

# Web APIs 第六天

BOM- 操作浏览器



# 目录

Contents

- ◆ Window对象
- ◆ swiper 插件
- ◆ 本地存储
- ◆ 综合案例

# 学习目标

Learning Objectives

1. 依托 BOM 对象实现对历史、地址、浏览器信息的操作或获取
2. 具备利用本地存储实现学生信息表案例的能力



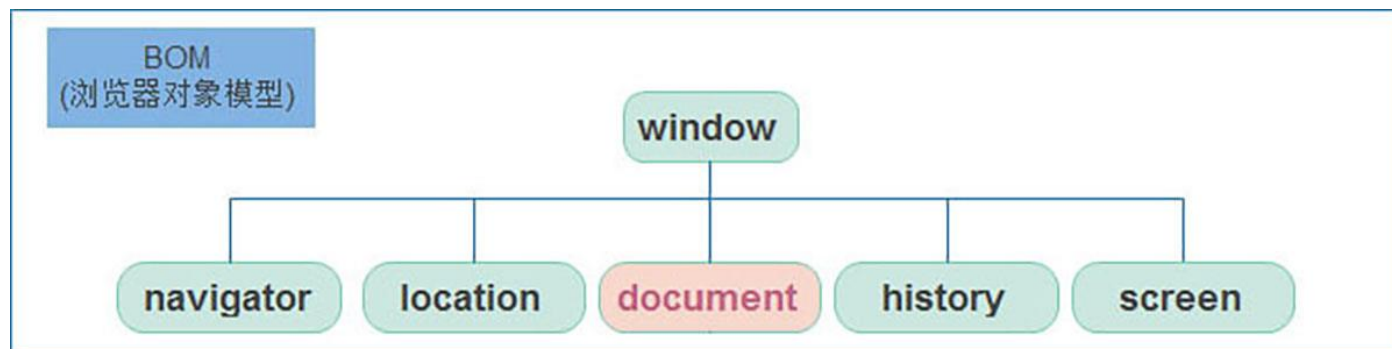
# Window对象

- BOM(浏览器对象模型)
  - 定时器-延时函数
  - JS执行机制
  - location对象
  - navigator对象
  - histroy对象
- 
- 目标：学习 window 对象的常见属性，知道各个 BOM 对象的功能含义

# 一、Window对象

## 1.1 BOM

- BOM(Browser Object Model ) 是浏览器对象模型



- window 是浏览器内置中的全局对象，我们所学习的所有 Web APIs 的知识内容都是基于 window 对象实现的
- window 对象下包含了 navigator、location、document、history、screen 5个属性，即所谓的 BOM（浏览器对象模型）
- document 是实现 DOM 的基础，它其实是依附于 window 的属性。
- 注：依附于 window 对象的所有属性和方法，使用时可以省略 window



# Window对象

- BOM(浏览器对象模型)
  - 定时器-延时函数
  - JS执行机制
  - location对象
  - navigator对象
  - histroy对象
- 
- 目标：学习 window 对象的常见属性，知道各个 BOM 对象的功能含义

# 一、Window对象

## 1.2 定时器-延时函数

- JavaScript 内置的一个用来让代码延迟执行的函数，叫 setTimeout
- 语法：

```
setTimeout(回调函数, 等待的毫秒数)
```

- setTimeout 仅仅只执行一次，所以可以理解为就是把一段代码延迟执行，平时省略window
- 清除延时函数：

```
let timer = setTimeout(回调函数, 等待的毫秒数)  
clearTimeout(timer)
```

## 案例

### 5秒钟之后消失的广告

需求：5秒钟之后，广告自动消失

分析：

①：设置延时函数

②：隐藏元素



## 1.2 定时器-延时函数

- 结合递归函数可以使用 setTimeout 实现 setInterval 一样的功能

```
<div class="clock"></div>
<script>
  let clock = document.querySelector('.clock')
  function myInterval() {
    let d = new Date();
    clock.innerText = d.toLocaleString();
    // 延时任务, 自调用
    setTimeout(myInterval, 1000);
  }
  // 启动定时任务
  myInterval();
</script>
```

```
<\script>
  myInterval();
```

## 1.2 定时器-延时函数

- 两种定时器对比:

- setInterval 的特征是重复执行, 首次执行会延时
- setTimeout 的特征是延时执行, 只执行 1 次
- setTimeout 结合递归函数, 能模拟 setInterval 重复执行
- clearTimeout 清除由 setTimeout 创建的定时任务



# Window对象

- BOM(浏览器对象模型)
  - 定时器-延时函数
  - JS执行机制
  - location对象
  - navigator对象
  - histroy对象
- 目标：学习 window 对象的常见属性，知道各个 BOM 对象的功能含义

# 一、Window对象

## 1.2 经典面试题

```
console.log(1111)
setTimeout(function () {
  console.log(2222)
}, 1000)
console.log(3333)
// 问，输出的结果是什么？
```

```
console.log(1111)
setTimeout(function () {
  console.log(2222)
}, 0)
console.log(3333)
// 问，输出的结果是什么？
```

### JS 是单线程

JavaScript 语言的一大特点就是**单线程**，也就是说，**同一个时间只能做一件事**。这是因为 Javascript 这门脚本语言诞生的使命所致——JavaScript 是为处理页面中用户的交互，以及操作 DOM 而诞生的。比如我们对某个 DOM 元素进行添加和删除操作，不能同时进行。应该先进行添加，之后再删除。

单线程就意味着，所有任务需要排队，前一个任务结束，才会执行后一个任务。这样所导致的问题是：如果 JS 执行的时间过长，这样就会造成页面的渲染不连贯，导致页面渲染加载阻塞的感觉。

### 同步和异步

为了解决这个问题，利用多核 CPU 的计算能力，HTML5 提出 Web Worker 标准，允许 JavaScript 脚本创建多个线程。于是，JS 中出现了**同步**和**异步**。

#### 同步

前一个任务结束后再执行后一个任务，程序的执行顺序与任务的排列顺序是一致的、同步的。比如做饭的同步做法：我们要烧水煮饭，等水开了（10分钟之后），再去切菜，炒菜。

#### 异步

你在做一件事情时，因为这件事情会花费很长时间，在做这件事的同时，你还可以去处理其他事情。比如做饭的异步做法，我们在烧水的同时，利用这10分钟，去切菜，炒菜。

**他们的本质区别：这条流水线上各个流程的执行顺序不同。**

### 同步和异步

#### 同步任务

同步任务都在主线程上执行，形成一个**执行栈**。

#### 异步任务

JS 的异步是通过回调函数实现的。

一般而言，异步任务有以下三种类型：

- 1、普通事件，如 click、resize 等
- 2、资源加载，如 load、error 等
- 3、定时器，包括 setInterval、setTimeout 等

异步任务相关添加到**任务队列**中（任务队列也称为消息队列）。

执行栈

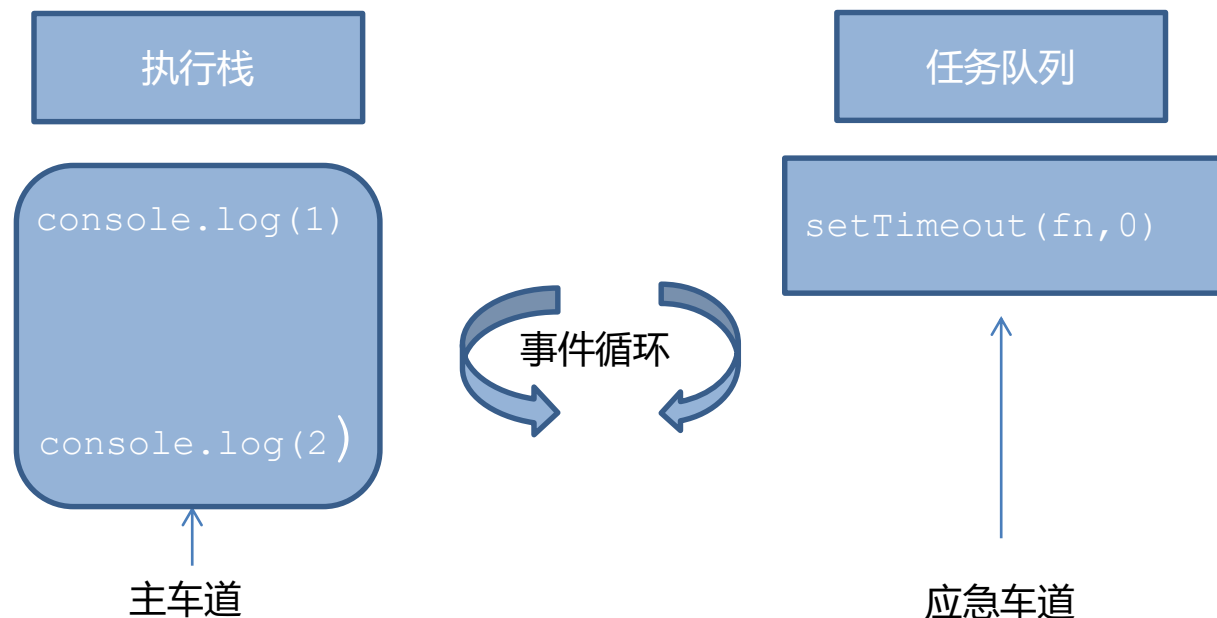
```
console.log(1)
setTimeout(fn, 0)
console.log(2)
```

任务队列

fn

### JS 执行机制

1. 先执行**执行栈中的同步任务**。
2. 异步任务放入任务队列中。
3. 一旦执行栈中的所有同步任务执行完毕，系统就会按次序读取**任务队列**中的异步任务，于是被读取的异步任务结束等待状态，进入执行栈，开始执行。





思考：

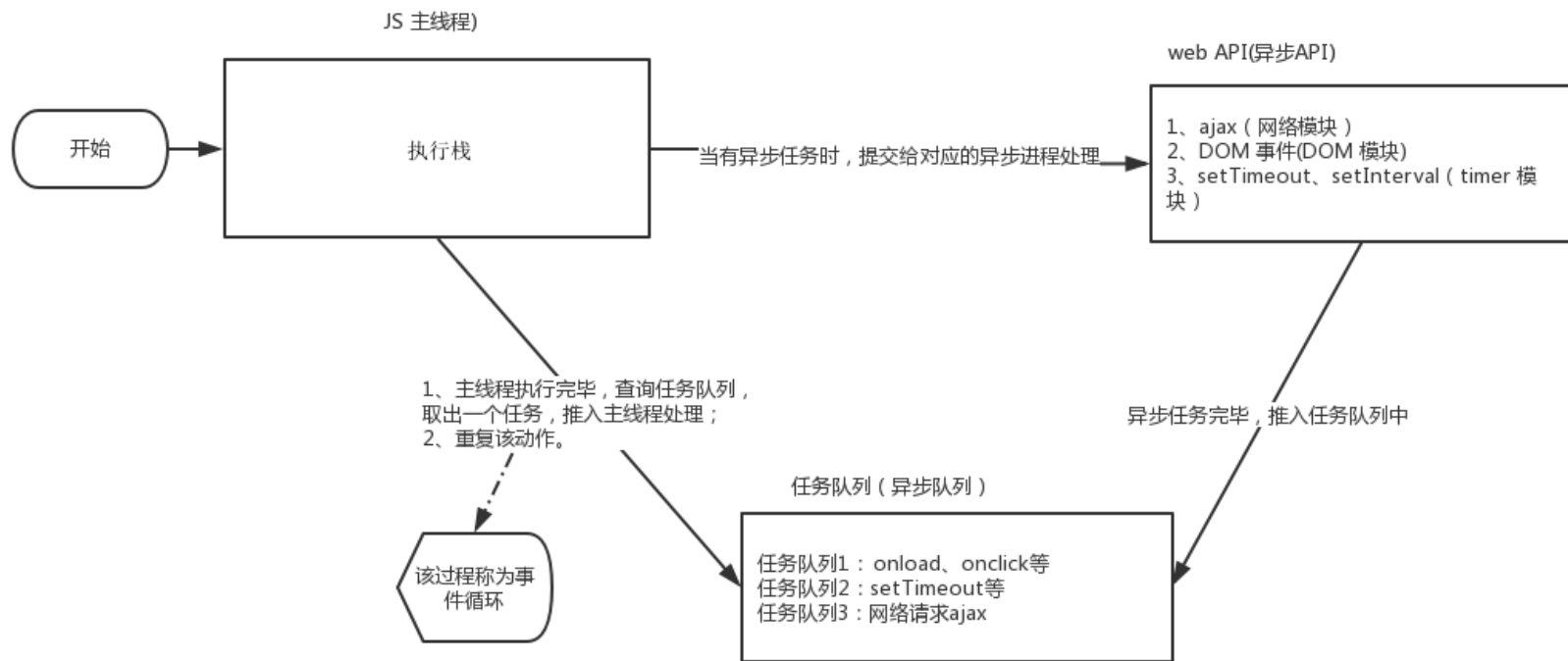
```
console.log(1)

document.addEventListener('click', function () {
  console.log(4)
})

console.log(2)

setTimeout(function () {
  console.log(3)
}, 3000)
```

### 图例小结



由于主线程不断的重复获得任务、执行任务、再获取任务、再执行，所以这种机制被称为**事件循环 (event loop)**。



# Window对象

- BOM(浏览器对象模型)
  - 定时器-延时函数
  - JS执行机制
  - location对象
  - navigator对象
  - histroy对象
- 目标：学习 window 对象的常见属性，知道各个 BOM 对象的功能含义

## 1.4 location对象

- location 的数据类型是对象，它拆分并保存了 URL 地址的各个组成部分
- **常用属性和方法：**
  - href 属性获取完整的 URL 地址，对其赋值时用于地址的跳转
  - search 属性获取地址中携带的参数，符号 ? 后面部分
  - hash 属性获取地址中的啥希值，符号 # 后面部分
  - reload 方法用来刷新当前页面，传入参数 true 时表示强制刷新

## 1.4 location对象

- location 的数据类型是对象，它拆分并保存了 URL 地址的各个组成部分
- 常用属性和方法：
  - href 属性获取完整的 URL 地址，对其赋值时用于地址的跳转

```
// 可以得到当前文件URL地址  
console.log(location.href)  
// 可以通过js方式跳转到目标地址  
location.href = 'http://www.itcast.cn'
```

## 案例

### 5秒钟之后跳转的页面

需求：用户点击可以跳转，如果不点击，则5秒之后自动跳转，要求里面有秒数倒计时

分析：

- ①：目标元素是链接
- ②：利用定时器设置数字倒计时
- ③：时间到了，自动跳转到新的页面

支付成功 1秒之后跳转回原网页

## 1.4 location对象

- location 的数据类型是对象，它拆分并保存了 URL 地址的各个组成部分
- 常用属性和方法：
  - search 属性获取地址中携带的参数，符号 ? 后面部分

```
console.log(location.search)
```

## 1.4 location对象

- location 的数据类型是对象，它拆分并保存了 URL 地址的各个组成部分
- 常用属性和方法：
  - hash 属性获取地址中的哈希值，符号 # 后面部分

```
console.log(location.hash)
```

- 后期vue路由的铺垫，经常用于不刷新页面，显示不同页面，比如 网易云音乐



## 1.4 location对象

- location 的数据类型是对象，它拆分并保存了 URL 地址的各个组成部分
- 常用属性和方法：
  - reload 方法用来刷新当前页面，传入参数 true 时表示强制刷新

```
<button>点击刷新</button>
<script>
    let btn = document.querySelector('button')
    btn.addEventListener('click', function () {
        location.reload(true)
        // 强制刷新 类似 ctrl + f5
    })
</script>
</script>
    })
```



# 总结

- location.href 属性获取完整的 URL 地址，对其赋值时用于地址的跳转
- search 属性获取地址中携带的参数，符号 ? 后面部分
- hash 属性获取地址中的啥希值，符号 # 后面部分
- reload 方法用来刷新当前页面，传入参数 true 时表示强制刷新



# Window对象

- BOM(浏览器对象模型)
  - 定时器-延时函数
  - JS执行机制
  - location对象
  - navigator对象
  - histroy对象
- 
- 目标：学习 window 对象的常见属性，知道各个 BOM 对象的功能含义

## 1.5 navigator对象

- navigator的数据类型是对象，该对象下记录了浏览器自身的相关信息
- 常用属性和方法：
  - 通过 userAgent 检测浏览器的版本及平台

```
// 检测 userAgent (浏览器信息)
!(function () {
    const userAgent = navigator.userAgent
    // 验证是否为Android或iPhone
    const android = userAgent.match(/(Android);?[\s\/]+([\d.]+)?/)
    const iphone = userAgent.match(/(iPhone\sOS)\s([\d_]+)/)

    // 如果是Android或iPhone, 则跳转至移动站点
    if (android || iphone) {
        location.href = 'http://m.itcast.cn'
    }
})();
```



## Window对象

- BOM(浏览器对象模型)
  - 定时器-延时函数
  - JS执行机制
  - location对象
  - navigator对象
  - **histroy对象**
- 目标：学习 window 对象的常见属性，知道各个 BOM 对象的功能含义

## 1.6 histroy对象

- history 的数据类型是对象，该对象与浏览器地址栏的操作相对应，如前进、后退、历史记录等
- 常用属性和方法：

| history对象方法 | 作用                                 |
|-------------|------------------------------------|
| back()      | 可以后退功能                             |
| forward()   | 前进功能                               |
| go(参数)      | 前进后退功能 参数如果是 1 前进1个页面 如果是-1 后退1个页面 |

history 对象一般在实际开发中比较少用，但是会在一些 OA 办公系统中见到。





# 目录

Contents

- ◆ Window对象
- ◆ swiper 插件
- ◆ 本地存储
- ◆ 综合案例

### 2.1 插件












- 插件: 就是别人写好的一些代码,我们只需要复制对应的代码,就可以直接实现对应的效果
- 学习插件的基本过程
  - 熟悉官网,了解这个插件可以完成什么需求 <https://www.swiper.com.cn/>
  - 看在线演示,找到符合自己需求的demo <https://www.swiper.com.cn/demo/index.html>
  - 查看基本使用流程 <https://www.swiper.com.cn/usage/index.html>
  - 查看API文档,去配置自己的插件 <https://www.swiper.com.cn/api/index.html>
  - 注意: 多个swiper同时使用的时候, 类名需要注意区分



## 二、swiper

### 2.1 插件

- 1. 本地文件

|  |                |
|--|----------------|
|  .github      | 2021/8/12 21:5 |
|  .nova        | 2021/8/12 21:5 |
|  .vscode      | 2021/8/12 21:5 |
|  build        | 2021/8/12 21:5 |
|  cypress      | 2021/8/12 21:5 |
|  demos        | 2021/8/12 21:5 |
|  package     | 2021/8/12 21:5 |
|  playground | 2021/8/12 21:5 |
|  scripts    | 2021/8/12 21:5 |
|  src        | 2021/8/12 21:5 |
|  ..         | ..             |

css和js文件在这里



# 目录

Contents

- ◆ Window对象
- ◆ swiper 插件
- ◆ 本地存储
- ◆ 综合案例

### 3.1 本地存储特性

随着互联网的快速发展，基于网页的应用越来越普遍，同时也变的越来越复杂，为了满足各种各样的需求，会经常性在本地存储大量的数据，HTML5规范提出了相关解决方案。

- 1、数据存储在用户浏览器中
- 2、设置、读取方便、甚至页面刷新不丢失数据
- 3、容量较大，sessionStorage和localStorage约 5M 左右

### 3.2 localStorage

- 1、生命周期永久生效，除非手动删除 否则关闭页面也会存在
- 2、可以多窗口（页面）共享（同一浏览器可以共享）
3. 以键值对的形式存储使用

### 3.2 localStorage

存储数据:

```
localStorage.setItem(key, value)
```

获取数据:

```
localStorage.getItem(key)
```

删除数据:

```
localStorage.removeItem(key)
```

### 3.2 localStorage

#### 存储复杂数据类型存储

本地只能存储字符串,无法存储复杂数据类型.需要将复杂数据类型转换成JSON字符串,在存储到本地

#### JSON.stringify(复杂数据类型)

- 将复杂数据转换成JSON字符串 **存储** 本地存储中

#### JSON.parse(JSON字符串)

- 将JSON字符串转换成对象 **取出** 时候使用

### 3.3 sessionStorage (了解)

- 1、生命周期为关闭浏览器窗口
- 2、在同一个窗口(页面)下数据可以共享
- 3、以键值对的形式存储使用
- 4、用法跟localStorage 基本相同



# 目录

Contents

- ◆ Window对象
- ◆ swiper 插件
- ◆ 本地存储
- ◆ 综合案例



## 案例

## 本地存储学习信息案例

需求：改为本次存储版本的学习信息表

### 新增学生信息

姓名:  年龄:  性别:  薪资:  就业城市:

### 就业榜

| 学号   | 姓名   | 年龄 | 性别 | 薪资    | 就业城市 | 操作 |
|------|------|----|----|-------|------|----|
| 1001 | 欧阳霸天 | 19 | 男  | 20000 | 上海   | 删除 |
| 1002 | 令狐霸天 | 29 | 男  | 30000 | 北京   | 删除 |
| 1003 | 诸葛霸天 | 39 | 男  | 2000  | 北京   | 删除 |



## 案例

## 本地存储学习信息案例

需求：改为本次存储版本的学习信息表

分析：

需求①：读取本地存储数据（封装函数）

如果本地存储有数据，则返回 JSON.parse() 之后的对象

如果本地存储没有数据，则默认写入三条数据，注意存储的利用JSON.stringify() 存储JSON 格式的数据

需求②：渲染模块

先读取本地存储数据，然后渲染

需求③：添加模块

注意，先取的最新的本地存储数据，然后追加

新增了数据，要把新数据存储到本地存储别，忘记转换

## 案例

### 本地存储学习信息案例

需求：改为本次存储版本的学习信息表

分析：

需求④：删除模块

注意，先**取的**最新的本地存储数据，然后追加

新增了数据，要把新数据**存储**到本地存储别忘记转换

### 自定义属性

#### 固有属性:

标签天生自带的属性 比如class id title等, 可以直接使用点语法操作

#### 自定义属性:

由程序员自己添加的属性,在DOM对象中找不到, 无法使用点语法操作,必须使用专门的API

getAttribute('属性名') // 获取自定义属性

setAttribute('属性名', '属性值') // 设置自定义属性

removeAttribute('属性名') // 删除自定义属性

```
<div class="box"></div>
<script>
  let box = document.querySelector('.box')
  box.setAttribute('myid', 10)
  console.log(box.getAttribute('myid'))
</script>
</script>
console.log(box.getAttribute('myid'))
```

### 自定义属性

#### data-自定义属性:

传统的自定义属性没有专门的定义规则,开发者随意定值,不够规范,所以在html5中推出来了专门的data-自定义属性 在标签上一律以data-开头

在DOM对象上一律以dataset对象方式获取

```
<div class="box" data-id="10"></div>
<script>
    let box = document.querySelector('.box')
    console.log(box.dataset.id) // 10
</script>
```

```
<\script>
```



思考

1. 整理今天笔记
2. 作业 请把微博发布案例改为本地存储版本

今天多一份拼搏，明日多一份欢笑



传智教育旗下高端IT教育品牌