JavaScript81道面试题（https://github.com/minsion）

面试题 1. 在 JavaScript中，为什么说函数是第一类对象?

推荐指数：★★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

第一类函数即 JavaScript中的函数。这通常意味着这些函数可以作为参数传递给其他函数，作为其他函数的值返回，分配给变量，也可以存储在数据结构中

面试题 2. JavaScript函数声明与函数表达式的区别?

推荐指数：★★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

在 JavaScript中，在向执行环境中加载数据时，解析器对函数声明和函数表达式并非是一视同仁的。解析器会首先读取函数声明，并使它在执行任何代码之前可用（可以访问）。至于函数表达式，则必须等到解析器执行到它所在的代码行，才会真正解析和执行它

面试题 3. JavaScript如何判断一个对象是否属于某个类?

推荐指数：★★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

使用 instanceof关键字，判断一个对象是否是类的实例化对象；使用 constructor属性，判断一个对象是否是类的构造函数

面试题 4. JavaScript中有一个函数，执行直接对象查找时，它始终不会查找原型，这个函数是什么?

推荐指数：★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

hasOwnProperty

面试题 5. 简述documen.wrte和 innerHTML的区别是什么?

推荐指数：★★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

document.wite重绘整个页面；innerHTML可以重绘页面的一部

面试题 6. JavaScript中读取文件的方法是什么?

推荐指数：★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

可以通过如下方式读取服务器中的文件内容。

```
function readAjaxEile（url） {
//创建xhr
var xhr =new XMLHttpRequest（）；
/监听状态
```

```
xhr. onreadystatechange=function（）{
//监听状态值是4
if（xhr. readystate == 4 && xhr. status = = =200）{
console. log（xhr. responseText）
}
//打开请求
xhr.open（'GET'， url, true）
//发送数据
xhr, send（null）
}
```

可以通过如下方式读取本地计算机中的内容。

```
function readInputFile（id） {
var file= document. getElementById（id）. files[0];
//实例化 FileReader
var reader=new FileReader（）；
//读取文件
reader. readAsText （file）
//监听返回
reader, onload= function （data） {
console. log （data, this .result）
}
}
```

面试题 7. JavaScript如何分配对象属性?

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

将属性分配给对象的方式与赋值给变量的方式相同。例如，表单对象的操作值以下列方式分配给" submit"：document.form. action=" submit'"

面试题 8. 请简述JavaScript语句的基本规范？

推荐指数：★★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

基本规范如下：
（1）不要在同一行声明多个变量。
（2）应使用==/！==来比较true/ false或者数值。
（3）使用对象字面量替代 new Array这种形式。
（4）不要使用全局函数。
（5） switch语句必须带有 default分支。
（6）函数不应该有时有返回值，有时没有返回值。
（7）for循环必须使用大括号括起来。
（8）if语句必须使用大括号括起来。
（9）for-in循环中的变量应该使用war关键字明确限定的作用域，从而避免作用域污染。

面试题 9. 列出不同浏览器中关于 JavaScript兼容性的两个常见问题？

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

（1）事件绑定兼容性问题。
IE8以下的浏览器不支持用 add Event Listener来绑定事件，使用 attachement可以解决这个问题

（2） stopPropagation兼容性问题
IE8以下的浏览器不支持用 e .stopPropagation()来阻止事件传播，使用 e .return Value =false可以解决这个问题

面试题 10. JavaScript中常用的逻辑运算符有哪些?

推荐指数：★★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

"and”（&&）运算符、“or”（||）运算符和"not"（！）运算符，它们可以在 JavaScript中使用

面试题 11. 如何将 JavaScript代码分解成几行?

推荐指数：★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

M：在字符串语句中可以通过在第一行末尾使用反斜杠“\”来完成，例如， document. write（"This is \a program"）。

如果不是在字符串语句中更改为新行，那么 JavaScript会忽略行中的断点下面的代码是完美的，但并不建议这样做，因为阻碍了调试。

```
var x=l, y=2,
z=
X+y;
```

面试题 12. 解释 JavaScript中定时器的工作，并说明使用定时器的缺点 ？

推荐指数：★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

定时器用于在设定的时间执行一段代码，或者在给定的时间间隔内重复该代码这通过使用函数 setTimeout、setInterval和 clearInterva来完成。

setTimeout（ function, delay）函数用于启动在所属延迟之后调用特定功能的定时器。

setInterval（ function,dlay）函数用于在提到的延迟中重复执行给定的功能，只有在取消时才停止。

clearInterval（id）函数指示定时器停止定时器在一个线程内运行，因此事件可能需要排队等待执行

面试题 13. JavaScript语言中ViewState和 SessionState有什么区别?

推荐指数：★★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

View State特定于会话中的页面； SessionState特定于可在Web应用程序中的所有页面上访问的用户特定数据

面试题 14. 如何在 JavaScript中将base字符串转换为 integer?

推荐指数：★★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

parseInt()函数解析一个字符串参数，并返回一个指定基数的整数。 parseInt()将要转换的字符串作为其第一个参数，第二个参数是给定字符串的转换进制基数。

为了将4F（基数16）转换为整数，可以使用代码 parrent（"4F"，16）。

面试题 15. JavaScript如何检测客户端机器上的操作系统?

推荐指数：★★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

为了检测客户端机器上的操作系统，应使用 navigator.app Version字符串（属性）

面试题 16. 解释JavaScript void（0）的作用是什么?

推荐指数：★★★★★ 试题难度： 高难 试题类型： 原理题

试题回答参考思路：

void操作符使表达式的运算结果返回 undefined。
void（0）用于防止页面刷新，并在调用时传递参数“0”。
void（0）用于调用另一种方法而不刷新页面

面试题 17. 如何强制页面加载 JavaScript中的其他页面?

推荐指数：★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

必须插入以下代码才能达到预期效果。

面试题 18. JavaScript转义字符的作用 ?

推荐指数：★★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

当使用特殊字符（如单引号、双引号、撇号和&符号）时，将使用转义字符（反斜杠）。在字符前放置反斜杠，使其显示。

下面给出两个示例
document. write"I m a "good"boy "
document. write"I m a\"good\"boy"

面试题 19. 解释JavaScript中， datatypes的两个基本组是什么?

推荐指数：★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

两个基本组是原始类型和引用类型。

原始类型包括数字和布尔类型。引用类型包括更复杂的类型，如字符串和日期

面试题 20. JavaScript中不同类型的错误有几种?

推荐指数：★★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

有3种类型的错误。
Load time errors，该错误发生于加载网页时，例如出现语法错误等状况，称为加载时间错误，并且会动态生成错误。
Run time errors，由于在HTML语言中滥用命令而导致的错误。
Logical errors，这是由于在具有不同操作的函数上执行了错误逻辑而发生的错误

面试题 21. JavaScript中，push方法的作用是什么?

推荐指数：★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

push方法用于将一个或多个元素添加或附加到数组的末尾。使用这种方法，可通过传递多个参数来附加多个元素

面试题 22. JavaScript中， unshift方法的作用是什么?

推荐指数：★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

unshift方法就像在数组开头工作的push方法。该方法用于将一个或多个元素添加到数组的开头

面试题 23. JavaScript如何获得 CheckBox状态?

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

alert（ document getElement Byld（'checkbox1'） .checked;
如果 CheckBox选中，此警告将返回TRUE

面试题 24. 请区分解释 window. onload和 onDocumentReady?

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

在载入页面的所有信息之前，不运行 window. onload。这导致在执行任何代码之前会出现延迟。

window.onDocumentReady在加载DOM之后加载代码。这允许代码更早地执行（早于 window. onload）

面试题 25. 简述JavaScript什么是构造函数？它与普通函数有什么区别?

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

构造函数是一种特殊的方法，主要用来创建对象时初始化对象，经常与new运算符一起使用，创建对象的语句中构造函数的名称必须与类名完全相同。
与普通函数相比，区别如下
（1）构造函数只能由new关键字调用
（2）构造函数可以创建实例化对象
（3）构造函数是类的标志

面试题 26. 请说出 JavaScript无阻塞加载的具体方式?

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

将脚本放在底部。
放在head中，以保证在 JavaScript代码加载前，能加载出正常显示的页面。
< script>标签放在前。
在阻塞脚本中，因为每个< script标签下载时都会阻塞页面的解析，所以限制页面的< script>总数也可以改善性能。
它适用于内嵌脚本和外链脚本。
在非阻塞脚本中，等页面完成加载后，再加载 Javascript代码。也就是说，在window.onload事件发出后开始加载代码。
其中， defer属性支持IE4和 Fierfox3.5及更高版本的浏览器。通过动态脚本元素，文档对象模型（DOM）允许使用JavaScript动态创建HTML的几乎全部文档内容，代码如下。

此技术的重点在于，无论在何处启动下载，即使在head里，文件的下载和运行都不会阻塞其他页面的处理过程。

面试题 27. 请解释一下JavaScript事件冒泡机制？

推荐指数：★★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

在一个对象上触发某类事件（比如onclick事件）时，如果此对象定义了此事件的处理程序，那么此事件就会调用这个处理程序；如果没有定义此事件处理程序或者事件返回true，那么这个事件会向这个对象的父级对象传播，从里到外，直至它被处理（父级对象所有同类事件都将被激活），或者它到达了对象层次的最顶层，即 document对象（有些浏览器中是 window）。
冒泡型事件触发顺序是指从最特定的事件目标（触发事件对象）到最不特定的事件目标对象（ document对象）。
JavaScript冒泡机制是指如果某元素定义了事件A，如 click事件，如果触发了事件之后，没有阻止冒泡事件，那么该事件将向父级元素传播，触发父类的 click事件

面试题 28. 请说出JavaScript阻止事件冒泡的方法?

推荐指数：★★★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

阻止事件冒泡的方法，包括兼容IE浏览器（e.cancle Bubble）和标准浏览器（e. stopProgation）。下面给出一段示例代码。

```
function stopBubble（e）{
var evt = e || window.event;
evt.stopPropagation? evt.stopPropagation():(evt. cancelBubble=true）；
```

面试题 29. 简述JavaScript什么是事件流?

推荐指数：★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

事件流是网页元素接收事件的顺序，"DOM2级事件"规定的事件流包括三个阶段：事件捕获阶段、处于目标阶段、事件冒泡阶段。 首先发生的事件捕获，为截获事件提供机会。然后是实际的目标接受事件。最后一个阶段是时间冒泡阶段，可以在这个阶段对事件做出响应。 虽然捕获阶段在规范中规定不允许响应事件，但是实际上还是会执行，所以有两次机会获取到目标对象

我是父元素 我是子元素

当容器元素及嵌套元素，即在 捕获阶段 又在 冒泡阶段 调用事件处理程序时：事件按DOM事件流的顺序执行事件处理程序：
父级捕获
子级冒泡
子级捕获
父级冒泡
且当事件处于目标阶段时，事件调用顺序决定于绑定事件的书写顺序，按上面的例子为，先调用冒泡阶段的事件处理程序，再调用捕获阶段的事件处理程序。依次alert出“子集冒泡”，“子集捕获”。

IE 兼容
attchEvent('on' + type, handler)
detachEvent('on' + type, handler)

面试题 30. JavaScript如何清除一个定时器?

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

清除定时器使用的方法是：window. clearInterval( )。清除循环定时器使用的方法x window. clearTimeout( )

面试题 31. JavaScript节点类型是有哪些？如何判断当前节点类型?

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

节点有以下类型。
元素节点
属性节点；
文本节点
注释节点；
文档节点。
用 nodeObject.nodeType判断节点类型。其中， nodObject为DOM节点（节点对象）
该属性返回用数字表示节点的类型，例如，元素节点返回1，属性节点返回2

面试题 32. 使用 typeof bar===" object"可以确定bar是不是对象的潜在陷阱，如何避免这个陷阱?

推荐指数：★★★★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

尽管 typeof bar===" object"可以是检查bar是不是对象的可靠方法，但是在 JavaScript中也认为nul是对象。因此，下面的代码将在控制台中输出true（而不是 false）。
var bar null;

console. log（typeof bar – object"）；//true
只要清楚了这一点，同时检查bar是否为nul，就可以很容易地避免问题console. log（（bar ！ == null） &&（typeof bar ===object"））；// false
要答全问题，还有其他两件事情值得注意。
首先，上述解决方案将返回 false，当bar是一个函数的时候，在大多数情况下，这是期望结果，但当你想对函数返回true的时候，可以这样修改上面的解决方案。
console. log（（bar ！ – null） &&（（typeof bar a"object "）ll （typeof bar ==="function"））
其次，当bar是一个数组（例如，当 var bar=口）的时候，上述解决方案将返回true。在大多数情况下，这是期望的结果，因为数组是真正的对象，但当你想对数组返回 false时，可以这样修改上面的解决方案。
console. log（（bar ！ == null） &&（typeof bar ==="object"）&&（Object. prototype tostr ing call（bar） ！ =="[object Array]"））
或者在ES5规范中输入如下内容。
console.log（（bar ！ == null） &&（typeof bar ===object"）&&（！ Array. isArray（bar））

面试题 33. 封装 JavaScript源文件的全部内容到一个函数块有什么意义?

推荐指数：★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

这是一个越来越普遍的做法，已被许多流行的 JavaScript库（ jQuery、 Node. js等）采用。这种技术创建了一个围绕文件全部内容的闭包。最重要的是，创建了一个私有的命名空间，有助于避免不同 JavaScript模块和库之间的命名冲突。
这种技术的另一个特点是，允许把一个易于引用的（更短的）别名用于全局变
量。例如， jQuery插件中， jQuery允许你使用 jQuery. no Conflict()，来禁止$引用到jQuery命名空间。在完成这项工作之后，利用这种闭包技术，代码仍然可以使用$，如下所示。
（function（s）{/* jQuery plugin code referencing s */}）（jQuery）

面试题 34. 说明下列代码将输出什么，并解释原因?

```
console. log（0. 1 +0.2）；
console.1og（0.1+0.2==0.3）；
```

推荐指数：★★★★ 试题难度： 高难 试题类型： 编程题

试题回答参考思路：

它会输出以下内容。
0.30000000000000004
false
原因如下。
十进制数0.1对应二进制数0.00011001100110011...（循环0011）。
十进制数0.2对应二进制数0.0011001100110011...（循环001）
两者相加得到达以下结果
0.00011001100110011001100110011001100110011001100110011001100110011001...
+0.00110011001l0011001100110011001100110011001100110011001100110011
=0.01001100110011001100110011001100110011001100110011001100110011001100...
转换成十进制之后得到0.300000000000004。

面试题 35. 简述如何调试 JavaScript代码?

推荐指数：★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

调试方式如下
（1） Javascript断点调试。
断点可以让程序在需要的地方中断，从而方便程序员分析。也可以在一次调试中设置断点，下一次只须让程序自动运

行到设置断点的位置，便可在上次设置断点的位置中断，这极大地方便了调试，同时节省了时间。
JavaScript断点调试，即是在浏览器开发者工具中为 JavaScript代码添加断点，让JavaScript执行到某一特定位置停住，方便开发者对该处代码段进行分析和调试。
（2） debugger断点调试。
通过在代码中添加“ debugger”语句，当代码执行到该语句的时候就会自动插入断点。
（3）DOM断点调试DOM断点就是在DOM元素上添加断点，进而达到调试的目的

面试题 36. 解释promises中的race method是什么意思?

推荐指数： ★★★★ 试题难度： 高难 试题类型： 原理题 ▸

试题回答参考思路：

Promise.race() 方法返回首先解决或拒绝的Promise。
让我们用一个例子来证明这一点，第二个promise 比第一个promise 更快地解决：

```
let p1 = new Promise(function(resolve, reject) {
setTimeout(resolve, 500, ‘the first promise’);
}
);
let p2 = new Promise(function(resolve, reject) {
setTimeout(resolve, 100, ‘the second promise’);
}
);
Promise.race([p1, p2]).then(function(value) {
console.log(value, ‘was faster’);
}
);
```

输出：

```
the se
cond promise was faster
```

面试题 37. 简述promise.all() 方法有什么作用?

推荐指数： ★★★★★ 试题难度： 初级 试题类型： 原理题 ▸

试题回答参考思路：

Promise.all 是一个将一系列promise 作为输入的promise 。它在以下情况下得到解决：

要么所有的输入promise 都得到解决。
或者其中任何一个被拒绝。
例如，promise.all 等待所有这三个promise 完成：

```
var prom1 = new Promise((resolve, reject) => {
setTimeout(() => {
resolve(“Yay!”);
}, 1000);
});
var prom2 = Promise.resolve(10);
var prom3 = 100;
Promise.all([prom1, prom2, prom3]).then(values => {
console.log(values);
});
```

一秒后输出：

```
[“Yay”, 10, 100]
```

面试题 38. 解释什么是JavaScript时间死区?

推荐指数：★★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

时间死区意味着变量无法访问，即使它已经在范围内。
让我们首先看一下当您尝试在未初始化的情况下将变量记录到控制台时会发生什么：
console.log(x);
var x = “Yay”;
输出：
undefined
您可能希望这会导致错误，但它会打印出 undefined。
发生这种情况是因为hoisting，这意味着所有声明都被移动到范围的顶部。由于hoisting，上面的代码在幕后表现如下：
var x;
console.log(x);
x = “Yay”;
这里 undefined 自动分配给顶部的变量，这使得在定义它之前使用它成为可能。
但是让我们看看当我们使用 let 而不是 var 做同样的事情时会发生什么：
console.log(x);
let x = 10;
输出：
error: Uncaught ReferenceError: Cannot access ‘x’ before initialization
发生这种情况是因为 let 的hoisting方式与 var 不同。当一个 let 变量被hoisting时，它不会变成未定义的。相反，它是不可访问的，或者在时间死区中，直到它被分配一个值。

面试题 39. 解释文档加载与 DOMContentLoaded?

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

DOMContentLoaded 事件在加载和解析 HTML 文档时触发，它不等待资产（例如样式表和图像）。
文档加载事件仅在加载整个页面（包括所有资产）后触发。
例如，下面是如何使用 DOMContentLoaded 在 DOM 完全加载时发出通知：
window.addEventListener(‘DOMContentLoaded’, (event) => {
console.log(‘DOM is now loaded!’);
}
);
这是一个示例，说明如何在加载特定页面时添加侦听器：
window.addEventListener(‘load’, (event) => {
console.log(‘The page is now loaded!’);
}
);

面试题 40. JavaScript 是区分大小写的语言吗?

推荐指数：★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

JavaScript 是一种区分大小写的语言。
关键字、变量、函数名等需要大写一致。
为了演示，这段代码有效

```
let i = 1;
while(i < 2) {
console.log(i);
i++;
```

```
}
```

但这并没有，因为 while 被大写了，尽管它不应该大写。

```
let i = 1;
WHILE(i < 2) {
console.log(i);
i++;
}
```

面试题 41. JavaScript 中有多少个线程?

推荐指数：★★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

JavaScript 使用单线程。它不允许编写解释器可以在多个线程或进程中并行运行的代码。
这意味着它按顺序执行代码并且必须执行完一段代码才能移动到下一段代码。
当您在网页上显示警报时，就是一个很好的例子。警报弹出后，您无法与页面交互，直到警报关闭。
alert(“Hello there!”);

面试题 42. 简述JavaScript什么是正则表达式?

推荐指数：★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

正则表达式，也称为 regex 或 regexp，是一组构成搜索模式的字符。它是一种模式匹配工具，常用于 JavaScript 和其他编程语言。
例如，让我们使用正则表达式从字符串中查找任意数字：

```
var regex = /\d+/g;
var string = “You have 100 seconds time to run”;
var matches = string.match(regex);
console.log(matches);
```

输出是所有匹配项的数组：

```
[100]
```

例如，正则表达式可用于在大型文本文件中搜索电子邮件或电话号码

面试题 43. 解释JavaScript调试代码时断点机制 ?

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

断点允许您在 JavaScript 代码中查找错误。
当执行调试器语句并出现调试器窗口时，您可以在代码中设置断点。
在断点处，JavaScript 停止执行并让您检查值和范围以解决可能的问题

面试题 44. 解释JavaScript能链接条件运算符吗?

推荐指数：★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

对的，这是可能的。有时它很有用，因为它可以使代码更易于理解。
让我们看一个例子：

```
function example() {
if (condition1) {
```

```
return value1;
} else if (condition2) {
return value2;
} else if (condition3) {
return value3;
} else {
return value4;
}
}
// Shorthand for the above function
function example() {
return condition1 ? value1
: condition2 ? value2
: condition3 ? value3
: value4;
}
```

面试题 45. JavaScript freeze() 方法有什么作用?

推荐指数: ★★★★ 试题难度: 中级 试题类型: 原理题

试题回答参考思路:

freeze() 方法冻结一个对象。它使对象不可变。
冻结对象后，无法向其添加新属性。
例如:

```
const item = { name: “test” };
Object.freeze(item);
item.name = “Something else”; // Error
```

面试题 46. JavaScript如何获取对象的键列表?

推荐指数: ★★★★ 试题难度: 初级 试题类型: 原理题

试题回答参考思路:

使用 Object.keys() 方法。
例如:

```
const student = {
name: ‘Mike’,
gender: ‘male’,
age: 23
}
;
console.log(Object.keys(student));
```

输出:

```
[“name”, “gender”, “age”]
```

面试题 47. 简述JavaScript 的原始数据类型有哪些?

推荐指数: ★★★★ 试题难度: 初级 试题类型: 原理题

试题回答参考思路:

原始数据类型具有原始值。JavaScript 中有七种不同的原始数据类型:

string——单词。例如，“John”。
number— 数值。例如，12。

boolean——真或假。例如，true。
null — 没有值。例如，let x = null;
undefined——声明变量但没有值的类型。例如，当以这种方式创建变量 x 时，let x; , x 变得undefined。
bigint — 用于表示大于 2^53−1 的整数的对象。例如 BigInt(121031393454720292)
symbol — 用于创建独特符号的内置对象。例如，let sym1 = Symbol('test')

面试题 48. 如何在页面加载后执行 JavaScript 代码?

推荐指数: ★★ 试题难度: 初级 试题类型: 原理题

试题回答参考思路:

您可以通过三种方式执行此操作:
将属性 window.onload 设置为在页面加载后执行的函数:
window.onload = function ...
将属性 document.onload 设置为在页面加载后执行的函数:
document.onload = function ...
将 HTML 属性的 onload 属性设置为 JS 函数:

面试题 49. JavaScript NoScript标签有什么作用?

推荐指数: ★★★★ 试题难度: 中级 试题类型: 原理题

试题回答参考思路:

Noscript 标签用于检测和响应禁用 JavaScript 的浏览器。
您可以使用 noscript 标签来执行一段通知用户的代码。
例如，您的 HTML 页面可以有一个像这样的 noscript 标签:

面试题 50. 简述Javascript gb2312转utf8 ?

推荐指数: ★★★ 试题难度: 中级 试题类型: 原理题

试题回答参考思路:

JavaScript中GB2312转UTF-8的实现方法
在JavaScript中，可以使用编码库或API来进行GB2312转UTF-8的转换。下面以示例代码的方式，介绍一下具体实现方法。
第一种实现方式：利用文本编码库
可以使用text-encoding库中的TextDecoder和TextEncoder对象来进行GB2312转UTF-8的编码转换。具体的实现步骤如下:

```
// 定义要转换的字符串
var gb2312Str = '这是一段测试字符串';
// 将gb2312编码的字符串转换为Uint8Array数组
var gb2312Array = new Uint8Array(gb2312Str.length);
for (var i = 0; i < gb2312Str.length; ++i) {
gb2312Array[i] = gb2312Str.charCodeAt(i);
}
// 利用TextDecoder对象将Uint8Array数组转换为UTF-8编码的字符串
var utf8Str = new TextDecoder('gb2312').decode(gb2312Array);
console.log(utf8Str);
// 输出：这是一段测试字符串
```

在这个例子中，首先将gb2312的字符串转换为Uint8Array数组，然后使用TextDecoder对象将其转换为UTF-8编码的字符串。
第二种实现方式：利用iconv-lite库

iconv-lite是一个可以在NodeJS和浏览器中使用的编码库。它支持多种编码方式的字符串转换，包括GB2312和UTF-8。具体的实现步骤如下：

```
// 导入 iconv-lite 库
const iconv = require('iconv-lite');
// 定义要转换的字符串
var gb2312Str = '这是一段测试字符串';
// 利用iconv-lite库将GB2312编码字符串转换为UTF-8编码的字符串
var utf8Str = iconv.decode(Buffer.from(gb2312Str), 'gb2312');
console.log(utf8Str);
// 输出：这是一段测试字符串
```

在这个例子中，我们通过iconv-lite库首先将GB2312字符串转换为Buffer对象，然后使用decode方法将其转换为UTF-8编码的字符串

面试题 51. Promise 的 finally 怎么实现的?

推荐指数：★★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

Promise.prototype.finally 方法是 ES2018 引入的一个方法，用于在 Promise 执行结束后无论成功与否都会执行的操作。在实际应用中，finally 方法通常用于释放资源、清理代码或更新 UI 界面等操作。

以下是一个简单的实现方式：

```
Promise.prototype.finally = function(callback) {
const P = this.constructor;
return this.then(
value => P.resolve(callback()).then(() => value),
reason => P.resolve(callback()).then(() => { throw reason })
);
}
```

我们定义了一个名为 finally 的函数，它使用了 Promise 原型链的方式实现了 finally 方法。该函数接收一个回调函数作为参数，并返回一个新的 Promise 对象。如果原始 Promise 成功，则会先调用 callback 函数，然后将结果传递给下一个 Promise；如果失败，则会先调用 callback 函数，然后将错误信息抛出。

可以看到，在实现中，我们首先通过 this.constructor 获取当前 Promise 实例的构造函数，然后分别处理 Promise 的 resolved 和 rejected 状态的情况。在 resolved 状态时，我们先调用 callback 函数，然后将结果传递给新创建的 Promise 对象；在 rejected 状态时，我们也是先调用 callback 函数，然后将错误信息抛出。

这样，我们就完成了 Promise.prototype.finally 方法的实现。

面试题 52. JavaScript 创建“原生”（native）方法？

给字符串对象定义一个repeatify功能。当传入一个整数n时，它会返回重复n次字符串的结果。例如：
console.log('hello'.repeatify(3));
应打印 hellohellohello

推荐指数：★★★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

一个可能的实现如下所示：

```
1: String.prototype.repeatify = String.prototype.repeatify || function(times) {
2: var str = '';
3: for (var i = 0; i < times; i++) {
4: str += this;
5:
}
```

```
6: return str;
7:
}
;
```

现在的问题测试开发者有关JavaScript继承和prototype的知识点。这也验证了开发者是否知道该如果扩展内置对象（尽管这不应该做的）。

这里的另一个要点是，你要知道如何不覆盖可能已经定义的功能。通过测试一下该功能定义之前并不存在：

```
String.prototype.repeatify = String.prototype.repeatify || function(times) {
/* code here */
}
;
```

当你被要求做好JavaScript函数兼容时这种技术特别有用

面试题 53. 列举JavaScript中的数据类型检测方案？

推荐指数：★★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

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

优点：能够快速区分基本数据类型
缺点：不能将Object、Array和Null区分，都返回object

```
2.instanceof
console.log(1 instanceof Number); // false
console.log(true instanceof Boolean); // false
console.log('str' instanceof String); // false
console.log([] instanceof Array); // true
console.log(function(){} instanceof Function); // true
console.log({} instanceof Object); // true
```

优点：能够区分Array、Object和Function，适合用于判断自定义的类实例对象
缺点：Number，Boolean，String基本数据类型不能判断

```
3.Object.prototype.toString.call()
var toString = Object.prototype.toString;
console.log(toString.call(1)); //[object Number]
console.log(toString.call(true)); //[object Boolean]
console.log(toString.call('mc')); //[object String]
console.log(toString.call([])); //[object Array]
console.log(toString.call({})); //[object Object]
console.log(toString.call(function(){})); //[object Function]
console.log(toString.call(undefined)); //[object Undefined]
console.log(toString.call(null)); //[object Null]
```

优点：精准判断数据类型
缺点：写法繁琐不容易记，推荐进行封装后使用

instanceof 的作用

用于判断一个引用类型是否属于某构造函数；
还可以在继承关系中用来判断一个实例是否属于它的父类型。

instanceof 和 typeof 的区别：
typeof在对值类型number、string、boolean 、null 、 undefined、 以及引用类型的function的反应是精准的；
但是，对于对象{ } 、数组[ ] 、null 都会返回object
为了弥补这一点，instanceof 从原型的角度，来判断某引用属于哪个构造函数，从而判定它的数据类型。

var && let && const

ES6之前创建变量用的是var,之后创建变量用的是let/const

三者区别：

var定义的变量，没有块的概念，可以跨块访问，不能跨函数访问。
let定义的变量，只能在块作用域里访问，不能跨块访问，也不能跨函数访问。
const用来定义常量，使用时必须初始化(即必须赋值)，只能在块作用域里访问，且不能修改。
var可以先使用，后声明，因为存在变量提升；let必须先声明后使用。
var是允许在相同作用域内重复声明同一个变量的，而let与const不允许这一现象。
在全局上下文中，基于let声明的全局变量和全局对象GO（window）没有任何关系 ；
var声明的变量会和GO有映射关系；
会产生暂时性死区：
暂时性死区是浏览器的bug：检测一个未被声明的变量类型时，不会报错，会返回undefined
如：console.log(typeof a) //undefined
而：console.log(typeof a)//未声明之前不能使用
let a

let /const/function会把当前所在的大括号(除函数之外)作为一个全新的块级上下文，应用这个机制，在开发项目的时候，遇到循环事件绑定等类似的需求，无需再自己构建闭包来存储，只要基于let的块作用特征即可解决

面试题 54. 请问什么是JavaScript箭头函数以及特性？

推荐指数：★★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

箭头函数没有自己的this，会捕获其所在的上下文的this值，作为自己的this值
箭头函数没有constructor，是匿名函数，不能作为构造函数，不能通过new 调用；
没有new.target 属性。在通过new运算符被初始化的函数或构造方法中，new.target返回一个指向构造方法或函数的引用。在普通的函数调用中，new.target 的值是undefined
箭头函数不绑定Arguments 对象。取而代之用rest参数...解决。由于 箭头函数没有自己的this指针，通过 call() 或 apply() 方法调用一个函数时，只能传递参数（不能绑定this），他们的第一个参数会被忽略。（这种现象对于bind方法同样成立）
箭头函数通过 call() 或 apply() 方法调用一个函数时，只传入了一个参数，对 this 并没有影响。
箭头函数没有原型属性 Fn.prototype 值为 undefined
箭头函数不能当做Generator函数,不能使用yield关键字
参考：箭头函数与普通函数的区别

面试题 55. 简述浏览器 JavaScript EventLoop 事件循环？

推荐指数：★★★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

JS是单线程的，为了防止一个函数执行时间过长阻塞后面的代码，所以会先将同步代码压入执行栈中，依次执行，将异步代码推入异步队列，异步队列又分为宏任务队列和微任务队列，因为宏任务队列的执行时间较长，所以微任务队列要优先于宏任务队列。微任务队列的代表就是，Promise.then，MutationObserver，宏任务的话就是setImmediate setTimeout setInterval

JS运行的环境。一般为浏览器或者Node。 在浏览器环境中，有JS 引擎线程和渲染线程，且两个线程互斥。 Node环境中，只有JS 线程。 不同环境执行机制有差异，不同任务进入不同Event Queue队列。 当主程结束，先执行准备好微任务，然后再执行准备好的宏任务，一个轮询结束

面试题 56. 简述Javascript 建造者模式 ?

推荐指数： ★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。
使用场景
相同的方法，不同的执行顺序，产生不同的事件结果时，可以采用建造者模式。
多个部件或零件，都可以装配到一个对象中，但是产生的运行结果又不相同时，则可以使用该模式。
产品类非常复杂，或者产品类中的调用顺序不同产生了不同的效能，这个时候使用建造者模式非常合适。
结构
Product 产品类：通常是实现了模板方法模式，也就是有模板方法和基本方法。
Builder 抽象建造者：规范产品的组建，一般是由子类实现。
ConcreteBuilder 具体建造者：实现抽象类定义的所有方法，并且返回一个组建好的对象。
Director 导演类：负责安排已有模块的顺序，然后告诉 Builder 开始建造。
代码示例

```
public class ConcreteProduct extends Builder {
private Product product = new Product();
//设置产品零件
public void setPart() {
/*
* 产品类内的逻辑处理
*/
}
//组建一个产品
public Product buildProduct() {
return product;
}
}
```

面试题 57. 简述创建对象有几种方法?

推荐指数： ★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

1、字面量对象 // 默认这个对象的原型链指向object

```
var o1 = {name: '01'};
```

2、通过new Object声明一个对象

```
var o11 = new Object({name: '011'});
```

3、使用显式的构造函数创建对象
o2的构造函数是M
o2这个普通函数，是M这个构造函数的实例

```
function sum (a, b, c) {
console.log(a + b + c);
}
sum(1, 2, 3); // 6
```

4、object.create()

原型、构造函数、实例、原型链
1、Object.prototype属性是整个原型链的顶端
2、原型链通过prototype原型和proto属性来查找的.
3、所有的引用类型（数组、对象、函数），都具有对象特性，即可自由扩展属性（除了“null”以外）。
4、所有的引用类型（数组、对象、函数），都有一个proto属性,属性值是一个普通的对象（null除外）。
5、所有的函数，都有prototype属性，属性值也是一个普通的对象。
6、所有的引用类型（数组、对象、函数），proto属性指向它的构造函数prototype属性值
7、实例本身的属性和方法如果没有找到，就会去找原型对象的属性和方法。如果在某一级找到了，就会停止查找，并返回结果

面试题 58. 简述实际开发中闭包的应用？

推荐指数：★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

闭包实际应用中主要用于封装变量，收敛权限
function isFirstLoad() {
var _list = []; // 有_的变量说明是私有变量，函数内部使用的
return function(id) {
12，如何理解js的单线程?
只有一个线程，同一时间只能做一件事情。
13，js为什么是单线程的?
避免dom渲染的冲突
1、浏览器需要渲染dom
2、js可以修改dom结构
3、js执行的时候，浏览器dom渲染会暂停
4、两段js也不能同时执行（都修改dom就冲突了）
5、webworder支持多线程，但是不能访问dom
14，同步和异步的区别是什么？分别举一个同步和异步的例子?
1、同步会阻塞代码执行，而异步不会。
2、alert是同步，setTimeout是异步。
同步：指一个进程在执行某个请求的时候，若该请求需要一段时间才能返回信息，那么这个进程将会一直等待下去，直到收到返回信息才继续执行下去；
异步：指进程不需要一直等下去，而是继续执行下面的操作，不管其他进程的状态。当有消息返回时系统会通知进程进行处理，这样可以提高执行的效率。
if (_list.indexOf(id) >=0) {
// 也可用includes
return false;
} else {
_list.push(id);
return true;
}
}
}
// 使用
var firstLoad = isFirstLoad();
console.log(firstLoad(10)); // true
console.log(firstLoad(10)); // false
console.log(firstLoad(20)); // true
// 你在isFirstLoad函数外面，根本不可能修改掉_list的值

面试题 59. 简述Javascript为什么是单线程的？

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

避免dom渲染的冲突
1、浏览器需要渲染dom
2、js可以修改dom结构
3、js执行的时候，浏览器dom渲染会暂停
4、两段js也不能同时执行（都修改dom就冲突了）
5、webworder支持多线程，但是不能访问dom

面试题 60. 简述JS判断数据类型的方法有哪四种?(列出四种即可）？

推荐指数：★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

在 ECMAScript 规范中，共定义了 7 种数据类型，分为 基本类型 和 引用类型 两大类，如下所示：
基本类型：String、Number、Boolean、Symbol、Undefined、Null
引用类型：Object
基本类型也称为简单类型，由于其占据空间固定，是简单的数据段，为了便于提升变量查询速度，将其存储在栈中，即按值访问。
引用类型也称为复杂类型，由于其值的大小会改变，所以不能将其存放在栈中，否则会降低变量查询速度，因此，其值存储在堆(heap)中，而存储在变量处的值，是一个指针，指向存储对象的内存处，即按址访问。引用类型除 Object 外，还包括 Function 、Array、RegExp、Date 等等。
鉴于 ECMAScript 是松散类型的，因此需要有一种手段来检测给定变量的数据类型。对于这个问题，JavaScript 也提供了多种方法，但遗憾的是，不同的方法得到的结果参差不齐。
下面介绍常用的4种方法，并对各个方法存在的问题进行简单的分析。

1、typeof
typeof 是一个操作符，其右侧跟一个一元表达式，并返回这个表达式的数据类型。返回的结果用该类型的字符串(全小写字母)形式表示，包括以下 7 种：number、boolean、symbol、string、object、undefined、function 等
typeof ''; // string 有效
typeof 1; // number 有效
typeof Symbol(); // symbol 有效
typeof true; //boolean 有效
typeof undefined; //undefined 有效
typeof null; //object 无效
typeof [] ; //object 无效
typeof new Function(); // function 有效
typeof new Date(); //object 无效
typeof new RegExp(); //object 无效

有些时候，typeof 操作符会返回一些令人迷惑但技术上却正确的值：
对于基本类型，除 null 以外，均可以返回正确的结果。
对于引用类型，除 function 以外，一律返回 object 类型。
对于 null ，返回 object 类型。
对于 function 返回 function 类型。
其中，null 有属于自己的数据类型 Null ， 引用类型中的 数组、日期、正则 也都有属于自己的具体类型，而 typeof 对于这些类型的处理，只返回了处于其原型链最顶端的 Object 类型，没有错，但不是我们想要的结果。

2、instanceof
instanceof 是用来判断 A 是否为 B 的实例，表达式为：A instanceof B，如果 A 是 B 的实例，则返回true,否则返回 false。 在这里需要特别注意的是：instanceof 检测的是原型，我们用一段伪代码来模拟其内部执行过程：
instanceof (A,B) = {
var L = A.__proto__;

```
var R = B.prototype;
if(L === R) {
// A的内部属性 __proto__ 指向 B 的原型对象
return true;
}
return false;
}
```

从上述过程可以看出，当 A 的 proto 指向 B 的 prototype 时，就认为 A 就是 B 的实例，我们再来看几个例子：

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

我们发现，虽然 instanceof 能够判断出 [ ] 是Array的实例，但它认为 [ ] 也是Object的实例，为什么呢?

我们来分析一下 [ ]、Array、Object 三者之间的关系：

从 instanceof 能够判断出 [ ].proto 指向 Array.prototype，而 Array.prototype.proto 又指向了 Object.prototype，最终 Object.prototype.proto 指向了null，标志着原型链的结束。因此，[]、Array、Object 就在内部形成了一条原型链

4、toString

toString() 是 Object 的原型方法，调用该方法，默认返回当前对象的 [[Class]] 。这是一个内部属性，其格式为 [object Xxx] ，其中 Xxx 就是对象的类型。

对于 Object 对象，直接调用 toString() 就能返回 [object Object] 。而对于其他对象，则需要通过 call / apply 来调用才能返回正确的类型信息。

```
Object.prototype.toString.call('') ; // [object String]
Object.prototype.toString.call(1) ; // [object Number]
Object.prototype.toString.call(true) ; // [object Boolean]
Object.prototype.toString.call(Symbol()); //[object Symbol]
Object.prototype.toString.call(undefined) ; // [object Undefined]
Object.prototype.toString.call(null) ; // [object Null]
Object.prototype.toString.call(new Function()) ; // [object Function]
Object.prototype.toString.call(new Date()) ; // [object Date]
Object.prototype.toString.call([]) ; // [object Array]
Object.prototype.toString.call(new RegExp()) ; // [object RegExp]
Object.prototype.toString.call(new Error()) ; // [object Error]
Object.prototype.toString.call(document) ; // [object HTMLDocument]
Object.prototype.toString.call(window) ; //[object global] window 是全局对象 global 的引用
```

面试题 61. 简述JS按照存储方式区分为哪些类型，并描述其特点？？

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

1、存储在栈中：值类型。

2、存储在堆中：引用类型

引用类型的”数据“存储在堆中，

引用类型”指向堆中的数据的指针“存储在栈中。

面试题 62. 简述Javascript数组怎么去重有哪些？（方法）？

推荐指数：★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

1、使用数组方法indexOf来判断

```
function sele(arr){
var temp = [];
for( var i = 0 ; i < arr.length ; i++ ){
if( temp.indexOf( arr[ i ] ) == -1 ){
temp.push( arr[ i ] );
}
}
return temp;
}
var arr = ['aa', 'bb', 'cc', '', 1, 0, '1', 1, 'bb', null, undefined, null];
console.log(sele(arr));
```

2、使用数组方法indexOf第二种方法 IE8一不兼容

```
function sele( arr ) {
var temp = [];
for( var i = 0 ; i < arr.length ; i++ ){
if( arr.indexOf( arr[ i ] ) == i ){
temp.push( arr[ i ] );
}
}
return temp;
}
```

3、循环

```
function sele( arr ) {
var temp = [];
for( var i = 0 ; i < arr.length ; i++ ){
for( var j = i + 1 ; j < arr.length ; j++ ){
if( arr[ i ] === arr[ j ] ){
j = ++i;
}
}
temp.push( arr[ i ] );
}return temp;
}
```

4、

```
function unique3(array) {
var result = [];
var hash = {};
for(var i=0; i var key = (typeof array[i]) + array[i];
if(!hash[key]) {
result.push(array[i]);
hash[key] = true;
}
}
return result;
}
var arr = ['aa', 'bb', 'cc', '', 1, 0, '1', 1, 'bb', null, undefined, null];
console.log(unique3(arr));
```

5、使用es6中includes方法

```
function sele( arr ) {
var temp = [];
arr.forEach(( v ) => {
temp.includes( v ) || temp.push( v );
})
return temp;
}
```

6、es6的set

```
function unique(array){
return Array.from(new Set(array));
}
// 或者写成（建议写法）
const unique = arr => [...new Set(arr)]
// 也可以是
const unique = arr => {return [...new Set(arr)]}
var arr = ['aa', , '', 1, 0, '1', 1, , null, undefined, null];
console.log(unique(arr));
```

面试题 63. 简述实现Javascript一句话数组去重？

推荐指数：★★★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

new Set()：Set本身是一个构造函数，用来生成Set数据结构

```
const unique = arr => [...new Set(arr)]
var arr = ['aa', , '', 1, 0, '1', 1, , null, undefined, null];
console.log(unique(arr));
```

面试题 64. 简述Js通用事件绑定/ 编写一个通用的事件监听函数？

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

```
function bindEvent(elem, type, selector, fn) {
if (fn == null) {
fn = selector; selector = null;
}
elem.addEventListner(type, function(e) {
var target;
if (selector) {
target = e.target;
if (target.matches(selector)) {
fn.call(target, e);
}
} else {
fn(e);
}
})
}// 使用代理
var div1 = document.getElementById('div1');
bindEvent(div1, 'click', 'a', function(e) {
console.log(this.innerHTML);
```

```
});
// 不使用代理
var a = document.getElementById('a1');
bindEvent(div1, 'click', function(e) {
console.log(a.innerHTML);
})
```

1、代理的好处
（1）代码简洁
（2）减少浏览器内存占用
2、事件冒泡
事件冒泡的应用：代理

面试题 65. 简述.JS 整数是怎么表示的？

推荐指数：★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

通过 Number 类型来表示，遵循 IEEE754 标准，通过 64 位来表示一个数字，（1 + 11 + 52），最大安全数字是 Math.pow(2, 53) – 1，对于 16 位十进制。（符号位 + 指数位 + 小数部分有效位）

面试题 66. 简述Number() 的存储空间是多大？如果后台发送了一个超过最大自己的数字怎么办？

推荐指数：★★★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

Math.pow(2, 53)，53 为有效数字，会发生截断，等于 JS 能支持的最大数字。

面试题 67. new 一个构造函数，如果函数返回 return {} 、 return null， return 1， return true 会发生什么情况？

推荐指数：★★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

如果函数返回一个对象，那么new 这个函数调用返回这个函数的返回对象，否则返回 new 创建的新对象

面试题 68. 如何判断一个对象是不是空对象？

推荐指数：★★★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

Object.keys(obj).length === 0
手写题：在线编程，getUrlParams(url,key)；就是很简单的获取url的某个参数的问题，但要考虑边界情况，多个返回值等等

面试题 69. 简述Set、Map、WeakSet 和 WeakMap 的区别？

推荐指数：★★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

Set——对象允许你存储任何类型的唯一值，无论是原始值或者是对象引用WeakSet——成员都是对象；成员都是弱引用，可以被垃圾回收机制回收，可以用来保存 DOM 节点，不容易造成内存泄漏；
Map——本质上是键值对的集合，类似集合；可以遍历，方法很多，可以跟各种数据格式转换。

WeakMap——只接受对象最为键名（null 除外），不接受其他类型的值作为键名；键名是弱引用，键值可以是任意的，键名所指向的对象可以被垃圾回收，此时键名是无效的；不能遍历，方法有 get、set、has、delete

面试题 70. 请分别用深度优先思想和广度优先思想实现一个拷贝函数?

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

```
let _toString = Object.prototype.toStringlet map = {
array: 'Array',
object: 'Object',
function: 'Function',
string: 'String',
null: 'Null',
undefined: 'Undefined',
boolean: 'Boolean',
number: 'Number'}let getType = (item) => {
return _toString.call(item).slice(8, = (item, type)
=> {
return && map[type]
```

面试题 71. Promise 构造函数是同步执行还是异步执行，那么 then 方法呢？

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

```
const promise = new Promise((resolve, reject) => {
console.log(1)
resolve()
console.log(2)})promise.then(() => {
console.log(3)})console.log(4)
```

执行结果是：1243，promise 构造函数是同步执行的，then 方法是异步执行的

面试题 72. 简述JavaScript中的常见编码方案？

推荐指数：★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

在编写J ava S c ri p t代码时，我们常常需要处理URL s，这时候理解J ava S c ri p t中的e s c a p e， e n c o d e U R I 和 e n c o d e U R I C o m p o n e n t 函数就显得尤为重要。这些函数用于将特殊字符转化为能在URL中安全传输的形式。本文将详细介绍这三个函数的用法和区别，以帮助你更准确的处理URL编码问题。

1. escape函数
首先，我们来了解一下 e s c a p e 函数。这是一个老旧的函数，现在已经不再推荐使用，因为它不能处理所有的Un i c o d e字符。 e s c a p e 函数会将传入的字符串转化为十六进制的e s c a p e序列，这样的序列以 %开头。
然而，这个函数只能正确处理ASCI I字符（字符代码小于等于2 5 5的字符）。对于ASCI I字符代码大于2 5 5的字符， e s c a p e 函数会先将其转化为Un i c o d e转义序列（例如， \ u 2 0A C ），然后再对这个转义序列进行编码。这种处理方式会导致一些问题。比如，对于欧元符号（ € ），它的Un i c o d e代码是 2 0A C， e s c a p e 函数会将其转化为 % u 2 0A C ，而不是正确的 % E2 % 8 2 % A C 。
因此，我们不应该再使用 e s c a p e 函数来处理URL编码。

2. encodeURI函数
接下来，我们来看看 e n c o d e U R I 函数。这个函数用于编码完整的URL。它会将非法的URL字符转化为各自的十六进制表示，以 % 开头。

然而， e n c o d e U R I 函数并不会对所有的字符进行编码。一些在URL中有特殊含义的字符，例如 / ， ： ， # 等，以及ASCI I字母，数字和一些符号（ - _ . ! ~ * ' ( ) ），不会被 e n c o d e U R I 函数编码。这是因为这些字符在URL中是合法的，可以直接使用。
下面是一个 e n c o d e U R I 函数的例子：
const url =
'https://example.com/Hello World!';
console.log(encodeURI(url)); // https://example.com/Hello%20World!
在这个例子中， e n c o d e U R I 函数将空格字符编码为 % 2 0 ，因为空格在URL中是不合法的。而其他的字符，如 / 和 ： 等，都没有被编码。

3. encodeURIComponent函数
最后，我们来看看 e n c o d e U R I C o m p o n e n t 函数。这个函数用于编码URL的组成部分，比如查询参数。它会将所有非法的URL字符以及一些有特殊含义的字符（如 / ， ： ， # 等）转化为各自的十六进制表示。
这意味着 e n c o d e U R I C o m p o n e n t 函数会对更多的字符进行编码。在大多数情况下，我们都应该使用` e n c o d eURICo m p o n e n t
`函数来编码URL的组成部分。
下面是一个 e n c o d e U R I C o m p o n e n t 函数的例子：
const query =
'/Hello World!';
console.log(encodeURIComponent(query)); // %2FHello%20World%21
在这个例子中， e n c o d e U R I C o m p o n e n t 函数将 / 和空格字符都编码了，因为这些字符在URL的查询参数中都是不合法的。

4. 总结
总的来说，当我们需要编码完整的URL时，应该使用 e n c o d e U R I 函数；而当我们需要编码URL的组成部分，比如查询参数，应该使用 e n c o d e U R I C o m p o n e n t 函数。不再推荐使用 e s c a p e 函数，因为它不能正确处理所有的字符。
理解和掌握这些函数的用法和区别对于正确处理URL编码问题来说是非常重要的。

面试题 73. 简述JavaScript修饰器？

推荐指数： ★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

在J ava S c ri p t中，修饰器（De c o r a t o r）是一种特殊的语法，用于修改类、方法或属性的行为。修饰器提供了一种简洁而灵活的方式来扩展和定制代码功能。本文将详细介绍J ava S c ri p t修饰器的概念、语法和应用场景，并提供相关的代码示例。
1. 修饰器简介
修饰器是一种用于修改类、方法或属性的语法，它可以在不修改原始代码的情况下增强其功能。修饰器可以实现横切关注点（c r o ss- c u t t i n g c o n c e r n s）的功能，例如日志记录、性能分析、缓存等。通过将这些功能与原始代码分离，我们可以更好地组织和维护代码，并实现更高的可重用性和可扩展性。
2. 修饰器语法
修饰器使用 @ 符号作为前缀，紧跟着修饰器函数或类。修饰器可以接收不同的参数，根据修饰的目标不同，参数也会有所区别。修饰器可以单独使用，也可以通过组合多个修饰器来实现更复杂的功能。
下面是一个基本的修饰器语法示例：
@decorator
class MyClass {
@propertyDecorator
myProperty = 123;
@methodDecorator
myMethod() {
// 代码逻辑
}
}
3. 类修饰器
类修饰器用于修改类的行为和属性。它可以在类定义之前应用，以修改类的构造函数或原型。
常见的应用场景包括：

「日志记录」：在类的方法执行前后记录日志信息。
「验证和授权」：对类的方法进行验证和授权操作。
「性能分析」：测量类的方法执行时间，进行性能分析。
「依赖注入」：为类的构造函数注入依赖项。

4. 方法修饰器
方法修饰器用于修改类的方法行为。它可以在方法定义之前应用，以修改方法的特性和行为。
常见的应用场景包括：
「日志记录」：在方法执行前后记录日志信息。
「验证和授权」：对方法进行验证和授权操作。
「性能分析」：测量方法执行时间，进行性能分析。
「缓存」：为方法添加缓存功能，提高性能。
下面是一个使用方法修饰器实现日志记录的示例：
应用场景
示例代码
```
function log(target, name, descriptor) {
const originalMethod = descriptor.value;
descriptor.value = function(...args) {
console.log(`Executing method ${name}`);
const result = originalMethod.apply(this, args);
console.log(`Method ${name} executed`);
return result;
};
return descriptor;
}
class MyClass {
@log
myMethod() {
// 代码逻辑
}
}
const myObj = new MyClass();
myObj.myMethod();
```
在上面的示例中，我们定义了一个名为 log 的修饰器函数。该修饰器函数接收三个参数，分别是 target（类的原型或构造函数）、name（方法名）和 descriptor（方法的属性描述符）。在修饰器函数内部，我们获取原始方法并将其保存到 originalMethod 中。然后，我们修改 descriptor.value，将其替换为一个新的函数，该函数在执行原始方法前后打印日志信息。最后，我们返回修改后的属性描述符。

5. 属性修饰器
属性修饰器用于修改类的属性行为。它可以在属性定义之前应用，以修改属性的特性和行为。
常见的应用场景包括：
「日志记录」：在属性读取或写入时记录日志信息。
「验证和授权」：对属性进行验证和授权操作。
「计算属性」：根据其他属性的值计算属性的值。
「缓存」：为属性添加
缓存功能，提高性能。
下面是一个使用属性修饰器实现日志记录的示例：
应用场景
示例代码
```
function log(target, name) {
let value;
const getter = function() {
console.log(`Getting value of property ${name}`);
return value;
};
const setter = function(newValue) {
console.log(`Setting value of property ${name}`);
value = newValue;
```

```
};
Object.defineProperty(target, name, {
get: getter,
set: setter,
enumerable: true,
configurable: true
});
}
class MyClass {
@log
myProperty;
}
const myObj = new MyClass();
myObj.myProperty = 123;
const value = myObj.myProperty;
```

在上面的示例中，我们定义了一个名为 l o g 的修饰器函数。该修饰器函数接收两个参数，分别是 t a r g e t（类的原型或构造函数）和 n a m e （属性名）。在修饰器函数内部，我们定义了一个名为 g e t t e r 的函数，用于获取属性值，并在获取属性值时打印日志信息。我们还定义了一个名为 s e t t e r 的函数，用于设置属性值，并在设置属性值时打印日志信息。最后，我们使用 Ob j e c t . d e fi n ePr o p e rt y 方法将修饰后的属性定义到类的原型上。

6. 参数修饰器

参数修饰器用于修改方法的参数行为。它可以在方法参数声明之前应用，以修改参数的特性和行为。

常见的应用场景包括：

「验证和授权」：对方法的参数进行验证和授权操作。

「日志记录」：在方法执行前后记录参数信息。

「参数转换」：对方法的参数进行类型转换或格式化操作。

下面是一个使用参数修饰器实现参数验证的示例：

应用场景

```
function validate(target, name, index, validator) {
const originalMethod = target[name];
target[name] = function(...args) {
const value = args[index];
if (validator(value)) {
return originalMethod.apply(this, args);
} else {
throw new Error(`Invalid value for parameter ${index} of method ${name}`);
}
};
}
class MyClass {
myMethod(@validate isNumber) {
// 代码逻辑
}
}
function isNumber(value) {
return typeof value ===
"number";
}
const myObj = new MyClass();
myObj.myMethod(123);
```

在上面的示例中，我们定义了一个名为 v a l i d a t e 的修饰器函数。该修饰器函数接收四个参数，分别是 t a r g e t （类的原型或构造函数）、 n a m e （方法名）、 i n d e x （参数索引）和 v a l i d a t o r （验证函数）。在修饰器函数内部，我们获取原始方法并将其保存到 o r i g i n a l M e t h o d 中。然后，我们修改 t a r g e t [n a m e]，将其替换为一个新的函数，该函数在执行原始方法之前对指定参数进行验证。如果参数通过验证，就继续执行原始方法；否则，抛出一个错误。最后，我们使用 @v a l i d a t e 修饰器应用参数验证。

7. 修饰器组合和执行顺序

可以通过组合多个修饰器来实现更复杂的功能。修饰器的执行顺序从上到下，从右到左。
以下是一个使用多个修饰器组合的示例：
function log(target, name, descriptor) {
// 日志记录逻辑
}
function validate(target, name, index, validator) {
// 参数验证逻辑
}
class MyClass {
@log
@validate(isNumber)
myMethod(@validate(isString) param1, @validate(isBoolean) param2) {
// 代码逻辑
}
}
在上面的示例中，我们通过使用 @l o g 修饰器和 @v a l i d a t e 修饰器组合，为类的方法和参数添加日志记录和验证功能。修饰器的执行顺序是从上到下，从右到左。

8. 常用修饰器库和工具
除了原生的修饰器语法，还有许多优秀的修饰器库和工具可供使用。一些常见的库和工具包括：
「cor e -de cora tors」：提供了一组常用的修饰器，如
@readonl y 、 @debounce 、 @throttle 等。GitHub 地址(https:// g ithub. com /j ayphe lps/ cor e -de cor a tors)
「loda sh-de cora tors」：基于Loda sh库的修饰器集合，提供了许多实用的修饰器。GitHub 地址（https:// g ithub. com /st e e lsoj k a /loda sh-de cor a tors)「m obx」：流行的状态管理库MobX使用修饰器来实现响应式数据和自动触发更新。官方文档（https://m obx .js.or g /README .htm l)
「ne stjs」：基于Node .js的框架Ne stJ S使用修饰器来实现依赖注入、路由定义等功能。官方文档（https://docs.ne stjs. com /)

9. 结论
J ava S c ri p t修饰器是一种强大的语法，它能够简化代码、增强功能，并提高代码的可维护性和可扩展性。通过使用修饰器，我们可以轻松地实现日志记录、验证和授权、性能分析等常见的功能，同时保持代码的整洁和可读性。修饰器在许多库和框架中得到了广泛的应用，为开发者提供了更好的开发体验和工具支持。

面试题 74. 简述ES6 的 class 和构造函数的区别？

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

class 的写法只是语法糖，和之前 prototype 差不多，但还是有细微差别的，下面看看：
1. 严格模式
类和模块的内部，默认就是严格模式，所以不需要使用 use strict 指定运行模式。只要你的代码写在类或模块之中，就只有严格模式可用。考虑到未来所有的代码，其实都是运行在模块之中，所以 ES6 实际上把整个语言升级到了严格模式。
2. 不存在提升
类不存在变量提升（hoist），这一点与 ES5 完全不同。
new Foo(); // ReferenceError
class Foo {}
3. 方法默认是不可枚举的
ES6 中的 class，它的方法（包括静态方法和实例方法）默认是不可枚举的，而构造函数默认是可枚举的。细想一下，这其实是个优化，让你在遍历时候，不需要再判断 hasOwnProperty 了
4. class 的所有方法（包括静态方法和实例方法）都没有原型对象 prototype，所以也没有[[construct]]，不能使用 new 来调用。
5. class 必须使用 new 调用，否则会报错。这是它跟普通构造函数的一个主要区别，后者不用 new 也可以执行。
6. ES5 和 ES6 子类 this 生成顺序不同
ES5 的继承先生成了子类实例，再调用父类的构造函数修饰子类实例。ES6 的继承先 生成父类实例，再

调用子类的构造函数修饰父类实例。这个差别使得 ES6 可以继承内置对象。
7. ES6可以继承静态方法，而构造函数不能

面试题 75. 简述拖拽功能的实现？

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

首先是三个事件，分别是 mousedown，mousemove，mouseup 当鼠标点击按下的时候，需要一个 tag 标识此时已经按下，可以执行
mousemove 里面的具体方法。clientX，clientY 标识的是鼠标的坐标，分别标识横坐标和纵坐标，并且我们用 offsetX 和 offsetY 来表示元素的
元素的初始坐标，移动的举例应该是：鼠标移动时候的坐标-鼠标按下去时候的坐标。也就是说定位信息为：鼠标移动时候的坐标-鼠标按下去时
候的坐标+元素初始情况下的 offetLeft.还有一点也是原理性的东西，也就是拖拽的同时是绝对定位，我们改变的是绝对定位条件下的 left 以及 top
等等值。补充：也可以通过 html5 的拖放（Drag 和 drop）来实现

面试题 76. 简述JS 监听对象属性的改变？

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

我们假设这里有一个 user 对象，
在 ES5 中可以通过 Object.defineProperty 来实现已有属性的监听
Object.defineProperty(user,'name',{ set：function(key,value){
}
})
缺点：如果 id 不在 user 对象中，则不能监听 id 的变化(2)在 ES6 中可以通过 Proxy 来实现
varuser = new Proxy({}, {
set：function(target,key,value,receiver){
}
})
这样即使有属性在 user 中不存在，通过 user.id 来定义也同样可以这样监听这个属性的变化哦

面试题 77. 简述自己实现一个 bind？

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

原理：通过 apply 或者 call 方法
来实现。(1)初始版本
Function.prototype.bind=func
tion(obj,arg){
var arg=Array.prototype.slice.call(arguments,1);
var context=this;
return
function(newArg){ arg=arg.concat(Arr
ay.prototype.slice.call(newArg));
return context.apply(obj,arg);
}
}
(2) 考虑到原型链
为什么要考虑？因为在 new 一个 bind 过生成的新函数的时候，必
须的条件是要继承原函数的原型

```
Function.prototype.bind=function(obj,arg){
var
arg=Array.prototype.slice.call(argu
ments,1); var context=this;
var
bound=function(newArg){ arg=arg.co
ncat(Array.prototype.slice.call(newArg)
); return context.apply(obj,arg);
}
var F=function(){}
//这里需要一个寄生组
合继承
F.prototype=context.
prototype;
bound.prototype=ne
w F(); return bound;
}
```

面试题 78. 简述怎么控制一次加载一张图片，加载完后再加载下一张？

推荐指数：★★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

```
<br />
(1)方法 1 <scripttype="text/javascript<br />
"> var obj=new Image();<br />
obj.src="http://www.phpernote.com/uploadfiles/editor/201107240502201179.jpg";<br />
obj.onload=function()<br />
{<br />
alert('图片的宽度为：'+obj.width+'；图片的高度为：'+obj.height);<br />
document.getElementById("mypic").innner<br />
HTML="<img src='"+this.src+"'/>";<br />
}<br />
</script><br />
<div id="mypic">onloading……</div><br />
方法 2<br />
<script type="text/javascript"> var obj=new<br />
Image();<br />
obj.src="http://www.phpernote.com/uploadfiles/editor/201107240502201179.jpg";<br />
obj.onreadystatecha<br />
nge=function(){<br />
if(this.readyState=="complete")<br />
<br />
alert('图片的宽度为：'+obj.width+'；图片的高度为：'+obj.height);<br />
document.getElementById("mypic").innner<br />
HTML="<img src='"+this.src+"' />";<br />
}<br />
}<br />
</script><br />
<div id="mypic">onloading……</div><br />
```

面试题 79. 简述.现实现 JS 中所有对象的深度克隆 （ 包装对象Date 对象 ， 正则对象）？

推荐指数：★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

通过递归可以简单实现对象的深度克隆，但是这种方法不管是 ES6还是 ES5 实现，都有同样的缺陷，就是只能实现特定的 object 的
深度复制（比如数组和函数），不能实现包装对象 Number，String ，Boolean，以及 Date 对象，RegExp 对象的复制。
(1) 前文的方法
function deepClone(obj){
var newObj= obj instanceof
Array?[]:{}; for(var i in obj){
newObj[i]=typeof
obj[i]=='object'?
deepClone(obj[i]):obj[i];
}
return newObj;
}
这种方法可以实现一般对象和数组对象的克隆，比如：
var arr=[1,2,3];
var newArr=deepClone(arr);
//
newArr->[1,2,
3] var obj={
x:1,
y:2
}
var newObj=deepClone(obj);
// newObj={x:1,y:2}
但是不能实现例如包装对象 Number,String,Boolean,以及正则对象 RegExp 和 Date 对象的克隆，比如：
//Number 包装对象
var num=new Number(1); typeof num // "object"
var newNum=deepClone(num);
//newNum -> {} 空对象
//String 包装对象
var str=new String("hello"); typeof str //"object"
var newStr=deepClone(str);
//newStr->{0:'h',1:'e',2:'l',3:'l',4:'o'};
//Boolean 包装对象
var bol=new Boolean(true); typeof bol //"object"

var newBol=deepClone(bol);
// newBol ->{} 空对象
....
(2) valueof()函数
所有对象都有 valueOf 方法，valueOf 方法对于：如果存在任意原
始值，它就默认将对象转换为表示它的原始值。对象是复合值，而且
大多数对象无法真正表示为一个原始值， 因此默认的 valueOf()方法
简单地返回对象本身，而不是返回一个原始值。数组、函数和正则表
达式简单地继承了这个默认方法，调用这些类型的实例的 valueOf()
方法只是简单返回这个对象本身。
对于原始值或者包装类： function baseClone(base){ return
base.valueOf();
}
//Number
var num=new Number(1);
var newNum=baseClone(num);
//newNum->1
//String
var str=new
String('hello'); var
newStr=baseClone

```
(str);
// newStr->"hello"
//Boolean
var bol=new
Boolean(true); var
newBol=baseClone(bol);
//newBol-> true
```

其实对于包装类，完全可以用=号来进行克隆，其实没有深度克隆一说， 这里用 valueOf 实现，语法上比较符合规范。

对于 Date 类型：

因为 valueOf 方法，日期类定义的 valueOf()方法会返回它的一个内部表示：1970 年 1 月 1 日以来的毫秒数.因此我们可以在 Date 的原型上定义克隆的方法： Date.prototype.clone=function(){

```
return new Date(this.valueOf());
}
var date=new
Date('2010'); var
newDate=date.clon
e();
// newDate-> Fri Jan 01 2010 08:00:00 GMT+0800
```

对于正则对象 RegExp：

```
RegExp.prototype.clone =
function() { var pattern =
this.valueOf();
var flags = '';
flags += pattern.global ? 'g' : '';
flags +=
pattern.ignoreCase ? 'i' : '';
flags += pattern.multiline ?
'm' : '';
return new RegExp(pattern.source, flags);
};
var reg=new
RegExp('/111/'); var
newReg=reg.clone()
;
//newReg-> /\/111\//
```

面试题 80. 简述 JS 的全排列？

推荐指数： ★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

```
function permutate(str) {
var result =[];
if(str.length> 1) { var left = str[0];
var rest = str.slice(1,
str.length); var
preResult =permutate(rest);
for(var i=0;
i { for(var j=0;
j var tmp = preResult[i],slice(0, j) + left + preResult[i].slice(j,
preResult[i].length);result.push(tmp);
}
}
```

```
} else if (str.length== 1) { return [str];
}
return result;
}
```

面试题 81. 简述document.write的用法？

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

document.write()方法可以用在两个方面：页面载入过程中用实时脚本创建页面内容，以及用延时脚本创建本窗口或新窗口的内容。
document.write只能重绘整个页面。innerHTML可以重绘页面的一部分
编写一个方法求一个字符串的字节长度
假设：一个英文字符占用一个字节，一个中文字符占用两个字节

```
functionGetBytes(str){
 var len = str.length;
 var bytes = len;
 for(var i=0; i    if (str.charCodeAt(i) >255) bytes++;
 }
 return bytes;
}
alert(GetBytes("你好,as"));
```