**作用域**

**1. 理解**

\* 就是一块"地盘", 一个代码段所在的区域

\* 它是静态的(相对于上下文对象), 在编写代码时就确定了

**2. 分类**

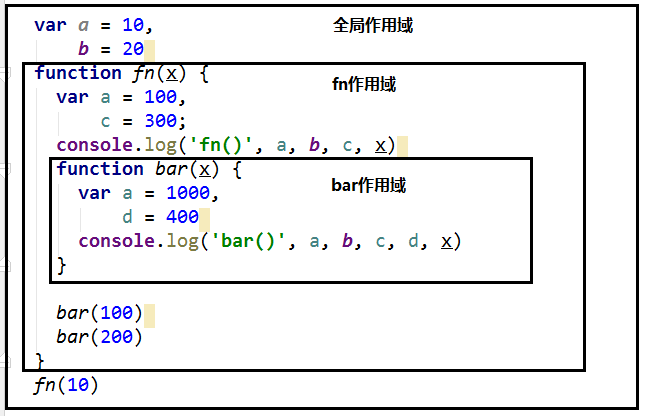
\* 全局作用域

\* 函数作用域

\* 块作用域(ES6)

**3. 作用**

\* 隔离变量，不同作用域下同名变量不会有冲突



**作用域与执行上下文**

**1. 区别1**

\* 全局作用域之外，每个函数都会创建自己的作用域，作用域在函数定义时就已经确定了。而不是在函数调用时

\* 全局执行上下文环境是在全局作用域确定之后, js代码马上执行之前创建

\* 函数执行上下文是在调用函数时, 函数体代码执行之前创建

**2. 区别2**

\* 作用域是静态的, 只要函数定义好了就一直存在, 且不会再变化

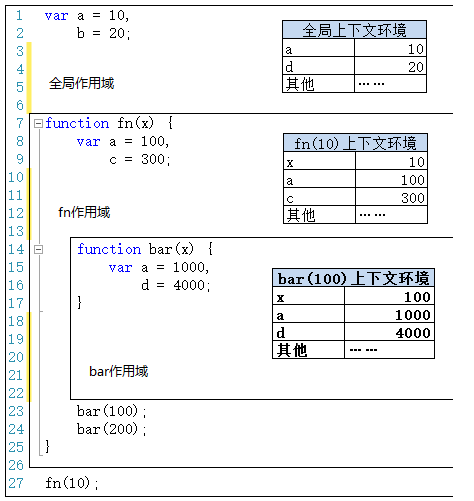
\* 执行上下文是动态的, 调用函数时创建, 函数调用结束时就会自动释放

**3. 联系**

\* 执行上下文(对象)是从属于所在的作用域

\* 全局上下文环境==>全局作用域

\* 函数上下文环境==>对应的函数使用域



**作用域链**

**1. 理解**

\* 多个上下级关系的作用域形成的链, 它的方向是从下向上的(从内到外)

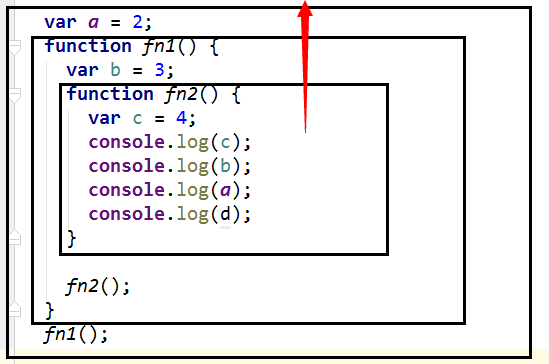
\* 查找变量时就是沿着作用域链来查找的

**2. 查找一个变量的查找规则**

\* 在当前作用域下的执行上下文中查找对应的属性, 如果有直接返回, 否则进入2

\* 在上一级作用域的执行上下文中查找对应的属性, 如果有直接返回, 否则进入3

\* 再次执行2的相同操作, 直到全局作用域, 如果还找不到就抛出找不到的异常



**面试题1**

var x = 10;

function fn() {

console.log(x);

}

function show(f) {

var x = 20;

f();

}

show(fn);//10

**面试题2**

var fn = function () {

    console.log(fn)

  }

  fn()//正常输出

  var obj = {

    fn2: function () {

     console.log(fn2)//报错

     //console.log(this.fn2) //正常输出

    }

  }

  obj.fn2()