面向对象这块知识点，是javascript中的重点与难点。许多库的实现都是基于面向对象思想，而且平常面试中，也是一个高频考点甚至是必考点。

很多文章一开头就各种介绍原型链、原型、构造函数等等，这样容易让一些javascript初学者感到混乱，因此本文采取渐进式的讲解，尽量简单易懂的描述来展示面向对象的相关知识，结合图文与例子来帮助大家快速理解面向对象。

## 对象的定义

ECMA-262 把对象定义为：“无序属性的集合，其属性可以包含基本值、对象或者函数。”严格来讲， 这就相当于说对象是一组没有特定顺序的值。简单的说，对象是由一些无顺序的key-value对组成的，其中value可以包含基本值，对象或者函数。

比如：

var person = {

name: 'Tom',

age: 18,

getName: function() {},

parent: {}

}

在该例子中，person就是一个对象。

**创建对象**

在javascript中，创建对象无非就两种形式：

1. new的方式创建

var obj = new Object();

2.对象字面量的形式创建一个简单的对象

var obj = {};

本质上，对象都是通过函数创建的，第2中创建对象的方式是第一种的语法糖而已。

**给对象添加方法**

给对象添加方法有两种方式：

// 第一种

var person = {};

person.name = "TOM";

person.getName = function() {

return this.name;

}

// 第二种

var person = {

name: "TOM",

getName: function() {

return this.name;

}

}

**对象属性和方法的访问**

比如有以下对象：

var student = {

name: 'TOM',

}

那么我们访问对象student的方式有两种：

1. student .name
2. student [‘name’]

这两种方式的区别是，第二种可以用于属性名为变量时候的访问，比如

var abc = ‘name’

student [abc] // 等同于abc

## 工厂模式-提升创建对象效率

虽然使用上述方式创建对象比较简单，但是如果我们需要100个student对象时，且100个student对象只是name不同，就不得不定义100个student对象，但只是name不同。这种方式会造成太多相似对象，代码变得十分臃肿。

工厂模式是软件工程领域一种广为人知的设计模式，这种模式抽象了创建具体对象的过程。提供一个对象的模子，像工厂一样，需要多少个对象就复制出多少个。

看代码：

function createPerson(name, age){

var o = new Object();

o.name = name;

o.age = age;

o.sayName = function(){

return this.name

};

return o;

}

var person1 = createPerson("Nicholas", 29);

var person2 = createPerson("Greg", 27);

createPerson内部实现了一个模子对象o，并支持函数入参作为模子对象o的一些属性，通过调用createPerson函数和传入对应参数，就可以得到对应的实例对象。一样的key，定制化的value。

工厂模式虽然解决了创建 多个相似对象的问题，但却没有解决对象识别的问题（即怎样知道一个对象的类型）。

var obj = {};

var foo = function() {}

console.log(obj instanceof Object); // true

console.log(foo instanceof Function); // true

随着 JavaScript 的发展，又一个新模式出现（ps：有同学留下学不动的泪水吗）

## 构造函数

构造函数模式的目的就是为了创建一个自定义类，并且创建这个类的实例。构造函数模式中拥有了类和实例的概念，并且实例和实例之间是相互独立的，简单的说就是不同的对象。

构造函数就是一个普通的函数，创建方式和普通函数没有区别，但是有两点与普通函数不同：

1. 构造函数习惯上首字母大写。
2. 2.就是调用方式的不同，普通函数是直接调用，而构造函数需要使用new关键字来调用。

由于new关键字调用，构造函数与普通函数的表现是不同。

function demo() {

console.log(this);

}

demo(); // window

new demo(); // demo

为了方便演示，我们没有使用Demo大写模式。

为什么会出现这种调用差异呢？new关键字到底做了什么呢？

手撕代码实现下new过程：

// 先一本正经的创建一个构造函数，其实该函数与普通函数并无区别

var student= function(name, age) {

this.name = name;

this.age = age;

this.getName = function() {

return this.name;

}

}

// 将构造函数以参数形式传入

function New(func) {

// 声明一个中间对象，该对象为最终返回的实例

var res = {};

if (func.prototype !== null) {

// 将实例的原型指向构造函数的原型

res.\_\_proto\_\_ = func.prototype;

}

// ret为构造函数执行的结果，这里通过apply，将构造函数内部的this指向修改为指向res，即为实例对象

var ret = func.apply(res, Array.prototype.slice.call(arguments, 1));

// 当我们在构造函数中明确指定了返回对象时，那么new的执行结果就是该返回对象

if ((typeof ret === "object" || typeof ret === "function") && ret !== null) {

return ret;

}

// 如果没有明确指定返回对象，则默认返回res，这个res就是实例对象

return res;

}

// 通过new声明创建实例，这里的p1，实际接收的正是new中返回的res

var p1 = New(student, 'tom', 20);

console.log(p1.getName());

// 当然，这里也可以判断出实例的类型了

console.log(p1 instanceof Person); // true