JavaScript 阶段

JavaScript核心

7天

Web APIs

6天

基础：

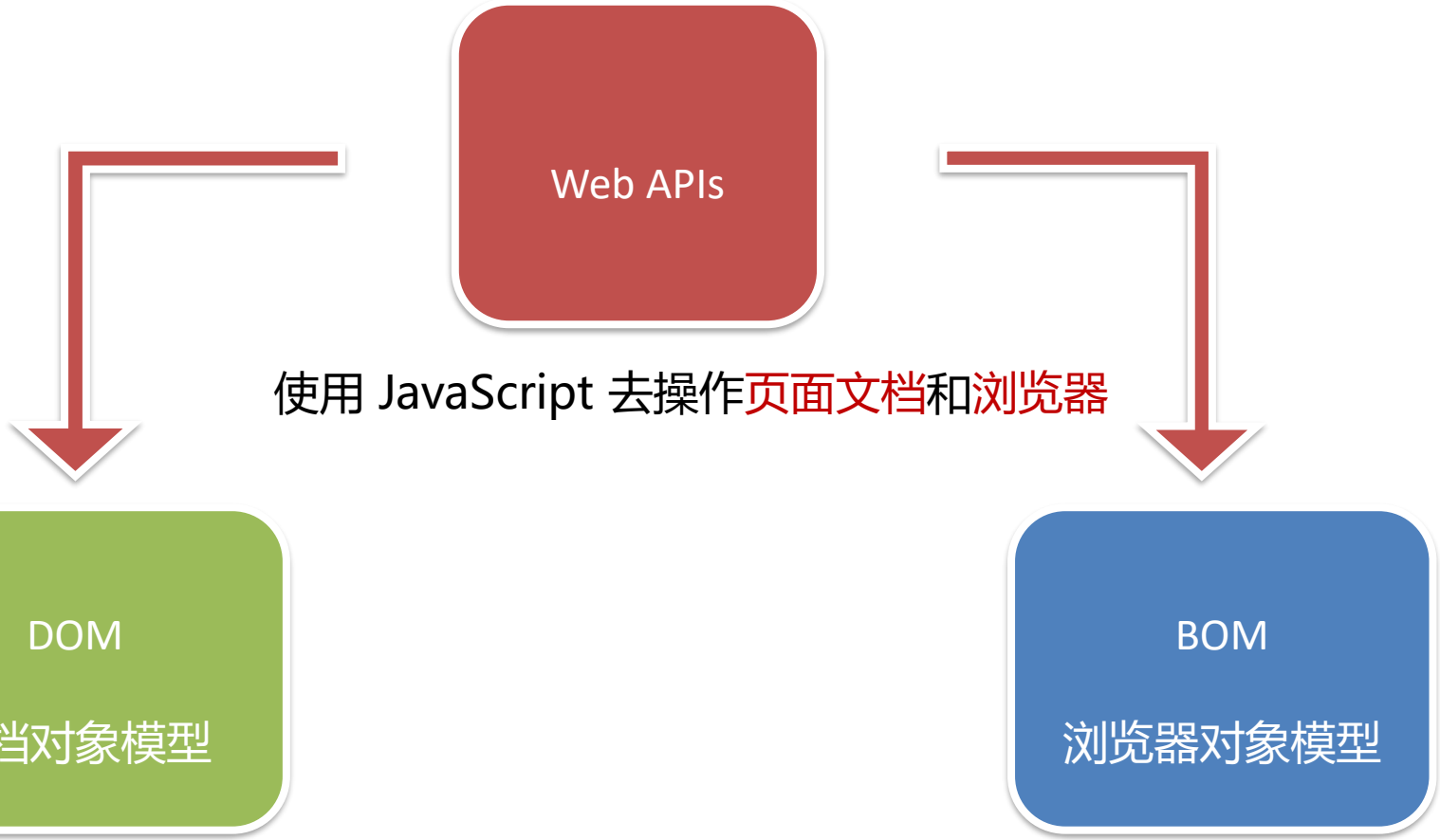
基本写法，知道怎么控制计算机去执行代码....

高级语法，简化代码

应用

操作页面元素

实现后台管理的操作



使用 JavaScript 去操作页面文档和浏览器

使用 JavaScript 去操作页面文档

使用 JavaScript 去操作浏览器


什么是 API ？

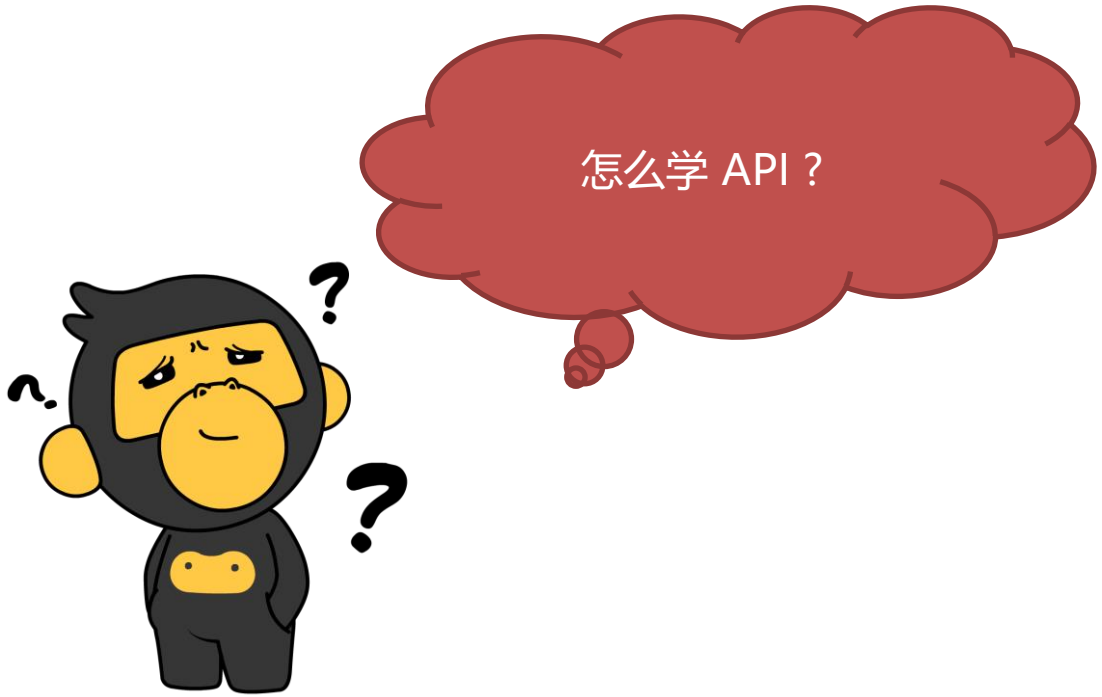
什么是API?

- **API**: 应用程序接口（Application Programming Interface）
- 接口：无需关心内部如何实现，程序员只需要调用就可以很方便实现某些功能

什么是API?

- **API**: 应用程序接口（Application Programming Interface）
- **作用**：开发人员使用 JavaScript提供的接口来操作网页元素和浏览器

```
alert('我会弹出一个警示框')
```


怎么学 API？

怎么学习API?

好习惯

理解API（是啥）

- 它能做什么?
- 它能解决什么需求?

查阅文档（咋用）

- API里面包含哪些属性和方法?
- 有哪些注意事项
- 怎么使用?
- MDN 强烈推荐
- https://developer.mozilla.org/zh-CN/

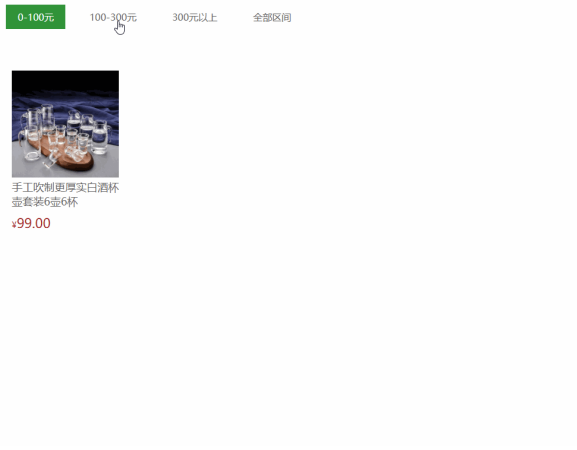

API 怎么安排的呢？

Web APIs 课程安排 6天

第一天

DOM-操作元素

获取元素、修改内容、属性、注册事件



第二天

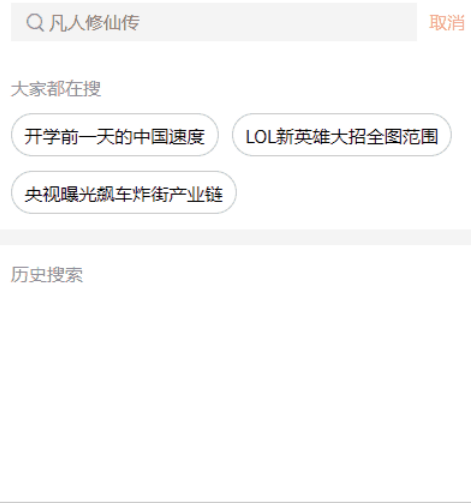
DOM-事件核心

事件类型、事件对象、事件流

第三天

BOM-操作浏览器

BOM、本地存储



Web APIs 课程安排 6天

第四天

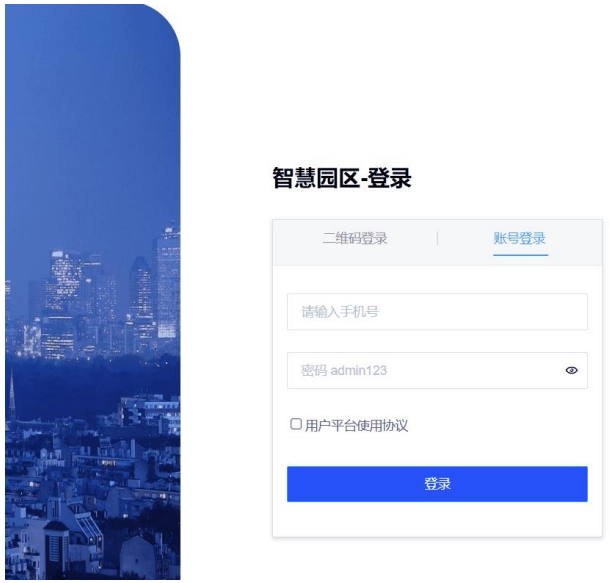
正则表达式

正则表达式、综合案例

第五天

元素尺寸、位置、节点操作

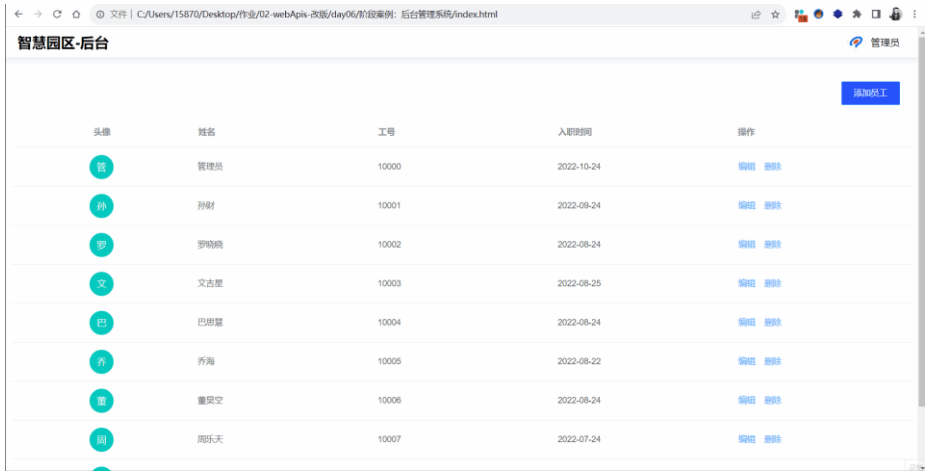
后台管理系统 – 登录页



第六天

阶段实战案例

后台管理系统



web APIs 第一天

获取元素、操作元素、注册事件

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

描述	属性/方法	效果
获取DOM对象	document.querySelector()	获取指定的第一个元素
	document.querySelectorAll()	获取指定的所有元素
操作元素内容	元素.innerText	操作元素内容，不解析标签
	元素.innerHTML	操作元素内容，解析标签
事件监听	元素.addEventListener()	事件监听，事件绑定，事件注册
环境对象	this	谁调用，指向谁
操作元素属性	常用属性	src、title、value、checked、disable
	样式属性：元素.style	通过style操作样式
	样式属性：元素.className	通过类名操作样式
	样式属性：元素.classList	add添加、remove移除、toggle切换类名
	自定义属性	结构中data-*，JS中 dataset操作自定义属性

学习目标

Learning Objectives

1. 掌握DOM属性操作，完成元素内容设置
2. 元素属性设置，控制元素样式
3. 掌握事件绑定，完成常见网页交互

目录

Contents

DOM 简介

- 什么是DOM

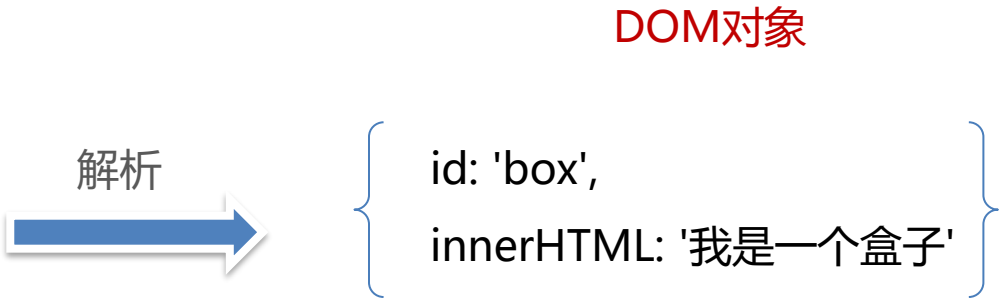
1. 什么是DOM

- DOM（Document Object Model——文档对象模型）
- 作用：DOM用来 操作网页文档，开发网页特效和实现用户交互
- DOM的核心思想就是把网页内容当做对象来处理，通过对象的属性和方法对网页内容操作

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>页面</title>
</head>
<body>

  <div id="box">我是一个盒子</div>

</body>
</html>
```



1. 什么是DOM

- DOM（Document Object Model——文档对象模型）
- 作用：DOM用来 操作网页文档，开发网页特效和实现用户交互
- DOM的核心思想就是把网页内容当做对象来处理，通过对象的属性和方法对网页内容操作

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>页面</title>
</head>
<body>

  <div id="box">我是box</div>

</body>
</html>
```

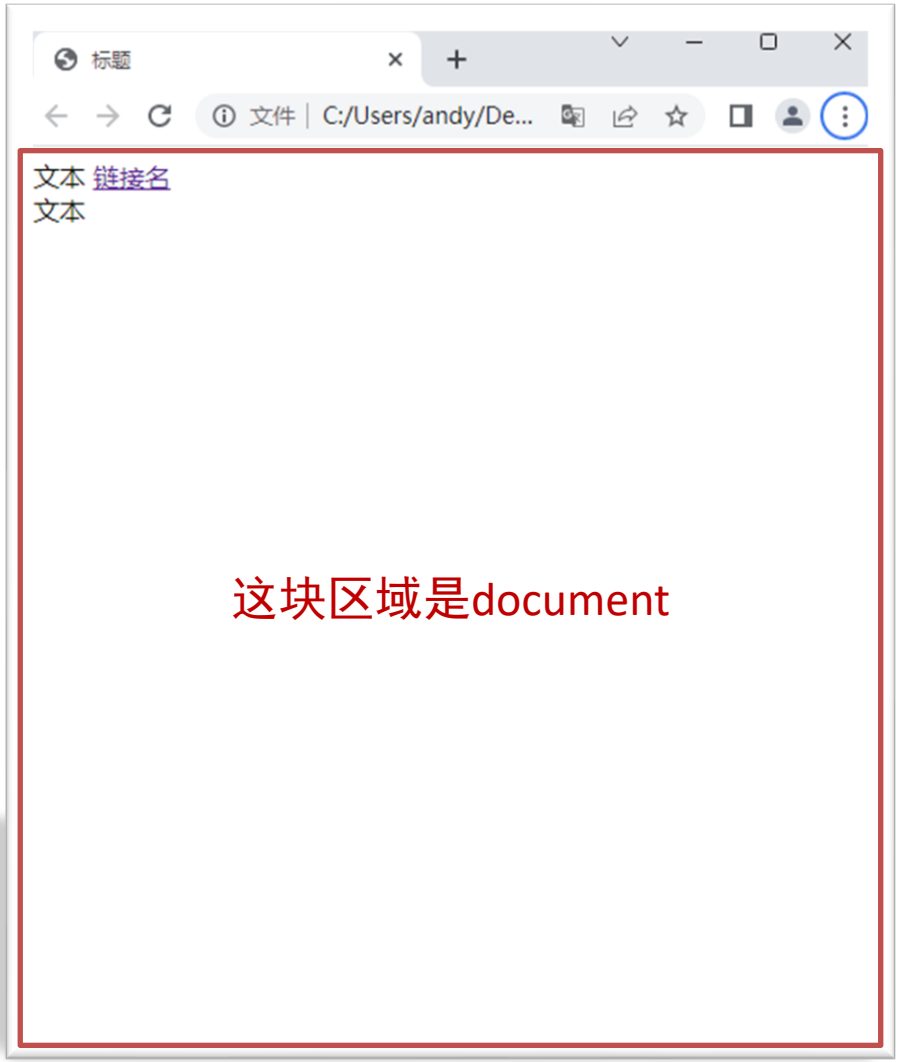
DOM对象

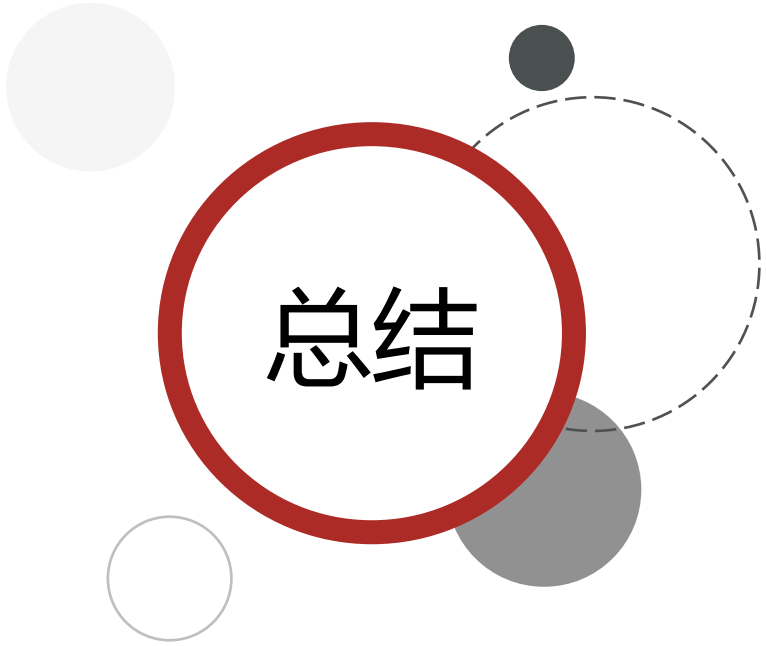
```
id: 'box',
innerHTML: '我是一个盒子'
```

对象.innerHTML = '我是box'

1. 什么是DOM

- document 对象
  - 是 DOM 里提供的一个**对象，**是DOM顶级对象
  - 作为网页内容的入口
  - 所以它提供的属性和方法都是**用来访问和操作网页内容的**
    - ✓ 例：document.querySelector()





1. DOM是什么？作用是什么？
   - 文档对象模型
   - 操作网页文档，开发网页特效和实现用户交互
2. DOM核心思想是什么？
   - DOM核心就是把网页内容当做对象来处理
   - 比如DOM会把标签解析成 DOM对象
   - 修改这个对象的属性会自动修改到标签身上，从而可以修改标签的结构、样式或者内容
3. DOM的顶级对象是 document 对象

目录

Contents

02

获取DOM元素

- 根据CSS选择器来获取DOM元素（重点）
- 其他获取DOM元素方法（了解）

2. 获取DOM元素

想要操作页面元素，那我们需要先利用DOM方式来获取（选择）这个元素

1. CSS选择器来获取DOM元素（重点）

document.querySelector()
document.querySelectorAll()

2. 其他获取DOM元素（了解）

document.getElementById()
document.getElementsByClassName()
document.getElementsByTagName()
document.getElementsByName()

2.1 根据CSS选择器来获取DOM元素（重点）

1. 选择匹配的第一个元素

语法：

```
document.querySelector('css选择器')
```

参数：

包含一个或多个有效的CSS选择器 字符串

返回值：

CSS选择器匹配的第一个元素**对象**

如果没有匹配到，则返回 null

多参看文档：https://developer.mozilla.org/zh-CN/docs/Web/API/Document/querySelector

2.1 根据CSS选择器来获取DOM元素（重点）

2. 选择匹配的多个元素对象

语法：

```
document.querySelectorAll('css选择器')
```

参数：

包含一个或多个有效的CSS选择器 **字符串**

返回值：

CSS选择器匹配的NodeList 伪数组

```
document.querySelectorAll('ul li')
```

2.1 伪数组

```
document.querySelectorAll('css选择器')
```

得到的是一个**伪数组**：

- 有长度有索引号的数组
- 但是没有 pop() push() 等数组方法

想要得到里面的每一个对象，则需要遍历（for）的方式获得

注意事项

哪怕只有一个元素，通过querySelectAll() 获取过来的也是一个**伪数组**，里面只有一个元素而已

1. DOM中获取页面中的元素我们最常用哪两种方式?
   - querySelectorAll()
   - querySelector()
   - 参数是利用css选择器实现，注意是字符串类型
2. 他们两者的区别是什么?
   - querySelector() 只能选择第一个元素
   - querySelectorAll() 可以选择多个元素，得到的是伪数组，需要遍历得到每一个元素
3. 什么是伪数组?
   - 外形是数组，并且有索引号和长度
   - **但是没有一些数组常用的方法，比如 pop 、push等**

2.2 其他获取DOM元素方法（了解）

语法	实例	描述
getElementById	document.getElementById('box')	根据id获取元素，单个元素
getElementsByTagName	document.getElementsByTagName('li')	根据标签名获取元素，伪数组
getElementsByClassName	document.getElementsByClassName('one')	根据类名获取元素，伪数组
getElementsByName	document.getElementsByName('sex')	根据name属性值获取元素，伪数组

目录

Contents

操作元素内容

- 对象.innerText 属性
- 对象.innerHTML 属性

三、操作元素内容

- DOM对象可以操作页面标签，所以本质上就是操作DOM对象（增删改查）
- 如果想要操作标签元素的**内容**，则可以使用如下2种方式：
  1. 对象.innerText 属性
  2. 对象.innerHTML 属性

三、操作元素内容

1. 元素.innerText 属性

- "渲染"文本内容到标签里面
- 显示纯文本，不解析标签

举例说明

```
// 首先需要获取这个dom对象
const box = document.querySelector('.box')
//  改
box.innerText = '迪丽热巴'
```

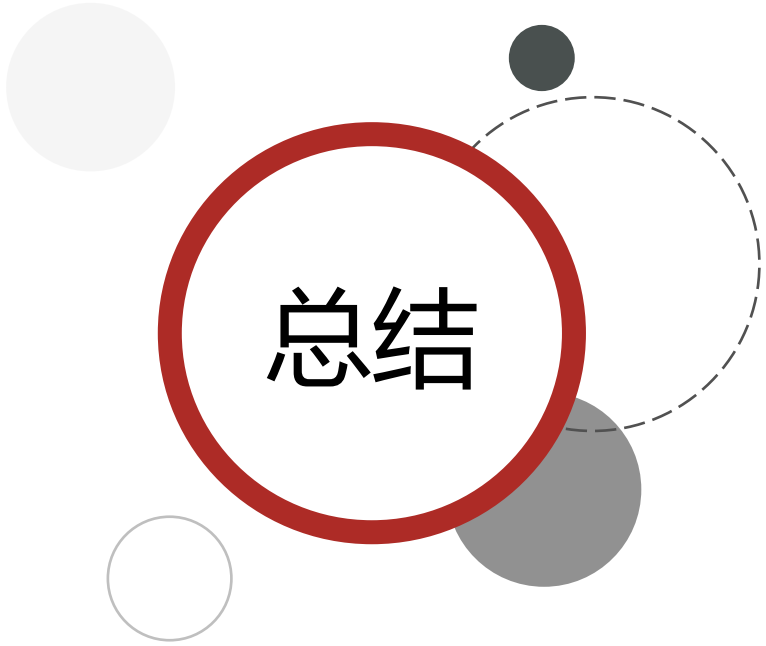
三、操作元素内容

2. 元素.innerHTML 属性

- "渲染"文本内容到标签里面
- 会解析标签

举例说明

```
// 2. 对象.innerHTML  会解析标签
box.innerHTML = '<strong>迪丽热巴</strong>'
```



1. 操作DOM元素内容我们学了哪两种方式?
   - 元素对象.innerText 属性
   - 元素对象.innerHTML 属性
2. 两者的区别是什么?
   - 元素.innerText 属性　只识别文本，不能解析标签
   - 元素.innerHTML 属性　能识别文本，能够解析标签
   - 如果还在纠结到底用谁，你可以选择innerHTML

案例

年会抽奖案例

需求：从数组随机抽取一等奖、二等奖和三等奖，显示到对应的标签里面

业务分析：

①：页面刷新随机抽取获奖姓名

②：不允许重复得奖

案例

年会抽奖案例

需求：从数组随机抽取一等奖、二等奖和三等奖，显示到对应的标签里面

分析：

①：**一等奖**:利用随机数选取数组中的名字（Math.random()）

②：把名字放入span里面 （innerHTML 或 innerText）

③：不允许重复抽奖，所以要把刚才选出来的名字从数组中删除（ splice ）

④： **二等奖、三等奖**依次类推同样操作

目录

Contents

事件监听

- 事件监听
- 环境对象-this
- 回调函数
- 拓展阅读-事件监听版本

事件

- 以前写的代码都是自动执行的，我们希望一段代码在某个特定的时机才去执行，比如
  - 点击按钮可以弹出警示框
  - 比如鼠标经过显示下拉菜单等等

事件

- 事件是程序在运行的时候，发生的特定动作或者特定的事情
- 比如点击按钮、
- 比如鼠标经过菜单等等

通常，当事件发生时，可以做些事情

比如点击按钮，可以 弹出警示框

比如鼠标经过某个盒子，可以 显示下拉菜单

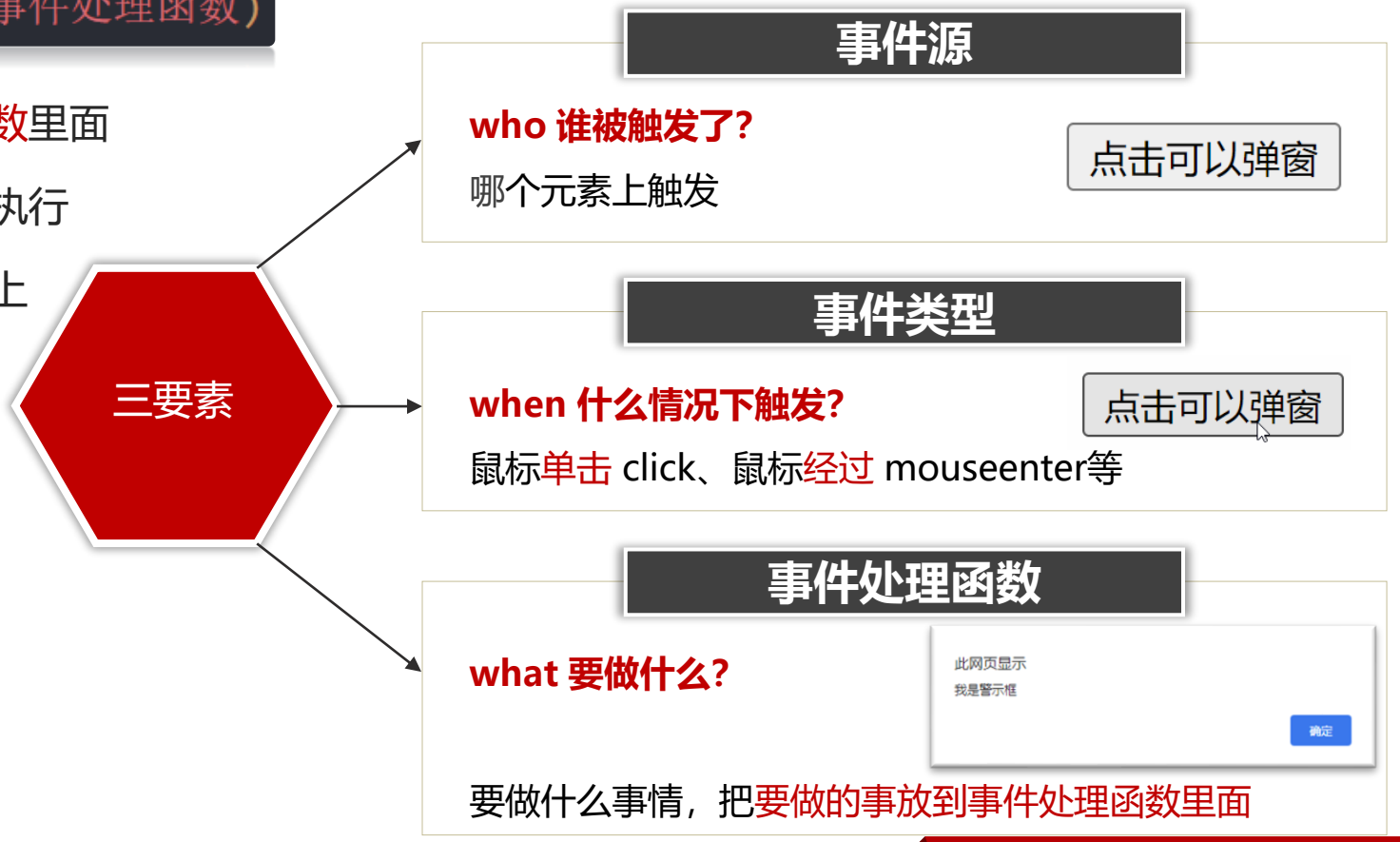
那我们怎么实现事件触发时 JavaScript 可以执行一些代码呢?

事件监听

- **语法：**

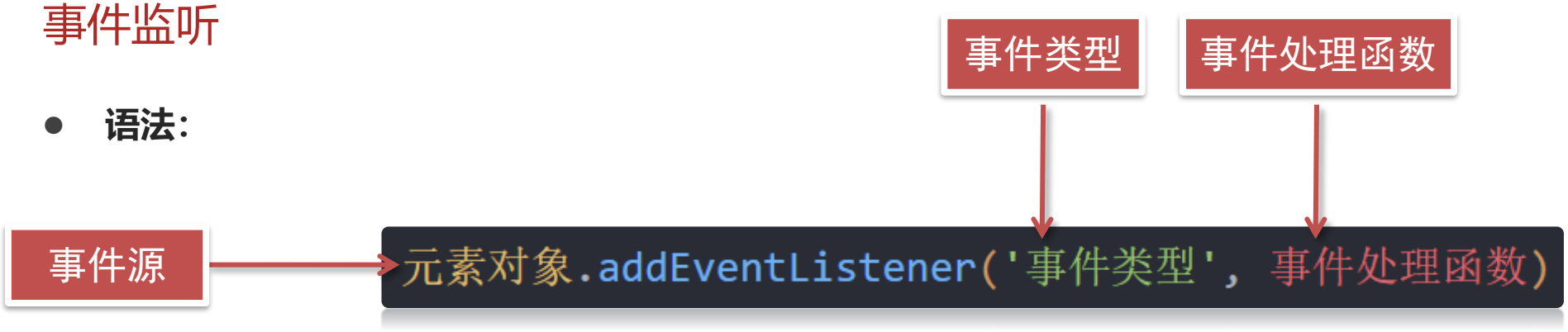
```
元素对象.addEventListener('事件类型', 事件处理函数)
```

- 事件发生后，想要执行的代码写到事件处理函数里面
- 当触发指定的事件时，则事件处理函数就会被执行
- 事件监听是将事件处理函数注册到元素对象身上
- 事件监听也称为: 事件注册、事件绑定



事件监听

- **语法：**

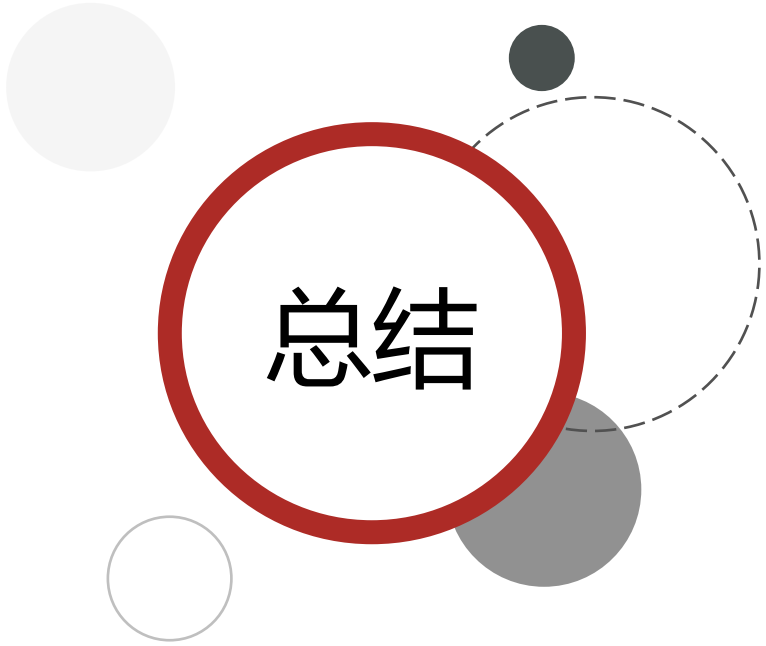


举例说明

```
<button>按钮</button>
<script>
  const btn = document.querySelector('.btn')
  // 修改元素样式
  btn.addEventListener('click', function () {
    alert('点击了~')
  })
</script>
```

注意：

1. 事件类型要加引号，小写
2. 函数是点击之后再去执行，每次点击都会执行一次



1. 我们为什么需要事件?
   - 我们希望一段代码在某个特定的时机才去执行的时候
2. 事件监听的语法是什么?
   - 将事件处理函数注册到元素对象身上，当触发指定的事件时，则事件处理函数就会被执行
   - 也称为事件注册、事件绑定
3. 事件监听三要素是什么?
   - 事件源 (谁被触发了)
   - 事件类型 (什么情况下触发，点击还是鼠标经过等)
   - 事件处理函数 (要做什么事情)

```
元素对象.addEventListener('事件类型', 事件处理函数)
```

04

事件监听

- 事件监听
- 环境对象-this
- 回调函数
- 拓展阅读-事件监听版本

环境对象-this

疑问：为什么事件监听的事件处理函数没有写成箭头函数？

这里的函数不推荐使用箭头函数

环境对象：指的是函数内部特殊的 **this**，它指向一个对象，并且受当前环境影响

作用：弄清楚this的指向，可以让我们代码更简洁

- 函数的调用方式不同，this 指代的对象也不同
- 【谁调用， this 就是谁】是判断 this 指向的粗略规则
- 箭头函数没有自己的this，this指向外层作用域中的this

  所以在事件处理函数，一般不用箭头函数

```
<button>按钮</button>
<script>
  const btn = document.querySelector('.btn')
  // 修改元素样式
  btn.addEventListener('click', function () {
    alert('点击了~')
  })
</script>
```

```
btn.addEventListener('click', () => {
  alert('点击了')
})
```

04

事件监听

- 事件监听
- 环境对象-this
- 回调函数
- 拓展阅读-事件监听版本

回调函数

回调函数：当一个函数当做参数来传递给另外一个函数的时候，这个函数就是回调函数（回头调用的函数）

作用：完成某些特定任务

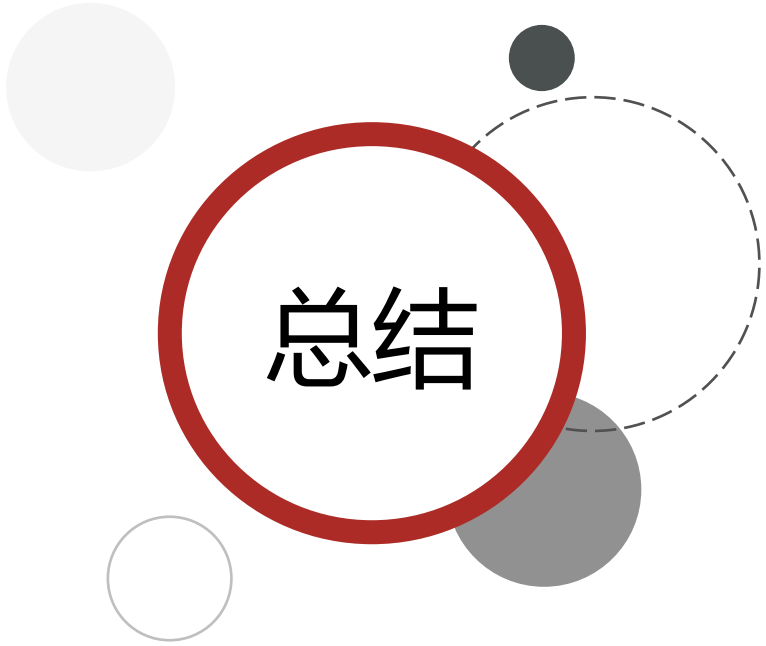
- 常见的使用场景：

```
box.addEventListener('click', function() {})
```

```
box.addEventListener('click', function() {
  // 执行特定任务，比如可以弹出警示框
})
```

```
arr.map(function () { })
```

```
arr.map(function (item, index) {
  // 为数组中的每个元素执行的函数
})
```



1. 回调函数
   - 把函数当做参数传递给另外一个函数，这个函数就叫回调函数
   - 回调函数本质还是函数，只不过把它当成参数使用
   - 使用匿名函数做为回调函数比较常见
   - 作用是完成某些特定任务

```
box.addEventListener('click', function () {
  console.log('我也是回调函数')
})
```

```
arr.map(function (item, index) {
  // 为数组中的每个元素执行的函数
})
```

事件监听

- 事件监听
- 环境对象-this
- 回调函数
- 拓展阅读-事件监听版本

事件监听版本

L：level 标准、层次

DOM L0

事件源.on事件类型 = function() { }

```
btn.onclick = function() {
  alert('我是弹框')
}
```

DOM L2

事件源.addEventListener(事件类型，事件处理函数)

```
btn.addEventListener('click', function() {
  alert('我是弹框')
})
```

- 区别：

  on 方式同名事件会被覆盖，addEventListener则不会，同时拥有事件更多特性，推荐使用

事件监听版本

- 发展史：
  - DOM L0：是 DOM 的发展的第一个版本，使用 on.事件类型 来注册事件
  - DOM L1：DOM级别1 于1998年10月1日成为W3C推荐标准
  - DOM L2：使用addEventListener注册事件
  - DOM L3： DOM3级事件模块在DOM2级事件的基础上重新定义了这些事件，也添加了一些新事件类型

```
btn.onclick = function() {
  alert('我是弹框')
}
```

```
btn.addEventListener('click', function() {
  alert('我是弹框')
})
```

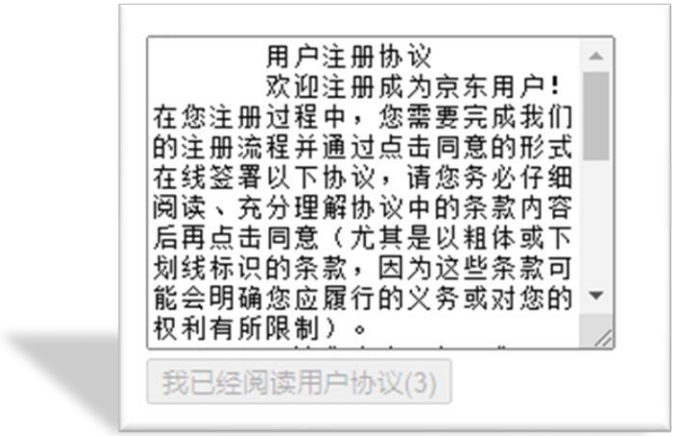
目录

Contents

操作元素属性

- 操作元素**常用**属性
- 操作元素**样式**属性
- 自定义属性

操作元素常用属性



- 还可以通过DOM操作元素属性，比如通过 src 更换 图片地址
- 最常见的属性，比如： href、title、src 、value、checked、disabled等等
- **语法：**

```
对象.属性 = 值
```

- **注意：**表单的disabled、checked、selected属性值一律使用布尔值表示

举例说明

```
// 1. 先获取这个元素
const img = document.querySelector('img')
// 2. 操作DOM元素常见属性
// 2.1 查
console.log(img.src)

// 2.2 改
img.src = './images/3.png'
```

```
// 2.1 禁用按钮
const button = document.querySelector('button')
button.disabled = true  // true 是禁用

// 2.2 勾选复选框
const agree = document.querySelector('[name=agree]')
agree.checked = false // flase 是不选中复选框
```

练习

点击按钮，随机更换图片

需求：当我们点击按钮的时候，页面中的图片随机显示不同的图片

分析：

①：监听按钮的点击事件

②：利用随机数抽取数组中的一个图片地址

③：修改图片元素的src地址（把图片地址赋值给src属性）

```
// 图片地址
const arr = [
  './images/1.png',
  './images/2.png',
  './images/3.png',
  './images/4.png'
]
```

操作元素属性

- 操作元素**常用**属性
- 操作元素**样式**属性
- 自定义属性

操作元素**样式**属性

- 还可以通过 DOM对象修改标签元素的样式属性
  - 比如通过 轮播图小圆点自动更换颜色 样式
  - 点击按钮可以滚动图片，这是移动的的位置 translateX 等等

style **类名** **classList**

style属性操作CSS **类名操作CSS** **classList操作类**

操作元素**样式**属性

1.通过 style 属性操作元素样式

- **语法：**

```
对象.style.样式属性 = 值
```

举例说明

```
const box = document.querySelector('.box')
// 修改元素样式
box.style.width = '200px'
box.style.marginTop = '15px'
box.style.backgroundColor = 'pink'
```

注意：

1. 修改样式通过style属性引出
2. 如果属性有-连接符，需要转换为小驼峰命名法
3. 赋值的时候，需要的时候不要忘记加css单位

总结

1. 操作元素样式属性通过_____style_____属性引出来?
2. 如果需要修改一个div盒子的样式，比如 padding-left, 如何写?
   - element.style.paddingLeft = '300px'
   - 小驼峰命名法
3. 因为我们是样式属性，一定别忘记，大部分数字后面都需要加单位

```
const box = document.querySelector('.box')
// 修改元素样式
box.style.width = '200px'
box.style.marginTop = '15px'
box.style.backgroundColor = 'pink'
```

练习 点击按钮，随机显示背景图片

需求：当我们点击按钮的时候，页面中的背景图片随机显示不同的图片

分析：

①：监听按钮的点击事件

②：利用随机数抽取数组中的一个图片地址

③：修改body元素的背景样式地址

```
// 背景图片地址数组
const arr = [
  './images/bg1.jpg',
  './images/bg2.jpg',
  './images/bg3.jpg',
  './images/bg4.jpg',
  './images/bg5.jpg'
]
```

操作元素**样式**属性

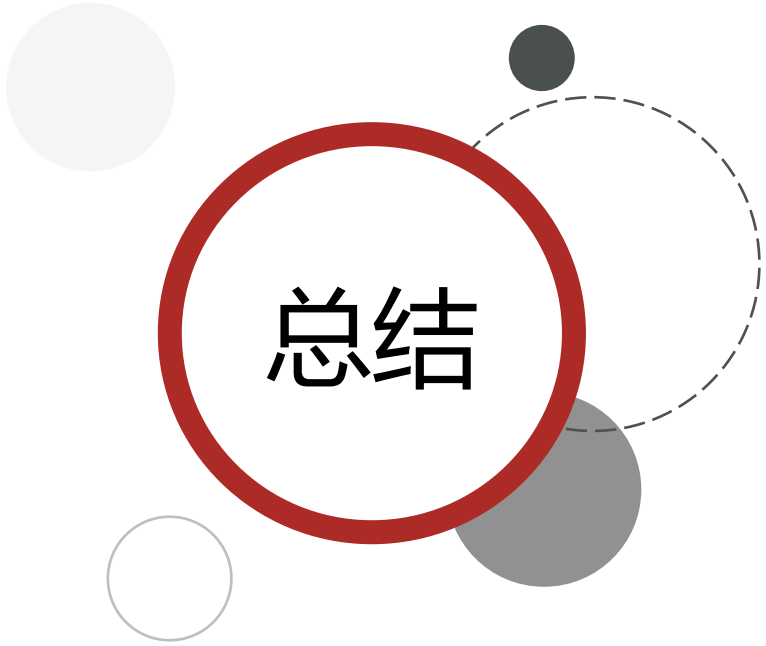
2. 通过类名(className) 操作元素样式（理解）

- 如果修改的样式比较多，直接通过style属性修改比较繁琐，我们可以通过借助于css类名的形式
- 核心：把多个样式放到css一个类中，然后把这个类添加到这个元素身上
- **语法：**

```
// active是一个类名
对象.className = 'active'
```

- **注意：**

1. 由于class是关键字, 所以使用className去代替
2. className是使用新值换旧值, 如果需要添加一个类,需要保留之前的类名



1. 使用 className 相比较 style 有什么好处?
   - 如果修改多个样式，会方便一些
2. 使用 className 有什么注意事项?
   - 直接使用 className 赋值会覆盖以前的类名

操作元素**样式**属性

3. 通过 classList 操作元素样式（**推荐**）

- 为了解决className 容易覆盖以前的类名，我们可以通过classList方式追加和删除类名
- **语法：**

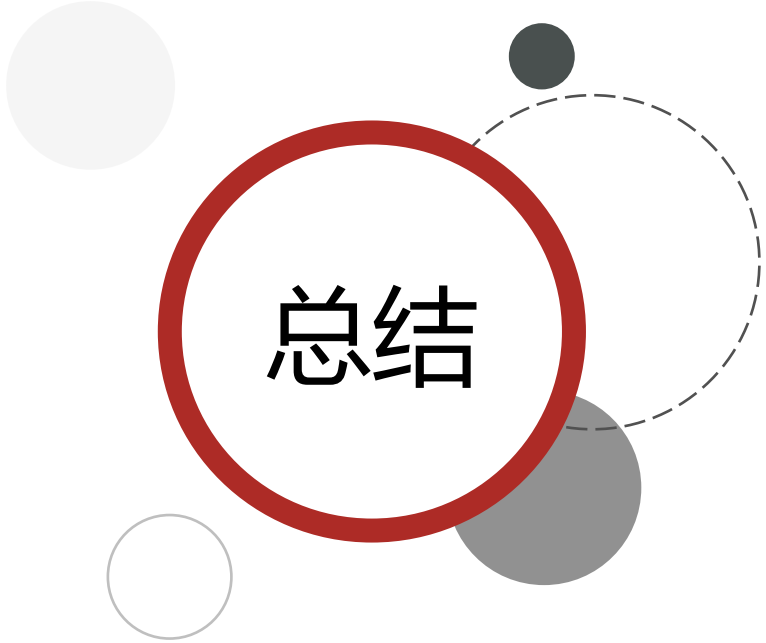
```
// 新增类名
DOM对象.classList.add('类名')

// 移除类名
DOM对象.classList.remove('类名')

// 切换类名
DOM对象.classList.toggle('类名')

// 判断是否有指定类名
DOM对象.classList.contains('类名')
```

- **注意：** 方法要加小括号



1. 使用 style、 className 和classList 怎么选择?
   - 修改样式很少的时候，使用 style
   - 修改大量样式的可以选择类：className / classList
   - classList 是追加和删除不影响以前类名，更提倡

案例 点击关闭王者荣耀登录窗口

需求：点击关闭按钮之后，关掉登录提示盒子

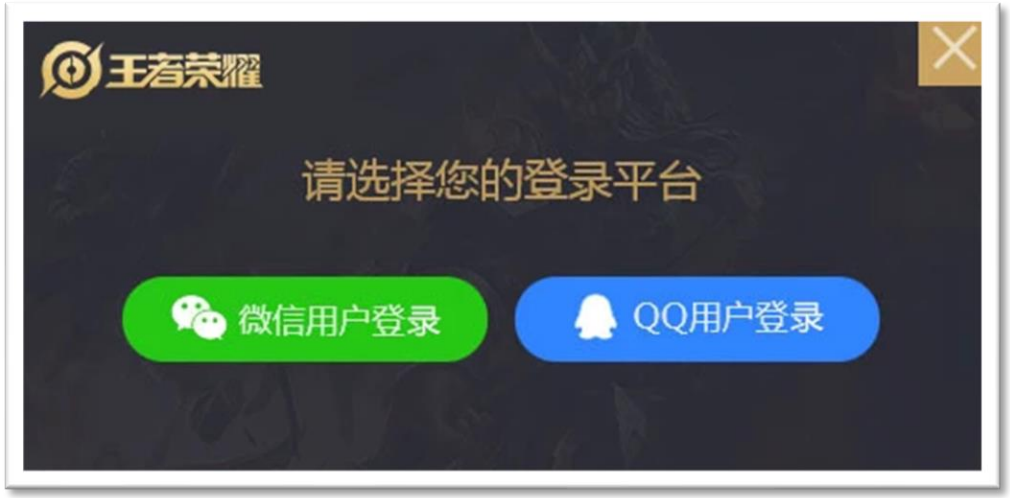
分析：

①：事件源： 关闭按钮 a链接

②：事件类型：鼠标点击 click

③：事件处理程序：关闭的是父盒子

核心：利用提供的 hide 类名 来实现隐藏功能

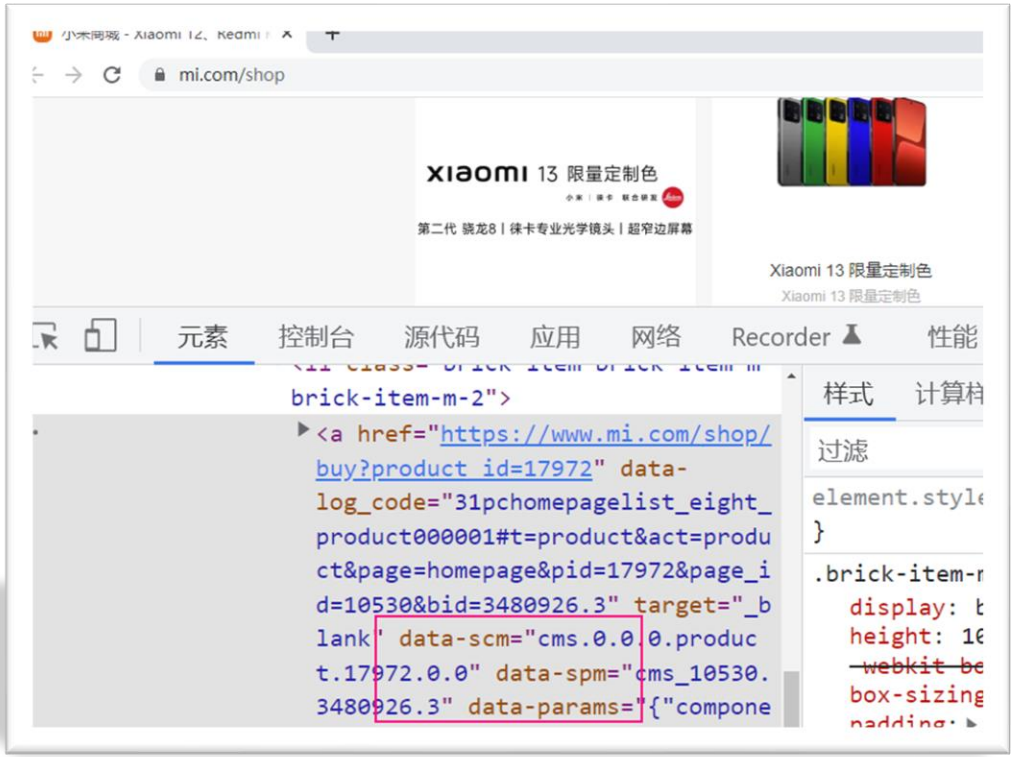


操作元素属性

- 操作元素**常用**属性
- 操作元素**样式**属性
- 自定义属性

自定义属性

- **标准属性:** 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比如：对象.title
- **自定义属性:**
  - 在html5中推出来了专门的data-自定义属性



自定义属性

- **标准属性:** 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比如：对象.title
- **自定义属性:**
  - 在html5中推出来了专门的data-自定义属性
  - 使用场景：通过自定义属性可以存储数据，后期可以使用这个数据

就业榜

学号	姓名	年龄	性别	薪资	就业城市	操作
1	迪丽热巴	18	女	10000	北京	删除
2	古丽扎娜	19	女	11000	北京	删除
3	佟丽丫丫	20	女	12000	北京	删除
4	马尔扎哈	21	女	13000	北京	删除

```
▼<td>
    <a href="javascript:" data-id="0">删除</a>
  </td>
</tr>
<tr>
▼<td>
    <a href="javascript:" data-id="1">删除</a>
  </td>
</tr>
<tr>
▼<td>
    <a href="javascript:" data-id="2">删除</a>
  </td>
</tr>
<tr>
▼<td>
    <a href="javascript:" data-id="3">删除</a>
  </td>
```

自定义属性

- **标准属性:** 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比如：对象.title
- **自定义属性:**
  - 在html5中推出来了专门的data-自定义属性
  - 使用场景：通过自定义属性可以存储数据，后期可以使用这个数据
  - 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

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

目录

Contents

案例

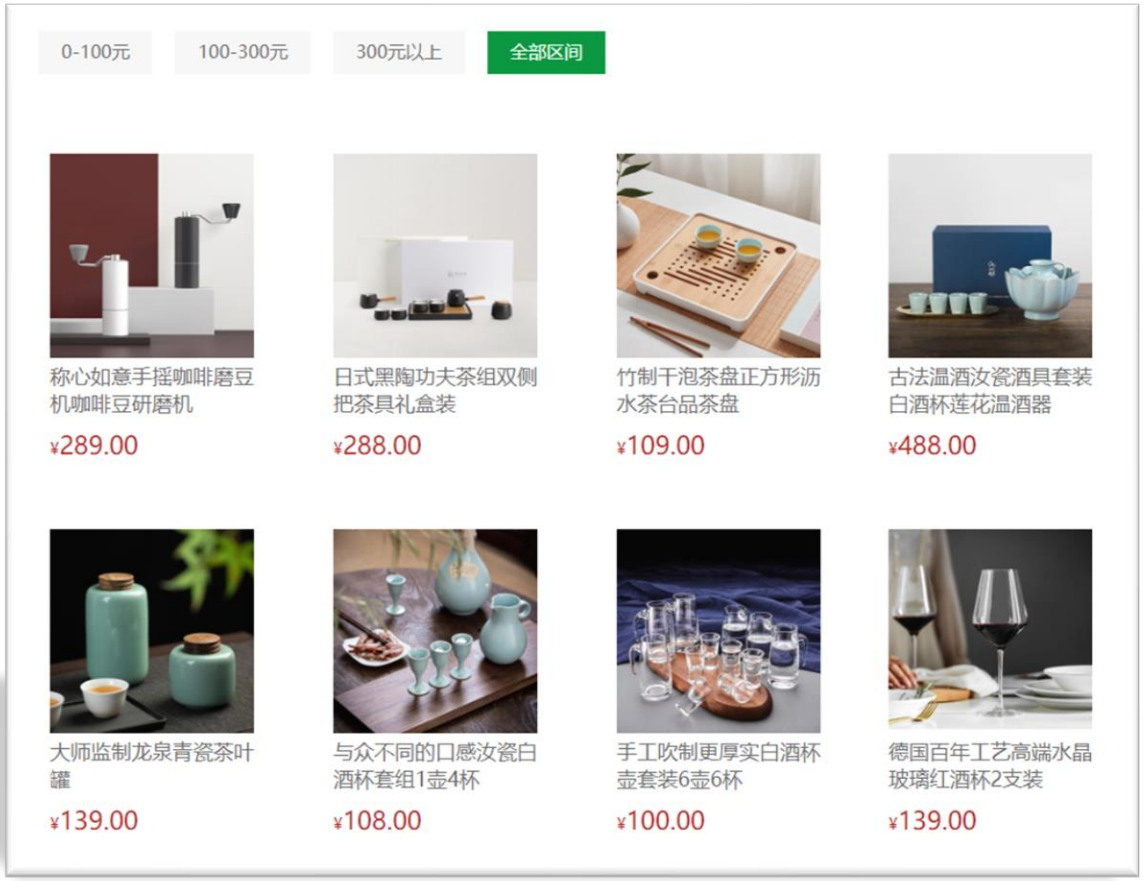
商品列表价格筛选

需求：

①：渲染数据列表

②：根据选择不同条件显示不同商品

案例 商品列表价格筛选



渲染业务

筛选业务（filter）

步骤 商品列表价格筛选

渲染业务

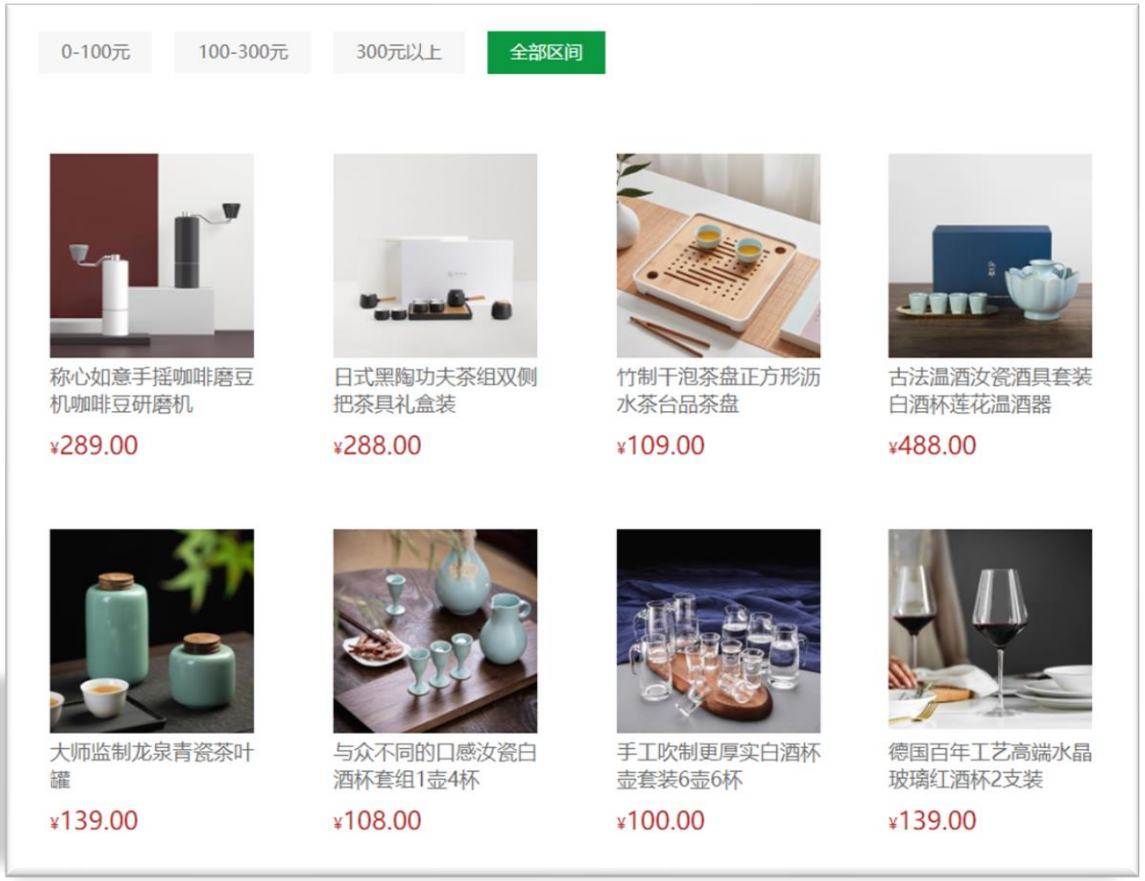
业务分析：页面初始渲染

①：封装 render 渲染函数

②：利用 数组 map + join 方法渲染页面

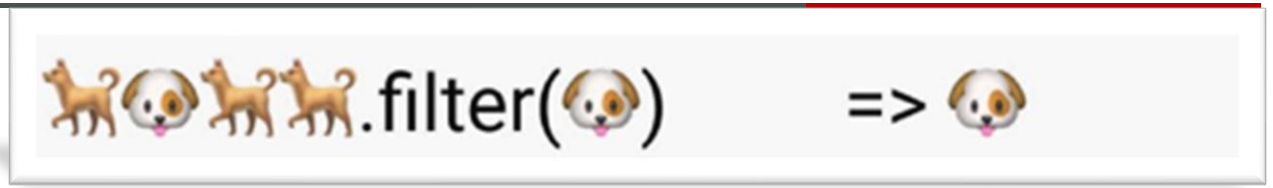
③：后期筛选业务，本次要求渲染函数增加参数，传递不同的数组

案例 商品列表价格筛选



- 渲染业务
- 筛选业务（filter）

回顾-数组 filter 方法



- filter() 方法创建一个新的数组，新数组中的元素是符合条件的所有元素
- 主要使用场景： 筛选数组符合条件的元素，并返回筛选之后元素的新数组，不影响原数组
- **语法：**

```
const newArr = arr.filter(function (element, index) {
  return  筛选条件
})
```

- element 是数组元素
- index 是数组元素的索引号

步骤 商品列表价格筛选

筛选业务

业务2：点击链接，筛选不同商品

(1) 给所有的导航a链接注册click事件

- 先拿到当前点击的序号（利用自定义属性 data-index ）

(2) 筛选的核心思路：

- 根据不同序号 利用filter 筛选数组，得到符合条件的新数组
- 新数组传递给渲染函数重新渲染页面即可
- 全部区间不需要筛选，直接把goodList渲染即可

传智教育旗下高端IT教育品牌