【前端面试】手写一个getType()函数用于判...

前言

这是一道很基础的面试题，经常用来考应届生或者实习生。但是看似如此简单的一道题，却让不少人在这上面翻了车。有些小伙伴可能觉得这无非就是考察如何判断数据类型，大家或者多少都知道一些，但是为什么会翻车呢?

原因很简单：说得不全，有些概念模棱两可!

今天就来总结一下如何判断数据类型，取得最优解!

1.JS 数据类型有哪些?

无非就是两大类，引用类型和值类型。

值类型:

- String
- Number
- Boolean
- Null
- Undefined
- Symbol

引用类型:

- Object
- Array
- Function
- RegExp
- Date

上面的类型我相信绝大多数小伙伴都能回答上来，但是面试官让我们写的这个函数真的就判断这些类型就行了吗?

答案不是的!

面试官可能还需要让我们判断这些特殊的：NaN、BigInt、Promise、Set、Map、weakMap、Error、Infinity 等等。

2.利用 typeof 实现 getType?

不会还有人不知道 typeof 吧！这是一个非常经典的判断 js 数据类型的方法，不少小伙伴都会考虑到用 typeof 来实现我们的 getType 方法，但是我们一定要先知道 typeof 方法的特点。

typeof 特点：

- 只能用来判断值类型（除 null），其它类型均返回 function 或者 object。
- 对于 null 返回 object。

看表格：

数据类型	返回结果
Null	”object”
Boolean	”boolean”
String	”string”
BigInt	”bigint”
Undefined	”undefined”
Number	”number”
Symbol	”symbol”
Function	”function”
其它对象	”object”

从上面的特点可以看出，typeof 完全不符合我们的预期！

但是，如果你非要利用 typeof 来实现我们的函数，也不是不行。我们可以结合 instanceof 来实现 getType 函数。

instanceof 简介：

> 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。

它的主要特点是用来判断某个数据是否符合某个类型，重在**判断！**

使用案例：

```
[] instanceof Array; // true
{} instanceof Object;// true
new Date() instanceof Date;// true
```

从上面的代码块可以看出，instanceof 返回的是一个 Boolean 值，它可以用来判断某个数据类型是否满足我们的要求。

所以结合上面的 typeof 和 instanceof 可以实现我们的 getType 函数。

示例代码：

```
<script>
  function getType(data) {
    // 先判断该数据是不是基础类型
    if (typeof data === 'object') {
      // 数组类型
      if (data instanceof Array) {
        return 'array'
      } else if (data instanceof Date) {
        return 'date'
      } else if (data instanceof Map) {
        return 'map'
      }
      // 还有超多的 else...if
    } else {
      return typeof data
    }
  }
  console.info(getType([1, 2, 3])); // array
</script>
```

上面的代码的确可以正确返回数据类型，但是小伙伴们可能发现我 else...if 没有写完，因为是在太多了，我也写不下去了！而且还有一个很大的问题，**instanceof 是无法判断 null 类型的！**

那么问题出在哪里呢?

原因就出在可变因素，你根本不知道传输的数据会以何种状态出现，不按照套路出牌！

总体看来，typeof+instanceof 方法有以下问题：

- 数据类型太多，要写超多的判断语句。
- 不可控，如果新出了数据类型，必须增加代码判断。
- 容易遗漏数据类型。

如果你能够写出上面的方法，面试官会觉得你掌握了数据类型这些知识，但是判断类型这一步做的还不够好。

3.Object.prototype.toString.call()

这个方法用来判断数据类型可以说是非常的完美了，它基本上能够判断出所有的数据类型，这也是我们最为推荐的一种。

官网描述：

> toString() 方法返回一个表示该对象的字符串。

详细介绍：

> 每个对象都有一个 toString() 方法，当该对象被表示为一个文本值时，或者一个对象以预期的字符串方式引用时自动调用。默认情况下，toString() 方法被每个 Object 对象继承。如果

> 此方法在自定义对象中未被覆盖，toString() 返回 "[object type]"，其中 type 是对象的类型。

通常情况下我们都是以 data.toString()的方式调用该方法，它可以返回调用该对象的字符串。但是大家注意详细介绍中的这句话：“如果此方法在自定义对象中未被覆盖，toString() 返回 "[object type]"”。

这里使用 call 是为了改变 toString()函数的内部 this 指向，当然也可以用 apply。

3.1 内部属性介绍

我们之所以能够借助 Object.prototype.toString.call()来判断数据类型，深层原因就是借助了toString()这个方法可以返回对象内部属性这个特征。

内部属性是在 JS 中主要用来判断属性特征的，比如判断属性是否可修改、删除、是否可枚举等等，内部属性通常就以[[]]的形式来进行标识。内部属性主要包括数据属性、访问器属性、类属性等等。所有 typeof 返回为"object"的对象都有[[class]]内部属性，可以通过Object.prototype.toString.call()获取。

Object.prototype.toString.call()返回示例：

```
console.info(Object.prototype.toString.call([])); // [object Array]
console.info(Object.prototype.toString.call("小猪课堂")); // [object String]
console.info(Object.prototype.toString.call({})); // [object Object]
```

对象常见的内部属性：

- [[Prototype]]：对象的原型
- [[Class]]：字符串对象的一种表现形式，可以通过 Object.prototype.toString.call()获取。
- [[Get]]——获得属性值的方法
- [[Put]]——设置属性值的方法
- [[CanPut]]——检查属性是否可写
- [[HasProperty]]——检查对象是否已经拥有该属性
- [[Delete]]——从对象删除该属性
- ……

了解了内部属性之后，我们就可以编写代码了。

3.2 getType函数

借助Object.prototype.toString.call()方法可以返回内部属性这个特征，我们就可以变向的获取数据的类型。

示例代码：

```
<script>
```

```
  function getType(data) {
    let originType = Object.prototype.toString.call(data); // 获取内部熟悉感
    let index = originType.indexOf(' '); // 以空格分割
    let type = originType.slice(index + 1, -1); // 截取
    return type.toLowerCase();
  }
  console.info(getType("小猪课堂")); // string
  console.info(getType(123)); // number
  console.info(getType(true)); // boolean
  console.info(getType(null)); // null
  console.info(getType(undefined)); // undefined
  console.info(getType({ name: "小猪课堂" })); // object
  console.info(getType([1,3,2])); // array
  console.info(getType(Promise.resolve())); // promise
  console.info(getType(new Set())); // set
  console.info(getType(new WeakMap())); // weakmap
  console.info(getType(new Date())); // date
  console.info(getType(() => {})); // function
  console.info(getType(new Map)); // map
  console.info(getType(BigInt(100))); // bigint
  console.info(getType(new RegExp(''))); // regexp
  console.info(getType((Symbol()))); // symbol
</script>
```

输出结果：

```
string
number
boolean
null
undefined
object
array
promise
set
weakmap
date
function
map
bigint
regexp
symbol
```

从上述输出结果来看，我们这个方法基本上可以正确判断所有的类型，而且最大的好处是如果后续新增了类型，那么这个函数也是可以判断的。

总结

判断数据类型是非常简单的，难得是想得周全。很多小伙伴知道如何判断类型，但是底层的依据可能比较模糊，这就是拉开差距的地方。