1. JavaScript的解析与执行过程

key:预处理（函数、变量覆盖）、执行这两项的顺序

1）全局预处理--扫描函数声明，再扫描变量声明

example：

```
console.log(a); // undefined
console.log(b); //ReferenceError: b is not defined
var a = 1;
b = 2;
```

同理，词法环境中只有用var定义的变量。

```
alert(f);                //function f(){alert('567');}
var f = 1;
function f() {
    alert('123');
}
var f = 2;
function f() {
    alert('567');
}
var f = function() {
    alert('890');
}
```

运行代码

命名冲突时，函数具有“优先权”，变量会直接被忽略，而函数会被覆盖。

2）全局执行过程

```
alert(a);        //undefined
alert(b);        //ReferenceError: b is not defined
alert(f);         //function f() {console.log('f');}
alert(g);        //undefined
var a = 1;
b = 2;
alert(b);           //2
function f() {
  console.log('f');
}
var g = function () {
  console.log('g');
}
alert(g);            //function () {console.log('g');}
```

为了分析方便我们可以使用词法环境：

```
LexicalEnvironment {} === window;
```

全局预处理时：

```
window {
    //先扫描函数声明：
    f: 指向函数，
    //再扫面变量声明：
    a: undefined，
    g: undefined
}
```

全局执行时：

```
window {
    f: 指向函数，
    a: 1，
    b: 2，
    g: 指向函数
}
```

example:

3）. 函数预处理阶段

- 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；

```
function f(a, b) {
  alert(a);              //1
  alert(b);              //funciton b(){}
  var a = 100;
  function b() {
  }
}
f(1, 2);
```

```
AO(f) {
    a: 1,                          //变量命名冲突，忽略
    b: 指向函数,                   //函数命名冲突，覆盖
}
```

4）. 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

2.函数声明和函数表达式与变量提升问题

https://javascriptweblog.wordpress.com/2010/07/06/function-declarations-vs-function-expressions/

知识点：变量提升/函数声明提升/函数表达式中的变量部分提升，表达式部分不提升

```
function foo(){
    function bar() {
        return 3;
    }
    return bar();
    function bar() {
        return 8;
    }
}
alert(foo());
```

e：函数提升//8

函数表达式中变量部分提升，函数表达式不提升//3

```
alert(foo());
function foo(){
    var bar = function() {
        return 3;
    };
    return bar();
    var bar = function() {
        return 8;
    };
}
```