目录

10 图片加载优化（下）

更新时间：2020-06-05 18:50:33

> 学习这件事不在乎有没有人教你，最重要的是在于你自己有没有觉悟和恒心。
>
> —— 法布尔

上一节我们主要从图片的概念和特点方面介绍了相关的优化方法，这一节我们从代码层面介绍相关的图片优化方法，我们主要介绍2个，一个是图片渐进显示，另外一个是图片懒加载。

图片渐进显示

引入

可能有的人对图片渐进显示这个效果不是很清楚，我们先来看如下录屏：

如上我们可以看到左边的是没有任何效果的普通网站，而右边则是使用了图片渐进显示效果的网站。图片渐进显示效果其实就是在图片完全加载完之前，使用低分辨率的模糊图片做预览图，让用户提前先看到模糊的轮廓，同时加载真正的高清图，高清图加载完成之后，然后进行替换。

这么做虽然CDN流量有所上升，因为有了加载了额外的图片，但是带来的用户体验非常好，国内这个技术用的比较多的网站是知乎，大家可以去网站上面亲自体验一下加深印象。

- **非代码方式**

上一节我们介绍了JPEG格式的图片，知道了它的一些应用场景，其实JPEG还可以细分为Baseline JPEG（标准型）、Progressive JPEG（渐进式）。两种格式有相同尺寸以及图像数据，他们的扩展名也是相同的，唯一的区别是二者显示的方式不同。

其中Baseline JPEG格式的显示方式如下所示，可以看到效果与我们上面录屏当中普通网页的显示效果相同。

- Progressive JPEG格式的显示方式如下，可以看到它和使用了渐进显示的网页显示效果是类似的，也就是我们也可以直接使用这种格式的图片来达到渐进式显示效果的目的。

> Tips：关于Progressive JPEG格式图片的获取，我们可以直接使用Photoshop，然后在保存为JPEG格式的时候，将连续这个选项勾选即可，这样我们得到的就是Progressive JPEG格式的图片了。

- **代码方式**

下面我们就来介绍如何用代码的方式实现录屏当中的渐进式显示效果，代码实现方式也分为两种，一种是使用canvas API来实现的，另外一种只需要JavaScript即可实现，我们主要介绍主流的JavaScript实现方式，canvas API的实现方式这里也会介绍大致原理，感兴趣的同学可以下去自行研究。

普通JavaScript实现

首先我们构造基本的HTML，结构如下:

```
<div class="progressive">
  //src是img本身属性，data-src是自定义属性，相应事件触发后用data-src的属性值
  <img class="preview lazy" data-src="origin.png" src="preview.png" />
</div>
```

目录

这里origin.png是原图，preview.png是等比例压缩后的模糊预览图。然后我们来写基本的样式，如下：

```
.progressive {
  position: relative;
  display: block;
  overflow: hidden;  // 形成BFC不影响其他部分
}
.progressive img {
  display: block;
  width: 100%;
  max-width: 100%;
  height: auto;
  border: 0 none;
}
.progressive img.preview {
    filter: blur(2vw);
    transform: scale(1.05);
 }
.progressive img.origin {
  position: absolute;
  left: 0;
  top: 0;
  animation: origin 1s ease-out;
}

@keyframes origin {
  0% {
    transform: scale(1.1);
    opacity: 0;
  }
  100% {
    transform: scale(1);
    opacity: 1;
  }
}
```

这里利用CSS filter:blur(20vw)来做玻璃模糊。由于低质量缩略图代替高质量大图会拉伸放大，看起来像二维码一样的，为了友好显示，我这里做个剥离模糊。然后在最下面的我们定义了origin动画效果，让其在加载过程中有一个渐变效果。最后我们再来介绍JavaScript代码，如下：

```
function checkImage() {
  //获取所有img
  const lazys = document.querySelectorAll('img.lazy')
  const l = lazys.length
  //如果存在进行遍历
  if(l>0){
    for (let i = 0; i < l; i++) {
      //获取rect对象
      let rect = lazys[i].getBoundingClientRect()
      //当图片出现在可视区域内加载图片
      if (rect.top < window.innerHeight && rect.bottom > 0 && rect.left < wi
        //loadImage定义了上面提到的用data-src的属性值代替src属性值
        loadImage(lazys[i])
      }
```

```
  }
}
```

> Tips: `Element.getBoundingClientRect` 方法返回一个rect对象，提供当前元素节点的大小、位置等信息，基本上就是 CSS 盒状模型的所有信息，如下：
>
> - `x`： 元素左上角相对于视口的横坐标
> - `y`： 元素左上角相对于视口的纵坐标
> - `height`： 元素高度
> - `width`： 元素宽度
> - `left`： 元素左上角相对于视口的横坐标，与 `x` 属性相等
> - `right`： 元素右边界相对于视口的横坐标（等于 `x + width`）
> - `top`： 元素顶部相对于视口的纵坐标，与 `y` 属性相等
> - `bottom`： 元素底部相对于视口的纵坐标（等于 `y + height`）

完整代码我放到了GitHub上面，大家可以下载下来进行阅读，这里篇幅有限，就不把所有代码全部罗列出来了。

使用Canvas API的方式

使用Canvas API流程基本原理和纯JavaScript类似，只是绘制的时候用Canvas API代替，大致流程描述如下：

- 在文档流中嵌入一个和原图一样大的 canvas，和一个不显示的 img ，后者负责载入一个非常小的图片
- 当JavaScript监听到上述 img 标签的 onload 事件时，将小的图片以高斯模糊的方式绘制到 canvas 上
- 当页面滚动到快要显示这张图片时，JavaScript 控制开始加载原图
- 原图加载完成后，被放在之前的 img 标签中
- 在 canvas 中应用 fadeOut 效果，逐渐露出原图，fadeOut 播放完成后，将 canvas display：none

具体的实现方式大家可以参考国外的 Medium 网站。

> Tips:阿里云的对象存储OSS也支持，图片渐进显示处理，实际开发当中我们常用这种方式进行处理。

图片懒加载

图片懒加载是现在最常用的性能优化手段之一，对于首屏用不到的图片，我们完全可以使用懒加载在用户下拉到对应位置的时候再进行加载，避免网页打开时一下子加载过多资源。

```
<div class="container">
    <div class="img-area">
    <img class="my-pic" alt="loading" data-src="./img/img1.png">
    </div>
  <div class="img-area">
      <img class="my-pic" alt="loading" data-src="./img/img2.png">
    </div>
    <div class="img-area">
      <img class="my-pic" alt="loading" data-src="./img/img3.png">
    </div>
    <div class="img-area">
      <img class="my-pic" alt="loading" data-src="./img/img4.png">
  </div>
    <div class="img-area">
    <img class="my-pic" alt="loading" data-src="./img/img5.png">
    </div>
    ......
  </div>
```

可以看到这个结构与我们上面渐进加载图片的HTML结构类似，其实这两者原理都是差不多，大家也可以类比进行学习。然后我们接着来看JavaScript，如下：

```
//图片什么时候出现在可视区域内
function isInSight(el) {
const rect = el.getBoundingClientRect();
  //这里我们只考虑向下滚动
const clientHeight = window.innerHeight;
   // 这里加50为了让其提前加载
  return rect.top <= clientHeight + 50;
}
//data-src的属性值代替src属性值
function loadImg(el) {
  if (!el.src) {
    const source = el.dataset.src;
    el.src = source;
  }
}
let index = 0;
function checkImgs() {
  const imgs = document.querySelectorAll('.my-pic');
  for (let i = index; i < imgs.length; i++) {
    if (isInSight(imgs[i])) {
    loadImg(imgs[i]);
      index = i;
  }
  }
}
```

通过分析上述代码，可以看到我们主要用到的还是 getBoundingClientRect 这个关键API，通过它来判断图片什么时候出现在可视化区域。

> 思考题：HTML5 有一个新的 `IntersectionObserver` API，它可以自动观察元素是否可见，那么大家可以查阅它的相关用法，试着用它来实现图片懒加载

同样这部分的完整代码我也放到了GitHub，有需要的直接下载即可。

这一节我们从代码层面介绍了两种图片优化方法，这两种方法都是非常常用的图片优化方法，大家可以根据自己的业务需求自行选择，可能你在开发当中也有常用的一些图片优化方法，可以写在评论区，大家一起进行交流。

← 9 图片加载优化（上）

11 十八般缓存 →

精选留言 2

欢迎在这里发表留言，作者筛选后可公开显示

空白处

学习了，感谢大佬的指点，期待后续的更新

0 回复 2019-08-12

慕粉1820144070

更新太慢啦！

1 回复 2019-08-11

千学不如一看，千看不如一练