1.作用域

作用域规定了变量能够被访问的"范围",离开了这个"范围"变量便不能够被访问

- **局部作用域**
- **全局作用域**
- **作用域链**
- **JS垃圾回收机制**
- **闭包**
- **变量提升**

1.1局部作用域

分为函数作用域和块作用域

1. 函数作用域:

在函数内部声明的变量只能在函数内部访问，外部无法直接访问。

总结:

1. 函数的参数也是函数内部的局部变量
2. 不同函数内部声明的变量无法互相访问
3. 函数执行完毕后,函数内部的变量实际被当作垃圾回收了

2. 块作用域:

在JavaScript中使用 '{ }' 包裹起来的代码称作代码块,代码块内部声明的变量外部将[**有可能**]无法被访问

总结:

1.let声明的变量会产生块作用域，var不会产生块作用域 2.const声明的常量也会产生块作用域 3.不同代码块之间的变量无法互相访问 4.推荐使用let和const

1.2全局作用域

<script>标签和.js文件的【最外层】就是所谓的全局作用域，在此声明的变量在函数内部也可以被访问。全局作用域中生明的变量，任何其它作用域都可以被访问

```
<script>
    // 全局作用域
    // 全局作用域下声明了num变量
    const num=10;
    function fn(){
        // 函数内部可以使用全局作用域的变量
        console.log(num);
```

```
}
// 此处全局作用域
```

注意: 1.为window对象动态添加的属性默认也是全局的 2.函数中未使用任何关键字声明的变量为全局变量， 3.尽可能少的声明全局变量，防止全局变量被污染

1.3作用域链

作用域链本质是最底层的变脸查找机制。 > 在函数被执行时，会优先查找当前函数作用域中查找变量 > 如果当前作用域查找不到则会依次逐级查找父级作用域直到全局作用域

总结:

1.嵌套关系的作用域串联起来形成了作用域链 2.相同作用域链中按着从小到大的规则查找变量 3.子作用域能够访问父作用域，父级作用域无法访问子级作用域

1.4垃圾回收机制

内存的生命周期

1.5闭包(closure)

2.函数进阶

3.解构赋值

4.综合案例