

# 36 个JS 面试题为你助力金九银十(面试必读)

原创 前端小智 大迂世界 2019-09-08 19:21

鸽子

宋冬野 - 安和桥北

来源: javapoint

译者: 前端小智

为了保证的可读性, 本文采用意译而非直译。

## 1.JS中`let`和`const`有什么用?

在现代js中, `let` & `const` 是创建变量的不同方式。在早期的js中, 咱们使用 `var` 关键字来创建变量。

`let` & `const` 关键字是在ES6版本中引入的, 其目的是在js中创建两种不同类型的变量, 一种是不可变的, 另一种是可变的。

**const:** 它用于创建一个不可变变量。不可变变量是指其值在程序的整个生命周期中永不改变的变量。

**let:** `let` 用于创建一个可变变量, 可变变量是像 `var` 这样的普通变量, 可以任意次数地更改。

## 2. JS 中的主要有哪几类错误

JS有三类的错误:


**加载时错误:** 加载web页面时出现的错误(如语法错误)称为加载时错误, 它会动态生成错误。

**运行时错误:** 由于滥用HTML语言中的命令而导致的错误。

**逻辑错误:** 这些错误是由于对具有不同操作的函数执行了错误的逻辑而导致的

## 3. 如何通过类别名获取 dom 元素

在 JS 中使用 `document.getElementsByClassName()` 方法来获取具有类名的元素。

getElementsByClassName()	
Method Name	getElementsByClassName
Syntax	document.getElementsByClassName('className')
Parameter	String (name of class)
Output	Array of HTMLCollection that have inputted className.  大迁世界

4. JS的作用域链是什么及其作用

一般情况下，变量取值到创建这个变量的函数的作用域中取值。但是如果在当前作用域中没有查到值，就会向上级作用域去查，直到查到**全局作用域**，这么一个查找过程形成的链条就叫做**作用域链**。

JS中的作用域链主要用于解析变量的值。如果没有这个，在不同的作用域内定义了许多变量，JS很难为变量选择某个值。

5. 解释JS中的`MUL`函数

**MUL** 表示数的简单乘法。在这种技术中，将一个值作为参数传递给一个函数，而该函数将返回另一个函数，将第二个值传递给该函数，然后重复继续。例如: `x*y*z` 可以表示为

```
function mul (x) {
  return function (y) {
    return function (z) {
      return x * y * z;
    }
  }
}
```

6.用纯JS编写一个程序来反转字符串

使用内置函数：内置函数 `reverse()` 直接反转字符串。

```
str="jQuery";
str = str.split("")
str = str.reverse()
str = str.join("")
alert(str);
```

首先将字符串拆分为数组，然后反转数组，最近将字符连接起来形成字符串。

**使用循环:**首先，计算字符串中的字符数，然后对原始字符串应用递减循环，该循环从最后一个字符开始，打印每个字符，直到count变为零。

7. JS中如何将页面重定向到另一个页面？

1. 使用 location.href: `window.location.href = "https://www.onlineinterviewquestions.com/"`
2. 使用 location.replace: `window.location.replace("https://www.onlineinterviewquestions.com/");`

## 8. 列出JS中的一些设计模式:

设计模式是软件设计中常见问题的通用可重用解决方案, 以下是一些设计模式是:

**创建模式:** 该模式抽象了对象实例化过程。

**结构型模式:** 这些模式处理不同的类和对象以提供新功能。

**行为模式:** 也称**发布-订阅模式**, 定义了一个被观察者和多个观察者的、一对多的对象关系。

**并行设计模式:** 这些模式处理多线程编程范例。

**架构设计模式:** 这些模式用于处理架构设计。

## 9. JS中的`Array.splice()`和`Array.slice()`方法有什么区别

话不多说, 来看第一个例子:

```
var arr=[0,1,2,3,4,5,6,7,8,9]; //设置一个数组
console.log(arr.slice(2,7)); //2,3,4,5,6
console.log(arr.splice(2,7)); //2,3,4,5,6,7,8
//由此我们简单推测数量两个函数参数的意义,
slice(start,end)第一个参数表示开始位置,第二个表示截取到的位置(不包含该位置)
splice(start,length)第一个参数开始位置,第二个参数截取长度
```

接着看第二个:

```
var x=y=[0,1,2,3,4,5,6,7,8,9]
console.log(x.slice(2,5)); //2,3,4
console.log(x); [0,1,2,3,4,5,6,7,8,9] 原数组并未改变
//接下来用同样方式测试splice
console.log(y.splice(2,5)); //2,3,4,5,6
console.log(y); // [0,1,7,8,9] 显示原数组中的数值被剔除掉了
```

`slice` 和 `splice` 虽然都是对于数组对象进行截取,但是二者还是存在明显区别,函数参数上 `slice` 和 `splice` 第一个参数都是截取开始位置, `slice` 第二个参数是截取的结束位置(不包含),而 `splice` 第二个参数(表示这个从开始位置截取的长度), `slice` 不会对原数组产生变化,而 `splice` 会直接剔除原数组中的截取数据!

## 10. 如何在JS中动态添加/删除对象的属性?

咱们可以使用 `object.property_name = value` 向对象添加属性, `delete object.property_name` 用于删除属性。

例如:

```
let user = new Object();
// adding a property
user.name='Anil';
user.age =25;
console.log(user);
delete user.age;
console.log(user);
```

## 11. 解释一下什么是 promise ?

**promise** 是js中的一个对象，用于生成可能在将来产生结果的值。值可以是已解析的值，也可以是说明为什么未解析该值的原因。

promise 可以有三种状态:

- pending: 初始状态，既不是成功也不是失败
- fulfilled: 意味着操作完全成功
- rejected: 意味着操作失败

一个等待状态的promise对象能够成功后返回一个值，也能失败后带回一个错误  
当这两种情况发生的时候，处理函数会排队执行通过then方法会被调用

## 12. 数组去重复的方法有哪些

### 1.使用 **set**

```
1 function uniquearray(array) {
2   let unique_array= Array.from(set(array))
3   return unique_array;
4 }
```

### 2.使用 **filter**

```
function unique_array (arr) {
  let unique_array = arr.filter(function(elem, index, self) {
    return index == self.indexOf(elem);
  })
  return unique_array;
}

console.log(unique_array(array_with_duplicates));
```

### 3.使用 **for** 循环

```
Array dups_names = ['Ron', 'Pal', 'Fred', 'Rongo', 'Ron'];
function dups_array(dups_names) {
  let unique = {};
  names.forEach(function(i) {
    If (!unique[i]) {
      unique[i] = true;
    }
  });
}
```

```
});  
return Object.keys(unique);} // Ron, Pal, Fred, Rongo  
Dups_array(names);
```

### 13. undefined, null 和 undeclared 有什么区别?

1.null表示"没有对象", 即该处不应该有值, 转为数值时为0。典型用法是:

(1) 作为函数的参数, 表示该函数的参数不是对象。

(2) 作为对象原型链的终点。

2.undefined表示"缺少值", 就是此处应该有一个值, 但是还没有定义, 转为数值时为NaN。典型用法是:

(1) 变量被声明了, 但没有赋值时, 就等于undefined。

(2) 调用函数时, 应该提供的参数没有提供, 该参数等于undefined。

(3) 对象没有赋值的属性, 该属性的值为undefined。

(4) 函数没有返回值时, 默认返回undefined。

3.undeclared: js语法错误, 没有申明直接使用, js无法找到对应的上下文。

### 14. 列出JS基本和非基本数据类型之间的一些区别?

1.目前JS中有 6 种基本数据类型: Undefined、Null、Boolean、Number、Symbol 和 String。还有1种复杂的数据类型——Object, Object 本质上是由一组无序的名值对组成的。Object、Array 和 Function 则属于引用类型。

2.基本数据类型是不可变的, 而非基本数据类型是可变的。

3.基本数据类型是不可变的, 因为它们一旦创建就无法更改, 但非基本数据类型刚可更改, 意味着一旦创建了对象, 就可以更改它。

4.将基本数据类型与其值进行比较, 这意味着如果两个值具有相同的数据类型并具有相同的值, 那么它们是严格相等的。

5.非基本数据类型不与值进行比较。例如, 如果两个对象具有相同的属性和值, 则它们严格不相等。

### 15. 如何在现有函数中添加新属性

只需给现有函数赋值, 就可以很容易地在现有函数中添加新属性。例如, 现有一个对象 person, 通过下面的代码来为 person 添加新的属性:

```
person.country= "India";
```

### 16. JS中的深拷贝与浅拷贝的区别?

- 深拷贝递归地复制新对象中的所有值或属性，而拷贝只复制引用。
- 在深拷贝中，新对象中的更改不会影响原始对象，而在浅拷贝中，新对象中的更改，原始对象中也会跟着改。
- 在深拷贝中，原始对象不与新对象共享相同的属性，而在浅拷贝中，它们具有相同的属性。

## 17. 如何在JavaScript中每x秒调用一个函数

在JS中，咱们使用函数 `setInterval()` 在每 `x` 秒内调用函数。如：

```
setInterval(function (){ alert("Hello"); }, 3000);
```

## 18. 解释一下JS的展开操作符？

展开运算符在需要多个参数/变量/元素的位置展开表达式，它用三个点（`...`）。如：

```
var mid = [3, 4];

var newarray = [1, 2, ...mid, 5, 6];

console.log(newarray);

// [1, 2, 3, 4, 5, 6]
```

## 19. JS中的宿主对象与原生对象有何不同？

**宿主对象**:这些是运行环境提供的对象。这意味着它们在不同的环境下是不同的。例如，浏览器包含像 `windows` 这样的对象，但是Node.js环境提供像 `Node List` 这样的对象。

**原生对象**:这些是JS中的内置对象。它们也被称为全局对象，因为如果使用JS，内置对象不受是运行环境影响。

## 20. 解释JS中的高阶函数？

高阶函数是JS函数式编程的最佳特性。它是以函数为参数并返回函数作为结果的函数。一些内置的高阶函数是 `map`、`filter`、`reduce` 等等。

## 21. JS 中 == 和 === 区别是什么？

1、对于 `string`，`number` 等基础类型，`==` 和 `===` 有区别

- 1) 不同类型间比较，`==` 之比较“转化成同一类型后的值”看“值”是否相等，`===` 如果类型不同，其结果就是不等。
- 2) 同类型比较，直接进行“值”比较，两者结果一样。

2、对于 `Array`，`Object` 等高级类型，`==` 和 `===` 没有区别

进行“指针地址”比较。

3、基础类型与高级类型，`==` 和 `===` 有区别

- 1) 对于 `==`，将高级转化为基础类型，进行“值”比较。
- 2) 因为类型不同，`===` 结果为 `false`。

## 22. JS中的匿名函数是什么？

匿名函数：就是没有函数名的函数，如：

```
(function(x, y){  
    alert(x + y);  
})(2, 3);
```

这里创建了一个匿名函数(在第一个括号内)，第二个括号用于调用该匿名函数，并传入参数。

## 23. 是否可以在JS中执行301重定向？

JS完全运行在客户端上。`301` 是服务器作为响应发送的响应代码。因此，在JS中不可能执行 `301` 重定向。

## 24. 解释JS中的事件冒泡和事件捕获

**事件捕获和冒泡：**在HTML DOM API中，有两种事件传播方法，它们决定了接收事件的顺序。两种方法是事件冒泡和事件捕获。第一个方法事件冒泡将事件指向其预期的目标，第二个方法称为事件捕获，其中事件向下到达元素。

### 事件捕获

捕获过程很少被使用，但是当它被使用时，它被证明是非常有用的。这个过程也称为 **滴流模式**。在这个过程中，事件首先由最外层的元素捕获，然后传播到最内部的元素。例如：

```
<div>  
  <ul>  
    <li></li>  
  </ul>  
</div>
```

从上面的示例中，假设单击事件发生在 `li` 元素中，在这种情况下，捕获事件将首先处理 `div`，然后处理 `ul`，最后命中目标元素 `li`。

### 事件冒泡

冒泡的工作原理与冒泡类似，事件由最内部的元素处理，然后传播到外部元素。

```
<div>  
  <ul>  
    <li></li>  
  </ul>  
</div>
```

从上面的例子中，假设 `click` 事件确实发生在冒泡模型中的 `li` 元素中，该事件将首先由 `li` 处理，然后由 `ul` 处理，最后由 `div` 元素处理。

## 24. 如何将文件的所有导出作为一个对象？

`import * as objectname from './file.js'` 用于将所有导出的成员导入为对象。可以使用对象的点 (`.`) 运算符来访问导出的变量或方法，如：

```
objectname.member1;
objectname.member2;
objectname.memberfunc();
```

## 25. 解释一下什么是箭头函数？

箭头函数是在 `es6` 或更高版本中编写函数表达式的简明方法。箭头函数不能用作构造函数，也不支持 `this`，`arguments`，`super` 或 `new.target` 关键字，它最适合非方法函数。通常，箭头函数看起来像 `const function_name = () => {}`。

```
const greet={()=>{console.log('hello')}};
greet();
```

## 25 解释 JS 中的函数提升

JS允许将声明移动到顶部的默认行为称为**提升**。JS中创建函数的两种方法是函数声明和函数表达式。

### 函数声明

具有特定参数的函数称为函数声明，在JS中创建变量称为声明。如：

```
hoisted(); // logs "foo"

function hoisted() {
  console.log('foo');
}
```

### 函数表达式

当使用表达式创建函数时，称为函数表达式。如：

```
notHoisted(); // TypeError: notHoisted is not a function

var notHoisted = function() {
  console.log('bar');
};
```

## 26. module.exports 和 exports 之间有什么区别？

`module` 和 `exports` 是 `Node.js` 给每个 `js` 文件内置的两个对象。可以通过 `console.log(module)` 和 `console.log(exports)` 打印出来。如果你在 `main.js` 中写入下面两行，然后运行 `$ node main.js`：



```
console.log(exports);//输出: {}  
console.log(module);//输出: Module {..., exports: {}, ...} (注: ...代表省略了其他一些属性
```



从打印咱们可以看出, `module.exports` 和 `exports` 一开始都是一个空对象 `{}`, 实际上, 这两个对象指向同一块内存。这也就是说 `module.exports` 和 `exports` 是等价的 (有个前提: 不去改变它们指向的内存地址)。

例如: `exports.age = 18` 和 `module.export.age = 18`, 这两种写法是一致的 (都相当于给最初的空对象 `{}` 添加了一个属性, 通过 `require` 得到的就是 `{age: 18}`)。

## 27. import 和 exports 是什么?

`import` 和 `exports` 帮助咱们编写模块化的JS代码。使用 `import` 和 `exports`, 咱们可以将代码分割成多个文件。 `import` 只允许获取文件的某些特定变量或方法。可以导入模块导出的方法或变量。

```
//index.js  
  
import name,age from './person';  
  
console.log(name);  
console.log(age);  
  
//person.js  
  
let name = 'Sharad', occupation='developer', age =26;  
  
export { name, age};
```

## 28. 列出一些单元测试框架

下面是一些最流行的JS单元测试框架:

- Unit.js
- Jasmine
- Karma
- Chai
- AVA
- Mocha
- JSUnit
- QUnit
- Jest

## 29. JS中有哪些不同类型的弹出框可用

在JS中有三种类型的弹出框可用, 分别是:

- Alert
- Confirm
- Prompt

### 30. 如何将 JS 日期转换为ISO标准

`toISOString()` 方法用于将js日期转换为ISO标准。它使用ISO标准将js Date对象转换为字符串。如：

```
var date = new Date();
var n = date.toISOString();
console.log(n);
// YYYY-MM-DDTHH:mm:ss.sssZ
```

### 31. 如何在JS中克隆对象

`Object.assign()` 方法用于在JS中克隆对象。如：

```
var x = {myProp: "value"};
var y = Object.assign({}, x);
```

### 32. 如何在JS中编码和解码 URL

`encodeURIComponent()` 函数用于在JS中对URL进行编码。它将 `url` 字符串作为参数并返回编码的字符串。

**注意：** `encodeURIComponent()` 不会编码类似这样字符： `/ ? : @ & = + $ #`，如果需要编码这些字符，请使用 `encodeURIComponent()`。用法：

```
var uri = "my profile.php?name=sammer&occupation=päntiNG";
var encoded_uri = encodeURIComponent(uri);
```

`decodeURI()` 函数用于解码js中的URL。它将编码的 `url` 字符串作为参数并返回已解码的字符串，用法：

```
var uri = "my profile.php?name=sammer&occupation=päntiNG";
var encoded_uri = encodeURIComponent(uri);
decodeURI(encoded_uri);
```

### 33. BOM 和 DOM 的关系

**BOM**全称 **Browser Object Model**，即浏览器对象模型，主要处理浏览器窗口和框架。

**DOM** 全称 **Document Object Model**，即文档对象模型，是 HTML 和XML 的应用程序接口（API），遵循 W3C 的标准，所有浏览器公共遵守的标准。

JS是通过访问**BOM**（Browser Object Model）对象来访问、控制、修改客户端(浏览器)，由于**BOM**的 `window` 包含了 `document`，`window` 对象的属性和方法是直接可以使用而且被感知的，因此可以直接使用 `window` 对象的 `document` 属性，通过document属性就可以访问、检索、修改XHTML文档内容与结构。因为 `document` 对象又是 **DOM** 的根节点。

可以说，**BOM** 包含了 **DOM** (对象)，浏览器提供出来给予访问的是 **BOM** 对象，从 **BOM** 对象再访问到 **DOM** 对象，从而js可以操作浏览器以及浏览器读取到的文档。

### 34. JS中的 ``substr()``和``substring()``函数有什么区别

`substr()` 函数的形式为 `substr(startIndex,length)`。它从 `startIndex` 返回子字符串并返回 `length` 个字符数。

```
var s = "hello";  
( s.substr(1,4) == "ello" ) // true
```

`substring()` 函数的形式为 `substring(startIndex,endIndex)`。它返回从 `startIndex` 到 `endIndex - 1` 的子字符串。

```
var s = "hello";  
( s.substring(1,4) == "ell" ) // true
```

### 35. 解释一下 "use strict" ?

“`use strict`”是Es5中引入的js指令。使用“`use strict`”指令的目的是强制执行严格模式下的代码。在严格模式下，咱们不能在不声明变量的情况下使用变量。早期版本的js忽略了“`use strict`”。

### 36. 解释 JS 事件委托模型?

在JS中，有一些很酷的东西。其中之一是委托模型。当捕获和冒泡时，允许函数在一个特定的时间实现一个处理程序到多个元素，这称为事件委托。事件委托允许将事件侦听器添加到父节点而不是指定的节点。这个特定的侦听器分析冒泡事件，以找到子元素上的匹配项。

原文：<https://www.javatpoint.com/javascript-tutorial>

代码部署后可能存在的BUG没法实时知道，事后为了解决这些BUG，花了大量的时间进行log 调试，这边顺便给大家推荐一个好用的BUG监控工具 Fundebug。

### 交流

我是小智，公众号「大迂世界」作者，对前端技术保持学习爱好者。我会经常分享自己所学所看的干货，在进阶的路上，共勉！

Vue 和 React 的优点分别是什么? (知乎大佬解答)

JS中，如何提高展开运算符的性能

JS 如何创建、读取和删除cookie

50 个JS 必须懂的面试题为你助力金九银十

面试 10

面试 · 目录

上一篇

50 个JS 必须懂的面试题为你助力金九银十

下一篇

常见的三个 JS 面试题

喜欢此内容的人还喜欢

90后程序员辞职搞灰产，一年获利超700万，结局很刑！

大迂世界



100 个鲜为人知的 CSS 技巧汇总整理合集

大迂世界



Vite 4.3 为何性能爆表？

大迂世界

