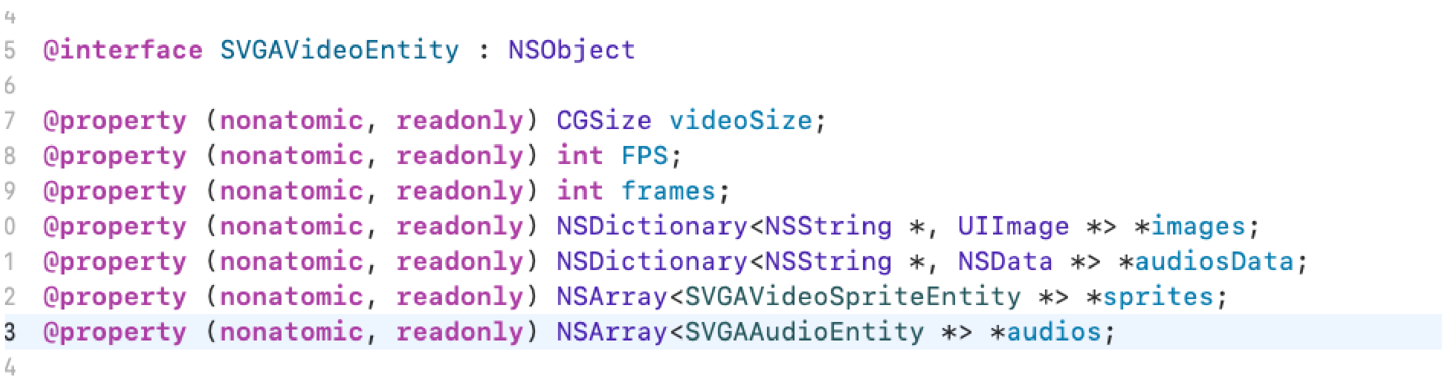
SVGA动画资源内存优化

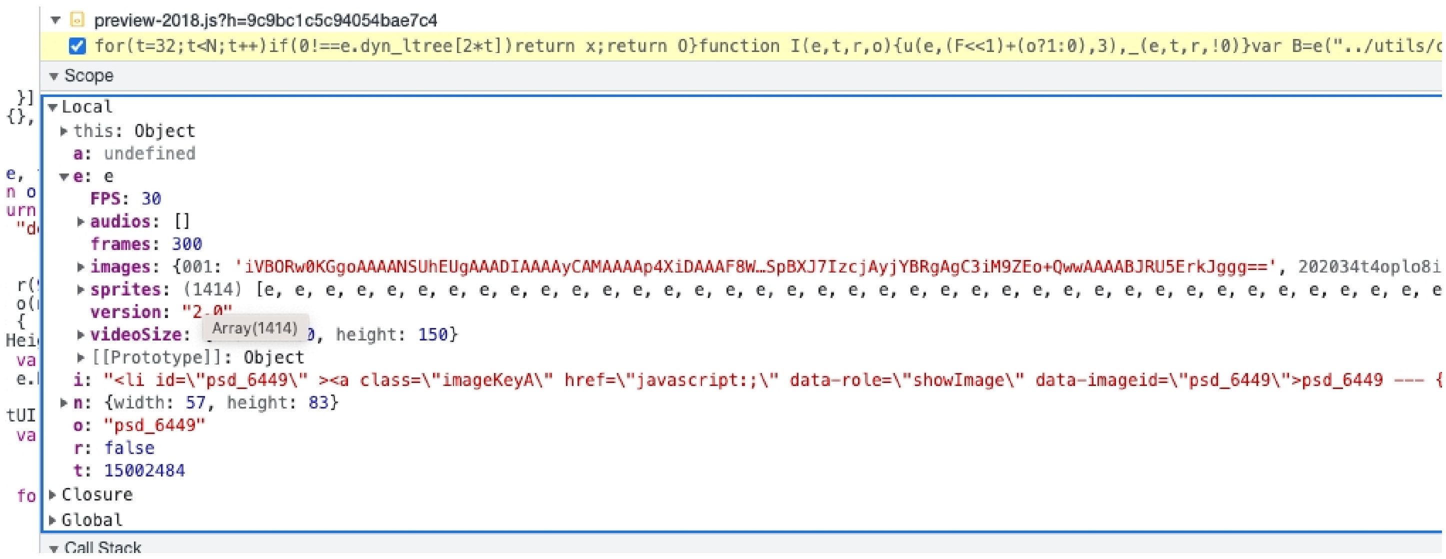
一、SVGA动画绘制原理

通过多个相同尺寸的图层叠加最终合成一张图成为动画的一帧，每个图层持有着相同的帧数，同步地进行切换，完成一帧帧的绘制连贯成动画。

动画数据格式：



```objc
@interface SVGAVideoEntity : NSObject

@property (nonatomic, readonly) CGSize videoSize;
@property (nonatomic, readonly) int FPS;
@property (nonatomic, readonly) int frames;
@property (nonatomic, readonly) NSDictionary<NSString *, UIImage *> *images;
@property (nonatomic, readonly) NSDictionary<NSString *, NSData *> *audiosData;
@property (nonatomic, readonly) NSArray<SVGAVideoSpriteEntity *> *sprites;
@property (nonatomic, readonly) NSArray<SVGAAudioEntity *> *audios;

```

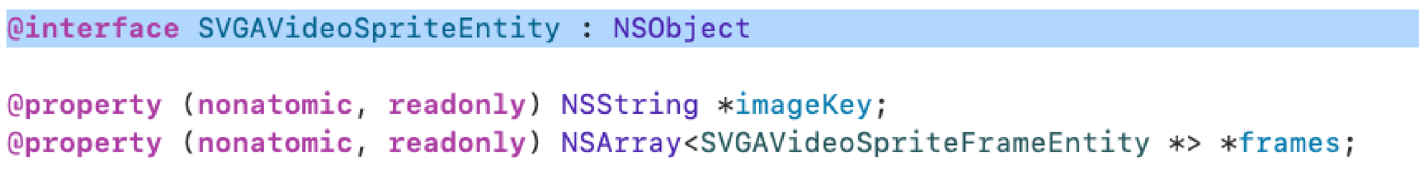


```
▼ preview-2018.js?h=9c9bc1c5c94054bae7c4
  ☑ for(t=32;t<N;t++)if(0!==e.dyn_ltree[2*t])return x;return 0}function I(e,t,r,o){u(e,(F<<1)+(o?1:0),3),_(e,t,r,!0)}var B=e("../utils/c
▼ Scope
▼ Local
  ▸ this: Object
    a: undefined
  ▼ e: e
      FPS: 30
    ▸ audios: []
      frames: 300
    ▸ images: {001: 'iVBORw0KGgoAAAANSUhEUgAAADIAAAAyCAMAAAAp4XiDAAAF8W…SpBXJ7IzcjAyjYBRgAgC3iM9ZEo+QwwAAAABJRU5ErkJggg==', 202034t4oplo8i
    ▸ sprites: (1414) [e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e
      version: "2.0"
    ▸ videoSize:  Array(1414) 0, height: 150}
    ▸ [[Prototype]]: Object
    i: "<li id=\"psd_6449\" ><a class=\"imageKeyA\" href=\"javascript:;\" data-role=\"showImage\" data-imageid=\"psd_6449\">psd_6449 --- {
  ▸ n: {width: 57, height: 83}
    o: "psd_6449"
    r: false
    t: 15002484
▸ Closure
▸ Global
▼ Call Stack
```

其中：

images存储着动画需要的图片，使用map存储

sprites存储着需要的图层，使用数组存储，数组中的元素通过imageKey从images里取出需要用到的图片，所以我们可以通过修改imagekey取图片的方式动态的添加我们设定的图片，例如替换头像等。



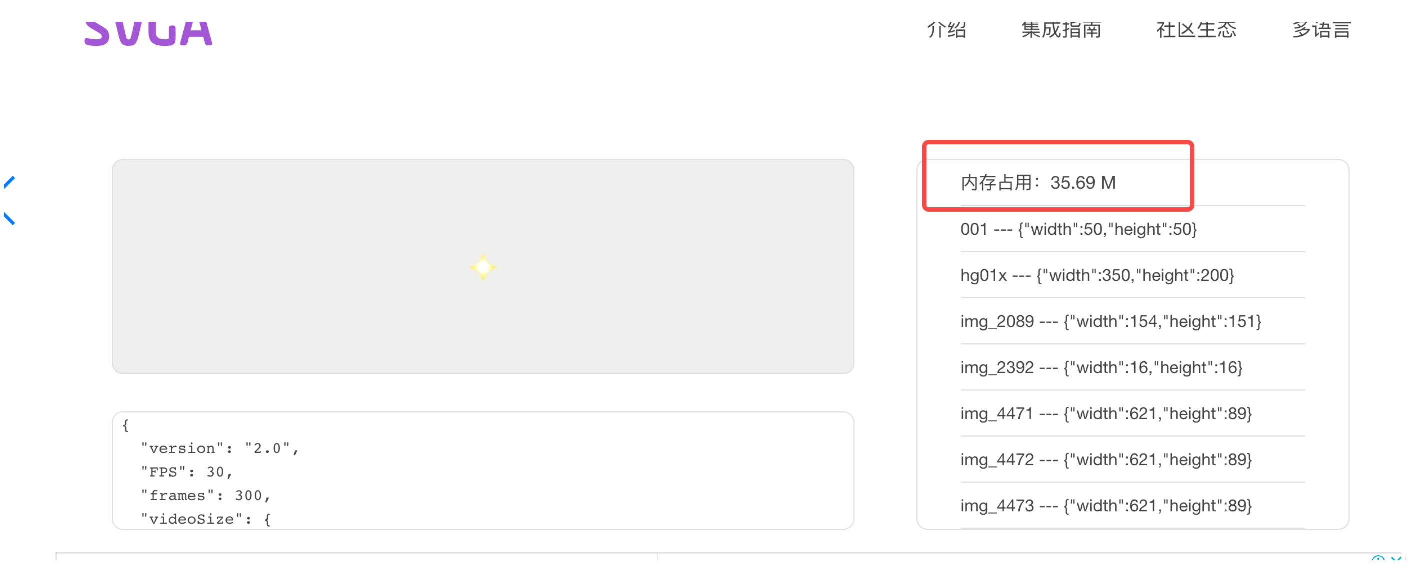
```objc
@interface SVGAVideoSpriteEntity : NSObject

@property (nonatomic, readonly) NSString *imageKey;
@property (nonatomic, readonly) NSArray<SVGAVideoSpriteFrameEntity *> *frames;
```

audios等数据则是音频相关，通过AVAudioPlayer进行播放，暂时因为没有使用到，不做考虑

二、内存占用

通过上面的分析，我们发现SVGA动画资源的内存占用主要在images和sprites，分别为图片bitmap占用和动画参数的占用。

1. bitmap占用我们可以直接从 https://svga.io 上查看到



内存占用：35.69 M

001 --- {"width":50,"height":50}

hg01x --- {"width":350,"height":200}

img_2089 --- {"width":154,"height":151}

img_2392 --- {"width":16,"height":16}

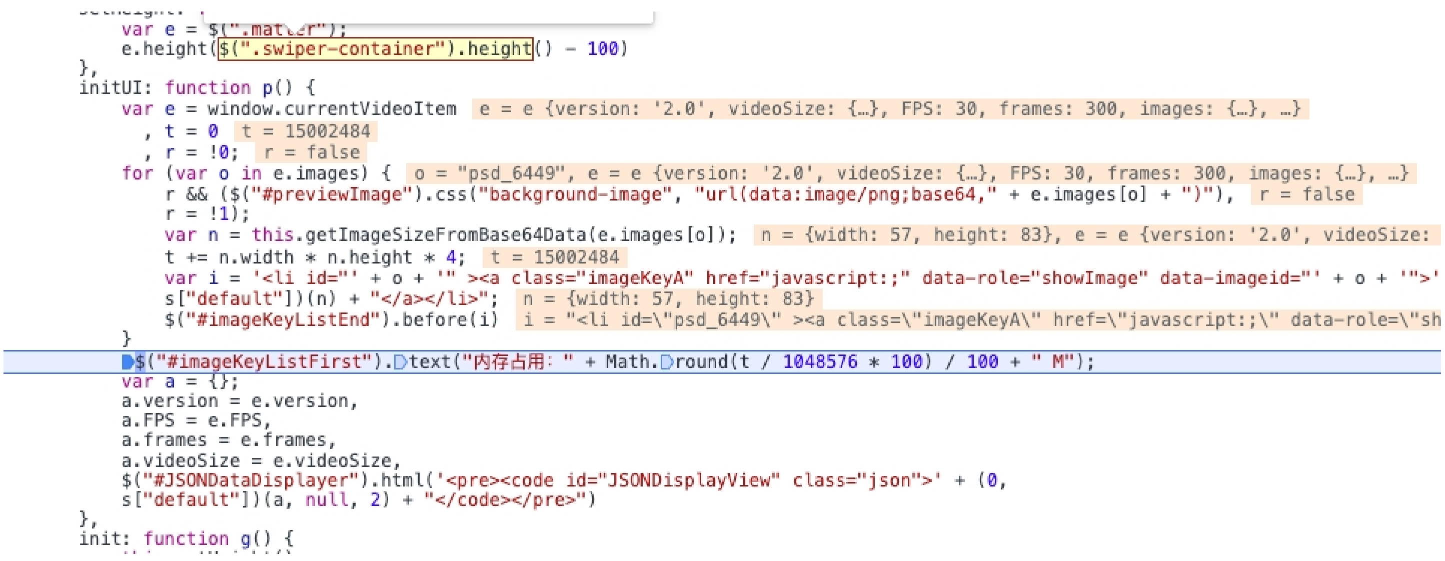
img_4471 --- {"width":621,"height":89}

img_4472 --- {"width":621,"height":89}

img_4473 --- {"width":621,"height":89}

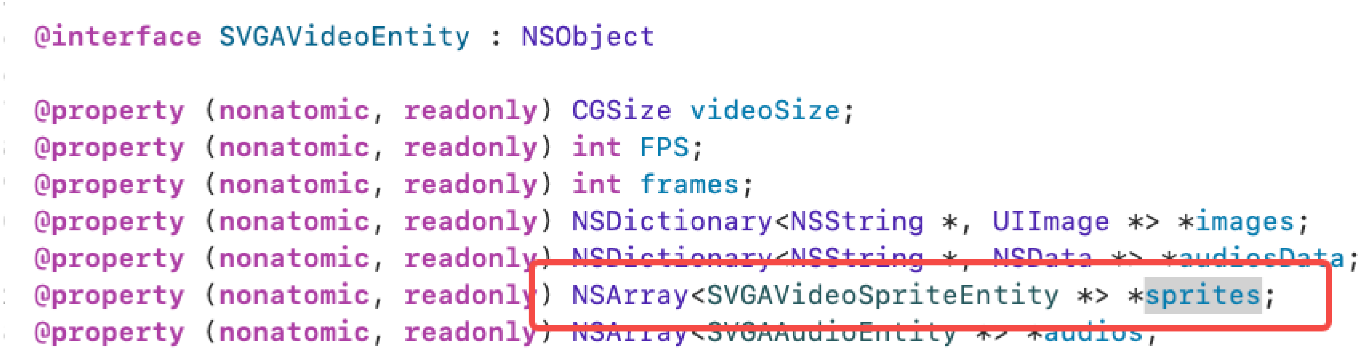
```
{
  "version": "2.0",
  "FPS": 30,
  "frames": 300,
  "videoSize": {
```

检查网页源码可以发现记录地确实是图片占用



```
        var e = $(".matter");
        e.height($(".swiper-container").height() - 100)
    },
    initUI: function p() {
        var e = window.currentVideoItem  e = e {version: '2.0', videoSize: {…}, FPS: 30, frames: 300, images: {…}, …}
          , t = 0  t = 15002484
          , r = !0;  r = false
        for (var o in e.images) {  o = "psd_6449", e = e {version: '2.0', videoSize: {…}, FPS: 30, frames: 300, images: {…}, …}
            r && ($("#previewImage").css("background-image", "url(data:image/png;base64," + e.images[o] + ")"),  r = false
            r = !1);
            var n = this.getImageSizeFromBase64Data(e.images[o]);  n = {width: 57, height: 83}, e = e {version: '2.0', videoSize:
            t += n.width * n.height * 4;  t = 15002484
            var i = '<li id="' + o + '" ><a class="imageKeyA" href="javascript:;" data-role="showImage" data-imageid="' + o + '">'
            s["default"])(n) + "</a></li>";  n = {width: 57, height: 83}
            $("#imageKeyListEnd").before(i)  i = "<li id=\"psd_6449\" ><a class=\"imageKeyA\" href=\"javascript:;\" data-role=\"sh
        }
        $("#imageKeyListFirst").text("内存占用: " + Math.round(t / 1048576 * 100) / 100 + " M");
        var a = {};
        a.version = e.version,
        a.FPS = e.FPS,
        a.frames = e.frames,
        a.videoSize = e.videoSize,
        $("#JSONDataDisplayer").html('<pre><code id="JSONDisplayView" class="json">' + (0,
        s["default"])(a, null, 2) + "</code></pre>")
    },
    init: function g() {
```

2. 动画参数的占用则主要是sprites数组

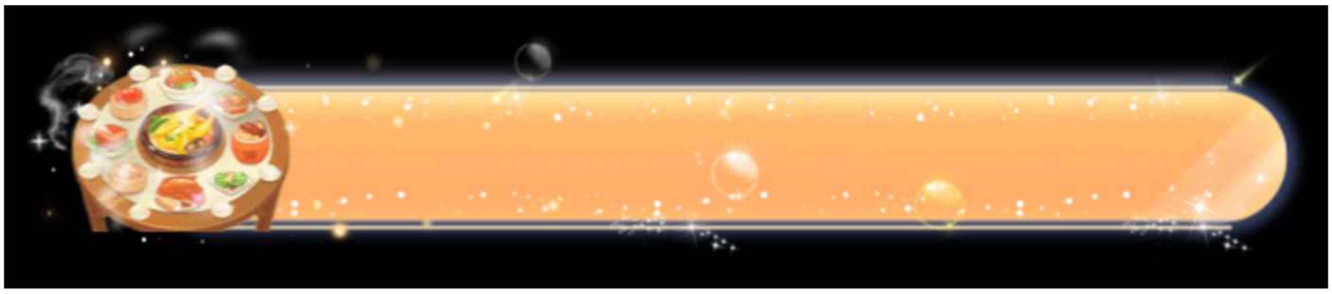


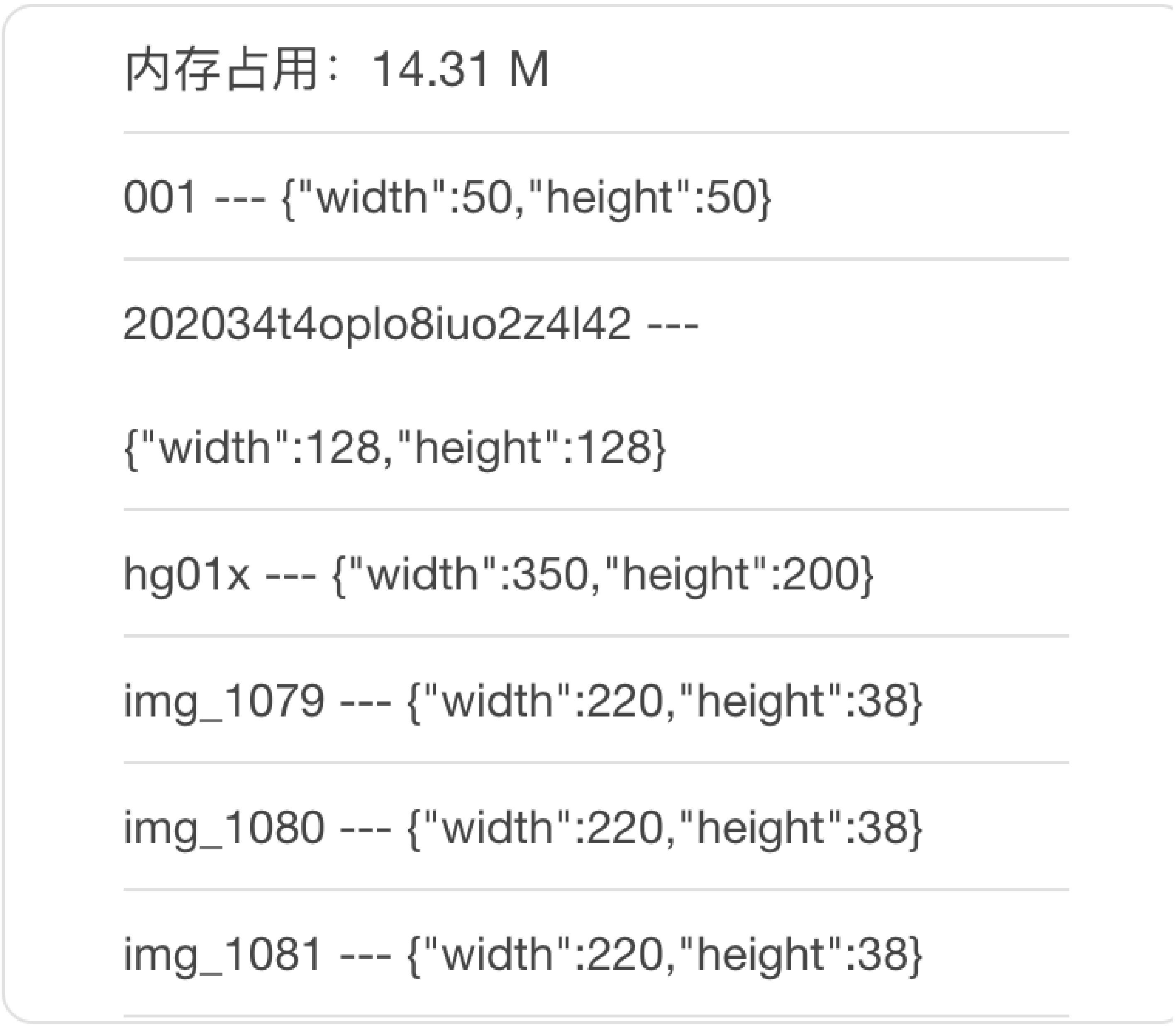
```
@interface SVGAVideoEntity : NSObject

@property (nonatomic, readonly) CGSize videoSize;
@property (nonatomic, readonly) int FPS;
@property (nonatomic, readonly) int frames;
@property (nonatomic, readonly) NSDictionary<NSString *, UIImage *> *images;
@property (nonatomic, readonly) NSDictionary<NSString *, NSData *> *audiosData;
@property (nonatomic, readonly) NSArray<SVGAVideoSpriteEntity *> *sprites;
@property (nonatomic, readonly) NSArray<SVGAAudioEntity *> *audios;
```

```
@interface SVGAVideoSpriteEntity : NSObject

@property (nonatomic, readonly) NSString *imageKey;
@property (nonatomic, readonly) NSArray<SVGAVideoSpriteFrameEntity *> *frames;
```

```
@interface SVGAVideoSpriteFrameEntity : NSObject

@property (nonatomic, readonly) CGFloat alpha;
@property (nonatomic, readonly) CGAffineTransform transform;
@property (nonatomic, readonly) CGRect layout;
@property (nonatomic, readonly) CGFloat nx;
@property (nonatomic, readonly) CGFloat ny;
@property (nonatomic, readonly) CALayer *maskLayer;
@property (nonatomic, readonly) NSArray *shapes;
```

例如 https://sca.tcsdzz.com/snake_file_1682602214_tftb.svga 动画，占用了88M内存





内存占用：14.31 M

001 --- {"width":50,"height":50}

202034t4oplo8iuo2z4l42 ---

{"width":128,"height":128}

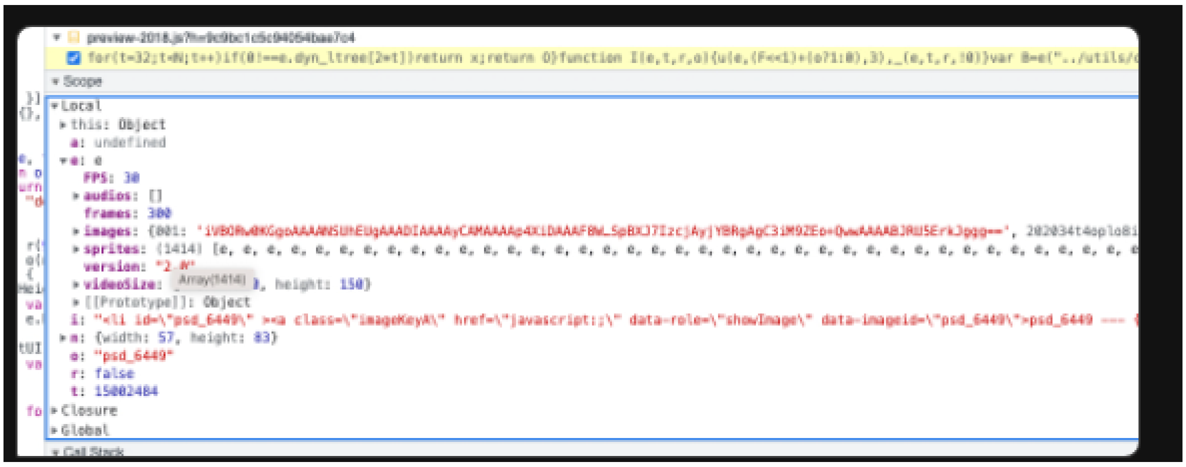
hg01x --- {"width":350,"height":200}

img_1079 --- {"width":220,"height":38}

img_1080 --- {"width":220,"height":38}

img_1081 --- {"width":220,"height":38}

图片占用为14.31M，动画参数占用则为 74M



```
▼ preview-2018.js?h=9c9bc1c5c94054bae7c4
☑ for(t=32;t<N;t++)if(0!==e.dyn_ltree[2*t])return x;return 0}function I(e,t,r,o){u(e,(F<<1)+(o?1:0),3),_(e,t,r,!0)}var B=e("../utils/c
▼ Scope
▼ Local
  ▸ this: Object
    a: undefined
  ▼ e: e
      FPS: 30
    ▸ audios: []
      frames: 300
    ▸ images: {001: 'iVBORw0KGgoAAAANSUhEUgAAADIAAAAyCAMAAAAp4XiDAAAF8W_SpBXJ7IzcjAyjYBRgAgC3iM9ZEo=QwwAAAABJRU5ErkJggg==', 202034t4oplo8i
    ▸ sprites: (1414) [e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e, e
      version: "2.0"
    ▸ videoSize: Array(1414) }, height: 150}
    ▸ [[Prototype]]: Object
    i: "<li id=\"psd_6449\" ><a class=\"imageKeyA\" href=\"javascript:;\" data-role=\"showImage\" data-imageid=\"psd_6449\">psd_6449 --- {
  ▸ m: {width: 57, height: 83}
    o: "psd_6449"
    r: false
    t: 15002484
▸ Closure
▸ Global
▼ Call Stack
```

究其原因，查看其动画参数发现需要 1414个图层，每个图层 300 帧，这不仅需要1414个图层叠加，也需要 1414 * 300 个对象进行存储。

三、如何优化?

1. 缓存优化

为什么需要缓存?

因为解析动画资源数据需要一定的时间，为了更快的展示出动画，通常需要进行缓存，尤其是列表页面，多个头像框动画，如果没有缓存，页面滑动会变的卡顿。

> 值得一提的是 1400多个甚至1600多个图层不是个好的选择，这给绘制也带来了极大的压力，如果需要这么多图层才能绘制，是不是可以考虑换一种格式?

我们对缓存加了一些限制，最多缓存20个，最大200M，这种缓存主要是针对列表头像框的，实测头像框的内存占用在 3 - 15M之间（未进行下述优化之前），所以上述缓存能够满足一般的页面需要。

2. 通用地对动画参数进行优化

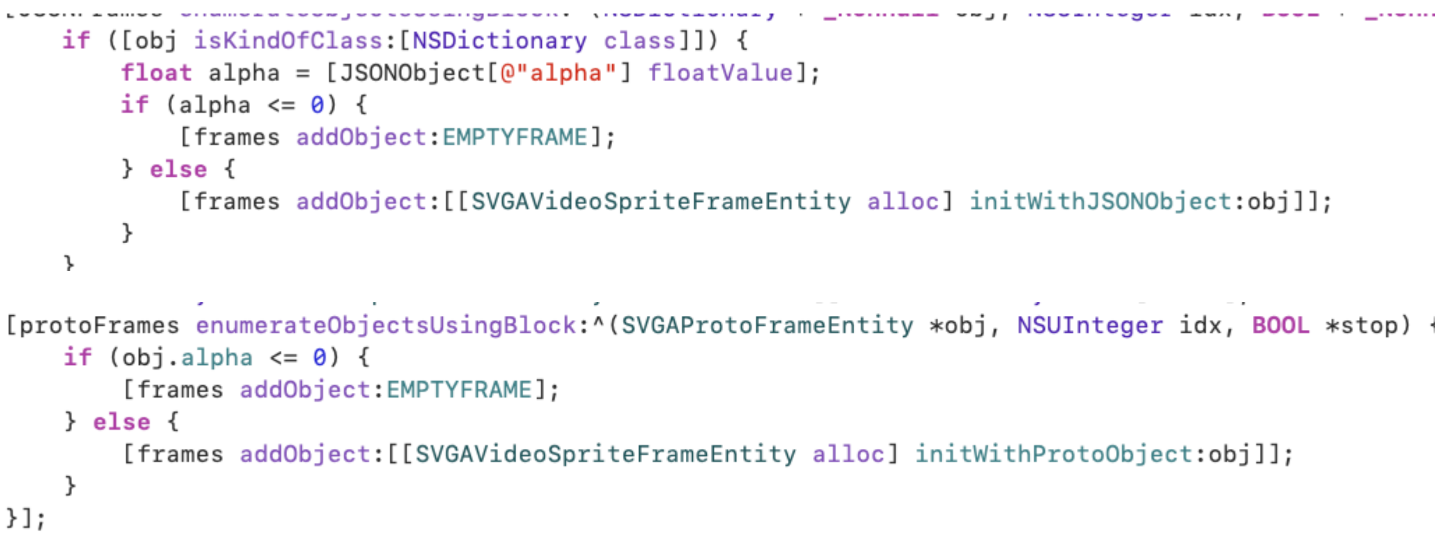
bitmap的占用属于动画必要的资源，从代码层面不太能对其进行优化，只能对动画参数进行优化

王威

分析解码中的每帧数据

```
@interface SVGAVideoSpriteFrameEntity : NSObject

@property (nonatomic, readonly) CGFloat alpha;
@property (nonatomic, readonly) CGAffineTransform transform;
@property (nonatomic, readonly) CGRect layout;
@property (nonatomic, readonly) CGFloat nx;
@property (nonatomic, readonly) CGFloat ny;
@property (nonatomic, readonly) CALayer *maskLayer;
@property (nonatomic, readonly) NSArray *shapes;
```

发现当alpha值为0时，我们会把相应的图层隐藏，不参与绘制。这一部分我们可以直接让其指向同一个空对象，这样可以减少大量的空对象存储。



```
    if ([obj isKindOfClass:[NSDictionary class]]) {
        float alpha = [JSONObject[@"alpha"] floatValue];
        if (alpha <= 0) {
            [frames addObject:EMPTYFRAME];
        } else {
            [frames addObject:[[SVGAVideoSpriteFrameEntity alloc] initWithJSONObject:obj]];
        }
    }

[protoFrames enumerateObjectsUsingBlock:^(SVGAProtoFrameEntity *obj, NSUInteger idx, BOOL *stop) {
    if (obj.alpha <= 0) {
        [frames addObject:EMPTYFRAME];
    } else {
        [frames addObject:[[SVGAVideoSpriteFrameEntity alloc] initWithProtoObject:obj]];
    }
}];
```

```
SVGAVideoSpriteFrameEntity *frameItem = self.frames[frame];
if ([frameItem isEqual:EMPTYFRAME]) {
    self.hidden = YES;
}else {
    self.hidden = NO;
    self.opacity = frameItem.alpha;
    CGFloat nx = frameItem.nx;
    CGFloat ny = frameItem.ny;
    self.position = CGPointMake(0, 0);
    self.transform = CATransform3DIdentity;
    self.frame = frameItem.layout;
    self.transform = CATransform3DMakeAffineTransform(frameItem.transform);
    CGFloat offsetX = self.frame.origin.x - nx;
    CGFloat offsetY = self.frame.origin.y - ny;
    self.position = CGPointMake(self.position.x - offsetX, self.position.y - offsetY);
    self.mask = frameItem.maskLayer;
    [self.bitmapLayer stepToFrame:frame];
    [self.vectorLayer stepToFrame:frame];
}
```

在修改SVGA源码中的解析部分后，我们发现很多动画中有80%以上的解码数据都是空对象，上述动画所需要的内存直接从88M降到了17M

另一个广播秀招财貔貅直接从 128M 变成了 38M （其中图片内存占用为 35.69M，1662个图层 300 帧）

3. 结合实际应用进行优化

由于业务中有很多动画只需要执行一次，然后保留着最后一帧的绘制，svga源码处只是停止了定时器进行帧切换，但是依旧保留了 SVGAVideoEntity 对象，并且保留了1414个图层(上例中)。

针对这种场景，我们直接重新绘制了最后一帧成为一张图片，然后释放掉图层和SVGAVideoEntity对象，只保留了一张图片进行展示，这样几乎可以将动画所需要的内存全部释放掉，转而类似于一个普通的图片控件

```
if (!self.clearsAfterStop && self.videoItem) { // 保留当前帧内容
    UIGraphicsBeginImageContextWithOptions(self.layer.frame.size, NO, 0.0);
    [self.layer renderInContext:UIGraphicsGetCurrentContext()];
    UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    self.layer.contents = (__bridge id _Nullable)([image CGImage]);
}
[_drawLayer removeFromSuperlayer];
_drawLayer = nil;
_audioLayers = nil;
_videoItem = nil;
[self clearDynamicObjects];
```

4. 具体针对特定场景（例如游戏等）优化 （暂时还没有做）

对于部分播放完成以后需要停止的动画我们采用了截取最后一帧来进行优化，但是部分动画控件是无限循环的，这种动画显然不太适合进行释放，但是我们依旧希望在进入游戏场景或者一些特定页面时

进行释放，退出以后再进行恢复。

思路：

1. 需要统一控件的初始化入口，记录页面中还持有的控件，在设置 SVGAVideoEntity 时记录下 url 或者 filePath 以便后续恢复
2. 在进入游戏页面时，我们对依旧持有着的控件停止播放动效，释放其动画资源对象
3. 在退出游戏页面时，我们根据记录的url或者filePath恢复解析到内存中，然后赋值给持有的控件，恢复播放动画。