31-JavaScript语法（三）：什么是表达式语句？

你好，我是winter。

不知道你有没有注意到，我们在语句部分，讲到了很多种语句类型，但是，其实最终产生执行效果的语句不多。

事实上，真正能干活的就只有表达式语句，其它语句的作用都是产生各种结构，来控制表达式语句执行，或者改变表达式语句的意义。

今天的课程，我们就深入到表达式语句中来学习一下。

什么是表达式语句

表达式语句实际上就是一个表达式，它是由运算符连接变量或者直接量构成的（关于直接量我们在下一节详细讲解）。

一般来说，我们的表达式语句要么是函数调用，要么是赋值，要么是自增、自减，否则表达式计算的结果没有任何意义。

但是从语法上，并没有这样的限制，任何合法的表达式都可以当做表达式语句使用。比如我们看下面的例子。

```
a + b;
```

这句代码计算了a和b相加的值，但是不会显示出来，也不会产生任何执行效果（除非a和b是getter），但是不妨碍它符合语法也能够被执行。

下面我们就一起来了解下都有哪些表达式，我们从粒度最小到粒度最大了解一下。

PrimaryExpression 主要表达式

首先我们来给你讲解一下表达式的原子项：Primary Expression。它是表达式的最小单位，它所涉及的语法结构也是优先级最高的。

Primary Expression包含了各种“直接量”，直接量就是直接用某种语法写出来的具有特定类型的值。我们已经知道，在运行时有各种值，比如数字123，字符串Hello world，所以通俗地讲，直接量就是在代码中把它们写出来的语法。

我们在类型部分，已经介绍过一些基本类型的直接量。比如，我们当时用null关键字获取null值，这个用法就是null直接量，这就是这里我们仅仅把它们简单回顾一下：

```
"abc";
123;
null;
```

```
true;
false;
```

除这些之外，JavaScript还能够直接量的形式定义对象，针对函数、类、数组、正则表达式等特殊对象类型，JavaScript提供了语法层面的支持。

```
({});
(function(){});
(class{ });
[];
/abc/g;
```

需要注意，在语法层面，function、{ 和class开头的表达式语句与声明语句有语法冲突，所以，我们要想使用这样的表达式，必须加上括号来回避语法冲突。

在JavaScript标准中，这些结构有的被称作直接量（Literal），有的被称作表达式（**Expression），在我看来，把它们都理解成直接量比较合适。

Primary Expression还可以是this或者变量，在语法上，把变量称作“标识符引用”。

```
this;
myVar;
```

任何表达式加上圆括号，都被认为是Primary Expression，这个机制使得圆括号成为改变运算优先顺序的手段。

```
(a + b);
```

这就是Primary Expression的几种形式了，接下来，我们讲讲由Primary Expression构成的更复杂的表达式：Member Expression。

MemberExpression 成员表达式

Member Expression通常是用于访问对象成员的。它有几种形式：

```
a.b;
a["b"];
new.target;
super.b;
```

前面两种用法都很好理解，就是用标识符的属性访问和用字符串的属性访问。而new.target是个新加入的语法，用于判断函数是否是被new调用，super则是构造函数中，用于访问父类的属性的语法。

从名字就可以看出，Member Expression最初设计是为了属性访问的，不过从语法结构需要，以下两种在JavaScript标准中当做Member Expression：

```
f`a${b}c`;
```

这是一个是带函数的模板，这个带函数名的模板表示把模板的各个部分算好后传递给一个函数。

```
new Cls();
```

另一个是带参数列表的new运算，注意，不带参数列表的new运算优先级更低，不属于Member Expression。

实际上，这两种被放入Member Expression，仅仅意味着它们跟属性运算属于同一优先级，没有任何语义上的关联。接下来我们看看Member Expression能组成什么。

NewExpression NEW表达式

这种非常简单，Member Expression加上new就是New Expression（当然，不加new也可以构成New Expression，JavaScript中默认独立的高优先级表达式都可以构成低优先级表达式）。

注意，这里的New Expression特指没有参数列表的表达式。我们看个稍微复杂的例子：

```
new new Cls(1);
```

直观看上去，它可能有两种意思：

```
new (new Cls(1));
```

```
new (new Cls)(1);
```

实际上，它等价于第一种。我们可以用以下代码来验证：

```
class Cls{
  constructor(n){
    console.log("cls", n);
    return class {
      constructor(n) {
        console.log("returned", n);
      }
    }
  }
}

new (new Cls(1));
```

这段代码最后得到了下面这样的结果。

```
cls 1
returned undefined
```

这里就说明了，1被当做调用Cls时的参数传入了。

CallExpression 函数调用表达式

除了New Expression，Member Expression还能构成Call Expression。它的基本形式是Member Expression后加一个括号里的参数列表，或者我们可以用上super关键字代替Member Expression。

```
a.b(c);
super();
```

这看起来很简单，但是它有一些变体。比如：

```
a.b(c)(d)(e);
a.b(c)[3];
a.b(c).d;
a.b(c)`xyz`;
```

这些变体的形态，跟Member Expression几乎是一一对应的。实际上，我们可以理解为，Member Expression中的某一子结构具有函数调用，那么整个表达式就成为了一个Call Expression。

而Call Expression就失去了比New Expression优先级高的特性，这是一个主要的区分。

LeftHandSideExpression 左值表达式

接下来，我们需要理解一个概念：New Expression 和 Call Expression 统称LeftHandSideExpression，左值表达式。

我们直观地讲，左值表达式就是可以放到等号左边的表达式。JavaScript语法则是下面这样。

```
a() = b;
```

这样的用法其实是符合语法的，只是，原生的JavaScript函数，返回的值都不能被赋值。因此多数时候，我们看到的赋值将会是Call Expression的其它形式，如：

```
a().c = b;
```

另外，根据JavaScript运行时的设计，不排除某些宿主会提供返回引用类型的函数，这时候，赋值就是有效的了。

左值表达式最经典的用法是用于构成赋值表达式，但是其实如果你翻一翻JavaScript标准，你会发现它出现在各种场合，凡是需要“可以被修改的变量”的位置，都能见到它的身影。

那么接下来我们就讲讲 AssignmentExpression 赋值表达式。

AssignmentExpression 赋值表达式

AssignmentExpression 赋值表达式也有多种形态，最基本的当然是使用等号赋值：

```
a = b
```

这里需要理解的一个稍微复杂的概念是，这个等号是可以嵌套的：

```
a = b = c = d
```

这样的连续赋值，是右结合的，它等价于下面这种：

```
a = (b = (c = d))
```

也就是说，先把d的结果赋值给c，再把整个表达式的结果赋值给b，再赋值给a。

当然，这并非一个很好的代码风格，我们讲解语法是为了让你理解这样的用法，而不是推荐你这样写代码。

赋值表达式的使用，还可以结合一些运算符，例如：

```
a += b;
```

相当于

```
a = a + b;
```

能有这样用的运算符有下面这几种：

*=、/=、%=、+=、-=、<<=、>>=、>>>=、&=、^=、|=、**=

我想你已经注意到了，赋值表达式的等号左边和右边能用的表达式类型不一样，在这一课，我们已经关注完了表达式的左边部分（左值表达式）的语法结构，下一节课，我们将会给你重点讲解表达式的右边部分。

Expression 表达式

赋值表达式可以构成Expression表达式的一部分。在JavaScript中，表达式就是用逗号运算符连接的赋值表达式。

在JavaScript中，比赋值运算优先级更低的就是逗号运算符了。我们可以把逗号可以理解为一种小型的分号。

```
a = b, b = 1, null;
```

逗号分隔的表达式会顺次执行，就像不同的表达式语句一样。“整个表达式的结果”就是“最后一个逗号后的表达式结果”。比如我们文中的例子，整个“`a = b, b = 1, null;`”表达式的结果就是“, ”后面的`null`。

在很多场合，都不允许使用带逗号的表达式，比如我们在前面课程中提到，export后只能跟赋值表达式，意思就是表达式中不能含有逗号。

结语

这节课我们开始讲解了运算符和表达式的一些相关知识，这节课上，我们已经学习了赋值表达式和赋值表达式的左边部分。下节课，我们将会讲一讲赋值表达式的右边部分。

最后给你留一个作业，把今天讲到的所有运算符按优先级排列成一个表格，下节课我们会补完剩下的部分。

精选留言：

- 木木 2019-04-04 09:46:13
  f`a${b}c`;
  没有明白这个是什么意思 [4赞]

  作者回复2019-04-09 18:20:40
  写一个函数f试试就知道了。

- 炒饭 2019-04-15 19:07:18
  @嗨海海 winter老师讲得是语言细节，这些应该都是基础知识，但实际上很多一线前端都忽略了这些。比起常见那些框架工具应用，这课在国内还是很难得的，特别还是winter老师开的，讲的透彻，感谢winter老师，让我受益匪浅 [2赞]

- 嗨海海 2019-04-12 06:13:47
  完全看不懂这个专栏莫名其妙，代码语法不在机器上跑一跑讲一讲，到处扯一扯，这跟开口讲photoshop差不多吧 [2赞]

- vspt 2019-04-04 20:24:41
  winter老师，问个问题，在react源码中经常看到如下写法，一直没太理解，请问这种写法有什么好处吗？

  ```
  var validateFormat = function () {};

  {
  validateFormat = function (format) {
  if (format === undefined) {
  throw new Error('invariant requires an error message argument');
  }
  ```

};
}
```[2赞]

- 桂马 2019-04-07 23:37:54
  平时不确定优先级的一般都加括号

- 王俊宇 2019-04-04 20:39:11

  默认的模板字符串就是拼接，但是可以写个函数来改变这种默认行为。之前的课有讲过的。感觉这个有点类似julia语言的@f_str宏

- 木木 2019-04-04 09:45:37
  f`a${b}c`;
```