匠人学院
JR ACADEMY

Web开发进阶班第1期

Lesson 7 JavaScript 2

2023.08.05

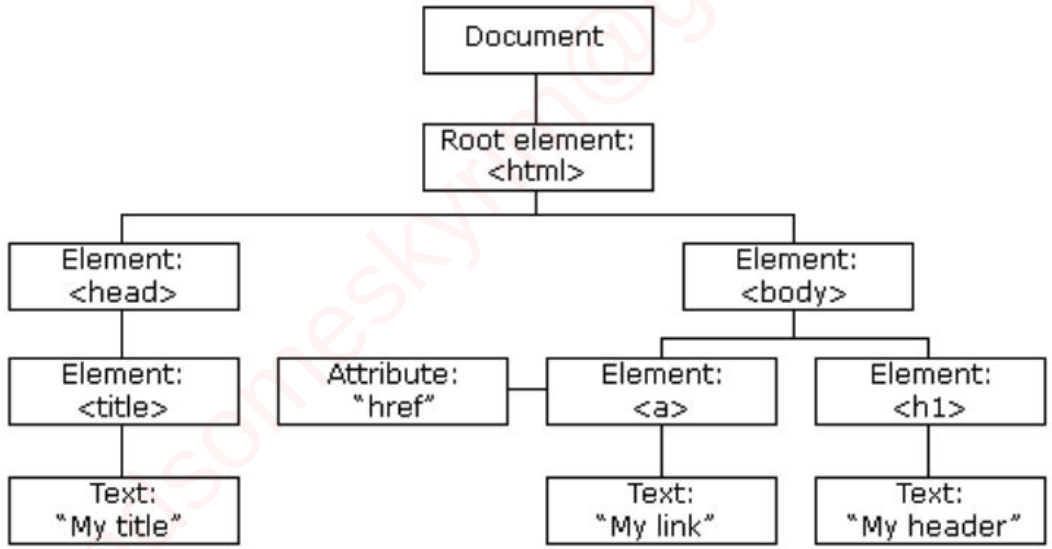
1. ES6介绍
2. 异步编程

补充知识

DOM

DOM的最小组成单位叫做节点（node）。文档的树形结构（DOM树），就是由各种不同类型的节点组成。每个节点可以看作是文档树的一片叶子。

Document：整个文档树的顶层节点
DocumentType：doctype标签（比如<!DOCTYPE html>）
Element：网页的各种HTML标签（比如<body>、<a>等）
Attribute：网页元素的属性（比如class="right"）
Text：标签之间或标签包含的文本
Comment：注释
DocumentFragment：文档的片段



补充知识

DOM Manipulation

1. querySelector
   return the first element found or null

2. querySelectorAll
   return all elements as NodeList object, or empty object

3. addEventListener
   assign function to certain event

4. removeEventListener
5. createElement
   create a new HTML element using the name of HTML tag

6. appendChild
   add an element as the last child to the element that is invoking this method

7. removeChild
8. insertBefore
   add an element before a child element

9. setAttribute
   add a new attribute to an HTML element

补充知识

匠人学院 JR ACADEMY

DOM Events

- `<h1 onclick="this.innerHTML = 'Ooops!'">Click on this text!</h1>`
- `element.addEventListener("click", function(){ alert("Hello World!"); });`
- ‘hover’

补充知识

The Browser Object Model (BOM)

There are no official standards for the Browser Object Model (BOM).

Since modern browsers have implemented (almost) the same methods and properties for JavaScript interactivity, it is often referred to, as methods and properties of the BOM.

- window.innerHeight - the inner height of the browser window (in pixels)
- window.innerWidth - the inner width of the browser window (in pixels)
- window.open() - open a new window
- window.close() - close the current window
- window.location.hostname returns the domain name of the web host
- window.location.pathname returns the path and filename of the current page
- window.location.protocol returns the web protocol used (http: or https:)
- history.back() - same as clicking back in the browser
- history.forward() - same as clicking forward in the browser

The window.alert() method can be written without the window prefix.

1. ES6介绍

What is ES6

ES6是JavaScript的一个版本，它于2015年发布。ES6是ECMAScript的第六个版本，因此它也被称为ES2015。ES6引入了许多新的语言特性，包括箭头函数、let和const关键字、模板字符串、解构赋值、默认参数、剩余参数、展开运算符、类、模块等等。这些新特性使得JavaScript更加现代化、易读易写、易维护。

1. ES6介绍

- let和const关键字：用于声明变量。使用场景：声明变量。

块级作用域

```
{
  let a = 10;
  var b = 1;
}

a // ReferenceError: a is not defined.
b // 1
```

```
for (let i = 0; i < 10; i++) {
  // ...
}

console.log(i);
// ReferenceError: i is not defined
```

```
var a = [];
for (var i = 0; i < 10; i++) {
  a[i] = function () {
    console.log(i);
  };
}
a[6](); // 10
```

1. ES6介绍

- let和const关键字：用于声明变量。使用场景：声明变量。

不存在变量提升（Hoisting）

```
// var 的情况
console.log(foo); // 输出undefined
var foo = 2;

// let 的情况
console.log(bar); // 报错ReferenceError
let bar = 2;
```

1. ES6介绍

- let和const关键字：用于声明变量。使用场景：声明变量。

  不允许重复声明

```
// 报错
function func() {
  let a = 10;
  var a = 1;
}

// 报错
function func() {
  let a = 10;
  let a = 1;
}
```

```
function func(arg) {
  let arg;
}
func() // 报错

function func(arg) {
  {
    let arg;
  }
}
func() // 不报错
```

1. ES6介绍

- 解构（Destructuring）

ES6 允许按照一定模式，从数组和对象中提取值，对变量进行赋值，这被称为解构（Destructuring）。

```
let a = 1;
let b = 2;
let c = 3;
```

```
let [a, b, c] = [1, 2, 3];
```

1. ES6介绍

- 解构(Destructuring)

ES6 允许按照一定模式，从数组和对象中提取值，对变量进行赋值，这被称为解构(Destructuring)。

```
let [foo, [[bar], baz]] = [1, [[2], 3]];
foo // 1
bar // 2
baz // 3

let [ , , third] = ["foo", "bar", "baz"];
third // "baz"

let [x, , y] = [1, 2, 3];
x // 1
y // 3

let [head, ...tail] = [1, 2, 3, 4];
head // 1
tail // [2, 3, 4]

let [x, y, ...z] = ['a'];
x // "a"
y // undefined
z // []
```

1. ES6介绍

- 解构(Destructuring)

解构不仅可以用于数组，还可以用于对象。

```
let { foo, bar } = { foo: 'aaa', bar: 'bbb' };
foo // "aaa"
bar // "bbb"
```

```
let { bar, foo } = { foo: 'aaa', bar: 'bbb' };
foo // "aaa"
bar // "bbb"

let { baz } = { foo: 'aaa', bar: 'bbb' };
baz // undefined
```

1. ES6介绍

- 模板字符串

```
$('#result').append(
  'There are <b>' + basket.count + '</b> ' +
  'items in your basket, ' +
  '<em>' + basket.onSale +
  '</em> are on sale!'
);
```

```
$('#result').append(`
  There are <b>${basket.count}</b> items
   in your basket, <em>${basket.onSale}</em>
  are on sale!
`);
```

1. ES6介绍

- 模板字符串

```
// 普通字符串
`In JavaScript '\n' is a line-feed.`

// 多行字符串
`In JavaScript this is
 not legal.`

console.log(`string text line 1
string text line 2`);

// 字符串中嵌入变量
let name = "Bob", time = "today";
`Hello ${name}, how are you ${time}?`
```

1. ES6介绍

- 模板字符串

```
let x = 1;
let y = 2;

`${x} + ${y} = ${x + y}`
// "1 + 2 = 3"

`${x} + ${y * 2} = ${x + y * 2}`
// "1 + 4 = 5"

let obj = {x: 1, y: 2};
`${obj.x + obj.y}`
// "3"
```

1. ES6介绍

- 字符串新增方法

传统上，JavaScript 只有indexOf方法，可以用来确定一个字符串是否包含在另一个字符串中。ES6 又提供了三种新方法。

includes()：返回布尔值，表示是否找到了参数字符串。
startsWith()：返回布尔值，表示参数字符串是否在原字符串的头部。
endsWith()：返回布尔值，表示参数字符串是否在原字符串的尾部。

```
let s = 'Hello world!';

s.startsWith('Hello') // true
s.endsWith('!') // true
s.includes('o') // true
```

```
let s = 'Hello world!';

s.startsWith('world', 6) // true
s.endsWith('Hello', 5) // true
s.includes('Hello', 6) // false
```

1. ES6介绍

- 箭头函数

  ES6 允许使用“箭头”(=>)定义函数。

```
var f = v => v;

// 等同于
var f = function (v) {
  return v;
};
```

```
var f = () => 5;
// 等同于
var f = function () { return 5 };

var sum = (num1, num2) => num1 + num2;
// 等同于
var sum = function(num1, num2) {
  return num1 + num2;
};
```

1. ES6介绍

- 箭头函数

ES6 允许使用“箭头”(=>)定义函数。

如果箭头函数的代码块部分多于一条语句，就要使用大括号将它们括起来，并且使用 `return` 语句返回。

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

由于大括号被解释为代码块，所以如果箭头函数直接返回一个对象，必须在对象外面加上括号，否则会报错。

```
// 报错
let getTempItem = id => { id: id, name: "Temp" };

// 不报错
let getTempItem = id => ({ id: id, name: "Temp" });
```

1. ES6介绍

- 箭头函数

  简化回调函数Callback

```
// 普通函数写法
[1,2,3].map(function (x) {
  return x * x;
});

// 箭头函数写法
[1,2,3].map(x => x * x);
```

```
// 普通函数写法
var result = values.sort(function (a, b) {
  return a - b;
});

// 箭头函数写法
var result = values.sort((a, b) => a - b);
```

1. ES6介绍

- 箭头函数 https://www.freecodecamp.org/news/the-difference-between-arrow-functions-and-normal-functions/

注意，箭头函数没有this指向，长期以来，JavaScript 语言的this对象一直是一个令人头痛的问题，在对象方法中使用this，必须非常小心。箭头函数”不会改变”this指向，很大程度上解决了这个困扰。

```
function Timer() {
  this.s1 = 0;
  this.s2 = 0;
  // 箭头函数
  setInterval(() => this.s1++, 1000);
  // 普通函数
  setInterval(function () {
    this.s2++;
  }, 1000);
}

var timer = new Timer();

setTimeout(() => console.log('s1: ', timer.s1), 3100);
setTimeout(() => console.log('s2: ', timer.s2), 3100);
// s1: 3
// s2: 0
```

1. ES6介绍

- 数组的扩展

  扩展运算符(spread)是三个点(...)。它好比 rest 参数的逆运算，将一个数组转为用逗号分隔的参数序列。

```
console.log(...[1, 2, 3])
// 1 2 3

console.log(1, ...[2, 3, 4], 5)
// 1 2 3 4 5

[...document.querySelectorAll('div')]
// [<div>, <div>, <div>]
```

1. ES6介绍

- 数组的扩展

扩展运算符(spread)应用

（1）复制数组

数组是复合的数据类型，直接复制的话，只是复制了指向底层数据结构的指针，而不是克隆一个全新的数组。

```
const a1 = [1, 2];
const a2 = a1;

a2[0] = 2;
a1 // [2, 2]
```

上面代码中，`a2`并不是`a1`的克隆，而是指向同一份数据的另一个指针。修改`a2`，会直接导致`a1`的变化。

1. ES6介绍

- 数组的扩展

扩展运算符(spread)应用

（1）复制数组

数组是复合的数据类型，直接复制的话，只是复制了指向底层数据结构的指针，而不是克隆一个全新的数组。

```
const a1 = [1, 2];
const a2 = a1;

a2[0] = 2;
a1 // [2, 2]
```

上面代码中，`a2`并不是`a1`的克隆，而是指向同一份数据的另一个指针。修改`a2`，会直接导致`a1`的变化。

1. ES6介绍

- 数组的扩展

扩展运算符(spread)应用

ES5 只能用变通方法来复制数组。

```
const a1 = [1, 2];
const a2 = a1.concat();

a2[0] = 2;
a1 // [1, 2]
```

上面代码中，`a1`会返回原数组的克隆，再修改`a2`就不会对`a1`产生影响。

扩展运算符提供了复制数组的简便写法。

```
const a1 = [1, 2];
// 写法一
const a2 = [...a1];
// 写法二
const [...a2] = a1;
```

上面的两种写法，`a2`都是`a1`的克隆。

1. ES6介绍

- 数组的扩展

Array.from()：将类数组对象或可迭代对象转换为数组。使用场景：将arguments对象转换为数组、将NodeList对象转换为数组等。

Array.prototype.find()：查找数组中符合条件的第一个元素。使用场景：查找数组中符合条件的第一个元素。

Array.prototype.findIndex()：查找数组中符合条件的第一个元素的索引。使用场景：查找数组中符合条件的第一个元素的索引。

Array.prototype.includes()：判断数组中是否包含某个元素。使用场景：判断数组中是否包含某个元素。

filter: [1,2,3,4].filter((item, index) => {return item %2 === 0})

map: [1,2,3,4].map((item, index) => item * item)

forEach: [1,2,3,4].forEach((item, index) => console.log(item, index))

1. ES6介绍

- 对象的扩展

属性的简介表示，ES6 允许在大括号里面，直接写入变量和函数，作为对象的属性和方法。这样的书写更加简洁。

```
const foo = 'bar';
const baz = {foo};
baz // {foo: "bar"}

// 等同于
const baz = {foo: foo};
```

1. ES6介绍

匠人学院 JR ACADEMY

- 对象的扩展

  Object.assign()方法用于对象的合并，将源对象(source)的所有可枚举属性，复制到目标对象(target)。

```
const target = { a: 1 };

const source1 = { b: 2 };
const source2 = { c: 3 };

Object.assign(target, source1, source2);
target // {a:1, b:2, c:3}
```

1. ES6介绍

- 对象的扩展

Object.assign()方法用于对象的合并，将源对象(source)的所有可枚举属性，复制到目标对象(target)。

`Object.assign()`方法的第一个参数是目标对象，后面的参数都是源对象。

注意，如果目标对象与源对象有同名属性，或多个源对象有同名属性，则后面的属性会覆盖前面的属性。

```
const target = { a: 1, b: 1 };

const source1 = { b: 2, c: 2 };
const source2 = { c: 3 };

Object.assign(target, source1, source2);
target // {a:1, b:2, c:3}
```

1. ES6介绍

- 对象的扩展

  Object.assign()方法用于对象的合并，将源对象（source）的所有可枚举属性，复制到目标对象（target）。

  Object.assign(target, {b:2} ,{c:3}) = still equals to target

  Copy a object
  Object.assign({}, target, {b:2} ,{c:3})
  Object.assign({...target},{b:2}, {c:3})

  {...target, ...{b: 2}, ...{c:3}} // recommend

2. 异步编程

- Promise

Promise 是异步编程的一种解决方案，比传统的解决方案——回调函数和事件——更合理和更强大。它由社区最早提出和实现，ES6 将其写进了语言标准，统一了用法，原生提供了Promise对象。

Promise对象有以下两个特点。

- （1）对象的状态不受外界影响。Promise对象代表一个异步操作，有三种状态：pending（进行中）、fulfilled（已成功）和rejected（已失败）。只有异步操作的结果，可以决定当前是哪一种状态，任何其他操作都无法改变这个状态。这也是Promise这个名字的由来，它的英语意思就是“承诺”，表示其他手段无法改变。
- （2）一旦状态改变，就不会再变，任何时候都可以得到这个结果。Promise对象的状态改变，只有两种可能：从pending变为fulfilled和从pending变为rejected。只要这两种情况发生，状态就凝固了，不会再变了，会一直保持这个结果，这时就称为 resolved（已定型）。如果改变已经发生了，你再对Promise对象添加回调函数，也会立即得到这个结果。这与事件（Event）完全不同，事件的特点是，如果你错过了它，再去监听，是得不到结果的。

2. 异步编程

- Promise

```
const promise = new Promise(function(resolve, reject) {
  // ... some code

  if (/* 异步操作成功 */){
    resolve(value);
  } else {
    reject(error);
  }
});
```

2. 异步编程

- Promise

```
promise.then(function(value) {
  // success
}, function(error) {
  // failure
});
```

```
function timeout(ms) {
  return new Promise((resolve, reject) => {
    setTimeout(resolve, ms, 'done');
  });
}

timeout(100).then((value) => {
  console.log(value);
});
```

2. 异步编程

- Promise
- Promise.all()方法用于将多个 Promise 实例，包装成一个新的Promise 实例。

```
const p = Promise.all([p1, p2, p3]);
```

```
// 生成一个Promise对象的数组
const promises = [2, 3, 5, 7, 11, 13].map(function (id) {
  return getJSON('/post/' + id + ".json");
});

Promise.all(promises).then(function (posts) {
  // ...
}).catch(function(reason){
  // ...
});
```

2. 异步编程

- Async Await
- Async/Await 是一种建立在 promises 基础上的，书写异步代码的新方式，所以它也是非阻塞的。
- 与我们之前异步编程所使用到的回调函数和 promises 相比较，最大的区别就是 Async/Await 使我们的异步代码看起来就像同步代码一样，这也正是它的厉害之处。

Eventloop:

synchronous

asynchronous:

- MacroTask
- MicroTask

2. 异步编程

匠人学院 JR ACADEMY

- Async Await
- **回调函数最典型的问题——回调地狱 callback hell**
- 在回调函数中嵌套回调函数，看起来就像这样：
-

```
asyncCallOne(() => {
  asyncCallTwo(() => {
    asyncCallThree(() => {
      asyncCallFour(() => {
        asyncCallFive(() => {
          asyncCallSix(() => {
            asyncCallSeven(() => {
              asyncCallEight(() => {
                asyncCallNine(() => {
                  asyncCallTen(() => {
                    // do something here...
                    // Maybe Channels and Generators?
                  })
                })
              })
            })
          })
        })
      })
    })
  })
})
```

2. 异步编程

- Async Await
-

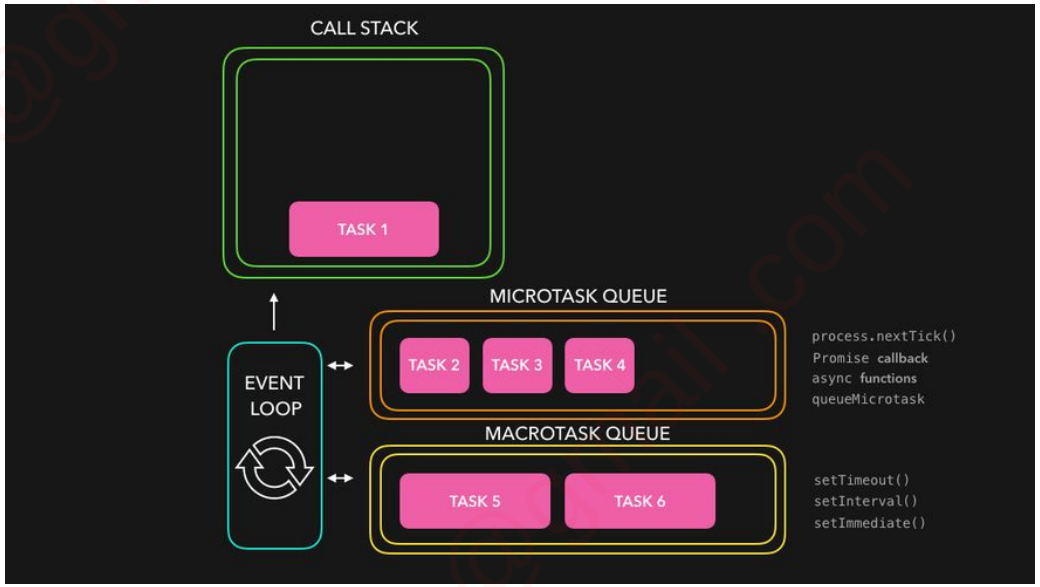
```
const asyncFunction = async () => {

// Code

await fetch()

}
```

2. 异步编程



Eventloop:

Synchronous

Asynchronous:

Microtask
A microtask is a short function which is executed after the function or program which created it exits and only if the JavaScript execution stack is empty.

- Promise callback
- queueMicrotask

https://medium.com/@saravanaeswari22/microtasks-and-macro-tasks-in-event-loop-7b408b2949e0

Macrotask
A macrotask is short function which is executed after JavaScript execution stack and microtask are empty.

- setTimeout
- setInterval
- setImmediate

1. ES6介绍

匠人学院
JR ACADEMY

Homework:

1. **if else => switch case**
2. **try catch**
3. **闭包**
4. **https://juejin.cn/post/7055460626923012104**

匠人学院
JR ACADEMY

Q & A