

19 | JavaScript执行（四）：try里面放return，finally还会执行吗？

2019-03-02 winter

重学前端

[进入课程 >](#)



讲述：winter

时长 11:22 大小 10.42M



你好，我是 winter。

在前面几篇文章中，我们已经了解了关于执行上下文、作用域、闭包之间的关系。

今天，我们则要说一说更为细节的部分：语句。


语句是任何编程语言的基础结构，与 JavaScript 对象一样，JavaScript 语句同样具有“看起来很像其它语言，但是其实一点都不一样”的特点。

我们比较常见的语句包括变量声明、表达式、条件、循环等，这些都是大家非常熟悉的东西，对于它们的行为，我在这里就不赘述了。

为了了解 JavaScript 语句有哪些特别之处，首先我们要看一个不太常见的例子，我会通过这个例子，来向你介绍 JavaScript 语句执行机制涉及的一种基础类型：Completion 类型。

Completion 类型

我们来看一个例子。在函数 foo 中，使用了一组 try 语句。我们可以先来做一个小实验，在 try 中有 return 语句，finally 中的内容还会执行吗？我们来看一段代码。


 复制代码

```
1 function foo(){
2   try{
3     return 0;
4   } catch(err) {
5
6   } finally {
7     console.log("a")
8   }
9 }
10
11 console.log(foo());
```

通过实际试验，我们可以看到，finally 确实执行了，而且 return 语句也生效了，foo() 返回了结果 0。

虽然 return 执行了，但是函数并没有立即返回，又执行了 finally 里面的内容，这样的行为违背了很多人的直觉。

如果在这个例子中，我们在 finally 中加入 return 语句，会发生什么呢？

 复制代码

```
1 function foo(){
2   try{
3     return 0;
4   } catch(err) {
5
6   } finally {
7     return 1;
8   }
9 }
```

```
10  
11 console.log(foo());
```

通过实际执行，我们看到，finally 中的 return “覆盖”了 try 中的 return。在一个函数中执行了两次 return，这已经超出了很多人的常识，也是其它语言中不会出现的一种行为。

面对如此怪异的行为，我们当然可以把它作为一个孤立的知识去记忆，但是实际上，这背后有一套机制在运作。

这一机制的基础正是 JavaScript 语句执行的完成状态，我们用一个标准类型来表示：Completion Record（我在类型一节提到过，Completion Record 用于描述异常、跳出等语句执行过程）。

Completion Record 表示一个语句执行完之后的结果，它有三个字段：

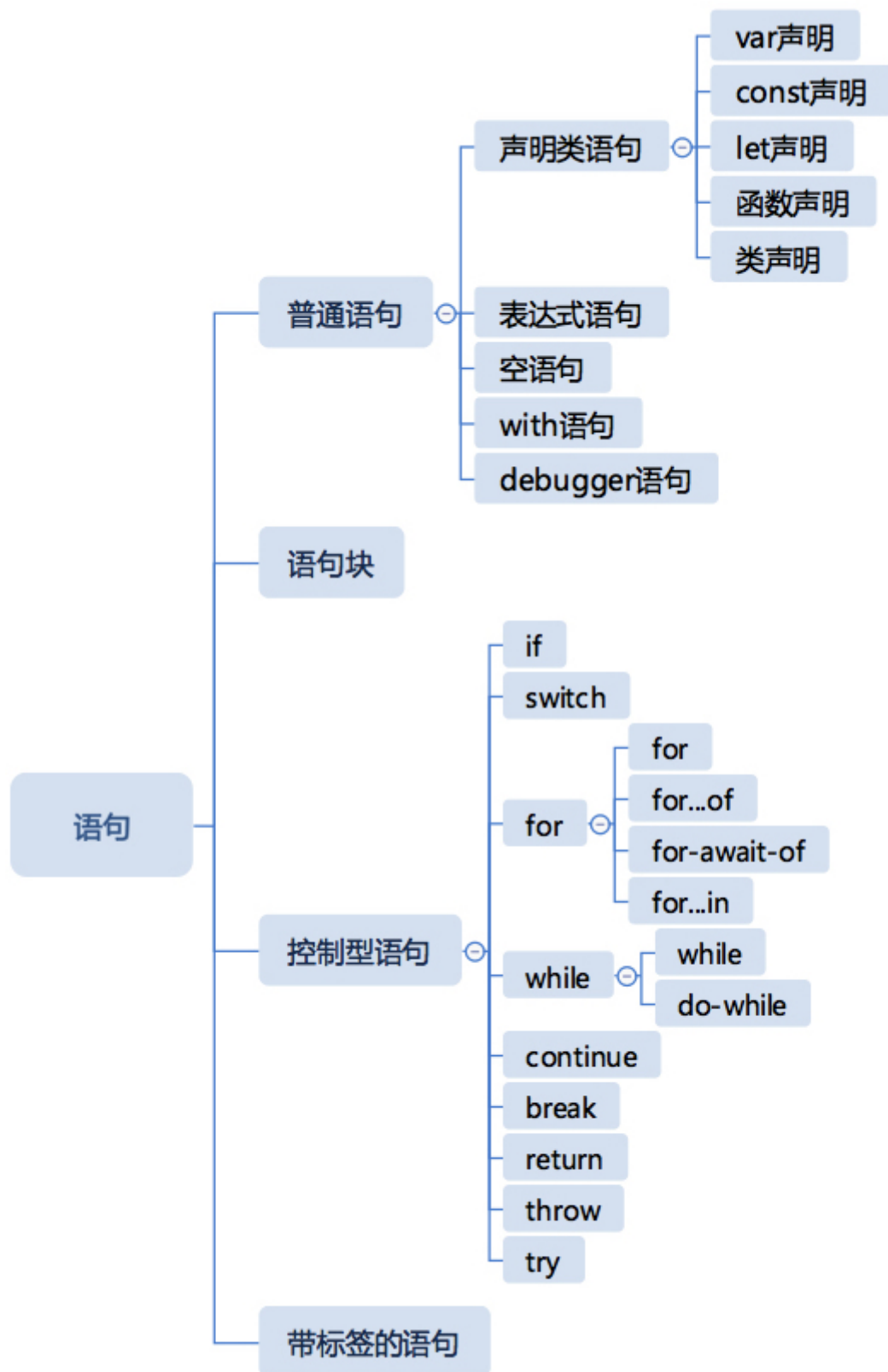
[[type]] 表示完成的类型，有 break continue return throw 和 normal 几种类型；

[[value]] 表示语句的返回值，如果语句没有，则是 empty；

[[target]] 表示语句的目标，通常是一个 JavaScript 标签（标签在后文会有介绍）。

JavaScript 正是依靠语句的 Completion Record 类型，方才可以在语句的复杂嵌套结构中，实现各种控制。接下来我们要来了解一下 JavaScript 使用 Completion Record 类型，控制语句执行的过程。

首先我们来看看语句有几种分类。



普通的语句

在 JavaScript 中，我们把不带控制能力的语句称为普通语句。普通语句有下面几种。

声明类语句

var 声明

const 声明

let 声明

函数声明

类声明

表达式语句

空语句

debugger 语句

这些语句在执行时，从前到后顺次执行（我们这里先忽略 var 和函数声明的预处理机制），没有任何分支或者重复执行逻辑。

普通语句执行后，会得到 [[type]] 为 normal 的 Completion Record，JavaScript 引擎遇到这样的 Completion Record，会继续执行下一条语句。

这些语句中，只有表达式语句会产生 [[value]]，当然，从引擎控制的角度，这个 value 并没有什么用处。

如果你经常使用 chrome 自带的调试工具，可以知道，输入一个表达式，在控制台可以得到结果，但是在前面加上 var，就变成了 undefined。

```
> var i = 1
< undefined
> i = 1
< 1
>
```

Chrome 控制台显示的正是语句的 Completion Record 的 [[value]]。

语句块

介绍完了普通语句，我们再来介绍一个比较特殊的语句：语句块。

语句块就是拿大括号括起来的一组语句，它是一种语句的复合结构，可以嵌套。


语句块本身并不复杂，我们需要注意的是语句块内部的语句的 Completion Record 的 `[[type]]` 如果不为 `normal`，会打断语句块后续的语句执行。

比如我们考虑，一个 `[[type]]` 为 `return` 的语句，出现在一个语句块中的情况。

从语句的这个 `type` 中，我们大概可以猜到它由哪些特定语句产生，我们就来说说最开始的例子中的 `return`。

`return` 语句可能产生 `return` 或者 `throw` 类型的 Completion Record。我们来看一个例子。


先给出一个内部为普通语句的语句块：

 复制代码

```
1 {  
2   var i = 1; // normal, empty, empty  
3   i ++; // normal, 1, empty  
4   console.log(i) //normal, undefined, empty  
5 } // normal, undefined, empty
```

在每一行的注释中，我给出了语句的 Completion Record。

我们看到，在一个 `block` 中，如果每一个语句都是 `normal` 类型，那么它会顺次执行。接下来我们加入 `return` 试试看。

 复制代码

```
1 {  
2   var i = 1; // normal, empty, empty  
3   return i; // return, 1, empty  
4   i ++;  
5   console.log(i)
```

```
6 } // return, 1, empty
```

但是假如我们在 block 中插入了一条 return 语句，产生了一个非 normal 记录，那么整个 block 会成为非 normal。这个结构就保证了非 normal 的完成类型可以穿透复杂的语句嵌套结构，产生控制效果。

接下来我们就具体讲讲控制类语句。

控制型语句

控制型语句带有 if、switch 关键字，它们会对不同类型的 Completion Record 产生反应。

控制类语句分成两部分，一类是对其内部造成影响，如 if、switch、while/for、try。另一类是对外部造成影响如 break、continue、return、throw，这两类语句的配合，会产生控制代码执行顺序和执行逻辑的效果，这也是我们编程的主要工作。

一般来说，for/while - break/continue 和 try - throw 这样比较符合逻辑的组合，是大家比较熟悉的，但是，实际上，我们需要控制语句跟 break、continue、return、throw 四种类型与控制语句两两组合产生的效果。

	break	continue	return	throw
if	穿透	穿透	穿透	穿透
switch	消费	穿透	穿透	穿透
for/while	消费	消费	穿透	穿透
function	报错	报错	消费	穿透
try	特殊处理	特殊处理	特殊处理	消费
catch	特殊处理	特殊处理	特殊处理	穿透
finally	特殊处理	特殊处理	特殊处理	穿透

通过这个表，我们不难发现知识的盲点，也就是我们最初的 case 中的 try 和 return 的组合了。


因为 finally 中的内容必须保证执行，所以 try/catch 执行完毕，即使得到的结果是非 normal 型的完成记录，也必须要执行 finally。

而当 finally 执行也得到了非 normal 记录，则会使 finally 中的记录作为整个 try 结构的结果。

带标签的语句

前文我重点讲了 type 在语句控制中的作用，接下来我们重点来讲一下最后一个字段：target，这涉及了 JavaScript 中的一个语法，带标签的语句。

实际上，任何 JavaScript 语句是可以加标签的，在语句前加冒号即可：

 复制代码

```
1    firstStatement: var i = 1;
```

大部分时候，这个东西类似于注释，没有任何用处。唯一有作用的时候是：与完成记录类型中的 target 相配合，用于跳出多层循环。

 复制代码

```
1    outer: while(true) {
2        inner: while(true) {
3            break outer;
4        }
5    }
6    console.log("finished")
```

break/continue 语句如果后跟了关键字，会产生带 target 的完成记录。一旦完成记录带了 target，那么只有拥有对应 label 的循环语句会消费它。

结语

我们以 Completion Record 类型为线索，为你讲解了 JavaScript 语句执行的原理。

因为 JavaScript 语句存在着嵌套关系，所以执行过程实际上主要在一个树形结构上进行，树形结构的每一个节点执行后产生 Completion Record，根据语句的结构和 Completion Record，JavaScript 实现了各种分支和跳出逻辑。

你遇到哪些语句中的执行的实际效果，是跟你想象的有所出入呢，你可以给我留言，我们一起讨论。



重学前端

每天 10 分钟，重构你的前端知识体系

winter 程劭非
前手机淘宝前端负责人



新版升级：点击「👤 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 18 | JavaScript 执行（三）：你知道现在有多少种函数吗？

下一篇 20 | CSS 选择器：如何选中svg里的a元素？

精选留言 (25)

写留言



有铭

2019-03-02

45

很感兴趣这些高级特性的知识老师是从哪里学到的，我翻过js高级编程那本书都没讲到过这些



Rushan-Ch...

2019-03-02

👍 35

请问老师，表格中的“穿透”和“消费”是什么意思？

展开 ▾



AICC

2019-03-02

👍 21

3楼你好，我的理解是，消费指对应的代码被有效的执行了，穿透指对应代码被跳过了，也就是对应控制的语句体被有效执行比如try catch,当try中出现了throw,能被有效捕获进而执行catch，这在我理解就是try被消费执行了，而当catch中还有throw时，由于catch不具备处理throw的能力，于是catch被中断跳出，也就是作者所说的穿透，希望能帮到你

展开 ▾



周小成

2019-03-12

👍 14

穿透和消费，报错应该是连贯的，“穿透”就是指不在当前这一层处理，向外逐层寻找可以“消费”的那一层，直到最后都没找到就报错，比如：function里面有while, while里面有switch, switch里面又有continue，按图表来看，switch-continue应该是穿透，向上层寻找消费，碰到while-continue,那就是消费，再如switch里面是return, switch-return穿透，向上层while-return穿透，最后function-return是消费。

展开 ▾



Dream.

2019-03-03

👍 7

第一次看见『消费』与『穿透』这样的描述。

这两个词的来源自哪里呢？

结合表格中的控制语句组合使用得到的结果来看，我的理解是...

展开 ▾





加利率的钟...

2019-04-17



```
``javascript
function test(){
  if(true){
    console.log("111");
    break;...
```

展开 ▾



夜空中最亮...

2019-03-02



老师，我昨天成功的把您的课推销出去了一份，哈哈😁高兴

展开 ▾



火云邪神00...

2019-03-04



老师在前面讲过，穿透就是去上一层的作用域或者控制语句找可以消费break，continue的执行环境，消费就是在这一层就执行了这个break或者continue

展开 ▾



Brigand

2019-03-04



Completion 类型是个神马鬼？

展开 ▾



K4SHIFZ

2019-03-29



老师，请出一份ES标准解读。带着我们学一次。必买！

展开 ▾



靠人品去赢

2019-03-20



这个把completion称作一个类型，感觉有点怪，首先这个不是我们自己去定义的，这个是因为我们执行语句都会有这个东西。小白看到这个“类型”会往前找，发现没这个类型，之前掌握的的语言系统也没有相关的类型，结果就是“我擦，这是啥，ES6的新特性

吗？”。关于这个，这是我在MDN上找的相关资料，希望大家指点一下（看run-to-completion这部分）<https://developer.mozilla.org/en-...>

展开 ▾



守候

2019-03-14

👍 1

学到了

展开 ▾



阿成

2019-03-02

👍 1

涨姿势啦

不过，从来没用过label...

甚至第一次知道js里没有goto...

展开 ▾



InfoQ_1ab8...

2019-05-29

👍

标准 <https://tc39.github.io/ecma262/>

展开 ▾



毛哥来了

2019-05-10

👍

比如

```
"test";var a = 1;
```

这一句在console中返回的值是"test",why?

展开 ▾



Geek_c11e9...

2019-05-06

👍

遇到过一些异步的情况，有些理解不了

展开 ▾



kovar

2019-04-17



搜了一下，Completion Record好像是ES2020里面的东西，老师果然够前卫，虽然我现在还看不懂就是了<https://tc39.github.io/ecma262/#sec-runtime-semantics>

展开 ∨



Jacky

2019-04-16



try finally 那个跟Java不是一样的吗

展开 ∨



张驰Terry

2019-04-02



这个章节的知识令人茅塞顿开

展开 ∨



Marvin

2019-04-02



太强了，太强了，越看发现自己越弱鸡~

展开 ∨