原型链类

- 创建对象有几种方法
- 原型,构造函数,实例,原型链
- instanceof原理
- new 运算符

创建对象有几种方法

1. 字面量对象，new Object()创建

```
var o1 = {name:'o1'};//字面量对象（默认原型链指向Object）
var o11 = new Object({name:'o11'});//new Object()创建对象
```

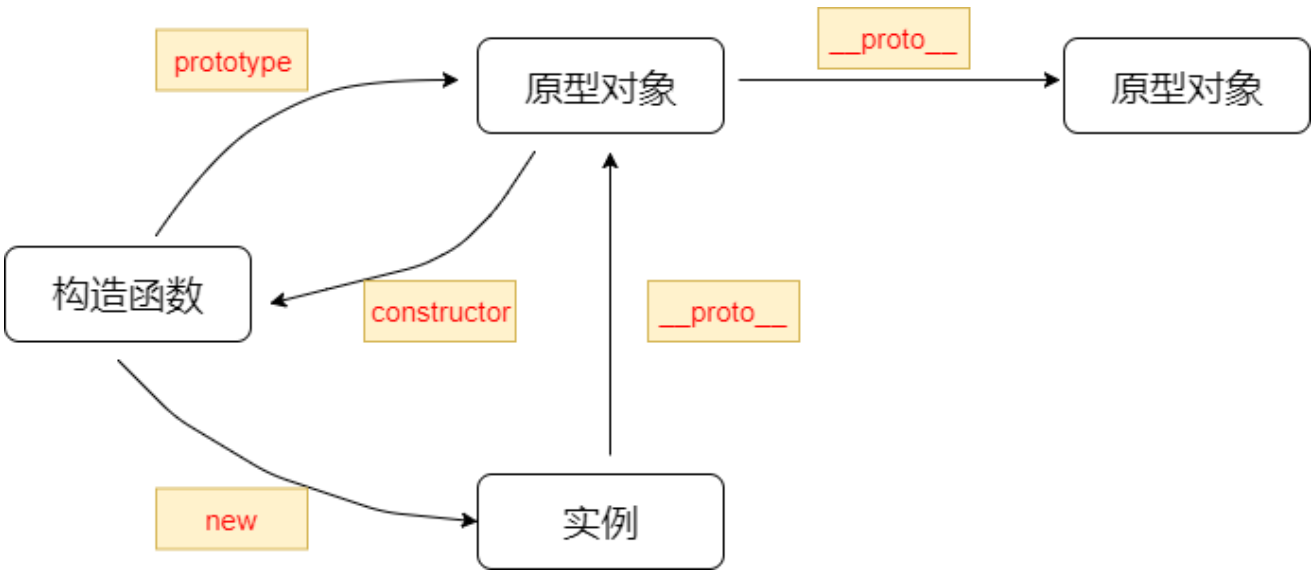
2. 显示构造函数创建对象

```
var M = function(){
    this.name='o2';
}
var o2 = new M();
```

3. Object.create()方法创建

```
var P = {name:'o3'};
var o3 = Object.create(P);
```

原型,构造函数,实例,原型链



1. 任何函数只要被new使用了就可以被称为构造函数，不用new的函数则是普通函数。
2. 当新建一个函数时，js引擎会为这个函数添加一个prototype属性，这个属性会创建一个空对象，也就是原型对象。

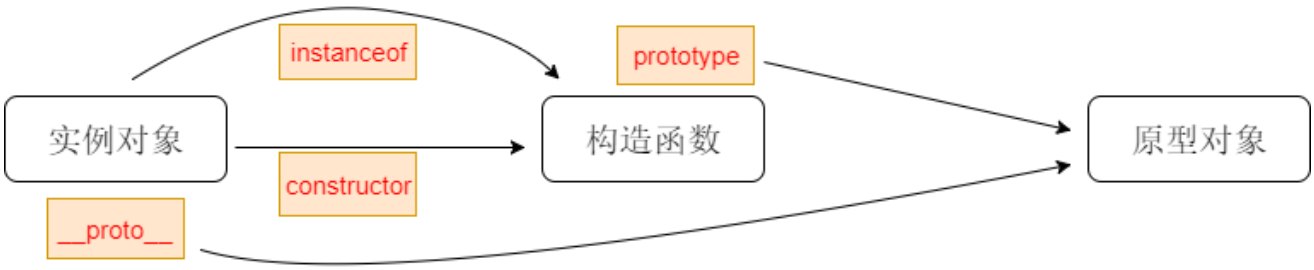
3. 原型对象中会有一个构造器constructor，默认指向你声明的那个函数

```
function M(name){
    this.name = name;
}
var o3 = new M('o3');
//print o3
M {name: "o3"}
-------------------------
//spread
name:"o3"
__proto__:Object
-------------------------
//spread more
name:"o3"
__proto__:
  constructor:ƒ M(name)
  __proto__:Object
-------------------------
//print M
ƒ M(name){
  this.name = name;
}
-------------------------
//print M.prototype
Object{
  constructor:ƒ M(name)
  __proto__:Object
}
-------------------------
M.prototype.constructor === M;//true
o3.__proto__ === M.prototype;//true
-------------------------
//原型链的顶端为 Object.prototype
-------------------------
M.prototype.say = function(){
   console.log('h1');
}
var o5 = new M('o5');
o3.say();//hi
o5.say();//hi
```

4. 在访问一个实例的时候，比如说在这个实例上有什么方法，在这个实例本身上没有找到这个方法和属性的话，它通过`__proto__`往它的原型对象上找（上一级原型对象），如果还没有找到，继续通过`__proto__`往再上一级原型对象找，找对应的方法后立即原路返回，如果还没有找到则继续往再上一级原型对象找知道`Object.prototype`。
5. 只有实例对象有`__proto__`属性，函数即是函数也是对象，所以函数也有`__proto__`属性。
6.
```
M.__proto__ === Function.prototype;//true
```

`M`这个普通函数的构造函数是`Function`，也可以理解为`M`这个普通函数是`Function`这个函数的实例。

instanceof 原理



1. `__proto__`实际关联的是构造函数的`prototype`属性，也就是构造函数的原型对象。
2. `instanceof`的运力是判断实例对象的`__proto__`属性和构造函数的`prototype`属性是不是同一个引用（原型对象）。
3. 在判断实例对象是否为该构造函数的实例的时候，实际上判断的是实例对象的`__proto__`和构造函数的`prototype`是不是引用同一个地址。

```
//print o3
M {name: "o3"}
-----------------------------
o3 instanceof M;//true
o3 instanceof Object;//true
//只要在原型链上的构造函数都会被instanceof看作是实例的构造函数
o3.__proto__ === M.prototype;//true
M.prototype.__proto__ === Object.prototype;//true
-----------------------------
o3.__proto__.constructor === M;//true
o3.__proto__.constructor === Object;//false
```

new 运算符

new 操作过程

1. 一个新对象（理解为空对象，字面量对象）被创建。它继承自`foo.prototype`（`foo`为构造函数）。
2. 构造函数`foo`被执行。执行的时候，相应的传参会被传入，同时上下文（`this`）会被指定为这个新实例。`new foo`等同于`new foo()`，只能用在不传递任何参数的情况下。
3. 如果构造函数返回了一个“对象”，那么这个对象会取代整个`new`出来的结果。如果构造函数没有返回值，那么`new`出来的结构为步骤1创建的对象。

```
//new 运算符工作原理
var new2 =function(func){
    var o = Object.create(func.prototype);
    var k = func.call(o);
    if(typeof k === 'Object'){
        return k
    }else{
        return o
    }
}
```