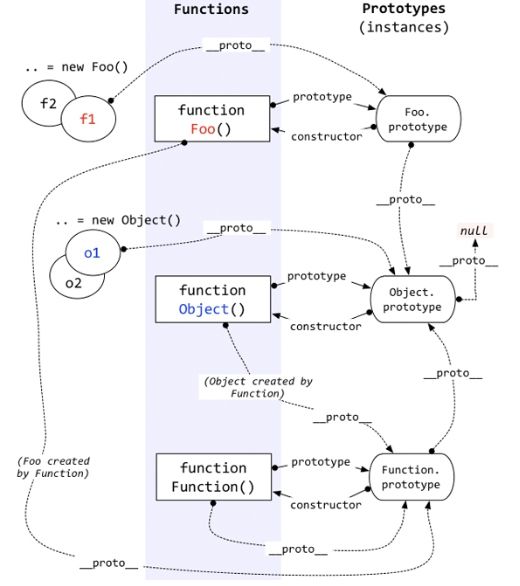
作者：doris  
链接：https://www.zhihu.com/question/34183746/answer/58155878  
来源：知乎  
著作权归作者所有，转载请联系作者获得授权。

是时候拿出我珍藏多年的这张图了：  


初学javascript的时候也跟题主一样搞不清楚，自己好好总结了一下：  
  
首先，要明确几个点：

**方法是对象,构造函数是对象,构造函数的原型是对象**

**Js中的string是基本类型没有对象的特性,而String是引用类型,做对象来处理。其它Date、Boolean、function()等引用类型的对象都有\_\_proto\_\_属性。**  
**1.JS的对象。**

**方法（Function）是对象**,**方法的原型(Function.prototype)是对象**。因此，**它们都会具有对象共有的特点**。  
即：**任何对象都具有属性\_\_proto\_\_**，可称为隐式原型，**对象的\_\_proto\_\_属性指向创建该对象的构造函数的原型对象，这保证了对象的实例能够访问到构造函数的原型对象中定义的属性和方法。**  
**2.方法(Function)**  
 **方法这个特殊的对象，除了和其他对象一样有上述\_proto\_属性之外，还有自己特有的属性——原型属性（prototype）**，**prototype属性指向该方法的原型对象。**

**3.原型对象**

**原型对象也有一个独特的属性，叫做constructor，这个属性包含了一个指针，指回原构造函数。原型对象是种特殊的对象形式，但并非函数对象，所以不具有prototype属性**  
好啦，知道了这两个基本点，我们来看看上面这副图。

对象的\_\_proto\_\_属性指向该对象的**构造函数**的**原型对象。**

**方法(构造函数)**的prototype属性指向该构造函数的**原型对象**。

**对象的方法和属性都来自原型对象，**  
1.构造函数Foo()  
**构造函数的原型属性Foo.prototype指向了原型对象，在原型对象里有共有的方法，所有构造函数声明的实例（这里是f1，f2）都可以共享这个方法**。  
  
2.原型对象Foo.prototype  
Foo.prototype表示**原型对象**，它保存着实例共享的方法，有一个指针**constructor指回构造函数。**  
3.实例  
f1和f2是Foo这个对象的两个实例，这两个对象也有属性\_\_proto\_\_，指向构造函数的原型对象，这样子就可以像上面1所说的访问原型对象的所有方法啦。  
  
另外：  
构造函数Foo()除了是方法，也是对象啊，它也有\_\_proto\_\_属性，指向谁呢？  
指向它的构造函数的原型对象呗。函数的构造函数不就是Function嘛，因此这里的\_\_proto\_\_指向了Function.prototype。  
其实除了Foo()，Function(), Object()也是一样的道理。  
  
**原型对象也是对象啊，它的\_\_proto\_\_属性**，又指向谁呢？  
同理，**指向它的构造函数的原型对象**呗。**即Object.prototype.**  
最后，**Object.prototype的\_\_proto\_\_属性指向null**。  
  
  
总结：  
1.对象有属性\_\_proto\_\_,指向该对象的构造函数的原型对象。  
2.**方法除了有属性\_\_proto\_\_,还有属性prototype，prototype指向该方法的原型对象。**  
  
讲完啦，欢迎各种批评指正完善探讨，共同进步~

**回调函数：**

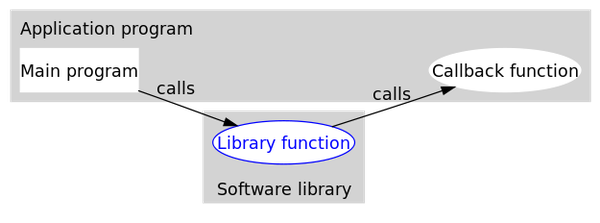
**你到一个商店买东西，刚好你要的东西没有货，于是你在店员那里留下了你的电话，过了几天店里有货了，店员就打了你的电话，然后你接到电话后就到店里去取了货。在这个例子里，你的电话号码就叫回调函数，你把电话留给店员就叫登记回调函数，店里后来有货了叫做触发了回调关联的事件，店员给你打电话叫做调用回调函数，你到店里去取货叫做响应回调事件。**

**什么是回调函数？**

我们绕点远路来回答这个问题。

编程分为两类：系统编程（system programming）和应用编程（application programming）。所谓系统编程，简单来说，就是编写**库**；而应用编程就是利用写好的各种库来编写具某种功用的程序，也就是**应用**。系统程序员会给自己写的库留下一些接口，即API（application programming interface，应用编程接口），以供应用程序员使用。所以在抽象层的图示里，库位于应用的底下。

当程序跑起来时，一般情况下，应用程序（application program）会时常通过API调用库里所预先备好的函数。但是有些库函数（library function）却要求应用先传给它一个函数，好在合适的时候调用，以完成目标任务。这个被传入的、后又被调用的函数就称为**回调函数**（callback function）。

打个比方，有一家旅馆提供叫醒服务，但是要求旅客自己决定叫醒的方法。可以是打客房电话，也可以是派服务员去敲门，睡得死怕耽误事的，还可以要求往自己头上浇盆水。这里，“叫醒”这个行为是旅馆提供的，相当于库函数，但是叫醒的方式是由旅客决定并告诉旅馆的，也就是回调函数。而旅客告诉旅馆怎么叫醒自己的动作，也就是把回调函数传入库函数的动作，称为**登记回调函数**（to register a 

可以看到，回调函数通常和应用处于同一抽象层（因为传入什么样的回调函数是在应用级别决定的）。而回调就成了一个高层调用底层，底层再**回**过头来**调**用高层的过程。（我认为）这应该是回调最早的应用之处，也是其得名如此的原因。

**回调机制的优势**

从上面的例子可以看出，回调机制提供了非常大的灵活性。请注意，从现在开始，我们把图中的库函数改称为**中间函数**了，这是因为回调并不仅仅用在应用和库之间。任何时候，只要想获得类似于上面情况的灵活性，都可以利用回调。

这种灵活性是怎么实现的呢？乍看起来，回调似乎只是函数间的调用，但仔细一琢磨，可以发现两者之间的一个关键的不同：在回调中，我们利用某种方式，把回调函数像参数一样传入中间函数。可以这么理解，在传入一个回调函数之前，中间函数是不完整的。换句话说，程序可以在运行时，通过登记不同的回调函数，来决定、改变中间函数的行为。这就比简单的函数调用要灵活太多了。请看下面这段Python写成的回调的简单示例：

`even.py`

#回调函数1

#生成一个2k形式的偶数

def double(x):

return x \* 2

#回调函数2

#生成一个4k形式的偶数

def quadruple(x):

return x \* 4

`callback\_demo.py`

from even import \*

#中间函数

#接受一个生成偶数的函数作为参数

#返回一个奇数

def getOddNumber(k, getEvenNumber):

return 1 + getEvenNumber(k)

#起始函数，这里是程序的主函数

def main():

k = 1

#当需要生成一个2k+1形式的奇数时

i = getOddNumber(k, double)

print(i)

#当需要一个4k+1形式的奇数时

i = getOddNumber(k, quadruple)

print(i)

#当需要一个8k+1形式的奇数时

i = getOddNumber(k, lambda x: x \* 8)

print(i)

if \_\_name\_\_ == "\_\_main\_\_":

main()

运行`callback\_demp.py`，输出如下：

3

5

9

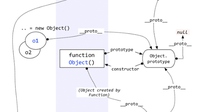
上面的代码里，给`getOddNumber`传入不同的回调函数，它的表现也不同，这就是回调机制的优势所在。值得一提的是，上面的第三个回调函数是一个匿名函数。

**易被忽略的第三方**

通过上面的论述可知，中间函数和回调函数是回调的两个必要部分，不过人们往往忽略了回调里的第三位要角，就是中间函数的调用者。绝大多数情况下，这个调用者可以和程序的主函数等同起来，但为了表示区别，我这里把它称为**起始函数**（如上面的代码中注释所示）。

之所以特意强调这个第三方，是因为我在网上读相关文章时得到一种印象，很多人把它简单地理解为两个个体之间的来回调用。譬如，很多中文网页在解释“回调”（callback）时，都会提到这么一句话：“If you call me, I will call you back.”我没有查到这句英文的出处。我个人揣测，很多人把起始函数和回调函数看作为一体，大概有两个原因：第一，可能是“回调”这一名字的误导；第二，给中间函数传入什么样的回调函数，是在起始函数里决定的。实际上，回调并不是“你我”两方的互动，而是ABC的三方联动。有了这个清楚的概念，在自己的代码里实现回调时才不容易混淆出错。

另外，回调实际上有两种：阻塞式回调和延迟式回调。两者的区别在于：阻塞式回调里，回调函数的调用一定发生在起始函数返回之前；而延迟式回调里，回调函数的调用有可能是在起始函数返回之后。这里不打算对这两个概率做更深入的讨论，之所以把它们提出来，也是为了说明强调起始函数的重要性。网上的很多文章，提到这两个概念时，只是笼统地说阻塞式回调发生在主调函数返回之前，却没有明确这个主调函数到底是起始函数还是中间函数，不免让人糊涂，所以这里特意说明一下。另外还请注意，本文中所举的示例均为阻塞式回调。延迟式回调通常牵扯到多线程，我自己还没有完全搞明白，所以这里就不多说了。

是时候拿出我珍藏多年的这张图了： 初学javascript的时候也跟题主一样搞不清楚，自己好好总结了一下： 首先，要明确几个点： 1.在JS里，万物皆对象。方法（Function）是对象，方法的原型(Function.prototype)是对象。因此，它们都会具有对象共有的特点。 即… [显示全部](file:///question/34183746/answer/58155878)