HTML 面试题

前端页面的三层结构

前端页面的三层结构主要分为：结构层（HTML）、表现层（CSS）、行为层（JavaScript）

1. **结构层（HTML）**

结构层是网页的骨架，使用 HTML 标记语言来描述页面的基本结构和内容

主要职责：

- 定义页面的基本结构
- 组织页面的内容层次
- 提供语义化的标签
- 确保内容的可访问性
- 为 SEO 提供基础

2. **表现层（CSS）**

表现层负责页面的视觉效果，使用 CSS 来定义页面的样式和布局

主要职责：

- 控制页面布局
- 定义视觉样式
- 实现响应式设计
- 处理动画效果
- 提供主题定制

3. **行为层（JavaScript）**

行为层负责页面的交互和动态功能，使用 JavaScript 来实现

主要职责：

- 处理用户交互
- 实现动态功能
- 进行数据处理
- 控制页面行为

H5 和 HTML5 的区别

HTML5 是 Web 技术的标准规范，包含了新的语义化标签、API、多媒体支持等技术标准

而 H5 是一个产品名词，特指移动端的网页应用，通常用于开发富交互的营销页面、小游戏等，它是一个综合技术的应用形态，包含了 HTML5、CSS3、JavaScript 等多种技术

说说 Web 标准和 W3C 标准

Web 标准是由 W3C 组织制定的一系列标准，目的是让网页更规范、更易维护、更利于搜索引擎抓取。它主要包含三个方面：

1. 结构标准（HTML）：规范页面的结构和内容
2. 表现标准（CSS）：规范页面的显示效果

3. 行为标准（JavaScript）：规范页面的交互行为

W3C(万维网联盟)是一个国际组织，它的主要工作是制定 Web 标准并推广。通过发布技术标准和指导原则，它帮助确保网页能在不同的浏览器、设备上正常工作，提高了网页的兼容性和可访问性

遵循 Web 标准可以提高代码质量、增强可维护性、提升用户体验，同时有利于搜索引擎优化（SEO）

对渐进增强与优雅降级的理解

渐进增强：

- 从基本功能开始构建
- 针对低版本浏览器构建核心功能
- 然后逐步添加高级功能增强体验
- 类似"由下而上"的开发方式

优雅降级：

- 从完整功能开始构建
- 针对高版本浏览器构建完整功能
- 然后做向下兼容处理
- 类似"由上而下"的开发方式

渐进增强的例子：

```
// 1. 基础功能：普通按钮点击
const button = document.querySelector(".action-button");
button.addEventListener("click", handleClick);

// 2. 增强功能：添加触摸支持
if ("ontouchstart" in window) {
  button.addEventListener("touchstart", handleTouch);
}

// 3. 更高级的增强：添加手势支持
if (window.GestureEvent) {
  button.addEventListener("gesturestart", handleGesture);
}
```

优雅降级的例子：

```
function createVideoPlayer() {
  // 1. 首选最新的视频播放方案
  if (supportsHLSPlayback()) {
    return new HLSVideoPlayer();
  }
  // 2. 降级到普通 HTML5 播放器
  else if (supportsHTML5Video()) {
    return new HTML5VideoPlayer();
  }
  // 3. 最后降级到 Flash 播放器
  else {
    return new FlashVideoPlayer();
  }
```

```
}
```

下面通过一个响应式图片加载的例子来展示这两种策略的实际应用：

```html
<!-- 渐进增强的响应式图片 -->
<picture>
  <!-- 基础图片 -->
  <img src="basic.jpg" alt="基础图片" />

  <!-- 逐步增强 -->
  <source media="(min-width: 768px)" srcset="medium.jpg" />
  <source media="(min-width: 1200px)" srcset="large.jpg" />
</picture>

<!-- 优雅降级的响应式图片 -->
<img
  src="large.jpg"
  onerror="this.onerror=null; this.src='basic.jpg';"
  alt="降级图片"
/>
```

CSS 中的应用示例：

```css
/* 渐进增强的 CSS */
.button {
  /* 基础样式 */
  background: #007bff;
  color: white;
  padding: 10px;

  /* 增强特性 */
  @supports (backdrop-filter: blur(10px)) {
    backdrop-filter: blur(10px);
    background: rgba(0, 123, 255, 0.8);
  }
}

/* 优雅降级的 CSS */
.modal {
  /* 现代浏览器特性 */
  display: grid;
  place-items: center;

  /* 降级方案 */
  display: flex;
  justify-content: center;
  align-items: center;
}
```

主要区别：

1. 开发思路

- 渐进增强：先实现基础功能，再逐步添加高级特性
- 优雅降级：先实现完整功能，再处理兼容性问题

2. 适用场景

- 渐进增强：面向移动端、追求性能和基础可用性
- 优雅降级：面向桌面端、追求完整功能体验

3. 维护成本

- 渐进增强：代码结构清晰，易于维护
- 优雅降级：需要维护多套降级方案，成本较高

4. 用户体验

- 渐进增强：基础功能保证，体验逐步提升
- 优雅降级：高端用户体验好，低端用户体验可能受限

HTML 标签语义化

在 HTML 标签中，有些标签本身传达了网页内容的语义信息,例如 `<header>` 代表页头， `<nav>` 代表导航等

因此在用 HTML 写网页结构的时候，应该用合适的 HTML 标签来表达内容的结构和含义，让代码的结构能够表达网页实际内容的结构,使其具有良好的自解释性，这就是 HTML 标签语义化

语义化的好处：

1. 提升代码可读性和可维护性，开发者更容易理解内容的层次与逻辑
2. 有利于 SEO 优化，搜索引擎更容易理解页面结构和内容的重要性
3. 在 CSS 失效时也能展现清晰的页面结构

语义化的实践：

- 使用 `<header>、<footer>、<main>、<article>` 等表示文档结构
- 使用 `<nav>` 表示导航区域
- 使用 `<section>` 对内容进行分区
- 使用 `<h1>-<h6>` 表示标题层级
- 使用 `<p>` 表示段落， `<ul>/<li>` 表示列表
- 使用 `<figure>/<figcaption>` 表示图片及其说明

DOCTYPE 的作用

DOCTYPE 必须放在 HTML 文档的最前面，在 HTML 标签之前，它会告诉浏览器使用哪种 HTML 或 XHTML 规范来解析页面（也就是 DTD）

`! DOCTYPE` 告诉浏览器以 HTML5 标准解析页面，如果不写，则进入怪异模式

标准模式

- 遵循 W3C 标准规范进行页面渲染
- 所有浏览器都按照统一的标准解析 HTML/CSS
- 这确保了跨浏览器的一致性和可预测性

怪异模式

- 页面以非标准的方式渲染
- 每个浏览器都可能有自己独特的解析规则
- 为了保持对老版浏览器的兼容性

- 这可能导致页面在不同浏览器中表现不一致

两者的常见区别：

- 在怪异模式下，设置元素的 width 与 height 会包含 padding 与 border
- 在怪异模式下，设置行内元素的 width 与 height 会生效
- 在怪异模式下，设置元素的样式 `margin:0 auto` 的居中效果会失效

> **Caution**
>
> `<!DOCTYPE html>` 不是 HTML 标签！！！
> HTML5 文档不存在严格模式与混杂模式！！！

说说 HTML5 新特性

1. **语义化标签**

- `<header>` - 定义文档或节的页眉
- `<nav>` - 定义导航链接集合
- `<main>` - 定义文档的主要内容
- `<article>` - 定义独立的内容
- `<section>` - 定义文档中的节
- `<aside>` - 定义页面内容之外的内容
- `<footer>` - 定义文档或节的页脚
- `<figure>` - 定义独立的流内容（图像、图表、照片、代码等）
- `<figcaption>` - 定义 figure 元素的标题
- `<mark>` - 定义重要或高亮显示的文本
- `<time>` - 定义日期/时间
- `<details>` - 定义用户可查看或隐藏的额外细节
- `<summary>` - 定义 details 元素的标题

2. **表单增强**

- 新增输入类型：
  - `type="email"` - 电子邮件输入框
  - `type="url"` - URL 输入框
  - `type="number"` - 数字输入框
  - `type="range"` - 范围滑块
  - `type="tel"` - 电话号码输入框
  - `type="search"` - 搜索框
  - `type="color"` - 颜色选择器
  - `type="date"` - 日期选择器
  - `type="month"` - 月份选择器
  - `type="week"` - 周选择器
  - `type="time"` - 时间选择器
  - `type="datetime-local"` - 本地日期时间选择器
- 新增表单属性：

- `placeholder` - 输入框提示
- `required` - 必填项
- `pattern` - 正则验证
- `min` 和 `max` - 最小值和最大值
- `step` - 数字间隔
- `autocomplete` - 自动完成
- `multiple` - 多文件上传
- `novalidate` - 关闭验证
- `autofocus` - 自动聚焦
- `form` - 表单外的表单元素

3. **多媒体支持**

- `<audio>` - 音频播放
  - 属性：src, controls, autoplay, loop, muted, preload
  - 支持多种音频格式：MP3, WAV, Ogg
- `<video>` - 视频播放
  - 属性：src, controls, width, height, poster, autoplay, loop, muted, preload
  - 支持多种视频格式：MP4, WebM, Ogg
- `<source>` - 为媒体元素定义多个源
- `<track>` - 为媒体元素添加字幕

4. **图形和效果**

- `<canvas>` - 可用于绘制图形、制作图片图像合成或者制作简单动画
- `<svg>` - 用于定义基于矢量的图形

5. **存储**

- localStorage - 永久本地存储
- sessionStorage - 临时会话存储
- IndexedDB - 索引数据库
- Web SQL Database (已废弃)

6. **API**

- Geolocation API - 地理位置
- Drag and Drop API - 拖放
- Web Workers - 后台 JavaScript
- Web Socket - 服务器推送
- Server-Sent Events - 服务器发送事件
- WebRTC - 实时通信
- History API - 历史记录管理
- File API - 文件操作
- Blob API - 二进制数据操作
- Fetch API - 网络请求
- requestAnimationFrame - 动画帧请求
- WebGL - 3D 图形

7. **其他新特性**

- `data-*` 自定义数据属性
- classList API - 操作类
- contenteditable - 可编辑内容
- hidden - 隐藏元素
- spellcheck - 拼写检查
- 新的选择器 querySelector/querySelectorAll

8. **浏览器支持**

- 支持离线 Web 应用
- Application Cache (已废弃，推荐使用 Service Workers)
- DNS 预解析 (`<link rel="dns-prefetch">`)
- 预加载资源 (`<link rel="preload">`)

9. **改进的支持**

- 更好的错误处理
- 全局属性的增加
- 更简单的 DOCTYPE 声明
- 更简单的字符编码声明
- 脚本异步加载 (async, defer)
- 更好的跨域资源共享（CORS）支持

> **Tip**
>
> 使用第三方库 HTML5 Shiv 可以解决 HTML5 新标签兼容问题：

```
<head>
<script src="https://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
</head>
```

说说 defer 和 async

defer 和 async 是 script 标签的两个加载脚本的属性，它们有不同的执行时机和用途

可以把 `<script>` 脚本标签分成以下三类：

1. **正常的 script（无属性）**

HTML 解析遇到 script 时会暂停解析，立即下载并执行脚本，执行完毕后继续解析 HTML。这一过程会阻塞页面渲染

样例：

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

2. **async 属性**

HTML 解析过程中异步下载脚本，下载完成后立即执行，执行时会阻塞 HTML 解析，且不保证多个脚本的执行顺序

样例：

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

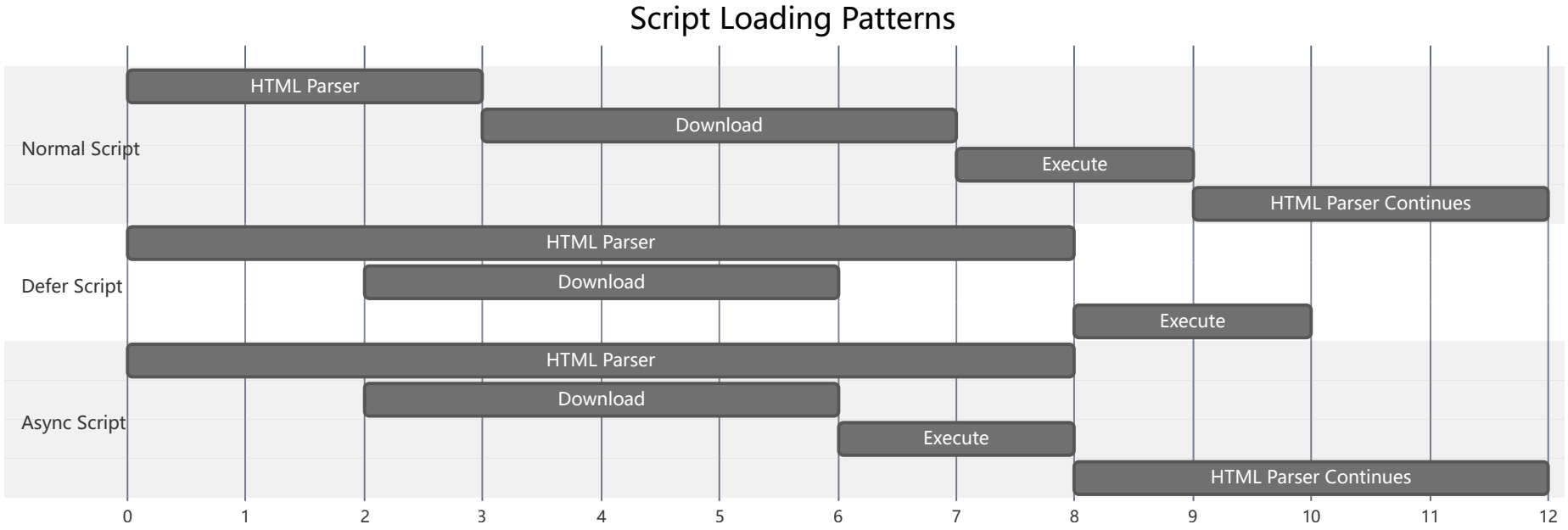
3. **defer 属性**

HTML 解析过程中异步下载脚本，等待 HTML 解析完成后，DOMContentLoaded 事件前执行，且多个 defer 脚本按照顺序执行，适用于需要操作 DOM 的脚本

样例：

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

总结一下就是：



说说 iframe 的优缺点

iframe 是一个 HTML 标签，可以在一个网页中嵌入另一个网页

用途：

- 在当前页面中嵌入其他页面内容
- 实现跨域通信
- 沙箱隔离不同的页面内容
- 加载第三方内容（如广告、地图等）

优点：

1. 嵌入的页面与主页面之间形成了天然的沙箱环境，CSS 和 JavaScript 的作用域可以完全隔离，不会相互影响，可以实现独立的会话管理
2. 可以在多个页面中嵌入相同的内容，便于维护共用的页面模块，实现更新一处，多处同步改变
3. 可以跨子域通信，比如看下面的例子

发送方（父页面）：

```
// 向iframe发送消息
document
  .getElementById("myIframe")
  .contentWindow.postMessage("消息内容", "https://子域.example.com");
```

接收方（子域 iframe）：

```
window.addEventListener(
  "message",
  function (event) {
    // 验证消息来源
    if (event.origin !== "https://主域.example.com") return;

    // 处理接收到的消息
    console.log("接收到消息:", event.data);
  },
  false
);
```

缺点：

1. iframe 会增加页面的加载时间，因为它需要加载额外的 HTML、CSS 和 JavaScript，且会阻塞主页面 `onload` 事件
2. 搜索引擎爬虫难以解析 iframe 内容，不利于网页的搜索引擎优化
3. 增加额外的 HTTP 请求开销，其独立的渲染上下文会增加浏览器渲染负担，内存消耗显著高于普通元素

src 和 href 的区别

从语义上看：

- src：源头，意味着直接加载和嵌入资源
- href：引用，意味着建立资源之间的关联和链接

`src` 加载资源时，浏览器会暂停解析，它的优先级较高，会阻塞页面渲染，用于多媒体资源嵌入

```
src 加载过程：
发起请求 → 下载资源 → 解析 → 替换当前元素
```

`href` 加载资源时，是创建资源间的关联，属于异步加载，不阻塞页面渲染，用于链接和资源引用

```
href 加载过程：
发起请求 → 下载资源 → 建立链接 → 不中断页面渲染
```

link 和 @import 的区别

link 标签定义文档与外部资源的关系。只能存在于 head 内部

1. **从属关系区别**：@import 是 CSS 提供的语法规则，只有导入样式表的作用；link 是 HTML 提供的标签，不仅可以加载 CSS 文件，还可以定义 RSS、rel 连接属性、引入网站图标等
2. **加载顺序区别**：加载页面时，link 标签引入的 CSS 被同时加载；@import 引入的 CSS 将在页面加载完毕后被加载
3. **兼容性区别**：@import 是 CSS2.1 才有的语法，故只可在 IE5+ 才能识别；link 标签作为 HTML 元素，不存在兼容性问题
4. **样式变化**：link 支持用 JS 控制 DOM 改变样式；@import 不支持

说说常见的 meta 标签

1. 基础语义信息

```html
<!-- 定义文档编码 -->
<meta charset="utf-8" />

<!-- 设置视口，移动端自适应 -->
<meta
  name="viewport"
  content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no"
/>
```

2. SEO 与搜索引擎优化

```html
<!-- 页面描述 -->
<meta name="description" content="详细描述网页内容，控制搜索结果摘要" />

<!-- 关键词 -->
<meta name="keywords" content="HTML,meta标签,web开发" />

<!-- 页面作者 -->
<meta name="author" content="author" />
```

3. 移动端和应用相关

```html
<!-- 针对iOS设备的web应用 -->
<meta name="apple-mobile-web-app-capable" content="yes" />
<meta name="apple-mobile-web-app-status-bar-style" content="black" />

<!-- Android设备 -->
<meta name="theme-color" content="#4285f4" />

<!-- 禁止跳转到移动版本 -->
<meta name="format-detection" content="telephone=no,email=no,adress=no" />

<!-- 移动端适配 -->
<meta
  name="viewport"
  content="width=device-width, initial-scale=1, maximum-scale=1"
/>
```

> **Tip**
>
> 对于 viewport， `content` 参数有以下几种：
>
> 1. `width viewport` ： 宽度(数值/device-width)
> 2. `height viewport` ： 高度(数值/device-height)
> 3. `initial-scale` ： 初始缩放比例
> 4. `maximum-scale` ： 最大缩放比例
> 5. `minimum-scale` ： 最小缩放比例

6. `user-scalable`：是否允许用户缩放

4. 安全与性能增强

```
<!-- 内容安全策略 -->
<meta http-equiv="Content-Security-Policy" content="default-src 'self'" />

<!-- 启用Internet Explorer的兼容性视图 -->
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />

<!-- 禁止百度转码 -->
<meta http-equiv="Cache-Control" content="no-siteapp" />
```

5. 社交媒体与分享优化

```
<!-- Open Graph协议，控制社交媒体分享 -->
<meta property="og:title" content="网页标题" />
<meta property="og:type" content="website" />
<meta property="og:url" content="https://example.com" />
<meta property="og:image" content="https://example.com/image.jpg" />

<!-- Twitter Card -->
<meta name="twitter:card" content="summary_large_image" />
<meta name="twitter:site" content="@username" />
```

6. 爬虫与索引控制

```
<!-- 控制搜索引擎爬虫行为 -->
<meta name="robots" content="index,follow" />

<!-- 谷歌特定爬虫指令 -->
<meta name="googlebot" content="index,follow" />
```

> **Tip**
>
> 对于 robot，`content` 参数有以下几种：
>
> 1. `index`：允许搜索引擎索引此网页
> 2. `noindex`：不允许搜索引擎索引此网页
> 3. `follow`：允许搜索引擎继续通过此网页的链接索引搜索其它的网页
> 4. `nofollow`：不允许搜索引擎通过此网页的链接索引搜索其它的网页
> 5. `all`：文件会被检索且链接可以被查询
> 6. `none`：文件将不被检索，且页面上的链接不可以被查询

7. 缓存控制

```
<!-- 缓存控制 -->
<meta http-equiv="Cache-Control" content="no-cache" />
<meta http-equiv="Pragma" content="no-cache" />
<meta http-equiv="Expires" content="0" />
```

8. 移动端浏览器扩展

```
<!-- 手机号识别 -->
<meta name="format-detection" content="telephone=no" />

<!-- 邮箱识别 -->
<meta name="format-detection" content="email=no" />
```

9. PWA 和离线应用

```
<!-- Web App Manifest链接 -->
<link rel="manifest" href="/manifest.json" />

<!-- 离线存储 -->
<meta name="apple-mobile-web-app-capable" content="yes" />
```

10. 跨域资源控制

```
<!-- 允许跨域访问 -->
<meta http-equiv="Access-Control-Allow-Origin" content="*" />
```

说说 HTML5 的离线存储

离线缓存允许 Web 应用在没有网络连接的情况下继续访问，它有以下优势：

- 离线浏览 - 用户可以在离线状态下访问页面
- 更快的访问速度 - 资源从本地加载
- 减轻服务器负载 - 浏览器只下载更新过的资源

当网络在线时，浏览器识别到 manifest 属性时，如果页面是首次加载，那么会按照该属性指向的文件中的内容进行下载并缓存到本地;如果在加载页面时，页面已经离线存储，那么浏览器会使用离线存储的资源加载页面，然后对比新旧 manifest。如果有更新，将更新后的数据重新缓存

当网络离线时，浏览器直接会使用离线存储的资源加载网页

用法:

```
<!-- 在 HTML 页面中引用 manifest 文件 -->
<html manifest="cache.manifest"></html>
```

manifest 文件的基本结构:

```
CACHE MANIFEST
# version 1.0.0

CACHE:
index.html
styles/main.css
scripts/main.js
images/logo.png

```

```
NETWORK:
*

FALLBACK:
/ offline.html
```

manifest 文件分为三个部分：

- `CACHE`： 必须缓存的资源列表
- `NETWORK`：永远不缓存、需要网络访问的资源
- `FALLBACK`：指定无法访问时的替代页面

可以在 JS 脚本中借助 `applicationCache` 来判断缓存的状态：

```
const appCache = window.applicationCache;

switch (appCache.status) {
  case 0: // UNCACHED
    console.log("未缓存");
    break;
  case 1: // IDLE
    console.log("闲置");
    break;
  case 2: // CHECKING
    console.log("检查中");
    break;
  case 3: // DOWNLOADING
    console.log("下载中");
    break;
  case 4: // UPDATEREADY
    console.log("更新就绪");
    break;
  case 5: // OBSOLETE
    console.log("废弃");
    break;
}
```

`applicationCache` 对象的事件：

```
// 更新检查
appCache.addEventListener("checking", function () {
  console.log("正在检查manifest是否更新");
});

// 更新完成
appCache.addEventListener("updateready", function () {
  if (appCache.status === appCache.UPDATEREADY) {
    appCache.swapCache(); // 切换到新缓存
    location.reload(); // 刷新页面
  }
});

// 错误处理
appCache.addEventListener("error", function (e) {
  console.log("缓存更新失败:", e);
```

```
});
```

缓存更新的方式：

1. 自动更新:

- 用户刷新页面
- manifest 文件被修改
- 通过 JavaScript 调用 update() 方法

2. 手动更新:

```
// 手动检查更新
applicationCache.update();

// 强制重新加载
applicationCache.swapCache();
```

行内元素、块级元素、空元素

1. **行内元素**

行内元素的特点：

- 不会独占一行
- 宽度由内容决定
- 不可设置宽高
- 只能包含文本或其他行内元素

典型代表：`a b span img input select strong button label textarea`

2. **块级元素**

块级元素的特点：

- 独占一行
- 可以设置宽高
- 可以包含行内元素和块级元素

典型代表：`div ul ol li dl dt dd h1 h2 h3 h4 h5 h6 p`

3. **空元素**

- 没有闭合标签
- 在 HTML5 中可以写成自闭合形式
- 不能包含任何内容

典型代表：`br hr img input link meta`

使用 display 属性可以实现元素类型的转换

- `block` 转换为块级元素
- `inline` 转换为行内元素
- `inline-block` 转换为行内块元素

img 标签 title、alt、srcset

1. **alt 属性**

`alt` 是 "alternate text" 的缩写,用于提供图片的文字描述:

```
<img src="cat.jpg" alt="一只橘色的猫咪正在睡觉" />
```

主要用途:

- 图片加载失败时显示替代文本
- 屏幕阅读器会读取 alt 内容
- 搜索引擎通过 alt 了解图片内容，有利于 SEO
- 在禁用图片的浏览器中显示文字描述

2. **title 属性**

`title` 提供图片的额外信息，鼠标悬停时显示提示文本:

```
<img src="cat.jpg" alt="睡觉的猫咪" title="这是我家的Tom猫，拍摄于2024年1月" />
```

主要用途:

- 提供补充说明或详细信息
- 鼠标悬停时显示提示
- 可以包含图片的拍摄时间、版权信息等

3. **srcset 属性**

`srcset` 用于响应式图片，允许浏览器根据设备特性选择最适合的图片版本

主要用途:

- 响应式图片加载，适配不同屏幕尺寸
- 支持高分辨率屏幕
- 优化性能，避免加载过大图片

srcset 有两种语法:

```
<!-- 1. 宽度描述符（w） -->
<img
  srcset="small.jpg 300w, medium.jpg 600w, large.jpg 900w"
  sizes="(max-width: 500px) 300px,
            (max-width: 900px) 600px,
            900px"
  src="fallback.jpg"
/>

<!-- 2. 像素密度描述符（x） -->
<img
  srcset="normal.jpg 1x, retina.jpg 2x, high-res.jpg 3x"
  src="fallback.jpg"
/>
```

搭配 sizes 属性：

```
<img
  srcset="cat-320w.jpg 320w, cat-480w.jpg 480w, cat-800w.jpg 800w"
  sizes="(max-width: 320px) 280px,
            (max-width: 480px) 440px,
            800px"
  src="cat-800w.jpg"
  alt="猫咪照片"
/>
```

这三个属性各有各的用处：

- alt 主要用于可访问性和降级体验
- title 用于提供额外信息
- srcset 用于性能优化和响应式设计

title 与 h1 的区别、b 与 strong 的区别、i 与 em 的区别

title 与 h1 的区别：

1. 作用不同：

- `<title>` 定义文档标题，显示在浏览器标签页上
- `<h1>` 定义页面内容的主标题，显示在页面内容中

2. 位置不同：

- `<title>` 必须在 `<head>` 标签中
- `<h1>` 在 `<body>` 标签中

3. SEO 意义：

- `<title>` 是搜索引擎最重要的参考信息之一
- `<h1>` 表示页面主标题，权重仅次于 title

```
<head>
  <title>我的网站 - 首页</title>
</head>
<body>
  <h1>欢迎访问我的网站</h1>
</body>
```

b 与 strong 的区别：

1. 语义不同：

- `<b>` 仅表示文字加粗，无语义化的标签
- `<strong>` 表示文本的重要性，是语义化标签

2. 用途区分：

- `<b>` 用于纯粹的视觉效果
- `<strong>` 用于强调文本的重要性

3. 可访问性：

- `<b>` 屏幕阅读器不会特殊处理

- `<strong>` 屏幕阅读器会加重语气读出

```
<p>这是一个<b>产品名称</b></p>
<p>这是一个<strong>警告信息</strong></p>
```

i 与 em 的区别：

1. 语义不同：

- `<i>` 表示不同于普通文本的内容，如外语词语、技术语言等
- `<em>` 表示强调或重读的内容，是语义化标签

2. 用途区分：

- `<i>` 用于标识专有名词、引用等
- `<em>` 用于表示文本中需要着重强调的部分

3. 可访问性：

- `<i>` 屏幕阅读器按普通文本读出
- `<em>` 屏幕阅读器会改变语调来强调

```
<p>我最喜欢的乐队是 <i>The Beatles</i></p>
<p>请<em>务必</em>在出发前确认时间</p>
```

说说 head 标签的作用

`<head>` 标签是 HTML 文档的容器，位于 `<html>` 和 `<body>` 标签之间，包含了文档的元数据

文档的头部描述了文档的各种属性和信息，包括文档的标题、在 Web 中的位置以及和其他文档的关系等。绝大多数文档头部包含的数据都不会真正作为内容显示给读者

head 标签示例：

```
<html>
  <head>
    <!-- 元数据内容 -->
  </head>
  <body>
    <!-- 页面内容 -->
  </body>
</html>
```

下面列举一些常用的 head 标签：

1. **title 标签**

在所有的 head 标签中，`<title>` 定义文档标题，是唯一必需的元素：

```
<head>
  <title>网页标题</title>
</head>
```

2. **meta 标签**

提供各种元数据信息：

```
<!-- 视口设置 -->
<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<!-- SEO 相关 -->
<meta name="description" content="网站描述" />
<meta name="keywords" content="关键词1,关键词2" />
<meta name="author" content="作者名" />

<!-- 搜索引擎抓取控制 -->
<meta name="robots" content="index,follow" />

<!-- 页面刷新/跳转 -->
<meta http-equiv="refresh" content="30" />

<!-- 兼容性设置 -->
<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<!-- Open Graph 协议 -->
<meta property="og:title" content="标题" />
<meta property="og:description" content="描述" />
<meta property="og:image" content="图片URL" />
```

3. **link 标签**

引入外部资源：

```
<!-- CSS 样式表 -->
<link rel="stylesheet" href="styles.css" />

<!-- 网站图标 -->
<link rel="icon" href="favicon.ico" />
<link rel="apple-touch-icon" href="apple-touch-icon.png" />

<!-- 预加载资源 -->
<link rel="preload" href="font.woff2" as="font" crossorigin />
<link rel="prefetch" href="secondary-page.html" />

<!-- RSS 订阅 -->
<link rel="alternate" type="application/rss+xml" title="RSS" href="/rss.xml" />
```

4. **style 标签和 script 标签**

定义内部样式和脚本：

```html
<style>
    body {
        margin: 0;
        padding: 0;
    }
</style>

<script>
    console.log('Hello World');
<script/>
```

5. **base 标签**

指定基础 URL：

```html
<base href="https://www.example.com/" /> <base target="_blank" />
```

6. **noscript 标签**

为不支持 JavaScript 的用户提供替代内容：

```html
<noscript>
  <p>请启用 JavaScript</p>
</noscript>
```

`<head>` 标签的正确使用有助于：

- 页面性能优化
- SEO 优化
- 移动设备适配
- 浏览器兼容性
- 用户体验提升

label 的作用

label 标签的主要作用是为表单元素定义标注/标签，它有两个主要用途：

1. 提升可用性：点击 label 时，关联的表单元素也会被选中/聚焦
2. 提升可访问性：屏幕阅读器可以读出标签，帮助视障用户理解表单含义

使用方式有两种：

1. 显式关联（使用 for 属性）：

```html
<label for="username">用户名：</label> <input type="text" id="username" />
```

2. 隐式关联（嵌套方式）：

```
<label>
  密码：
  <input type="password" />
</label>
```

说说 Web Worker

传统的 js 代码在运行时会阻塞页面的响应，直到 js 执行完毕后页面才会响应

Web Worker 是 HTML5 引入的一个重要特性，它允许网页在后台运行 JavaScript 脚本，而不会影响页面的响应性。本质上，它创建了一个独立于主线程的后台线程。

主要特点和作用：

1. 并行处理

- 可以执行耗时的计算操作而不会阻塞主线程
- 适合处理大量数据、复杂计算等 CPU 密集型任务
- 保持页面的流畅响应，提升用户体验

2. 独立的执行环境

- 有自己独立的上下文，不能直接访问 DOM
- 不能使用 window 对象的大部分方法和属性
- 可以使用 setTimeout、setInterval 等计时器功能
- 可以使用 XMLHttpRequest 发起请求

下面通过代码示例来展示 Web Worker 的基本使用

主线程代码（main.js）：

```
// 创建 Worker
const worker = new Worker("worker.js");

// 发送消息给 Worker
worker.postMessage({
  type: "compute",
  data: [1, 2, 3, 4, 5],
});

// 接收 Worker 返回的消息
worker.onmessage = function (e) {
  console.log("从 Worker 收到结果:", e.data);
};

// 错误处理
worker.onerror = function (error) {
  console.error("Worker 错误:", error);
};
```

Worker 线程代码（worker.js）：

```
self.onmessage = function (e) {
  if (e.data.type === "compute") {
    // 执行耗时计算
    const result = e.data.data.reduce((sum, num) => sum + num, 0);

    // 将结果发送回主线程
    self.postMessage(result);
  }
};
```

使用注意事项：

1. 通信开销

- Worker 与主线程之间的通信会有性能开销
- 对于小型计算任务，创建 Worker 的开销可能大于收益

2. 资源限制

- Worker 数量应该适度，避免创建过多
- 需要考虑内存占用，特别是处理大量数据时

3. 兼容性处理

```
// 检查浏览器是否支持 Web Worker
if (typeof Worker !== "undefined") {
  // 支持 Web Worker
  const worker = new Worker("worker.js");
} else {
  // 降级处理
  console.log("当前浏览器不支持 Web Worker");
}
```

canvas 与 svg 的区别

canvas	svg
画布,通过 js 绘制的 2D 图像,逐像素进行渲染,位置改变时,会重新渲染	可伸缩矢量图形,基于 XML 描述的 2D 图形语言, SVG 对象属性发生改变,会重新绘制
依赖分辨率	不依赖分辨率
不支持事件处理器	支持事件处理器
适合图像密集型游戏	不适合用于游戏
弱文本渲染能力	适合带有大型渲染区域的应用程序
能够以 .png，.jpg 格式保存图像	复杂度高会减慢渲染速度

说说 HTML5 中的 drag API

HTML5 Drag API 提供了原生的拖放功能实现机制，让我们可以轻松实现元素的拖拽交互。下面通过代码和实例来展示其具体用法：

在元素标签中添加了 `draggable=true` 属性后,元素将会成为一个可拖放元素,该元素通常与另一个拖放目标区域元素配合使用

- `dragstart` 事件的主体是被拖放元素，开始拖放时触发
- `dragend` 事件的主体是被拖放元素，结束拖放式触发
- `drag` 事件的主体是被拖放元素，拖放中触发
- `dragenter` 事件主体是目标元素，进入目标元素时触发
- `dragleave` 事件主体是目标元素，离开目标元素时触发
- `dragover` 事件主体是目标元素，在目标元素中拖放时触发
- `dop` 事件主体是目标元素，目标元素完全接受被拖放元素时触发