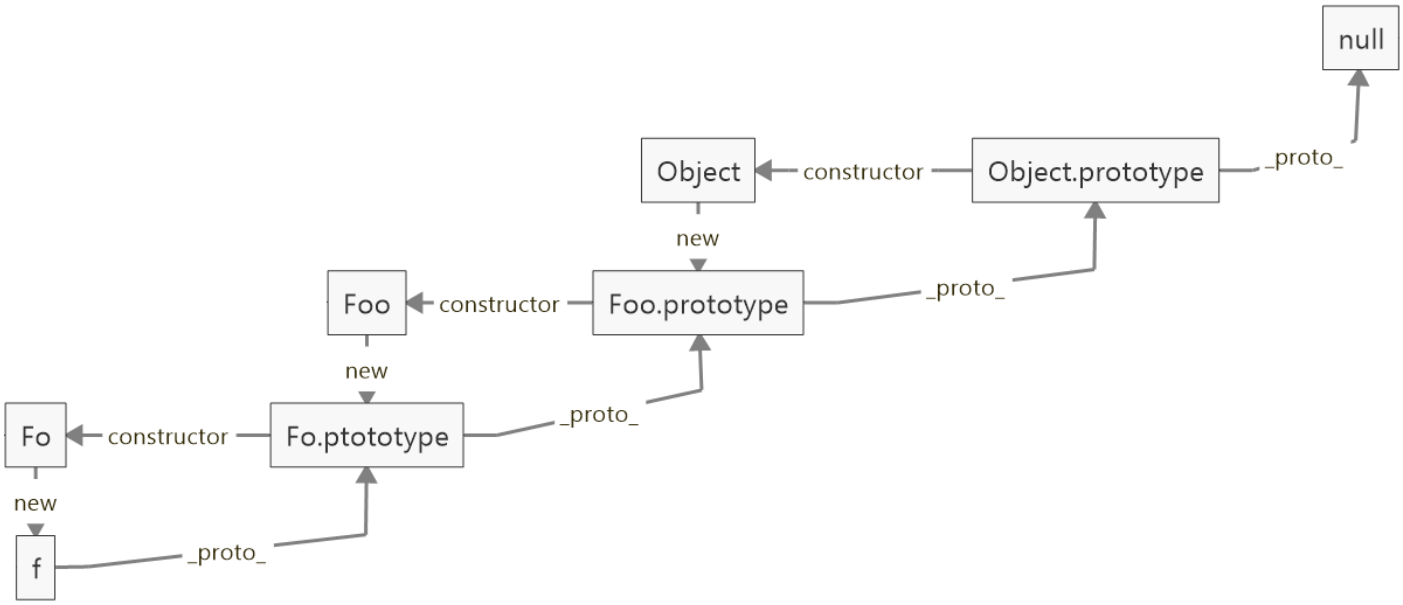
1.原型对象
只要创建一个新函数, 就会为这个函数创建一个 prototype 属性, 该属性值为一个指向函数的原型对象的指针.
原型对象会获得一个constructor 属性, 值指向其所在函数.
调用构造函数创建一个实例后, 该实例内部将包含一个指针(ECMA-262中被叫做[[prototype]], 即 __proto__, 指向其构造函数的原型对象.
由同一构造函数创建的实例共享函数原型对象里的属性和方法

```
> function Father(name, age) {
      this.name = name
      this.age = age
      this.sayName = function() {
          log(this.name)
      }
  }
< undefined
> Father.prototype
< {constructor: f}
    constructor: f Father(name, age)
      arguments: null
      caller: null
      length: 2
      name: "Father"
    ▶ prototype: {constructor: f}
    ▶ __proto__: f ()
      [[FunctionLocation]]: VM368:1
    ▶ [[Scopes]]: Scopes[1]
  ▶ __proto__: Object
> f = new Father('dad', '18')
< Father {name: "dad", age: "18", sayName: f}
    age: "18"
    name: "dad"
  ▶ sayName: f ()
  ▼ __proto__:
    ▶ constructor: f Father(name, age)
    ▶ __proto__: Object
```

2.原型链
当访问一个对象的某个属性时，会先在这个对象本身属性上查找，如果没有找到，则会去它的__proto__隐式原型上查找，即它的构造函数的prototype，如果还没有找到就会再在构造函数的prototype的__proto__中查找，这样一层一层向上查找就会形成一个链式结构，我们称为原型链。
图示:



可以通过重写原型对象来实现继承

3.继承的实现方式

1)原型链

```
function Animal(type) {
    this.type = type
    this.colors = ['red', 'Black', 'blue']
    this.logType = function() {
        log(this.type)
    }
}

function Cat(name) {
    this.name = name
    this.age = age
}

Cat.prototype = new Animal('cat')
Cat.prototype.constructor = Cat
```

缺点：父级函数引用类型的属性为子级函数的实例所共享

2)借用构造函数

```
function Animal(type) {
    this.type = type
    this.colors = ['red', 'Black', 'blue']
    this.logType = function() {
        log(this.type)
    }
}

function Cat(name) {
```

```
    Animal.call(this, 'cat')
    this.name = name
    this.age = age
}
```

优点: 解决了原型链继承的父级函数引用属性共享的问题
可以向父级函数传递参数
缺点: 方法都在构造函数中定义, 未实现函数复用
父级函数的原型属性和方法在子级函数中不可见

3)组合继承

```
function Animal(type) {
    this.type = type
    this.logType = function() {
        log(this.type)
    }
}

function Cat(name) {
    Animal.call(this, 'cat') // 第二次调用 => Cat 的实例属性 this.type = 'cat'
}

Cat.prototype = new Animal('cat')  // 第一次调用 => Cat.prototype.type = 'cat'
Cat.prototype.constructor = Cat
```

优点: 结合原型链和借用构造函数:
原型链实现对原型属性和方法的继承, 实现函数复用,
借用构造函数实现对父级构造函数中属性的继承, 保证子级函数的每个实例都有自己的属性
缺点: 会调用两次父级构造函数

4)寄生组合继承

```
function Animal(type) {
    this.type = type
    this.logType = function() {
        log(this.type)
    }
}

function Cat(name) {
    Animal.call(this, 'cat') // 仅一次调用
}

function inheritPrototype(Cat, Animal) {
    var o = object(Animal.prototype)
    o.constructor = Cat
    Cat.prototype = o
}

inheritPrototype(Cat, Animal)
```

优点：解决了组合继承调用两次父级构建函数的问题

5)ES6 类

```
class Animal {
    constructor(type) {
        this.type = type
    }
    logType() {
        log(this.type)
    }
}

class Cat extends Animal {
    constructor(type) {
        super(type)
    }
}
```

相关面试题:

1. 实现继承的方式有哪些?
2. var F = function() {}
   Object.prototype.a =function() {}
   Function.prototype.b= function() {}
   f = new F()
   f 有 a 方法么? (有) f (no) => f.__proto__ == F.prototype(no) => F.prototype.__proto__ == Object.prototype(yes)

   有 b 方法么? (没有) f (no) => f.__proto__ == F.prototype(no) => F.prototype.__proto__ == Object.prototype(no)