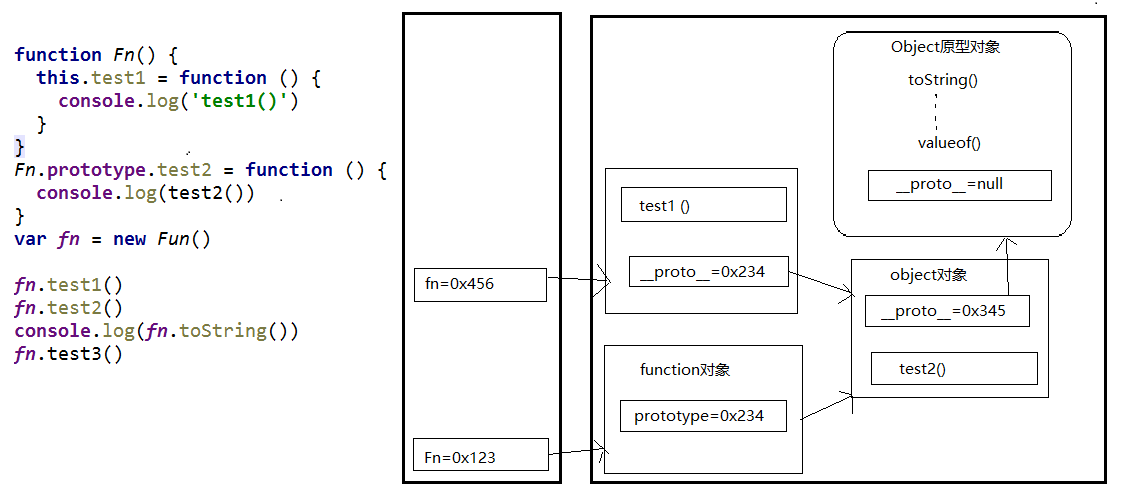
**1. 函数的prototype属性**

\* 每个函数都有一个prototype属性, 它默认指向一个Object空对象(即称为: 原型对象)

\* 原型对象中有一个属性constructor, 它指向函数对象



**2. 给原型对象添加属性(一般都是方法)**

\* 作用: 函数的所有实例对象自动拥有原型中的属性(方法)

**3. 显式原型和隐式原型**

1. 每个函数function都有一个prototype，即显式原型(属性)

2. 每个实例对象都有一个\_\_proto\_\_，可称为隐式原型(属性)

3. 对象的隐式原型的值为其对应构造函数的显式原型的值

4. 内存结构(图)

5. 总结:

\* 函数的prototype属性: 在定义函数时自动添加的, 默认值是一个空Object对象

\* 对象的\_\_proto\_\_属性: 创建对象时自动添加的, 默认值为构造函数的prototype属性值

\* 程序员能直接操作显式原型, 但不能直接操作隐式原型(ES6之前)

**4. 原型链**

\* 访问一个对象的属性时，

\* 先在自身属性中查找，找到返回

\* 如果没有, 再沿着\_\_proto\_\_这条链向上查找, 找到返回

\* 如果最终没找到, 返回undefined

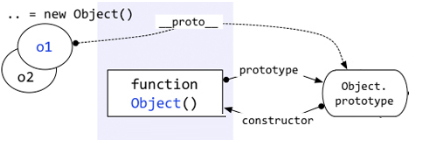
\* 别名: 隐式原型链

\* 作用: 查找对象的属性(方法)

**5. 构造函数/原型/实体对象的关系**

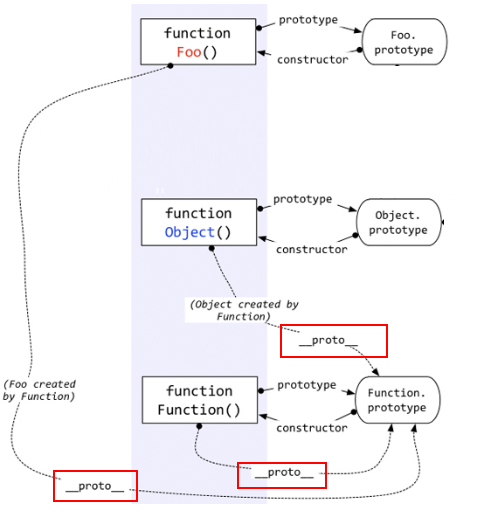
var o1 = new Object();

var o2 = {};



**6.构造函数/原型/实体对象的关系2**

function Foo(){ }



**7.原型链 属性问题**

1. 读取对象的属性值时: 会自动到原型链中查找

2. 设置对象的属性值时: 不会查找原型链, 如果当前对象中没有此属性, 直接添加此属性并设置其值

3. 方法一般定义在原型中, 属性一般通过构造函数定义在对象本身上

**8. instanceof是如何判断的?**

\* 表达式: A instanceof B

\* 如果B函数的显式原型对象在A对象的原型链上, 返回true, 否则返回false

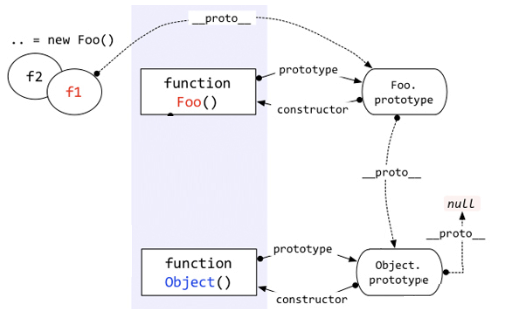
**注意：**

1.所有函数的显式原型都是空的Object的实例对象（但是Object()不满足）

2.所有函数都是Function()的实例化对象(包括Function()自己)

3.Object()的原型对象prototype是原型链的尽头

Object.prototype.\_\_proto\_\_ === null



**9. Function是通过new自己产生的实例**

