**JavaScript高级教程笔记**

**[2018.1.25]**

目录

[一、 JavaScript 3](#_Toc507004867)

[二、 使用规范（请看基础笔记） 3](#_Toc507004868)

[三、 Number数值数据类型 3](#_Toc507004869)

[四、 函数 4](#_Toc507004870)

[五、 全局变量和局部变量 6](#_Toc507004871)

[六、 数组 7](#_Toc507004872)

[七、 字符串 8](#_Toc507004873)

[八、 神奇的eval用法（语法结构） 8](#_Toc507004874)

[九、 DOM 8](#_Toc507004875)

[十、 事件操作 11](#_Toc507004876)

[十一、 作用域链 14](#_Toc507004877)

[十二、 闭包 16](#_Toc507004878)

[十三、 面向对象 17](#_Toc507004879)

[十四、 面向对象特性 20](#_Toc507004880)

[十五、 正则表达式 24](#_Toc507004881)

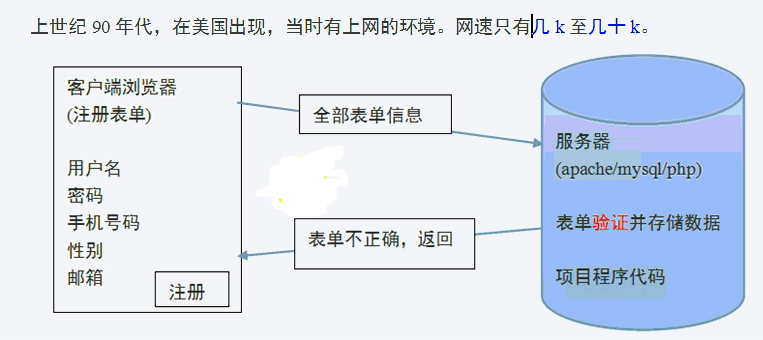
[十六、 案例：贪吃蛇游戏 27](#_Toc507004882)

# JavaScript

1. **定义**

基于事件和对象驱动，并具有安全性能的脚本语言。

该语言运行在客户端浏览器里边，也有运行在服务器端的Node.js

1. **出现背景**
2. **发明JavaScript语言的公司**

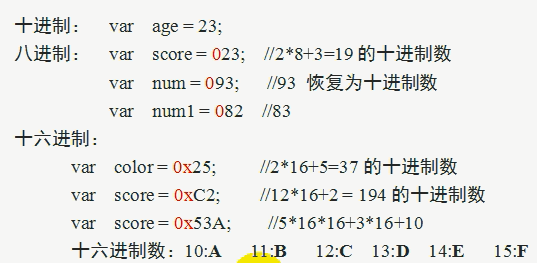
网景公司（navigator）在1995年发布了JavaScript语言，起初名字为livescript

微软公司自己的脚本语言：Jscrpit（JavaScript拷贝版本）

# 使用规范（请看基础笔记）

# Number数值数据类型

1. **各种进制表示**



1. **浮点数**

**带小数点的数**

1. **最大数和最小数**

最大：NumberMAX\_VALUE

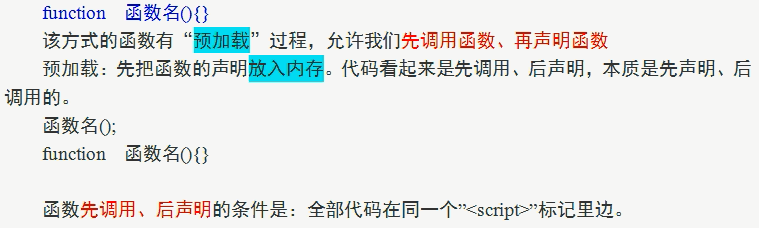
# 函数

1. **什么是函数**

**有一定功能代码体的集合**

1. **函数封装**
2. 传统方式

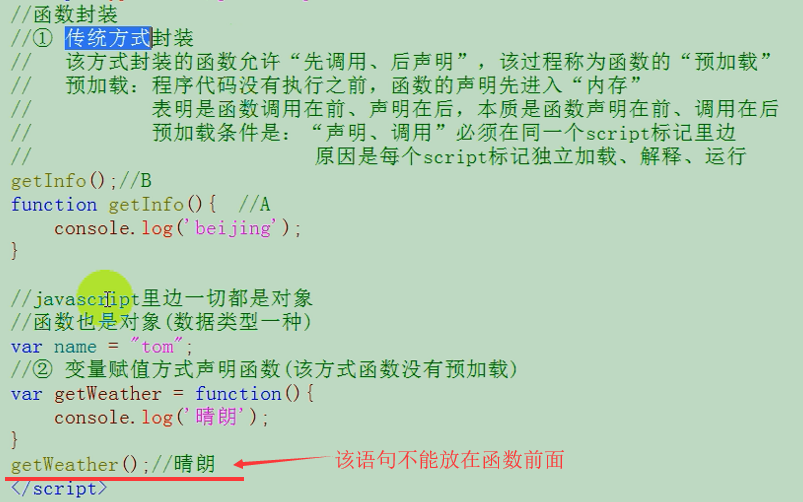
function 函数名（）{ 代码 }



1. 变量赋值方式声明函数（匿名函数）

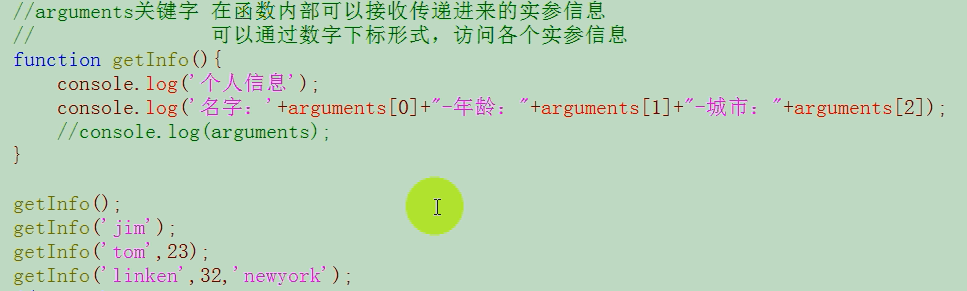


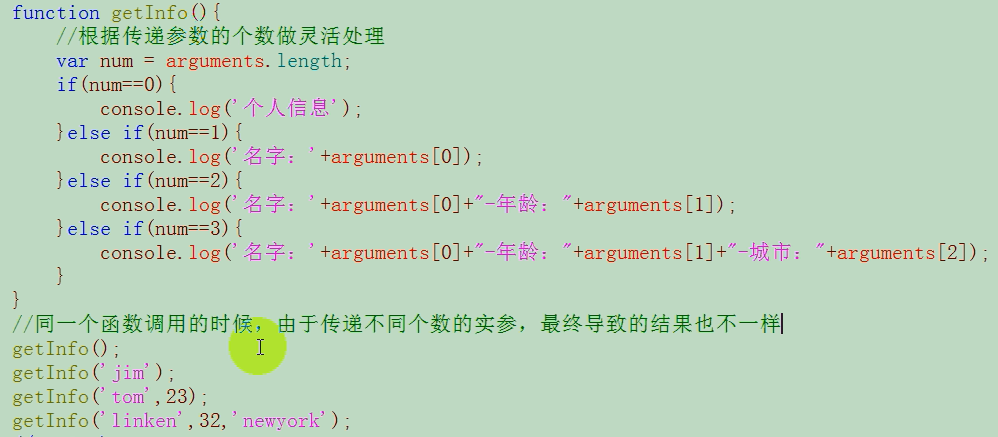
1. EG：



1. **函数中arguments关键字（模仿方法重载）**

Arguments关键字，在函数内部可以接受传进来的实参信息，可以通过数组下标形式，访问各个实参信息



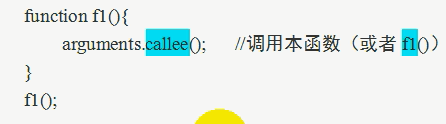


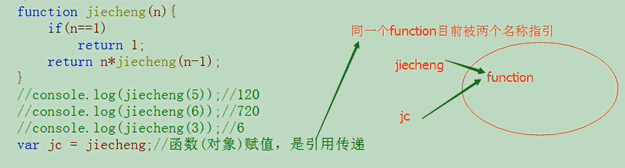
1. **callee关键字**

在函数内部使用，代表当前函数的引用（名字）。是arguments的属性

作用：自己调用自己，降低代码的耦合度。

EG：





1. **函数返回值return**

一个函数执行完毕可以通过return关键字返回一定的信息，该信息可以直接输出，也可以进行变量赋值，return本身可以接受函数的执行。

在一定意义上看，全部的数据类型（数值、字符串、布尔、对象、null）信息都可以返回（undefined类型无需返回，本身无意义）

在JavaScript里边函数除了return可以返回基本类型的信息，其还可以返回function函数。

在JavaScript里边，一切都是对象。

在一个函数内部，可以声明数值、字符串、布尔、对象等局部变量信息，言外之意就可以声明函数（函数内部还要嵌套函数），因为函数是对象，并且函数可以被return给返回出来。

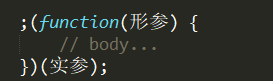
1. **函数调用**
2. 传统方式函数调用  
   函数名（）；
3. 匿名函数自调用  
   使得一下匿名函数发生执行

特点：程序代码没有停顿，立即执行

好处：可以避免变量污染，函数内的所有变量和方法被保护起来。

变量污染：同名函数和变量同时使用会出现覆盖现象

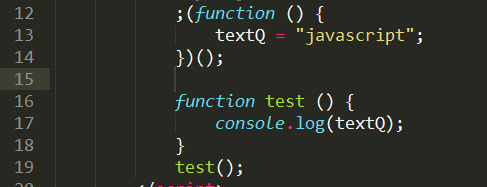
格式：



# 全局变量和局部变量

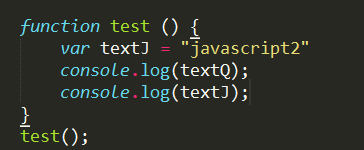
1. **全局变量**

JavaScript：在函数内部声明的全局变量，变量前不要加“var”关键字



1. **局部变量**

JavaScript：在函数内部声明的局部变量，变量前要加“var”关键字



# 数组

1. **什么是数组**

具有名称和数据类型一致的元素集合

（在实际应用中，数组内部各个元素的数据类型可以是不相同的）

1. **三种声明数组**

* var arr = [ element0, element1, ..., elementn ];
* var arr = new Array();
* var arr = new Array(element0, element1, ..., elementn);

注意：

* JavaScript数组下标都是数字；
* 1和3两种方式声明数组，各个元素不能显示设置下标
* 2方式给数组可以给设置下标（不是数组就是设置成员属性）



* JavaScript中的数组本身是对象，内部数组元素部分也有对象成员部分

1. **获取数组长度**

Arr.length;

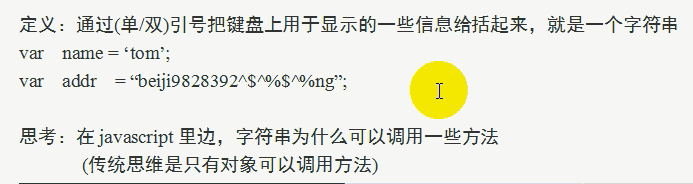
Length属性在数组里边是获得数组元素 “最大数字下标” 加1 的信息。（原因：0也算一个）

1. **数组遍历**

按照一定顺序对数组内部元素做一次仅且一次访问，称为遍历。

* 1. for循环 遍历 ：合适遍历下标是规则、连续的数组
  2. for-in遍历 ： 数组、对象都可以遍历，并且数组的下标没有具体要求

# 字符串



# 7神奇的eval用法（语法结构）

eval（参数字符串）

该eval可以把内部参数字符串当成表达式，在上下文环境中运行。

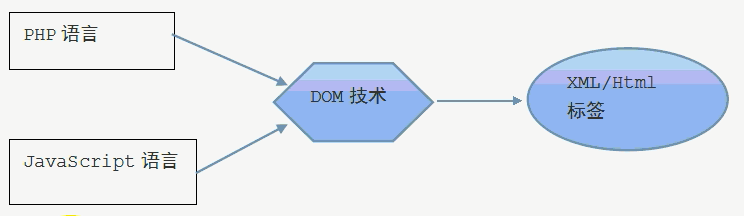
该eval（）经常用于把其他用户传递过来的字符串信息转为JavaScript的实体（对象、数组等）信息。

eval( 参数字符串 )：参数要求必须符合js语法规则。

# DOM

1. **介绍**

DOM：document object model 文档对象类型



DOM可以让我们通过JavaScript语言对html文档进行增、删、改、查操作。

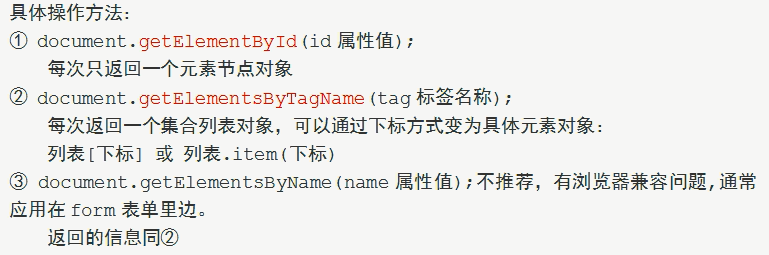
为了方便JavaScript语言通过DOM操作HTML文档比较方便

把HTML文档的各个组成内容划分为各种节点（对象）

* **文档节点（document）：**其实是HTML根节点的父节点。
* **元素节点（element）**
* **文本节点**
* **属性节点**
* **注释节点**



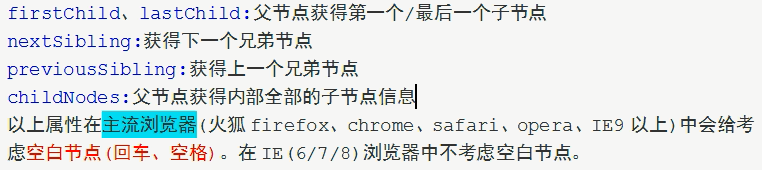
1. **元素节点获取**



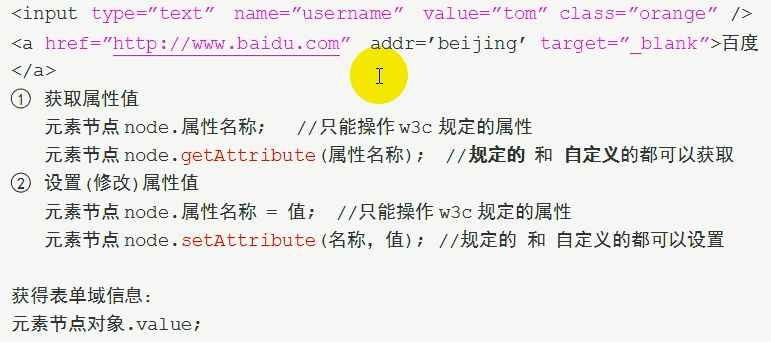
1. **文本节点获取**

需要借助div元素节点再获得其内部的文本节点

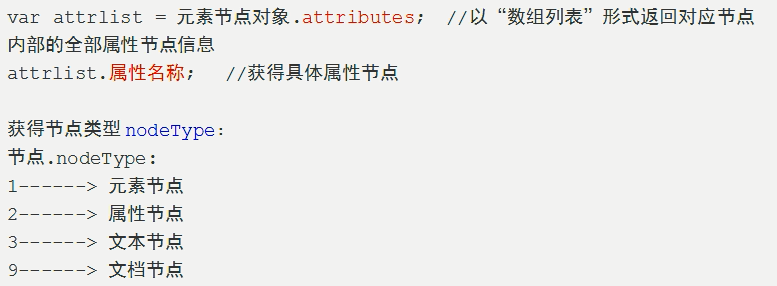
1. **子节点/兄弟节点**



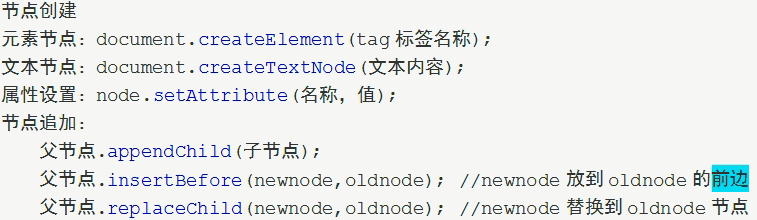
1. **属性操作**
2. 属性值操作



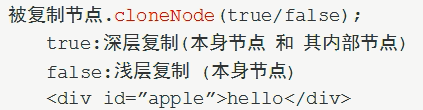
1. 属性节点获取



1. **节点创建和追加**



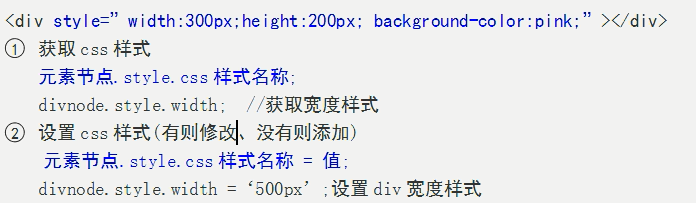
1. **节点复制操作**



1. **节点删除**

父节点.removeChild()

1. **DOM与CSS样式操作**





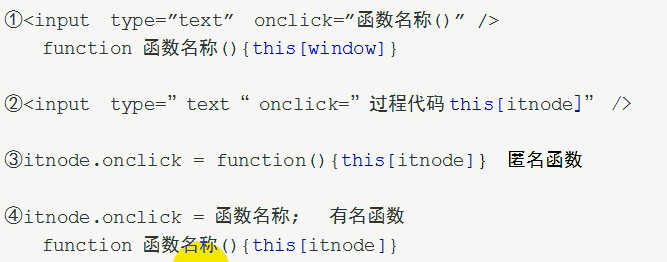
# 事件操作

1. **什么是事件**

通过鼠标、键盘对浏览器页面所做的动作就是事件。

事件一旦发生需要时间处理，该处理称为“事件驱动”，事件驱动通常有函数担任

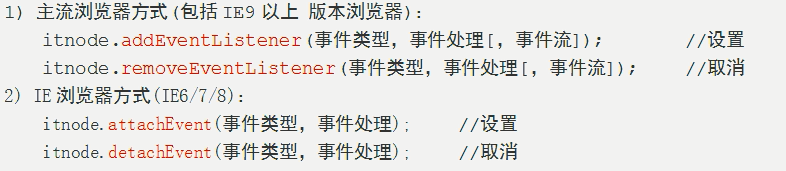
* onclick：鼠标点击
* onmouseover：鼠标移入
* onmouseout：鼠标移出
* onkeyup：键盘按下并抬起
* onkeydowm：键盘按下
* onchange：内容改变
* onblur：失去焦点
* onfocus：获得焦点
* onsubmit：表单提交

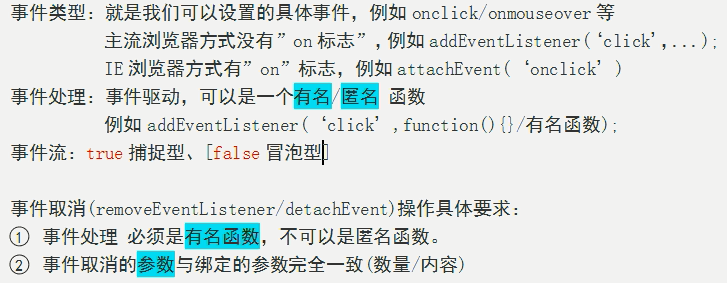
1. **设置事件**
2. DOM 1级方式设置  
   

取消1级事件：

Itnode.onclick = null;

以上是dom1级事件设置的4种具体表现，this关键字除了第一种其代表window对象，其他三种都代表事件节点对象本身

1. DOM 2级方式设置（重点）  
   



1. DOM2级事件设罝的特点：

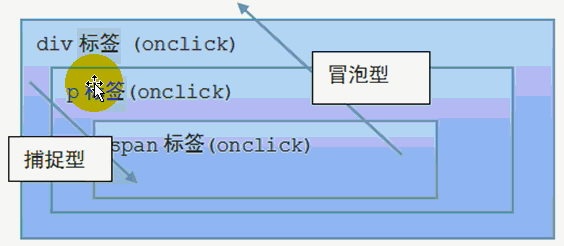
①可以为同一个对象设置多个同类型事件

②事件取消也非常灵活。

③对事件流也有很好的处理。

1. **事件流**

多个彼此嵌套元素，他们拥有相同的事件，最内部元素事件被触发后，外边多个元素的同类型事件也会被触发，多个元素他们同类事件同时执行的效果称为事件流



事件流分为两种类型：

* 冒泡型：事件从内部往外部依次执行，
* 捕捉型：事件从外部往内部依次执行，

**IE浏览器**从开始到后期事件流始终是“冒泡型”的，直到IE9以后版本两种都开始支持。 **网景的Navigator浏览器**(现在火狐浏览器的许多血统来源于navigator浏览器)一开始的 事件流是“捕捉型”。后期事件流有改进，针对捕捉型、冒泡型都可以支持。

1. **事件对象（Event）**

事件对象，每个事件(包括鼠标、键盘事件)触发执行的过程中，都有对应的事件对象，通过 事件对象可以获得鼠标相对页面的坐标信息、通过事件对象也可以感知什么键子被触发执行、通过事件对象还可以阻止事件流产生、阻止浏览器默认动作。

1. 获得：
2. 主流浏览器(IE9以上版本浏览器）：

事件处理函数的第一个形参就是事件对象 例如：

node, onclick = function (evt) { evt 就是事件对象}

addEventListener( 类型，function( evt ){ } / 函数名字〉；

function 函数名称( evt ) { }

红色的evt就是事件对象

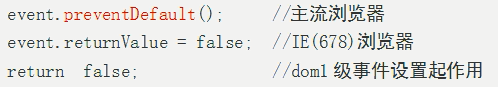
1. IE (6/7/8)浏览器

node, onclick = function() {window, event 事件对象}

全局变量event就是事件对象

全局变量直接上级对象是window。可以通过window访问全局变量信息。

window.document.getElementByld()

1. 事件对象的作用
   1. 获取鼠标的坐标信息  
      event.clientX / clientY //相对DOM区域坐标  
      event.pageX / pageY //相对DOM区域坐标，给考虑滚动条距离  
      event.screenX / screen //相对屏幕坐标
   2. 阻止事件流  
      event.stopPropagetion() //针对主流浏览器  
      阻止事件流，到该事件后不再继续执行其他事件，只阻止冒泡型的事件流，才有意义！
   3. 阻止浏览器默认动作  
       浏览器默认动作，注册form表单页面，提交表单的时候，浏览器的页面会根据action 属性值进行跳转，这个动作称为“浏览器默认动作'  
      
   4. 感知被触发键盘键子信息  
      event.keyCode 获取键盘对应的键值码信息  
      通过实践触发时候获得keyCode数值码信息可以对应键盘的键子信息。
2. **加载事件onload**
3. 什么是加载事件：  
   js代码执行时候，需要html&css的支持，就让html代码先执行(先进入内存），js代码后执行js代码在最后执行的过程就是“加载过程”，通常通过“加载事件”实现加载过程  
   加载事件onload可以保证js代码后于html&css执行，其要在最后执行。

具体设置：  
<body onload=” 加载函数” >

window, onload =匿名/有名函数；//推荐

# 作用域链

1. **定义**

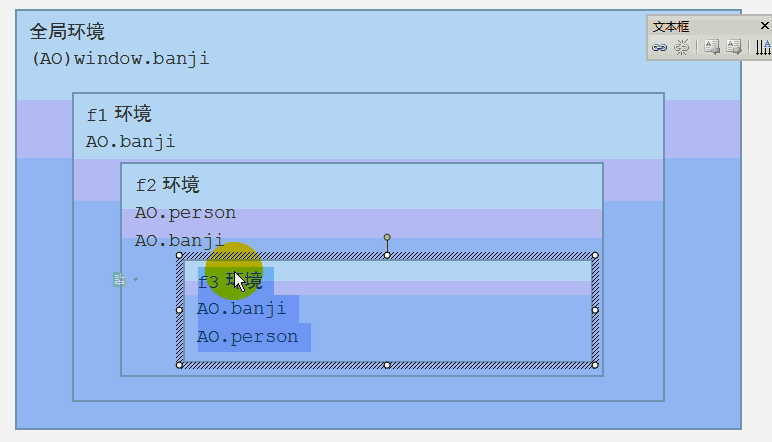
变量在当前环境now、内部环境f1、内部深层环境f2/f3....都起 作用的现象形成了一个链条，这个P条就称为变量的"作用域链"

1. **作用域链的作用**
2. 变量必须“先声明、后使用”  
    函数可以“先使用、后声明”，原因是函数有“预加载”过程（函数声明先于其他执行 代码进入内存）。本质还是函数的声明在前，使用在后。
3. 内部环境可以访问外部环境的变量，反之不然  
   环境：每一个函数内部都是一个环境，最外边是全局环境  
   两种类型： 函数环境、全局环境。
4. 变量的作用域是声明时决定的，而不是运行时
5. **AO活动对象**

AO：Active Object 活动对象（抽象的）

执行环境：

* Js代码执行是有环境的（全局环境、函数内部环境）
* 该环境定义了其他有权访问的其他数据
* 环境有一个与之关联的“活动对象AO”
* 全局环境是最外围的环境，活动对象就是window对象
* 执行环境中的代码执行完毕后就被销毁了

Eg：

1. **变量的类型及优先顺序**

执行环境可以访问变量的类型和优先顺序：

内部变量 => 内部函数 => 形参 => 外部变量（变量、函数、形参)

1. **重新认识全局变量和局部变量**
   1. 全局变量

声明：

* 在函数外部声明的变量
* 在函数内部不能使用var变量的变量（提前是函数执行之后才起作用）

重新认识：

全局变量是活动对象window的成员信息。在访问的时候。Window是否设置都可以，如果设置window可以提高代码可读性

* 1. 局部变量

声明：

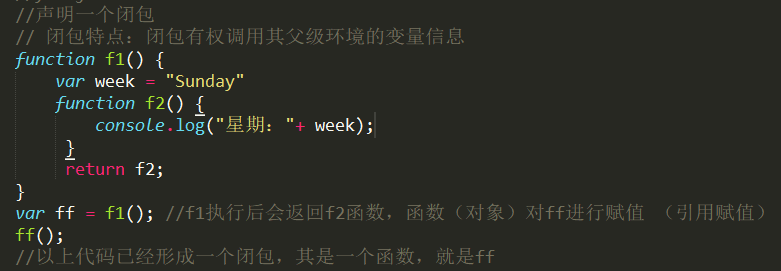
* 在函数内部通过var声明的变量就是局部变量

重新认识：

局部变量相对概念上看，在一定范围内可以看做是“全局变量”。其在本环境、内部环境、内部更深层环境都可以被访问

# 闭包

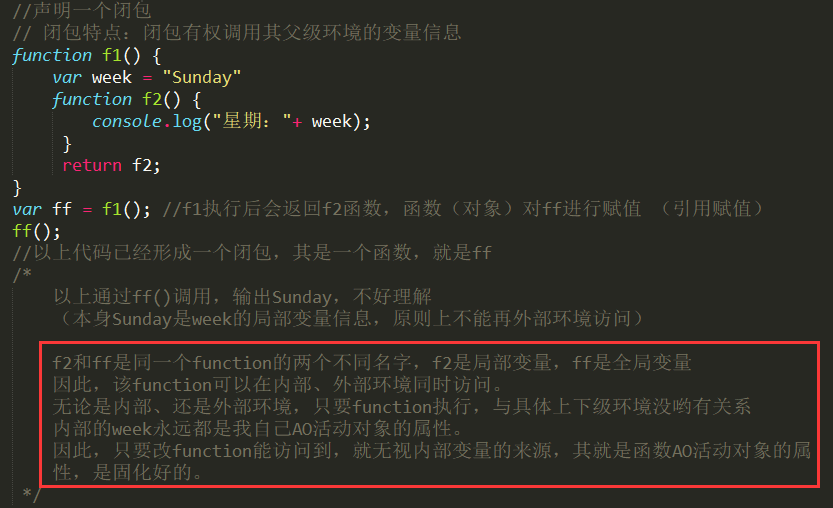
1. **什么是闭包**

闭包就是一个函数，两个函数彼此嵌套，内部函数就是闭包。形成闭包条件是内部函数需要通过return给放回出来。  


1. **闭包特点**

闭包有权调用其父级环境的变量信息。父级环境的信息已经固化为了本身AO的成员了.

闭包为什么可以调用父级环境的变量信息：

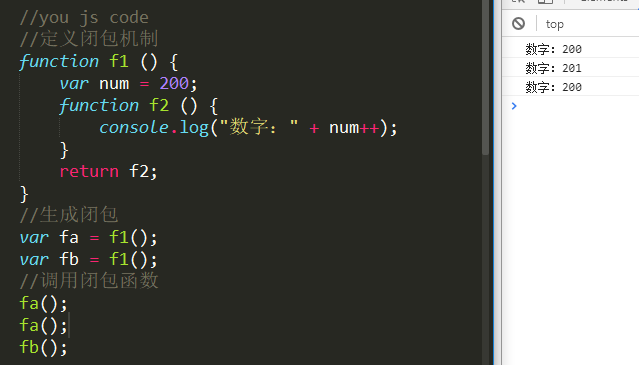


1. **闭包使用规则**

同一个闭包机制可以创建多个闭包函数，它们彼此没有联系，都是独立的。

并且每一个闭包都可以保护自己的个性化信息。

从以下例子可以看出来：



同一个闭包机制生成的多个闭包可以通过以下图理解（彼此独立、没有联系、内补各有各的信息）



1. **闭包案例**
   1. 闭包生成数组元素

失败案例，生成的多个function都访问同一个i变量，效果与预期不符

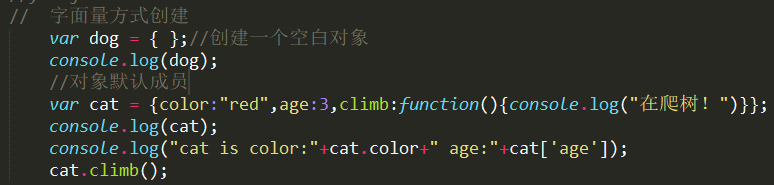
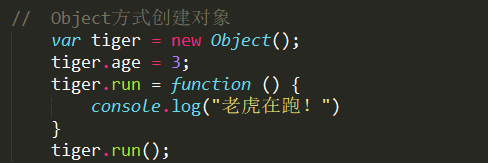
成功案例，生成的多个function都访问不同的下标

* 1. 闭包事件操作

# 面向对象

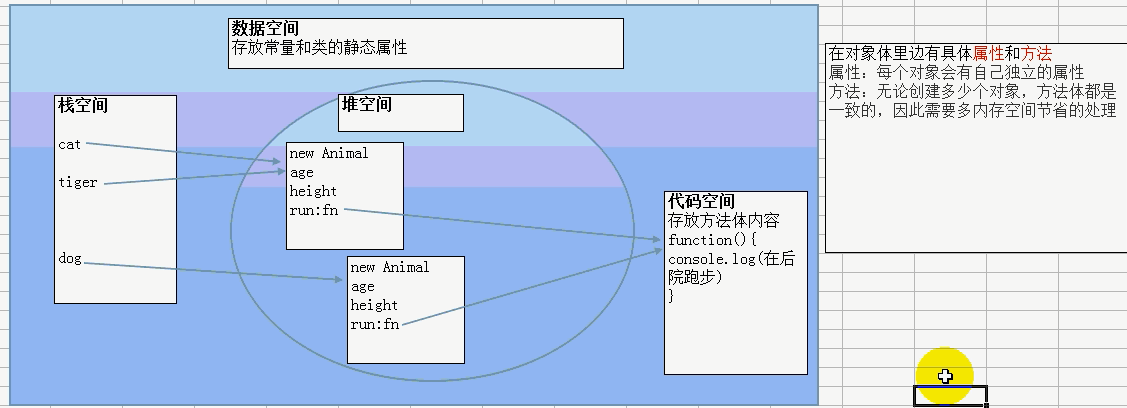
JavaScript中没有类的概念，可以直接创建一个对象（或通过函数）出来，对象可以有默认成员，后期也可以给对象丰富成员出来。

1. **创建对象**

* 字面量方式创建  
  var obj = { 名称：值，名称：值，名称：值…… }  
  
* 构造函数创建对象  
  var obj = new 函数名（）；  
  
* Object方式创建对象  
  var obj = new Object（）；  
  

1. **对象在内存分配**

与对象有关的内存空间：

1. **栈内存**  
   存放的数据大小比较小，一般大小固定的信息适合存放在该空间，例如：整型、boolean、对象引用（名称）  
   var dog = new Animal（）；//对象名称
2. **堆空间**该空间存储的数据比较，空间较大，一般数据长度不固定的信息在该空间存放，例如：string、Array、对象实体。  
   var dog = new Animal（）；
3. **数据空间**该空间存放常量、类的静态属性
4. **代码空间**存放函数体代码、方法体代码  
     
   上图代码创建的对象在内存中的具体分布如下：  
   
5. **对象调用其他函数或其他对象成员**

定义个一个函数名将其函数的引用赋值给该变量。

1. **构造函数与普通函数**

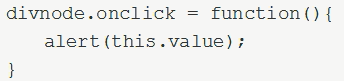
构造函数与普通函数的区别：

没有区别，就看使用，new就是构造函数，函数名（）就是普通函数调用

1. **函数的各种执行方式**
2. 普通函数调用（包括匿名函数自调用）
3. 构造函数执行new函数
4. 作为对象的成员方法执行
5. 通过call或apply执行  
   call和apply使得函数执行的时候可以控制变量污染的风险。  
   （对象调用其他函数/方法的时候）  
   函数的执行方式：call和apply方式：

函数.call(函数内部this的指引，[实参1，实参2，·····]);

函数.apply(函数内部this的指引，[实参1，实参2，·····]);

1. **this都是谁**
2. 在函数/方法 里边this代表调用该函数/方法的当前对象
3. 在事件中，this代表元素节点对象  
   
4. 代表window。
5. 可以任意代表其他对象  
   在call和apply使用的时候，可以任意设置被执行函数内部this的代表
6. **获取构造器**

构造器：使用什么元素实例化的对象，元素就称为该对象的构造器

var cat = new Animal（）；

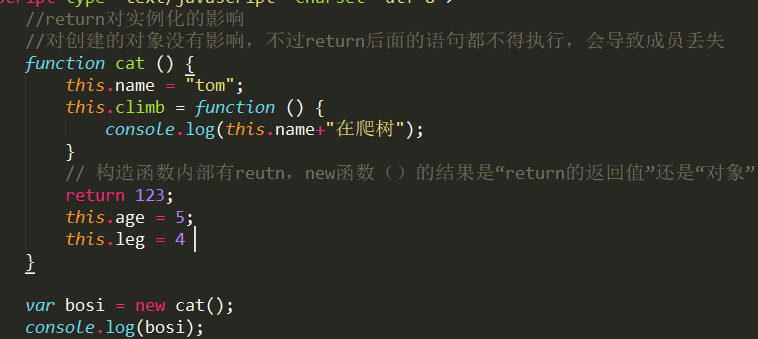
Animal就称为cat对象的构造器。

获取构造器方式：

对象.constructor;

1. **Return对实例化对象的影响**

对创建的对象没有影响，不过return后面的语句都不得执行，会导致成员丢失



# 面向对象特性

三大特性：封装、继承、多态

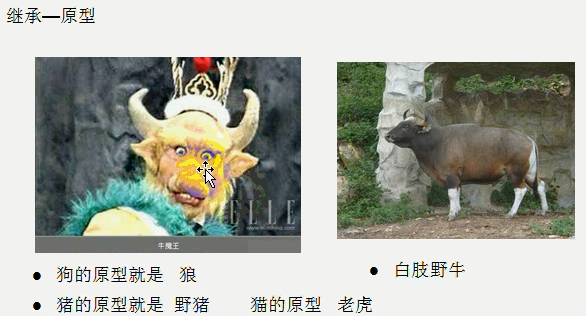
1. **封装特性**

在JavaScript里面，封装只提醒public、private

1. **Public（公开的）**，在构造函数里面通过this关键字声明的成员都是公开的，函数内部、外部都可以调用
2. **Private（私有的）**，在构造函数里面声明局部变量，就是私有成员
3. **继承特性**
   1. **原型继承介绍**

JavaScript里面：（构造）函数可以继承另外一个对象，构造函数实例化出来的对象除了本身成员外还会拥有被继承对象的的成员。

原型继承关键字：prototype



* 1. **实现原型继承**

1. 函数.prototype.成员名称 = 值； //继承单一成员
2. 函数.prototype = 对象； //继承一个对象



* 1. **原型继承注意**

1. 对象和单一成员同时继承，需要西安机场对象，再继承成员
2. 多个对象同时继承，最后对象起作用
3. 继承成员名称与本成员名称一致，本成员的有效。（重写）
4. **对象访问成员的类型及顺序**

对象本身成员 》 构造函数内部成员 》 构造函数继承对象本身成员 》 构造函数继承对象的构造函数内部成员

1. **原型链**

什么是原型链：

构造函数 继承 一个对象

继承对象也有构造函数，其构造函数还去继承其他对象

其他对象也有构造函数，构造函数还去继承其他对象

其他对象也有构造函数，构造函数还去继承其他对象

如此反复。。。。。

形成一个继承的链条，称为“原型链”，原型链顶端是Object对象

1. **对象遍历级成员类型判断**

for-in遍历

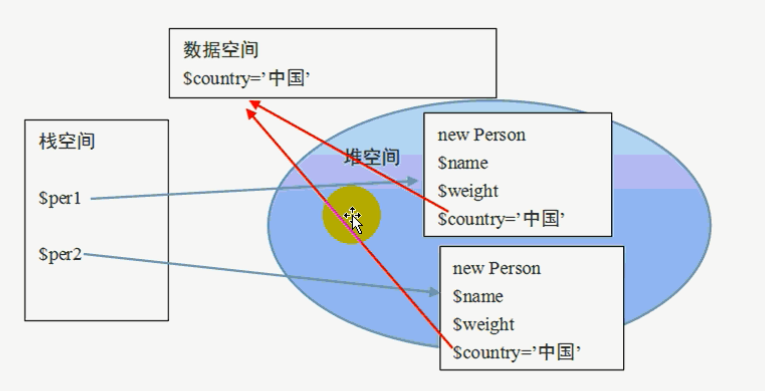
1. **原型衍生继承-复制继承**

特点：继承非常灵活，可以根据实际情况需要为对象去继承成员。如果有的对象不要额外成员，就不继承即可。同时继承允许同时继承“多个对象”



1. **静态成员**

如果一个类里面，有这样的属性，其值不随便发生变化，全部对象都共用一个值，该属性就声明static静态的，其在内存中无论创建多少个对象（或者说其与具体对象无关）都值占据一份空间。



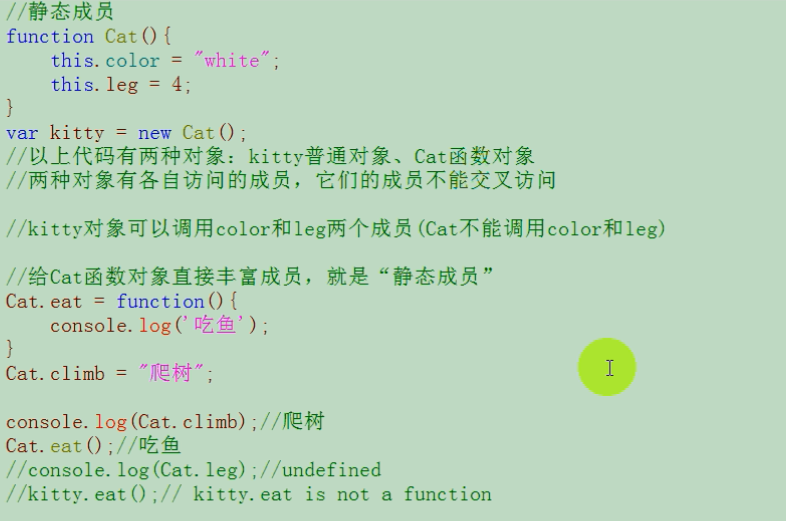
为什么使用静态成员

1. 静态成员归属到类本身，可以明显的加快访问速度。
2. 节省内存

一个类，每实例化一个对象，在堆空间都要给每个成员分配内存空间，如果有一个成员无论创建多少个对象其的信息值也不发生变化，为了节省内存空间的使用，就把这样的成员设置为“静态成员”。其在内存中就只占据一份空间

在JavaScript里面如果一个成员是通过函数直接调用，就称其为“静态”

函数对象和普通对象两种对象有各自访问的成员，它们不能交叉访问。



1. **异常Exception**
2. **什么是异常**

异常是程序运行时的一种错误。

在没有异常的时候，程序遇到“问题、错误”通常就停止运行，给用户了一种不好的体验

使用异常后，即使程序有运动错误，我们也要把这个程序完整运行完毕，以便给用户好的使用效果！

1. **多态**

多态就是多种状态

JavaScript体现多态

1. 在函数内部可以通过arguments关键字进行多态处理，传递一个不同的参数，通过arguments判断，进而有不同的处理。
2. this关键字可以体现多态。其实可以任意代表其他对象（call和apply的使用）

# 正则表达式

1. **什么是正则表达式**

正则表达式就是记录文本规则的代码

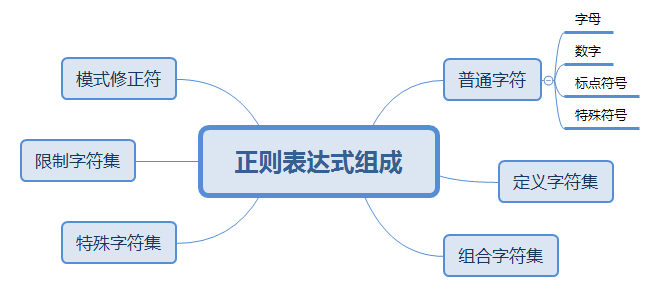
正则表达式是由普通字符（例如字符aA到zZ）以及特殊字符（称为元字符）组成了文字模式

正则表达式作为一个模板，将某个字符模式与所搜索的字符串进行匹配。

在处理程序与网页是，经常会有查找或替换符合某些复杂规则的字符串的需要。

1. **正则表达式作用（用途）**
2. 表单域验证
3. 网络爬虫（信息采集）
4. 获取天气预报信息（筛网）
5. 内容替换、获取
6. **使用正则表达式**

正则表达式组成内容：



1. **定义字符组成**

**[ a-e ]** 代表a到e这些字符中你的某一个字符

**[a-z]**26个字母之一

**[aeiou]**表示aeiou这5个字符其中的某一个字符

**[a-zA-Z]**表示大写、小写字母中的某一个字符

**[a-zA-Z0-9\_-]**

**[0-9]**表示0到9之间任意一个数字

1. **特别字符组成**

**$**：匹配输入字符串的结尾位置

**^（托字符）**:表示字符串的开始位置（托字符）

**()**：标记一个子表达式的开始和结束位置

**\***：其前面那个单元出现0次或以上（任意次数）

**+**：其前面那个单元出现1次或以上

**?**：其前面那个单元出现0次或1次

**.点**：匹配除换行符\n之外的任何单字符号

**\**：这个符号是用来转义的

**|**：指明两项之间的一个选择

^ 与 $ 的深层含义：

* 存在于全部的正确的正则表达式里边
* 可以保证正则对目标字符串的 “完整” 匹配
* 在有些场合要求正则匹配必须是 “完整匹配”

（ ）模式单元：作用：

* 提高表达式优先级
* 从一个大字符串里拆分小的字符串出来

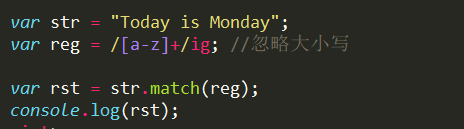
1. **模式修正符组成**

**var 模式名称 = /模式内容/模式修正符**

模式修正符：

**i -------->忽略大小写  
 g-------->全局匹配（进行全局匹配的时候不给体现“模式单元”内容）**

注意：i与g可以同时使用！



1. **（）小括号使用**

作用：

1. 提高子表达式优先级
2. 从打的字符串中拆分小的内容出来（模式单元）
3. 小括号模式单元，重复使用

{ m } \* + 表示前面的单元重复出现多次，信息的数据类型一致即可

\s 重复匹配第s个小括号模式单元内容，匹配和被匹配内容的数据类和大小都要一致

1. **（）小括号计数方法**

横向：从左向右依次计数

纵向：从外向内部依次计数

横向-纵向：横向1（纵向）-----横向2（纵向）------横向3（纵向）

1. **组合字符**

**\d**：表示数字[ 0-9 ]

**\D**：表示非数字[ ^0-9 ]

**\w**：表示字母、数字、下划线其中一个[a-zA-Z0-9\_-]

**\W**：非字母、数字、下划线

**\s**：表示空格

**\S**：表示非空格

**\b**：表示单词边界

**\B**：表示非单词边界

1. **限制字符**

**{m}**：其前面一个单元严格出现m次

**{m,}**：其前面一个单元至少出现m次，至多不限制

**{m,n}**：其前面一个单元至少出现n次，至多m次

**[^表达式 ]**：中括号内有^（托字符）是取反的意思

1. **正向预查**

辅助条件都在查找内容的右边

1. **正向匹配**

我们要要找出来的内容，右边必须出现指定的信息，并且其是辅助条件，不是结果的一部分。

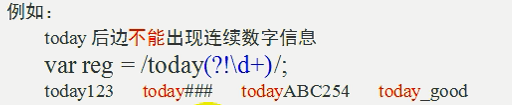
**var reg = /pattern(?=模式 )/；**蓝色是辅助条件（可以通过具体“正则模式”进行设置）



1. **正向不匹配**

我们要要找出来的内容，右边不能出现指定的信息，并且其是辅助条件，非结果的一部分。

**var reg = /pattern(?！模式 )/；**蓝色是辅助条件（可以通过具体“正则模式”进行设置）



1. **反向预查（只有PHP支持）**
2. **反向匹配**

左边必须出现指定的信息，并且其是辅助条件，非结果的一部分。

**var reg = / (?<=模式 ) pattern /；**蓝色是辅助条件（可以通过具体“正则模式”进行设置）

1. **反向不匹配**

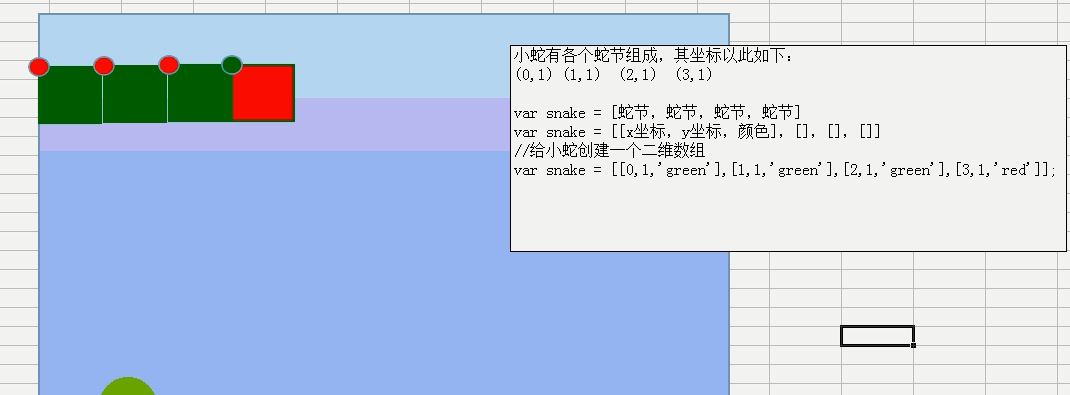
左边不能出现指定的信息，并且其是辅助条件，非结果的一部分。

**var reg = / (?<！模式 ) pattern /；**蓝色是辅助条件（可以通过具体“正则模式”进行设置）

# 案例：贪吃蛇游戏

涉及技术点：面向对象、DOM操作、事件操作、定时器

1. 绘制地图
2. 绘制食物
3. 绘制蛇身



1. 小蛇移动
2. 控制小蛇移动  
   （通过“上下左右”键的键盘事件（onkeydown）事件来完成）
3. 小蛇吃食物  
   头部碰到食物（两个坐标一致）  
   原食物消失，在其他为准重新生成一个食物  
   吃食物增加蛇节