原型

2019/04/23 10:58

father.__proto__ 查看对象原型

father.constructor 查看对象的构造函数

原型是function对象的一个属性，他定义了构造函数制造出的对象的公共祖先。通过该构造函数产生的对象，可以继承原型的属性和方法。原型也是对象。

```
//Person.prototype     --原型
//构造函数出生之前就已经被定义好，系统自带
//Person.prototype = {}                    //祖先
//可以赋给原型属性，对象person就会有相应属性
//Person.prototype.name = "hehe";
function Person(){                         //父亲
    //var this = {
    //  __proto__ : Person.prototype
    //}

}
var person = new Person();                 //儿子
var person1 = new Person();                //另一个儿子
//拥有同样的祖先所以他也能接收到祖先的属性
```

特点：

1. 原型和构造函数拥有相同的属性则与构造函数属性相一致
2. 可以将公有的固定的属性写在原型中，防止每创建一个对象就执行一次代码，降低效率

eg:

```
Car.prototpye.carname = "BMW";
Car.prototpye.height = 1400;
Car.prototpye.long = 4800;
//简化
Car.prototype = {
    carname : "BMW",
    height : 1400,
    long : 4800,                  //注意对象中的语句结尾都是逗号！！！
 }
funciton Car (color,owner){
    this.color = color;
    this.owner = owner;
}
var car1 = Car('red','Ji');
var car2 = Car('green','Deng');
```

原型中有的属性而构造函数中没有，可以访问，但是修改并不会改变原型中的值而会在构造函数中添加一个新属性，除非调用Person.prototype.属性名，来改变

构造函数可以删除不存在的属性，返回结果为TRUE。

原型并不是完全空对象

constructor构造器，　　可以手动更改，可以找出对象的所属类，

```
function Car (){

}
var car = Car();
car.consturctor
//返回
//function Car(){    }
```

构造函数的原型也可以替换，this中的__proto__属性中保存着构造函数的prototype

```
Person.prototype.name = 'abc';
function Person    () {
    var this = {
        __proto__ : Person.prototype
    }
}
var obj = {
    name : 'sunny'
}
var person = new Person();
person.name    //sunny
```

eg1

```
Person.prototype.name = 'sunny';
function Person() {
    //var this = {
        //__proto__ : Person.prototype
    //}
    return this
}
Person.prototype = {
    name : 'cherry'
}
//写在这里结果是cherry因为执行顺序，new之后才var this return this，
//而此时prototype已经改变
var person = new Person();
Person.prototype = {
    name : 'cherry'
}
```

```
//相当于下面这个部分person等于Person.prototype
var obj = {name : "a"};
var obj1 = obj;
obj = {name : "b"};
//结果Person.name为sunny
```

对比

eg2

```
Person.prototype.name = "sunny";
function Person(){

}
Person.prototype.name = 'cherry'
var person = new Person();
Person.prototype.name = 'cherry';
//无论是5还是7结果Person.name都为cherry
//而这个是因为后执行的覆盖先执行的
```

构造函数原型也有__proto__属性说明原型也有原型！！

所有对象的最终原型都是Object.prototype

谁调用this方法this就指向谁！

```
Person.prototype = {
    name : "a",
    sayName : function () {
        console.log(this.name);
    }
}
function (){
    this.name = "b";
}
var person = new Person();
person.sayName();
//结果是b
person.prototype.sayName();
//a
```

指定原型创建对象

```
var obj = {name : "sunny",age : 123};
var obj1 = Object.create(obj);
//obj1的原型就成了obj
//原始值不可放入
```

绝大多数对象的最终都会继承自Object.prototype

因为Object.create()括号内部也可以填null，即没有原型，即便人为添加__proto__属性，也不会有继承特性

只有underfined null 没有原型且不是对象

包装类

123.toString();　　有原型且有toString方法，但是这么调用系统首先会定义其为浮点型

num = 123　num.toString();　　作用是将其变成字符串

document.write();实际上是先调用toString()方法

```
var obj = Object.create(null);
obj.toString = function (){
   return "laodengshentihao"
}
document.write(obj)
```

call /apply

作用：改变this指向

```
function Person(name,age){
   //var this = obj;
   this.name = name;
   this.age = age;
}
var person =new Person();
var obj = {
}
Person.call(obj,'cheng',300); //this是对象

Person(); 等同于Person.call();
```

通过call来写重复的部分

我写的

```
function Person(name,age,sex) {
       this.name = name;
       this.age = age;
       this.sex = sex;
   }
   function Student (tel,grade){
       this.tel = tel;
       this.grade = grade;
   }
   var student = new Student(8843333,3);
   // var person = new Person();

   Person.call(student,'ji',330,'male');
```

老师写的

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

```
    this.sex = sex;
}
function Student (name,age,sex,tel,grade){
    Person.call(this,name,age,sex);
    this.tel = tel;
    this.grade = grade;
}
var student =new Student('sunny',123,'male',139,2018);
```

apply

cal和apply没有本质区别 区别是传参不同

call需要把实参按照形参的个数传进去

apply需要传一个arguments列表