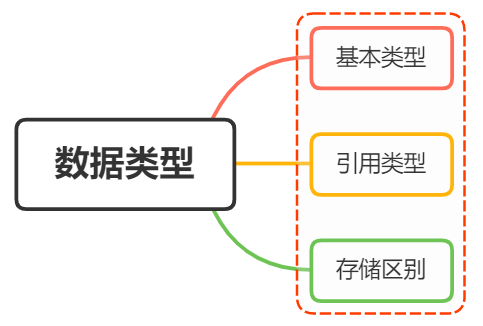
# 面试官：说说JavaScript中的数据类型？存储上的差别？



## 前言

在JavaScript中，我们可以分成两种类型：

* 基本类型
* 复杂类型

两种类型的区别是：存储位置不同

## 一、基本类型

基本类型主要为以下6种：

* Number
* String
* Boolean
* Undefined
* null
* symbol

### Number

数值最常见的整数类型格式则为十进制，还可以设置八进制（零开头）、十六进制（0x开头）

let intNum = 55 // 10进制的55  
let num1 = 070 // 8进制的56  
let hexNum1 = 0xA //16进制的10

浮点类型则在数值汇总必须包含小数点，还可通过科学计数法表示

let floatNum1 = 1.1;  
let floatNum2 = 0.1;  
let floatNum3 = .1; // 有效，但不推荐  
let floatNum = 3.125e7; // 等于 31250000

在数值类型中，存在一个特殊数值NaN，意为“不是数值”，用于表示本来要返回数值的操作失败了（而不是抛出错误）

console.log(0/0); // NaN  
console.log(-0/+0); // NaN

### Undefined

Undefined 类型只有一个值，就是特殊值 undefined。当使用 var或 let声明了变量但没有初始化时，就相当于给变量赋予了 undefined值

let message;  
console.log(message == undefined); // true

包含undefined 值的变量跟未定义变量是有区别的

let message; // 这个变量被声明了，只是值为 undefined  
  
console.log(message); // "undefined"  
console.log(age); // 没有声明过这个变量，报错

### String

字符串可以使用双引号（"）、单引号（'）或反引号（`）标示

let firstName = "John";  
let lastName = 'Jacob';  
let lastName = `Jingleheimerschmidt`

字符串是不可变的，意思是一旦创建，它们的值就不能变了

let lang = "Java";  
lang = lang + "Script"; // 先销毁再创建

### Null

Null类型同样只有一个值，即特殊值 null

逻辑上讲， null 值表示一个空对象指针，这也是给typeof传一个 null 会返回 "object" 的原因

let car = null;  
console.log(typeof car); // "object"

undefined 值是由 null值派生而来

console.log(null == undefined); // true

只要变量要保存对象，而当时又没有那个对象可保存，就可用 null来填充该变量

### Boolean

Boolean（布尔值）类型有两个字面值： true 和false

通过Boolean可以将其他类型的数据转化成布尔值

规则如下：

数据类型 转换为 true 的值 转换为 false 的值  
 String 非空字符串 ""   
 Number 非零数值（包括无穷值） 0 、 NaN   
 Object 任意对象 null  
Undefined N/A （不存在） undefined

### Symbol

Symbol （符号）是原始值，且符号实例是唯一、不可变的。符号的用途是确保对象属性使用唯一标识符，不会发生属性冲突的危险

let genericSymbol = Symbol();  
let otherGenericSymbol = Symbol();  
console.log(genericSymbol == otherGenericSymbol); // false  
  
let fooSymbol = Symbol('foo');  
let otherFooSymbol = Symbol('foo');  
console.log(fooSymbol == otherFooSymbol); // false

## 二、引用类型

复杂类型统称为Object，我们这里主要讲述下面三种：

* Object
* Array
* Function

### Object

创建object常用方式为对象字面量表示法，属性名可以是字符串或数值

let person = {  
 name: "Nicholas",  
 "age": 29,  
 5: true  
};

### Array

JavaScript数组是一组有序的数据，但跟其他语言不同的是，数组中每个槽位可以存储任意类型的数据。并且，数组也是动态大小的，会随着数据添加而自动增长

let colors = ["red", 2, {age: 20 }]  
colors.push(2)

### Function

函数实际上是对象，每个函数都是 Function类型的实例，而 Function也有属性和方法，跟其他引用类型一样

函数存在三种常见的表达方式：

* 函数声明

// 函数声明  
function sum (num1, num2) {  
 return num1 + num2;  
}

* 函数表达式

let sum = function(num1, num2) {  
 return num1 + num2;  
};

* 箭头函数

函数声明和函数表达式两种方式

let sum = (num1, num2) => {  
 return num1 + num2;  
};

### 其他引用类型

除了上述说的三种之外，还包括Date、RegExp、Map、Set等......

## 三、存储区别

基本数据类型和引用数据类型存储在内存中的位置不同：

* 基本数据类型存储在栈中
* 引用类型的对象存储于堆中

当我们把变量赋值给一个变量时，解析器首先要确认的就是这个值是基本类型值还是引用类型值

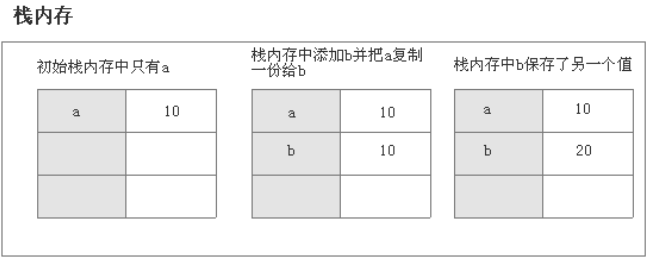
下面来举个例子

### 基本类型

let a = 10;  
let b = a; // 赋值操作  
b = 20;  
console.log(a); // 10值

a的值为一个基本类型，是存储在栈中，将a的值赋给b，虽然两个变量的值相等，但是两个变量保存了两个不同的内存地址

下图演示了基本类型赋值的过程：



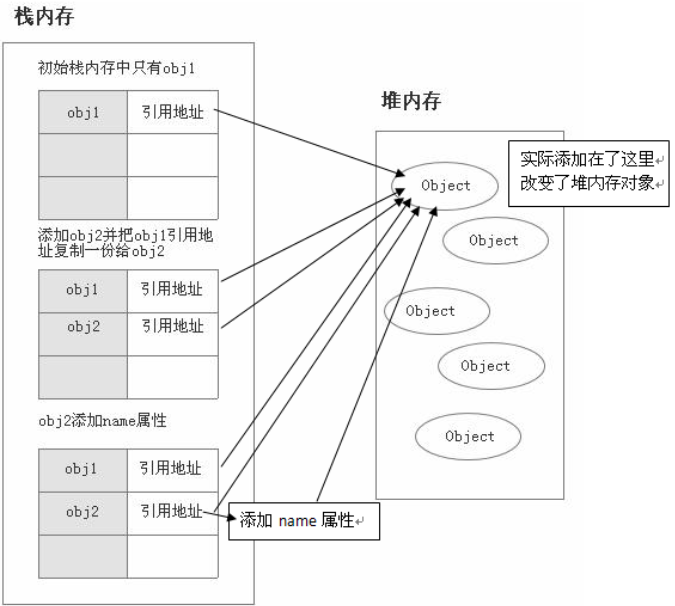
### 引用类型

var obj1 = {}  
var obj2 = obj1;  
obj2.name = "Xxx";  
console.log(obj1.name); // xxx

引用类型数据存放在内对内中，每个堆内存中有一个引用地址，该引用地址存放在栈中

obj1是一个引用类型，在赋值操作过程汇总，实际是将堆内存对象在栈内存的引用地址复制了一份给了obj2，实际上他们共同指向了同一个堆内存对象，所以更改obj2会对obj1产生影响

下图演示这个引用类型赋值过程



### 小结

* 声明变量时不同的内存地址分配：
* 简单类型的值存放在栈中，在栈中存放的是对应的值
* 引用类型对应的值存储在堆中，在栈中存放的是指向堆内存的地址
* 不同的类型数据导致赋值变量时的不同：
* 简单类型赋值，是生成相同的值，两个对象对应不同的地址
* 复杂类型赋值，是将保存对象的内存地址赋值给另一个变量。也就是两个变量指向堆内存中同一个对象