目录

[一、 知识点汇总： 1](#_Toc79010394)

[1、 闭包： 1](#_Toc79010395)

[2、 JavaScript为什么是单线程？ 2](#_Toc79010396)

[3、 基本类型和引用类型的区别 3](#_Toc79010397)

[4、 null和undefined的异同 3](#_Toc79010398)

[5、 let、const、var 3](#_Toc79010399)

[6、 箭头函数和普通函数有什么区别 4](#_Toc79010400)

[7、 页面大量图片，如何优化？ 4](#_Toc79010401)

# 知识点汇总：

## 闭包：

**作用域**（即scope, 其实更确切地说是lexical scope）：就是一套变量访问规则，这些规则包括变量如何存储和访问，也就是规定了哪些变量可以被访问，哪些变量不可以被访问。

**词法环境对象**由两部分组成：

1、环境记录（Environment Record） —— 一个**存储所有局部变量**作为其属性（包括一些其他信息，例如 this 的值）的对象。

2、对 外部词法环境的引用，与外部代码相关联。

**词法作用域：**函数和对其周围状态（词法环境）的引用捆绑构成闭包，函数在创建的时候会产生一个隐藏属性[[environment]]词法环境对象，这个属性用于记录当前函数的词法环境对象、外层词法环境对象和全局词法环境对象，形成了作用域链。这些词法环境对象构成了该函数的词法作用域。

**闭包**：词法作用域＋函数，每当函数被创建，就会在函数生成时生成闭包。

例：当foo函数被调用时返回的内部函数记录了被定义时的词法作用域，因此可以访问函数foo内的变量。

解决内存泄漏问题：如果一个对象不被引用，这个对象就会被回收，在内部函数的词法环境内拷贝引用外层环境的函数的变量，再清空外层函数的变量值，这个外层变量就会被回收了。

JavaScript是一种编译型的语言，但它不是提前编译的，它的编译发生在在代码执行前的几微秒。传统的编译过程分为三个阶段：

词法分析——词法分析阶段主要将程序代码拆分成一个个的**词法单元**。

比如将 var a = 2; 拆分成 var、a、=、2、；

语法分析——语法分析阶段主要将词法单元按照程序中的嵌套逻辑转换成“**抽象语法树**”。

代码生成——代码生成就是将抽象语法树转换为**可执行代码（字节码）**的过程。

## JavaScript为什么是单线程？

(<https://blog.csdn.net/baidu_24024601/article/details/51861792>)

假定JavaScript同时有两个线程，一个线程在某个DOM节点上添加内容，另一个线程删除了这个节点，这时浏览器应该以哪个线程为准？

这跟历史有关系。JavaScript从诞生起就是单线程。原因大概是不想让浏览器变得太复杂，因为多线程需要共享资源、且有可能修改彼此的运行结果，对于一种网页脚本语言来说，这就太复杂了。后来就约定俗成，JavaScript为一种单线程语言。（Worker API可以实现多线程，但是JavaScript本身始终是单线程的。）

单线程就意味着，所有任务需要排队，前一个任务结束，才会执行后一个任务。如果前一个任务耗时很长，后一个任务就不得不一直等着。

如果排队是因为计算量大，CPU忙不过来，倒也算了，但是很多时候CPU是闲着的，因为IO设备（输入输出设备）很慢（比如Ajax操作从网络读取数据），不得不等着结果出来，再往下执行。

JavaScript语言的设计者意识到，这时主线程完全可以不管IO设备，挂起处于等待中的任务，先运行排在后面的任务。等到IO设备返回了结果，再回过头，把挂起的任务继续执行下去。

于是，所有任务可以分成两种，一种是同步任务（synchronous），另一种是异步任务（asynchronous）。同步任务指的是，在主线程上排队执行的任务，只有前一个任务执行完毕，才能执行后一个任务；异步任务指的是，不进入主线程、而进入“任务队列”（task queue）的任务，只有“任务队列”通知主线程，某个异步任务可以执行了，该任务才会进入主线程执行。

3、异步执行机制：

（1）所有同步任务都在主线程上执行，形成一个执行栈（execution context stack）。

（2）主线程之外，还存在一个"任务队列"（task queue）。只要异步任务有了运行结果，就在"任务队列"之中放置一个事件。

（3）一旦"执行栈"中的所有同步任务执行完毕，系统就会读取"任务队列"，看看里面有哪些事件。那些对应的异步任务，于是结束等待状态，进入执行栈，开始执行。

（4）主线程不断重复上面的第三步。

只要主线程空了，就会去读取“任务队列”，这就是JavaScript的运行机制。

Event Loop： <https://segmentfault.com/a/1190000016278115>

异步、宏任务、微任务：https://gongchenghuigch.github.io/2019/09/14/awat/

## 基本类型和引用类型的区别

栈：速度快，内存小

堆：速度慢，内存大

基本类型创建时已经确定大小，引用类型创建时无法确定大小，因此基本内存放在栈中，引用类型放在堆中，同时栈中内存保存引用类型的变量标识符和指向堆内存中该对象的指针，即该对象在堆内存的地址。

## null和undefined的异同

null表示没有对象，即该处不应该有值

1） 作为函数的参数，表示该函数的参数不是对象

2） 作为对象原型链的终点

undefined表示缺少值，即此处应该有值，但没有定义

1）定义了形参，没有传实参，显示undefined

2）对象属性名不存在时，显示undefined

3）函数没有写返回值，即没有写return，拿到的是undefined

4）写了return，但没有赋值，拿到的是undefined

相同：

null和 undefined的值相等但是类型不同，所以两者相等但是不绝对相等。（null==undefined 但null !== undefined）

if条件下的null和undefined值都为false。

不同：

null转化为number数据类型为0，undefined转化为number类型为NaN。

## let、const、var

1. **let不存在变量提升**

必须遵循“先声明，后使用”的原则。否则会报错（ReferenceError），如果是var声明的变量，则不会报错（undefined）。

1. **let声明的变量，存在块级作用域**

let声明的变量只在所声明的代码块内有效。块级作用域由 { } 包括，if语句和for语句里面的{}也属于块作用域。

1. **let不允许在同一作用域内重复声明同一个变量**

在同一作用域内，如果使用var声明同一个变量，则后面的覆盖前面的

但是let重复声明会报错（Uncaught SyntaxError:Identifier 'a' has already been declared）

1. **暂时性死区：在代码块内，使用let声明变量之前，该变量都是不可以使用**

只要在同一作用域内存在let命令，他所声明的变量就“绑定”在这个作用域内，不管外部有没有声明

1. **const声明的常量不能改变**

意味着const一旦声明常量，就必须同时初始化。不能先声明，后初始化，这样会报 错

1. **const对于对象：**

因为Object类型是引用类型。用const声明常量保存的是对象的地址，不可变的是地址，而修改对象的属性，并不会改变对象的地址，因此用const声明的对象，属性可以修改。

1. **不推荐使用var：**

因为var创建的变量是全局变量挂载在window顶级对象上面的，如果跟之前定义的对象重复则会改变之前定义的值，变量会造成污染

## 箭头函数和普通函数有什么区别

1. 箭头函数的this对象是定义时所在的对象，用call apply bind也不能改变this指向
2. 不能当作构造函数
3. 不可以使用arguments对象，该对象在函数体内不存在。如果要用，可以用 rest 参数代替。
4. 不能使用yield命令，因此箭头函数不能用作 Generator 函数。
5. 箭头函数没有原型对象prototype

## 页面大量图片，如何优化？

1. 图片懒加载。在页面的未可视区域添加一个滚动事件，判断图片位置与浏览器顶端的距离与页面的距离，如果前者小于后者，优先加载。
2. 如果为幻灯片、相册等，可以使用图片预加载技术，将当前展示图片的前一张和后一张优先下载
3. 如果图片为css图片，可以使用CSSsprite，SVGsprite等技术。
4. 如果图片展示区域小于图片的真实大小，应在服务器端根据业务需要先进行图片压缩，图片压缩后大小与展示一致。（减小图片内存大小）