web APIs 第三天

BOM操作 (本地存储)

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

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综合案例

目录

Contents

- BOM操作
- 综合案例

BOM

- window对象
- 延时器、定时器
- location对象
- navigator对象
- histroy对象
- 本地存储

- 目标：学习 window 对象的常见属性，知道各个 BOM 对象的功能含义

JavaScript的组成

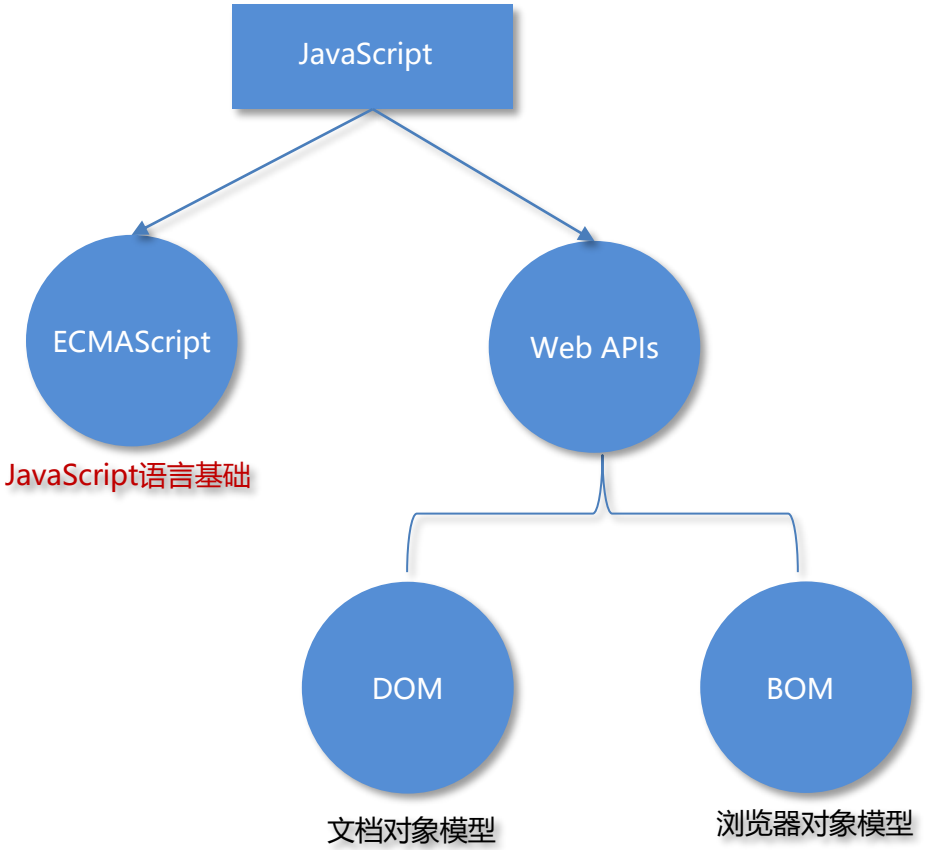
- **ECMAScript:**

  规定了js基础语法核心知识。

  - 比如：变量、分支语句、循环语句、对象等等

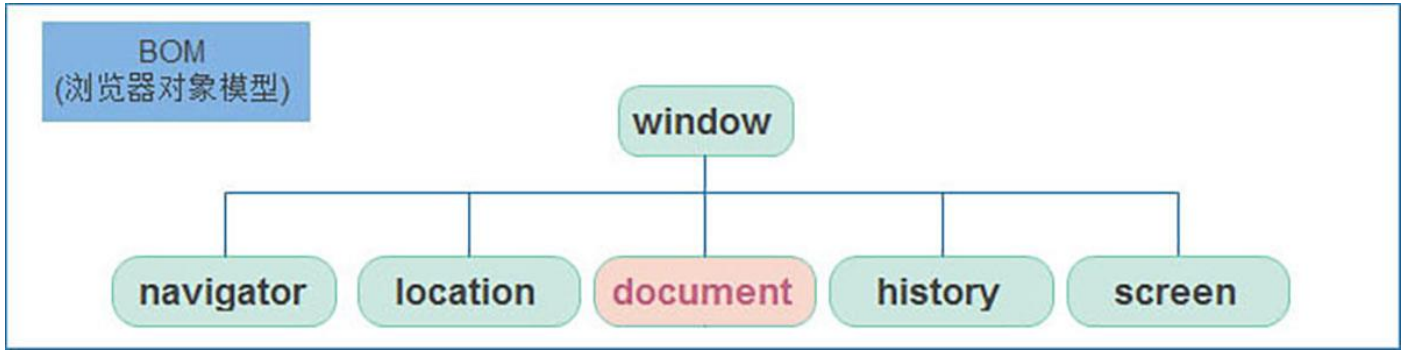
- **Web APIs :**

  - DOM 文档对象模型， 定义了一套操作HTML文档的API
  - BOM 浏览器对象模型，定义了一套操作浏览器窗口的API

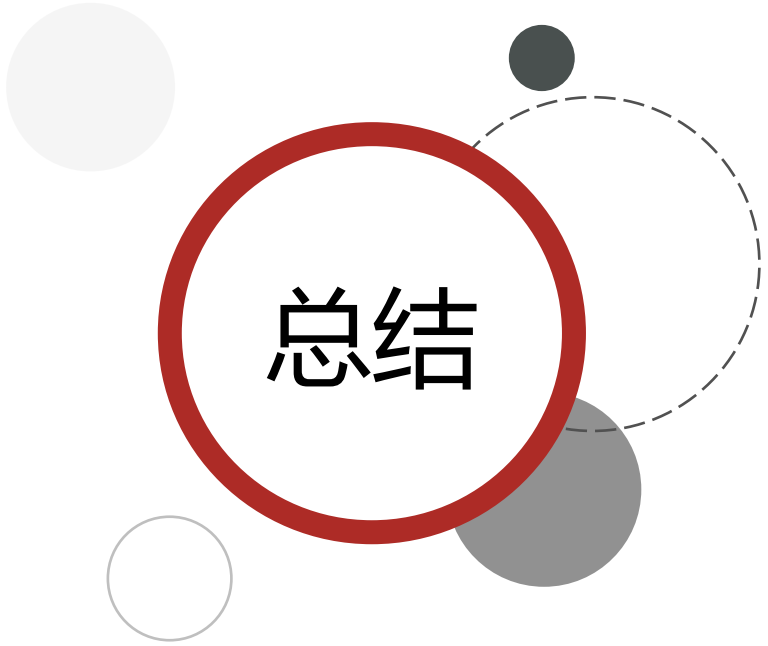


1.1 BOM

- BOM (Browser Object Model) 是浏览器对象模型



- window对象是一个全局对象，也可以说是JavaScript中的顶级对象
- 像document、alert()、console.log()这些都是window的属性，基本BOM的属性和方法都是window的
- 所有通过var定义在全局作用域中的变量、函数都会变成window对象的属性和方法
- window对象下的属性和方法调用的时候可以省略window



1. JavaScript有几部分组成的?
   - ECMAScript 定义基本js语法
   - web APIs （DOM、BOM）
2. BOM是什么?
   - 浏览器对象模型，定义了一套操作浏览器窗口的API
3. window对象是什么？ 平时可以省略吗?
   - 是一个全局对象，也可以说是JavaScript中的顶级对象
   - window对象下的属性和方法调用的时候可以省略window

BOM

- window对象
- 延时器、定时器
- location对象
- navigator对象
- histroy对象
- 本地存储

- 目标：学习 window 对象的常见属性，知道各个 BOM 对象的功能含义

1.2 延迟器

- 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
- **清除延时器**：

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

- **注意点**
  - 延时函数需要等待,所以后面的代码先执行
  - 返回值是一个正整数，表示定时器的编号

案例

5秒钟之后消失的广告

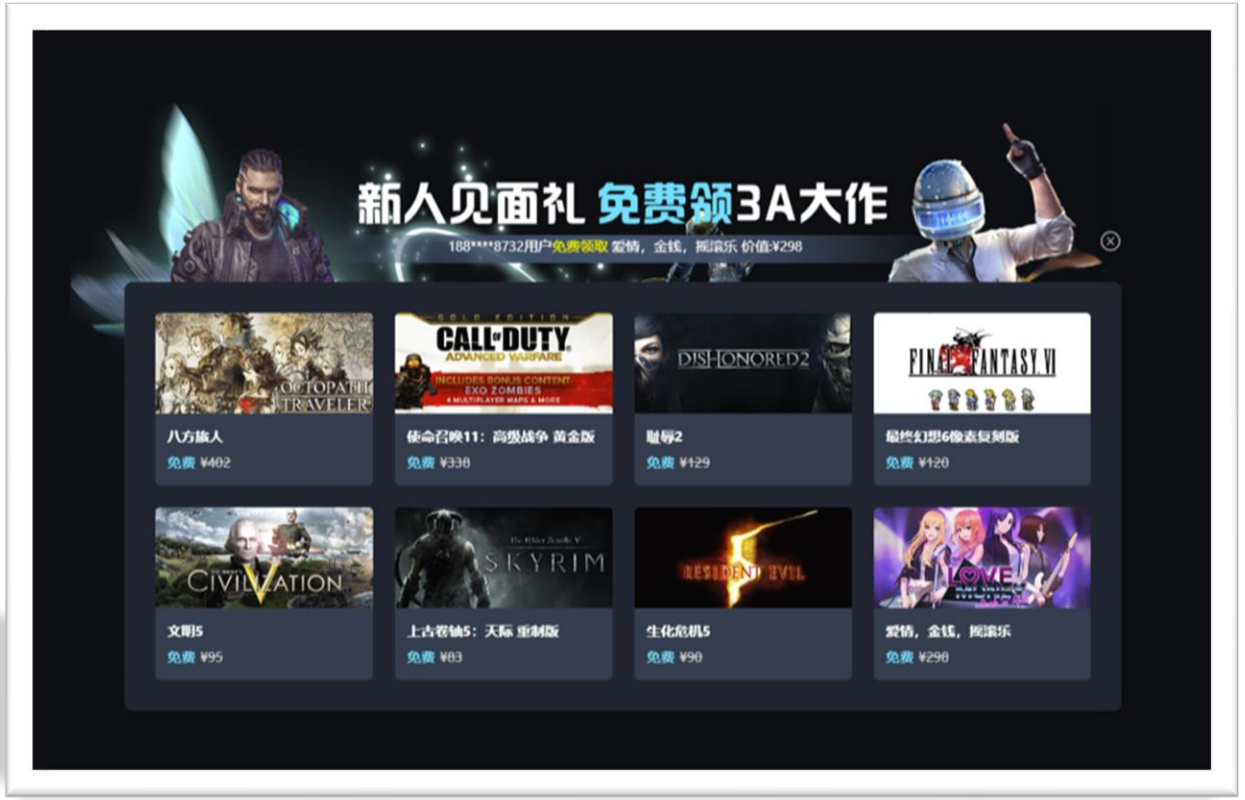
需求：5秒钟之后，广告自动消失

分析：

①：点击关闭按钮可以关闭

②：设置延迟器时间为5秒

③：调用点击事件 click()



定时器

- 网页中经常会需要一种功能：每隔一段时间需要自动执行一段代码，不需要我们手动去触发
- 例如：网页中的倒计时
- 要实现这种需求，需要定时器

定时器

定时器函数可以开启和关闭定时器

1. 开启定时器

```
setInterval(函数, 间隔时间)
```

- 作用：每隔一段时间调用这个函数
- 注意：间隔时间单位是毫秒

举例说明

```
function repeat() {
    console.log('前端程序员，就是头发多咋滴~~')
}
// 每隔一秒调用repeat函数
setInterval(repeat, 1000)
```

注意：

1. 函数名字**不需要加括号**
2. **定时器返回的是一个id数字**

定时器

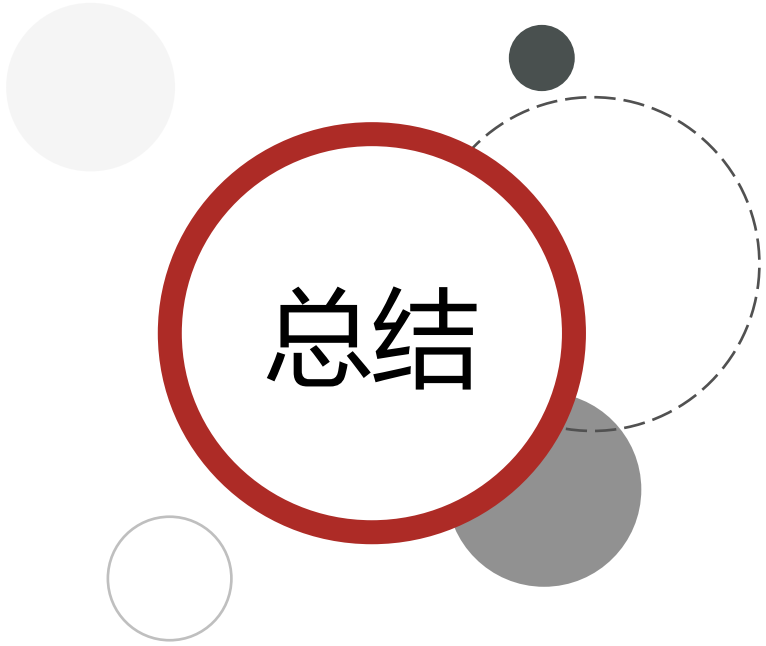
定时器函数可以开启和关闭定时器

2. 关闭定时器

```
let 变量名 = setInterval(函数, 间隔时间)
clearInterval(变量名)
```

一般不会刚创建就停止，而是满足一定条件再停止

```
let timer = setInterval(function() {
  console.log('hi~~')
}, 1000)
clearInterval(timer)
```



1. 定时器有什么作用?
   - 可以每隔指定时间自动重复执行某些代码
2. 定时器如何开启?
   - setInterval(函数名, 时间)
3. 定时器如何关闭?
   - 需要定时器变量名来关闭
   - 返回的是一个唯一的数字

```
let 变量名 = setInterval(函数, 间隔时间)
clearInterval(变量名)
```

案例

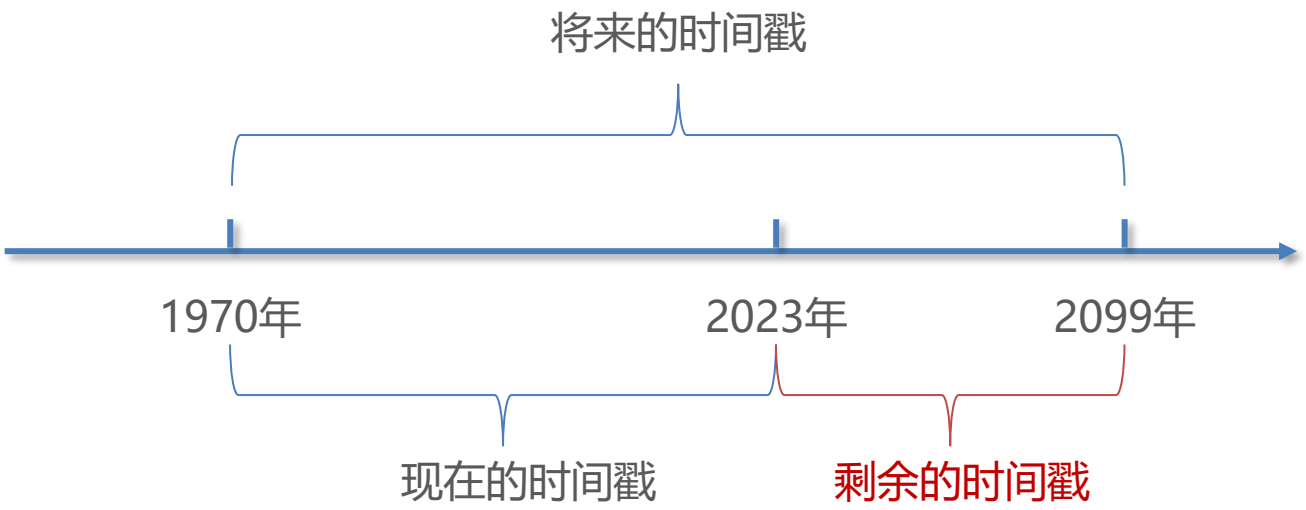
倒计时效果

需求：计算到下课还有多少时间（18:30）

需要 定时器 和 时间戳 一并实现

时间戳

- **什么是时间戳:**
  - 是指1970年01月01日00时00分00秒起至现在的总毫秒数(数字型)，它是一种特殊的计量时间的方式
- **使用场景：** 计算倒计时效果，需要借助于时间戳完成
- **算法:**
- 将来的时间戳 - 现在的时间戳 = 剩余时间毫秒数
- 剩余时间毫秒数转换为年月日时分秒 就是倒计时时间



时间戳

三种方式获取时间戳：

getTime()

日期对象来调用

```
const date = new Date()
console.log(date.getTime())
```

+new Date()

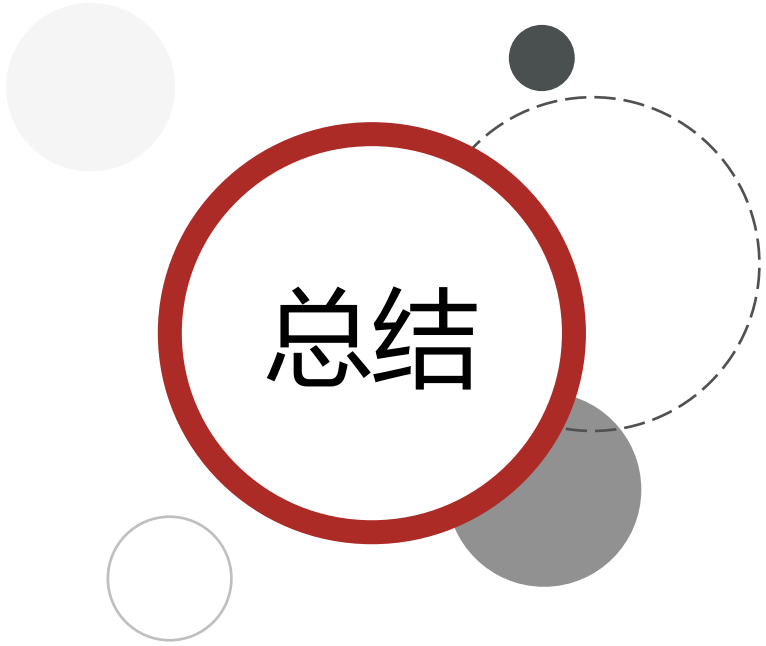
本质转换为数字

```
console.log(+new Date())
```

Date.now()

只能得到当前的时间戳

```
console.log(Date.now())
```



1. 为什么需要时间戳?
   - 计算倒计时效果，需要借助于时间戳完成
2. 时间戳是什么?
   - 是计量时间的方式，是指1970年01月01日00时00分00秒起至现在的总毫秒数
3. 获取时间戳有哪三种方式？重点记住那个?
   - date.getTime()
   - +new Date()　　使用较为简单
   - Date.now()

案例

倒计时效果

需求：计算到下课还有多少时间（18:30）

分析：

①：核心： 使用将来的时间戳减去现在的时间戳，封装函数 getTimer

②：把剩余的时间戳转换为 时、分、秒

③：把计算时分秒写到对应 span 盒子里面

④： 添加定时器效果自动变化时间

注意：

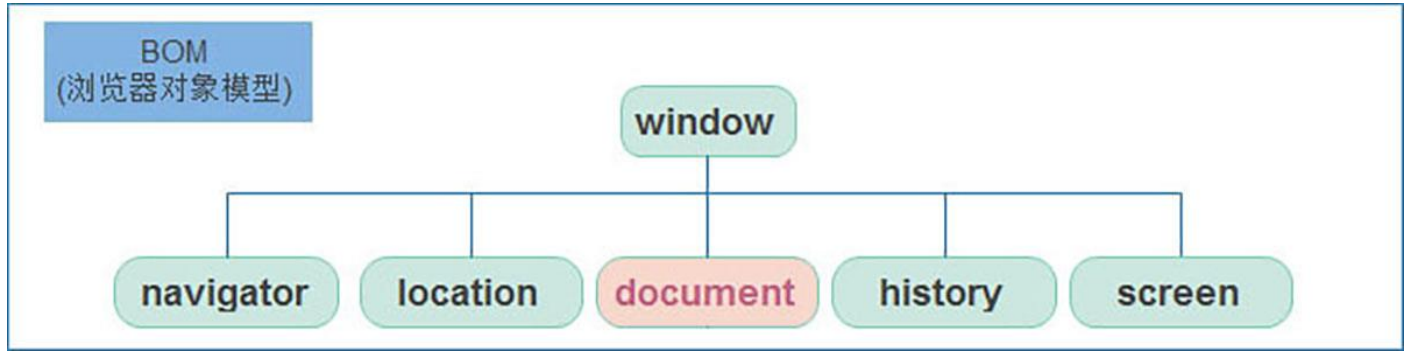
1. 通过时间戳得到是毫秒，需要转换为秒在计算
2. 转换公式：
   - h = parseInt(总秒数/ 60/60 %24) // 计算小时
   - m = parseInt(总秒数 /60 %60) // 计算分数
   - s = parseInt(总秒数%60) // 计算当前秒数

BOM

- window对象
- 延时器、定时器
- location对象
- navigator对象
- histroy对象
- 本地存储

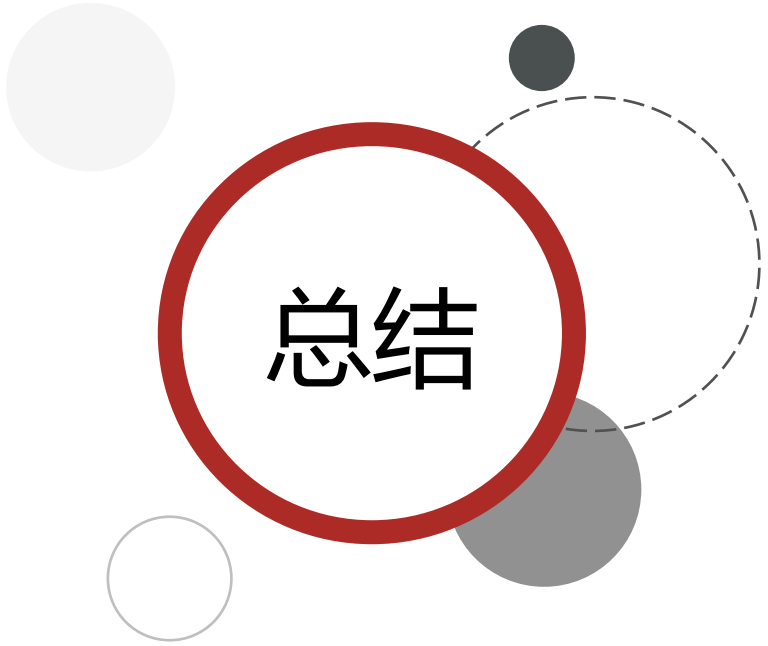
- 目标：学习 window 对象的常见属性，知道各个 BOM 对象的功能含义

1.3 location对象



- location (地址) 它拆分并保存了 URL 地址的各个组成部分， 它是一个对象
- **常用属性和方法:**

属性/方法	说明
href	属性，获取完整的 URL 地址，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



location 是对象，它拆分并保存了 URL 地址的各个组成部分

属性/方法	说明
href	属性，获取完整的 URL 地址，赋值时用于地址的跳转（重点）
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

案例

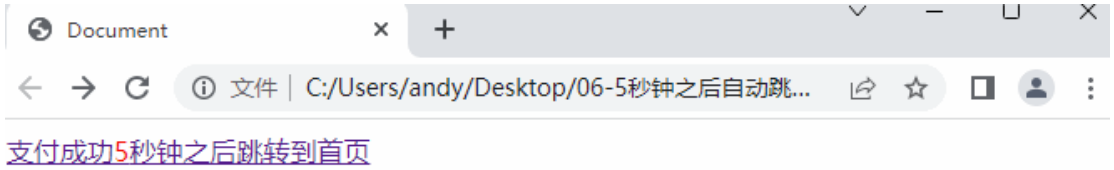
5秒钟之后跳转页面

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：利用定时器设置链接里面的数字倒计时

②：时间到了，关闭定时器同时自动跳转到新的页面 （location.href）



BOM

- window对象
- 定时器-延时函数
- location对象
- navigator对象
- histroy对象
- 本地存储

- 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'
    }
})();
```

BOM

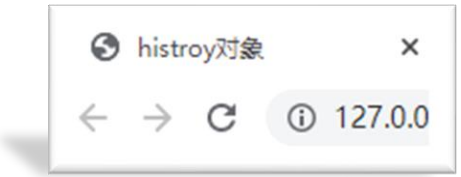
- window对象
- 定时器-延时函数
- location对象
- navigator对象
- histroy对象
- 本地存储

- 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 histroy对象

- history (历史)是对象，主要管理历史记录， 该对象与浏览器地址栏的操作相对应，如前进、后退等
- **使用场景**

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。



- **常见方法：**

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

BOM

- window对象
- 定时器-延时函数
- location对象
- navigator对象
- histroy对象
- 本地存储

- 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 本地存储（今日重点）

本地存储：将数据存储在本地浏览器中

常见的使用场景：

https://todomvc.com/examples/vanilla-es6/　页面刷新数据不丢失

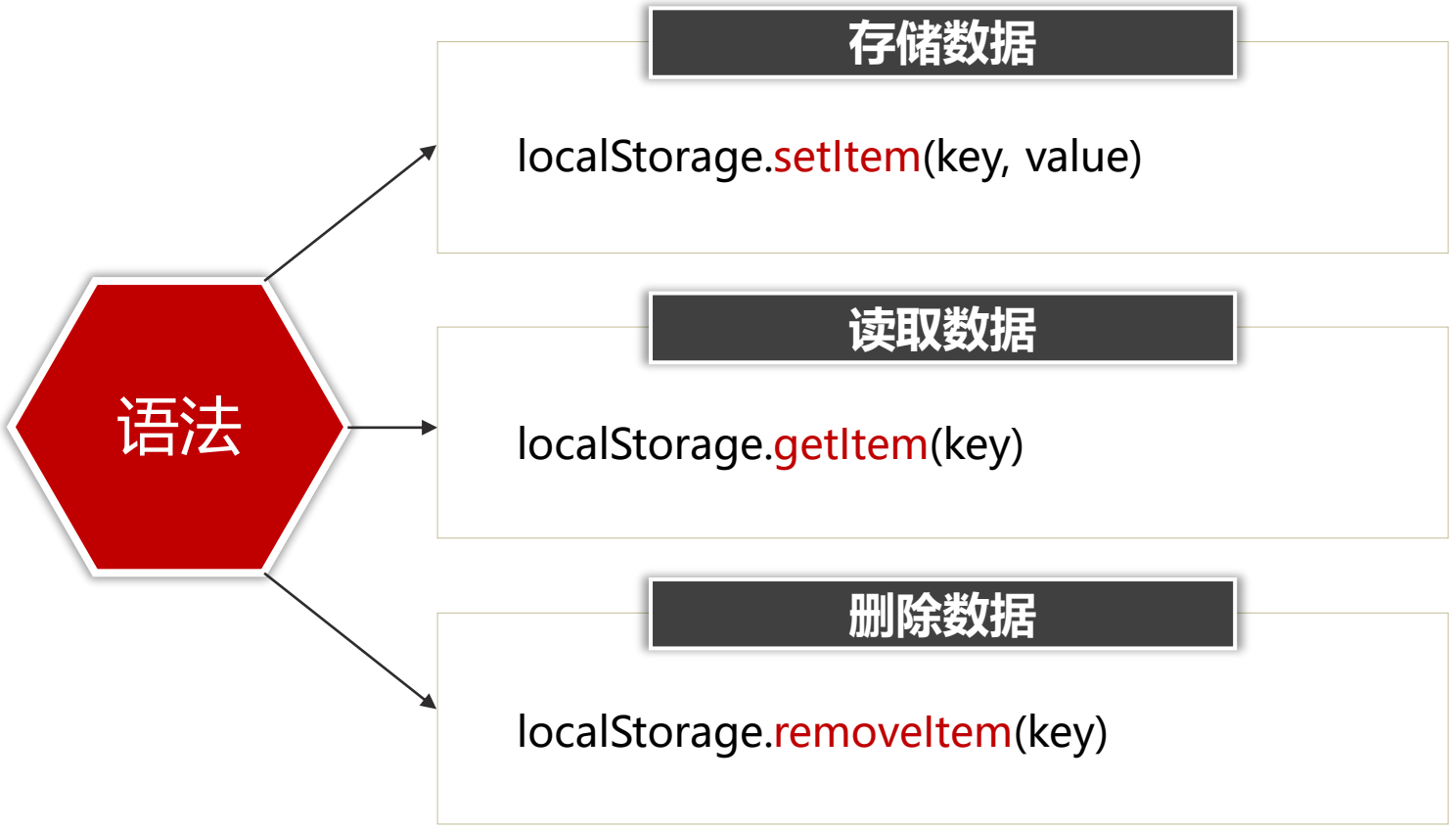
好处：

1、页面刷新或者关闭不丢失数据，实现数据持久化

2、容量较大，sessionStorage和 localStorage 约 5M 左右

1.6 本地存储分类- localStorage（重点）

- **作用:** 数据可以长期保留在本地浏览器中，刷新页面和关闭页面，数据也不会丢失
- **特性:** 以键值对的形式存储，并且存储的是字符串， 省略了window



1.6 本地存储分类- sessionStorage（了解）

- **特性：**
  - 用法跟localStorage 基本相同
  - 区别是：当页面浏览器被关闭时，存储在 sessionStorage 的数据会被清除

存储：sessionStorage.setItem(key, value)

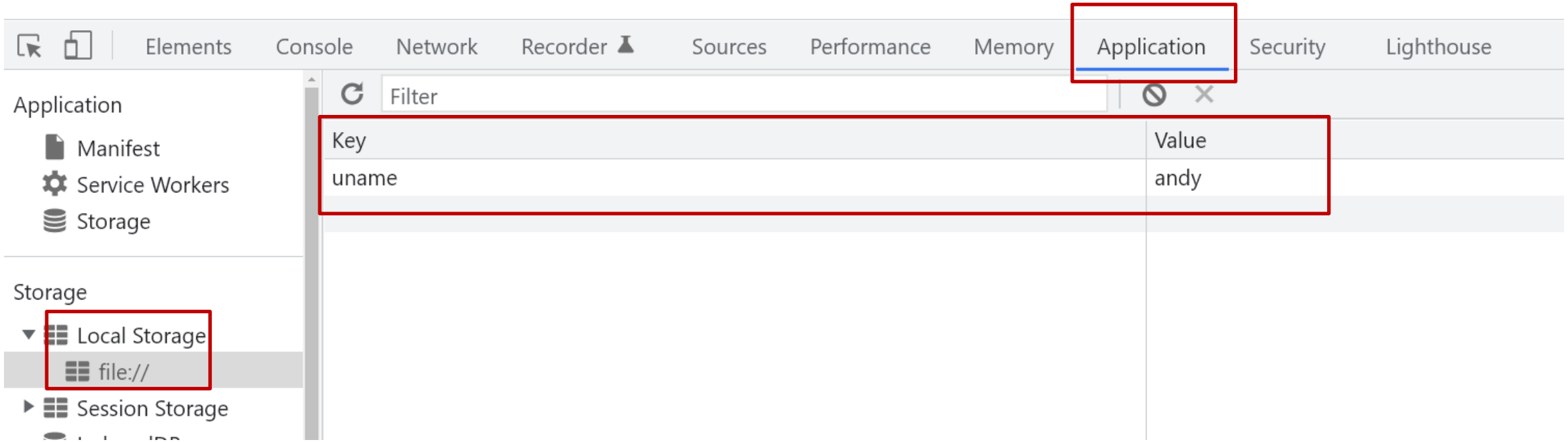
获取：sessionStorage.getItem(key)

删除：sessionStorage.removeItem(key)

1. localStorage 作用是什么?
   - 数据可以长期保留在本地浏览器中，刷新页面和关闭页面，数据也不会丢失
2. localStorage 存储，获取，删除的语法是什么?
   - 存储：localStorage.setItem(key, value)
   - 获取：localStorage.getItem(key)
   - 删除：localStorage.removeItem(key)

1.6 本地存储分类- localStorage

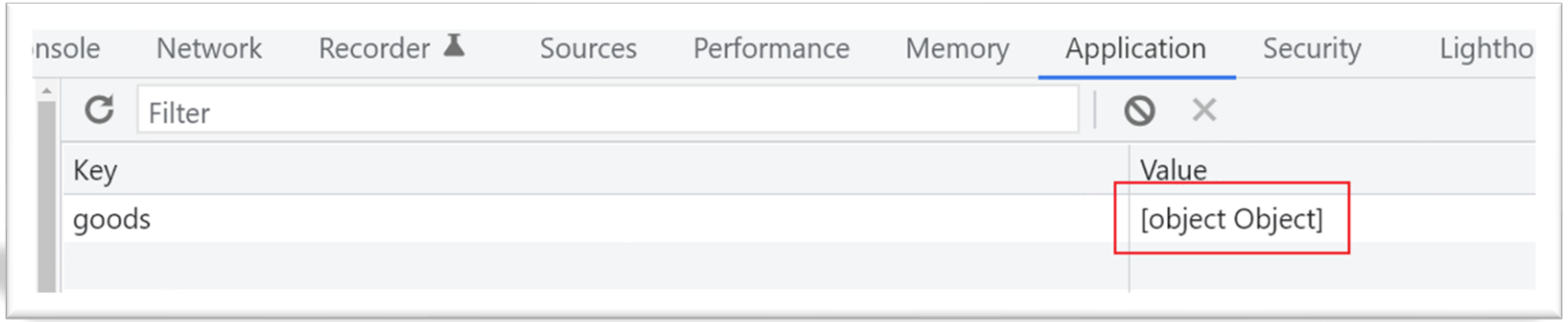
- **浏览器查看本地数据:**



1.6 localStorage 存储复杂数据类型

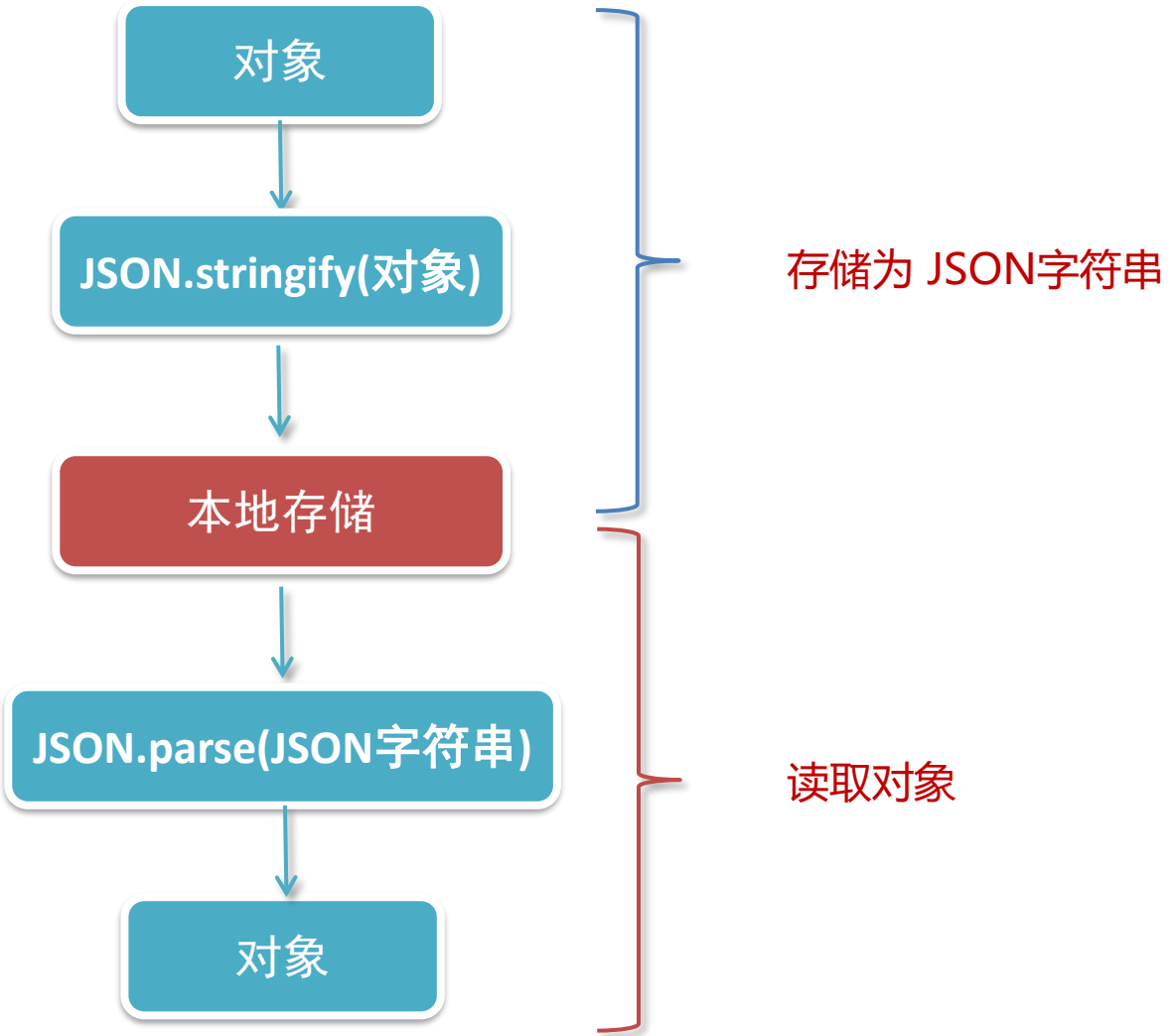
- **问题：**本地只能存储字符串,无法存储复杂数据类型.

```
const goods = {
  name: '小米10',
  price: 1999
}
localStorage.setItem('goods', goods)
```



1.6 本地存储分类- localStorage

- **解决方案：**



1.6 本地存储分类- localStorage

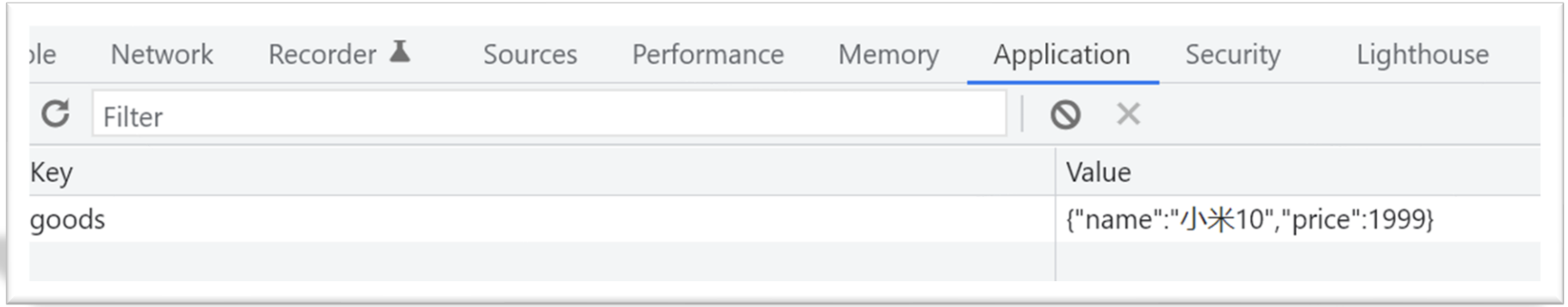
- **解决：**需要将复杂数据类型转换成 JSON字符串,在存储到本地
- **语法：JSON.stringify(复杂数据类型)**

```
const goods = {
  name: '小米10',
  price: 1999
}
localStorage.setItem('goods', JSON.stringify(goods))
```

JSON字符串：

- 首先是1个字符串
- 属性名使用双引号引起来，不能单引号
- 属性值如果是字符串型也必须双引号

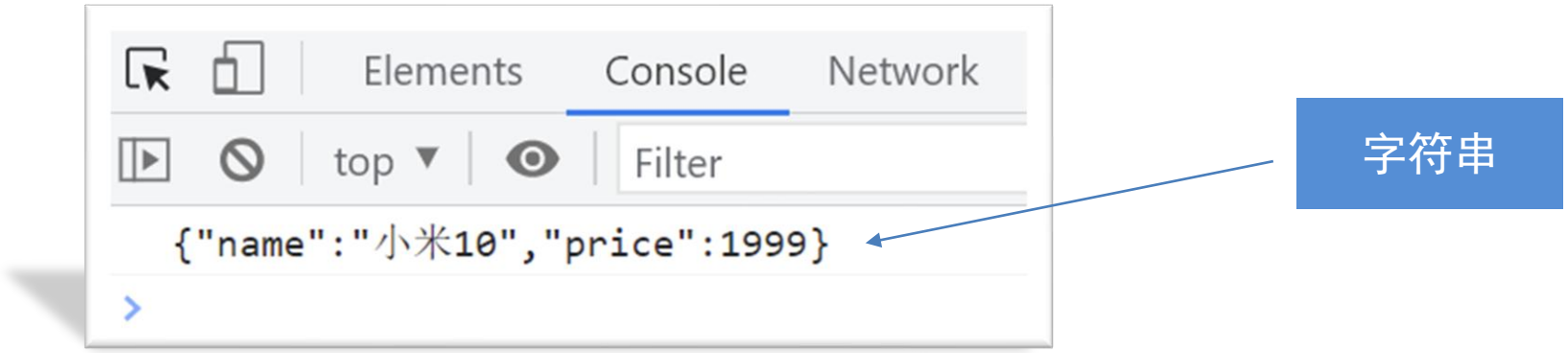
- 将复杂数据转换成JSON字符串 **存储** 本地存储中



1.6 本地存储分类- localStorage

- **问题：**因为本地存储里面取出来的是字符串，不是对象，无法直接使用

```
const obj = localStorage.getItem('goods')
console.log(obj)
```

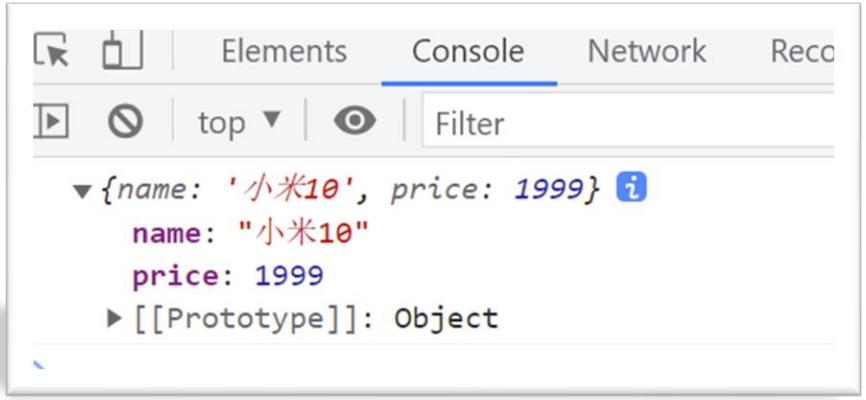


1.6 本地存储分类- localStorage

- **解决：**把取出来的字符串转换为对象
- **语法：JSON.parse(JSON字符串)**

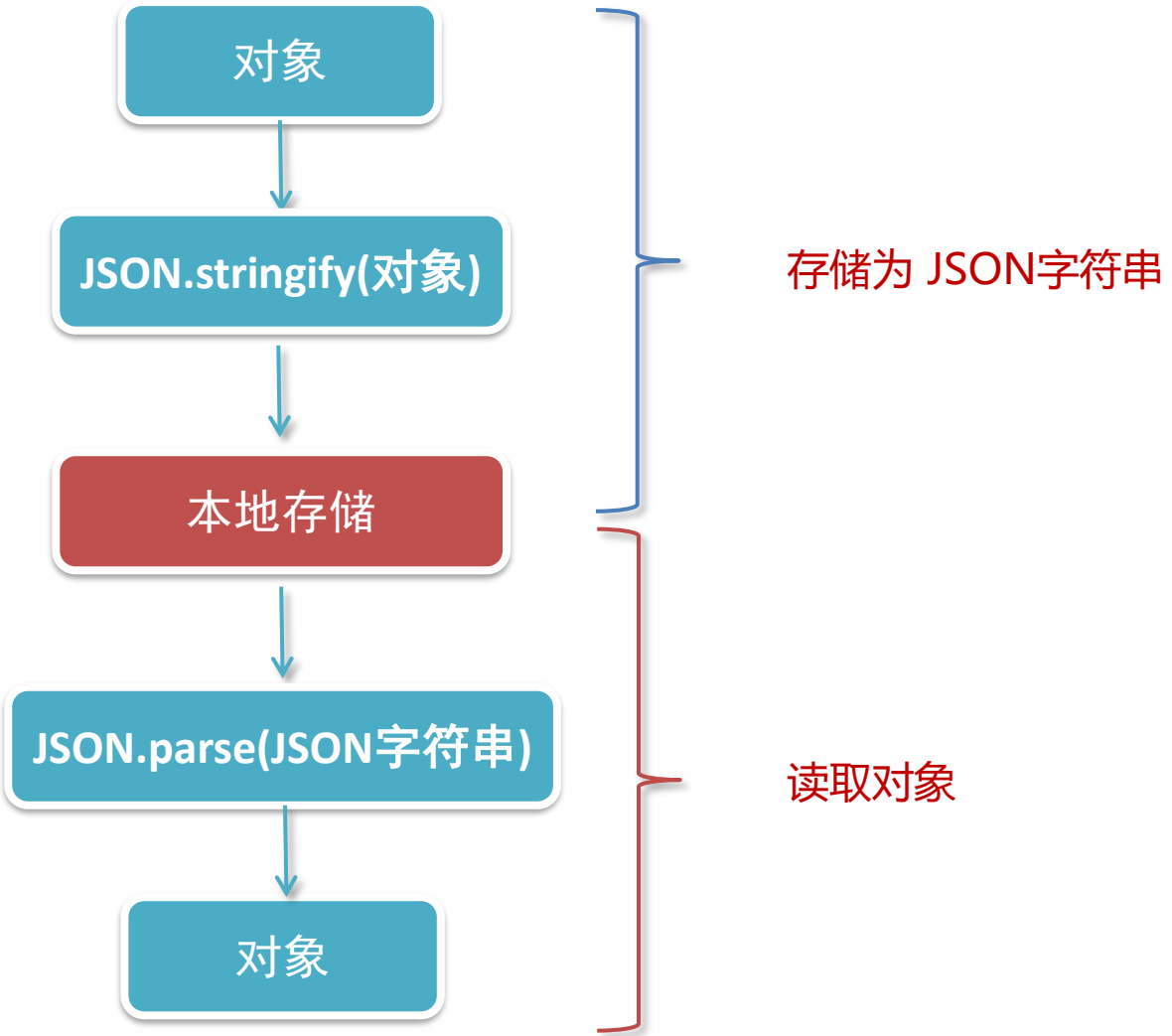
```
const obj = JSON.parse(localStorage.getItem('goods'))
console.log(obj)
```

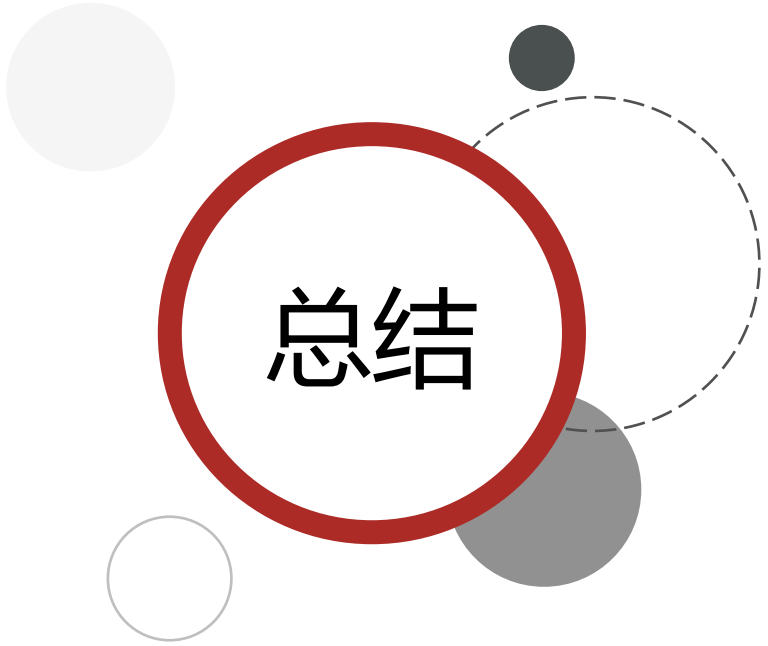
➢ 将JSON字符串转换成对象　　**取出** 时候使用



1.6 本地存储分类- localStorage

- **解决方案：**





1. localStorage 如何存储复杂数据类型?

   - 把对象转换为JSON字符串。 JSON.stringify(复杂数据类型)

```
const goods = {
  name: '小米10',
  price: 1999
}
localStorage.setItem('goods', JSON.stringify(goods))
```

2. localStorage 如何读取复杂数据类型?

   - 把JSON字符串转换为对象。JSON.parse()

```
const obj = JSON.parse(localStorage.getItem('goods'))
console.log(obj)
```

目录

Contents

- BOM操作
- 综合案例

案例

历史搜索记录案例

需求：本地存储历史搜索记录案例

业务：

①：在输入框中按键为回车的时候，本地存储该搜索记录

- 并且需要在历史搜索中展示该搜索记录
- 最新搜索记录在最前面

②：点击清除历史记录，清空所有的历史搜索记录

③：历史搜索需要去重

步骤

历史搜索记录案例

需求：本地存储历史搜索记录案例

业务1分析：输入框中按键为回车 功能

①：准备一个数组，用于存储搜索记录

②：监听输入框的keyup事件

③：当按键为Enter的时候，获取输入框内容，添加到数组中

- 将数组渲染展示到历史搜索列表中
- 并将数组进行本地存储
- 清空输入框内容

④：封装 renderHistory 函数，用于渲染历史搜索列表

步骤

历史搜索记录案例

需求：本地存储历史搜索记录案例

业务2分析：点击清除历史记录，清空所有的历史搜索记录

①：给清除历史记录按钮的父级元素注册click事件（事件委托）

②：如果点击的当前元素是 清除历史记录按钮，就清空数组

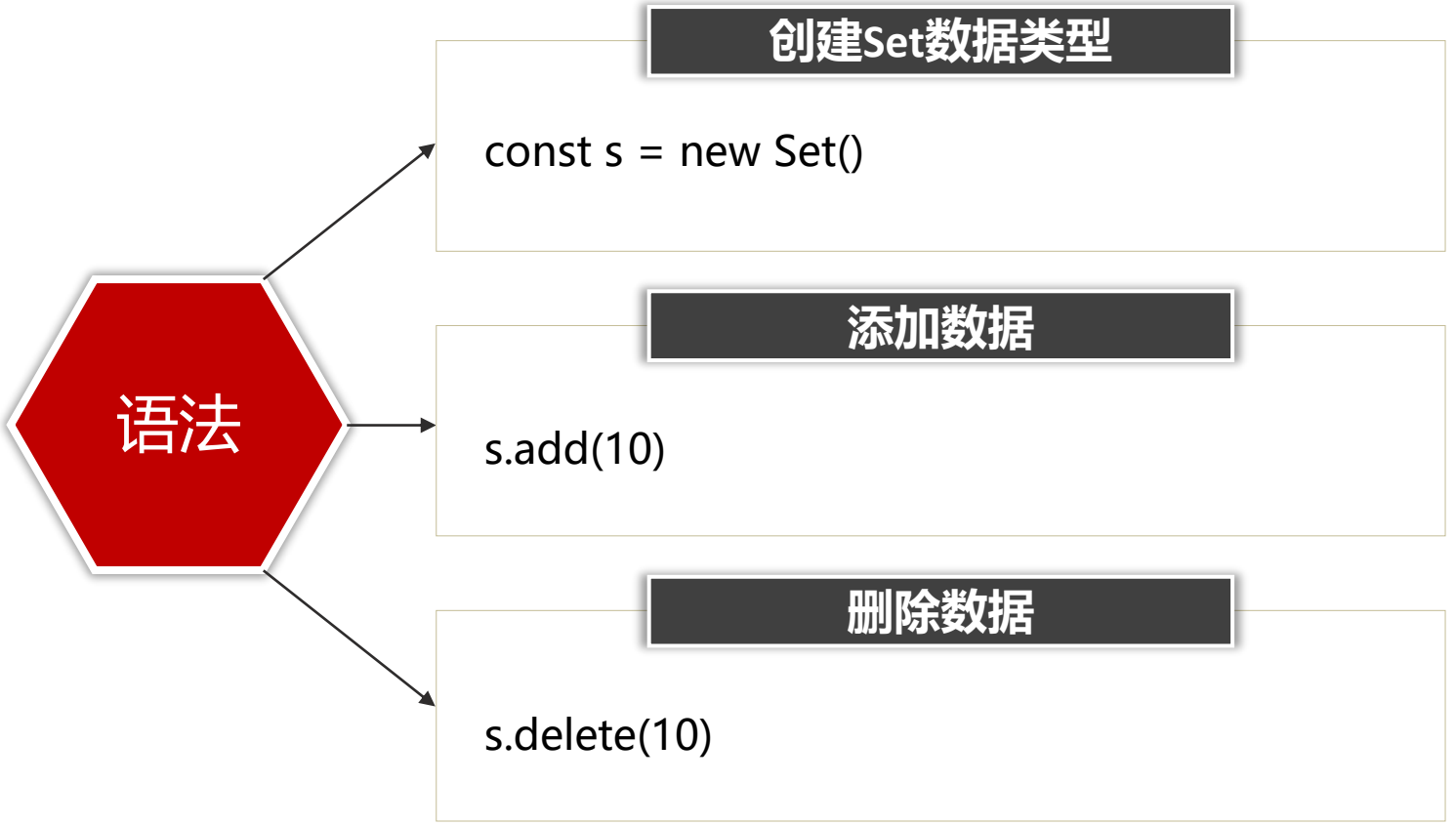
- 渲染历史搜索列表
- 以及本地存储存储清空后的数组

③：在renderHistory 函数，添加判断显示隐藏 清除历史记录

Set 数据类型

常见的使用场景：数组去重

典型特征：不允许包含重复的元素



实现数组的去重: [...new Set(数组)]

步骤

历史搜索记录案例

需求：本地存储历史搜索记录案例

业务3分析：历史搜索记录去重

①：在按键回车的时候，需要对搜索进行去重

- 历史搜索记录数据在数组中，需要给数组去重

②：把数组传递给Set数据类型进行去重 new Set(数组)

③：得到了 去重后的 Set类型的数据，将其展开到 新数组中，得到去重后的数组数据

传智教育旗下高端IT教育品牌