

面试官：请你手写一下！ 懒加载

懒加载

今天，我们来学习一个面试经常考到的知识点：懒加载！

也是我们优化页面的一种手段，大家不妨学习一下！


懒加载的概念

懒加载（Lazy Loading）是一种延迟加载技术，指的是在需要使用某个对象或数据时才进行加载，而不是在系统启动或加载时就立即加载。懒加载可以在一定程度上提高系统的性能和资源利用率。

例如：图片的加载，我们看不见的图片就不加载，能看到的图片就加载，加载完的图片就存入到浏览器的缓存当中，这样一个小小的功能就是懒加载！


我们进入到PC端的淘宝网官网：

猜你喜欢 个性推荐




华科加厚搪瓷盆 洗脸盆热水盆36cm40cm
送运费险 淘金币抵0.9元

¥ 32.55




宇治抹茶 调味含糖抹茶星冰乐抹茶拿铁甜点冰淇淋
送运费险

¥ 39




无印良品 MUJI 聚丙烯储物箱抽屜式 收纳箱

¥ 145




包邮桂冠亲肠 桂冠香肠 台湾风味香肠金装豆捞火
包邮 多省1包包邮

¥ 67




日本可乐CLOVER服装立裁针 缝纫用大头针手工固定

¥ 25.01




Monet莫奈原木杆猪鬃毛扇形笔专业水粉画笔 硬毛

¥ 32




新乐福咸味海苔饼干批整箱发咸味薄饼薄脆早餐饼
淘金币抵0.5元

¥ 19.6



美国进口Ball Mason Jar玻璃密封罐梅森瓶蔬菜沙拉
送运费险 淘金币抵0.3元

¥ 12



灵活舒适 薄尼龙PU印花涂掌涂指浸胶透气防滑耐磨摘花椒胶
淘金币抵0.3元

¥ 13

LYRA 艺雅官方旗舰店 德国LYRA艺雅天琴旗舰店 云南特产猫哆哩酸角糕好 尺寸定制 定制布艺收纳箱儿童家用

我们可以发现，当我们在往下滚动的时候，下方商品的图片在你看不到之前是没有加载的，而是当你滚到它的时候，这个图片才加载，这就是懒加载的一个应用！

实现懒加载

懒加载思路

首先，我们说一下思路：

- 要实现懒加载，我们可以先让图片全部**不显示**
- **如何让图片加载**：只要img的src属性里面有图片，就会加载，所以我们可以给img的src属性设置一个空 的图片，这样图片就不会加载了。
- 然后拿到用户的屏幕尺寸有多高，图片有多高，以及判断用户的屏幕能够放多少张图片！
- 先让能放的下图片有值，放不下的图片不放值（本质上就是判断图片是否在可视区域，然后把图片的值（图片地址）放到src上面）

接下来，我们就可以开始尝试手写懒加载了！

开始手写懒加载

首先，我们放十个 `img` 标签，并且自行搜索十张图片的地址！并且设置一个类名 `img-item`

```
1 html
2 复制代码
3 <body    
```

我们将 `src` 属性先置为空！然后我们自己定义一个 `data-original` 属性存我们图片的地址！

第二步：拿到用户可视界面的高度，图片相关的几何信息

接下来，我们开始第二步，拿到用户可视界面的高度，图片相关的几何信息，我们可以这样写一段js代码

```
1 html
2 复制代码
3 <script    // 拿到可视区域的高度    innerHeight是window上独有的方法 别的用outerHeight
或者innerHeight    let viewHeight = window.innerHeight;    // 拿到所有的img元素
```

```
img[data-original]只要具有data-original属性的img元素    let imgs =
document.querySelectorAll('img[data-original]');    imgs.forEach(el=>{
// getBoundingClientRect() 专门获取容器的几何信息    let rect =
el.getBoundingClientRect()    })/</script>
```

我们可以通过 `window.innerHeight`; 拿到用户可视区域的高度

注意！！ `innerHeight`是window上独有的方法 别的用`offsetHeight`或者`innerHeight`

再通过 `let imgs = document.querySelectorAll('img[data-original]');` 拿到所有具有 `data-original` 属性的 `img` 元素

`getBoundingClientRect()`:专门获取容器的几何信息！

我们看看浏览器的输出,方便理解:

```
> document.querySelector('img')
<
  <img class="img-item" src data-original="https://img.zcool.cn/community/011e0e5c231dcaa8
  0121df9063b03f.jpg@1280w_1l_2o_100sh.jpg" alt>
> document.querySelector('img').getBoundingClientRect()
< ▼ DOMRect {x: 8, y: 50, width: 636.7999877929688, height: 300, top: 50, ...} ⓘ
  bottom: 350
  height: 300
  left: 8
  right: 644.7999877929688
  top: 50
  width: 636.7999877929688
  x: 8
  y: 50
  ► [[Prototype]]: DOMRect
>
```

| bottom是指图片底部到可视界面顶部的距离 > top是指图片顶部到可视界面顶部的距离

我们这里没有用 `querySelectorAll` 而是 `querySelector` 则拿到的是第一个 `img` 元素！

第三步：判断图片是否在可视区域，然后把图片的值（图片地址）放到src上面

最后就是判断图片是否在可视区域，然后把图片的值（图片地址）放到src上面，我们再第二步的基础上可以封装一个方法！

```
1 html
2 复制代码
3 <script    // 拿到可视觉区域的高度    let viewHeight = window.innerHeight;
function lazyLoad(){    // 拿到所有的img元素 img[data-original]只要具有data-
original属性的img元素    let imgs = document.querySelectorAll('img[data-
original]');    imgs.forEach(el=>{    // getBoundingClientRect()专门
获取容器的几何信息    let rect = el.getBoundingClientRect()
}
```

```
if(rect.top<viewHeight){           //进行赋值操作           }  
    }    })    }</script>
```

我们可以想象一下，只要到图片顶部的距离小于可视区域的高度，我们就应该加载图片了！

所以这里用的是 `if(rect.top<viewHeight)`

如果是正常情况下，`data-original` 的值应该用一个中间变量来承接，而这个中间变量应该是一个 `image` 标签，一般情况下我们还需要使用 `createElement` 创建一个标签，但是 `image` 标签比较特殊，它天生具有构造函数！可以创建一个图片对象，所以我们接下来可以这样写：

```
1 js  
2 复制代码  
3 if(rect.top<viewHeight){           //进行赋值操作           let image = new Image()  
    image.src = el.dataset.original;}
```

值得注意的是： `data-original` 在js中有专有写法，写成 `dataset.original`

接下来就是赋值了！

你可能会想直接这样写：

```
1 js  
2 复制代码  
3 el.src = image.src
```

但是我们会这样写，而是写成

```
1 js  
2 复制代码  
3 if(rect.top<viewHeight){           let image = new Image()           image.src =  
    el.dataset.original;           image.onload = function(){           el.src = image.src  
    } }
```

为什么这样写呢？这样写我们就通过 `onload` 函数的执行，当我们创造的 `image` 标签加载完毕的时候，我们才去把 `el.src` 赋值为 `image.src`。

如果我们直接写 `el.src = image.src`，图片加载的时候还是会空白一下，因为没有提前加载好，而加载是需要时间的，用 `onload` 封装的话，图片可以提前加载完毕，存入浏览器的缓存当中，能够更好的节约性能，节省时间！

当图片加载完毕了之后，我们再把 `data-original` 属性移除


```

1 js
2 复制代码
3 if(rect.top<viewHeight){    let image = new Image()    image.src =
  el.dataset.original;    image.onload = function(){    el.src = image.src
  }    // 图片加载完毕就移除属性    el.removeAttribute('data-original')}

```

写到这里，功能我们就实现了，我们调用一遍这个函数，同时添加一个事件监听器监听滚动事件，随着页面的滚动来加载图片，最终我们 `js` 部分就写成了这样！

```

1 html
2 复制代码
3 <script    // 拿到可视区域的高度    let viewHeight = window.innerHeight;
  function lazyLoad(){    // 拿到所有的img元素    let imgs =
  document.querySelectorAll('img[data-original]');    imgs.forEach(el=>{
    // getBoundingClientRect() 专门获取容器的几何信息    let rect =
  el.getBoundingClientRect()    if(rect.top<viewHeight){
  // img元素自带一个构造函数，可以创建一个图片对象    let image = new
  Image()    // js专有写法dataset.original; = data-original
    image.src = el.dataset.original;    image.onload = function()
  {    el.src = image.src    }    // 图片
  加载完毕就移除属性    el.removeAttribute('data-original')
  }    })    }    lazyLoad()    // 添加滚动事件监听器
  document.addEventListener('scroll', lazyLoad)</script>

```

我们来看看现在页面的效果！

