重新認識JavaScript

kevin

JavaScript 型別-原始型別

數值 (number)

字串 (string)

布林 (boolean)

null

undefined

JavaScript 型別-物件型別

原生物件 (Native)

原始型別包裹物件

Array

Date

Math

RegExp

function

宿主物件 (window document ……等所有非ECMAScript的實作)

原始型別 VS 物件型別

原始型別無法自由擴增屬性，也沒有對應的屬性可使用。

物件型別就可以自由擴增屬性，並且也可以刪除屬性。

相等與全等

'123' == 123 =>true

'123' === 123 =>false

NaN == NaN =>false

NaN === NaN =>false

基本型別與物件類別

'123' == new String('123'); =>true

'123' === new String('123'); =>false

typeof '123'.substr; =>'function'

typeof new String('123').substr; =>'function'

轉型

!!0 =>false

!!-1 =>true

!!Infinity =>true

!!-Infinity =>true

轉型

!!"" =>false

!!"0" =>true

!!"false" =>true

!!NaN =>false

轉型

!![] =>true

!!{} =>true

!!undefined =>false

!!null =>false

轉型

!!function(){}	=>true
!!new Boolean(false)	=>true
!!new Number(0)	=>true
!!new String('')	=>true

Higher-order function

某函數如果可以接受函式當作參數，或者以函式為傳出值，我們就稱這樣的函式為「較高次方函式」

```
var people = [{name:'kevin',age:26},{name:'aaron',age:31}];

people.filter(

    function(person){return person.age > 27;}

);
```

Global variable(全域變數)

```
var globalVar = '1';
function showVar(){
    console.log(globalVar);
}
```

Local variable(區域變數)

```
var myVar = '1';
function showVar(){
    var myVar = '2';
    console.log(myVar);
}
```

Block variable(區塊變數)？

```
var myVar = 1;

function scopeTest(){

    if(true){ var myVar = 2; } return myVar;

}

scopeTest();
```

變數提升-Variable Hoisting

```
function scopeTest(){
    var myVar;
    if(true){ myVar = 2; } return myVar;
}
scopeTest();
```

變數提升-Variable Hoisting

```
var var1 = 123;

function hoistingFunc(){

        if(!var1){  var var1=456; }

        return var1;

}

hoistingFunc();
```

變數提升-Variable Hoisting

```
var var1 = 123;

function hoistingFunc(){

        var var1;

        if(!var1){  var1=456; }

        return var1;

}

hoistingFunc();
```

函式宣告

```
functionDeclaration();

function functionDeclaration(){
    console.log('functionDeclaration');
}
```

函式表達式(匿名表達式)

```
var functionExpression = function(){

        console.log('functionExpression');

};

functionExpression();
```

函式表達式(具名表達式)

```
var functionExpression = function func1(){

        console.log('functionExpression');

};

functionExpression();

console.log(functionExpression.name);
```

函式宣告 vs 函式表達式

通常來說，建議使用函式表達式不建議使用函式宣告

ps:在不支援或未啟用ES6的瀏覽器跑跑看(EX:IE10 or Chrome Ver48以下)

```
function funcA(){  alert('fn1');}

if(false){

        function funcA(){    alert('new fn1');  }

}

funcA();
```

範圍鏈-Scope Chain

JavaScript程式在執行時會產生一個執行環境(Execution context)

每一個function會有一個自己的執行環境(Function execution context)

JavaScript在尋找變數時會沿著範圍鏈尋找context中的變數

範圍鏈-Scope Chain

```
var a = 0;

function outer(){

    var b = 1;//在outer中可以存取變數 a b

    function inner(){

        var c = 2;//在inner 中可以存取a b c

    }

}
```

立即函式-IIFE

立即函式(Immediately invoke function expression)
即立即執行一個函式

```
;(function(){
    //TODO
})();
```

立即函式-sample

```
;(function (name){

        console.log('hi,my name is ' + name);

})('kevin');
```

閉包closure

```
var person = (function (name){
        return {
                intor:function(){  console.log('hi,my name is ' + name); }
        };
})('kevin');
person.intor();
```

有無閉包的差別

```
for(var i =0; i < 5 ; i++){
        $('button' + i).on('click',function(){ console.log(i); });
}
for(var j =0; j < 5 ; j++){
        ;(function(j){
                $('button' + j).on('click',function(){ console.log(j); });
        })(j);
}
```

this in JavaScript

JavaScript的this會依據不同的運行情境指向不同物件

1.this指向於調用該函式之物件

2.this指向全域物件(瀏覽器：window物件、node.js：GLOBAL物件)

3.this指向利用call或apply所指派給this的物件

4.this指向new所產生之新物件

this in JavaScript

1.this指向於調用該函式之物件

```
var person = {
        name:'kevin',
        intor:function(){ console.log('hi my name is ' + this.name);}
};
person.intor()
```

this in JavaScript

2.this指向全域物件(瀏覽器：window物件、node.js：GLOBAL物件)

```
//呼叫a時不是以物件.XXX的形式，this指向全域物件
function a(){
    return this===window;
}
a();
```

this in JavaScript

3.this指向利用call或apply所指派給this的物件

```
var person = {
        intor:function(helloworld,mark){
                console.log(helloworld + mark+' ' +this.name);
        }
};
person.intor.call({name:'kevin'},'hi','!');//hi! kevin
person.intor.apply({name:'kevin'},['hi','!']);//hi! kevin
```

this in JavaScript

4.this指向new所產生之新物件

在解釋建構式時在說

建構式-Constructor

建構式呼叫的方法＝> new function(){}

一個建構式被呼叫時會有四個動作

1.建立一個空物件

2.將新物件的constructor指定成建構他的函式

3.指定新物件的prototype

4.呼叫函式(this指向新物件)

建構式-Constructor

1.建立一個空物件

```
function person(name){
	var _name = name;
	var _fnIntor = function(){console.log('hi my name is ' + name);};
	this.intor = function(){_fnIntor();};
};
var p = {};
```

建構式-Constructor

2.將新物件的constructor指定成建構他的函式

```
function person(name){
	var _name = name;
	var _fnIntor = function(){console.log('hi my name is ' + name);};
	this.intor = function(){_fnIntor();};
};
var p = {};
p.constructor = person;
```

建構式-Constructor

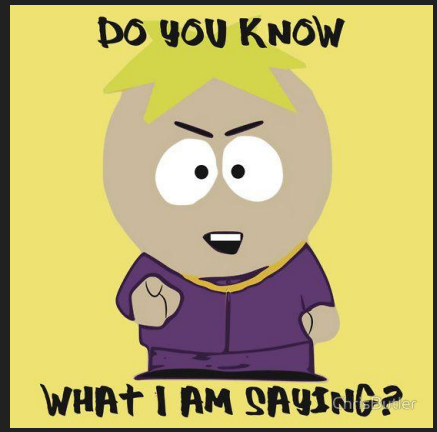
3.指定新物件的prototype

```
function person(name){
	var _name = name;
	var _fnIntor = function(){console.log('hi my name is ' + name);};
	this.intor = function(){_fnIntor();};
};
var p = {};
p.constructor = person;
p.prototype = person.prototype;
```

建構式-Constructor

4.呼叫函式(this指向新物件)

```
function person(name){
	var _name = name;
	var _fnIntor = function(){console.log('hi my name is ' + name);};
	this.intor = function(){ _fnIntor();};
};
var p = {};
p.constructor = person;
p.prototype = person.prototype;
person.call(p, 'kevin');
```



JavaScript 物件導向

JavaScript 是物件導向程式語言
=>所有物件皆繼承至Object
=>透過原形鏈(prototype chain)繼承

JavaScript 物件導向

```
var person = function(name){
    var _name = name;
    var _fnIntor = function(){console.log('hi my name is ' + name);};
    this.intor = function(){_fnIntor();};
};
new person('kevin').intor();
```

JavaScript 物件導向

```
var person = function(name){
    var _name = name;
    var _fnIntor = function(){console.log('hi my name is ' + name);};
    this.intor = function(){_fnIntor();};
};
person.prototype.sayHi = function (){console.log('hi');}
new person('alien').sayHi();
new person('kevin').intor();
```

JavaScript 物件導向-繼承

```
var gaPerson = function(){

        this.ga=function(){ console.log('ga');};

};

gaPerson.prototype = new person('asa');//person function 參考上一頁

new gaPerson().ga();

new gaPerson().intor();
```

HOW TO:
DRAW A HORSE

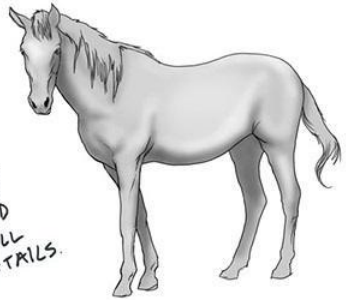
BY VAN OKTOP

(1) DRAW 2 CIRCLES

(2) DRAW THE LEGS

(3) DRAW THE FACE

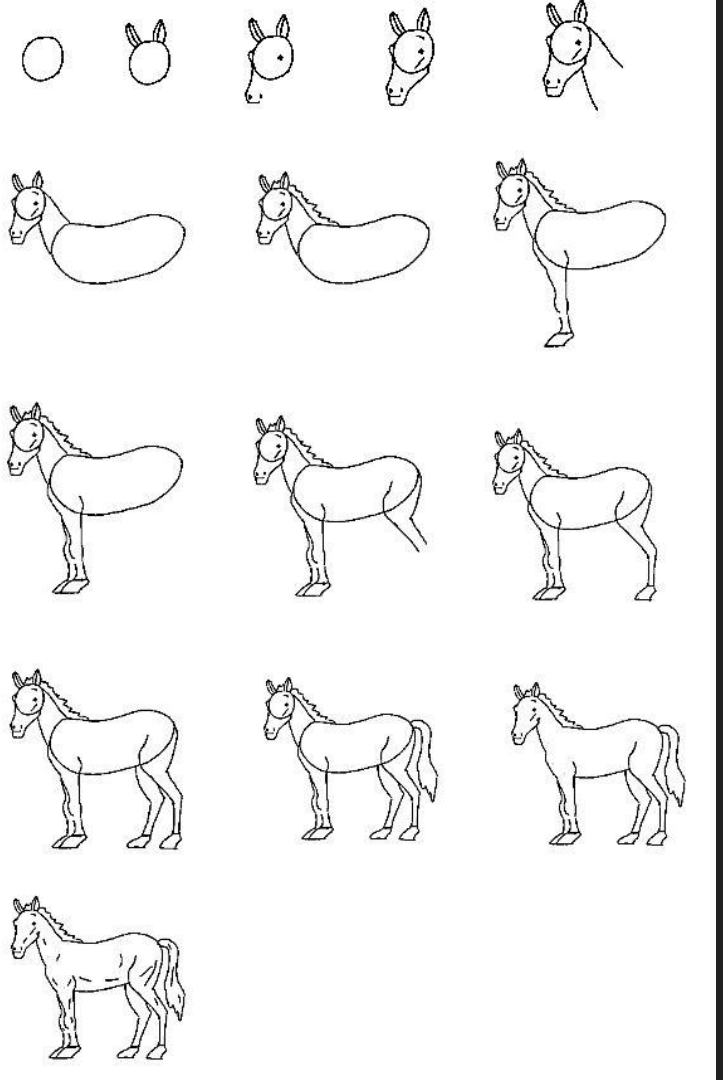
(4) DRAW THE HAIR



(5) ADD SMALL DETAILS.


大家不要緊張


因為我還沒發功啊!



Before we demo

Q & A