title: 浏览器图片机制
date: 2020-11-06 18:48:40
categories:

tags: web

前言

浏览器中引用图片是一种很常见的情况，使用方式的不同，他们的意义也不同。比如————作为“内容主体”、“背景”、“图标”等，而设计师有时候也会提供不同的格式图片（img/png/svg/）。在不同的场景，我们对同一份图片素材，要根据图片在web界面中的意义合理运用。个人会列举浏览器常用使用图片的方式。

- img
- background-image
- icon-font
- svg
- webpack 与 img
- base64

<!-- more -->

img、background-image

HTML的img标签、css的background-image样式是最原始的使用图片方式，在H5时代前，相当长的一段时间都是用这两方式引用图片资源的。

<img>标签，将图片作为内容主体引入web页面，故其是占位的；而background-image样式，起修饰作用，不占位。

基本用法

img

```
<img src="https://mdn.mozillademos.org/files/7693/catfront.png" />
```

background-image

```
<div class="background"></div>
.background {
  width: 30px;
  height: 64px;
  margin: 0 auto;
  background-image: url('https://mdn.mozillademos.org/files/7693/catfront.png');
}
```


<style>
.background {

margin: 0 auto;
background-image: url('https://mdn.mozillademos.org/files/7693/catfront.png');
}
</style>

虽然呈现的效果一致，意义却不一样。

<!-- - <img>: 图像嵌入元素 -->
<!-- - background-image -->

img之尺寸、居中

img标签提供了关于设置尺寸的属性，分别是width和height，单位可以是css像素，也可以是百分比。

```
<img src="https://mdn.mozillademos.org/files/7693/catfront.png"  width="100%" height="100%" />
```

然而尺寸的百分比单位并不是相对于图片资源的比例，而是其容器的百分比。

所以并不推荐使用img标签的width及height设置属性，推荐使用css的width及height属性编写。

其实很多web开发者设置100%的本意是想让图片**按父容器宽度自动缩放，并保持图片原本的长宽比。**

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

除此之外，img的居中方式也是很容易让人误解，因img的display属性为inline-block，其居中方式（水平、垂直都是）更是让人误解。

```
<div class="block">
  <img src="https://mdn.mozillademos.org/files/7693/catfront.png" />
</div>
.block {
  width:150px;
  height: 150px;
  border: 1px solid #333;
  /* 垂直居中 */
  display: table-cell;
  vertical-align: middle;
  /* 水平居中 */
  text-align: center;
}

img {
    max-width: 50%;
    max-height: 50%;
}
```

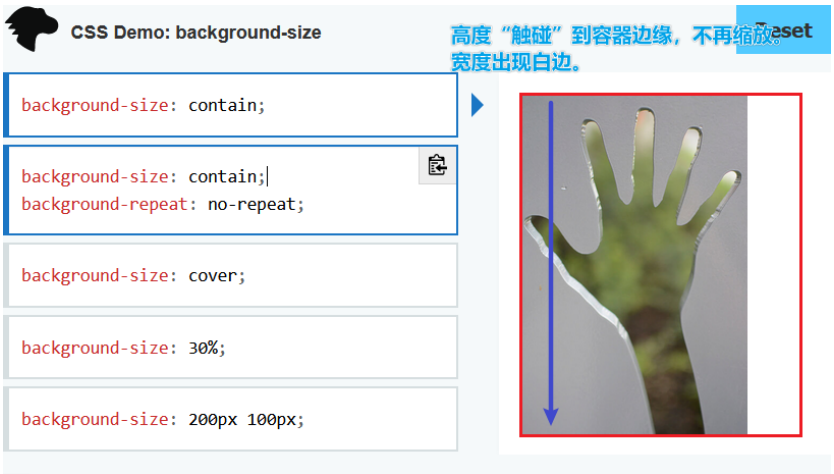
没想吧？居然是加在img的容器标签上，虽然绝对居中（水平、垂直都同时居中）还有其他的方法，常见的margin auto居中还有绝对定位50%。不过个人觉得这是最值得开发着去记住的。除此之外，也建议给容器设置font-size: 0;，这可以解决两个相邻的img标签之间的空隙问题。

说了那么多的img，现在得回过头来谈论background-image了。

background-image之位置、尺寸及重复

虽然前面我们说的background-image一直说的是css的样式特性，然而background-image只能指定使用的图片资源（可以是一张、也可以是多张）。背景图片样式（如本节小标题所说的位置、尺寸及重复）的设置，往往还需要结合其他css特性。

1. background-position可以给背景图片定义位置，设置的是其图片的左上角要在容器的哪个偏移度位置。
2. background-size 可设置背景图片大小。contain理解为等比例缩放图片，高度/宽度其一先与容器尺寸相等，则停止缩放，若图片和容器宽高比例不一致，会出现白边；cover也是等比例缩放，高度/宽度其一先与容器尺寸相等，继续缩放，（此时图片溢出），直到另一方向的尺寸占满容器，停止缩放。



更简易的理解： contain为最小化等比例缩放图片，cover则为最大化等比例缩放。

除了这两个关键字，也可以用两个单位值指定背景图片的宽高，对于绝对单位（px、em、rem）没啥好说的，对于相对单位（百分比），是相对于容器的尺寸来计算的，有意思的是100% 100%，这代表着破坏原比例，把图片拉伸/挤压到容器的尺寸。（很多css属性的相对单位都是根据容器来计算的，或许有特殊的属性我忘了。;-）

1. background-repeat 设置图片重复使用的方式。

以上就是关于背景常用的css样式特性，完整的css背景样式如下，并不复杂。

- background-attachment
- background-clip
- background-color
- background-image
- background-origin
- background-position
- background-position-x
- background-position-y
- background-repeat
- background-size

不建议使用css简写属性background一次性设置背景特性。

CSS Sprites

CSS Sprite(CSS 精灵)，又名雪碧图，是一种图片合并技术，我们可以把一些小图，整合放在一张大图中，每次单独使用小图的时候，裁剪出指定位置，尺寸即可正常显示。

像上图就可以作为雪碧图的素材使用，以实现改方案。

简单分析一下这张图片：

1. 尺寸：134 * 44

2. 小图数量：3
3. 规范：固定大小，水平排列

那我们可以定义一个通用的class，设置小图的尺寸；再定义一个class，设置图片裁剪位置即可。

```
.css-sprite {
    width: 44px;
    height: 44px;
    background: url("./CSS-Sprites.gif");
    background-repeat: no-repeat;
}

.hourse {
    background-position-x: 0;
}
.left-arrow {
    background-position-x: -44px;
}
.right-arrow {
    background-position-x: -88px;
}
```

特性

1. 减少服务器压力：多图合并成一张，只发送一次HTTP请求，并且可以被缓存，有助于提升页面加载性能
2. 维护困难： 后期维护成本较高，添加一张图片需要重新制作。
3. 应用麻烦：每应用一张图片都需要调整位置，误差要求严格。
4. 局限：只能用在背景图片background-image上，不能用<img>标签来使用。

不同方式实现 CSS Sprites

如果会使用gulp、webkack 等构建工具，可以借助工具自动生成雪碧图。

spritesmith，是一个node工具，可以将多张图片合成一张图片——雪碧图，也提供了grup和 gulp插件，甚至是命令行工具。

- gulp 结合spritesmith的插件 gulp.spritesmith
- webpack结合对应的loader webpack-spritesmith
- svg svg-sprite-loader

icon-font

首先我们得明白，icont-font本质上不是图片，而是一种比较特殊字体，这种字体，以图标的方式显示。

web字体

得益于css3的新特性“web字体”，我们可以为自己的网页定义在线字体，无论用户是否安装了我们指定的字体，我们都可以让网页呈现出我们想要的字体，突破了传统Web-safe 字体的限制。

```
/* 定义名为“Open Sans”字体 */
@font-face {
  font-family: "Open Sans";
  src: url("/fonts/OpenSans-Regular-webfont.woff2") format("woff2"),
       url("/fonts/OpenSans-Regular-webfont.woff") format("woff");
}

/* 应用于网页 */
body {
  font-family: "Open Sans";
}
```

web字体不是我们这次要重点的谈论范围，了解即可，这里提供了一些相关资料：

- Web 字体
- @font-face

- font-family

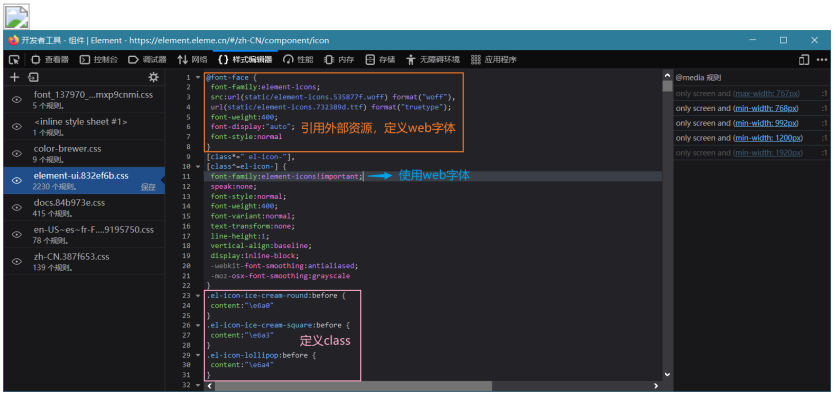
所以icon-font，指的是使用自定义的字体展示图标。运用了上述介绍的web字体技术。

基本使用

字体图标技术已经是web主流使用icon的一种方案了，很多UI库都内置提供了一套图标库供开发者使用，当然也可以独立使用开源的图标库，或者使用工具生成自定义的图标库。

这里用常用的UI框架elemment-ui举例，其UI库提供了icon组件，内置了图标库，使用方式也很简单。

基本上，正确引用了icon-font，直接设置的类名即可展示对应的UI，那么我们究竟引用了什么东西呢？在icon页面上，使用F12打开开发者工具、找到element-ui-*.css源码。



独立的开源图标库有很多，名气比较大的有Font Awesome。使用方法也是类比UI库，引入相关的css源文件即可。css源文件也是类似element-ui-*.css的格式，引用外部资源、定义web字体、使用web字体，内置定义了class。直接在相关dom中使用class即可。

实际项目中，这种开源的集成图标库往往不能满足需求设计稿，我们需要使用一些自定义图标。我们可以使用一些工具：iconfont、fontello、icomoon，都是很优秀的在线生成图标库，具体使用方式网站也有介绍，不再累述。教程中引入的css，也是跟element-ui-*.css的格式类似。

特性

icon-font最大的特性就是样式有更多的灵活性。**我们可以像处理文字一样处理图标的样式。**使用font-size控制图标的尺寸，text-align、line-height控制其居中方式，甚至是color为图标设置不同的样式。

一般使用icon-font都是一套一套的使用，而不是一个一个独立使用，所以这对减少网络请求次数也有优势。

矢量图形也意味着我们可以随意调整图标大小而不用担心其失真。

不过icon-font只适用于纯色图标，当然渐变效果也可以使用css样式编写。

svg

svg——可缩放矢量图形(Scalable Vector Graphics)，是一种文件格式， 用XML 的格式定义图像。我们可以使用代码编辑器编辑svg文件，使用浏览器可直接预览。

```
<svg version="1.1"
  baseProfile="full"
  width="300" height="200"
  xmlns="http://www.w3.org/2000/svg">
```

```
  <rect width="100%" height="100%" fill="red" />
  <circle cx="150" cy="100" r="80" fill="green" />
  <text x="150" y="125" font-size="60" text-anchor="middle" fill="white">SVG</text>
</svg>
```

可以先将其拷贝保存为x.svg，待会用到的 。

将在浏览器中呈现...

基本使用

有几种使用方式，第一种是直接作为标签嵌入HTML源码中。

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>

  <svg version="1.1"
     baseProfile="full"
     width="300" height="200"
     xmlns="http://www.w3.org/2000/svg">
     <rect width="100%" height="100%" fill="red" />
     <circle cx="150" cy="100" r="80" fill="green" />
     <text x="150" y="125" font-size="60" text-anchor="middle" fill="white">SVG</text>
  </svg>

</body>
</html>
```

第二种是将svg代码保存为一个单独的文件，如同png，jpg，git等图片资源一样使用。

```
<h1>img svg</h1>
<img src="x.svg" alt="">
<h1>object svg</h1>
<object data="x.svg" type=""></object>
<h1>iframe svg</h1>
<iframe src="x.svg" frameborder="0"></iframe>
```

背景图片也是可以的。

```
<div id="div-svg"></div>
<style>
  #div-svg {
     width: 300px;
     height: 200px;
     background: url('./x.svg');
  }
</style>
```

特性

svg基于XML语法实现，可以用DOM选择器获取该DOM对象。前提是用第一种方式直接将svg嵌入HTML。

```
<!-- 先给刚才的svg加上id属性 -->
  <svg
    id="dom-svg"
    ...>
    ...
  </svg>
const domSvg = document.querySelector('#dom-svg')
console.log(domSvg) // <svg version="1.1" baseProfile="full" width="300" height="200" xmlns="http://www.w3.org/2000/svg">
console.log(domSvg.childNodes) // NodeList(7) [ #text, rect, #text, circle, #text, text, #text ]
const divSvg = document.querySelector('#div-svg')
```

如果有用过icon-font，会知道我们可以用多个svg制作成一套字体图标库。虽然字体图标然比传统的img，background方式有着更好的css样式灵活性，可终究直接使用的时候是纯色的。而svg有着更好的色彩表现能力，同样也是矢量图形，且可以进行DOM操作，这也意味着我们可以**随时动态地改变图片的结构。而且svg不仅仅可以制作成字体图标库，也可以转换成png、jpg等传统图片格式甚至是**canvas。

svg文件格式现在已经是主流web开发图片使用方案了，而且是目前最灵活的图片文件格式。

webpack 与 img

webpack是一个前端打包工具，前端项目的每个静态资源都是一个单独的模块，webpack内部会**自动管理这些依赖关系**，编译源码时会自动根据这些依赖关系进行打开，最终生成*bundle*。其特点是拥有**模块化机制**、loader可以对各种类型的模块加载时运动不同的任务、**插件化**更是令其可以跟其他的构建工具（grunt、gulp等）结合使用、**模块热替换****更是大大加大了开发速度，模块的更新无需重新加载整个页面。

webpack功能多样且强大，我们本次将重点放在webpack是如何处理图片资源的。webpack一般是跟vue或者react框架集成使用，当然也可以独立使用。原理都差不多，框架的脚手架会基于webpack进行对框架场景的适合或者说扩展。为了方便，这里以vue为例。

基本使用

vue-cli已有相关介绍，简单来说，就是在template中，还有在js中有不同的使用图片方式。

原生html 跟vue的template语法是一样的。

```
<img src="./image.png">
```

script 或者叫js中是这样使用的。

```
imgURL = require('./image.png')
<img :src="require('./image.png')">
```

background也类同<img>。

裂图/空图

裂图：指定的图片资源地址不存在，或者加载失败（404），此时界面出现一张小的占位“裂图”；

空图：不指定图片资源地址，如果样式有设置尺寸大小，会根据img/background的原生特性占位。

裂图一般是bug，我们需要根据bug的场景去解决。

空图也有其使用的场景：如，初始img标签，动态加载不同的图片资源地址，默认占位。

实现空图也很简单，src="" :src="null" 即可，也可以直接不使用src属性。

有意思的是，当src的属性值为空时，chrome 和 firefox 渲染的DOM略有差异。

```
<!-- chrome -->
<img src(unknown)>
<!-- firefox -->
<img>
```

小图自动转成base64

有时候我们会发现webpack自动把一些小内存的图片自动转换成databas64格式的编码。在vue-cli也有相关的资料介绍 —— 从相对路径导入。

我们知道vue-cli是一个集成了webpack常用的功能实现的vue脚手架，为了是配置管理vue项目更加方便快捷，内置静默配置了webpack的常用功能。

究其原因还是使用了webpack的file-loader处理资源最终引用的路径。url-loader将小于**指定大小**的资源转成内联（这里包括css、javascript、图片字体等静态资源），css javascript 都拥有html对用的标签，图片资源则是处理成base64格式。为了是节约 HTTP 请求数量。

vue-cli可以使用chainWebpack设置指定大小，如果是单独使用webpack，则应该配置url-loader，以下是参考的url-loader配置。

```
module: {
    loaders: [
        {
            test: /\.(png|jpg)$/,
            loader: 'url-loader?limit=8192'
        }
    ]
}
```

base64

base64是一种编码的方法，可以用来表示**二进制**数据。所以图片也可以被编码成base64，形成一条字符串。

试试将下面的这串字符复制在浏览器地址栏直接访问，看看是本文的出现的哪张图片。

```

```

这条字符串有特定的格式的，这种格式叫Data URL scheme，意为data协议（类同http协议）。base64编码不仅仅可以表示图片，也可以表示其他类型的数据。

```
data:[<mediatype>][;base64],<data>
```

特性

1. 体积会比原来大1/3
2. 不需要请求服务器资源，减少HTTP请求次数
3. 编码、解码方便，算法可逆，不适用于私密信息通信
4. 无法缓存，不建议使用在改动频繁的地方

js创建base64的方法：

1. canvas 将图片转化为 base64 编码
2. FileReader 将图片转化base64格式

使用场景

base64的使用场景比较少，个人之前的工作经历是img url 有token认证机制，直接使用<img src="https://HOST/xx.jpg">，接口状态码返回403。而后端整个api系统都是有token认证的机制，不太好改，img标签又不支持添加HTTP请求头。最后用js发起http请求，设置token请求头。

1. 使用XHR发送http请求，设置响应类型 xhr.responseType = 'blob'
2. 构建 new FileReader()实例，实例事件 onloadend获取base64
3. 将base64赋值给<img>标签的src属性

其他常见的编码格式

- Unicode
- UTF-8
- URL12
- Unix时间戳
- Ascii/Native
- Hex
- Html

小结

我们介绍了从各个话题介绍了浏览器中使用的图片，现在是时候来一波总结了。

1. img / background - 原生html、css实现，分别作为**内容主体**、**样式装饰**功能
2. CSS Sprites （技术方案） - 一种优化图片方案，有不同的实现方式
3. icon-font - 基于h5的web-font特性实现，使用在线工具可生成一套矢量图片库，样式控制灵活

4. svg - 一种很灵活的图片格式，浏览器原生支持，可转换成传统图片格式或者制作成icon-font
5. webpack中使用img - img在工程化中的简单实现
6. base64 - 一种编码方式，格式为Data URL scheme， 可表示图片等二进制文件流

结尾

此本从开发者拿到一张图片素材开始。以在不同场景在，选择最合适的实现方式。尽管写得比较长，可还是少了一点东西。比如canvas没有讲，不过一般canvas用于脚本绘制渲染可视化数据多一些，当然能渲染成img，场景少，就没在这里讨论。本意也是觉得一张图片在浏览器上有多种使用方式，就整理了一下，写下以上内容，算是个人的知识总结回顾吧。