面向对象

2016年6月21日 星期二　　下午10:20

数据属性

Configurable：能否修改属性的特性，能否通过delete删除属性从而重新定义属性（true）
Enumerable：能否通过 for-in 循环返回属性（true）
Writable：能否修改属性的值（true）
Value：包含这个属性的数据值（undefined）

```
Object.defineProperty(person, "name", {
        configurable: false, //改不回来了，不可配置
        value: "Nicholas"
});
```

访问器属性

Configurable
Enumerable
Get：读取属性时调用（undefined）
Set：写入属性时调用（undefined）

```
var book = { _year: 2004, edition: 1 }; //_year表示只能通过对象方法访问的属性
Object.defineProperty(book, "year", { //setter、getter最好同时给出
        get: function(){ return this._year; },
        set: function(newValue){
                if (newValue > 2004) {
                        this._year = newValue;
                        this.edition += newValue - 2004;
                }
        }
}); //旧方法：book.__defineGetter__("year", function(){ ...};
Object.defineProperties(book, {...});
Object.getOwnPropertyDescriptor(book, "_year");
```

工厂模式：解决了创建多个相似对象的问题，但不能识别对象类型

构造函数模式：通过执行环境（隐式传入this）调用函数创建，但每个方法都要在每个实例上重

new Person(...); // 当作构造函数使用

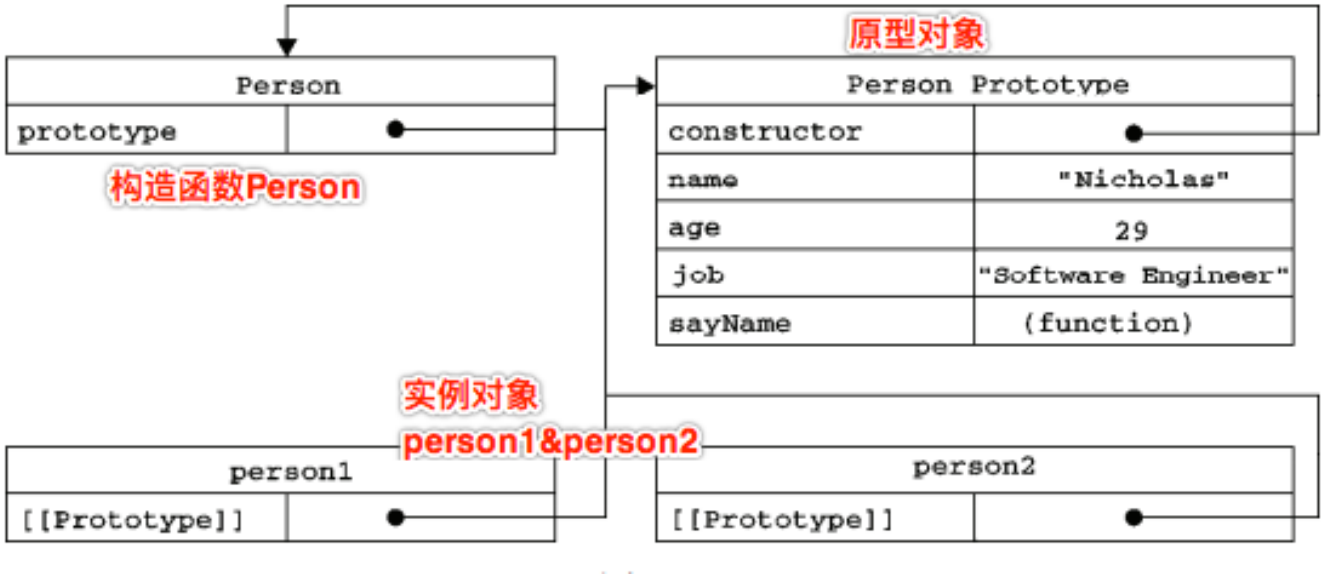
Person(...); // 作为普通函数调用，添加到window

原型模式：prototype指针指向函数的原型对象，多个对象实例共享原型所保存的属性和方法

isPrototypeOf()

Object.getPrototypeOf()

查找对象属性和方法：现在当前实例中查找，找不到再去原型中找



hasOwnProperty() 只在属性存在于实例中（非原型中）时才返回true

Object.keys() 返回一个包含所有可枚举属性的字符串数组

Object.getOwnPropertyNames() 可得到所有实例属性，无论它是否可枚举

原型的动态性

为原型添加属性和方法，并且修改能够立即在所有对象实例中反映出来

重写原型对象切断了现有原型与任何之前已经存在的对象实例之间的联系；它们引用的仍然是最

原生对象的原型：String.prototype.startsWith = function (text) { ...} 在原有原型上扩展，不

组合使用构造函数模式和原型模式（常用）

```
// 构造函数模式用于定义实例属性
function Person(name, age, job){
        // 继承了 SuperType，同时还传递了参数
```

重新创建一遍

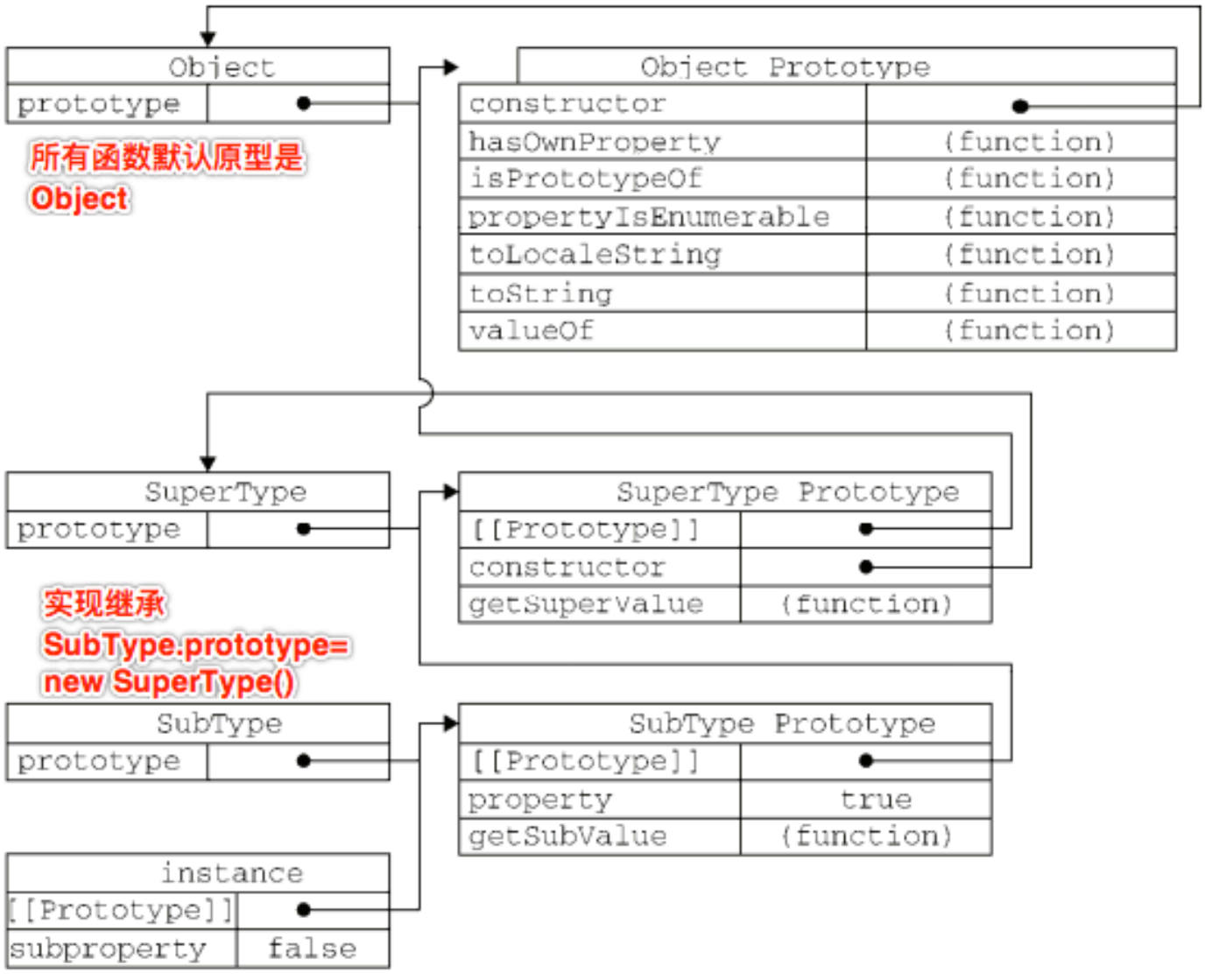
最初的原型

不会重写

```
        SuperType.call(this, name);
        this.age = age;
        this.friends = ["Shelby", "Court"];
}
// 原型模式用于定义方法和共享的属性
Person.prototype = { // 使用字面量构造原型，会重写原型
        constructor : Person,
        sayName : function(){ alert(this.name); }
}
```

原型链

构造函数、原型和实例的关系：每个构造函数都有一个原型对象，原型对象都包含一个指向构造
例都包含一个指向原型对象的内部指针。



造函数的指针，而实

原型式继承 Object.create(用作新对象原型的对象，为新对象定义额外属性)

闭包：在函数内部定义了其他函数时，闭包有权访问包含函数内部的所有变量

1、闭包的作用域链包含着它自己的作用域、包含函数的作用域和全局作用域

2、当函数返回了一个闭包时，这个函数的作用域将会一直在内存中保存到闭包不存在为止

```
function assignHandler(){
        var element  = document.getElementById("someElement");
        var id = element.id;  // 消除闭包中的循环引用
        element.onclick  = function(){
                alert(id);
        };
        element  = null;
}
```

匿名函数可以用来模仿块级作用域以避免变量污染

```
(function(){
// 这里是块级作用域，定义的任何变量会在执行结束时被销毁
// 将函数声明包含在一对圆括号中，表示它实际上是一个函数表达式
})(); // 紧随其后的另一对圆括号会立即调用这个函数
```

```
(function() { return 'hello objc' }) 闭包，指向函数的指针，返回函数
function() { return 'hello objc' } 直接执行报错，必须有变量接收
```