<!DOCTYPE> 声明是 HTML 文档中的一个重要部分，它告诉浏览器当前文档使用的 HTML 版本和标准模式。它的主要作用是确保浏览器以正确的模式渲染页面，避免浏览器使用旧的、不标准的渲染模式（称为“怪异模式”）。

**作用**

1. **指定文档类型**：<!DOCTYPE> 声明告诉浏览器当前文档是基于哪个 HTML 版本编写的。
2. **触发标准模式**：它确保浏览器使用标准模式（也称为“严格模式”）来渲染页面，而不是怪异模式。标准模式下，浏览器会严格按照 HTML 和 CSS 规范来解析和渲染页面，从而避免一些兼容性问题。
3. HTML5 的 <!DOCTYPE> 声明非常简洁：
4. html
5. 复制
6. <!DOCTYPE html>

Html4的Doc

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

Xhtml 1.0  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

### 1. ****HTML4****

HTML4 是 HTML 的第四个版本，发布于 1997 年，是 Web 开发早期的主要标准。

#### 特点：

* **语法宽松**：HTML4 的语法相对宽松，标签不区分大小写，属性可以不加引号，甚至可以省略某些标签（如 <p> 的结束标签）。
* **功能有限**：HTML4 主要用于结构化文档，支持基本的文本、图像、表格、表单等元素，但缺乏现代 Web 应用所需的高级功能（如多媒体、本地存储、语义化标签等）。
* **兼容性强**：HTML4 在所有浏览器中都有很好的兼容性，适合需要兼容老旧浏览器的场景。

### 2. ****XHTML****

XHTML 是 HTML 的 XML 版本，发布于 2000 年，旨在将 HTML 标准化为严格的 XML 语法。

#### 特点：

* **严格语法**：XHTML 要求所有标签必须小写，所有属性必须加引号，所有标签必须正确闭合（包括自闭合标签如 <img />）。
* **基于 XML**：XHTML 是基于 XML 的，因此必须符合 XML 的语法规则，任何语法错误都会导致页面无法正确解析。
* **功能与 HTML4 类似**：XHTML 的功能与 HTML4 基本相同，但语法更严格。

### 3. ****HTML5****

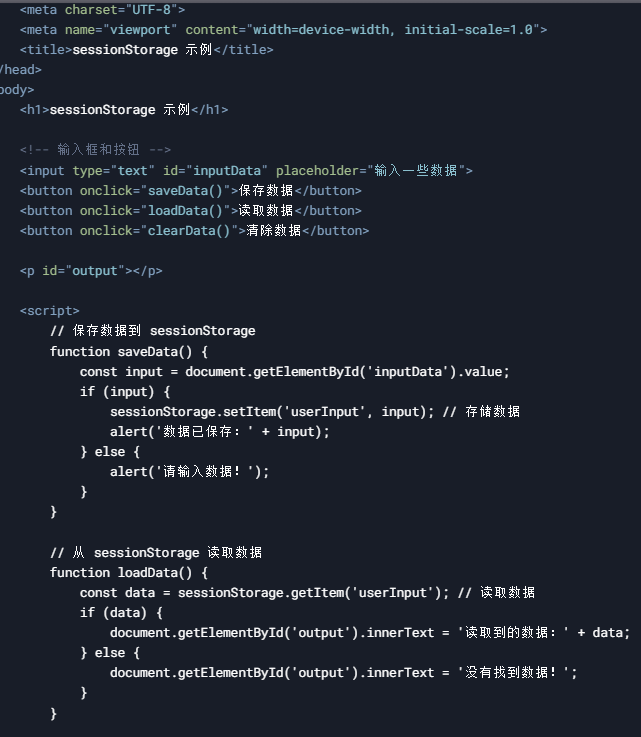
HTML5 是 HTML 的第五个版本，发布于 2014 年，是目前 Web 开发的主流标准。

#### 特点：

* **语法简化**：HTML5 的语法比 XHTML 宽松，但比 HTML4 更严格。标签不区分大小写，但推荐小写；属性可以不加引号，但推荐加引号。
* **新增功能**：**HTML5 引入了大量新功能，如语义化标签（<header>、<footer>、<article> 等）、多媒体支持（<audio>、<video>）、本地存储（**localStorage**、sessionStorage）、Canvas 绘图、Web Workers 等。**
* **跨平台支持**：HTML5 支持移动端和桌面端，适合开发现代 Web 应用。
* **向后兼容**：HTML5 兼容 HTML4 和 XHTML 的语法，同时提供了更好的浏览器兼容性。
* HTML5 支持更多的新元素，如 **<header>, <footer>, <article>, <section>**等，用于更好的语义化。
* HTML5 不需要DTD（文档类型定义），因此更加简洁。
* HTML5 支持视频和音频标签 <video> 和 <audio>，而HTML4需要使用插件。
* HTML5 引入了新的表单元素和属性，如 <input type="date">。
* HTML5 支持离线应用，通过 <canvas> 元素支持绘图和图形渲染。

# sessionStorage 是 HTML5 提供的一种客户端存储机制，用于在浏览器会话期间存储数据。它的特点是：

* 数据仅在当前会话期间有效，关闭浏览器标签页或窗口后数据会被清除。
* 存储容量通常为 5MB（具体取决于浏览器）。
* 数据存储在客户端，不会发送到服务器。



# Localstorage:



1. **安全性**：localStorage 存储的数据仅在客户端有效，不会发送到服务器，因此适合存储一些持久化数据（如用户偏好设置、缓存数据等）。
2. **跨页面共享**：localStorage 的数据可以在同一域名下的所有页面之间共享。如果需要跨页面共享数据，可以使用 localStorage。
3. **存储容量**：localStorage 的存储容量通常为 5MB 到 10MB（具体取决于浏览器）。如果需要存储大量数据，可以考虑使用 IndexedDB。

这句话并不自相矛盾，它准确地描述了 localStorage 的工作原理。让我来解释一下：

跨页面共享：指的是在同一个浏览器中，在同一域名下的不同页面之间可以共享数据。例如，如果您在一个网页（比如 example.com/page1.html）上设置了 localStorage 数据，那么当用户导航到同一域名下的另一个网页（比如 example.com/page2.html）时，这个数据仍然可用。

局限性：需要注意的是，localStorage 的数据共享仅限于相同的协议（HTTP 或 HTTPS）、域名和端口。这意味着 http://example.com 和 https://example.com 会被视为不同的域，example.com 和 sub.example.com 也会被视为不同的域。此外，localhost:3000 和 localhost:8080 由于端口号不同，也被视为不同的域。

因此，localStorage 确实可以在同一域名下的所有页面之间共享数据，只要这些页面满足上述条件。如果需要跨不同域名或子域名共享数据，则不能直接使用 localStorage，而可能需要其他机制，如通过服务器端处理、使用 postMessage API 来安全地传递信息给不同源的窗口，或者使用 cookie（带有适当的域设置）等方法。

# CSS 盒子模型（Box Model）

是网页布局的基础，它描述了网页元素如何在页面上呈现。每个HTML元素都被视为一个矩形的盒子，这个盒子由四个部分组成：内容区（content）、内边距（padding）、边框（border）和外边距（margin）。下面是对每一部分的详细解释：

内容区（Content）：

这是盒子中实际的内容区域，包括文本、图片等。width 和 height 属性定义的是内容区的尺寸。

内边距（Padding）：

内边距位于内容区周围，用来创建内容与其他部分之间的空间。它影响到盒子内部的空间，增加了内容区与边框之间的距离。内边距也是透明的，并不会影响到盒子的实际位置。

边框（Border）：

边框围绕着内边距和内容区，用来分隔不同的元素或者装饰元素。可以通过设置边框的颜色、样式（如实线、虚线等）和宽度来改变其外观。

外边距（Margin）：

外边距位于边框之外，用于控制盒子与其他盒子之间的间距。它是透明的，并且可以用来防止或创建元素之间的重叠。外边距还可以导致相邻的垂直外边距发生合并（称为外边距折叠），从而形成一个单一的外边距。

在CSS中，默认情况下，当您为一个元素设置了宽度和高度时，这些值仅应用于内容区。如果您还指定了内边距、边框和外边距，那么整个元素（即“盒子”）的总宽度和高度将是内容区宽度/高度加上左右内边距、左右边框宽度、以及左右外边距之和（对于宽度而言；对于高度则是上下方向的相应参数相加）。

此外，CSS 提供了一个名为 box-sizing 的属性，它可以改变这种计算方式：

box-sizing: content-box;（默认行为）：宽度和高度只适用于内容区。

box-sizing: border-box;：宽度和高度则同时适用于内容区、内边距和边框，但不包括外边距。这意味着如果给定一个固定宽度，增加内边距或边框宽度不会使元素的总宽度超出该固定宽度。

理解盒子模型对于正确地设计和布局网页至关重要，因为它直接影响元素在页面上的大小和位置。

# CSS Grid

.grid-container {

display: grid;

grid-template-columns: 1fr 3fr; /\* 定义两列，第一列占1份空间，第二列占3份 \*/

grid-template-rows: auto 1fr auto; /\* 定义三行，高度根据内容自动调整 \*/

gap: 10px; /\* 设置网格线之间的间距 \*/

}

.item1 {

grid-column: 1 / span 2; /\* Header 跨越两列 \*/

background-color: #f06d06;

padding: 20px;

text-align: center;

}

是的，CSS 伪类是预先定义好的，它们是 CSS 规范的一部分，旨在提供一种标准的方式来选择和样式化特定状态下的元素。这意味着开发者不需要自己创建这些伪类；相反，他们可以直接在样式表中使用这些预定义的伪类来增强页面元素的行为和外观。

# 预定义的伪类

以下是一些常见的预定义伪类：

:link 和 :visited：分别用于未访问和已访问的链接。

:hover：当用户将鼠标悬停在元素上时应用。

:active：当元素被激活（如点击）时应用。

:focus：当元素获得焦点时应用，例如通过键盘导航选择的输入框。

:target：当URL的片段标识符匹配元素的ID时应用，即当用户通过链接跳转到页面中的某个部分时。

:nth-child(n)、:nth-of-type(n)：用于选择父元素下的第n个子元素或特定类型的第n个子元素。

:first-child 和 :last-child：分别用于选择作为其父元素的第一个或最后一个子元素的元素。

:first-of-type 和 :last-of-type：分别用于选择作为其类型第一个或最后一个兄弟元素的元素。

:only-child 和 :only-of-type：分别用于选择唯一一个子元素或唯一一个该类型的兄弟元素。

:empty：用于选择没有任何子元素（包括文本节点）的元素。

:not(selector)：用于否定选择器，排除与指定选择器匹配的元素。

:root：匹配文档的根元素，在HTML中通常是<html>元素。

:checked：用于选择被选中的<input>元素（如单选按钮或复选框）。

:disabled 和 :enabled：用于选择禁用或启用的表单元素。

:read-only 和 :read-write：用于选择只读或可编辑的表单元素。

:placeholder-shown：用于选择当前显示占位符文本的 <input> 或 <textarea> 元素。

:fullscreen：用于选择处于全屏模式的元素。

@media (max-width: 600px) {

/\* 在此定义小屏幕上的样式 \*/

}

<meta name="viewport" content="width=device-width, initial-scale=1">

闭包的核心在于：

1. **函数嵌套**：一个函数内部定义了另一个函数。
2. **外部变量引用**：内部函数引用了外部函数中的变量。
3. **返回内部函数**：外部函数返回内部函数，使得内部函数可以在外部函数的作用域之外执行。

### 原型链（Prototype Chain）

当访问一个对象的属性或方法时，如果这个对象本身没有这个属性或方法，解释器就会去该对象的原型（prototype）中查找。如果原型中也没有，就会继续查找原型的原型，这样一直回溯到Object.prototype。这就是所谓的原型链

在JavaScript中，每当创建一个函数时，这个**函数就会自动拥有一个prototype属性**，这是一个**带有constructor属性的对象**，**constructor属性指向函数自身**。

# New Class 时的具体操作:

1. 创建一个新对象：

**javascript**

复制

var obj = {};

1. 设置原型链：

**javascript**

复制

obj.\_\_proto\_\_ = MyClass.prototype;

1. 绑定this并执行构造函数：

**javascript**

复制

MyClass.call(obj);

这一步会执行MyClass函数体中的代码，其中this指向obj。

1. 如果构造函数返回了一个对象，那么这个对象会被返回；如果没有返回对象，则返回新创建的对象obj。

### 浅拷贝（Shallow Copy）

浅拷贝只复制对象的第一层属性，如果属性值是基本类型，则拷贝的是基本类型的值；如果属性值是引用类型，则拷贝的是内存地址，即新旧对象共享同一个引用类型的值。

使用Object.assign实现浅拷贝

**javascript**

复制

let obj = { a: 1, b: { c: 2 } };

let shallowCopy = Object.assign({}, obj);

console.log(shallowCopy); *// { a: 1, b: { c: 2 } }*

使用扩展运算符实现浅拷贝

**javascript**

复制

let obj = { a: 1, b: { c: 2 } };

let shallowCopy = { ...obj };

console.log(shallowCopy); *// { a: 1, b: { c: 2 } }*

深拷贝（Deep Copy）

深拷贝会复制对象的所有层级属性，新旧对象不共享任何东西，即使属性值是引用类型，也会被复制一份新的。

使用JSON.parse和JSON.stringify实现深拷贝

**javascript**

复制

let obj = { a: 1, b: { c: 2 } };

let deepCopy = JSON.parse(JSON.stringify(obj));

console.log(deepCopy); *// { a: 1, b: { c: 2 } }*

注意：这种方法有局限性，它不能复制函数、undefined、循环引用的对象，以及不能正确处理Date对象和正则表达式对象。

### 事件冒泡（Event Bubbling）

事件冒泡是指事件从触发该事件的元素开始，然后逐级向上传播到较为顶层的元素。也就是说，当一个事件发生在一个元素上时，会首先运行该元素的事件处理程序，然后运行其父元素的事件处理程序，接着是父元素的父元素的事件处理程序，以此类推，直到到达document对象。

例如，假设有如下HTML结构：

**html**

复制

<div id="outer">

<div id="inner">

<button id="myButton">点击我</button>

</div>

</div>

如果点击了按钮，那么事件的传播顺序是：

复制

button -> inner -> outer -> document

如果每个元素都有相应的事件处理程序，那么它们将按照这个顺序被触发。

**事件捕获（Event Capturing）**

与事件冒泡相反，事件捕获是从最顶层的元素开始，然后逐级向下传播到具体的元素。在事件捕获阶段，事件从document对象开始，然后向下传播到目标元素的父元素，最终到达目标元素。

使用同样的HTML结构，事件捕获阶段的事件传播顺序是：

复制

document -> outer -> inner -> button

**DOM事件流**

现代浏览器的事件流包括三个阶段：事件捕获阶段、目标阶段和事件冒泡阶段。

1. **事件捕获阶段**：事件从document对象向下传播到目标元素。
2. **目标阶段**：事件到达目标元素，触发目标元素的事件处理程序。
3. **事件冒泡阶段**：事件从目标元素向上传播回document对象。

**addEventListener的第三个参数**

在使用addEventListener方法绑定事件处理程序时，可以指定一个布尔值作为第三个参数，该参数决定了事件处理程序是在捕获阶段还是冒泡阶段被触发。

* addEventListener(event, function, useCapture)

如果useCapture为false（默认值），则事件处理程序在冒泡阶段被触发。

如果useCapture为true，则事件处理程序在捕获阶段被触发。

**addEventListener的第三个参数**

在使用addEventListener方法绑定事件处理程序时，可以指定一个布尔值作为第三个参数，该参数决定了事件处理程序是在捕获阶段还是冒泡阶段被触发。

addEventListener(event, function, useCapture)

如果useCapture为false（默认值），则事件处理程序在冒泡阶段被触发。

如果useCapture为true，则事件处理程序在捕获阶段被触发。

Function.prototype.bind 是JavaScript语言中的一个方法，它用于创建一个新的函数，这个新的函数在被调用时，其this关键字会被绑定到传入bind方法的第一个参数上，而其余的参数将作为新函数的参数序列。

以下是bind方法的一些核心特点和用途：

**语法**

**javascript**

复制

function.bind(thisArg[, arg1[, arg2[, ...]]])

参数

* thisArg: 当绑定函数被调用时，该参数会作为this的值。
* arg1, arg2, ...（可选）: 当绑定函数被调用时，这些参数将先于其他实参被传入。

**返回值**

返回一个原函数的拷贝，并拥有指定的this值和初始参数。

**用途**

1. **改变函数的this指向**：在JavaScript中，函数的this关键字通常取决于函数是如何被调用的。使用bind可以确保函数无论怎样被调用，this都指向你指定的对象。
2. **偏函数应用（Partial Function Application）**：通过bind可以预设一些函数的参数，生成一个新函数，这样在调用时只需要传入剩余的参数。
3. **作为构造函数使用的绑定函数**：当你使用new操作符调用绑定函数时，提供的thisArg将被忽略，而new操作符的常规规则适用。

**示例**

**javascript**

复制

function showName(label) {

console.log(`${label}:${this.name}`);

}

var person = { name: '张三' };

*// 使用bind创建一个新函数，this指向person对象*

var showPersonName = showName.bind(person);

showPersonName('Person'); *// 输出: Person: 张三*

*// 使用bind预设参数*

var sayHelloToPerson = showName.bind(person, 'Hello');

sayHelloToPerson(); *// 输出: Hello: 张三*

在上述示例中，bind方法使得showName函数在调用时this总是指向person对象，而不管它是如何被调用的。

bind方法在事件处理、回调函数、异步操作等场景中非常有用，它帮助开发者确保函数执行时的上下文是正确的，尤其是在面对复杂的代码结构时。

# Web性能优化

是指通过各种技术和实践来提高网站或Web应用的加载速度和响应时间，从而提升用户体验。以下是一些常见的Web性能优化方法：

1. 内容分发网络（CDN）

使用CDN可以将静态资源分布到全球多个服务器节点上，用户可以从距离最近的节点获取资源，减少延迟。

2. 减少HTTP请求

减少页面中的元素数量，合并CSS、JavaScript文件，使用CSS Sprites等技术可以有效降低HTTP请求数量。

3. 压缩资源

对HTML、CSS、JavaScript文件进行压缩，去除不必要的空格、注释等内容；同时对图片等媒体文件使用适当的压缩工具，在不影响质量的前提下减小文件大小。

4. 缓存策略

利用浏览器缓存，为静态资源设置合适的Cache-Control和Expires头信息，使得用户在再次访问时可以直接从本地缓存读取资源。

5. 使用异步加载

对于非关键路径上的脚本，采用async或<script src="script.js" defer></script>

属性，使它们不会阻塞页面解析，从而加快页面渲染速度。

6. 代码分割与懒加载

将应用程序拆分为更小的块，并仅在需要时加载特定部分（如路由组件），以减少初始加载时间。

7. 图片优化

使用现代图像格式（如WebP），根据设备特性提供适当分辨率的图片，以及考虑使用srcset和sizes属性实现响应式图片。

8. 减少重定向

避免不必要的URL重定向，因为每次重定向都会增加额外的HTTP请求。

9. DNS预取（DNS Prefetching）

提前解析可能的外部资源域名，以便当这些资源真正被请求时，DNS查找已经完成。

10. Server-Side Rendering (SSR) 或 Static Site Generation (SSG)

对于动态内容，可以采用服务端渲染；对于静态内容，生成静态页面，两者都能显著提高首屏加载速度。

11. 减少DOM复杂度

简化HTML结构，避免过于复杂的DOM树，这有助于提高浏览器解析和渲染效率。

12. HTTP/2 和 HTTP/3

这两个协议支持多路复用等功能，能够显著改善资源并行下载的能力，减少延迟。

13. 消除渲染阻塞资源

确保样式表位于文档头部，脚本尽可能靠后放置或者标记为异步加载，确保页面尽快开始渲染。

14. 使用Service Workers

Service Workers 可以拦截网络请求并在客户端缓存资源，允许离线访问并加速后续页面加载。

实施上述优化措施时，应该结合实际情况选择最适合的方法，并且持续监控和测试以确保最佳效果。

# Defer的用法示例:

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

1. **前端工程化的概念及其重要性** 前端工程化是指将软件工程的方法和实践应用于前端开发的过程，它包括模块化、组件化、自动化构建、性能优化、代码质量控制、持续集成和部署等。其重要性在于提高开发效率、保证代码质量、优化产品性能、降低维护成本，并且使得前端开发规模化、标准化和可协作。
2. **Web性能优化方法**

* 资源压缩与合并：**减少HTTP请求，减小文件体积**。
* 缓存利用：**使用浏览器缓存、CDN等。**
* 代码优化：减少DOM操作，避免重绘和回流。
* 图片优化：使用适当格式的图片，懒加载。
* CSS和JavaScript优化：压缩、合并、懒加载。
* 使用HTTP/2：多路复用，头部压缩等特性提高加载速度。
* 服务端渲染（SSR）或静态站点生成（SSG）。

1. **前端模块化方案**

* AMD（Asynchronous Module Definition）：异步模块定义，代表库如RequireJS。
* CMD（Common Module Definition）：通用模块定义，代表库如SeaJS。
* CommonJS：主要用于服务器端，Node.js采用此规范。
* ES6模块化：使用import和export语法，现代浏览器原生支持。

1. **前端组件化的理解及实现** 前端组件化是将UI拆分成独立的、可复用的组件，每个组件包含自己的逻辑和样式。实现组件化可以通过以下方式：

* 使用Web Components标准：自定义元素、Shadow DOM等。
* 使用框架提供的组件系统，如React的JSX、Vue的.vue文件、Angular的组件。

1. **模块化规范的区别**

* AMD和CMD都是异步模块定义，但CMD推崇依赖就近，AMD推崇依赖前置。
* CommonJS主要用于服务器端，同步加载模块，适用于Node.js环境。
* ES6模块化是语言层面的规范，支持静态分析和Tree Shaking，适用于现代浏览器。

1. **事件循环（Event Loop）、微任务和宏任务** 事件循环是JavaScript的执行机制，它负责监听调用栈和任务队列，并按顺序执行代码。微任务是ES6中的Promise和MutationObserver等，它们在宏任务之后、下一次事件循环之前执行。宏任务包括script代码执行、setTimeout、setInterval、I/O操作等。
2. **浏览器的渲染过程**

* 解析HTML生成DOM树。
* 解析CSS生成CSSOM树。
* 将DOM树和CSSOM树结合生成渲染树。
* 布局（Layout）：计算元素的位置和大小。
* 绘制（Painting）：将元素绘制在屏幕上。

1. **前端安全性问题及防范**

* XSS（跨站脚本攻击）：通过转义用户输入、使用CSP（内容安全策略）等防范。
* CSRF（跨站请求伪造）：使用CSRF Token、检查Referer或使用SameSite Cookie属性等防范。

1. **前端跨域请求方法**

* JSONP：利用<script>标签无跨域限制的特性。
* CORS（跨域资源共享）：服务器设置Access-Control-Allow-Origin响应头。
* 代理服务器：通过同源服务器转发请求。
* Websocket：支持跨域的实时通信协议。

1. **响应式设计及实现** 响应式设计是指网站能够根据不同设备和屏幕尺寸自动调整布局和内容。实现方法：

* 使用媒体查询（Media Queries）调整不同屏幕下的样式。
* 使用百分比、vw/vh单位定义布局。
* 使用弹性布局（Flexbox）或网格布局（Grid）。

1. **前端构建工具的了解及作用** Webpack是一个模块打包器，它将模块和依赖打包成一个或多个bundle。Gulp是一个任务运行器，用于自动化前端工作流程。它们的作用包括模块打包、代码转译、文件优化、自动化测试等。
2. **前端框架的理解及区别** React是一个用于构建用户界面的JavaScript库，Vue是一个渐进式JavaScript框架，Angular是一个完整的MEAN栈框架。区别在于设计理念、语法、组件化和状态管理等方面。
3. **前端性能监控方法**

* 使用浏览器性能API，如Performance API。
* 使用第三方性能监控服务，如Google Analytics、Lighthouse。
* 自定义性能监控脚本，统计页面加载时间、资源加载时间等。

1. **PWA的优势** PWA（Progressive Web Apps）优势包括：

* 可靠：即使在网络不稳定的情况下也能快速加载。
* 快速：流畅的用户体验，类似原生应用的交互。
* engagement：通过推送通知等方式增强用户粘性。

1. **Serverless架构的理解及在前端中的应用** Serverless架构允许开发者专注于编写和部署代码，无需管理服务器。在前端应用中，可以使用Serverless函数处理API请求、数据计算等，降低运维成本。
2. **前后端分离及实现** 前后端分离是指前端和后

**重绘（Repaint）**： 重绘是指浏览器重新绘制页面的一部分，但不改变页面的布局。当元素的样式发生变化，但并不影响其在文档流中的位置时，例如改变元素的背景颜色、文字颜色等，浏览器会触发重绘。重绘是一个相对轻量级的过程，因为它只涉及到样式的变化，而不涉及到布局。

**回流（Reflow）**： 回流（也称为重排）是指浏览器重新计算元素的布局位置和大小，然后重新绘制受影响的部分。回流是一个比较重的操作，因为它可能涉及到整个页面的布局变化。以下操作可能会导致回流：

* 改变DOM结构（添加或删除元素）。
* 改变元素的尺寸（宽高、边距、边框等）。
* 内容变化，如文本输入或图片加载。
* 访问某些属性，如offsetWidth或scrollTop。

回流通常会导致重绘，因为布局变化后需要重新绘制页面，但重绘不一定需要回流