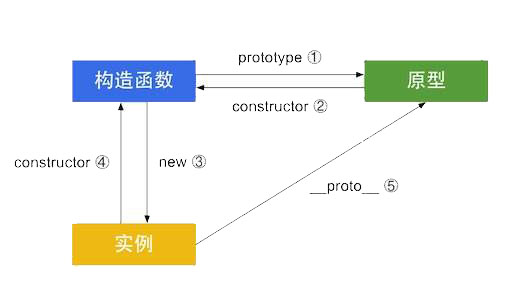
**原型链**

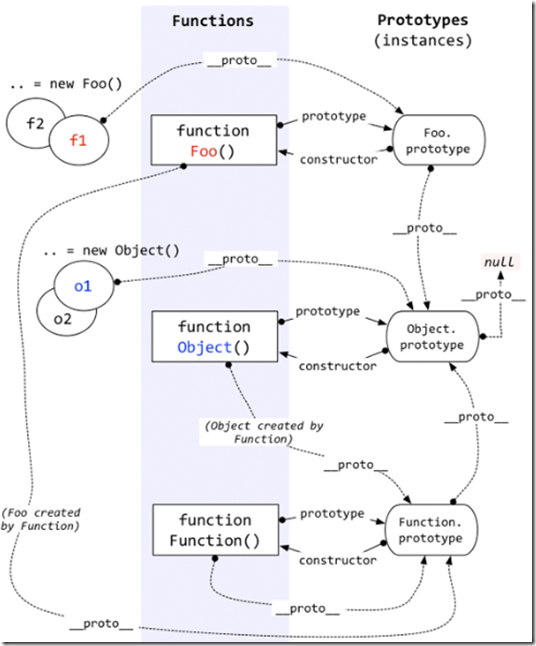
1. **原型对象**

每创建了一个新函数，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。在默认情况下，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。



1. **原型链**

在创建对象的时候，都会有一个内置属性\_\_proto\_\_，用于指向创建它的函数对象的prototype,也就是原型对象；原型对象作为一个对象，也有\_\_proto\_\_这样一个内置属性，指向它的原型对象，依次类推就形成了原型链。



1. **原型链的特点**

3.1 原型对象上的属性和方法被所有继承它的对象所共享；

3.2 对对象属性的操作不会影响到原型对象；

3.3 当读取对象的某一属性或者方法时，会从该对象的属性开始查找，如果没有找到，会在其原型链上查找，依次类推，直到找到为止，如果最终没有找到则返回undefined；

练习：

1. var a = { }; a.\_\_proto\_\_.\_\_proto\_\_;

2. 如何遍历原型链上所有属性？

3.var F = function(){};

Object.prototype.a = function a(){};

Function.prototype.b = function b(){};

var f = new F();

console.log(f.a)

console.log(f.b)

**继承**

1. **原型链继承**

思想：使一个构造函数的原型等于另一个构造函数的实例；

例：

function SuperType(){ this.property = true; }

function SubType(){ this.subproperty = false; }

SubType.prototype = new SuperType();

var instance = new SubType();

alert(instance.subproperty);//true

优点：原型链上所有的属性和方法都可以被继承；

缺点：所有实例都共享原型对象的属性和方法，对于属性值是引用类型时，会存在很大问题；

2．子类无法向父类传参；

**2.借用构造函数**；

思想;在子类构造函数内部借助call/apply方法调用父类构造函数；

例：

function SuperType(){ this.name=’Join’ }

function SubType(){ SuperType SuperType.call(this);this.age=’32’ }

var instance = new SubType();

缺点：1.所有的实例各自都拥有一个继承下来的属性的副本，无法做到复用；

2.父类的原型对象上的属性和方法对子类是不可见的；

优点：可以向父类构造函数传参；

**3.组合继承**

思想：是将原型链和借用构造函数的 技术组合到一块，从而发挥二者之长的一种继承模式。具体思路：使用原型链实现对原型属性和方法的继承，而通过借用构造函数来实现对实例属性的继承。

例:

function SuperType(name){ this.name = name; }

function SubType(name, age){ SuperType.call(this, name); this.age = age; }

SubType.prototype = new SuperType();

SubType.prototype.constructor = SubType;

缺点：两次调用了父类构造函数：一次是在创建子类型原型的时候，另一次是 在子类型构造函数内部；调用子类构造函数时重写了父类实例属性；

1. **原型式继承**

思路：是借助原型可以基于已有的对象创建新对象，同时还不必因此创建自定义类型。

例：

function object(o){ function F(){} F.prototype = o; return new F(); }

var animal = { name: 'xiaohua'};

var another = object(animal);

console.log(another.name)//xiaohua

优点：实现继承时不必创建自定义类型；

缺点：无法判断属于哪个类；

1. **寄生式继承**

思路：创建一个仅用于封装继承过程的函数，该函数在内部以某种方式来增强对象；

例:

function createAnother(original){

var clone = object(original); //通过调用函数创建一个新对象

clone.sayHi = function(){ //以某种方式来增强这个对象 alert("hi"); };

return clone; //返回这个对象 }

缺点：无法判断类；

1. **寄生组合式继承**

思路：不必为了指定子类型的原型而调用超类型的构造函数，我们所需要的无非就是超类型 原型的一个副本而已。可以通过借用构造函数来继承属性，通过原型链的混成形式来继承方法。

例：

function inheritPrototype(subType, superType){

var prototype = Object.create(superType.prototype); //创建对象

prototype.constructor = subType; //增强对象

subType.prototype = prototype; //指定对象

return subType

}

function SuperType(name){

this.name = name;

}

function SubType(name, age){

SuperType.call(this, name);

this.age = age;

}

inheritPrototype(SubType, SuperType);