WEB 图像

高效的图像处理 - 倪创伟

前言

1. 图像优化必要性：图像占页面的比重 1/2
2. 常见图像格式：
    a. jpeg： 46% 24位
    b. png ： 30% ，分为 png8、png24、png32，透明
    c. gif ： 19% 8 位 + 透明
3. 光栅、矢量图像
4. 有损压缩

图像以什么样的形式存储的？300*300像素大小的图，若按照rgb各8位，需要270k

压缩质量 q 从何而来?

jpeg压缩-色彩模型与采样

1. 从三维的 RGB 模型转到 Y Cb Cr 模型转换

对于人眼图像的明暗变化更容易被感知，视杆细胞远远多于视锥细胞

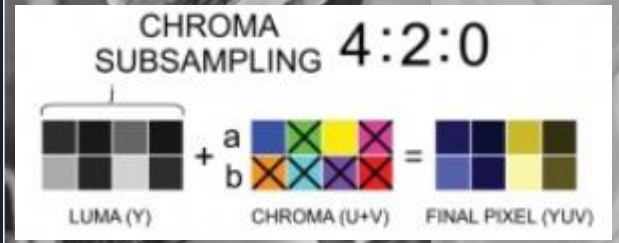
$$\begin{bmatrix} Y \\ \end{bmatrix} \quad \begin{bmatrix} 0.29900 & 0.58700 & 0.11400 \end{bmatrix} \begin{bmatrix} R \end{bmatrix}$$

2. 采样

Y Cb Cr

保留亮度信息，对 Cb 和 Cr 进行采样，4:2:2以及4:2:0

8*8的采样问题



大量数据表示，约有一半的jpeg格式采用4:2:0采样。

4:2:0 采样可以减少 17% 的体积大小

jpeg压缩-DCT与量化

3. 离散余弦变化 DCT

将复杂的数据变换成工整的规律，敏感部分和不敏感部分分离

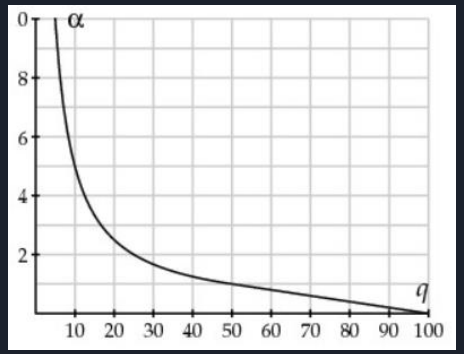
4. 量化

离散后，对频率系数进行量化，为 8*8 矩阵

同时分为色度量化和亮度量化

```
if (quality < 50) {
        sf = Math.floor(5000 / quality);
} else {
        sf = Math.floor(200 - quality*2);
}
```

$$Q_Y = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$




```
for (var i = 0; i < 64; i++) {
        var t = ffloor((YQT[i]*sf+50)/100);
        if (t < 1) {
                t = 1;
        } else if (t > 255) {
                t = 255;
        }
        YTable[ZigZag[i]] = t;
}
```

jpeg压缩-DCT与量化

20	-7	-1	1	-2	1	0	0
1	0	0	0	1	0	0	0
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Y

Q = 50 压缩后效果

高频部分被较高系数相除取整，剩下 0；

从而实现高频滤波

从图像的角度：颜色变化快速的部分会被模糊化，并且低质量下会出现块状色斑

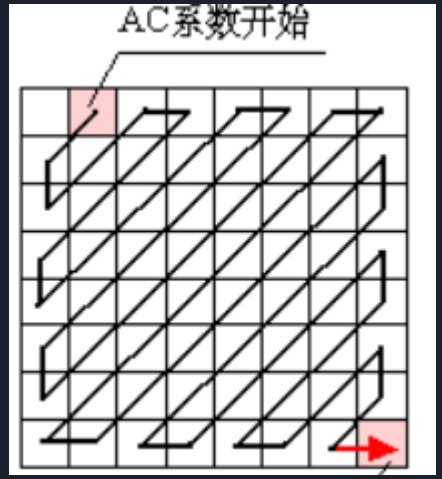
jpeg压缩-编码

5.DC/AC 编码

DC部分系数大，相邻模块变化小，采用插值编码

AC部分采用Z字排版，增加连续 0 系数的个数。

RLE压缩：不需要把所有 0 都记录下来



6.Huffman编码

给出现几率或频率较高的符号较短的编码，以提高压缩率

范式哈夫曼编码

最后 jpeg 保存的信息有：
图片标识、量化信息表、图像信息、huffman表、图像数据

jpeg-js

编译器-jpeg

最常见的编译器：mozJPEG（渐进式jpeg、量化设定、更好的量化表）jpegtran

基线jpeg和渐进式jpeg

```
const imagemin = require('imagemin');
const imageminMozjpeg = require('imagemin-mozjpeg');

(async () => {
  await imagemin(["images/*.jpg"], "build/images", {
    use: [
      imageminMozjpeg({
        progressive: true,
        quality: 85
      })
    ]
  });
})();
```

Image-webpack-loader：

```
test: /\.(gif|png|jpe?g|svg)$/i,
use: [
  'file-loader',
  {
    loader: 'image-webpack-loader',
    options: {
      mozjpeg: {
        progressive: true,
        quality: 65
      }
    }
  }
]
```

Guetzli：谷歌2016年底开源的编译器，获得更好的压缩质量，图像压缩后比同质量的libjpeg要小20%-30%。源于一个图片视觉差异评价工具 Butteraugli，分别进行全局微调和高频处理

imagemin-guetzli

编译器-其他

Png：无损压缩算法：LZ77

常见编译器 pngquant/optipng：有损编译器，采用量化，将24位或32位的RGBA PNG图转换成8位PNG图并保留透明度通道。

将总体颜色数减少到256以下（使用相近的颜色来代替），然后使用索引色彩来编码整张图片

Png8 为 256 索引色

Tinypng：采用量化技术，同时用了多种压缩技术：pngquant、optipng、advpng等，比 pngquant 要号10%。无法设置参数。

imageOptim：客户端，支持多个图片格式的压缩技术。仅支持 Mac

Imagine：Electron写得开源项目，支持多平台的压缩工具，基于 pngquant和mozjpeg

squoosh.app：在线或者本地处理单个图片进行调优，并支持格式转换对比

Webp与svg

Webp：2010年谷歌推出的图片格式，图像大小平均比png小26%，比同等质量的jpeg小25%-34%；

目前仅mac、ios以及IE不支持；ps：支持到 android 4.1 浏览器

如何应用webp?

1. 采用picture标签，使用多个source元素，达到兼容的目的
2. 服务端支持
    1. 浏览器检查是否支持webp格式，采用canvas或图片加载来判断是否支持webp
    2. 服务端通过请求头Accept来判断

对于安卓，colorOS 7.0 可以直接用 webp 图像

svg：矢量图像，可以采用svgo来压缩，简化路径、使用svg形状标签、降低精度

自适应图像

1. **art direction problem**，视觉风格，不同的屏幕不同的图像
2. **resolution switching problem**，像素密度，

srcset：浏览器选择图像

Sizes：定义媒体查询

```
<img
  srcset="./1@1.jpg 320w, ./1@2.jpg 640w, ./1@3.jpg 750w"
  sizes="(max-width: 320px) 280px,
         (max-width: 480px) 440px,
         800px"
/>
```

```
<body>
  <img srcset="./1@1.jpg, ./1@2.jpg 1.7x, ./1@3.jpg 2.3x" />
</body>
```

如何同时适配不同像素密度、不同大小的屏幕？

采用picture + source模式如：

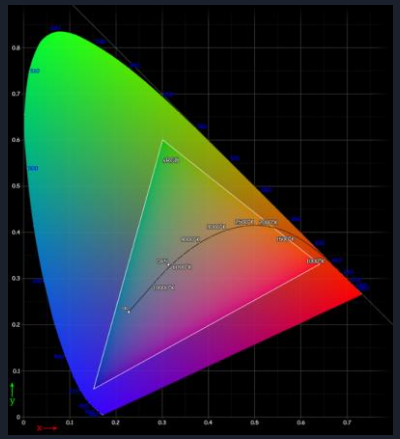
```
<picture>
  <source srcset="./1@3.jpeg, ./1@4.jpeg 2x" media="(min-width: 1000px)" />
  <img srcset="./1@1.jpeg, ./1@2.jpeg 2x" />
</picture>
```

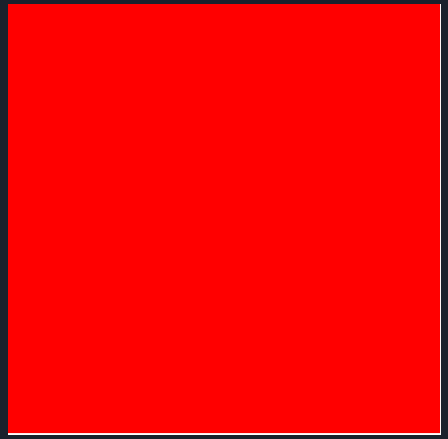
颜色

色彩空间：色彩空间提供了一个能够定义和比较颜色的环境，比如：LMS色彩空间（描述的是对视锥细胞的刺激），XYZ色彩空间，RGB色彩空间。

srgb: 标准 RGB 空间，以帮助确保在媒体之间忠实地传输色彩数据。sRGB是几乎在任何地方使用的默认色彩空间，也是浏览器使用的标准色彩空间。Css标准中指定的色彩空间。

色彩深度：电脑使用不同的精度或者深度来描述颜色，比如 css 里面的 rgb 语法，每条通道深度是 8 位。



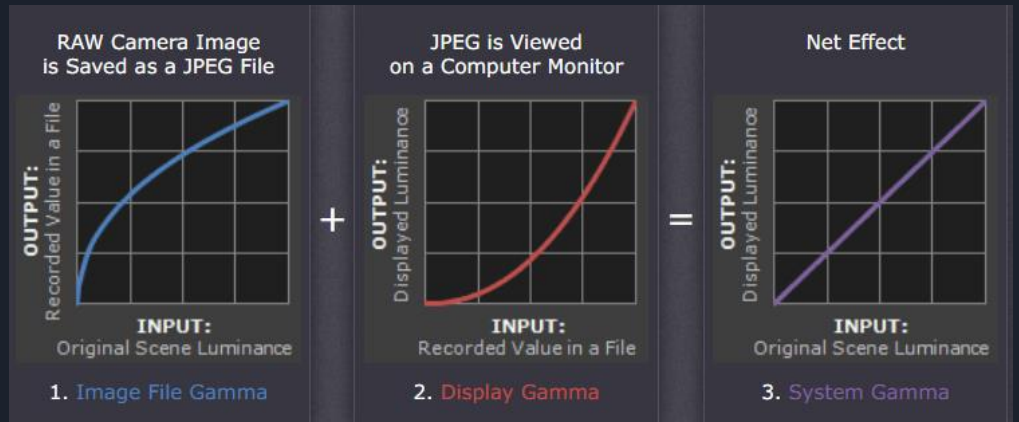


WebKit会用sRGB标准处理图片

中间的红色并不是全红色，用浏览器扫码右图

颜色-伽玛编码与校正

1. 对亮度的敏感程度
2. 有效的存储色调



什么时候才不用伽玛?

懒加载-滚动

常见的滚动方案：

性能问题触发重绘

```
window.addEventListener('scroll', () => {
    const rect = elem.getBoundingClientRect();
    const inViewport = rect.bottom > 0 && rect.right > 0 &&
                       rect.left < window.innerWidth &&
                       rect.top < window.innerHeight;
});
```

Intersection Observer

```
const onIntersection = (entries) => {
  for (const entry of entries) {
    if (entry.isIntersecting) {
      console.log(entry);
    }
  }
};

const observer = new IntersectionObserver(onIntersection);
observer.observe(document.querySelector('#some-target'));
```

存在问题?

无法解决遮盖问题，opatcity、transform等

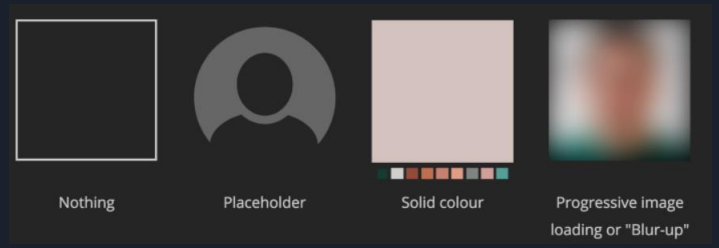
Intersection Observer：
trackVisibility 、delay
包含 isVisible 字段

常见库：lazysizes 是功能全面的延迟加载库

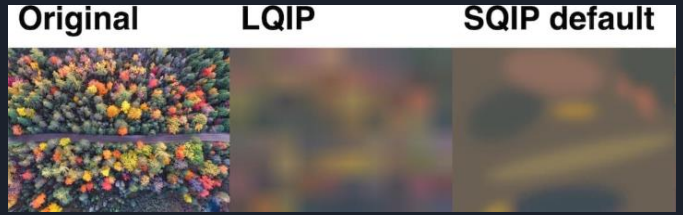
懒加载-占位符

常见的占位符：

知乎的占位符



1. 空白占位符：data:image/svg+xml;utf8，开头的svg 随后为黄色 div
2. LQIP(Low Quality Image Placeholders)， 1.7 kb 图
3. 高斯模糊
4. SQIP： a SVG-based LQIP technique



5. Svg 高斯模糊
6. 具体步骤：

Primitive

Blur

Svgo

data-uri

Transition 实现模糊渐变

No-script问题

懒加载-例子

Lazysizes： 支持 **LQIP**

```
<img
  data-sizes="auto"
  src="lqip-src.jpg"
  srcset="
  data-srcset="
    lqip-src.jpg 220w,
    image2.jpg 300w,
    image3.jpg 600w,
    image4.jpg 900w"
  class="lazyload"
/>
```

Vue-lazyload

```
Vue.use(VueLazyload, {
  preLoad: 1.3,
  error: 'dist/error.png',
  loading: 'dist/loading.gif',
  attempt: 1,
  observer: true,
  observerOptions: {
    rootMargin: '0px',
    threshold: 0.1
  }
})
<img
  v-lazy="'img.400px.jpg'"
  data-srcset="
    img.400px.jpg 400w,
    img.800px.jpg 800w,
    img.1200px.jpg 1200w"
>
```

The-platform

```
<React.Suspense maxDuration={300} fallback={'loading...'}>
  <Img src="img.jpg" srcset="" />
</React.Suspense>
```

Vue-lazyload可以添加

transition实现模糊渐变

思考题 – 大尺寸透明背景图

图片体积达到 501k，具有透明通道的 png 图片，无法融合到背景，必须透明，如何优化?

Tinypng 将 32位png 压成 8位png，效果极差，其他压缩无法达到高质量效果

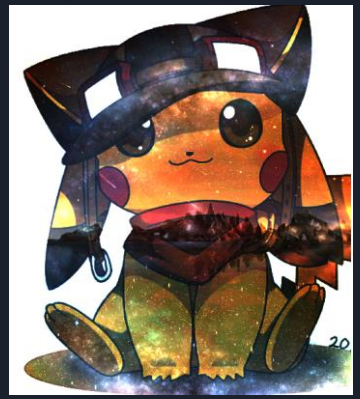
淘宝前端团队 给出webgl方案 webgl 实现透明通道，RGB生成jpeg图像，压缩

图像与css

通过CSS来处理图像的效果，只有CSS的混合模式和 滤镜相关的属性

mix-blend-mode：叠加色彩的混合，和ps的图层混合模式类似，有 normal darken lighten screen overlay 等模式

background-blend-mode：背景混合



Filter：滤镜，css 3 的滤镜，包括饱和度、灰度、模糊、invert等等

Invert 可用于黑夜模式

如：filter: brightness(2) invert(1)

https://www.cssfilters.co/

问答

学无止境