**理论知识**

JavaScript是一种描述型的脚本语言，它不同于java或C#等编译性语言,它不需要进行编译成中间语言,而是由浏览器进行动态地解析与执行。所以理解javaScript语言的运行机制，掌握javascript的执行顺序，有利于我们学好JavaScript。

**预处理阶段**

js在预处理阶段，创建一个词法环境（LexicalEnvironment），扫描JS中的用***声明的方式声明的函数***，**用var定义的变量**并将它们加到预处理阶段的词法环境中去。扫描顺序：先扫描函数声明，再扫描变量声明

用var定义的变量：undefined;

声明的函数： 对函数引用

其中遇到命名冲突时，处理原则为：

1.处理函数声明有冲突时，会覆盖；

2.处理变量声明有冲突时，会忽略

**执行阶段**

在执行阶段中，js会先扫描函数声明后扫描变量，然后给预处理阶段中执行上下文对象中的成员赋值，如果没有用var声明的变量，会成为最外部执行上下文的成员。

**全局**

**全局预处理阶段**

比如说下面的这段代码：  
<pre>  
var a = 1;//用var定义的变量，以赋值  
var b;//用var定义的变量，未赋值  
c = 3;//未定义，直接赋值  
function d(){//用声明的方式声明的函数  
console.log('hello');  
}  
var e = function(){//函数表达式  
console.log('world');  
}  
</pre>  
在预处理时，创建的作用域标示为：

<pre>

LE{  
a:undefined  
b:undefined  
没有c  
d：对函数引用  
没有e  
}

</pre>

***Note***  
一定区分 ***声明式的函数*** 和 ***函数表达式*** 概念

**命名冲突的处理**

<pre>  
console.log(f);  
var f = 1;  
function f(){  
console.log('foodoir');  
}  
</pre>

输出结果：function f(){console.log('foodoir');}

<pre>  
console.log(f);  
function f(){  
console.log('foodoir');  
}  
var f = 1;  
</pre>

输出结果：function f(）{console.log('foodoir');}

<pre>  
console.log(f);  
var f = 1;  
var f = 2;  
console.log(f);  
</pre>

输出结果：先undefined 后 2

***总结一句话：函数是js里的第一等公民***

**全局**

**全局执行阶段**

<pre>  
**console**.log(***a***);  
**console**.log(***b***);  
**console**.log(*c*);  
**console**.log(*d*);  
**var *a*** = 1;  
***b*** = 2;  
**console**.log(**b**);  
**function** *c*() {  
 **console**.log(**'c'**);  
}  
  
**var** *d* = **function** () {  
 **console**.log(**'d'**);  
}  
**console**.log(*d*);

</pre>

1.首先我们先进行与处理阶段分析  
<pre>  
LE{  
a:undefined  
没有b  
c:对函数的一个引用  
d:undefined  
}  
</pre>

结果：  
<pre>  
此时，我们可以得到前四行代码得到的结果分别为：  
　　undefined  
　　报错  
　　function c(){console.log('c');  
　　undefined  
</pre>

2、当执行完预处理后，代码开始一步步被解析（将第二行报错的代码注释掉）  
<pre>  
LE{  
a:1  
b:2  
c:指向函数  
d:指向函数  
}  
</pre>

其中变量b比较特殊，变为了***最外部执行上下文的成员***

**函数**

**函数预处理阶段**

函数在预处理阶段与全局的基本一致，函数中有个arguments

* 每调用一次，产生一个词法环境（或执行上下文Execution Context）；
* 先传入函数的参数，若参数值为空，初始化undefined；
* 然后是内部函数声明，若发生命名冲突，会覆盖；
* 接着就是内部var变量声明，若发生命名冲突，会忽略；

让我们来看下面的例子

<pre>  
function f(a,b){  
alert(a);  
alert(b);

var b = 100;

function a(){}

}  
f(1,2);

</pre>

我们先来分析函数的预处理，它和全局的预处理类似，它的词法结构如下：

<pre>  
LE{  
b:2  
a:指向函数的引用  
arguments：2  
}  
//arguments，调用函数时实际调用的参数个数

</pre>

再结合之前的那句话：处理函数声明有冲突时，会覆盖；处理变量声明时有冲突，会忽略。  
故结果分别为：function a(){}和2

其中有两种特殊情况  
1.实际参数少于形参，  
f(1)

这个时候语法结构  
<pre>

LE{  
b:undefined  
a:对函数的一个引用  
arguments：1  
}  
故结果分别为：function a(){}和undefined  
</pre>  
2.没有用var声明的变量

<pre>  
function a(){  
function b(){  
g = 12;  
}  
b();  
}  
a();  
console.log(g);//12  
</pre>

如果没有用var声明的变量，会变成最外部LE的成员，即全局变量

**函数执行阶段**

* 给预处理阶段的成员赋值；
* 无var声明的变量，会成为全局成员；