我在写一篇图解prototype和\_\_proto\_\_的区别时，搜资料搜到了一个有意思的现象，下面这两个运算返回的结果是一样的：

Function instanceof Object;//true

Object instanceof Function;//true

这个是怎么一回事呢？要从运算符instanceof说起。

一、instanceof究竟是运算什么的？

我曾经简单理解instanceof只是检测一个对象是否是另个对象new出来的实例（例如var a = new Object()，a instanceof Object返回true），但实际instanceof的运算规则上比这个更复杂。

首先w3c上有官方解释（传送门，有兴趣的同学可以去看看），但是一如既往地让人无法一目了然地看懂……

知乎上有同学把这个解释翻译成人能读懂的语言（传送门），看起来似乎明白一些了：

//假设instanceof运算符左边是L，右边是R

L instanceof R

//instanceof运算时，通过判断L的原型链上是否存在R.prototype

L.\_\_proto\_\_.\_\_proto\_\_ ..... === R.prototype ？

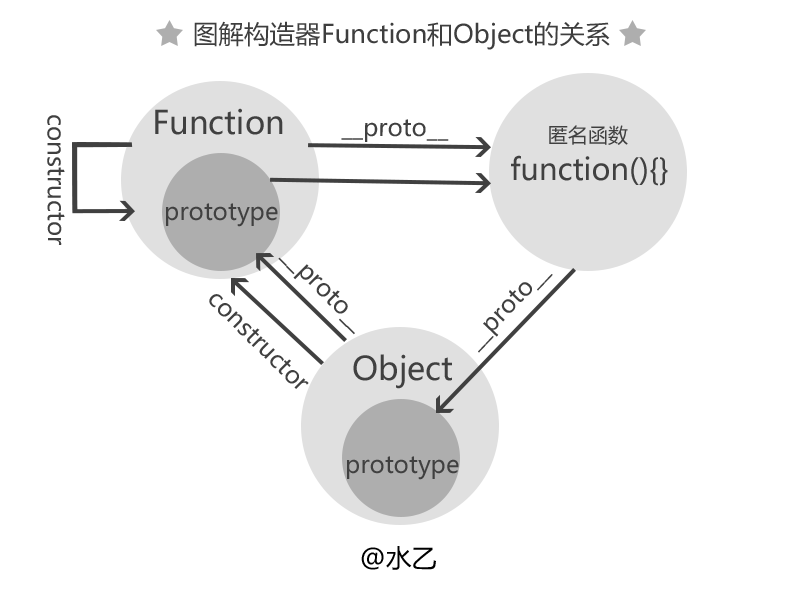
//如果存在返回true 否则返回false

注意：instanceof运算时会递归查找L的原型链，即L.\_\_proto\_\_.\_\_proto\_\_.\_\_proto\_\_.\_\_proto\_\_...直到找到了或者找到顶层为止。

所以一句话理解instanceof的运算规则为：

instanceof检测左侧的\_\_proto\_\_原型链上，是否存在右侧的prototype原型。

二、图解构造器Function和Object的关系



图解构造器Function和Object的关系

我们再配合代码来看一下就明白了：

//①构造器Function的构造器是它自身

Function.constructor=== Function;//true

//②构造器Object的构造器是Function（由此可知所有构造器的constructor都指向Function）

Object.constructor === Function;//true

//③构造器Function的\_\_proto\_\_是一个特殊的匿名函数function() {}

console.log(Function.\_\_proto\_\_);//function() {}

//④这个特殊的匿名函数的\_\_proto\_\_指向Object的prototype原型。

Function.\_\_proto\_\_.\_\_proto\_\_ === Object.prototype//true

//⑤Object的\_\_proto\_\_指向Function的prototype，也就是上面③中所述的特殊匿名函数

Object.\_\_proto\_\_ === Function.prototype;//true

Function.prototype === Function.\_\_proto\_\_;//true

三、当构造器Object和Function遇到instanceof

我们回过头来看第一部分那个“奇怪的现象”，从上面那个图中我们可以看到：

Function.\_\_proto\_\_.\_\_proto\_\_ === Object.prototype;//true

Object.\_\_proto\_\_ === Function.prototype;//true

所以再看回第一点中我们说的instanceof的运算规则，Function instanceof Object 和 Object instanceof Function运算的结果当然都是true啦！

如果看完以上，你还觉得上面的关系看晕了的话，只需要记住下面两个最重要的关系，其他关系就可以推导出来了：

1、**所有的构造器的constructor都指向Function**

2、**Function的prototype指向一个特殊匿名函数，而这个特殊匿名函数的\_\_proto\_\_指向Object.prototype**