1. 作用域
2. 全局作用域

在script下面声明一个变量，那么在整个作用域下，这个变量均有效，

在函数中也可以访问这个变量，

1. 局部作用域（在function中），

在函数内部声明一个变量，这个只能在function中访问，外面无法访问。函数中传入的参数也算作局部参数。

1. 全局作用域

声明变量为全局变量，可以在script下全局访问。

注意：作用域只限于全局作用域和函数的局部作用域，for循环没有作用域，在for循环中声明的变量相当于全局变量

函数=》全局函数，可以全局访问

例：

Outer（）；

Function outer（）{alert（1）}；

结果为弹出1.

调用可以写在任何地方，甚至是函数中执行的代码。

注意：

1）在script下面定义的变量和函数均可以通过window.来访问，全局变量等同于window.变量，函数执行 函数名();也等同于window.函数名（）；

全局变量相当于window下面的一个属性，全局函数相当于window下面的一个方法（属性定义为函数）。

1. 在不同的位置写入script标签，在一个中写入声明，在另外一个中写入调用，能够成功（但是声明必须在调用的前面）

因为在全局的概念下，只有一个window，那么在这个window下，a相当于一个属性，先在一个script中声明，那么这个a可以在另一个script中调用。

注：必须先声明后调用，因为js解析顺序是从上到下，当在上面调用的时候，还未解析到声明处，所以会报错

外部引入也遵循这个规则。

1. 局部作用域

局部变量：在函数中声明的变量

局部函数：在函数中声明的函数

局部变量和局部函数只能在外部函数之内访问，外部无法访问。

例如：

Function fn（）{

Var a=10;

Function inner(){

Alert( a );

}

Inner();

}

Fn();

内部函数可以调用外部声明的变量

总结：函数中的变量=》局部变量，内部可以访问

函数中的函数=》局部函数，内部可以访问

注意：1)内部的可以发访问外部的内容

1. 若在外部定义一个变量，在内部也定义一个变量，那么采用就近原则，找到最近的，内部变量会覆盖外部变量。
2. 若使用变量先调用后声明，那么结果就会为undefined

例如：

Function fn(){

Alert( a );//解析的时候还没有声明

Var a=2;

}

Fn();

结果为undefined

1. 作用域链

在函数中，如果有层层套用

例：

Var a=90;

Function fn1(){

Function fn2(){

Function fn3(){

Alert( a );

}

Fn3();

}

Fn2();

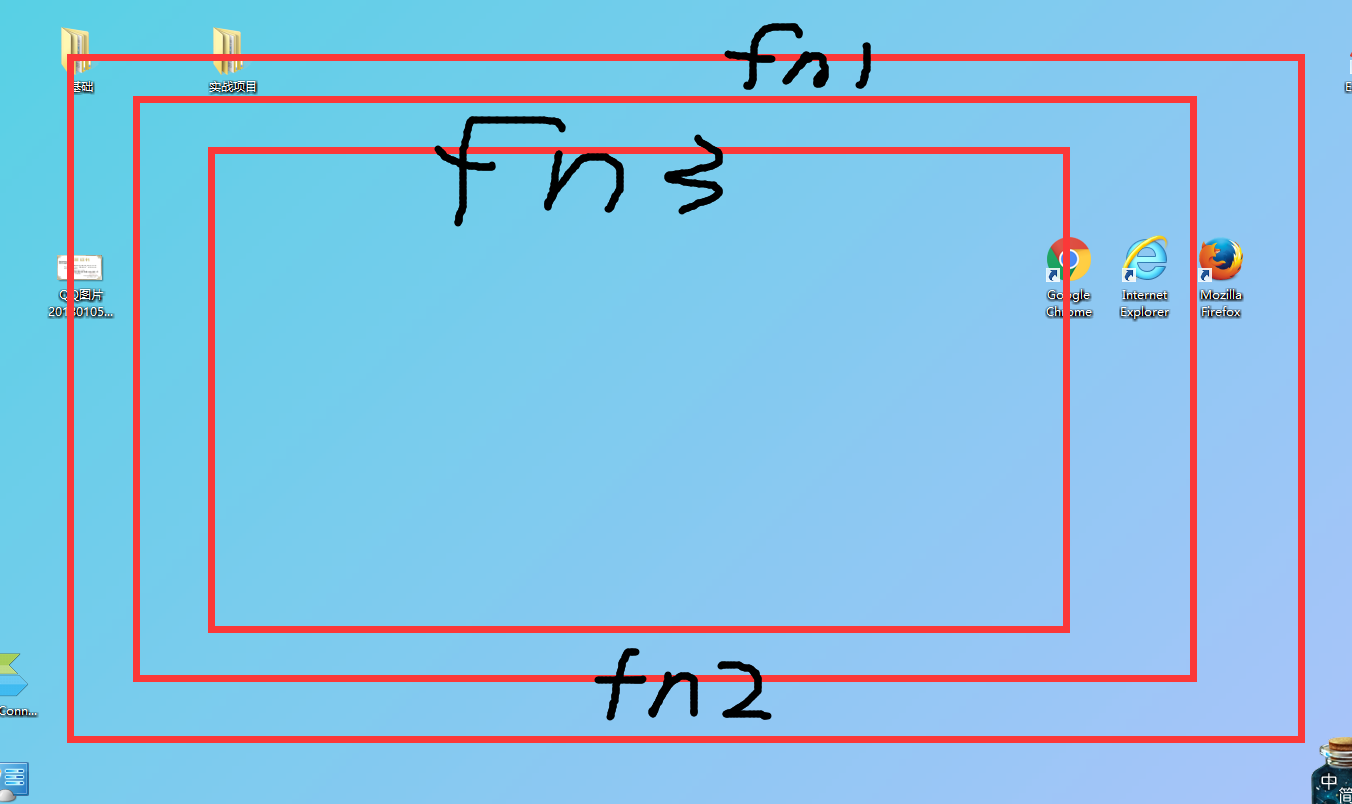
}

Fn1();

那么这个函数套用的结果为弹出a；

在调用函数的时候，每个函数会形成一个空间，在层层套用的函数里面，会形成下图所示的层层套用空间，在最里面的函数中的代码，在调用的时候会寻找变量，当找不到的时候，就会进入外部的空间中继续找，找不到再往外面找，直到找到为止。就如同上面的例子一样，fn3的代码找a，找不到就往fn2的空间里面找，找不到就往fn1的空间中找，找不到就到全局找，最终找到。

那么作用域链就是从内往外找。



1. js解析
2. 解析顺序：从上往下
3. 预解析：

Js在解析之前，会进行预解析，在预解析的时候，js会识别代码中的var 和function函数关键词，并将之保存在一个空间中，

若是var关键词，那么就将变量保存在空间中，并赋值为undefined，

若是函数，就整个保存下来。

所以在后面执行的时候，若先调用变量，那么就会返回一个undefined，若是函数，则就执行。

原因：在预解析之后，进行代码执行，执行到变量或函数调用的时候，就会从预解析形成的空间里面找相应的量，找到变量就是undefined，找到函数就是执行该函数，这就是为什么先调用变量是undefined，先调用函数能够执行的原因。

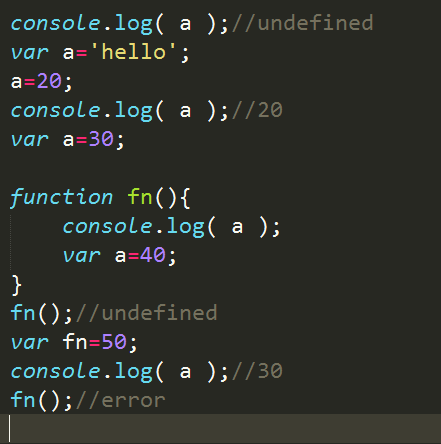
3特殊

1. 变量和函数重名，看谁在后面，保留谁，谁在后面谁的优先级高
2. 若有两个函数同名，那么后面的函数覆盖前面的函数。这个可以用预解析来解释。
3. 注意：函数表达式相当于比那辆，若先被调用，那么就会被视为变量。
4. 变量提升=》在预解析获得变量并赋值undefined的过程中，

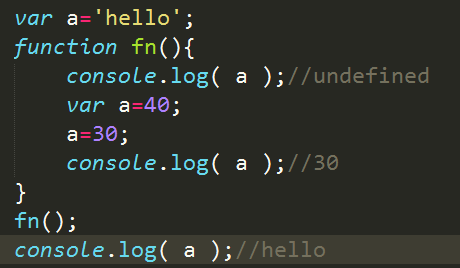
函数提升=》预解析获得函数的过程

1. 案例：

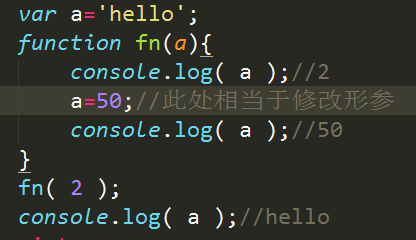
1、



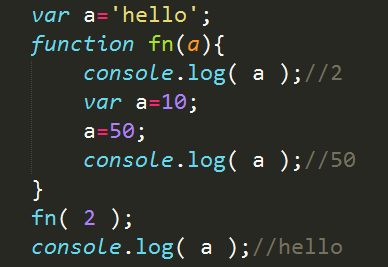
2、



3、



4、

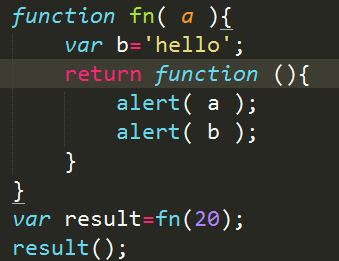


1. 闭包
2. 垃圾回收机制

不在使用的变量，函数，会被js垃圾回收机制回收

1. 闭包，将使用过的值保存起来，不被回收

例：



这个可以将使用过的a和b保存在result中，待到要使用的时候在使用、

3、使用的条件

1）函数内嵌函数

2）return后面接内嵌函数

3）内嵌函数的变量参数不会被垃圾回收机制回收。