Web 开发技术-JavaScript

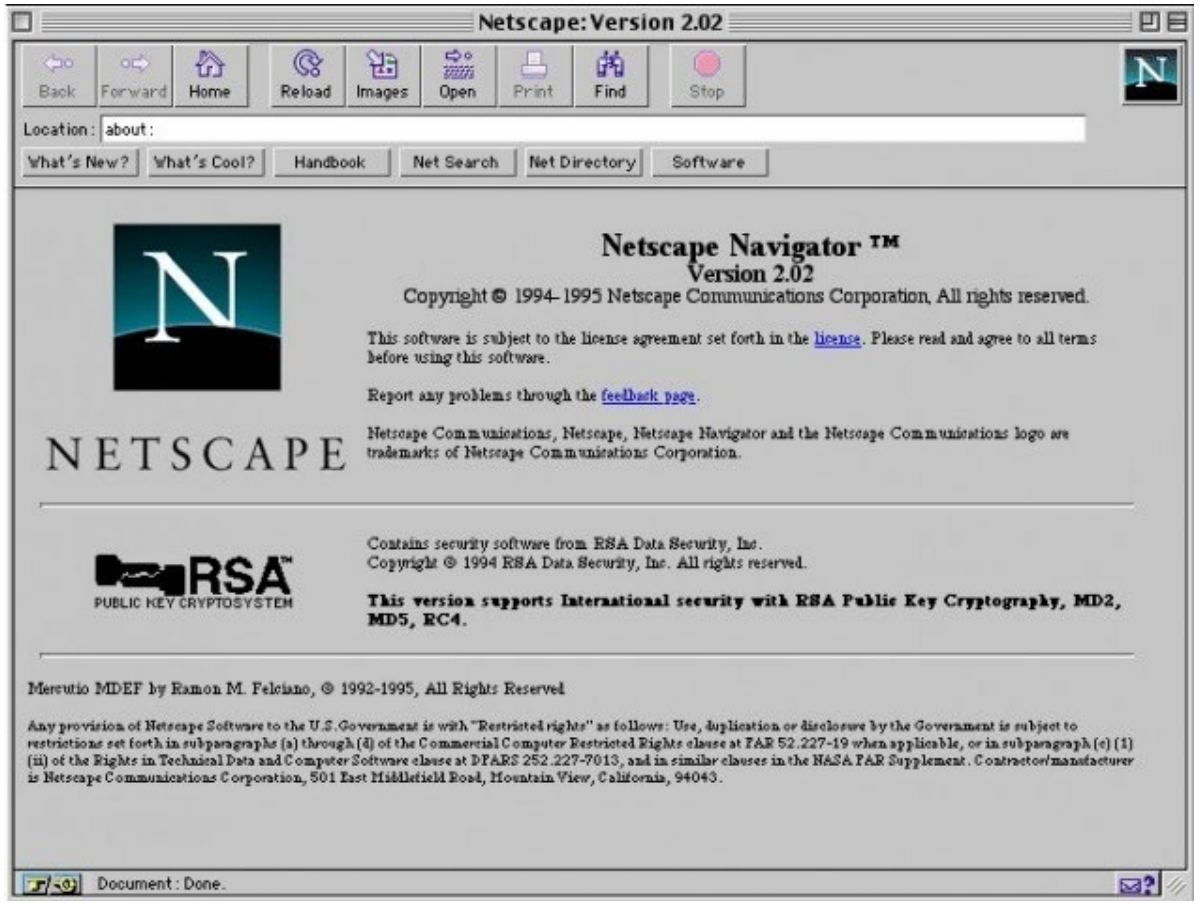
1.概述

内容提要

- 1 JavaScript 简史
- 2 JavaScript 功能、脚本语言
- 3 JavaScript 实现
  - ECMAScript
  - DOM
  - BOM
- 4 在 HTML 中使用 JavaScript
  - 直接嵌入
  - 外部引入
  - JS 代码压缩

1 JavaScript 简史

- 时代背景
  - 1995 年，拨号上网，速度仅 28.8kbit/s
- 诞生目的
  - 用于表单验证，网速很慢，最好能将服务器语言负责的表单输入验证操作转为客户端独立完成，减少客户端 - 服务器数据交换量，节约大量时间。



Netscape Navigator 2, 1995

人类首个支持 JavaScript 和 gif 动图的浏览器

1. JavaScript 简史

- 发展过程
  - ***1995***：LiveScript/JavaScript 始于网景
    - 网景通信公司（Netscape）开发 LiveScript 1.0，与 Sun 公司组成开发联盟。由于 Java 正流行，LiveScript 发布前临时改名 JavaScript.
  - ***1996***：Microsoft 采用
    - 在 Internet Explorer 3 中加入名为 JScript 的 JavaScript 实现.
  - ***1997***：JavaScript 标准化 – ECMAScript
    - 欧洲计算机制造商协会（ECMA） 39 号技术委员会（TC39）负责“标准化一种通用、跨平台、供应商中立的脚本语言的语法和语义”。
    - 定义一种名为 ECMAScript 的新脚本语言标准。
  - ***1998***：ISO/IEC 采用 ECMAScript 作为标准
    - 浏览器开发商开始致力于将 ECMAScript 作为各自 JavaScript 实现基础。

2. JavaScript 和脚本语言

- **JavaScript**
  - 一种高级的，面向对象的脚本语言。
  - JavaScript 代码可直接嵌入 HTML 文件或使用外部文件，随网页一起传送到客户端浏览器，然后通过浏览器的解释器来解释执行。
  - JavaScript 不支持 I/O（如网络、存储和图形），但可以由宿主环境（如浏览器）提供支持。

- **脚本语言（Scripting language）**
  - 是为了缩短传统的“编写、编译、链接、运行”（edit-compile-link-run）过程而创建的计算机编程语言。
  - 可直接用任何的文本编辑器开发完成。是一种不必事先编译，只要利用适当的解释器(Interpreter)就可以执行的简单的解释式程序

2. JavaScript 功能

- 脚本编写语言
- 基于对象的语言
- 简单性

主要功能

- 安全性
- 动态性
- 跨平台

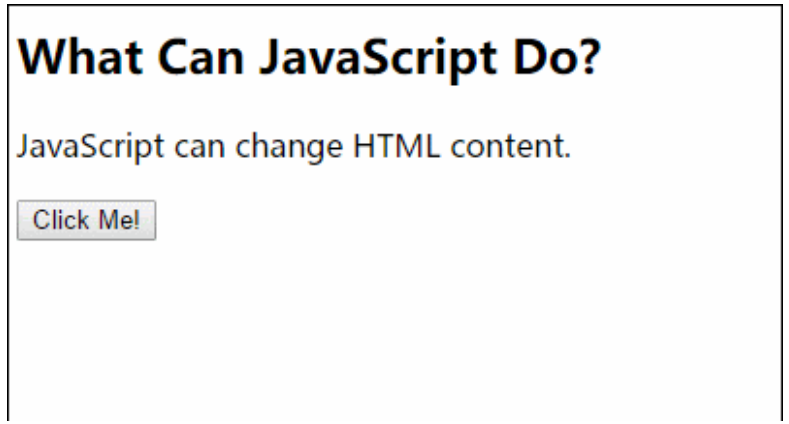
  - 向 HTML 页面添加交互行为
  - 读写 HTML 元素
  - 在数据被提交到服务器之前验证数据
  - 检测访客的浏览器信息
  - 控制 cookies 和本地存储

- 主要功能
  - 向 HTML 页面添加交互行为
  - 读写 HTML 元素
  - 在数据被提交到服务器之前验证数据
  - 检测访客的浏览器信息
  - 控制 cookies 和本地存储

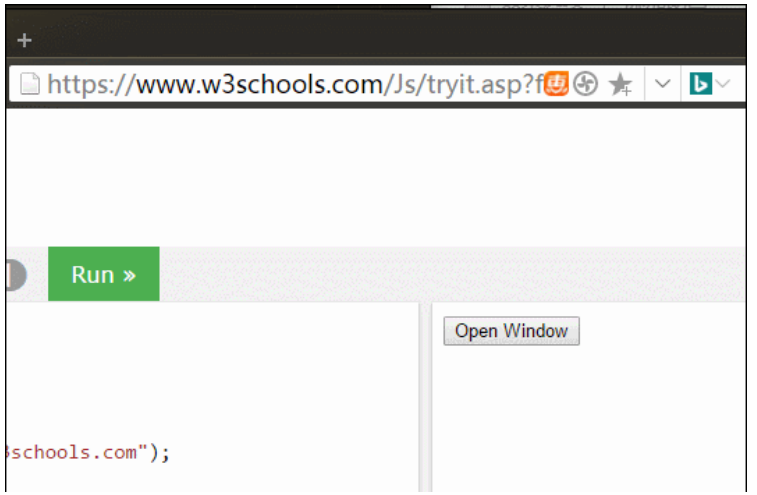
2. JavaScript 功能

- 其他功能，如
  - 网络服务器（Node.js）
    - 处理 Web 应用的 HTTP 请求
  - 机器学习（TensorFlow.js）
    - 在浏览器/Node.js 上开发、训练和部署 ML 模型
  - 开发桌面应用（ELECTRON）
    - 使用 JS/HTML/CSS 构建跨平台的桌面应用

2. JavaScript *Demo*



向 HTML 页面添加交互行为

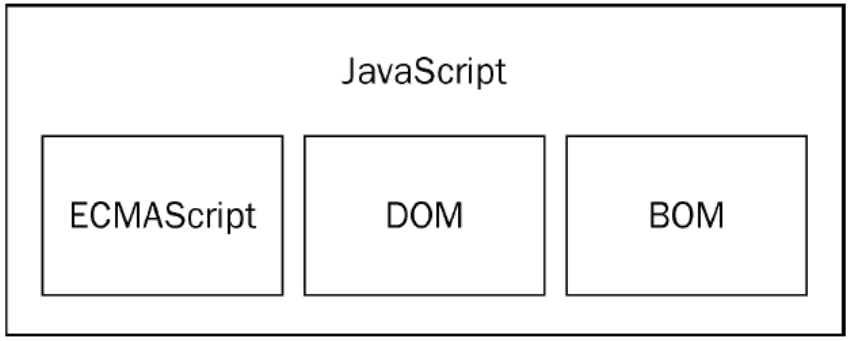


检测浏览器信息

本地表单验证

3. JavaScript 实现

JavaScript 的完整实现 =



ECMAScript
提供语言核心功能

文档对象模型（DOM）
提供访问和操作网页的方法和接口

浏览器对象模型（BOM）
提供与浏览器交互的方法和接口

3.1 ECMAScript 简介

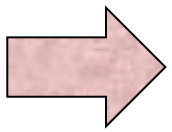
- ECMAScript 是一种**标准**，定义了 JavaScript 语言的语法、类型、语句、关键字、保留字、操作符和对象等。
- JavaScript 是 ECMAScript 的一种**实现**。
- 与 Web 浏览器没有依赖关系，Web 浏览器只是 ECMAScript 实现可能的**宿主环境**之一。
- 宿主环境提供基本的 ECMAScript 实现和扩展（如 DOM、BOM 等）。

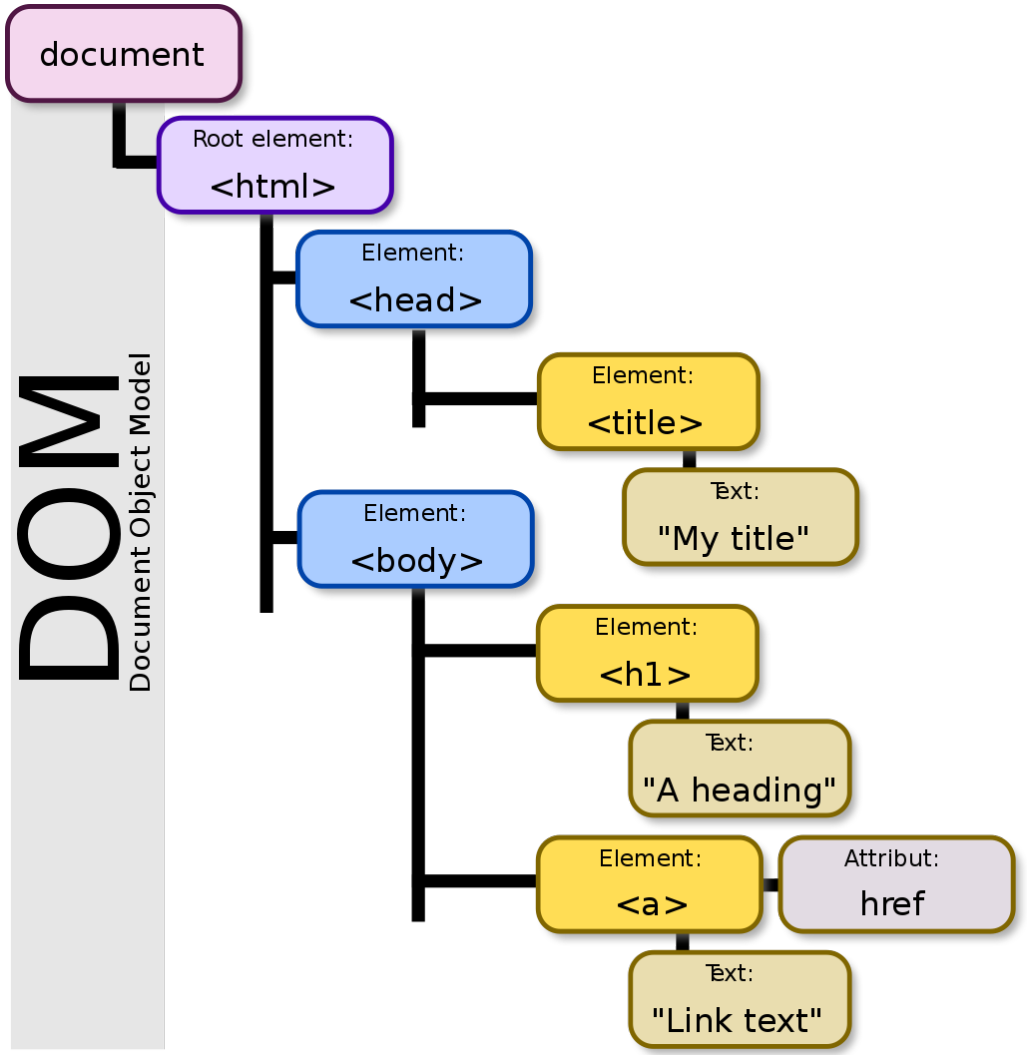
3.2 DOM（文档对象模型）简介

- DOM（Document Object Model）是一种跨平台、独立于编程语言的应用程序编程接口（API）。DOM 标准由 W3C 设定。
- DOM 将 HTML/XHTML/XML 文档视为树结构，每个节点代表文档中的一个组成部分，这些节点又包含不同数据类型的结构。
- 通过 DOM 创建的树形图，开发人员获得了控制页面内容和结构的主动权。开发者可以增加、删除、修改或替换任何节点。

3.2 DOM（文档对象模型）简介

```
<html>
  <head>
    <title>My Title</title>
  </head>
  <body>
    <h1>A heading</h1>
    <a href="shaorongwang.com">
      Link text
    </a>
  </body>
</html>
```





3.3 BOM（浏览器对象模型）简介

- BOM（Browser Object Model）提供了很多对象，用于访问浏览器的功能，这些功能与任何网页内容无关。

- BOM 的常用用途
  - 弹出新浏览器窗口
  - 移动、缩放和关闭浏览器窗口
  - 获取浏览器详情信息，如浏览器版本、语言等
  - 获取浏览器所加载页面的详细信息，如 URL 等
  - 获取用户显示分辨率的详细信息，如显示屏幕的高度、宽度等
  - 对 cookie 的支持

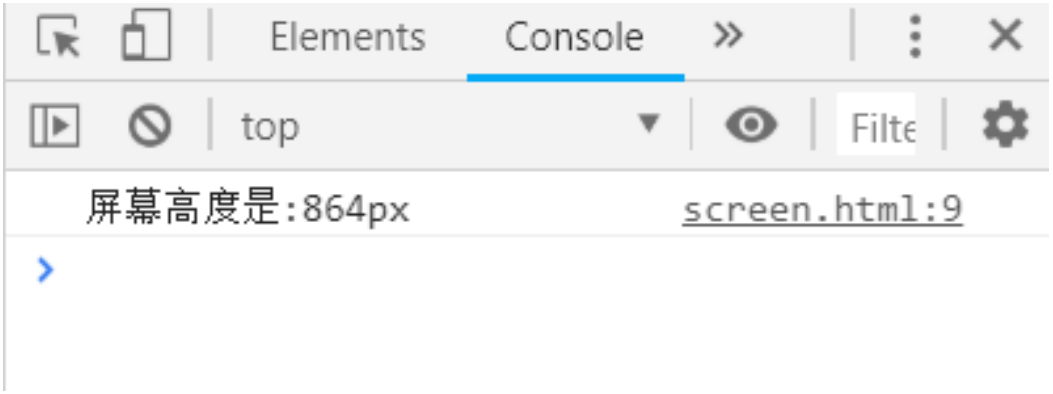
- HTML5 将主要的 BOM 方法纳入规范。

3.3 BOM（浏览器对象模型）简介

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Screen Object</title>
</head>
<body>
  <script>
    console.log("屏幕高度是:" +
screen.height + "px");
  </script>
</body>
</html>
```

Screen 对象实例

属性



4 在 HTML 中使用 JavaScript

- 两种方式
  - 直接嵌入：将 JS 代码写入 HTML 元素的属性中，或将代码写入 <script> 标签内

Demo 1.1

```
<html>
  <body>
    <p onmouseover="console.log('Hello World!')">鼠标移到这里</p>
  </body>
</html>
```

Demo 1.2

```
<script>
  console.log("屏幕高度是:" + screen.height + "px");
</script>
```

  - (推荐) 外部脚本：在 HTML 中使用 <script> 的 src 属性从外部 JS 文件引入

```
<script src="main.js"></script>
```

4.1 <script> 标签

- HTML5 中，<script> 标签保留 4 个有效属性
  - **async**：可选。定义脚本是否异步执行，表示应立即下载脚本，但不应妨碍页面中的其他操作，比如下载其他资源或等待加在其他脚本。只对外部脚本有效。
  - **type**：可选。指示脚本的 MIME（描述消息内容类型的因特网标准） 类型，默认为 text/javascript
  - **defer**：可选。表示脚本可以延迟到文档完全被解析和显示之后再执行。仅对外部脚本有效。
  - **src**：可选。定义指向包含脚本文件的 URL。

4.2 嵌入脚本

- 执行顺序
  - 包含在 <script> 元素内部的 JavaScript 代码从上至下依次解释。如遇语法错误，后续代码无法继续执行。

- 转义字符“\”
  - 不能在 <script> 元素内部的 JS 代码中出现“</script>”字符串，否则会认为出现闭标签。应用转义字符解决。

```
<script>
  console.log(“<\/script>”);
</script>
```

4.3 引入外部文件

- 将代码独立存储为以 .js 为扩展名的文件，利用 <script> 元素的 src 属性将该文件调入。

```
<script src="welcome.js"></script>

// welcome.js 文件内容如下:
document.write("欢迎您学习JavaScript!");
```

- 优点
  - **可维护性**：可在不触及 HTML 标记的情况下，集中精力编辑 JS 代码
  - **可缓存**：浏览器可缓存链接的所有外部 JS 文件。若多个页面使用同一个文件，该文件只需下载一次。

4.3 引入外部文件

标签位置

- 浏览器遇到 <body> 标签才开始呈现内容。
- <script> 标签应放在 <body> 元素中页面内容的后面，这样在下载并解析 JavaScript 代码前，页面内容将完全呈现在浏览器中，不会等待下载完 JS 后再渲染页面，减少显示空白页的时间。

```
<html>
  <body>
    <p>Hello World</p>
    <!-- 其他内容 -->
    <script src="main.js"></script>
  </body>
</html>
```

4.3 引入外部文件

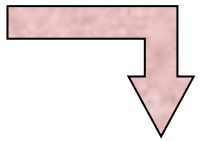
JavaScript 代码压缩

- 优势：去除 JavaScript 文件中的注释和不必要的空格，并简化命名标识符。它通常减少了一半的文件大小，从而导致更快的下载速度，并增加了一定的安全性。
- 扩展名：*.min.js
- 压缩工具：YUI Compressor、在线压缩（https://www.css-js.com/）等

```
(function ($, sr) {
    // debouncing function from John Hann
    // http://unscriptable.com/index.php/2009/03/20/debouncing-javascript-methods/
    var debounce = function (func, threshold, execAsap) {
        var timeout;

        return function debounced() {
            var obj = this,
                args = arguments;
```

custom.js



custom.min.js

```
(function(c,b){var a=function(g,d,e){var h;return function f(){var k=this,j=arguments;
```