# Javascript学习笔记（二）

## 1、自定义构造函数。

必须强制使用new的模式来生成构造函数。

如：function Student(name,age){   
this.name=name;   
this.age=age;   
}

Var \_stu = new Student(“liu”,25);

Var \_stu2 = Student(“lisi”,26); //这样会丢掉原型链。

自调用构造函数。 //在构造函数内部new一个自己的实例，可以不用再在外部new出来，并使得原型（prototype）属性可在实例对象中可用。

function Student(name,age){

if( !(this instanceof Student) ){

return new Student(name, age);

}  
this.name=name;   
this.age=age;   
}

## 2、函数的arguments属性。

## 3、回调函数和其作用域。特别注意的是this的引用。

## 4、Javascript中的调用函数的call和apply的异同。

个人理解call和apply的作用就是**切换函数的对象上下文**。

Call就是直接从第二个参数开始接受多个参数。

Apply就只有两个参数，第二个参数是一个数组，存放多个参数。

## 5、javascript中的全局函数和局部函数。

function testPartFun(){

var self = this;

console.log("这是一个局部函数。testPartFun");

}

var testFun = function (){

var myself = this;

var obj = {

name:"liuyaoao"

};

console.log("这是一个局部函数。testFun");

}

testVar = function (){

var self2 = this;

console.log("这是一个全局函数。testVar");

}

## 6、json的有关问题。

最后去[json官网](http://json.org/json-zh.html)才发现规定了json里面的字符串必须要用双引号。

## 7、点击按钮排序的逻辑。

///通过某一列的属性值来排序。目前固定按排序值来排序。

//实现了点一次是正序，再点一次反序。

orderByPower: function ($context){

if(!page.data.pageDataArr){ return;}

var isAscend = ($context.attr("ascendOrder")==="1")?1:0;

$context.attr("ascendOrder",1-isAscend);

page.data.pageDataArr.sort(function (item1,item2){

return isAscend ? item1.power<item2.power : item1.power>item2.power;

});

page.data.dataTable.setData(page.data.pageDataArr || []);

},

// var $form = page.data.$newDialog.find(".form");

// var params = $form.formData();

// page.formatContentData(params, function (){

// $('#contentType\_3').find("[value]").val("");

// alert("成功添加了一个跳转链接！");

// });

## 8、返回 function 函数的防反跳版本: getDebounceFun;

将延迟函数的执行(真正的执行)在函数最后一次调用时刻的 **wait** 毫秒之后. 对于必须在一些输入（多是一些用户操作）停止到达*之后*执行的行为有帮助。

传参 **immediate** 为 true， **getDebounceFun**会在 **wait** 时间间隔的开始调用这个函数 。（注：并且在 waite 的时间之内，不会再次调用。）在类似不小心点了提交按钮两下而提交了两次的情况下很有用。

getNowTime :**function**(){  
 **return** Date.now || **new** Date().getTime();  
},  
getDebounceFun : **function**(func, wait, immediate) {  
 **var** timeout, args, context, timestamp, result;  
 **var** later = **function**() {  
 **var** last = page.getNowTime() - timestamp;  
  
 **if** (last < wait && last >= 0) {  
 timeout = setTimeout(later, wait - last);  
 } **else** {  
 timeout = **null**;  
 **if** (!immediate) {  
 result = func.apply(context, args);  
 **if** (!timeout) context = args = **null**;  
 }  
 }  
 };  
 **return function**() {  
 context = **this**;  
 args = arguments;  
 timestamp = page.getNowTime()();  
 **var** callNow = immediate && !timeout;  
 **if** (!timeout) timeout = setTimeout(later, wait);  
 **if** (callNow) {  
 result = func.apply(context, args);  
 context = args = **null**;  
 }  
 **return** result;  
 };  
}

## 理解typeof 和instanceof 验证数据类型的区别；

**一.instanceof运算符: 双目运算符** 此运算符可以判断一个变量是否是某个对象（类）的实例，返回值是布尔类型的。

使用示列： var obj = {}; obj instanceof Object 等于true;

说明： 该运算符必须是验证是引用类型的对象。即是通过构造函数创建的对象。

正确使用： var str = new String(“字符串”); str instanceof String 为 true .

错误使用： var str = “字符串”; str instanceof String 为 false。

1. **typeof运算符: 单目运算符**此运算符可以返回一个字符串，用语说明元算数的类型，它的返回值有如下可能:number,boolean,string,function,object,undefined 。

可以看到没有数组的类型， 因为数组是作为object使用的。

说明：一般用来验证基本数据类型。引用类型的数据一般返回的是object字符串。但是函数除外，可以返回function。

二者的区别： 验证区别在自定义的类对象上instanceof 可以验证自定义的类对象类型。

function Cat (){....}; var tempCat = new Cat();

Cat instanceof Function == tue; tempCat instanceof Cat == true;

还有数组比较特殊：需要用instanceof 来判断类型。 Var arr = []; arr instanceof Array == true;

## 10、深拷贝对象：

　　function deepCopy(p, c) {

　　　　var c = c || {};

　　　　for (var i in p) {

　　　　　　if (typeof p[i] === 'object') {

　　　　　　　　c[i] = (p[i].constructor === Array) ? [] : {};

　　　　　　　　deepCopy(p[i], c[i]);

　　　　　　} else {

　　　　　　　　　c[i] = p[i];

　　　　　　}  
　　　　}

　　　　return c;  
　　}