



# 奇趣百科性能优化分享

Chrome DevTools 中的 Timeline 、 Profiles 等工具使用介绍

ELF + GRUNT



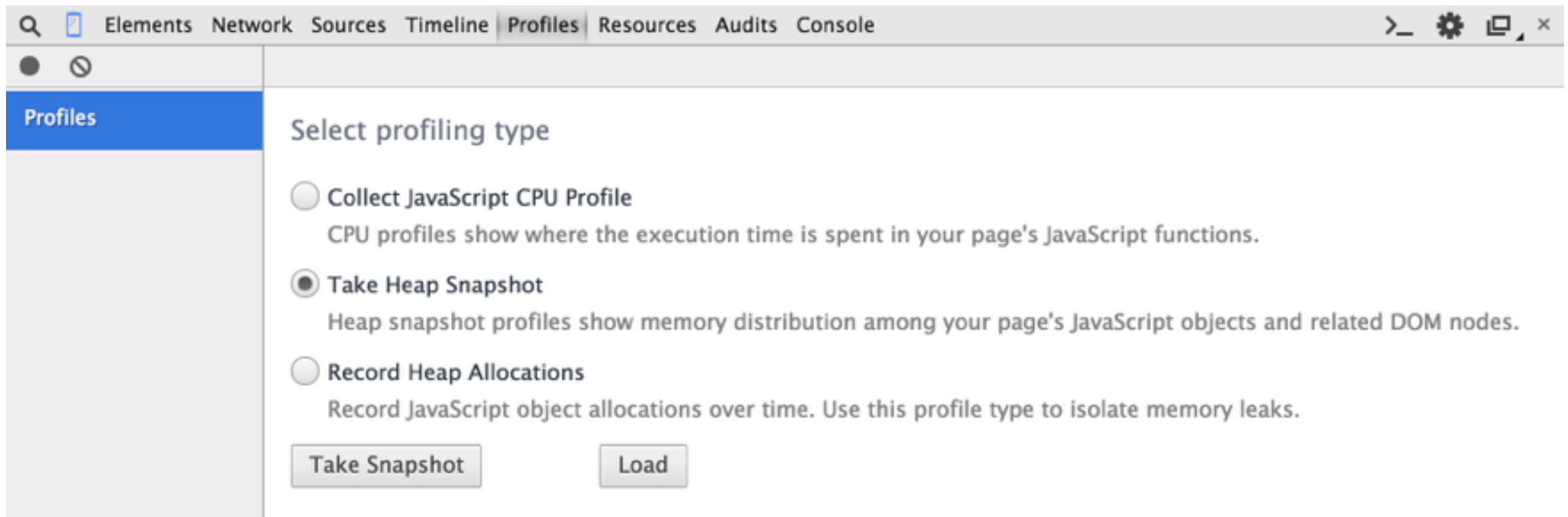
VUE + SCRAT



占用内存过多

# Profiles

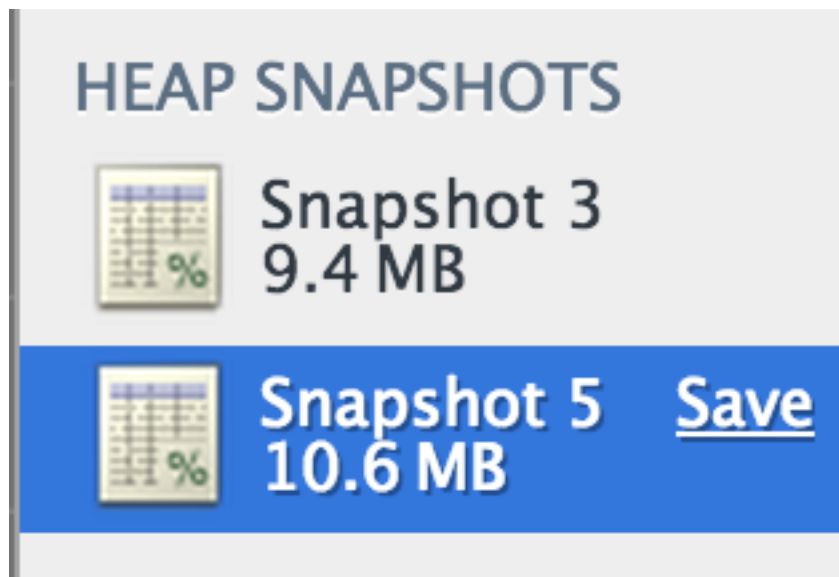
利用 Profiles 工具发现潜在内存问题



★ Collect Javascript Cpu Profile    ★ Take Heap Snapshot    ★ Record Heap Allocations

# Take Heap Snapshot

改版前后内存对比

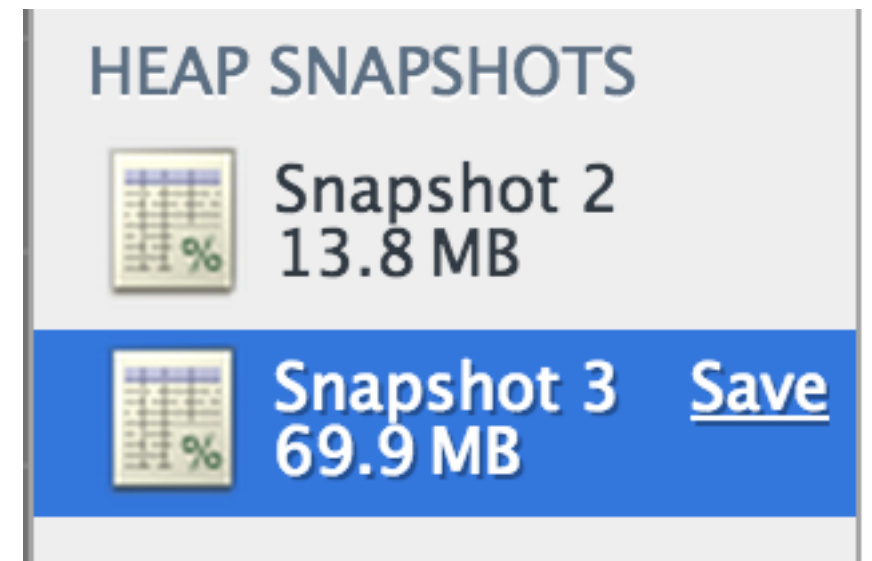


改版前

首屏占用内存9.4MB

加载30次数据后占用内存10.6MB

VS



改版前

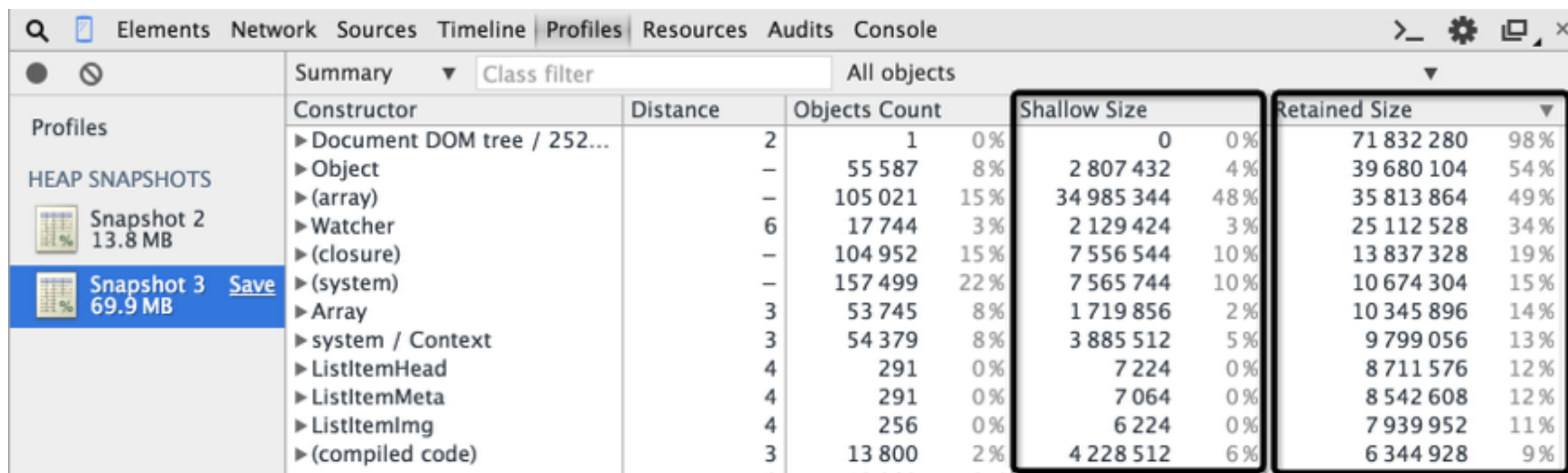
首屏占用内存13.8MB

加载30次数据后占用内存69.9MB

内存泄露! 🤯

# Summary 视图

## 名词解释



The screenshot shows the Chrome DevTools Summary view for a heap snapshot. The left sidebar lists 'Profiles' and 'HEAP SNAPSHOTS' with 'Snapshot 3' (69.9 MB) selected. The main table displays memory usage statistics for various object types. Two columns, 'Shallow Size' and 'Retained Size', are highlighted with black boxes.

Constructor	Distance	Objects Count	Shallow Size	Retained Size
Document DOM tree / 252...	2	1 0%	0 0%	71 832 280 98%
Object	-	55 587 8%	2 807 432 4%	39 680 104 54%
(array)	-	105 021 15%	34 985 344 48%	35 813 864 49%
Watcher	6	17 744 3%	2 129 424 3%	25 112 528 34%
(closure)	-	104 952 15%	7 556 544 10%	13 837 328 19%
(system)	-	157 499 22%	7 565 744 10%	10 674 304 15%
Array	3	53 745 8%	1 719 856 2%	10 345 896 14%
system / Context	3	54 379 8%	3 885 512 5%	9 799 056 13%
ListItemHead	4	291 0%	7 224 0%	8 711 576 12%
ListItemMeta	4	291 0%	7 064 0%	8 542 608 12%
ListItemImage	4	256 0%	6 224 0%	7 939 952 11%
(compiled code)	3	13 800 2%	4 228 512 6%	6 344 928 9%

## Shallow Size

对象直接持有的内存大小,通常情况下应该只有字符串和数组类型可能拥有一个较大的

## Shallow Size

## Retained Size

当前对象所引用的其他对象占用的内存大小. 当前对象被销毁时, 这一部分的内存会被释放.



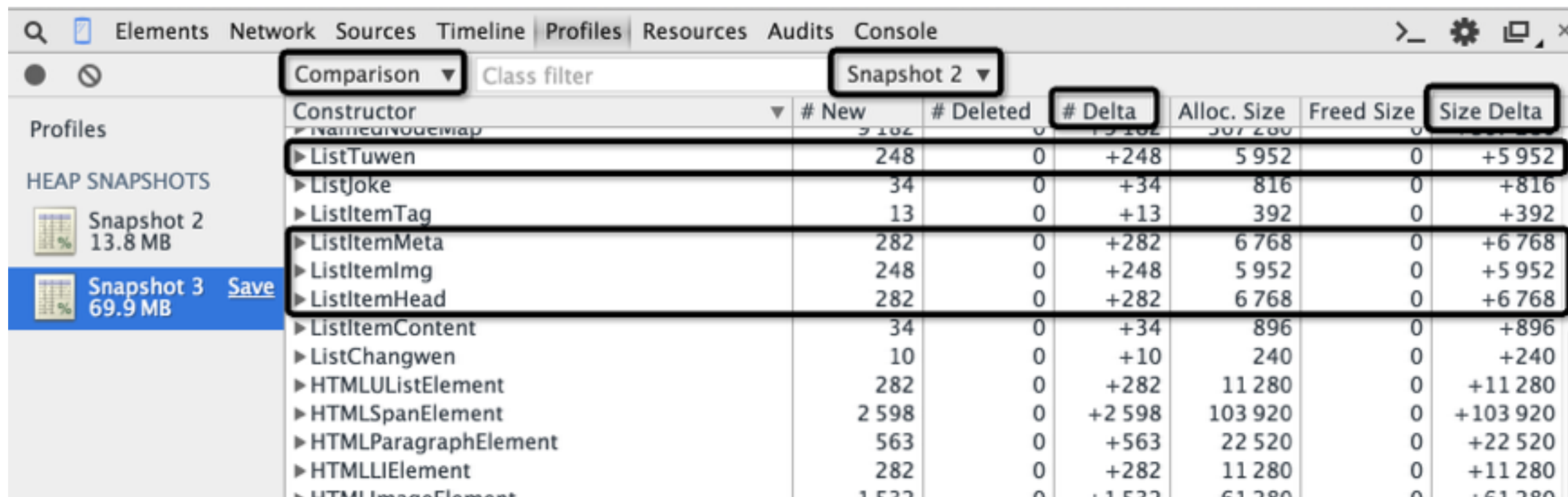
# Summary 视图

找出内存泄露的原因

Elements Network Sources Timeline Profiles Resources Audits Console									
Summary		Class filter			All objects				
Profiles		Constructor	Distance	Objects Count		Shallow Size		Retained Size	
HEAP SNAPSHOTS		► Document DOM tree / 25...	2	1	0%	0	0%	71 832 280	98%
Snapshot 2 13.8 MB		► Object	—	55 587	8%	2 807 432	4%	39 680 104	54%
		► (array)	—	105 021	15%	34 985 344	48%	35 813 864	49%
		► Watcher	6	17 744	3%	2 129 424	3%	25 112 528	34%
		► (closure)	—	104 952	15%	7 556 544	10%	13 837 328	19%
Snapshot 3 69.9 MB		► (system)	—	157 499	22%	7 565 744	10%	10 674 304	15%
		► Array	3	53 745	8%	1 719 856	2%	10 345 896	14%
		► system / Context	3	54 379	8%	3 885 512	5%	9 799 056	13%
		► ListItemHead	4	291	0%	7 224	0%	8 711 576	12%
		► ListItemMeta	4	291	0%	7 064	0%	8 542 608	12%
		► ListItemImg	4	256	0%	6 224	0%	7 939 952	11%
		► (compiled code)	3	13 800	2%	4 228 512	6%	6 344 928	9%
		► Directive	6	13 982	2%	2 125 272	3%	6 120 000	8%
		► Dep	6	13 962	2%	558 480	1%	2 805 864	4%
		► Text	3	15 908	2%	636 320	1%	1 788 216	2%
		► ListTuwen	4	256	0%	6 224	0%	1 715 944	2%
		► VueComponent	4	317	0%	7 752	0%	1 366 824	2%
		► NodeList	4	24 837	4%	993 480	1%	993 480	1%
		► (string)	—	22 691	3%	936 416	1%	936 416	1%

# Comparison 视图

对比首屏和加载30次数据后的内存差异



The screenshot shows the Chrome DevTools Comparison view. The 'Comparison' dropdown is set to 'Snapshot 2'. The 'Class filter' is empty. The table displays memory differences between Snapshot 2 (13.8 MB) and Snapshot 3 (69.9 MB). The columns are: Constructor, # New, # Deleted, # Delta, Alloc. Size, Freed Size, and Size Delta. The table is filtered to show only the 'List' items.

Constructor	# New	# Deleted	# Delta	Alloc. Size	Freed Size	Size Delta
▶ ListTuwen	248	0	+248	5 952	0	+5 952
▶ ListJoke	34	0	+34	816	0	+816
▶ ListItemTag	13	0	+13	392	0	+392
▶ ListItemMeta	282	0	+282	6 768	0	+6 768
▶ ListItemImg	248	0	+248	5 952	0	+5 952
▶ ListItemHead	282	0	+282	6 768	0	+6 768
▶ ListItemContent	34	0	+34	896	0	+896
▶ ListChangwen	10	0	+10	240	0	+240
▶ HTMLUListElement	282	0	+282	11 280	0	+11 280
▶ HTMLSpanElement	2 598	0	+2 598	103 920	0	+103 920
▶ HTMLParagraphElement	563	0	+563	22 520	0	+22 520
▶ HTMLLIElement	282	0	+282	11 280	0	+11 280
▶ HTMLImageElement	1 522	0	+1 522	61 280	0	+61 280

奇趣百科

趣图

段子


发现

小贱日报

4529

分享

listTuwen




忧郁的橙子

listItemHead

姑娘吃的一口好香蕉!

listItemImg



99

34

listItemMeta

分享

百思不得姐



```

▼ ListItemImg
  ▼ ListItemImg @524163
    ▶ $root :: VueComponent @15393
    ▶ $parent :: ListTuwen @524129
    ▶ _data :: @534343
    ▶ properties :: (object properties)[] @53436
    ▶ _directives :: Array @534351
    ▶ _events :: @534355
    ▶ _containerUnlinkFn :: function function un
    ▶ _children :: Array @534361
    ▶ _watcherList :: Array @534363
    ▶ $options :: @534357
    ▶ get gifSrc :: function function() @457555
    ▶ get imgSizeStyle :: function function() @4
    ▶ get isGif :: function function() @457551
    ▶ get isLongImg :: function function() @4575
    ▶ get preloadGif :: function function() @457
    ▶ get src :: function function() @457547
    ▶ _watchers :: @534345
    ▶ $el :: HTMLDivElement @400069
    ▶ get detailUrl :: function function proxyGe
    ▶ get id :: function function proxyGetter()
    ▶ get image :: function function proxyGetter
    ▶ get notPreloadGif :: function function pro
    ▶ get nt :: function function proxyGetter()
    ▶ get title :: function function proxyGetter
    ▶ set detailUrl :: function function proxySe
    ▶ set gifSrc :: function function noop() @15
    ▶ set id :: function function proxySetter()
    ▶ set image :: function function proxySetter
    ▶ set imgSizeStyle :: function function noop
    ▶ set isGif :: function function noop() @153
    ▶ set isLongImg :: function function noop()

```

# For example

ListItemImg 组件:

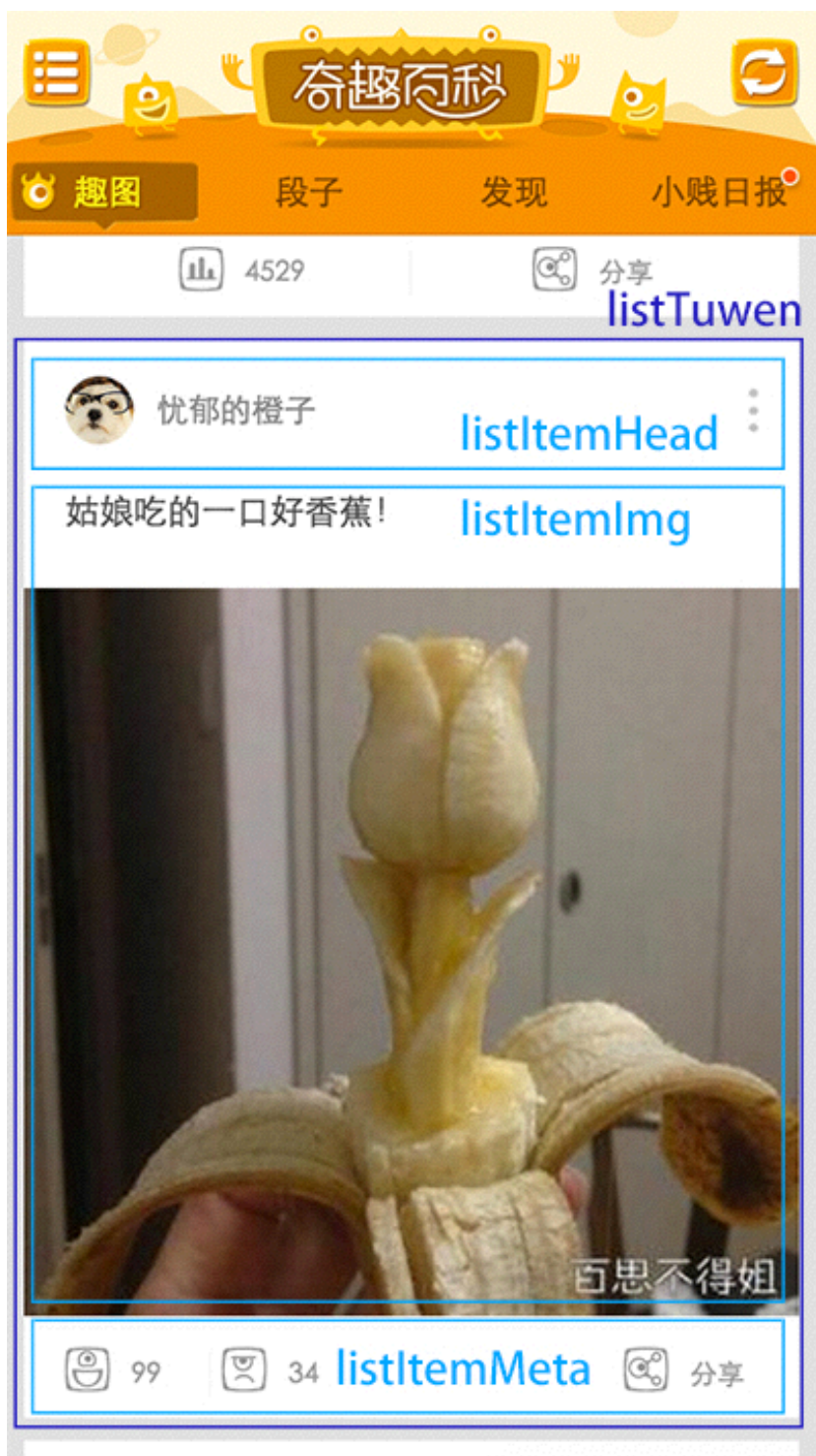
- 大量的 Vue 定义的对象: directives, watchers 等
- 大量的 getter 和 setter 方法等

同一页面中被大量使用的组件尽量不要嵌套组件，  
不然内存占用会随着组件的增多而快速上升。

# 优化方案

卡片内不使用组件, 一个卡片一个组件

卡片的事件托管到列表上



# 优化结果

优化前70MB, 优化后40MB, 此处应有掌声 🙌🙌🙌

Q

Elements

Network

Sources

Timeline

Profiles

Resources

Audits

Console

>

⚙

🖨

✕

●

🚫

Summary

▼


Class filter


All objects

▼

Profiles

HEAP SNAPSHOTS

Snapshot 4  
13.4 MB

Snapshot 5  
42.2 MB

Save

Constructor

Distance

Objects Count

Shallow Size

Retained Size

▼

▶ Watcher

6

7 2322 %

867 9842 %

8 531 90419 %

▶ system / Context

3

35 3367 %

2 412 4885 %

8 176 93618 %

▶ Array

3

33 7937 %

1 081 3922 %

6 750 76815 %

▶ (compiled code)

3

14 6743 %

4 377 57610 %

6 648 09615 %

▶ (system)

—

95 00020 %

4 281 25610 %

6 641 44015 %

▶ ListTuwen

4

2610 %

6 4240 %

6 044 04014 %

▶ Directive

6

7 2752 %

1 164 0003 %

5 070 62411 %

▶ Text

3

18 7014 %

748 0402 %

1 904 7924 %

▶ Document DOM tree / 23780 entries

2

10 %

00 %

1 753 7204 %



300张卡片占用内存40M左右, 下降了39.6%

Keep going



越往下加载, DOM 越多, 想不卡都难! 🥲

Inspiration — 手机淘宝

# 优化方案

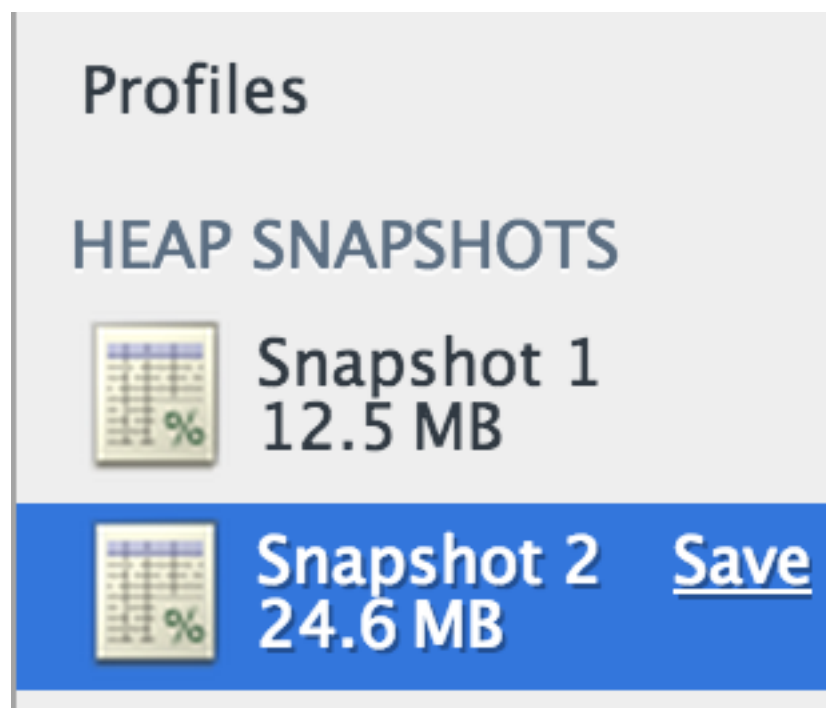
移除视窗外不可见的DOM.

也就是说列表最多只有30张卡片, DOM不会随着页面越滚动到下面越多.

```
▼ <div class="list">
  ▼ <div data-group="0" data-show="0" style="min-height: 6448px;">
    <!--v-repeat-->
  </div>
  ▼ <div data-group="1" data-show="0" style="min-height: 5679px;">
    <!--v-repeat-->
  </div>
  ▼ <div data-group="2" data-show="0" style="min-height: 5801px;">
    <!--v-repeat-->
  </div>
  ▶ <div data-group="3" data-show="0" style="min-height: 5168px;">...</div>
  ▶ <div data-group="4" data-show="0" style="min-height: 5596px;">...</div>
  ▶ <div data-group="5" data-show="1" style="min-height: 0px;">...</div>
  ▶ <div data-group="6" data-show="1" style="min-height: 0px;">...</div>
  ▶ <div data-group="7" data-show="1" style="min-height: 0px;">...</div>
  <!--v-repeat-->
</div>
<!--v-component-->
</div>
```

# 优化结果

优化前40MB, 优化后25MB

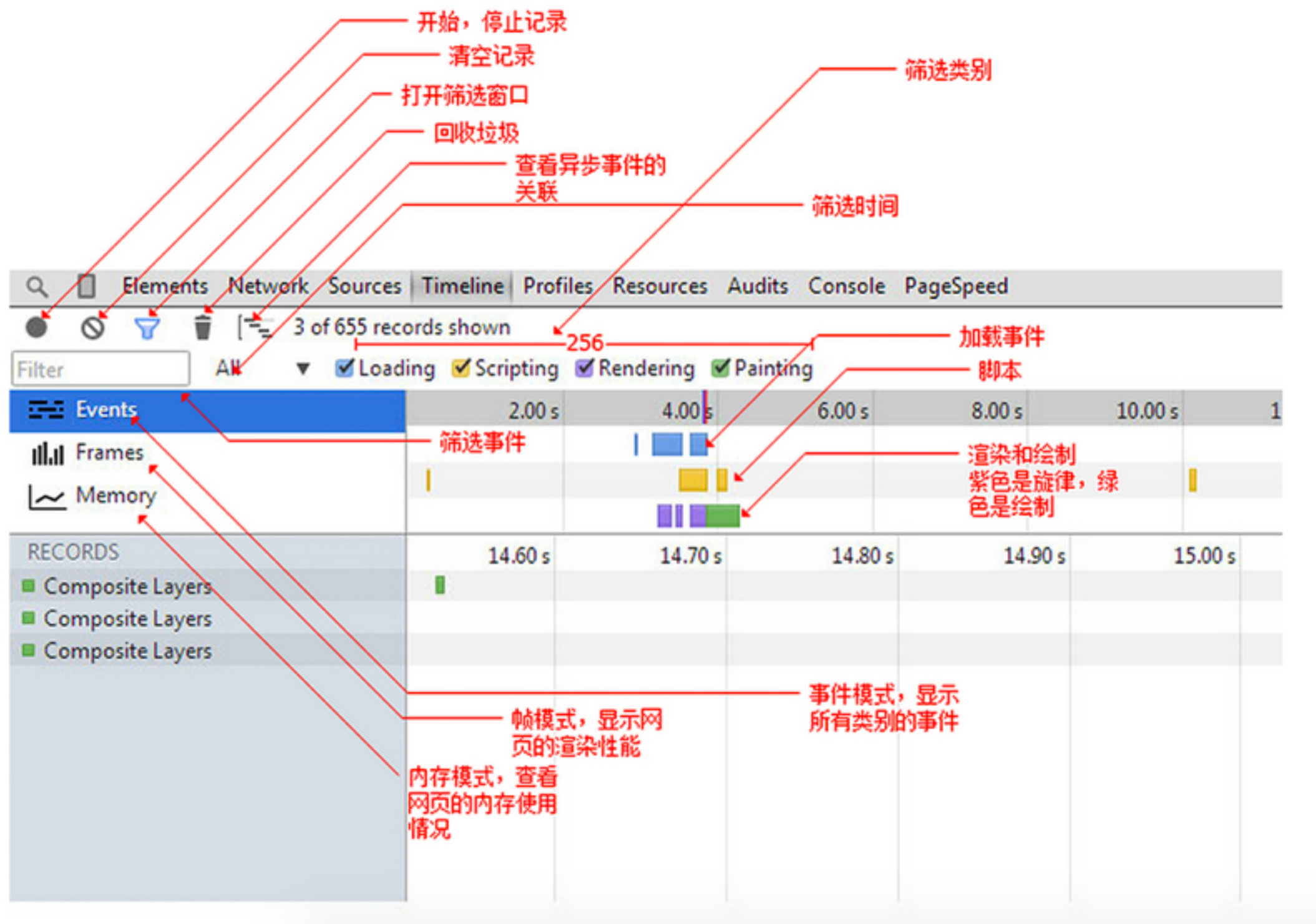


30张卡片占用内存25M左右, 下降了21.4%

换个角度

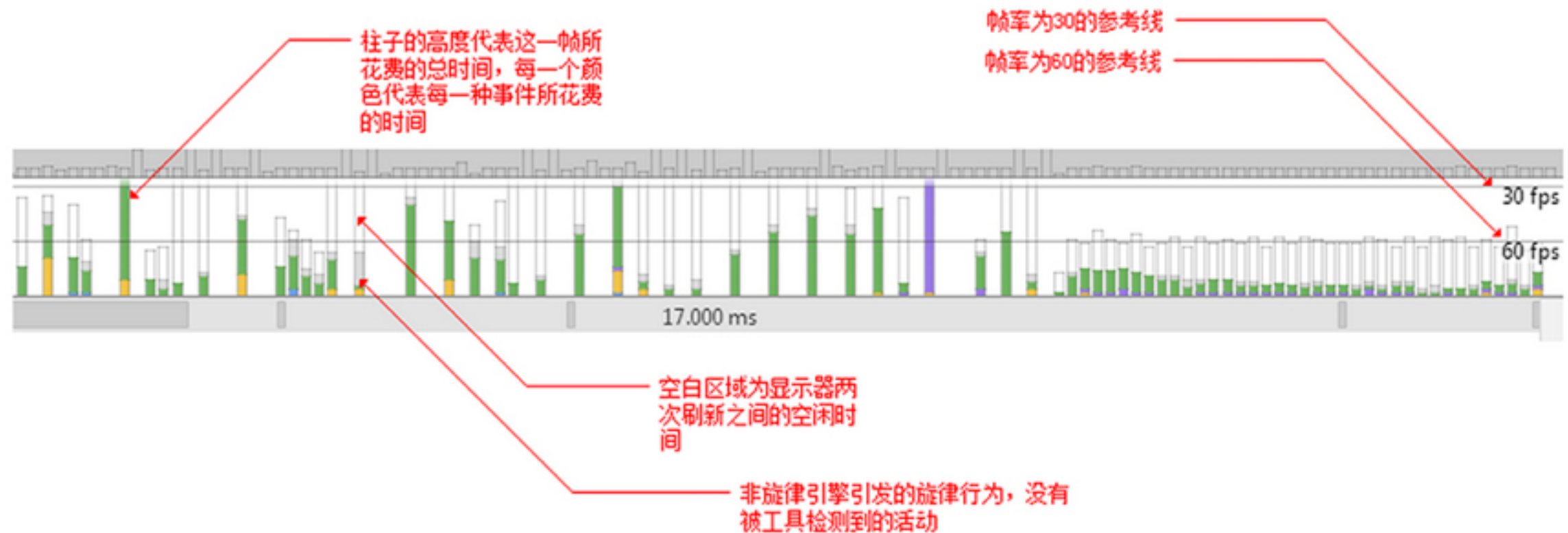
# Timeline

## Timeline 控制板介绍



# Timeline – Frames

## Frames 模式介绍



一帧(一条柱子)代表了在一帧(GPU渲染画面)内展现内容所要完成的工作,包括执行JS,处理事件,更新 DOM, 改变样式和布局还有绘制页面.

60FPS

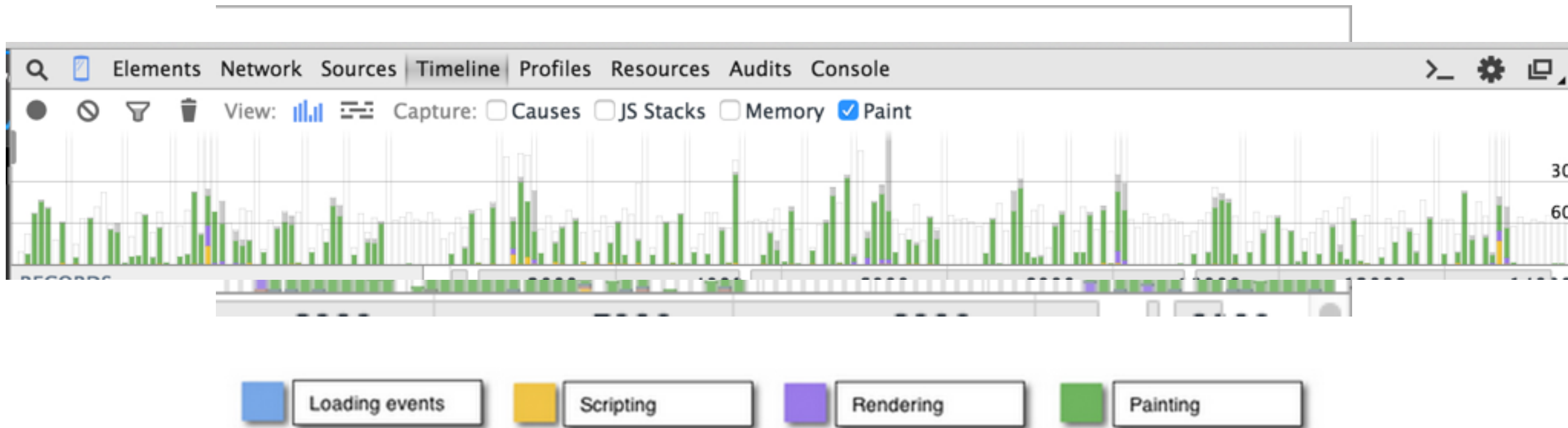


**显示器刷新频率: 60Hz**

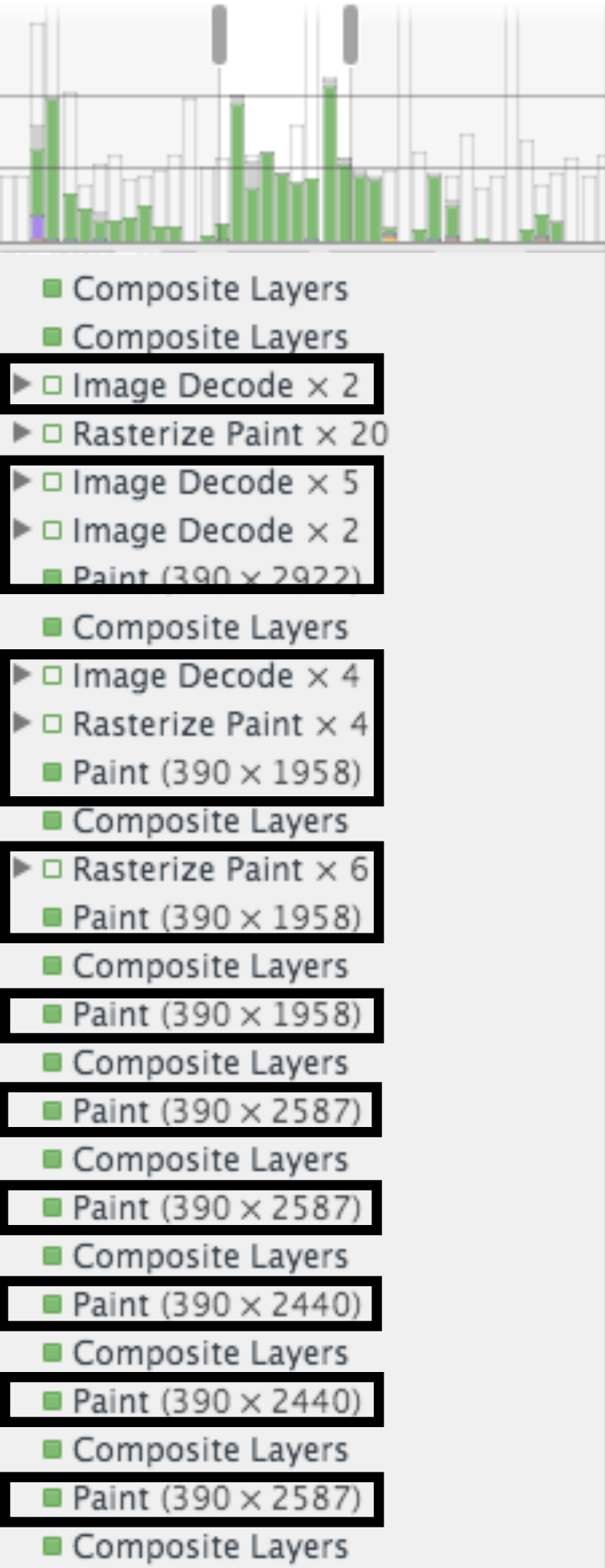
**GPU渲染页面频率: 60FPS**

# Frames

往下滚动加载两次数据的 Frames 情况



- 蓝色: 网络和HTML解析
- 黄色: JavaScript 脚本运行
- 紫色: 样式重计算和布局 ( Layout , Recalculate Style, Update Layer tree)
- 绿色: 绘制和合成 ( Paint , Composite Layers)



超过60FPS的柱子一般包括多个 Image Decode 事件

Image Decode 事件在一个图片资源完成解码后触发

大量的图片加载完成后进而大量触发 Paint 事件

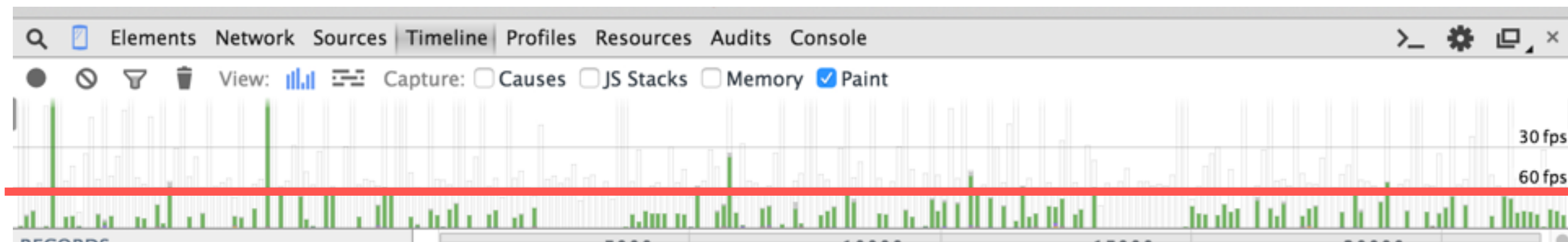
Paint事件是合并后的层被绘制到对应显示区域后触发

图片分开加载, 把Paint事件分散, 高(高于60FPS)  
的柱子拆成多个矮(低于60FPS)的柱子, 这样就能  
进一步提升流畅度.

# 图片懒加载

# 优化结果

实现了图片懒加载的 Frames 情况



绝大部分 Frame 达到60FPS, 实际操作流畅

但是



图片懒加载会让用户在整体上觉得卡顿,

从用户体验的角度出发,

我们的优化方案最终并没有使用图片懒加载.



# 其他优化

细小却同样重要

- 避免使用 border-radius, box-shadow, 渐变等性能杀手
- 动画尽量使用 opacity 和 transform 完成

合并 Vue 组件

内存占用减少40%



性能优化

移除视图外的 DOM

内存占用进一步减少21%

图片懒加载

优化 Frames 情况, 实际操作流畅

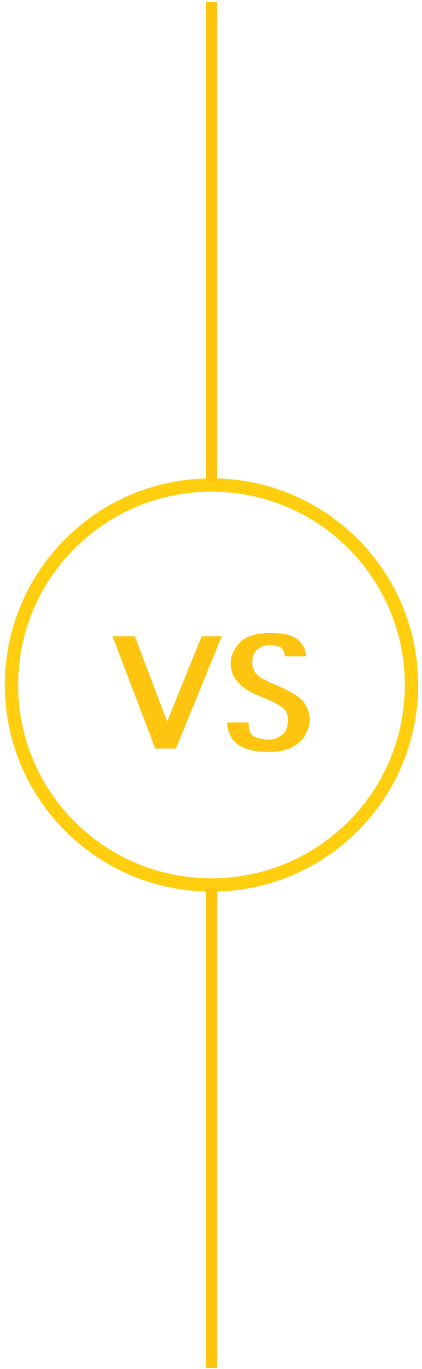
# 优化前后数据对比

## 优化前

人均停留时长: 14.5分钟

人均加载次数: 9.3

次日留存率: 41.45%



## 优化后

人均停留时长: 17.0分钟

人均加载次数: 11.0

次日留存率: 44.11%

最后的广告👁👁

<http://xinranliu.me>

Q & A

THX 🙏