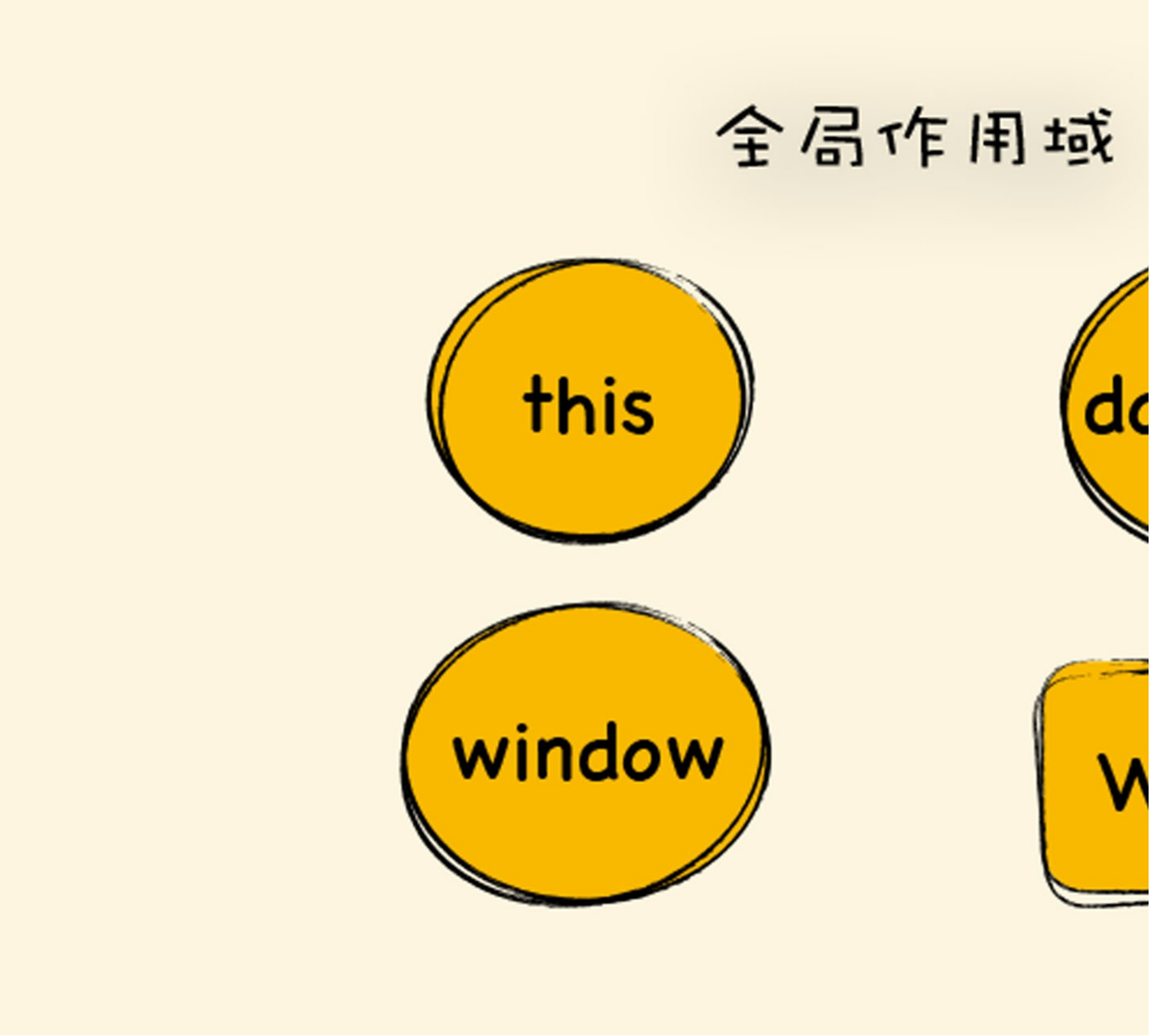
作用域链是怎么工作的？

理解了作用域和作用域链，我们再回过头来看文章开头的那道思考题：“foo函数中打印出来的变量type是bar函数中的呢，还是全局环境中的呢?”我把这段代码复制到下面：

```
var name = '极客时间'
```

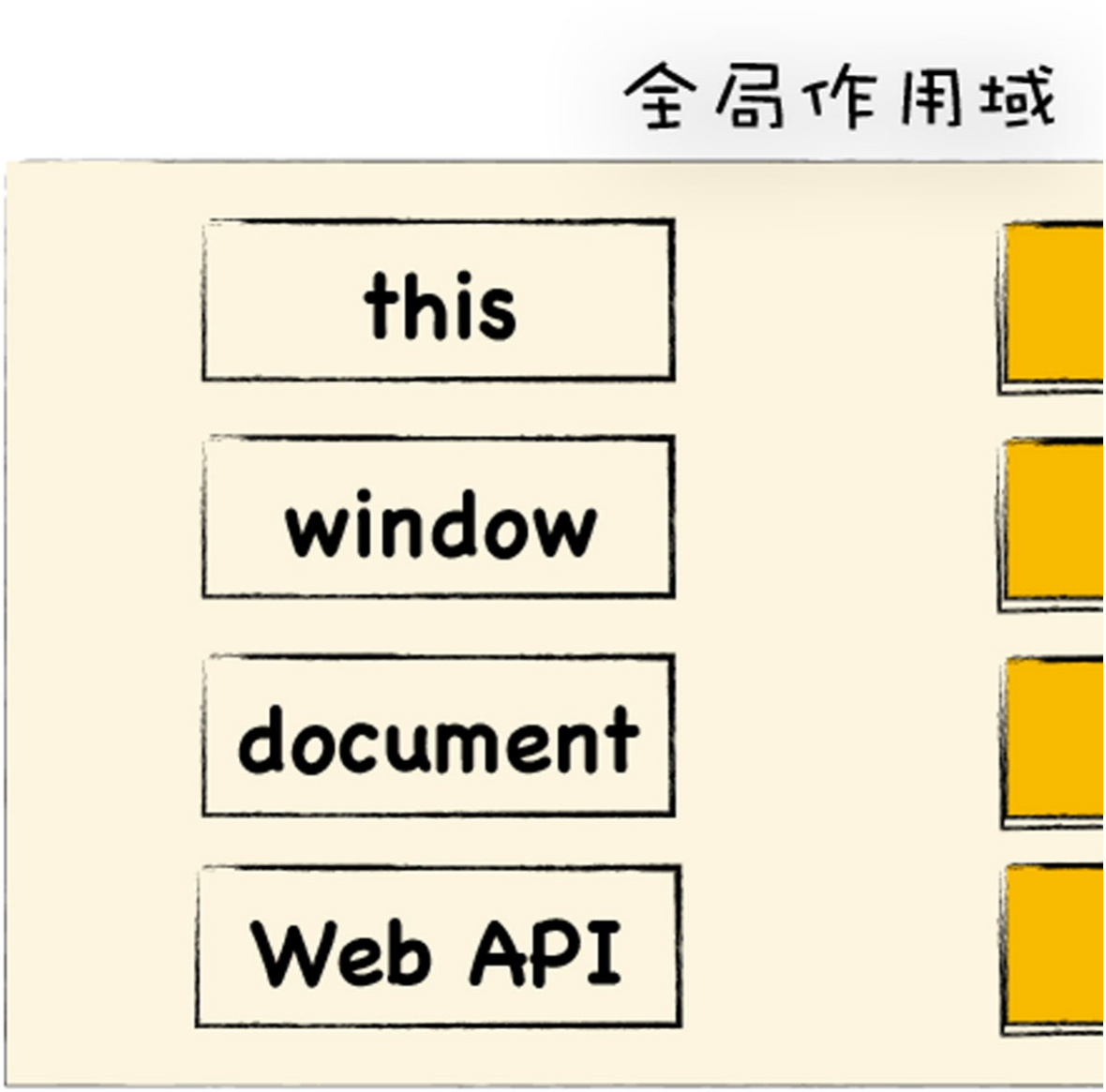
```
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

现在，我们结合V8执行这段代码的流程来具体分析下。首先当V8启动时，会创建全局作用域，全局作用域中包括了this、window等变量，还有一些全局的Web API接口，创建的作用域如下图所示：



V8启动之后，消息循环系统便开始工作了，这时候，我输入了这段代码，让其执行。

V8会先编译顶层代码，在编译过程中会将顶层定义的变量和声明的函数都添加到全局作用域中，最终的全局作用域如下图所示：

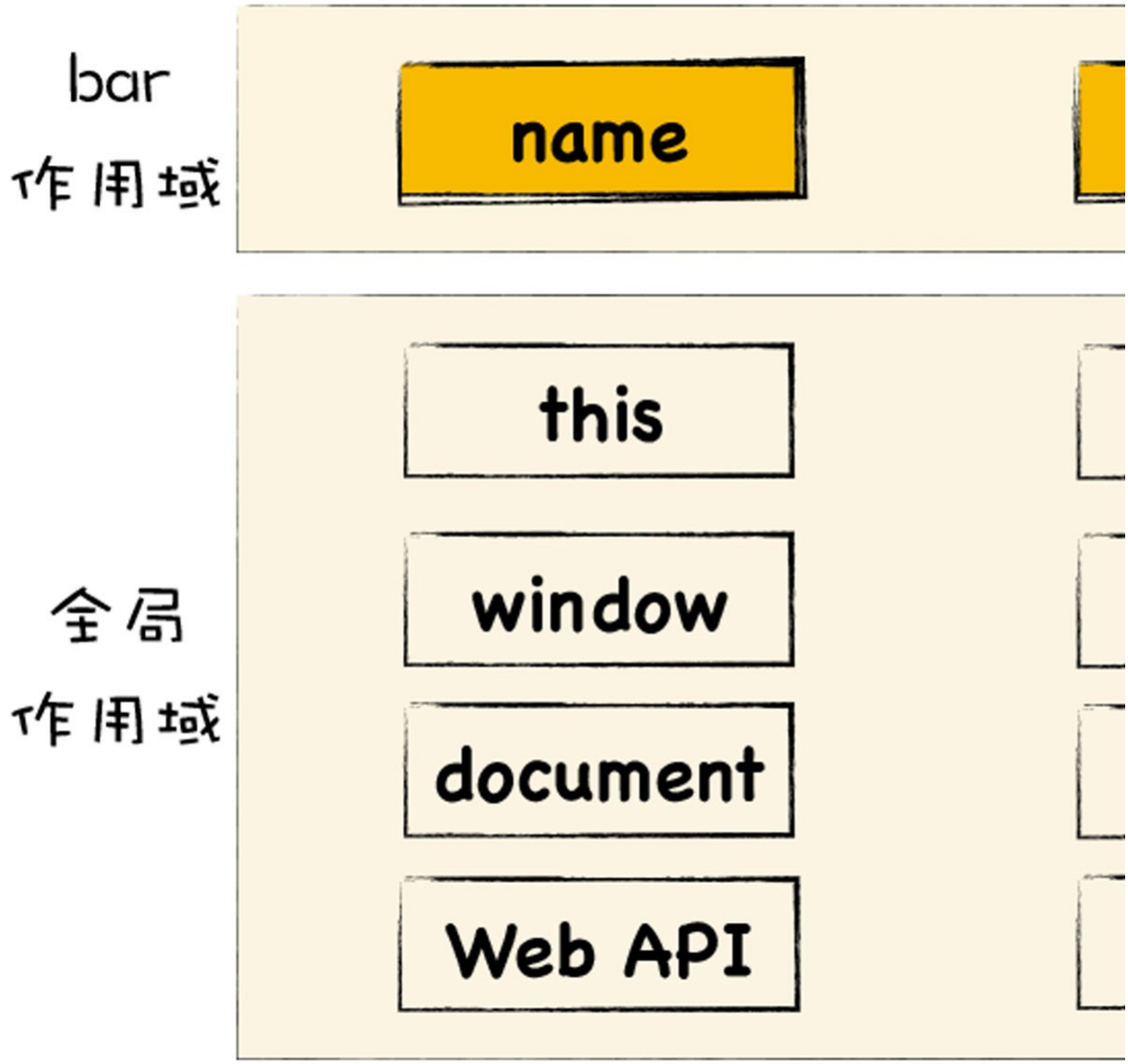


全局作用域创建完成之后，V8便进入了执行状态。前面我们介绍了变量提升，因为变量提升的原因，你可以把上面这段代码分解为如下两个部分：

```
//======解析阶段--实现变量提升=======
var name = undefined
var type = undefined
function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}
function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}



//====执行阶段========
name = '极客时间'
type = 'global'
bar()
```

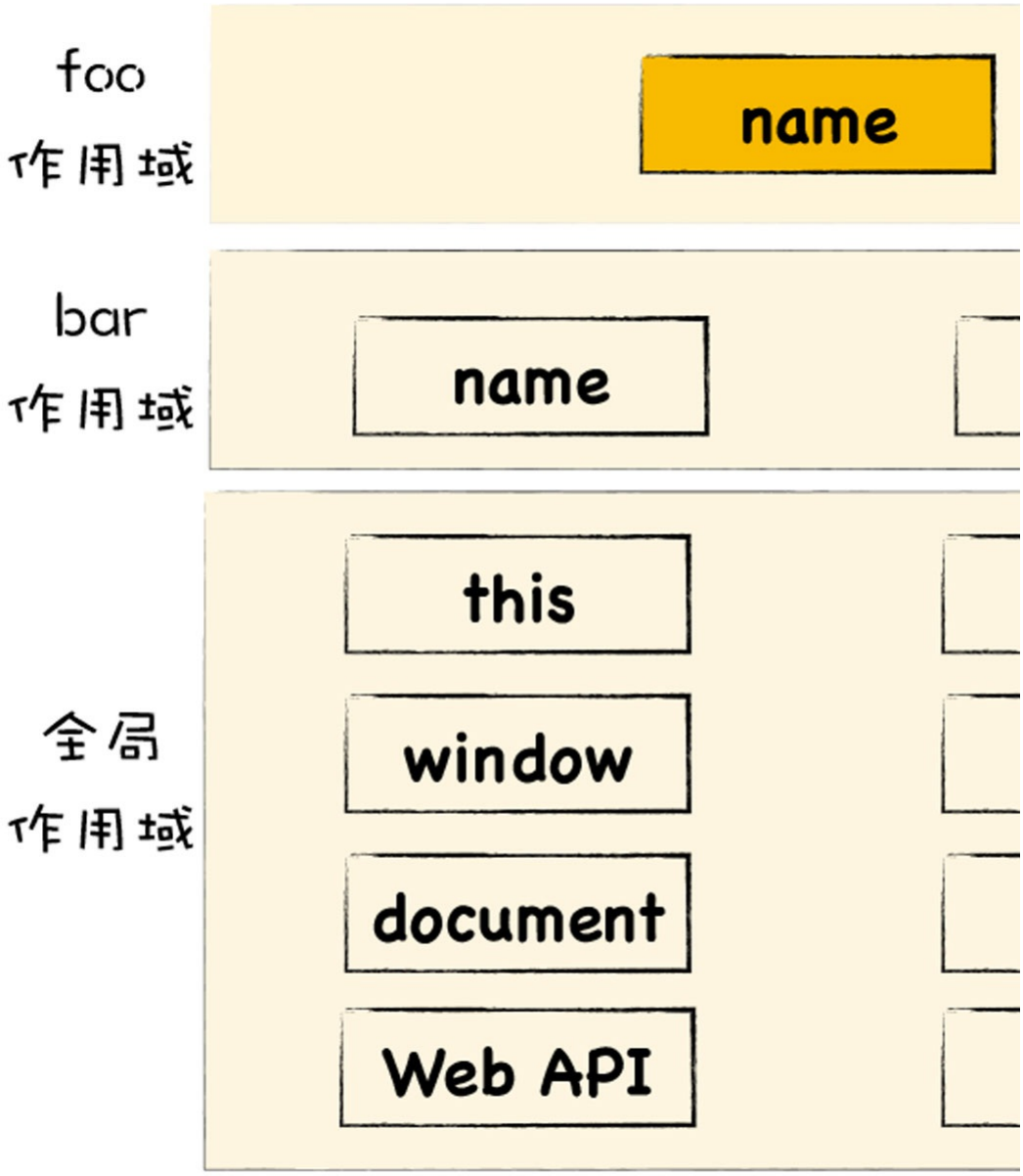
第一部分是在编译过程中完成的，此时全局作用中两个变量的值依然是undefined，然后进入执行阶段；第二部代码就是执行时的顺序，首先全局作用域中的两个变量赋值“极客时间”和“global”，然后就开始执行函数bar的调用了。

当V8执行bar函数的时候，同样需要经历两个阶段：编译和执行。在编译阶段，V8会为bar函数创建函数作用域，最终效果如下所示：



然后进入了bar函数的执行阶段。在bar函数中，只是简单地调用foo函数，因此V8又开始执行foo函数了。

同样，在编译foo函数的过程中，会创建foo函数的作用域，最终创建效果如下图所示：



好了，这时候我们就有了三个作用域了，分别是全局作用域、bar的函数作用域、foo的函数作用域。

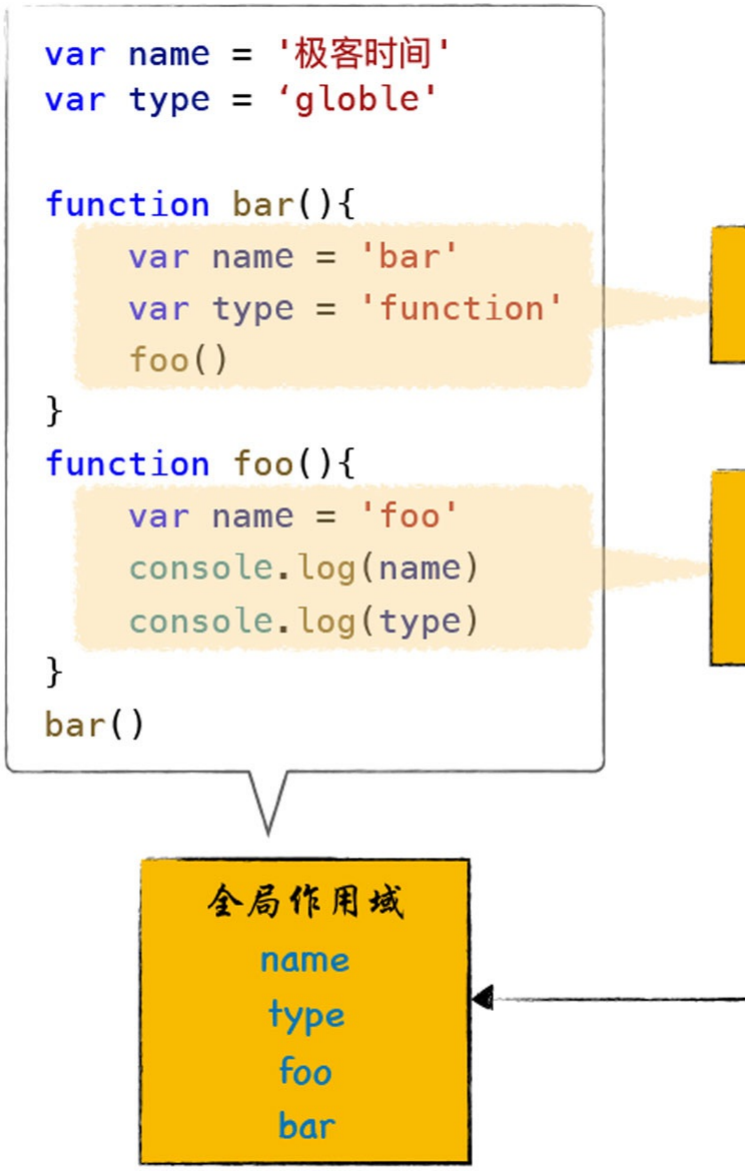
现在我们就可以将刚才提到的问题转换为作用域链的问题了：foo函数查找变量的路径到底是什么？

- 沿着foo函数作用域–>bar函数作用域–>全局作用域;
- 还是，沿着foo函数作用域—>全局作用域？

因为JavaScript是基于词法作用域的，词法作用域就是指，查找作用域的顺序是按照函数定义时的位置来决定的。bar和foo函数的外部代码都是全局代码，所以无论你是在bar函数中查找变量，还是在foo函数中查找变量，其查找顺序都是按照当前函数作用域–>全局作用域这个路径来的。

由于我们代码中的foo函数和bar函数都是在全局下面定义的，所以在foo函数中使用了type，最终打印出来的值就是全局作用域中的type。

你可以参考下面这张图：



另外，我再展开说一些。因为词法作用域是根据函数在代码中的位置来确定的，作用域是在声明函数时就确定好的了，所以我们也将词法作用域称为静态作用域。

和静态作用域相对的是动态作用域，动态作用域并不关心函数和作用域是如何声明以及在何处声明的，只关心它们从**何处调用**。换句话说，作用域链是基于调用栈的，而不是基于函数定义的位置的。（动态作用域不是本文讨论的重点，如果你感兴趣，可以参考《浏览器工作原理与实践》专栏中的《10 | 作用域链和闭包：代码中出现相同的变量，JavaScript引擎是如何选择的？》这一节。）

总结

今天，我们主要解释了一个问题，那就是在一个函数中，如果使用了一个变量，或者调用了另外一个函数，V8将会怎么去查找该变量或者函数。

为了解释清楚这个问题，我们引入了作用域的概念。作用域就是用来存放变量和函数的地方，全局作用域中存放了全局环境中声明的变量和函数，函数作用域中存放了函数中声明的变量和函数。当在某个函数中使用某个变量时，V8就会去这些作用域中查找相关变量。沿着这些作用域查找的路径，我们就称为作用域链。

要了解查找路径，我们需要明白词法作用域，词法作用域是按照代码定义时的位置决定的，而JavaScript所采用的作用域机制就是词法作用域，所以作用域链的路径就是按照词法作用域来实现的。

思考题

我将文章开头那段代码稍微调整了下，foo函数并不是在全局环境中声明的，而是在bar函数中声明的，改造后的代码如下所示：

```
var name = '极客时间'
var type = 'global'
function bar() {
    var type = 'function'
    function foo() {
        console.log(type)
    }
    foo()
}
bar()
```

那么执行这段代码之后，打印出来的内容是什么？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

在前面我们介绍了JavaScript的继承是基于原型链的，原型链将一个个原型对象串起来，从而实现对象属性的查找，今天我们要聊一个和原型链类似的话题，那就是作用域链。

作用域链就是将一个个作用域串起来，实现变量查找的路径。讨论作用域链，实际就是在讨论按照什么路径查找变量的问题。

我们知道，作用域就是存放变量和函数的地方，全局环境有全局作用域，全局作用域中存放了全局变量和全局函数。每个函数也有自己的作用域，函数作用域中存放了函数中定义的变量。

当在函数内部使用一个变量的时候，V8便会去作用域中去查找。我们通过一段在函数内部查找变量的代码来具体看一下：

```
var name = '极客时间'
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

在这段代码中，我们在全局环境中声明了变量name和type，同时还定义了bar函数和foo函数，在bar函数中又再次定义了变量name和type，在foo函数中再次定义了变量name。

函数的调用关系是：在全局环境中调用bar函数，在bar函数中调用foo函数，在foo函数中打印出来变量name和type的值。

当执行到foo函数时，首先需要打印出变量name的值，而我们在三个地方都定义了变量name，那么究竟应该使用哪个变量呢？

在foo函数中使用了变量name，那么V8就应该先使用foo函数内部定义的变量name，最终的结果确实如此，也符合我们的直觉。

接下来，foo函数继续打印变量type，但是在foo函数内部并没有定义变量type，而是在全局环境中和调用foo函数的bar函数中分别定义了变量type，那么这时候的问题来了，你觉得foo函数中打印出来的变量type是bar函数中的，还是全局环境中的呢？

什么是函数作用域和全局作用域？

要解释清楚这个问题，我们需要从作用域的工作原理讲起。

每个函数在执行时都需要查找自己的作用域，我们称为函数作用域，在执行阶段，在执行一个函数时，当该函数需要使用某个变量或者调用了某个函数时，便会优先在该函数作用域中查找相关内容。

我们再来看一段代码：

```
var x = 4
var test
function test_scope() {
    var name = 'foo'
    console.log(name)
    console.log(type)
    console.log(test)
    var type = 'function'
    test = 1
    console.log(x)
}
test_scope()
```

在上面的代码中，我们定义了一个test_scope函数，那么在V8执行test_scope函数的时候，在编译阶段会为test_scope函数创建一个作用域，在test_scope函数中定义的变量和声明的函数都会丢到该作用域中，因为我们在test_scope函数中定了三个变量，那么常见的作用域就包含有这三个变量。

你可以通过Chrome的控制台来直观感受下test_scope函数的作用域，先打开包含这段代码的页面，然后打开开发者工具，接着在test_scope函数中的第二段代码加上断点，然后刷新该页面。当执行到该断点时，V8会暂停整个执行流程，这时候我们就可以通过右边的区域面板来查看当前函数的执行状态。

你可以参考图中右侧的Scope项，然后点击展开该项，这个Local就是当前函数test_scope的作用域。在test_scope函数中定义的变量都包含到了Local中，如变量name、type，另外系统还为我们添加了另外一个隐藏变量this，V8还会默认将隐藏变量this存放到作用域中。

另外你还需要注意下，第一个test1，我并没有采用var等关键字来声明，所以test1并不会出现在test_scope函数的作用域中，而是属于this所指向的对象。（this的工作机制不是本文讨论的重点，不展开介绍。如果你感兴趣，可以在《浏览器工作原理与实践》专栏中《11 | this：从JavaScript执行上下文的视角讲清楚this》这一讲查看。）

那么另一个问题来了，我在test_scope函数使用了变量x，但是在test_scope函数的作用域中，并没有定义变量x，那么V8应该如何获取变量x?

如果在当前函数作用域中没有查找到变量，那么V8会去全局作用域中去查找，这个查找的线路就称为作用域链。

全局作用域和函数作用域类似，也是存放变量和函数的地方，但是它们还是有点不一样：**全局作用域是在V8启动过程中就创建了，且一直保存在内存中不会被销毁的，直至V8退出。 而函数作用域是在执行该函数时创建的，当函数执行结束之后，函数作用域就随之被销毁掉了。**

全局作用域中包含了很多全局变量，比如全局的this值，如果是浏览器，全局作用域中还有window、document、opener等非常多的方法和对象，如果是node环境，那么会有Global、File等内容。

V8启动之后就进入正常的消息循环状态，这时候就可以执行代码了，比如执行到上面那段脚本时，V8会先解析顶层(Top Level)代码，我们可以看到，在顶层代码中定义了变量x，这时候V8就会将变量x添加到全局作用域中。

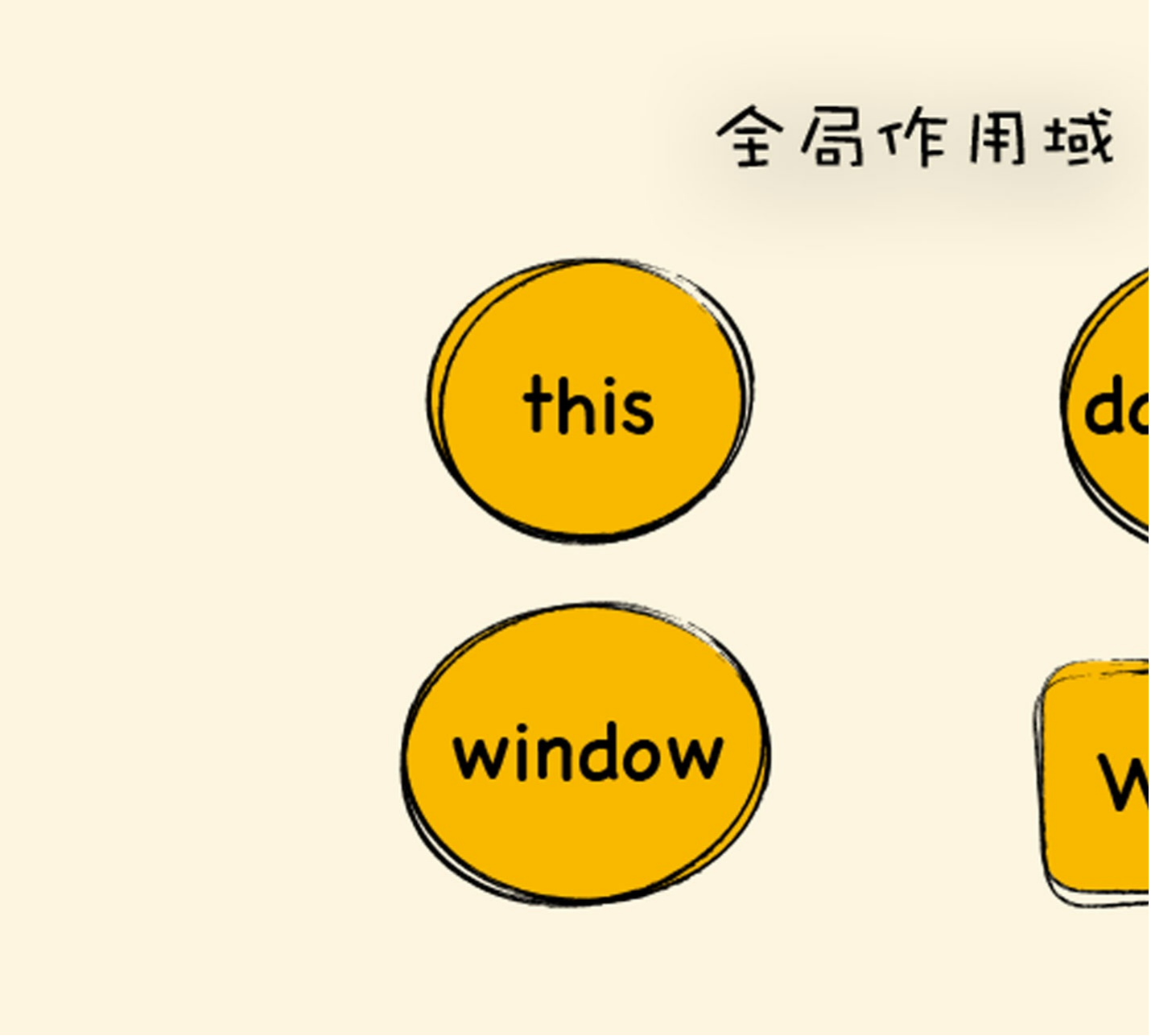
作用域链是怎么工作的？

理解了作用域和作用域链，我们再回过头来看文章开头的那道思考题：“foo函数中打印出来的变量type是bar函数中的呢，还是全局环境中的呢?”我把这段代码复制到下面：

```
var name = '极客时间'
```

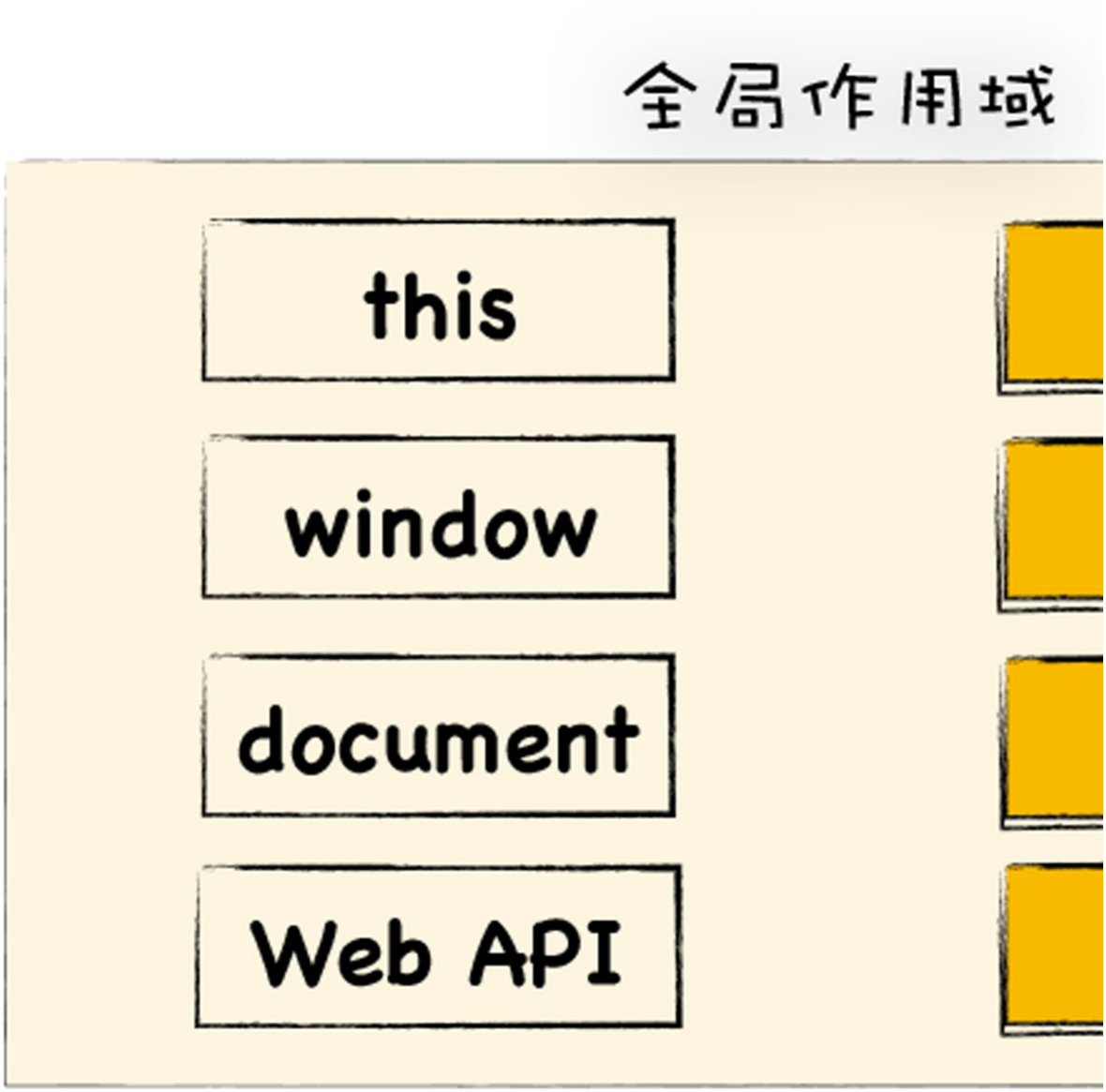
```
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

现在，我们结合V8执行这段代码的流程来具体分析下。首先当V8启动时，会创建全局作用域，全局作用域中包括了this、window等变量，还有一些全局的Web API接口，创建的作用域如下图所示：



V8启动之后，消息循环系统便开始工作了，这时候，我输入了这段代码，让其执行。

V8会先编译顶层代码，在编译过程中会将顶层定义的变量和声明的函数都添加到全局作用域中，最终的全局作用域如下图所示：

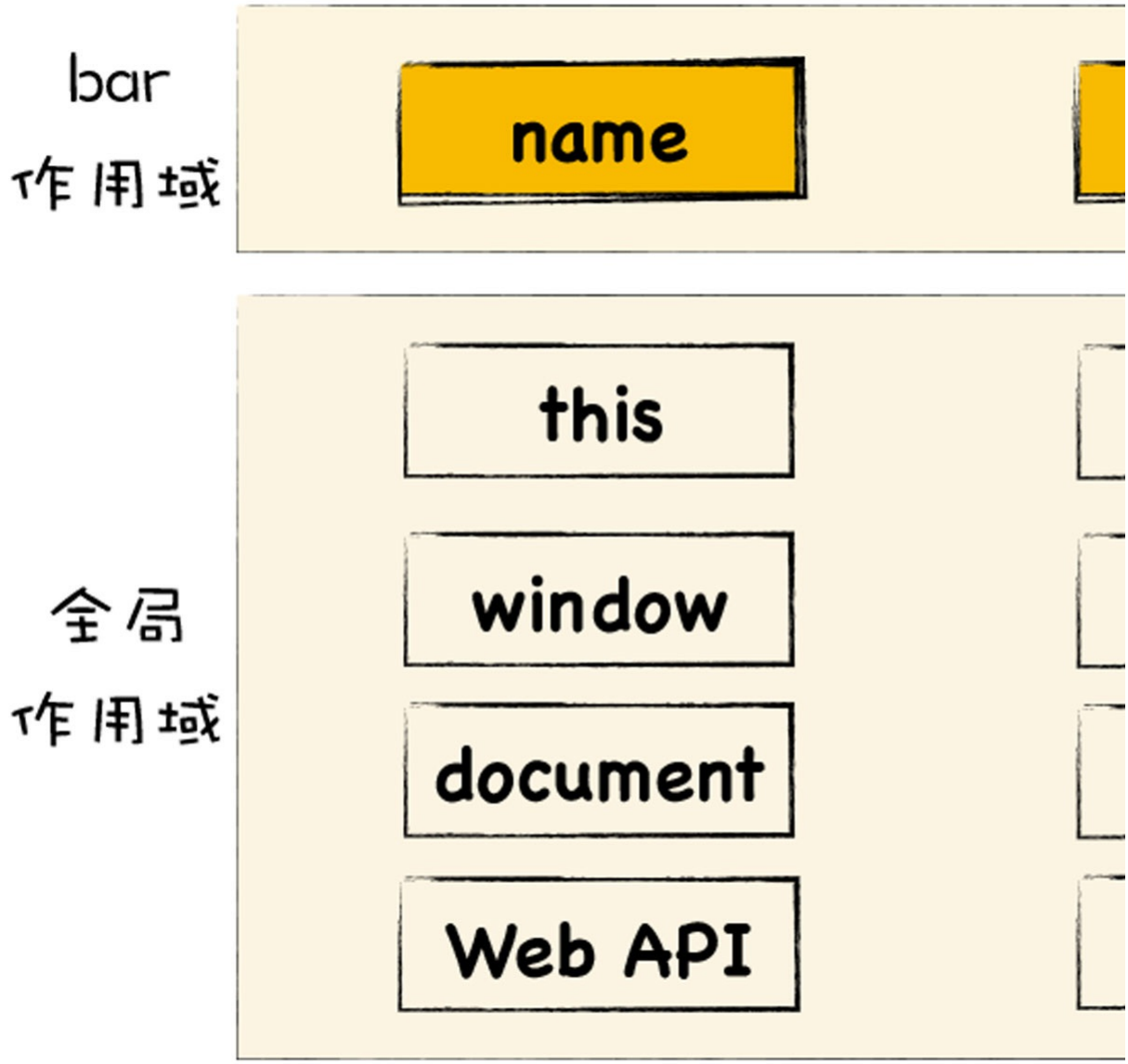


全局作用域创建完成之后，V8便进入了执行状态。前面我们介绍了变量提升，因为变量提升的原因，你可以把上面这段代码分解为如下两个部分：

```
//======解析阶段--实现变量提升=======
var name = undefined
var type = undefined
function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}
function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}



//====执行阶段========
name = '极客时间'
type = 'global'
bar()
```

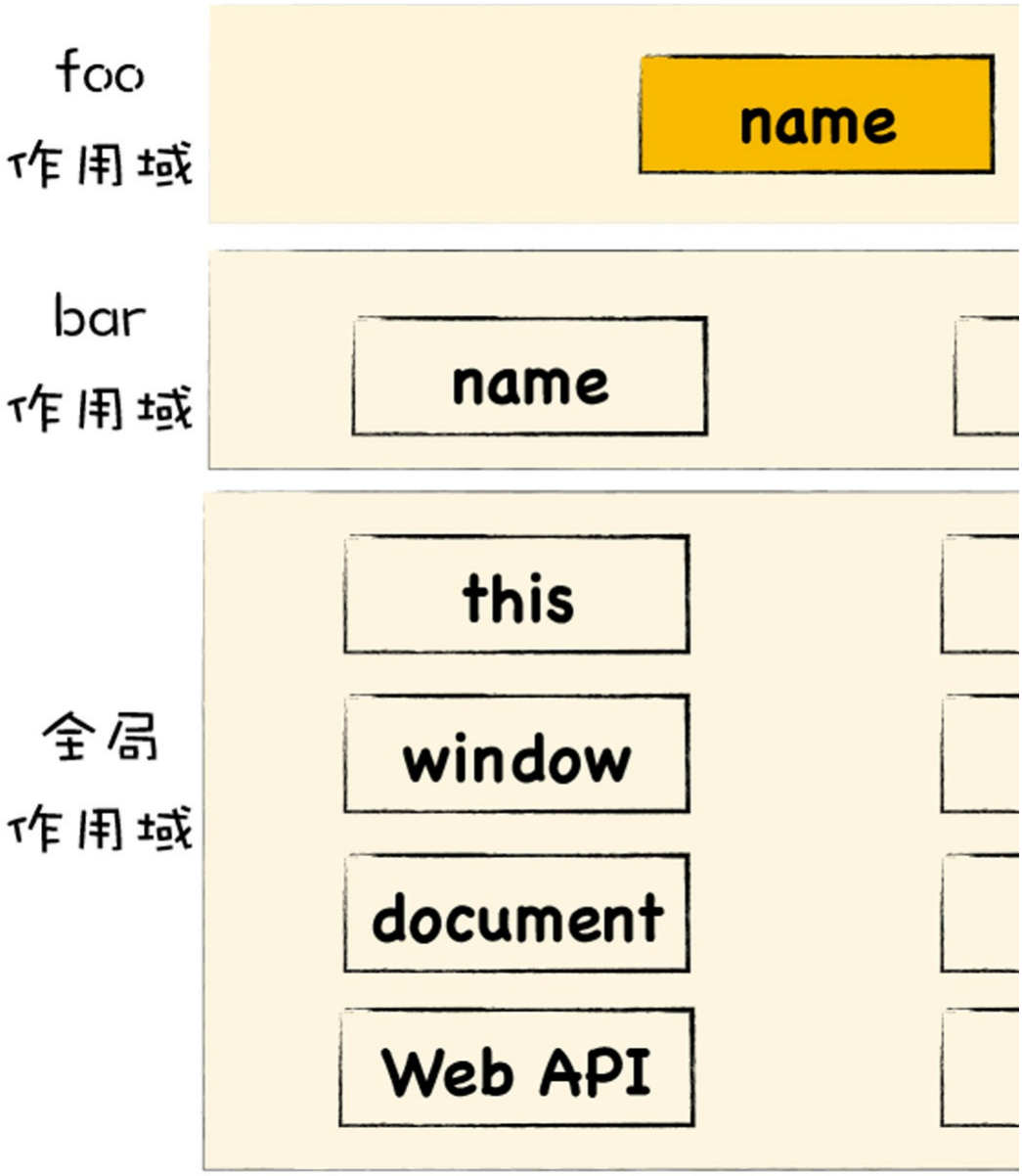
第一部分是在编译过程中完成的，此时全局作用中两个变量的值依然是undefined，然后进入执行阶段；第二部代码就是执行时的顺序，首先全局作用域中的两个变量赋值“极客时间”和“global”，然后就开始执行函数bar的调用了。

当V8执行bar函数的时候，同样需要经历两个阶段：编译和执行。在编译阶段，V8会为bar函数创建函数作用域，最终效果如下所示：



然后进入了bar函数的执行阶段。在bar函数中，只是简单地调用foo函数，因此V8又开始执行foo函数了。

同样，在编译foo函数的过程中，会创建foo函数的作用域，最终创建效果如下图所示：



好了，这时候我们就有了三个作用域了，分别是全局作用域、bar的函数作用域、foo的函数作用域。

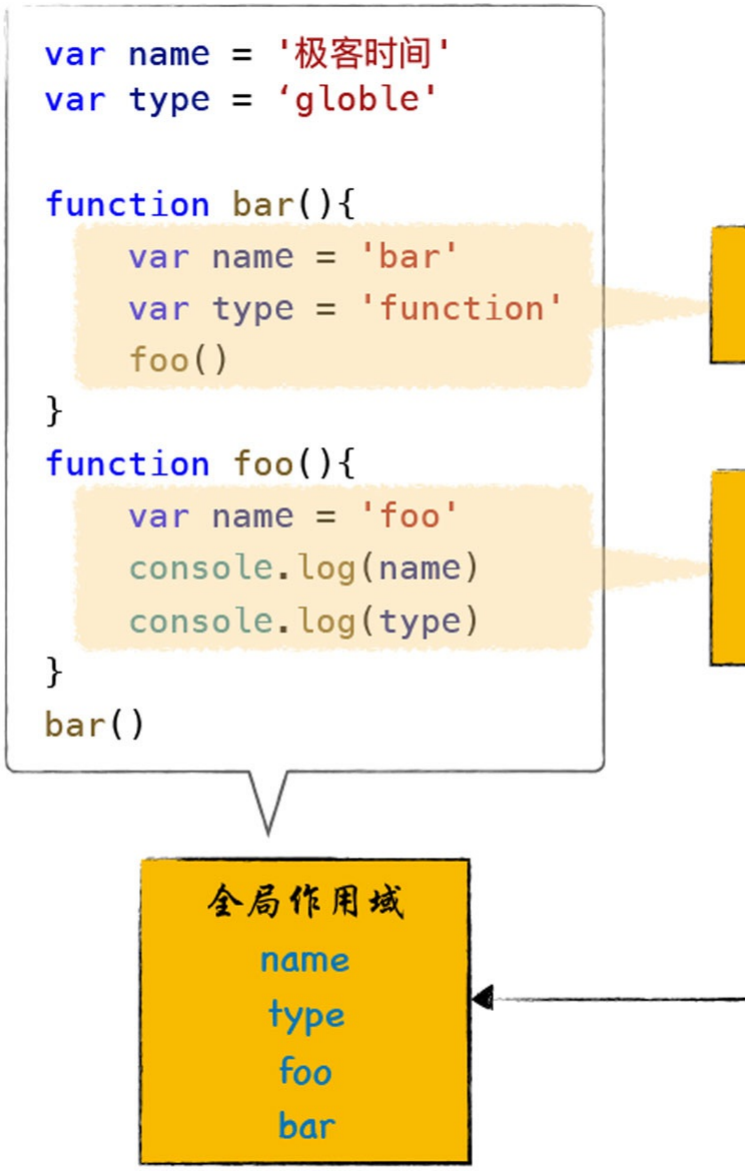
现在我们就可以将刚才提到的问题转换为作用域链的问题了：foo函数查找变量的路径到底是什么？

- 沿着foo函数作用域–>bar函数作用域–>全局作用域;
- 还是，沿着foo函数作用域—>全局作用域？

因为JavaScript是基于词法作用域的，词法作用域就是指，查找作用域的顺序是按照函数定义时的位置来决定的。bar和foo函数的外部代码都是全局代码，所以无论你是在bar函数中查找变量，还是在foo函数中查找变量，其查找顺序都是按照当前函数作用域–>全局作用域这个路径来的。

由于我们代码中的foo函数和bar函数都是在全局下面定义的，所以在foo函数中使用了type，最终打印出来的值就是全局作用域中的type。

你可以参考下面这张图：



另外，我再展开说一些。因为词法作用域是根据函数在代码中的位置来确定的，作用域是在声明函数时就确定好的了，所以我们也将词法作用域称为静态作用域。

和静态作用域相对的是动态作用域，动态作用域并不关心函数和作用域是如何声明以及在何处声明的，只关心它们从**何处调用**。换句话说，作用域链是基于调用栈的，而不是基于函数定义的位置的。（动态作用域不是本文讨论的重点，如果你感兴趣，可以参考《浏览器工作原理与实践》专栏中的《10 | 作用域链和闭包：代码中出现相同的变量，JavaScript引擎是如何选择的？》这一节。）

总结

今天，我们主要解释了一个问题，那就是在一个函数中，如果使用了一个变量，或者调用了另外一个函数，V8将会怎么去查找该变量或者函数。

为了解释清楚这个问题，我们引入了作用域的概念。作用域就是用来存放变量和函数的地方，全局作用域中存放了全局环境中声明的变量和函数，函数作用域中存放了函数中声明的变量和函数。当在某个函数中使用某个变量时，V8就会去这些作用域中查找相关变量。沿着这些作用域查找的路径，我们就称为作用域链。

要了解查找路径，我们需要明白词法作用域，词法作用域是按照代码定义时的位置决定的，而JavaScript所采用的作用域机制就是词法作用域，所以作用域链的路径就是按照词法作用域来实现的。

思考题

我将文章开头那段代码稍微调整了下，foo函数并不是在全局环境中声明的，而是在bar函数中声明的，改造后的代码如下所示：

```
var name = '极客时间'
var type = 'global'
function bar() {
    var type = 'function'
    function foo() {
        console.log(type)
    }
    foo()
}
bar()
```

那么执行这段代码之后，打印出来的内容是什么？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

在前面我们介绍了JavaScript的继承是基于原型链的，原型链将一个个原型对象串起来，从而实现对象属性的查找，今天我们要聊一个和原型链类似的话题，那就是作用域链。

作用域链就是将一个个作用域串起来，实现变量查找的路径。讨论作用域链，实际就是在讨论按照什么路径查找变量的问题。

我们知道，作用域就是存放变量和函数的地方，全局环境有全局作用域，全局作用域中存放了全局变量和全局函数。每个函数也有自己的作用域，函数作用域中存放了函数中定义的变量。

当在函数内部使用一个变量的时候，V8便会去作用域中去查找。我们通过一段在函数内部查找变量的代码来具体看一下：

```
var name = '极客时间'
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

在这段代码中，我们在全局环境中声明了变量name和type，同时还定义了bar函数和foo函数，在bar函数中又再次定义了变量name和type，在foo函数中再次定义了变量name。

函数的调用关系是：在全局环境中调用bar函数，在bar函数中调用foo函数，在foo函数中打印出来变量name和type的值。

当执行到foo函数时，首先需要打印出变量name的值，而我们在三个地方都定义了变量name，那么究竟应该使用哪个变量呢？

在foo函数中使用了变量name，那么V8就应该先使用foo函数内部定义的变量name，最终的结果确实如此，也符合我们的直觉。

接下来，foo函数继续打印变量type，但是在foo函数内部并没有定义变量type，而是在全局环境中和调用foo函数的bar函数中分别定义了变量type，那么这时候的问题来了，你觉得foo函数中打印出来的变量type是bar函数中的，还是全局环境中的呢？

什么是函数作用域和全局作用域？

要解释清楚这个问题，我们需要从作用域的工作原理讲起。

每个函数在执行时都需要查找自己的作用域，我们称为函数作用域，在执行阶段，在执行一个函数时，当该函数需要使用某个变量或者调用了某个函数时，便会优先在该函数作用域中查找相关内容。

我们再来看一段代码：

```
var x = 4
var test
function test_scope() {
    var name = 'foo'
    console.log(name)
    console.log(type)
    console.log(test)
    var type = 'function'
    test = 1
    console.log(x)
}
test_scope()
```

在上面的代码中，我们定义了一个test_scope函数，那么在V8执行test_scope函数的时候，在编译阶段会为test_scope函数创建一个作用域，在test_scope函数中定义的变量和声明的函数都会丢到该作用域中，因为我们在test_scope函数中定了三个变量，那么常见的作用域就包含有这三个变量。

你可以通过Chrome的控制台来直观感受下test_scope函数的作用域，先打开包含这段代码的页面，然后打开开发者工具，接着在test_scope函数中的第二段代码加上断点，然后刷新该页面。当执行到该断点时，V8会暂停整个执行流程，这时候我们就可以通过右边的区域面板来查看当前函数的执行状态。

你可以参考图中右侧的Scope项，然后点击展开该项，这个Local就是当前函数test_scope的作用域。在test_scope函数中定义的变量都包含到了Local中，如变量name、type，另外系统还为我们添加了另外一个隐藏变量this，V8还会默认将隐藏变量this存放到作用域中。

另外你还需要注意下，第一个test1，我并没有采用var等关键字来声明，所以test1并不会出现在test_scope函数的作用域中，而是属于this所指向的对象。（this的工作机制不是本文讨论的重点，不展开介绍。如果你感兴趣，可以在《浏览器工作原理与实践》专栏中《11 | this：从JavaScript执行上下文的视角讲清楚this》这一讲查看。）

那么另一个问题来了，我在test_scope函数使用了变量x，但是在test_scope函数的作用域中，并没有定义变量x，那么V8应该如何获取变量x?

如果在当前函数作用域中没有查找到变量，那么V8会去全局作用域中去查找，这个查找的线路就称为作用域链。

全局作用域和函数作用域类似，也是存放变量和函数的地方，但是它们还是有点不一样：**全局作用域是在V8启动过程中就创建了，且一直保存在内存中不会被销毁的，直至V8退出。 而函数作用域是在执行该函数时创建的，当函数执行结束之后，函数作用域就随之被销毁掉了。**

全局作用域中包含了很多全局变量，比如全局的this值，如果是浏览器，全局作用域中还有window、document、opener等非常多的方法和对象，如果是node环境，那么会有Global、File等内容。

V8启动之后就进入正常的消息循环状态，这时候就可以执行代码了，比如执行到上面那段脚本时，V8会先解析顶层(Top Level)代码，我们可以看到，在顶层代码中定义了变量x，这时候V8就会将变量x添加到全局作用域中。

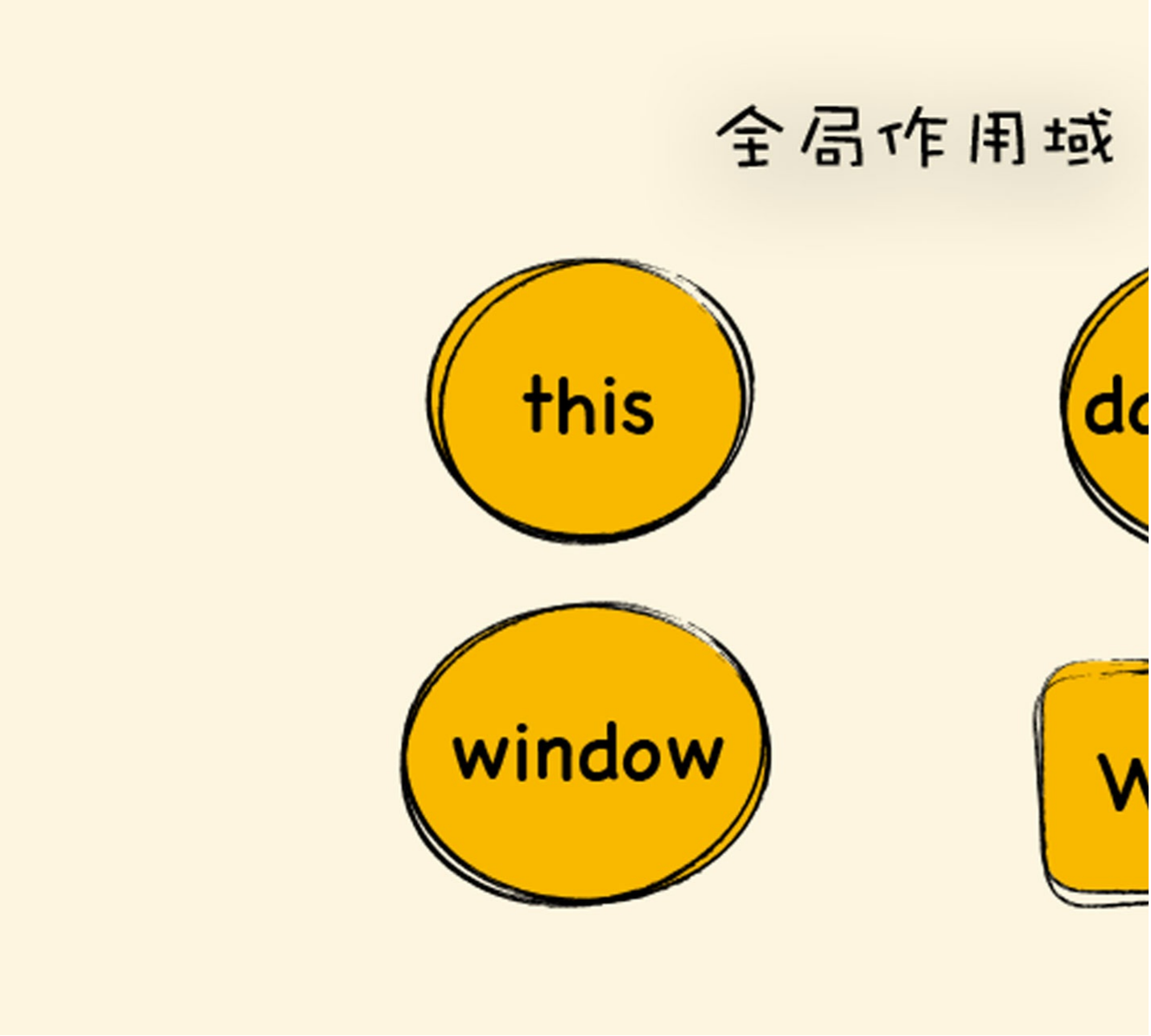
作用域链是怎么工作的？

理解了作用域和作用域链，我们再回过头来看文章开头的那道思考题：“foo函数中打印出来的变量type是bar函数中的呢，还是全局环境中的呢?”我把这段代码复制到下面：

```
var name = '极客时间'
```

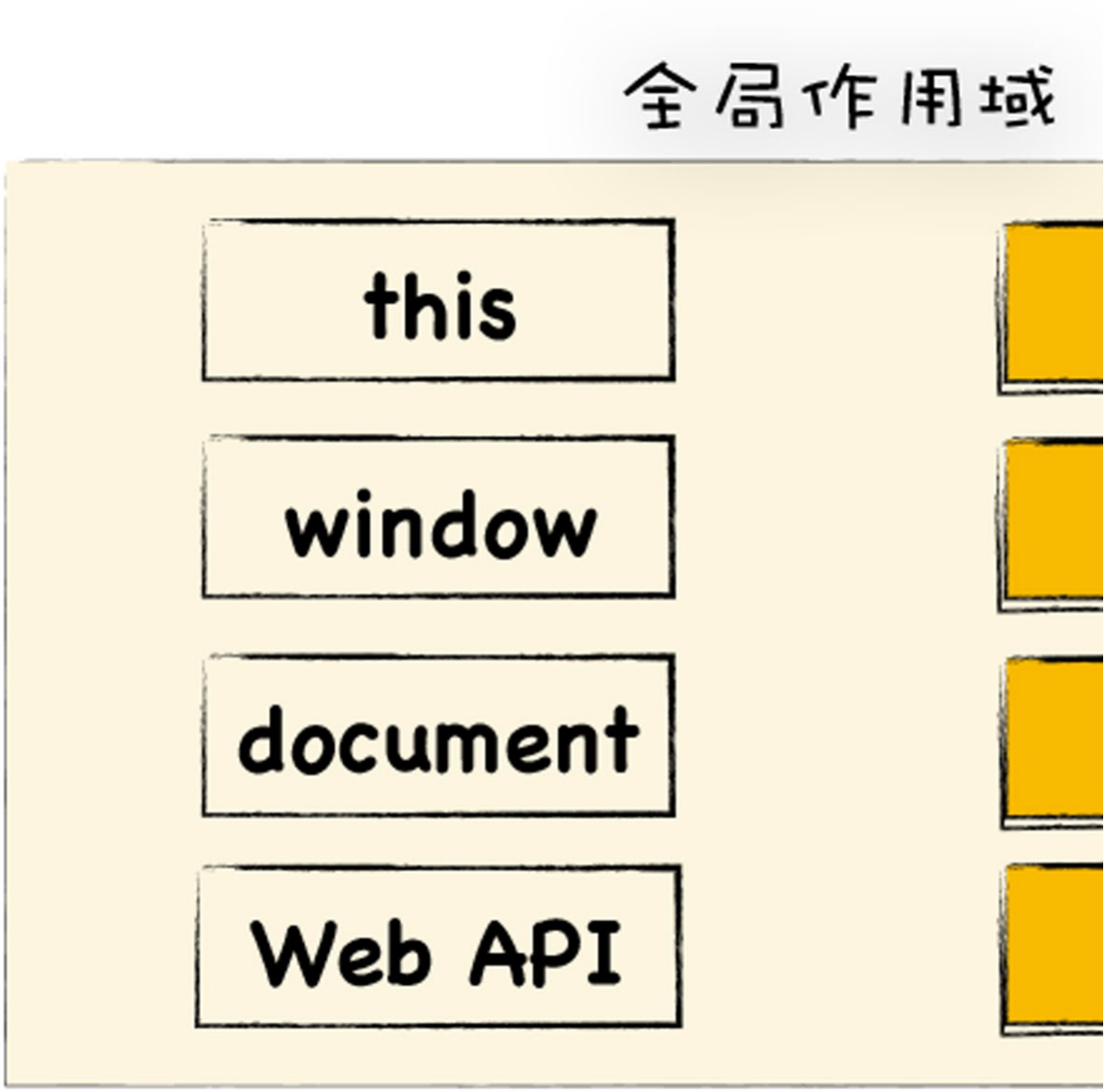
```
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

现在，我们结合V8执行这段代码的流程来具体分析下。首先当V8启动时，会创建全局作用域，全局作用域中包括了this、window等变量，还有一些全局的Web API接口，创建的作用域如下图所示：



V8启动之后，消息循环系统便开始工作了，这时候，我输入了这段代码，让其执行。

V8会先编译顶层代码，在编译过程中会将顶层定义的变量和声明的函数都添加到全局作用域中，最终的全局作用域如下图所示：

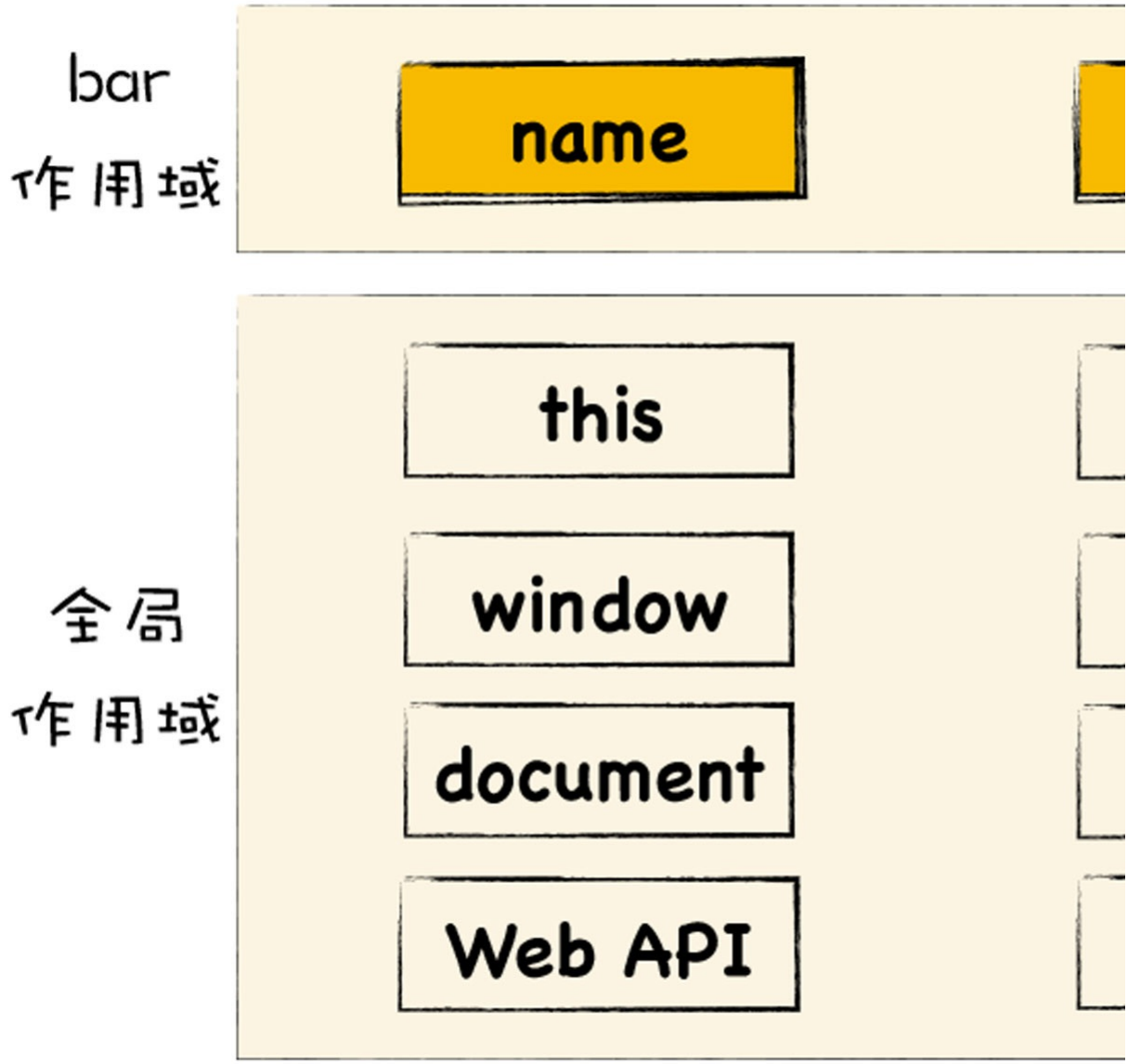


全局作用域创建完成之后，V8便进入了执行状态。前面我们介绍了变量提升，因为变量提升的原因，你可以把上面这段代码分解为如下两个部分：

```
//======解析阶段--实现变量提升=======
var name = undefined
var type = undefined
function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}
function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}



//====执行阶段========
name = '极客时间'
type = 'global'
bar()
```

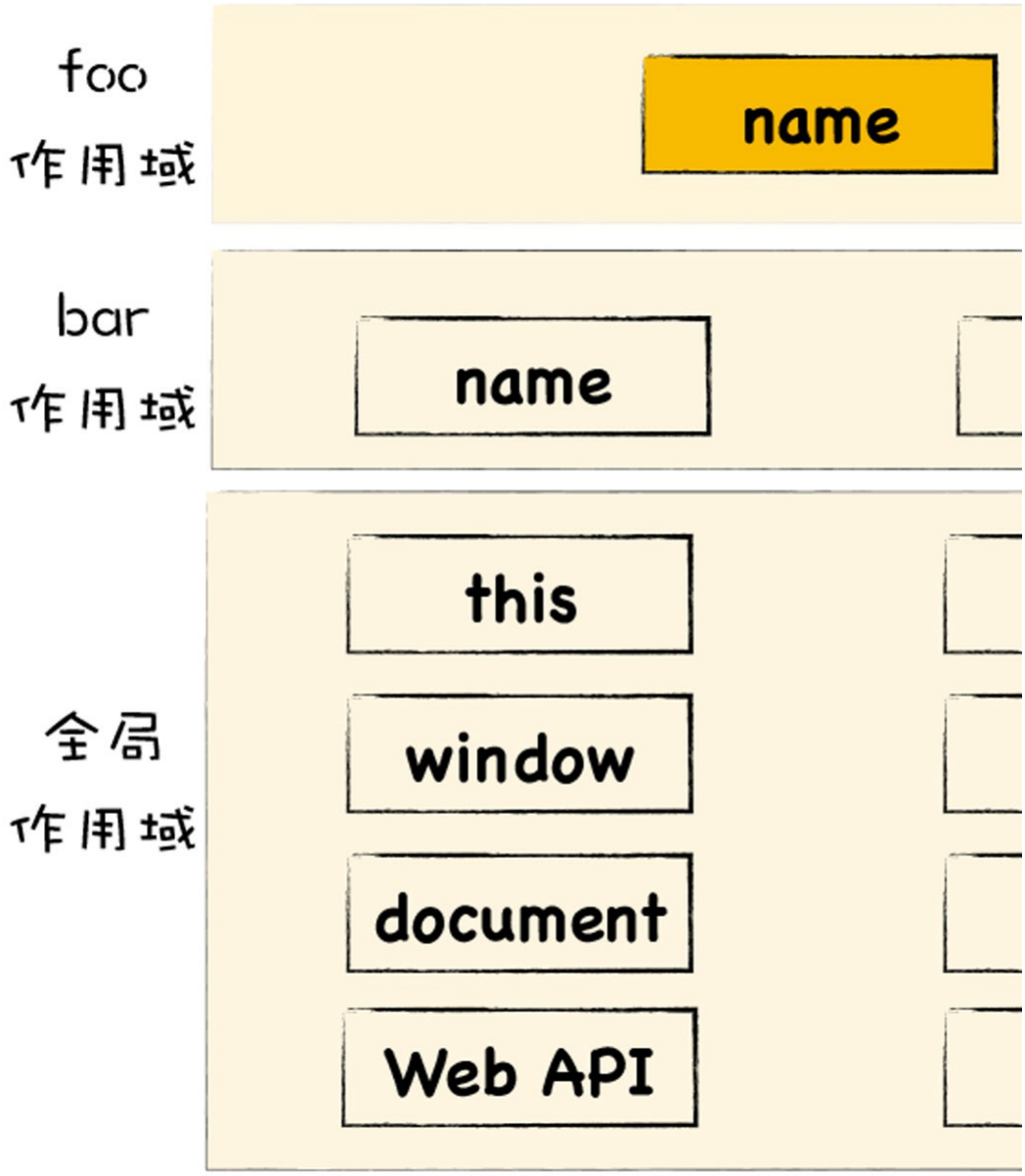
第一部分是在编译过程中完成的，此时全局作用中两个变量的值依然是undefined，然后进入执行阶段；第二部代码就是执行时的顺序，首先全局作用域中的两个变量赋值“极客时间”和“global”，然后就开始执行函数bar的调用了。

当V8执行bar函数的时候，同样需要经历两个阶段：编译和执行。在编译阶段，V8会为bar函数创建函数作用域，最终效果如下所示：



然后进入了bar函数的执行阶段。在bar函数中，只是简单地调用foo函数，因此V8又开始执行foo函数了。

同样，在编译foo函数的过程中，会创建foo函数的作用域，最终创建效果如下图所示：



好了，这时候我们就有了三个作用域了，分别是全局作用域、bar的函数作用域、foo的函数作用域。

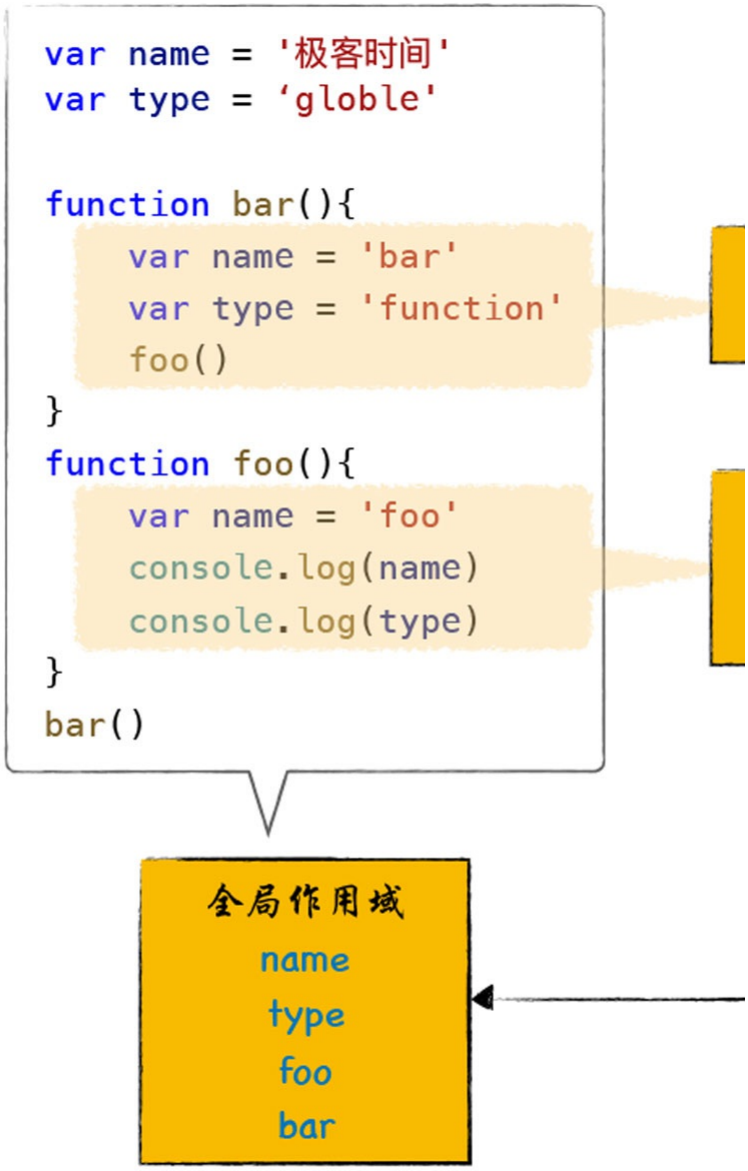
现在我们就可以将刚才提到的问题转换为作用域链的问题了：foo函数查找变量的路径到底是什么？

- 沿着foo函数作用域–>bar函数作用域–>全局作用域;
- 还是，沿着foo函数作用域—>全局作用域？

因为JavaScript是基于词法作用域的，词法作用域就是指，查找作用域的顺序是按照函数定义时的位置来决定的。bar和foo函数的外部代码都是全局代码，所以无论你是在bar函数中查找变量，还是在foo函数中查找变量，其查找顺序都是按照当前函数作用域–>全局作用域这个路径来的。

由于我们代码中的foo函数和bar函数都是在全局下面定义的，所以在foo函数中使用了type，最终打印出来的值就是全局作用域中的type。

你可以参考下面这张图：



另外，我再展开说一些。因为词法作用域是根据函数在代码中的位置来确定的，作用域是在声明函数时就确定好的了，所以我们也将词法作用域称为静态作用域。

和静态作用域相对的是动态作用域，动态作用域并不关心函数和作用域是如何声明以及在何处声明的，只关心它们从**何处调用**。换句话说，作用域链是基于调用栈的，而不是基于函数定义的位置的。（动态作用域不是本文讨论的重点，如果你感兴趣，可以参考《浏览器工作原理与实践》专栏中的《10 | 作用域链和闭包：代码中出现相同的变量，JavaScript引擎是如何选择的？》这一节。）

总结

今天，我们主要解释了一个问题，那就是在一个函数中，如果使用了一个变量，或者调用了另外一个函数，V8将会怎么去查找该变量或者函数。

为了解释清楚这个问题，我们引入了作用域的概念。作用域就是用来存放变量和函数的地方，全局作用域中存放了全局环境中声明的变量和函数，函数作用域中存放了函数中声明的变量和函数。当在某个函数中使用某个变量时，V8就会去这些作用域中查找相关变量。沿着这些作用域查找的路径，我们就称为作用域链。

要了解查找路径，我们需要明白词法作用域，词法作用域是按照代码定义时的位置决定的，而JavaScript所采用的作用域机制就是词法作用域，所以作用域链的路径就是按照词法作用域来实现的。

思考题

我将文章开头那段代码稍微调整了下，foo函数并不是在全局环境中声明的，而是在bar函数中声明的，改造后的代码如下所示：

```
var name = '极客时间'
var type = 'global'
function bar() {
    var type = 'function'
    function foo() {
        console.log(type)
    }
    foo()
}
bar()
```

那么执行这段代码之后，打印出来的内容是什么？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

在前面我们介绍了JavaScript的继承是基于原型链的，原型链将一个个原型对象串起来，从而实现对象属性的查找，今天我们要聊一个和原型链类似的话题，那就是作用域链。

作用域链就是将一个个作用域串起来，实现变量查找的路径。讨论作用域链，实际就是在讨论按照什么路径查找变量的问题。

我们知道，作用域就是存放变量和函数的地方，全局环境有全局作用域，全局作用域中存放了全局变量和全局函数。每个函数也有自己的作用域，函数作用域中存放了函数中定义的变量。

当在函数内部使用一个变量的时候，V8便会去作用域中去查找。我们通过一段在函数内部查找变量的代码来具体看一下：

```
var name = '极客时间'
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

在这段代码中，我们在全局环境中声明了变量name和type，同时还定义了bar函数和foo函数，在bar函数中又再次定义了变量name和type，在foo函数中再次定义了变量name。

函数的调用关系是：在全局环境中调用bar函数，在bar函数中调用foo函数，在foo函数中打印出来变量name和type的值。

当执行到foo函数时，首先需要打印出变量name的值，而我们在三个地方都定义了变量name，那么究竟应该使用哪个变量呢？

在foo函数中使用了变量name，那么V8就应该先使用foo函数内部定义的变量name，最终的结果确实如此，也符合我们的直觉。

接下来，foo函数继续打印变量type，但是在foo函数内部并没有定义变量type，而是在全局环境中和调用foo函数的bar函数中分别定义了变量type，那么这时候的问题来了，你觉得foo函数中打印出来的变量type是bar函数中的，还是全局环境中的呢？

什么是函数作用域和全局作用域？

要解释清楚这个问题，我们需要从作用域的工作原理讲起。

每个函数在执行时都需要查找自己的作用域，我们称为函数作用域，在执行阶段，在执行一个函数时，当该函数需要使用某个变量或者调用了某个函数时，便会优先在该函数作用域中查找相关内容。

我们再来看一段代码：

```
var x = 4
var test
function test_scope() {
    var name = 'foo'
    console.log(name)
    console.log(type)
    console.log(test)
    var type = 'function'
    test = 1
    console.log(x)
}
test_scope()
```

在上面的代码中，我们定义了一个test_scope函数，那么在V8执行test_scope函数的时候，在编译阶段会为test_scope函数创建一个作用域，在test_scope函数中定义的变量和声明的函数都会丢到该作用域中，因为我们在test_scope函数中定了三个变量，那么常见的作用域就包含有这三个变量。

你可以通过Chrome的控制台来直观感受下test_scope函数的作用域，先打开包含这段代码的页面，然后打开开发者工具，接着在test_scope函数中的第二段代码加上断点，然后刷新该页面。当执行到该断点时，V8会暂停整个执行流程，这时候我们就可以通过右边的区域面板来查看当前函数的执行状态。

你可以参考图中右侧的Scope项，然后点击展开该项，这个Local就是当前函数test_scope的作用域。在test_scope函数中定义的变量都包含到了Local中，如变量name、type，另外系统还为我们添加了另外一个隐藏变量this，V8还会默认将隐藏变量this存放到作用域中。

另外你还需要注意下，第一个test1，我并没有采用var等关键字来声明，所以test1并不会出现在test_scope函数的作用域中，而是属于this所指向的对象。（this的工作机制不是本文讨论的重点，不展开介绍。如果你感兴趣，可以在《浏览器工作原理与实践》专栏中《11 | this：从JavaScript执行上下文的视角讲清楚this》这一讲查看。）

那么另一个问题来了，我在test_scope函数使用了变量x，但是在test_scope函数的作用域中，并没有定义变量x，那么V8应该如何获取变量x?

如果在当前函数作用域中没有查找到变量，那么V8会去全局作用域中去查找，这个查找的线路就称为作用域链。

全局作用域和函数作用域类似，也是存放变量和函数的地方，但是它们还是有点不一样：**全局作用域是在V8启动过程中就创建了，且一直保存在内存中不会被销毁的，直至V8退出。 而函数作用域是在执行该函数时创建的，当函数执行结束之后，函数作用域就随之被销毁掉了。**

全局作用域中包含了很多全局变量，比如全局的this值，如果是浏览器，全局作用域中还有window、document、opener等非常多的方法和对象，如果是node环境，那么会有Global、File等内容。

V8启动之后就进入正常的消息循环状态，这时候就可以执行代码了，比如执行到上面那段脚本时，V8会先解析顶层(Top Level)代码，我们可以看到，在顶层代码中定义了变量x，这时候V8就会将变量x添加到全局作用域中。

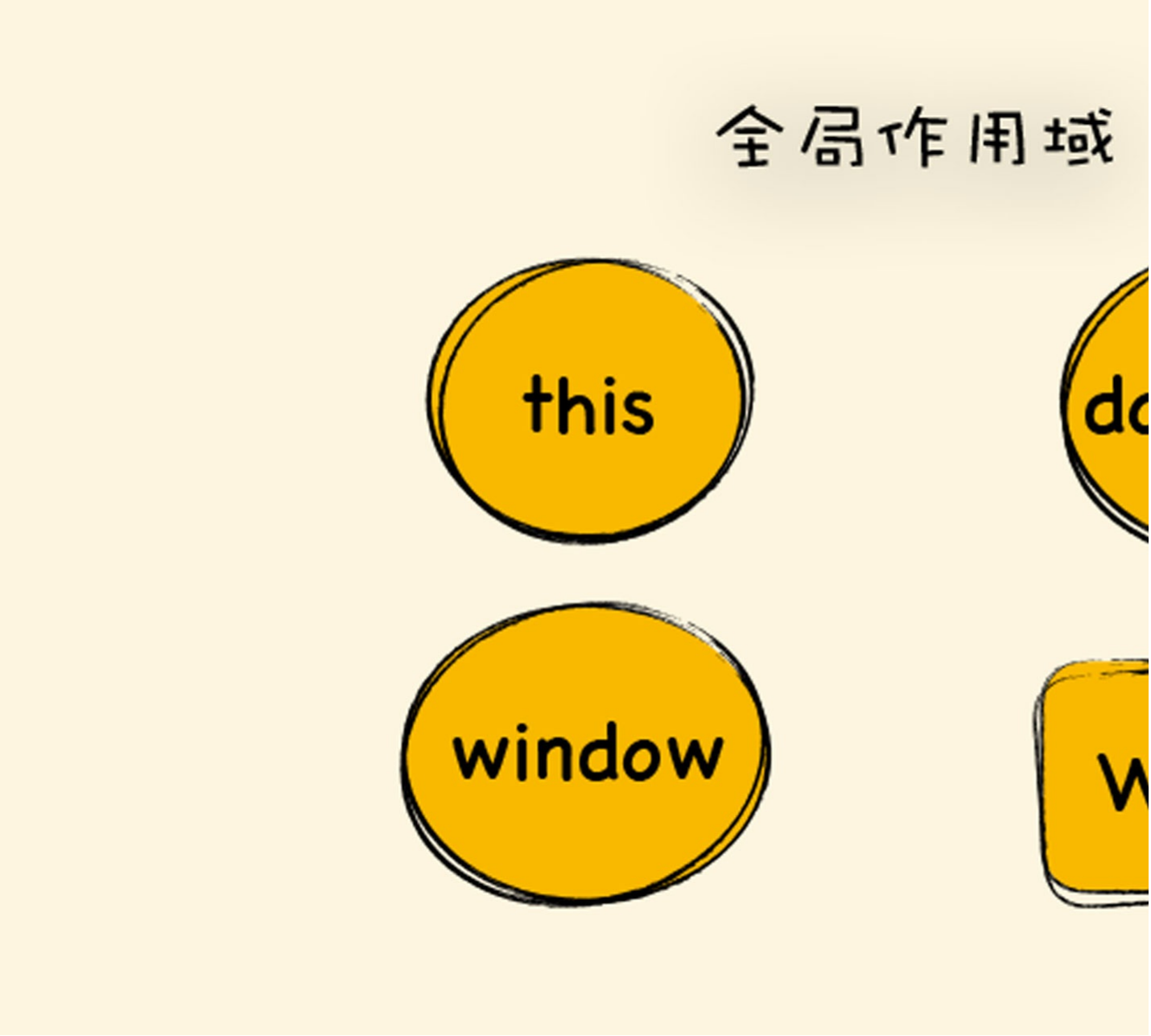
作用域链是怎么工作的？

理解了作用域和作用域链，我们再回过头来看文章开头的那道思考题：“foo函数中打印出来的变量type是bar函数中的呢，还是全局环境中的呢?”我把这段代码复制到下面：

```
var name = '极客时间'
```

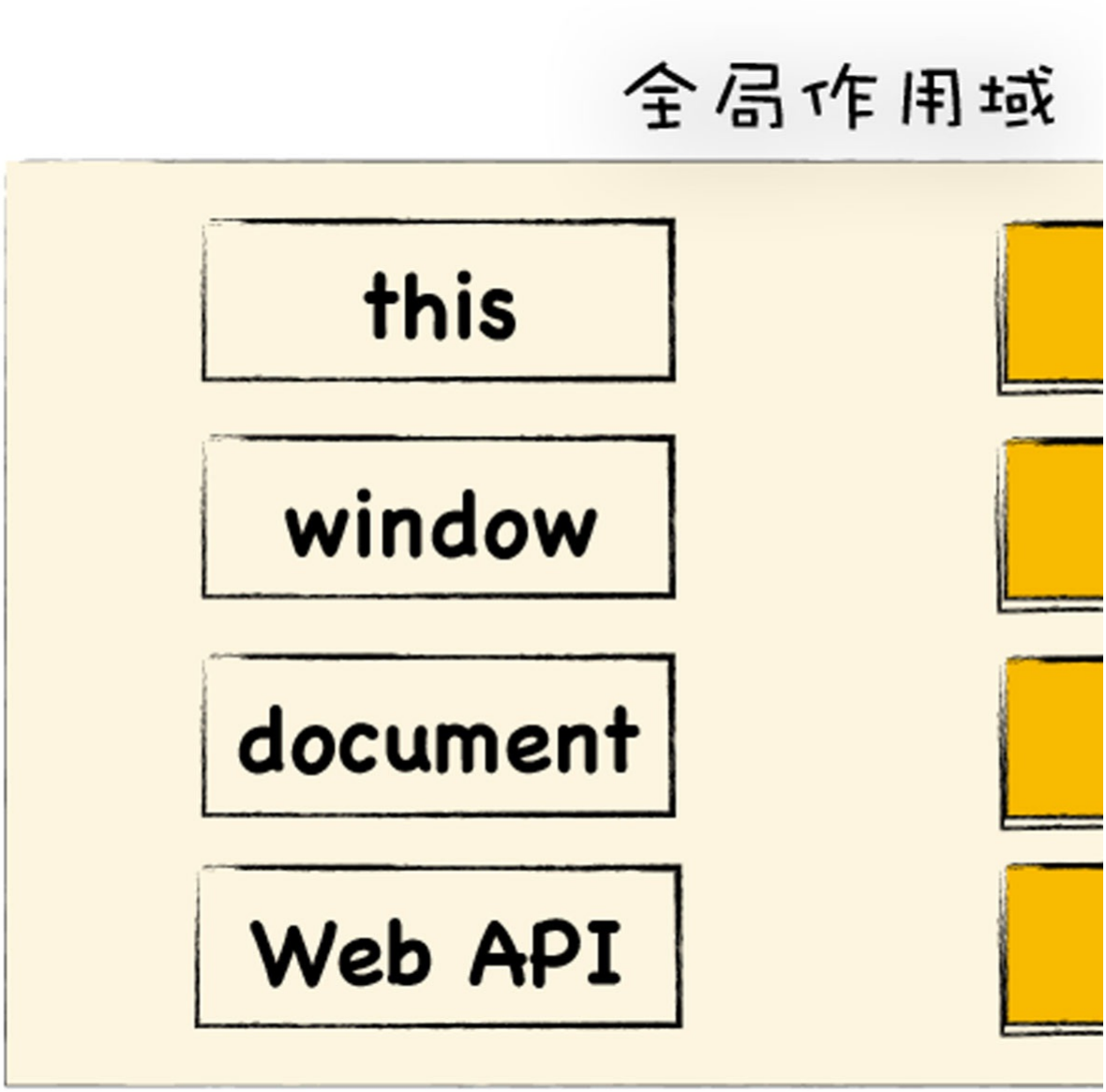
```
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

现在，我们结合V8执行这段代码的流程来具体分析下。首先当V8启动时，会创建全局作用域，全局作用域中包括了this、window等变量，还有一些全局的Web API接口，创建的作用域如下图所示：



V8启动之后，消息循环系统便开始工作了，这时候，我输入了这段代码，让其执行。

V8会先编译顶层代码，在编译过程中会将顶层定义的变量和声明的函数都添加到全局作用域中，最终的全局作用域如下图所示：

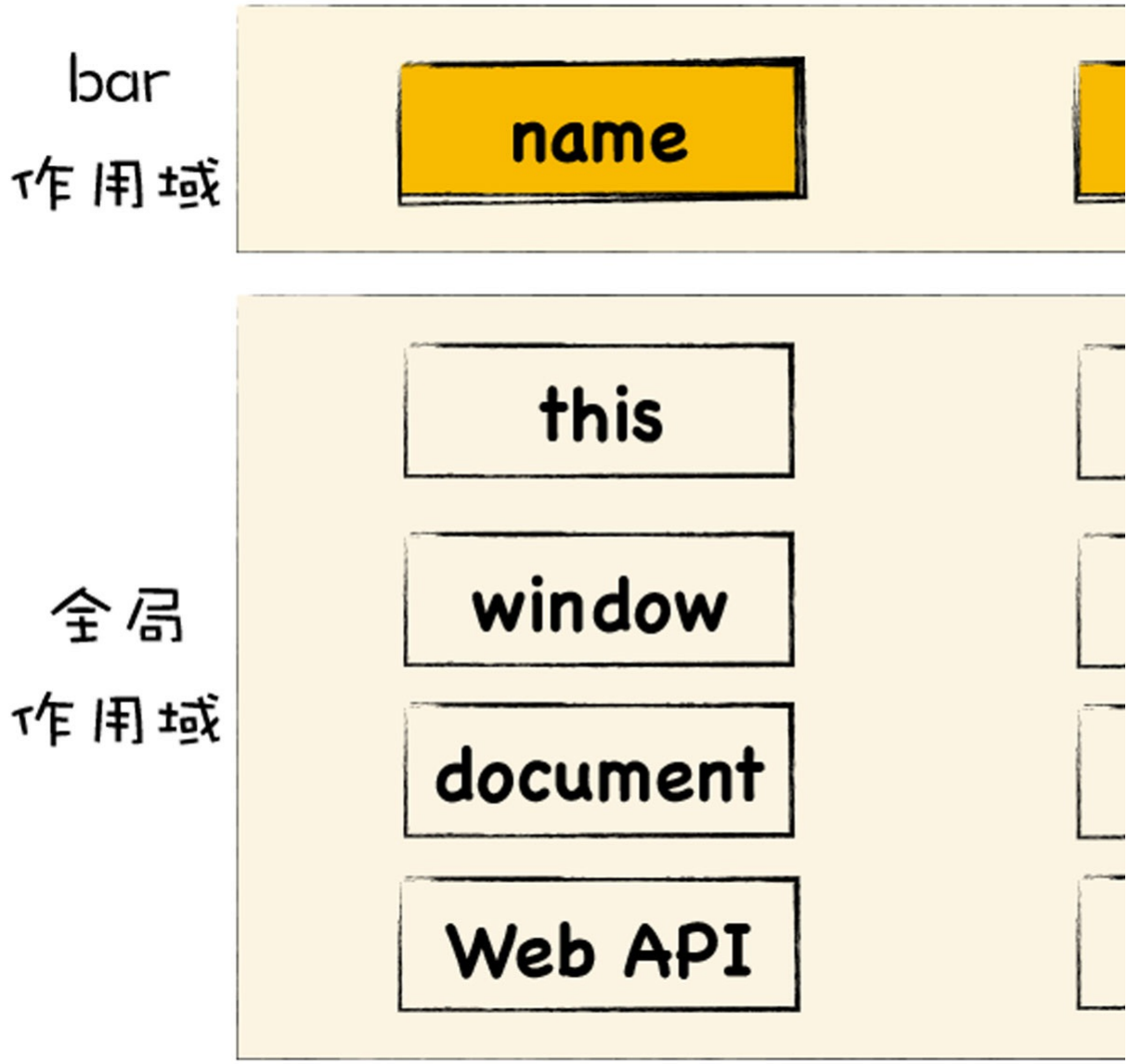


全局作用域创建完成之后，V8便进入了执行状态。前面我们介绍了变量提升，因为变量提升的原因，你可以把上面这段代码分解为如下两个部分：

```
//======解析阶段--实现变量提升=======
var name = undefined
var type = undefined
function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}
function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}



//====执行阶段========
name = '极客时间'
type = 'global'
bar()
```

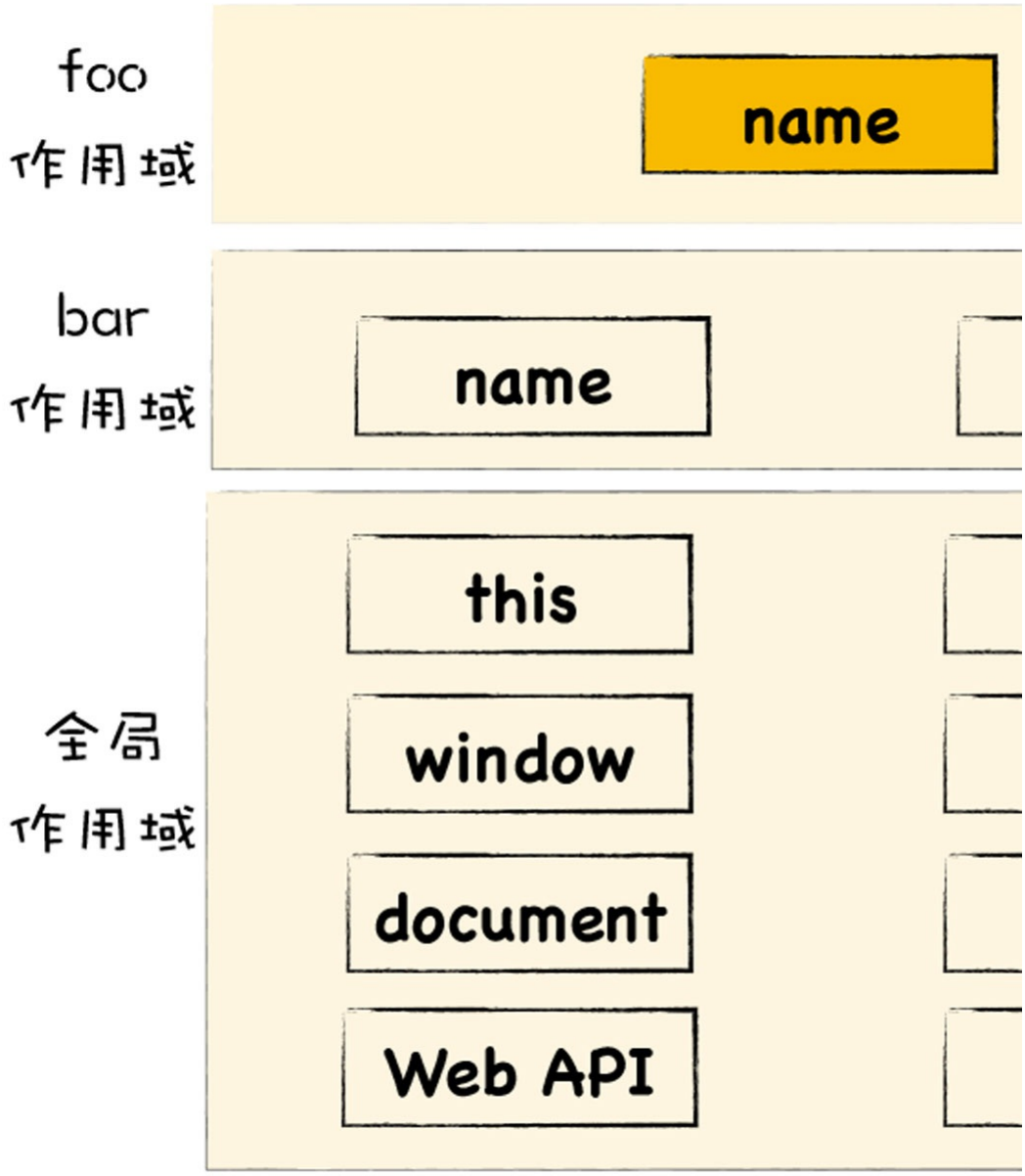
第一部分是在编译过程中完成的，此时全局作用中两个变量的值依然是undefined，然后进入执行阶段；第二部代码就是执行时的顺序，首先全局作用域中的两个变量赋值“极客时间”和“global”，然后就开始执行函数bar的调用了。

当V8执行bar函数的时候，同样需要经历两个阶段：编译和执行。在编译阶段，V8会为bar函数创建函数作用域，最终效果如下所示：



然后进入了bar函数的执行阶段。在bar函数中，只是简单地调用foo函数，因此V8又开始执行foo函数了。

同样，在编译foo函数的过程中，会创建foo函数的作用域，最终创建效果如下图所示：



好了，这时候我们就有了三个作用域了，分别是全局作用域、bar的函数作用域、foo的函数作用域。

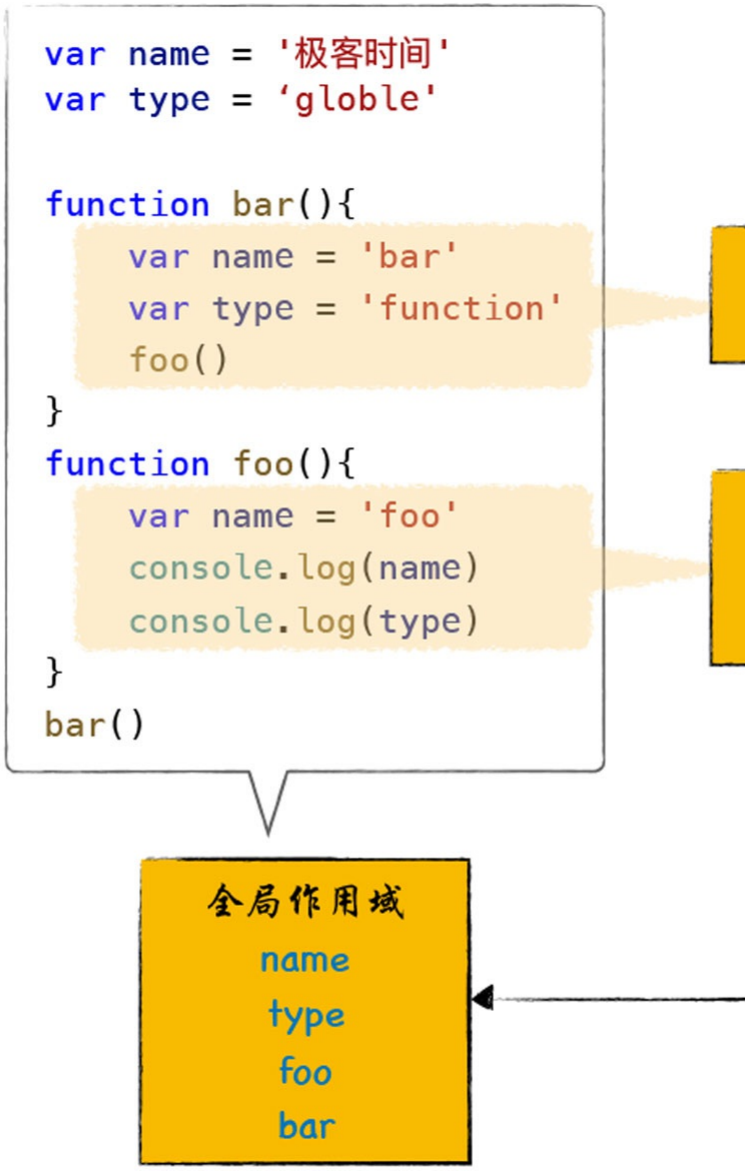
现在我们就可以将刚才提到的问题转换为作用域链的问题了：foo函数查找变量的路径到底是什么？

- 沿着foo函数作用域–>bar函数作用域–>全局作用域;
- 还是，沿着foo函数作用域—>全局作用域？

因为JavaScript是基于词法作用域的，词法作用域就是指，查找作用域的顺序是按照函数定义时的位置来决定的。bar和foo函数的外部代码都是全局代码，所以无论你是在bar函数中查找变量，还是在foo函数中查找变量，其查找顺序都是按照当前函数作用域–>全局作用域这个路径来的。

由于我们代码中的foo函数和bar函数都是在全局下面定义的，所以在foo函数中使用了type，最终打印出来的值就是全局作用域中的type。

你可以参考下面这张图：



另外，我再展开说一些。因为词法作用域是根据函数在代码中的位置来确定的，作用域是在声明函数时就确定好的了，所以我们也将词法作用域称为静态作用域。

和静态作用域相对的是动态作用域，动态作用域并不关心函数和作用域是如何声明以及在何处声明的，只关心它们从**何处调用**。换句话说，作用域链是基于调用栈的，而不是基于函数定义的位置的。（动态作用域不是本文讨论的重点，如果你感兴趣，可以参考《浏览器工作原理与实践》专栏中的《10 | 作用域链和闭包：代码中出现相同的变量，JavaScript引擎是如何选择的？》这一节。）

总结

今天，我们主要解释了一个问题，那就是在一个函数中，如果使用了一个变量，或者调用了另外一个函数，V8将会怎么去查找该变量或者函数。

为了解释清楚这个问题，我们引入了作用域的概念。作用域就是用来存放变量和函数的地方，全局作用域中存放了全局环境中声明的变量和函数，函数作用域中存放了函数中声明的变量和函数。当在某个函数中使用某个变量时，V8就会去这些作用域中查找相关变量。沿着这些作用域查找的路径，我们就称为作用域链。

要了解查找路径，我们需要明白词法作用域，词法作用域是按照代码定义时的位置决定的，而JavaScript所采用的作用域机制就是词法作用域，所以作用域链的路径就是按照词法作用域来实现的。

思考题

我将文章开头那段代码稍微调整了下，foo函数并不是在全局环境中声明的，而是在bar函数中声明的，改造后的代码如下所示：

```
var name = '极客时间'
var type = 'global'
function bar() {
    var type = 'function'
    function foo() {
        console.log(type)
    }
    foo()
}
bar()
```

那么执行这段代码之后，打印出来的内容是什么？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

在前面我们介绍了JavaScript的继承是基于原型链的，原型链将一个个原型对象串起来，从而实现对象属性的查找，今天我们要聊一个和原型链类似的话题，那就是作用域链。

作用域链就是将一个个作用域串起来，实现变量查找的路径。讨论作用域链，实际就是在讨论按照什么路径查找变量的问题。

我们知道，作用域就是存放变量和函数的地方，全局环境有全局作用域，全局作用域中存放了全局变量和全局函数。每个函数也有自己的作用域，函数作用域中存放了函数中定义的变量。

当在函数内部使用一个变量的时候，V8便会去作用域中去查找。我们通过一段在函数内部查找变量的代码来具体看一下：

```
var name = '极客时间'
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

在这段代码中，我们在全局环境中声明了变量name和type，同时还定义了bar函数和foo函数，在bar函数中又再次定义了变量name和type，在foo函数中再次定义了变量name。

函数的调用关系是：在全局环境中调用bar函数，在bar函数中调用foo函数，在foo函数中打印出来变量name和type的值。

当执行到foo函数时，首先需要打印出变量name的值，而我们在三个地方都定义了变量name，那么究竟应该使用哪个变量呢？

在foo函数中使用了变量name，那么V8就应该先使用foo函数内部定义的变量name，最终的结果确实如此，也符合我们的直觉。

接下来，foo函数继续打印变量type，但是在foo函数内部并没有定义变量type，而是在全局环境中和调用foo函数的bar函数中分别定义了变量type，那么这时候的问题来了，你觉得foo函数中打印出来的变量type是bar函数中的，还是全局环境中的呢？

什么是函数作用域和全局作用域？

要解释清楚这个问题，我们需要从作用域的工作原理讲起。

每个函数在执行时都需要查找自己的作用域，我们称为函数作用域，在执行阶段，在执行一个函数时，当该函数需要使用某个变量或者调用了某个函数时，便会优先在该函数作用域中查找相关内容。

我们再来看一段代码：

```
var x = 4
var test
function test_scope() {
    var name = 'foo'
    console.log(name)
    console.log(type)
    console.log(test)
    var type = 'function'
    test = 1
    console.log(x)
}
test_scope()
```

在上面的代码中，我们定义了一个test_scope函数，那么在V8执行test_scope函数的时候，在编译阶段会为test_scope函数创建一个作用域，在test_scope函数中定义的变量和声明的函数都会丢到该作用域中，因为我们在test_scope函数中定了三个变量，那么常见的作用域就包含有这三个变量。

你可以通过Chrome的控制台来直观感受下test_scope函数的作用域，先打开包含这段代码的页面，然后打开开发者工具，接着在test_scope函数中的第二段代码加上断点，然后刷新该页面。当执行到该断点时，V8会暂停整个执行流程，这时候我们就可以通过右边的区域面板来查看当前函数的执行状态。

你可以参考图中右侧的Scope项，然后点击展开该项，这个Local就是当前函数test_scope的作用域。在test_scope函数中定义的变量都包含到了Local中，如变量name、type，另外系统还为我们添加了另外一个隐藏变量this，V8还会默认将隐藏变量this存放到作用域中。

另外你还需要注意下，第一个test1，我并没有采用var等关键字来声明，所以test1并不会出现在test_scope函数的作用域中，而是属于this所指向的对象。（this的工作机制不是本文讨论的重点，不展开介绍。如果你感兴趣，可以在《浏览器工作原理与实践》专栏中《11 | this：从JavaScript执行上下文的视角讲清楚this》这一讲查看。）

那么另一个问题来了，我在test_scope函数使用了变量x，但是在test_scope函数的作用域中，并没有定义变量x，那么V8应该如何获取变量x?

如果在当前函数作用域中没有查找到变量，那么V8会去全局作用域中去查找，这个查找的线路就称为作用域链。

全局作用域和函数作用域类似，也是存放变量和函数的地方，但是它们还是有点不一样：**全局作用域是在V8启动过程中就创建了，且一直保存在内存中不会被销毁的，直至V8退出。 而函数作用域是在执行该函数时创建的，当函数执行结束之后，函数作用域就随之被销毁掉了。**

全局作用域中包含了很多全局变量，比如全局的this值，如果是浏览器，全局作用域中还有window、document、opener等非常多的方法和对象，如果是node环境，那么会有Global、File等内容。

V8启动之后就进入正常的消息循环状态，这时候就可以执行代码了，比如执行到上面那段脚本时，V8会先解析顶层(Top Level)代码，我们可以看到，在顶层代码中定义了变量x，这时候V8就会将变量x添加到全局作用域中。

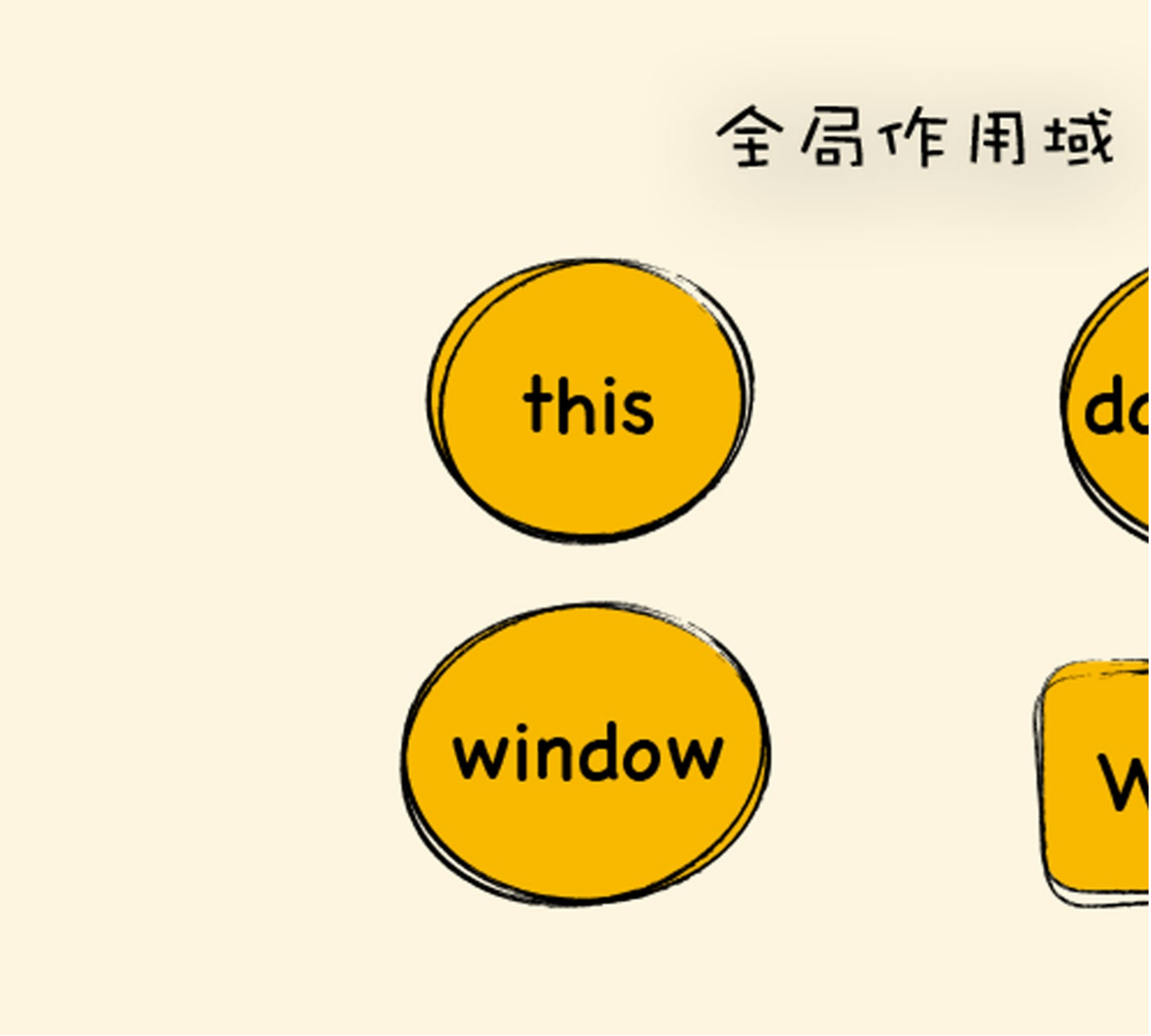
作用域链是怎么工作的？

理解了作用域和作用域链，我们再回过头来看文章开头的那道思考题：“foo函数中打印出来的变量type是bar函数中的呢，还是全局环境中的呢?”我把这段代码复制到下面：

```
var name = '极客时间'
```

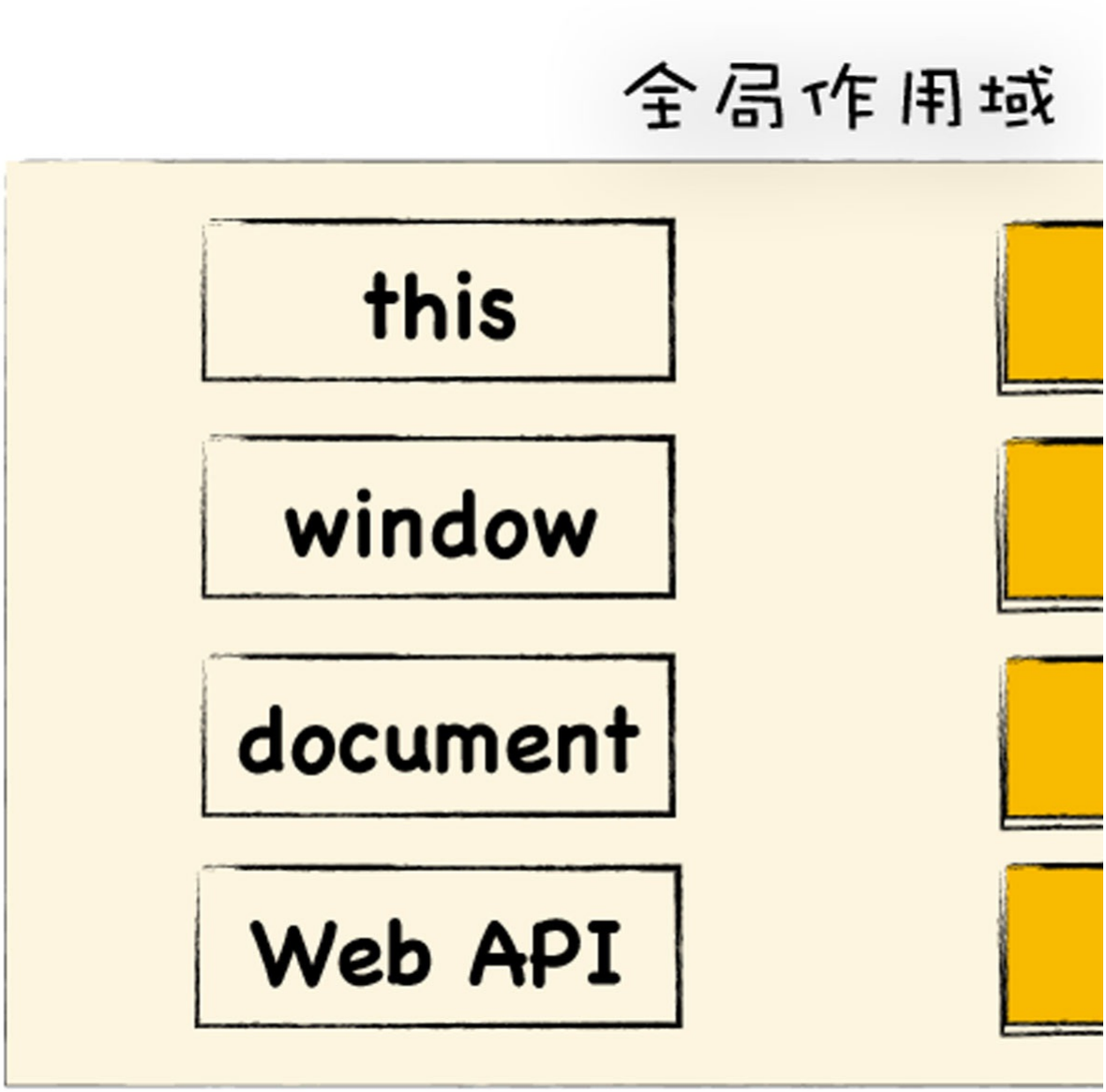
```
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

现在，我们结合V8执行这段代码的流程来具体分析下。首先当V8启动时，会创建全局作用域，全局作用域中包括了this、window等变量，还有一些全局的Web API接口，创建的作用域如下图所示：



V8启动之后，消息循环系统便开始工作了，这时候，我输入了这段代码，让其执行。

V8会先编译顶层代码，在编译过程中会将顶层定义的变量和声明的函数都添加到全局作用域中，最终的全局作用域如下图所示：

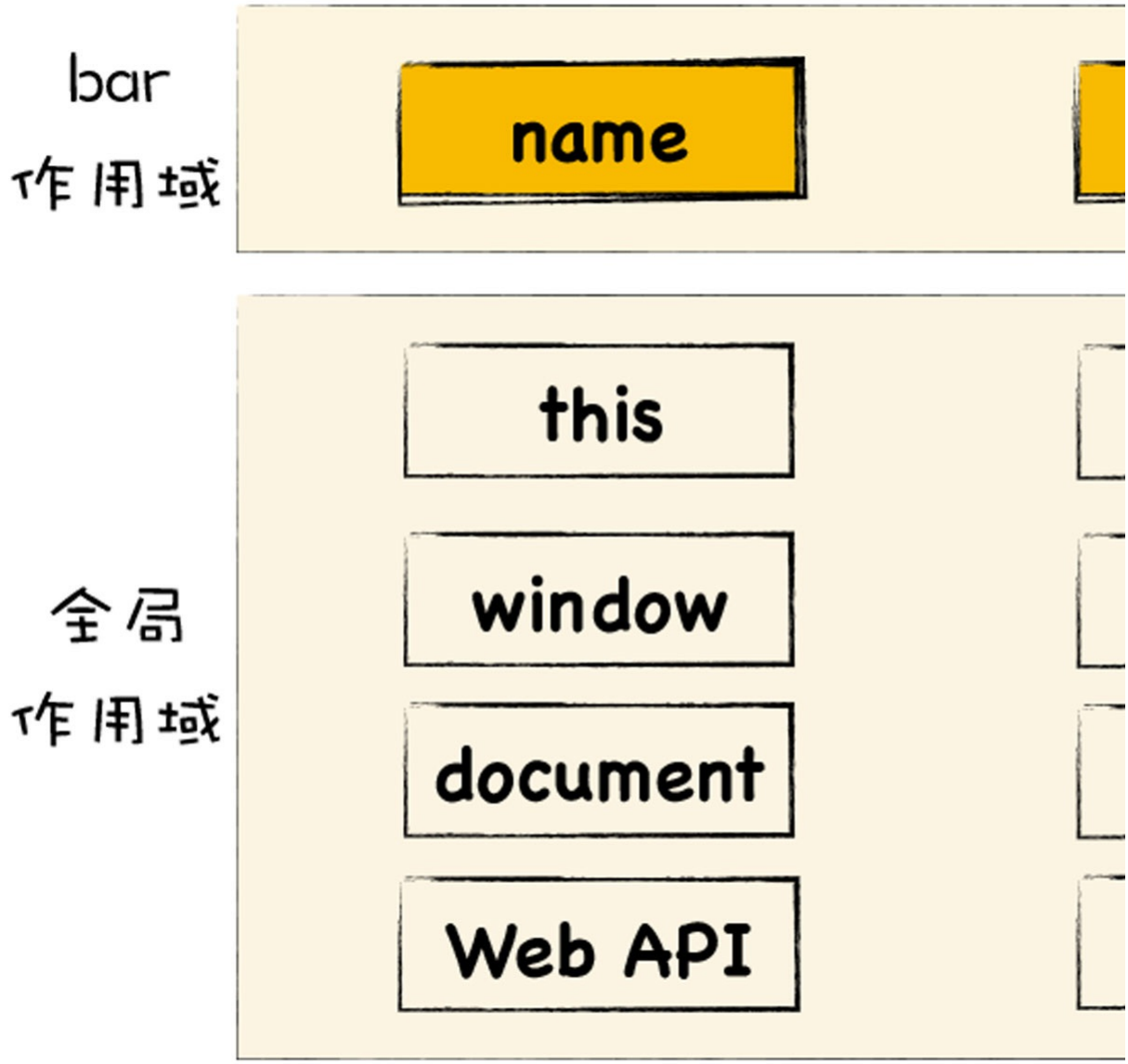


全局作用域创建完成之后，V8便进入了执行状态。前面我们介绍了变量提升，因为变量提升的原因，你可以把上面这段代码分解为如下两个部分：

```
//======解析阶段--实现变量提升=======
var name = undefined
var type = undefined
function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}
function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}



//====执行阶段========
name = '极客时间'
type = 'global'
bar()
```

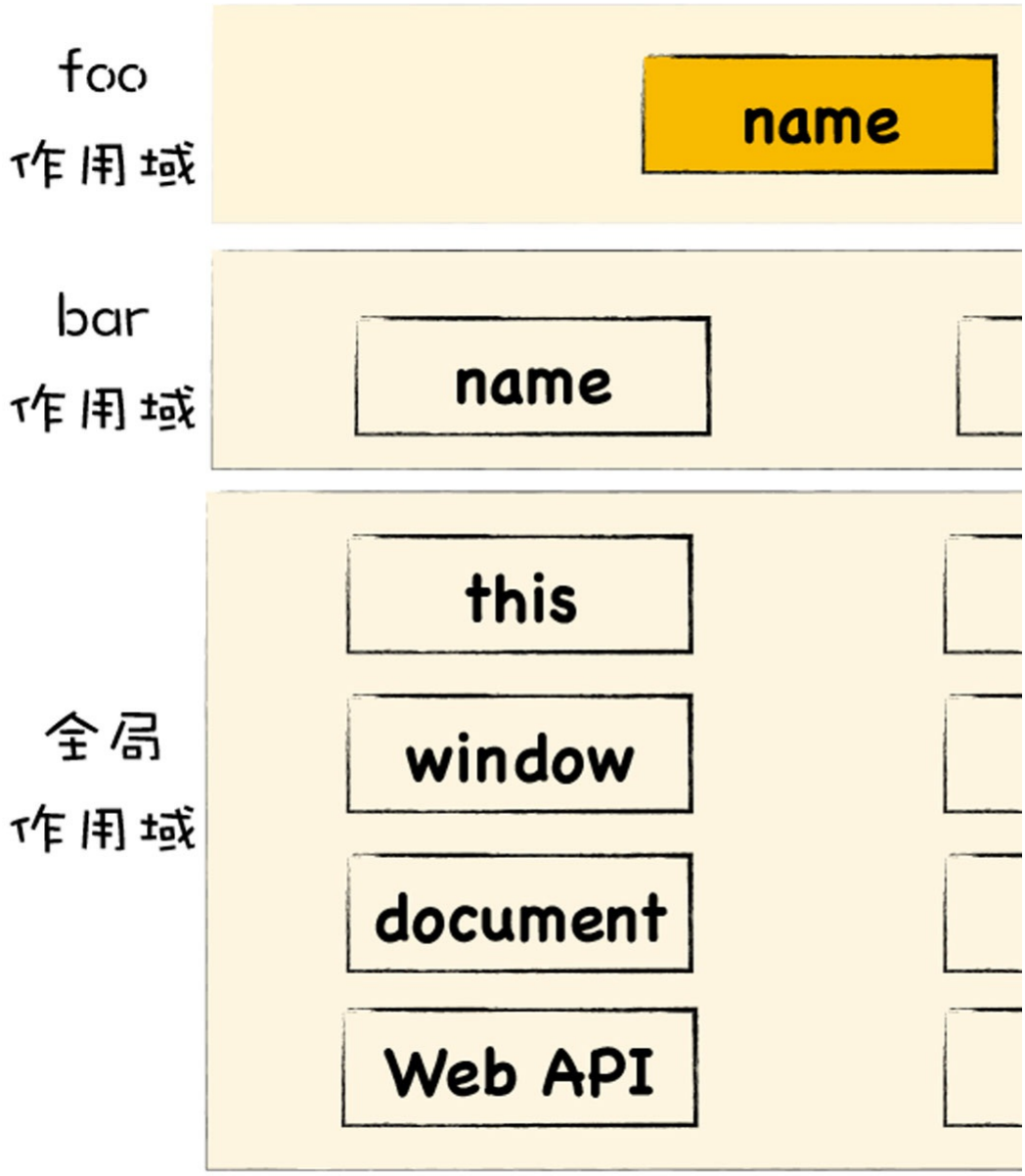
第一部分是在编译过程中完成的，此时全局作用中两个变量的值依然是undefined，然后进入执行阶段；第二部代码就是执行时的顺序，首先全局作用域中的两个变量赋值“极客时间”和“global”，然后就开始执行函数bar的调用了。

当V8执行bar函数的时候，同样需要经历两个阶段：编译和执行。在编译阶段，V8会为bar函数创建函数作用域，最终效果如下所示：



然后进入了bar函数的执行阶段。在bar函数中，只是简单地调用foo函数，因此V8又开始执行foo函数了。

同样，在编译foo函数的过程中，会创建foo函数的作用域，最终创建效果如下图所示：



好了，这时候我们就有了三个作用域了，分别是全局作用域、bar的函数作用域、foo的函数作用域。

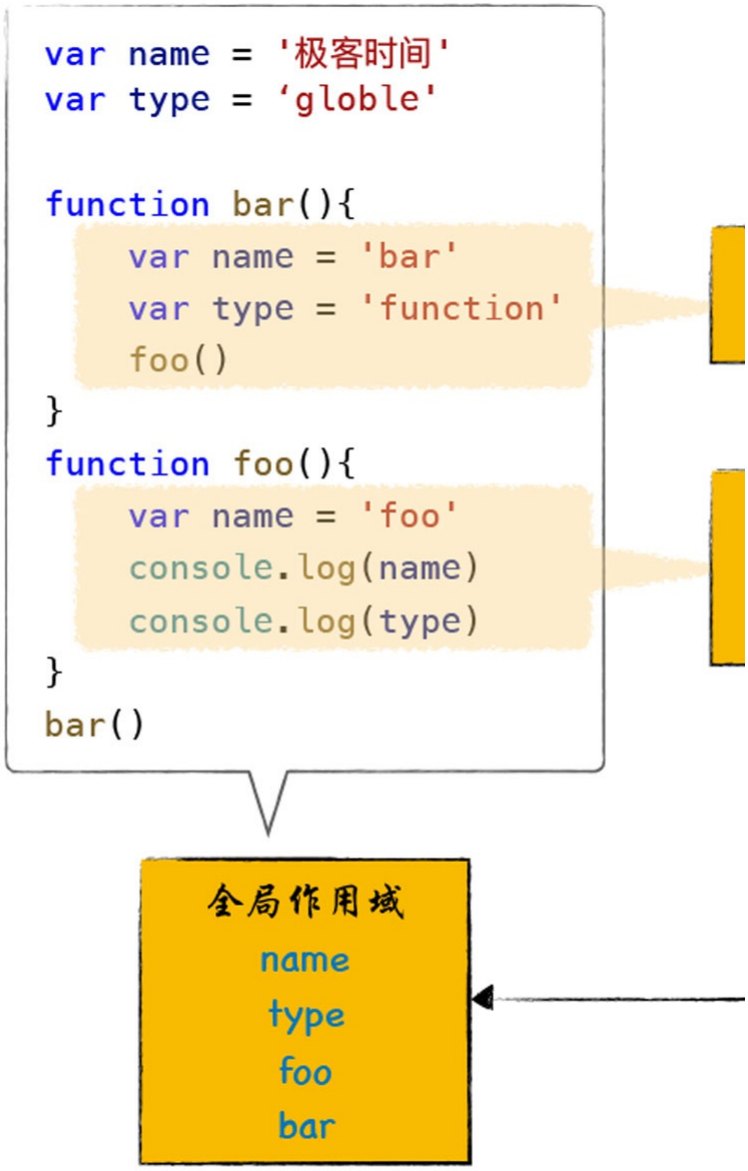
现在我们就可以将刚才提到的问题转换为作用域链的问题了：foo函数查找变量的路径到底是什么？

- 沿着foo函数作用域–>bar函数作用域–>全局作用域;
- 还是，沿着foo函数作用域—>全局作用域？

因为JavaScript是基于词法作用域的，词法作用域就是指，查找作用域的顺序是按照函数定义时的位置来决定的。bar和foo函数的外部代码都是全局代码，所以无论你是在bar函数中查找变量，还是在foo函数中查找变量，其查找顺序都是按照当前函数作用域–>全局作用域这个路径来的。

由于我们代码中的foo函数和bar函数都是在全局下面定义的，所以在foo函数中使用了type，最终打印出来的值就是全局作用域中的type。

你可以参考下面这张图：



另外，我再展开说一些。因为词法作用域是根据函数在代码中的位置来确定的，作用域是在声明函数时就确定好的了，所以我们也将词法作用域称为静态作用域。

和静态作用域相对的是动态作用域，动态作用域并不关心函数和作用域是如何声明以及在何处声明的，只关心它们从**何处调用**。换句话说，作用域链是基于调用栈的，而不是基于函数定义的位置的。（动态作用域不是本文讨论的重点，如果你感兴趣，可以参考《浏览器工作原理与实践》专栏中的《10 | 作用域链和闭包：代码中出现相同的变量，JavaScript引擎是如何选择的？》这一节。）

总结

今天，我们主要解释了一个问题，那就是在一个函数中，如果使用了一个变量，或者调用了另外一个函数，V8将会怎么去查找该变量或者函数。

为了解释清楚这个问题，我们引入了作用域的概念。作用域就是用来存放变量和函数的地方，全局作用域中存放了全局环境中声明的变量和函数，函数作用域中存放了函数中声明的变量和函数。当在某个函数中使用某个变量时，V8就会去这些作用域中查找相关变量。沿着这些作用域查找的路径，我们就称为作用域链。

要了解查找路径，我们需要明白词法作用域，词法作用域是按照代码定义时的位置决定的，而JavaScript所采用的作用域机制就是词法作用域，所以作用域链的路径就是按照词法作用域来实现的。

思考题

我将文章开头那段代码稍微调整了下，foo函数并不是在全局环境中声明的，而是在bar函数中声明的，改造后的代码如下所示：

```
var name = '极客时间'
var type = 'global'
function bar() {
    var type = 'function'
    function foo() {
        console.log(type)
    }
    foo()
}
bar()
```

那么执行这段代码之后，打印出来的内容是什么？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

在前面我们介绍了JavaScript的继承是基于原型链的，原型链将一个个原型对象串起来，从而实现对象属性的查找，今天我们要聊一个和原型链类似的话题，那就是作用域链。

作用域链就是将一个个作用域串起来，实现变量查找的路径。讨论作用域链，实际就是在讨论按照什么路径查找变量的问题。

我们知道，作用域就是存放变量和函数的地方，全局环境有全局作用域，全局作用域中存放了全局变量和全局函数。每个函数也有自己的作用域，函数作用域中存放了函数中定义的变量。

当在函数内部使用一个变量的时候，V8便会去作用域中去查找。我们通过一段在函数内部查找变量的代码来具体看一下：

```
var name = '极客时间'
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

在这段代码中，我们在全局环境中声明了变量name和type，同时还定义了bar函数和foo函数，在bar函数中又再次定义了变量name和type，在foo函数中再次定义了变量name。

函数的调用关系是：在全局环境中调用bar函数，在bar函数中调用foo函数，在foo函数中打印出来变量name和type的值。

当执行到foo函数时，首先需要打印出变量name的值，而我们在三个地方都定义了变量name，那么究竟应该使用哪个变量呢？

在foo函数中使用了变量name，那么V8就应该先使用foo函数内部定义的变量name，最终的结果确实如此，也符合我们的直觉。

接下来，foo函数继续打印变量type，但是在foo函数内部并没有定义变量type，而是在全局环境中和调用foo函数的bar函数中分别定义了变量type，那么这时候的问题来了，你觉得foo函数中打印出来的变量type是bar函数中的，还是全局环境中的呢？

什么是函数作用域和全局作用域？

要解释清楚这个问题，我们需要从作用域的工作原理讲起。

每个函数在执行时都需要查找自己的作用域，我们称为函数作用域，在执行阶段，在执行一个函数时，当该函数需要使用某个变量或者调用了某个函数时，便会优先在该函数作用域中查找相关内容。

我们再来看一段代码：

```
var x = 4
var test
function test_scope() {
    var name = 'foo'
    console.log(name)
    console.log(type)
    console.log(test)
    var type = 'function'
    test = 1
    console.log(x)
}
test_scope()
```

在上面的代码中，我们定义了一个test_scope函数，那么在V8执行test_scope函数的时候，在编译阶段会为test_scope函数创建一个作用域，在test_scope函数中定义的变量和声明的函数都会丢到该作用域中，因为我们在test_scope函数中定了三个变量，那么常见的作用域就包含有这三个变量。

你可以通过Chrome的控制台来直观感受下test_scope函数的作用域，先打开包含这段代码的页面，然后打开开发者工具，接着在test_scope函数中的第二段代码加上断点，然后刷新该页面。当执行到该断点时，V8会暂停整个执行流程，这时候我们就可以通过右边的区域面板来查看当前函数的执行状态。

你可以参考图中右侧的Scope项，然后点击展开该项，这个Local就是当前函数test_scope的作用域。在test_scope函数中定义的变量都包含到了Local中，如变量name、type，另外系统还为我们添加了另外一个隐藏变量this，V8还会默认将隐藏变量this存放到作用域中。

另外你还需要注意下，第一个test1，我并没有采用var等关键字来声明，所以test1并不会出现在test_scope函数的作用域中，而是属于this所指向的对象。（this的工作机制不是本文讨论的重点，不展开介绍。如果你感兴趣，可以在《浏览器工作原理与实践》专栏中《11 | this：从JavaScript执行上下文的视角讲清楚this》这一讲查看。）

那么另一个问题来了，我在test_scope函数使用了变量x，但是在test_scope函数的作用域中，并没有定义变量x，那么V8应该如何获取变量x?

如果在当前函数作用域中没有查找到变量，那么V8会去全局作用域中去查找，这个查找的线路就称为作用域链。

全局作用域和函数作用域类似，也是存放变量和函数的地方，但是它们还是有点不一样：**全局作用域是在V8启动过程中就创建了，且一直保存在内存中不会被销毁的，直至V8退出。 而函数作用域是在执行该函数时创建的，当函数执行结束之后，函数作用域就随之被销毁掉了。**

全局作用域中包含了很多全局变量，比如全局的this值，如果是浏览器，全局作用域中还有window、document、opener等非常多的方法和对象，如果是node环境，那么会有Global、File等内容。

V8启动之后就进入正常的消息循环状态，这时候就可以执行代码了，比如执行到上面那段脚本时，V8会先解析顶层(Top Level)代码，我们可以看到，在顶层代码中定义了变量x，这时候V8就会将变量x添加到全局作用域中。

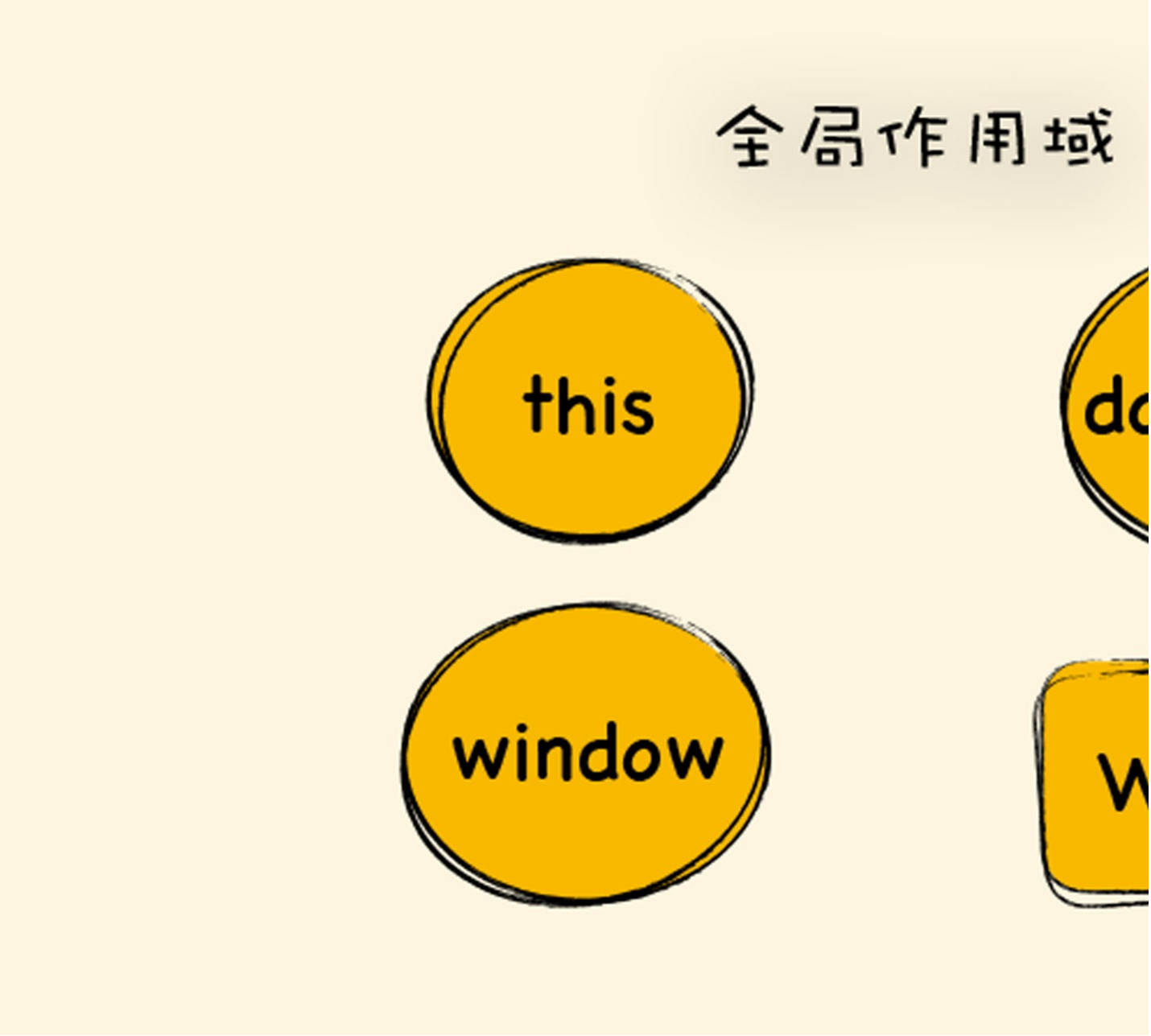
作用域链是怎么工作的？

理解了作用域和作用域链，我们再回过头来看文章开头的那道思考题：“foo函数中打印出来的变量type是bar函数中的呢，还是全局环境中的呢?”我把这段代码复制到下面：

```
var name = '极客时间'
```

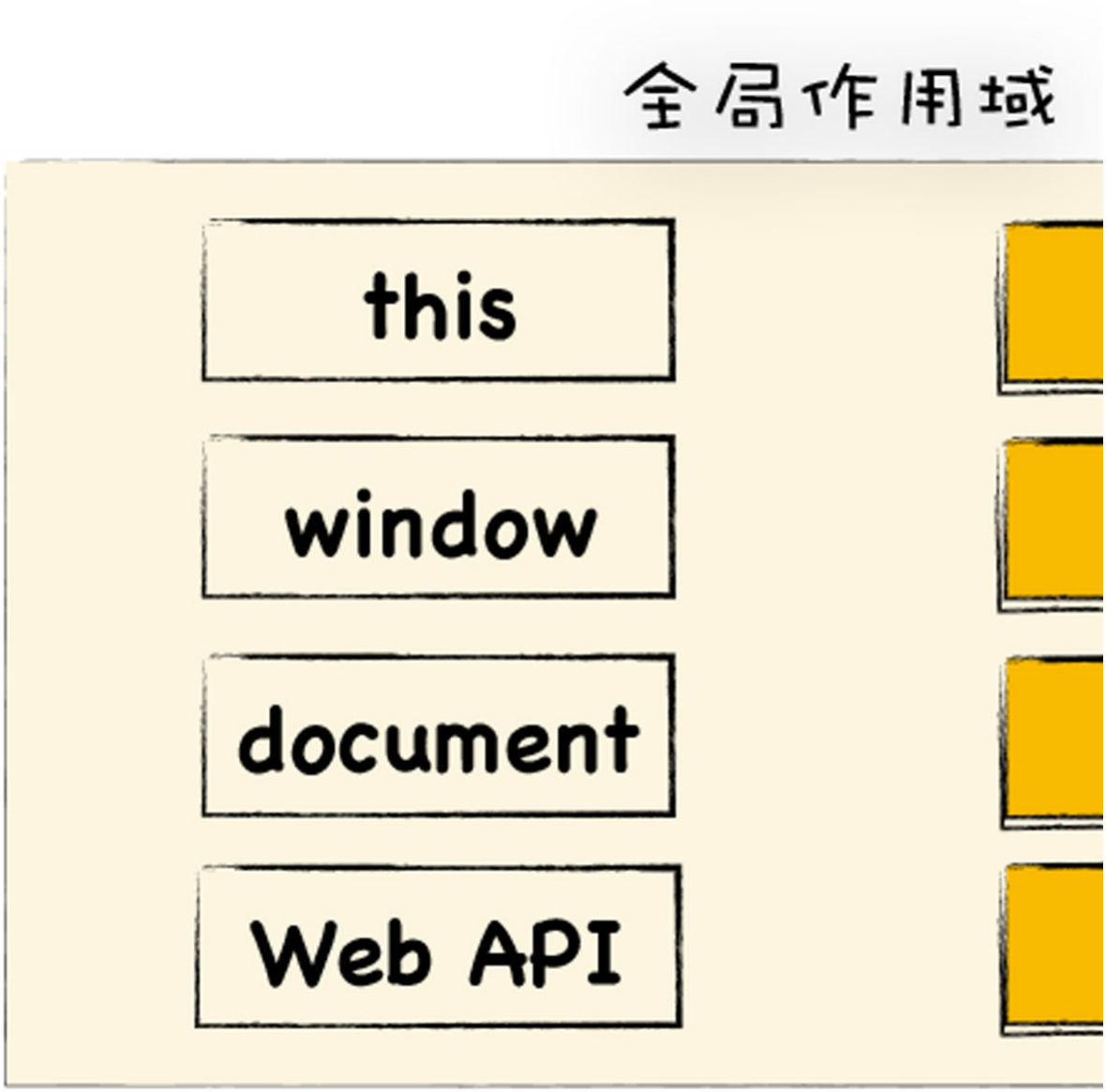
```
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

现在，我们结合V8执行这段代码的流程来具体分析下。首先当V8启动时，会创建全局作用域，全局作用域中包括了this、window等变量，还有一些全局的Web API接口，创建的作用域如下图所示：



V8启动之后，消息循环系统便开始工作了，这时候，我输入了这段代码，让其执行。

V8会先编译顶层代码，在编译过程中会将顶层定义的变量和声明的函数都添加到全局作用域中，最终的全局作用域如下图所示：

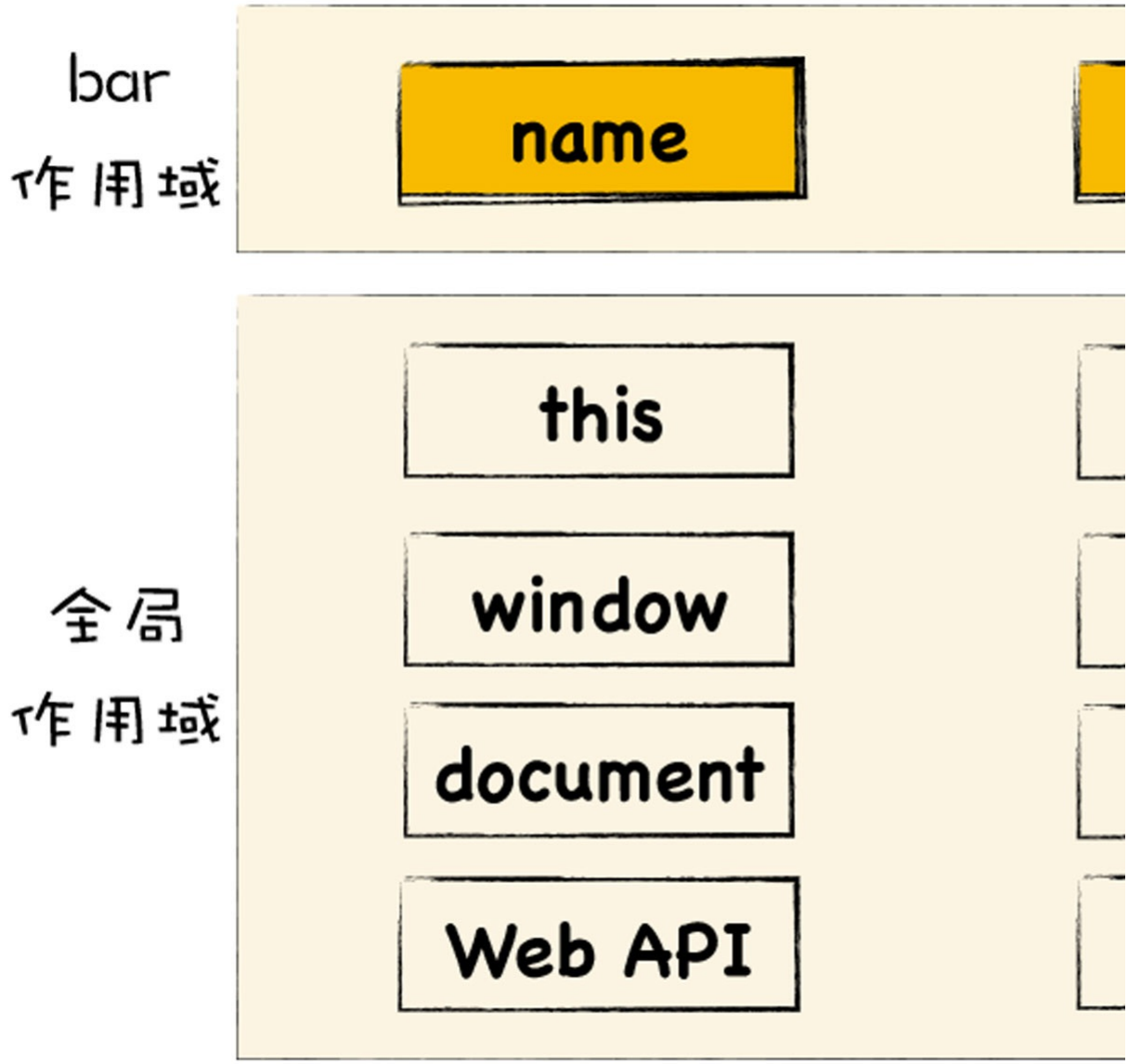


全局作用域创建完成之后，V8便进入了执行状态。前面我们介绍了变量提升，因为变量提升的原因，你可以把上面这段代码分解为如下两个部分：

```
//======解析阶段--实现变量提升=======
var name = undefined
var type = undefined
function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}
function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}



//====执行阶段========
name = '极客时间'
type = 'global'
bar()
```

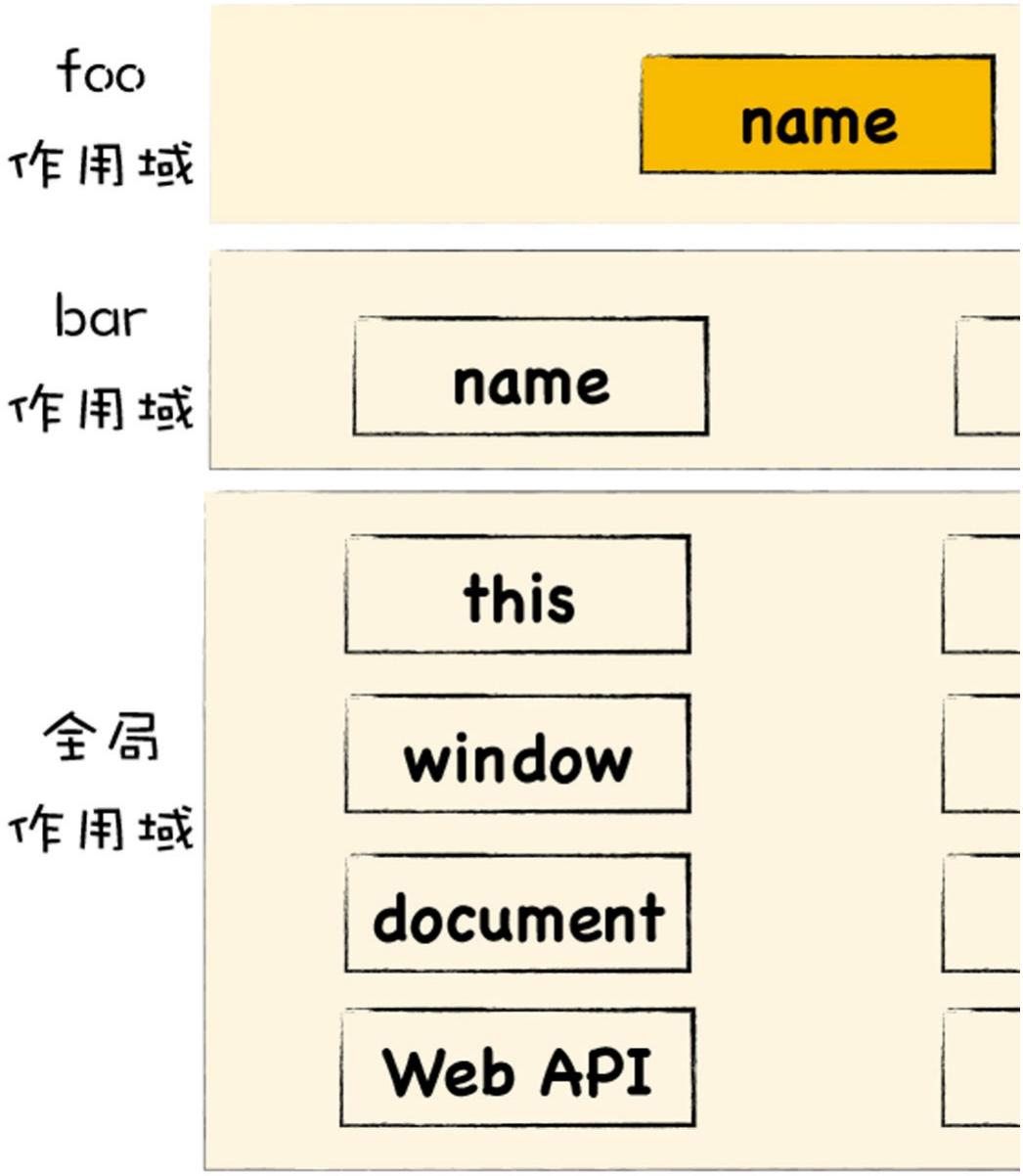
第一部分是在编译过程中完成的，此时全局作用中两个变量的值依然是undefined，然后进入执行阶段；第二部代码就是执行时的顺序，首先全局作用域中的两个变量赋值“极客时间”和“global”，然后就开始执行函数bar的调用了。

当V8执行bar函数的时候，同样需要经历两个阶段：编译和执行。在编译阶段，V8会为bar函数创建函数作用域，最终效果如下所示：



然后进入了bar函数的执行阶段。在bar函数中，只是简单地调用foo函数，因此V8又开始执行foo函数了。

同样，在编译foo函数的过程中，会创建foo函数的作用域，最终创建效果如下图所示：



好了，这时候我们就有了三个作用域了，分别是全局作用域、bar的函数作用域、foo的函数作用域。

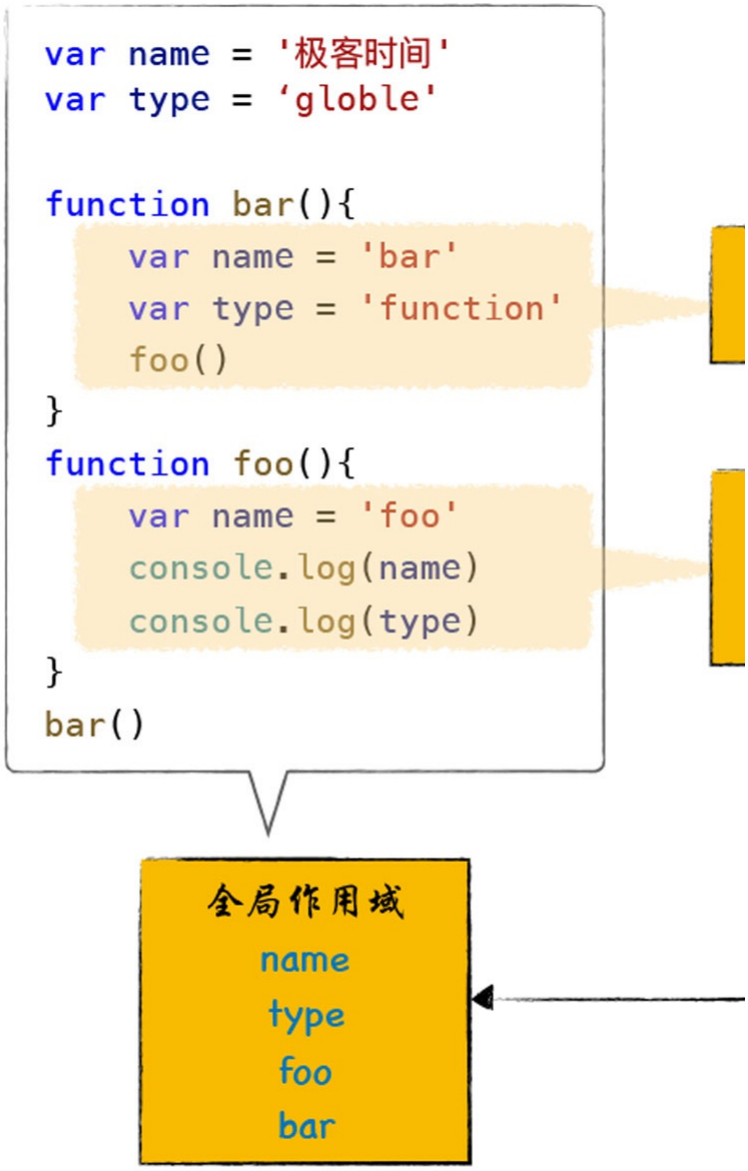
现在我们就可以将刚才提到的问题转换为作用域链的问题了：foo函数查找变量的路径到底是什么？

- 沿着foo函数作用域–>bar函数作用域–>全局作用域;
- 还是，沿着foo函数作用域—>全局作用域？

因为JavaScript是基于词法作用域的，词法作用域就是指，查找作用域的顺序是按照函数定义时的位置来决定的。bar和foo函数的外部代码都是全局代码，所以无论你是在bar函数中查找变量，还是在foo函数中查找变量，其查找顺序都是按照当前函数作用域–>全局作用域这个路径来的。

由于我们代码中的foo函数和bar函数都是在全局下面定义的，所以在foo函数中使用了type，最终打印出来的值就是全局作用域中的type。

你可以参考下面这张图：



另外，我再展开说一些。因为词法作用域是根据函数在代码中的位置来确定的，作用域是在声明函数时就确定好的了，所以我们也将词法作用域称为静态作用域。

和静态作用域相对的是动态作用域，动态作用域并不关心函数和作用域是如何声明以及在何处声明的，只关心它们从**何处调用**。换句话说，作用域链是基于调用栈的，而不是基于函数定义的位置的。（动态作用域不是本文讨论的重点，如果你感兴趣，可以参考《浏览器工作原理与实践》专栏中的《10 | 作用域链和闭包：代码中出现相同的变量，JavaScript引擎是如何选择的？》这一节。）

总结

今天，我们主要解释了一个问题，那就是在一个函数中，如果使用了一个变量，或者调用了另外一个函数，V8将会怎么去查找该变量或者函数。

为了解释清楚这个问题，我们引入了作用域的概念。作用域就是用来存放变量和函数的地方，全局作用域中存放了全局环境中声明的变量和函数，函数作用域中存放了函数中声明的变量和函数。当在某个函数中使用某个变量时，V8就会去这些作用域中查找相关变量。沿着这些作用域查找的路径，我们就称为作用域链。

要了解查找路径，我们需要明白词法作用域，词法作用域是按照代码定义时的位置决定的，而JavaScript所采用的作用域机制就是词法作用域，所以作用域链的路径就是按照词法作用域来实现的。

思考题

我将文章开头那段代码稍微调整了下，foo函数并不是在全局环境中声明的，而是在bar函数中声明的，改造后的代码如下所示：

```
var name = '极客时间'
var type = 'global'
function bar() {
    var type = 'function'
    function foo() {
        console.log(type)
    }
    foo()
}
bar()
```

那么执行这段代码之后，打印出来的内容是什么？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

在前面我们介绍了JavaScript的继承是基于原型链的，原型链将一个个原型对象串起来，从而实现对象属性的查找，今天我们要聊一个和原型链类似的话题，那就是作用域链。

作用域链就是将一个个作用域串起来，实现变量查找的路径。讨论作用域链，实际就是在讨论按照什么路径查找变量的问题。

我们知道，作用域就是存放变量和函数的地方，全局环境有全局作用域，全局作用域中存放了全局变量和全局函数。每个函数也有自己的作用域，函数作用域中存放了函数中定义的变量。

当在函数内部使用一个变量的时候，V8便会去作用域中去查找。我们通过一段在函数内部查找变量的代码来具体看一下：

```
var name = '极客时间'
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

在这段代码中，我们在全局环境中声明了变量name和type，同时还定义了bar函数和foo函数，在bar函数中又再次定义了变量name和type，在foo函数中再次定义了变量name。

函数的调用关系是：在全局环境中调用bar函数，在bar函数中调用foo函数，在foo函数中打印出来变量name和type的值。

当执行到foo函数时，首先需要打印出变量name的值，而我们在三个地方都定义了变量name，那么究竟应该使用哪个变量呢？

在foo函数中使用了变量name，那么V8就应该先使用foo函数内部定义的变量name，最终的结果确实如此，也符合我们的直觉。

接下来，foo函数继续打印变量type，但是在foo函数内部并没有定义变量type，而是在全局环境中和调用foo函数的bar函数中分别定义了变量type，那么这时候的问题来了，你觉得foo函数中打印出来的变量type是bar函数中的，还是全局环境中的呢？

什么是函数作用域和全局作用域？

要解释清楚这个问题，我们需要从作用域的工作原理讲起。

每个函数在执行时都需要查找自己的作用域，我们称为函数作用域，在执行阶段，在执行一个函数时，当该函数需要使用某个变量或者调用了某个函数时，便会优先在该函数作用域中查找相关内容。

我们再来看一段代码：

```
var x = 4
var test
function test_scope() {
    var name = 'foo'
    console.log(name)
    console.log(type)
    console.log(test)
    var type = 'function'
    test = 1
    console.log(x)
}
test_scope()
```

在上面的代码中，我们定义了一个test_scope函数，那么在V8执行test_scope函数的时候，在编译阶段会为test_scope函数创建一个作用域，在test_scope函数中定义的变量和声明的函数都会丢到该作用域中，因为我们在test_scope函数中定了三个变量，那么常见的作用域就包含有这三个变量。

你可以通过Chrome的控制台来直观感受下test_scope函数的作用域，先打开包含这段代码的页面，然后打开开发者工具，接着在test_scope函数中的第二段代码加上断点，然后刷新该页面。当执行到该断点时，V8会暂停整个执行流程，这时候我们就可以通过右边的区域面板来查看当前函数的执行状态。

你可以参考图中右侧的Scope项，然后点击展开该项，这个Local就是当前函数test_scope的作用域。在test_scope函数中定义的变量都包含到了Local中，如变量name、type，另外系统还为我们添加了另外一个隐藏变量this，V8还会默认将隐藏变量this存放到作用域中。

另外你还需要注意下，第一个test1，我并没有采用var等关键字来声明，所以test1并不会出现在test_scope函数的作用域中，而是属于this所指向的对象。（this的工作机制不是本文讨论的重点，不展开介绍。如果你感兴趣，可以在《浏览器工作原理与实践》专栏中《11 | this：从JavaScript执行上下文的视角讲清楚this》这一讲查看。）

那么另一个问题来了，我在test_scope函数使用了变量x，但是在test_scope函数的作用域中，并没有定义变量x，那么V8应该如何获取变量x?

如果在当前函数作用域中没有查找到变量，那么V8会去全局作用域中去查找，这个查找的线路就称为作用域链。

全局作用域和函数作用域类似，也是存放变量和函数的地方，但是它们还是有点不一样：**全局作用域是在V8启动过程中就创建了，且一直保存在内存中不会被销毁的，直至V8退出。 而函数作用域是在执行该函数时创建的，当函数执行结束之后，函数作用域就随之被销毁掉了。**

全局作用域中包含了很多全局变量，比如全局的this值，如果是浏览器，全局作用域中还有window、document、opener等非常多的方法和对象，如果是node环境，那么会有Global、File等内容。

V8启动之后就进入正常的消息循环状态，这时候就可以执行代码了，比如执行到上面那段脚本时，V8会先解析顶层(Top Level)代码，我们可以看到，在顶层代码中定义了变量x，这时候V8就会将变量x添加到全局作用域中。

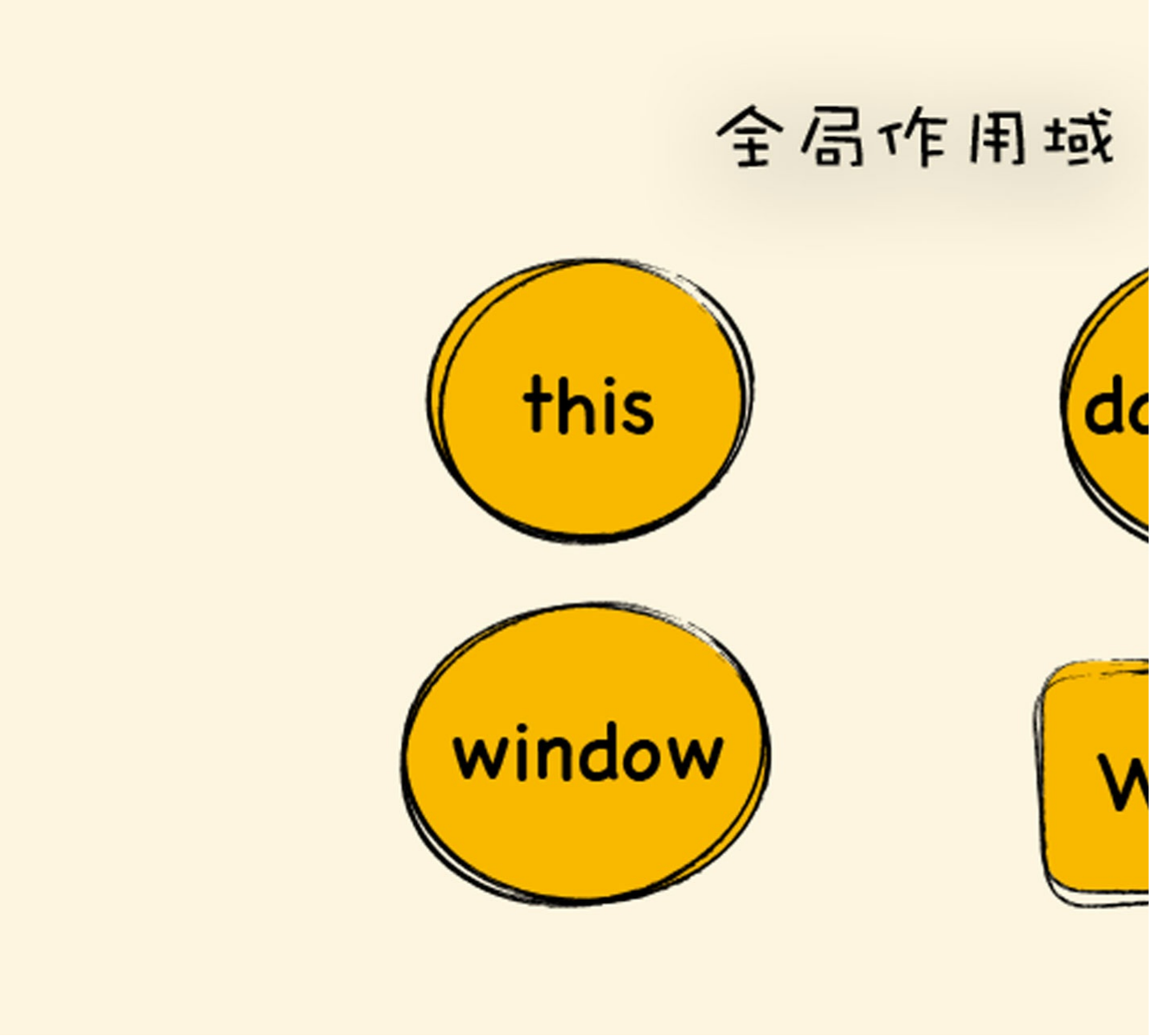
作用域链是怎么工作的？

理解了作用域和作用域链，我们再回过头来看文章开头的那道思考题：“foo函数中打印出来的变量type是bar函数中的呢，还是全局环境中的呢?”我把这段代码复制到下面：

```
var name = '极客时间'
```

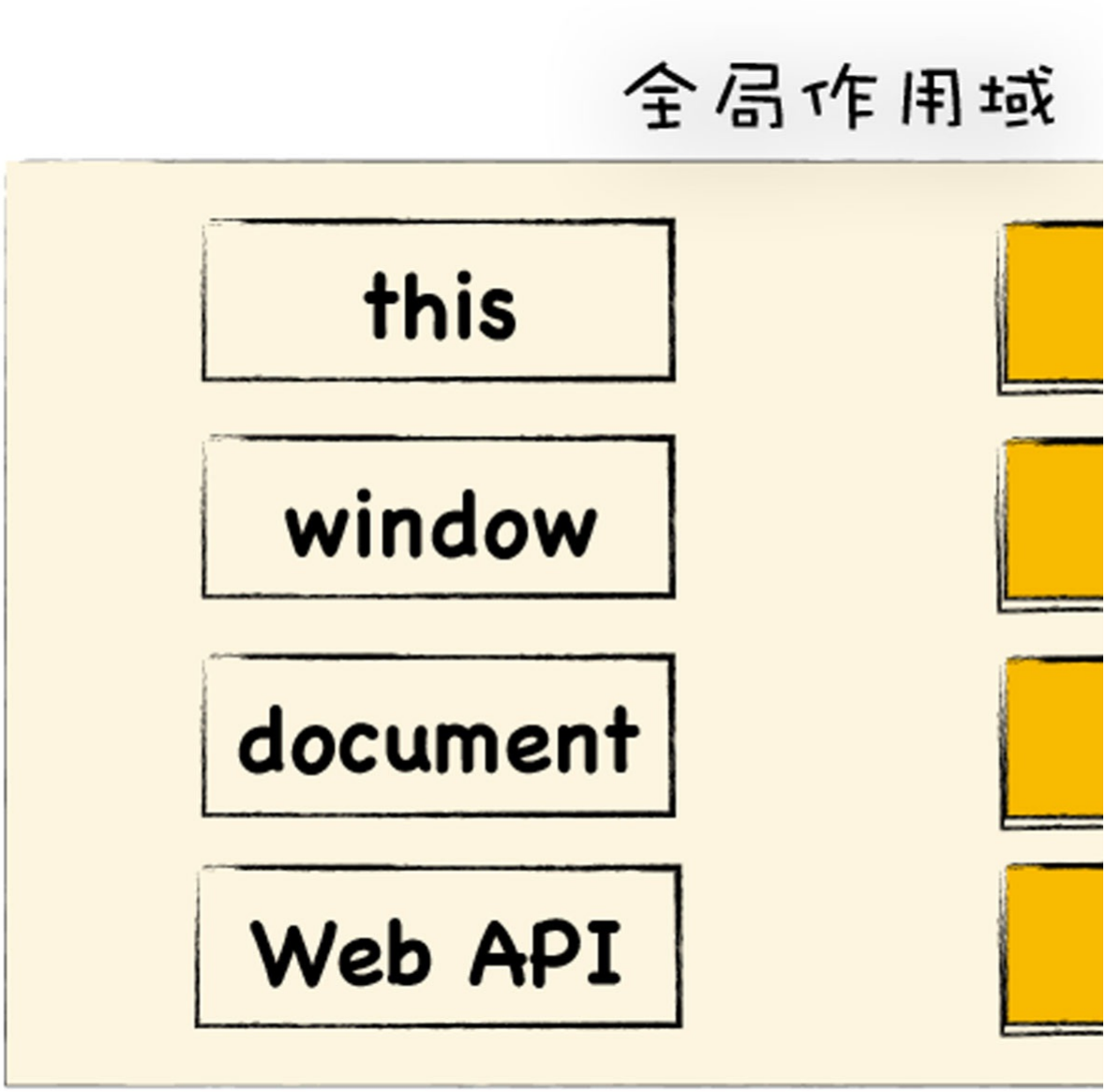
```
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

现在，我们结合V8执行这段代码的流程来具体分析下。首先当V8启动时，会创建全局作用域，全局作用域中包括了this、window等变量，还有一些全局的Web API接口，创建的作用域如下图所示：



V8启动之后，消息循环系统便开始工作了，这时候，我输入了这段代码，让其执行。

V8会先编译顶层代码，在编译过程中会将顶层定义的变量和声明的函数都添加到全局作用域中，最终的全局作用域如下图所示：

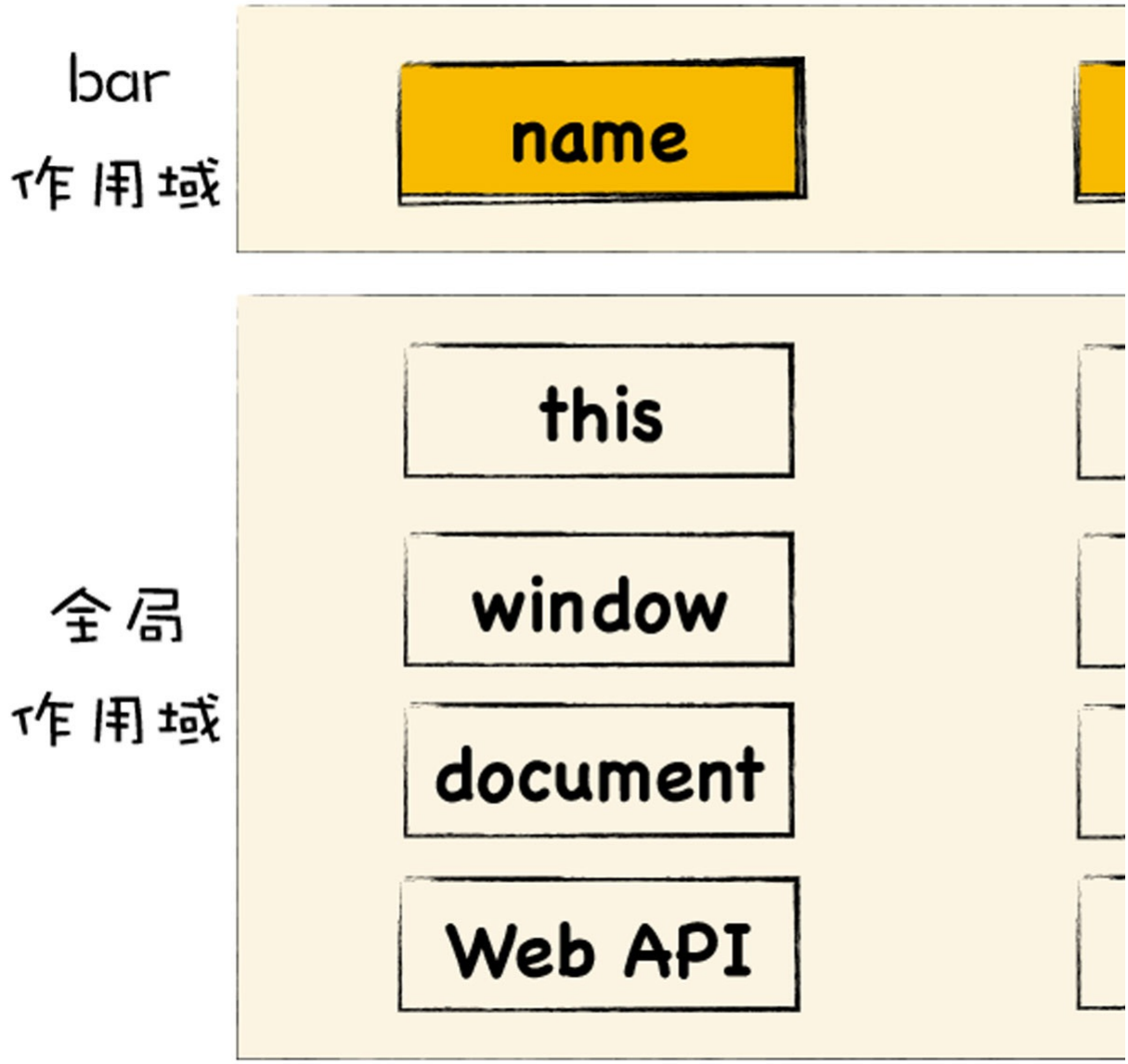


全局作用域创建完成之后，V8便进入了执行状态。前面我们介绍了变量提升，因为变量提升的原因，你可以把上面这段代码分解为如下两个部分：

```
//======解析阶段--实现变量提升=======
var name = undefined
var type = undefined
function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}
function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}



//====执行阶段========
name = '极客时间'
type = 'global'
bar()
```

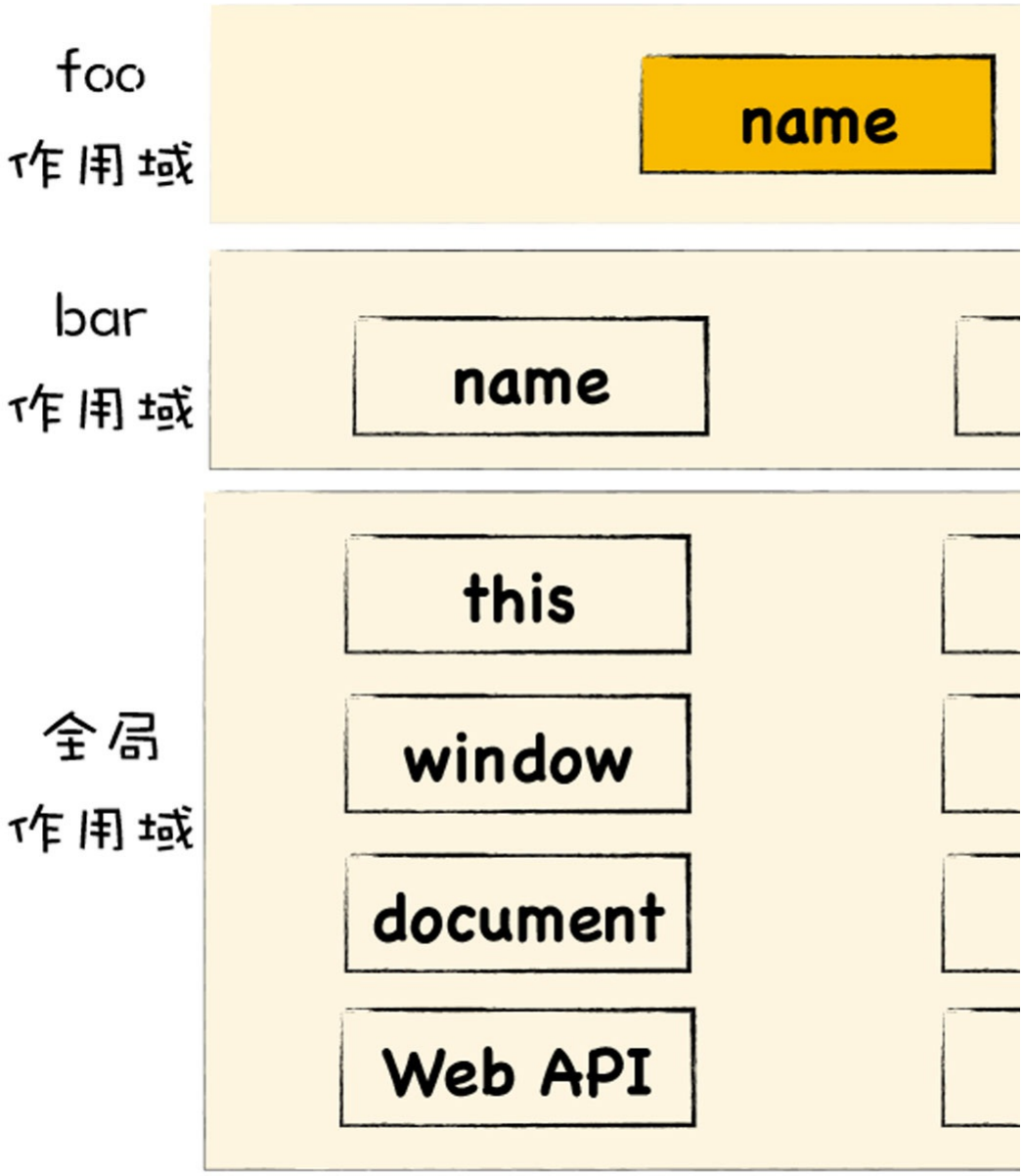
第一部分是在编译过程中完成的，此时全局作用中两个变量的值依然是undefined，然后进入执行阶段；第二部代码就是执行时的顺序，首先全局作用域中的两个变量赋值“极客时间”和“global”，然后就开始执行函数bar的调用了。

当V8执行bar函数的时候，同样需要经历两个阶段：编译和执行。在编译阶段，V8会为bar函数创建函数作用域，最终效果如下所示：



然后进入了bar函数的执行阶段。在bar函数中，只是简单地调用foo函数，因此V8又开始执行foo函数了。

同样，在编译foo函数的过程中，会创建foo函数的作用域，最终创建效果如下图所示：



好了，这时候我们就有了三个作用域了，分别是全局作用域、bar的函数作用域、foo的函数作用域。

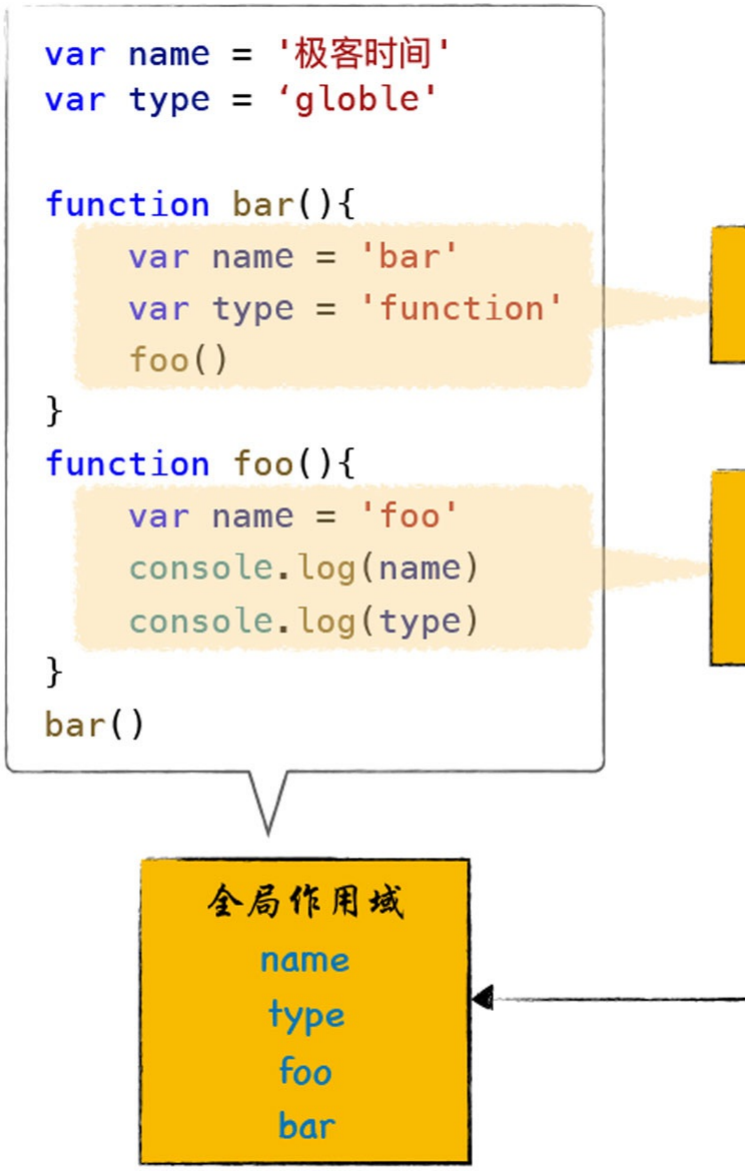
现在我们就可以将刚才提到的问题转换为作用域链的问题了：foo函数查找变量的路径到底是什么？

- 沿着foo函数作用域–>bar函数作用域–>全局作用域;
- 还是，沿着foo函数作用域—>全局作用域？

因为JavaScript是基于词法作用域的，词法作用域就是指，查找作用域的顺序是按照函数定义时的位置来决定的。bar和foo函数的外部代码都是全局代码，所以无论你是在bar函数中查找变量，还是在foo函数中查找变量，其查找顺序都是按照当前函数作用域–>全局作用域这个路径来的。

由于我们代码中的foo函数和bar函数都是在全局下面定义的，所以在foo函数中使用了type，最终打印出来的值就是全局作用域中的type。

你可以参考下面这张图：



另外，我再展开说一些。因为词法作用域是根据函数在代码中的位置来确定的，作用域是在声明函数时就确定好的了，所以我们也将词法作用域称为静态作用域。

和静态作用域相对的是动态作用域，动态作用域并不关心函数和作用域是如何声明以及在何处声明的，只关心它们从**何处调用**。换句话说，作用域链是基于调用栈的，而不是基于函数定义的位置的。（动态作用域不是本文讨论的重点，如果你感兴趣，可以参考《浏览器工作原理与实践》专栏中的《10 | 作用域链和闭包：代码中出现相同的变量，JavaScript引擎是如何选择的？》这一节。）

总结

今天，我们主要解释了一个问题，那就是在一个函数中，如果使用了一个变量，或者调用了另外一个函数，V8将会怎么去查找该变量或者函数。

为了解释清楚这个问题，我们引入了作用域的概念。作用域就是用来存放变量和函数的地方，全局作用域中存放了全局环境中声明的变量和函数，函数作用域中存放了函数中声明的变量和函数。当在某个函数中使用某个变量时，V8就会去这些作用域中查找相关变量。沿着这些作用域查找的路径，我们就称为作用域链。

要了解查找路径，我们需要明白词法作用域，词法作用域是按照代码定义时的位置决定的，而JavaScript所采用的作用域机制就是词法作用域，所以作用域链的路径就是按照词法作用域来实现的。

思考题

我将文章开头那段代码稍微调整了下，foo函数并不是在全局环境中声明的，而是在bar函数中声明的，改造后的代码如下所示：

```
var name = '极客时间'
var type = 'global'
function bar() {
    var type = 'function'
    function foo() {
        console.log(type)
    }
    foo()
}
bar()
```

那么执行这段代码之后，打印出来的内容是什么？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

在前面我们介绍了JavaScript的继承是基于原型链的，原型链将一个个原型对象串起来，从而实现对象属性的查找，今天我们要聊一个和原型链类似的话题，那就是作用域链。

作用域链就是将一个个作用域串起来，实现变量查找的路径。讨论作用域链，实际就是在讨论按照什么路径查找变量的问题。

我们知道，作用域就是存放变量和函数的地方，全局环境有全局作用域，全局作用域中存放了全局变量和全局函数。每个函数也有自己的作用域，函数作用域中存放了函数中定义的变量。

当在函数内部使用一个变量的时候，V8便会去作用域中去查找。我们通过一段在函数内部查找变量的代码来具体看一下：

```
var name = '极客时间'
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

在这段代码中，我们在全局环境中声明了变量name和type，同时还定义了bar函数和foo函数，在bar函数中又再次定义了变量name和type，在foo函数中再次定义了变量name。

函数的调用关系是：在全局环境中调用bar函数，在bar函数中调用foo函数，在foo函数中打印出来变量name和type的值。

当执行到foo函数时，首先需要打印出变量name的值，而我们在三个地方都定义了变量name，那么究竟应该使用哪个变量呢？

在foo函数中使用了变量name，那么V8就应该先使用foo函数内部定义的变量name，最终的结果确实如此，也符合我们的直觉。

接下来，foo函数继续打印变量type，但是在foo函数内部并没有定义变量type，而是在全局环境中和调用foo函数的bar函数中分别定义了变量type，那么这时候的问题来了，你觉得foo函数中打印出来的变量type是bar函数中的，还是全局环境中的呢？

什么是函数作用域和全局作用域？

要解释清楚这个问题，我们需要从作用域的工作原理讲起。

每个函数在执行时都需要查找自己的作用域，我们称为函数作用域，在执行阶段，在执行一个函数时，当该函数需要使用某个变量或者调用了某个函数时，便会优先在该函数作用域中查找相关内容。

我们再来看一段代码：

```
var x = 4
var test
function test_scope() {
    var name = 'foo'
    console.log(name)
    console.log(type)
    console.log(test)
    var type = 'function'
    test = 1
    console.log(x)
}
test_scope()
```

在上面的代码中，我们定义了一个test_scope函数，那么在V8执行test_scope函数的时候，在编译阶段会为test_scope函数创建一个作用域，在test_scope函数中定义的变量和声明的函数都会丢到该作用域中，因为我们在test_scope函数中定了三个变量，那么常见的作用域就包含有这三个变量。

你可以通过Chrome的控制台来直观感受下test_scope函数的作用域，先打开包含这段代码的页面，然后打开开发者工具，接着在test_scope函数中的第二段代码加上断点，然后刷新该页面。当执行到该断点时，V8会暂停整个执行流程，这时候我们就可以通过右边的区域面板来查看当前函数的执行状态。

你可以参考图中右侧的Scope项，然后点击展开该项，这个Local就是当前函数test_scope的作用域。在test_scope函数中定义的变量都包含到了Local中，如变量name、type，另外系统还为我们添加了另外一个隐藏变量this，V8还会默认将隐藏变量this存放到作用域中。

另外你还需要注意下，第一个test1，我并没有采用var等关键字来声明，所以test1并不会出现在test_scope函数的作用域中，而是属于this所指向的对象。（this的工作机制不是本文讨论的重点，不展开介绍。如果你感兴趣，可以在《浏览器工作原理与实践》专栏中《11 | this：从JavaScript执行上下文的视角讲清楚this》这一讲查看。）

那么另一个问题来了，我在test_scope函数使用了变量x，但是在test_scope函数的作用域中，并没有定义变量x，那么V8应该如何获取变量x?

如果在当前函数作用域中没有查找到变量，那么V8会去全局作用域中去查找，这个查找的线路就称为作用域链。

全局作用域和函数作用域类似，也是存放变量和函数的地方，但是它们还是有点不一样：**全局作用域是在V8启动过程中就创建了，且一直保存在内存中不会被销毁的，直至V8退出。 而函数作用域是在执行该函数时创建的，当函数执行结束之后，函数作用域就随之被销毁掉了。**

全局作用域中包含了很多全局变量，比如全局的this值，如果是浏览器，全局作用域中还有window、document、opener等非常多的方法和对象，如果是node环境，那么会有Global、File等内容。

V8启动之后就进入正常的消息循环状态，这时候就可以执行代码了，比如执行到上面那段脚本时，V8会先解析顶层(Top Level)代码，我们可以看到，在顶层代码中定义了变量x，这时候V8就会将变量x添加到全局作用域中。

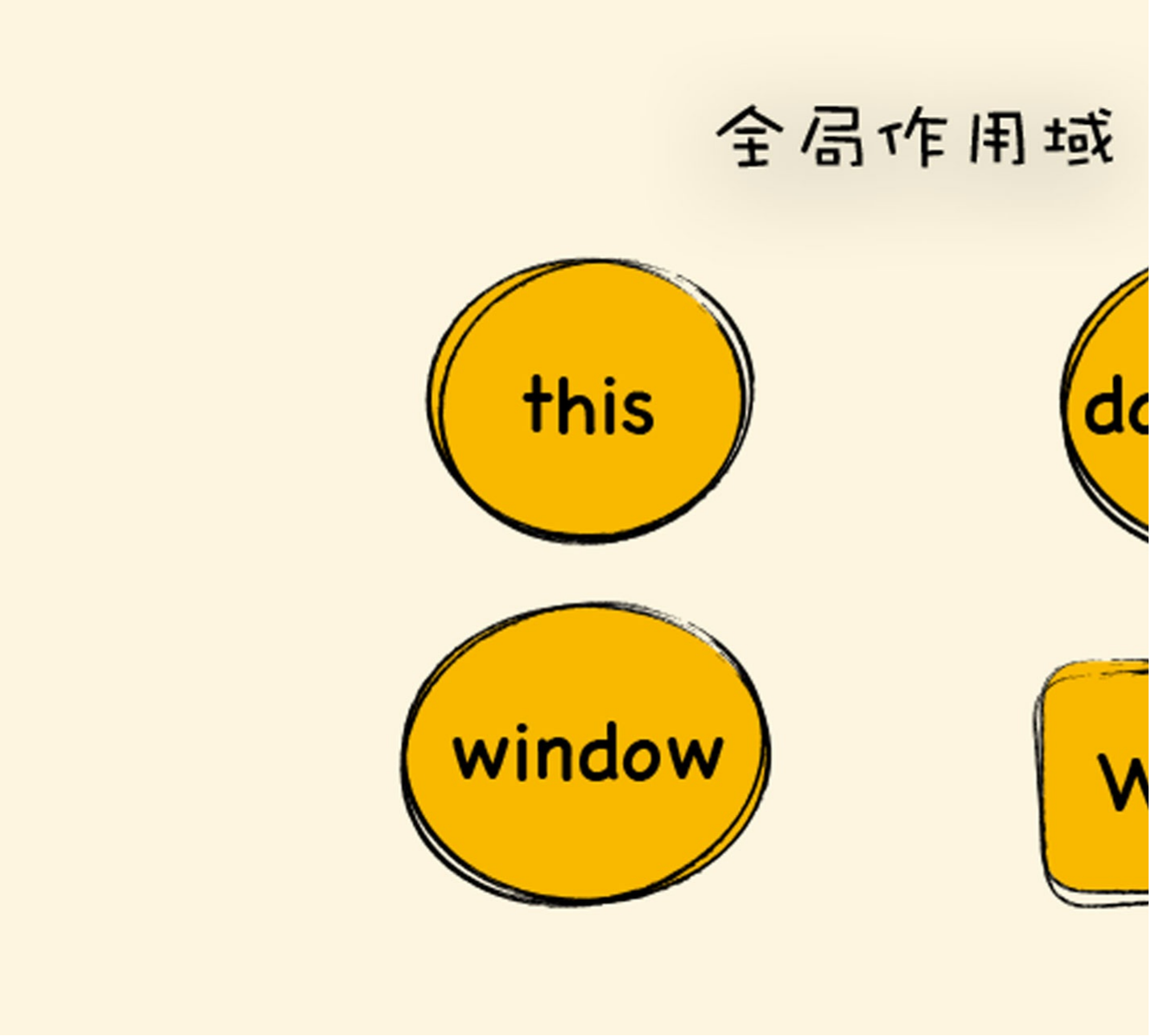
作用域链是怎么工作的？

理解了作用域和作用域链，我们再回过头来看文章开头的那道思考题：“foo函数中打印出来的变量type是bar函数中的呢，还是全局环境中的呢?”我把这段代码复制到下面：

```
var name = '极客时间'
```

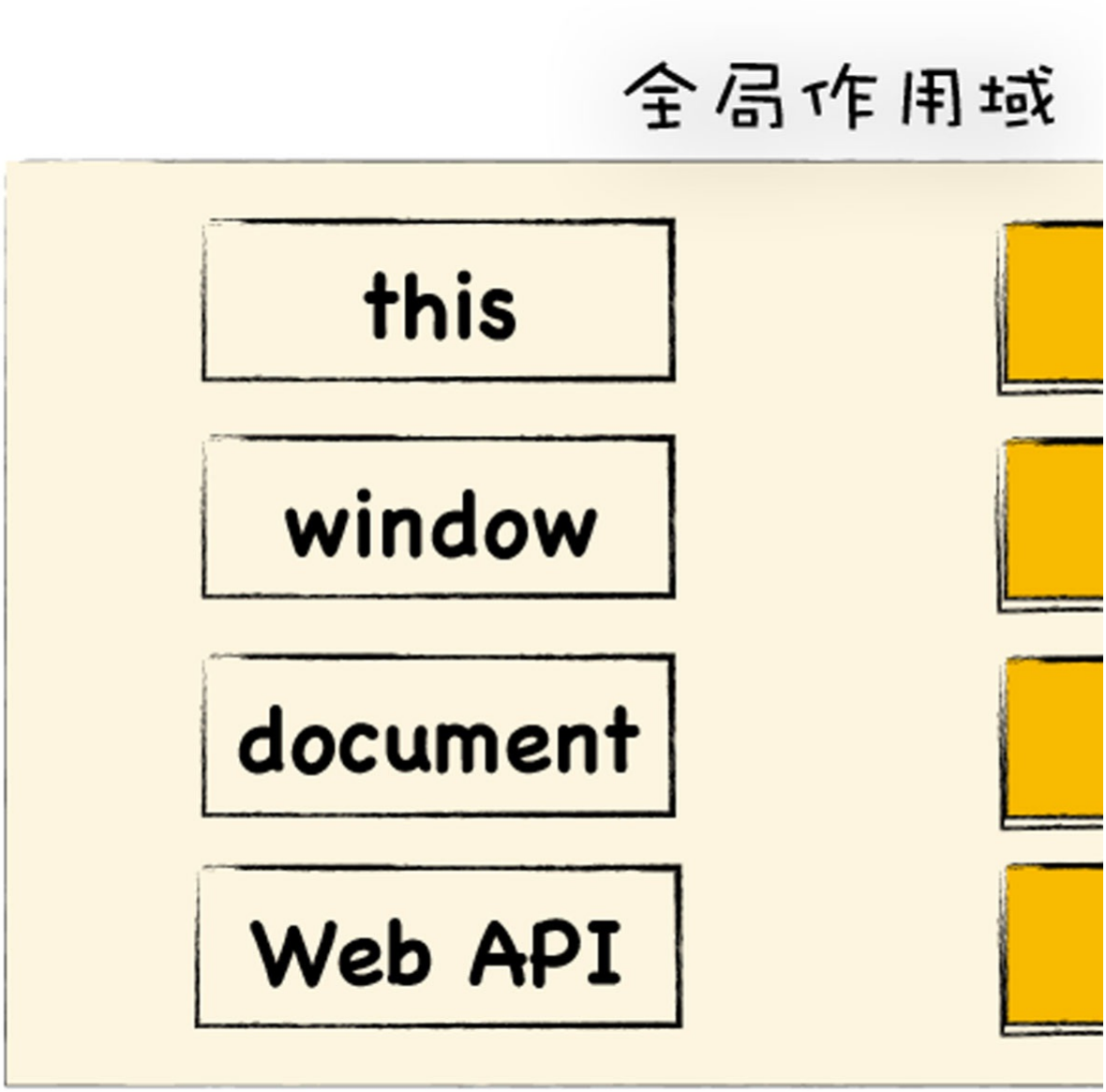
```
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

现在，我们结合V8执行这段代码的流程来具体分析下。首先当V8启动时，会创建全局作用域，全局作用域中包括了this、window等变量，还有一些全局的Web API接口，创建的作用域如下图所示：



V8启动之后，消息循环系统便开始工作了，这时候，我输入了这段代码，让其执行。

V8会先编译顶层代码，在编译过程中会将顶层定义的变量和声明的函数都添加到全局作用域中，最终的全局作用域如下图所示：

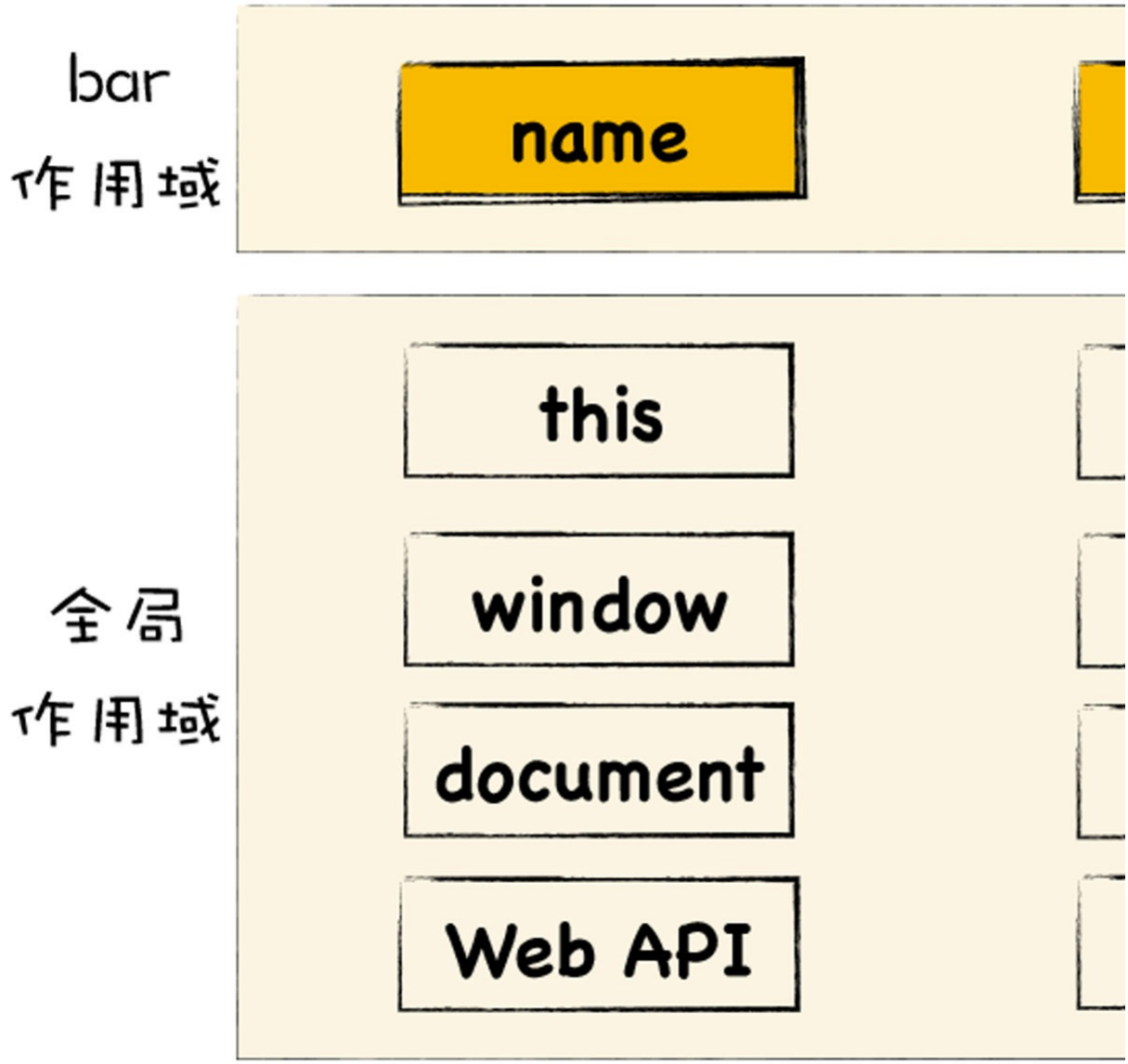


全局作用域创建完成之后，V8便进入了执行状态。前面我们介绍了变量提升，因为变量提升的原因，你可以把上面这段代码分解为如下两个部分：

```
//======解析阶段--实现变量提升=======
var name = undefined
var type = undefined
function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}
function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}



//====执行阶段========
name = '极客时间'
type = 'global'
bar()
```

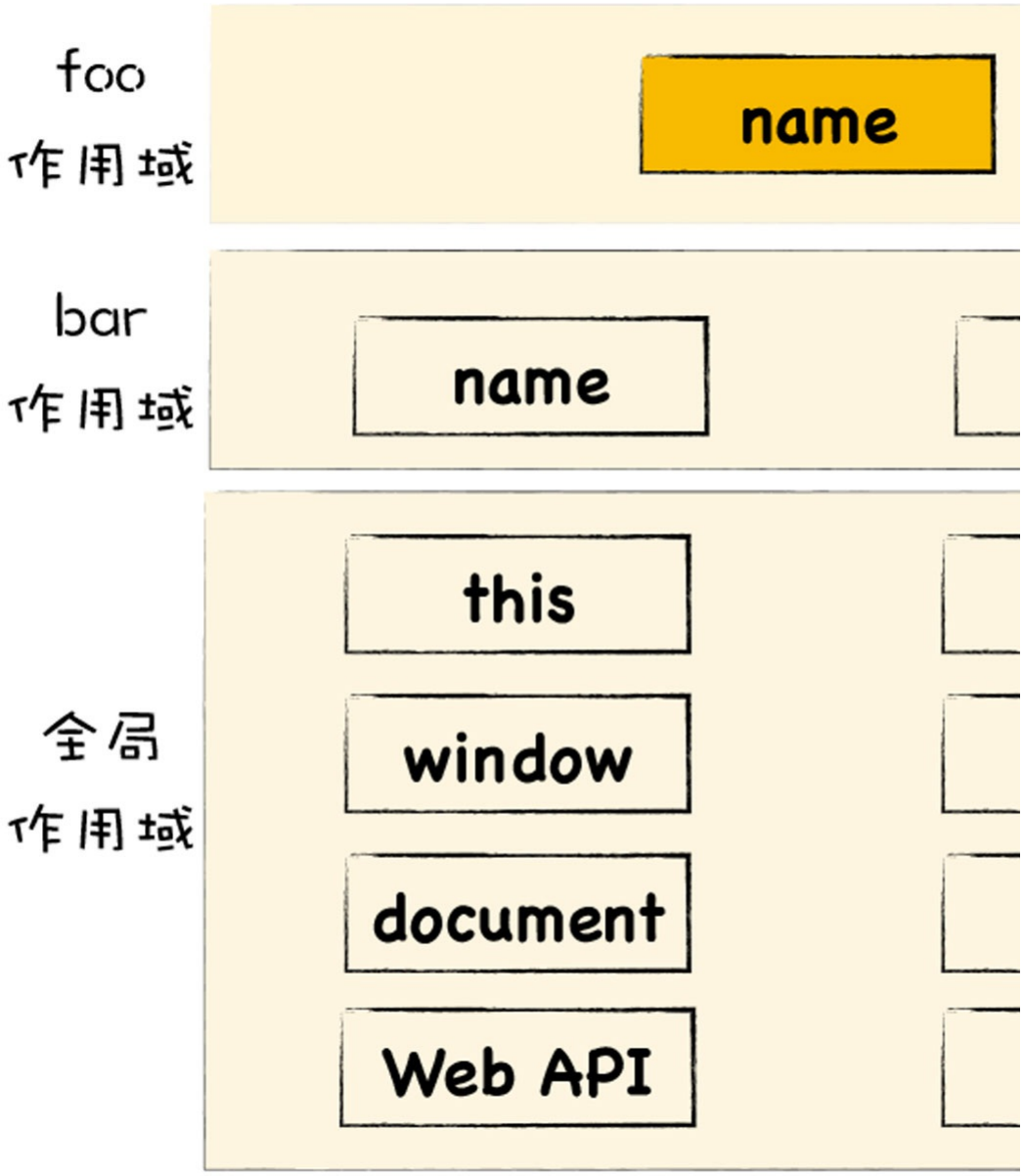
第一部分是在编译过程中完成的，此时全局作用中两个变量的值依然是undefined，然后进入执行阶段；第二部代码就是执行时的顺序，首先全局作用域中的两个变量赋值“极客时间”和“global”，然后就开始执行函数bar的调用了。

当V8执行bar函数的时候，同样需要经历两个阶段：编译和执行。在编译阶段，V8会为bar函数创建函数作用域，最终效果如下所示：



然后进入了bar函数的执行阶段。在bar函数中，只是简单地调用foo函数，因此V8又开始执行foo函数了。

同样，在编译foo函数的过程中，会创建foo函数的作用域，最终创建效果如下图所示：



好了，这时候我们就有了三个作用域了，分别是全局作用域、bar的函数作用域、foo的函数作用域。

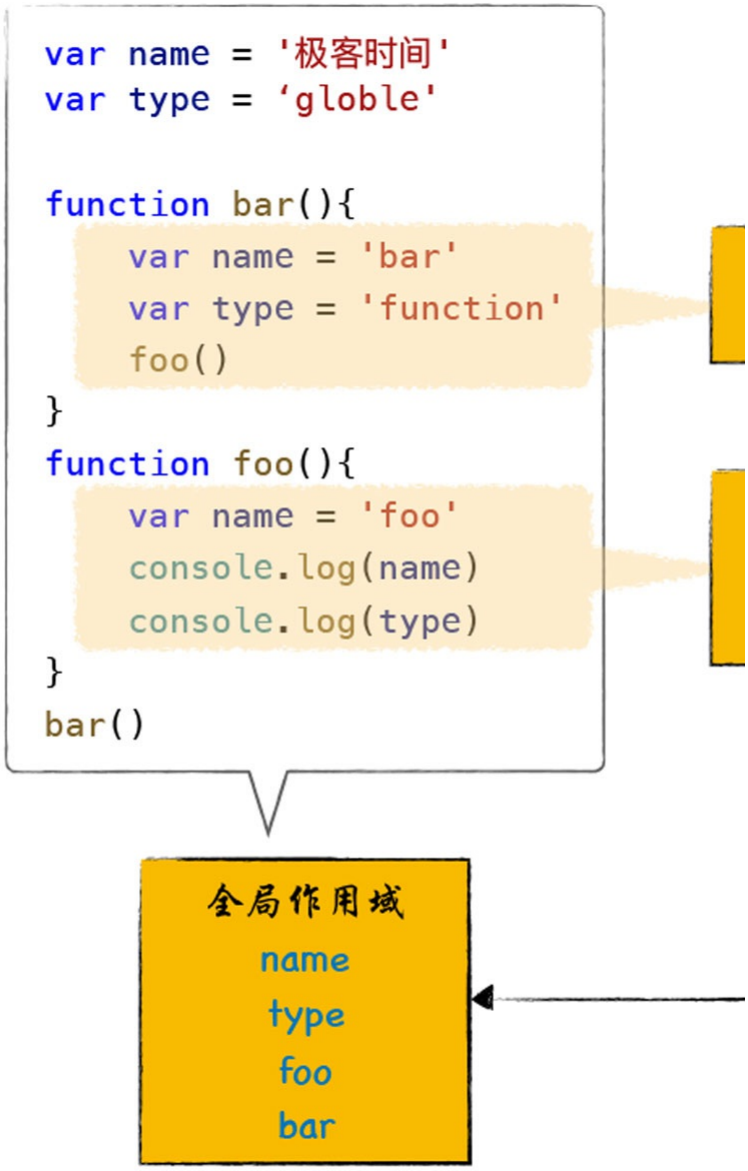
现在我们就可以将刚才提到的问题转换为作用域链的问题了：foo函数查找变量的路径到底是什么？

- 沿着foo函数作用域–>bar函数作用域–>全局作用域;
- 还是，沿着foo函数作用域—>全局作用域？

因为JavaScript是基于词法作用域的，词法作用域就是指，查找作用域的顺序是按照函数定义时的位置来决定的。bar和foo函数的外部代码都是全局代码，所以无论你是在bar函数中查找变量，还是在foo函数中查找变量，其查找顺序都是按照当前函数作用域–>全局作用域这个路径来的。

由于我们代码中的foo函数和bar函数都是在全局下面定义的，所以在foo函数中使用了type，最终打印出来的值就是全局作用域中的type。

你可以参考下面这张图：



另外，我再展开说一些。因为词法作用域是根据函数在代码中的位置来确定的，作用域是在声明函数时就确定好的了，所以我们也将词法作用域称为静态作用域。

和静态作用域相对的是动态作用域，动态作用域并不关心函数和作用域是如何声明以及在何处声明的，只关心它们从**何处调用**。换句话说，作用域链是基于调用栈的，而不是基于函数定义的位置的。（动态作用域不是本文讨论的重点，如果你感兴趣，可以参考《浏览器工作原理与实践》专栏中的《10 | 作用域链和闭包：代码中出现相同的变量，JavaScript引擎是如何选择的？》这一节。）

总结

今天，我们主要解释了一个问题，那就是在一个函数中，如果使用了一个变量，或者调用了另外一个函数，V8将会怎么去查找该变量或者函数。

为了解释清楚这个问题，我们引入了作用域的概念。作用域就是用来存放变量和函数的地方，全局作用域中存放了全局环境中声明的变量和函数，函数作用域中存放了函数中声明的变量和函数。当在某个函数中使用某个变量时，V8就会去这些作用域中查找相关变量。沿着这些作用域查找的路径，我们就称为作用域链。

要了解查找路径，我们需要明白词法作用域，词法作用域是按照代码定义时的位置决定的，而JavaScript所采用的作用域机制就是词法作用域，所以作用域链的路径就是按照词法作用域来实现的。

思考题

我将文章开头那段代码稍微调整了下，foo函数并不是在全局环境中声明的，而是在bar函数中声明的，改造后的代码如下所示：

```
var name = '极客时间'
var type = 'global'
function bar() {
    var type = 'function'
    function foo() {
        console.log(type)
    }
    foo()
}
bar()
```

那么执行这段代码之后，打印出来的内容是什么？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

在前面我们介绍了JavaScript的继承是基于原型链的，原型链将一个个原型对象串起来，从而实现对象属性的查找，今天我们要聊一个和原型链类似的话题，那就是作用域链。

作用域链就是将一个个作用域串起来，实现变量查找的路径。讨论作用域链，实际就是在讨论按照什么路径查找变量的问题。

我们知道，作用域就是存放变量和函数的地方，全局环境有全局作用域，全局作用域中存放了全局变量和全局函数。每个函数也有自己的作用域，函数作用域中存放了函数中定义的变量。

当在函数内部使用一个变量的时候，V8便会去作用域中去查找。我们通过一段在函数内部查找变量的代码来具体看一下：

```
var name = '极客时间'
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

在这段代码中，我们在全局环境中声明了变量name和type，同时还定义了bar函数和foo函数，在bar函数中又再次定义了变量name和type，在foo函数中再次定义了变量name。

函数的调用关系是：在全局环境中调用bar函数，在bar函数中调用foo函数，在foo函数中打印出来变量name和type的值。

当执行到foo函数时，首先需要打印出变量name的值，而我们在三个地方都定义了变量name，那么究竟应该使用哪个变量呢？

在foo函数中使用了变量name，那么V8就应该先使用foo函数内部定义的变量name，最终的结果确实如此，也符合我们的直觉。

接下来，foo函数继续打印变量type，但是在foo函数内部并没有定义变量type，而是在全局环境中和调用foo函数的bar函数中分别定义了变量type，那么这时候的问题来了，你觉得foo函数中打印出来的变量type是bar函数中的，还是全局环境中的呢？

什么是函数作用域和全局作用域？

要解释清楚这个问题，我们需要从作用域的工作原理讲起。

每个函数在执行时都需要查找自己的作用域，我们称为函数作用域，在执行阶段，在执行一个函数时，当该函数需要使用某个变量或者调用了某个函数时，便会优先在该函数作用域中查找相关内容。

我们再来看一段代码：

```
var x = 4
var test
function test_scope() {
    var name = 'foo'
    console.log(name)
    console.log(type)
    console.log(test)
    var type = 'function'
    test = 1
    console.log(x)
}
test_scope()
```

在上面的代码中，我们定义了一个test_scope函数，那么在V8执行test_scope函数的时候，在编译阶段会为test_scope函数创建一个作用域，在test_scope函数中定义的变量和声明的函数都会丢到该作用域中，因为我们在test_scope函数中定了三个变量，那么常见的作用域就包含有这三个变量。

你可以通过Chrome的控制台来直观感受下test_scope函数的作用域，先打开包含这段代码的页面，然后打开开发者工具，接着在test_scope函数中的第二段代码加上断点，然后刷新该页面。当执行到该断点时，V8会暂停整个执行流程，这时候我们就可以通过右边的区域面板来查看当前函数的执行状态。

你可以参考图中右侧的Scope项，然后点击展开该项，这个Local就是当前函数test_scope的作用域。在test_scope函数中定义的变量都包含到了Local中，如变量name、type，另外系统还为我们添加了另外一个隐藏变量this，V8还会默认将隐藏变量this存放到作用域中。

另外你还需要注意下，第一个test1，我并没有采用var等关键字来声明，所以test1并不会出现在test_scope函数的作用域中，而是属于this所指向的对象。（this的工作机制不是本文讨论的重点，不展开介绍。如果你感兴趣，可以在《浏览器工作原理与实践》专栏中《11 | this：从JavaScript执行上下文的视角讲清楚this》这一讲查看。）

那么另一个问题来了，我在test_scope函数使用了变量x，但是在test_scope函数的作用域中，并没有定义变量x，那么V8应该如何获取变量x?

如果在当前函数作用域中没有查找到变量，那么V8会去全局作用域中去查找，这个查找的线路就称为作用域链。

全局作用域和函数作用域类似，也是存放变量和函数的地方，但是它们还是有点不一样：**全局作用域是在V8启动过程中就创建了，且一直保存在内存中不会被销毁的，直至V8退出。 而函数作用域是在执行该函数时创建的，当函数执行结束之后，函数作用域就随之被销毁掉了。**

全局作用域中包含了很多全局变量，比如全局的this值，如果是浏览器，全局作用域中还有window、document、opener等非常多的方法和对象，如果是node环境，那么会有Global、File等内容。

V8启动之后就进入正常的消息循环状态，这时候就可以执行代码了，比如执行到上面那段脚本时，V8会先解析顶层(Top Level)代码，我们可以看到，在顶层代码中定义了变量x，这时候V8就会将变量x添加到全局作用域中。

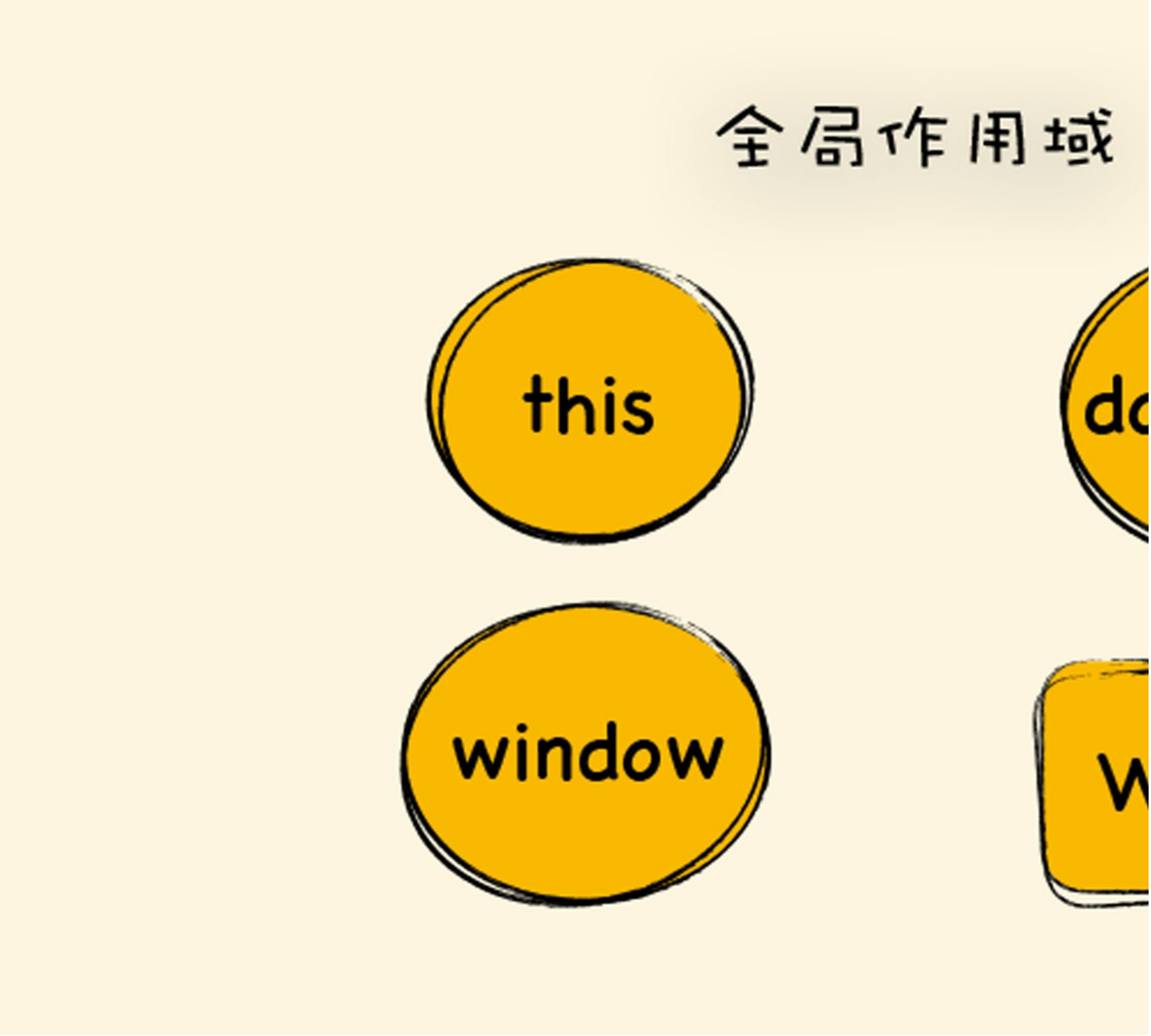
作用域链是怎么工作的？

理解了作用域和作用域链，我们再回过头来看文章开头的那道思考题：“foo函数中打印出来的变量type是bar函数中的呢，还是全局环境中的呢?”我把这段代码复制到下面：

```
var name = '极客时间'
```

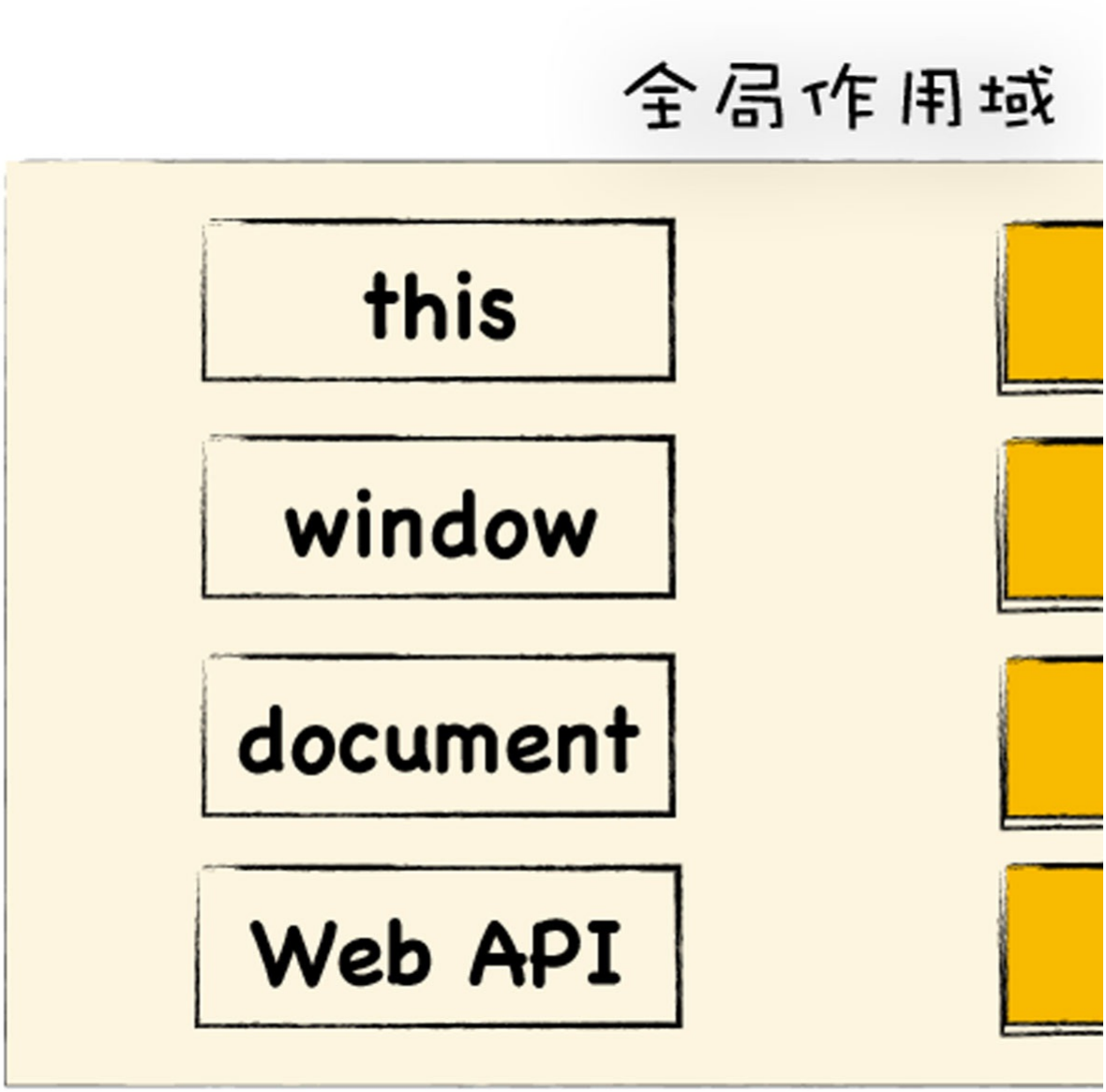
```
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

现在，我们结合V8执行这段代码的流程来具体分析下。首先当V8启动时，会创建全局作用域，全局作用域中包括了this、window等变量，还有一些全局的Web API接口，创建的作用域如下图所示：



V8启动之后，消息循环系统便开始工作了，这时候，我输入了这段代码，让其执行。

V8会先编译顶层代码，在编译过程中会将顶层定义的变量和声明的函数都添加到全局作用域中，最终的全局作用域如下图所示：

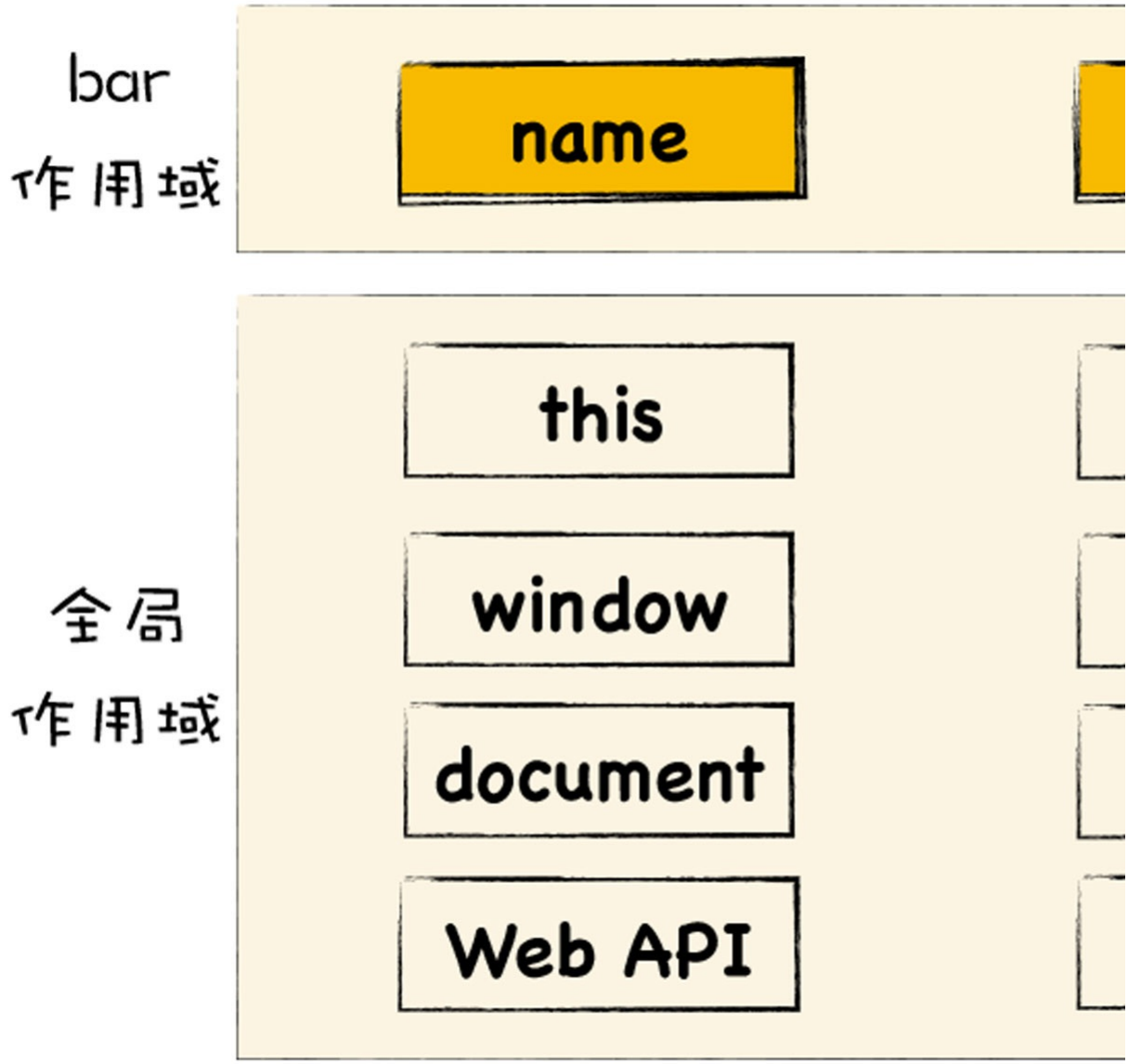


全局作用域创建完成之后，V8便进入了执行状态。前面我们介绍了变量提升，因为变量提升的原因，你可以把上面这段代码分解为如下两个部分：

```
//======解析阶段--实现变量提升=======
var name = undefined
var type = undefined
function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}
function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}



//====执行阶段========
name = '极客时间'
type = 'global'
bar()
```

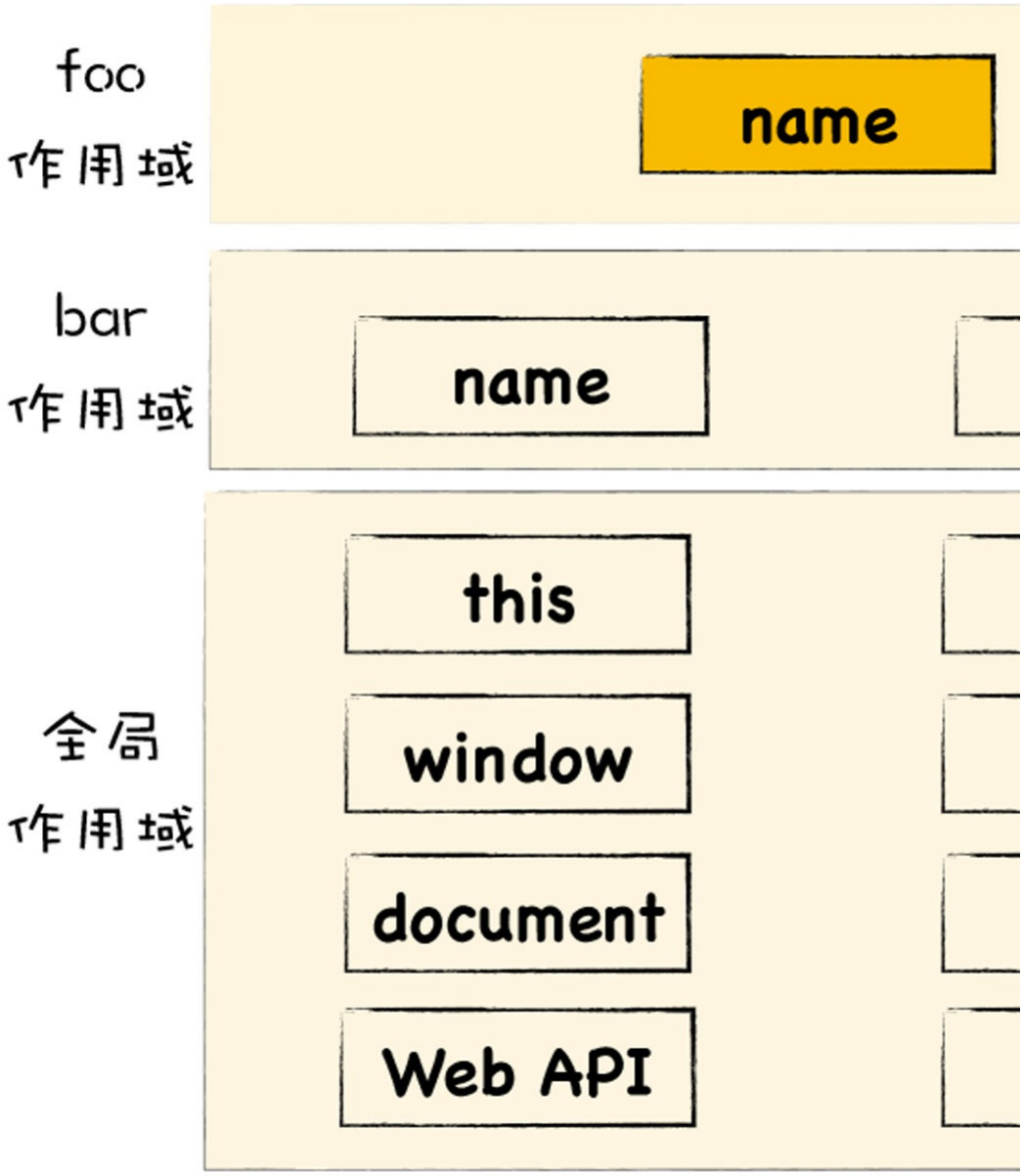
第一部分是在编译过程中完成的，此时全局作用中两个变量的值依然是undefined，然后进入执行阶段；第二部代码就是执行时的顺序，首先全局作用域中的两个变量赋值“极客时间”和“global”，然后就开始执行函数bar的调用了。

当V8执行bar函数的时候，同样需要经历两个阶段：编译和执行。在编译阶段，V8会为bar函数创建函数作用域，最终效果如下所示：



然后进入了bar函数的执行阶段。在bar函数中，只是简单地调用foo函数，因此V8又开始执行foo函数了。

同样，在编译foo函数的过程中，会创建foo函数的作用域，最终创建效果如下图所示：



好了，这时候我们就有了三个作用域了，分别是全局作用域、bar的函数作用域、foo的函数作用域。

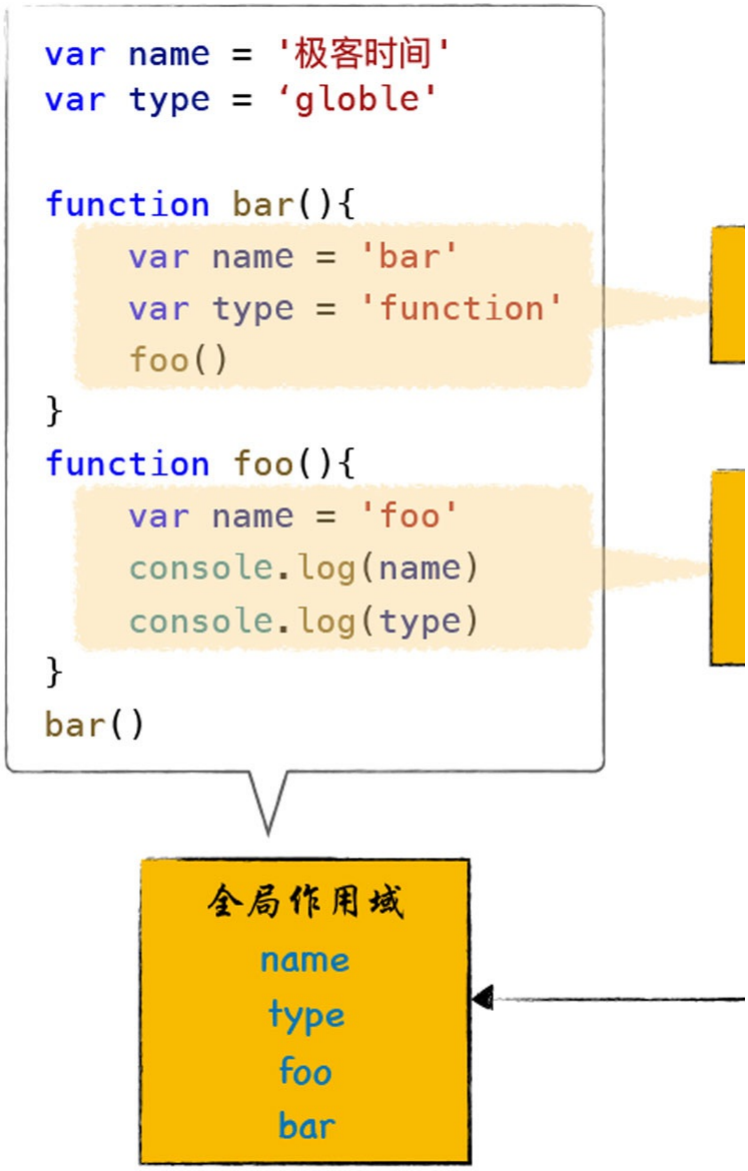
现在我们就可以将刚才提到的问题转换为作用域链的问题了：foo函数查找变量的路径到底是什么？

- 沿着foo函数作用域–>bar函数作用域–>全局作用域;
- 还是，沿着foo函数作用域—>全局作用域？

因为JavaScript是基于词法作用域的，词法作用域就是指，查找作用域的顺序是按照函数定义时的位置来决定的。bar和foo函数的外部代码都是全局代码，所以无论你是在bar函数中查找变量，还是在foo函数中查找变量，其查找顺序都是按照当前函数作用域–>全局作用域这个路径来的。

由于我们代码中的foo函数和bar函数都是在全局下面定义的，所以在foo函数中使用了type，最终打印出来的值就是全局作用域中的type。

你可以参考下面这张图：



另外，我再展开说一些。因为词法作用域是根据函数在代码中的位置来确定的，作用域是在声明函数时就确定好的了，所以我们也将词法作用域称为静态作用域。

和静态作用域相对的是动态作用域，动态作用域并不关心函数和作用域是如何声明以及在何处声明的，只关心它们从**何处调用**。换句话说，作用域链是基于调用栈的，而不是基于函数定义的位置的。（动态作用域不是本文讨论的重点，如果你感兴趣，可以参考《浏览器工作原理与实践》专栏中的《10 | 作用域链和闭包：代码中出现相同的变量，JavaScript引擎是如何选择的？》这一节。）

总结

今天，我们主要解释了一个问题，那就是在一个函数中，如果使用了一个变量，或者调用了另外一个函数，V8将会怎么去查找该变量或者函数。

为了解释清楚这个问题，我们引入了作用域的概念。作用域就是用来存放变量和函数的地方，全局作用域中存放了全局环境中声明的变量和函数，函数作用域中存放了函数中声明的变量和函数。当在某个函数中使用某个变量时，V8就会去这些作用域中查找相关变量。沿着这些作用域查找的路径，我们就称为作用域链。

要了解查找路径，我们需要明白词法作用域，词法作用域是按照代码定义时的位置决定的，而JavaScript所采用的作用域机制就是词法作用域，所以作用域链的路径就是按照词法作用域来实现的。

思考题

我将文章开头那段代码稍微调整了下，foo函数并不是在全局环境中声明的，而是在bar函数中声明的，改造后的代码如下所示：

```
var name = '极客时间'
var type = 'global'
function bar() {
    var type = 'function'
    function foo() {
        console.log(type)
    }
    foo()
}
bar()
```

那么执行这段代码之后，打印出来的内容是什么？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

在前面我们介绍了JavaScript的继承是基于原型链的，原型链将一个个原型对象串起来，从而实现对象属性的查找，今天我们要聊一个和原型链类似的话题，那就是作用域链。

作用域链就是将一个个作用域串起来，实现变量查找的路径。讨论作用域链，实际就是在讨论按照什么路径查找变量的问题。

我们知道，作用域就是存放变量和函数的地方，全局环境有全局作用域，全局作用域中存放了全局变量和全局函数。每个函数也有自己的作用域，函数作用域中存放了函数中定义的变量。

当在函数内部使用一个变量的时候，V8便会去作用域中去查找。我们通过一段在函数内部查找变量的代码来具体看一下：

```
var name = '极客时间'
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

在这段代码中，我们在全局环境中声明了变量name和type，同时还定义了bar函数和foo函数，在bar函数中又再次定义了变量name和type，在foo函数中再次定义了变量name。

函数的调用关系是：在全局环境中调用bar函数，在bar函数中调用foo函数，在foo函数中打印出来变量name和type的值。

当执行到foo函数时，首先需要打印出变量name的值，而我们在三个地方都定义了变量name，那么究竟应该使用哪个变量呢？

在foo函数中使用了变量name，那么V8就应该先使用foo函数内部定义的变量name，最终的结果确实如此，也符合我们的直觉。

接下来，foo函数继续打印变量type，但是在foo函数内部并没有定义变量type，而是在全局环境中和调用foo函数的bar函数中分别定义了变量type，那么这时候的问题来了，你觉得foo函数中打印出来的变量type是bar函数中的，还是全局环境中的呢？

什么是函数作用域和全局作用域？

要解释清楚这个问题，我们需要从作用域的工作原理讲起。

每个函数在执行时都需要查找自己的作用域，我们称为函数作用域，在执行阶段，在执行一个函数时，当该函数需要使用某个变量或者调用了某个函数时，便会优先在该函数作用域中查找相关内容。

我们再来看一段代码：

```
var x = 4
var test
function test_scope() {
    var name = 'foo'
    console.log(name)
    console.log(type)
    console.log(test)
    var type = 'function'
    test = 1
    console.log(x)
}
test_scope()
```

在上面的代码中，我们定义了一个test_scope函数，那么在V8执行test_scope函数的时候，在编译阶段会为test_scope函数创建一个作用域，在test_scope函数中定义的变量和声明的函数都会丢到该作用域中，因为我们在test_scope函数中定了三个变量，那么常见的作用域就包含有这三个变量。

你可以通过Chrome的控制台来直观感受下test_scope函数的作用域，先打开包含这段代码的页面，然后打开开发者工具，接着在test_scope函数中的第二段代码加上断点，然后刷新该页面。当执行到该断点时，V8会暂停整个执行流程，这时候我们就可以通过右边的区域面板来查看当前函数的执行状态。

你可以参考图中右侧的Scope项，然后点击展开该项，这个Local就是当前函数test_scope的作用域。在test_scope函数中定义的变量都包含到了Local中，如变量name、type，另外系统还为我们添加了另外一个隐藏变量this，V8还会默认将隐藏变量this存放到作用域中。

另外你还需要注意下，第一个test1，我并没有采用var等关键字来声明，所以test1并不会出现在test_scope函数的作用域中，而是属于this所指向的对象。（this的工作机制不是本文讨论的重点，不展开介绍。如果你感兴趣，可以在《浏览器工作原理与实践》专栏中《11 | this：从JavaScript执行上下文的视角讲清楚this》这一讲查看。）

那么另一个问题来了，我在test_scope函数使用了变量x，但是在test_scope函数的作用域中，并没有定义变量x，那么V8应该如何获取变量x?

如果在当前函数作用域中没有查找到变量，那么V8会去全局作用域中去查找，这个查找的线路就称为作用域链。

全局作用域和函数作用域类似，也是存放变量和函数的地方，但是它们还是有点不一样：**全局作用域是在V8启动过程中就创建了，且一直保存在内存中不会被销毁的，直至V8退出。 而函数作用域是在执行该函数时创建的，当函数执行结束之后，函数作用域就随之被销毁掉了。**

全局作用域中包含了很多全局变量，比如全局的this值，如果是浏览器，全局作用域中还有window、document、opener等非常多的方法和对象，如果是node环境，那么会有Global、File等内容。

V8启动之后就进入正常的消息循环状态，这时候就可以执行代码了，比如执行到上面那段脚本时，V8会先解析顶层(Top Level)代码，我们可以看到，在顶层代码中定义了变量x，这时候V8就会将变量x添加到全局作用域中。

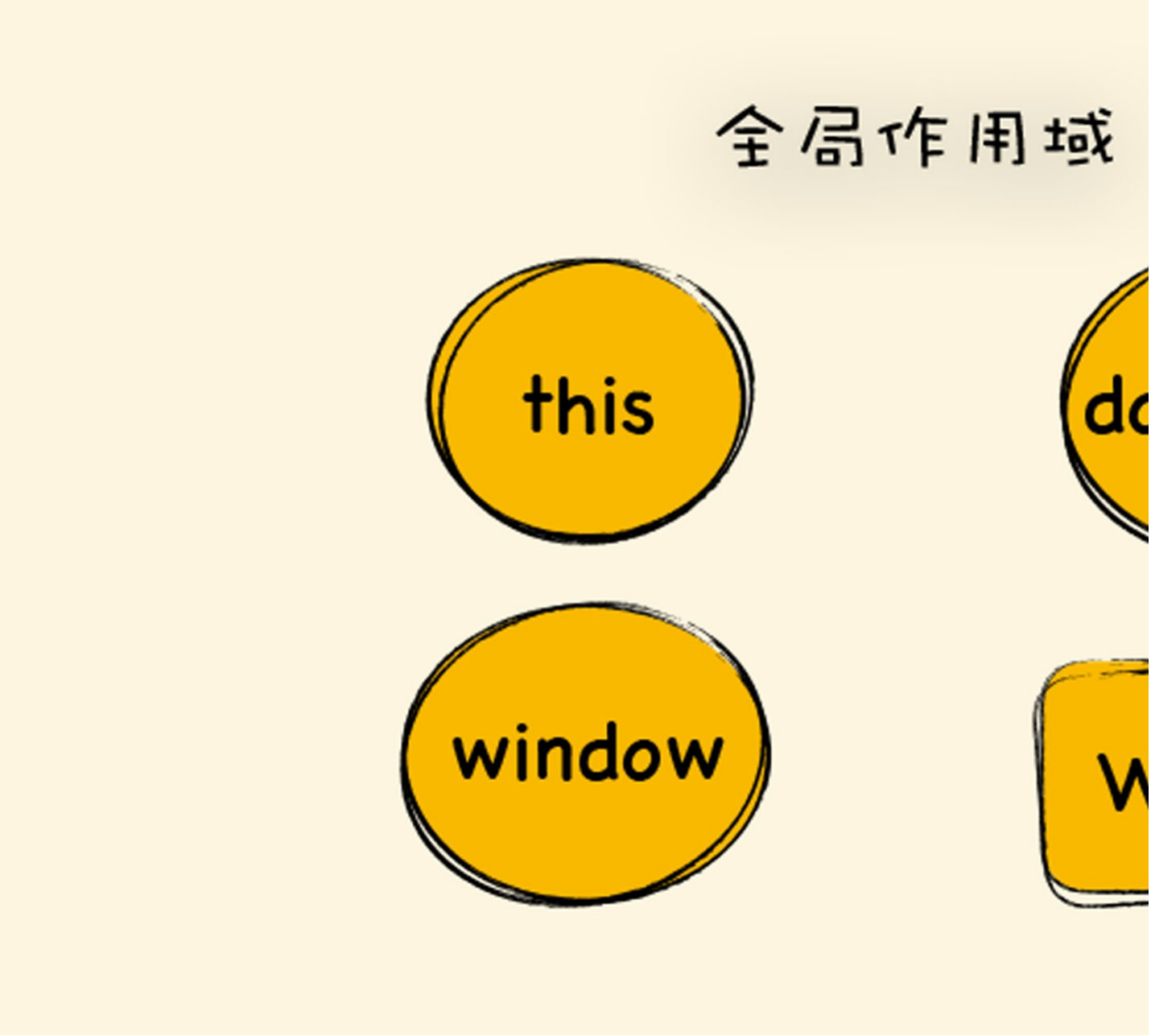
作用域链是怎么工作的？

理解了作用域和作用域链，我们再回过头来看文章开头的那道思考题：“foo函数中打印出来的变量type是bar函数中的呢，还是全局环境中的呢?”我把这段代码复制到下面：

```
var name = '极客时间'
```

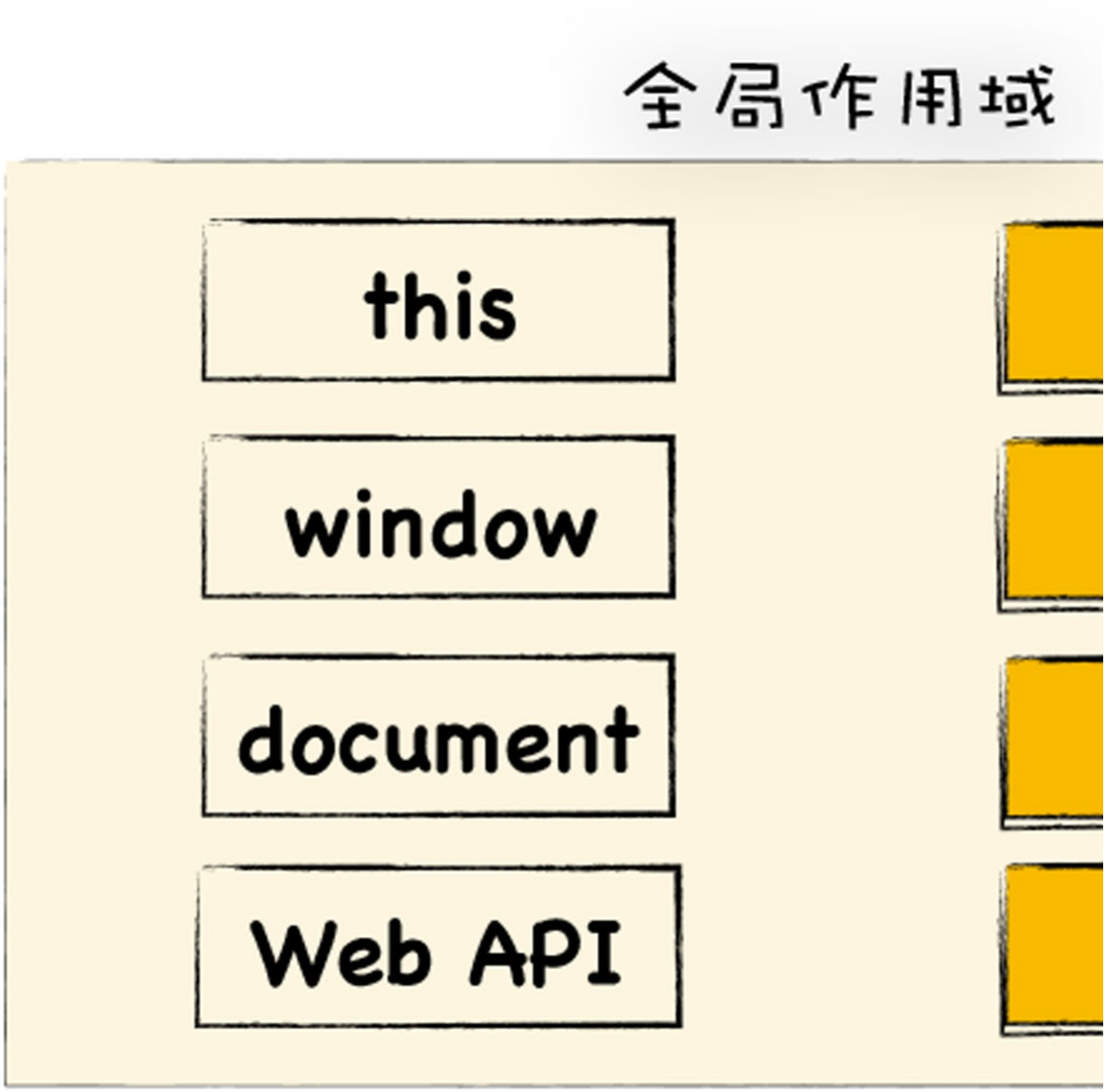
```
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

现在，我们结合V8执行这段代码的流程来具体分析下。首先当V8启动时，会创建全局作用域，全局作用域中包括了this、window等变量，还有一些全局的Web API接口，创建的作用域如下图所示：



V8启动之后，消息循环系统便开始工作了，这时候，我输入了这段代码，让其执行。

V8会先编译顶层代码，在编译过程中会将顶层定义的变量和声明的函数都添加到全局作用域中，最终的全局作用域如下图所示：

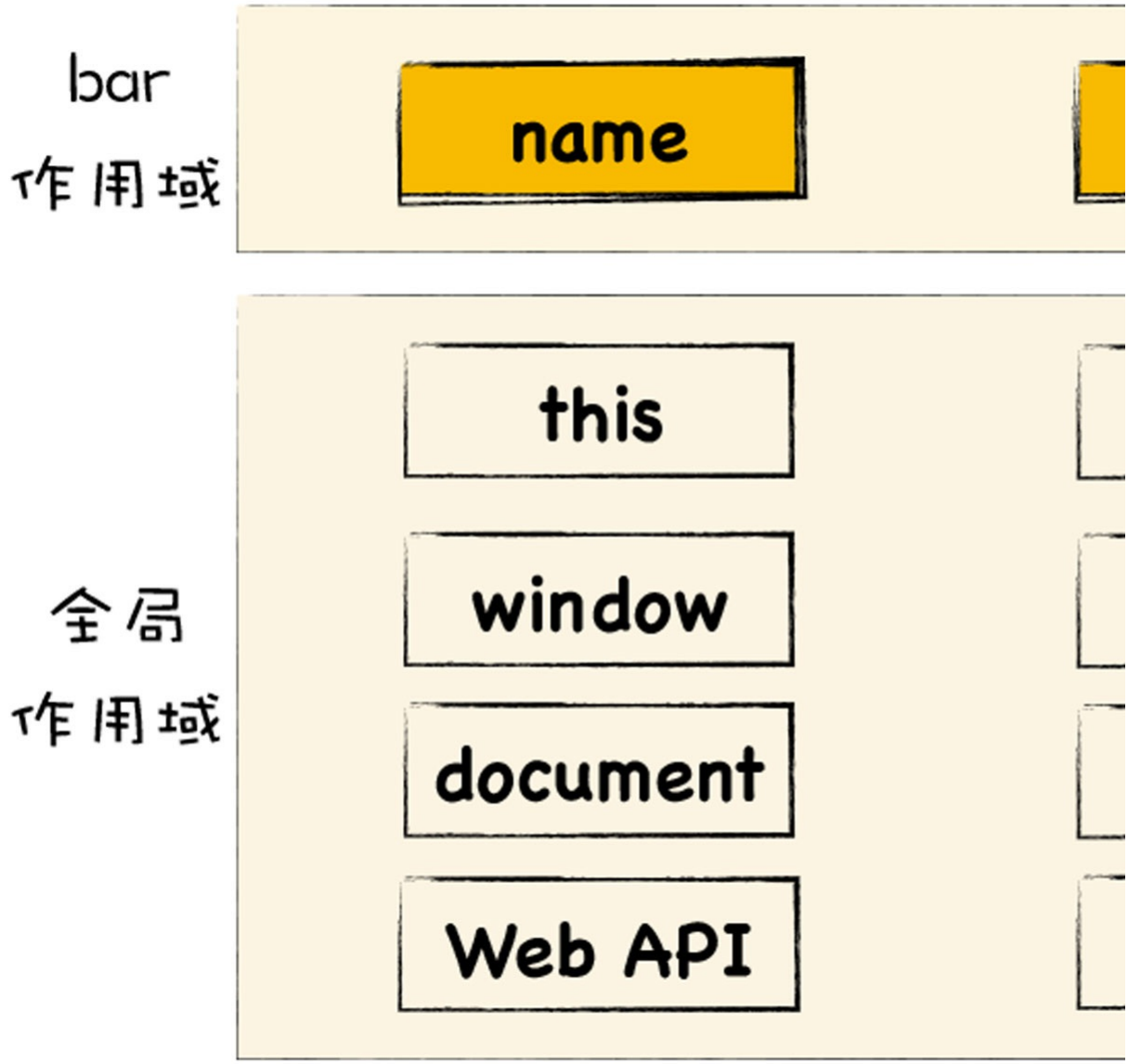


全局作用域创建完成之后，V8便进入了执行状态。前面我们介绍了变量提升，因为变量提升的原因，你可以把上面这段代码分解为如下两个部分：

```
//======解析阶段--实现变量提升=======
var name = undefined
var type = undefined
function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}
function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}



//====执行阶段========
name = '极客时间'
type = 'global'
bar()
```

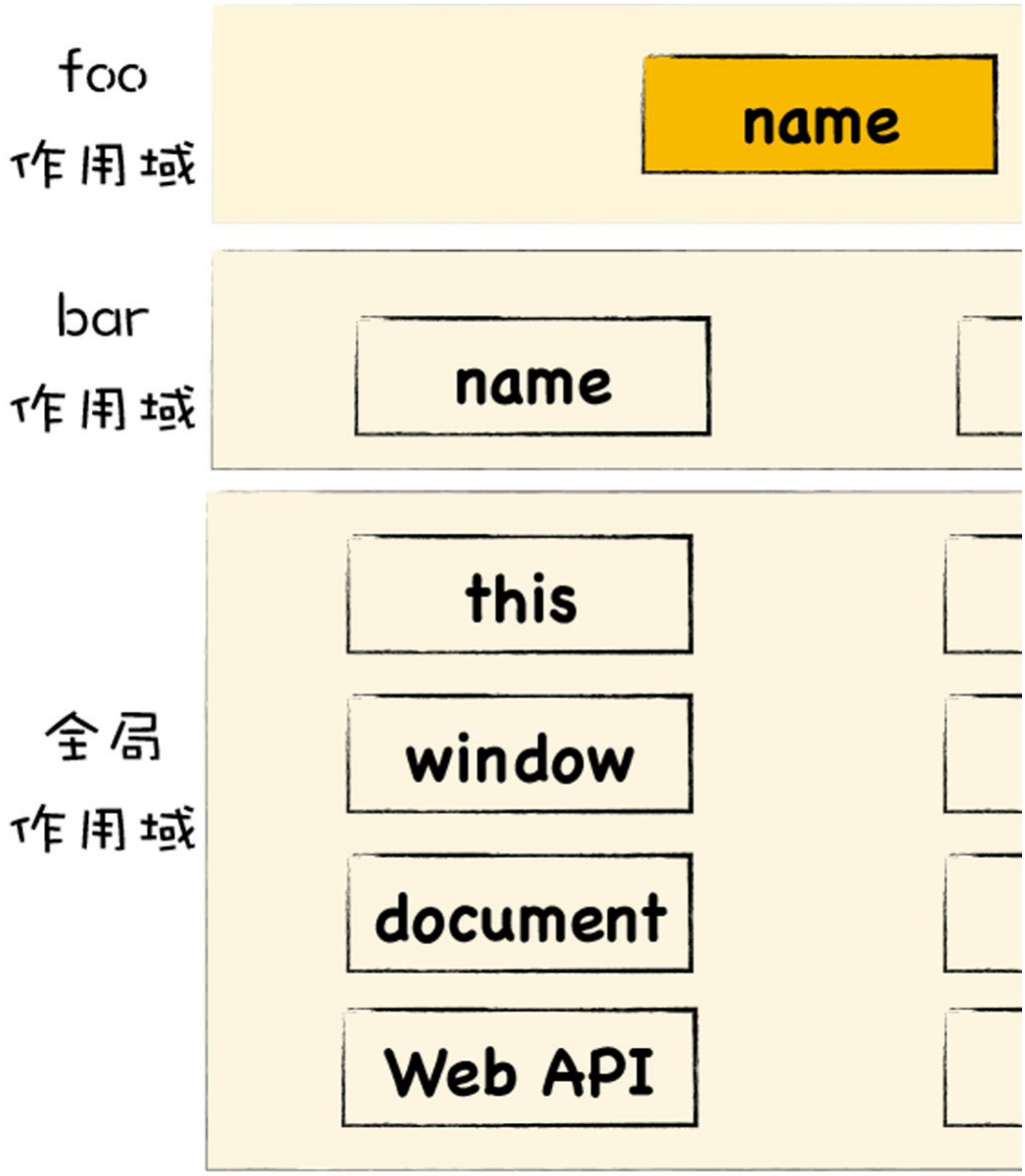
第一部分是在编译过程中完成的，此时全局作用中两个变量的值依然是undefined，然后进入执行阶段；第二部代码就是执行时的顺序，首先全局作用域中的两个变量赋值“极客时间”和“global”，然后就开始执行函数bar的调用了。

当V8执行bar函数的时候，同样需要经历两个阶段：编译和执行。在编译阶段，V8会为bar函数创建函数作用域，最终效果如下所示：



然后进入了bar函数的执行阶段。在bar函数中，只是简单地调用foo函数，因此V8又开始执行foo函数了。

同样，在编译foo函数的过程中，会创建foo函数的作用域，最终创建效果如下图所示：



好了，这时候我们就有了三个作用域了，分别是全局作用域、bar的函数作用域、foo的函数作用域。

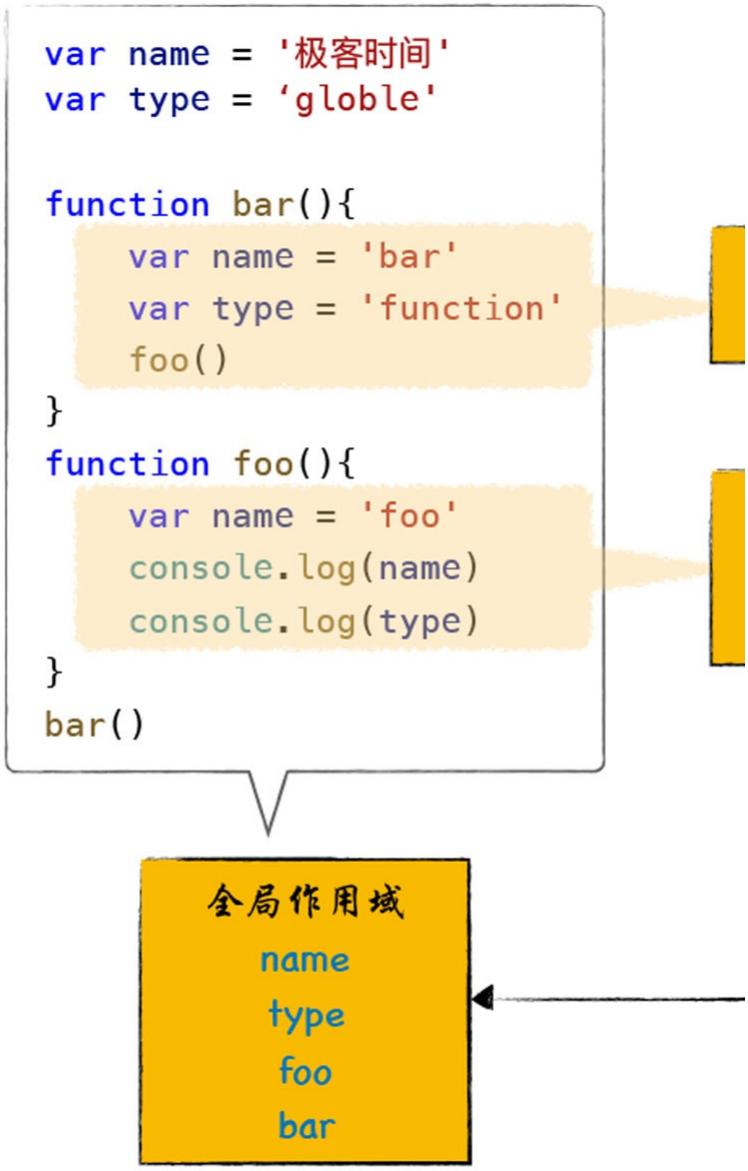
现在我们就可以将刚才提到的问题转换为作用域链的问题了：foo函数查找变量的路径到底是什么？

- 沿着foo函数作用域–>bar函数作用域–>全局作用域;
- 还是，沿着foo函数作用域—>全局作用域？

因为JavaScript是基于词法作用域的，词法作用域就是指，查找作用域的顺序是按照函数定义时的位置来决定的。bar和foo函数的外部代码都是全局代码，所以无论你是在bar函数中查找变量，还是在foo函数中查找变量，其查找顺序都是按照当前函数作用域–>全局作用域这个路径来的。

由于我们代码中的foo函数和bar函数都是在全局下面定义的，所以在foo函数中使用了type，最终打印出来的值就是全局作用域中的type。

你可以参考下面这张图：



另外，我再展开说一些。因为词法作用域是根据函数在代码中的位置来确定的，作用域是在声明函数时就确定好的了，所以我们也将词法作用域称为静态作用域。

和静态作用域相对的是动态作用域，动态作用域并不关心函数和作用域是如何声明以及在何处声明的，只关心它们从**何处调用**。换句话说，作用域链是基于调用栈的，而不是基于函数定义的位置的。（动态作用域不是本文讨论的重点，如果你感兴趣，可以参考《浏览器工作原理与实践》专栏中的《10 | 作用域链和闭包：代码中出现相同的变量，JavaScript引擎是如何选择的？》这一节。）

总结

今天，我们主要解释了一个问题，那就是在一个函数中，如果使用了一个变量，或者调用了另外一个函数，V8将会怎么去查找该变量或者函数。

为了解释清楚这个问题，我们引入了作用域的概念。作用域就是用来存放变量和函数的地方，全局作用域中存放了全局环境中声明的变量和函数，函数作用域中存放了函数中声明的变量和函数。当在某个函数中使用某个变量时，V8就会去这些作用域中查找相关变量。沿着这些作用域查找的路径，我们就称为作用域链。

要了解查找路径，我们需要明白词法作用域，词法作用域是按照代码定义时的位置决定的，而JavaScript所采用的作用域机制就是词法作用域，所以作用域链的路径就是按照词法作用域来实现的。

思考题

我将文章开头那段代码稍微调整了下，foo函数并不是在全局环境中声明的，而是在bar函数中声明的，改造后的代码如下所示：

```
var name = '极客时间'
var type = 'global'
function bar() {
    var type = 'function'
    function foo() {
        console.log(type)
    }
    foo()
}
bar()
```

那么执行这段代码之后，打印出来的内容是什么？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

在前面我们介绍了JavaScript的继承是基于原型链的，原型链将一个个原型对象串起来，从而实现对象属性的查找，今天我们要聊一个和原型链类似的话题，那就是作用域链。

作用域链就是将一个个作用域串起来，实现变量查找的路径。讨论作用域链，实际就是在讨论按照什么路径查找变量的问题。

我们知道，作用域就是存放变量和函数的地方，全局环境有全局作用域，全局作用域中存放了全局变量和全局函数。每个函数也有自己的作用域，函数作用域中存放了函数中定义的变量。

当在函数内部使用一个变量的时候，V8便会去作用域中去查找。我们通过一段在函数内部查找变量的代码来具体看一下：

```
var name = '极客时间'
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

在这段代码中，我们在全局环境中声明了变量name和type，同时还定义了bar函数和foo函数，在bar函数中又再次定义了变量name和type，在foo函数中再次定义了变量name。

函数的调用关系是：在全局环境中调用bar函数，在bar函数中调用foo函数，在foo函数中打印出来变量name和type的值。

当执行到foo函数时，首先需要打印出变量name的值，而我们在三个地方都定义了变量name，那么究竟应该使用哪个变量呢？

在foo函数中使用了变量name，那么V8就应该先使用foo函数内部定义的变量name，最终的结果确实如此，也符合我们的直觉。

接下来，foo函数继续打印变量type，但是在foo函数内部并没有定义变量type，而是在全局环境中和调用foo函数的bar函数中分别定义了变量type，那么这时候的问题来了，你觉得foo函数中打印出来的变量type是bar函数中的，还是全局环境中的呢？

什么是函数作用域和全局作用域？

要解释清楚这个问题，我们需要从作用域的工作原理讲起。

每个函数在执行时都需要查找自己的作用域，我们称为函数作用域，在执行阶段，在执行一个函数时，当该函数需要使用某个变量或者调用了某个函数时，便会优先在该函数作用域中查找相关内容。

我们再来看一段代码：

```
var x = 4
var test
function test_scope() {
    var name = 'foo'
    console.log(name)
    console.log(type)
    console.log(test)
    var type = 'function'
    test = 1
    console.log(x)
}
test_scope()
```

在上面的代码中，我们定义了一个test_scope函数，那么在V8执行test_scope函数的时候，在编译阶段会为test_scope函数创建一个作用域，在test_scope函数中定义的变量和声明的函数都会丢到该作用域中，因为我们在test_scope函数中定了三个变量，那么常见的作用域就包含有这三个变量。

你可以通过Chrome的控制台来直观感受下test_scope函数的作用域，先打开包含这段代码的页面，然后打开开发者工具，接着在test_scope函数中的第二段代码加上断点，然后刷新该页面。当执行到该断点时，V8会暂停整个执行流程，这时候我们就可以通过右边的区域面板来查看当前函数的执行状态。

你可以参考图中右侧的Scope项，然后点击展开该项，这个Local就是当前函数test_scope的作用域。在test_scope函数中定义的变量都包含到了Local中，如变量name、type，另外系统还为我们添加了另外一个隐藏变量this，V8还会默认将隐藏变量this存放到作用域中。

另外你还需要注意下，第一个test1，我并没有采用var等关键字来声明，所以test1并不会出现在test_scope函数的作用域中，而是属于this所指向的对象。（this的工作机制不是本文讨论的重点，不展开介绍。如果你感兴趣，可以在《浏览器工作原理与实践》专栏中《11 | this：从JavaScript执行上下文的视角讲清楚this》这一讲查看。）

那么另一个问题来了，我在test_scope函数使用了变量x，但是在test_scope函数的作用域中，并没有定义变量x，那么V8应该如何获取变量x?

如果在当前函数作用域中没有查找到变量，那么V8会去全局作用域中去查找，这个查找的线路就称为作用域链。

全局作用域和函数作用域类似，也是存放变量和函数的地方，但是它们还是有点不一样：**全局作用域是在V8启动过程中就创建了，且一直保存在内存中不会被销毁的，直至V8退出。 而函数作用域是在执行该函数时创建的，当函数执行结束之后，函数作用域就随之被销毁掉了。**

全局作用域中包含了很多全局变量，比如全局的this值，如果是浏览器，全局作用域中还有window、document、opener等非常多的方法和对象，如果是node环境，那么会有Global、File等内容。

V8启动之后就进入正常的消息循环状态，这时候就可以执行代码了，比如执行到上面那段脚本时，V8会先解析顶层(Top Level)代码，我们可以看到，在顶层代码中定义了变量x，这时候V8就会将变量x添加到全局作用域中。

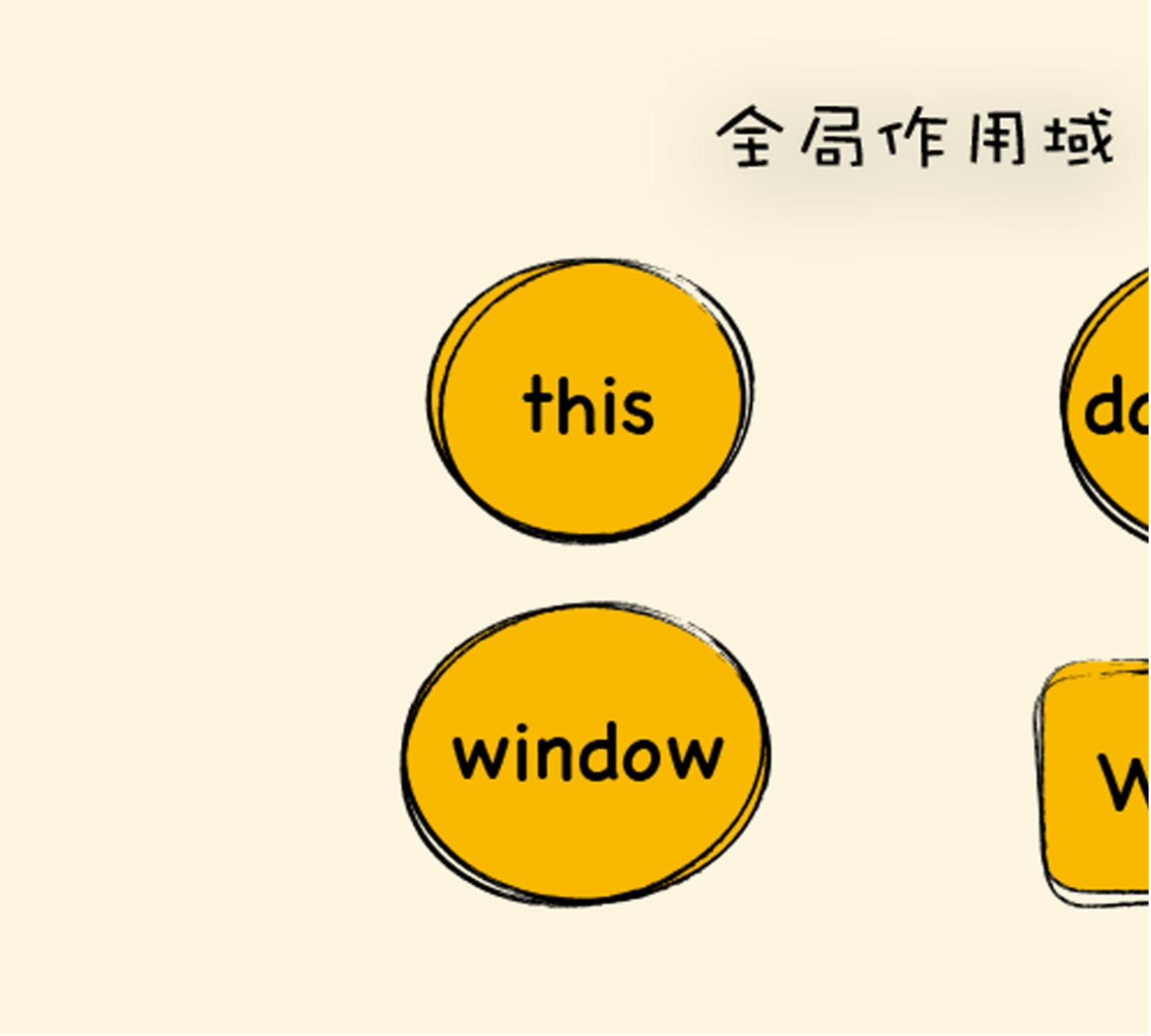
作用域链是怎么工作的？

理解了作用域和作用域链，我们再回过头来看文章开头的那道思考题：“foo函数中打印出来的变量type是bar函数中的呢，还是全局环境中的呢?”我把这段代码复制到下面：

```
var name = '极客时间'
```

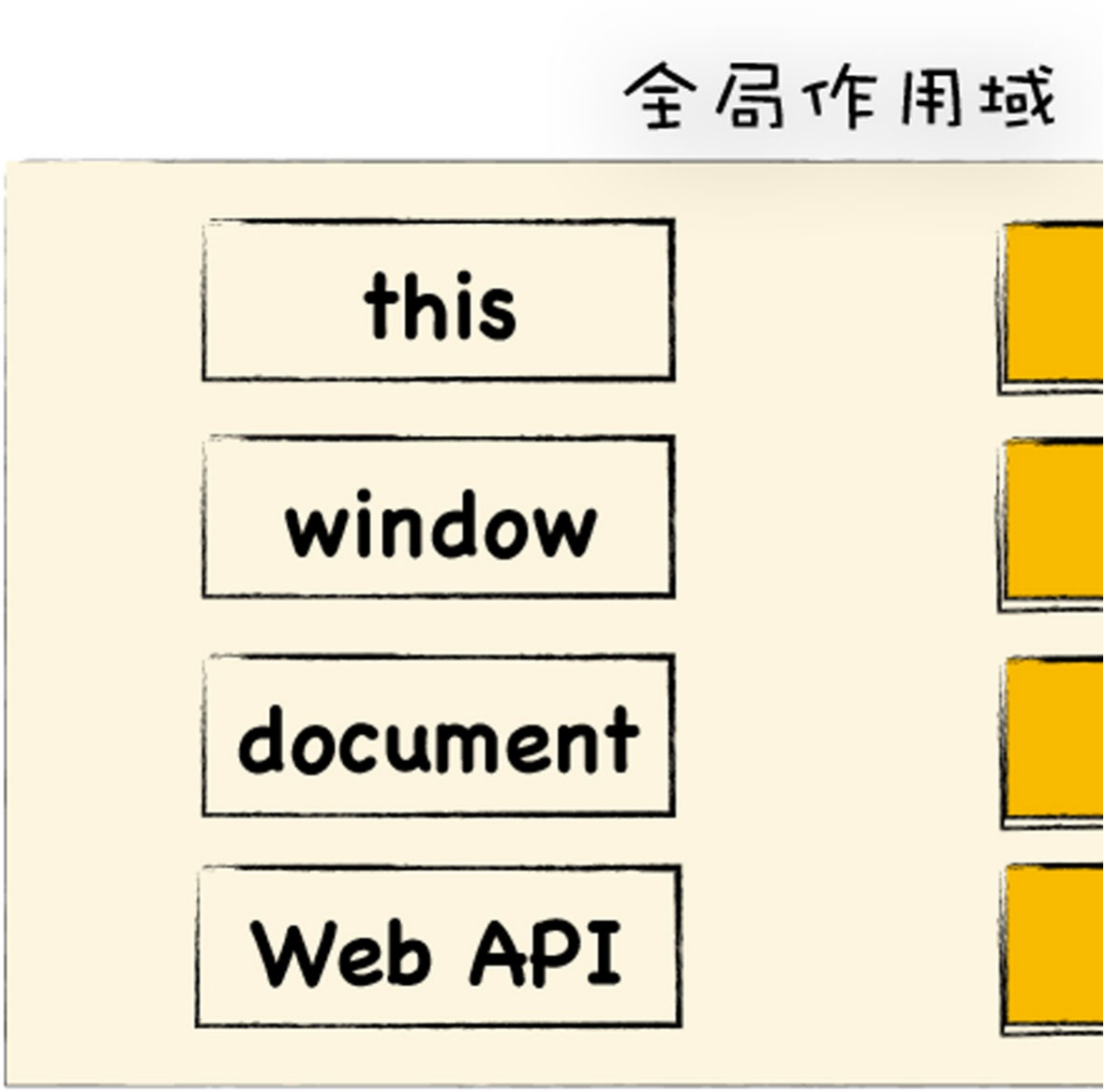
```
var type = 'global'


function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}


function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}
bar()
```

现在，我们结合V8执行这段代码的流程来具体分析下。首先当V8启动时，会创建全局作用域，全局作用域中包括了this、window等变量，还有一些全局的Web API接口，创建的作用域如下图所示：



V8启动之后，消息循环系统便开始工作了，这时候，我输入了这段代码，让其执行。

V8会先编译顶层代码，在编译过程中会将顶层定义的变量和声明的函数都添加到全局作用域中，最终的全局作用域如下图所示：

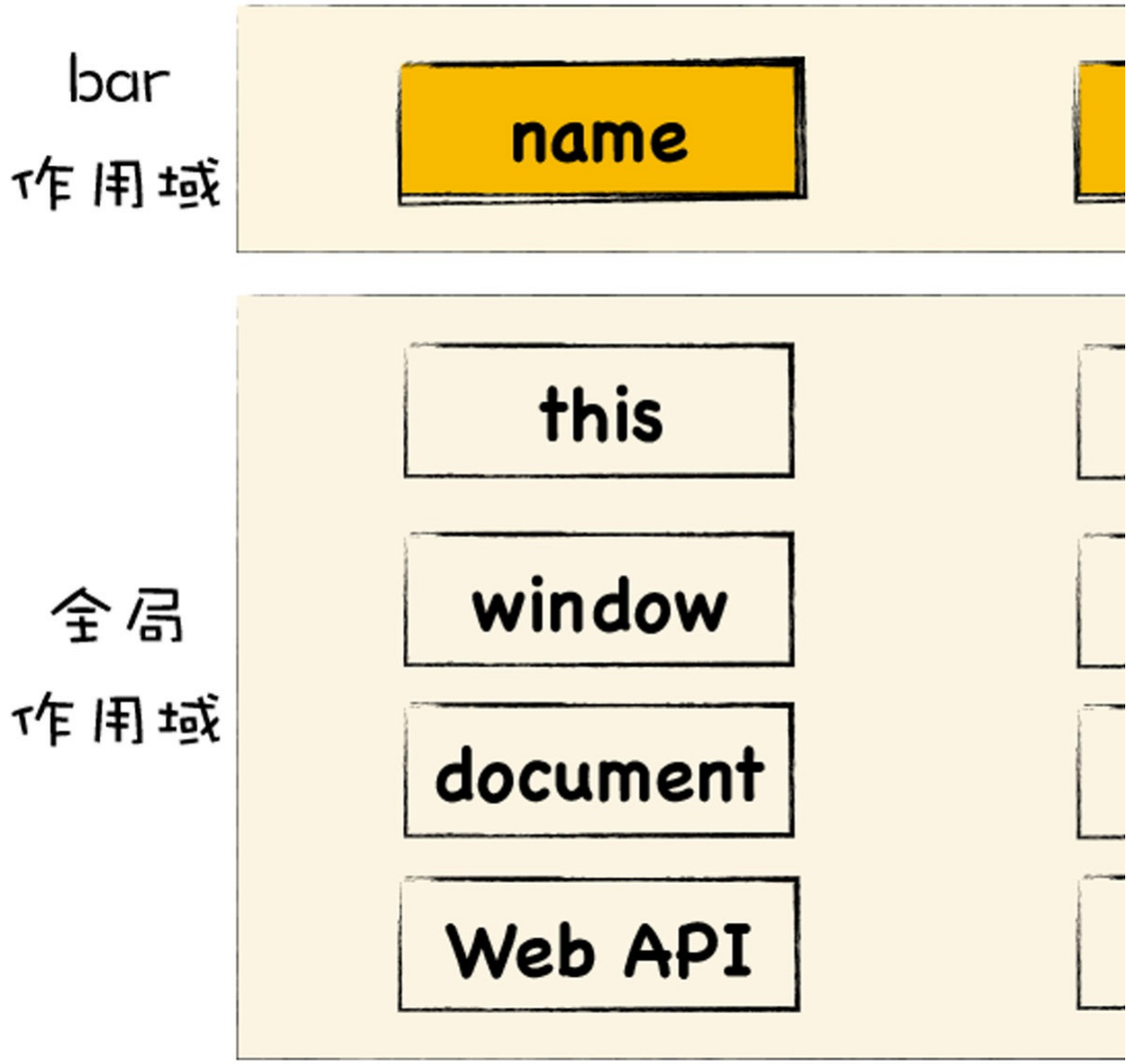


全局作用域创建完成之后，V8便进入了执行状态。前面我们介绍了变量提升，因为变量提升的原因，你可以把上面这段代码分解为如下两个部分：

```
//======解析阶段--实现变量提升=======
var name = undefined
var type = undefined
function foo(){
    var name = 'foo'
    console.log(name)
    console.log(type)
}
function bar(){
    var name = 'bar'
    var type = 'function'
    foo()
}



//====执行阶段========
name = '极客时间'
type = 'global'
bar()
```

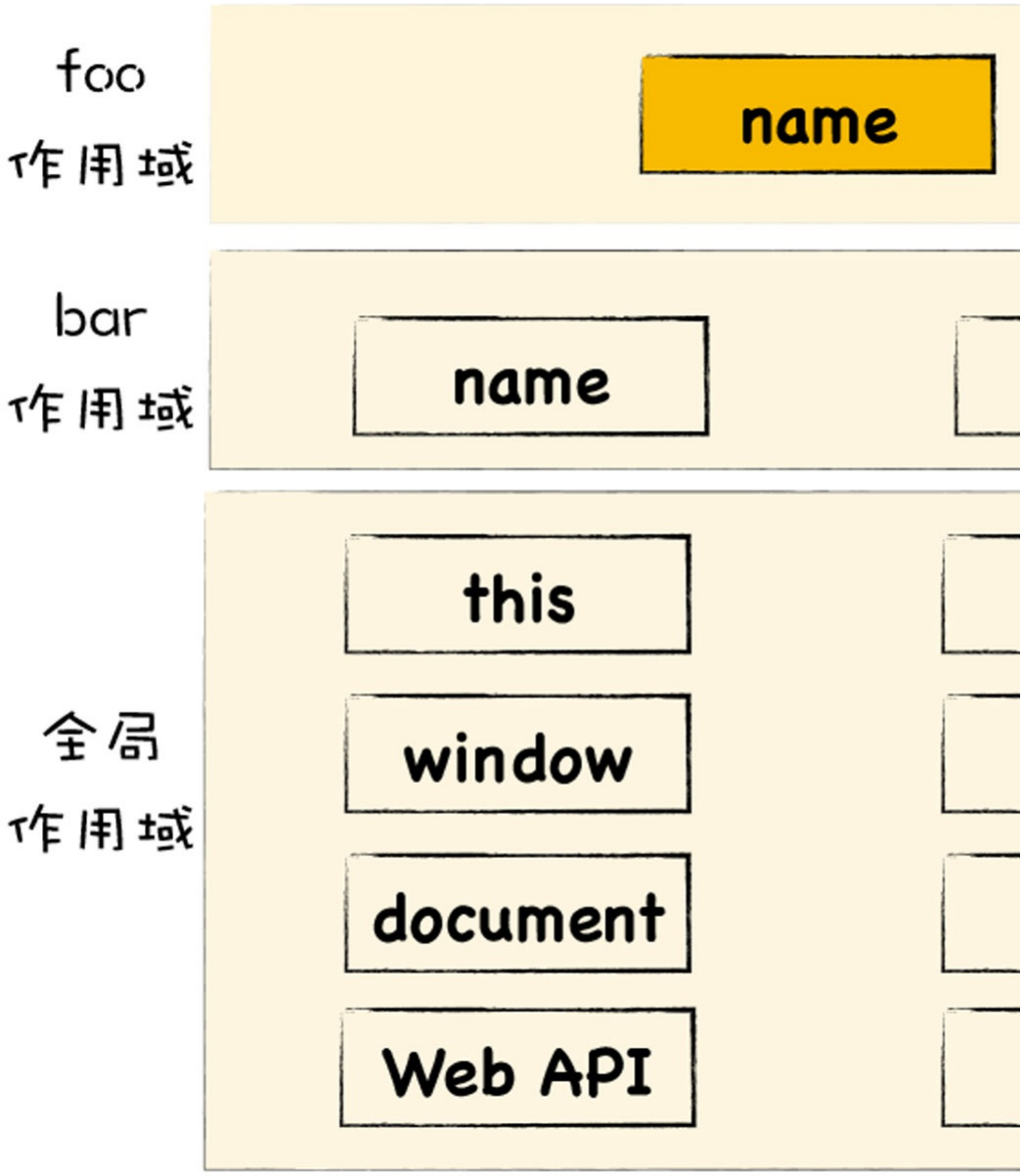
第一部分是在编译过程中完成的，此时全局作用中两个变量的值依然是undefined，然后进入执行阶段；第二部代码就是执行时的顺序，首先全局作用域中的两个变量赋值“极客时间”和“global”，然后就开始执行函数bar的调用了。

当V8执行bar函数的时候，同样需要经历两个阶段：编译和执行。在编译阶段，V8会为bar函数创建函数作用域，最终效果如下所示：



然后进入了bar函数的执行阶段。在bar函数中，只是简单地调用foo函数，因此V8又开始执行foo函数了。

同样，在编译foo函数的过程中，会创建foo函数的作用域，最终创建效果如下图所示：



好了，这时候我们就有了三个作用域了，分别是全局作用域、bar的函数作用域、foo的函数作用域。

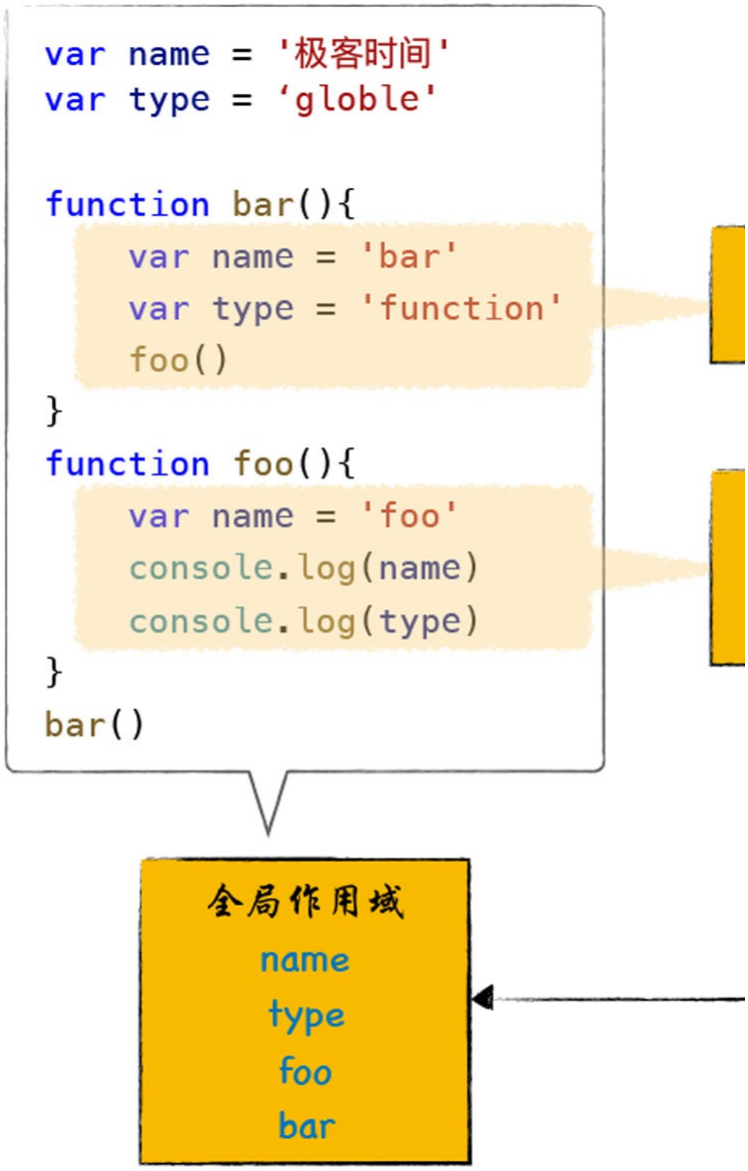
现在我们就可以将刚才提到的问题转换为作用域链的问题了：foo函数查找变量的路径到底是什么？

- 沿着foo函数作用域–>bar函数作用域–>全局作用域;
- 还是，沿着foo函数作用域—>全局作用域？

因为JavaScript是基于词法作用域的，词法作用域就是指，查找作用域的顺序是按照函数定义时的位置来决定的。bar和foo函数的外部代码都是全局代码，所以无论你是在bar函数中查找变量，还是在foo函数中查找变量，其查找顺序都是按照当前函数作用域–>全局作用域这个路径来的。

由于我们代码中的foo函数和bar函数都是在全局下面定义的，所以在foo函数中使用了type，最终打印出来的值就是全局作用域中的type。

你可以参考下面这张图：



另外，我再展开说一些。因为词法作用域是根据函数在代码中的位置来确定的，作用域是在声明函数时就确定好的了，所以我们也将词法作用域称为静态作用域。

和静态作用域相对的是动态作用域，动态作用域并不关心函数和作用域是如何声明以及在何处声明的，只关心它们从**何处调用**。换句话说，作用域链是基于调用栈的，而不是基于函数定义的位置的。（动态作用域不是本文讨论的重点，如果你感兴趣，可以参考《浏览器工作原理与实践》专栏中的《10 | 作用域链和闭包：代码中出现相同的变量，JavaScript引擎是如何选择的？》这一节。）

总结

今天，我们主要解释了一个问题，那就是在一个函数中，如果使用了一个变量，或者调用了另外一个函数，V8将会怎么去查找该变量或者函数。

为了解释清楚这个问题，我们引入了作用域的概念。作用域就是用来存放变量和函数的地方，全局作用域中存放了全局环境中声明的变量和函数，函数作用域中存放了函数中声明的变量和函数。当在某个函数中使用某个变量时，V8就会去这些作用域中查找相关变量。沿着这些作用域查找的路径，我们就称为作用域链。

要了解查找路径，我们需要明白词法作用域，词法作用域是按照代码定义时的位置决定的，而JavaScript所采用的作用域机制就是词法作用域，所以作用域链的路径就是按照词法作用域来实现的。

思考题

我将文章开头那段代码稍微调整了下，foo函数并不是在全局环境中声明的，而是在bar函数中声明的，改造后的代码如下所示：

```
var name = '极客时间'
var type = 'global'
function bar() {
    var type = 'function'
    function foo() {
        console.log(type)
    }
    foo()
}
bar()
```

那么执行这段代码之后，打印出来的内容是什么？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。