JavaScript 押题基础篇

JS 的数据类型有哪些?

纯记忆题，答案有 8 个词，建议背诵 10 次。

字符串、数字、布尔、undefined、null、大整数、符号、对象

string、number、boolean、undefined、null、bigint、symbol、object

> 提了就零分的答案有：数组、函数、日期。这些是类 class，不是类型 type

原型链是什么?

大概念题，答题思路为大概念化成小概念（分割），抽象化成具体（举例）。

我的答题思路如下：

哦，原型链涉及到的概念挺多的，**我举例说明一下吧。**

假设我们有一个**普通**对象 `x={}`，这个 `x` 会有一个隐藏属性，叫做 `__?????__`，这个属性会指向 `Object.prototype`，即

```
x.__?????__ === Object.prototype // 原型
```

此时，我们说 `x 的原型` 是 Object.prototype，或者说 Object.prototype 是 x 的原型。

而这个 `__?????__` 属性的唯一作用就是用来指向 x 的原型的。

如果没有 `__?????__` 属性，x 就不知道自己的原型是谁了。

> 为什么我用问号来表示？因为这样你不容易晕。

接下来我来说说原型链，我还是举例说明吧。

假设我们有一个数组对象 `a=[]`，这个 `a` 也会有一个隐藏属性，叫做 `__?????__`，这个属性会指向 `Array.prototype`，即

```
a.__?????__ === Array.prototype
```

此时，我们说 a 的原型是 `Array.prototype`，跟上面的 x 一样。但又有一点不一样，那就是 Array.prototype 也有一个隐藏属性 `__?????__`，指向 `Object.prototype`，即

```
// 用 x 表示 Array.prototype
x.__?????__ === Object.prototype
```

这样一来，a 就有两层原型：

1. a 的原型是 Array.prototype
2. a 的原型的原型是 Object.prototype

于是就通过隐藏属性 `__?????__` 形成了一个链条：

```
a ===> Array.prototype ===> Object.prototype
```

这就是原型链。

以上我对「原型链是什么」的回答。

怎么做：

看起来只要改写 x 的隐藏属性 `__?????__` 就可以改变 x 的原型（链）

```
x.__?????__ = 原型
```

但是这不是标准推荐的写法，为了设置 `x.__?????___`，推荐的写法是

```
const x = Object.create(原型)
// 或
const x = new 构造函数() // 会导致 x.__?????__ === 构造函数.prototype
```

> 没错，JS 就是这么别扭。

解决了什么问题：

在没有 Class 的情况下实现「继承」。以 `a ===> Array.prototype ===> Object.prototype` 为例，我们说：

1. a 是 Array 的实例，a 拥有 Array.prototype 里的属性
2. Array 继承了 Object（注意专业术语的使用）
3. a 是 Object 的间接实例，a 拥有 Object.prototype 里的属性

这样一来，a 就既拥有 Array.prototype 里的属性，又拥有 Object.prototype 里的属性。

优点：

简单、优雅。

缺点：

跟 class 相比，不支持私有属性。

怎么解决缺点：

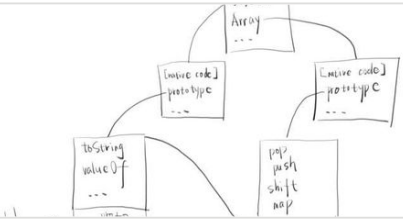
使用 class 呗。但 class 是 ES6 引入的，不被旧 IE 浏览器支持。

> 建议熟读这篇文章：

JS 中 __proto__ 和 prototype 存在的意义是什么？

你的 JS 代码还没运行的时候，JS 环境里已经有一个 window 对象了 window 对象有一个 Object 属性，window.Object 是一个函数

知 https://www.zhihu.com/question/56770432/answer/31534...



这段代码中的 this 是多少？

把判断依据背下来才能全对

```
var length = 4;
function callback() {
  console.log(this.length); // => 打印出什么?
}

const obj = {
  length: 5,
  method(callback) {
    callback();
  }
};

obj.method(callback, 1, 2);
```

> 建议熟读这篇文章：

this 的值到底是什么？一次说清楚

你可能遇到过这样的 JS 面试题： var obj = { foo: function(){ console.log(this) } } var bar = obj.foo obj.foo() // 打印出的 this 是

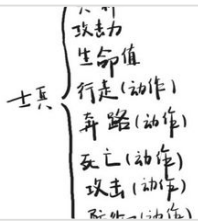
知 https://zhuanlan.zhihu.com/p/23804247

JS 的 new 做了什么？

记忆题，建议博客，甩链接

1. 创建临时对象/新对象
2. 绑定原型
3. 指定 this = 临时对象
4. 执行构造函数
5. 返回临时对象

> 建议熟读这篇文章：



JS 的立即执行函数是什么?

概念题，「是什么、怎么做、解决了什么问题、优点是、缺点是、怎么解决缺点」

是什么：

声明一个匿名函数，然后立即执行它。这种做法就是立即执行函数。

怎么做：

```
(function(){alert('我是匿名函数')} ())  // 用括号把整个表达式包起来
(function(){alert('我是匿名函数')}) ()  // 用括号把函数包起来
!function(){alert('我是匿名函数')}()    // 求反，我们不在意值是多少，只想通过语法
+function(){alert('我是匿名函数')}()
-function(){alert('我是匿名函数')}()
~function(){alert('我是匿名函数')}()
void function(){alert('我是匿名函数')}()
new function(){alert('我是匿名函数')}()
var x = function(){return '我是匿名函数'}()
```

上面每一行代码都是一个立即执行函数。（举例法）

解决了什么问题：

在 ES6 之前，只能通过它来「创建局部作用域」。

优点：

兼容性好。

缺点：

丑。为什么这么丑？看视频分析。

怎么解决缺点：

使用 ES6 的 block + let 语法，即

```
{
  let a = '我是局部变量啦'
  console.log(a) // 能读取 a
}
console.log(a) // 找不到 a
```

JS 的闭包是什么？怎么用？

概念题，「是什么、怎么做、解决了什么问题、优点是、缺点是、怎么解决缺点」

是什么

闭包是 JS 的一种**语法特性**。

> 闭包 = 函数 + 自由变量

对于一个函数来说，变量分为：全局变量、本地变量、自由变量

怎么做

```
let count
function add (){ // 访问了外部变量的函数
  count += 1
}
```

把上面代码放在「非全局环境」里，就是闭包。

> 注意，闭包不是 count，闭包也不是 add，闭包是 count + add 组成的整体。

怎么制造一个「非全局环境」呢？答案是立即执行函数：

```
const x = function (){
  var count
  function add (){ // 访问了外部变量的函数
    count += 1
  }
}()
```

但是这个代码什么用也没有，所以我们需要 `return add` ，即：

```
const add2 = function (){
  var count
  return function add (){ // 访问了外部变量的函数
    count += 1
  }
}()
```

此时 add2 其实就是 add，我们可以调用 add2

```
add2()
// 相当于
add()
// 相当于
count += 1
```

至此，我们就实现了一个完整的「闭包的应用」。

> 注意：闭包 ≠ 闭包的应用，但面试官问你「闭包」的时候，你一定要答「闭包的应用」，这是规矩。

解决了什么问题：

1. 避免污染全局环境。（因为用的是局部变量）
2. 提供对局部变量的间接访问。（因为只能 count += 1 不能 count -= 1）
3. 维持变量，使其不被垃圾回收。

优点：

简单，好用。

缺点：

闭包**使用不当**可能造成内存泄露。

注意，重点是「使用不当」，不是闭包。

「闭包造成内存泄露」这句话以讹传讹很多年了，曾经旧版本 IE 的 bug 导致的问题，居然被传成这样了。

举例说明：

```
function test() {
  var x = {name: 'x'};
  var y = {name: 'y', content: "-----这里很长，有一万三千五百个字符那么长----"
  return function fn() {
    return x;
  };
}

const myFn = test() // myFn 就是 fn 了
const myX = myFn() // myX 就是 x 了
// 请问，y 会消失吗?
```

对于一个正常的浏览器来说，y 会在**一段时间**后自动消失（被垃圾回收器给回收掉）。

但旧版本的 IE 并不是正常的浏览器，所以是 IE 的问题。

当然，你可以说

> 君子不立于危墙之下，我们应该尽量少用闭包，因为有些浏览器对闭包的支持不够好

但你不可以说「闭包造成内存泄露」。对吗?

怎么解决缺点：

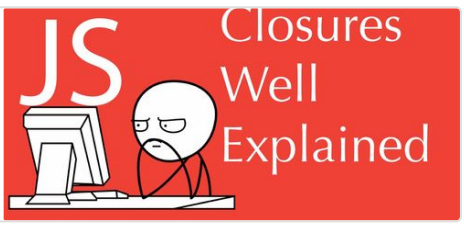
慎用，少用，不用。（我偏要用）

> 建议熟读这篇文章：

「每日一题」JS 中的闭包是什么?

大名鼎鼎的闭包！这一题终于来了，面试必问。 请用自己的话简述 由于评论里不停有人给出错误的答案，这里先声明： 如果你对

知 https://zhuanlan.zhihu.com/p/22486908



JS 如何实现类?

方法一：使用原型

```
function Dog(name){
  this.name = name
  this.legsNumber = 4
}
Dog.prototype.kind = '狗'
Dog.prototype.say = function(){
  console.log(`汪汪汪~ 我是${this.name}, 我有${this.legsNumber}条腿。`)
}
Dog.prototype.run = function(){
  console.log(`${this.legsNumber}条腿跑起来。`)
}
const d1 = new Dog('啸天') // Dog 函数就是一个类
d1.say()
```

请试着实现一个 Chicken 类，没 name 会 say 会 fly。

方法二：使用 class

```
class Dog {
  kind = '狗' // 等价于在 constructor 里写 this.kind = '狗'
  constructor(name) {
    this.name = name
    this.legsNumber = 4
    // 思考：kind 放在哪，放在哪都无法实现上面的一样的效果
  }
  say(){
    console.log(`汪汪汪~ 我是${this.name}, 我有${this.legsNumber}条腿。`)
  }
  run(){
    console.log(`${this.legsNumber}条腿跑起来。`)
  }
}
const d1 = new Dog('啸天')
d1.say()
```

请试着实现一个 Chicken 类，没 name 会 say 会 fly。

JS 如何实现继承?

方法一：使用原型链

```
function Animal(legsNumber){
  this.legsNumber = legsNumber
}
Animal.prototype.kind = '动物'

function Dog(name){
  this.name = name
  Animal.call(this, 4) // 关键代码1
}
Dog.prototype.__proto__ = Animal.prototype // 关键代码2，但这句代码被禁用了，怎

Dog.prototype.kind = '狗'
Dog.prototype.say = function(){
  console.log(`汪汪汪~ 我是${this.name}，我有${this.legsNumber}条腿。`)
}

const d1 = new Dog('啸天') // Dog 函数就是一个类
console.dir(d1)
```

如果面试官问被 ban 的代码如何替换，就说下面三句：

```
var f = function(){ }
f.prototype = Animal.prototype
Dog.prototype = new f()
```

方法二：使用 class

```
class Animal{
  constructor(legsNumber){
    this.legsNumber = legsNumber
  }
  run(){}
}
class Dog extends Animal{
  constructor(name) {
    super(4)
    this.name = name
  }
  say(){
    console.log(`汪汪汪~ 我是${this.name}, 我有${this.legsNumber}条腿。`)
  }
}
```