媒体查询入门指南

CSS 媒体查询为你提供了一种应用 CSS 的方法，仅在浏览器和设备的环境与你指定的规则相匹配的时候 CSS 才会真的被应用，例如"视口宽于 480 像素"的时候。媒体查询是响应式 Web 设计的关键部分，因为它允许你按照视口的尺寸创建不同的布局，不过它也可以用来探测和你的站点运行的环境相关联的其他条件，比如用户是在使用触摸屏还是鼠标。在本节课，你将会先学习到媒体查询的语法，然后继续在一个被安排好的示例中使用它，这个示例还会告诉你一个简单的设计是可以怎么被弄成响应式的。

学习前提：	HTML 基础知识（学习Introduction to HTML），对 CSS 工作方式的大致了解（学习CSS first steps和CSS building blocks）
目标：	理解如何使用媒体查询和用它建立响应式设计的最常见方法。

媒体查询基础

最简单的媒体查询语法看起来是像这样的：

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

它由以下部分组成：

- 一个媒体类型，告诉浏览器这段代码是用在什么类型的媒体上的（例如印刷品或者屏幕）；
- 一个媒体表达式，是一个被包含的 CSS 生效所需的规则或者测试；

- 一组 CSS 规则，会在测试通过且媒体类型正确的时候应用。

媒体类型

你可以指定的媒体类型为：

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

下面的媒体查询将会在页面被打印的时候把 body 设定为只有 12pt 大小。当页面在浏览器中载入的时候，它将不会生效。

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

> **备注：**这里的媒体类型是和所谓的MIME type不同的东西。

> **备注：** 在第三级媒体查询规范中，定义了其他一些媒体类型，它们已经不被建议使用，而且应该被避免使用。

> **备注：**媒体类型是可选的，如果你没有在媒体查询中指示一个媒体类型的话，那么

媒体特征规则

在指定了类型以后，你可以用一条规则指向一种媒体特征。

宽和高

为了建立响应式设计（已经广受浏览器支持），我们一般最常探测的特征是视口宽度，而且我们可以使用 `min-width` 、 `max-width` 和 `width` 媒体特征，在视口宽度大于或者小于某个大小——或者是恰好处于某个大小——的时候，应用 CSS。

这些特征是用来创建响应不同屏幕大小的布局的。例如，要想在视口正好是 600 像素的时候，让 body 的文本变为红色，你可能会使用下面的媒体查询。

CSS

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

在浏览器中打开这个示例 ，或者查看源代码 。

`width` （和 `height` ）媒体特征可以以数值范围使用，于是就有了 `min-` 或者 `max-` 的前缀，指示所给的值是最小值还是最大值。例如，要让颜色在视口窄于 400 像素的时候变成蓝色的话，可以用 `max-width` ：

CSS

```
@media screen and (max-width: 400px) {
  body {
    color: blue;
  }
}
```

在浏览器中打开示例 ，或者查看源代码 。

实践中，使用最小值和最大值对响应式设计有很多的用处，所以你会很少见到 `width` 或 `height` 单独使用的情况。

还有许多其他媒体特征可以供你测试，尽管于 4 级和 5 级媒体查询规范中引入了一些新特征，它们受浏览器支持仍然有限。在 MDN 上，每个特征都已经同浏览器支持信息一同记载下来，你可以在使用媒体查询：媒体特征中找到一张完整的列表。

朝向

一个受到良好支持的媒体特征是 `orientation`，我们可以用它测得竖放（portrait mode）和横放（landscape mode）模式。要在设备处于横向的时候改变 body 文本颜色的话，可使用下面的媒体查询。

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

在浏览器中打开此示例 ，或者查看源代码 。

标准的桌面视图是横放朝向的，在这种朝向上能够表现良好的设计，在处于竖放模式的手机或平板电脑上可能不会表现得这么好。对朝向的测试可以帮你建立一个为竖放设备优化的布局。

使用指点设备

作为四级规范的一部分， `hover` 媒体特征被引入了进来。这种特征意味着你可以测试用户是否能在一个元素上悬浮，这也基本就是说他们正在使用某种指点设备，因为触摸屏和键盘导航是没法实现悬浮的。

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

在浏览器中打开此示例 ，或者查看源代码 。

如果我们知道用户不能悬浮的话，我们可以默认显示一些交互功能。对于能够悬浮的用户，我们可以选择在悬浮在链接上的时候，让这些功能可用。

还是在四级规范中，出现了 `pointer` 媒体特征。它可取三个值： `none` 、 `fine` 和 `coarse` 。 `fine` 指针是类似于鼠标或者触控板的东西，它让用户可以精确指向一片小区域。 `coarse` 指针是你在触摸

屏上的手指。 none 值意味着，用户没有指点设备，也许是他们正只使用键盘导航，或者是语音命令。

使用 pointer 可以在用户使用屏幕时进行交互时，帮你更好地设计响应这种交互的界面。例如，如果你知道用户正在用触摸屏设备交互的时候，你可以建立更大的响应区域。

更复杂的媒体查询

有了所有不同的可用的媒体查询，你可能想要把它们混合起来，或者建立查询列表——其中的任何一个都可以匹配生效。

媒体查询中的“与”逻辑

为了混合媒体特征，你可以以与在上面使用 and 很相同的方式，用 and 来混合媒体类型和特征。例如，我们可能会想要测得 min-width 和 orientation ，而 body 的文字只会在视口至少为 400 像素宽，且设备横放时变为蓝色。

CSS

```css
@media screen and (min-width: 400px) and (orientation: landscape) {
  body {
    color: blue;
  }
}
```

在浏览器中打开此示例 ，或者查看源代码 。

媒体查询中的“或”逻辑

如果你有一组查询，且要其中的任何一个都可以匹配的话，那么你可以使用逗号分开这些查询。在下面的示例中，文本会在视口至少为 400 像素宽的时候**或者**设备处于横放状态的时候变为蓝色。如果其中的任何一项成立，那么查询就匹配上了。

CSS

```css
@media screen and (min-width: 400px), screen and (orientation: landscape) {
  body {
    color: blue;
  }
}
```

在浏览器中打开此示例 ，或者查看源代码 。

媒体查询中的“非”逻辑

你可以用 `not` 操作符让整个媒体查询失效。这就直接反转了整个媒体查询的含义。因而在下面的例子中，文本只会在朝向为竖着的时候变成蓝色。

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

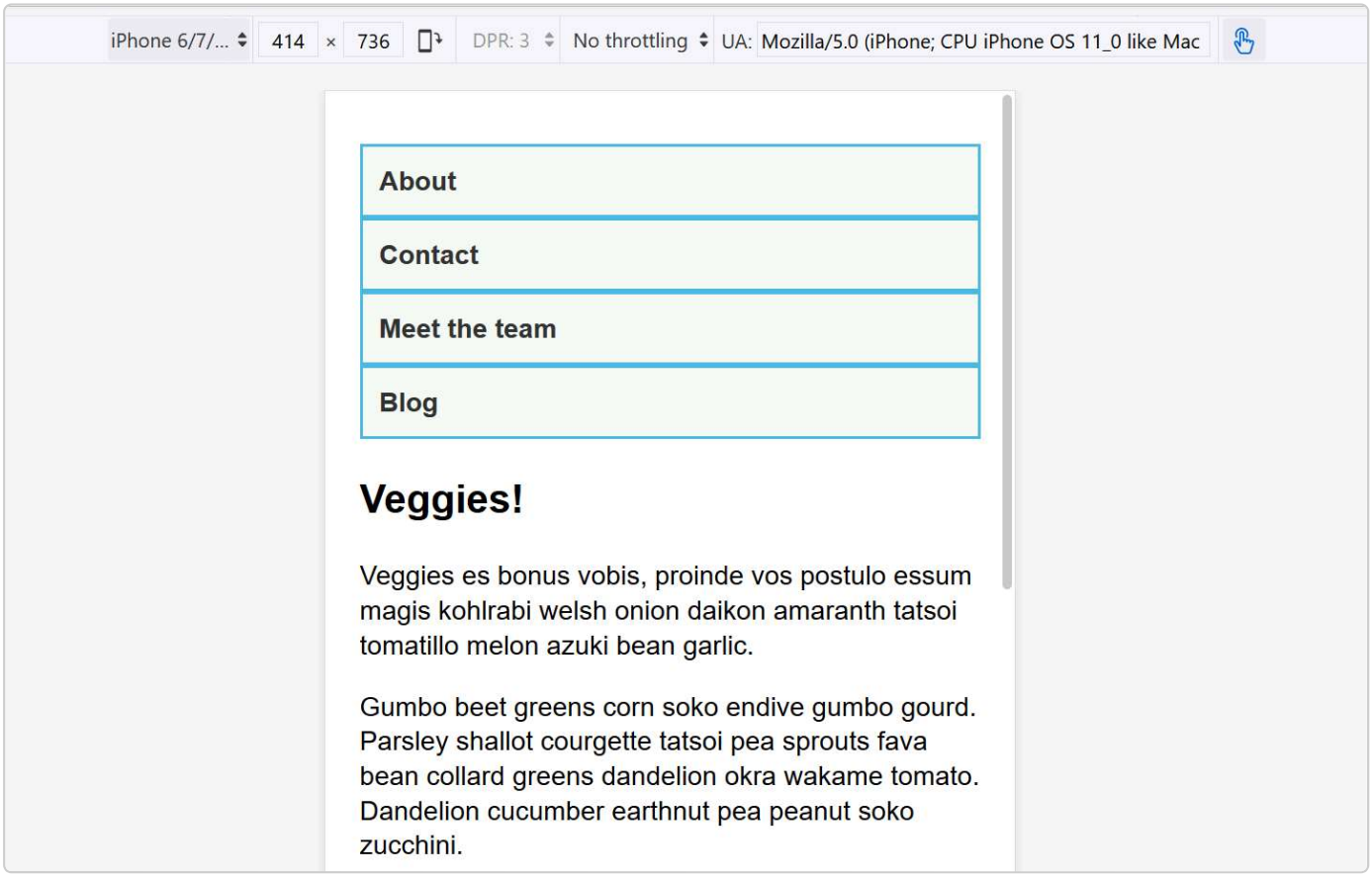
在浏览器中打开此示例 ，或者查看源代码 。

怎么选择断点

响应式设计的早期，许多设计者会尝试指向非常特定的屏幕尺寸。人们公布了流行的手机和平板的屏幕尺寸列表，以让设计者创建可以整齐地放在那些视口里面的设计。

现在有多得多的设备，以及多种多样的尺寸，让这种事变得不再可行。这也就是说，将所有的设计用在特定的尺寸上以外，一个更好的方法是在内容某种程度上开始变得混乱的时候，改变尺寸的设计。也许线太长了，或者盒子状的外侧栏开始挤在一起而难以阅读。那就是你想要使用媒体查询，将设计变得对剩余可用空间更加友好的时候。这种方式意味着，它无关使用的设备的确切大小，每个范围都被照顾到了。引入媒体查询的点就叫做**断点**。

火狐开发者工具中的响应式设计模式 能很好地帮助弄清楚断点应该设置在哪里。你能容易就能让视口变大和变小，然后看下可以在哪里加入媒体查询、调整设计，从而改善内容。



主动学习：移动优先的响应式设计

泛泛地说，你可以采用两种方式实现响应式设计。你可以从桌面或者最宽的视图开始，然后随着视口变得越来越小，加上断点，把物件挪开；你也可以从最小的视图开始，随着视口变得越来越大，增添布局内容。第二种方式被叫做**移动优先**的响应式设计，很多时候是最值得仿效的做法。

用在最小的那个设备上的视图很多时候都是一个简单的单列内容，很像正常文本流显示的那样。这意味着，你很可能不需要为小设备做多少布局设计，合适地安排下你的源代码，默认情况下你就可以得到可读的布局。

下面的教程会领你用一个非常简单的布局熟悉这种方式。在生产站点上，你的媒体查询中可能会有更多的东西需要调整，但是它们的方法是完全一样的。

教程：一个简单的移动优先布局

我们的起始点是一个 HTML 文档，上面应用了一些 CSS，为布局的各部分加入了背景颜色。

CSS

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

```css
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;
}
```

我们没有改变过任何布局，但是文件的源代码是以让内容可读的方式排列的。这个开头是重要的，也是能够确保内容在由屏幕阅读器读出来的时候，让其可以理解的一步。

HTML

```
<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>
```

这个简单的布局在移动端上也能表现得很好。如果我们在开发者工具中的响应式设计模式里面查看这个布局的话，我们可以看到，它作为一个直截了当的站点移动版布局来说，表现得相当优秀。

在浏览器里打开步骤一 ，或者查看源代码 。

如果你想要在我们继续的时候，按步骤来并尝试这个示例，在你的电脑上建立一个step1.html的本地副本。

从这里开始，脱拽响应式设计的窗口，让它变得变得更宽，直到你看到一行变得非常长，有足够空间把导航栏放在一个水平行里面。这是我们加入第一个媒体查询的地方。我们将会使用 em，因为这意味着，如果用户已经增加了文本的大小，断点会在行差不多也是这样长，但是视口更宽的时候产生；而文本更小的时候，视口也会更窄。

将下面的代码加到你的 step1.html 的 CSS 底部。

CSS

```
@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;
  }
}
```

这个 CSS 让我们的文章里面有了个两列布局，两栏分别是文章的内容和在 aside 元素中相关的信息。我们也已经用弹性盒把导航栏放在了一行里面。

在浏览器中打开步骤二 ，或者查看源代码 。

让我们继续增加宽度，直到我们觉得这里有了足够多的空间来放置侧栏，再形成一列。在媒体查询中，我们会让 main 元素变成两栏网格。我们之后需要移除文章上的 `margin-bottom` ，让两个侧栏和彼此对齐，然后我们将会往页脚的顶部加上一个 `border` 。一般来说，为了让设计看起来好看，这些小调整是你将会在每一个断点都需要做的。

再往你的 step1.html 的 CSS 的底部加入下面的代码：

CSS

```
@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;
  }
}
```

在浏览器中打开步骤三 ，或者查看源代码 。

如果你在不同的宽度下，看下最后的示例，你会看到设计是如何响应的，在可用的宽度下是如何表现为单栏、双栏或者三栏的。这是一个移动优先的响应式设计的非常简单的示例。

你真的需要媒体查询吗？

弹性盒、网格和多栏布局都给了你建立可伸缩的甚至是响应式组件的方式，而不需要媒体查询。这些布局方式能否在不加入媒体查询的时候实现你想要的设计，总是值得考虑的一件事。例如，你可能想要一组卡片，至少为二百像素宽，并在主文章里尽可能多地放下这些二百像素的卡片。这可以用网格布局实现，而完全不使用媒体查询。

这可以由以下代码实现：

HTML

```
<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>
```

CSS

```
.grid {
  list-style: none;
  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 ，或者查看源代码 。

在你的浏览器里打开这个示例，让屏幕变宽变窄，看一看列轨数目的变化。这个方法里面的好事是，网格不是靠视口宽度判断的，而是可以容纳组件的宽度。对媒体查询这章节的建议就是，你可能根本不需要它！但是，实践中你会发现，由媒体查询改进的现代布局方式的恰当使用，将会产生最佳效果。

技能测试！

你已经到了此文的结尾，但是你能记住最重要的信息吗？你可以在继续之前，找一个测试来验证下你是否已经掌握了这些信息。见技能测试：响应式 Web 设计 (en-US)。

小结

本节课中，你已经学到了媒体查询的知识，也发现了如何在实践中使用它们，来建立一个移动优先的响应式设计。

你可以使用我们已经建立的起始点，试出更多的媒体查询，例如，也许你能使用 `pointer` 媒体特征，在你探测到访客有一个模糊指针的时候，改变导航栏的大小。

你还能通过加入不同的组件进行实验，看下加入媒体查询，或者使用类似弹性盒或者网格的布局方式，哪个是让组件可响应的最佳途径。经常是没有什么对错的，你应该实验下，看看哪个对你的设计和内容效果最好。