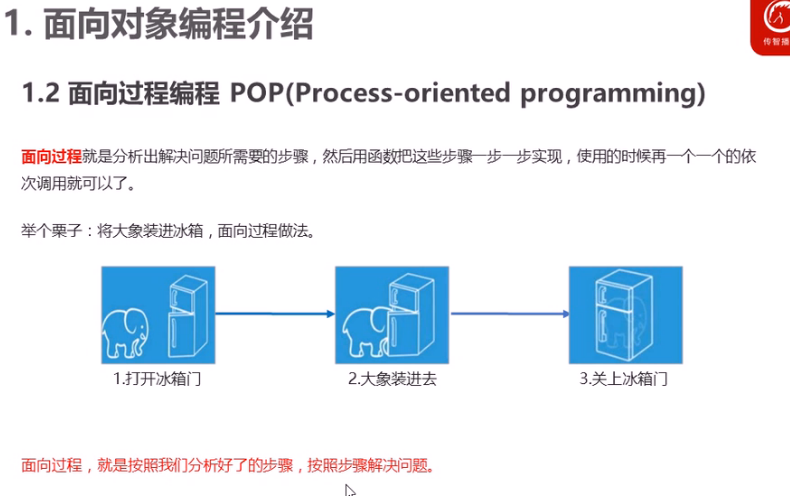
面向編程 POP 面向對象OOP Object Oriented Programming

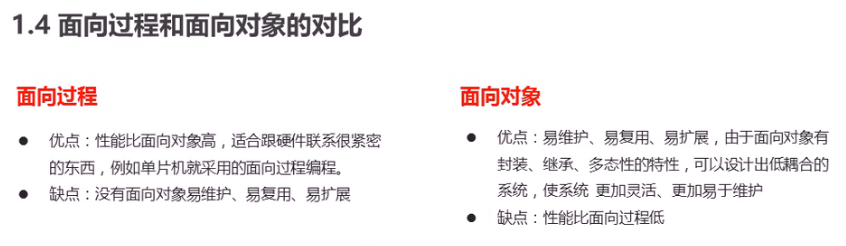




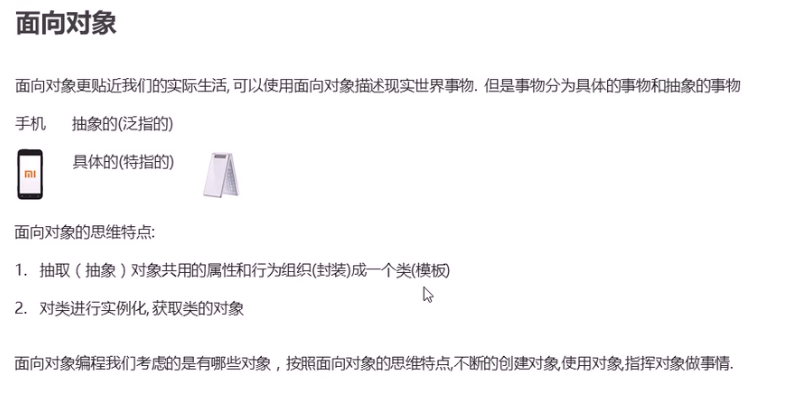
OOP 封裝性,繼承性,多態性



POP vs OOP



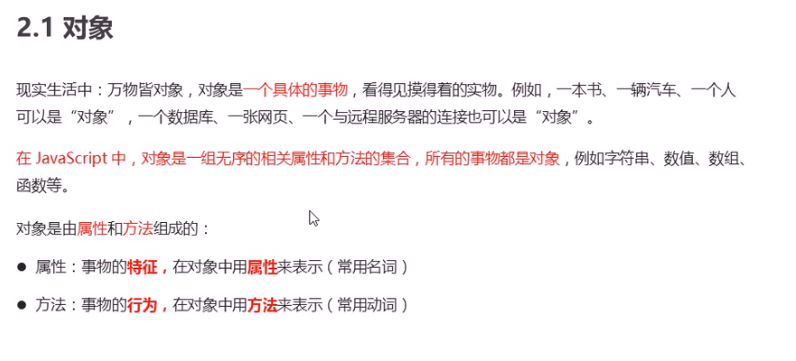
面向對象 類和對象



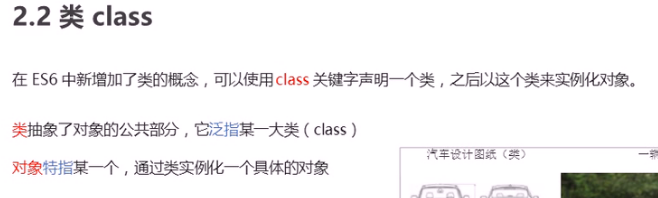
對象下有屬性和方法。

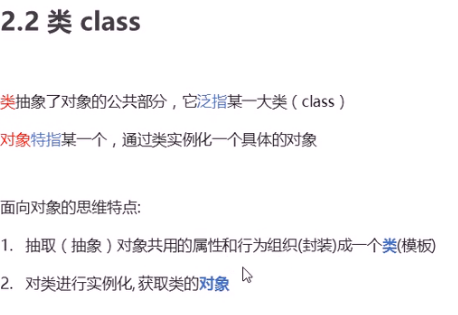
比如: Vue的element-ui 下有一個$message對象, 下面有error()方法,success()方法。

this.$message.error(xxx);



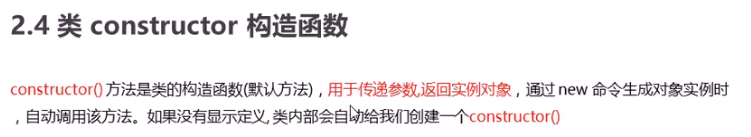
es6 新增了class 類





1.0創建類和生成實例

constructor 構造函數



new 生成實例

類的創建:

class 名 { 屬性,方法, constructor構造函數..}

Tab欄案例:

element.insertAdjacentHTML(position,text);

position 是相對於元素的位置

beforebegin

afterbegin

beforeend

afterend

四種position

appendChild不支持追加字符串,而insertAdjacentHTML是支持追加字符串元素。

   addTab() {

var li = '<li class="liactive"><span>新測試</span><img src="close.png" alt="" class="close"></li>';

that.ul.insertAdjacentHTML('beforeend', li); }

that.lis[index] && that.lis[index].click();

*// 如果&&前 為真 ,就調用 &&後*

*// 如果&&前 為假 ,就不調用了*

*e*.stopPropagation();

取消冒泡

window.getSelection ? window.getSelection().removeAllRanges() : document.getSelection.empty();

this.innerHTML = '<input type="text" />';

利用innerHTML 生成標簽

input.select(); *// 文本框里面的文字處於選定狀態*

監聽onblur \*\*\*\*\*.

input.onblur = function() {

*// 這里的this 是input*

                    this.parentNode.innerHTML = this.value;

                }

*// 接下enter也可以給span值 enter的ASCII碼 13*

input.onkeyup = function(*e*) {

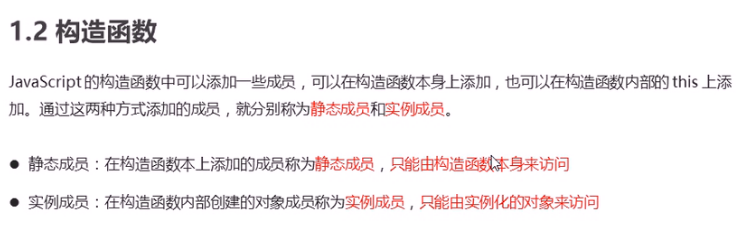
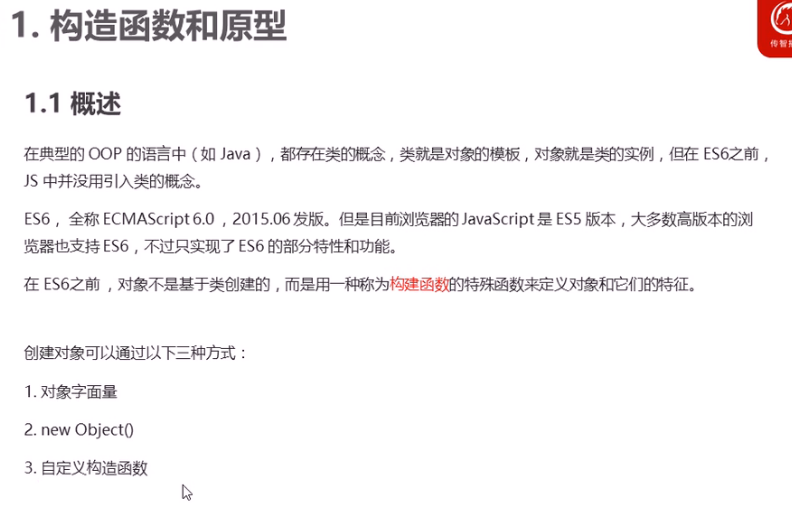
                if (*e*.keyCode == 13) {

       \*\*\*\*\*        this.blur(); *//直接調用onblur*

                }

            }

構造函數和原型 面試會問到!!!



實例成員 就是構造函數內部通過this添加的成員

只能通過實例化對象來訪問

比如peter.sing(); peter.uname;

是通過實例化的peter訪問成員

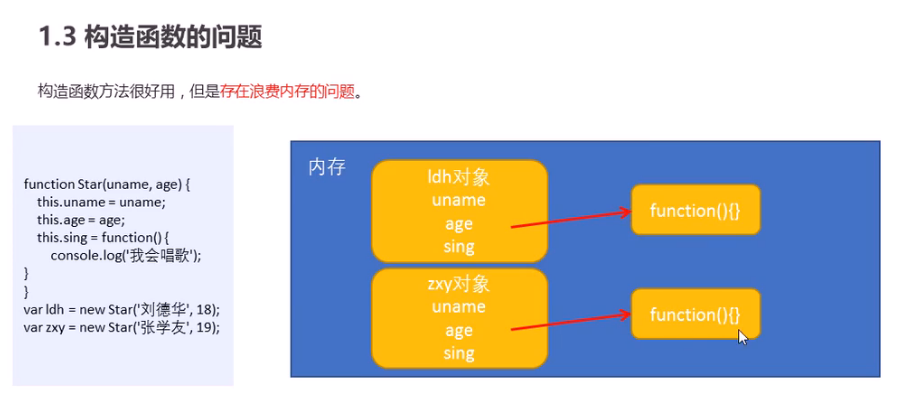
靜態成員 在構造函數本身上添加的成員

構造函數Star

比如Star.sex = “男”;

靜態成員 只能通過構造函數來訪問

原型對象prototype



我們希望所有的對象使用同一個函數, 這樣就比較省內存空間。

這時我們就要使用原型prototype

構造函數通過原型分配的函數是所有對象所共享的。

*// 把方法放到prototype原型對象內*

        Star.prototype.dance = function() {

            console.log(this.uname + '跳舞');

        }



**原型的作用: 共享方法, 節省內存**。

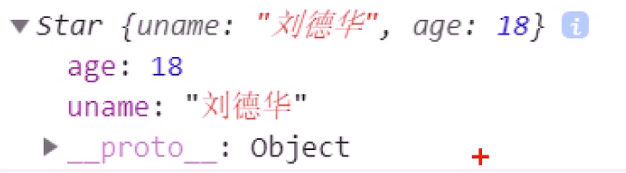
原型protoype :

不同的對象(由同一種構造函數new出來的),會在同一個構造函數內的prototype原型對象內,調用方法(在同一個地址)。 十分省內存。

如果不是在prototype內寫的方法, 因為方法是複雜類型, 每new一個對象就會有一個新的方法,也就開出新的內存去存放新對象的方法。

對象原型\_\_proto\_\_

對象會有一個屬性\_\_proto\_\_

****

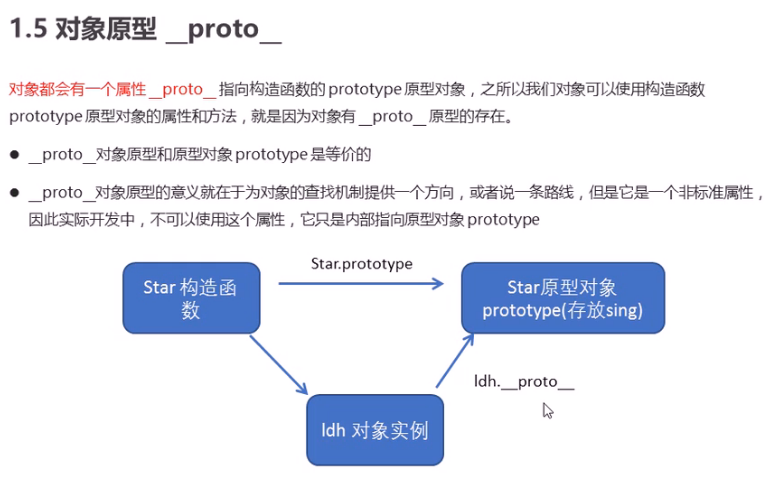
console.log(peter.\_\_proto\_\_ === Star.prototype);

*// 對象的\_\_proto\_\_ 與 構造函數的prototype 是同一個內存地址的東西*

*// 對象內如果有\_\_proto\_\_的方法 它就會去構造函數的prototype身上查找 這個方法*

**prototype 原型對象**

**\_\_proto\_\_ 對象原型**



constructor 構造函數

對象原型\_\_proto 和原型對象 prototype 內都有一個constructor 屬性,它指回構造函數本身。

行多情況下,我們需要手動利用constructor 這個屬性指回 回來的構造函數

*// 利用對象字面量, 但是會複蓋走prototype和\_\_proto\_\_內的constructor, 所以我們要手動利用constructor指回來*

        Star.prototype = {

*// 手動constructor 指回 構造函數本身。*

            constructor: Star,

            sing: function() {

                console.log(this.uname + '唱歌');

            },

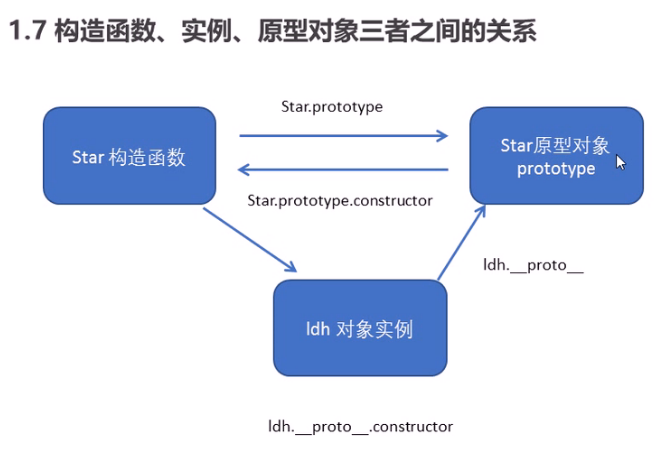
            movie: function() {

                console.log(this.uname + '演電影');

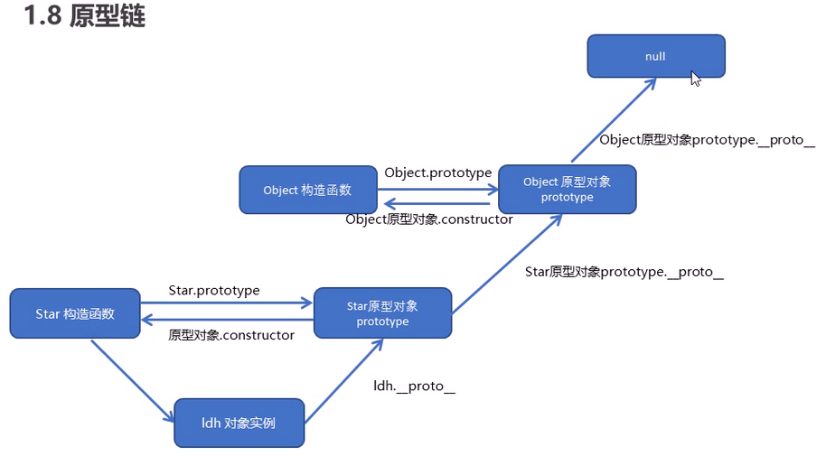
            }

        };

構造函數, 實例, 原型對象 三者之間的關系



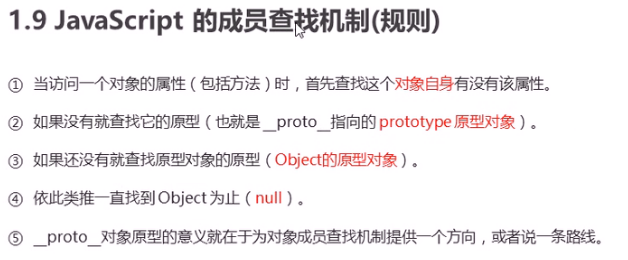
原型鏈 面試常問!!



Object是所有類的父類 (最大的)

Object(js內置構造函數) > Star(自定義構造函數) > ldh(利用Star實例化的對象)

原型鏈: 對象成員查找規則



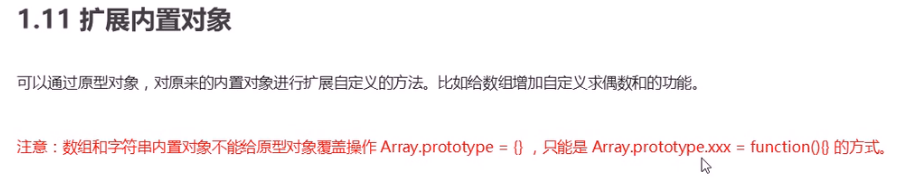
就近原則,如果近有該屬性,就直接返回。 沒有的話就向下查找。

比如 只有Object有toString() 這個方法,

Star對象和Star實例化的ldh都沒有toString(),

但是Star和ldh都可以調用toString(), 因為繼承性。

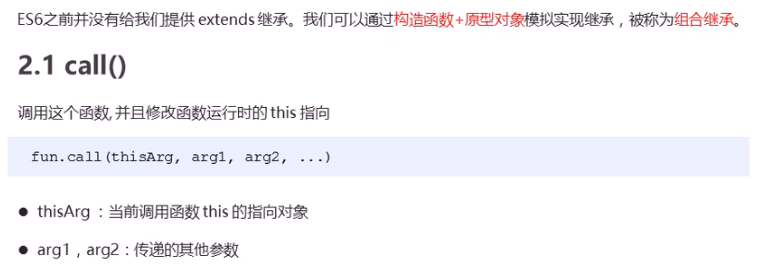
11 利用原型對象擴展內置對象方法



會連創建數組的方法都複蓋走。 直接報錯 無法創建數組。

繼承:

12 call方法的作用(繼承)



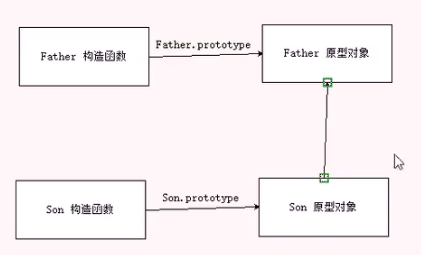
13 借用構造函數繼承父類型**屬性**

es6之前沒有extends繼承 , 我們可以通過**構造函數 + 原型對象** 實現繼承,也稱為**組合繼承**。



通過call() 把父類型的this指向 子類型的this 實現繼承

14 利用父構造函數繼承**方法**

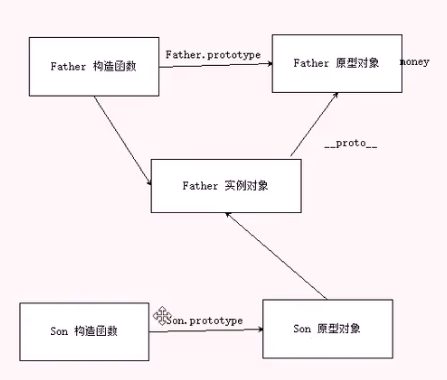


*// 賦值prototype 實現 方法的繼承  (不好, 因為這使son.prototype的內存地址 = father.prototype的內存地址)  之後我們添加方法給son的prototype時,會因為地址問題 father同時添加*

*// Son.prototype = Father.prototype;*

利用

Son.prototype = new Father();



*// 如果利用對象的形式修改了原型對象, 一定要加回constructor 指回原來的原型對象*

        Son.prototype.constructor = Son;

這樣我們添加方法給Son原型對象prototype時, 就不會影響到Father原型對象prototype了

15 類的本質

es6之前, 我們用構造函數 + 原型實現面向對象 編程

es6之後 通過類 實現面向對象編程

class 類 的本質 還是function (js 語法糖)

1類和構造函數 都有 原型對象prototype

2類和構造函數 的prototype 里面都有一個constructor 指向自己本身

3類和構造函數 都可以通過原型對象添加方法

4類和構造函數 創建實例對象有 \_\_proto\_\_ 指向 類/構造函數的原型對象prototype

es6 大多數功能 es5都可以做到, 所以es6 是一個 語法糖。

es5 新增的方法

數組方法: forEach() map() filter() some() every()

1.forEach()

array.forEach(function(currentValue,index,arr));

currentValue 數組當前項的值

index 數組當前的索引

arr 數組對象本身

2.filter()

array.filter(function(currentValue,index,array) { \*\*\*array可省略\*\*\*

return \*\*\*condition\*\*\*;

});

filter方法 創建一個新數組, 新數組中的元素是通過檢查指定數組中 符合條件的所有元素, **filter()主要用於篩選數組**。

**filter 返回一個新數組**

currentValue 數組當前項的值

index 數組當前的索引

arr 數組對象本身

**記得用一個新的數組去接收filter()**  返回return的新數組。

**var arr2**=arr.filter(function(*value*,*index*){

            return *value* > 20;

        });

3.some()

array.some(function(currentValue,index,array){

return \*\*\*condition\*\*\*;

});

some()方法用於檢測數組中的元素是否滿足指定條件, 滿足條件的元素**是否存在**

some()返回值是boolean, 如果滿足條件就true,不滿足就false;

**如果找到第一個滿足條件的元素,則終止循環, 返回true**。\*\*\*

currentValue 數組當前項的值

index 數組當前的索引

arr 數組對象本身

**記得用一個新的variable去接收 some()的 return返回**

 var flag = arr.some(function(*value*, *index*, *array*) {

            return *value* >= 20;

        });

some() forEach() 分別

forEach() 內的 return不會終止疊代(編歷)

some() 內的return 會直接終止函數的執行。

但some() 記得return true; 如果return false; 函數不會終止。

filter() 的return 也不會終止疊代(編歷)

20 trim方法去除字符串兩側空格

str.trim();

trim() 方法不影響原字符串本身, 因為它**返回一個新的字符串**。

實用例子

btn.onclick = function() {

            if (input.value.trim() === '') {

                alert('please enter something!');

            } else {

                console.log(input.value.trim());

                console.log(input.value.trim().length);

                div.innerHTML = input.value.trim();

            }

        }

21 對象方法Object.keys()方法

Object.keys(obj) 用於**獲取對象自身所有的屬性(返回新數組)**。

取鍵不取值。

效果類似 for…in

返回一個由屬性名組成的數組

22 Object.defineProperty方法 \*\*\* 數據劫持\*\*\*

Object.defineProperty() 定義新屬性或修改原有的屬性

Object.defineProperty(obj,prop,descriptor);

三個參數都是 必需寫的

obj 目標對象

prop 新定義/修改的屬性名(鍵)

descriptor 目標屬性擁有的特性

**descriptor的說明:** 以對象形式 { } 書寫

value: 設置屬性的值 默認為undefined

writable: 值是否可以重寫, true | false 默認為false

enumerable: 目標屬性是否可以被枚舉(編歷), true | false 默認為false

configurable: 目標屬性是否可以被刪除 或 是否可以再次修改特性 true | false 默認為false。

writable: false 比如id, index 我們想不可以修改, 就用writable: false;

configurable: true 使屬性 不可刪除 不可修改

enumerable: true 使屬性 在Object.keys(obj)下 不可編歷

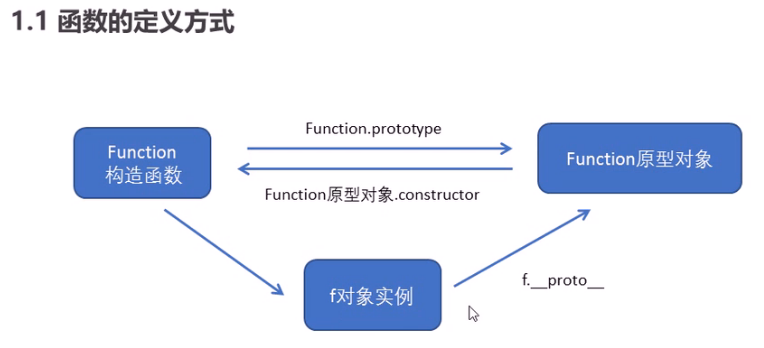
但console.log(obj) 下 還是可以log出的…

23 函數進階

學習目標



23 函數進階-函數定義與調用



24 函數的調用方法(6種)

1. 普通函數

2. 對象的方法

3. 構造函數

4. 綁定事件函數

5. 定時器函數

6. 立即執行函數

25 函數內部的this指向



window 頂級對象

原型對象里面的方法

Star.prototype.sing = function() {xxx}; 也指向實例對象

比如: ldh.sing();

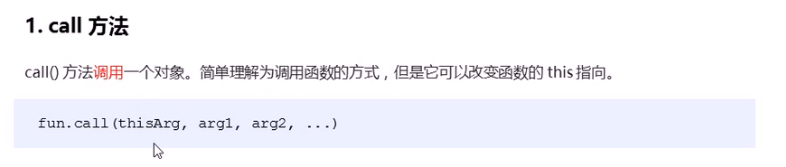
window.fn();

window.setTimeout(function(){xxxxx},1000);

三種改變函數內部 this指向

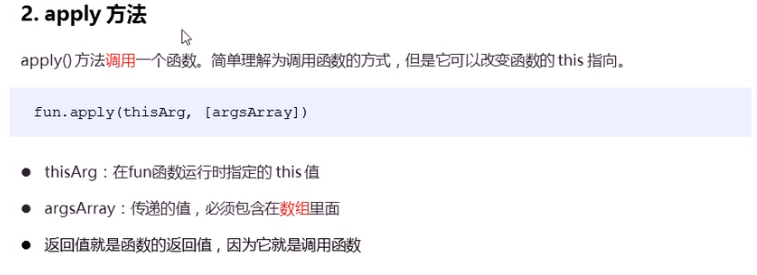
常用的有 call() apply() bind() 三種

26 call方法及其應用



26 call方法及其應用

27 apply方法及其應用



apply 的參數是數組(偽數組)

返回值 就是函數的返回值, 因為它就是調用函數

*// apply主要應用 比如說我們可利用 apply 借助於數學內置對象*

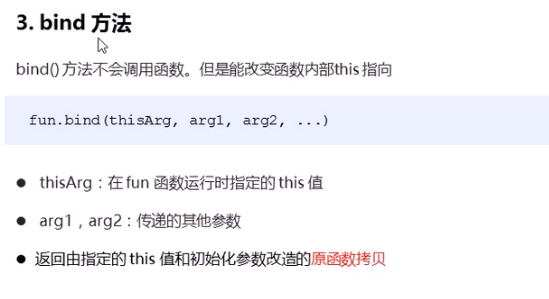
*// Math.max();*

        var arr1 = [1, 66, 4, 33, 50]

        var res = Math.max.apply(null, arr1);

        console.log(res);

28 bind方法及其應用



\*\* bind方法,不會調用函數

返回由指定的this值的初始化參數改造的**原函數拷貝**。

call 改變指向 繼承父函數

apply 改變指向 調用函數 以數組形式傳參

bind 改變指向 不會調用函數 返回新指向的原函數(拷貝)

bind 應用

 btn.onclick = function() {

            this.disabled = true;

*// var that = this;*

            setTimeout(function() { *// setTimeout的this 指向window   所以我們利用bind()*

*// that.disabled = false;*

                this.disabled = false;

            }.bind(this), 2000); *// 這bind(this) 就指向了btn*

        }

bind(this) 的this是指向btn, 改變了setTimeout內的this指向,

由本身(指向window) 改變為(指向btn)。

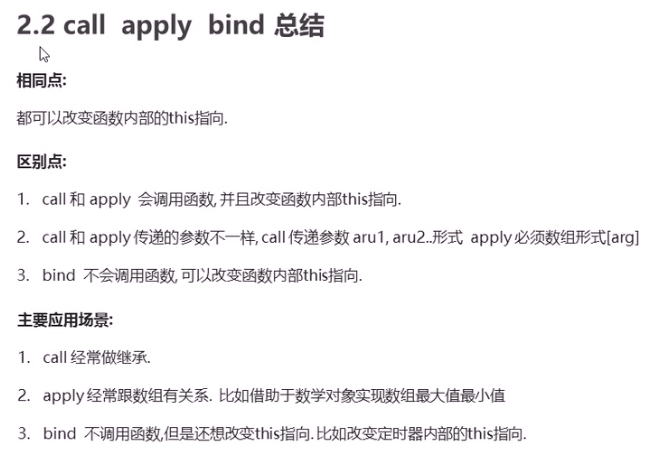
定時器是異步函數

<https://www.bilibili.com/video/BV1Kt411w7MP?p=58>

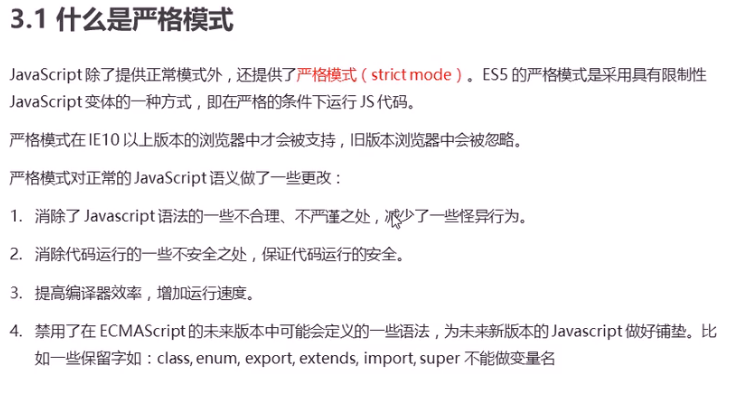
bind 應用對象tab欄, bind(this.lis[i],this)

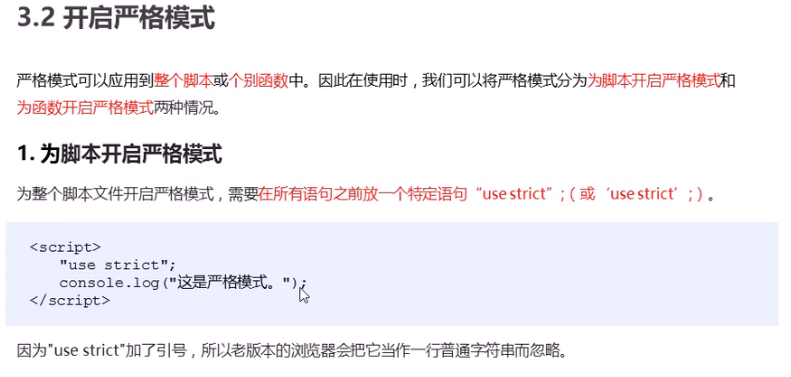
如果用到改變兩個以上的this指向, 可以以參數傳遞的方式傳指向

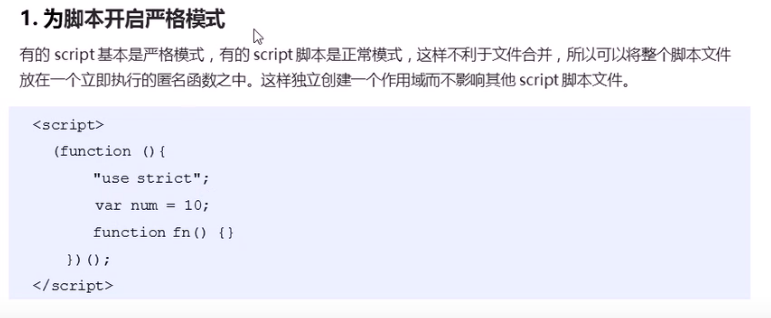
call apply bind 總結 應用

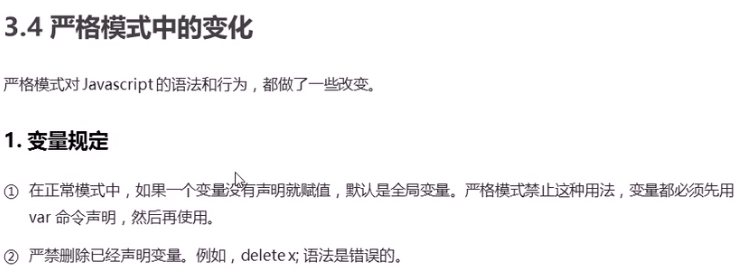


29 嚴格模式strict mode (主要為了提高性能,減少編譯)

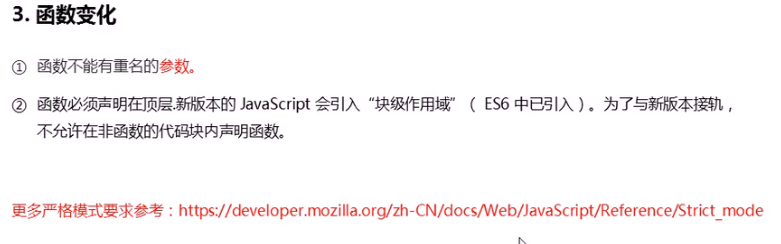












'use strict';

if (true) {

function f() { } // !!! syntax error

f();

}

for (var i = 0; i < 5; i++) {

function f2() { } // !!! syntax error

f2();

}

function baz() { // kosher

function eit() { } // also kosher

}

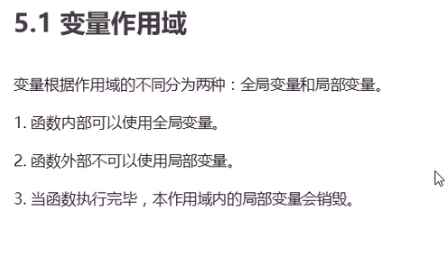
30 高階函數

高階函數: 參數 或 返回值 是函數的函數

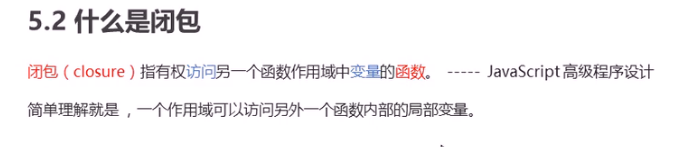


31 閉包

js中 閉包和異步是兩大難點



閉包(closure)指 有權被**另一個函數作用域**訪問自己的變量的函數。 (上圖是錯的)



作用域(scope) 局部(local) 全局(global)

*// fn 就是閉包函數 fn有權被fn2訪問*

        function fn() {

            var num = 10;

            function fn2() {

                console.log(num);

            }

            fn2();

        }

        fn();

*// fn 就是閉包函數 fn有權被fn2訪問*

閉包的主要作用: 延伸了變量的作用範圍

deep copy 跟shallow copy的差別是 shallow copy會把變數的整個記憶體位置複製給它使得只要更動其中一個變數的值兩個變數就都會更動，而deep copy則會再空出一個記憶體位置給它且只會把另一個變數的值複製給它，這樣就不會有兩個變數同時更動的問題了！

32 閉包應用 (同步任務,異步任務)

用立即執行函數

 for (var i = 0; i < lis.length; i++) {

*// 利用for循環 創建4個立即執行函數*

            (function(*i*) {

*//立即執行函數  立即執行函數的參數i 會傳遞到這個函數內*

                lis[*i*].onclick = function() {

                    console.log(*i*);

                }

            })(i);

        }

實現閉包 把異步任務(點擊綁定事件) 變成 同步任務

利用立即執行函數 解決異步任務的問題\*\*\*\*\*

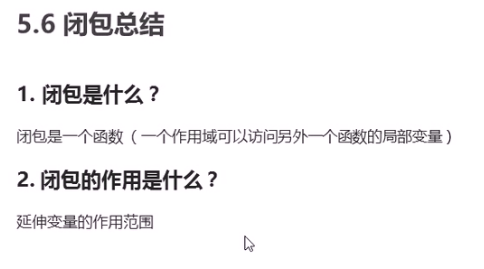
eg. setTimeout,for循環內的綁定事件(點擊時才執行的異步函數)\*\*\*\*\*

34 閉包應用-計算打車價格

變量所在的函數是閉包(被訪問的變量)

35 思考題

*// 立即執行函數,匿名函數  的this 都是指向window*



局部變量在函數完成後,會立即燒毀。 所以一個函數寫在另一個函數內, 是延伸了變量的作用範圍。

38 遞歸, 遞歸函數



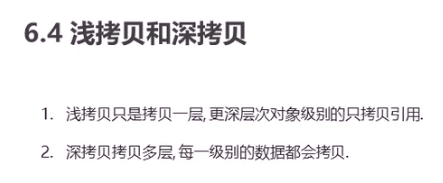
遞歸可能會發生 棧溢出(stack overflow), 所以我們必須要加退出條件 return;

39 利用遞歸求階乘

41 利用遞歸編歷數據

\*\*\*\*\*(遞歸查詢子部門數據十分有效)

42 淺拷貝



淺拷貝只是拷貝一層,更深層對象級別的只拷貝引用。

深拷貝 拷貝多層,每一級別的數據都會拷貝

43 淺拷貝方法 Object.assign()

Object.assign(target, …sources) es6新增方法,可以淺拷貝

44 深拷貝

\*\*\*\*\*

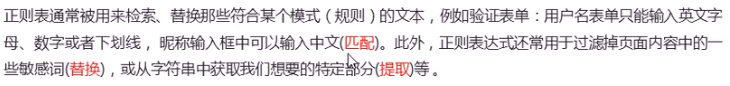
對象,數組 是複雜數據類型, 我們利用遞歸函數 實現深拷貝

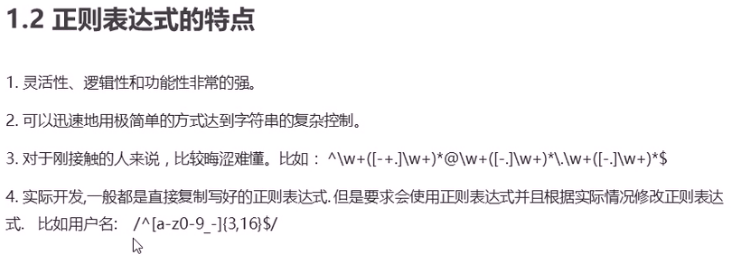
如果沒有利用遞歸函數,就可能是淺拷貝(複雜數據類型會直接拷貝地址)。

45 正則表達式

正則表達式 Regular Expression 是用於匹配字符串中字符組合的模式,

在JS中,正則表達式也是對象。





在JS中,有兩種方法創建正則表達式對象

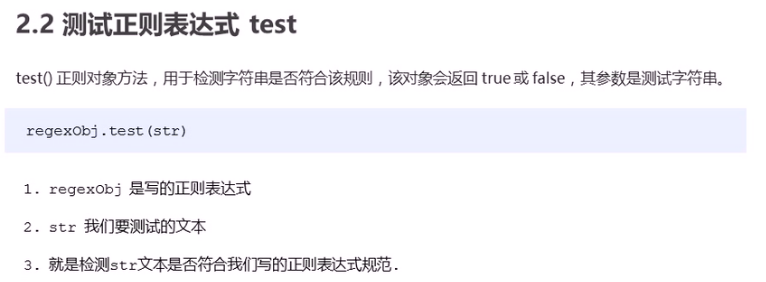
1. 調用RegExp對象的構造函數創建

var xxx = new RegExp(/表達式/);

1. 利用字面量創建

var xxx = /表達式/ ;

regexObj.test(str) 測試正則表達式



46 邊界符



^ 表示匹配行首的文本(以誰開始)

$ 表示匹配行尾的文本(以誰結束)

^和$ 一起用 實現精確匹配

47 字符類

[內容]

字符類: [] 表示有一系列字符可供選擇, 只要匹配**其中一個**就可以了

比如var rg = /[abc]/ 只要包含有 a 或 b 或c 都返回true;

[-] 中括號內部 範圍符-

[^]中括號內部 ^ 表示取反

[] 只能是單個字符

48量詞符



49 量詞重複某個模式的次數

量詞符中間記得不要有空格!!!!\*\*\*\*\*

50 用戶名表單驗證

51 括號總結與正則驗證工具

*// 中括號 字符集合,匹配中括號內任意字符*

*// 大括號 量詞符,里面表示重複次數*

*// 小括號 表示優先級*

52 預定義類與座機號碼驗證



\s和\S 重要。

53 表單驗證案例

可以封裝 表單驗證 function RegExp(input,reg){…. this.nextElementSibiling.className = ‘success/wrong’

this.nextElementSibiling.innerHTML= ‘正確/錯誤’};

nextElementSibiling就是input之後的span文字提示。

中文正則表達式 : ^[\u4e00-\u9fa5]{2,8}$

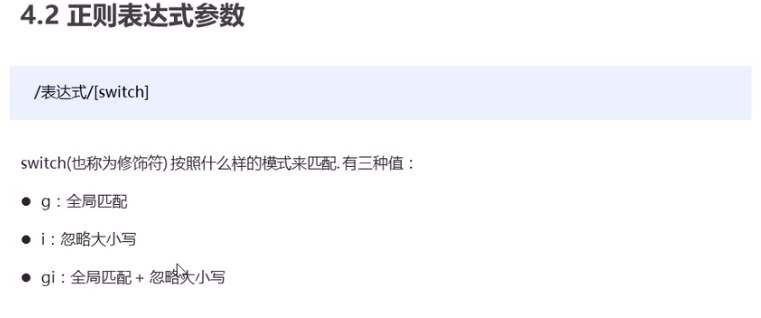
54 正則替換

replace() 可實現 替換字符串操作,

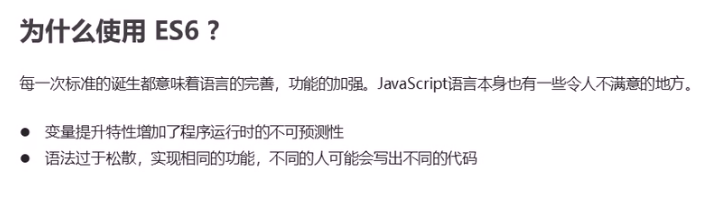
用來替換的參數可以是一個**字符串或是正則表達式**。

stringObject.replace(regexp/substr , replacement);

返回值是新的字符串



55 es6~es11 開始







56 let變量聲明

*// 1.let 變量不能重複聲明 (可理解為 strict mode)   但var 可以(不夠嚴格)*

*// 2.塊級作用域(只在代碼塊內有效) 全局,函數,eval*

*// if else while for 內的{}都是塊級作用域*

 {

            let girl = 'BBchan';

        }

        console.log(girl);  *// 報錯, 無法調用girl*

*// 3. let 沒有變量提升*

*// 4.不影響作用域鏈 (是塊級作用域, 但不影響作用域鏈)*

{

            let school = 'school';

            function fn() {

                console.log(school); *//可以調用school*

            }

            fn();

        }

a

57 let案例與實踐

*// for內 let i 會使 let 分為3個不同的作用域*

        {

            let i = 0;

            items[i].onclick = function() {

                items[i].style.background = 'pink'; *// 異步*

            }

        } {

            let i = 1;

            items[i].onclick = function() {

                items[i].style.background = 'pink'; *// 異步*

            }

        } {

            let i = 2;

            items[i].onclick = function() {

                items[i].style.background = 'pink'; *// 異步*

            }

        }

解決異步問題

58 const常量聲明與特點

*// 5.對於數組和對象的元素修改, 不算做對常量的修改, 不會報錯*

*// const 對象和數組保存的是內存地址(對象和數組內的元素改變了,但地址沒變 所以沒有報錯)*

*// 引用的地址值 地址值沒變*

59 變量的解構賦值

es6允許按照一定模式 從數組和對象中提取值, 對變量進行賦值, 這被稱為 解構賦值

60 模板字符串

61 對象簡化寫法\*\*

62 箭頭函數

*// 1. this 是靜態的  箭頭函數this始終指向函數聲明時{所在作用域}下的 this的值*

*// 2.不能作為構造函數 實例化對象*

*// 3.箭頭函數 不能使用arguments 變量*

*// 4.箭頭函數 簡寫 (不好用)*

63 箭頭函數的實踐與應用

*// 箭頭函數 適合與 this無關的回調, 定時器, 數組的方法回調*

*// 箭頭函數 不適合與 this有關的回調, 事件回調, 對象的方法*

64 函數參數的默認值設置

*// 1.形參初始值  具有默認值的參數, 一般規範放後面*

*// 2.與解構賦值結合*

*// 與解構賦值結合寫法*

        function connect({

*host* = '127.0.0.1', *// 如果沒有傳host 就給默認值127.0.0.1*

*username*,

*password*,

*port*

        }) {

            console.log(host);

            console.log(username);

            console.log(password);

            console.log(port);

        }

        connect({

*// host: 'google.com',*

            username: 'root',

            password: 'root',

            port: 3306

        })

65 rest …參數

rest …放在 函數聲明的 **形參**內

66 … 擴展運算符

擴展運算符 … 放在 函數調用的 **實參**內

67 … 擴展運算符的應用

*// 1.數組的合并*

*// 2.數組的copy   如果有引用類型數據,還是淺拷貝*

*// 3.將偽數組 轉換為 真正的數組*

JavaScript 中共有 6 種基本數據類型：Undefined、Null、Boolean、Number、String、Symbol (new in ES 6) ！

除過上面的 6 種基本數據類型外，剩下的就是引用類型了，統稱為 Object 類型。細分的話，有：Object 類型、Array 類型、Date 類型、RegExp 類型、Function 類型 等。

簡單(基本)數據類型: Number、String、Boolean、Undefined、Null

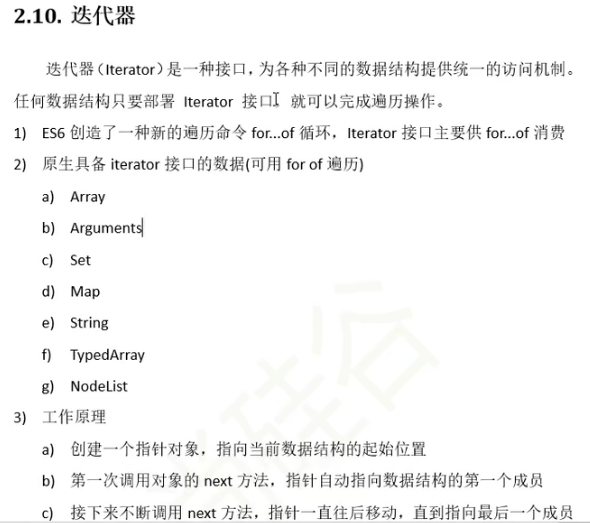
複雜(引用)數據類型: Object、Array、Date、RegExp、Function等

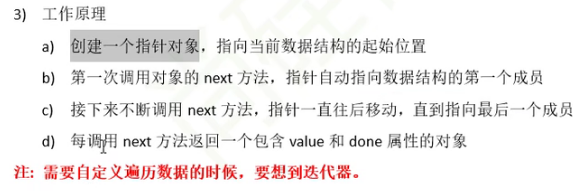
68 Symbol的介紹與創建

70 Symbol內置屬性



71 疊代器 iterator 介紹 for of for…of





[Symbol.iterator]() { *// 手動設計iterator*

*// 索引變量*

                let index = 0;

*//*

                let \_this = this;

                return {

                    next: function() {

                        if (index < \_this.stus.length) {

                            const result = {

                                value: \_this.stus[index],

                                done: false *// done是 是否結束*

                            };

*// 下標自增*

                            index++;

*// return結果*

                            return result;

                        } else {

                            return {

                                value: undefined,

                                done: true

                            };

                        }

                    }

                };

            }

73 生成器函數聲明和調用

生成器函數是es6提供的一種異步編程解決方案, 語法行為與傳統函數完全不同

74 生成器函數的參數傳遞

77 Promise基本介紹與使用

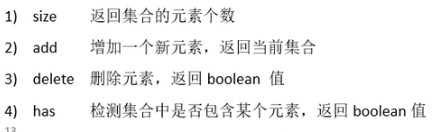
面試重點



83 set集合介紹與API

es6提供了新的數據結構 set(集合) ,set類似於數組,但成員的值都是唯一的,集合實現了iterator接口, 所以可以使用[擴展運算符]和for…of 進行編歷。

set集合的屬性和方法:



5. clear 清空set集合

集合沒有重複性, 所以會自動去除重複的值。

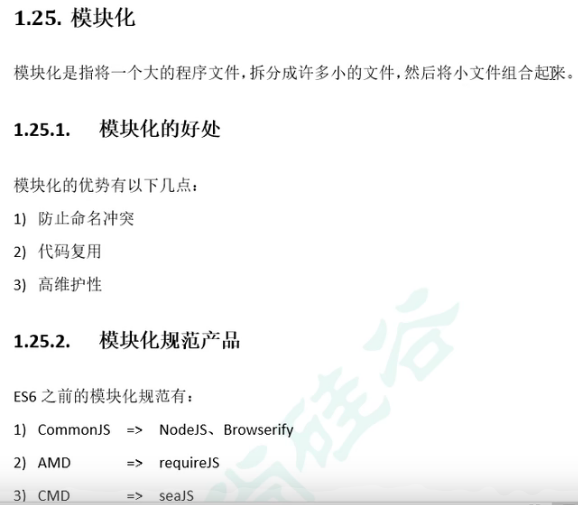
85 Map的介紹與實踐

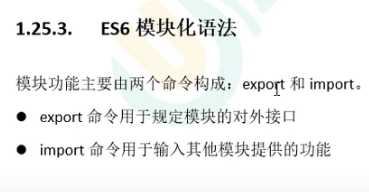


92 es6的數值擴展

Math.sign()*判斷一個數為 正數 負數 或0*

94 es6模塊化介紹,優勢和產品





100 es7新特性

3.1Array.prototype.includes

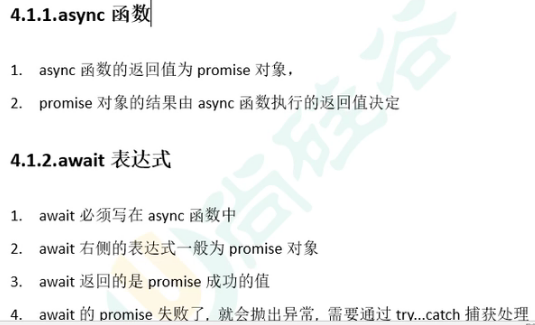
includes 用來檢測數組中是否包含某個元素,返回boolean值。

3.2指數操作符

\*\*, 用來實現 幕運算(次方), 與Math.pow結果相同

101 es8 async函數

async與await兩種語法結合, 可以讓異步代碼僕同步代碼一樣



102 es8 await函數

105 es8 Object對象方法

Object.values() 方法,返回一個給定對象的所有可枚舉屬性值的數組

Object.entries() 方法,返回一個給定對象自身可編歷屬性[key,value]的數組

Object.getOwnPropertyDescriptors

可獲取對象的屬性描述 eg: writable,configurable,enumerable

可看22 Object.defineProperty方法 \*\*\* 數據劫持\*\*\*

106 es9擴展運算符與rest參數

107 es9正則擴展-命名捕獲分組

let str2 = '<a href="http://www.google.com">google.com</a>';

        const reg2 = /<a href="(?<url>.\*)">(?<text>.\*)<\/a>/;

        const result2 = reg2.exec(str2);

108 es9正則擴展,反向斷言.html

109 es9正則擴展,dotAll模式

*// dot . 元字符  表示 除了換行符以外的任意單個字符*

高級正則表達式

. 表示 匹配除换行符 \n 之外的任何单字符

\*表示零次或多次

所以.\*在一起就表示任意字符出现零次或多次

?表示贪婪模式

比如a.\*b，它将会匹配最长的以a开始，以b结束的字符串。如果用它来搜索aabab的话，它会匹配整个字符串aabab。这被称为贪婪匹配。

a.\*?b匹配最短的，以a开始，以b结束的字符串。如果把它应用于aabab的话，它会匹配aab（第一到第三个字符）和ab（第四到第五个字符）。

**圓括號()是組**，主要應用在限制多選結構的范圍/分組/捕獲文本/環視/特殊模式處理

示例：  
1、(abc|bcd|cde)，表示這一段是abc、bcd、cde三者之一均可，順序也必須一致  
2、(abc)?，表示這一組要麼一起出現，要麼不出現，出現則按此組內的順序出現  
3、(?:abc)表示找到這樣abc這樣一組，但不記錄，不保存到$變量中，否則可以通過$x取第幾個括號所匹配到的項，比如：(aaa)(bbb)(ccc)(?:ddd)(eee)，可以用$1獲取(aaa)匹配到的內容，而$3則獲取到了(ccc)匹配到的內容，而$4則獲取的是由(eee)匹配到的內容，因為前一對括號沒有保存變量  
4、a(?=bbb) 順序環視 表示a後面必須緊跟3個連續的b  
5、(?i:xxxx) 不區分大小寫 (?s:.\*) 跨行匹配.可以匹配回車符

**方括號是單個匹配**，字符集/排除字符集/命名字符集  
示例：  
1、[0-3]，表示找到這一個位置上的字符只能是0到3這四個數字，與(abc|bcd|cde)的作用比較類似，但圓括號可以匹配多個連續的字符，而一對方括號只能匹配單個字符  
2、[^0-3]，表示找到這一個位置上的字符只能是除了0到3之外的所有字符

116 String.prototype.matchAll方法 \*\*\*十分實用

for of 83章與71章

5.3 for-in 語句

for-in 常用於 對象的編歷。

*//for-in 常用於 對象的編歷*

        var obj = {

            name: 'peter',

            age: 18,

            sex: '男'

        } // *k for key*

        for (var k in obj) { *//    for-in 是個循環*

            console.log(k); *// 輸出 屬性名*

            console.log(obj[k]); *//輸出 屬性值*

        }

for-in 取鍵名(屬性名)key

for (var key in *student*) {

            objStudent[key] = *student*[key];

        }

數據上 objStudent對象的鍵名順序 與 student的鍵名順序不同。

但for-in 可以因對應的鍵名 而 賦值對應的值。

for-of 取值(屬性值)value

for-of 是iterator疊代器

需要自定義編歷數組時,就要想到for-of iterator 。

for in vs for of

