07-变量提升：JavaScript代码是按顺序执行的吗？

讲解完宏观视角下的浏览器后，从这篇文章开始，我们就进入下一个新的模块了，这里我会对JavaScript执行原理做深入介绍。

今天在该模块的第一篇文章，我们主要讲解**执行上下文**相关的内容。那为什么先讲执行上下文呢？它这么重要吗？可以这么说，**只有理解了JavaScrip的执行上下文，你才能更好地理解JavaScript语言本身**，比如变量提升、作用域和闭包等。不仅如此，理解执行上下文和调用栈的概念还能助你成为一名更合格的前端开发者。

不过由于我们专栏不是专门讲JavaScript语言的，所以我并不会对JavaScript语法本身做过多介绍。本文主要是从JavaScript的顺序执行讲起，然后**一步步带你了解JavaScript是怎么运行的**。

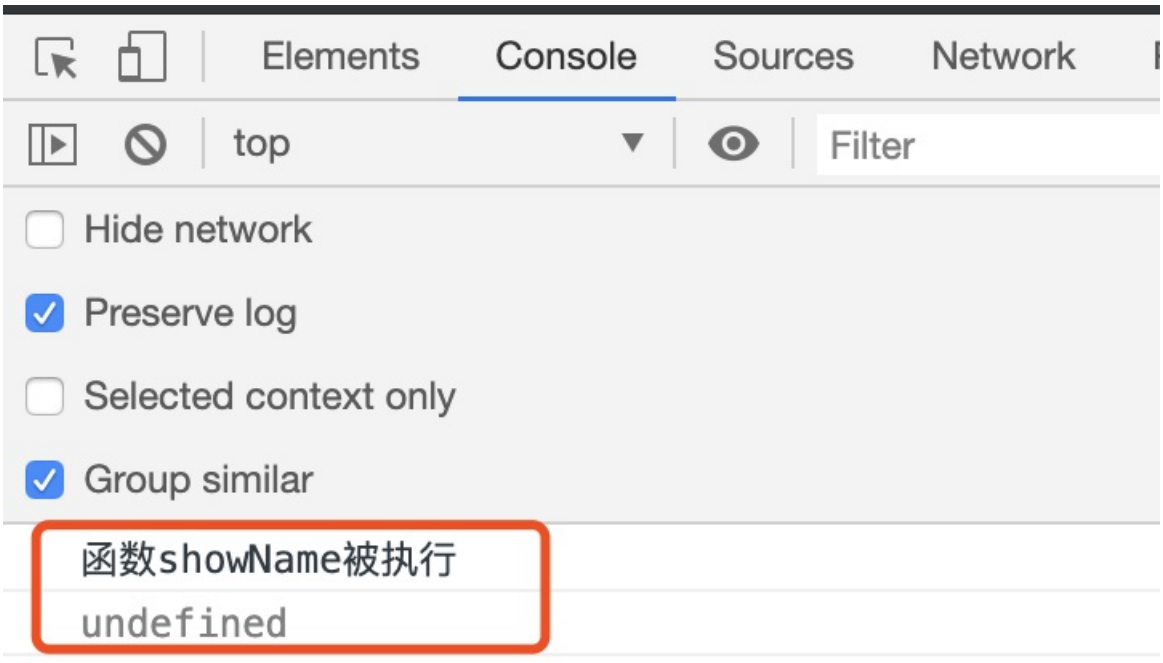
接下来咱们先看段代码，你觉得下面这段代码输出的结果是什么？

```
showName()
console.log(myname)
var myname = '极客时间'
function showName() {
    console.log('函数showName被执行');
}
```

使用过JavaScript开发的程序员应该都知道，JavaScript是按顺序执行的。若按照这个逻辑来理解的话，那么：

- 当执行到第1行的时候，由于函数showName还没有定义，所以执行应该会报错；
- 同样执行第2行的时候，由于变量myname函数也未定义，所以同样也会报错。

然而实际执行结果却并非如此， 如下图：



在声明之前使用函数和变量的执行结果

第1行输出“函数showName被执行”，第2行输出“undefined”，这和前面想象中的顺序执行有点不一样啊！

通过上面的执行结果，你应该已经知道了函数或者变量可以在定义之前使用，那如果使用没有定义的变量或者函数，JavaScript代码还能继续执行吗？为了验证这点，我们可以删除第3行变量myname的定义，如下所示：

```
showName()
console.log(myname)
function showName() {
    console.log('函数showName被执行');
}
```

然后再次执行这段代码时，JavaScript引擎就会报错，结果如下：

使用了未定义的变量——执行报错

从上面两段代码的执行结果来看，我们可以得出如下三个结论。

1. 在执行过程中，若使用了未声明的变量，那么JavaScript执行会报错。
2. 在一个变量定义之前使用它，不会出错，但是该变量的值会为undefined，而不是定义时的值。
3. 在一个函数定义之前使用它，不会出错，且函数能正确执行。

第一个结论很好理解，因为变量没有定义，这样在执行JavaScript代码时，就找不到该变量，所以JavaScript会抛出错误。

但是对于第二个和第三个结论，就挺让人费解的：

- 变量和函数为什么能在其定义之前使用？这似乎表明JavaScript代码并不是一行一行执行的。
- 同样的方式，变量和函数的处理结果为什么不一样？比如上面的执行结果，提前使用的showName函数能打印出来完整结果，但是提前使用的myname变量值却是undefined，而不是定义时使用的“极客时间”这个值。

变量提升（Hoisting）

要解释这两个问题，你就需要先了解下什么是变量提升。

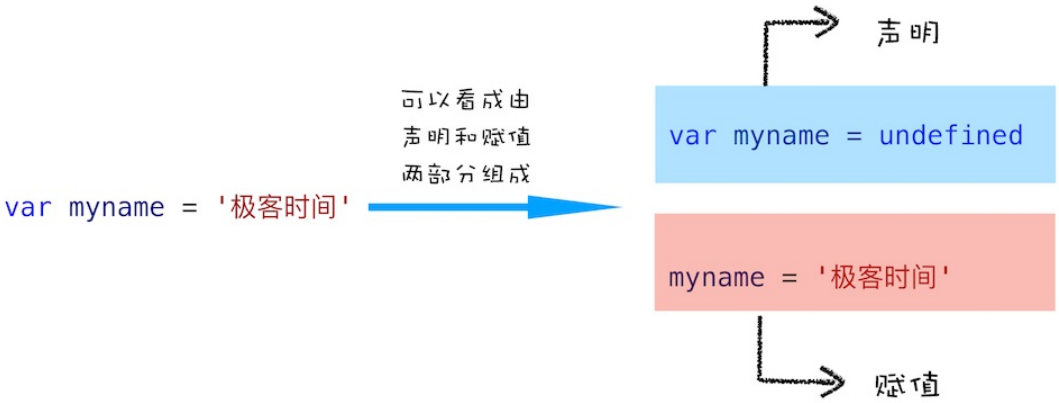
不过在介绍变量提升之前，我们先通过下面这段代码，来看看什么是JavaScript中的**声明**和**赋值**。

```
var myname = '极客时间'
```

这段代码你可以把它看成是两行代码组成的：

```
var myname    //声明部分
myname = '极客时间'  //赋值部分
```

如下图所示：



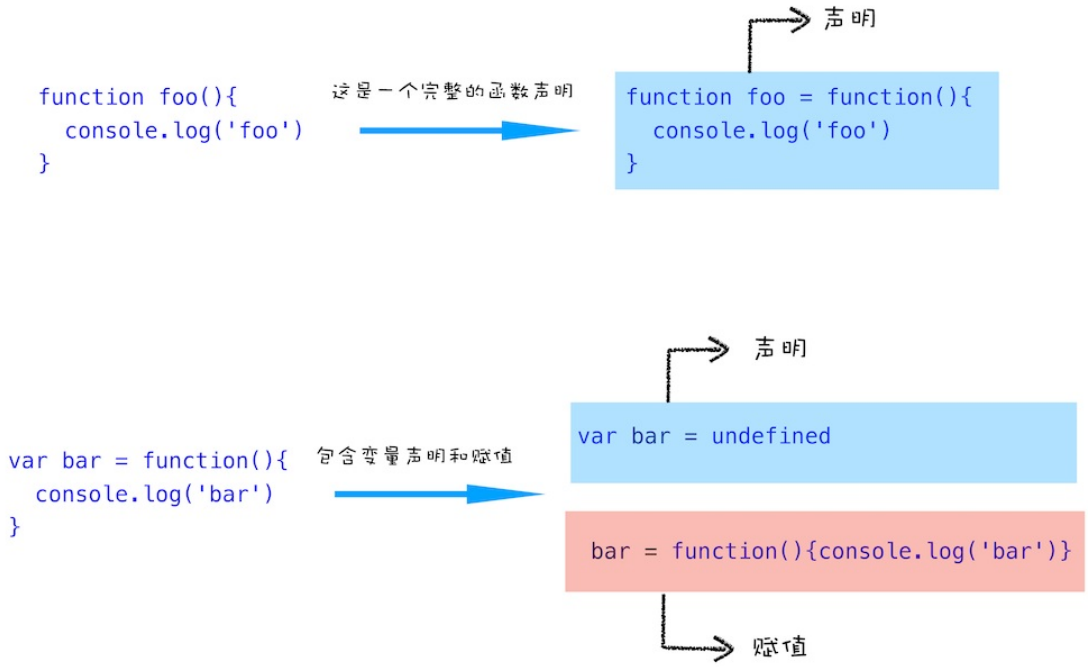
如何理解`var myname = '极客时间'`

上面是**变量**的声明和赋值，那接下来我们再来看看**函数**的声明和赋值，结合下面这段代码：

```
function foo(){
  console.log('foo')
}

var bar = function(){
  console.log('bar')
}
```

第一个函数foo是一个完整的函数声明，也就是说没有涉及到赋值操作；第二个函数是先声明变量bar，再把`function(){console.log('bar')}`赋值给bar。为了直观理解，你可以参考下图：



函数的声明和赋值

好了，理解了声明和赋值操作，那接下来我们就可以聊聊什么是变量提升了。

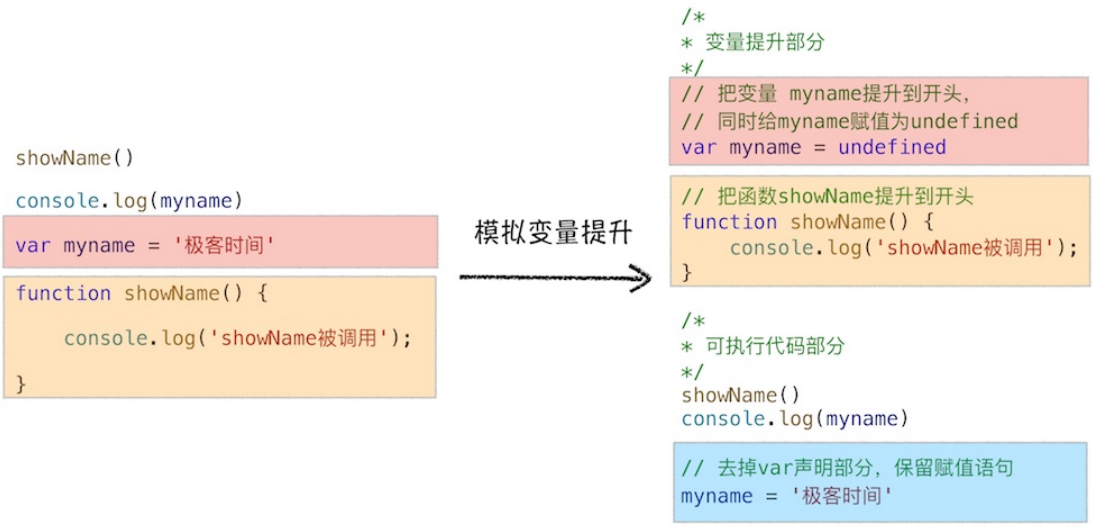
所谓的变量提升，是指在JavaScript代码执行过程中，JavaScript引擎把变量的声明部分和函数的声明部分提升到代码开头的“行为”。变量被提升后，会给变量设置默认值，这个默认值就是我们熟悉的undefined。

下面我们来模拟下实现：

```
/*
* 变量提升部分
*/
// 把变量 myname提升到开头，
// 同时给myname赋值为undefined
var myname = undefined
// 把函数showName提升到开头
function showName() {
    console.log('showName被调用');
}

/*
* 可执行代码部分
*/
showName()
console.log(myname)
// 去掉var声明部分，保留赋值语句
myname = '极客时间'
```

为了模拟变量提升的效果，我们对代码做了以下调整，如下图：



模拟变量提升示意图

从图中可以看出，对原来的代码主要做了两处调整：

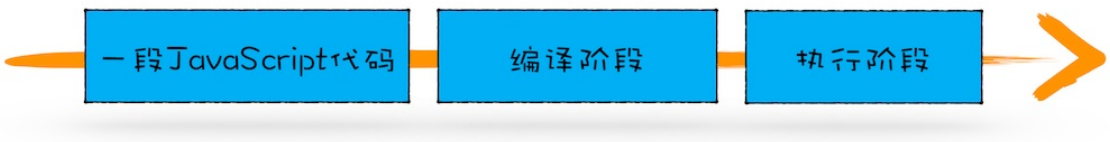
- 第一处是把声明的部分都提升到了代码开头，如变量myname和函数showName，并给变量设置默认值undefined；
- 第二处是移除原本声明的变量和函数，如`var myname = '极客时间'`的语句，移除了var声明，整个移除showName的函数声明。

通过这两步，就可以实现变量提升的效果。你也可以执行这段模拟变量提升的代码，其输出结果和第一段代码应该是完全一样的。

通过这段模拟的变量提升代码，相信你已经明白了可以在定义之前使用变量或者函数的原因——**函数和变量在执行之前都提升到了代码开头**。

JavaScript代码的执行流程

从概念的字面意义上来看，“变量提升”意味着变量和函数的声明会在物理层面移动到代码的最前面，正如我们所模拟的那样。但，这并不准确。**实际上变量和函数声明在代码里的位置是不会改变的，而且是在编译阶段被JavaScript引擎放入内存中**。对，你没听错，一段JavaScript代码在执行之前需要被JavaScript引擎编译，**编译**完成之后，才会进入**执行**阶段。大致流程你可以参考下图：



JavaScript的执行流程图

1. 编译阶段

那么编译阶段和变量提升存在什么关系呢？

为了搞清楚这个问题，我们还是回过头来看上面那段模拟变量提升的代码，为了方便介绍，可以把这段代码分成两部分。

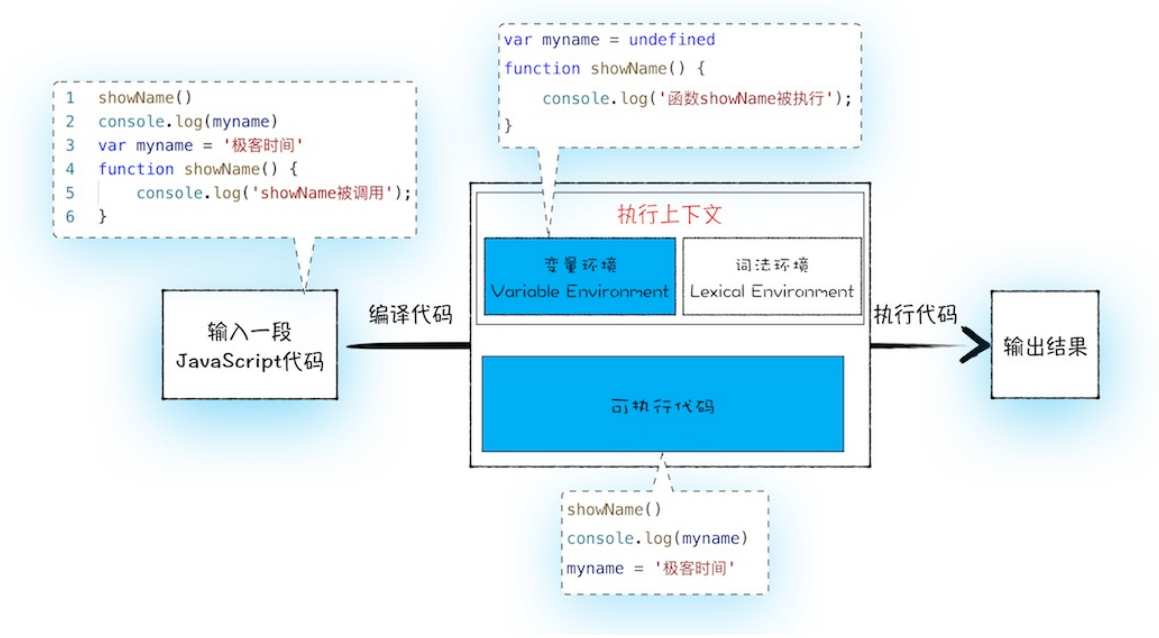
第一部分：变量提升部分的代码。

```
var myname = undefined
function showName() {
    console.log('函数showName被执行');
}
```

第二部分：执行部分的代码。

```
showName()
console.log(myname)
myname = '极客时间'
```

下面我们就可以把JavaScript的执行流程细化，如下图所示：



JavaScript执行流程细化图

从上图可以看出，输入一段代码，经过编译后，会生成两部分内容：**执行上下文（Execution context）和可执行代码**。

执行上下文是JavaScript执行一段代码时的运行环境，比如调用一个函数，就会进入这个函数的执行上下文，确定该函数在执行期间用到的诸如this、变量、对象以及函数等。

关于执行上下文的细节，我会在下一篇文章《08 | 调用栈：为什么JavaScript代码会出现栈溢出？》做详细

介绍，现在你只需要知道，在执行上下文中存在一个**变量环境的对象**（Viriable Environment），该对象中保存了变量提升的内容，比如上面代码中的变量myname和函数showName，都保存在该对象中。

你可以简单地把变量环境对象看成是如下结构：

```
VariableEnvironment:
     myname -> undefined,
     showName ->function : {console.log(myname)
```

了解完变量环境对象的结构后，接下来，我们再结合下面这段代码来分析下是如何生成变量环境对象的。

```
showName()
console.log(myname)
var myname = '极客时间'
function showName() {
    console.log('函数showName被执行');
}
```

我们可以一行一行来分析上述代码：

- 第1行和第2行，由于这两行代码不是声明操作，所以JavaScript引擎不会做任何处理；
- 第3行，由于这行是经过var声明的，因此JavaScript引擎将在环境对象中创建一个名为myname的属性，并使用undefined对其初始化；
- 第4行，JavaScript引擎发现了一个通过function定义的函数，所以它将函数定义存储到堆(HEAP）中，并在环境对象中创建一个showName的属性，然后将该属性值指向堆中函数的位置（不了解堆也没关系，JavaScript的执行堆和执行栈我会在后续文章中介绍）。

这样就生成了变量环境对象。接下来JavaScript引擎会把声明以外的代码编译为字节码，至于字节码的细节，我也会在后面文章中做详细介绍，你可以类比如下的模拟代码：

```
showName()
console.log(myname)
myname = '极客时间'
```

好了，现在有了执行上下文和可执行代码了，那么接下来就到了执行阶段了。

2. 执行阶段

JavaScript引擎开始执行“可执行代码”，按照顺序一行一行地执行。下面我们就来一行一行分析下这个执行过程：

- 当执行到showName函数时，JavaScript引擎便开始在变量环境对象中查找该函数，由于变量环境对象中存在该函数的引用，所以JavaScript引擎便开始执行该函数，并输出“函数showName被执行”结果。
- 接下来打印“myname”信息，JavaScript引擎继续在变量环境对象中查找该对象，由于变量环境存在myname变量，并且其值为undefined，所以这时候就输出undefined。
- 接下来执行第3行，把“极客时间”赋给myname变量，赋值后变量环境中的myname属性值改变为“极客时间”，变量环境如下所示：

```
VariableEnvironment:
     myname -> "极客时间",
     showName ->function : {console.log(myname)
```

好了，以上就是一段代码的编译和执行流程。实际上，编译阶段和执行阶段都是非常复杂的，包括了词法分析、语法解析、代码优化、代码生成等，这些内容我会在《14 | 编译器和解释器：V8是如何执行一段JavaScript代码的？》那节详细介绍，在本篇文章中你只需要知道JavaScript代码经过编译生成了什么内容就可以了。

代码中出现相同的变量或者函数怎么办？

现在你已经知道了，在执行一段JavaScript代码之前，会编译代码，并将代码中的函数和变量保存到执行上下文的变量环境中，那么如果代码中出现了重名的函数或者变量，JavaScript引擎会如何处理？

我们先看下面这样一段代码：

```
function showName() {
    console.log('极客邦');
}
showName();
function showName() {
    console.log('极客时间');
}
showName();
```

在上面代码中，我们先定义了一个showName的函数，该函数打印出来“极客邦”；然后调用showName，并定义了一个showName函数，这个showName函数打印出来的是“极客时间”；最后接着继续调用showName。那么你能分析出来这两次调用打印出来的值是什么吗？

我们来分析下其完整执行流程：

- **首先是编译阶段**。遇到了第一个showName函数，会将该函数体存放到变量环境中。接下来是第二个showName函数，继续存放至变量环境中，但是变量环境中已经存在一个showName函数了，此时，**第二个showName函数会将第一个showName函数覆盖掉**。这样变量环境中就只存在第二个showName函数了。
- **接下来是执行阶段**。先执行第一个showName函数，但由于是从变量环境中查找showName函数，而变

量环境中只保存了第二个showName函数，所以最终调用的是第二个函数，打印的内容是“极客时间”。第二次执行showName函数也是走同样的流程，所以输出的结果也是“极客时间”。

综上所述，**一段代码如果定义了两个相同名字的函数，那么最终生效的是最后一个函数**。

总结

好了，今天就到这里，下面我来简单总结下今天的主要内容：

- JavaScript代码执行过程中，需要先做**变量提升**，而之所以需要实现变量提升，是因为JavaScript代码在执行之前需要先**编译**。
- 在**编译阶段**，变量和函数会被存放到**变量环境**中，变量的默认值会被设置为undefined；在代码**执行阶段**，JavaScript引擎会从变量环境中去查找自定义的变量和函数。
- 如果在编译阶段，存在两个相同的函数，那么最终存放在变量环境中的是最后定义的那个，这是因为后定义的会覆盖掉之前定义的。

以上就是今天所讲的主要内容，当然，学习这些内容并不是让你掌握一些JavaScript小技巧，其主要目的是让你清楚JavaScript的执行机制：**先编译，再执行**。

如果你了解了JavaScript执行流程，那么在编写代码时，你就能避开一些陷阱；在分析代码过程中，也能通过分析JavaScript的执行过程来定位问题。

思考时间

最后，看下面这段代码：

```
showName()
var showName = function() {
    console.log(2)
}
function showName() {
    console.log(1)
}
```

你能按照JavaScript的执行流程，来分析最终输出结果吗？

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

精选留言：

- 万字辈儿老小 2019-08-20 02:04:58
  沙发

- wuqilv 2019-08-20 01:39:38
  输出 1

  提升部分代码
  var showName = undefined
  function showName() {
  console.log(1)
  }

  执行部分代码
  showName()
  showName = function() {
  console.log(2)
  }

- leitong 2019-08-20 00:51:42
  一、
  showName()
  var showName = function() {
  console.log(2)
  }
  function showName() {
  console.log(1)
  }
  编译阶段，第一个showName存入变量环境中，自动赋值undefined
  第二个showName函数体也存入了变量环境中，但是是一个完整的函数声明赋值
  执行阶段，Javacript引擎从变量环境中查找到showName函数体直接执行

输出结果：1

- leitong 2019-08-20 00:45:02
  二、
  showName()
  function showName(){
  console.log(1)
  }
  var showName=function(){
  console.log(2)
  }
  showName()
  编译阶段，showName函数体存入变量环境
  showName变量存入变量环境，赋值undefined
  执行阶段，第一个showName()查找到函数体直接执行
  输出结果：1
  执行到第二个showName()时，showName变量已经赋值了function(){console.log(2)}
  输出结果：2