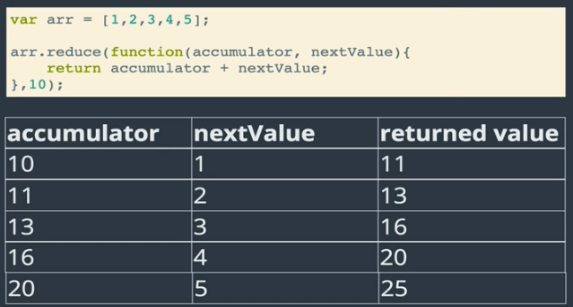
基本使用

* 5种原始类型
  + Number
  + String
    - 长度属性length
    - 单元素访问[x]
  + Boolean
  + null
    - 表示空值，但与undefined不同
  + undefined
    - 变量生成后的默认值
* 定义变量
  + var x = “value”;
    - 变量在Js内可以重复定义不会提示错误
* 内建函数
  + alert 弹出文本窗口提示用户
  + prompt弹出文本输入框
  + console.log产生日志至console setInterval(function(),time) 定时调用function()
  + console.dir显示该标记的所有属性、事件、样式等
* 数组的内建函数
  + 见MDN <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array>
  + shift() 所有对象向前移动，删除最后位
  + unshift(“xxx”) 将xxx插入数组第一位，其他向后移动一位
  + indexOf(“xxx”)在数组中的位置，如有两个一样的值，值提供先出现的，找不到为-1
  + Slice(1,3) 将数组进行切分（不对原有数组产生变化）并获得子数组，如1,3意为从序号1开始到序号3（不包含序号3）
  + forEach(function(element,i,arr){}); 遍历函数,最多可以支持3个参数，分别为当前元素，当前索引，当前数组
* 特殊操作
  + JS不是强类型语言，所以在定义变量的时候不用指定变量类型，但是这样在对比时就会产生障碍，所以JS提供了两种特殊的比较符号
    - === 用于比对值和类型是否都相等
    - !==值或者类型不同
  + 数组的定义方法
    - var friendsList = [“element1”,”element2”,”...”] ;
    - 与其他语言不同的是，JS内并没有强制定义数组的长度，成员可以是任意类型的不同数据，所以可以任意的在数组后添加数据，如果出现跨越中间的序号，则会自动将中间的内容定义为undefined
      * friendsList [length+x]
    - 二维数组的使用为
      * friendsList [x][y]
  + 当一个变量值定义为null是，如果判断是否和undefined相等，从值的角度上(==)是相等的，从类型的角度上(===)是不等的
  + 1为true，其他数字（其他正负数）均为false
  + NaN符号不能作为逻辑对比对象
  + 特殊false值
    - 空字符串为false
    - null为false
    - undefined为false
    - NaN为false
  + typeof 操作符可以确认变量的类型可和由字符串组成的变量类型名称对比
  + 强制转换
    - 转换数字型 Number(var)
    - 其他类型用法同数字型转换，类型名称注意大小写
  + 函数定义方法
    - function doSomething()
    - {
      * Xxx.xx.xxx;  
        }
  + 匿名函数的定义方法
    - var xx = function(yy)
    - {
      * return zz;
    - };
  + 确认函数参数的长度
    - arguments.length
  + 函数内的同名零时变量不会修改同名全局变量的值
  + 访问对象的方法(两种方法等效)
    - 方法1：Obj[“attribute”]=value;
    - 方法2：Obj.attribute = value;
  + 申明对象的方法(两种方法等效)
    - 方法1：var person = {name:value,....};
    - 方法2：
    - var person = new Object();
    - person.name = value;
  + in操作符
    - 用于确认某个对象中是佛存在某个属性，如 prop in object
* 其他
  + Chrome浏览器与IE浏览器不同，直到alert，prompt，confirm结束之前，它不会出现任何HTML界面，但本身不推荐在页面加载前就使用alert，prompt，confirm等功能函数
    - 可以使用window.setTImeout(function(){},time)绕开这个问题
    - 更优雅的方法是使用jQuery $(document).ready功能
  + JavaScript+HTML DOM+CSS特效网站https://patatap.com

数组的高级用法

* arr.forEach(function(item,index,arr){})
  + 遍历
* arr.map(function(item,index,arr){ return item/\*处理后的\*/;})
  + 遍历并处理每个数据
  + 返回新的数组
* arr.filter(function(item,index,arr){ return Boolean;})
  + 筛选item并将符合条件的插入新的数组，返回true表示需要此item，返回false表示移除此item
  + 返回新数组
* arr.some(function(item,index,arr){return Boolean;})
  + 如数组中的某个item满足条件则返回true
  + 不返回新数组，仅返回true or false
* arr.every(function(item,index,arr){return Boolean;})
  + 如数组中的某个item不满足条件则返回false
  + 不返回新数组，仅返回true or false
* arr.reduce(function(accumulator,next,index,arr){ return xxx; },firstNum)
  + 可以实现累加/乘等运算多种功能（见下图累加）
  + 返回累加/乘等运算后的值

对象封装的方法（实现成员变量私有化）

* 建立构建器

function myObj(){

var privateVariable = data; //内部成员变量

return {

setValue(value):function(value) { …… }, //对象方法

getValue():function() { …… }, //对象方法

}

}

* this 关键字的注意事项
  + 不要使用作为属性直接返回this，这会返回上一级对象
* 显示绑定
  + call
    - 取代函数中的this,使用参数的对象作为上下文中的this

如下，myObj取代了info，成为了printData函数中的thisvar myObj = {

data:123,

info:{

printData : function (param) {

console.log(param);

}

}};

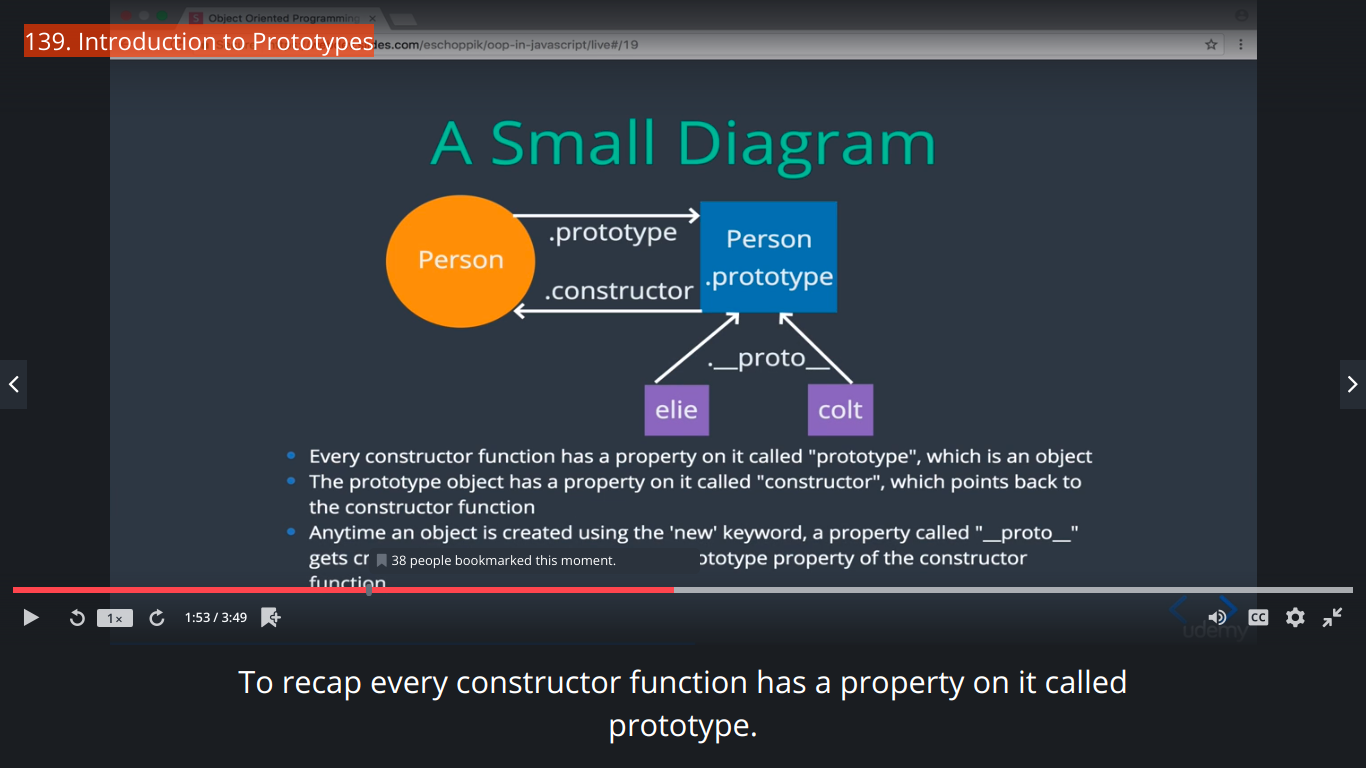
myObj.info.printData.call(myObj/\*取代this的对象\*/, param/\*函数的参数\*/);

* + apply
    - 和call基本功能一致，只是apply的参数是由数组组成的，对于参数较长的函数，apply有优势
  + bind
    - 功能和call一样，不同的是call是立即执行的，而bind则是会返回新的函数用做callback使用。
* new
  + 使用new关键字后，会自动将对象中的this变为对象自身。
* prototype
  + Prototype,\_\_proto\_\_,constructor之间的关系

实力化对象的\_\_proto\_\_和对象构建器的prototype相同

对象的prototype的构建器与构建器相同

具体关系见图



* + Prototype chain
    - 在对象的prototype中添加/更改属性后，实力化的对象会立即应用更改
    - 如:  
      function Person(){this.name = “Kevin”;}  
        
      var me = new Person();
    - Person.prototype.xx = “123”; //为Person原型添加xx属性
    - me.xx; //me也有相同的xx属性，且值继承自Person
  + 通过原型继承属性
    - function TypeA(name) {this.name = name;}
    - TypeA.prototype.ShowSelf = function() {console.log(this.name);}

function TypeB(name) { return TypeA.apply(this,arguments);}

TypeB.prototype = Object.create(TypeA.prototype)

//此时TypeB的实例继承了TypeA的ShowSelf方法

//修改TypeB的属性不会影响TypeA，但是修改TypeA仍会影响TypeB，因为继承的关系

//错误的使用方法：

// 1.TypeB.prototype = TypeA.prototype

// 修改TypeB的属性会更改TypeA的属性