JAVASCRIPT的OO及继承实现

—— by liuxin.rkl 2014-09-05

什么是对象?

An object is a collection of properties and has a single prototype object. The prototype may be either an object or the null value.

```
var foo = {
  x: 10,
  y: 20
};
```

什么是原型链?

A prototype chain is a finite chain of objects which is used to implement inheritance and shared properties.

特殊属性__PROTO__

- 标准：ECMAScript5 [[Prototype]]
- from [FireFox, V8, Safari] to ECMASctript6
- Object.getPrototypeOf()
- Object.create()

```
//创建及读取
var myProto = {};

var obj = Object.create(myProto);

Object.getPrototypeOf(obj) === myProto
```

```
//指定对象原型
var obj = {
  __proto__: myProto,
  foo: 123,
  bar: "abc"
};
```

原型`prototype`和原型链`__proto__`的区别?

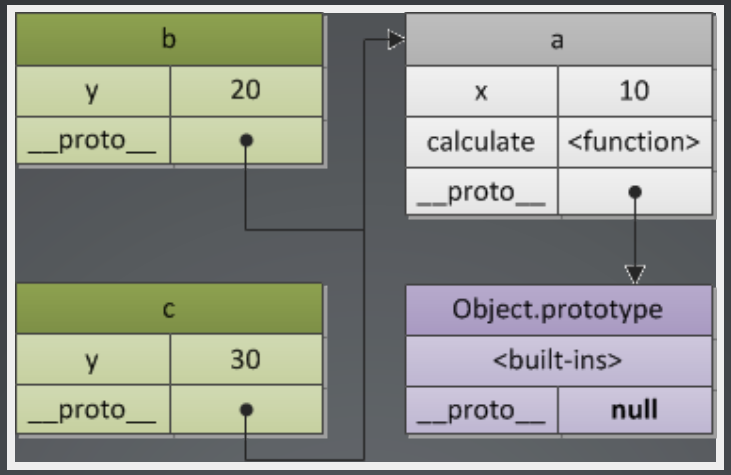
原型是包含若干属性的对象，原型链是对原型对象的引用，是javascript实现继承的关键

```
var a = {
  x: 10,
  calculate: function (z) {
    return this.x + this.y + z
  }
};

var b = {
  y: 20,
  __proto__: a
};

var c = {
  y: 30,
  __proto__: a
};

// call the inherited method
```


b
y
20
__proto__
a
x
10
calculate
<function>
__proto__
c
y
30
__proto__
Object.prototype
<built-ins>
__proto__
null

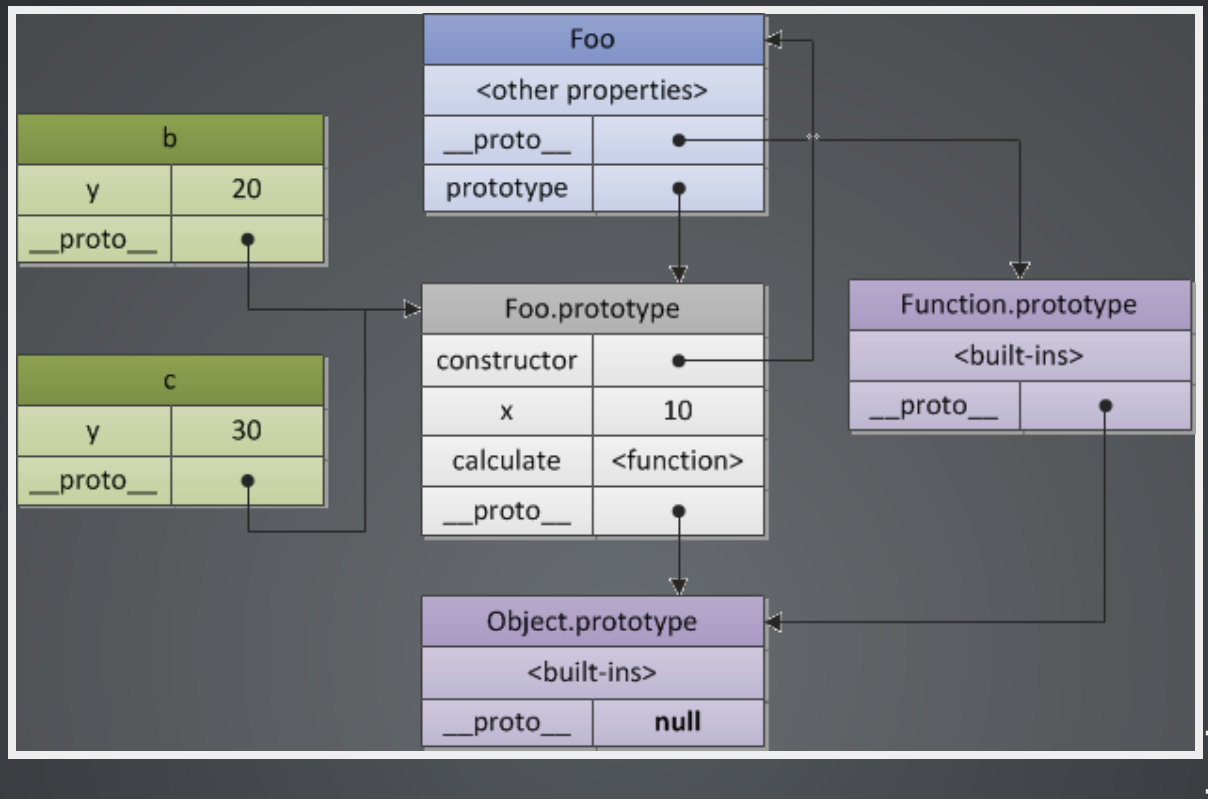
JAVASCRIPT的OO实现

通过函数来实现类的定义，函数被定义出来后，一定包含prototype属性，是这个函数的原型对象（详细），这是javascript赋予Function可以被作为构造器使用的关键。

```
// Foo class
function Foo(y) {
  this.y = y;
}
Foo.prototype.x = 10;
Foo.prototype.calculate = function (z) {
  return this.x + this.y + z;
}

var b = new Foo(20);
var c = new Foo(30);

// 结论
console.log(b.__proto__ === Foo.prototype);
console.log(c.__proto__ === Foo.prototype);
console.log(b.constructor === Foo);
console.log(b.hasOwnProperty('constructor') === false);
console.log(b.__proto__.hasOwnProperty('constructor') === true);
```


Foo
<other properties>
__proto__
prototype
b
y
20
__proto__
Foo.prototype
constructor
x
10
calculate
<function>
__proto__
Function.prototype
<built-ins>
__proto__
c
y
30
__proto__
Object.prototype
<built-ins>
__proto__
null

OO及继承实现

- class 类定义
- extends 可继承

class 定义

- 公共成员
- 私有成员
- 特权成员

```
function Foo(info) {
  var city = 'hangzhou';
  this.info = info + ' from ' + city;
  this.sayCity = function () {
    console.log(city);
  };
}
Foo.prototype.sayInfo = function () {
  console.log(this.info);
}
```

extends继承

```
function Class(o) {
  // Convert existed function to Class
  if (typeof o === 'function') {
    o.extends = Class.extends;
    return o;
  }
}

Class.extends = function (properties) {...}
module.exports = Class;
```

- 子类构造器函数
- 继承父类static properties
- 继承父类原型链
- 扩展自身原型链

```
var parent = this;

// 子类构造器函数
function SubClass() {
  parent.apply(this, arguments);
  if (this.constructor === SubClass && this.__constructor) {
    this.__constructor.apply(this, arguments);
  }
}

// 子类继承父类static properties
mix(SubClass, parent);

// 子类继承父类原型链
var proto = createProto(parent.prototype);
proto.constructor = SubClass;
SubClass.prototype = proto;
SubClass.__super__ = parent.prototype;
```

createProto 性能对比

Testing in Chrome 37.0.2062.94 on OS X 10.9.4

Test		Ops/sec
new ctor	`getProto(Parent.prototype, Parent)`	435,955 ±5.55% 86% slower
object.create	`getProto2(Parent.prototype, Parent)`	380,855 ±11.92% 88% slower
__proto__	`getProto3(Parent.prototype, Parent)`	2,931,959 ±2.57% fastest

coffee兼容

- `__super__`父类原型对象
- Class(coffee class)

示例

```
// 定义新的class
var Class = require('class.js');

var Animal = Class.extends({
  __constructor: function () {},
  move: function () {}
});

module.exports = Animal;
```

```
// 已有的function类转化为class，可被继承
var Class = require('class.js');

function Event () {...}

Event.prototype.on = function () {};

module.exports = Class(Event);
```

ES6 CLASS

Notice: in ES6 the concept of a “class” is standardized, and is implemented as exactly a syntactic sugar on top of the constructor functions as described above.

```js
// ES6
class Foo {
  constructor(name) {
    this._name = name;
  }
  getName() {
    return this._name;
  }
}
class Bar extends Foo {
  getName() {
    return super.getName() + ' Doe';
  }
}
var bar = new Bar('John');
console.log(bar.getName()); // John Doe
```

参考

- javascript core
- difference between `__proto__` and prototype
- Function MDN
- Function.prototype MDN
- `__proto__`
- Private Members in JavaScript

谢谢大家