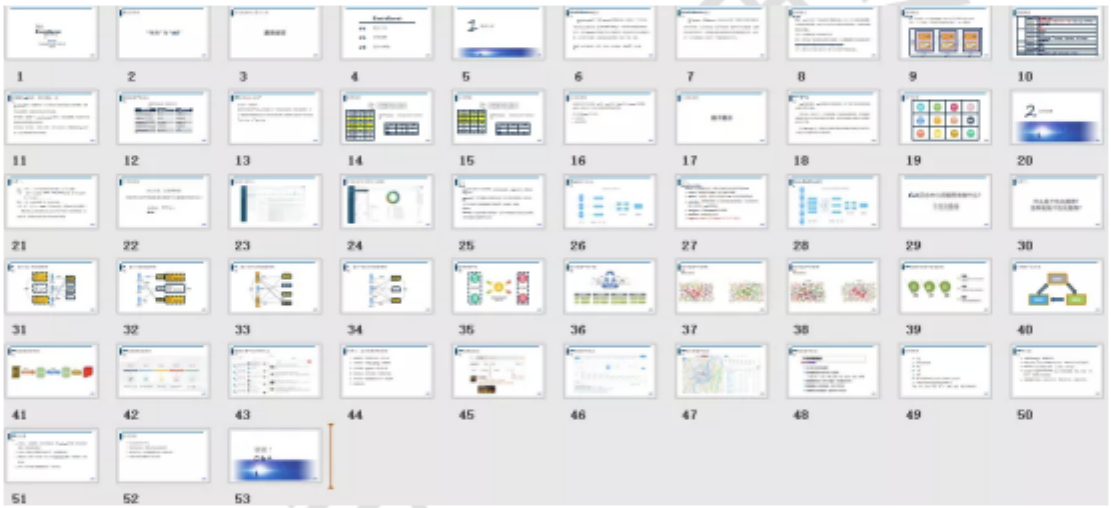
第二版：JavaScript 43 道

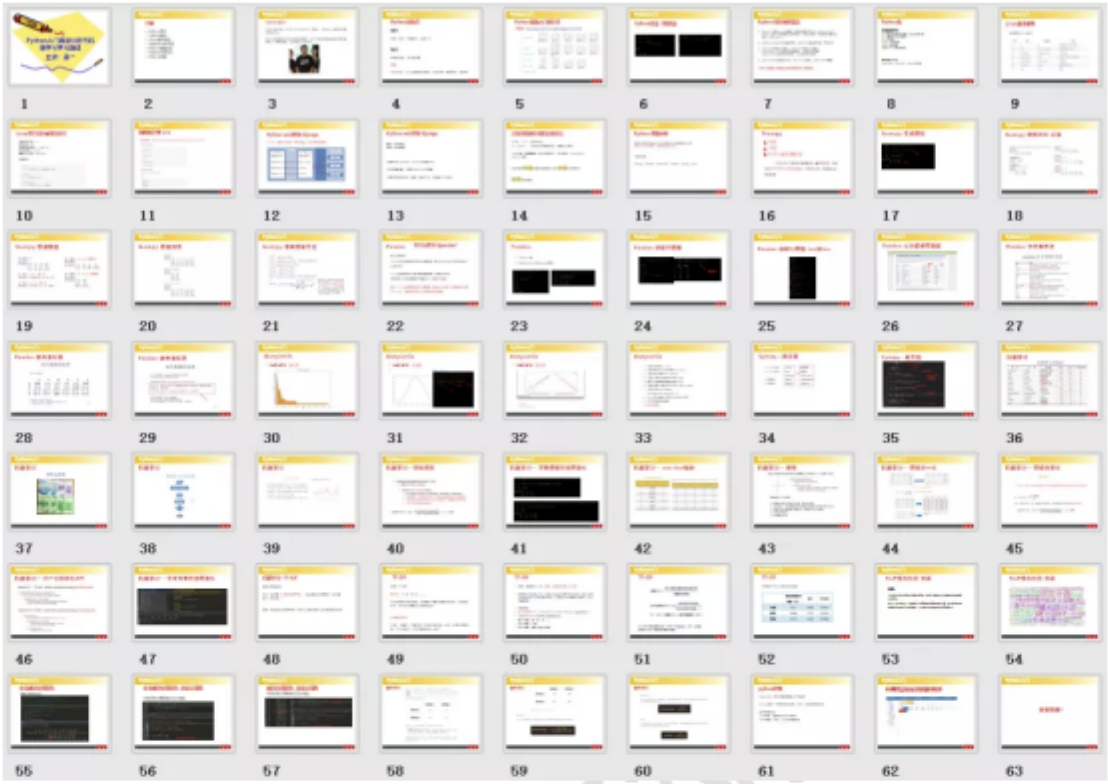
目录

46 页《JVM 深度调优，演讲 PPT》

53 页《Elasticsearch 调优演讲 PPT》



63 页《Python 数据分析入门 PPT》



1. 下面代码的输出是什么?

```
function sayHi() {
  console.log(name);
  console.log(age);
  var name = "Lydia";
  let age = 21;
}

sayHi();
```

- A: Lydia 和 undefined
- B: Lydia 和 ReferenceError
- C: ReferenceError 和 21
- D: undefined 和 ReferenceError

答案

答案: D

在函数中，我们首先使用 var 关键字声明了 name 变量。 这意味着变量在创建阶段会被提升（JavaScript 会在创建变量创建阶段为其分配内存空间），默认值为 undefined，直到我们实际执行到使用该变量的行。 我们还没有为 name 变量赋值，所以它仍然保持 undefined 的值。

使用 let 关键字（和 const）声明的变量也会存在变量提升，但与 var 不同，初始化没有被提升。 在我们声明（初始化）它们之前，它们是不可访问的。 这被称为“暂时死区”。 当我们在声明变量之前尝试访问变量时，JavaScript 会抛出一个 ReferenceError。

译者注：

关于 let 的是否存在变量提升，我们何以用下面的例子来验证：

```
let name = 'ConardLi'
{
  console.log(name) // Uncaught ReferenceError: name is not defined
  let name = 'code 秘密花园'
}
```

let 变量如果不存在变量提升，console.log(name)就会输出 ConardLi，结果却抛出了 ReferenceError，那么这很好的说明了，let 也存在变量提升，但是它存在一个“暂时死区”，在变量未初始化或赋值前不允许访问。

变量的赋值可以分为三个阶段：

- 创建变量，在内存中开辟空间
- 初始化变量，将变量初始化为 undefined
- 真正赋值

关于 let、var 和 function：

- let 的「创建」过程被提升了，但是初始化没有提升。
- var 的「创建」和「初始化」都被提升了。
- function 的「创建」「初始化」和「赋值」都被提升了。

---

2. 下面代码的输出是什么?

```
for (var i = 0; i < 3; i++) {
```

```
  setTimeout(() => console.log(i), 1);
}

for (let i = 0; i < 3; i++) {
  setTimeout(() => console.log(i), 1);
}
```

- A: 0 1 2 and 0 1 2
- B: 0 1 2 and 3 3 3
- C: 3 3 3 and 0 1 2

答案

答案: C

由于 JavaScript 中的事件执行机制，setTimeout 函数真正被执行时，循环已经走完。 由于第一个循环中的变量 i 是使用 var 关键字声明的，因此该值是全局的。 在循环期间，我们每次使用一元运算符++都会将 i 的值增加 1。 因此在第一个例子中，当调用 setTimeout 函数时，i 已经被赋值为 3。

在第二个循环中，使用 let 关键字声明变量 i：使用 let（和 const）关键字声明的变量是具有块作用域的（块是{}之间的任何东西）。 在每次迭代期间，i 将被创建为一个新值，并且每个值都会存在于循环内的块级作用域。

---

3. 下面代码的输出是什么?

```
const shape = {
  radius: 10,
```

```
  diameter() {
    return this.radius * 2;
  },
  perimeter: () => 2 * Math.PI * this.radius
};

shape.diameter();
shape.perimeter();
```

- A: 20 and 62.83185307179586
- B: 20 and NaN
- C: 20 and 63
- D: NaN and 63

答案

答案: B

请注意，diameter 是普通函数，而 perimeter 是箭头函数。

对于箭头函数，this 关键字指向是它所在上下文（定义时的位置）的环境，与普通函数不同！ 这意味着当我们调用 perimeter 时，它不是指向 shape 对象，而是指其定义时的环境（window）。没有值 radius 属性，返回 undefined。

---

4. 下面代码的输出是什么?

```
+true;
!"Lydia";
```

- A: 1 and false
- B: false and NaN
- C: false and false

答案

答案: A

一元加号会尝试将 boolean 类型转换为数字类型。 true 被转换为 1，false 被转换为 0。

字符串'Lydia'是一个真值。 我们实际上要问的是“这个真值是假的吗？”。 这会返回 false。

---

5. 哪个选项是不正确的?

```
const bird = {
  size: "small"
};

const mouse = {
  name: "Mickey",
  small: true
};
```

- A: mouse.bird.size

- B: mouse[bird.size]
- C: mouse[bird["size"]]
- D: All of them are valid

答案

答案: A

在 JavaScript 中，所有对象键都是字符串（除了 Symbol）。尽管有时我们可能不会给定字符串类型，但它们总是被转换为字符串。

JavaScript 解释语句。当我们使用方括号表示法时，它会看到第一个左括号[，然后继续，直到找到右括号]。只有在那个时候，它才会对这个语句求值。

mouse [bird.size]：首先它会对 bird.size 求值，得到 small。 mouse [“small”]返回 true。

但是，使用点表示法，这不会发生。 mouse 没有名为 bird 的键，这意味着 mouse.bird 是 undefined。 然后，我们使用点符号来询问 size：mouse.bird.size。 由于 mouse.bird 是 undefined，我们实际上是在询问 undefined.size。 这是无效的，并将抛出 Cannot read property "size" of undefined。

---

6. 下面代码的输出是什么?

```
let c = { greeting: "Hey!" };
```

```
let d;

d = c;
c.greeting = "Hello";
console.log(d.greeting);
```

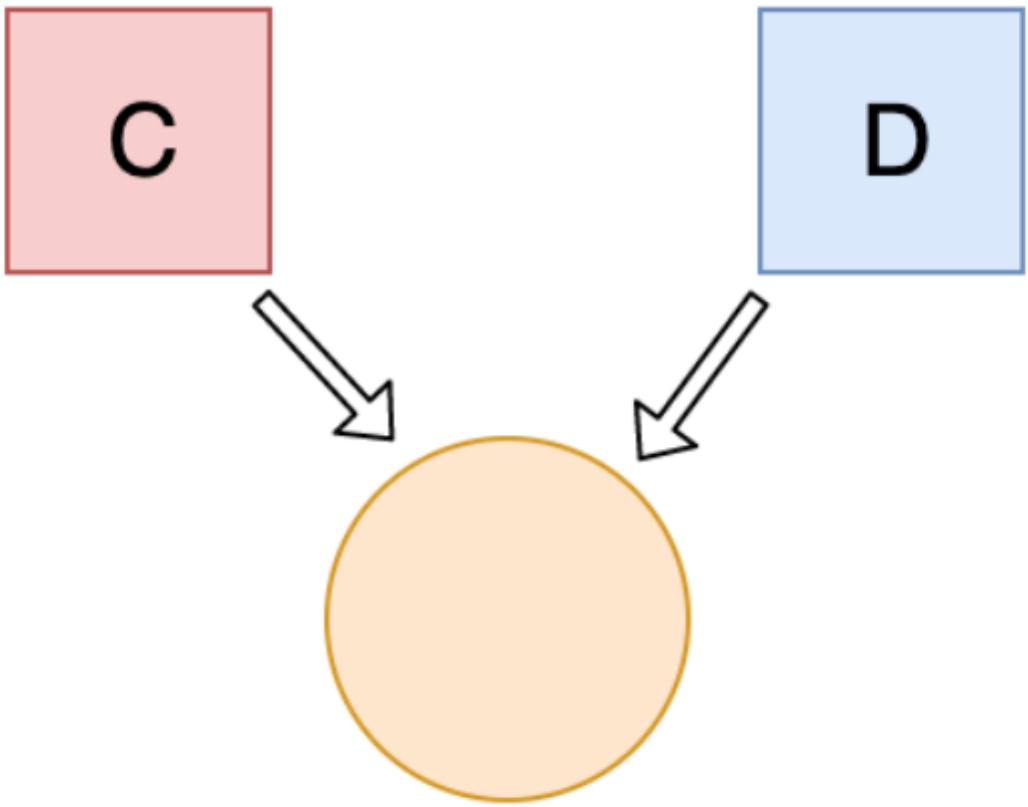
- A: Hello
- B: undefined
- C: ReferenceError
- D: TypeError

答案

答案: A

在 JavaScript 中，当设置它们彼此相等时，所有对象都通过引用进行交互。

首先，变量 c 为对象保存一个值。 之后，我们将 d 指定为 c 与对象相同的引用。



更改一个对象时，可以更改所有对象。

---

7. 下面代码的输出是什么?

```
let a = 3;
let b = new Number(3);
let c = 3;

console.log(a == b);
console.log(a === b);
console.log(b === c);
```

- A: true false true
- B: false false true
- C: true false false
- D: false true true

答案

答案: C

new Number（）是一个内置的函数构造函数。 虽然它看起来像一个数字，但它并不是一个真正的数字：它有一堆额外的功能，是一个对象。

当我们使用==运算符时，它只检查它是否具有相同的值。 他们都有 3 的值，所以它返回 true。

译者注：==会引发隐式类型转换，右侧的对象类型会自动拆箱为 Number 类型。

然而，当我们使用===操作符时，类型和值都需要相等，new Number()不是一个数字，是一个对象类型。两者都返回 false。

---

8. 下面代码的输出是什么?

```
class Chameleon {
  static colorChange(newColor) {
    this.newColor = newColor;
  }

  constructor({ newColor = "green" } = {}) {
```

```
    this.newColor = newColor;
  }
}

const freddie = new Chameleon({ newColor: "purple" });
freddie.colorChange("orange");
```

- A: orange
- B: purple
- C: green
- D: TypeError

答案

答案: D

colorChange 方法是静态的。 静态方法仅在创建它们的构造函数中存在，并且不能传递给任何子级。 由于 freddie 是一个子级对象，函数不会传递，所以在 freddie 实例上不存在 freddie 方法：抛出 TypeError。

---

9. 下面代码的输出是什么?

```
let greeting;
greetign = {}; // Typo!
console.log(greetign);
```

- A: {}

- B: ReferenceError: greetign is not defined
- C: undefined

答案

答案: A

控制台会输出空对象，因为我们刚刚在全局对象上创建了一个空对象！ 当我们错误地将 greeting 输入为 greetign 时，JS 解释器实际上在浏览器中将其视为 global.greetign = {}（或 window.greetign = {}）。

为了避免这种情况，我们可以使用“use strict”。 这可以确保在将变量赋值之前必须声明变量。

---

10. 当我们这样做时会发生什么?

```
function bark() {
  console.log("Woof!");
}

bark.animal = "dog";
```

- A: Nothing, this is totally fine!
- B: SyntaxError. You cannot add properties to a function this way.
- C: undefined
- D: ReferenceError

答案

答案: A

这在 JavaScript 中是可能的，因为函数也是对象！（原始类型之外的所有东西都是对象）

函数是一种特殊类型的对象。您自己编写的代码并不是实际的函数。 该函数是具有属性的对象，此属性是可调用的。

---

11. 下面代码的输出是什么?

```
function Person(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
}

const member = new Person("Lydia", "Hallie");
Person.getFullName = () => this.firstName + this.lastName;

console.log(member.getFullName());
```

- A: TypeError
- B: SyntaxError
- C: Lydia Hallie
- D: undefined undefined

答案

答案: A

您不能像使用常规对象那样向构造函数添加属性。 如果要一次向所有对象添加功能，则必须使用原型。 所以在这种情况下应该这样写：

```
Person.prototype.getFullName = function () {
  return `${this.firstName} ${this.lastName}`;
}
```

这样会使 member.getFullName()是可用的，为什么样做是对的？ 假设我们将此方法添加到构造函数本身。 也许不是每个 Person 实例都需要这种方法。这会浪费大量内存空间，因为它们仍然具有该属性，这占用了每个实例的内存空间。相反，如果我们只将它添加到原型中，我们只需将它放在内存中的一个位置，但它们都可以访问它！

---

12. 下面代码的输出是什么?

```
function Person(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
}

const lydia = new Person("Lydia", "Hallie");
const sarah = Person("Sarah", "Smith");

console.log(lydia);
console.log(sarah);
```

- A: Person {firstName: "Lydia", lastName: "Hallie"} and undefined
- B: Person {firstName: "Lydia", lastName: "Hallie"} and Person {firstName: "Sarah", lastName: "Smith"}
- C: Person {firstName: "Lydia", lastName: "Hallie"} and {}
- D:Person {firstName: "Lydia", lastName: "Hallie"} and ReferenceError

答案

答案: A

对于 sarah，我们没有使用 new 关键字。 使用 new 时，它指的是我们创建的新空对象。 但是，如果你不添加 new 它指的是全局对象！

我们指定了 this.firstName 等于'Sarah 和 this.lastName 等于 Smith。 我们实际做的是定义 global.firstName ='Sarah'和 global.lastName ='Smith。 sarah 本身的返回值是 undefined。

---

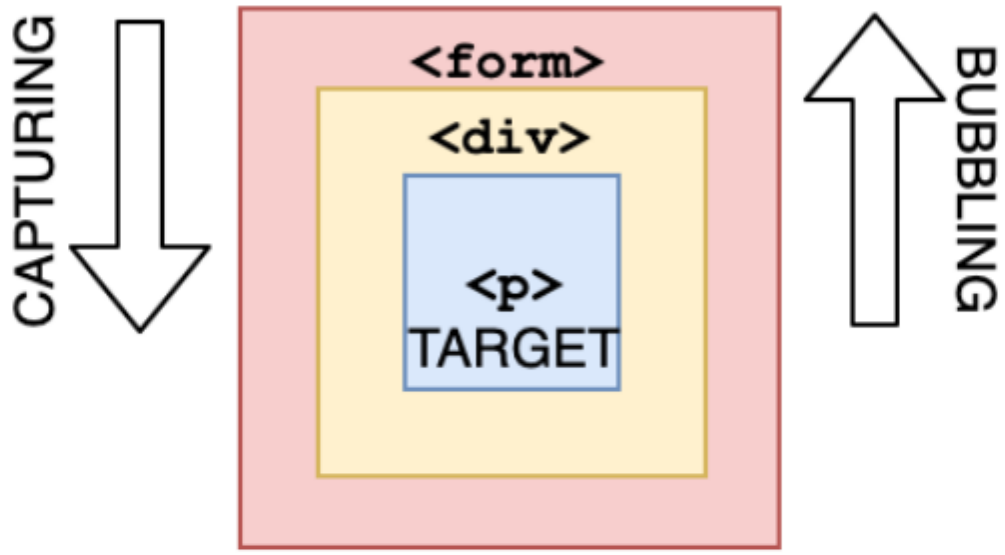
12. 事件传播的三个阶段是什么？？

- A: 目标 > 捕获 > 冒泡
- B: 冒泡 > 目标 > 捕获
- C: 目标 > 冒泡 > 捕获
- D: 捕获 > 目标 > 冒泡

答案

答案: D

在捕获阶段，事件通过父元素向下传递到目标元素。 然后它到达目标元素，冒泡开始。



---

13. 所有对象都有原型.

- A: 对
- B: 错误

答案

答案: B

除**基础对象**外，所有对象都有原型。 基础对象可以访问某些方法和属性，例如.toString。 这就是您可以使用内置 JavaScript 方法的原因！ 所有这些方法

都可以在原型上找到。 虽然 JavaScript 无法直接在您的对象上找到它，但它会沿着原型链向下寻找并在那里找到它，这使您可以访问它。

译者注：基础对象指原型链终点的对象。基础对象的原型是 null。

---

14. 下面代码的输出是什么?

```
function sum(a, b) {
  return a + b;
}

sum(1, "2");
```

- A: NaN
- B: TypeError
- C: "12"
- D: 3

答案

答案: C

JavaScript 是一种**动态类型语言**：我们没有指定某些变量的类型。 在您不知情的情况下，值可以自动转换为另一种类型，称为隐式类型转换。 **强制**从一种类型转换为另一种类型。

在此示例中，JavaScript 将数字 1 转换为字符串，以使函数有意义并返回值。 在让数字类型（1）和字符串类型（'2'）相加时，该数字被视为字符串。 我们可以

连接像“Hello” + “World”这样的字符串，所以这里发生的是“1” + “2”返回“12”。

---

15. 下面代码的输出是什么?

```
let number = 0;
console.log(number++);
console.log(++number);
console.log(number);
```

- A: 1 1 2
- B: 1 2 2
- C: 0 2 2
- D: 0 1 2

答案

答案: C

后缀一元运算符++：

1、 返回值（返回 0）
2、 增加值（数字现在是 1）

前缀一元运算符++：

1、 增加值（数字现在是 2）
2、 返回值（返回 2）

所以返回 0 2 2。

---

16. 下面代码的输出是什么?

```
function getPersonInfo(one, two, three) {
  console.log(one);
  console.log(two);
  console.log(three);
}

const person = "Lydia";
const age = 21;

getPersonInfo`${person} is ${age} years old`;
```

- A: Lydia 21 ["", "is", "years old"]
- B: ["", "is", "years old"] Lydia 21
- C: Lydia ["", "is", "years old"] 21

答案

答案: B

如果使用标记的模板字符串，则第一个参数的值始终是字符串值的数组。 其余参数获取传递到模板字符串中的表达式的值！

---

17. 下面代码的输出是什么?

```
function checkAge(data) {
  if (data === { age: 18 }) {
    console.log("You are an adult!");
  } else if (data == { age: 18 }) {
    console.log("You are still an adult.");
  } else {
    console.log(`Hmm.. You don't have an age I guess`);
  }
}

checkAge({ age: 18 });
```

- A: You are an adult!
- B: You are still an adult.
- C: Hmm.. You don't have an age I guess

答案

答案: C

在比较相等性，原始类型通过它们的值进行比较，而对象通过它们的引用进行比较。JavaScript 检查对象是否具有对内存中相同位置的引用。

我们作为参数传递的对象和我们用于检查相等性的对象在内存中位于不同位置，所以它们的引用是不同的。

这就是为什么{ age: 18 } === { age: 18 }和 { age: 18 } == { age: 18 } 返回 false 的原因。

---

18. 下面代码的输出是什么?

```
function getAge(...args) {
  console.log(typeof args);
}

getAge(21);
```

- A: "number"
- B: "array"
- C: "object"
- D: "NaN"

答案

答案: C

扩展运算符（... args）返回一个带参数的数组。 数组是一个对象，因此 typeof args 返回 object。

---

20. 下面代码的输出是什么?

```
function getAge() {
  "use strict";
  age = 21;
  console.log(age);
}

getAge();
```

- A: 21
- B: undefined
- C: ReferenceError
- D: TypeError

答案

答案: C

使用“use strict”，可以确保不会意外地声明全局变量。 我们从未声明变量 age，因为我们使用`use strict'，它会引发一个 ReferenceError。 如果我们不使用“use strict”，它就会起作用，因为属性 age`会被添加到全局对象中。

---

21. 下面代码的输出是什么?

```
const sum = eval("10*10+5");
```

- A: 105
- B: "105"

- C: TypeError
- D: "10*10+5"

答案

答案: A

eval 会为字符串传递的代码求值。 如果它是一个表达式，就像在这种情况下一样，它会计算表达式。 表达式为 10 * 10 + 5 计算得到 105。

---

22. cool_secret 可以访问多长时间?

```
sessionStorage.setItem("cool_secret", 123);
```

- A：永远，数据不会丢失。
- B：用户关闭选项卡时。
- C：当用户关闭整个浏览器时，不仅是选项卡。
- D：用户关闭计算机时。

答案

答案: B

关闭选项卡后，将删除存储在 sessionStorage 中的数据。

如果使用 localStorage，数据将永远存在，除非例如调用 localStorage.clear()。

---

23. 下面代码的输出是什么?

```
var num = 8;
var num = 10;

console.log(num);
```

- A: 8
- B: 10
- C: SyntaxError
- D: ReferenceError

答案

答案: B

使用 var 关键字，您可以用相同的名称声明多个变量。然后变量将保存最新的值。

您不能使用 let 或 const 来实现这一点，因为它们是块作用域的。

---

24. 下面代码的输出是什么?

```
const obj = { 1: "a", 2: "b", 3: "c" };
const set = new Set([1, 2, 3, 4, 5]);

obj.hasOwnProperty("1");
obj.hasOwnProperty(1);
```

```
set.has("1");
set.has(1);
```

- A: false true false true
- B: false true true true
- C: true true false true
- D: true true true true

答案

答案: C

所有对象键（不包括 Symbols）都会被存储为字符串，即使你没有给定字符串类型的键。 这就是为什么 obj.hasOwnProperty（'1'）也返回 true。

上面的说法不适用于 Set。 在我们的 Set 中没有“1”：set.has（'1'）返回 false。 它有数字类型 1，set.has（1）返回 true。

---

25. 下面代码的输出是什么?

```
const obj = { a: "one", b: "two", a: "three" };
console.log(obj);
```

- A: { a: "one", b: "two" }
- B: { b: "two", a: "three" }
- C: { a: "three", b: "two" }
- D: SyntaxError

答案

答案: C

如果对象有两个具有相同名称的键，则将替前面的键。它仍将处于第一个位置，但具有最后指定的值。

---

26. JavaScript 全局执行上下文为你创建了两个东西:全局对象和 this 关键字.

- A: 对
- B: 错误
- C: 视情况而定

答案

答案: A

基本执行上下文是全局执行上下文:它是代码中随处可访问的内容。

---

27. 下面代码的输出是什么?

```
for (let i = 1; i < 5; i++) {
```

```
  if (i === 3) continue;
  console.log(i);
}
```

- A: 1 2
- B: 1 2 3
- C: 1 2 4
- D: 1 3 4

答案

答案: C

如果某个条件返回 true，则 continue 语句跳过迭代。

---

28. 下面代码的输出是什么?

```
String.prototype.giveLydiaPizza = () => {
  return "Just give Lydia pizza already!";
};

const name = "Lydia";

name.giveLydiaPizza();
```

- A: "Just give Lydia pizza already!"
- B: TypeError: not a function

- C: SyntaxError
- D: undefined

答案

答案: A

String 是一个内置的构造函数，我们可以为它添加属性。 我刚给它的原型添加了一个方法。 原始类型的字符串自动转换为字符串对象，由字符串原型函数生成。因此，所有字符串（字符串对象）都可以访问该方法！

译者注：

当使用基本类型的字符串调用 giveLydiaPizza 时，实际上发生了下面的过程：

- 创建一个 String 的包装类型实例
- 在实例上调用 substring 方法
- 销毁实例

---

29. 下面代码的输出是什么?

```
const a = {};
const b = { key: "b" };
const c = { key: "c" };

a[b] = 123;
a[c] = 456;
```

```
console.log(a[b]);
```

- A: 123
- B: 456
- C: undefined
- D: ReferenceError

答案

答案: B

对象键自动转换为字符串。我们试图将一个对象设置为对象 a 的键，其值为 123。

但是，当对象自动转换为字符串化时，它变成了[Object object]。 所以我们在这里说的是 a["Object object"] = 123。 然后，我们可以尝试再次做同样的事情。 c 对象同样会发生隐式类型转换。那么，a["Object object"] = 456。

然后，我们打印 a[b]，它实际上是 a["Object object"]。 我们将其设置为 456，因此返回 456。

---

30. 下面代码的输出是什么?

```
const foo = () => console.log("First");
const bar = () => setTimeout(() => console.log("Second"));
const baz = () => console.log("Third");

bar();
```

```
foo();
baz();
```

- A: First Second Third
- B: First Third Second
- C: Second First Third
- D: Second Third First

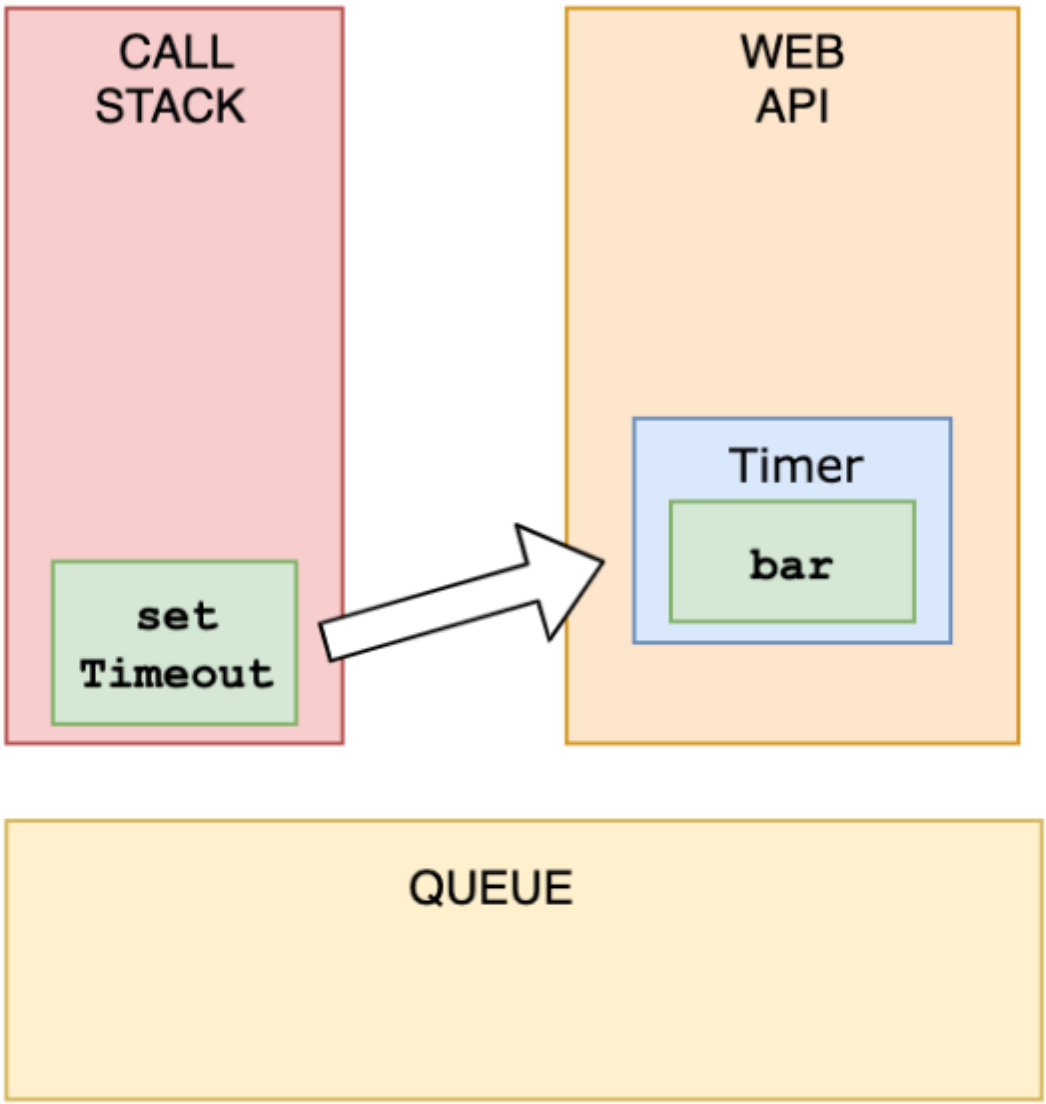
答案

答案: B

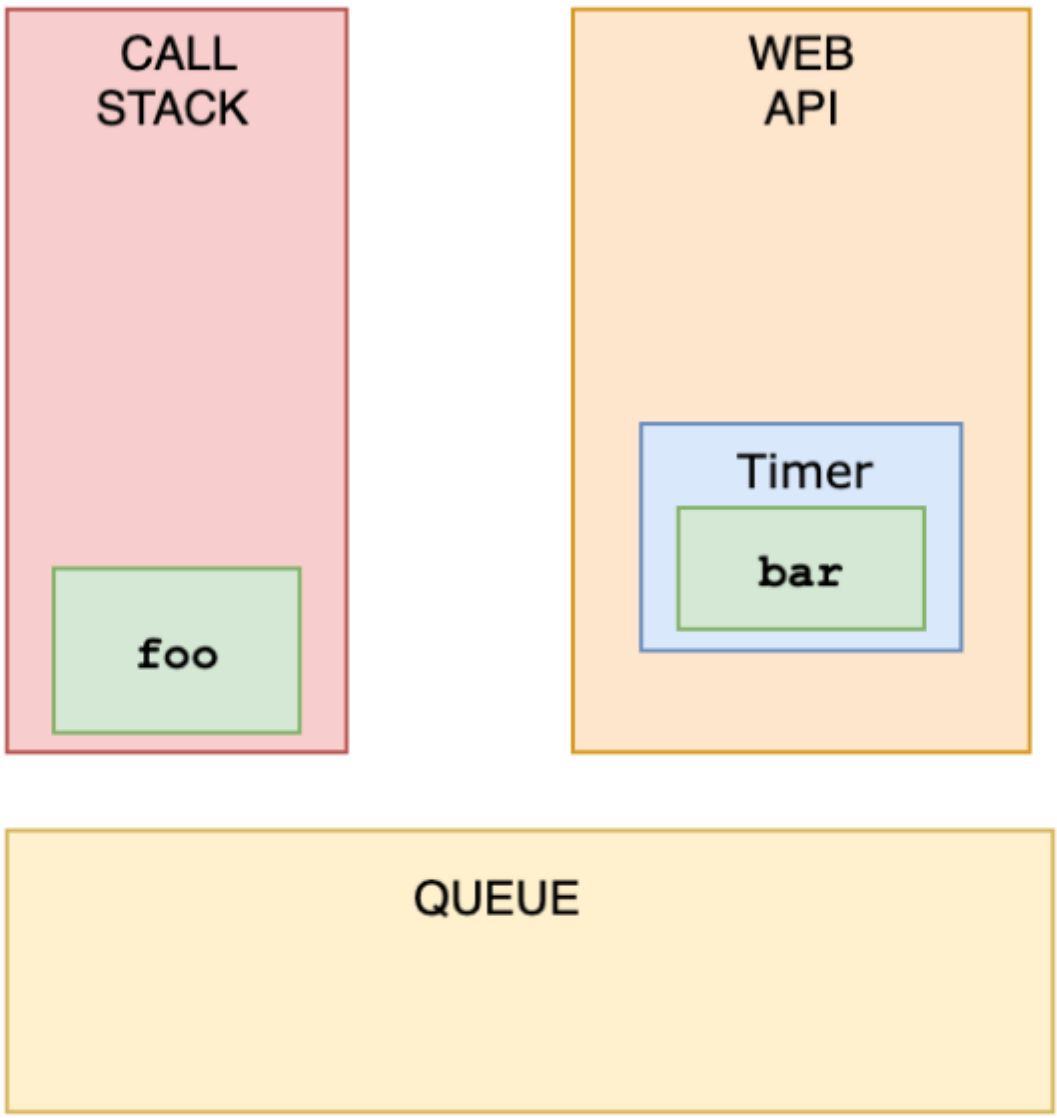
我们有一个 setTimeout 函数并首先调用它。 然而却最后打印了它。

这是因为在浏览器中，我们不只有运行时引擎，我们还有一个叫做 WebAPI 的东西。WebAPI 为我们提供了 setTimeout 函数，例如 DOM。

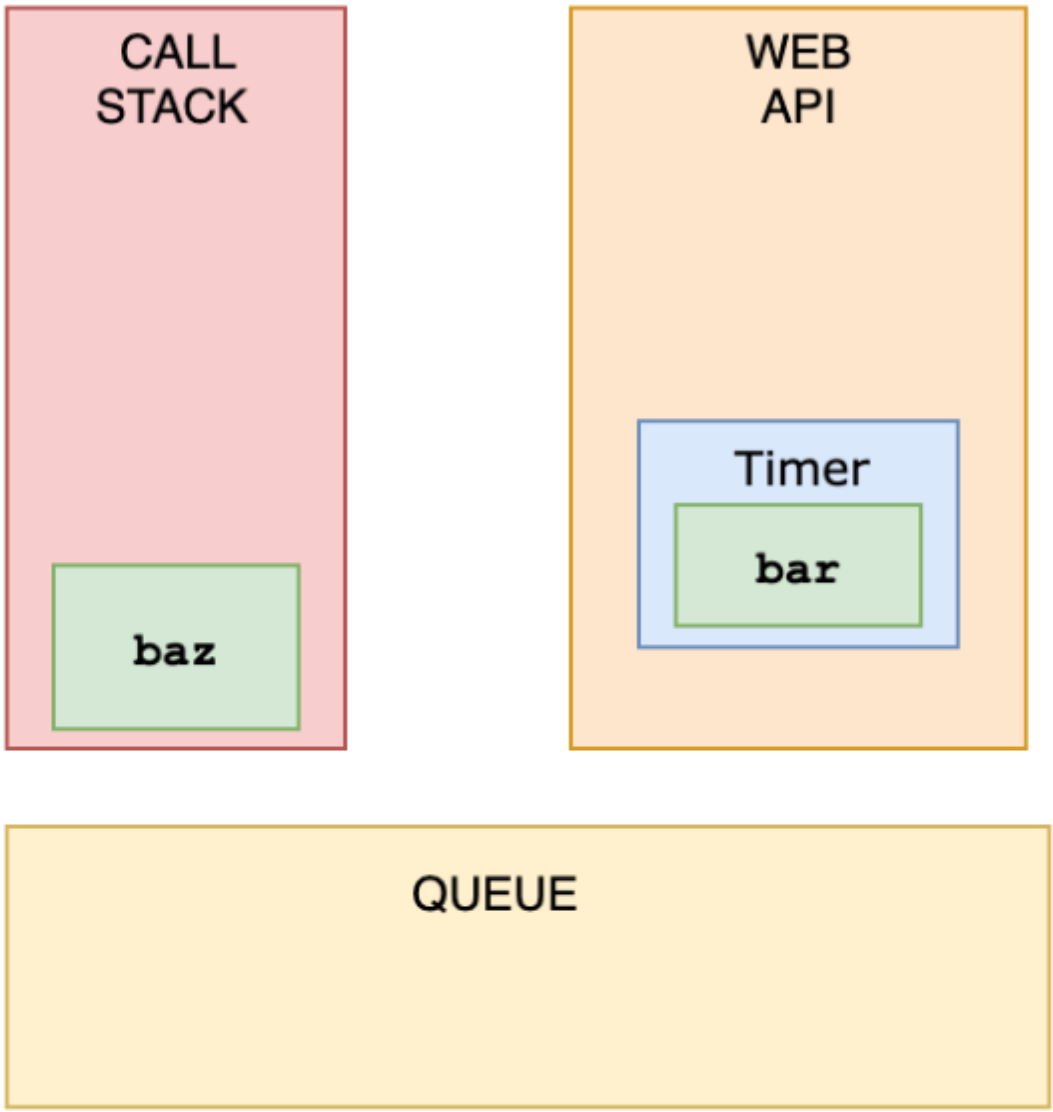
将 callback 推送到 WebAPI 后，setTimeout 函数本身（但不是回调！）从堆栈中弹出。



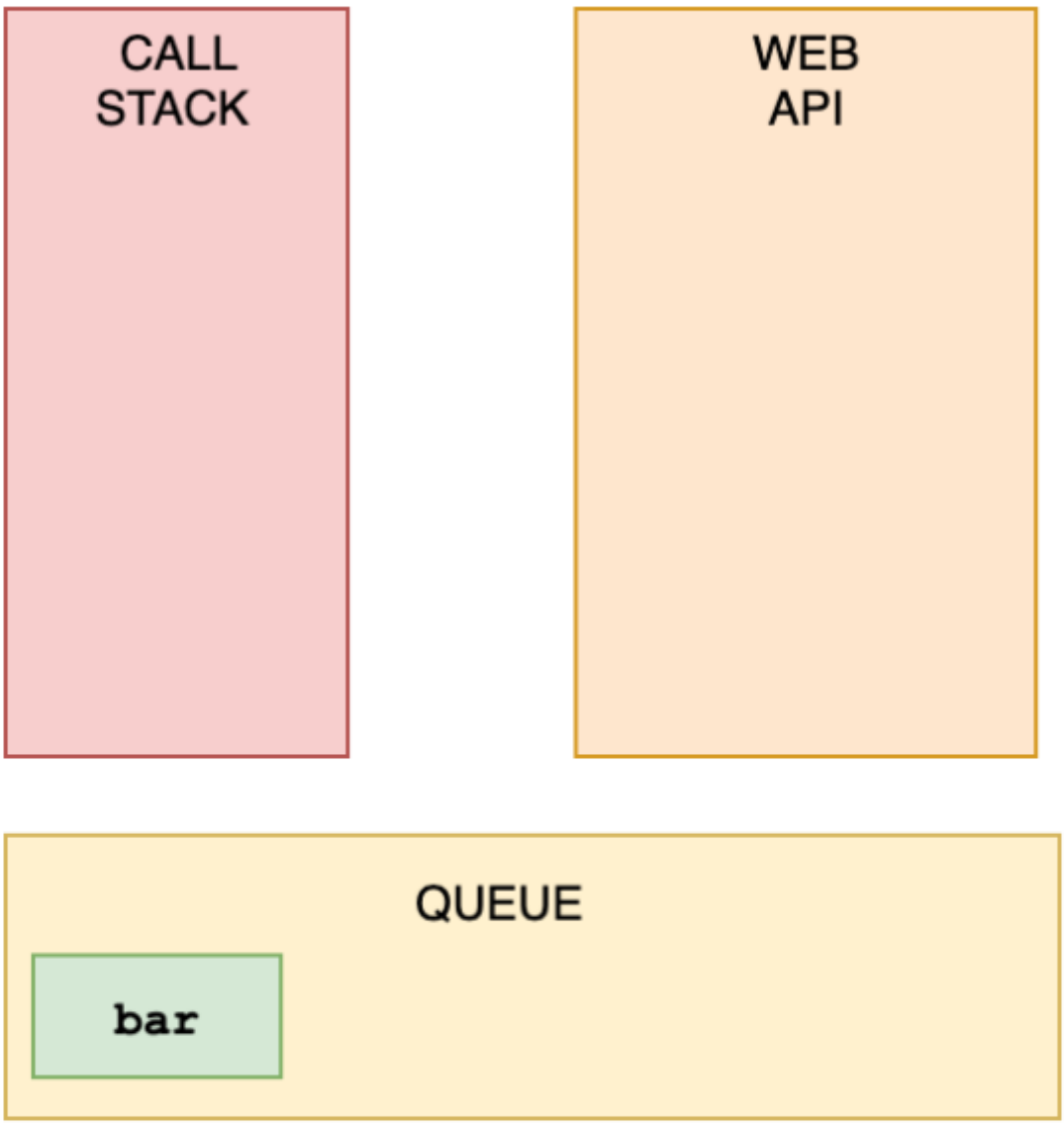
现在，调用 foo，并打印 First。



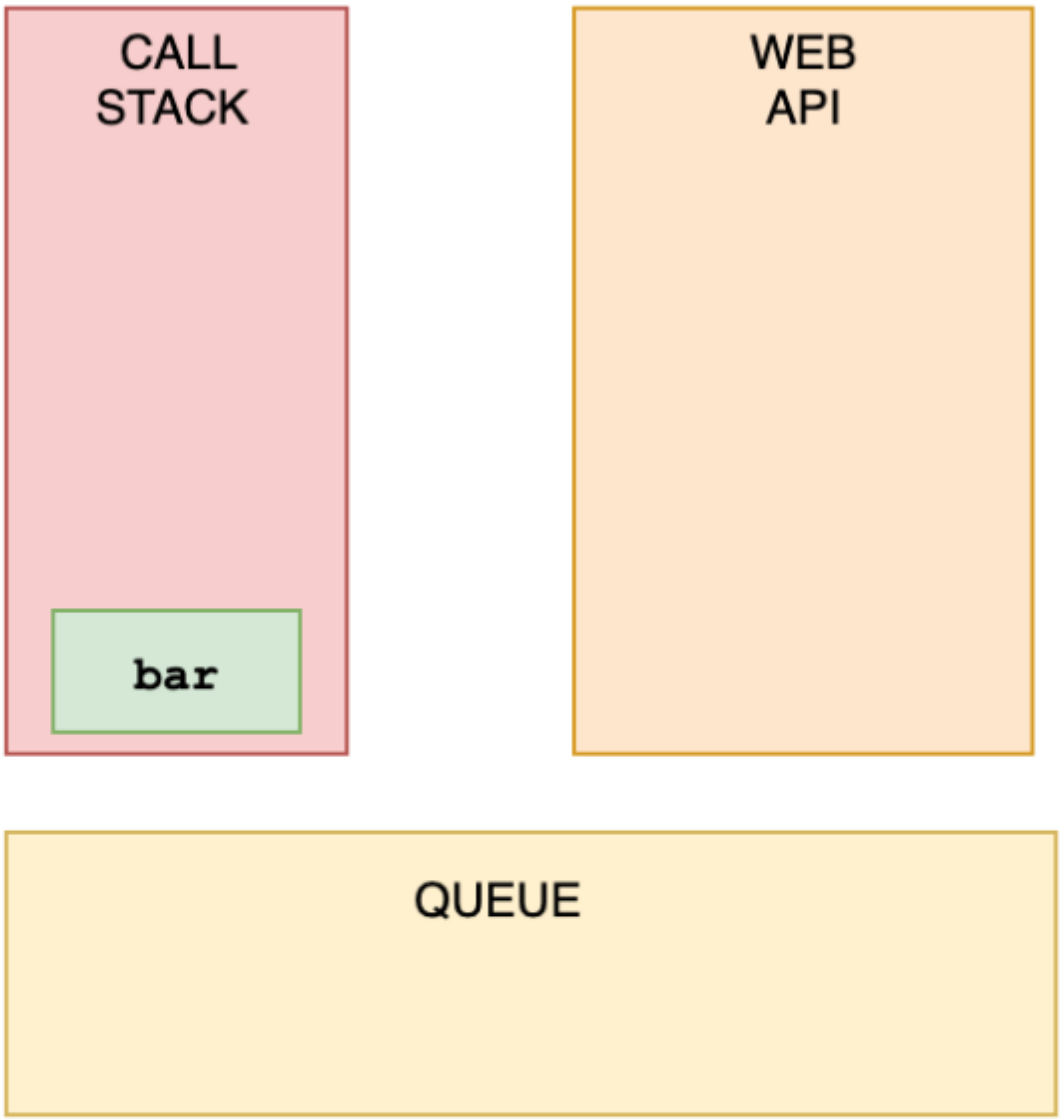
foo 从堆栈弹出，baz 被调用，并打印 Third。



WebAPI 不能只是在准备就绪时将内容添加到堆栈中。 相反，它将回调函数推送到一个称为任务队列的东西。



这是事件循环开始工作的地方。 **事件循环**查看堆栈和任务队列。 如果堆栈为空，则会占用队列中的第一个内容并将其推送到堆栈中。



bar 被调用，Second 被打印，它从栈中弹出。

---

31. 单击按钮时 event.target 是什么?

```
<div onclick="console.log('first div')">
  <div onclick="console.log('second div')">
    <button onclick="console.log('button')">
```

```
      Click!
    </button>
  </div>
</div>
```

- A: div 外部
- B: div 内部
- C: button
- D: 所有嵌套元素的数组.

答案

答案: C

导致事件的最深嵌套元素是事件的目标。你可以通过 event.stopPropagation 停止冒泡

---

32. 单击下面的 html 片段打印的内容是什么?

```
<div onclick="console.log('div')">
  <p onclick="console.log('p')">
    Click here!
  </p>
</div>
```

- A: p div
- B: div p

- C: p
- D: div

答案

答案: A

如果我们单击 p，我们会看到两个日志：p 和 div。在事件传播期间，有三个阶段：捕获，目标和冒泡。 默认情况下，事件处理程序在冒泡阶段执行（除非您将 useCapture 设置为 true）。 它从最深的嵌套元素向外延伸。

---

33. 下面代码的输出是什么?

```
const person = { name: "Lydia" };

function sayHi(age) {
  console.log(`${this.name} is ${age}`);
}

sayHi.call(person, 21);
sayHi.bind(person, 21);
```

- A: undefined is 21 Lydia is 21
- B: function function
- C: Lydia is 21 Lydia is 21
- D: Lydia is 21 function

答案

答案: D

使用两者，我们可以传递我们想要 this 关键字引用的对象。 但是，.call 方法会立即执行！

.bind 方法会返回函数的拷贝值，但带有绑定的上下文！ 它不会立即执行。

---

34. 下面代码的输出是什么?

```
function sayHi() {
  return (() => 0)();
}

typeof sayHi();
```

- A: "object"
- B: "number"
- C: "function"
- D: "undefined"

答案

答案: B

sayHi 函数返回立即调用的函数（IIFE）的返回值。 该函数返回 0，类型为数字。

仅供参考：只有 7 种内置类型：null，undefined，boolean，number，string，object 和 symbol。 function 不是一个类型，因为函数是对象，它的类型是 object。

---

35. 下面这些值哪些是假值?

```
0;
new Number(0);
("");
(" ");
new Boolean(false);
undefined;
```

- A: 0, '', undefined
- B: 0, new Number(0), '', new Boolean(false), undefined
- C: 0, '', new Boolean(false), undefined
- D: 所有都是假值

答案

答案: A

JavaScript 中只有 6 个假值：

- undefined
- null
- NaN

- 0
- '' (empty string)
- false

函数构造函数，如 new Number 和 new Boolean 都是真值。

---

36. 下面代码的输出是什么?

```
console.log(typeof typeof 1);
```

- A: "number"
- B: "string"
- C: "object"
- D: "undefined"

答案

答案: B

typeof 1 返回 "number". typeof "number" 返回 "string"

---

37. 下面代码的输出是什么?

```
const numbers = [1, 2, 3];
numbers[10] = 11;
```

```
console.log(numbers);
```

- A: [1, 2, 3, 7 x null, 11]
- B: [1, 2, 3, 11]
- C: [1, 2, 3, 7 x empty, 11]
- D: SyntaxError

答案

答案: C

当你为数组中的元素设置一个超过数组长度的值时，JavaScript 会创建一个名为“空插槽”的东西。 这些位置的值实际上是 undefined，但你会看到类似的东西：

[1, 2, 3, 7 x empty, 11]

这取决于你运行它的位置（每个浏览器有可能不同）。

---

38. 下面代码的输出是什么?

```
(() => {
  let x, y;
  try {
    throw new Error();
  } catch (x) {
    (x = 1), (y = 2);
    console.log(x);
```

```
  }
  console.log(x);
  console.log(y);
})();
```

- A: 1 undefined 2
- B: undefined undefined undefined
- C: 1 1 2
- D: 1 undefined undefined

答案

答案: A

catch 块接收参数 x。当我们传递参数时，这与变量的 x 不同。这个变量 x 是属于 catch 作用域的。

之后，我们将这个块级作用域的变量设置为 1，并设置变量 y 的值。 现在，我们打印块级作用域的变量 x，它等于 1。

在 catch 块之外，x 仍然是 undefined，而 y 是 2。 当我们想在 catch 块之外的 console.log(x)时，它返回 undefined，而 y 返回 2。

---

39. JavaScript 中的所有内容都是...

- A：原始或对象
- B：函数或对象

- C：技巧问题！只有对象
- D：数字或对象

答案

答案: A

JavaScript 只有原始类型和对象。

原始类型是 boolean，null，undefined，bigint，number，string 和 symbol。

---

40. 下面代码的输出是什么？

```
[[0, 1], [2, 3]].reduce(
  (acc, cur) => {
    return acc.concat(cur);
  },
  [1, 2]
);
```

- A: [0, 1, 2, 3, 1, 2]
- B: [6, 1, 2]
- C: [1, 2, 0, 1, 2, 3]
- D: [1, 2, 6]

答案

答案: C

[1,2]是我们的初始值。 这是我们开始执行 reduce 函数的初始值，以及第一个 acc 的值。 在第一轮中，acc 是[1,2]，cur 是[0,1]。 我们将它们连接起来，结果是[1,2,0,1]。

然后，acc 的值为[1,2,0,1]，cur 的值为[2,3]。 我们将它们连接起来，得到[1,2,0,1,2,3]。

---

41. 下面代码的输出是什么?

```
!!null;
!!"";
!!1;
```

- A: false true false
- B: false false true
- C: false true true
- D: true true false

答案

答案: B

null 是假值。 !null 返回 true。 !true 返回 false。

""是假值。 !""返回 true。 !true 返回 false。

1 是真值。 !1 返回 false。 !false 返回 true。

---

42. setInterval 方法的返回值什么?

```
setInterval(() => console.log("Hi"), 1000);
```

- A：一个唯一的 id
- B：指定的毫秒数
- C：传递的函数
- D：undefined

答案

答案: A

它返回一个唯一的 id。 此 id 可用于使用 clearInterval()函数清除该定时器。

---

43. 下面代码的返回值是什么?

```
[..."Lydia"];
```

- A: ["L", "y", "d", "i", "a"]
- B: ["Lydia"]
- C: [[], "Lydia"]
- D: [["L", "y", "d", "i", "a"]]

答案

答案: A

字符串是可迭代的。 扩展运算符将迭代的每个字符映射到一个元素。