CSS面试题库（44题）

1. CSS中的animation、transition、transform有什么区别?

参考答案:

在CSS中，`animation`、`transition`和`transform`是用来创建动画效果的关键属性，它们各自具有不同的作用和特点。

animation:

- 用于创建复杂的动画序列，可以控制多个关键帧
- 可以设置动画的持续时间、延迟、重复次数、播放方向等
- 需要配合@keyframes规则定义动画的各个阶段

transition:

- 用于指定在元素状态改变时，要以何种方式过渡到新状态
- 通过指定过渡的属性、持续时间、动画方式、延迟时间等来控制过渡效果
- 适用于元素从一种状态平滑过渡到另一种状态

transform:

- 用于对元素进行变形，例如平移、旋转、缩放、倾斜等
- 通常与transition或animation结合使用，使得变形动画更加平滑
- 不会影响文档流，只是视觉上的变化

---

2. 怎么做移动端的样式适配?

参考答案:

移动端样式适配的常见方法：

1. 响应式设计（Responsive Design）：

- 使用媒体查询(@media)针对不同屏幕尺寸设置不同样式
- 采用流式布局，使用百分比、vw/vh等相对单位
- 设置合适的viewport meta标签

2. 弹性布局：

- 使用Flexbox和Grid布局
- 相对单位：rem、em、vw、vh等
- 避免使用固定像素值

3. 移动端优先：

- 先设计移动端样式，再适配桌面端
- 渐进增强的设计理念

4. 图片和多媒体适配：

- 使用响应式图片（srcset、picture标签）
- 压缩优化图片大小
- 使用矢量图标（SVG、字体图标）

---

3. 相邻的两个inline-block节点为什么会出现间距，该怎么解决？

参考答案：

出现间距的原因：

inline-block元素之间的空白符（空格、换行、制表符）会被浏览器解析为一个空格字符，从而产生间距。

解决方法：

1. 移除空格：

```
代码块
1   <div>item1</div><div>item2</div>
```

2. 使用font-size: 0：

```
代码块
1   .parent {
2       font-size: 0;
3   }
4   .child {
5       font-size: 14px;
6   }
```

3. 使用margin负值：

```
代码块
1   .inline-block {
2       margin-right: -4px;
3   }
```

4. 使用flexbox：

```
代码块
1   .parent {
2       display: flex;
3   }
```

5. 使用float：

```
代码块
1   .inline-block {
2       float: left;
3   }
```

---

4. CSS Grid网格布局的基本概念和使用方法？

参考答案：

基本概念：

- Grid是二维布局系统，可以同时处理行和列
- 由网格容器（Grid Container）和网格项目（Grid Items）组成
- 通过网格线（Grid Lines）划分网格轨道（Grid Tracks）

容器属性：

```
代码块
1   .grid-container {
2       display: grid;
3       grid-template-columns: 1fr 2fr 1fr; /* 列的定义 */
4       grid-template-rows: 100px 200px; /* 行的定义 */
5       grid-gap: 10px; /* 网格间距 */
6       grid-template-areas: /* 区域命名 */
7           "header header header"
8           "sidebar content content";
9   }
```

项目属性：

```css
.grid-item {
    grid-column: 1 / 3; /* 占据列1到列3 */
    grid-row: 1 / 2; /* 占据行1到行2 */
    grid-area: header; /* 指定区域 */
}
```

兼容性：

- 现代浏览器支持良好
- IE 10+部分支持，需要使用-ms-前缀

---

5. CSS3新增了哪些特性？

参考答案：

1. 新增选择器：

- 属性选择器：`[attr^="value"]`、`[attr$="value"]`、`[attr*="value"]`
- 伪类选择器：`:nth-child()`、`:nth-of-type()`、`:not()`
- 伪元素选择器：`::before`、`::after`

2. 边框和背景：

- `border-radius`：圆角边框
- `box-shadow`：盒子阴影
- `border-image`：边框图片
- 多重背景：`background-image`支持多个值

3. 文字效果：

- `text-shadow`：文字阴影
- `word-wrap`：文字换行

- `@font-face`：自定义字体

4. 颜色：

- RGBA颜色：`rgba(255, 0, 0, 0.5)`
- HSLA颜色：`hsla(120, 100%, 50%, 0.3)`
- 渐变：`linear-gradient()`、`radial-gradient()`

5. 动画和变换：

- `transition`：过渡效果
- `transform`：2D/3D变换
- `animation`：关键帧动画

6. 布局：

- Flexbox弹性布局
- Grid网格布局
- Multi-column多列布局

---

6. CSS实现动画的方式有哪些?

参考答案：

1. Transition过渡动画：

代码块

```
.box {
    transition: all 0.3s ease;
}
.box:hover {
    transform: scale(1.2);
}
```

2. Transform变换动画：

代码块

```
.box {
    transform: translateX(100px) rotate(45deg) scale(1.5);
}
```

3. Animation关键帧动画：

代码块

```
@keyframes slideIn {
    from { transform: translateX(-100%); }
    to { transform: translateX(0); }
}
.box {
    animation: slideIn 1s ease-in-out;
}
```

4. JavaScript动画：

- 使用 `requestAnimationFrame`
- 修改元素的style属性
- 使用动画库如GSAP、Anime.js

5. SVG动画：

- SMIL动画
- CSS动画应用于SVG元素
- JavaScript控制SVG动画

6. Canvas动画：

- 使用Canvas API绘制动画帧
- WebGL 3D动画

7. 什么是回流和重绘？如何减少回流和重绘？

参考答案：

回流（Reflow）：

- 当元素的几何信息发生变化时，浏览器需要重新计算元素的位置和大小
- 触发条件：添加/删除元素、改变尺寸、改变位置、改变字体大小等

重绘（Repaint）：

- 当元素的外观发生变化但几何信息不变时，浏览器重新绘制元素
- 触发条件：改变颜色、背景、阴影等样式属性

减少回流和重绘的方法：

1. 批量修改DOM：

代码块

```
// 不好的做法
element.style.width = '100px';
element.style.height = '100px';
element.style.background = 'red';

// 好的做法
element.className = 'new-style';
```

2. 使用文档片段：

代码块

```
const fragment = document.createDocumentFragment();
// 在fragment中操作
document.body.appendChild(fragment);
```

3. 使用transform和opacity：

```
代码块
1   /* 使用transform代替改变位置 */
2   .move {
3       transform: translateX(100px);
4   }
```

4. 避免频繁读取会引起回流的属性：

- offsetTop、offsetLeft、offsetWidth、offsetHeight
- clientTop、clientLeft、clientWidth、clientHeight
- scrollTop、scrollLeft、scrollWidth、scrollHeight

---

8. CSS选择器的优先级是怎样的?

参考答案：

优先级计算规则：

1. 内联样式：1000
2. ID选择器：100
3. 类选择器、属性选择器、伪类选择器：10
4. 标签选择器、伪元素选择器：1
5. 通配符选择器：0

特殊规则：

- `!important` 具有最高优先级
- 相同优先级时，后定义的样式覆盖先定义的

- 继承的样式优先级最低

示例：

```
代码块
1   /* 优先级：1 + 10 + 1 = 12 */
2   div.container p { color: red; }
3
4   /* 优先级：100 */
5   #header { color: blue; }
6
7   /* 优先级：1000 */
8   <div style="color: green;">
9
10  /* 最高优先级 */
11  .text { color: yellow !important; }
```

最佳实践：

- 避免使用!important
- 尽量使用类选择器
- 保持选择器简洁
- 使用CSS预处理器管理复杂样式

---

9. 什么是BFC（块级格式化上下文）？

参考答案：

BFC定义：

Block Formatting Context，块级格式化上下文，是Web页面中盒模型布局的CSS渲染模式，指一个独立的渲染区域。

BFC的特性：

1. 内部的Box会在垂直方向一个接一个地放置
2. Box垂直方向的距离由margin决定，同一个BFC的相邻Box的margin会发生重叠
3. BFC的区域不会与float box重叠
4. BFC是页面上的一个隔离的独立容器
5. 计算BFC的高度时，浮动元素也参与计算

触发BFC的条件：

- 根元素（html）
- float属性不为none
- position为absolute或fixed
- display为inline-block、table-cell、table-caption、flex、inline-flex
- overflow不为visible

BFC的应用：

1. 解决margin重叠：

代码块

```
.bfc {
    overflow: hidden; /* 创建BFC */
}
```

2. 清除浮动：

代码块

```
.clearfix {
    overflow: hidden; /* 包含浮动子元素 */
}
```

3. 防止文字环绕：

代码块

```
.sidebar {
    float: left;
    width: 200px;
}
.content {
    overflow: hidden; /* 创建BFC，不与浮动元素重叠 */
}
```

---

10. Flexbox布局的主要特性和使用方法?

参考答案:

基本概念:

- 主轴（main axis）和交叉轴（cross axis）
- 容器（flex container）和项目（flex items）
- 一维布局系统，主要处理一个方向上的布局

容器属性:

代码块

```
.flex-container {
    display: flex;
    flex-direction: row | column; /* 主轴方向 */
    flex-wrap: nowrap | wrap; /* 是否换行 */
    justify-content: flex-start | center | space-between; /* 主轴对齐 */
    align-items: stretch | center | flex-start; /* 交叉轴对齐 */
    align-content: stretch | center; /* 多行对齐 */
}
```

项目属性:

代码块

```
.flex-item {
    flex-grow: 1; /* 放大比例 */
    flex-shrink: 1; /* 缩小比例 */
    flex-basis: auto; /* 基础大小 */
    flex: 1; /* flex-grow, flex-shrink, flex-basis的简写 */
    align-self: auto | center; /* 单独的对齐方式 */
    order: 0; /* 排列顺序 */
}
```

常用布局模式：

1. 水平垂直居中：

代码块

```
.center {
    display: flex;
    justify-content: center;
    align-items: center;
}
```

2. 等分布局：

代码块

```
.equal {
    display: flex;
}
.equal > div {
    flex: 1;
}
```

3. 固定侧边栏：

代码块

```
.layout {
    display: flex;
}
```

```
.sidebar {
    flex: 0 0 200px;
}
.content {
    flex: 1;
}
```

---

11. CSS中的position属性有哪些值?

参考答案:

1. static（默认值）：

- 正常文档流定位
- top、right、bottom、left属性无效
- 不会创建新的层叠上下文

2. relative（相对定位）：

- 相对于元素在正常文档流中的位置定位
- 不脱离文档流，原位置保留
- 可以使用z-index

代码块

```
.relative {
    position: relative;
    top: 10px;
    left: 20px;
}
```

3. absolute（绝对定位）：

- 相对于最近的已定位祖先元素定位
- 脱离文档流
- 如果没有已定位祖先，则相对于初始包含块定位

代码块

```
.absolute {
    position: absolute;
    top: 0;
    right: 0;
}
```

4. fixed（固定定位）：

- 相对于视口定位
- 脱离文档流
- 滚动时位置不变

代码块

```
.fixed {
    position: fixed;
    bottom: 20px;
    right: 20px;
}
```

5. sticky（粘性定位）：

- 根据滚动位置在relative和fixed之间切换
- 需要指定top、right、bottom、left中的一个

代码块

```
.sticky {
    position: sticky;
    top: 0;
```

```
}
```

---

12. 如何实现元素的水平垂直居中?

参考答案:

1. Flexbox方法:

代码块

```
.center {
    display: flex;
    justify-content: center;
    align-items: center;
}
```

2. Grid方法:

代码块

```
.center {
    display: grid;
    place-items: center;
}
```

3. 绝对定位 + transform:

代码块

```
.center {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
```

```
}
```

4. 绝对定位 + margin：

代码块

```
.center {
    position: absolute;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    margin: auto;
    width: 200px;
    height: 100px;
}
```

5. table-cell方法：

代码块

```
.center {
    display: table-cell;
    text-align: center;
    vertical-align: middle;
}
```

6. line-height方法（单行文本）：

代码块

```
.center {
    height: 100px;
    line-height: 100px;
    text-align: center;
}
```

---

13. CSS盒模型是什么?

参考答案:

盒模型组成:

CSS盒模型由内容（content）、内边距（padding）、边框（border）、外边距（margin）四部分组成。

两种盒模型:

1. 标准盒模型（content-box）:

- 宽度 = 内容宽度
- 总宽度 = width + padding + border + margin

2. IE盒模型（border-box）:

- 宽度 = 内容宽度 + padding + border
- 总宽度 = width + margin

box-sizing属性:

代码块

```
/* 标准盒模型 */
.standard {
    box-sizing: content-box;
}

/* IE盒模型 */
.border-box {
    box-sizing: border-box;
}
```

示例:

代码块

```
.box {
    width: 200px;
    padding: 20px;
    border: 5px solid #000;
    margin: 10px;
}

/* content-box: 总宽度 = 200 + 40 + 10 + 20 = 270px */
/* border-box: 总宽度 = 200 + 20 = 220px */
```

最佳实践：

代码块

```
* {
    box-sizing: border-box;
}
```

---

14. CSS中的float属性及其清除浮动的方法？

参考答案：

float属性：

- left：元素向左浮动
- right：元素向右浮动
- none：默认值，不浮动

浮动的特性：

1. 脱离文档流
2. 向左或向右移动，直到碰到容器边缘或另一个浮动元素

3. 浮动元素会尽可能向上移动
4. 浮动元素不会超出包含块

清除浮动的方法：

1. 使用clear属性：

代码块

```
.clear {
    clear: both; /* left | right | both */
}
```

2. 父元素添加overflow：

代码块

```
.clearfix {
    overflow: hidden; /* 或 auto */
}
```

3. 使用伪元素清除：

代码块

```
.clearfix::after {
    content: "";
    display: table;
    clear: both;
}
```

4. 父元素也浮动：

代码块

```
.parent {
    float: left;
    width: 100%;
```

```
4   }
```

5. 使用display: flow-root：

```
代码块
1   .clearfix {
2       display: flow-root;
3   }
```

现代替代方案：

- 使用Flexbox布局
- 使用Grid布局
- 避免使用float进行布局

---

15. CSS中的z-index属性如何工作？

参考答案：

z-index基本概念：

- 控制元素在z轴（垂直于屏幕）上的堆叠顺序
- 只对定位元素（position不为static）有效
- 数值越大，元素越靠前

层叠上下文（Stacking Context）：

创建层叠上下文的条件：

- 根元素（html）
- position为absolute或relative且z-index不为auto
- position为fixed或sticky

- flex项目且z-index不为auto
- opacity小于1
- transform不为none
- filter不为none

层叠顺序（从底到顶）：

1. 层叠上下文的根
2. z-index为负值的定位元素
3. 非定位的块级元素
4. 非定位的浮动元素
5. 非定位的行内元素
6. z-index为auto的定位元素
7. z-index为正值的定位元素

示例：

代码块

```
.context {
    position: relative;
    z-index: 1;
}

.child1 {
    position: absolute;
    z-index: 100;
}

.child2 {
    position: absolute;
    z-index: 200;
}
```

注意事项：

- 子元素的z-index只在父级层叠上下文内有效
- 不要滥用过大的z-index值

- 建议使用合理的z-index分层策略

---

16. CSS预处理器（Sass/Less）的优势是什么？

参考答案：

主要优势：

1. 变量（Variables）：

代码块

```
// Sass
$primary-color: #3498db;
$font-size: 16px;

.button {
    background-color: $primary-color;
    font-size: $font-size;
}
```

2. 嵌套（Nesting）：

代码块

```
.navbar {
    background: #333;

    ul {
        margin: 0;
        padding: 0;
    }

    li {
        list-style: none;

        a {
```

```
            text-decoration: none;
            color: white;

            &:hover {
                color: #ccc;
            }
        }
    }
}
```

3. 混合（Mixins）：

代码块

```
@mixin border-radius($radius) {
    -webkit-border-radius: $radius;
    -moz-border-radius: $radius;
    border-radius: $radius;
}

.button {
    @include border-radius(5px);
}
```

4. 继承（Inheritance）：

代码块

```
.message {
    border: 1px solid #ccc;
    padding: 10px;
    color: #333;
}

.success {
    @extend .message;
    border-color: green;
}
```

5. 函数和运算：

代码块

```
$base-font-size: 16px;

.title {
    font-size: $base-font-size * 1.5;
    margin-bottom: $base-font-size / 2;
}
```

6. 模块化：

代码块

```
// _variables.scss
$primary-color: #3498db;

// _mixins.scss
@mixin flex-center {
    display: flex;
    justify-content: center;
    align-items: center;
}

// main.scss
@import 'variables';
@import 'mixins';
```

其他优势：

- 更好的代码组织和维护
- 减少代码重复
- 支持条件语句和循环
- 丰富的内置函数
- 更好的团队协作

---

17. 如何实现响应式设计？

参考答案：

1. 媒体查询（Media Queries）：

代码块

```
/* 移动端优先 */
.container {
    width: 100%;
    padding: 10px;
}

/* 平板 */
@media (min-width: 768px) {
    .container {
        width: 750px;
        margin: 0 auto;
    }
}

/* 桌面端 */
@media (min-width: 1200px) {
    .container {
        width: 1170px;
    }
}
```

2. 流式布局：

代码块

```
.container {
    max-width: 1200px;
    width: 100%;
    margin: 0 auto;
}

.column {
    width: 48%;
    float: left;
    margin: 1%;
}
```

3. 弹性图片：

代码块

```
img {
    max-width: 100%;
    height: auto;
}

/* 响应式背景图 */
.hero {
    background-image: url('mobile.jpg');
}

@media (min-width: 768px) {
    .hero {
        background-image: url('desktop.jpg');
    }
}
```

4. 相对单位：

代码块

```
.text {
    font-size: 1rem; /* 相对于根元素 */
    padding: 2em; /* 相对于当前元素 */
    width: 50vw; /* 相对于视口宽度 */
    height: 100vh; /* 相对于视口高度 */
}
```

5. Flexbox和Grid：

代码块

```
.flex-container {
    display: flex;
    flex-wrap: wrap;
}

```

```
.flex-item {
    flex: 1 1 300px; /* 最小宽度300px */
}

.grid-container {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
    gap: 20px;
}
```

6. 视口设置：

```
代码块
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

断点策略：

- 移动端：320px - 767px
- 平板：768px - 1023px
- 桌面：1024px+

---

18. CSS中的伪类和伪元素有什么区别？

参考答案：

伪类（Pseudo-classes）：

用于选择处于特定状态的元素，以单冒号 `:` 表示。

常用伪类：

```
代码块
```

```
/* 链接状态 */
a:link { color: blue; }
a:visited { color: purple; }
a:hover { color: red; }
a:active { color: orange; }

/* 结构伪类 */
li:first-child { font-weight: bold; }
li:last-child { margin-bottom: 0; }
li:nth-child(2n) { background: #f0f0f0; }
li:nth-child(odd) { background: white; }

/* 状态伪类 */
input:focus { border-color: blue; }
input:disabled { opacity: 0.5; }
input:checked + label { color: green; }

/* 否定伪类 */
p:not(.special) { color: gray; }
```

伪元素（Pseudo-elements）：

用于创建和样式化不存在于HTML中的元素，以双冒号 `::` 表示（CSS3规范，但单冒号也兼容）。

常用伪元素：

代码块

```
/* 首字母和首行 */
p::first-letter {
    font-size: 2em;
    float: left;
}

p::first-line {
    font-weight: bold;
}

/* 前后插入内容 */
.quote::before {
    content: """;
    font-size: 2em;
}

```

```
.quote::after {
    content: """;
    font-size: 2em;
}

/* 选中文本 */
::selection {
    background: yellow;
    color: black;
}

/* 占位符 */
input::placeholder {
    color: #999;
    font-style: italic;
}
```

主要区别：

1. **概念**：伪类选择存在的元素的特定状态，伪元素创建虚拟元素
2. **语法**：伪类用单冒号，伪元素用双冒号（CSS3）
3. **数量**：一个元素可以有多个伪类，但只能有一个::before和一个::after
4. **DOM**：伪类不创建新元素，伪元素创建虚拟元素

---

19. 什么是CSS Sprites？有什么优缺点？

参考答案：

CSS Sprites定义：

CSS Sprites是一种网页图片应用处理方式，它将多个小图片合并成一张大图片，然后通过CSS的background-position属性来显示所需的图片部分。

实现方法：

```css
.sprite {
    background-image: url('sprites.png');
    background-repeat: no-repeat;
    display: inline-block;
}

.icon-home {
    width: 32px;
    height: 32px;
    background-position: 0 0;
}

.icon-user {
    width: 32px;
    height: 32px;
    background-position: -32px 0;
}

.icon-settings {
    width: 32px;
    height: 32px;
    background-position: -64px 0;
}
```

优点：

1. **减少HTTP请求**：多个图片合并为一个，减少服务器请求次数
2. **提高加载速度**：减少网络延迟，提升页面性能
3. **减少服务器压力**：降低并发请求数量
4. **缓存友好**：一次加载，多次使用

缺点：

1. **维护困难**：添加或修改图标需要重新生成整个Sprite图
2. **内存占用**：加载整张大图，即使只使用部分图标
3. **不够灵活**：图标大小固定，难以适应响应式设计
4. **开发复杂**：需要精确计算坐标位置

现代替代方案：

1. 字体图标：

代码块

```
@font-face {
    font-family: 'iconfont';
    src: url('iconfont.woff2') format('woff2');
}

.icon {
    font-family: 'iconfont';
}

.icon-home::before {
    content: '\e001';
}
```

2. SVG图标：

代码块

```
<svg class="icon">
    <use xlink:href="#icon-home"></use>
</svg>
```

3. Base64内联：

代码块

```
.icon {
    background-image:
url('
QVR42mP8/5+hHgAHggJ/PchI7wAAAABJRU5ErkJggg==');
}
```

20. CSS中的单位有哪些?

参考答案:

绝对单位:

1. px（像素）:

代码块

```
.box {
    width: 300px;
    height: 200px;
}
```

2. pt（点）:

- 1pt = 1/72英寸
- 主要用于打印样式

3. pc（派卡）:

- 1pc = 12pt

4. in（英寸）、cm（厘米）、mm（毫米）:

代码块

```
@media print {
    .page {
        width: 8.5in;
        height: 11in;
    }
}
```

相对单位:

1. em：

- 相对于当前元素的字体大小

```
代码块
1   .parent {
2       font-size: 16px;
3   }
4   .child {
5       font-size: 1.5em; /* 24px */
6       padding: 1em; /* 24px */
7   }
```

2. rem：

- 相对于根元素的字体大小

```
代码块
1   html {
2       font-size: 16px;
3   }
4   .text {
5       font-size: 1.2rem; /* 19.2px */
6       margin: 2rem; /* 32px */
7   }
```

3. %（百分比）：

```
代码块
1   .container {
2       width: 80%; /* 相对于父元素宽度 */
3       font-size: 120%; /* 相对于父元素字体大小 */
4   }
```

视口单位：

1. vw（视口宽度）：

- 1vw = 视口宽度的1%

```
代码块
1   .full-width {
2       width: 100vw;
3   }
```

2. vh（视口高度）：

- 1vh = 视口高度的1%

```
代码块
1   .full-height {
2       height: 100vh;
3   }
```

3. vmin和vmax：

```
代码块
1   .square {
2       width: 50vmin; /* 视口较小尺寸的50% */
3       height: 50vmin;
4   }
```

新单位（CSS3+）：

1. ch：

- 相对于字符"0"的宽度

```
代码块
1   .monospace {
2       width: 40ch; /* 大约40个字符宽度 */
3   }
```

2. ex：

- 相对于字母"x"的高度

使用建议：

- 字体大小：rem
- 间距：rem或em
- 边框：px
- 布局宽度：%或vw
- 响应式设计：相对单位优先

---

21. 如何实现CSS三角形?

参考答案：

基本原理：

利用border属性，将元素的宽高设为0，通过设置不同方向的border来形成三角形。

1. 向上的三角形：

代码块

```
.triangle-up {
    width: 0;
    height: 0;
    border-left: 25px solid transparent;
    border-right: 25px solid transparent;
    border-bottom: 40px solid #333;
}
```

2. 向下的三角形：

代码块

```
.triangle-down {
    width: 0;
    height: 0;
    border-left: 25px solid transparent;
    border-right: 25px solid transparent;
    border-top: 40px solid #333;
}
```

3. 向左的三角形：

代码块

```
.triangle-left {
    width: 0;
    height: 0;
    border-top: 25px solid transparent;
    border-bottom: 25px solid transparent;
    border-right: 40px solid #333;
}
```

4. 向右的三角形：

代码块

```
.triangle-right {
    width: 0;
    height: 0;
    border-top: 25px solid transparent;
    border-bottom: 25px solid transparent;
    border-left: 40px solid #333;
}
```

其他形状：

5. 等腰直角三角形：

代码块

```
.triangle-right-angle {
    width: 0;
    height: 0;
    border-top: 50px solid #333;
    border-right: 50px solid transparent;
}
```

6. 聊天气泡：

代码块

```
.chat-bubble {
    position: relative;
    background: #333;
    padding: 10px 15px;
    border-radius: 10px;
    color: white;
}

.chat-bubble::after {
    content: '';
    position: absolute;
    top: 100%;
    left: 20px;
    width: 0;
    height: 0;
    border-left: 10px solid transparent;
    border-right: 10px solid transparent;
    border-top: 10px solid #333;
}
```

现代替代方案：

1. 使用clip-path：

代码块

```
.triangle-clip {
    width: 100px;
    height: 100px;
    background: #333;
    clip-path: polygon(50% 0%, 0% 100%, 100% 100%);
```

```
}
```

2. 使用transform：

代码块

```
.triangle-transform {
    width: 50px;
    height: 50px;
    background: #333;
    transform: rotate(45deg);
}
```

3. 使用SVG：

代码块

```
<svg width="100" height="100">
    <polygon points="50,0 0,100 100,100" fill="#333" />
</svg>
```

---

22. CSS中的层叠规则是什么？

参考答案：

层叠（Cascade）的含义：

CSS的"层叠"指的是当多个规则应用到同一个元素时，如何确定最终应用哪个规则的机制。

层叠顺序（按优先级从低到高）：

1. 浏览器默认样式：

```
/* 浏览器默认样式 */
p { margin: 1em 0; }
```

2. 用户样式表：

- 用户在浏览器中设置的样式

3. 作者样式表（网站开发者编写的样式）：

```
/* 外部样式表 */
@import url('style.css');

/* 内部样式表 */
<style>
p { color: blue; }
</style>

/* 内联样式 */
<p style="color: red;">
```

4. 作者!important声明：

```
p {
    color: green !important;
}
```

5. 用户!important声明：

- 用户设置的!important样式

6. 浏览器!important声明：

- 浏览器默认的!important样式

特殊性（Specificity）计算：

计算规则：

- 内联样式：1000
- ID选择器：100
- 类、属性、伪类选择器：10
- 元素、伪元素选择器：1

示例：

代码块

```
/* 特殊性: 0001 */
p { color: black; }

/* 特殊性: 0010 */
.text { color: blue; }

/* 特殊性: 0100 */
#title { color: red; }

/* 特殊性: 0111 */
#title.text p { color: green; }

/* 特殊性: 1000 */
<p style="color: yellow;">

/* 最高优先级 */
p { color: purple !important; }
```

层叠解决冲突的步骤：

1. 找出所有相关规则
2. 按来源和重要性排序
3. 按特殊性排序
4. 按源码顺序排序（后来居上）

最佳实践：

- 避免使用!important
- 保持选择器简洁
- 使用有意义的类名
- 遵循CSS架构方法（BEM、OOCSS等）

---

23. 什么是CSS-in-JS？有什么优缺点？

参考答案：

CSS-in-JS定义：

CSS-in-JS是一种将CSS样式直接写在JavaScript代码中的技术，通常与React等组件化框架一起使用。

常见的CSS-in-JS库：

1. Styled-components：

代码块

```
import styled from 'styled-components';

const Button = styled.button`
    background: ${props => props.primary ? 'blue' : 'white'};
    color: ${props => props.primary ? 'white' : 'blue'};
    font-size: 1em;
    margin: 1em;
    padding: 0.25em 1em;
    border: 2px solid blue;
    border-radius: 3px;
    cursor: pointer;

    &:hover {
        background: ${props => props.primary ? 'darkblue' : 'lightblue'};
    }
`;
```

```
17
18  // 使用
19  <Button primary>Primary Button</Button>
```

2. Emotion：

代码块

```
1   import { css } from '@emotion/react';
2
3   const buttonStyle = css`
4       background: hotpink;
5       &:hover {
6           background: pink;
7       }
8   `;
9
10  <button css={buttonStyle}>Click me</button>
```

3. JSS：

代码块

```
1   import { createUseStyles } from 'react-jss';
2
3   const useStyles = createUseStyles({
4       button: {
5           background: 'blue',
6           color: 'white',
7           '&:hover': {
8               background: 'darkblue'
9           }
10      }
11  });
12
13  function Button() {
14      const classes = useStyles();
15      return <button className={classes.button}>Click me</button>;
16  }
```

优点：

1. 组件化：

- 样式与组件紧密绑定
- 更好的封装性和可维护性

2. 动态样式：

```
代码块
1  const Button = styled.button`
2      background: ${props => props.theme.primary};
3      opacity: ${props => props.disabled ? 0.5 : 1};
4  `;
```

3. 自动前缀：

- 自动添加浏览器前缀
- 处理兼容性问题

4. 死代码消除：

- 未使用的样式会被自动移除
- 减少最终打包大小

5. 主题支持：

```
代码块
1  const theme = {
2      primary: 'blue',
3      secondary: 'green'
4  };
5
6  <ThemeProvider theme={theme}>
7      <App />
8  </ThemeProvider>
```

缺点：

1. 学习成本：

- 需要学习新的API和语法
- 与传统CSS开发方式不同

2. 运行时开销：

- 样式在运行时生成
- 可能影响性能

3. 调试困难：

- 生成的类名不直观
- 调试工具支持有限

4. 服务端渲染复杂：

- SSR配置相对复杂
- 需要额外的设置

5. 工具链依赖：

- 依赖JavaScript构建工具
- 增加了项目复杂度

适用场景：

- React/Vue等组件化项目
- 需要动态样式的应用
- 大型团队协作项目
- 需要严格样式隔离的场景

---

24. 如何实现CSS动画的性能优化?

参考答案:

1. 使用transform和opacity:

这两个属性不会触发重排（reflow），只会触发重绘（repaint）或合成（composite）。

代码块

```
/* 好的做法 */
.animate {
    transform: translateX(100px);
    opacity: 0.5;
    transition: transform 0.3s, opacity 0.3s;
}

/* 避免的做法 */
.animate {
    left: 100px; /* 会触发重排 */
    width: 200px; /* 会触发重排 */
}
```

2. 启用硬件加速:

代码块

```
.accelerated {
    transform: translateZ(0); /* 或 translate3d(0,0,0) */
    /* 或者 */
    will-change: transform;
}
```

3. 使用will-change属性:

代码块

```
.element {
    will-change: transform, opacity;
```

```
}

/* 动画结束后移除 */
.element.animation-finished {
    will-change: auto;
}
```

4. 避免动画期间的重排属性：

代码块

```
/* 会触发重排的属性（避免动画） */
.bad {
    animation: badAnimation 1s;
}

@keyframes badAnimation {
    from { width: 100px; height: 100px; }
    to { width: 200px; height: 200px; }
}

/* 好的替代方案 */
.good {
    animation: goodAnimation 1s;
}

@keyframes goodAnimation {
    from { transform: scale(1); }
    to { transform: scale(2); }
}
```

5. 使用CSS动画而非JavaScript：

代码块

```
/* CSS动画 - 更好的性能 */
.css-animation {
    animation: slideIn 0.3s ease-out;
}

@keyframes slideIn {
    from { transform: translateX(-100%); }
    to { transform: translateX(0); }
```

```
}
```

6. 合理使用动画时长和缓动函数：

代码块

```
.smooth {
    transition: transform 0.3s cubic-bezier(0.4, 0, 0.2, 1);
}

/* 避免过长的动画时间 */
.too-slow {
    transition: transform 2s; /* 用户可能感到不耐烦 */
}
```

7. 减少同时运行的动画数量：

代码块

```
// 使用requestAnimationFrame控制动画
function animate() {
    // 批量更新DOM
    elements.forEach(el => {
        el.style.transform = `translateX(${getNewPosition()}px)`;
    });

    requestAnimationFrame(animate);
}
```

8. 使用transform3d强制开启GPU加速：

代码块

```
.gpu-accelerated {
    transform: translate3d(0, 0, 0);
    /* 或者 */
    transform: translateZ(0);
    /* 或者 */
    backface-visibility: hidden;
}
```

9. 避免在动画中使用box-shadow：

代码块

```
/* 性能较差 */
.shadow-animation {
    transition: box-shadow 0.3s;
}
.shadow-animation:hover {
    box-shadow: 0 10px 20px rgba(0,0,0,0.3);
}

/* 更好的替代方案 */
.pseudo-shadow {
    position: relative;
}
.pseudo-shadow::after {
    content: '';
    position: absolute;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    box-shadow: 0 10px 20px rgba(0,0,0,0.3);
    opacity: 0;
    transition: opacity 0.3s;
}
.pseudo-shadow:hover::after {
    opacity: 1;
}
```

10. 性能监控和调试：

代码块

```
// 使用Performance API监控
performance.mark('animation-start');
// 动画代码
performance.mark('animation-end');
performance.measure('animation-duration', 'animation-start', 'animation-end');
```

最佳实践总结：

- 优先使用transform和opacity
- 合理使用will-change
- 避免在动画中修改布局属性
- 使用CSS动画替代JavaScript动画
- 监控和测试动画性能

---

25. CSS中的@media查询有哪些常用的特性？

参考答案：

基本语法：

```
代码块
1   @media media-type and (media-feature) {
2       /* CSS规则 */
3   }
```

媒体类型（Media Types）：

```
代码块
1   @media screen { /* 屏幕设备 */ }
2   @media print { /* 打印设备 */ }
3   @media speech { /* 语音合成器 */ }
4   @media all { /* 所有设备（默认） */ }
```

常用媒体特性：

1. 宽度和高度：

代码块

```
/* 视口宽度 */
@media (max-width: 768px) {
    .container { width: 100%; }
}

@media (min-width: 1200px) {
    .container { width: 1170px; }
}

/* 设备宽度 */
@media (max-device-width: 480px) {
    body { font-size: 14px; }
}

/* 高度 */
@media (max-height: 600px) {
    .header { height: 40px; }
}
```

2. 方向：

代码块

```
/* 横屏 */
@media (orientation: landscape) {
    .sidebar { width: 300px; }
}

/* 竖屏 */
@media (orientation: portrait) {
    .sidebar { width: 100%; }
}
```

3. 分辨率：

代码块

```
/* 高分辨率屏幕 */
@media (min-resolution: 2dppx) {
```

```
    .logo { background-image: url('logo@2x.png'); }
}

/* Retina屏幕 */
@media (-webkit-min-device-pixel-ratio: 2) {
    .icon { background-size: 50% 50%; }
}
```

4. 颜色：

代码块

```
/* 彩色屏幕 */
@media (color) {
    .colorful { color: red; }
}

/* 黑白屏幕 */
@media (monochrome) {
    .image { filter: grayscale(100%); }
}
```

5. 指针设备：

代码块

```
/* 触摸设备 */
@media (pointer: coarse) {
    .button { min-height: 44px; }
}

/* 鼠标等精确指针 */
@media (pointer: fine) {
    .button { min-height: 32px; }
}

/* 悬停支持 */
@media (hover: hover) {
    .button:hover { background: #ccc; }
}
```

6. 暗色模式：

代码块

```
/* 暗色主题 */
@media (prefers-color-scheme: dark) {
    body {
        background: #333;
        color: white;
    }
}

/* 亮色主题 */
@media (prefers-color-scheme: light) {
    body {
        background: white;
        color: black;
    }
}
```

7. 动画偏好：

代码块

```
/* 用户偏好减少动画 */
@media (prefers-reduced-motion: reduce) {
    * {
        animation-duration: 0.01ms !important;
        animation-iteration-count: 1 !important;
        transition-duration: 0.01ms !important;
    }
}
```

逻辑操作符：

1. and操作符：

代码块

```
@media screen and (min-width: 768px) and (max-width: 1023px) {
```

```
    .tablet-only { display: block; }
}
```

2. or操作符（逗号）：

```
代码块
@media (max-width: 768px), (orientation: portrait) {
    .mobile-or-portrait { width: 100%; }
}
```

3. not操作符：

```
代码块
@media not screen {
    .no-screen { display: none; }
}
```

4. only操作符：

```
代码块
@media only screen and (max-width: 768px) {
    .mobile-only { display: block; }
}
```

常用断点：

```
代码块
/* 移动端 */
@media (max-width: 767px) { }

/* 平板 */
@media (min-width: 768px) and (max-width: 1023px) { }

/* 桌面端 */
```

```
@media (min-width: 1024px) { }

/* 大屏幕 */
@media (min-width: 1200px) { }
```

最佳实践：

- 移动端优先设计
- 使用相对单位
- 测试各种设备和屏幕尺寸
- 考虑用户偏好设置
- 合理组织媒体查询代码

---

26. 如何实现CSS的垂直居中？

参考答案：

1. Flexbox方法（推荐）：

代码块

```
.container {
    display: flex;
    align-items: center; /* 垂直居中 */
    justify-content: center; /* 水平居中 */
    height: 100vh;
}
```

2. Grid方法：

代码块

```
.container {
    display: grid;
```

```
    place-items: center;
    height: 100vh;
}

/* 或者 */
.container {
    display: grid;
    align-items: center;
    justify-items: center;
    height: 100vh;
}
```

3. 绝对定位 + transform：

代码块

```
.container {
    position: relative;
    height: 100vh;
}

.centered {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
}
```

4. 绝对定位 + margin auto：

代码块

```
.container {
    position: relative;
    height: 100vh;
}

.centered {
    position: absolute;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
```

```
    margin: auto;
    width: 200px; /* 需要固定宽度 */
    height: 100px; /* 需要固定高度 */
}
```

5. table-cell方法：

代码块

```
.container {
    display: table-cell;
    vertical-align: middle;
    text-align: center;
    width: 100vw;
    height: 100vh;
}
```

6. line-height方法（单行文本）：

代码块

```
.container {
    height: 100px;
    line-height: 100px;
    text-align: center;
}

.centered {
    display: inline-block;
    vertical-align: middle;
    line-height: normal;
}
```

7. 伪元素方法：

代码块

```
.container {
    text-align: center;
    height: 100vh;
```

```
}

.container::before {
    content: '';
    display: inline-block;
    height: 100%;
    vertical-align: middle;
}

.centered {
    display: inline-block;
    vertical-align: middle;
}
```

8. CSS Grid（单个子元素）：

代码块

```
.container {
    display: grid;
    height: 100vh;
}

.centered {
    margin: auto;
}
```

9. 现代方法 - place-content：

代码块

```
.container {
    display: grid;
    place-content: center;
    height: 100vh;
}
```

10. 多行文本垂直居中：

代码块

```
.container {
    display: table;
    height: 200px;
    width: 100%;
}

.centered {
    display: table-cell;
    vertical-align: middle;
    text-align: center;
}
```

选择建议：

- **现代浏览器**：优先使用Flexbox或Grid
- **需要兼容老浏览器**：使用绝对定位 + transform
- **单行文本**：使用line-height
- **已知尺寸**：使用绝对定位 + margin auto
- **表格布局**：使用table-cell

兼容性考虑：

代码块

```
/* 兼容性方案 */
.container {
    /* 老浏览器回退 */
    display: table-cell;
    vertical-align: middle;
    text-align: center;

    /* 现代浏览器 */
    display: flex;
    align-items: center;
    justify-content: center;
}
```

---

27. CSS中的calc()函数有什么用途?

参考答案:

calc()函数定义:

calc()允许在CSS中进行数学计算，可以混合使用不同的单位进行运算。

基本语法:

代码块

```
.element {
    width: calc(expression);
}
```

支持的运算符:

- `+` (加法)
- `-` (减法)
- `*` (乘法)
- `/` (除法)

常用场景:

1. 混合单位计算:

代码块

```
.container {
    width: calc(100% - 200px); /* 百分比减去固定像素 */
    height: calc(100vh - 60px); /* 视口高度减去头部高度 */
    margin: calc(1rem + 5px); /* rem加上像素 */
}
```

2. 响应式布局:

代码块

```
.sidebar {
    width: 300px;
    float: left;
}

.content {
    width: calc(100% - 300px); /* 剩余宽度 */
    float: right;
}

/* 三栏布局 */
.left { width: 200px; }
.right { width: 150px; }
.center { width: calc(100% - 350px); }
```

3. 网格布局计算：

代码块

```
.grid-item {
    width: calc(33.333% - 20px); /* 三列布局，减去间距 */
    margin-right: 20px;
}

/* 考虑边距的等分布局 */
.four-columns {
    width: calc((100% - 60px) / 4); /* 四列，总间距60px */
}
```

4. 垂直居中计算：

代码块

```
.centered {
    position: absolute;
    top: calc(50% - 100px); /* 50%减去元素高度的一半 */
    left: calc(50% - 150px); /* 50%减去元素宽度的一半 */
    width: 300px;
    height: 200px;
}
```

5. 字体大小响应式：

代码块

```
.responsive-text {
    font-size: calc(16px + 1vw); /* 基础16px加上视口宽度的1% */
}

.title {
    font-size: calc(1.5rem + 2vw); /* 响应式标题 */
}
```

6. 动态间距：

代码块

```
.section {
    padding: calc(2rem + 5vh) calc(1rem + 2vw);
}

.card {
    margin-bottom: calc(1em + 1vh);
}
```

7. 表单布局：

代码块

```
.form-group {
    width: calc(50% - 10px); /* 两列表单，考虑间距 */
    display: inline-block;
    margin-right: 20px;
}

.input-with-button {
    width: calc(100% - 120px); /* 输入框宽度，减去按钮宽度 */
}

.button {
    width: 100px;
}
```

8. 复杂计算：

代码块

```
.complex {
    width: calc((100% - 40px) / 3 - 20px);
    /* 三等分，减去容器左右边距40px，再减去元素间距20px */

    height: calc(100vh - 80px - 2em);
    /* 视口高度减去头部80px再减去2em的底部间距 */
}
```

注意事项：

1. 运算符两边必须有空格：

代码块

```
/* 正确 */
width: calc(100% - 20px);

/* 错误 */
width: calc(100%-20px);
```

2. 除法运算的除数不能为0：

代码块

```
/* 错误 */
width: calc(100px / 0);
```

3. 嵌套使用：

代码块

```
.nested {
    width: calc(calc(100% / 3) - 20px);
```

```
    /* 可以嵌套，但建议简化 */

    /* 更好的写法 */
    width: calc(100% / 3 - 20px);
}
```

4. 与CSS变量结合：

代码块

```
:root {
    --sidebar-width: 250px;
    --header-height: 60px;
}

.content {
    width: calc(100% - var(--sidebar-width));
    height: calc(100vh - var(--header-height));
}
```

浏览器兼容性：

- IE 9+支持
- 现代浏览器完全支持
- 移动端浏览器支持良好

---

28. 什么是CSS变量（自定义属性）？

参考答案：

CSS变量定义：

CSS变量（CSS Custom Properties）允许开发者定义可重复使用的值，并在整个文档中引用这些值。

基本语法：

```
代码块
1  /* 定义变量 */
2  :root {
3      --primary-color: #3498db;
4      --font-size: 16px;
5      --margin: 20px;
6  }
7
8  /* 使用变量 */
9  .button {
10     background-color: var(--primary-color);
11     font-size: var(--font-size);
12     margin: var(--margin);
13 }
```

变量作用域：

1. 全局变量：

```
代码块
1  :root {
2      --global-color: #333;
3      --global-font: 'Arial, sans-serif';
4  }
5
6  /* 在任何地方都可以使用 */
7  body {
8      color: var(--global-color);
9      font-family: var(--global-font);
10 }
```

2. 局部变量：

```
代码块
1  .component {
2      --local-bg: #f0f0f0;
3      --local-padding: 15px;
```

```
    background: var(--local-bg);
    padding: var(--local-padding);
}

/* 子元素可以继承父元素的变量 */
.component .child {
    background: var(--local-bg); /* 可以使用 */
}
```

3. 变量继承和覆盖：

代码块

```
:root {
    --color: blue;
}

.parent {
    --color: red; /* 覆盖全局变量 */
}

.child {
    color: var(--color); /* 使用父元素的red */
}
```

高级用法：

1. 回退值：

代码块

```
.element {
    color: var(--undefined-color, #000); /* 如果变量不存在，使用黑色 */
    font-size: var(--font-size, 16px);
}
```

2. 变量嵌套：

代码块

```
:root {
    --base-size: 16px;
    --large-size: calc(var(--base-size) * 1.5);
    --primary: #3498db;
    --primary-dark: color-mix(in srgb, var(--primary) 80%, black);
}
```

3. 动态主题切换：

代码块

```
:root {
    --bg-color: white;
    --text-color: black;
    --border-color: #ccc;
}

[data-theme="dark"] {
    --bg-color: #333;
    --text-color: white;
    --border-color: #555;
}

body {
    background: var(--bg-color);
    color: var(--text-color);
    border-color: var(--border-color);
}
```

4. 响应式变量：

代码块

```
:root {
    --container-width: 1200px;
    --grid-columns: 4;
    --gap: 20px;
}

@media (max-width: 768px) {
    :root {
        --container-width: 100%;
        --grid-columns: 2;
```

```
        --gap: 10px;
    }
}

.container {
    max-width: var(--container-width);
}

.grid {
    grid-template-columns: repeat(var(--grid-columns), 1fr);
    gap: var(--gap);
}
```

JavaScript交互：

1. 读取CSS变量：

代码块

```
// 获取CSS变量值
const primaryColor = getComputedStyle(document.documentElement)
    .getPropertyValue('--primary-color');

console.log(primaryColor); // #3498db
```

2. 设置CSS变量：

代码块

```
// 设置CSS变量
document.documentElement.style.setProperty('--primary-color', '#e74c3c');

// 或者在特定元素上设置
element.style.setProperty('--local-var', 'value');
```

3. 动态主题切换：

代码块

```
function toggleTheme() {
    const isDark = document.documentElement.getAttribute('data-theme') === 'dark';
    document.documentElement.setAttribute('data-theme', isDark ? 'light' : 'dark');
}
```

实际应用场景：

1. 设计系统：

代码块

```
:root {
    /* 颜色系统 */
    --primary-50: #e3f2fd;
    --primary-100: #bbdefb;
    --primary-500: #2196f3;
    --primary-900: #0d47a1;

    /* 间距系统 */
    --space-xs: 4px;
    --space-sm: 8px;
    --space-md: 16px;
    --space-lg: 24px;
    --space-xl: 32px;

    /* 字体系统 */
    --font-size-sm: 0.875rem;
    --font-size-base: 1rem;
    --font-size-lg: 1.125rem;
    --font-size-xl: 1.25rem;
}
```

2. 组件库：

代码块

```
.button {
    --button-bg: var(--primary-500);
    --button-color: white;
    --button-padding: var(--space-sm) var(--space-md);
```

```
    --button-border-radius: 4px;

    background: var(--button-bg);
    color: var(--button-color);
    padding: var(--button-padding);
    border-radius: var(--button-border-radius);
}

.button--secondary {
    --button-bg: transparent;
    --button-color: var(--primary-500);
}
```

浏览器兼容性：

- IE不支持
- 现代浏览器完全支持
- 可以使用PostCSS插件提供兼容性

优势：

- 原生CSS支持，无需预处理器
- 可以通过JavaScript动态修改
- 支持继承和级联
- 更好的性能表现
- 便于主题切换和维护

---

29. CSS中的object-fit属性有什么作用？

参考答案：

object-fit定义：

object-fit属性用于指定可替换元素（如img、video）的内容应该如何适应其容器的高度和宽度。

语法：

```
代码块
1   .element {
2       object-fit: fill | contain | cover | none | scale-down;
3   }
```

各个值的含义：

1. fill（默认值）：

```
代码块
1   .image-fill {
2       width: 300px;
3       height: 200px;
4       object-fit: fill;
5   }
6   /* 内容拉伸填满整个容器，可能导致变形 */
```

2. contain：

```
代码块
1   .image-contain {
2       width: 300px;
3       height: 200px;
4       object-fit: contain;
5   }
6   /* 保持宽高比，完整显示内容，可能有空白区域 */
```

3. cover：

```
代码块
1   .image-cover {
2       width: 300px;
```

```
    height: 200px;
    object-fit: cover;
}
/* 保持宽高比，填满容器，可能裁剪部分内容 */
```

4. none：

代码块

```
.image-none {
    width: 300px;
    height: 200px;
    object-fit: none;
}
/* 保持原始尺寸，可能溢出或显示不完整 */
```

5. scale-down：

代码块

```
.image-scale-down {
    width: 300px;
    height: 200px;
    object-fit: scale-down;
}
/* 相当于none或contain中较小的那个 */
```

配合object-position使用：

object-position属性：

代码块

```
.image {
    width: 300px;
    height: 200px;
    object-fit: cover;
    object-position: center top; /* 定位到顶部中心 */
}
```

```
/* 使用百分比 */
.image-position {
    object-fit: cover;
    object-position: 25% 75%; /* 从左25%，从顶75% */
}

/* 使用像素值 */
.image-pixel {
    object-fit: none;
    object-position: -50px 20px; /* 向左偏移50px，向下偏移20px */
}
```

实际应用场景：

1. 响应式图片画廊：

代码块

```
.gallery {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
    gap: 10px;
}

.gallery img {
    width: 100%;
    height: 200px;
    object-fit: cover;
    object-position: center;
    border-radius: 8px;
}
```

2. 用户头像：

代码块

```
.avatar {
    width: 60px;
    height: 60px;
    border-radius: 50%;
    object-fit: cover;
```

```
    object-position: center;
}
```

3. 卡片图片：

代码块

```
.card {
    width: 300px;
    border-radius: 12px;
    overflow: hidden;
}

.card-image {
    width: 100%;
    height: 200px;
    object-fit: cover;
    object-position: center;
}
```

4. 视频封面：

代码块

```
.video-container {
    position: relative;
    width: 100%;
    height: 400px;
}

.video-background {
    width: 100%;
    height: 100%;
    object-fit: cover;
    object-position: center;
}
```

5. 产品展示：

代码块

```
.product-image {
    width: 250px;
    height: 250px;
    object-fit: contain; /* 保证产品完整显示 */
    object-position: center;
    background: #f5f5f5; /* 空白区域背景色 */
}
```

与background-size的对比：

background-size方式：

代码块

```
.bg-image {
    width: 300px;
    height: 200px;
    background-image: url('image.jpg');
    background-size: cover;
    background-position: center;
    background-repeat: no-repeat;
}
```

object-fit方式：

代码块

```
.obj-image {
    width: 300px;
    height: 200px;
    object-fit: cover;
    object-position: center;
}
```

兼容性处理：

CSS回退方案：

代码块

```
.image {
    width: 300px;
    height: 200px;

    /* 不支持object-fit的浏览器回退 */
    background-size: cover;
    background-position: center;
    background-repeat: no-repeat;

    /* 支持object-fit的浏览器 */
    object-fit: cover;
    object-position: center;
}

/* 隐藏不支持object-fit时的img */
@supports not (object-fit: cover) {
    .image {
        background-image: attr(src url);
        font-size: 0; /* 隐藏alt文本 */
    }
}
```

JavaScript检测：

代码块

```
// 检测是否支持object-fit
function supportsObjectFit() {
    return 'objectFit' in document.documentElement.style;
}

if (!supportsObjectFit()) {
    // 使用polyfill或回退方案
    document.querySelectorAll('img[data-object-fit]').forEach(img => {
        const container = img.parentNode;
        container.style.backgroundImage = `url(${img.src})`;
        container.style.backgroundSize = img.dataset.objectFit;
        img.style.opacity = 0;
    });
}
```

浏览器兼容性：

- IE不支持
- Chrome 32+
- Firefox 36+
- Safari 10+
- 移动端支持良好

---

30. CSS中的clip-path属性如何使用?

参考答案:

clip-path定义:

clip-path属性用于创建一个剪切路径，只有路径内的部分会被显示，路径外的部分会被隐藏。

基本语法:

```
代码块
1   .element {
2       clip-path: <clip-source> | <basic-shape> | <geometry-box> | none;
3   }
```

基本形状（basic-shape）:

1. circle()圆形:

```
代码块
1   .circle {
2       clip-path: circle(50px at center);
3       /* circle(半径 at 圆心位置) */
4   }
5
6   .circle-percentage {
```

```
    clip-path: circle(50% at 50% 50%);
}

.circle-offset {
    clip-path: circle(60px at 30% 70%);
}
```

2. ellipse()椭圆：

代码块

```
.ellipse {
    clip-path: ellipse(100px 50px at center);
    /* ellipse(水平半径 垂直半径 at 中心位置) */
}

.ellipse-percentage {
    clip-path: ellipse(50% 25% at 50% 50%);
}
```

3. polygon()多边形：

代码块

```
/* 三角形 */
.triangle {
    clip-path: polygon(50% 0%, 0% 100%, 100% 100%);
}

/* 梯形 */
.trapezoid {
    clip-path: polygon(20% 0%, 80% 0%, 100% 100%, 0% 100%);
}

/* 六边形 */
.hexagon {
    clip-path: polygon(50% 0%, 100% 25%, 100% 75%, 50% 100%, 0% 75%, 0% 25%);
}

/* 星形 */
.star {
    clip-path: polygon(50% 0%, 61% 35%, 98% 35%, 68% 57%, 79% 91%, 50% 70%,
21% 91%, 32% 57%, 2% 35%, 39% 35%);
```

```
}
```

4. inset()矩形：

代码块

```
.inset-basic {
    clip-path: inset(20px);
    /* 四边都内缩20px */
}

.inset-detailed {
    clip-path: inset(10px 20px 30px 40px);
    /* 上 右 下 左 */
}

.inset-rounded {
    clip-path: inset(20px round 10px);
    /* 内缩20px，圆角10px */
}
```

实际应用场景：

1. 图片裁剪效果：

代码块

```
.image-clip {
    width: 300px;
    height: 200px;
    clip-path: polygon(0 0, 100% 0, 85% 100%, 0 100%);
    transition: clip-path 0.3s ease;
}

.image-clip:hover {
    clip-path: polygon(0 0, 100% 0, 100% 100%, 0 100%);
}
```

2. 按钮特殊形状：

代码块

```
.arrow-button {
    background: #3498db;
    color: white;
    padding: 10px 20px;
    clip-path: polygon(0 0, calc(100% - 20px) 0, 100% 50%, calc(100% - 20px) 100%, 0 100%);
    border: none;
    cursor: pointer;
}
```

3. 卡片切角效果：

代码块

```
.card {
    background: white;
    padding: 20px;
    clip-path: polygon(0 0, calc(100% - 20px) 0, 100% 20px, 100% 100%, 0 100%);
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}
```

4. 加载动画：

代码块

```
.loading-circle {
    width: 50px;
    height: 50px;
    background: #3498db;
    clip-path: circle(25px at center);
    animation: pulse 1.5s infinite;
}

@keyframes pulse {
    0% { clip-path: circle(0px at center); }
    50% { clip-path: circle(25px at center); }
    100% { clip-path: circle(0px at center); }
}
```

5. 文字遮罩效果：

代码块

```
.text-reveal {
    font-size: 4rem;
    font-weight: bold;
    background: linear-gradient(45deg, #ff6b6b, #4ecdc4);
    -webkit-background-clip: text;
    -webkit-text-fill-color: transparent;
    clip-path: inset(0 100% 0 0);
    animation: reveal 2s ease-in-out forwards;
}

@keyframes reveal {
    to { clip-path: inset(0 0 0 0); }
}
```

动画效果：

1. 形状变换动画：

代码块

```
.morph {
    width: 200px;
    height: 200px;
    background: #e74c3c;
    clip-path: circle(50% at center);
    transition: clip-path 0.5s ease;
}

.morph:hover {
    clip-path: polygon(50% 0%, 100% 50%, 50% 100%, 0% 50%);
}
```

2. 进场动画：

代码块

```
.slide-in {
    clip-path: inset(0 100% 0 0);
```

```
    animation: slideIn 1s ease-out forwards;
}

@keyframes slideIn {
    to { clip-path: inset(0 0 0 0); }
}
```

与SVG结合使用：

1. 使用SVG路径：

代码块

```
<svg width="0" height="0">
    <defs>
        <clipPath id="myClip">
            <path d="M50,0 L100,50 L50,100 L0,50 Z"/>
        </clipPath>
    </defs>
</svg>
```

代码块

```
.svg-clip {
    clip-path: url(#myClip);
}
```

工具和资源：

1. 在线生成工具：

- Clippy (https://bennettfeely.com/clippy/)
- CSS clip-path maker

2. 调试技巧：

```
.debug {
    /* 添加边框查看裁剪效果 */
    border: 2px solid red;

    /* 或者添加背景色 */
    background: rgba(255, 0, 0, 0.2);
}
```

浏览器兼容性：

- Chrome 55+
- Firefox 54+
- Safari 9.1+
- IE不支持
- 需要-webkit-前缀的旧版本浏览器

性能注意事项：

- clip-path会创建新的层叠上下文
- 复杂路径可能影响性能
- 避免在动画中使用过于复杂的路径
- 可以使用will-change优化动画性能

---

31. 什么是CSS的层叠上下文（Stacking Context）？

参考答案：

层叠上下文定义：

层叠上下文是HTML元素的三维概念，这些HTML元素在一条假想的相对于面向视窗或网页的用户的z轴上延伸，HTML元素依据其自身属性按照优先级顺序占用层叠上下文的空间。

创建层叠上下文的条件：

1. 根元素（html）：

```
代码块
1   <html> <!-- 根层叠上下文 -->
```

2. position + z-index：

```
代码块
1   .context {
2       position: relative; /* 或 absolute, fixed */
3       z-index: 1; /* z-index不为auto */
4   }
```

3. flex/grid项目 + z-index：

```
代码块
1   .flex-container {
2       display: flex;
3   }
4
5   .flex-item {
6       z-index: 1; /* flex项目且z-index不为auto */
7   }
```

4. opacity小于1：

```
代码块
1   .transparent {
2       opacity: 0.9; /* 创建层叠上下文 */
3   }
```

5. transform不为none：

代码块

```
.transformed {
    transform: translateZ(0); /* 或任何transform值 */
}
```

6. filter不为none：

代码块

```
.filtered {
    filter: blur(5px); /* 或任何filter值 */
}
```

7. mix-blend-mode不为normal：

代码块

```
.blended {
    mix-blend-mode: multiply;
}
```

8. isolation为isolate：

代码块

```
.isolated {
    isolation: isolate;
}
```

9. will-change指定任何会创建层叠上下文的属性：

代码块

```
.will-change {
    will-change: transform, opacity;
}
```

10. contain为layout、paint或包含它们的复合值：

```
代码块
1   .contained {
2       contain: layout;
3       /* 或 contain: paint; */
4       /* 或 contain: layout paint; */
5   }
```

层叠顺序（从底到顶）：

```
代码块
1   /* 在同一个层叠上下文中的层叠顺序 */
2   .stacking-order {
3       /* 1. 层叠上下文的根元素 */
4       /* 2. z-index为负值的定位元素（及其子元素） */
5       /* 3. 非定位的块级元素 */
6       /* 4. 非定位的浮动元素 */
7       /* 5. 非定位的行内元素 */
8       /* 6. z-index为auto的定位元素（及其子元素） */
9       /* 7. z-index为正值的定位元素（及其子元素） */
10  }
```

示例演示：

1. 基本层叠上下文：

```
代码块
1   <div class="container">
2       <div class="item item-1">Item 1 (z-index: 1)</div>
3       <div class="item item-2">Item 2 (z-index: 2)</div>
4       <div class="item item-3">Item 3 (z-index: 3)</div>
5   </div>
```

代码块

```
.container {
    position: relative; /* 创建层叠上下文 */
    z-index: 0;
}

.item {
    position: absolute;
    width: 100px;
    height: 100px;
}

.item-1 {
    background: red;
    z-index: 1;
    top: 0;
    left: 0;
}

.item-2 {
    background: green;
    z-index: 2;
    top: 20px;
    left: 20px;
}

.item-3 {
    background: blue;
    z-index: 3;
    top: 40px;
    left: 40px;
}
```

2. 嵌套层叠上下文的陷阱：

代码块

```
<div class="parent-1">
    <div class="child-1">Child 1 (z-index: 100)</div>
</div>
<div class="parent-2">
    <div class="child-2">Child 2 (z-index: 1)</div>
</div>
```

代码块

```
.parent-1 {
    position: relative;
    z-index: 1; /* 创建层叠上下文 */
    background: rgba(255, 0, 0, 0.3);
}

.parent-2 {
    position: relative;
    z-index: 2; /* 创建层叠上下文 */
    background: rgba(0, 255, 0, 0.3);
}

.child-1 {
    position: relative;
    z-index: 100; /* 在parent-1的层叠上下文中 */
    background: red;
}

.child-2 {
    position: relative;
    z-index: 1; /* 在parent-2的层叠上下文中 */
    background: green;
}

/* 结果：child-2会在child-1上面，因为parent-2的z-index更大 */
```

3. opacity创建的层叠上下文：

代码块

```
.opacity-context {
    opacity: 0.99; /* 创建层叠上下文 */
    position: relative;
}

.opacity-child {
    position: relative;
    z-index: -1; /* 在opacity-context的层叠上下文中 */
    background: blue;
}
```

4. transform创建的层叠上下文：

代码块

```
.transform-context {
    transform: translateZ(0); /* 创建层叠上下文 */
    position: relative;
}

.transform-child {
    position: relative;
    z-index: -1;
    background: purple;
}
```

调试层叠上下文：

1. 使用浏览器开发者工具：

代码块

```
// 检查元素是否创建了层叠上下文
function checkStackingContext(element) {
    const computed = getComputedStyle(element);

    const conditions = [
        computed.position !== 'static' && computed.zIndex !== 'auto',
        computed.opacity !== '1',
        computed.transform !== 'none',
        computed.filter !== 'none',
        computed.mixBlendMode !== 'normal',
        computed.isolation === 'isolate'
    ];

    return conditions.some(condition => condition);
}
```

2. CSS调试技巧：

```
/* 给所有可能创建层叠上下文的元素添加边框 */
*[style*="z-index"],
*[style*="opacity"],
*[style*="transform"],
*[style*="filter"] {
    outline: 2px solid red !important;
}
```

常见问题和解决方案：

1. z-index不生效：

代码块

```
/* 问题：z-index对static元素无效 */
.problem {
    z-index: 999; /* 无效 */
}

/* 解决：给元素定位 */
.solution {
    position: relative; /* 或 absolute, fixed */
    z-index: 999;
}
```

2. 子元素无法超越父元素的层叠上下文：

代码块

```
/* 问题：子元素被限制在父元素的层叠上下文中 */
.parent {
    position: relative;
    z-index: 1;
}

.child {
    position: relative;
    z-index: 9999; /* 无法超越其他z-index为2的元素 */
}

/* 解决：调整父元素的z-index或结构 */
.parent {
```

```
14      position: relative;
15      z-index: 10; /* 提高父元素的z-index */
16  }
```

最佳实践：

- 理解层叠上下文的创建条件
- 避免过度使用z-index
- 使用有意义的z-index分层策略
- 在复杂布局中谨慎使用transform、opacity等属性
- 使用CSS架构方法管理层叠关系

---

32. CSS中的contain属性有什么作用？

参考答案：

contain属性定义：

contain属性允许开发者指定特定的DOM元素和它的子元素，让它们能够独立于整个DOM树结构之外。这个属性对于性能优化非常有用。

语法：

代码块

```
1   .element {
2       contain: none | strict | content | size | layout | style | paint | inline-
    size;
3   }
```

各个值的含义：

1. none（默认值）：

代码块

```
.no-contain {
    contain: none; /* 不应用任何包含 */
}
```

2. size：

代码块

```
.size-contain {
    contain: size;
    /* 元素的尺寸计算不依赖于其子元素的内容 */
    /* 必须明确指定元素的尺寸 */
    width: 300px;
    height: 200px;
}
```

3. layout：

代码块

```
.layout-contain {
    contain: layout;
    /* 元素外部无法影响其内部布局，反之亦然 */
    /* 元素建立独立的格式化上下文 */
}
```

4. style：

代码块

```
.style-contain {
    contain: style;
    /* 计数器和引用的作用域限制在该元素内 */
}
```

5. paint：

代码块

```
.paint-contain {
    contain: paint;
    /* 元素的后代不会显示在其边界之外 */
    /* 类似于 overflow: hidden */
}
```

6. inline-size：

代码块

```
.inline-size-contain {
    contain: inline-size;
    /* 元素的内联尺寸计算不依赖于其子元素 */
}
```

复合值：

7. content：

代码块

```
.content-contain {
    contain: content;
    /* 等价于 contain: layout style paint */
}
```

8. strict：

代码块

```
.strict-contain {
    contain: strict;
    /* 等价于 contain: size layout style paint */
}
```

实际应用场景：

1. 独立组件优化：

代码块

```
.widget {
    contain: layout style paint;
    /* 小部件的内部变化不会影响页面其他部分 */
    width: 300px;
    height: 200px;
    border: 1px solid #ccc;
    overflow: hidden;
}

.widget-content {
    /* 内部内容的变化被包含在widget内 */
    position: relative;
}
```

2. 无限滚动列表：

代码块

```
.infinite-list {
    contain: layout style paint;
    height: 400px;
    overflow-y: auto;
}

.list-item {
    contain: layout paint;
    height: 50px;
    /* 每个列表项的变化不会影响其他项 */
}
```

3. 卡片组件：

```
.card {
    contain: layout paint;
    width: 250px;
    min-height: 200px;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);
    overflow: hidden;
}

.card-content {
    padding: 16px;
}
```

4. 模态框：

代码块

```
.modal {
    contain: layout style paint;
    position: fixed;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    width: 500px;
    max-height: 80vh;
    background: white;
    border-radius: 8px;
    overflow: auto;
}
```

5. 图表组件：

代码块

```
.chart-container {
    contain: size layout paint;
    width: 400px;
    height: 300px;
    /* 图表内部的重新渲染不会影响页面其他部分 */
}
```

性能优化效果：

1. 减少重排（Reflow）：

代码块

```
.optimized-section {
    contain: layout;
    /* 内部元素的布局变化不会触发外部重排 */
}

.dynamic-content {
    /* 内容的动态变化被包含 */
    height: auto;
}
```

2. 减少重绘（Repaint）：

代码块

```
.paint-optimized {
    contain: paint;
    /* 内部绘制操作不会影响外部 */
    animation: colorChange 2s infinite;
}

@keyframes colorChange {
    0% { background: red; }
    50% { background: blue; }
    100% { background: red; }
}
```

3. 样式计算优化：

代码块

```
.style-optimized {
    contain: style;
    /* CSS计数器等样式计算被限制在内部 */
    counter-reset: item;
```

```
}

.item {
    counter-increment: item;
}

.item::before {
    content: counter(item) ". ";
}
```

与其他CSS属性的关系：

1. 与overflow的区别：

代码块

```
/* overflow只是裁剪视觉内容 */
.overflow-hidden {
    overflow: hidden;
    width: 200px;
    height: 200px;
}

/* contain: paint 还会创建包含块 */
.paint-contained {
    contain: paint;
    width: 200px;
    height: 200px;
}
```

2. 与position的配合：

代码块

```
.positioned-container {
    position: relative;
    contain: layout;
    /* 为内部绝对定位元素提供包含块 */
}

.absolute-child {
    position: absolute;
```

```
    top: 0;
    left: 0;
    /* 相对于.positioned-container定位 */
}
```

注意事项和限制：

1. size包含的要求：

代码块

```
.size-contained {
    contain: size;
    /* 必须明确指定尺寸，否则可能为0 */
    width: 300px;
    height: 200px;
}
```

2. 可访问性考虑：

代码块

```
.accessible-contained {
    contain: layout paint;
    /* 确保辅助技术仍能访问内容 */
    /* 避免过度使用可能影响屏幕阅读器 */
}
```

3. 调试困难：

代码块

```
/* 开发时可以临时禁用contain */
.debug-mode .contained {
    contain: none !important;
}
```

浏览器兼容性：

- Chrome 52+
- Firefox 69+
- Safari 15.4+
- IE不支持

检测支持：

```
代码块
1  // 检测contain属性支持
2  function supportsContain() {
3      return CSS.supports('contain', 'layout');
4  }
5
6  if (supportsContain()) {
7      document.body.classList.add('supports-contain');
8  }
```

最佳实践：

- 在独立组件上使用contain
- 避免在根元素或大容器上使用strict
- 结合性能监控工具测试效果
- 在复杂动画组件中优先考虑使用
- 注意可访问性影响

---

33. CSS中的aspect-ratio属性如何使用？

参考答案：

aspect-ratio定义：

aspect-ratio属性用于设置元素的首选宽高比，浏览器会根据这个比例自动计算元素的尺寸。

基本语法：

```
代码块
1   .element {
2       aspect-ratio: <ratio> | auto;
3   }
```

使用方式：

1. 数字比例：

```
代码块
1   .square {
2       aspect-ratio: 1; /* 1:1 正方形 */
3       width: 200px; /* 高度自动为200px */
4   }
5
6   .rectangle {
7       aspect-ratio: 16/9; /* 16:9 宽屏比例 */
8       width: 100%;
9   }
10
11  .portrait {
12      aspect-ratio: 3/4; /* 3:4 竖屏比例 */
13      height: 400px; /* 宽度自动为300px */
14  }
```

2. 小数比例：

```
代码块
1   .golden-ratio {
2       aspect-ratio: 1.618; /* 黄金比例 */
3       width: 300px;
4   }
5
6   .custom-ratio {
```

```
    aspect-ratio: 2.5; /* 自定义比例 */
    height: 200px;
}
```

3. auto值：

代码块

```
.auto-ratio {
    aspect-ratio: auto; /* 使用内容的自然宽高比 */
}

.conditional-ratio {
    aspect-ratio: auto 16/9; /* 有内容时用auto，否则用16/9 */
}
```

实际应用场景：

1. 响应式视频容器：

代码块

```
.video-container {
    aspect-ratio: 16/9;
    width: 100%;
    background: #000;
    position: relative;
}

.video-container iframe,
.video-container video {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
}
```

2. 图片占位符：

代码块

```
.image-placeholder {
    aspect-ratio: 4/3;
    width: 100%;
    background: #f0f0f0;
    display: flex;
    align-items: center;
    justify-content: center;
}

.image-placeholder img {
    max-width: 100%;
    max-height: 100%;
    object-fit: cover;
}
```

3. 卡片布局：

代码块

```
.card-grid {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
    gap: 20px;
}

.card {
    aspect-ratio: 1.2; /* 稍微宽一点的矩形 */
    background: white;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);
    padding: 20px;
    display: flex;
    flex-direction: column;
}
```

4. 社交媒体帖子：

代码块

```
.instagram-post {
    aspect-ratio: 1; /* Instagram正方形 */
```

```
    max-width: 400px;
    background: #fff;
    border: 1px solid #ddd;
}

.story-preview {
    aspect-ratio: 9/16; /* 竖屏故事比例 */
    width: 100px;
    background: linear-gradient(45deg, #ff6b6b, #4ecdc4);
    border-radius: 8px;
}
```

5. 产品展示：

代码块

```
.product-image {
    aspect-ratio: 1;
    width: 100%;
    background: #f8f9fa;
    border-radius: 12px;
    overflow: hidden;
}

.product-image img {
    width: 100%;
    height: 100%;
    object-fit: cover;
}
```

与其他属性的配合：

1. 与Grid布局：

代码块

```
.photo-grid {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
    gap: 16px;
}

```

```
.photo-item {
    aspect-ratio: 1;
    background: #eee;
    border-radius: 8px;
    overflow: hidden;
}
```

2. 与Flexbox：

代码块

```
.flex-container {
    display: flex;
    gap: 20px;
    flex-wrap: wrap;
}

.flex-item {
    aspect-ratio: 16/9;
    flex: 1;
    min-width: 250px;
    background: #007bff;
    border-radius: 8px;
}
```

3. 与object-fit：

代码块

```
.media-container {
    aspect-ratio: 21/9; /* 超宽屏比例 */
    width: 100%;
    overflow: hidden;
    border-radius: 12px;
}

.media-container img,
.media-container video {
    width: 100%;
    height: 100%;
    object-fit: cover;
}
```

响应式应用：

1. 不同屏幕尺寸的比例：

代码块

```
.responsive-ratio {
    aspect-ratio: 4/3; /* 默认比例 */
    width: 100%;
}

@media (max-width: 768px) {
    .responsive-ratio {
        aspect-ratio: 1; /* 移动端改为正方形 */
    }
}

@media (min-width: 1200px) {
    .responsive-ratio {
        aspect-ratio: 21/9; /* 大屏幕使用超宽比例 */
    }
}
```

2. 容器查询配合：

代码块

```
.container {
    container-type: inline-size;
}

.adaptive-ratio {
    aspect-ratio: 16/9;
}

@container (max-width: 400px) {
    .adaptive-ratio {
        aspect-ratio: 1;
    }
}
```

替代方案（兼容性处理）：

1. 使用padding-top技巧：

代码块

```
/* 传统方法：16:9比例 */
.aspect-ratio-16-9 {
    position: relative;
    width: 100%;
    padding-top: 56.25%; /* 9/16 * 100% */
}

.aspect-ratio-16-9 > * {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
}

/* 现代方法 */
.modern-aspect-ratio {
    aspect-ratio: 16/9;
    width: 100%;
}
```

2. JavaScript回退：

代码块

```
// 检测aspect-ratio支持
if (!CSS.supports('aspect-ratio', '1')) {
    // 使用JavaScript实现回退
    function maintainAspectRatio(element, ratio) {
        const updateHeight = () => {
            const width = element.offsetWidth;
            element.style.height = `${width / ratio}px`;
        };

        updateHeight();
        window.addEventListener('resize', updateHeight);
```

```
    }

    document.querySelectorAll('.aspect-ratio-fallback').forEach(el => {
        const ratio = parseFloat(el.dataset.ratio) || 1;
        maintainAspectRatio(el, ratio);
    });
}
```

动画效果：

代码块

```
.animated-ratio {
    aspect-ratio: 1;
    width: 200px;
    background: #3498db;
    transition: aspect-ratio 0.3s ease;
}

.animated-ratio:hover {
    aspect-ratio: 2; /* 悬停时变为2:1 */
}
```

浏览器兼容性：

- Chrome 88+
- Firefox 89+
- Safari 15+
- IE不支持

最佳实践：

- 优先使用aspect-ratio而非padding-top技巧
- 结合object-fit处理媒体内容
- 在响应式设计中灵活调整比例
- 为不支持的浏览器提供回退方案
- 考虑内容的实际需求选择合适比例

34. CSS中的scroll-behavior属性有什么作用？

参考答案：

scroll-behavior定义：

scroll-behavior属性用于设置滚动框中滚动行为的表现，特别是通过导航或CSSOM滚动API触发的滚动。

语法：

代码块

```
.element {
    scroll-behavior: auto | smooth;
}
```

属性值：

1. auto（默认值）：

代码块

```
.auto-scroll {
    scroll-behavior: auto;
    /* 立即跳转，没有平滑动画 */
}
```

2. smooth：

代码块

```
.smooth-scroll {
    scroll-behavior: smooth;
    /* 平滑滚动动画 */
```

```
}
```

应用场景：

1. 全局平滑滚动：

代码块

```
html {
    scroll-behavior: smooth;
}

/* 所有锚点链接都会平滑滚动 */
```

2. 特定容器的平滑滚动：

代码块

```
.scroll-container {
    height: 300px;
    overflow-y: auto;
    scroll-behavior: smooth;
}
```

3. 导航菜单应用：

代码块

```
<nav class="navbar">
    <a href="#section1">Section 1</a>
    <a href="#section2">Section 2</a>
    <a href="#section3">Section 3</a>
</nav>

<section id="section1">Content 1</section>
<section id="section2">Content 2</section>
<section id="section3">Content 3</section>
```

代码块

```
html {
    scroll-behavior: smooth;
}

.navbar {
    position: fixed;
    top: 0;
    width: 100%;
    background: white;
    padding: 10px;
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

.navbar a {
    margin-right: 20px;
    text-decoration: none;
    color: #333;
}

section {
    height: 100vh;
    padding: 80px 20px 20px;
}
```

4. 侧边栏滚动：

代码块

```
.sidebar {
    height: 100vh;
    overflow-y: auto;
    scroll-behavior: smooth;
}

.sidebar-nav a {
    display: block;
    padding: 10px;
    text-decoration: none;
    color: #666;
}
```

5. 表格滚动：

代码块

```
.table-container {
    max-height: 400px;
    overflow-y: auto;
    scroll-behavior: smooth;
}

.table-nav button {
    margin: 5px;
    padding: 5px 10px;
    background: #007bff;
    color: white;
    border: none;
    cursor: pointer;
}
```

与JavaScript的配合：

1. scrollIntoView方法：

代码块

```
// CSS设置了scroll-behavior: smooth时，这些方法会自动平滑滚动
document.getElementById('target').scrollIntoView();

// 也可以在方法中指定行为
document.getElementById('target').scrollIntoView({
    behavior: 'smooth',
    block: 'start',
    inline: 'nearest'
});
```

2. scrollTo方法：

代码块

```
// 平滑滚动到顶部
```

```
window.scrollTo({
    top: 0,
    behavior: 'smooth'
});

// 滚动到指定位置
window.scrollTo({
    top: 1000,
    left: 0,
    behavior: 'smooth'
});
```

3. 自定义滚动按钮：

代码块

```
<button id="scrollToTop">回到顶部</button>
<button id="scrollToBottom">滚动到底部</button>
```

代码块

```
html {
    scroll-behavior: smooth;
}

#scrollToTop, #scrollToBottom {
    position: fixed;
    right: 20px;
    padding: 10px 15px;
    background: #007bff;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}

#scrollToTop {
    bottom: 70px;
}

#scrollToBottom {
    bottom: 20px;
```

```
}
```

代码块

```
document.getElementById('scrollToTop').addEventListener('click', () => {
    window.scrollTo({ top: 0, behavior: 'smooth' });
});

document.getElementById('scrollToBottom').addEventListener('click', () => {
    window.scrollTo({ top: document.body.scrollHeight, behavior: 'smooth' });
});
```

高级应用：

1. 条件性平滑滚动：

代码块

```
/* 默认不平滑 */
html {
    scroll-behavior: auto;
}

/* 用户偏好减少动画时保持auto */
@media (prefers-reduced-motion: no-preference) {
    html {
        scroll-behavior: smooth;
    }
}
```

2. 不同容器不同行为：

代码块

```
.instant-scroll {
    scroll-behavior: auto;
}

.smooth-scroll {
```

```
    scroll-behavior: smooth;
}

.modal {
    overflow-y: auto;
    scroll-behavior: smooth;
}

.code-editor {
    overflow: auto;
    scroll-behavior: auto; /* 代码编辑器通常需要即时滚动 */
}
```

3. 响应式滚动行为：

代码块

```
html {
    scroll-behavior: auto;
}

/* 只在大屏幕上启用平滑滚动 */
@media (min-width: 768px) {
    html {
        scroll-behavior: smooth;
    }
}
```

性能考虑：

1. 用户偏好检测：

代码块

```
/* 尊重用户的动画偏好 */
@media (prefers-reduced-motion: reduce) {
    html {
        scroll-behavior: auto;
    }
}

@media (prefers-reduced-motion: no-preference) {
```

```
    html {
        scroll-behavior: smooth;
    }
}
```

2. JavaScript检测：

代码块

```
// 检测用户是否偏好减少动画
const prefersReducedMotion = window.matchMedia('(prefers-reduced-motion: reduce)');

function updateScrollBehavior() {
    if (prefersReducedMotion.matches) {
        document.documentElement.style.scrollBehavior = 'auto';
    } else {
        document.documentElement.style.scrollBehavior = 'smooth';
    }
}

updateScrollBehavior();
prefersReducedMotion.addEventListener('change', updateScrollBehavior);
```

兼容性处理：

1. 特性检测：

代码块

```
// 检测scroll-behavior支持
function supportsScrollBehavior() {
    return 'scrollBehavior' in document.documentElement.style;
}

if (!supportsScrollBehavior()) {
    // 使用polyfill或自定义实现
    console.log('需要scroll-behavior polyfill');
}
```

2. Polyfill实现：

代码块

```
// 简单的平滑滚动polyfill
function smoothScrollTo(target, duration = 800) {
    const targetElement = typeof target === 'string'
        ? document.querySelector(target)
        : target;

    if (!targetElement) return;

    const targetPosition = targetElement.offsetTop;
    const startPosition = window.pageYOffset;
    const distance = targetPosition - startPosition;
    let startTime = null;

    function animation(currentTime) {
        if (startTime === null) startTime = currentTime;
        const timeElapsed = currentTime - startTime;
        const run = ease(timeElapsed, startPosition, distance, duration);
        window.scrollTo(0, run);
        if (timeElapsed < duration) requestAnimationFrame(animation);
    }

    function ease(t, b, c, d) {
        t /= d / 2;
        if (t < 1) return c / 2 * t * t + b;
        t--;
        return -c / 2 * (t * (t - 2) - 1) + b;
    }

    requestAnimationFrame(animation);
}
```

浏览器兼容性：

- Chrome 61+
- Firefox 36+
- Safari 14+
- IE不支持

最佳实践：

- 考虑用户的动画偏好设置
- 在长页面和单页应用中使用
- 避免在需要精确控制的场景中使用
- 结合JavaScript API获得更好的控制
- 为不支持的浏览器提供polyfill

---

35. 什么是CSS的逻辑属性（Logical Properties）？

参考答案：

逻辑属性定义：

CSS逻辑属性是相对于元素的书写模式、方向性和文本方向的属性，而不是相对于屏幕的物理方向。这使得样式能够更好地适应不同的语言和书写方向。

物理属性 vs 逻辑属性：

传统物理属性：

代码块

```
.physical {
    margin-top: 10px;
    margin-right: 20px;
    margin-bottom: 10px;
    margin-left: 20px;

    padding-top: 5px;
    padding-right: 15px;
    padding-bottom: 5px;
    padding-left: 15px;

    border-top: 1px solid red;
```

```
    border-right: 2px solid blue;
    border-bottom: 1px solid red;
    border-left: 2px solid blue;
}
```

对应的逻辑属性：

代码块

```
.logical {
    margin-block-start: 10px;    /* margin-top */
    margin-inline-end: 20px;     /* margin-right */
    margin-block-end: 10px;      /* margin-bottom */
    margin-inline-start: 20px;   /* margin-left */

    padding-block-start: 5px;    /* padding-top */
    padding-inline-end: 15px;    /* padding-right */
    padding-block-end: 5px;      /* padding-bottom */
    padding-inline-start: 15px;  /* padding-left */

    border-block-start: 1px solid red;    /* border-top */
    border-inline-end: 2px solid blue;    /* border-right */
    border-block-end: 1px solid red;      /* border-bottom */
    border-inline-start: 2px solid blue;  /* border-left */
}
```

简写属性：

1. margin和padding：

代码块

```
.shorthand {
    /* 物理属性 */
    margin: 10px 20px;
    padding: 5px 15px;

    /* 逻辑属性 */
    margin-block: 10px;      /* margin-block-start + margin-block-end */
    margin-inline: 20px;     /* margin-inline-start + margin-inline-end */
    padding-block: 5px;      /* padding-block-start + padding-block-end */
    padding-inline: 15px;    /* padding-inline-start + padding-inline-end */
```

```
}
```

2. border：

代码块

```
.border-logical {
    /* 逻辑边框 */
    border-block: 1px solid red;        /* 上下边框 */
    border-inline: 2px solid blue;      /* 左右边框 */
    border-block-start: 3px solid green; /* 顶部边框 */
    border-inline-end: 1px dashed orange; /* 右侧边框 */
}
```

尺寸逻辑属性：

代码块

```
.size-logical {
    /* 物理尺寸 */
    width: 300px;
    height: 200px;
    max-width: 500px;
    min-height: 100px;

    /* 逻辑尺寸 */
    inline-size: 300px;      /* width */
    block-size: 200px;       /* height */
    max-inline-size: 500px;  /* max-width */
    min-block-size: 100px;   /* min-height */
}
```

定位逻辑属性：

代码块

```
.position-logical {
```

```
    position: absolute;

    /* 物理定位 */
    top: 10px;
    right: 20px;
    bottom: 10px;
    left: 20px;

    /* 逻辑定位 */
    inset-block-start: 10px;    /* top */
    inset-inline-end: 20px;     /* right */
    inset-block-end: 10px;      /* bottom */
    inset-inline-start: 20px;   /* left */
}

/* 简写形式 */
.position-shorthand {
    position: absolute;
    inset-block: 10px;     /* top + bottom */
    inset-inline: 20px;    /* left + right */
    /* 或者 */
    inset: 10px 20px;      /* 所有方向 */
}
```

书写模式的影响：

1. 水平书写模式（默认）：

代码块

```
.horizontal {
    writing-mode: horizontal-tb;
    direction: ltr;

    margin-inline-start: 20px; /* 等于 margin-left */
    margin-inline-end: 10px;   /* 等于 margin-right */
    margin-block-start: 15px;  /* 等于 margin-top */
    margin-block-end: 5px;     /* 等于 margin-bottom */
}
```

2. 垂直书写模式：

代码块

```
.vertical {
    writing-mode: vertical-rl;

    margin-inline-start: 20px; /* 等于 margin-top */
    margin-inline-end: 10px;   /* 等于 margin-bottom */
    margin-block-start: 15px;  /* 等于 margin-right */
    margin-block-end: 5px;     /* 等于 margin-left */
}
```

3. RTL（从右到左）书写：

代码块

```
.rtl {
    direction: rtl;

    margin-inline-start: 20px; /* 等于 margin-right */
    margin-inline-end: 10px;   /* 等于 margin-left */
}
```

实际应用场景：

1. 国际化网站：

代码块

```
.article {
    /* 使用逻辑属性确保在不同语言下都正确显示 */
    padding-inline: 20px;
    margin-block: 15px;
    border-inline-start: 3px solid #007bff;
}

/* 阿拉伯语或希伯来语 */
[dir="rtl"] .article {
    /* 逻辑属性会自动适应RTL方向 */
    /* border-inline-start 会变成右边框 */
}
```

2. 响应式设计：

代码块

```
.card {
    inline-size: 100%;
    max-inline-size: 400px;
    padding-block: 20px;
    padding-inline: 15px;
    margin-block-end: 20px;
}

@media (min-width: 768px) {
    .card {
        inline-size: 48%;
        margin-inline-end: 4%;
    }
}
```

3. 组件库开发：

代码块

```
.button {
    padding-block: 8px;
    padding-inline: 16px;
    border: 1px solid transparent;
    border-radius: 4px;
}

.button--icon {
    padding-inline-start: 12px;
}

.button--icon::before {
    margin-inline-end: 8px;
}
```

4. 表单布局：

代码块

```css
.form-group {
    margin-block-end: 16px;
}

.form-label {
    display: block;
    margin-block-end: 4px;
    font-weight: 500;
}

.form-input {
    inline-size: 100%;
    padding-block: 8px;
    padding-inline: 12px;
    border: 1px solid #ccc;
    border-radius: 4px;
}

.form-help {
    margin-block-start: 4px;
    font-size: 0.875rem;
    color: #666;
}
```

与Flexbox和Grid的结合：

代码块

```css
.flex-container {
    display: flex;
    gap: 16px;
    padding-inline: 20px;
}

.flex-item {
    flex: 1;
    padding-block: 12px;
    padding-inline: 16px;
    border-inline-start: 2px solid #007bff;
}

.grid-container {
    display: grid;
```

```
    grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
    gap: 20px;
    padding-inline: 20px;
}

.grid-item {
    padding-block: 16px;
    padding-inline: 20px;
    border-block-start: 3px solid #28a745;
}
```

浏览器兼容性：

- Chrome 69+
- Firefox 41+（部分支持）
- Safari 12.1+
- IE不支持

渐进增强策略：

代码块

```
.progressive-enhancement {
    /* 物理属性作为回退 */
    margin-left: 20px;
    margin-right: 10px;
    padding-top: 15px;
    padding-bottom: 15px;

    /* 逻辑属性覆盖（支持的浏览器） */
    margin-inline-start: 20px;
    margin-inline-end: 10px;
    padding-block: 15px;
}
```

检测支持：

代码块

```
// 检测逻辑属性支持
```

```
function supportsLogicalProperties() {
    return CSS.supports('margin-inline-start', '0px');
}

if (supportsLogicalProperties()) {
    document.body.classList.add('supports-logical-props');
}
```

最佳实践：

- 在新项目中优先使用逻辑属性
- 为不支持的浏览器提供物理属性回退
- 在国际化项目中必须使用逻辑属性
- 组件库开发时使用逻辑属性提高复用性
- 结合writing-mode和direction属性测试效果

---

36. CSS中的scroll-snap属性如何实现滚动吸附效果？

参考答案：

scroll-snap定义：

CSS Scroll Snap允许开发者创建滚动体验，其中滚动位置会"吸附"到特定的位置，而不是在任意位置停止。

基本属性：

1. scroll-snap-type（容器属性）：

代码块

```
.scroll-container {
    scroll-snap-type: none | x | y | block | inline | both | mandatory | proximity;
```

```
}
```

2. scroll-snap-align（子项属性）：

代码块

```
.scroll-item {
    scroll-snap-align: none | start | end | center;
}
```

基本用法：

1. 水平滚动吸附：

代码块

```
.horizontal-scroll {
    display: flex;
    overflow-x: auto;
    scroll-snap-type: x mandatory;
    gap: 20px;
    padding: 20px;
}

.horizontal-scroll .item {
    flex: 0 0 300px;
    height: 200px;
    background: #f0f0f0;
    border-radius: 8px;
    scroll-snap-align: start;
}
```

2. 垂直滚动吸附：

代码块

```
.vertical-scroll {
    height: 400px;
    overflow-y: auto;
```

```
    scroll-snap-type: y mandatory;
}

.vertical-scroll .section {
    height: 100vh;
    scroll-snap-align: start;
    display: flex;
    align-items: center;
    justify-content: center;
    font-size: 2rem;
}
```

scroll-snap-type详解：

1. mandatory vs proximity：

代码块

```
/* 强制吸附 - 滚动必须停在吸附点 */
.mandatory {
    scroll-snap-type: x mandatory;
}

/* 接近吸附 - 只有在接近吸附点时才吸附 */
.proximity {
    scroll-snap-type: x proximity;
}
```

2. 方向控制：

代码块

```
.x-axis { scroll-snap-type: x mandatory; }      /* 水平轴 */
.y-axis { scroll-snap-type: y mandatory; }      /* 垂直轴 */
.both-axis { scroll-snap-type: both mandatory; } /* 双轴 */
.block-axis { scroll-snap-type: block mandatory; } /* 块轴（逻辑属性） */
.inline-axis { scroll-snap-type: inline mandatory; } /* 内联轴（逻辑属性） */
```

scroll-snap-align详解：

代码块

```
.align-start { scroll-snap-align: start; }    /* 对齐到开始位置 */
.align-center { scroll-snap-align: center; }  /* 对齐到中心位置 */
.align-end { scroll-snap-align: end; }        /* 对齐到结束位置 */
.align-none { scroll-snap-align: none; }      /* 不参与吸附 */
```

实际应用场景：

1. 图片轮播：

代码块

```
<div class="carousel">
    <div class="slide">Slide 1</div>
    <div class="slide">Slide 2</div>
    <div class="slide">Slide 3</div>
    <div class="slide">Slide 4</div>
</div>
```

代码块

```
.carousel {
    display: flex;
    overflow-x: auto;
    scroll-snap-type: x mandatory;
    scroll-behavior: smooth;
    -webkit-overflow-scrolling: touch; /* iOS平滑滚动 */
}

.slide {
    flex: 0 0 100%;
    height: 300px;
    scroll-snap-align: start;
    background: linear-gradient(45deg, #ff6b6b, #4ecdc4);
    display: flex;
    align-items: center;
    justify-content: center;
    font-size: 2rem;
```

```
    color: white;
}
```

2. 卡片滚动：

代码块

```
.card-scroll {
    display: flex;
    overflow-x: auto;
    scroll-snap-type: x mandatory;
    gap: 16px;
    padding: 20px;
}

.card {
    flex: 0 0 280px;
    height: 200px;
    background: white;
    border-radius: 12px;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    scroll-snap-align: start;
    padding: 20px;
}

/* 移动端优化 */
@media (max-width: 768px) {
    .card {
        flex: 0 0 250px;
    }
}
```

3. 全屏滚动：

代码块

```
.fullpage-scroll {
    height: 100vh;
    overflow-y: auto;
    scroll-snap-type: y mandatory;
    scroll-behavior: smooth;
}

```

```
.section {
    height: 100vh;
    scroll-snap-align: start;
    display: flex;
    align-items: center;
    justify-content: center;
    font-size: 3rem;
}

.section:nth-child(1) { background: #ff6b6b; }
.section:nth-child(2) { background: #4ecdc4; }
.section:nth-child(3) { background: #45b7d1; }
.section:nth-child(4) { background: #96ceb4; }
```

4. 产品展示：

代码块

```
.product-gallery {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
    gap: 20px;
    height: 400px;
    overflow-y: auto;
    scroll-snap-type: y proximity;
    padding: 20px;
}

.product-item {
    scroll-snap-align: start;
    background: white;
    border-radius: 8px;
    padding: 16px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}
```

高级功能：

1. scroll-snap-stop：

代码块

```
.scroll-item {
    scroll-snap-align: start;
    scroll-snap-stop: normal; /* 默认，可以跳过 */
    /* 或 */
    scroll-snap-stop: always; /* 强制停止，不能跳过 */
}
```

2. 混合对齐：

代码块

```
.mixed-alignment .item:first-child {
    scroll-snap-align: start;
}

.mixed-alignment .item:last-child {
    scroll-snap-align: end;
}

.mixed-alignment .item {
    scroll-snap-align: center;
}
```

3. 响应式吸附：

代码块

```
.responsive-snap {
    overflow-x: auto;
    scroll-snap-type: x proximity;
}

@media (max-width: 768px) {
    .responsive-snap {
        scroll-snap-type: x mandatory;
    }
}

.responsive-snap .item {
    scroll-snap-align: start;
}

@media (max-width: 768px) {
```

```
    .responsive-snap .item {
        scroll-snap-align: center;
    }
}
```

与JavaScript的结合：

1. 编程式滚动：

代码块

```
// 滚动到特定元素
function scrollToSlide(index) {
    const container = document.querySelector('.carousel');
    const slide = container.children[index];

    slide.scrollIntoView({
        behavior: 'smooth',
        block: 'nearest',
        inline: 'start'
    });
}

// 监听滚动事件
const container = document.querySelector('.carousel');
container.addEventListener('scroll', () => {
    // 可以在这里更新指示器等
    console.log('Scrolled to:', container.scrollLeft);
});
```

2. 滚动指示器：

代码块

```
<div class="carousel-container">
    <div class="carousel">
        <div class="slide">Slide 1</div>
        <div class="slide">Slide 2</div>
        <div class="slide">Slide 3</div>
    </div>
    <div class="indicators">
        <button class="indicator active"></button>
```

```
        <button class="indicator"></button>
        <button class="indicator"></button>
    </div>
</div>
```

代码块

```
const carousel = document.querySelector('.carousel');
const indicators = document.querySelectorAll('.indicator');

// 更新指示器
function updateIndicators() {
    const slideWidth = carousel.offsetWidth;
    const currentSlide = Math.round(carousel.scrollLeft / slideWidth);

    indicators.forEach((indicator, index) => {
        indicator.classList.toggle('active', index === currentSlide);
    });
}

carousel.addEventListener('scroll', updateIndicators);

// 点击指示器滚动
indicators.forEach((indicator, index) => {
    indicator.addEventListener('click', () => {
        const slideWidth = carousel.offsetWidth;
        carousel.scrollTo({
            left: index * slideWidth,
            behavior: 'smooth'
        });
    });
});
```

性能优化：

代码块

```
.optimized-scroll {
    overflow-x: auto;
    scroll-snap-type: x mandatory;

```

```
    /* 优化滚动性能 */
    -webkit-overflow-scrolling: touch;
    overscroll-behavior-x: contain;

    /* 硬件加速 */
    transform: translateZ(0);
    will-change: scroll-position;
}

.scroll-item {
    scroll-snap-align: start;

    /* 避免重绘 */
    contain: layout style paint;
}
```

浏览器兼容性：

- Chrome 69+
- Firefox 68+
- Safari 11+
- IE不支持

最佳实践：

- 结合scroll-behavior: smooth使用
- 在移动端特别有用
- 注意性能，避免过多的吸附点
- 提供视觉指示器帮助用户导航
- 考虑用户的滚动习惯和期望
- 在不支持的浏览器中提供回退方案

---

37. CSS中的@supports规则如何使用？

参考答案：

@supports定义：

@supports规则（也称为特性查询）允许开发者检测浏览器是否支持特定的CSS属性和值，从而实现渐进增强和优雅降级。

基本语法：

代码块

```
@supports (property: value) {
    /* 支持时的样式 */
}

@supports not (property: value) {
    /* 不支持时的样式 */
}
```

基本用法：

1. 检测单个属性：

代码块

```
/* 检测是否支持Grid布局 */
@supports (display: grid) {
    .container {
        display: grid;
        grid-template-columns: repeat(3, 1fr);
        gap: 20px;
    }
}

/* 检测是否支持Flexbox */
@supports (display: flex) {
    .flex-container {
        display: flex;
        justify-content: space-between;
        align-items: center;
    }
}
```

2. 检测CSS变量：

代码块

```
@supports (--css: variables) {
    :root {
        --primary-color: #007bff;
        --secondary-color: #6c757d;
    }

    .button {
        background-color: var(--primary-color);
        color: white;
    }
}
```

3. 检测复杂属性：

代码块

```
/* 检测clip-path支持 */
@supports (clip-path: polygon(0 0, 100% 0, 100% 100%, 0 100%)) {
    .clipped-element {
        clip-path: polygon(25% 0%, 100% 0%, 75% 100%, 0% 100%);
    }
}

/* 检测backdrop-filter支持 */
@supports (backdrop-filter: blur(10px)) {
    .glass-effect {
        backdrop-filter: blur(10px);
        background: rgba(255, 255, 255, 0.1);
    }
}
```

逻辑操作符：

1. and操作符：

代码块

```
@supports (display: flex) and (gap: 20px) {
    .modern-flex {
        display: flex;
        gap: 20px; /* 现代Flexbox的gap属性 */
    }
}

@supports (display: grid) and (grid-template-areas: "header header") {
    .grid-layout {
        display: grid;
        grid-template-areas:
            "header header"
            "sidebar content"
            "footer footer";
    }
}
```

2. or操作符：

代码块

```
@supports (display: -webkit-flex) or (display: flex) {
    .flexible {
        display: -webkit-flex;
        display: flex;
    }
}

@supports (transform: rotate(45deg)) or (-webkit-transform: rotate(45deg)) {
    .rotated {
        -webkit-transform: rotate(45deg);
        transform: rotate(45deg);
    }
}
```

3. not操作符：

代码块

```
@supports not (display: grid) {
    /* Grid不支持时的回退样式 */
    .fallback-layout {
        display: table;
```

```
        width: 100%;
    }

    .fallback-item {
        display: table-cell;
        vertical-align: top;
        width: 33.333%;
    }
}
```

实际应用场景：

1. 现代布局的渐进增强：

代码块

```
/* 基础布局（所有浏览器） */
.layout {
    width: 100%;
}

.layout .item {
    float: left;
    width: 33.333%;
    padding: 10px;
    box-sizing: border-box;
}

/* Flexbox增强 */
@supports (display: flex) {
    .layout {
        display: flex;
        flex-wrap: wrap;
        gap: 20px;
    }

    .layout .item {
        float: none;
        flex: 1;
        min-width: 250px;
    }
}

/* Grid进一步增强 */
```

```
@supports (display: grid) {
    .layout {
        display: grid;
        grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
        gap: 20px;
    }

    .layout .item {
        flex: none;
        min-width: auto;
    }
}
```

2. CSS形状和遮罩：

代码块

```
.shape-element {
    /* 回退样式 */
    border-radius: 50%;
    overflow: hidden;
}

@supports (clip-path: circle(50%)) {
    .shape-element {
        border-radius: 0;
        clip-path: circle(50%);
        overflow: visible;
    }
}

@supports (mask: url(#mask)) {
    .masked-element {
        mask: url(#complex-mask);
    }
}
```

3. 现代字体特性：

代码块

```
.text {
    font-family: Arial, sans-serif;
```

```
}

@supports (font-variation-settings: "wght" 400) {
    .text {
        font-family: 'Inter Variable', Arial, sans-serif;
        font-variation-settings: "wght" 400, "slnt" 0;
    }
}

@supports (font-feature-settings: "liga" 1) {
    .text {
        font-feature-settings: "liga" 1, "kern" 1;
    }
}
```

4. 滤镜和混合模式：

代码块

```
.image {
    /* 基础样式 */
    opacity: 0.8;
}

@supports (filter: blur(5px)) {
    .image {
        opacity: 1;
        filter: blur(2px) contrast(1.2);
    }
}

@supports (mix-blend-mode: multiply) {
    .overlay {
        mix-blend-mode: multiply;
    }
}
```

5. 滚动相关特性：

代码块

```
.scroll-container {
    overflow-y: auto;
```

```
}

@supports (scroll-behavior: smooth) {
    html {
        scroll-behavior: smooth;
    }
}

@supports (scroll-snap-type: y mandatory) {
    .scroll-container {
        scroll-snap-type: y mandatory;
    }

    .scroll-item {
        scroll-snap-align: start;
    }
}
```

复杂的特性检测：

1. 检测选择器支持：

代码块

```
@supports selector(:has(> img)) {
    .card:has(> img) {
        padding-top: 0;
    }
}

@supports selector(:is(h1, h2, h3)) {
    :is(h1, h2, h3) {
        margin-top: 0;
    }
}
```

2. 检测@规则支持：

代码块

```
@supports (container-type: inline-size) {
    .container {
```

```
        container-type: inline-size;
    }

    @container (min-width: 400px) {
        .card {
            display: flex;
        }
    }
}
```

与JavaScript结合：

1. JavaScript中的特性检测：

代码块

```
// 检测CSS属性支持
function supportsCSS(property, value) {
    return CSS.supports(property, value);
}

// 检测Grid支持
if (supportsCSS('display', 'grid')) {
    document.body.classList.add('supports-grid');
}

// 检测CSS变量支持
if (supportsCSS('--custom', 'property')) {
    document.body.classList.add('supports-css-vars');
}

// 检测复杂属性
if (supportsCSS('clip-path', 'polygon(0 0, 100% 0, 100% 100%, 0 100%)')) {
    document.body.classList.add('supports-clip-path');
}
```

2. 动态样式应用：

代码块

```
// 根据支持情况动态添加样式
const modernFeatures = [
```

```
    { property: 'display', value: 'grid', class: 'has-grid' },
    { property: 'backdrop-filter', value: 'blur(10px)', class: 'has-backdrop-filter' },
    { property: 'scroll-snap-type', value: 'x mandatory', class: 'has-scroll-snap' }
];

modernFeatures.forEach(feature => {
    if (CSS.supports(feature.property, feature.value)) {
        document.documentElement.classList.add(feature.class);
    }
});
```

最佳实践：

1. 渐进增强策略：

代码块

```
/* 1. 基础样式（所有浏览器） */
.component {
    background: #f0f0f0;
    padding: 20px;
    margin: 10px;
}

/* 2. 现代特性增强 */
@supports (backdrop-filter: blur(10px)) {
    .component {
        background: rgba(240, 240, 240, 0.8);
        backdrop-filter: blur(10px);
    }
}

/* 3. 最新特性 */
@supports (color: color(display-p3 1 0 0)) {
    .component {
        background: color(display-p3 0.94 0.94 0.94);
    }
}
```

2. 组合使用：

```
代码块
1  @supports (display: grid) and (gap: 20px) and (aspect-ratio: 1) {
2      .modern-grid {
3          display: grid;
4          grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
5          gap: 20px;
6      }
7
8      .modern-grid .item {
9          aspect-ratio: 1;
10     }
11 }
```

浏览器兼容性：

- Chrome 28+
- Firefox 22+
- Safari 9+
- IE不支持

注意事项：

- 不要过度使用，影响代码可读性
- 优先使用CSS特性检测而非浏览器检测
- 结合JavaScript进行更复杂的特性检测
- 始终提供合理的回退样式
- 测试各种浏览器和设备的表现

---

38. 什么是CSS的容器查询（Container Queries）？

参考答案：

容器查询定义：

CSS容器查询允许开发者根据容器元素的尺寸来应用样式，而不是根据视口尺寸。这使得组件能够根据其父容器的大小自适应，实现真正的组件级响应式设计。

基本语法：

代码块

```
/* 定义容器 */
.container {
    container-type: inline-size | block-size | size | normal;
    container-name: sidebar; /* 可选的容器名称 */
}

/* 容器查询 */
@container (min-width: 400px) {
    .card {
        display: flex;
    }
}
```

container-type属性：

1. inline-size：

代码块

```
.container {
    container-type: inline-size;
    /* 只能查询内联方向的尺寸（通常是宽度） */
}

@container (min-width: 300px) {
    .card {
        flex-direction: row;
    }
}
```

2. block-size：

代码块

```
.container {
    container-type: block-size;
    /* 只能查询块方向的尺寸（通常是高度） */
}

@container (min-height: 200px) {
    .content {
        padding: 20px;
    }
}
```

3. size：

代码块

```
.container {
    container-type: size;
    /* 可以查询两个方向的尺寸 */
}

@container (min-width: 300px) and (min-height: 200px) {
    .card {
        display: grid;
        grid-template-columns: 1fr 2fr;
    }
}
```

实际应用场景：

1. 响应式卡片组件：

代码块

```
<div class="card-container">
    <div class="card">
        <img src="image.jpg" alt="Card image">
        <div class="card-content">
            <h3>Card Title</h3>
```

```
            <p>Card description...</p>
            <button>Read More</button>
        </div>
    </div>
</div>
```

代码块

```
.card-container {
    container-type: inline-size;
    width: 100%; /* 可以是任意宽度 */
}

.card {
    background: white;
    border-radius: 8px;
    overflow: hidden;
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

/* 小容器：垂直布局 */
.card img {
    width: 100%;
    height: 200px;
    object-fit: cover;
}

.card-content {
    padding: 16px;
}

/* 大容器：水平布局 */
@container (min-width: 400px) {
    .card {
        display: flex;
    }

    .card img {
        width: 150px;
        height: auto;
        flex-shrink: 0;
    }

    .card-content {
```

```
        flex: 1;
        display: flex;
        flex-direction: column;
        justify-content: space-between;
    }
}
```

2. 侧边栏自适应：

代码块

```
.sidebar {
    container-type: inline-size;
    container-name: sidebar;
    width: 250px; /* 可变宽度 */
}

.nav-menu {
    list-style: none;
    padding: 0;
}

.nav-item {
    padding: 8px 12px;
}

.nav-link {
    display: flex;
    align-items: center;
    text-decoration: none;
    color: #333;
}

.nav-icon {
    margin-right: 8px;
}

.nav-text {
    display: none; /* 默认隐藏文字 */
}

/* 宽度足够时显示文字 */
@container sidebar (min-width: 200px) {
    .nav-text {
        display: block;
```

```
    }
}

/* 更宽时调整布局 */
@container sidebar (min-width: 300px) {
    .nav-item {
        padding: 12px 16px;
    }

    .nav-link {
        font-size: 1.1rem;
    }
}
```

3. 数据表格响应式：

代码块

```
.table-container {
    container-type: inline-size;
    overflow-x: auto;
}

.data-table {
    width: 100%;
    border-collapse: collapse;
}

.data-table th,
.data-table td {
    padding: 8px;
    text-align: left;
    border-bottom: 1px solid #ddd;
}

/* 小容器：隐藏次要列 */
.secondary-column {
    display: none;
}

@container (min-width: 600px) {
    .secondary-column {
        display: table-cell;
    }
}
```

```
}

@container (min-width: 800px) {
    .data-table th,
    .data-table td {
        padding: 12px;
    }

    .tertiary-column {
        display: table-cell;
    }
}
```

4. 媒体对象组件：

代码块

```
.media-container {
    container-type: inline-size;
}

.media-object {
    display: flex;
    gap: 16px;
}

.media-image {
    flex-shrink: 0;
    width: 60px;
    height: 60px;
    border-radius: 50%;
    object-fit: cover;
}

.media-content {
    flex: 1;
}

/* 大容器时增大图片 */
@container (min-width: 400px) {
    .media-image {
        width: 80px;
        height: 80px;
    }

    .media-object {
```

```
        gap: 20px;
    }
}

/* 超大容器时垂直居中 */
@container (min-width: 600px) {
    .media-object {
        align-items: center;
    }

    .media-image {
        width: 100px;
        height: 100px;
    }
}
```

命名容器：

代码块

```
.main-content {
    container-type: inline-size;
    container-name: main;
}

.sidebar {
    container-type: inline-size;
    container-name: sidebar;
}

/* 针对特定容器的查询 */
@container main (min-width: 800px) {
    .article {
        columns: 2;
        column-gap: 40px;
    }
}

@container sidebar (max-width: 200px) {
    .widget {
        display: none;
    }
}
```

容器查询单位：

代码块

```
.container {
    container-type: size;
}

@container (min-width: 400px) {
    .element {
        /* 容器查询单位 */
        width: 50cqw;      /* 50% 容器宽度 */
        height: 25cqh;     /* 25% 容器高度 */
        font-size: 4cqi;   /* 4% 容器内联尺寸 */
        margin: 2cqb;      /* 2% 容器块尺寸 */
        padding: 1cqmin;   /* 1% 容器较小尺寸 */
        border-width: 0.5cqmax; /* 0.5% 容器较大尺寸 */
    }
}
```

与媒体查询的结合：

代码块

```
.responsive-component {
    container-type: inline-size;
}

/* 移动端基础样式 */
@media (max-width: 768px) {
    .component {
        padding: 10px;
    }

    /* 移动端的容器查询 */
    @container (min-width: 300px) {
        .component {
            padding: 15px;
        }
```

```
    }
}

/* 桌面端样式 */
@media (min-width: 769px) {
    .component {
        padding: 20px;
    }

    /* 桌面端的容器查询 */
    @container (min-width: 500px) {
        .component {
            padding: 30px;
            display: grid;
            grid-template-columns: 1fr 2fr;
        }
    }
}
```

JavaScript交互：

代码块

```
// 检测容器查询支持
function supportsContainerQueries() {
    return CSS.supports('container-type', 'inline-size');
}

if (supportsContainerQueries()) {
    console.log('Container queries are supported');
} else {
    // 提供polyfill或回退方案
    console.log('Container queries not supported');
}

// 动态设置容器类型
function setupContainerQuery(element, type = 'inline-size') {
    if (supportsContainerQueries()) {
        element.style.containerType = type;
    }
}
```

性能考虑：

代码块

```
/* 避免过深的嵌套 */
.container {
    container-type: inline-size;
    contain: layout style; /* 优化性能 */
}

/* 合理使用容器查询 */
@container (min-width: 300px) {
    .component {
        /* 避免触发大量重排的属性 */
        transform: scale(1.1);
        opacity: 1;
    }
}
```

浏览器兼容性：

- Chrome 105+
- Firefox 110+
- Safari 16+
- IE不支持

Polyfill方案：

代码块

```
// 简单的容器查询polyfill概念
class ContainerQueryPolyfill {
    constructor() {
        this.containers = new Map();
        this.observer = new ResizeObserver(this.handleResize.bind(this));
    }

    observe(element, queries) {
        this.containers.set(element, queries);
        this.observer.observe(element);
```

```
    }

    handleResize(entries) {
        entries.forEach(entry => {
            const element = entry.target;
            const queries = this.containers.get(element);
            const width = entry.contentRect.width;

            queries.forEach(query => {
                const matches = width >= query.minWidth;
                element.classList.toggle(query.className, matches);
            });
        });
    }
}
```

最佳实践：

- 优先考虑使用容器查询而非媒体查询
- 合理设置container-type避免性能问题
- 为不支持的浏览器提供回退方案
- 结合CSS Grid和Flexbox使用
- 避免过度嵌套容器查询
- 测试各种容器尺寸下的表现

---

39. CSS中的color()函数和新的颜色空间如何使用？

参考答案：

新颜色空间概述：

CSS引入了新的颜色空间和color()函数，支持更广色域的颜色表示，包括Display P3、Rec2020等专业色彩空间。

color()函数语法：

代码块

```
.element {
    color: color(colorspace r g b / alpha);
}
```

支持的颜色空间：

1. sRGB（默认）：

代码块

```
.srgb-color {
    /* 传统RGB */
    color: rgb(255, 0, 0);

    /* 使用color()函数的sRGB */
    color: color(srgb 1 0 0);
    color: color(srgb 1 0 0 / 0.8); /* 带透明度 */
}
```

2. Display P3：

代码块

```
.p3-color {
    /* Display P3色彩空间，更广的色域 */
    color: color(display-p3 1 0 0);
    background: color(display-p3 0.8 0.2 0.9);
    border-color: color(display-p3 0.5 0.7 0.3 / 0.6);
}
```

3. Rec2020：

代码块

```
.rec2020-color {
```

```
    /* Rec2020色彩空间，用于HDR内容 */
    color: color(rec2020 1 0 0);
    background: color(rec2020 0.9 0.1 0.8);
}
```

4. ProPhoto RGB：

代码块

```
.prophoto-color {
    /* ProPhoto RGB，摄影专业色彩空间 */
    color: color(prophoto-rgb 1 0 0);
    background: color(prophoto-rgb 0.8 0.3 0.9);
}
```

新的颜色函数：

1. oklch()函数：

代码块

```
.oklch-colors {
    /* oklch(lightness chroma hue / alpha) */
    color: oklch(0.7 0.15 180); /* 青色调 */
    background: oklch(0.9 0.1 120 / 0.8); /* 浅绿色，带透明度 */
    border-color: oklch(0.5 0.2 300); /* 紫色调 */
}
```

2. oklab()函数：

代码块

```
.oklab-colors {
    /* oklab(lightness a b / alpha) */
    color: oklab(0.7 -0.1 0.1); /* 绿红轴和蓝黄轴 */
    background: oklab(0.9 0.05 -0.05 / 0.9);
}
```

3. lch()函数：

```
代码块
1   .lch-colors {
2       /* lch(lightness chroma hue / alpha) */
3       color: lch(70% 50 180); /* CIE LCH色彩空间 */
4       background: lch(90% 20 120 / 0.8);
5   }
```

4. lab()函数：

```
代码块
1   .lab-colors {
2       /* lab(lightness a b / alpha) */
3       color: lab(70% -20 30); /* CIE LAB色彩空间 */
4       background: lab(90% 10 -10 / 0.9);
5   }
```

实际应用场景：

1. 高质量显示器优化：

```
代码块
1   .hero-image {
2       /* 回退到标准sRGB */
3       background-color: rgb(255, 100, 150);
4   }
5
6   /* 支持P3显示器时使用更鲜艳的颜色 */
7   @supports (color: color(display-p3 1 0 0)) {
8       .hero-image {
9           background-color: color(display-p3 1 0.4 0.6);
10      }
11  }
```

2. 品牌色彩精确控制：

代码块

```
:root {
    /* 品牌主色 - 标准显示器 */
    --brand-primary: rgb(0, 120, 255);
    --brand-secondary: rgb(255, 80, 120);
}

@supports (color: color(display-p3 1 0 0)) {
    :root {
        /* 品牌主色 - 广色域显示器 */
        --brand-primary: color(display-p3 0 0.5 1);
        --brand-secondary: color(display-p3 1 0.3 0.5);
    }
}

.brand-button {
    background: var(--brand-primary);
    color: white;
}
```

3. 渐变中的新颜色空间：

代码块

```
.gradient-p3 {
    /* 标准渐变 */
    background: linear-gradient(45deg,
        rgb(255, 0, 0),
        rgb(0, 255, 0)
    );
}

@supports (color: color(display-p3 1 0 0)) {
    .gradient-p3 {
        /* P3色彩空间渐变，颜色更鲜艳 */
        background: linear-gradient(45deg,
            color(display-p3 1 0 0),
            color(display-p3 0 1 0)
        );
    }
}
```

4. 主题系统中的应用：

代码块

```
/* 浅色主题 */
[data-theme="light"] {
    --bg-primary: oklch(0.98 0.02 180);
    --text-primary: oklch(0.2 0.05 270);
    --accent: oklch(0.6 0.15 200);
}

/* 深色主题 */
[data-theme="dark"] {
    --bg-primary: oklch(0.15 0.02 270);
    --text-primary: oklch(0.9 0.03 180);
    --accent: oklch(0.7 0.2 200);
}

.card {
    background: var(--bg-primary);
    color: var(--text-primary);
    border-left: 4px solid var(--accent);
}
```

颜色空间转换和混合：

1. color-mix()函数：

代码块

```
.mixed-colors {
    /* 在sRGB空间中混合 */
    color: color-mix(in srgb, red 70%, blue 30%);

    /* 在P3空间中混合 */
    background: color-mix(in display-p3,
        color(display-p3 1 0 0) 60%,
        color(display-p3 0 0 1) 40%
    );

    /* 在oklch空间中混合 */
    border-color: color-mix(in oklch,
```

```
        oklch(0.8 0.15 120) 80%,
        oklch(0.6 0.2 240) 20%
    );
}
```

2. 相对颜色语法：

代码块

```
.relative-colors {
    --base-color: color(display-p3 0.8 0.2 0.9);

    /* 基于基础颜色创建变体 */
    color: color(from var(--base-color) display-p3 r g b / 0.8);
    background: color(from var(--base-color) display-p3 calc(r * 0.8) g b);
    border-color: color(from var(--base-color) display-p3 r calc(g * 1.2) b);
}
```

响应式颜色：

代码块

```
.responsive-colors {
    /* 基础颜色 */
    --primary: oklch(0.6 0.15 200);
    --secondary: oklch(0.8 0.1 120);
}

/* 高对比度模式 */
@media (prefers-contrast: high) {
    .responsive-colors {
        --primary: oklch(0.3 0.2 200);
        --secondary: oklch(0.9 0.05 120);
    }
}

/* 暗色模式 */
@media (prefers-color-scheme: dark) {
    .responsive-colors {
        --primary: oklch(0.7 0.18 200);
        --secondary: oklch(0.85 0.12 120);
```

```
    }
}
```

JavaScript中的颜色处理：

代码块

```
// 检测颜色空间支持
function supportsColorSpace(colorSpace) {
    try {
        return CSS.supports('color', `color(${colorSpace} 1 0 0)`);
    } catch {
        return false;
    }
}

// 检测各种颜色空间
const colorSpaceSupport = {
    p3: supportsColorSpace('display-p3'),
    rec2020: supportsColorSpace('rec2020'),
    oklch: CSS.supports('color', 'oklch(0.5 0.1 180)'),
    colorMix: CSS.supports('color', 'color-mix(in srgb, red, blue)')
};

console.log('Color space support:', colorSpaceSupport);

// 动态应用颜色
function applyOptimalColors() {
    const root = document.documentElement;

    if (colorSpaceSupport.p3) {
        root.style.setProperty('--brand-color', 'color(display-p3 1 0.3 0.8)');
    } else {
        root.style.setProperty('--brand-color', 'rgb(255, 76, 204)');
    }
}
```

性能和兼容性考虑：

1. 渐进增强：

代码块

```
.progressive-color {
    /* 基础颜色（所有浏览器） */
    background: #ff4080;

    /* 现代颜色空间（支持的浏览器） */
    background: oklch(0.7 0.15 340);
}

@supports (color: color(display-p3 1 0 0)) {
    .progressive-color {
        background: color(display-p3 1 0.25 0.5);
    }
}
```

2. 媒体查询检测：

代码块

```
/* 检测显示器色域 */
@media (color-gamut: srgb) {
    .adaptive-color {
        color: rgb(255, 0, 100);
    }
}

@media (color-gamut: p3) {
    .adaptive-color {
        color: color(display-p3 1 0 0.4);
    }
}

@media (color-gamut: rec2020) {
    .adaptive-color {
        color: color(rec2020 1 0 0.3);
    }
}
```

浏览器兼容性：

- Chrome 111+ (color(), oklch(), oklab())

- Firefox 113+ (部分支持)
- Safari 15+ (color(), Display P3)
- IE不支持

最佳实践：

- 始终提供sRGB回退颜色
- 使用@supports检测功能支持
- 在高端显示器上测试颜色效果
- 考虑色彩无障碍访问性
- 使用相对颜色语法创建一致的色彩系统
- 在专业显示场景中优先使用新颜色空间

---

40. CSS中的:has()伪类选择器如何使用？

参考答案：

:has()伪类定义：

:has()伪类选择器（也被称为"父选择器"）允许开发者基于元素的后代、兄弟元素或其他相关元素来选择该元素，实现了"向上"选择的能力。

基本语法：

```
代码块
1   .parent:has(.child) {
2       /* 选择包含.child的.parent元素 */
3   }
```

基础用法：

1. 选择包含特定子元素的父元素：

代码块

```
/* 选择包含图片的文章 */
.article:has(img) {
    padding-top: 0;
}

/* 选择包含视频的容器 */
.container:has(video) {
    background: #000;
    padding: 20px;
}

/* 选择包含表单的section */
.section:has(form) {
    border: 2px solid #007bff;
    border-radius: 8px;
}
```

2. 基于直接子元素选择：

代码块

```
/* 选择直接包含h1的div */
div:has(> h1) {
    margin-bottom: 30px;
}

/* 选择直接包含按钮的表单组 */
.form-group:has(> button) {
    text-align: right;
}
```

3. 基于兄弟元素选择：

代码块

```
/* 选择后面跟着.error的input */
input:has(+ .error) {
    border-color: red;
}
```

```
5
6   /* 选择前面有label的input */
7   input:has(~ label) {
8       margin-top: 5px;
9   }
```

实际应用场景：

1. 卡片组件自适应：

代码块

```
1   <div class="card">
2       <img src="image.jpg" alt="Card image">
3       <div class="card-content">
4           <h3>Card Title</h3>
5           <p>Card description</p>
6       </div>
7   </div>
8
9   <div class="card">
10      <div class="card-content">
11          <h3>Card without image</h3>
12          <p>This card has no image</p>
13      </div>
14  </div>
```

代码块

```
1   .card {
2       background: white;
3       border-radius: 8px;
4       overflow: hidden;
5       box-shadow: 0 2px 4px rgba(0,0,0,0.1);
6   }
7
8   /* 有图片的卡片 */
9   .card:has(img) {
10      display: flex;
11      flex-direction: column;
12  }
```

```
13
14  .card:has(img) img {
15      width: 100%;
16      height: 200px;
17      object-fit: cover;
18  }
19
20  /* 没有图片的卡片 */
21  .card:not(:has(img)) {
22      padding: 20px;
23      border-left: 4px solid #007bff;
24  }
```

2. 表单验证状态：

代码块

```
1  <div class="form-field">
2      <label for="email">Email</label>
3      <input type="email" id="email" required>
4      <span class="error">Please enter a valid email</span>
5  </div>
```

代码块

```
1  .form-field {
2      margin-bottom: 20px;
3  }
4
5  .form-field input {
6      width: 100%;
7      padding: 8px 12px;
8      border: 1px solid #ccc;
9      border-radius: 4px;
10 }
11
12 /* 包含错误信息的表单字段 */
13 .form-field:has(.error:not(:empty)) {
14     margin-bottom: 30px;
15 }
16
17 .form-field:has(.error:not(:empty)) input {
```

```
    border-color: #dc3545;
    background-color: #fff5f5;
}

.form-field:has(.error:not(:empty)) label {
    color: #dc3545;
}

/* 包含有效输入的表单字段 */
.form-field:has(input:valid) input {
    border-color: #28a745;
}

.form-field:has(input:valid) label {
    color: #28a745;
}
```

3. 导航菜单状态：

代码块

```
/* 包含活跃链接的导航项 */
.nav-item:has(.nav-link.active) {
    background-color: #e3f2fd;
    border-radius: 6px;
}

/* 包含下拉菜单的导航项 */
.nav-item:has(.dropdown-menu) {
    position: relative;
}

.nav-item:has(.dropdown-menu):hover .dropdown-menu {
    display: block;
}

/* 包含徽章的导航项 */
.nav-item:has(.badge) .nav-link {
    padding-right: 30px;
    position: relative;
}
```

4. 内容布局优化：

代码块

```
/* 包含侧边栏的主容器 */
.main-container:has(.sidebar) {
    display: grid;
    grid-template-columns: 250px 1fr;
    gap: 30px;
}

/* 没有侧边栏的主容器 */
.main-container:not(:has(.sidebar)) .content {
    max-width: 800px;
    margin: 0 auto;
}

/* 包含多个section的article */
.article:has(.section:nth-child(3)) {
    column-count: 2;
    column-gap: 40px;
}
```

5. 购物车和电商应用：

代码块

```
/* 包含商品的购物车 */
.shopping-cart:has(.cart-item) {
    border: 2px solid #28a745;
    background-color: #f8fff9;
}

.shopping-cart:has(.cart-item) .empty-message {
    display: none;
}

/* 空购物车 */
.shopping-cart:not(:has(.cart-item)) .checkout-button {
    opacity: 0.5;
    pointer-events: none;
}

/* 包含折扣的商品 */
.product:has(.discount) {
```

```
    border: 2px solid #ff6b35;
    position: relative;
}

.product:has(.discount)::before {
    content: "SALE";
    position: absolute;
    top: 10px;
    right: 10px;
    background: #ff6b35;
    color: white;
    padding: 4px 8px;
    border-radius: 4px;
    font-size: 0.8rem;
}
```

复杂选择器组合：

1. 多条件选择：

代码块

```
/* 同时包含图片和视频的容器 */
.media-container:has(img):has(video) {
    display: grid;
    grid-template-columns: 1fr 1fr;
    gap: 20px;
}

/* 包含标题但不包含图片的文章 */
.article:has(h1):not(:has(img)) {
    text-align: center;
    max-width: 600px;
    margin: 0 auto;
}
```

2. 嵌套选择：

代码块

```
/* 选择包含active链接的导航容器的父元素 */
.header:has(.nav:has(.nav-link.active)) {
```

```
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

/* 选择包含错误表单字段的表单 */
.form:has(.form-field:has(.error:not(:empty))) {
    border-left: 4px solid #dc3545;
    padding-left: 16px;
}
```

3. 状态组合：

代码块

```
/* 包含焦点输入框的表单组 */
.form-group:has(input:focus) {
    background-color: #f8f9fa;
    border-radius: 4px;
    padding: 8px;
    margin: -8px;
}

/* 包含悬停按钮的卡片 */
.card:has(button:hover) {
    transform: translateY(-2px);
    box-shadow: 0 4px 8px rgba(0,0,0,0.15);
    transition: all 0.2s ease;
}
```

响应式应用：

代码块

```
/* 在大屏幕上，包含多个子项的容器使用网格布局 */
@media (min-width: 768px) {
    .container:has(.item:nth-child(4)) {
        display: grid;
        grid-template-columns: repeat(2, 1fr);
        gap: 20px;
    }
}

```

```
@media (min-width: 1200px) {
    .container:has(.item:nth-child(7)) {
        grid-template-columns: repeat(3, 1fr);
    }
}
```

性能考虑：

代码块

```
/* 避免过于复杂的选择器 */
/* 不推荐 */
.container:has(.item:has(.content:has(.title:has(.icon)))) {
    /* 过于复杂，影响性能 */
}

/* 推荐 */
.container:has(.item.has-icon) {
    /* 使用类名简化选择器 */
}
```

JavaScript配合使用：

代码块

```
// 检测:has()支持
function supportsHas() {
    try {
        return CSS.supports('selector(:has(div))');
    } catch {
        return false;
    }
}

if (!supportsHas()) {
    // 提供JavaScript回退方案
    document.querySelectorAll('.card').forEach(card => {
        if (card.querySelector('img')) {
            card.classList.add('has-image');
        }
```

```
    });
}

// 动态更新基于:has()的样式
function updateCardStates() {
    document.querySelectorAll('.card').forEach(card => {
        const hasImage = card.querySelector('img');
        const hasVideo = card.querySelector('video');

        card.classList.toggle('has-media', hasImage || hasVideo);
        card.classList.toggle('text-only', !hasImage && !hasVideo);
    });
}
```

浏览器兼容性：

- Chrome 105+
- Firefox 121+
- Safari 15.4+
- IE不支持

Polyfill方案：

代码块

```
// 简单的:has()polyfill概念
function hasPolyfill() {
    if (!CSS.supports('selector(:has(div))')) {
        const style = document.createElement('style');
        document.head.appendChild(style);

        // 模拟:has()行为
        document.querySelectorAll('[data-has]').forEach(element => {
            const selector = element.dataset.has;
            if (element.querySelector(selector)) {
                element.classList.add('has-match');
            }
        });
    }
}
```

最佳实践：

- 避免过度复杂的嵌套选择器
- 优先使用直接子选择器(>)提高性能
- 结合类名使用，避免纯结构依赖
- 为不支持的浏览器提供回退方案
- 在组件设计中充分利用:has()的能力
- 注意选择器的性能影响，避免过于复杂的查询

---

41. CSS中的@layer规则如何管理样式层叠？

参考答案：

@layer规则定义：

@layer规则允许开发者明确定义CSS的层叠顺序，创建命名的层级，从而更好地控制样式的优先级和组织结构。

基本语法：

代码块

```
/* 声明层级顺序 */
@layer reset, base, components, utilities;

/* 在层级中定义样式 */
@layer base {
    body {
        font-family: Arial, sans-serif;
        line-height: 1.6;
    }
}
```

层级声明方式：

1. 预先声明层级顺序：

代码块

```
/* 声明层级的优先级顺序（从低到高） */
@layer reset, normalize, base, layout, components, utilities, overrides;
```

2. 直接在层级中定义样式：

代码块

```
@layer reset {
    * {
        margin: 0;
        padding: 0;
        box-sizing: border-box;
    }
}

@layer base {
    body {
        font-family: 'Inter', sans-serif;
        color: #333;
        background: #fff;
    }

    h1, h2, h3 {
        margin-bottom: 1rem;
    }
}
```

3. 嵌套层级：

代码块

```
@layer framework {
    @layer reset {
        * { margin: 0; padding: 0; }
    }

```

```
    @layer base {
        body { font-family: Arial, sans-serif; }
    }

    @layer components {
        .button { padding: 8px 16px; }
    }
}
```

实际应用场景：

1. 设计系统架构：

代码块

```
/* 声明设计系统的层级结构 */
@layer reset, tokens, base, layout, components, utilities, overrides;

/* Reset层 - 最低优先级 */
@layer reset {
    *, *::before, *::after {
        box-sizing: border-box;
        margin: 0;
        padding: 0;
    }

    html {
        -webkit-text-size-adjust: 100%;
    }

    body {
        line-height: 1.5;
    }
}

/* 设计令牌层 */
@layer tokens {
    :root {
        --color-primary: #007bff;
        --color-secondary: #6c757d;
        --spacing-sm: 0.5rem;
        --spacing-md: 1rem;
        --spacing-lg: 1.5rem;
        --border-radius: 0.375rem;
```

```
    }
}

/* 基础样式层 */
@layer base {
    body {
        font-family: system-ui, -apple-system, sans-serif;
        color: var(--color-text);
        background: var(--color-background);
    }

    h1, h2, h3, h4, h5, h6 {
        font-weight: 600;
        line-height: 1.25;
        margin-bottom: var(--spacing-sm);
    }

    a {
        color: var(--color-primary);
        text-decoration: none;
    }

    a:hover {
        text-decoration: underline;
    }
}
```

2. 组件库管理：

代码块

```
/* 组件库的层级结构 */
@layer reset, base, layout, components, utilities;

/* 布局组件 */
@layer layout {
    .container {
        max-width: 1200px;
        margin: 0 auto;
        padding: 0 var(--spacing-md);
    }

    .grid {
        display: grid;
        gap: var(--spacing-md);
```

```
    }

    .flex {
        display: flex;
        gap: var(--spacing-sm);
    }
}

/* UI组件 */
@layer components {
    .button {
        display: inline-flex;
        align-items: center;
        justify-content: center;
        padding: var(--spacing-sm) var(--spacing-md);
        border: 1px solid transparent;
        border-radius: var(--border-radius);
        font-size: 0.875rem;
        font-weight: 500;
        text-decoration: none;
        cursor: pointer;
        transition: all 0.2s ease;
    }

    .button--primary {
        background: var(--color-primary);
        color: white;
        border-color: var(--color-primary);
    }

    .button--secondary {
        background: transparent;
        color: var(--color-primary);
        border-color: var(--color-primary);
    }

    .card {
        background: white;
        border: 1px solid #e5e7eb;
        border-radius: var(--border-radius);
        padding: var(--spacing-lg);
        box-shadow: 0 1px 3px rgba(0, 0, 0, 0.1);
    }

    .card__header {
        margin-bottom: var(--spacing-md);
        padding-bottom: var(--spacing-sm);
```

```
            border-bottom: 1px solid #e5e7eb;
        }

        .card__title {
            font-size: 1.125rem;
            font-weight: 600;
            margin-bottom: 0;
        }
    }

    /* 工具类 */
    @layer utilities {
        .text-center { text-align: center; }
        .text-right { text-align: right; }
        .font-bold { font-weight: 700; }
        .hidden { display: none; }
        .sr-only {
            position: absolute;
            width: 1px;
            height: 1px;
            padding: 0;
            margin: -1px;
            overflow: hidden;
            clip: rect(0, 0, 0, 0);
            white-space: nowrap;
            border: 0;
        }
    }
```

3. 第三方库集成：

代码块

```
/* 主应用样式层级 */
@layer reset, vendor, base, components, pages, utilities;

/* 第三方库样式 */
@layer vendor {
    /* 引入第三方CSS库 */
    @import url('bootstrap.css');
    @import url('prism.css');
}

/* 覆盖第三方样式 */
@layer components {
```

```
    /* 自定义Bootstrap按钮样式 */
    .btn-custom {
        background: linear-gradient(45deg, #ff6b6b, #4ecdc4);
        border: none;
        color: white;
    }

    /* 自定义代码高亮样式 */
    .code-block {
        border-radius: 8px;
        overflow: hidden;
    }
}
```

4. 主题系统：

代码块

```
@layer reset, themes, base, components, utilities;

/* 主题层 */
@layer themes {
    /* 默认主题 */
    :root {
        --theme-bg: #ffffff;
        --theme-text: #1a1a1a;
        --theme-primary: #007bff;
        --theme-border: #e5e7eb;
    }

    /* 暗色主题 */
    [data-theme="dark"] {
        --theme-bg: #1a1a1a;
        --theme-text: #ffffff;
        --theme-primary: #4dabf7;
        --theme-border: #374151;
    }

    /* 高对比度主题 */
    [data-theme="high-contrast"] {
        --theme-bg: #000000;
        --theme-text: #ffffff;
        --theme-primary: #ffff00;
        --theme-border: #ffffff;
    }
```

```
}

@layer base {
    body {
        background: var(--theme-bg);
        color: var(--theme-text);
        transition: background-color 0.2s, color 0.2s;
    }
}

@layer components {
    .card {
        background: var(--theme-bg);
        border-color: var(--theme-border);
        color: var(--theme-text);
    }

    .button--primary {
        background: var(--theme-primary);
    }
}
```

层级优先级规则：

代码块

```
/* 层级顺序决定优先级 */
@layer A, B, C;

@layer A {
    .element { color: red; }
}

@layer B {
    .element { color: blue; } /* 优先级高于层级A */
}

@layer C {
    .element { color: green; } /* 优先级最高 */
}

/* 无层级的样式优先级最高 */
.element { color: purple; } /* 优先级高于所有层级 */
```

匿名层级：

代码块

```
/* 匿名层级，按出现顺序排列优先级 */
@layer {
    .element { color: red; }
}

@layer {
    .element { color: blue; } /* 优先级更高 */
}

/* 命名层级可以在匿名层级之间插入 */
@layer named {
    .element { color: green; }
}
```

条件层级：

代码块

```
/* 结合媒体查询 */
@media (max-width: 768px) {
    @layer mobile {
        .container {
            padding: var(--spacing-sm);
        }

        .button {
            width: 100%;
        }
    }
}

/* 结合特性查询 */
@supports (display: grid) {
    @layer modern {
        .layout {
```

```
            display: grid;
            grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
        }
    }
}
```

JavaScript交互：

代码块

```
// 检测@layer支持
function supportsLayers() {
    return CSS.supports('@layer', 'base');
}

// 动态添加层级样式
function addLayerStyles(layerName, styles) {
    if (supportsLayers()) {
        const styleSheet = new CSSStyleSheet();
        styleSheet.insertRule(`@layer ${layerName} { ${styles} }`);
        document.adoptedStyleSheets = [...document.adoptedStyleSheets,
styleSheet];
    } else {
        // 回退方案
        const style = document.createElement('style');
        style.textContent = styles;
        document.head.appendChild(style);
    }
}

// 使用示例
addLayerStyles('dynamic', `
    .dynamic-component {
        background: #f0f0f0;
        padding: 1rem;
        border-radius: 4px;
    }
`);
```

调试和开发工具：

代码块

```
/* 开发时显示层级信息 */
@layer debug {
    [data-debug="true"] * {
        position: relative;
    }

    [data-debug="true"] *::before {
        content: attr(class);
        position: absolute;
        top: 0;
        left: 0;
        font-size: 10px;
        background: rgba(255, 0, 0, 0.8);
        color: white;
        padding: 2px 4px;
        pointer-events: none;
        z-index: 9999;
    }
}
```

最佳实践：

代码块

```
/* 推荐的层级结构 */
@layer
    reset,           /* CSS重置 */
    normalize,       /* 标准化样式 */
    tokens,          /* 设计令牌 */
    base,           /* 基础元素样式 */
    layout,         /* 布局组件 */
    components,     /* UI组件 */
    patterns,       /* 复合组件 */
    utilities,      /* 工具类 */
    overrides;      /* 特殊覆盖 */

/* 保持层级内容聚焦 */
@layer components {
    /* 只包含组件相关样式 */
    .button { /* ... */ }
```

```
    .card { /* ... */ }
    .modal { /* ... */ }
}

/* 避免跨层级依赖 */
@layer base {
    /* 不要依赖components层的样式 */
    body { font-family: Arial, sans-serif; }
}
```

浏览器兼容性：

- Chrome 99+
- Firefox 97+
- Safari 15.4+
- IE不支持

回退策略：

代码块

```
/* 不支持@layer时的回退 */
@supports not at-rule(@layer) {
    /* 使用传统的特异性管理 */
    .reset-styles { /* 低特异性 */ }
    .component-styles { /* 中等特异性 */ }
    .utility-styles { /* 高特异性 */ }
}
```

优势总结：

- 明确的样式层级管理
- 更好的团队协作
- 减少特异性冲突
- 更容易维护和调试
- 支持大型项目的CSS架构
- 与现有工具和框架良好集成

42. CSS中的subgrid如何实现嵌套网格布局？

参考答案：

subgrid定义：

CSS Subgrid是CSS Grid Layout的扩展，允许网格项目继承其父网格的行或列轨道，实现更复杂和一致的嵌套网格布局。

基本语法：

代码块

```
.parent-grid {
    display: grid;
    grid-template-columns: repeat(4, 1fr);
    grid-template-rows: repeat(3, 100px);
}

.child-grid {
    display: grid;
    grid-column: 2 / 4;  /* 占据父网格的列2-4 */
    grid-row: 1 / 3;     /* 占据父网格的行1-3 */

    /* 继承父网格的列轨道 */
    grid-template-columns: subgrid;
    /* 继承父网格的行轨道 */
    grid-template-rows: subgrid;
}
```

基本用法示例：

1. 继承列轨道：

代码块

```css
.main-grid {
    display: grid;
    grid-template-columns: 200px 1fr 200px;
    gap: 20px;
}

.content-section {
    grid-column: 1 / -1; /* 跨越所有列 */
    display: grid;
    grid-template-columns: subgrid; /* 继承父网格的3列 */
}

.content-section .item {
    /* 这些项目会自动对齐到父网格的列 */
}
```

2. 继承行轨道：

代码块

```css
.layout-grid {
    display: grid;
    grid-template-rows: auto 1fr auto;
    min-height: 100vh;
}

.main-content {
    display: grid;
    grid-template-rows: subgrid; /* 继承父网格的行结构 */
    grid-row: 1 / -1; /* 跨越所有行 */
}
```

实际应用场景：

1. 卡片网格对齐：

代码块

```
<div class="card-grid">
    <div class="card">
        <img src="image1.jpg" alt="Image 1">
        <h3>Short Title</h3>
```

```
        <p>Brief description</p>
        <button>Read More</button>
    </div>
    <div class="card">
        <img src="image2.jpg" alt="Image 2">
        <h3>This is a Much Longer Title That Spans Multiple Lines</h3>
        <p>This is a longer description that provides more detail about the content</p>
        <button>Read More</button>
    </div>
    <div class="card">
        <img src="image3.jpg" alt="Image 3">
        <h3>Medium Length Title</h3>
        <p>Standard description length</p>
        <button>Read More</button>
    </div>
</div>
```

代码块

```
.card-grid {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
    gap: 20px;
    /* 定义隐式行，让所有卡片内容对齐 */
    grid-template-rows: repeat(auto-fit, auto);
}

.card {
    display: grid;
    grid-template-rows: subgrid; /* 继承父网格的行结构 */
    grid-row: span 4; /* 每个卡片占据4行 */

    background: white;
    border-radius: 8px;
    padding: 20px;
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

.card img {
    width: 100%;
    height: 200px;
    object-fit: cover;
    border-radius: 4px;
```

```
}

.card h3 {
    margin: 15px 0 10px 0;
    /* 标题会自动对齐到同一行 */
}

.card p {
    margin-bottom: 15px;
    flex-grow: 1; /* 描述区域填充可用空间 */
}

.card button {
    margin-top: auto; /* 按钮始终在底部对齐 */
    padding: 8px 16px;
    background: #007bff;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}
```

2. 表单布局对齐：

代码块

```
<form class="form-grid">
    <div class="form-section">
        <label for="firstName">First Name</label>
        <input type="text" id="firstName" required>
        <span class="error-message">This field is required</span>
    </div>
    <div class="form-section">
        <label for="lastName">Last Name</label>
        <input type="text" id="lastName" required>
        <span class="error-message"></span>
    </div>
    <div class="form-section">
        <label for="email">Email Address</label>
        <input type="email" id="email" required>
        <span class="error-message">Please enter a valid email</span>
    </div>
</form>
```

代码块

```
.form-grid {
    display: grid;
    grid-template-columns: repeat(2, 1fr);
    gap: 20px;
    /* 定义每个表单字段的行结构 */
    grid-template-rows: repeat(auto-fit, auto auto auto);
}

.form-section {
    display: grid;
    grid-template-rows: subgrid; /* 继承父网格的行结构 */
    grid-row: span 3; /* 每个section占据3行：label, input, error */
    gap: 5px;
}

.form-section label {
    font-weight: 500;
    color: #333;
    /* 所有标签都对齐到第一行 */
}

.form-section input {
    padding: 8px 12px;
    border: 1px solid #ccc;
    border-radius: 4px;
    /* 所有输入框都对齐到第二行 */
}

.form-section .error-message {
    font-size: 0.875rem;
    color: #dc3545;
    min-height: 1.2em; /* 确保即使为空也占据空间 */
    /* 所有错误信息都对齐到第三行 */
}
```

3. 复杂页面布局：

代码块

```
.page-layout {
    display: grid;
```

```
    grid-template-columns: 250px 1fr 200px;
    grid-template-rows: 60px 1fr 40px;
    grid-template-areas:
        "header header header"
        "sidebar main aside"
        "footer footer footer";
    min-height: 100vh;
    gap: 20px;
}

.main-content {
    grid-area: main;
    display: grid;
    grid-template-columns: subgrid; /* 继承主网格的列结构 */
    grid-template-rows: subgrid;    /* 继承主网格的行结构 */
    gap: inherit; /* 继承父网格的间距 */
}

.article-grid {
    display: grid;
    grid-template-columns: subgrid;
    grid-column: 1 / -1; /* 跨越所有可用列 */
    gap: 15px;
}

.article {
    background: white;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}
```

4. 数据表格对齐：

代码块

```
.data-grid {
    display: grid;
    grid-template-columns: 100px 1fr 120px 80px;
    gap: 1px;
    background: #e5e7eb; /* 网格线颜色 */
}

.table-header,
.table-row {
```

```
    display: grid;
    grid-template-columns: subgrid; /* 继承父网格的列结构 */
    grid-column: 1 / -1; /* 跨越所有列 */
    background: white;
}

.table-header {
    font-weight: 600;
    background: #f3f4f6;
}

.table-cell {
    padding: 12px 16px;
    border-right: 1px solid #e5e7eb;
}

.table-cell:last-child {
    border-right: none;
}

/* 响应式调整 */
@media (max-width: 768px) {
    .data-grid {
        grid-template-columns: 1fr 100px;
    }

    .table-cell:nth-child(3),
    .table-cell:nth-child(4) {
        display: none;
    }
}
```

命名网格线的继承：

代码块

```
.parent-grid {
    display: grid;
    grid-template-columns:
        [start] 200px
        [content-start] 1fr
        [content-end] 200px
        [end];
```

```
    grid-template-rows:
        [header-start] 60px
        [main-start] 1fr
        [main-end] 40px
        [footer-end];
}

.content-area {
    grid-column: content-start / content-end;
    grid-row: main-start / main-end;

    display: grid;
    grid-template-columns: subgrid; /* 继承命名的网格线 */
    grid-template-rows: subgrid;
}

.nested-item {
    /* 可以使用继承的网格线名称 */
    grid-column: content-start;
    grid-row: main-start;
}
```

gap的继承和覆盖：

代码块

```
.parent-grid {
    display: grid;
    grid-template-columns: repeat(4, 1fr);
    gap: 20px;
}

.subgrid-container {
    grid-column: 1 / -1;
    display: grid;
    grid-template-columns: subgrid;
    gap: inherit; /* 继承父网格的gap */
}

.custom-gap-subgrid {
    grid-column: 2 / 4;
    display: grid;
    grid-template-columns: subgrid;
```

```
    gap: 10px; /* 覆盖父网格的gap */
}
```

与其他CSS特性的结合：

1. 与容器查询结合：

代码块

```
.responsive-subgrid {
    container-type: inline-size;
    display: grid;
    grid-template-columns: repeat(4, 1fr);
    gap: 20px;
}

.adaptive-section {
    display: grid;
    grid-template-columns: subgrid;
    grid-column: 1 / -1;
}

@container (max-width: 600px) {
    .adaptive-section {
        grid-template-columns: 1fr; /* 在小容器中不使用subgrid */
    }
}
```

2. 与CSS动画结合：

代码块

```
.animated-subgrid {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    transition: grid-template-columns 0.3s ease;
}

.animated-subgrid:hover {
    grid-template-columns: 2fr 1fr 1fr;
}

```

```
.subgrid-item {
    display: grid;
    grid-template-columns: subgrid;
    grid-column: 1 / -1;
    /* 子网格会自动适应父网格的动画变化 */
}
```

调试和开发工具：

代码块

```
/* 开发时显示网格线 */
.debug-grid {
    background-image:
        linear-gradient(rgba(255,0,0,0.1) 1px, transparent 1px),
        linear-gradient(90deg, rgba(255,0,0,0.1) 1px, transparent 1px);
    background-size: 20px 20px;
}

.debug-subgrid {
    background-image:
        linear-gradient(rgba(0,255,0,0.1) 1px, transparent 1px),
        linear-gradient(90deg, rgba(0,255,0,0.1) 1px, transparent 1px);
    background-size: inherit;
}
```

浏览器兼容性：

- Firefox 71+
- Chrome/Safari: 尚未支持（截至2024年）
- IE不支持

回退方案：

代码块

```
.grid-container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
```

```
    gap: 20px;
}

.grid-item {
    display: grid;
    /* 回退：不使用subgrid */
    grid-template-columns: 1fr;
    gap: 10px;
}

/* 支持subgrid时使用 */
@supports (grid-template-columns: subgrid) {
    .grid-item {
        grid-template-columns: subgrid;
        grid-column: 1 / -1;
        gap: inherit;
    }
}
```

最佳实践：

- 在需要对齐嵌套网格内容时使用subgrid
- 合理使用gap的继承和覆盖
- 为不支持的浏览器提供回退方案
- 结合命名网格线提高代码可读性
- 避免过深的subgrid嵌套
- 在复杂布局中优先考虑subgrid解决方案

---

43. CSS中的accent-color属性如何自定义表单控件样式？

参考答案：

accent-color定义：

accent-color属性允许开发者自定义表单控件的强调色，包括复选框、单选按钮、范围滑块、进度条等元素的主题颜色。

基本语法：

```
代码块
.element {
    accent-color: <color> | auto;
}
```

支持的表单控件：

1. 复选框（checkbox）：

```
代码块
input[type="checkbox"] {
    accent-color: #007bff;
    width: 20px;
    height: 20px;
}

.custom-checkbox {
    accent-color: #28a745;
}

.danger-checkbox {
    accent-color: #dc3545;
}
```

2. 单选按钮（radio）：

```
代码块
input[type="radio"] {
    accent-color: #6f42c1;
    width: 18px;
    height: 18px;
}
```

```
6
7   .radio-group input[type="radio"] {
8       accent-color: #fd7e14;
9       margin-right: 8px;
10  }
```

3. 范围滑块（range）：

代码块

```
1   input[type="range"] {
2       accent-color: #e83e8c;
3       width: 100%;
4       height: 8px;
5   }
6
7   .volume-slider {
8       accent-color: #20c997;
9   }
10
11  .brightness-slider {
12      accent-color: #ffc107;
13  }
```

4. 进度条（progress）：

代码块

```
1   progress {
2       accent-color: #17a2b8;
3       width: 100%;
4       height: 20px;
5   }
6
7   .upload-progress {
8       accent-color: #28a745;
9   }
10
11  .loading-progress {
12      accent-color: #6c757d;
13  }
```

实际应用场景：

1. 主题化表单：

代码块

```
<form class="themed-form">
    <div class="form-group">
        <label>
            <input type="checkbox" name="terms">
            I agree to the terms and conditions
        </label>
    </div>

    <div class="form-group">
        <label>Choose your plan:</label>
        <label><input type="radio" name="plan" value="basic"> Basic</label>
        <label><input type="radio" name="plan" value="pro"> Pro</label>
        <label><input type="radio" name="plan" value="enterprise">
Enterprise</label>
    </div>

    <div class="form-group">
        <label for="volume">Volume:</label>
        <input type="range" id="volume" min="0" max="100" value="50">
    </div>

    <div class="form-group">
        <label for="progress">Upload Progress:</label>
        <progress id="progress" value="75" max="100">75%</progress>
    </div>
</form>
```

代码块

```
:root {
    --primary-color: #007bff;
    --success-color: #28a745;
    --warning-color: #ffc107;
    --danger-color: #dc3545;
}

```

```css
.themed-form {
    accent-color: var(--primary-color);
    padding: 20px;
    background: #f8f9fa;
    border-radius: 8px;
}

.form-group {
    margin-bottom: 20px;
}

.form-group label {
    display: block;
    margin-bottom: 5px;
    font-weight: 500;
}

.form-group input[type="radio"] + label,
.form-group input[type="checkbox"] + label {
    display: inline;
    margin-left: 8px;
    font-weight: normal;
}

/* 特定控件的自定义颜色 */
input[name="terms"] {
    accent-color: var(--success-color);
}

input[name="plan"] {
    accent-color: var(--primary-color);
}

#volume {
    accent-color: var(--warning-color);
}

#progress {
    accent-color: var(--success-color);
}
```

2. 品牌色彩系统：

```
代码块
```

```
/* 品牌主色调 */
.brand-primary {
    accent-color: #ff6b35;
}

.brand-secondary {
    accent-color: #4ecdc4;
}

.brand-accent {
    accent-color: #45b7d1;
}

/* 状态颜色 */
.status-success {
    accent-color: #96ceb4;
}

.status-warning {
    accent-color: #feca57;
}

.status-error {
    accent-color: #ff6b6b;
}

/* 应用到不同表单控件 */
.settings-form .brand-primary {
    /* 复选框和单选按钮使用品牌主色 */
}

.preferences-form .brand-secondary {
    /* 滑块使用品牌次色 */
}

.upload-form .status-success {
    /* 进度条使用成功色 */
}
```

3. 暗色主题适配：

代码块

```
/* 浅色主题 */
:root {
```

```
    --accent-primary: #007bff;
    --accent-secondary: #6c757d;
    --accent-success: #28a745;
}

.form-controls {
    accent-color: var(--accent-primary);
}

/* 暗色主题 */
@media (prefers-color-scheme: dark) {
    :root {
        --accent-primary: #4dabf7;
        --accent-secondary: #adb5bd;
        --accent-success: #51cf66;
    }
}

/* 手动暗色主题切换 */
[data-theme="dark"] {
    --accent-primary: #4dabf7;
    --accent-secondary: #adb5bd;
    --accent-success: #51cf66;
}

[data-theme="dark"] .form-controls {
    accent-color: var(--accent-primary);
}
```

4. 交互状态增强：

代码块

```
.interactive-controls input[type="checkbox"],
.interactive-controls input[type="radio"] {
    accent-color: #6c757d;
    transition: accent-color 0.2s ease;
    transform: scale(1);
    transition: accent-color 0.2s ease, transform 0.1s ease;
}

.interactive-controls input[type="checkbox"]:hover,
.interactive-controls input[type="radio"]:hover {
    accent-color: #007bff;
    transform: scale(1.1);
```

```
}

.interactive-controls input[type="checkbox"]:focus,
.interactive-controls input[type="radio"]:focus {
    accent-color: #0056b3;
    outline: 2px solid rgba(0, 123, 255, 0.25);
    outline-offset: 2px;
}

.interactive-controls input[type="range"] {
    accent-color: #6c757d;
    transition: accent-color 0.2s ease;
}

.interactive-controls input[type="range"]:hover {
    accent-color: #007bff;
}

.interactive-controls input[type="range"]:active {
    accent-color: #0056b3;
}
```

5. 动态颜色变化：

代码块

```
<div class="color-picker-demo">
    <input type="color" id="colorPicker" value="#007bff">
    <div class="demo-controls">
        <label><input type="checkbox" class="demo-checkbox"> Checkbox</label>
        <label><input type="radio" name="demo" class="demo-radio"> Radio 1</label>
        <label><input type="radio" name="demo" class="demo-radio"> Radio 2</label>
        <input type="range" class="demo-range" min="0" max="100" value="50">
        <progress class="demo-progress" value="60" max="100">60%</progress>
    </div>
</div>
```

代码块

```
.demo-controls {
```

```
    margin-top: 20px;
}

.demo-controls > * {
    margin: 10px 0;
    display: block;
}

/* JavaScript会动态更新这个颜色 */
.demo-checkbox,
.demo-radio,
.demo-range,
.demo-progress {
    accent-color: #007bff;
    transition: accent-color 0.3s ease;
}
```

代码块

```
const colorPicker = document.getElementById('colorPicker');
const demoControls = document.querySelectorAll('.demo-checkbox, .demo-radio, .demo-range, .demo-progress');

colorPicker.addEventListener('input', (e) => {
    const selectedColor = e.target.value;
    demoControls.forEach(control => {
        control.style.accentColor = selectedColor;
    });
});
```

与CSS变量结合：

代码块

```
:root {
    --form-accent: #007bff;
    --form-accent-hover: #0056b3;
    --form-accent-focus: #004085;
}

.form-container {
```

```
    accent-color: var(--form-accent);
}

/* 基于用户偏好动态调整 */
@media (prefers-contrast: high) {
    :root {
        --form-accent: #0000ff;
        --form-accent-hover: #0000cc;
        --form-accent-focus: #000099;
    }
}

/* 基于颜色方案调整 */
@media (prefers-color-scheme: dark) {
    :root {
        --form-accent: #66b3ff;
        --form-accent-hover: #4da6ff;
        --form-accent-focus: #3399ff;
    }
}
```

可访问性考虑：

代码块

```
/* 确保足够的对比度 */
.accessible-form {
    accent-color: #0066cc; /* WCAG AA标准对比度 */
}

/* 高对比度模式 */
@media (prefers-contrast: high) {
    .accessible-form {
        accent-color: #000080; /* 更高对比度 */
    }
}

/* 减少动画偏好 */
@media (prefers-reduced-motion: reduce) {
    .form-controls input {
        transition: none;
    }
}
```

```
19
20  /* 焦点指示器增强 */
21  .form-controls input:focus-visible {
22      outline: 2px solid currentColor;
23      outline-offset: 2px;
24  }
```

自定义表单组件：

代码块

```
1   .custom-form-group {
2       position: relative;
3       margin-bottom: 20px;
4   }
5
6   .custom-checkbox {
7       accent-color: #e91e63;
8       width: 18px;
9       height: 18px;
10      margin-right: 10px;
11  }
12
13  .custom-checkbox:checked + .custom-label::after {
14      content: "✓";
15      position: absolute;
16      left: 4px;
17      top: 0;
18      color: white;
19      font-size: 12px;
20      font-weight: bold;
21  }
22
23  .custom-radio {
24      accent-color: #9c27b0;
25      margin-right: 8px;
26  }
27
28  .custom-range {
29      accent-color: #ff5722;
30      height: 6px;
31      border-radius: 3px;
32      background: #e0e0e0;
```

```
}

.custom-progress {
    accent-color: #4caf50;
    height: 12px;
    border-radius: 6px;
    background: #f5f5f5;
    border: 1px solid #e0e0e0;
}
```

浏览器兼容性：

- Chrome 93+
- Firefox 92+
- Safari 15.4+
- IE不支持

回退方案：

代码块

```
/* 不支持accent-color的回退 */
@supports not (accent-color: red) {
    /* 使用传统的自定义样式方法 */
    input[type="checkbox"] {
        appearance: none;
        width: 18px;
        height: 18px;
        border: 2px solid #007bff;
        border-radius: 3px;
        background: white;
        position: relative;
    }

    input[type="checkbox"]:checked {
        background: #007bff;
    }

    input[type="checkbox"]:checked::after {
        content: "✓";
        position: absolute;
        top: -2px;
        left: 2px;
```

```
        color: white;
        font-size: 12px;
    }
}
```

最佳实践：

- 确保accent-color与品牌色彩保持一致
- 考虑不同主题和用户偏好
- 提供足够的颜色对比度
- 为不支持的浏览器提供回退方案
- 结合CSS变量实现动态主题切换
- 测试各种表单控件的视觉效果
- 保持整个应用的颜色一致性

---

44. CSS中的@property规则如何定义自定义属性？

参考答案：

@property规则定义：

@property规则允许开发者明确定义CSS自定义属性（CSS变量）的语法、初始值、是否继承等特性，使自定义属性能够参与动画和过渡，并提供更好的类型安全性。

基本语法：

代码块

```
@property --property-name {
    syntax: '<color>';
    initial-value: #000;
    inherits: false;
}
```

语法组成部分：

1. syntax（语法定义）：

代码块

```
@property --my-color {
    syntax: '<color>'; /* 颜色类型 */
    initial-value: red;
    inherits: false;
}

@property --my-length {
    syntax: '<length>'; /* 长度类型 */
    initial-value: 0px;
    inherits: false;
}

@property --my-number {
    syntax: '<number>'; /* 数字类型 */
    initial-value: 0;
    inherits: false;
}

@property --my-percentage {
    syntax: '<percentage>'; /* 百分比类型 */
    initial-value: 0%;
    inherits: false;
}

@property --my-angle {
    syntax: '<angle>'; /* 角度类型 */
    initial-value: 0deg;
    inherits: false;
}
```

2. 复合语法类型：

代码块

```
@property --gradient-colors {
    syntax: '<color>+'; /* 一个或多个颜色 */
```

```
    initial-value: red, blue;
    inherits: false;
}

@property --border-style {
    syntax: 'solid | dashed | dotted'; /* 枚举值 */
    initial-value: solid;
    inherits: false;
}

@property --transform-values {
    syntax: '<number> | <length> | <percentage>'; /* 多种类型 */
    initial-value: 0;
    inherits: false;
}

@property --any-value {
    syntax: '*'; /* 任意值 */
    initial-value: initial;
    inherits: true;
}
```

实际应用场景：

1. 可动画的颜色属性：

代码块

```
@property --theme-color {
    syntax: '<color>';
    initial-value: #007bff;
    inherits: false;
}

@property --gradient-start {
    syntax: '<color>';
    initial-value: #ff6b6b;
    inherits: false;
}

@property --gradient-end {
    syntax: '<color>';
    initial-value: #4ecdc4;
    inherits: false;
```

```
}

.animated-gradient {
    background: linear-gradient(45deg, var(--gradient-start), var(--gradient-end));
    transition: --gradient-start 0.5s ease, --gradient-end 0.5s ease;
}

.animated-gradient:hover {
    --gradient-start: #ff9ff3;
    --gradient-end: #54a0ff;
}
```

2. 可动画的数值属性：

代码块

```
@property --rotation {
    syntax: '<angle>';
    initial-value: 0deg;
    inherits: false;
}

@property --scale {
    syntax: '<number>';
    initial-value: 1;
    inherits: false;
}

@property --blur-radius {
    syntax: '<length>';
    initial-value: 0px;
    inherits: false;
}

.transformable-element {
    transform: rotate(var(--rotation)) scale(var(--scale));
    filter: blur(var(--blur-radius));
    transition: --rotation 0.3s ease, --scale 0.3s ease, --blur-radius 0.3s ease;
}

.transformable-element:hover {
    --rotation: 180deg;
    --scale: 1.2;
```

```
    --blur-radius: 2px;
}
```

3. 渐变动画：

代码块

```
@property --gradient-angle {
    syntax: '<angle>';
    initial-value: 0deg;
    inherits: false;
}

@property --gradient-position {
    syntax: '<percentage>';
    initial-value: 0%;
    inherits: false;
}

.animated-background {
    background: conic-gradient(
        from var(--gradient-angle),
        #ff6b6b var(--gradient-position),
        #4ecdc4 calc(var(--gradient-position) + 50%),
        #45b7d1 calc(var(--gradient-position) + 100%)
    );
    animation: rotateGradient 3s linear infinite;
}

@keyframes rotateGradient {
    from {
        --gradient-angle: 0deg;
        --gradient-position: 0%;
    }
    to {
        --gradient-angle: 360deg;
        --gradient-position: 100%;
    }
}
```

4. 主题系统：

```css
@property --primary-hue {
    syntax: '<number>';
    initial-value: 210; /* 蓝色色相 */
    inherits: true;
}

@property --primary-saturation {
    syntax: '<percentage>';
    initial-value: 100%;
    inherits: true;
}

@property --primary-lightness {
    syntax: '<percentage>';
    initial-value: 50%;
    inherits: true;
}

:root {
    --primary-color: hsl(var(--primary-hue), var(--primary-saturation), var(--primary-lightness));
    --primary-light: hsl(var(--primary-hue), var(--primary-saturation), 75%);
    --primary-dark: hsl(var(--primary-hue), var(--primary-saturation), 25%);
}

.theme-switcher {
    transition: --primary-hue 0.5s ease, --primary-saturation 0.5s ease, --primary-lightness 0.5s ease;
}

/* 主题变体 */
.theme-blue { --primary-hue: 210; }
.theme-green { --primary-hue: 120; }
.theme-red { --primary-hue: 0; }
.theme-purple { --primary-hue: 270; }

.theme-vibrant { --primary-saturation: 100%; }
.theme-muted { --primary-saturation: 50%; }

.theme-light { --primary-lightness: 70%; }
.theme-dark { --primary-lightness: 30%; }
```

5. 复杂动画效果：

```css
@property --wave-amplitude {
    syntax: '<length>';
    initial-value: 10px;
    inherits: false;
}

@property --wave-frequency {
    syntax: '<number>';
    initial-value: 1;
    inherits: false;
}

@property --wave-phase {
    syntax: '<angle>';
    initial-value: 0deg;
    inherits: false;
}

.wave-effect {
    position: relative;
    overflow: hidden;
}

.wave-effect::before {
    content: '';
    position: absolute;
    top: 0;
    left: 0;
    right: 0;
    height: 4px;
    background: linear-gradient(90deg,
        transparent,
        var(--primary-color),
        transparent
    );
    transform: translateY(
        calc(
            var(--wave-amplitude) *
            sin(var(--wave-phase) + var(--wave-frequency) * 360deg)
        )
    );
    animation: waveMotion 2s ease-in-out infinite;
}

@keyframes waveMotion {
    0% { --wave-phase: 0deg; }
    100% { --wave-phase: 360deg; }
```

```
}
```

6. 响应式数值：

代码块

```
@property --container-padding {
    syntax: '<length>';
    initial-value: 16px;
    inherits: false;
}

@property --grid-columns {
    syntax: '<number>';
    initial-value: 1;
    inherits: false;
}

.responsive-container {
    padding: var(--container-padding);
    display: grid;
    grid-template-columns: repeat(var(--grid-columns), 1fr);
    gap: calc(var(--container-padding) / 2);
    transition: --container-padding 0.3s ease, --grid-columns 0.3s ease;
}

@media (min-width: 768px) {
    .responsive-container {
        --container-padding: 24px;
        --grid-columns: 2;
    }
}

@media (min-width: 1200px) {
    .responsive-container {
        --container-padding: 32px;
        --grid-columns: 3;
    }
}
```

JavaScript交互：

代码块

```
// 检测@property支持
function supportsPropertyRule() {
    return CSS.supports('@property', '--test: 0');
}

// 动态注册自定义属性
function registerProperty(name, syntax, initialValue, inherits = false) {
    if ('registerProperty' in CSS) {
        try {
            CSS.registerProperty({
                name: name,
                syntax: syntax,
                initialValue: initialValue,
                inherits: inherits
            });
        } catch (e) {
            console.warn('Failed to register property:', name, e);
        }
    }
}

// 注册动画属性
registerProperty('--dynamic-color', '<color>', '#000000');
registerProperty('--dynamic-size', '<length>', '0px');
registerProperty('--dynamic-opacity', '<number>', '1');

// 动态更新属性值
function animateProperty(element, property, fromValue, toValue, duration =
1000) {
    if (!supportsPropertyRule()) {
        // 回退到传统动画
        return;
    }

    element.style.setProperty(property, fromValue);
    element.style.transition = `${property} ${duration}ms ease`;

    requestAnimationFrame(() => {
        element.style.setProperty(property, toValue);
    });
}

// 使用示例
const element = document.querySelector('.animated-element');
animateProperty(element, '--dynamic-color', '#ff0000', '#00ff00', 2000);
```

类型验证和错误处理：

代码块

```
@property --validated-number {
    syntax: '<number>';
    initial-value: 0;
    inherits: false;
}

.validated-element {
    /* 有效值 */
    --validated-number: 42;
    transform: scale(var(--validated-number));
}

.invalid-element {
    /* 无效值会回退到initial-value */
    --validated-number: "not a number";
    transform: scale(var(--validated-number)); /* 使用initial-value: 0 */
}
```

性能优化：

代码块

```
/* 避免过于复杂的语法 */
@property --simple-color {
    syntax: '<color>';
    initial-value: red;
    inherits: false;
}

/* 而不是 */
@property --complex-syntax {
    syntax: '<color> | <length> | <percentage> | <number>';
    initial-value: red;
    inherits: false;
}
```

```
14
15  /* 合理使用inherits */
16  @property --theme-base {
17      syntax: '<color>';
18      initial-value: #000;
19      inherits: true; /* 主题色应该继承 */
20  }
21
22  @property --animation-value {
23      syntax: '<number>';
24      initial-value: 0;
25      inherits: false; /* 动画值通常不需要继承 */
26  }
```

浏览器兼容性：

- Chrome 85+
- Firefox: 尚未支持
- Safari: 尚未支持
- IE不支持

Polyfill和回退方案：

代码块

```
1   // 简单的polyfill概念
2   if (!('registerProperty' in CSS)) {
3       // 为不支持的浏览器提供基础功能
4       window.CSS = window.CSS || {};
5       CSS.registerProperty = function(definition) {
6           // 基础实现，仅设置初始值
7           document.documentElement.style.setProperty(
8               definition.name,
9               definition.initialValue
10          );
11      };
12  }
```

最佳实践：

- 为可动画的数值使用@property
- 选择合适的syntax类型提高性能
- 合理设置inherits属性
- 提供有意义的initial-value
- 为不支持的浏览器提供回退方案
- 避免过于复杂的语法定义
- 在大型项目中统一管理自定义属性定义

---

45. CSS中的inert属性如何实现元素的惰性状态?

参考答案:

inert属性定义:

inert是一个HTML属性，当应用到元素上时，会使该元素及其所有后代元素变为"惰性"状态，即不可交互、不可聚焦，并且对辅助技术隐藏。CSS可以通过:inert伪类选择器来样式化这些惰性元素。

基本用法:

代码块

```
<!-- HTML中使用inert属性 -->
<div inert>
    <button>This button is inert</button>
    <input type="text" placeholder="This input is inert">
    <a href="#">This link is inert</a>
</div>
```

代码块

```
/* CSS中样式化惰性元素 */
:inert {
    opacity: 0.5;
```

```
    pointer-events: none;
    user-select: none;
}
```

inert的效果：

1. 元素不能接收焦点
2. 元素不响应用户交互（点击、键盘等）
3. 元素对屏幕阅读器等辅助技术隐藏
4. 元素的所有后代也会变为惰性

实际应用场景：

1. 模态框背景内容：

代码块

```
<div class="page-content" id="mainContent">
    <header>
        <h1>Main Page Content</h1>
        <nav>
            <a href="#home">Home</a>
            <a href="#about">About</a>
            <a href="#contact">Contact</a>
        </nav>
    </header>
    <main>
        <button id="openModal">Open Modal</button>
        <form>
            <input type="text" placeholder="Name">
            <input type="email" placeholder="Email">
            <button type="submit">Submit</button>
        </form>
    </main>
</div>

<div class="modal" id="modal" hidden>
    <div class="modal-content">
        <h2>Modal Dialog</h2>
        <p>This is a modal dialog.</p>
        <button id="closeModal">Close</button>
    </div>
```

```
</div>
```

代码块

```
/* 惰性内容的样式 */
:inert {
    opacity: 0.3;
    filter: blur(2px);
    pointer-events: none;
    user-select: none;
    transition: opacity 0.3s ease, filter 0.3s ease;
}

.modal {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: rgba(0, 0, 0, 0.5);
    display: flex;
    align-items: center;
    justify-content: center;
    z-index: 1000;
}

.modal[hidden] {
    display: none;
}

.modal-content {
    background: white;
    padding: 30px;
    border-radius: 8px;
    max-width: 500px;
    width: 90%;
}
```

代码块

```
const openModalBtn = document.getElementById('openModal');
const closeModalBtn = document.getElementById('closeModal');
```

```
const modal = document.getElementById('modal');
const mainContent = document.getElementById('mainContent');

openModalBtn.addEventListener('click', () => {
    modal.hidden = false;
    mainContent.inert = true; // 使背景内容变为惰性
});

closeModalBtn.addEventListener('click', () => {
    modal.hidden = true;
    mainContent.inert = false; // 恢复背景内容的交互性
});
```

2. 加载状态管理：

代码块

```
<div class="app-container">
    <div class="content" id="appContent">
        <h1>Application Content</h1>
        <form id="dataForm">
            <input type="text" placeholder="Enter data">
            <button type="submit">Submit</button>
        </form>
        <div class="data-list">
            <div class="data-item">Item 1</div>
            <div class="data-item">Item 2</div>
            <div class="data-item">Item 3</div>
        </div>
    </div>

    <div class="loading-overlay" id="loadingOverlay" hidden>
        <div class="spinner"></div>
        <p>Loading...</p>
    </div>
</div>
```

代码块

```
.app-container {
    position: relative;
    min-height: 100vh;
```

```
}

/* 加载时的惰性样式 */
:inert {
    opacity: 0.4;
    filter: grayscale(100%);
    transition: opacity 0.3s ease, filter 0.3s ease;
}

.loading-overlay {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: rgba(255, 255, 255, 0.8);
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    z-index: 100;
}

.loading-overlay[hidden] {
    display: none;
}

.spinner {
    width: 40px;
    height: 40px;
    border: 4px solid #f3f3f3;
    border-top: 4px solid #007bff;
    border-radius: 50%;
    animation: spin 1s linear infinite;
}

@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}
```

代码块

```
function showLoading() {
```

```
    const content = document.getElementById('appContent');
    const loading = document.getElementById('loadingOverlay');

    content.inert = true;
    loading.hidden = false;
}

function hideLoading() {
    const content = document.getElementById('appContent');
    const loading = document.getElementById('loadingOverlay');

    content.inert = false;
    loading.hidden = true;
}

// 模拟异步操作
document.getElementById('dataForm').addEventListener('submit', async (e) => {
    e.preventDefault();
    showLoading();

    // 模拟API调用
    await new Promise(resolve => setTimeout(resolve, 2000));

    hideLoading();
});
```

3. 分步表单：

代码块

```
<div class="multi-step-form">
    <div class="step-indicators">
        <div class="step active">1</div>
        <div class="step">2</div>
        <div class="step">3</div>
    </div>

    <div class="form-step active" id="step1">
        <h2>Step 1: Personal Information</h2>
        <input type="text" placeholder="First Name">
        <input type="text" placeholder="Last Name">
        <button onclick="nextStep(2)">Next</button>
    </div>

    <div class="form-step" id="step2" inert>
```

```
16
```