媒体查询初学者指南

CSS**媒体查询**为您提供了一种仅当浏览器和设备环境匹配您指定的规则（例如“视口宽度大于 480 像素”）时才应用 CSS 的方法。媒体查询是响应式网页设计的关键部分，因为它们允许您根据视口的大小创建不同的布局，但它们也可用于检测有关您网站运行环境的其他信息，例如用户使用的是触摸屏而不是鼠标。在本课中，您将首先了解媒体查询中使用的语法，然后继续在一个工作示例中使用它们，展示如何使简单的设计成为响应式的。

先决条件：	HTML 基础知识（研究 HTML 简介），以及 CSS 工作原理的概念（研究 CSS 第一步和 CSS 构建块。）
客观的：	了解如何使用媒体查询，以及使用它们创建响应式设计的最常用方法。

媒体查询基础

最简单的媒体查询语法如下所示：

```
@media media-type and (media-feature-rule) {
  /* CSS rules go here */
}
```

它包括：

- 一种媒体类型，它告诉浏览器此代码适用于哪种媒体（例如打印或屏幕）。
- 媒体表达，它是规则或测试，必须通过才能应用包含的 CSS。
- 如果测试通过并且媒体类型正确，将应用一组 CSS 规则。

媒体类型

您可以指定的可能媒体类型是：

- `all`
- `print`
- `screen`

如果打印页面，以下媒体查询只会将正文设置为 12pt。当页面在浏览器中加载时，它将不适用。

```
@media print {
  body {
    font-size: 12pt;
  }
}
```

> **注意：**这里的媒体类型不同于所谓的MIME 类型。

> **注意：** Level 3 媒体查询规范中定义了许多其他媒体类型；这些已被弃用，应避免使用。

> **注意：**媒体类型是可选的；如果你没有在你的媒体查询中指明媒体类型，那么媒体查询将默认为所有媒体类型。

媒体功能规则

指定类型后，您可以使用规则定位媒体功能。

宽度和高度

为了创建响应式设计（并且具有广泛的浏览器支持），我们最常检测的功能是视口宽度，如果视口高于或低于某个宽度（或精确宽度），我们可以应用 CSS，使用 `min-width` , `max-width` 和 `width` 媒体功能。

这些功能用于创建响应不同屏幕尺寸的布局。例如，如果视口恰好是 600 像素，要将正文文本颜色更改为红色，您可以使用以下媒体查询。

```
@media screen and (width: 600px) {
  body {
    color: red;
  }
}
```

在浏览器中打开这个例子，或者 查看源码 。

width (and) 媒体特征 height 可以用作范围，因此以 min- or为前缀 max- 来指示给定值是最小值或最大值。例如，如果视口为 600 像素或更窄，要使颜色变为蓝色，请使用 max-width：

```
@media screen and (max-width: 600px) {
  body {
    color: blue;
  }
}
```

在浏览器中打开这个例子，或者 查看源码 。

在实践中，使用最小值或最大值对于响应式设计更有用，因此您很少会单独看到 width 或 height 使用它们。

尽管媒体查询规范的第 4 级和第 5 级中引入的一些较新的功能对浏览器的支持有限，但您还可以测试许多其他媒体功能。MDN 上记录了每个功能以及浏览器支持信息，您可以在使用媒体查询：语法中找到完整列表。

方向

一个得到很好支持的媒体功能是 orientation ，它允许我们测试纵向或横向模式。如果设备处于横向，要更改正文文本颜色，请使用以下媒体查询。

```
@media (orientation: landscape) {
  body {
    color: rebeccapurple;
```

```
  }
}
```

在浏览器中打开这个例子，或者 查看源码 。

标准桌面视图具有横向方向，在该方向上运行良好的设计在以纵向模式在手机或平板电脑上查看时可能效果不佳。方向测试可以帮助您创建针对纵向模式设备优化的布局。

定点设备的使用

作为 Level 4 规范的一部分， `hover` 引入了媒体功能。此功能意味着您可以测试用户是否能够将鼠标悬停在某个元素上，这实际上意味着他们正在使用某种定点设备；触摸屏和键盘导航不会悬停。

```
@media (hover: hover) {
  body {
    color: rebeccapurple;
  }
}
```

在浏览器中打开这个例子，或者 查看源码 。

如果我们知道用户不能悬停，我们可以默认显示一些交互功能。对于可以悬停的用户，我们可能会选择在悬停链接时让他们可用。

4 级中还有 `pointer` 媒体功能。这需要三个可能的值 `none` ， `fine` 和 `coarse` 。指针 `fine` 类似于鼠标或触控板。它使用户能够精确地瞄准一个小区域。指针 `coarse` 是触摸屏上的手指。该值 `none` 表示用户没有指针设备；也许他们仅使用键盘或语音命令进行导航。

使用 `pointer` 可以帮助您设计更好的界面，以响应用户与屏幕的交互类型。例如，如果您知道用户正在与作为触摸屏的设备进行交互，您可以创建更大的点击区域。

更复杂的媒体查询

对于所有可能的不同媒体查询，您可能希望将它们组合起来，或者创建查询列表——其中任何一个都可以匹配。

媒体查询中的“和”逻辑

要组合媒体功能，您可以使用与我们上面使用的组合媒体类型和功能 `and` 大致相同的方式。 `and` 例如，我们可能想要测试 a `min-width` 和 `orientation` 。仅当视口宽度至少为 600 像素且设备处于横向模式时，正文文本才会显示为蓝色。

```
@media screen and (min-width: 600px) and (orientation:
landscape) {
  body {
    color: blue;
  }
}
```

在浏览器中打开这个例子，或者 查看源码 。

媒体查询中的“或”逻辑

如果您有一组查询，其中任何一个都可以匹配，那么您可以用逗号分隔这些查询。在下面的示例中，如果视口宽度至少为 600 像素或设备处于横向，则文本将为蓝色。如果其中任何一个为真，则查询匹配。

```
@media screen and (min-width: 600px), screen and (orientation:
landscape) {
  body {
    color: blue;
  }
}
```

在浏览器中打开这个例子，或者 查看源码 。

媒体查询中的“非”逻辑

您可以使用运算符否定整个媒体查询 `not` 。这颠倒了整个媒体查询的含义。因此，在下一个示例中，只有当方向为纵向时，文本才会显示为蓝色。

```
@media not all and (orientation: landscape) {
  body {
    color: blue;
  }
}
```

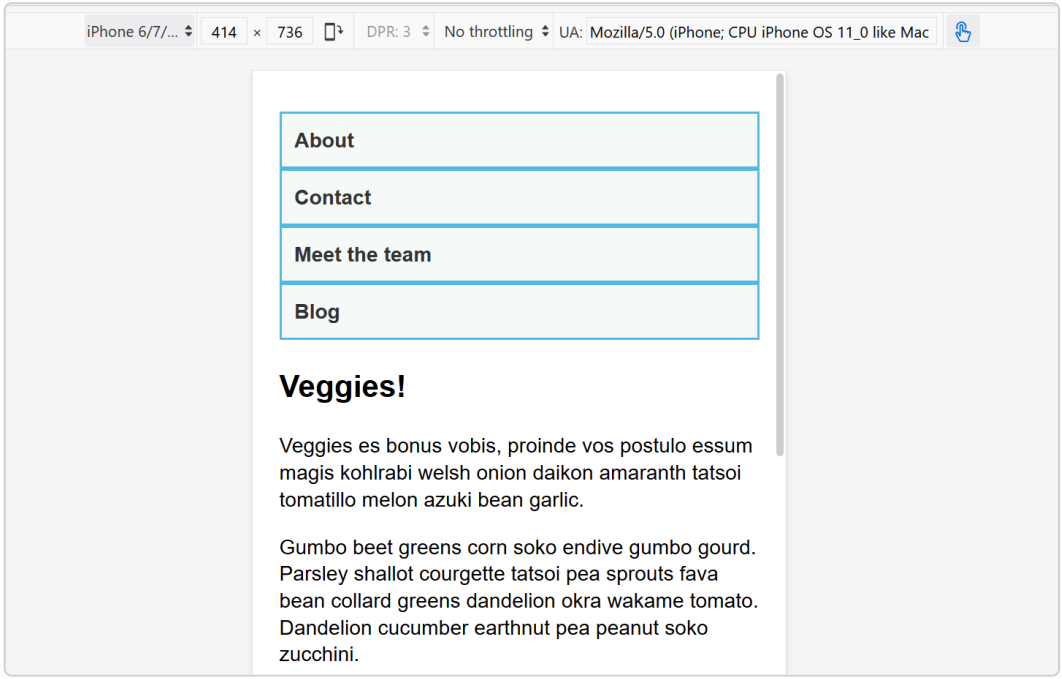
在浏览器中打开这个例子，或者 查看源码 。

如何选择断点

在响应式设计的早期，许多设计师会尝试针对非常具体的屏幕尺寸。发布了流行手机和平板电脑屏幕尺寸的列表，以便可以创建与这些视口完美匹配的设计。

现在有太多的设备，尺寸各不相同，无法实现。这意味着，与其针对所有设计的特定尺寸，更好的方法是在内容开始以某种方式中断的尺寸处更改设计。也许行的长度变得太长，或者一个盒装的侧边栏被压扁并且难以阅读。这就是您想要使用媒体查询将设计更改为适合可用空间的更好设计的时刻。这种方法意味着使用的设备的确切尺寸是多少并不重要，每个范围都可以满足。引入媒体查询的点称为**断点**。

Firefox DevTools 中的响应式设计模式 对于确定这些断点的位置非常有用。您可以轻松地使视口变小或变大，以查看通过添加媒体查询和调整设计可以改进内容的地方。



主动学习：移动优先响应式设计

从广义上讲，您可以采用两种方法来进行响应式设计。您可以从桌面或最宽的视图开始，然后随着视口变小添加断点来移动内容，或者您可以从最小的视图开始并随着视口变大添加布局。第二种方法被描述为**移动优先**响应式设计，通常是最好的方法。

最小设备的视图通常是一个简单的单列内容，就像它在正常流程中出现的那样。这意味着您可能不需要为小型设备做很多布局 — 对您的源进行良好排序，默认情况下您将拥有可读的布局。

下面的演练将通过非常简单的布局向您介绍这种方法。在生产站点中，您可能需要在媒体查询中调整更多内容，但方法完全相同。

演练：一个简单的移动优先布局

我们的起点是一个 HTML 文档，其中应用了一些 CSS 来为布局的各个部分添加背景颜色。

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

body {
  width: 90%;
  margin: 2em auto;
  font: 1em/1.3 Arial, Helvetica, sans-serif;
}

a:link,
a:visited {
  color: #333;
}

nav ul,
aside ul {
  list-style: none;
  padding: 0;
}
```

```
nav a:link,
nav a:visited {
  background-color: rgba(207, 232, 220, 0.2);
  border: 2px solid rgb(79, 185, 227);
  text-decoration: none;
  display: block;
  padding: 10px;
  color: #333;
  font-weight: bold;
}

nav a:hover {
  background-color: rgba(207, 232, 220, 0.7);
}

.related {
  background-color: rgba(79, 185, 227, 0.3);
  border: 1px solid rgb(79, 185, 227);
  padding: 10px;
}

.sidebar {
  background-color: rgba(207, 232, 220, 0.5);
  padding: 10px;
}

article {
  margin-bottom: 1em;
}
```

我们没有对布局进行任何更改，但是文档的来源以一种使内容可读的方式进行了排序。这是重要的第一步，它确保如果内容由屏幕阅读器读出，它是可以理解的。

```
<body>
  <div class="wrapper">
    <header>
      <nav>
        <ul>
          <li><a href="">About</a></li>
          <li><a href="">Contact</a></li>
          <li><a href="">Meet the team</a></li>
          <li><a href="">Blog</a></li>
```

```
        </ul>
      </nav>
    </header>
    <main>
      <article>
        <div class="content">
          <h1>Veggies!</h1>
          <p>…</p>
        </div>
        <aside class="related">
          <p>…</p>
        </aside>
      </article>

      <aside class="sidebar">
        <h2>External vegetable-based links</h2>
        <ul>
          <li>…</li>
        </ul>
      </aside>
    </main>

    <footer><p>&copy;2019</p></footer>
  </div>
</body>
```

这种简单的布局也适用于移动设备。如果我们在 DevTools 中以响应式设计模式查看布局，我们可以看到它作为站点的直接移动视图运行得非常好。

在浏览器中打开步骤1，或者 查看源码 。

如果您想在我们进行时继续并实施此示例，请在您的计算机上制作 step1.html的本地副本。

从这一点开始，开始将响应式设计模式视图拖得更宽，直到您可以看到线的长度变得很长，并且我们有空间让导航显示在水平线上。这是我们将添加第一个媒体查询的地方。我们将使用 ems，因为这意味着如果用户增加了他们的文本大小，断点将发生在与文本大小较小的用户相似的行长但更宽的视口处。

将以下代码添加到 step1.html CSS 的底部。

```
@media screen and (min-width: 40em) {
  article {
    display: grid;
    grid-template-columns: 3fr 1fr;
    column-gap: 20px;
  }

  nav ul {
    display: flex;
  }

  nav li {
    flex: 1;
  }
}
```

此 CSS 为我们提供了文章内部的两列布局，文章内容和相关信息位于 aside 元素中。我们还使用 flexbox 将导航排成一行。

在浏览器中打开第2步、或者 查看源码 。

让我们继续扩大宽度，直到我们觉得有足够的空间让侧边栏也形成一个新的列。在媒体查询中，我们会将主要元素制作成两列网格。然后我们需要删除 `margin-bottom` 文章上的 以便两个侧边栏彼此对齐，我们将 `border` 在页脚顶部添加一个。通常，这些小调整是您为使设计在每个断点处看起来都不错而要做的事情。

同样，将以下代码添加到 step1.html CSS 的底部。

```
@media screen and (min-width: 70em) {
  main {
    display: grid;
    grid-template-columns: 3fr 1fr;
    column-gap: 20px;
  }

  article {
    margin-bottom: 0;
  }
```

```
  footer {
    border-top: 1px solid #ccc;
    margin-top: 2em;
  }
}
```

在浏览器中打开第3步、或者 查看源码 。

如果您查看不同宽度的最后一个示例，您可以看到设计如何响应并作为单列、两列或三列工作，具体取决于可用宽度。这是移动优先响应式设计的一个非常简单的示例。

视口元标记

如果您查看上面示例中的 HTML 源代码，您会看到文档头部包含以下元素：

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

这是视口元标记——它作为一种控制移动浏览器如何呈现内容的方式而存在。这是必需的，因为默认情况下，大多数移动浏览器都会谎报其视口宽度。在窄视口中呈现时，非响应式网站通常看起来非常糟糕，因此移动浏览器通常默认使用比真实设备宽度更宽的视口宽度（通常为 980 像素）呈现网站，然后缩小呈现结果以使其适合在显示中。

这一切都很好，但这意味着响应式网站无法按预期工作。如果视口宽度报告为 980 像素，则移动布局（例如使用 的媒体查询创建 `@media screen and (max-width: 600px) { }`）将不会按预期呈现。

为了解决这个问题，在您的页面上包含一个像上面那样的视口元标记，告诉浏览器“不要使用 980 像素的视口来呈现内容——而是使用真实的设备宽度来呈现它，并设置默认的初始比例级别以获得更好的效果一致性。”然后媒体查询将按预期启动。

您可以将许多其他选项放入 `content` 视口元标记的属性中——有关更多详细信息，请参阅使用视口元标记控制移动浏览器上的布局。

你真的需要媒体查询吗?

Flexbox、Grid 和多列布局都为您提供了无需媒体查询即可创建灵活且响应迅速的组件的方法。这些布局方法是否可以在不添加媒体查询的情况下实现您想要的效果，总是值得考虑的。例如，您可能想要一组宽度至少为 200 像素的卡片，并且这 200 像素中的尽可能多的卡片将适合主要文章。这可以通过网格布局来实现，根本不使用媒体查询。

这可以使用以下方法实现：

```
<ul class="grid">
  <li>
    <h2>Card 1</h2>
    <p>…</p>
  </li>
  <li>
    <h2>Card 2</h2>
    <p>…</p>
  </li>
  <li>
    <h2>Card 3</h2>
    <p>…</p>
  </li>
  <li>
    <h2>Card 4</h2>
    <p>…</p>
  </li>
  <li>
    <h2>Card 5</h2>
    <p>…</p>
  </li>
</ul>
```



```
  margin: 0;
  padding: 0;
  display: grid;
  gap: 20px;
  grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
```

```
}

.grid li {
  border: 1px solid #666;
  padding: 10px;
}
```

在浏览器中打开网格布局示例，或 查看源代码 。

在浏览器中打开示例后，将屏幕变宽和变窄以查看列轨道数的变化。这种方法的好处是网格不查看视口宽度，而是查看它可用于该组件的宽度。在关于媒体查询的部分结束时提出您可能根本不需要媒体查询的建议似乎很奇怪！然而，在实践中，您会发现良好地使用现代布局方法，并通过媒体查询进行增强，将提供最佳结果。

测试你的技能！

您已读完本文，但您还记得最重要的信息吗？在继续之前，您可以找到一个测试来验证您是否记住了这些信息——请参阅测试您的技能：响应式网页设计和媒体查询。

概括

在本课中，您学习了媒体查询，还发现了如何在实践中使用它们来创建移动优先的响应式设计。

您可以使用我们创建的起点来测试更多媒体查询。例如，如果您使用媒体 `pointer` 功能检测到访问者的指针粗略，也许您可以更改导航的大小。

您还可以尝试添加不同的组件，看看添加媒体查询或使用 flexbox 或 grid 等布局方法是否是使组件响应的最合适方法。很多时候没有正确或错误的方法——你应该试验并看看哪种方法最适合你的设计和内容。

此页面最后修改于 2023 年 3 月 22 日由MDN 贡献者提供。