Web 前端水印实践

内容：

在作业二基础上，新增一个二级页面实现前端水印，包括：

1. 动态水印：基于 svg 的方案，水印需包含个人昵称+学号，可见；
2. 频域水印：基于傅立叶变换的方案，不可见。可参考文献 4

提交：

源文件（html、css、js 文件），readme（包括 1 的结果水印图像，2 的原始图像，以及使用方法的说明，如果是自行实现请注明，会有分数上的体现）。

Web 前端水印方案

前言

Web 水印技术在信息安全和版权保护等领域有着广泛的应用，对防止信息泄露或知识产品被侵犯有重要意义。

水印的添加根据环境可以分为两大类，前端浏览器环境添加和后端服务环境添加

水印根据可见性可分为可见水印和不可见水印（盲水印）

前端方案;

服务器运算量、内存的减少，能够快速响应请求
安全性较低，有心人很容易通过各种骚操作直接获取到源文件

后端方案

安全性较高，无法获取到源文件
当遇到大文件密集水印，或是复杂水印，占用服务器内存、运算量，请求时间过长

显性水印/可见水印

容易处理，算法较为简单
攻击者就可以通过裁剪、模糊等操作对水印进行攻击消除，同时显性水印也会破坏图片的完整性

创建时间	修改人	修改时间
2022-08-18 18:49:11	fransli	2022-08-18 18:49:11
2022-08-18 17:55:11	fransli	2022-08-18 17:55:11
2022-08-18 17:50:15	fransli	2022-08-18 17:50:15
2022-08-18 17:50:02		2022-08-18 17:50:02
2022-08-18 17:36:52	fransli	2022-08-18 17:36:52
2022-08-18 17:36:10	fransli	2022-08-18 17:36:10
2022-08-18 17:35:35	fransli	2022-08-18 17:35:35
2022-08-18 17:35:27		2022-08-18 17:35:27
2022-08-18 17:34:45		2022-08-18 17:34:45
2022-08-17 15:43:39		2022-08-17 15:43:39

`background-image:url("./logo.png");`
这里想要的效果就是一个浅浅的 logo 平铺展示。实现起来也比较简单，只需制作一个半透明的 logo 图片，设为文章或者表格的背景图片即可。仅需一行 CSS 声明。
实现图片平铺关键的 CSS 属性是 background-repeat，值为 repeat 时是平铺，这也是它的默认值，所以可以省略。

全页面水印

照葫芦画瓢，如果要给整个 Web 页面加上水印，是不是给页面的 body 元素设置背景图片平铺展示就可以了呢?
然而通常并不会这么处理，因为文章和表格内容多以文本为主，不会明显遮挡水印，而一个完整的页面往往还包含很多其他页面元素，比如图片、视频、控件等等，它们很可能会遮挡住背景图片，从而影响水印效果。
所以，为了避免被其他元素遮挡，针对页面的水印一般会使用一个层级比较高且覆盖整个页面的元素来承载。

```
div.watermark{
  position: fixed;
  left:0;
  top:0;
  width: 100vw;
  height: 100vh;
  background-image:url("./logo.png");
  opacity: .5;
  z-index: 3000;
}
```

这样一来，其他元素就遮挡不住水印了。不过，这个 div 反过来可能会遮挡页面其他元素，影响页面元素操作。还需要一条关键的 CSS 声明来破解这个问题 ：

pointer-events: none;

这个 CSS 声明会使该元素“可穿透”，“看得见、摸不着”，不再影响页面操作。

动态水印

很多时候，给页面加水印的目的并不是申明版权，而是为了支持溯源。此时水印的内容并不会只是一个 logo，通常会包含用户信息，比如用户名、UID、手机号等等。这就意味着，每个用户的水印内容是不同的，无法通过提前准备好一张图片来满足了。这种场景往往需要根据用户信息动态生成图片。

几种主流的动态生成水印图片的方式：

服务端方案

传统的方式是在服务端生成图片。页面上发起的图片请求中可以附带用户信息，服务端根据这些参数动态生成图片，并将图片数据作为该请求的响应返给页面，页面拿到后将其用作水印。
这种方式的优点是兼容性好，缺点是需要前后端配合，增加了页面请求和服务端资源开销，防攻击能力也较差。

***Canvas** 方案*

HTML5 引入 Canvas 特性使得浏览器自身具备了绘图能力。经过多年的发展，主流浏览器基本都已可以提供良好的支持。通过 Canvas 可以轻松绘制图片，并可将图片数据导出，用于页面图片或背景。

```
const canvasElement = document.createElement('canvas');
const context = canvasElement.getContext('2d');
canvasElement.width = 200;
canvasElement.height = 200;
context.rotate((-30 * Math.PI) / 180);
context.font = '400 26px Arial';
context.fillStyle = '#B9C0CA';
context.textAlign = 'center';
context.textBaseline = 'middle';
context.fillText('水印文字', 70, 130);
const watermark = canvasElement.toDataURL('image/png');
```

通过上述示例代码可拿到水印图片的 data URI 数据，用作水印承载元素的背景图片平铺展示即可。
这种方式不需要服务端配合，在前端就可以完成，且有助于减少请求和服务端资源开销。曾经面临的浏览器兼容问题现在也不再是问题，该方案已逐渐流行起来。

***SVG** 方案*

对于纯文字的水印来说，有没有办法不生成图片而直接实现平铺呢？

动态创建大量 DOM 节点，通过 CSS 控制排列当然可以实现，但是繁琐且性能差，优雅更无从谈起。
不妨换个角度思考，有没有办法让文字不转成图片就可以用作 background-image 属性的值呢？这样就可以利用 background-repeat 实现平铺效果了。
这时候可以考虑使用 SVG，因为 SVG 具有文本和图像的双重特性。看上去是文本，然而在很多场景可以当做图片使用。
我们可以通过 SVG 的相关属性精准控制字体位置、大小、颜色、透明度和旋转角度等参数。如：

```
<svg width="200" height="200" xmlns="http://www.w3.org/2000/svg">
  <text x="50%" y="50%" font-size="30" fill="#a2a9b6" fill-opacity="0.3" font-family="system-ui, sans-serif" text-anchor="middle" dominant-baseline="middle" transform='rotate(-45, 100 100)'>水印文字</text>
</svg>
```

考虑到浏览器兼容性，用作背景图片时，建议将 SVG 编码为 Base64（或转义特定字符）：

```
background-image: url("");
```

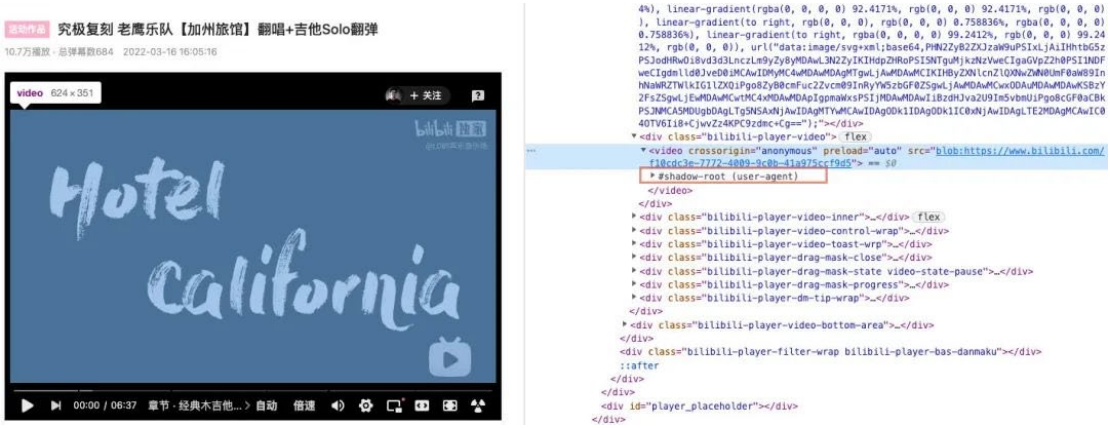
权限类型	生效时间	失效时间
普通账户	2022-01-17 10:51:12	2023-01-17 10:51:12
普通账户	2021-06-10 17:04:26	2022-07-02 00:00:00
普通账户	2019-05-31 16:10:26	2022-08-23 00:00:00
普通账户	2019-05-31 16:10:26	2022-08-23 00:00:00
普通账户	2019-05-31 16:10:26	2022-08-23 00:00:00
普通账户	2019-05-31 16:10:26	2022-08-23 00:00:00
普通账户	2019-05-31 16:10:26	2022-08-23 00:00:00
普通账户	2019-05-31 16:10:26	2022-08-23 00:00:00
普通账户	2019-05-31 16:10:26	2022-08-23 00:00:00
普通账户	2019-05-31 16:10:26	2022-08-23 00:00:00

水印安全
水印是用来保护信息安全的。信息要安全，首先要确保水印自身的安全，提高水印的防攻击（篡改、删除等）能力。
可见水印大都是基于 DOM 的，找到这个 DOM 节点，通过浏览器插件、抓包工具等在页面上注入一段 JavaScript 或者 CSS 代码对其进行篡改或删除并不困难。
防止外部代码篡改，一种思路是把水印元素封装起来，与外部环境进行隔离。

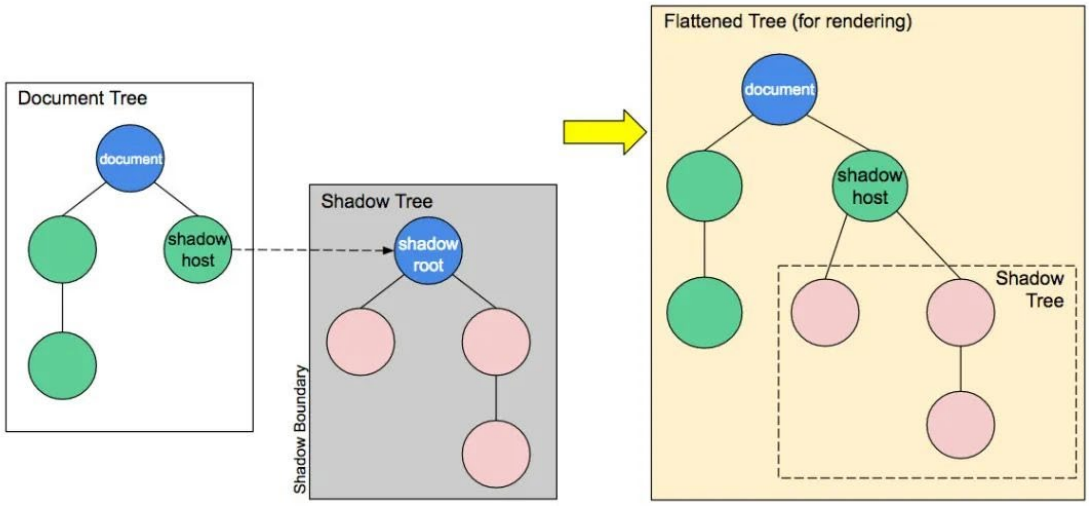
Shadow DOM

在 Chrome 逐步统治浏览器江湖之后，谷歌正野心勃勃的推广 Web Components 技术。该技术允许在 Web 中创建可重用的小部件或组件。组件化开发在前端业界已经流行相当长一段时间了，这主要得益于前端三大框架的推崇，但具体组件标准是由框架各自制定的，而 Web Components 可理解为 Web 标准化的组件。

Web Components 的一个重要特性就是“封装”，即可以将标记结构、样式和行为隐藏起来，并与页面上的其他代码相隔离。比如我们熟悉的 video 元素，它的进度条、按钮等控件都已被封装。



Shadow DOM 接口是“封装”特性的关键所在，它可以将一个隐藏的、独立的 DOM 附加到一个元素上。



为了提高 web 水印的隐蔽性，同时避免受外部代码影响，从而在一定程度上防止篡改，可以考虑把水印元素放在 Shadow DOM 中。

来看下 Shadow DOM 的基本用法。使用 Element.attachShadow() 方法来将一个 shadow root 附加到任何一个元素上。它接受一个配置对象作为参数，该对象有一个 mode 属性，值可以是 open 或者 closed 。open 表示可以通过页面内的 JavaScript 方法来获取 Shadow DOM。而 closed 则表示不可以从外部获取 Shadow DOM 。

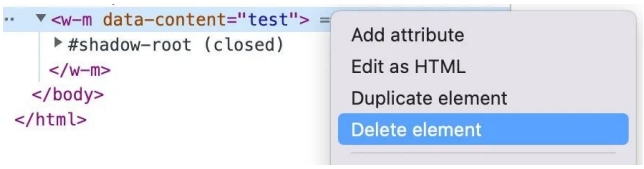
```
Element.attachShadow({mode: 'closed'});
```

```
<w-m data-content="test">
  #shadow-root (closed)
</w-m>
```

样式怎么隔离呢？Shadow DOM 中的样式本身就是隔离的，除非主动使用 CSS 变量、part 属性等暴露，外部样式是不会影响到组件内的。

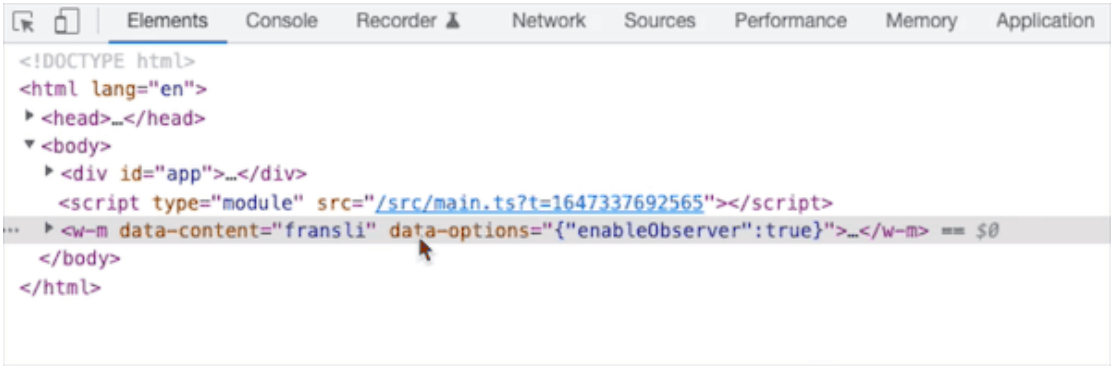
Mutation Observer

Shadow DOM 提高了水印的隐蔽性，同时可以防止外部代码修改。除此之外，还有一种常见的攻击场景——人为修改，比如在浏览器控制台直接修改或删除对应的 DOM 元素。



可以考虑“监听”这种行为，一旦发生就马上修复，比如重新插入一个。那怎么实现这种“监听”呢？现代浏览器中有多种观察者（Observer），比如 IntersectionObserver、PerformanceObserver、ResizeObserver、ReportingObserver、MutationObserver 等。其中，MutationObserver 就可以用来监听 DOM 变动，DOM 的任何变动，比如节点的增减、属性的变动、文本内容的变动，通过该 API 都可以得到通知。

所以可以使用 MutationObserver API 来监听水印元素 DOM 变化，一旦监听到 DOM 元素被修改或者删除，就立即重新插入一个。



不可见水印（盲水印）

不可见水印也叫盲水印、隐水印，顾名思义是一种看不到的水印，看不到还要它做什么呢？其实，不可见水印在一些对安全性要求较高的场景意义还是蛮大的。不可见水印通常具有比可见水印更好的隐蔽性和抗攻击性。虽不可见，但通过一定的技术手段是可以将水印信息从其载体上提取出来的，这就使得其载体具备了溯源能力，在关键时刻往往能发挥大作用。

前段时间阿里凭截图查到了月饼事件的泄密者，其实就是用了盲水印。AlloyTeam 团队写过一篇 不能说的秘密——前端也能玩的图片隐写术 ，通过 Canvas 给图片加上了“隐水印”，针对用户保存的图片，是可以轻松还原里面隐含的内容，但是对于截图或者处理过的照片却无能为力，不过对于一些机密图片文件展示，是可以偷偷用上该技术的。

当然，优缺点也需要分情况来看，各个方案都拥有自己的优缺点，需要使用者在安全性、性能之间衡量

没有最好的方案，只有根据环境与需求，使用当前最适合的方案

总结不可见水印相对可见水印至少有以下三个明显的优势：

1. 更好的观感。可见水印总给人一种“膏药感”，甚至会引起部分人的不适，而不可见水印则不会有这个问题。
2. 更佳的隐蔽性。用户基本感知不到水印的存在。
3. 更强的抗攻击性。可见水印更容易受到攻击，而不可见水印除了隐蔽性比较强之外，其自身往往还具备比较强的抗攻击能力。

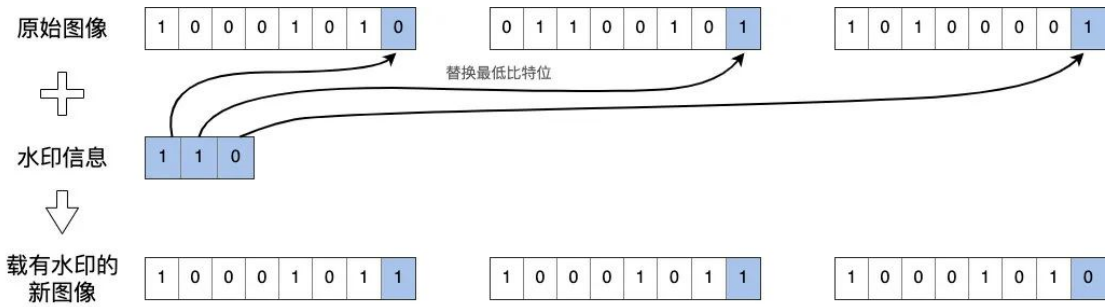
不可见水印（盲水印）属于信息隐匿技术（也叫隐写术），历史悠久，手段繁多。在现代，随着计算机网络技术的发展，数字产品的信息安全和版权保护也已成为信息隐匿技术的一个重要课题。隐写术在数字音频、数字视频和数字图像领域有着非常广泛的应用。Web 上基于 DOM 的盲水印大都不靠谱，而另一方面数字图像是信息隐藏和数字水印领域研究最多和最早的一种载体，**相较于 Web，数字图像领域有着更为成熟的数字水印算法**。我们不妨先来看下数字图像领域的常见盲水印技术。
在说数字水印之前，这里介绍一些数字图像的基础知识。
数字图像（位图）是由像素（pixel）组成。

- 非黑即白的二值图像，1 个 bit 即可表示 1 个像素（黑白两种状态）。所以 1 个字节（byte）可以表示 8 个像素。
- 灰度图，1 个像素有 256 种状态（2 的 8 次方），需要 8 个 bit，即 1 个字节。
- 彩色的 RGB 图，有 R / G / B 三个通道，每个通道 256 种状态，使用 1 个字节表示，共需 3 个字节表示 1 个像素。
- RGBA 图，在 R / G / B 三个通道基础上增加了一个透明度通道，256 种状态，额外需要 1 个字节，共需要 4 个字节表示一个像素。

通常，考虑到计算速度和性能，图像处理和图像识别大都会将图像转成灰度图或者选择其中一个通道进行。

LSB 水印

如上文所述，灰度图像的一个像素有 256 种状态，通常用灰度值（0-255）表示，0 表示黑色，255 表示白色，灰度值越大表示亮度越高。
灰度可用一个字节，即 8 比特二进制数表示，其中最高位对图像的贡献最大，最低位对图像的贡献最小，称为最低比特位（Least Significant Bit，LSB）。
如果将一个图像所有像素的比特位抽出来，就构成了 8 个不同的位平面，从 LSB（最低有效位 0）到 MSB（最高有效位 7）。位平面从低位到高位，图像的特征逐渐变得复杂，细节不断增加，相邻比特的相关性也越强。而比特位越低包含的图像信息就越少，最低位平面类似于随机噪声。因此，改变低位对图像的成像质量影响不大。
LSB 水印就是利用了这一点，用水印信息替换载体图像的最低比特位，这样原图像的 7 个高位平面就与表示水印信息的最低位平面组成了新的图像。



LSB 水印鲁棒性（防攻击性）较差，水印信息容易被抹去。

频域水印

将数字图像用一个矩阵来表示，是图像的空间域表示方法，LSB 就是在图像的空间域隐藏信息，鲁棒性较差。而在图像信号的频域（变换域）中隐藏信息要比在空间域中隐藏信息具有更好的鲁棒性。那么如何把图像信号从空间域转换到频域呢？这里就需要用到大名鼎鼎的 傅里叶变换 了。
法国数学家傅里叶大家一定不陌生，高数里就有傅里叶级数。

傅里叶提出的傅里叶变换（Fourier transform）理论，表示能将满足一定条件的某个函数表示成三角函数（正弦和/或余弦函数）或者它们的积分的线性组合，可用于把信号从时间域（或空间域）变换到频率域。

对于N点序列$\{x[n]\}_{0\leq n<N}$，它的离散傅里叶变换（DFT）为

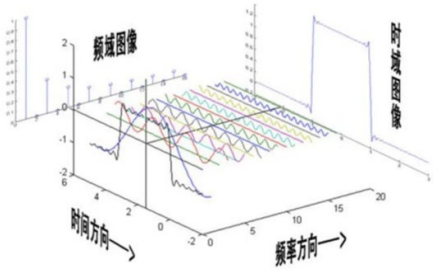
$$\hat{x}[k]=\sum_{n=0}^{N-1}e^{-i\frac{2\pi}{N}nk}x[n]\qquad k=0,1,\ldots,N-1.$$

其中e是自然对数的底数，i是虚数单位。

离散傅里叶变换的逆变换（IDFT）为：

$$x\left[n\right]=\frac{1}{N}\sum_{k=0}^{N-1}e^{i\frac{2\pi}{N}nk}\hat{x}[k]\qquad n=0,1,\ldots,N-1.$$

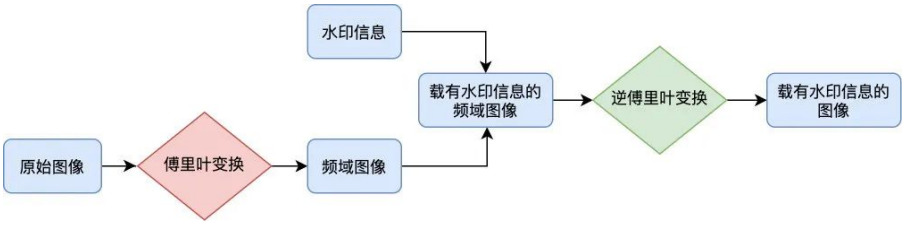
在此之前人们对信号的分析主要集中在空间域，傅里叶变换的提出为频域分析奠定了基础，有助于解决许多图像的问题。



傅里叶变换在数字图像处理领域有着极为重要的应用，图像领域变换的实质是把图像函数展开成具有不同空间频率的正、余弦信号的叠加，也就是说任何图像都可以分解为若干个频率不同的亮度呈正弦变化的图像之和。把图像从空间域变换到频率域后，就能够实现对图像数据进行不同频率成分的提取。对于图像信号来说，可以把灰度（亮度）看做频率，傅里叶变换可作为图像灰度值形成的空间域与其频率域的桥梁。

在频域中隐藏信息就是傅里叶变换在数字图像处理领域的一个典型应用场景。通常多选择在图像频域的中频部分嵌入信息，因为高频部分易于被各种信号处理方法破坏，而人的视觉又对低频部分比较敏感，容易察觉低频部分的变化。

在图像频域嵌入水印信息的大致流程为：把原始图像通过离散傅里叶变换转换到频域（变换域），把水印文字信息混入，再经过离散傅里叶变换的逆变换，便得到了载有水印信息的图像。水印信息隐匿性较好。



Web 中的数字水印应用

上面介绍了几种常见的不可见水印（盲水印）实现方式，其中鲁棒性比较好的是基于频域的数字图像盲水印，但这种水印主要是针对数字图像，而 Web 上的内容载体并不一定都是图片，常见的需要加水印的载体除了图片还有文本、表格等，这些场景该如何应用频域盲水印呢？

或许，Canvas 就是答案。

Reference

- Web Components
- shadow DOM
- 如何让文字作为 CSS 背景图片显示？
- 《数字图像隐写分析》
- 《数字图像处理原理与实践》
- 《数据隐藏技术揭秘》

显性水印+隐藏水印方案

数字水印

Base64 编码原理
Base64 实现例子
傅里叶转换数字水印
canvas 添加水印教程
Data URL
Mutationobserver

参考文献:

1. 探秘 Web 水印技术，https://mp.weixin.qq.com/s/FNuVlGLt3Zi8COVAdGBKRw

2. 不能说的秘密——前端也能玩的图片隐写术，
http://www.alloyteam.com/2016/03/image-steganography/

3. 隐写术、盲水印入门知识体系
https://www.guofei.site/2020/11/14/blind_watermark.html

4. https://github.com/zeruniverse/CryptoStego

附录
Data URL 示例
<img
src="
byI/PjwhRE9DVFlQRSBzd
mcgUFVCTElDICItLy9XM0MvL0RURCBTVkcgMS4xLy9FTiIgImh0dHA6Ly93d3cudzMub3J
nL0dyYXBoaWNzL1NWRy8xLjEvRFREL3
N2ZzExLmR0ZCI+PHN2ZyB0PSIxNTgwODkyMjMyMDU3IiBjbGFzcz0iaWNvbiIgdmlld0Jv
eD0iMCAwIDEwMjQgMTAyNCIgdmVyc2I
vbj0iMS4xIiB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHAtaWQ9IjIwNz
Y0IiB4bWxuczp4bGluaz0iaHR0cDov
L3d3dy53My5vcmcvMTk5OS94bGluayIgd2lkdGg9IjEyOCIgaGVpZ2h0PSIxMjgiPjxzd
PjxzdHlsZSB0eXBlPSJ0ZXh0L2Nzc
yI+PC9zdHlsZT48L2RlZnM+PHBhdGggZD0iTTIyNi40MDY0IDMyNi43NTg0bS00MC45NiA
wYTQwLjk2IDQwLjk2IDAgMSAxIDgxLj
kyIDAgNDAuOTYgNDAuOTYgMCAxIDAtODEuOTIgMFoiIHAtaWQ9IjIwNzY1IiBmaWxsP
SIjRkY1NzIyIj48L3BhdGg+PHBhdGggZD0
iTTgzOC4zNDg4IDQzNC4yNzg0YTIwLjQ4IDIwLjQ4IDAgMCAxLTE0LjQzODQtNi4wNDE2
TDczNy4yOCAzNDEuMjk5MmEyMC40OCAy
MC40OCAwIDAgMSAwLTI4Ljc3NDRMODI0LjExNTlgMjI1Ljl4YTIwLjQ4IDIwLjQ4IDAgMC
AxIDI4Ljk3OTlgMjguOTc5MmwtNzluN
jAxNiA3Mi42MDE2TDg1Mi43ODcyIDM5OS4zNmEyMC40OCAyMC40OCAwIDAgMS0xNC40Mzg0IDM0LjkxODR6TTUxMiA4MDQuNzYxNm

MtMTUwLjkzNzYgMC0yNzkuNTUyLTEwMS43ODU2LTMwNS45NzEyLTI0MS45NzEyYTIw
LjQ4IDIwLjQ4IDAgMCAxIDQwLjI0MzItNy4
1Nzc2QzI2OS4yMDk2IDY3NS44NCAzODAuOTI4IDc2My44MDE2IDUxMiA3NjMuODAxN
lM3NTUuMiA2NzUuODQgNzc4LjI0IDU1NS4y
MTI4YTIwLjQ4IDIwLjQ4IDAgMCAxIDM5LjkzNiA3Ljk4NzJDNzkxLjc1NjggNzAyLjk3NiA2Nj
MuMTQyNCA4MDQuNzYxNiA1MTIgO
DA0Ljc2MTZ6IiBwLWlkPSIyMDc2NiIgZmlsbD0iI0ZGNTcyMiI+PC9wYXRoPjwvc3ZnPg=="
>

Data URL to Canvas

```
function dataURLToCanvas(dataurl, cb){
	var canvas = document.createElement('CANVAS');
	var ctx = canvas.getContext('2d');
	var img = new Image();
	img.onload = function(){
		canvas.width = img.width;
		canvas.height = img.height;
		ctx.drawImage(img, 0, 0);
		cb(canvas);
	};
	img.src = dataurl;
}
```