4-24

原型继承

> 通过把父类的原型地址赋值给一个中间函数的原型，然后再将这个中间函数的实例化对象赋值给子类的原型；这样就实现了子类的原型属性也继承了父类的原型属性，并且改变子类属性的同时还不影响父类的属性；

```
<script>
    function Fruit(name){
        this.name = name;
        console.log(1);
    }
    Fruit.prototype.chi = function(){
        console.log(this.name + '能吃');
    }
    Fruit.prototype.卖钱 = function(){
        console.log('能卖钱');
    }


    let apple = new Fruit('苹果');

    //椰汁

    function Yezhi(name){
        Fruit.call(this,name);
        this.baoz = '包装加工';
    }


    function p(){
        function Ph(){}
        Ph.prototype = Fruit.prototype;
        return new Ph;
    }
    Yezhi.prototype = p();
    // Yezhi.prototype = new Fruit;

    /*
        实例的方法:
```

```
            this.xx
            p = new PH
            p.xx

            let p2 = new PH
        */
        // Yezhi.prototype = new Ph;
        /*
            yz.chi -> yz.__proto__ -> Yezhi.prototype
            Yezhi.prototype -> p -> p.__proto__
            Ph.prototype = Fruit.prototype
        */
        Yezhi.prototype.he = function(){
            console.log('每天一杯白白壮壮');
        }
        Yezhi.prototype.chi = function(){
            console.log('吃椰肉');
        }
        let yz = new Yezhi('椰子');
        yz.he();
        yz.卖钱();

        yz.chi();
        apple.chi();
    </script>
```

寄生式继承

`Object.create({})`必须传入一个对象,返回值为一个新的对象，这个对象的原型链指向传入的参数(相当于新建一个对象)

```
    function Person(name,age){
            this.name = name;
            this.age = age;
        }

        Person.prototype.say = function(){
            alert(this.name+'会说话');
        }


        function Coder(name,age,job) {
            Person.call(this,name,age); //属性继承
            this.job = job;
        }
```

```
    //方法继承

    function create(obj){
        function ph(){}
        let o = new ph;
        o.__proto__ = obj;
        return o;
    }


    Coder.prototype = create(Person.prototype);//Object.create(Pers
on.prototype);
    //手动修正指向
    Coder.prototype.constructor = Coder;

    // console.log(Object.create(Person.prototype))

    Coder.prototype.say = function(){
        alert('哈哈');
    }


    /*
        p.say -> Coder.prototype -> {}

        {}.__proto__ =
    */

    let o = Object.create({name:'haha'});
    let p = new Coder('xx',20);

    // console.log(p.constructor);
    // console.log(p  instanceof  Coder);

    // p.say();
    // new Person('oo',28).say();
```

Class类（常用）

语法：`class 类名 { }`使用`constructor`去接收参数`constructor(a,b){ }`
方法直接写在class'类下

```
class Fn {
        constructor(name,age){
```

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

        //方法:  方法名 () {}
        say(){
            console.log(1);
        }
    }
    let f = new Fn('刘泉',18);
    f.say();
    console.log(f);
```

Class继承法（常用）

语法: `class Cat//子类 extends Animal//父类`
`super()`用来父类接受参数

```
class Cat extends Animal {
        constructor() {
            /*
                super下面才能写this，不然就报错
            */
        //    console.log(arguments);
            super(...arguments); //super括号中传入父类使用的参数
            this.jiao = '喵喵';
            this.ming = '九条命';

        }
        skill(){
            console.log('抓老鼠');
        }

        chi(){
            console.log('吃鱼');
        }
    }

    let tom = new Cat('汤姆');
    tom.chi();
    new Animal().chi()
    console.log(tom);
```

三点（...）剩余运算符/扩展运算符

扩展运算符

```
...arr[];扩展运算符，可以把arr数组内的数据扩展出来，（相当于去掉中括号）
[1,2,...3.4.5],可以把数组中1,2，后边的
```

剩余运算符

```
function aa(a,...arw){//除了第一个参数外剩余的所有参数自动存放在arw的数组中
        console.log(arw)//输出arw数组显示['b''c']
        alert(a)//弹出a显示‘a’
    }
    aa('a','b','c')
```