2019/4/23

- setInterval(function,1000,函数的参数)

undefined出现的情况
1.对象没有属性的时候
2.函数没有返回值
3.形参没有实参
4.变量没有赋值
5.简单类型的自定义属性

**只有在引用类型下才能添加属性或者方法

```
let str = '';
str.num = 0;
str.num++;
console.log(str.num);
undefined
这种是不能添加属性或方法
```

包装对象:
当简单类型去使用某个属性或者方法的时候,
内部会偷偷地转成对象(new 内置类)把属性

或者方法提供使用者，然后再悄悄地销毁
这个过程就叫包装对象。

```
let str = ‘1234567’;
let str2 = new String(‘1234567’);
```

自身属性

for in不但会枚举本对象，还会枚举原型，此时就会多出来一些莫名其妙的东西
但是我们不想要那么不是对象上的东西.

obj.hasOwnProperty(‘属性名’)
查看某个属性是不是对象自身的

返回值:
布尔值，是就为true，不是就为false

```
Object.prototype.say = function(){
        console.log('咩');
    }
    let obj = {
        name:'小马',
        age:16,
        job:'前端'
    }
    for(let attr in obj){
        if(obj.hasOwnProperty(attr)){
            console.log(attr);//obj[attr]
        }
    }
    console.dir(obj);
```

修改THIS

一个函数，天生就自带一些属性和方法
其中有:
apply()

call()
bind()
他们都能改变this指向

- call:
  有无数个参数
  第一个参数:
  改变this指向（写啥是啥）
  null和undefined为window
  第二个参数之后:
  就是实参
- apply:
  有2个参数
  第一个参数:
  改变this指向（写啥是啥）
  null和undefined为window
  第二个参数:
  数组[1,2,3]
  数组中放参数
- bind:
  有无数个参数
  第一个参数:
  改变this指向（写啥是啥）
  null和undefined为window
  第二个参数之后:
  就是实参
  使用bind不能立马执行函数，会返回一个新函数，这个函数的this是改变了的，得执行这个新函数才能输出代码。

JSON方法

我不会

```
JSON
```

```
        '{
            name:'小明',
            age:18
        }'
        '[]'

    对象:
        {
            name:'小明',
            age:18
        }

    xml:
        <person>
            <name>小明</name>
            <age>18</age>
        </person>

    JSON -> [] | {}

        JSON.parse()
            能够把json转成对象或者数组

            json必须是一个标准格式的json,不然转不出来

            json中不能放函数、不能为undefined

            '{"name":12,"nn":"ds"}'

        JSON.stringify()
            把对象或者数组转成json
            对象中不能放函数、不能为undefined


        低版本可以使用json2.js

        eval 能够把字符串尽量转成js能执行的代码。

        new Function('','console.log()')
```

、

赋值赋址

因为赋值的时候第一层为简单类型，简单类型的
赋值就是赋值，如果第一层有引用类型，
那么引用类型的赋值为赋址。

如果是引用类型，就把引用类型中的值取出来
如果值还是为引用类型，那么继续循环取值
直到全部都为简单类型为止

deepclone -> 深度克隆（深拷贝）

```
    let arr = [1, 2, 3, 4, [5, { ary: [{ name: '小强' }]
}]];
    Object.prototype.xxoo = '哈哈';
    function deepClone(obj) {
        let o = obj.push ? [] : {};
        for (let attr in obj) {
            if (obj.hasOwnProperty(attr)) {
                if (typeof obj[attr] === 'object') {
                    o[attr] = deepClone(obj[attr])
                } else {
                    o[attr] = obj[attr];
                }
            }
        }
        return o;
    }
    let arr2 = deepClone(arr);
    arr2[4][1].ary[0].name = '小弓';
    arr2[4].push(6);
    console.log(arr2)
    console.log(arr)
```

思路：
//先声明一个数组，去存克隆出来的内容
//判断obj是否为数组，是数组就o就为[],否则为{}
// for(let i=0;i<arr.length;i++){
//判断对象中的某个值是否为引用类型
//如果是，就继续调用deepClone把引用值传到函数中

```
//如果是简单类型就直接赋值
```

继承

子类继承了父类的一些特征，还有自己的一套自己的特征。

- 为什么继承：？
  为了代码能够更好的复用，组合起来生成一个新的类别

类式继承

> 能够把父类看作一个函数，调用父类并且通过call去改变this指向**.call(zhis)**，把指向改为子类。

原型继承

```

```

寄生式组合继承

```

```

对象继承

```

```

方法继承

- 扩展式继承

  > 子类原型 = {...父类原型}
  > 不能进行深度拷贝

- deep clone 深度克隆(深拷贝)

for in deep clone（父类原型）