目录

[ES6 面向对象 2](#_Toc104931130)

[1.静态成员直接在类名上建立 只能通过类名访问 2](#_Toc104931131)

[2.prototype给原型加入新的属性或方法 原型中的一个属性 2](#_Toc104931132)

[3.\_\_protot\_\_ 一样 在实例化对象上 由浏览器添加 2](#_Toc104931133)

[4. super 调用父类中的函数 2](#_Toc104931134)

[5. 2](#_Toc104931135)

[6. class Son extends Father // extends 继承 //extends--扩展 3](#_Toc104931136)

[7. constructor 里面的this 指向的是 创建的实例对象 3](#_Toc104931137)

[8.类里面有个constructor 函数,可以接受传递过来的参数,同时返回实例对象 3](#_Toc104931138)

[9.//如果我们修改了原来的原型对象,给原型对象赋值的是一个对象,则必须手动的利用 3](#_Toc104931139)

[10. fn.call(o, 1, 2); fn是函数名 1和2是传参 o更改指向的对象 4](#_Toc104931140)

[11.创建一个类 4](#_Toc104931141)

[12.arr.forEach(function(value, index, array) { 4](#_Toc104931142)

[13.filter--过滤器 筛选数组 返回符合要求的拷贝至newArr 5](#_Toc104931143)

[14.fird--some 查找元素返回布尔值 5](#_Toc104931144)

[15.trim--修剪 删除首尾空格 ，中间空格不会删除 5](#_Toc104931145)

[16.Object.keys(对象) 会遍历对象中所有元素 并返回 成为一个数组 5](#_Toc104931146)

[17. Object.defineProperty() //对象,修改或增添属性名,内容 5](#_Toc104931147)

[18.// 2. 函数表达式 (匿名函数) 6](#_Toc104931148)

[19.this 指向调用者 普通函数直接调用 调用者是windows 6](#_Toc104931149)

[20.改变函数内this 指向又call() apply() bind() 6](#_Toc104931150)

[21.'use strict' 开启严格模式 7](#_Toc104931151)

[22.闭包的使用 普通是给每一个元素个index属性赋值 再访问 ，下面写法未使用到index属性 7](#_Toc104931152)

[23.//个人理解 闭包:将一块函数封装起来，隔离所有，只留下外界的调用函数,里面的东西与外界无关 8](#_Toc104931153)

[24.可以将函数作为返回值 8](#_Toc104931154)

[25.深浅拷贝 9](#_Toc104931155)

[26.正则表达式 new RegExp 获取 字面量直接创建 9](#_Toc104931156)

[27. let const 声明常量具有块级作用域 即 大括号内才有作用 10](#_Toc104931157)

[28.数据解构 快速赋值 11](#_Toc104931158)

[30. ... 剩余参数 最后面是定义一个数组 存储剩余的数据(可以是如下解构，也可以是函数传参) 11](#_Toc104931159)

[31. ... // 利用扩展运算符将伪数组转换为真正的数组 11](#_Toc104931160)

[32.面试题 12](#_Toc104931161)

[33. from 将伪数组(可迭代的对象)转化为正真的数组 12](#_Toc104931162)

[33.find(函数(传参--key)) 查找 返回查找到的对象或值 12](#_Toc104931163)

[34. ${fn()} 占位符 13](#_Toc104931164)

[35. 字符串的查询方法 string str=""; 13](#_Toc104931165)

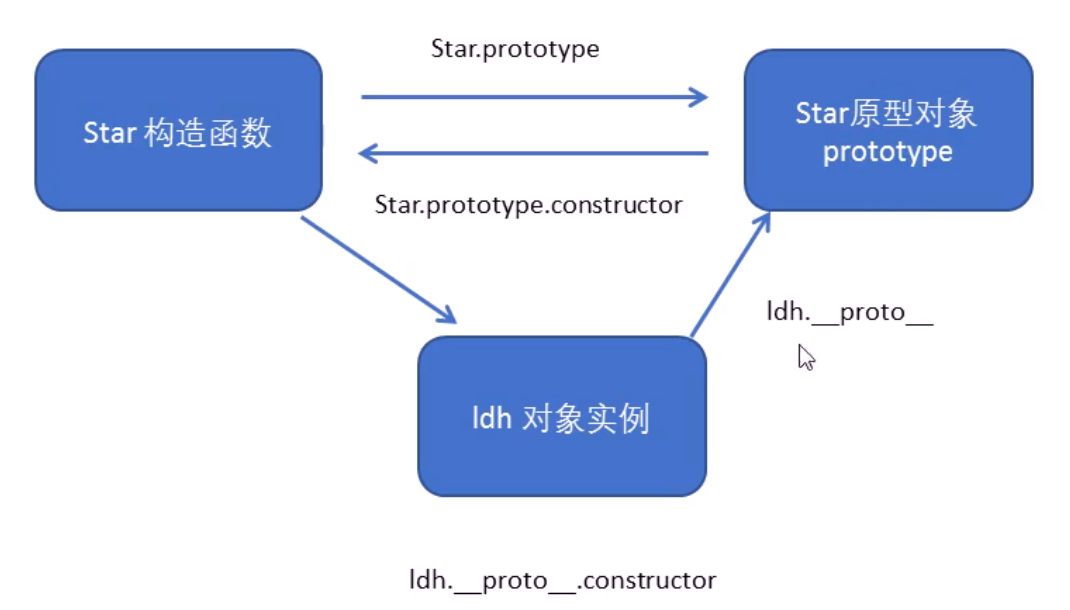
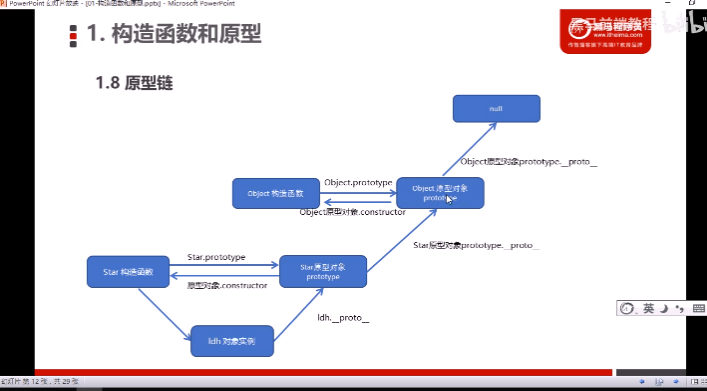
[36. 复制字符串的一种方式 13](#_Toc104931166)

[37.set 集合 不允许有重复元素 13](#_Toc104931167)

# ES6 面向对象

小知识

proto – 原型



## 1.静态成员直接在类名上建立 只能通过类名访问

class aut ;

aut.strtic=66;

## 2.prototype给原型加入新的属性或方法 原型中的一个属性

Star.prototype.sing = function() {

console.log('我会唱歌');

}

## 3.\_\_protot\_\_ 一样 在实例化对象上 由浏览器添加 指向原型

## 4. super 调用父类中的函数(子类想调用时使用)

super 调用父亲中的普通函数

//子类在调用自己的构造函数时，必须先调用父亲的构造函数

## 5.

## 6. class Son extends Father // extends 继承 //extends--扩展

// 父类有加法方法

class Father {

constructor(x, y) {

this.x = x;

this.y = y;

}

sum() {

console.log(this.x + this.y);

}

}

// 子类继承父类加法方法 同时 扩展减法方法

class Son extends Father {

constructor(x, y) {

// 利用super 调用父类的构造函数

// super 必须在子类this之前调用 必须先调用父类构造函数 才能调用自己的构造函数

super(x, y);

this.x = x;

this.y = y;

}

subtract() {

console.log(this.x - this.y);

}

}

var son = new Son(5, 3);

son.subtract();

son.sum();

console.log(son.window.x);

## 7. constructor 里面的this 指向的是 创建的实例对象

constructor(x, y) 构造函数

## 8.类里面有个constructor 函数,可以接受传递过来的参数,同时返回实例对象

class Star { //constructor 给类中变量赋值

constructor(uname, age) {

this.uname = uname;

this.age = age;

}

}

## 9.//如果我们修改了原来的原型对象,给原型对象赋值的是一个对象,则必须手动的指回原来的构造函数

constructor指回原来的构造函数

//constructor 指向构造函数 当以对象的形式修改原型对象时，

//会覆盖掉原构造函数 需要利用 constructor 指回原来的构造函数

//构造函数原型对象prototype 里面有constructor 指向构造函数本身

Star.prototype = {

constructor: Star,//此处修改

// Star: Star.prototype,

sing: function() {

console.log('我会唱歌');

}

}

使用继承来获取 别人的行为和属性 需要指回去this constructor: Son

        // 借用父构造函数继承属性

        // 1. 父构造函数

        function Father(*uname*, *age*) {

            // this 指向父构造函数的对象实例

*this*.uname = *uname*;

*this*.age = *age*;

        } // money--钱

        Father.prototype.money = function() {

            console.log(100000);

        };

        // 2 .子构造函数

        function Son(*uname*, *age*, *score*) {

            // this 指向子构造函数的对象实例

            Father.call(*this*, *uname*, *age*);

*this*.score = *score*;

        }

        // Son.prototype = Father.prototype;  这样直接赋值会有问题,如果修改了子原型对象,父原型对象也会跟着一起变化

        Son.prototype = **new** Father();

        // 如果利用对象的形式修改了原型对象,别忘了利用constructor 指回原来的构造函数

        Son.prototype.constructor = Son; //构造时还是采用Son的构造函数

        // 这个是子构造函数专门的方法

        Son.prototype.exam = function() {

            console.log('孩子要考试');

        }

        var son = **new** Son('刘德华', 18, 100);

        console.log(son);

        console.log(Father.prototype);

        console.log(Son.prototype.constructor);

        console.log(Son.prototype.money);

        console.log(Father.prototype.money);

## 10. fn.call(o, 1, 2); fn是函数名 1和2是传参 o更改指向的对象

// 借用父构造函数继承属性 // 1. 父构造函数

function Father(uname, age) {

// this 指向父构造函数的对象实例

this.uname = uname;

this.age = age;

console.log('!');

}

// 2 .子构造函数

function Son(uname, age, score) {

// this 指向子构造函数的对象实例.

//调用父亲构造函数 并将父亲的构造函数中this指向自己,

//在实例化时能挂载自己身上 不然对象将挂载到父亲实例上(父亲没有实例化)报错

Father.call(this, uname, age);

this.score = score;

}

var son = new Son('刘德华', 18, 100);

console.log(son);

## 11.创建一个类

class Star {

}

## 12.arr.forEach(function(value, index, array) { //循环

console.log('每个数组元素' + value);

console.log('每个数组元素的索引号' + index);

console.log('数组本身' + array);

sum += value;

})

## 13.filter--过滤器 筛选数组 返回符合要求的拷贝至newArr

var arr = [12, 66, 4, 88, 3, 7];

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

// return value >= 20;

return value % 2 === 0;

});//这里的return 不会终止迭代

## 14.fird--some 查找元素返回布尔值

arr1.some(function(value) {

return value == 'pink';

});//由于只需要查找一个 故终止迭代会结束查找 (提升效率)

## 15.trim--修剪 删除首尾空格 ，中间空格不会删除

## 16.Object.keys(对象) 会遍历对象中所有元素 并返回 成为一个数组

## 17. Object.defineProperty() //对象,修改或增添属性名,内容

Object.defineProperty(obj, 'address', {

value: '中国山东蓝翔技校xx单元',

// 如果只为false 不允许修改这个属性值 默认值也是false

writable: false,

// enumerable 如果值为false 则不允许遍历, 默认的值是 false

enumerable: false,

// configurable 如果为false 则不允许删除这个属性 不允许在修改第三个参数里面的特性 默认为false

configurable: false

});

## 18.// 2. 函数表达式 (匿名函数)

// 这个匿名函数是不可以调用的(会立即执行) fun(4,1)

var fun = function(a, b) {

console.log('天花乱坠');

console.log(a + b)

};

//这个只能匿名函数只调用一次 前面的符号告诉这是匿名函数 符号可以是+ - ~ {} [] 都可以

~ function(a, b) {

console.log(a + b + 'heihie')

}(4, 3)

## 19.this 指向调用者 普通函数直接调用 调用者是windows

## 20.改变函数内this 指向又call() apply() bind()

apply: 应用 运用的意思 使用时也会调用原来的函数

fun.apply(o, ['pink']); 传入需要是一个伪数组 所以有一个正括号

//可以应用在求最值上： arr是一个数组

Math.max.apply(Math,arr)=同等效力=Math.max.apply(null,arr)

//但是还有更简便的写法

Math.math(..arr)

bind: 绑定的意思 不会调用原函数，但会改变this指向 用在需要更改this但不需要立即调用处

使用方式与call以致

disabled--残废 (true禁用按钮属性)

var btns = document.querySelectorAll('button');

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

btns[i].onclick = function(e) {

console.log(e.target); //

this.disabled = true;

setTimeout(function() {

this.disabled = false; //(重要) bind会更改this从window指向btn

}.bind(this), 2000); //只会更改不会调用

}

}

## 21.'use strict' 开启严格模式

我们的变量名必须先声明再使用

num = 10;

console.log(num);

我们不能随意删除已经声明好的变量

var num=10;

delete num;

严格模式下全局作用域中函数中的 this 是 undefined而不是window。

function fn() {

console.log(this); // undefined。

}

fn();

严格模式下,如果 构造函数不加new调用, this 指向的是undefined 如果给他赋值则 会报错.

function Star() {

this.sex = '男';

}

Star();

定时器 this 还是指向 window

严格模式下函数里面的参数不允许有重名

## 22.闭包的使用 普通是给每一个元素个index属性赋值 再访问 ，下面写法未使用到index属性

lis 是一个装元素的数组

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

// 利用for循环创建了4个立即执行函数

// 立即执行函数也成为小闭包因为立即执行函数里面的任何一个函数都可以使用它的i这变量

(function(i) {//在点击时 将整个匿名函数看做一个整体 里面的点击还是在各自的匿名函数重运行 故i可以访问

// console.log(i);

lis[i].onclick = function() {

console.log(i);

}

})(i);

}

## 23.//个人理解 闭包:将一块函数封装起来，隔离所有，只留下外界的调用函数,里面的东西与外界无关

//闭包后 无法访问包内 var //与let局部变量

var car = (function() {

var start = 13; // 起步价 局部变量

var total = 0; // 总价 局部变量

return {

// 正常的总价

price: function(n) {

if (n <= 3) {

total = start;

} else {

total = start + (n - 3) \* 5

}

return total;

},

// 拥堵之后的费用

yd: function(flag) {

return flag ? total + 10 : total;

}

}

})();

console.log(car.price(5)); // 23

console.log(car.yd(true)); // 33

console.log(car.price(1)); // 13

console.log(car.yd(false)); // 13

## 24.可以将函数作为返回值

var object = {

name: "My Object",

getNameFunc: function() {

return function() {

return this.name;

};

}

};

console.log(object.getNameFunc()())//将函数作为返回值

var f = object.getNameFunc();

console.log(f());

## 25.深浅拷贝

• 空值(null)

• 未定义(undefined)

• 布尔值( boolean)

• 数字(number)

• 字符串(string)

• 符号(symbol，ES6 中新增)

复杂对象，数组(也是对象)

直接=赋值 会获得引用地址 (对于对象间)

=== 比较 会比较引用地址(对于对象间的比较)

(深拷贝需要重新new一块地址，消除浅拷贝,也就是引用地址)

function deepCopy(newobj, oldobj) {

for (var k in oldobj) {

// 判断我们的属性值属于那种数据类型

// 1. 获取属性值 oldobj[k]

var item = oldobj[k];

// 2. 判断这个值是否是数组

if (item instanceof Array) {

newobj[k] = [];

deepCopy(newobj[k], item)

} else if (item instanceof Object) {

// 3. 判断这个值是否是对象

newobj[k] = {};

deepCopy(newobj[k], item)

} else {

// 4. 属于简单数据类型

newobj[k] = item;

}

}

}

## 26.正则表达式 new RegExp 获取 字面量直接创建

test--测验

^ $ 前后边界符

[abc] 表示匹配里面任意字符即可

/^[abc]$/; // 三选一 这三个字母才返回 true

/^[a-z]$/; // 26个英文字母任何一个字母返回 true - 表示的是a 到z 的范围

/^[a-zA-Z0-9\_-]$/; // 26个英文字母(大写和小写都可以)任何一个字母返回 true

/^[^a-zA-Z0-9\_-]$/ //如果中括号里面有^ 表示取反的意思 千万和 我们边界符 ^ 别混淆

/^a\*$/ //\* 相当于 >= 0 可以出现0次或者很多次

/^a+$/ //+ 相当于 >= 1 可以出现1次或者很多次

/^a?$/ // ? 相当于 1 || 0

/^a{3}$/ //{3 } 就是重复3次

/^a{3,}$/ //{3, } 大于等于3

/^a{3,6}$/ // {3,16} 大于等于3 并且 小于等于6

/^[a-zA-Z0-9\_-]{6,16}$/ 用户名验证

/^(abc){3}$/; // 它是让abcc重复三次 'abcabcabc'true

/^\d{3,4}-\d{7,8}$/ \d 表示匹配所有数字 \D非数字

\b 匹配一个单词边界 \B 匹配非单词边界

/激情|gay/g | 是或者 匹配左边或者右边

{x} 重复前面的字符, 精确 x 次数.

{x,} 重复前面的字符, 至少 x 次数.

{0,x} 重复前面的字符, 最多 x 次数.

{x, y} 重复前面的字符, 在 x 与 y 之间的次数, x、y包括在内.

. 表示 匹配除换行符 \n 之外的任何单字符

\* 重复前面的字符, 0 或更多次数. 等价于 {0,}

+ 重复前面的字符, 1 或更多次数. 等价于 {1,}

? 可能会或可能不会出现的前一字符. 等价于 {0, 1}

? (在一个重复字符之后) 查找最少的匹配而非最多的 贪婪匹配 <img src=``test.jpg` width=`60px` height=`80px`/> 时，将会返回 src=``。

替换：text.value.replace(/激情|gay/g, '\*\*'); 执行多次 /g 表示会全局匹配

## 27. let const 声明常量具有块级作用域 即 大括号内才有作用

const 声明即需要赋初始值 声明后不可更改

//特别的是这个问题

var num = 10

var op = 90;

if (true) {

console.log(op);

let num = 20;

console.log(num);

}

## 28.数据解构 快速赋值

let ary = [1,2,3];

let [a, b, c, d, e] = ary;

console.log(a)

console.log(b)

console.log(c)

console.log(d)

console.log(e)

29.对象结构

let person = {name: 'lisi', age: 30, sex: '男'};

//普通匹配

let { name, age, sex } = person;

//带别名的匹配

//myName 是name的别名 //会匹配是否有name属性名 有则将内容给到myName

let {name: myName } = person;

console.log(name)

## 30. ... 剩余参数 最后面是定义一个数组 存储剩余的数据(可以是如下解构，也可以是函数传参)

let ary1 = ['张三', '李四', '王五'];

let [s1, s3, s4, ...s2] = ary1;

console.log(s1)

console.log(s2)

## 31. ... // 利用扩展运算符将伪数组转换为真正的数组

var oDivs = document.getElementsByTagName('div');

console.log(oDivs)

var ary = [...oDivs];

ary.push('a');

console.log(ary);

## 32.面试题

//在调用下面函数 会找i 当前作用域找不到则往上级作用域找

//每一个let i 都会形成一个作用域 故调用各自的

//当 是var i 时 ，i为全局变量 大家都会相同

//因为变量提升的缘故 在这里会有些区别

let arr = [];

for (let i = 0; i < 2; i++) {

arr[i] = function() {

console.log(i);

}

}

arr[0]();

arr[1]();

## 33. from 将伪数组(可迭代的对象)转化为正真的数组

var ary = Array.from(arrayLike);

var ary = Array.from(arrayLike, item => {

return item \* 2;

})

/\*\*\*\*迭代map数组\*\*\*\*\*\*/

const map = new Map([

[1, 2],

[2, 4],

[4, 8]

]);

console.log(Array.from(map));

// [[1, 2], [2, 4], [4, 8]]

const mapper = new Map([

['1', 'a'],

['2', 'b']

]);

console.log(Array.from(mapper.values()))

## 33.find(函数(传参--key)) 查找 返回查找到的对象或值

数组.findIndex(函数(传参--key)) 查找到返回下标

includes()方法用来判断一个数组是否包含一个指定的值，根据情况，

//cludes---包括 如果包含则返回true ，否则返回false

## 34. ${fn()} 占位符

## 35. 字符串的查询方法 string str="";

start---开始 width---具有

str.startsWith(查询字符串 , 下标开始查找(可选))

str.endsWith 从后面

## 36. 复制字符串的一种方式

repeat() 方法字符串复制指定次数。

## 37.set 集合 不允许有重复元素

add() 添加元素

delete() 删除元素

has() 查找元素返回布尔值

cleas() 清空容器

forEach() 遍历容器