

CSS 变量：从基础使用到高级技巧

马浩琨

Nov 25, 2025

用原生 CSS 打造动态、可维护与惊艳的现代 Web 界面

1 开篇：为什么我们需要 CSS 变量？

在传统 CSS 开发中，维护全局样式值如主题颜色或字体大小通常需要手动复制粘贴和查找替换，这在小型项目中尚可应付，但在大型项目中却演变成一场维护噩梦。每次设计变更都意味着遍历无数文件，不仅效率低下，还极易引入错误。CSS 自定义属性，俗称 CSS 变量，正是为解决这些问题而生。它的核心价值在于提升可维护性，因为只需在一处修改变量值，所有引用该变量的地方都会自动更新；增强可读性，语义化的变量名如 `--primary-color` 远比魔术数字如 `#a23e4f` 更易于理解；提供灵活性，支持动态修改，尤其在 JavaScript 配合下，其能力远超预处理器变量；此外，独特的作用域机制允许变量在不同上下文中被覆盖和控制，实现了强大的级联能力。

2 基础篇：语法与核心概念

定义 CSS 变量时，我们使用以两个减号开头的语法，例如 `--main-color: #3498db;` 这行代码定义了一个名为 `--main-color` 的变量，并将其值设置为蓝色。变量名可以自定义，但建议使用描述性名称以提高代码可读性。使用变量时，我们通过 `var()` 函数引用，其语法为 `var(--variable-name, fallback-value)`，其中第二个参数是可选的默认值。例如，`color: var(--main-color);` 会将元素的文本颜色设置为变量值，而 `height: var(--header-height, 100px);` 则在变量未定义时回退到 100 像素高度，这确保了样式的健壮性。

变量的作用域与级联是 CSS 变量的关键特性。全局变量通常在 `:root` 伪类上定义，使其在整个文档中可用，例如 `:root { --global-color: red; }` 这行代码将 `--global-color` 定义为红色，并可在任何元素中引用。局部变量则可以在任何选择器内定义，只在该选择器及其子元素中生效，例如 `.component { --local-bg: blue; background: var(--local-bg); }` 这里，`--local-bg` 变量仅在 `.component` 类及其子元素中可用，背景色被设置为蓝色。通过演示案例，我们可以看到当局部变量与全局变量同名时，局部变量会覆盖全局变量，这遵循 CSS 的级联规则，例如在某个元素内定义 `--color: green;` 会优先于 `:root` 中的 `--color: red;`。

3 进阶篇：实战技巧与应用场景

构建动态主题是 CSS 变量的一个典型应用。思路是在 `:root` 上定义多套主题变量，例如亮色和暗色主题。实现时，通过切换 `body` 元素的类名，如添加 `.theme-dark`，利用 CSS 优先级覆盖全局变量。例如，在 `:root` 中定义 `--bg-color: white;` 和 `--text-color: black;` 作为默认亮色主题，然后在 `.theme-dark` 类中重新定义 `--bg-color: black;` 和 `--text-color: white;`；当通过 JavaScript 切换类名时，页面背景和文本颜色会自动更新，无需重新加载或修改大量代码。这种方法的优势在于它纯 CSS 驱动，减少了 JavaScript 的依赖，同时保持了高性能。

与 JavaScript 联动是 CSS 变量的另一大亮点。核心 API 包括 `setProperty` 和 `getPropertyValue` 方法。例如，使用 `getComputedStyle(element).getPropertyValue('--my-variable')` 可以读取元素上变量的当前值，这在需要获取动态样式时非常有用。而 `element.style.setProperty('--my-variable', 'newValue')` 则允许我们修改变量值，例如在用户交互时实时调整布局。实战案例中，我们可以创建一个简单的控制面板，通过滑块调整 `--spacing` 变量值来改变页面元素的间距；另一个案例是基于鼠标位置动态改变背景色，通过 JavaScript 计算坐标并更新 `--bg-hue` 变量，实现流畅的视觉反馈。

用于 `calc()` 计算时，CSS 变量可以与数学运算结合，实现动态布局。例如，定义 `--spacing: 10;` 然后使用 `calc(var(--spacing) * 1px)` 来设置边距，结果为 10 像素。这允许我们通过修改变量值来轻松调整布局，而无需硬编码单位。另一个例子是 `width: calc(100% - var(--sidebar-width));`；这里，元素的宽度会根据 `--sidebar-width` 变量的值动态计算，确保布局的适应性。这种方法的灵活性在于它支持复杂的表达式，例如结合多个变量进行加减乘除运算。

在伪类和媒体查询中使用 CSS 变量，可以进一步增强样式的动态性。例如，在 `:hover` 伪类中修改变量，可以实现悬停效果，如 `.button:hover { --scale: 1.1; transform: scale(var(--scale)); }` 这会使按钮在悬停时放大。在媒体查询中，我们可以重新定义变量以适应不同屏幕尺寸，例如 `@media (max-width: 768px) { :root { --font-size: 14px; --container-padding: 10px; } }` 这行代码在屏幕宽度小于 768 像素时，将字体大小和容器内边距调整为更小的值，实现响应式设计。这种用法确保了样式在不同设备上的一致性，同时减少了代码重复。

4 高级篇：探索边界与最佳实践

用变量控制多个值是 CSS 变量的一个高效技巧。例如，将 `box-shadow` 的多个参数存储在一个变量中，如 `--shadow: 0 2px 4px rgba(0,0,0,.1);`；然后在样式中使用 `box-shadow: var(--shadow);`；这简化了样式的管理，因为只需修改变量值即可统一更新所有阴影效果。类似地，我们可以将渐变、边框等复杂值定义为变量，提高代码的可维护性和一致性。

构建微型 CSS 框架时，CSS 变量提供了高度可配置性。例如，定义一个按钮组件，通过变量控制其外观：`--button-bg-color: #007bff;` 和 `--button-padding: 10px 20px;`；然后在按钮样式中引用这些变量，如 `background: var(--button-bg-color); padding: var(--button-padding);`；用户只需修改变量值，即可自定义按钮的背景色和内边距，而无需修改 CSS 规则。这种方法适用于构建可重用的 UI 库，通过暴露变量接口，允许开发者轻松定制主题。

实现动态动画时，CSS 变量可以与 JavaScript 结合，创造交互式体验。例如，定义 `--animation-duration: 1s;`；然后在动画中使用 `animation-duration: var(--animation-duration);`；通过 JavaScript 动态

修改 `--animation-duration` 的值，例如基于用户输入调整为 2 秒，可以改变动画速度。另一个例子是使用变量控制关键帧，如 `@keyframes slide { from { left: var(--start-pos); } to { left: var(--end-pos); } }` 这允许我们通过修改变量来调整动画的起始和结束位置，实现非线性的动态效果。在注意事项与最佳实践中，命名规范至关重要。建议使用语义化、一致的命名，例如遵循 BEM 风格：`--component-property-state` 如 `--button-background-hover` 这提高了代码的可读性和可维护性。提供备用方案是另一个关键点，始终在 `var()` 函数中使用第二个参数作为降级值，例如 `color: var(--primary-color, black);` 这确保了在变量未定义时样式仍能正常显示。单位处理方面，存储无单位的值并在使用时通过 `calc()` 添加单位，例如 `--spacing: 10; margin: calc(var(--spacing) * 1px);` 这提供了最大灵活性，因为我们可以轻松切换单位而不修改变量定义。浏览器支持方面，现代浏览器对 CSS 变量有高支持度，但对于旧版浏览器，应采用渐进增强策略，例如使用 `@supports` 规则检测支持情况，并提供回退样式。

回顾本文要点，CSS 变量从基础语法到高级应用，都展现了其在提升可维护性、灵活性和动态性方面的强大能力。通过定义和使用变量，开发者可以构建更易于管理的样式系统；结合 JavaScript 和计算函数，更能实现复杂的交互效果。鼓励读者在下一个项目中大胆尝试 CSS 变量，无论是简单的主题切换还是复杂的动态布局，它都能成为现代 Web 开发中不可或缺的工具。展望未来，CSS 变量与 Houdini 等新兴技术的结合，将开启更多可能性，例如通过自定义绘制和布局 API，实现更高效的样式控制。