

Web 性能优化： 图片优化让网站大小减少 62%

前端  程序员 图片处理 web性能优化 阅读约 16 分钟

阿里云最近在做活动，低至2折，有兴趣可以看看：

<https://promotion.aliyun.com/...>

为了保证的可读性，本文采用意译而非直译。

这是 Web 性能优化的第二篇，上一篇在下面看点击查看：

1. [Web 性能优化： 使用 Webpack 分离数据的正确方法](#)

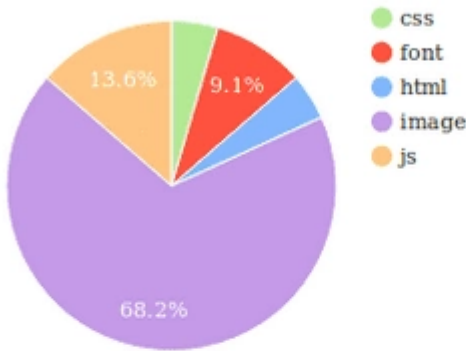
图像是web上提供的最基本的内容类型之一。他们说一张图片胜过千言万语。但是如果你不小心的话，图片大小有时高达几十兆。

因此，虽然网络图像需要清晰明快，但它们尺寸可以缩小压缩的，使用加载时间保持在可接受的水平。

在我的网站上，我注意到我的主页的页面大小 超过了 **1.1MB**，图片占了约88%，我还注意到我提供的图像比它们需要的大(在分辨率方面)，显然，还有很多改进的空间。

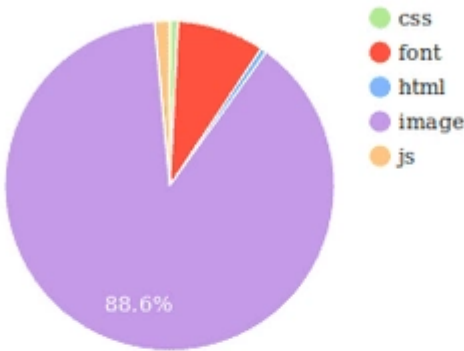
Content breakdown by MIME type (First View)

Requests



MIME Type	Requests ▼
image	15
js	3
font	2
css	1
html	1
flash	0
other	0
video	0

Bytes



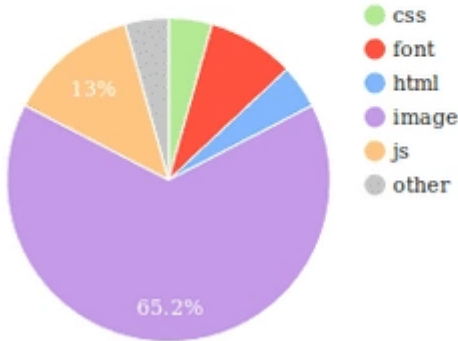
MIME Type	Bytes ▼	Uncompressed
image	1,041,690	1,038,826
font	100,300	99,996
js	17,487	43,539
css	9,755	55,537
html	6,981	25,819
flash	0	0
other	0	0
video	0	0

我开始阅读 Addy Osmani 的优秀 [Essential Image Optimization](#)电子书，并开始在我的网站上按照他们的建议做了一些图片的优化。，然后再对响应式图像进行了一些研究并应用了它。

这使得页面大小减少到 **445kb**，约 **62%**！

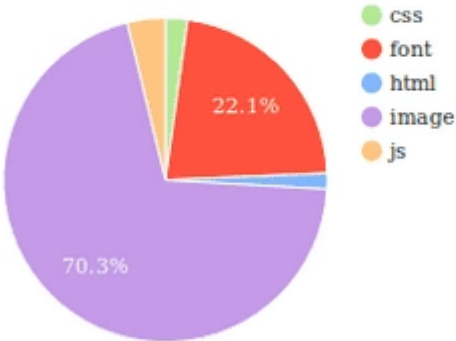
Content breakdown by MIME type (First View)

Requests



MIME Type	Requests ▼
image	15
js	3
font	2
css	1
html	1
other	1
flash	0
video	0

Bytes



MIME Type	Bytes ▼	Uncompressed
image	318,754	316,740
font	100,300	99,996
js	17,256	42,135
css	9,999	57,955
html	7,096	26,415
flash	0	0
other	0	0
video	0	0

什么是图像压缩？

压缩图像就是在图片保持在可接受的清晰度范围内同时减少文件大小，我使用 [imagemin](#) 来压缩站点上的图像。

要使用 [imagemin](#)，确保你已经安装了 Node.js，然后打开一个终端窗口，`cd` 进入项目，并运行以下命令：

```
npm install imagemin
```

然后创建一个名为 `imagemin.js` 的新文件，写入下面的内容：

```
const imagemin = require('imagemin');
const PNGImages = 'assets/images/*.png';
const JPEGImages = 'assets/images/*.jpg';
const output = 'build/images';
```

你可以根据自己的需要更改 `PNGImages`、`JPEGImages` 和 `output` 的值，以符合你的项目结构。

此外要执行图片压缩，还需要根据要压缩的图像类型安装对应的插件。

JPEG/JPG

JPG 的优点

JPG 最大的特点是 **有损压缩**。这种高效的压缩算法使它成为了一种非常轻巧的图片格式。另一方面，即使被称为“有损”压缩，JPG的压缩方式仍然是一种高质量的压缩方式：当我们把图片体积压缩至原有体积的 50% 以下时，JPG 仍然可以保持住 60% 的品质。此外，JPG 格式以 24 位存储单个图，可以呈现多达 1600 万种颜色，足以应对大多数场景下对色彩的要求，这一点决定了它压缩前后的质量损耗并不容易被我们人类的肉眼所察觉——前提是你用对了业务场景。

JPG 使用场景

JPG 适用于呈现色彩丰富的图片，在我们日常开发中，JPG 图片经常作为大的背景图、轮播图或 Banner 图出现。

JPG 的缺陷

有损压缩在上文所展示的轮播图上确实很难露出马脚，但当它处理矢量图形和 Logo 等线条感较强、颜色对比强烈的图像时，人为压缩导致的图片模糊会相当明显。

此外，JPEG 图像不支持透明度处理，透明图片需要召唤 PNG 来呈现。

使用 MozJPEG 压缩 jpeg

这里使用 Mozilla 的 [MozJPEG](#) 工具，该工具可以通过 [imagemin-mozjpeg](#) 作为 Imagemin 插件使用。你可以通过运行以下命令来安装它:

```
npm install imagemin-mozjpeg
```

然后将以下内容添加到的 `imagemin.js` 中:

```
const imageminMozjpeg = require('imagemin-mozjpeg');
const optimiseJPEGImages = () =>
  imagemin([JPEGImages], output, {
    plugins: [
      imageminMozjpeg({
        quality: 70,
      }),
    ],
  });
optimiseJPEGImages()
  .catch(error => console.log(error));
```

可以通过在终端中运行 `node imagemin.js` 来运行脚本。这将处理所有 JPEG 图像，并将优化后的版本放 `build/images` 文件夹中。

我发现将 `quality` 设置为 70 在大多数情况下可以产生足够清晰的图像，但你的项目需求可能不同，可以自行设置合适的值。

默认情况下，MozJPEG [生成渐进式 jpeg](#)，这会导致图像从低分辨率逐渐加载到高分辨率，直到图片完全加载为止。由于它们的编码方式，它们也比原始的 jpeg 略小。

你可以使用 Sindre Sorhus 提供的这个[命令行工具](#)来检查 JPEG 图像是否是渐进式的。

Addy Osmani 已经很好地总结了使用渐进式 jpeg 的[优缺点](#)。对我来说，我觉得利大于弊，所以我坚持使用默认设置。

如果你更喜欢使用原始的 jpeg，可以在 `options` 对象中将 `progressive` 设置为 `false`。另外，请确保 [imagemin-mozjpeg](#) 版本的变化，请重新查看对应文档。

PNG (PNG-8 与 PNG-24)

PNG 的优缺点

PNG（可移植网络图形格式）是一种无损压缩的高保真的图片格式。8 和 24，这里都是二进制数的位数。按照我们前置知识里提到的对应关系，8 位的 PNG 最多支持 256 种颜色，而 24 位的可以呈现约 1600 万种颜色。

PNG 图片具有比 JPG 更强的色彩表现力，对线条的处理更加细腻，对透明度有良好的支持。它弥补了上文我们提到的 JPG 的局限性，唯一的缺点就是 **体积太大**。

PNG 应用场景

前面我们提到，复杂的、色彩层次丰富的图片，用 PNG 来处理的话，成本会比较高，我们一般会交给 JPG 去存储。

考虑到 PNG 在处理线条和颜色对比度方面的优势，我们主要用它来呈现小的 Logo、颜色简单且对比强烈的图片或背景等。

使用 pngquant 优化 PNG 图像

[pngquant](#) 是我优化 PNG 图像的首选工具，你可以通过 [imagemin-pngquant](#) 使用它:

```
npm install imagemin-pngquant
```

然后将以下内容添加到 `imagemin.js` 文件中:

```
const imageminPngquant = require('imagemin-pngquant');
const optimisePNGImages = () =>
  imagemin([PNGImages], output, {
    plugins: [
      imageminPngquant({ quality: '65-80' })
    ],
  });
optimiseJPEGImages()
  .then(() => optimisePNGImages())
  .catch(error => console.log(error));
```

我发现将 `quality` 设置为 `65-80` 可以在文件大小和图像质量之间较好的折衷方案。

有了这些设置，我可以得到一个屏幕截图，我的网站从 913kb 到 187kb，没有任何明显的视觉损失，**惊人的79% 的降幅!**

这是两个文件。看一看，自己判断一下:

- [原图\(913kb\)](#)
- [优化后的图像\(187kb\)](#)

WebP

WebP 的优点

WebP 像 JPEG 一样对细节丰富的图片信手拈来，像 PNG 一样支持透明，像 GIF 一样可以显示动态图片——它集多种图片文件格式的优点于一身。

WebP 的官方介绍对这一点有着更权威的阐述：

与 PNG 相比，WebP 无损图像的尺寸缩小了 26%。在等效的 SSIM 质量指数下，WebP 有损图像比同类 JPEG 图像小25-34%。无损 WebP 支持透明度（也称为 alpha 通道），仅需 22% 的额外字节。对于有损 RGB 压缩可接受的情况，有损 WebP 也支持透明度，与 PNG 相比，通常提供 3 倍的文件大小。

将 WebP 图像提供给支持它们的浏览器

[WebP](#) 是谷歌引入的一种相对较新的格式，它的目标是通过以无损和有损格式编码图像来提供更小的文件大小，使其成为 JPEG 和 PNG 的一个很好的替代方案。

WebP 图像的清晰度通常可以与 JPEG 和 PNG相提并论，而且文件大小要小得多。例如，当我将屏幕截图从上面转换到 WebP 时，我得到了一个 88kb 的文件，其质量与 913kb 的原始图像相当，**减少了90%！**

看看这三张图片，你能说出区别吗？

- [原图PNG \(913kb\)](#)
- [优化PNG图像\(187kb\)](#)
- [WebP图像\(88kb, 可在Chrome或Opera浏览器中浏览\)](#)

就我个人而言，我认为视觉效果是可以比较的，而且节省下来的大小是不容忽视的。

既然我们已经认识到在可能的情况下使用WebP格式是有价值的，那么很重要的一点是—它不能完全替代 JPEG 和 PNG，因为浏览器对 WebP 支持并不普遍。

在撰写本文时，Firefox、Safari 和 Edge 都是不支持WebP的浏览器。



然而，根据 caniuse.com 的数据，全球超过70%的用户使用支持WebP的浏览器。这意味着，通过使用 WebP 图像，可以为大约 70% 的客户提供更快的 web 页面及更好的体验。

安装它，运行以下命令：

```
npm install imagemin-webp
```

然后将以下内容添加到你的 `imagemin.js` 文件中:

```
const imageminWebp = require('imagemin-webp');
const convertPNGToWebp = () =>
  imagemin([PNGImages], output, {
    use: [
      imageminWebp({
        quality: 85,
      }),
    ]
  });
const convertJPGToWebp = () =>
  imagemin([JPGImages], output, {
    use: [
      imageminWebp({
        quality: 75,
      }),
    ]
  });
optimiseJPEGImages()
  .then(() => optimisePNGImages())
  .then(() => convertPNGToWebp())
  .then(() => convertJPGToWebp())
  .catch(error => console.log(error));
```

我发现，将 `quality` 设置为 `85` 会生成质量与 PNG 相当但小得多的 WebP 图像。对于 jpeg，我发现将 `quality` 设置为 `75` 可以在视觉和文件大小之间取得很好的平衡。

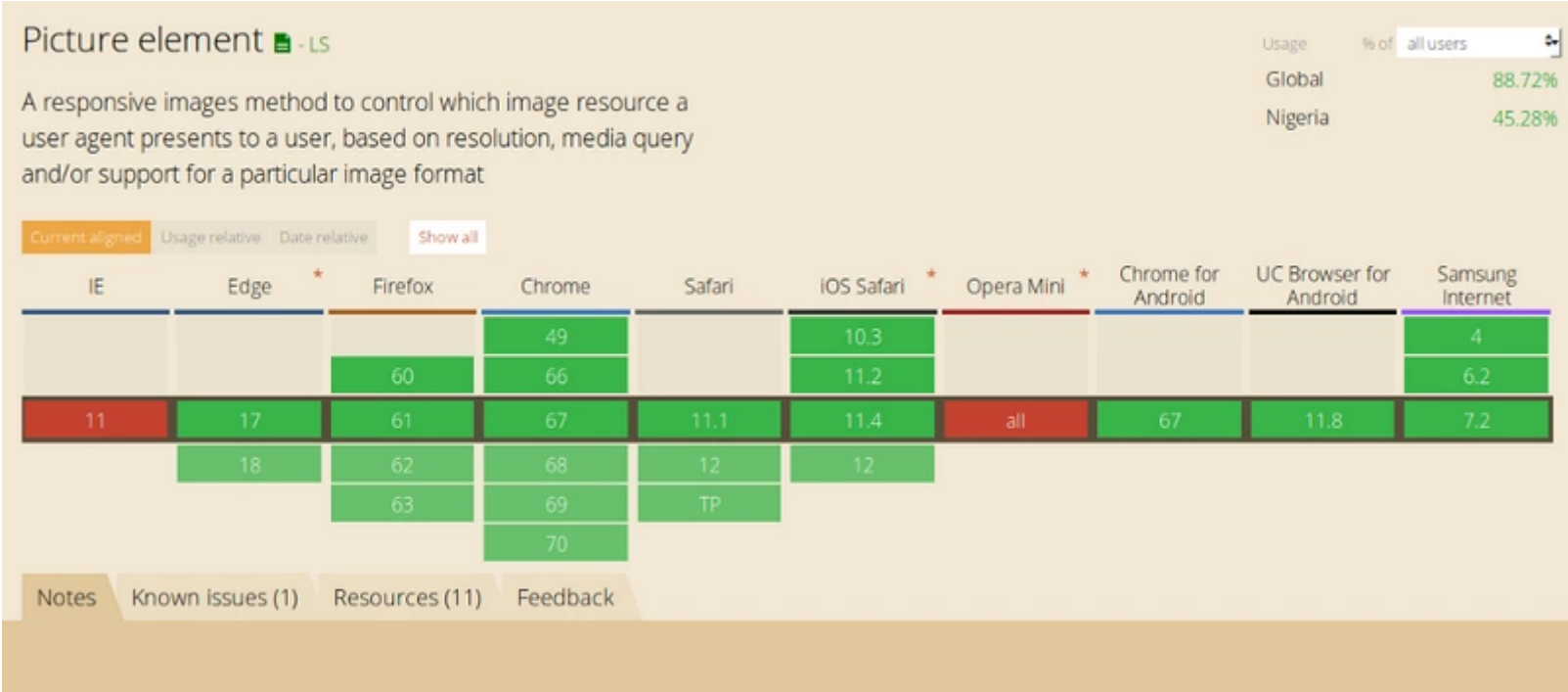
提供 HTML格式的WebP图像

一旦有了 WebP 图像，可以使用以下标记将它们提供给可以使用它们的浏览器，同时向不兼容 WebP 的浏览器使用 png 或者 jpeg。

```
<picture>
  <source srcset="sample_image.webp" type="image/webp">
  <source srcset="sample_image.jpg" type="image/jpeg">
  
</picture>
```

使用此标记，理解 `image/webp` 媒体类型的浏览器将下载 Webp 图片并显示它，而其他浏览器将下载 JPEG 图片。

任何不支持 `<picture>` 的浏览器都将跳过所有 `source` 标签，并加载底部 `img` 标签。因此，我们通过提供对所有浏览器类的支持，逐步增强了我们的页面。



请注意，在所有情况下，`img` 标记都是实际呈现给页面的内容，因此它确实是语法的必需部分。 如果省略 `img` 标记，则不会渲染任何图像。

`<picture>` 标签和其中定义的所有 `source` 都在那里，以便浏览器可以选择要使用的图片的路径。 选择源图像后，其 URL 将传给 `img` 标记，这就是显示的内容。

这意味着你无需设置 `<picture>` 或 `source` 标记的样式，因为浏览器不会渲染这些标记。 因此，你可以像以前一样继续使用 `img` 标签进行样式设置。

总结

正如你所看到的，优化 web 上使用的图像的过程并不复杂，通过减少页面加载时间，可以为客户带来更好的用户体验，希望本文对你有帮助，共进步！

代码部署后可能存在的BUG没法实时知道，事后为了解决这些BUG，花了大量的时间进行log 调试，这边顺便给大家推荐一个好用的BUG监控工具 [Fundebug](#)。

原文：

<https://medium.freecodecamp.o...>

你的点赞是我持续分享好东西的动力，欢迎点赞！

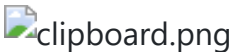
交流

干货系列文章汇总如下，觉得不错点个Star，欢迎 加群 互相学习。


<https://github.com/qg44924588...>


我是小智，公众号「大迂世界」作者，对前端技术保持学习爱好者。我会经常分享自己所学所看的干货，在进阶的路上，共勉！

关注公众号，后台回复**福利**，即可看到福利，你懂的。



阅读 7.3k • 更新于 9月25日

 赞 206

 收藏 151

 赞赏

 分享

本作品系 翻译 ([阅读原文](#))， 采用《署名-非商业性使用-禁止演绎 4.0 国际》许可协议



前端小智

前端开发工程师 37.9k

关注作者

2 条评论

得票 · 时间



撰写评论 ...

提交评论



rocky191： 性能优化没有穷尽啊

1 · 回复 · 3月6日



jifengg： 图像压缩确实是一个非常值得学习的方向。至少在项目中也要学会使用，现在流量最贵，能省则省

· 回复 · 3月6日

推荐阅读

web性能优化——优化内容

与桌面应用不同，网络应用不需要单独的安装过程：只需输入网址，便可启动和运行-这是网络的一个关键特色。不过，要做到这...

[Pines Cheng](#) · 阅读 20

前端性能优化(1)--减少HTTP请求

为什么减少HTTP请求能够优化性能？ HTTP请求建立和释放需要时间HTTP请求从建立到关闭一共经过以下步骤，这些步骤都是需要...

[桃子 Lisa](#) · 阅读 789

网站性能优化—CRP

为了把HTML、CSS和JavaScript转化成活灵活现、绚丽多彩的网页，浏览器需要处理一系列的中间过程，优化性能其实就是了解这...

[无名小贝勒](#) · 阅读 51

网站优化之路---图片优化，图片从模糊到清晰

作为一个有追求有信仰的程序员，做一个网站绝不是仅仅能用就行了，当我们实现功能后，或者在写代码的过程中就要考虑怎么去...

[leizore](#) · 阅读 16

网站性能优化-前端篇

我们的目标是优先显示与用户要在网页上执行的主要操作有关的内容。上面这句话出自《关键呈现路径》。前端性能，即页面性能...

[欧雷](#) · 阅读 46

web-性能优化

由于个人网站的博客没有阅读量，哈哈哈，就将所有博客移步到思否上面，同时也有人指出我的错误，让我能够发现我的问题，及...

[Winer](#) · 阅读 921

web前端优化之图片优化

开发前端也有几年了，一直很忙，课下看书，或者做一些笔记，看看别人的见解，也会做一些笔记记录，有时间就来刷刷掘金，逛...

[nyflxp](#) · 阅读 24

web性能优化

如何进行前端性能优化性能黄金法则：只有10%-20%的最终用户响应时间花在接收请求的HTML文档上，剩下的80%-90%时间花在...

灰熊 · 阅读 10

终身学习者

用户专栏

我要先坚持分享20年，大家来一起见证吧。

28766 人关注 242 篇文章

关注专栏

专栏主页



产品

[热门问答](#)
[热门专栏](#)
[热门课程](#)
[最新活动](#)
[技术圈](#)
[酷工作](#)
[移动客户端](#)

课程

[Java 开发课程](#)
[PHP 开发课程](#)
[Python 开发课程](#)
[前端开发课程](#)
[移动开发课程](#)

资源

[每周精选](#)
[用户排行榜](#)
[徽章](#)
[帮助中心](#)
[声望与权限](#)
[社区服务中心](#)

合作

[关于我们](#)
[广告投放](#)
[职位发布](#)
[讲师招募](#)
[联系我们](#)
[合作伙伴](#)

关注

[产品技术日志](#)
[社区运营日志](#)
[市场运营日志](#)
[团队日志](#)
[社区访谈](#)

条款

[服务条款](#)
[隐私政策](#)

[下载 App](#)

Copyright © 2011-2019 SegmentFault. 当前呈现版本 19.02.27

浙ICP备 15005796号-2 浙公网安备 33010602002000号 杭州堆栈科技有限公司版权所有

CDN 存储服务由 [又拍云](#) 赞助提供

