# Js基本数据类型和引用数据类型及其内存分配

## 1.1基本数据类型有哪些？

1、undefined 类型表示不存在定义，声明变量但没有初始化，这个变量的值就是undefined； 注意：在任何一个引用变量值设置为undefined都是错误的

2、null 类型，表示一个值被定义了，定义为空值； 使用场景为 定义变量准备在将来用于保存对象；所以引用值可以是null而不会是undefined；

undefined和null的区别：js诞生的时候只设置了null作为“无”的值。最初的设计是null是表示一个“无”的对象，转为数值时为0； | undefined表示“无”的原始值，转为数值时为NaN；红宝书上说引入undefined就是为了正式的区分空对象指针与未经初始化的变量，变量设置为null就是空对象指针，没有设置就是未经初始化！

3、Boolean类型，字面值为true和false

4、number类型，字面量格式可以是十进制、八进制（八进制第一位必须是0）、十六进制（前两位必须是0x）

5、String类型 由零个或多个16位Unicode字符组成的字符序列

6、symbol类型，ES5 的对象属性名都是字符串，这容易造成属性名的冲突。比如，你使用了一个他人提供的对象，但又想为这个对象添加新的方法（mixin 模式），新方法的名字就有可能与现有方法产生冲突。如果有一种机制，保证每个属性的名字都是独一无二的就好了，这样就从根本上防止属性名的冲突。这就是ES6 引入Symbol的原因

## 1.2 引用数据类型有哪些

引用类型统称为object类型，细分的话有：Object 类型、Array 类型、Date 类型、RegExp 类型、Function 类型 等。

## 1.3基本和引用数据类型的区别

主要体现在两种类型在添加属性时候的区别

引用类型可以动态的给添加属性

但是基本类型的值是不可变也不可以复制的

基本数据类型的值是不可变的，指的是，当一个值被赋值给一个变量后，这个值在栈中存储的实际位置上的值是不可变的。

例如

Var a = 3

a = 4

console.log(a) //会输出4

a的值不是改变了吗？

请注意，这里的逻辑是，当处理a = 3的时候，会在栈中开辟一块空间，这块空间上放上一个值为3，当a=4的时候，并不是把这块空间上的3修改为4，而是另外开辟一块新的空间，在这块新的空间上放一个值为4，但是原来地址上的值3 ，始终都没有改变。在合适的时间会被js的内存机制回收掉。

## 1.4 基本和引用的变量内存分配

##### 变量的不同内存分配

原始值：存储在栈（stack）中的简单数据段，也就是说，它们的值直接存储在变量访问的位置。这是因为这些原始类型占据的空间是固定的，所以可将他们存储在较小的内存区域 – 栈中。这样存储便于迅速查寻变量的值。栈有一个很重要的特殊性，就是**存在栈中的数据可以共享。**

引用值：存储在堆（heap）中的对象，也就是说，存储在变量处的值是一个指针（point），指向存储对象的内存地址。这是因为：引用值的大小会改变，所以不能把它放在栈中，否则会降低变量查寻的速度。相反，放在变量的栈空间中的值是该对象存储在堆中的地址。地址的大小是固定的，所以把它存储在栈中对变量性能无任何负面影响。

# 2 清空数组的方法

**1 . 将length赋值为0**

var arr = [1,2,3];

arr.length = 0;

console.log(arr);//[]

**2 . splice splice() 方法向/从数组中添加/删除项目，然后返回被删除的项目。**

var arr = [1,2,3];

arr.splice(0,arr.length);

console.log(arr);//[]

**3 . 使用 空数组 [] 赋值**

实际上是将arr指向了一个新的数组，这个数组中是空的，所以看起来像是被清空了，实际上并不能算是清空数组了。那么之前的数组如果再没有其他的引用指向它，将被垃圾回收。

var arr = [1 ,2 ,3];

arr = [];

console.log(arr);

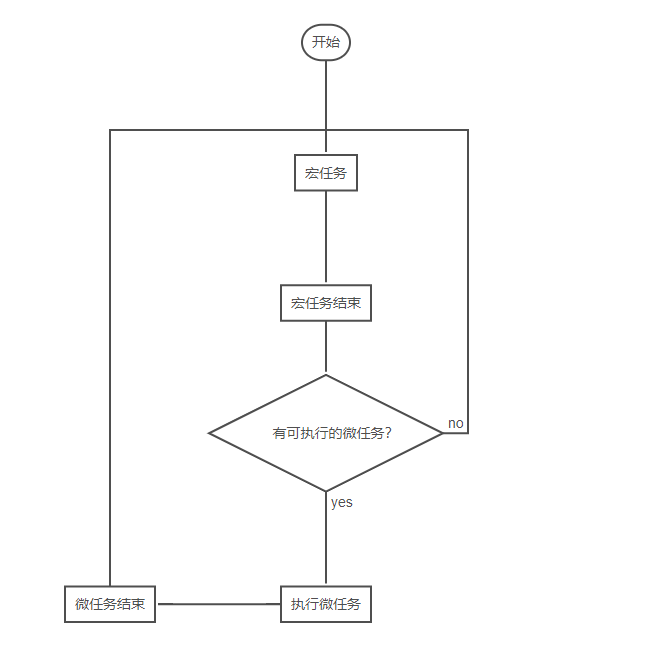
# 3 js的event loop

**问题1：为什么 js 是单线程？**

因为 js 是面向客户端的一门语言，主要是用户交互，操作dom，渲染数据。试想一下，如果是多线程，我们在一个线程删除了一个dom节点，另外一个线程添加了一个dom节点，以那个线程为主呢，就会出现混乱的情况。当然你可以说我们在操作一个dom之后加上锁，只允许一个线程操作，这样其实增加了程序的复杂度，并不是一个好办法。所以，js 是一个 单线程语言。

**问题2：单线程产生的问题？**

js 的单线程产生了一个问题，那就是必须要等待前一个程序执行完毕才执行下一个，所以将程序分为了两类：同步任务和异步任务。异步任务又可以分为宏任务和微任务。



# 4 死锁产生的必要条件

* 互斥条件：进程要求对所分配的资源进行排它性控制，即在一段时间内某资源仅为一进程所占用。
* 请求和保持条件：当进程因请求资源而阻塞时，对已获得的资源保持不放。
* 不剥夺条件：进程已获得的资源在未使用完之前，不能剥夺，只能在使用完时由自己释放。
* 环路等待条件：在发生死锁时，必然存在一个进程–资源的环形链。

# 5进程和线程的区别

* 进程：进程是系统进行资源分配和调度的一个独立单位，是系统中的并发执行的单位。
* 线程：线程是进程的一个实体，也是 CPU 调度和分派的基本单位，它是比进程更小的能独立运行的基本单位，有时又被称为轻权进程或轻量级进程。
* 进程是资源分配的最小单位，而线程是 CPU 调度的最小单位。
* 创建进程或撤销进程，系统都要为之分配或回收资源，操作系统开销远大于创建或撤销线程时的开销。
* 不同进程地址空间相互独立，同一进程内的线程共享同一地址空间。一个进程的线程在另一个进程内是不可见的。
* 进程间不会相互影响，而一个线程挂掉将可能导致整个进程挂掉。