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。

```
function foo(){
  try{
    return 0;
  } catch(err) {

  } finally {
    console.log("a")
  }
}

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？

```
function foo(){
  try{
    return 0;
  } catch(err) {

  } finally {
    return 1;
  }
}

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。

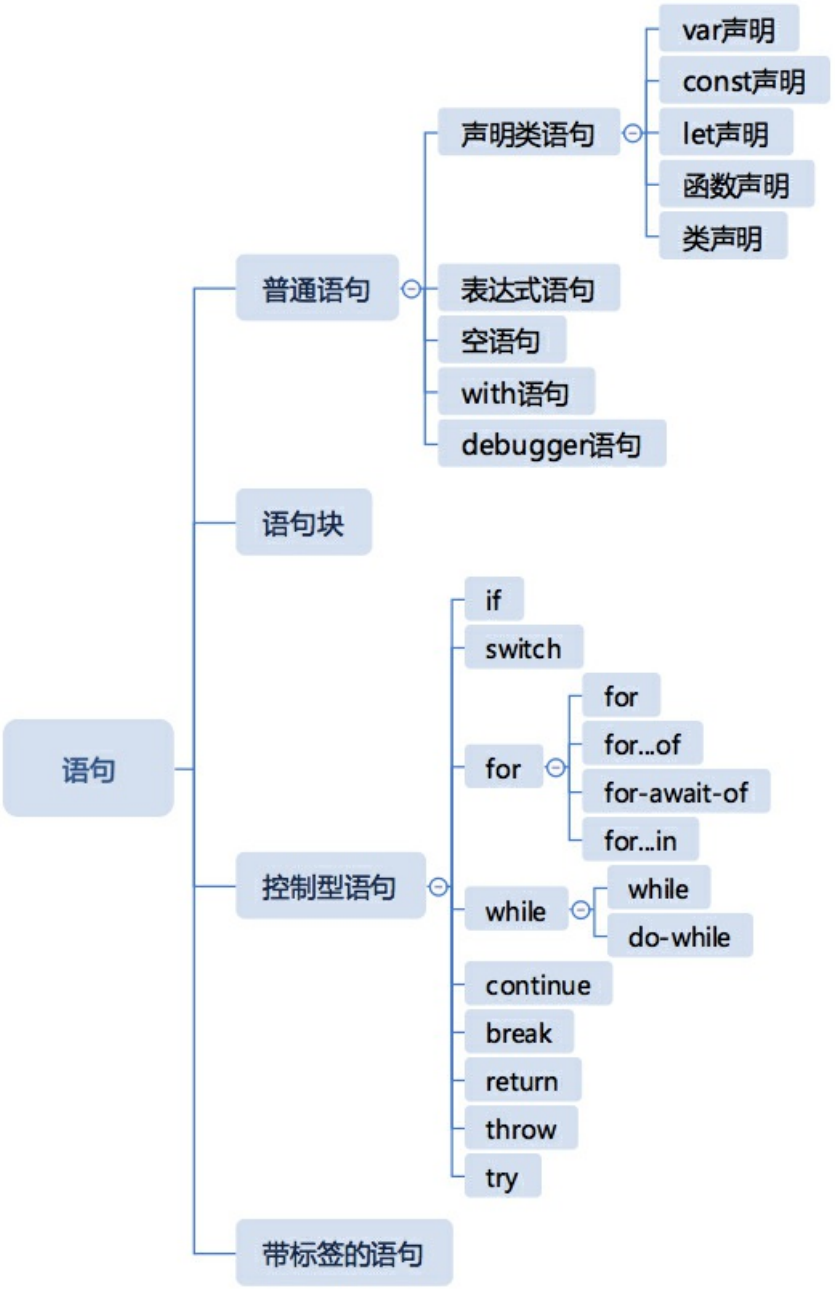
这一机制的基础正是JavaScript语句执行的完成状态，我们用一个标准类型来表示：Completion Record（我在类型一节提到过，Completion Record用于描述异常、跳出等语句执行过程）。

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。

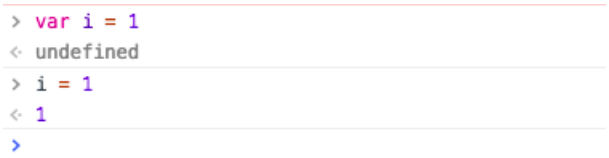
- 声明类语句
  - var声明
  - const声明
  - let声明
  - 函数声明
  - 类声明
- 表达式语句
- 空语句
- debugger语句

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。



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：

```
{
  var i = 1; // normal, empty, empty
  i ++; // normal, 1, empty
  console.log(i) //normal, undefined, empty
} // 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。

```
{
  var i = 1; // normal, empty, empty
  return i; // return, 1, empty
  i ++;
  console.log(i)
} // 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：

```
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。

```
outer: while(true) {
  inner: while(true) {
      break outer;
  }
}
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。

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。

```
function foo(){
  try{
    return 0;
  } catch(err) {

  } finally {
    console.log("a")
  }
}

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？

```
function foo(){
  try{
    return 0;
  } catch(err) {

  } finally {
    return 1;
  }
}

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。

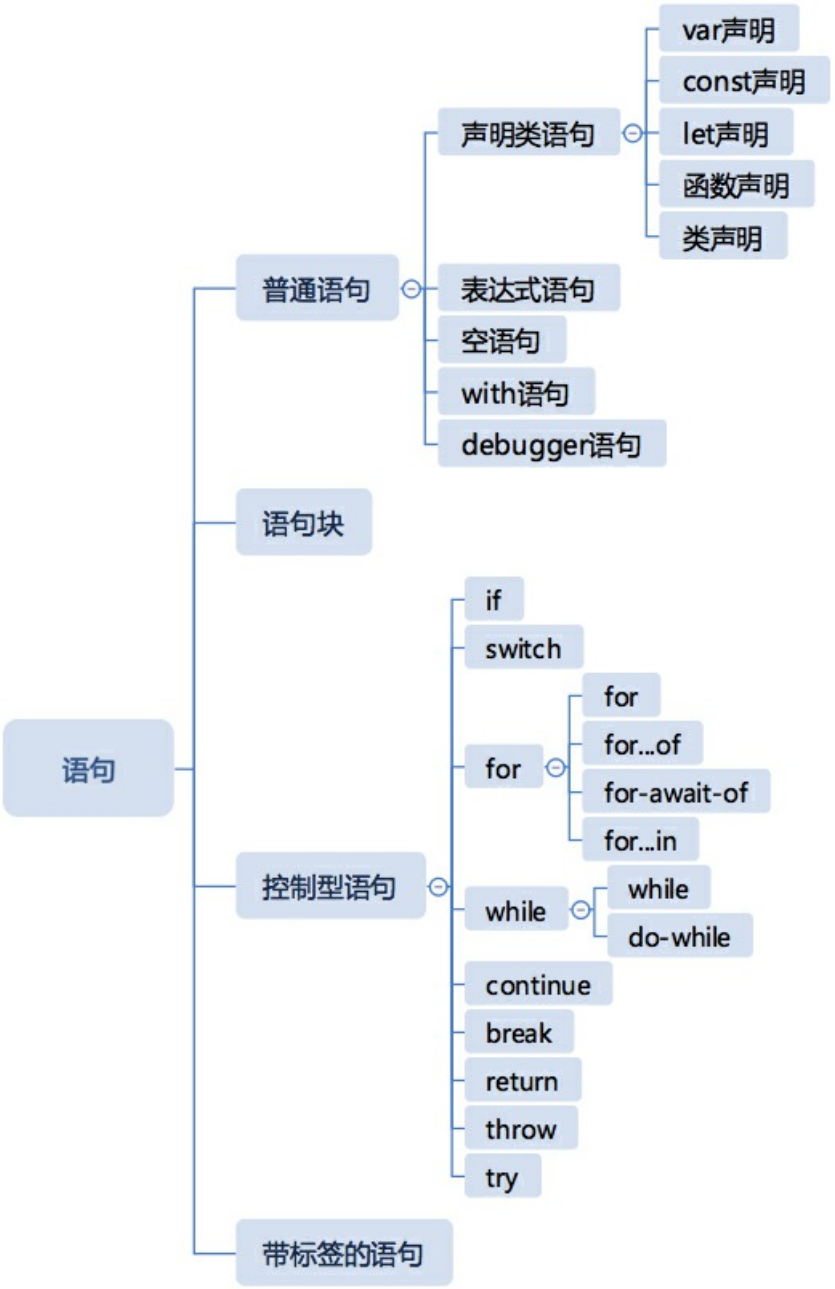
这一机制的基础正是JavaScript语句执行的完成状态，我们用一个标准类型来表示：Completion Record（我在类型一节提到过，Completion Record用于描述异常、跳出等语句执行过程）。

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。

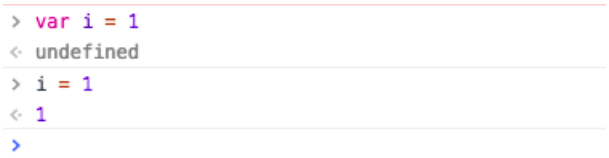
- 声明类语句
  - var声明
  - const声明
  - let声明
  - 函数声明
  - 类声明
- 表达式语句
- 空语句
- debugger语句

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。



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：

```
{
  var i = 1; // normal, empty, empty
  i ++; // normal, 1, empty
  console.log(i) //normal, undefined, empty
} // 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。

```
{
  var i = 1; // normal, empty, empty
  return i; // return, 1, empty
  i ++;
  console.log(i)
} // 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：

```
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。

```
outer: while(true) {
  inner: while(true) {
      break outer;
  }
}
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。

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。

```
function foo(){
  try{
    return 0;
  } catch(err) {

  } finally {
    console.log("a")
  }
}

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？

```
function foo(){
  try{
    return 0;
  } catch(err) {

  } finally {
    return 1;
  }
}

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。

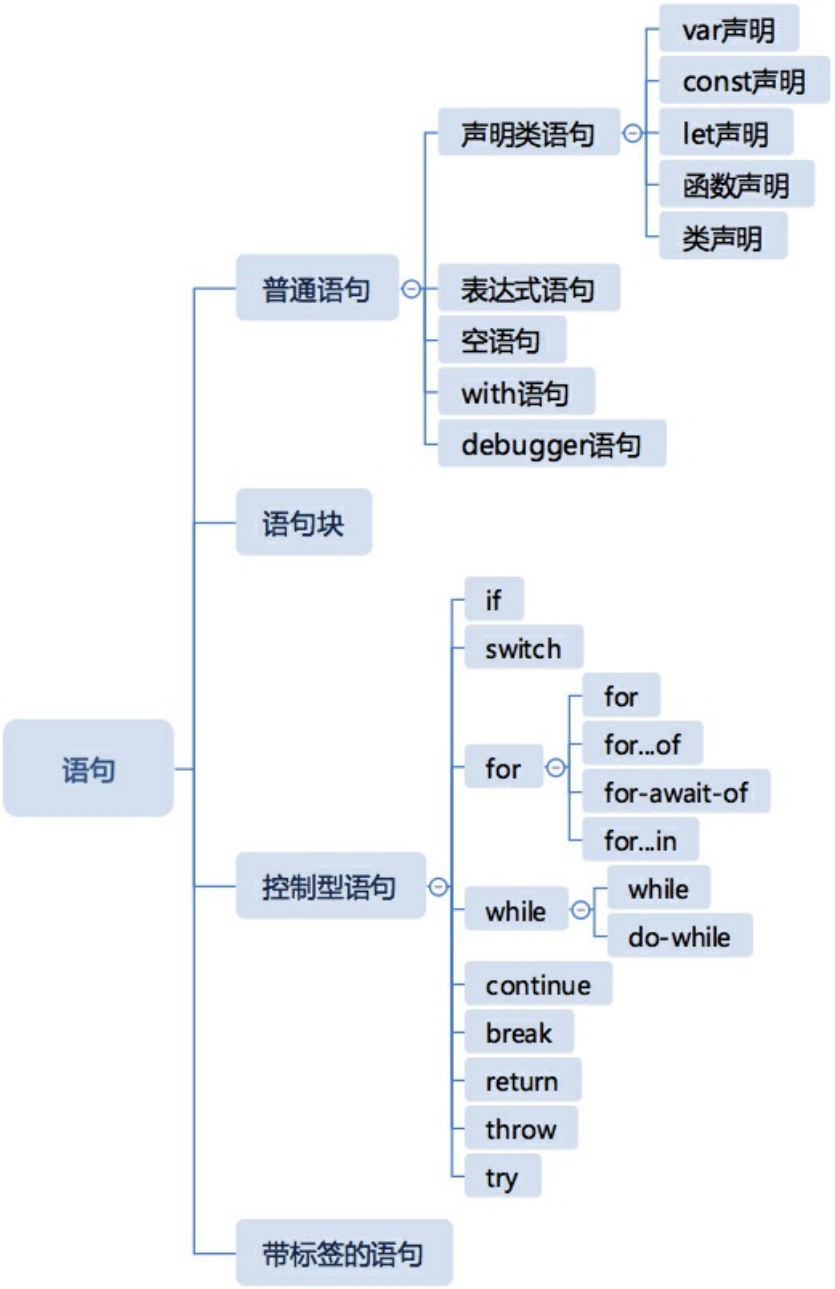
这一机制的基础正是JavaScript语句执行的完成状态，我们用一个标准类型来表示：Completion Record（我在类型一节提到过，Completion Record用于描述异常、跳出等语句执行过程）。

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。

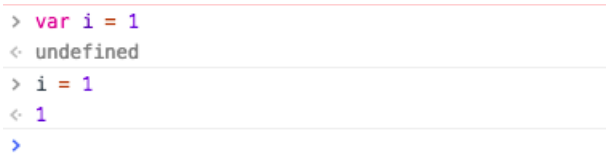
- 声明类语句
  - var声明
  - const声明
  - let声明
  - 函数声明
  - 类声明
- 表达式语句
- 空语句
- debugger语句

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。



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：

```
{
  var i = 1; // normal, empty, empty
  i ++; // normal, 1, empty
  console.log(i) //normal, undefined, empty
} // 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。

```
{
  var i = 1; // normal, empty, empty
  return i; // return, 1, empty
  i ++;
  console.log(i)
} // 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：

```
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。

```
outer: while(true) {
  inner: while(true) {
      break outer;
  }
}
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。

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。

```
function foo(){
  try{
    return 0;
  } catch(err) {

  } finally {
    console.log("a")
  }
}

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？

```
function foo(){
  try{
    return 0;
  } catch(err) {

  } finally {
    return 1;
  }
}

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。

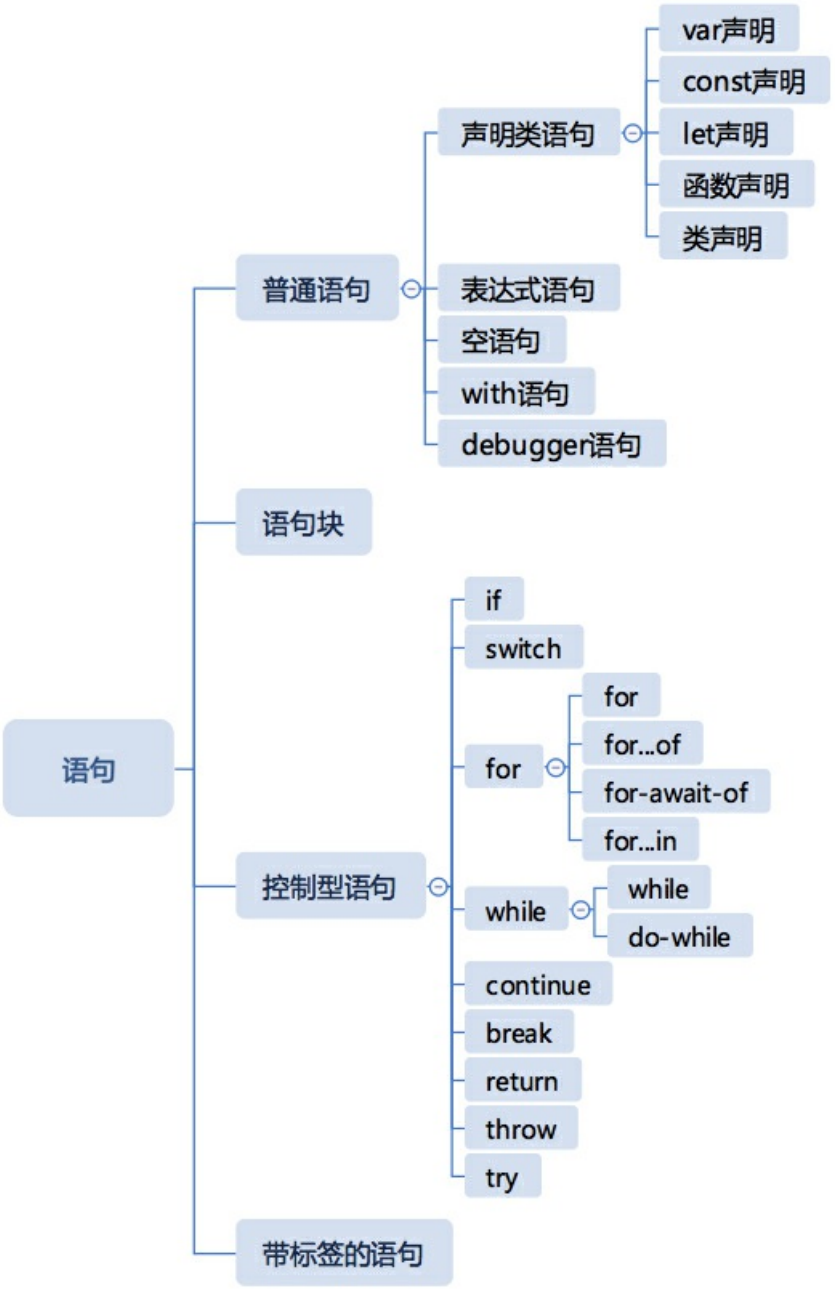
这一机制的基础正是JavaScript语句执行的完成状态，我们用一个标准类型来表示：Completion Record（我在类型一节提到过，Completion Record用于描述异常、跳出等语句执行过程）。

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。

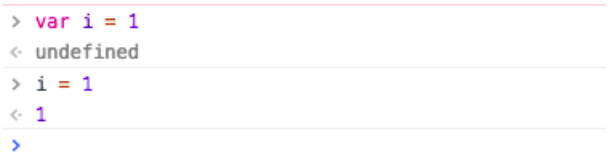
- 声明类语句
  - var声明
  - const声明
  - let声明
  - 函数声明
  - 类声明
- 表达式语句
- 空语句
- debugger语句

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。



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：

```
{
  var i = 1; // normal, empty, empty
  i ++; // normal, 1, empty
  console.log(i) //normal, undefined, empty
} // 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。

```
{
  var i = 1; // normal, empty, empty
  return i; // return, 1, empty
  i ++;
  console.log(i)
} // 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：

```
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。

```
outer: while(true) {
  inner: while(true) {
      break outer;
  }
}
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。

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。

```
function foo(){
  try{
    return 0;
  } catch(err) {

  } finally {
    console.log("a")
  }
}

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？

```
function foo(){
  try{
    return 0;
  } catch(err) {

  } finally {
    return 1;
  }
}

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。

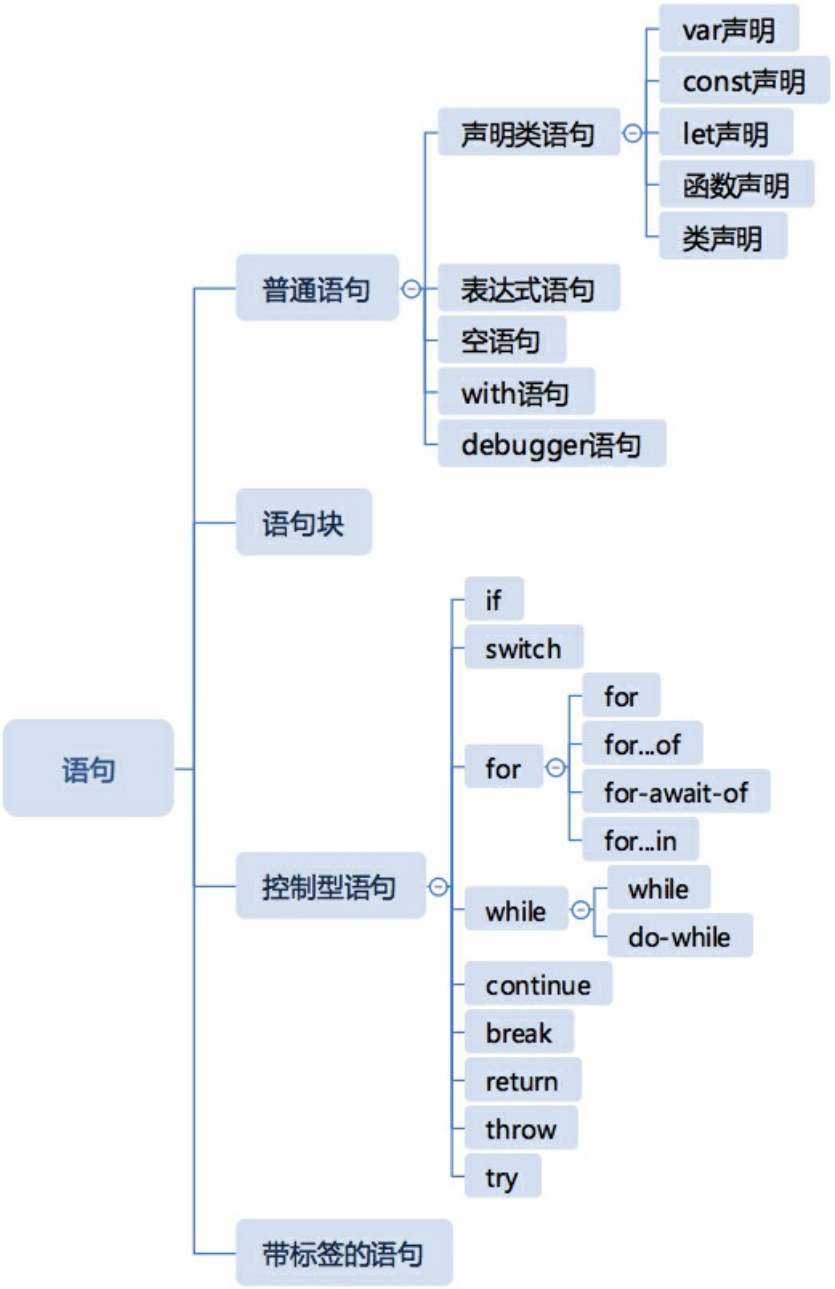
这一机制的基础正是JavaScript语句执行的完成状态，我们用一个标准类型来表示：Completion Record（我在类型一节提到过，Completion Record用于描述异常、跳出等语句执行过程）。

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。

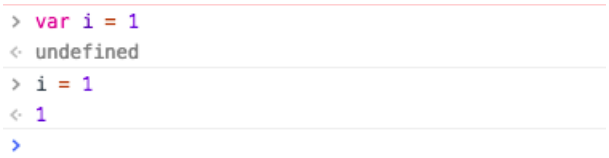
- 声明类语句
  - var声明
  - const声明
  - let声明
  - 函数声明
  - 类声明
- 表达式语句
- 空语句
- debugger语句

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。



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：

```
{
  var i = 1; // normal, empty, empty
  i ++; // normal, 1, empty
  console.log(i) //normal, undefined, empty
} // 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。

```
{
  var i = 1; // normal, empty, empty
  return i; // return, 1, empty
  i ++;
  console.log(i)
} // 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：

```
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。

```
outer: while(true) {
  inner: while(true) {
      break outer;
  }
}
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。

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。

```
function foo(){
  try{
    return 0;
  } catch(err) {

  } finally {
    console.log("a")
  }
}

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？

```
function foo(){
  try{
    return 0;
  } catch(err) {

  } finally {
    return 1;
  }
}

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。

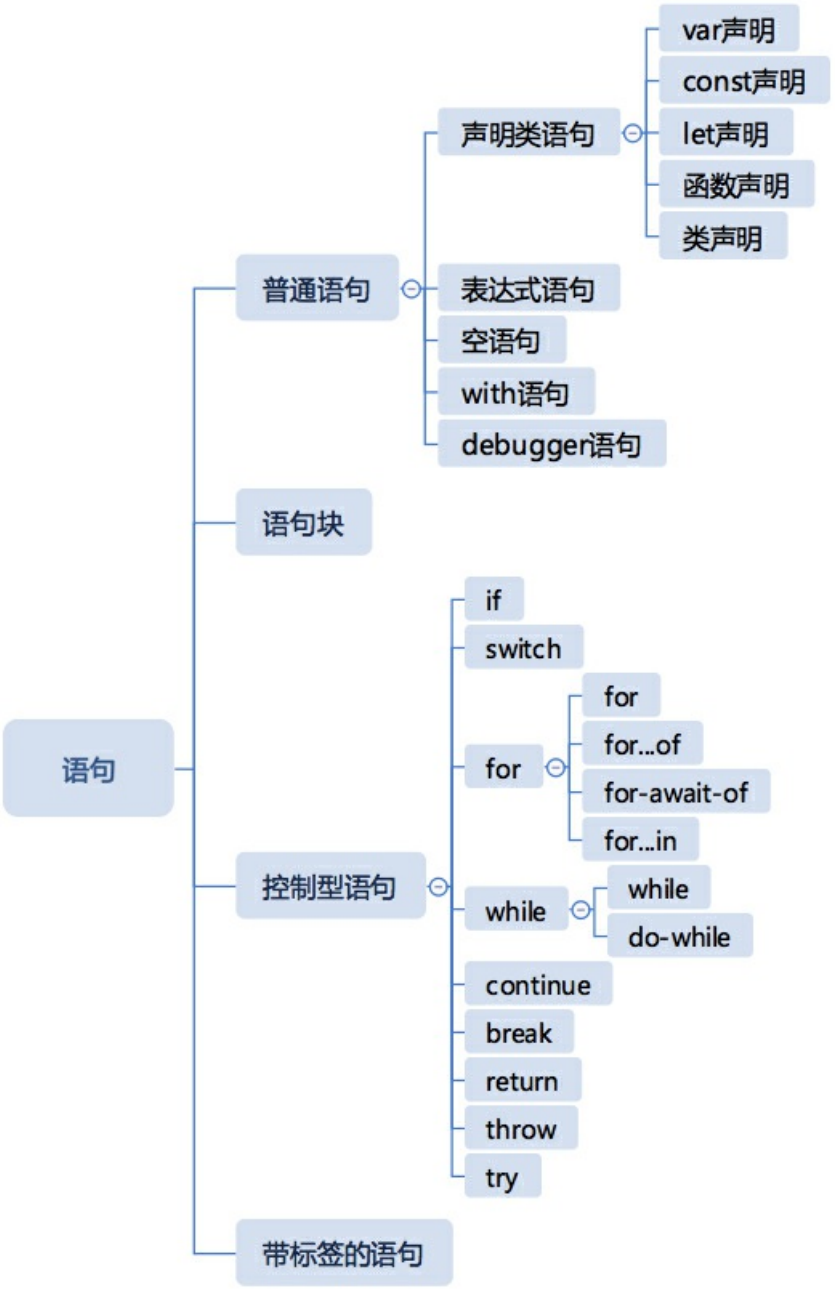
这一机制的基础正是JavaScript语句执行的完成状态，我们用一个标准类型来表示：Completion Record（我在类型一节提到过，Completion Record用于描述异常、跳出等语句执行过程）。

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。

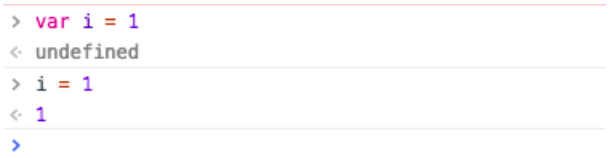
- 声明类语句
  - var声明
  - const声明
  - let声明
  - 函数声明
  - 类声明
- 表达式语句
- 空语句
- debugger语句

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。



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：

```
{
  var i = 1; // normal, empty, empty
  i ++; // normal, 1, empty
  console.log(i) //normal, undefined, empty
} // 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。

```
{
  var i = 1; // normal, empty, empty
  return i; // return, 1, empty
  i ++;
  console.log(i)
} // 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：

```
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。

```
outer: while(true) {
  inner: while(true) {
      break outer;
  }
}
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。

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。

```
function foo(){
  try{
    return 0;
  } catch(err) {

  } finally {
    console.log("a")
  }
}

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？

```
function foo(){
  try{
    return 0;
  } catch(err) {

  } finally {
    return 1;
  }
}

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。

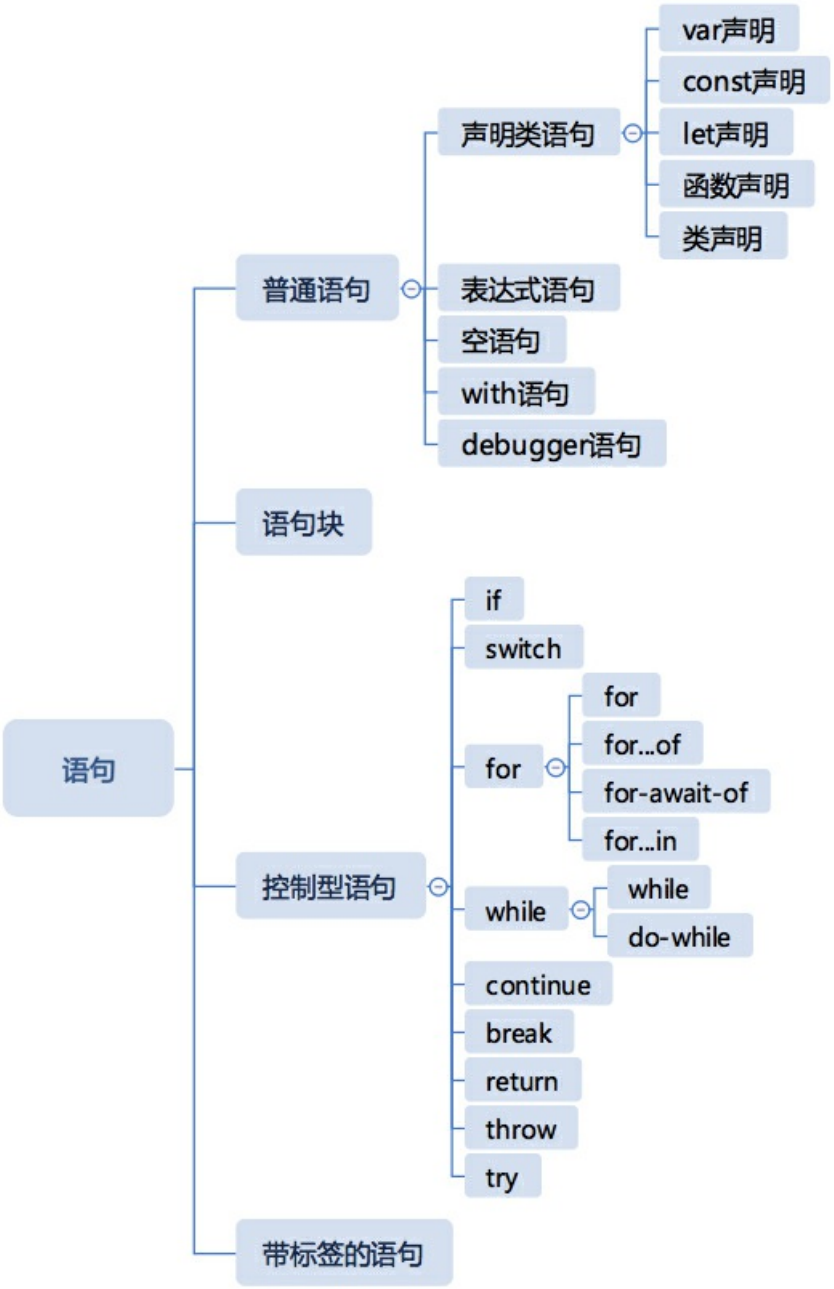
这一机制的基础正是JavaScript语句执行的完成状态，我们用一个标准类型来表示：Completion Record（我在类型一节提到过，Completion Record用于描述异常、跳出等语句执行过程）。

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。

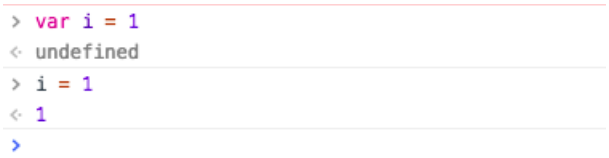
- 声明类语句
  - var声明
  - const声明
  - let声明
  - 函数声明
  - 类声明
- 表达式语句
- 空语句
- debugger语句

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。



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：

```
{
  var i = 1; // normal, empty, empty
  i ++; // normal, 1, empty
  console.log(i) //normal, undefined, empty
} // 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。

```
{
  var i = 1; // normal, empty, empty
  return i; // return, 1, empty
  i ++;
  console.log(i)
} // 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：

```
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。

```
outer: while(true) {
  inner: while(true) {
      break outer;
  }
}
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。

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。

```
function foo(){
  try{
    return 0;
  } catch(err) {

  } finally {
    console.log("a")
  }
}

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？

```
function foo(){
  try{
    return 0;
  } catch(err) {

  } finally {
    return 1;
  }
}

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。

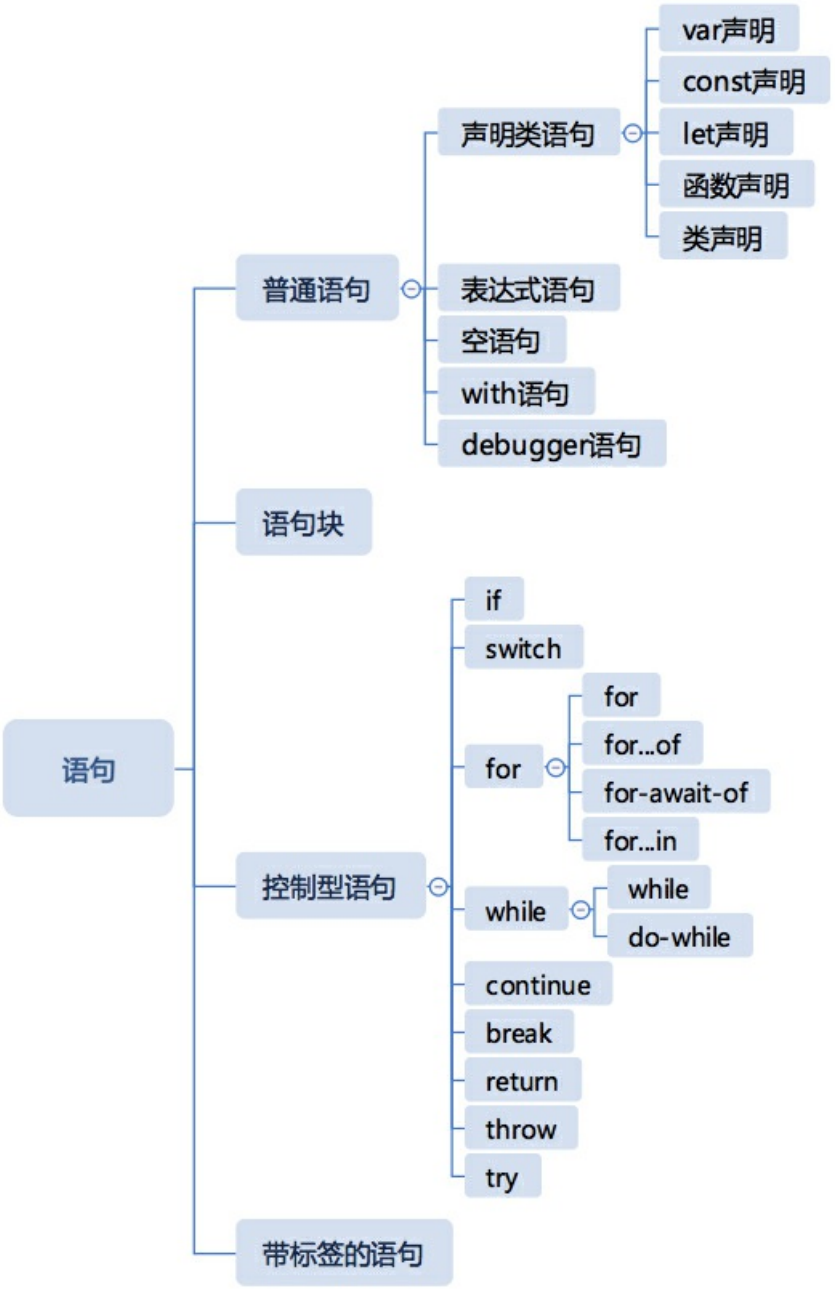
这一机制的基础正是JavaScript语句执行的完成状态，我们用一个标准类型来表示：Completion Record（我在类型一节提到过，Completion Record用于描述异常、跳出等语句执行过程）。

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。

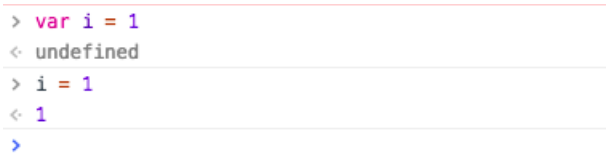
- 声明类语句
  - var声明
  - const声明
  - let声明
  - 函数声明
  - 类声明
- 表达式语句
- 空语句
- debugger语句

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。



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：

```
{
  var i = 1; // normal, empty, empty
  i ++; // normal, 1, empty
  console.log(i) //normal, undefined, empty
} // 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。

```
{
  var i = 1; // normal, empty, empty
  return i; // return, 1, empty
  i ++;
  console.log(i)
} // 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：

```
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。

```
outer: while(true) {
  inner: while(true) {
      break outer;
  }
}
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。