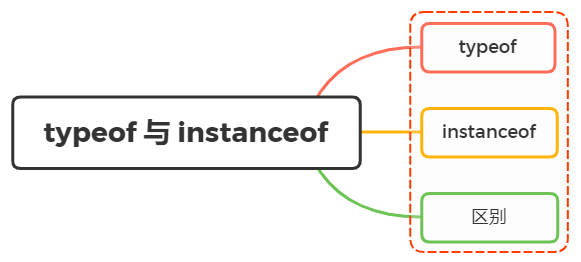
# 面试官：typeof 与 instanceof 区别



## 一、typeof

typeof 操作符返回一个字符串，表示未经计算的操作数的类型

使用方法如下：

typeof operand  
typeof(operand)

operand表示对象或原始值的表达式，其类型将被返回

举个例子

typeof 1 // 'number'  
typeof '1' // 'string'  
typeof undefined // 'undefined'  
typeof true // 'boolean'  
typeof Symbol() // 'symbol'  
typeof null // 'object'  
typeof [] // 'object'  
typeof {} // 'object'  
typeof console // 'object'  
typeof console.log // 'function'

从上面例子，前6个都是基础数据类型。虽然typeof null为object，但这只是JavaScript 存在的一个悠久 Bug，不代表null就是引用数据类型，并且null本身也不是对象

所以，null在 typeof之后返回的是有问题的结果，不能作为判断null的方法。如果你需要在 if 语句中判断是否为 null，直接通过===null来判断就好

同时，可以发现引用类型数据，用typeof来判断的话，除了function会被识别出来之外，其余的都输出object

如果我们想要判断一个变量是否存在，可以使用typeof：(不能使用if(a)， 若a未声明，则报错)

if(typeof a != 'undefined'){  
 //变量存在  
}

## 二、instanceof

instanceof 运算符用于检测构造函数的 prototype 属性是否出现在某个实例对象的原型链上

使用如下：

object instanceof constructor

object为实例对象，constructor为构造函数

构造函数通过new可以实例对象，instanceof能判断这个对象是否是之前那个构造函数生成的对象

// 定义构建函数  
let Car = function() {}  
let benz = new Car()  
benz instanceof Car // true  
let car = new String('xxx')  
car instanceof String // true  
let str = 'xxx'  
str instanceof String // false

关于instanceof的实现原理，可以参考下面：

function myInstanceof(left, right) {  
 // 这里先用typeof来判断基础数据类型，如果是，直接返回false  
 if(typeof left !== 'object' || left === null) return false;  
 // getProtypeOf是Object对象自带的API，能够拿到参数的原型对象  
 let proto = Object.getPrototypeOf(left);  
 while(true) {   
 if(proto === null) return false;  
 if(proto === right.prototype) return true;//找到相同原型对象，返回true  
 proto = Object.getPrototypeof(proto);  
 }  
}

也就是顺着原型链去找，直到找到相同的原型对象，返回true，否则为false

## 三、区别

typeof与instanceof都是判断数据类型的方法，区别如下：

* typeof会返回一个变量的基本类型，instanceof返回的是一个布尔值
* instanceof 可以准确地判断复杂引用数据类型，但是不能正确判断基础数据类型
* 而typeof 也存在弊端，它虽然可以判断基础数据类型（null 除外），但是引用数据类型中，除了function 类型以外，其他的也无法判断

可以看到，上述两种方法都有弊端，并不能满足所有场景的需求

如果需要通用检测数据类型，可以采用Object.prototype.toString，调用该方法，统一返回格式“[object Xxx]”的字符串

如下

Object.prototype.toString({}) // "[object Object]"  
Object.prototype.toString.call({}) // 同上结果，加上call也ok  
Object.prototype.toString.call(1) // "[object Number]"  
Object.prototype.toString.call('1') // "[object String]"  
Object.prototype.toString.call(true) // "[object Boolean]"  
Object.prototype.toString.call(function(){}) // "[object Function]"  
Object.prototype.toString.call(null) //"[object Null]"  
Object.prototype.toString.call(undefined) //"[object Undefined]"  
Object.prototype.toString.call(/123/g) //"[object RegExp]"  
Object.prototype.toString.call(new Date()) //"[object Date]"  
Object.prototype.toString.call([]) //"[object Array]"  
Object.prototype.toString.call(document) //"[object HTMLDocument]"  
Object.prototype.toString.call(window) //"[object Window]"

了解了toString的基本用法，下面就实现一个全局通用的数据类型判断方法

function getType(obj){  
 let type = typeof obj;  
 if (type !== "object") { // 先进行typeof判断，如果是基础数据类型，直接返回  
 return type;  
 }  
 // 对于typeof返回结果是object的，再进行如下的判断，正则返回结果  
 return Object.prototype.toString.call(obj).replace(/^\[object (\S+)\]$/, '$1');   
}

使用如下

getType([]) // "Array" typeof []是object，因此toString返回  
getType('123') // "string" typeof 直接返回  
getType(window) // "Window" toString返回  
getType(null) // "Null"首字母大写，typeof null是object，需toString来判断  
getType(undefined) // "undefined" typeof 直接返回  
getType() // "undefined" typeof 直接返回  
getType(function(){}) // "function" typeof能判断，因此首字母小写  
getType(/123/g) //"RegExp" toString返回