对象都是通过函数创建的，对象属性可以随时改动的。

**注意：构造函数的函数名第一个字母大写（规则约定）**

**对象创建表达式：创建一个对象并调用一个函数（这个函数称做构造函数）初始化新对象的属性，对象创建表达式和函数调用表达式非常类似，只是对象创建表达式之前多了一个关键字new:**

**new Object()**

**new Date()**

调用构造函数的一个重要特征：构造函数的prototype属性被用做新对象的原型。意味着通过同一个构造函数创建的所有对象都继承自一个相同的对象，因此它们都是同一个类的成员。

构造函数的调用和普通函数调用是不尽相同的，构造函数就是用来“构造新对象”，它必须通过关键字“new”调用。

函数是一种对象，也是属性的集合，可以对函数进行自定义属性。

JS中所有东西都对象，任何一个对象都是有一个原型的

所有通过对象字面量创建的对象都连接到object.prototype这个JS中标准的对象

**构造函数或自定义函数才有prototype**

**constructor指针指向函数本身**

**实例化对象才有\_\_proto\_\_**

查看原型使用 \_\_proto\_\_（内部原型对象）

\_\_proto\_\_：每个对象都有一个名为\_\_proto\_\_的内部隐藏属性，引用了创建这个对象的函数的prototype fn.\_\_proto\_\_ === Fn.prototype,**指向于创建它的函数的原型对象（prototype）**。原型链正是基于\_\_proto\_\_才得形成

函数是Function的实例化对象，由Function构造出来，函数的原型Function.prototype

Function是Object的实例对象，原型最终指向Object.prototype

Object.prototype.\_\_proto\_\_ === null 原型链至此结束。

Function/Object：JS自带的函数对象

prototype：每一个函数对象都有一个显式的prototype属性，它代表了对象的原型

prototype的属性值是一个对象（属性的集合），默认只一个constructor的属性，指向这个函数本身（F.prototype.constructor === F）

function Fn(){

Fn.prototype.name = 'pschen';

Fn.prototype.getYear = function(){

return 1979;

};

}

var fn = new Fn();

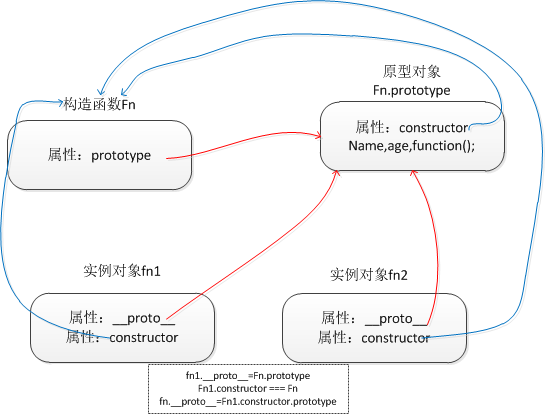
实例化fn对象有三步：

1、创建fn对象：var fn = new Fn();

2、将fn的内部属性\_\_proto\_\_指向构造它的函数Fn的属性prototype，fn. constructor === Fn === Fn.prototype. constructor。fn是没有prototype属性的，但有\_\_proto\_\_

3、将fn作为this调用构造函数Fn，从而设置成员（即对象属性、方法）并初始化。

C:\Users\Pschen\Documents\1.png



**prototype的作用**：基于由构造函数创建的对象都会继承原型属性和原型方法，可以给原型对象添加属性、方法让所有的实例化对象共享

**原型可以用来实现方法和属性的共享，也可以用来拓展对象的方法（内置函数的）**

字面量方式创建

User.prototype ={

constructior:User,

addr: “广西容县”,

show:function(){

console.log(this.name + “ | ” +this.age + “ | ” +this.addr);

}

};

例子：

function User(name,age){ //创建构造函数

this.name = name; //添加对象属性(全局变量)

this.age = age;

this.addr = “广西玉林”;

User.prototype.addr = “广西容县”; //在原型中添加属性

User.prototype.show = function(){ //在原型中添加方法

console.log(this.name + “ | ” +this.age + “ | ” +this.addr);

};

}

如果我们同时申明了对象属性、原型属性和实例属性，那么调用时显示的优先级应该是：实例属性>对象属性>原型属性。这就是采用了就近原则：调用时首先查找实例中是否直接定义了这个属性，有则返回实例属性；如果实例属性中没有就去构造函数中查找，有则返回；如果前面两者都没有就去原型对象中查找，如果没有则返回undefined。

var user1 = new User(“pschen”,38);

var user2 = new User(“ellen”,36);

user2.addr = “广东深圳”; //覆盖对象属性,但不会修改原型那个属性

//delete user2.addr; //删除对象属性后返回原型属性

user1.show();

user2.show();

console.log(user1.show === user2.show);

user1.hasOwnProperty(“addr”);判断属性是存在于实例还是存在于原型，存在于实例中返回true

原型和in操作符

两种方式使用in操作符，单独使用、for-in循环中使用，单独使用时，in操作符会在通过对象能够访问给定属性时返回true，无论该属性存在于实例中还是原型中

console.log(“addr” in user1);

function isPro(object,addr){

return !object.hasOwnProperty(addr) && (addr in object);

}

user2.addr = “广东深圳”;

console.log(isPro(user2,”addr”));

User.prototype等于使用一个对象字面量形式创建的新对象覆盖了原来的原型对象，这样造成constructor属性不再指向User(因为创建一个函数时同时创建它的prototype对象，而prototype对象的constructor属性指向这个函数)，要把constructor属性重新指向User

函数也是一种对象，是由"Function"创建

function fn1(x,y){

return x + y;

};

console.log(fn1(10,20));

var fn2 = new Function("x","y","return x + y;"); //不推荐使用此方式

console.log(fn2(5,6));

继承：JS的中的继承是通过原型链来体现的。

访问一个对象的属性时，先在基本属性中查找，如果没有，再沿着\_\_proto\_\_这条链向上找，这就是原型链。

如何判断属性是基本还是原型中找到的，使用hasOwnProperty()

理解原型对象

只要创建了一个新函数，就会根据一组特定的规则为该函数创建一个prototype属性，这个属性指向函数的原型对象，默认情况下，所有原型对象都会自动获得一个constructor属性，这个属性包含一个指向prototype属性所在函数的指针。

创建自定义的构造函数之后，其原型对象默认只会取得constructor属性，其他方法则都是从object继承而来，当调用构造函数创建一个新实例后，该实例内部将包含一个指针（\_\_proto\_\_），指向构造函数的原型对象

