



老马经典JavaScript教程

@马伦-flydragon

课程内容简介

- 第一章：开发环境搭建
- 第二章：HTML5各种标签及语义化
- 第三章：CSS3及整站开发
- 第四章：JavaScript基础
- 第五章：JavaScript高级
- 第六章：移动端开发[css3 + html5]
- 第七章：前端高级框架
- 第八章：NodeJs
- 第九章：前端 workflow 及工程化

老马自我介绍

老马联系方式

QQ : 515154084

邮箱 : malun666@126.com

微博 : <http://weibo.com/flydragon2010>

百度传课 :

<https://chuanke.baidu.com/s5508922.html>



第一节：认识JavaScript

01

什么是JavaScript

JavaScript脚本语言
JavaScript的作用
JavaScript运行的范围
无敌的JavaScript

02

JavaScript历史

JavaScript的历史演变以及
它与版本

03

JavaScript组成

ECMAScript 规范
DOM
BOM
NodeJS
....

JavaScript是？

01

世界上最流行的脚本语言

JavaScript是世界上用的最多的脚本语言。脚本语言：不需要编译，直接运行时边解析边执行的语言。

02

增强浏览器交互

JavaScript用的最多的就是改善用户与浏览器上浏览的网页进行交互，增强用户体验。

03

运行在服务器端的JS

NodeJS的存在，让JavaScript可以直接运行在后台，让前端开发工程师直接占领后台开发人员的领地，极大拓展了前端开发人员的领域。让js无所不能。

04

移动端的JavaScript

HTML5让JavaScript赋予了神一般的能力，穿透了所有的端（ios、android、windows、mac、linux...）

JavaScript历史

- 34岁的系统程序员
Brendan Eich，1995年4月，网景公司录用了他
- 1995年5月，网景公司做出决策，未来的网页脚本语言必须"看上去与Java足够相似"，但是比Java简单，使得非专业的网页作者也能很快上手。
- 他只用10天时间就把Javascript设计出来。



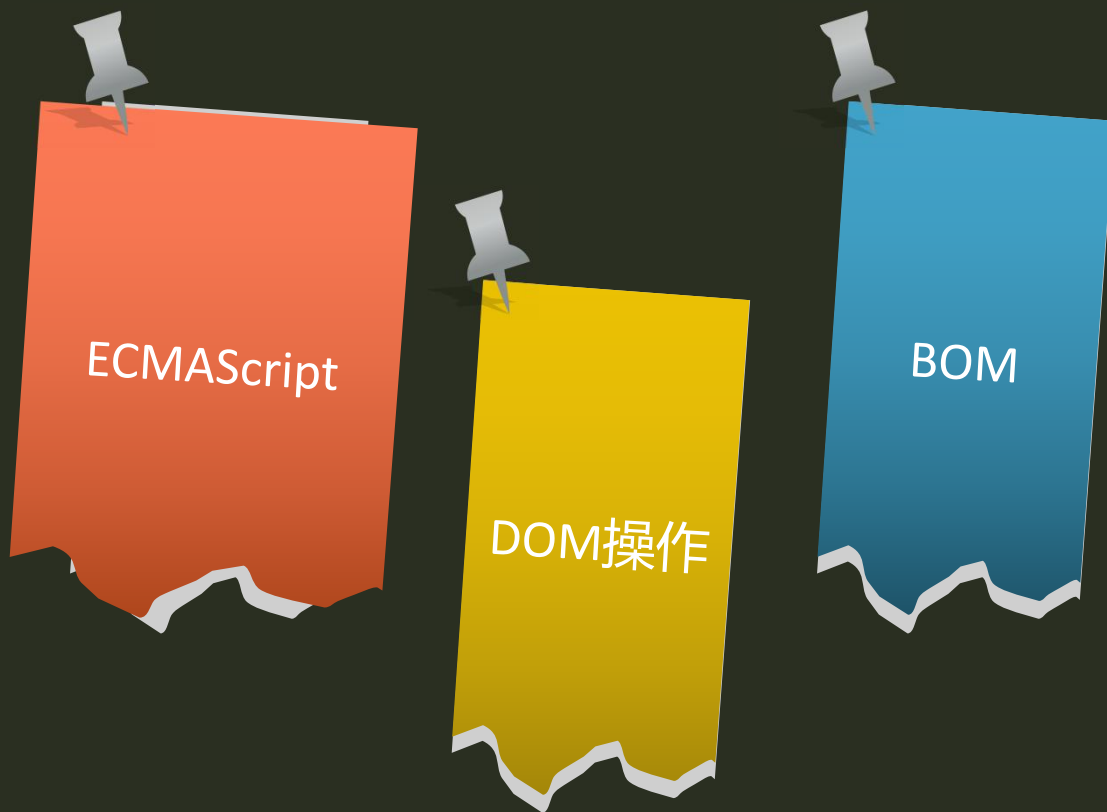
- (1) 借鉴C语言的基本语法；
- (2) 借鉴Java语言的数据类型和内存管理；
- (3) 借鉴Scheme语言，将函数提升到"第一公民"（first class）的地位；
- (4) 借鉴Self语言，使用基于原型（prototype）的继承机制。

JavaScript历史

- 1995.2月 Netscape公司发布LiveScript，后临时改为JavaScript，为了赶上Java的热浪。
- 欧洲计算机制造商协会（ECMA）英文名称是European Computer Manufacturers Association
- 1997年，以JavaScript 1.1为基础。由来自 Netscape、Sun、微软、Borland 和其他一些对脚本编程感兴趣的公司的程序员组成的TC39（ECMA的小组）锤炼出了 ECMA-262，也就是 ECMAScript 1.0。
- 1998年6月，ECMAScript 2.0版发布。
- 1999年12月，ECMAScript 3.0版发布，成为JavaScript的通行标准，得到了广泛支持。
- 2007年10月，ECMAScript 4.0版草案发布：分歧太大，失败告终。
- 2009年12月，ECMAScript 5.0版正式发布
- 2015年6月17日，ECMAScript 6发布正式版本，即ECMAScript 2015。

JavaScript的组成

- ECMAScript : JavaScript的语法。
- DOM : JavaScript操作网页上的元素的API
- BOM : JavaScript操作浏览器的部分功能的API
- Nodejs



第二节：引入JavaScript

- 行内JavaScript脚本
- Script标签
- 外部脚本
- 非常类似于CSS
 - 行内样式：写在标签上的CSS
 - 嵌入样式：style标签
 - 外部样式：link标签引入的CSS文件

JavaScript CSS HTML关系

- 结构：
- 样式：
- 行为：

Script标签的使用

- 在HTML页面中的所有JavaScript的脚本都必须放到Script标签中。
- 例如：

```
<script type="text/javascript">  
console.log("hello ");  
</script>
```

script标签的属性

属性	值	描述
async	async	立即异步下载脚本，但是不影响下载HTML（仅适用于外部脚本）。
charset	utf-8...	规定在外部脚本文件中使用的字符编码。
defer	defer	规定是否对脚本执行进行延迟加载，直到页面加载为止。
src	URL	规定外部脚本文件的 URL。

```
<script type="text/javascript" defer="defer" charset="utf-8" ></script>
```

```
<script type="text/javascript" src="" async="async" charset="utf-8" ></script>
```

<!--蓝色部分都可以直接省略-->

外部引用JavaScript脚本文件

- 创建的JS脚本文件的后缀都是.js结尾
- JavaScript文件也是普通的文本文件
- 引入JavaScript通过script标签的src属性
- 例如：

```
<script type="text/javascript" src="script/main.js"> </script>
```

Script标签放置位置

- 放置在<head> </head>标签中。
- 如果大型网站，加载的JS文件非常多的时候，会导致，在加载JS文件的时候整个页面是空白的，造成用户浏览不友好。
- 目前大型网站优化会将JS文件不影响布局和加载整体页面的js文件放置在body的结束标签之前。

第三节：JavaScript语法

- 语法
- 数据类型

JavaScript语法

- JavaScript是区分大小写（HTML、CSS不区分）
 - a A
- 标识符规范：
 - 标识符也就是变量、函数、属性、参数等的名字。
 - 标识符的规定：
 - 第一个字符必须是字母、下划线或者美元符号\$
 - 其他的字符可以是字母、下划线、数字
 - 例如：footerBar bannerArea _userName \$home2
 - 错误例子：88Bar &demo address(Info)

直接量

- 直接量：就是在程序中直接使用的数据的值。

18 //数字

1.2//小数

"web.sd.cn"//字符串

true false//布尔值

{ a: 8; } //

[] //数组

JavaScript注释

- 行内注释：//注释内容
- 多行注释，跟CSS一致。
- /*
 注释内容
- */

JavaScript语句

- JavaScript语句就是开发人员让程序执行的命令。JavaScript程序由语句组成。
- JavaScript会忽略多余的空格和换行符。

```
var x = 19;
```

定义一个变量 x，它的值是 19；

= 号是赋值运算符，把19的值给x

var定义一个变量的关键字

```
var y = x + 20;
```

- 每条语句用";"结束，非必须，但是我们要求必须添加";"号。

- 例如：

```
var x,y;
```

```
x = 10;
```

```
y = 19;
```

```
y = y * x;
```

```
console.log(y);
```

变量

- 源于生活：超市，仓库
- 变量：源于数学的概念。变量就是计算机内存中存储数据的标志符，变量的标识符指向的计算机存储空间可以存储相关的数据。
- JavaScript中的变量是弱类型的，可以存储任何数据类型的数据。
- 变量命名的语法：`var a = 10;` //var是变量定义的关键字。语句的含义为：定义一个变量a,并且给a赋值 10，也就是a指向的计算机存储空间中存储了10这个值。




变量的补充

- 变量可以在声明的时候赋值，也可以稍后赋值。
 - 例如：`var a;` //定义变量a
`a= "1222";` //给a赋值字符串。
- 变量可以随时改变存储的数据，甚至类型不一致都可以。
 - `var a = 9; a = "cn";` //弱类型
 - 弱类型：变量的类型可以随时改变，不强迫类型一旦确定不能更改。
- 如果定义多个变量用 “ , ” 隔开
 - 例如：`var a, b = 9, c = 123;` //都是局部变量
- 特殊情况：`var a = b = c = 9;` //var 只能作用到a上，b、c都是全局变量。

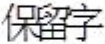
关键字和保留字

- 关键字：JavaScript语言用于程序控制或者执行特定操作的英语单词。
- 保留字：ECMAScript规范中，预留的某些词汇，以便于以后某个时间会用于关键字。
- 我们写的代码起名字的时候不能用跟关键字和保留字重名。不然会报错。

break	do	instanceof	typeof
case	else	new	var
catch	finally	return	void
continue	for	switch	while
debugger*	function	this	with
default	if	throw	
delete	in	try	



abstract	enum	int	short
boolean	export	interface	static
byte	extends	long	super
char	final	native	synchroniz
class	float	package	throws
const	goto	private	transient
debugger	implements	protected	volatile
double	import	public	



数据类型详解

- 数值类型(number)
 - 123 , 0.2 100.3 -4 -9.0
 - 10e10 => 10的10次方
- 布尔类型(boolean)
 - true false
- 字符串类型(string)
 - '1dddd' , "ssssssljldj"
- 未定义类型(undefined)
 - undefined
- null类型 (本质上是一个特殊的object)
 - null
 - var t = null; // t就是null类型
 - var m ; // m == undefined
- object类型 (引用类型)
- function类型 (函数类型)

简单类型

复杂类型

typeof取数据类型

- typeof是一种操作符。操作就是：获取变量或者字面量的数据类型。
- 使用方法：
 - typeof(变量|直接量)
 - typeof 变量|直接量

```
typeof "123"  
"string"
```

```
typeof 333  
"number"
```

```
typeof .22  
"number"
```

```
typeof {}  
"object"
```

```
typeof Function  
"function"
```


Number数值类型

- Number类型，包括了所有的数字类型，包括：小数、整数、正负数、实数等。
- 整数：`var num = 10;`
 - 合法的整数的范围： 2^{-53} 到 2^{53}
- 小数（浮点数）：`var num2 = - 1.33;`
 - 小数如果是0开头，可以省略0.例：`var b = .33;`
- 表示十六进制：以0x开头的数据。
 - 十六进制数字表示从 0 到9，A到 F
 - `var b = 0x4B;`// => 十进制: $4 \times 16 + 11$

科学表示法表示数字

- 可以使用e表示10的指数

8.88e22 //8.88*10²²

3.3e-22//3.3*10⁻²²

- 浮点数的最高精度只有17位小数。一般的情况足够

- 浮点数的原理：

- 一个浮点数由：符号位，指数，尾数表示。
- demo：+ -29 1.893 => +1.893 * 2⁻²⁹
- 浮点数不能精确表示小数，只能取近似值。
- 比如：0.3-0.2 == 0.2 - 0.1 //false

数据的表示范围

Number.MAX_VALUE

1.7976931348623157e+308

Number.MIN_VALUE

- 5e-324
- 超过以上的范围后：Infinity
 - 正Infinity
 - 负Infinity

NaN

- NaN : not a number , 非数字值 , 是数字类型 , 但是非常特殊的数字类型值。
- `parseInt("ss")` // 把ss字符串转成整数 , 此时转换失败会返回NaN
- 涉及到NaN的所有的操作都会返回NaN
- 判断是否是NaN使用isNaN方法
- 数字除以0的时候 , 都返回NaN

算数运算符

- + : 加法。 $10 + 10 = 20$
- - 减法
- * 乘法
- / 除法
- % 求余 (整除后的余数) $3\%2 = 1$

算数运算符综合练习

1. 定义变量：a,b ,c , d
2. 变量a = 10, b = 8, c = 9;
3. 变量d的值为 a 与b的 和。
4. 让变量c 的值改变为：变量d 和 变量a的乘机。
5. 让变量c的值翻倍。
6. 定义变量f 为 abcd的和，并输出f的值

复杂的算数运算

- 可以使用Math对象进行复杂的算数运算。

```
Math.pow(2,53)           // => 9007199254740992: 2 的 53次幂
Math.round(.6)           // => 1.0: 四舍五入
Math.ceil(.6)            // => 1.0: 向上求整
Math.floor(.6)           // => 0.0: 向下求整
Math.abs(-5)             // => 5: 求绝对值
Math.max(x,y,z)          // 返回最大值
Math.min(x,y,z)          // 返回最小值
Math.random()            // 生成一个大于等于0小于1.0的伪随机数
Math.PI                  //  $\pi$ : 圆周率
Math.E                   // e: 自然对数的底数
Math.sqrt(3)             // 3的平方根
Math.pow(3, 1/3)         // 3的立方根
Math.sin(0)              // 三角函数: 还有Math.cos, Math.atan等
Math.log(10)              // 10的自然对数
Math.log(100)/Math.LN10  // 以10为底100的对数
Math.log(512)/Math.LN2   // 以2为底512的对数
Math.exp(3)              // e的三次幂
```

boolean类型

- 布尔类型只有两个值：真true 和 假 false。注意区分大小写，都是小写。
- `var x = true;`//真
- `var y = false;`//假
- `var z = 10 > 9; // z=>true`
- `var m = (NaN == NaN); // m => false;`
- `var t = (1 === '1');` // t => false 三个等号比较值和类型。
- `var t = (1 == '1');` // t => true

字符串类型

- 字符串：就是由字符组成的串（羊肉）
- 定义一个字符串直接量：“字符串” '字符串'.
- 字符串都是用双引号或者单引号引起来。
- `var a = "1233"` 或者 `a = '123';`
- `var a = 123; // 数字`
- 字符串直接量必须写在同一行（EC5中可以用/链接不同行）
- `toString()`方法将其他类型转为字符串类型。
- `" + 值 => 转字符串`

字面量转义符

- 在字符串中某些特殊的字符用转义符表示。
- 比如：如何在字符串中表示双引号？
- `var a = "ssss"sss\n\t \b ss";`//错误
- `var a = "ssss\"ssssss";`//转义符： `\` => `"`

字 面 量	
<code>\n</code>	换行
<code>\t</code>	制表
<code>\b</code>	空格
<code>\r</code>	回车
<code>\f</code>	进纸
<code>\\</code>	斜杠
<code>\'</code>	单引号 (')
<code>\"</code>	双引号 (")

Undefined类型

- Undefined类型只用一个特殊的值：undefined，中文的意思就是未定义。
- undefined值出现的情况：
 - 当声明了一个变量，而未赋值的时候则变量的值为undefined。
 - 当未声明变量，直接对变量操作的时候（函数的参数未定义直接使用），变量会初始化一个值：undefined
 - 显式的给变量赋值 undefined
 - 当调用一个对象的属性或者函数执行没有任何返回值时返回undefined
- null: 代表空对象。对象类型的一个值。

Null也是object类型的一个值

- 只有一个值null：空对象。
- null表示对象为空，或者是无对象。
- `typeof null => "object"`, 所以null是一个特殊的object
- 使用方法：`var a = null;`//定义一个空对象。
- `null == undefined` //true，都表示空
- `null === undefined` // false，类型不同

Object和Function

- Object类型是JavaScript的引用类型。
- `var a = {} ; var a = new object();`
- 定义函数使用function关键字
- `function f () {}` 那么f就是function类型的实例。

第三节：运算符

- 一元运算符
- 算数运算符
- 位操作符
- 关系操作符
- 相等运算符
- 逻辑运算符
- 位操作符*

复习算术运算符

- + : 加法。 $10 + 10 = 20$
- - 减法
- * 乘法
- / 除法
- % 求余 (整除后的余数) $3\%2 = 1$

乘法运算符*

在处理特殊值的情况下，乘法操作符遵循下列特殊的规则：

- ❑ 如果操作数都是数值，执行常规的乘法计算，即两个正数或两个负数相乘的结果还是正数，而如果只有一个操作数有符号，那么结果就是负数。如果乘积超过了 ECMAScript 数值的表示范围，则返回 `Infinity` 或 `-Infinity`；
- ❑ 如果有一个操作数是 `NaN`，则结果是 `NaN`；
- ❑ 如果是 `Infinity` 与 `0` 相乘，则结果是 `NaN`；
- ❑ 如果是 `Infinity` 与非 `0` 数值相乘，则结果是 `Infinity` 或 `-Infinity`，取决于有符号操作数的符号；
- ❑ 如果是 `Infinity` 与 `Infinity` 相乘，则结果是 `Infinity`；
- ❑ 如果有一个操作数不是数值，则在后台调用 `Number()` 将其转换为数值，然后再应用上面的规则。

```
<top frame>
> 4 * -4
< -16
> 8 * NaN
< NaN
> Infinity * 0
< NaN
> Infinity * 7
< Infinity
> Infinity * Infinity
< Infinity
> "2" * "5"
< 10
```


除法的特殊情况

与乘法操作符类似，除法操作符对特殊的值也有特殊的处理规则。这些规则如下：

- ❑ 如果操作数都是数值，执行常规的除法计算，即两个正数或两个负数相除的结果还是正数，而如果只有一个操作数有符号，那么结果就是负数。如果商超过了 ECMAScript 数值的表示范围，则返回 `Infinity` 或 `-Infinity`；
- ❑ 如果有一个操作数是 `NaN`，则结果是 `NaN`；
- ❑ 如果是 `Infinity` 被 `Infinity` 除，则结果是 `NaN`；
- ❑ 如果是零被零除，则结果是 `NaN`；
- ❑ 如果是非零的有限数被零除，则结果是 `Infinity` 或 `-Infinity`，取决于有符号操作数的符号；
- ❑ 如果是 `Infinity` 被任何非零数值除，则结果是 `Infinity` 或 `-Infinity`，取决于有符号操作数的符号；

```
<top frame>
> 4 / -3
< -1.3333333333333333
> 5 / NaN
< NaN
> Infinity / Infinity
< NaN
> 0 / 0
< NaN
> 0 / 2
< 0
> 33 / 0
< Infinity
> -22 / Infinity
< -0
> Infinity / -2
< -Infinity
```

求余运算的特殊情况

- ❑ 如果操作数都是数值，执行常规的除法计算，返回除得的余数；
- ❑ 如果被除数是无穷大值而除数是有限大的数值，则结果是 NaN；
- ❑ 如果被除数是有限大的数值而除数是零，则结果是 NaN；
- ❑ 如果是 Infinity 被 Infinity 除，则结果是 NaN；
- ❑ 如果被除数是有限大的数值而除数是无穷大的数值，则结果是被除数；
- ❑ 如果被除数是零，则结果是零；
- ❑ 如果有一个操作数不是数值，则在后台调用 `Number()` 将其转换为数值，然后再应用上面的规则。

```
> Infinity % 2
< NaN
> Infinity % 0
< NaN
> 333 % 0
< NaN
> Infinity % Infinity
< NaN
> 33 % Infinity
< 33
> 0 % 3
< 0
> "3" % 2
< 1
```

自增|减运算符

- 递增运算符

```
var a,i = 0;
```

```
i++;           //i=1;自增1，相当于：i=i+1;
```

```
a = 3+(i++); //a = 4; i=2;先计算后自增
```

```
a = 3(++i); //先自增，后参与计算
```

前置自增会影响到语句的运算结果，后置自增等语句执行完成后
再对变量进行增加处理。

- 递减运算符

- 递减跟递增基本一致。

加法运算的二义性与一元加法

- 加法运算分为：字符串连接运算和算数加法运算。
- `var a = 1 + 2; // a => 3` 算数运算加法
- `var a = "我是" + "老马"; // a = "我是老马"` 字符串连接
- 当运算的变量中有字符串的时候**优先字符串连接**，后考虑算法运算。
- `var a = 1 + "22"; // => a="122",字符串连接操作`
- `var t = "23" ;`
- `var m = +t; // m => 23数字`

减法运算符

- 减法运算符：只有数学减法运算。不具备字符串链接功能。
- 比如：`var t = 3 - '2'; // => 1`
- 减法运算符也具备将变量转换为数值类型的作用

- ❑ 如果两个操作符都是数值，则执行常规的算术减法操作并返回结果；
- ❑ 如果有一个操作数是 NaN，则结果是 NaN；
- ❑ 如果是 Infinity 减 Infinity，则结果是 NaN；
- ❑ 如果是 -Infinity 减 -Infinity，则结果是 NaN；
- ❑ 如果是 Infinity 减 -Infinity，则结果是 Infinity；
- ❑ 如果是 -Infinity 减 Infinity，则结果是 -Infinity；
- ❑ 如果是 +0 减 +0，则结果是 +0；
- ❑ 如果是 +0 减 -0，则结果是 -0；

```
> 4-3
< 1
> NaN - 3
< NaN
> Infinity - Infinity
< NaN
> -Infinity - (-Infinity)
< NaN
> Infinity - (-Infinity)
< Infinity
> -Infinity - Infinity
< -Infinity
> 0-0
< 0
> 0-(-0)
< 0
> -0-0
< -0
```

关系运算符

- 关系运算符返回的都是boolean类型结果
- 大于 `>`
 - `2 > 3 //false`
 - `"32" > "223" // true`,字符串比较的是字符串的编码大小。
- 小于 `<`
 - `2 < 3 //true`
- 小于等于 `<=`
 - `var a = 22, b = 22; a<=b//true`
- 大于等于 `>=`
 - `22 >=22 //true`
 - `33>=22 //true`

关系运算符规则

- ❑ 如果两个操作数都是数值，则执行数值比较。
- ❑ 如果两个操作数都是字符串，则比较两个字符串对应的字符编码值。
- ❑ 如果一个操作数是数值，则将另一个操作数转换为一个数值，然后执行数值比较。
- ❑ 如果一个操作数是对象，则调用这个对象的 `valueOf()` 方法，用得到的结果按照前面的规则执行比较。如果对象没有 `valueOf()` 方法，则调用 `toString()` 方法，并用得到的结果根据前面比较。

操作数是布尔值，则先将其转换为数值，然后再执行比较。

```
> 4 > 8
< false
> 10 > 0
< true
> 9 <= 9
< true
> 'a' < 'A'
< false
> 'A' > 'aA'
< false
> 1 > '2'
< false
> 10 > '2'
< true
> '10' > '2'
< false
> 2 > false
< true
> -1 > false
< false
> Number(false)
< 0
```

Boolean操作符

- 逻辑非 **!**

- 逻辑非的符号用 “!” 取相反的意思。
- 比如：`var a = true; a = !a; //a = false;`

- 逻辑与 **&&**

- 逻辑与是用两个 “&&” 符号运算
- `var a = true, b = true; a&&b;//true`
- 同时为真时才返回true，其他false

- 逻辑或 **||**

- 用两个竖线连接两个变量运算（||）
- `var a = true, b = true; a||b //true`
- 两个变量只要有一个为真，那么就返回true。

逻辑运算符补充

- &&短路表达式 ***

- 由于进行逻辑运算的变量不一定
- 非Boolean类型的&&逻辑运算返回者最后一个真值的操作数（直接可以不是Boolean类型。

- 比如：

89 && (9 < 3) // => true

0 && 98 // => 0, 注意

null && 1 // => null

- 类似短路表达式，||运算符可以返回后一个假值。

- || 类型安全返回处理

- 有的时候为了防止变量为空，可
- 果。

- t = max || max-width || 500;

```
> 1 && ''
< ""
> Boolean("")
< false
> // 返回第一个假值, 或者最后一个真值
< undefined
> 1 && true && 'ssss' && 0 && 20
< 0
> 1 && true && 'ssss' && 2 && 20
< 20
> a && (+a)
✖ ▶ Uncaught ReferenceError: a is not defined
  at <anonymous>:1:1
> var a = "22";
< undefined
> a && (+a)
< 22
> a = null;
< null
> a && (+a)
< null
> 7 || "ssss" || 0 || null
< 7
> 0 || "" || 7 || "ssss" || 0 || null
< 7
```

相等运算符

- 数值相等 ==

类型可以不相同，值相等就可以。

```
var a = 3, b = "3"; a==b;//true
```

- 不等于 !=

值不等就返回true，值相等就直接返回false

```
3!=4 //true
```

```
3 != "3" //false
```

- 特殊情况：

NaN跟任何其他变量都不等，甚至不等于自己。

```
NaN != NaN//true
```

```
null == undefined ; // =>true 都是空的意思。
```

全等与不全等

- 全等就是用三个等号：`===`
- 全等就是数值相等而且类型相同。
 - `55 == "55" //true`
 - `55 === "55" //false`,值转换后相等，但是类型不同。
- 不全等：`!==`
- 只要类型不同或者值不等那么就返回true

条件运算符

- 条件运算符： 判断值？true返回值：false返回值。
- 语法： 表达式1？表达式2：表达式3；
- 结果：表达式1是真的吗？如果是：返回表达式2，反之返回表达式3；

```
var b = a > 3 ? 2 : 1;
```

//语义：a大于3 吗？如果大于返回2，如果不大于返回1；

```
var b = 判断表达式 ? 值1 : 值2 ;  
当判断表达式为 true的时候，返回值1。
```

赋值运算符

- 等号，就是最常用的赋值运算符。
 - `var a = 9;`
 - `var a = 9, b = 3, c = 4;`
- 赋值和算术运算符结合简写
 - `a = a + 10;`//当`+-*/`运算符跟自己进行运算时可用简写成
 - `a += 10;`// 跟 `a = a + 10` 效果一样。
 - `a *= 2; a -= 3; a /= 3; a %= 2;`

逗号运算符

- 逗号运算符，可用让js在一条语句中执行多次操作。

- 比如：

`var a = 2, b = 3, c = 4;`//一次性定义3个变量，并分别给变量赋值。

- 逗号运算符在用于赋值操作的时候，永远只返回最后一个值

- 比如：

`a = (2,3,4);`//a=4

运算符的优先级

- 结合性：

- 一元+，一元减，后置++，后置--，！求反，typeof，=赋值，+=，*=，/=，%= 是从右向左结合，其他都是从左向右结合。
- +a; -c; a++; c--; !true; typeof a; a=b=c=9;
- 其他的都是：从左向右结合。比如： c = a+b;

- 优先级

- 第一级：（ ） 圆括号 => (a+b)*c 先运算a+b
- 第二级：后置递增和递减 a++ b--
- 第三级：逻辑！，一元+-，前置递增和递减，typeof
- 第四级：* / %
- 第五级：二元+ -
- 第六级：< > >= <= => a+b > c*d
- 第七级：&& || => a>b || c<d
- 第八级：？： 条件运算符 => c= a && b ? 1: 2
- 第九级：赋值运算符 = += *= -= /= % =
- 第十级：逗号 a = 1, b = 2;

运算符综合练习

练习1：

- `var a = '1', b = 3, c = true;`
- 求下面的值：
 - `a > b` => false 转数字比较
 - `a >= c` => false 转成字符比较
 - `!(b)` => false
 - `!!a` => true
 - `a+b` => '13'
 - `b + c` => 4
 - `b - a` => 3-1 => 2
 - `b && a` => true
 - `!(a || b)` => false

练习2：

- `var a = '1', b = 3, c = true;`
- 求下面的值：
 - `a > b && c < a` => false
 - `+c` => 1
 - `++a+c` => 3
 - `++b + 'ssss'` => '4ssss'
 - `6 / 0` => Infinity
 - `NaN * 0` => NaN
 - `b > c ? ++a: c;` => 3

第四节：类型转换

- 字符串与各种类型转换
- 数字跟各种类型转换
- boolean类型跟各种类型转换

显式的类型转换

- 转换为数值类型：
 - Number(mix)、
 - parseInt(string,radix)
 - parseFloat(string)
- 转换为字符串类型：
 - String(mix)
- 转换为布尔类型：
 - Boolean(mix)

Number()函数转换

- Number(mix)函数，可以将任意类型的参数mix转换

- 如果是布尔值，true和false分别被转换为1和0

- Number(true)// 1

- 如果是数字值，返回本身。

- 如果是null，返回0.

- 如果是undefined，返回NaN。

- 如果是字符串，遵循以下规则：

- 如果字符串中只包含数字，则将其转换为十进制（忽略前导0）

- 如果字符串中包含有效的浮点格式，将其转换为浮点数值（忽略前导0）

- 如果是空字符串，将其转换为0

- 如果字符串中包含非以上格式，则将其转换为NaN

- 单加法操作跟Number函数效果一致：

- var b, a = false;

- b = +a;// b = 0 => b= Number(a);

- b = + "123" // b = 123 ,把字符串转为数字



```
Elements Network Sources
<top frame>
> Number(true)
< 1
> Number(false)
< 0
> Number(4)
< 4
> Number(null)
< 0
> Number(undefined)
< NaN
> Number("")
< 0
> Number("123")
< 123
> Number("s123")
< NaN
> Number("0123")
< 123
> Number("01.23e33")
< 1.23e+33
>
```

parseInt转换数字

- `parseInt(string, radix)`函数，将字符串转换为整数类型的数值。它也有一定的规则：
 - 忽略字符串前面的空格，直至找到第一个非空字符
 - 如果第一个字符不是数字符号或者负号，返回NaN
 - 如果第一个字符是数字，则继续解析直至字符串解析完毕或者遇到一个非数字符号为止
 - 如果上步解析的结果以0开头，则将其当作八进制来解析；如果以0x开头，则将其当作十六进制来解析
- `parseFloat(string)`函数，将字符串转换为浮点数类型的数值。

```
<top frame>
> parseInt(true)
< NaN
> parseInt(4)
< 4
> parseInt(null)
< NaN
> parseInt("123")
< 123
> parseInt("0123")
< 123
> parseInt("s123")
< NaN
> parseInt("12sss3")
< 12
> parseInt("01.23e33")
< 1
```

开头不能是非法数字，空格可以

可以忽略非法数字

String方法转换

- String(mix)函数，将任何类型的值转换为字符串，其规则为：
 - 如果有toString()方法，则调用该方法（不传递radix参数）并返回结果

```
var a = 4; a.toString();//"4"
```
 - 如果是null，返回" null"

```
a = null; a.toString();// "null"
```
 - 如果是undefined，返回" undefined"
- 也可以之间用变量+"的方法转换类型为字符串类型，同String函数。
- 例如：`a + "" ; // =>String(a);`

Boolean(mix)函数转换

- Boolean(mix)函数，将任何类型的值转换为布尔值。

- 以下值会被转换为false：

false、""（空字符串）、0、NaN、null、undefined，其余任何值都会被转换为true。

```
var t = Boolean(123); // t => true
```

```
var m = Boolean(""); // m => false
```

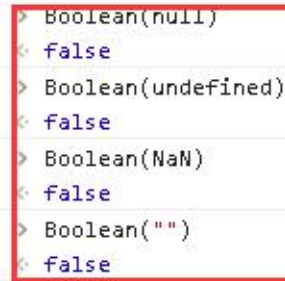
- !!两次取饭跟Boolean方法效果一致

```
var a = 3, b = 0, c = "";
```

```
!!a; // true
```

```
!!b; // false
```

```
!!c; // false
```



```
> Boolean(null)
< false
> Boolean(undefined)
< false
> Boolean(NaN)
< false
> Boolean("")
< false
> Boolean(Infinity)
< true
> Boolean(2)
< true
> Boolean("22")
< true
```

隐式的类型转换

- 隐式类型转换：就是当表达式进行运算时，如果类型不一致，JavaScript引擎会自动根据规则把类型进行转换后再进行运算。
 - `var b = +a;` // 相当于 `b = Number(a);`
 - `var b = !!a;` // `b = Boolean(a);`
 - `var b = a + '';` // `String(a)`
 - `var c = 1 + true;` // `c => 2`
 - `var t = true + '';` // `t => true`

第五节：循环分支语句

- 表达式语句
- 空语句与复合语句
- 声明语句
- 分支语句（条件语句）
- 循环语句
- 跳转语句

JavaScript语句概述

- JavaScript程序由基本的语句组成。默认情况下语句是从第一条往后顺序执行。为了应对复杂需求，出现了分支语句、循环语句、声明语句、跳转语句等。
- 语句是编程的核心，要练的非常熟练！！！！

空语句及复合语句

- 表达式语句

- 123; 'sss' ; true; false; null; undefined;
- 12+true; a+b; Number(true);

- 单行语句

```
var a = 9;    // 单行的赋值语句  
b = a * 9;   // 单行语句
```

- 复合语句（语句块）

复合语句，就是多条语句用花括号括起来，组成的整体就是一个语句块，也称为复合语句，在JavaScript中会把语句块当成一条语句执行。

```
{  
    var a = 2;  
    a++ ;  
    b = !!a;  
}
```

声明语句

- 变量声明语句

- 关键字var
- 语法格式：`var 变量1[=值1][, 变量2[=值2]]....;`
- 例如：

```
var a,      // a => undefined  
    b = 2,  
    c = 3;
```

- 函数声明语句

```
function f() {  
    var a = 123;  
    a *= 3;  
    return a;  
}
```

if语句

- if语句：条件分支语句。

- 语法：

```
if (表达式) 语句；
```

```
if(表达式)  
    语句;
```

```
if(表达式) {  
    语句块;
```

```
}
```

语义：当表达式为真的时候，执行后面的语句。

demo:

```
if( 233 * 2 > 456) {  
    a = 4 ;  
}
```

注意：当if语句的执行语句只有单行的时候，可以省略花括号，但是建议不要省略。

if else语句

- 如果我需要实现：如果小明的年龄小于18岁，那么不给小明啤酒喝，否则给小明啤酒喝？
- else：否则的意思
- if else语句的格式：

```
if(表达式) {  
    语句;  
}else {  
    语句;  
}
```

if else if语句

- 如果有多个条件，分别对应执行多个操作的时候怎么用if和else实现呢？
- 比如：a = 1[2,3--7]; 输出a= 1,那么输出星期一，2=>星期二.....
- 语法格式：

```
if(表达式) {  
    语句;  
}else if(表达式) {  
    语句;  
}else if {  
    语句;  
} else {  
    语句;  
}
```

If语句嵌套

- If语句内部的语句块可以嵌套其他语句及if语句
- 语法格式：

```
if(表达式) {  
    if(表达式2 ) {  
        if(表达式3 ) {  
            console.log(1);  
        }  
    }  
}
```

- 案例：
 - 如果变量a是数字类型，那么判断其是否大于9，如果大于9则输出'大于9'，否则输出:'非法类型'

if语句练习

- 备注：Math.random()方法会随机返回 0-1之间的小数
- 1. 声明两个变量a, b, 随机赋值 0-10之间的数, 然后如果 $a > b$ 那么输出 a, 如果 $a = b$ 输出=号, 如果 $a < b$ 输出 老马
- 2. 给定一个数值变量 (1-7之间) 如果是6、7则输出周末, 否则输出工作日.
- 3. 判断用户名是否为空。
- 4. 判断一个变量是不是字符串, 如果是字符串请将字符串转为数值类型并返回结果。否则, 直接把变量转成 boolean类型输出结果。

while语句

- 如何实现用js计算1到100的和呢？

- $1+2+3+4+...?$

- while就是循环语句的一种。

- 语法：

```
while(表达式) {
```

```
    语句；
```

```
}//语义：当满足条件时，循环执行语句，直到表达式不成立
```

demo：

```
var i = 0,sum = 0;
```

```
while( i <= 50){
```

```
    sum += i;
```

```
    i++;
```

```
}
```

while语句练习

- 案例1：输出10个（0-100之间）随机数
 - Math.random() // => 0-1之间的一个数。
 - 0-100的随机数，那么就需要 *100

- 案例2：实现1到100的阶乘？

```
var i = 1, result = 1;
while( i <= 10 ) {
    result *= i;
    i++;
}
```

do while语句

- do {
 语句 ;
- } while(表达式);
- do while语句就是先执行一个语句块然后在做判断，如果表达式成立继续下一次的语句的执行。
- do while语句较少使用。

for循环语句

- for循环语句是前测试语句，其实本质是while循环语句的变种。

- 语法

```
for(初始化语句;判断语句;循环结束执行语句) {  
    循环体执行的语句;  
}
```

for语句，现在执行初始化语句，只执行一次。

然后根据 判断语句是否为true，如果是true则执行循环体，最后执行 结束执行语句，在进行执行判断语句，如果false，则for循环结束。

for循环语句

- demo: 是1 到100的和

```
var i, result = 0;  
for( i = 1; i <= 100; i++ ) {  
    result += i;  
}
```

本质就是while语句变种。

for循环的变种：

```
for( ; i < 100; ){  
    result += i; i++; //把结束执行语句放到语句块  
    中。  
}
```

综合案例1

- 求100以内，可以对3求余为0的正整数的个数。

综合案例2

- 斐波那契数列，又称黄金分割数列，指的是这样一个数列：0、1、1、2、3、5、8、13、21、34、.....在数学上，斐波纳契数列以如下被以递归的方法定义： $F(0) = 0$ ， $F(1) = 1$ ， $F(n) = F(n-1) + F(n-2)$ ($n \geq 2$ ， $n \in \mathbb{N}^*$) 在现代物理、准晶体结构、化学等领域，斐波纳契数列都有直接的应用。
- 规律：前面两数的和等于后面的值。
- 请用JavaScript实现，求 $F(n)$ 的值， $n \geq 3$;
- 规定： $f(1) = 1$ ， $f(2) = 1$ ， $f(3) = 2 \dots f(n)$
- 分别用for循环和while循环实现

break语句

break语句可以用于精确控制循环语句

当js遇到break后，会立即结束当前的循环语句，并强制立即执行循环后面的语句；

```
for(var i =0; i<10;i++) {  
    if(i %7 == 0) {  
        break; //立即结束循环  
    }  
}
```


continue语句

- continue语句也是为了精确控制循环语句。
- js遇到continue语句后，立即结束当前循环轮次，直接把程序跳转到下一个循环的轮次。

```
for(var i =0; i<10;i++) {  
    if(i %7 == 0) {  
        continue; //立即执行 结束执行语句，并执行下次循环  
    }  
}
```

- 计算1-100的和，去除掉对3取余为0的数字。

switch语句

- switch 语句用于基于不同的条件来执行不同的动作。

语法

```
switch(n)
{
    case 1:
        执行代码块 1
        break;
    case 2:
        执行代码块 2
        break;
    default:
        n 与 case 1 和 case 2 不同时执行的代码
}
```

• de

言。

switch案例

- 把给一个数字，对7取余后，根据数字返回相应的星期，
1：星期一，2：星期二...

综合案例2

- 质数 (prime number) 又称素数，有无限个。一个大于1的自然数，除了1和它本身外，不能被其他自然数整除，换句话说就是该数除了1和它本身以外不再其他的因数；否则称为合数。
- 有一个数：781，判断此数是否是质数？11

第六节：函数

- 函数由数学中定义中引申来。
- 函数就是，一个功能对应的所有语句的集合。
- 函数关键字：function
- 函数不能定义在循环、分支等语句中。
- 语法：

```
function 函数名(参数1, 参数2....) {  
    函数体;  
    // JavaScript语句  
}
```

函数名();//调用函数执行。

// 标识符：变量名、函数名、属性名、类名、参数名。

// 必须以字母、下划线、或者\$开头，后面可以跟着数字、字母、下划线

函数的封装

- 计算1到n的和，可能用到很多次，每写一次都要重复吗？

```
function sumNum(n) {  
    var i = 1, result = 0;  
    if( n && n >= 1){  
        while( i <= n) {  
            result += i;  
        }  
    }  
    return result;//返回结果  
} //能改进一下吗？
```

函数调用：

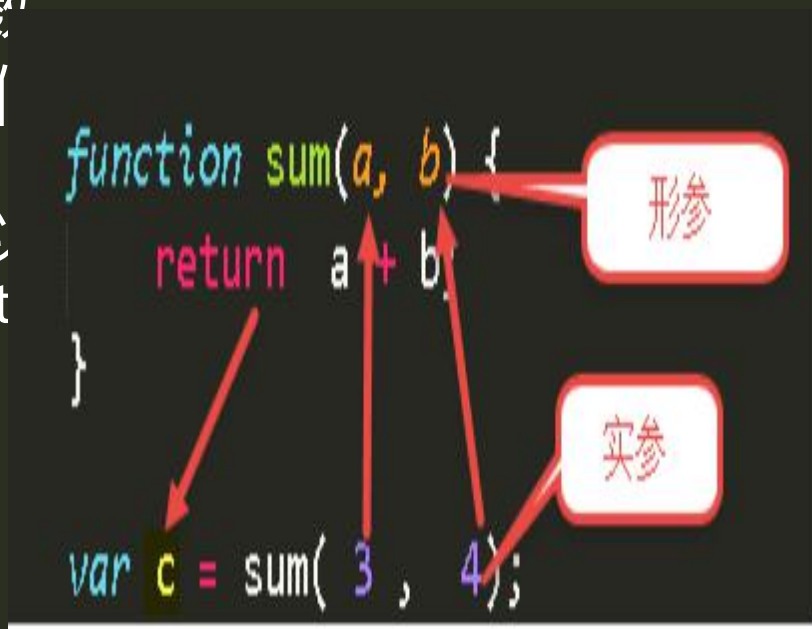
sumNum(20); //自动返回result，以后直接拿来用就可以了。

return子句

- return标识函数的结束。
- 当程序遇到return时，函数立即结束并返回相应的值。否则函数执行到}结束。
- 如果函数没有任何的返回值，JavaScript会默认返回undefined

函数的参数

- 形参和实参匹配传递
- 形参就是：函数定义的参数。
- 实参：就是函数执行传递的参数
- 形参和实参个数不匹配的时候
- 优先从左向右进行匹配
- 实参个数可以少于形参，也可以
- 所有的参数都会放到argument



函数练习

1. 给一个数字，返回相应的星期。

2 : 星期二 'sss' => 不是星期的格式

`getWeekName(3);`

2. 用函数实现给一个整数 n ，返回从1到 n 的和。

3. 给定一个整数 n ($n \geq 3$)，实现求斐波那契数列 $f(n)$ 。

4. 给定一个整数 n ($n \geq 2$)，求 n 的阶乘

匿名函数

- `var a = function (a) {
 return a*a;
}`
- `a();`
- 匿名函数可以作为一个变量来使用。
- 匿名还是可以作为一个参数进行传递。

第七节：对象类型

- Object类型
- 引用类型
- 创建对象的方式
- 对象封装模拟
- 对象的定义及添加属性和方法
- 删除属性
- 属性检测
- 对象的原型
- 对象的方法
- 关于EC5的部分在后续的教程中专门讲授。

对象

- 对象是JavaScript中一组属性和方法的无序集。
- 对象一般是针对一功能载体的描述。比如：狗对象、猫对象。狗对象：拥有常见age、name、color、type属性，还有奔跑run（）等行为。
- JavaScript中的对象是动态的，可以随时添加属性和删除属性。
- 除了字符串、数值类型、布尔类型、null、undefined之外的都是对象类型。
- 对象是引用类型（后面还会讲）

object对象创建方式

- Object类型是我们用的最多的引用类型，我们接触到的大部分都是Object类型的实例。

- 创建Object类型实例的方法：

第一种：new 运算符创建对象

```
var a = new Object();//new 运算符，创建一个Object对象。  
a.age = 18;  
a.name = "itt";
```

第二种：对象字面量

```
var a = {  
    name : "itct",  
    show : "http://www.ss.cn"  
};  
a.age = 19;
```

对象的动态添加属性及方法

- JavaScript的对象非常灵活，可以随时添加新的属性和方法

```
var t = {};  
t.age = 19; // t['age'] = 19; // 可以达到相同效果  
t.show = function() {};
```

- 对象属性的读写

```
t.age = 19; // 等同于 t['age'] = 19;  
console.log(t.age); // t['age']也可以读取属性值
```

- 对象方法的执行调用

```
t.show();
```

- 重复定义的属性和方法会被后面的覆盖***。

引用类型

- 引用类型是JavaScript中复杂类型，是一种数据结构。
- JavaScript中所有的东西都是对象。
- object就是最长用的引用类型
- 引用类型在内存中模型：
 - 栈内存
 - 堆内存
- 简单类型只存储在栈上（boolean、number）

对象的动态删除属性(很少用了了解)

- JavaScript的对象非常灵活，可以随时删除属性
- 删除属性的运算符是delete关键字
- 语法结构：
 - `delete object.property`
 - `delete object['property']`
- delete 操作符会从某个对象上移除指定属性。成功删除的时候回返回 true，否则返回 false。
 - 如果你删除的属性在对象上不存在，那么delete将不会起作用，但仍会返回true
 - 如果 delete 操作符删除成功，则被删除的属性将从所属的对象上彻底消失。
 - 任何使用 var 声明的属性不能从全局作用域或函数的作用域中删除。
 - 除了在全局作用域中的函数不能被删除，在对象(object)中的函数是能够用delete操作删除的。

检测属性

- JavaScript的对象检测属性用 in 运算符

```
var t = {};  
t.age = 19;  
  'age' in t; // => true  
delete t.age;  
  'age' in t; // => false
```

会把继承原型的属性也会遍历到。

枚举对象的属性

- JavaScript的对象可以使用for in 循环遍历对象中的所有属性。
- `var t = {};` // 定义一个对象
- `t.age = 20;` // 设置一个自定义属性为age = 20 ;
- `for(var i in t) { console.log(i);}`

对象的原型与构造函数

- 每个对象都有自己的私有原型对象（除了
- 每个对象都会从自己的私有原型对象上继承属性，可以直接对象可以直接使用。
- 对象调用一个属性或者方法的时候先搜索方法，如果没有那么就去搜索原型上的属性，如果有就直接使用，如果没有就直接抛出错误。
- 对象的私有原型都是通过 `__proto__` 联系在一起的，不是一个标准的属性，但是所有浏览器都支持。
- 构造函数就是构造对象的时候执行的函数。比如：`Object`
- 对象的原型可以通过它的构造函数的`prototype`获得的。
- 后面讲原型和继承的时候还会详细讲

```
> var m = new Object();  
< undefined  
  
> typeof Object  
< "function"  
  
> Object  
< f Object() { [native code] }  
  
> m.__proto__ === Object.prototype  
< true  
  
> m.__proto__.constructor === Object  
< true  
  
> m.age = 19;  
< 19  
  
> m.hasOwnProperty('age')  
< true  
  
> m.hasOwnProperty('age222')  
< false  
>
```

Obeject的原型属性和方法

- 原型属性
- toString() 转换成字符串
- toLocalString() 转换成本地化对应的字符串
- valueOf() 获取它的值。
- hasOwnProperty() 判断属性是否是自己创建添加的。

对象练习

- 封装一个猫对象。猫拥有：颜色、重量、名字、ID等属性，还拥有跑、跳、叫等行为。并调用猫猫的跑方法。
- 输出猫猫的所有的自定义属性的名称
- 封装一个数学对象，对象中包括： π 属性，获取1-n的和的方法，获取n的阶乘的方法。

第八节：Array类型

- 数组对象的作用是：使用单独的变量名来存储一系列的值。
- 数组就是一堆数据的分组或者集合。
- 数组的对象是Array，也是非常常用的引用类型
- 例如：[1, 3, 'sss', 333]
- Object , Array
- EC5的新方法在EC5&6的专题中再专门讲

创建数组

- 使用new操作符创建数组对象

- `var arr = new Array(); // => []`
- `arr[0] = 19; // 数组的索引从0开始 [19]`
- `arr[1] = "www.hamkd.com"; // [19, 'www...'] length=2`
- `arr[2] = "www.aicoder.com";`

- 数组中可以存储任何数据类型的数据（其他语言中数组的数据类型一旦确定只能存储特定类型的数据）

- 通过[]和索引来访问和设置数组的内容

- 数组的索引是从0开始！！！！

- JavaScript中的数字的容量可以动态改变

- 数组的容量最大为：4294967295个 ($2^{32}-1$)

- 求幂运算符：****** （补充） $2^{**}32$ $2^{**}11$

构造函数创建数组

- 数组的构造函数可以传递数组的容量的参数
 - `var arr = new Array();`// 创建一个空数组
 - `var arr = new Array(3);`// 定义数组的容量为3个
 - `var arr = new Array("a","b","c");`
 - `// 定义一个数字, 有a, b, c 三个字符串`
- 数组的长度可以通过`length`数组来获取。
- `length`是可读和可以进行设置, 可以对数组进行截断操作。不要这么用!!!!

数组字面量创建数组

- `var arr = [];`//创建一个空数组
- `var arr = [1,2,3];`//创建三个数字的数字。
- `var arr=[1,,2];`//中间的省略的是undefined
- `var arr = [1, " dd" ,true, [1,3], { age: 19}, 33];`
- 数组的元素类型可以是任意类型。

遍历数组

- length属性
 - length属性,如果是连续数组,那么length就是数组元素的个数
 - 如果是稀疏数组,那么length不能代表元素的个数
- 遍历数组
 - for循环方式遍历数组
 - for in循环的方式
 - 数组的索引是从0开始!!!不是从1开始
- 注意:从原型上继承来的属性也会被for in遍历,所以如果需要去掉这部分属性可以通过hasOwnProperty()进行过滤。
- 案例:一个数组,合法值为1,2或3,其余为不合法,统计合法及不合法的个数。
- `var t = [1, 4, 9, 'sss', 3, '2', 2, 3, 2, 1];`

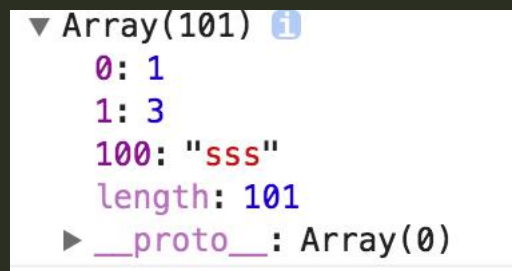
```
> Object.prototype.temp = '222';
< "222"
> var t = new Object();
< undefined
> t
< {}
> t.age = 20;
< 20
> for(var k in t) {
  if(t.hasOwnProperty(k)) {
    console.log(k);
  }
}
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
```

稀疏数组

- 数组中的数据索引不一定是连续的。非连续的数组就是稀疏数组。

- 比如：

- `var t = [1, 3];`
- `t[100] = true;`
- `// => t.length = 101;`
- `// => t = [1, 3, undefined*98, true]`
- `// 实际上有数据的只有 1, 3, true 中间有98个undefined`

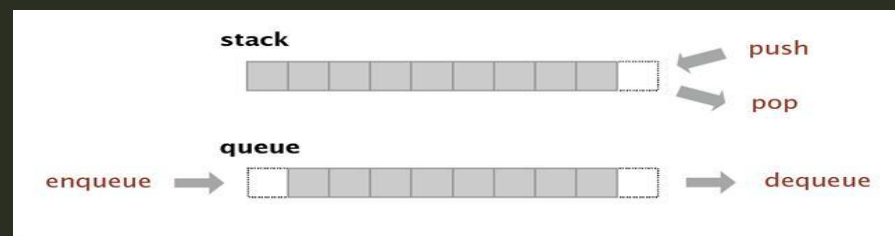


- 可以使用for in循环越过undefined的数据。
- 也就是数组的元素个数不一定跟length相等

- 稀疏数组尽量少用！！！！！或者就是不用！！！！！

数组方法-栈队列和排序

- 常用方法：
- `pop()`; //从数组尾部弹出一个元素并返回弹出的元素
 - `length`就会减1。 尾部：是指索引最大的地方。
- `push()`; //从数组的尾部压入一个元素，并返回数组的新长度
- // 栈方法：可以对比弹夹
- `shift()`; //从数组的头部弹出一个元素，并返回此元素
- `unshift()`; //从数组的头部压入一个元素，并返回`length`
- `sort()`; //转成字符串排序
- `reverse()`; //对原数组进行逆序



函数作为参数进行传递

- 数组的sort方法只能按照字符串类型的比较方法进行排序。如果是其他类型数据的排序会转成字符串然后再进行比较。
- 某些数值类型进行排序就不按照值类型大小排序。
 - 比如：[10, 5, 2, 13, 11, 20] => [10, 11, 13, 2, 20, 5]
- 函数本身也是一个对象，也是可以作为参数传递给其他函数进行使用。【函数式编程】
- 数组的sort方法可以接收一个类型为函数的参数，此函数接收两个参数，要求返回值为：负数，0，正数。如果返回负数代表第一个参数小于第二个参数，0为相等，正数则大于

```
[1, 3, 20, 11, 9].sort(function (a, b) {  
    if (a > b) {  
        return 1;  
    } else if (a == b) {  
        return 0;  
    } else {  
        return -1;  
    }  
});
```

数组的方法-连接方法

- `concat()` ;

- 连接原数组的元素和传递的参数形成一个新数组并返回，**不影响原来的数组。**
- 如果传入的参数是数组，会把数组中的元素跟原数组的元素进行合并成一新数组。
- `[1, 2, 3].concat(9, 1, 4); // => [1, 2, 3, 9, 1, 4]`
- `[1, 2, 3].concat([9, true]); // => [1, 2, 3, 9, true]`
- `[1, 2, 3].concat([9, true, ['22', 4, 9], 33])`
 - `//=> [1, 2, 3, 9, true, Array(3), 33]`

- `join()`;

- 可以把数组的元素（项）连接成字符串，接收一个参数为连接符号，默认是逗号，返回的结果为字符串。
- `[1, 2, 3].join(); //=> 1,2,3`
- `[4, true, 3].join("-"); // => 4-true-3`

数组的方法-获取切片方法

- slice() ; //复制数组的一部分
 - 截取数组的一个片段或者子数组
 - 接收1个到2个参数。参数：截取数组起始索引和结束索引
 - 如果只指定一个参数代表：从索引位置到数组结尾。
 - 参数如果是负数代表从数组末尾计算索引位置。
 - 此方法只能从数组前面往后面截取，如果第二个参数在第一个参数的前面则返回空数组[];
 - 数组只能往后截取，不能向前截取，如果向前截取返回[]
 - 此方法对原数组没有影响。
 - `m = [1,2,3,4,5];`
 - `m.slice(2); //=> [3, 4, 5]`
 - `m.slice(-3); //=> [3, 4, 5]`
 - `m.slice(3, 4); //=> [4]`
 - `m.slice(-3, -1); //=> [3, 4]`

数组的方法-切片方法

- splice () ;

- 在原数组上进行插入或者删除数组元素，会影响原来数组。
- 返回的结果是删除的元素组成的数组。
- 参数：可以接受1个参数，2个参数或者2个以上的参数。
 - 第一个参数是删除数据的索引位置(start)
 - 第二个参数是要删除数组元素的个数(end)
 - 第三个参数开始是要插入到原数组中的元素，插入的位置从第一个参数所在的索引开始。

- 删除数据

- `[1, 2, 3, 4, 5].splice(2);` // => `[3, 4, 5]` 原数组：`[1,2]`
- `[1, 2, 3, 4, 5].splice(3,2);` // => `[4, 5]` 原数组：`[1,2,3]`
- `[1, 2, 3, 4, 5].splice(-2);` // => `[4, 5]` 原数组：`[1,2,3]`
- `a = [1,2,3,4,5];`
- `a.splice(3,2,33,'222',[99,98]);` // => `[4,5]`
- `//a=>[1, 2, 3, 33, "222", Array(2)]`
- 插入数据：`a=[1,2,3]; a.splice(1,0,5,6);` // => `[]` `a=[1,5,6,23]`
- 替换数据：`a=[1,2,3];a.splice(1,1,4);` // => `[2]` `a=[1,4,3]`

数组继承的方法

- toString()方法
 - 它会将数组中的每个元素转成字符串并用逗号连接起来。
 - 类似join()方法的默认情况。
 - `[1, 3, 5].toString();//=> "1,3,5"`
- toLocalString()方法
- valueOf()方法

数组案例

数组数据：[90, 8, 34, 2, 39, 87, 22, 10]

- 1.将数组内容进行反序
- 2.求一个数据数组中的最小值及它的索引
- 3.求一个数组中的数据的平均值与和。
- 4.数组的数据进行排序
- 5.给定一个数组，请去掉数组中的重复数据。
- 6.冒泡排序算法

第九节：基本包装类型

- Boolean类型
- Number类型
- String类型
- 字符处理
- 字符串类型String
- 字符串的常用方法

基本包装类型概述

- 先看一个现象：

- `var t = 9.187; // 编辑，设置`
- `t.toString(); // => "9.187"`
- 为什么一个数值类型有`toString`方法？

- 针对布尔类型、数值类型、字符串类型JavaScript都提供了对应的包装类型。当三种类型的变量在读取操作的时候，JavaScript执行引擎会自动创建一个临时包装对象，帮助它可以访问包装对象的方法，使用完毕后立即销毁包装对象。

- 例如：

- `var t = 19; // 创建数值类型的变量。`
- `t.age = 19; // 创建临时对象，但是临时对象用完即毁`
- `console.log(t.age); // undefined`
- `// 临时创建了包装对象，并销毁了`

Boolean包装类型

- Boolean类型是引用类型
- 创建方法：
 - `var t = new Boolean(false);` // t是引用类型
 - `!!t => true` 非空对象转成boolean类型是true
 - `typeof t` // => object
 - `t instanceof Boolean` // => true
 - `var m = true;`
 - `m.toString();` // => true ,临时创建包装类型
 - `typeof m` // => boolean
- 创建的true值的Boolean类型在转成boolean的简单类型的时候却成了false，所以会引起很大误解。所以一般不会这样使用！！！！

Number包装类型（上）

- Number类型是引用类型
- 创建方法：
 - `var t = new Number(0);` // t是引用类型
 - `typeof t` // => object
 - `t instanceof Number` // => true
 - `var m = 9;`
 - `m.toString()` // "9"
- Number的方法
 - `toString()` // 重写了object的toString方法，可以接受进制的参数。
 - `var t = 2;`
 - `t.toString(2)` // => "10" 输出二进制的数值2
 - `t.toString()` // => "2" 默认是10进制
 - `t.toString(8)` // => "2" 输出8进制的数值2

Number包装类型（下）

- Number的方法
 - toString(); // 只是将数值转成字符串
 - `var t = 2;`
 - `t.toString(); // => "2"`
 - valueOf(); //返回自身
 - toFixed(); //接受一个参数，限定小数的位数。
 - // toFixed()有bug，所以参考：<http://qkxue.net/info/109055/toFixed-bug>
 - IE8及之前的版本对：[-0.94到-0.5] [0.5-0.94]返回0，不是1
 - `a = 1.23; a.toFixed(1); // => " 1.2"`
 - `a = 2.3; a.toFixed(3); // => "2.300"` 会自动补充0
 - toExponential(3);
 - 按指数的形式输出字符串
 - 传入的参数是保留小数的位数
 - `a = 3; a.toExponential(3); // "3.000*e+0"`
 - toPrecision();
 - 接受一个参数。参数是输出数据的总位数。
 - 它会自动根据情况调用toExponential或者toFixed方法
 - `a = 96;`
 - `a.toPrecision(3); // => 96.0`
 - `a.toPrecision(1); // => "1e+2"`

字符串类型String

- 字符串包装类型String
- `var a = new String("123");`
- `length`属性返回字符串字符的个数。
- 字符串是不可变的。对字符串的所有操作都会返回一个新字符串，原字符串不变。
 - ie6-7 : `'abc' + 'cdf'`;
- `charAt(index)`方法:获取某个索引位置的字符,把字符串当字符数组
- `charCodeAt(index)`方法 : 获取索引位置的字符 的编码
- `stringValue[index]` ; 类似数组的使用 , 获取索引位置的字符 , IE8 以上才支持。
- `concat()`方法 : 连接字符串,可以传入多个参数。
 - `"22".concat('2' ,333, 9); //=> "2223339 "`
- `slice()` 类似数组的用法。字符串切片。
 - 一个参数 , 从索引位置到最后。负数从结尾结算。
`'12345'.slice(2);=>" 345"`
 - 两个参数 , 返回从第一个参数索引位置到最后一个参数索引位置之前的一个字符
 - `'12345'.slice(2,4);=>" 34"`

字符串类型String

- `replace()`;
 - 可以传入两个参数。第一个是要替换的原字符串，第二个是替换的新字符串
 - 对原字符串没有影响，返回新字符串。
 - `"12345".replace('4', 'ss');// => "123ss5"`
 - 后面会介绍正则表达式
- `substring()`
 - 截取字符串，对原字符串没有影响
 - 可以接受两个参数或者一个参数。第一个参数：截取索引位置，第二个是结束的位置。都是正数的情况跟slice保持一致。负数有点不同。如果出现负数，substring都将它们看作0处理。
- `substr()`
 - 截取字符串，跟substring区别就是第二个参数是截取字符串的长度
 - 如果第二个参数是负数直接看做0处理
 - 第一个参数是负数从末位计算。【IE8有bug】不要用
- `trim()`
 - 方法会从一个字符串的两端删除空白字符
 - 并不影响原字符串本身，它返回的是一个新的字符串
 - `var orig = 'foo'; console.log(orig.trim()); // 'foo'`
 - IE9以上才支持

```
"123456".substring(3)
"456"
"123456".substring(3, -2)
"123"
"123456".substring(3, 2)
"3"
"123456".substring(3, 5)
"45"
"123456".substr(3, 2)
"45"
"123456".substr(3, -9)
""
"123456".substr(-3, 2)
"45"
```

字符串类型String

- `toLowerCase()` 会将调用该方法的字符串值转为小写形式，并返回。对原字符串没有影响。
 - `"ALPHABET".toLowerCase() ;//=> "alphabet"`
- `toUpperCase()` 将调用该方法的字符串值转换为大写形式，并返回。
 - `console.log("alphabet".toUpperCase()); // "ALPHABET"`
- `toLocaleLowerCase()`和`toLocaleUpperCase()`与上面类似，只有比如土耳其语言有点区别，大部分都没有区别。
- `split()` 方法使用指定的分隔符字符串将一个String对象分割成字符串数组，以将字符串分隔为子字符串，以确定每个拆分的位置。
 - 语法：`str.split([separator[, limit]])`
 - `separator`：指定表示每个拆分应发生的点的字符串
 - `limit`：一个整数，限定返回的分割片段数量；
 - 找到分隔符后，将其从字符串中删除，并将子字符串的数组返回。
 - `'abcdef'.split('c') ;// => ["ab", "def"]`
 - 如果没有找到或者省略了分隔符，则该数组包含一个由整个字符串组成的元素。
 - `'abcdef'.split('k'); // => ["abcdef"]`
 - 如果分隔符为空字符串，则将str转换为字符数组。
 - `'abcdef'.split(''); // => ["a", "b", "c", "d", "e", "f"]`
 - 如果分隔符出现在字符串的开始或结尾，或两者都分开，分别以空字符串开头，结尾或两者开始和结束。因此，如果字符串仅由一个分隔符实例组成，则该数组由两个空字符串组成
 - `'abcdef'.split('a'); //=> ["", "bcdef"]`
 - `'abcdef'.split('f'); //=> ["abcde", ""]`
 - `'abcdef'.split('abcdef');//=>["", ""]`

字符串的indexOf方法

- indexOf() 方法返回调用字符串对象中第一次出现的指定值的索引，开始在 fromIndex 进行搜索。如果未找到该值，则返回 -1
- 语法格式：`str.indexOf(searchValue[, fromIndex])`
- 参数：
 - searchValue 一个字符串表示被查找的值。
 - fromIndex 可选表示调用该方法的字符串中开始查找的位置。可以是任意整数。默认值为 0。如果 fromIndex < 0 则查找整个字符串（如同传进了 0）。如果 fromIndex >= str.length，则该方法返回 -1，除非被查找的字符串是一个空字符串，此时返回 str.length。
- 返回值：
 - 指定值的第一次出现的索引；如果没有找到 -1。
- 案例

```
"Blue Whale".indexOf("Blue"); // returns 0
"Blue Whale".indexOf("Blute"); // returns -1
"Blue Whale".indexOf("Whale", 0); // returns 5
"Blue Whale".indexOf("Whale", 5); // returns 5
"Blue Whale".indexOf("", 9); // returns 9
"Blue Whale".indexOf("", 10); // returns 10
"Blue Whale".indexOf("", 11); // returns 10
```
- lastIndexOf() 跟 indexOf 方法类似。要查询的是最后一个字符。

字符串练习

- 截取字符串abcdefg的efg
- 请编写代码实现字符串逆序。至少2种方法。
- 判断一个字符串中出现次数最多的字符，统计这个次数
- 问题：
 - 输入两个字符串，从第一个字符串中删除第二个字符串中的所有字符。不可以使用replace
 - <!--例如：输入 “They are students” 和 “aeiou” -->
 - <!--则删除之后的第一个字符串变成 “Thy r stdnts” -->

第九节：单体对象与日期对象

- 日期对象
- Math对象
- 全局对象

日期类型

- Date是JavaScript中处理日期的对象。它的值是从1970.1.1午夜零时起的毫秒数。
- 世界协调时间：UTC
- 创建日期对象可以通过构造函数：
 - 语法
 - `new Date();` // 当前时间
 - `new Date(value);`
 - `new Date(dateString);`
 - `new Date(year, month[, day[, hour[, minutes[, seconds[, milliseconds]]]]];`
 - 案例：
 - `var today = new Date();`
 - `var today = new Date(1453094034000);`
 - `var birthday = new Date("December 17, 1995");`
 - `var birthday = new Date("1995-12-17T03:24:00");`
 - `var birthday = new Date(1995,11,17,3,24,0);`
 - **0-11数字表示1-12月: `var a= new Date(2006,5,6)` 结果是2006-6-6**
0-6表示星期

日期的方法

- `getDate()`
 - 根据本地时间返回指定日期对象的月份中的第几天（1-31）。
- `getDay()`
 - 根据本地时间返回指定日期对象的星期中的第几天（0-6）。
- `getFullYear()`
 - 根据本地时间返回指定日期对象的年份（四位数年份时返回四位数字）。
- `getHours()`
 - 根据本地时间返回指定日期对象的小时（0-23）。
- `getMilliseconds()`
 - 根据本地时间返回指定日期对象的毫秒（0-999）。
- `getMinutes()`
 - 根据本地时间返回指定日期对象的分钟（0-59）。
- `getMonth()`
 - 根据本地时间返回指定日期对象的月份（0-11）。
- `getSeconds()`
 - 根据本地时间返回指定日期对象的秒数（0-59）。
- `getTime()`
 - 返回从1970-1-1 00:00:00 UTC（协调世界时）到该日期经过的毫秒数，对于1970-1-1 00:00:00 UTC之前的时间返回负值
- **所有的get换成set就是设置方法**

日期的方法

- `toDateString()`
 - 以人类易读 (human-readable) 的形式返回该日期对象日期部分的字符串。
- `toISOString()`
 - 把一个日期转换为符合 ISO 8601 扩展格式的字符串。
- `toGMTString()`
 - 返回一个基于 GMT (UT) 时区的字符串来表示该日期。请使用 `toUTCString()` 方法代替。
- `toLocaleDateString()`
 - 返回一个表示该日期对象日期部分的字符串，该字符串格式与系统设置的地区关联 (locality sensitive) 。
- `toLocaleString()`
 - 返回一个表示该日期对象的字符串，该字符串与系统设置的地区关联 (locality sensitive) 。覆盖了 `Object.prototype.toLocaleString()` 方法。
- `toLocaleTimeString()`
 - 返回一个表示该日期对象时间部分的字符串，该字符串格式与系统设置的地区关联 (locality sensitive) 。
- `toString()`
 - 返回一个表示该日期对象的字符串。覆盖了 `Object.prototype.toString()` 方法。
- `getTimeString()`
 - 以人类易读格式返回日期对象时间部分的字符串。
- `toUTCString()`
 - 把一个日期对象转换为一个以UTC时区计时的字符串。
- `valueOf()`
 - 获取原始的时间毫秒数

日期对象的格式化

- 给一个日期对象输出：xx年xx月xx日

Math对象

- JavaScript提供了Math内置对象方便我们进行数学运算
- 它具有数学常数和函数的属性和方法。
- 常用属性：
 - `Math.PI`; //圆周率=> 3.141592653589793
 - `Math.E` //自然对数底，数学中e的值2.718281828459045
 - `Math.LN10`; //10的自然对数，约等于2.303.
 - `Math.LN2`; // 2的自然对数，约等于0.693
 - `Math.LOG2E`; //以2为底e的对数，约等于 1.443.
 - `Math.LOG10E`; //以10为底e的对数，约等于 0.434.
 - `Math.SQRT1_2` ; 1/2的平方根, 约等于 0.707.
 - `Math.SQRT2` ; 2的平方根,约等于 1.414.
- 常用方法：
 - `max()`与`min()`。求一组数的最大值和最小值。
 - `Math.min(10,9,8,22)`; //=> 8
 - `Math.max(10,9,22)`; //22

Math对象的常用方法

- 舍入方法

- `Math.ceil();` // 向上舍入 `Math.ceil(2.34);`//3
- `Math.floor();` // 向下舍入 `Math.floor(2.3);`//2
- `Math.round();` //四舍五入 `Math.round(2.7);`//3

- `Math.random();` ; 获取随机数 (0-1)

- `Math.abs(num);`//求num绝对值

- `Math.exp(num);`//求e的num次幂

- `Math.log(num);`//求num的自然数对数

- `Math.pow(num,power);`//求num的power次方

- `Math.sqrt(num);`//求num的平方根

- 三角函数`sin()`, `cos()`, `tan()`,`asin()`, `acos()`, `atan()`参数是弧度 (0-2 π)

全局对象

- 全局对象Global，是JavaScript的兜底对象，所有没有对象的方法和属性都会归到全局对象上。
- 在浏览器端：window对象实现了Global对象，所以在浏览器环境中全局对象就是window对象。但是：在其他环境中不是这样的，比如Nodejs环境。
- 常用方法：
 - isNaN
 - parseInt
 - parseFloat
 - encodeURI: 对uri进行编码，uri中特殊符号保留。
 - encodeURIComponent：直接进行编码
 - decodeURI decodeURIComponent
 - eval
 - 函数会将传入的字符串当做JavaScript代码进行执行。

JavaScript高级

- 函数高级
- 作用域深入
- 原型链
- 闭包
- 垃圾回收机制
- 异常处理
- 正则表达式

JavaScript面向对象

- 面向对象的概述
- JavaScript继承
- JavaScript模块化
- 高级课程请大家关注地址：
- 百度传课：
- <https://chuanke.baidu.com/s5508922.html>
- QQ：515154084
- 邮箱：malun666@126.com
- 微博：<http://weibo.com/flydragon2010>



Thanks

@马伦-flydragon