JS的原型和原型链

github.com/tsrot 原型 原型链

地址：http://blog.xieliqun.com/2016/08/20/prototype/

> 在JavaScript的使用过程中，我们经常会遇到prototype，可能了解一点，它是一个对象的原型，用来做原型继承的。这样去理解就有点不准确了，今天就让我们深入的去了解它。

了解一些概念

在读这篇文章之前，你应该去在自己的脑海问几个问题：
1、什么是原型？
2、什么事原型链？
3、prototype与__proto__有什么不同，有什么联系？
4、constructor与面两个有什么联系，怎么用？

如果你把上面这四个问题都解决了，那你就真正了解了JS的原型和原型链。接下来，咱们一个一个问题去解决。

什么是原型

JavaScript 中，万物皆对象！但对象也是有区别的。分为普通对象和函数对象，Object ，Function是JS自带的函数对象。每个对象都有原型（null和undefined除外），你可以把它理解为对象的默认属性和方法。

你可以把下面的代码在浏览器打印出来看一下。

```
console.log(Object.prototype); //Object{}

var o = new Object();
console.log(o.prototype);  //undefined

console.log(Array.prototype); //[Symbol(Symbol.unscopables): Object]

console.log(Function.prototype); //function(){}

function hello(){
    console.log("hello");
}
hello.prototype = "hello world";
console.log(hello.prototype); //hello world
```

Object：Object是一个函数对象，Object的原型就是一个Object对象，它里面存在着一些对象的方法和属性，例如最常见的toString方法。

新建对象：用new Object或者{}建的对象是普通对象，它没有prototype属性，只有__proto__属性，它指向Object.prototype。

Array：Array也是一个函数对象，它的原型就是Array.prototype，它里面存在着一些数组的方法和属性，例如常见的push，pop等方法。

Function：Function也是一个函数对象，但它有点特殊，它的原型就是一个function空函数。

自定义函数：它的原型就是你给它指定的那个东西。如果你不指定，那它的原型就是一个Object.prototype。

什么是原型链

在 javaScript 中，每个对象都有一个指向它的原型（prototype）对象的内部链接。这个原型对象又有自己的原型，直到某个对象的原型为 null 为止（也就是不再有原型指向），组成这条链的最后一环。这种一级一级的链结构就称为原型链（prototype chain）。

JavaScript 对象是动态的属性“包”（指其自己的属性）。JavaScript 对象有一个指向一个原型对象的链。当试图访问一个对象的属性时，它不仅仅在该对象上搜寻，还会搜寻该对象的原型，以及该对象的原型的原型，依此层层向上搜索，直到找到一个名字匹配的属性或到达原型链的末尾。

```
var o = {
    a:1,
    b:2
};
console.log(o.toString()); //不报错，o上没有toString方法，但是Object上有

console.log(o.push("c")); //报错，o上没有这个方法，Object上也没有这个方法。

console.log(o.a); //1
console.log(o.c); //undefined
```

当你用new Object或者直接定义一个对象时，它的原型链就是：
o ==》 Object.prototype ==》 null
但你访问o上没有的属性或方法时，JS会往Object.prototype上寻找该属性和方法。如果有则直接返回，如果没有，方法则报错，这个方法未定义，属性则返回undefined。

```
function Person(name){
    this.name = name;
}
Person.prototype = {age:24};

var tsrot = new Person("tsrot");

console.log(tsrot.name); //tsrot
console.log(tsrot.age);  //24
console.log(tsrot.toString()); //[object Object]
```

当你用构造函数（构造函数我们一般首字母大写）建立一个对象时，它的原型链就是：
tsrot ==》 Person.prototype ==》 Object.prototype ==》 null
如果没有定义Person.prototype这一环，则直接跳到下一环。

来点更复杂的。

```
function Parent(){
    this.name = "i am parent";
}
Parent.prototype = {age:24};

function Child(){
    this.name = "i am child";
}

Child.prototype = Object.create(Parent.prototype); //让Child的原型指向Paren
t的原型
Child.prototype.constructor = Child; //把child的构造函数指向回来，否则它将指向P
arent。虽然在这没什么影响，但要养成代码的严谨性

var child = new Child();

console.log(child.name); //i am child
console.log(child.age);   //24
console.log(child.toString()); //[object Object]
```

当你需要父类的属性和方法时，你可以把它的原型指向父类的原型。此时的原型链就是：
child ==》 Parent.prototype ==》 Object.prototype ==》 null

```
var arr = [1,2,3];

console.log(arr); //[1,2,3]
arr.push(4);
console.log(arr); //[1,2,3,4]
```

数组也是一个对象，不过它是由Array构造函数new而来的，所以它的原型链就是：
arr ==》 Array.prototype ==》 Object.prototype ==》 null

```
var fun = function(){
    var hello = "i am function";
}

console.log(fun.name); //fun
```

fun是一个函数对象，它是由Function构造函数new而来的，所以它的原型链就是：
fun ==》 Function.prototype ==》 Object.prototype ==》 null
fun它没有name属性，但是Function它有，所以这个name就是Function原型上的。

prototype与__proto__

在Javascript中，每个函数都有一个原型属性prototype指向自身的原型，而由这个函数创建的对象也有一个__proto__属性指向这个原型，而函数的原型是一个对象（函数点prototype也是一个普通对象，Function.prototype除外,它是函数对象，但它很特殊，他没有prototype属性），所以这个对象也会有一个__proto__指向自己的原型，这样逐层深入直到Object对象的原型，这样就形成了原型链。普通对象没有prototype，但有__proto__属性。

```
function f1(){};

console.log(f1.prototype) //Object{}

console.log(typeof f1.prototype) //Object

console.log(typeof Function.prototype) // Function，这个特殊

console.log(typeof Object.prototype) //Object

console.log(typeof Function.prototype.prototype) //undefined
```

JS在创建对象（不论是普通对象还是函数对象）的时候，都有一个叫做__proto__的内置属性，用于指向创建它的函数对象的原型对象prototype。

普通对象的__proto__

```
var o = {name:"tsrot"};

console.log(o.__proto__); //Object{}
console.log(o.prototype); //undefined
console.log(o.__proto__ === Object.prototype);  //true
```

构造对象的__proto__

```
function Parent(){
    this.name = "i am parent";
}
Parent.prototype = {age:24};

function Child(){
    this.name = "i am child";
}

Child.prototype = Object.create(Parent.prototype);
Child.prototype.constructor = Child;

var child = new Child();

console.log(child.__proto__); //Object{}

console.log(Child.prototype); //Object{}

console.log(child.__proto__ === Child.prototype); //true

console.log(Parent.prototype.__proto__ === Object.prototype); //true
```

数组的__proto__

```
var arr = [1,2,3];

console.log(arr.__proto__);  //[Symbol(Symbol.unscopables): Object]

console.log(Array.prototype); //[Symbol(Symbol.unscopables): Object]

console.log(arr.__proto__ === Array.prototype); //true
```

函数的__proto__

```
var fun = function(){
    var hello = "i am function"
}

fun.prototype = {name:"tsrot"};

console.log(fun.prototype); //Object {name: "tsrot"}

console.log(fun.__proto__); //function(){}

console.log(fun.prototype === fun.__proto__); //false

console.log(fun.__proto__ === Function.prototype); //true
```

constructor属性

原型对象prototype中都有个预定义的constructor属性，用来引用它的函数对象。这是一种循环引用：

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

console.log(Person.prototype.constructor === Person); //true

console.log(Function.prototype.constructor === Function); //true

console.log(Object.prototype.constructor === Object); //true
```

用构造函数创建的对象，它的constructor属性就是它的构造函数。

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

var person = new Person();

console.log(person.constructor === Person); //true
```

参考文章

1、JavaScript Prototype Chains
2、Understanding JavaScript Prototypes
3、继承与原型链