## 2-1变量类型和计算

* **变量大的两种类型：** 值类型、引用类型（包括null，存储方式与值类型不同，存储时是以指针的方式存储）
* **变量的详细类型有几个（使用typeof判断，只能区分值类型）：**

六个 undefined number string boolean object function

* **变量的计算，强制类型转换（发生在值类型中）**

四种情景：字符串拼接、==、if语句、逻辑运算

100==’100‘ //true 0==’‘ //true null==undefined //true

var a=100 a=true a=’’; if(a) 转换a变量为布尔值

&& || ！

* **关于何时使用==和===的问题**

if（obj.a==null）可以使用==，因为此时可以理解为obj.a===null||obj.a===undefined。

剩下所有的情况，**全部使用===，**可以避免不必要的问题。

* **页面渲染过程**

onload事件触发时所有页面上的东西已经全部加载完毕。

DOMContetntLoaded事件触发时仅仅是DOM加载完毕。

* **JS自带已经封装好的函数**

Array、Boolean、Number、String、Function、Date、RegExp、Error、Math（对象）、JSON（对象）

## 2-2原型和原型链

* **构造函数 显式原型 隐式原型 继承 原型链**

首先构造一个函数Fun()，函数名称开头要大写，写完后new这个构造函数并且赋给一个新的对象a。当new构造函数时会返回到构造函数里面传参并且执行里面的程序，此时里面的this是一个空对象，执行时为this的属性赋值，之后构造函数默认会返回this，返回this后就指向了新的对象a，所以此时新的对象a就有了新的属性和方法了。此时a不光有了新的属性和方法，并且也被默认的赋予了隐式原型\_\_proto\_\_，而构造函数被默认赋予了一个显示原型prototype。这些原型也是对象，也能被赋变量和方法，也有它们的隐式原型。当查找a的一个属性时，如果在它的属性里找不到，就会自动到\_\_proto\_\_中去查找，如果还是找不到，就会去\_\_proto\_\_的\_\_proto\_\_中再找，但是不会永远循环下去，当还是找不到时，第二级的\_\_proto\_\_就为null了，js设计者把这里掐住了，通过原型向上查找的过程就形成了原型链。js里有一个特性，新对象a的隐式原型===构造函数Fun()的显式原型prototype。instanceof就是利用新特性来判断一个函数是不是一个变量的构造函数。原型链继承的原理就是让子类型的原型指向父类型的实例。即”子类型.prototype = new 父类型()”。这样新对象的隐式原型里就有了父类型的属性和方法。

## 3-1作用域和闭包

* **执行上下文（也是对变量的提升）**

在一个<script>标签中或者是一个函数中，执行程序的第一步是先把这个范围内的所有定义的变量（不包括它的值）和函数的声明（就是整个函数和它里面定义的东西）放在这个范围的最前面。在函数中除了这些还会确定this、argument的值。

* **this关键字的指向问题**

**this的值会随着函数使用场合不同而不同，但是总有一个原则，this指向的永远是调用函数的那个对象。四个场景：**

1、普通函数 this指向的是window。

2、作为对象的方法来调用 this指向的是这个对象

3、在构造函数中使用 this指向的是新对象

4、使用call、bind等方法时 this指向的是方法的第一个参数

* **作用域 作用域链**

在js中作用域只分为**函数作用域**和**全局作用域**。并没有块级作用域。

当在一个作用域中调用一个并没有被定义的变量时，这个变量就成为了**自由变量**，函数就会到上一级的作用域查找（在哪个作用域中定义这个函数哪个作用域就是父级作用域），并且一直到全局作用域中才会停止。向上查找的过程就形成了**作用域链**。

* **闭包的主要应用场景**

1、函数作为返回值

2、函数作为参数来传递

## 4-1异步和单线程

* **异步 单线程 同步**

JS语言是一种单线程语言，就是同一个时间我只能干一件事情，多的事情就要排队，我一个一个去做。所以产生了一些问题，程序有十行，从上往下一个一个去运行（**同步**），运行到第六行有一个settimeout定时器，需要等待十秒才能执行，如果执行这一行代码那么后面所有的程序都要等待十秒，**就形成了阻塞**，而且体验效果很差。所以不如先把这第六行的代码单独拿出来放一边，继续执行剩下的代码。等程序都执行完了在回头看看被单独拿出来的代码，执行他们。这就是**异步**执行。

需要异步的几个场景：settimeout()、setInterval()、网络请求（Ajax请求、img加载）、事件绑定。

## 4-5数组API和日期API

* **数组api**

forEach(function(item,index){})

every(function(item,index){})

some(function(item,index){})

sort(function(a,b))

map(function(item,index){})

filter(function(item,index){})

* **对象api**

**for**(var i **in** obj)

* **日期api**

Date.now()

dt.getTime()

dt.getFullYear()

dt.getMonth()

dt.getDate()

dt.getHours

dt.getMinutes

dt.getSeconds