17 图片查看器组件

更新时间：2019-08-28 15:36:15

> 完成工作的方法，是爱惜每一分钟。
>
> ——达尔文

朋友圈首页 UI - 图片查看器组件

图片查看器是一个移动 web 应用最常见的组件之一，当然有很多开源的组件可以供我们直接使用，但是对于我们这次项目来说，我将会带领大家重新开发一个功能完整的图片查看器组件，便于大家理解移动 web 端的一些基本事件和动画原理，并采用原始的 JavaScript 来实现（这样我们这个组件就不限于在 vue 里使用，在 react.js 或者是 zepto.js 项目中也可以使用），在学习本章内容之前，建议大家先预习一下移动 web 中 touch 基本事件的知识，可以参考这个教程，下面就进入开发吧。

本章节完整源代码地址，大家可以事先浏览一下：

Github

图片查看器 slider.js:

首先我们需要明确一下在移动端，一个图片查看器的基本功能有：

1. 点击查看大图。
2. 前后拖动查看上一张和下一张。
3. 双击或双指放大查看。

基于上述功能，我们开发一个可以跨框架的，利用原生 javascript 实现的图片查看器组件 slider.js，使用效果如下图：

在前端项目中的的 public 文件夹下，在 lib 目录下，新建一个 slider 文件夹，同时新建 slider.js，图片查看器的代码，都写在这里。

初始化：

我们采取面相对象的方式来封装 Slider 对象，这里设置了一些初始值，代码如下：

```
// 构造函数
function Slider (opts) {
  // 构造函数需要的参数
  var imgWrap = document.createElement('DIV')
  imgWrap.style.cssText = '-webkit-transform:translate3d(0,0,3px);-webkit-transition:opacity 200ms;opacity:0;position:fixed;top:0;left:0;right:0;bottom:0;background-color: #000;z-index:999;'
  this.wrap = imgWrap
  this.list = opts.list
  // 初始在哪一页
  this.idx = opts.page || 0
  // 初始化方法
  this.init()
  // 渲染dom
  this.renderDOM()
  // 绑定事件
  this.bindDOM()
}
```

```
// 第一步 -- 初始化
Slider.prototype.init = function () {
  // 设定窗口比率
  this.radio = window.innerHeight / window.innerWidth
  // 设定一页的宽度 +10 代表每张图片流一定的间距
  this.scaleW = window.innerWidth + 10
  // 放大时的最大倍数
  this.scaleMax = 2;
}
```

查看器一般是全屏的，我们利用 `window.innerWidth` 来获取屏幕宽度 `window.innerHeight` 获取屏幕高度。`window.innerHeight / window.innerWidth` 表示窗口比率，在后面渲染图片时会用到。

渲染 dom:

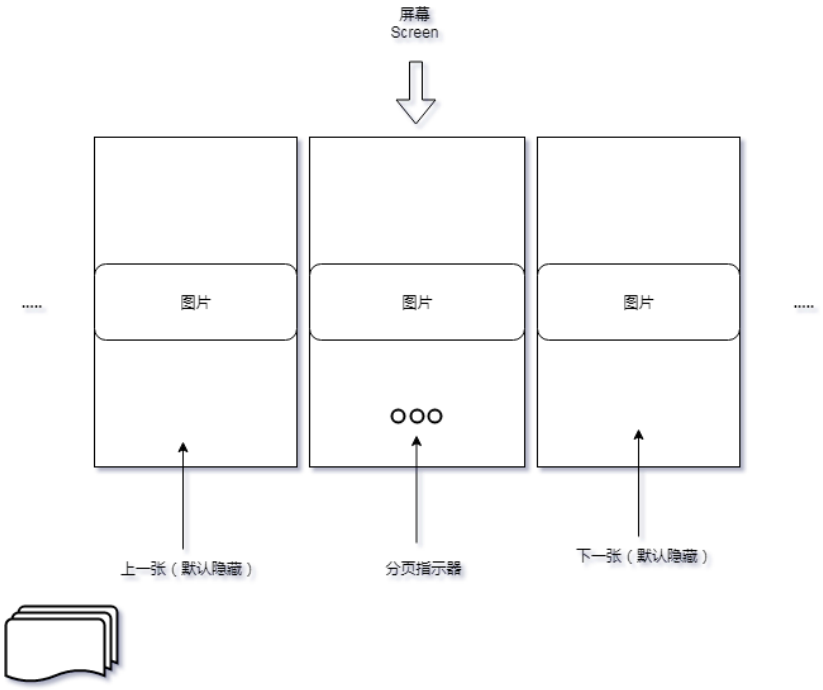
在进行一些初始化操作后，我们需要将 dom 元素渲染出来，根据外部传来的图片数据，代码如下：

```
Slider.prototype.renderDOM = function () {
  var wrap = this.wrap
  //图片的数据
  var data = this.list
  var len = data.length

  this.outer = document.createElement('ul')
  this.outer.style.cssText = 'height:100%;overflow:hidden;'
  // 根据元素的
  for (var i = 0; i < len; i++) {
    var li = document.createElement('li')
    li.style.cssText = 'position:absolute;display:flex;align-items:center;overflow:hidden;height:100%;'
    var item = data[i]
    li.style.width = window.innerWidth + 'px'
    li.style.webkitTransform = 'translate3d(' + (i - this.idx) * this.scaleW + 'px, 0, 0)'
    if (item) {
      // 根据窗口的比例与图片的比例来确定
      // 图片是根据宽度来等比缩放还是根据高度来等比缩放
      if (item['height'] / item['width'] > this.radio) {
        li.innerHTML = '<img style="max-width:100%;max-height:100%;height:'+window.innerHeight+'px;margin: 0 auto;"  class="lazyload" src="" data-original="' + item['img'] + '">'
      } else {
        li.innerHTML = '<img style="max-width:100%;max-height:100%;width:'+window.innerWidth+'px;margin: 0 auto;"  class="lazyload" src="" data-original="' + item['img'] + '">'
      }
    }
    this.outer.appendChild(li)
  }

  // UL的宽度和画布宽度一致
  this.outer.style.cssText = 'width:' + this.scaleW + 'px;height:100%;overflow:hidden;'

  wrap.style.height = window.innerHeight + 'px'
  wrap.appendChild(this.outer)

  this.divider = document.createElement('ul')
  this.divider.style.cssText = 'position: absolute;bottom: 24px;left: 50%;font-size:19px;-webkit-transform: translateX(-50%);color: rgb(109, 109, 109);'

  //渲染分页的UI和样式
  for (var k = 0; k < len; k++) {
    var dividerItem = document.createElement('li')
    dividerItem.innerText = '•'
    dividerItem.style.cssText = 'float:left;margin-right:5px;'
    if (k === this.idx) {
      dividerItem.style.color = '#fff'
    }

    this.divider.appendChild(dividerItem)
  }

  //当传入的图片列表大于等于2，才显示分页组件
  if (len >= 2) {
    wrap.appendChild(this.divider)
  }
}
```

上面的 dom 渲染，主要完成了下面的逻辑：

1. 创建一个 ul 元素，其中内部的 li 用来承载每个图片。
2. 通过外部传来的数据，将 url 赋值给 img 元素的 src，并根据之前的窗口比例来设置图片的宽高。
3. 将图片利用 CSS3 的 transform:translate3d 定位当前显示的图片。
4. 最后渲染分页器的 UI 样式。

具体代码的含义，在注释中可以查看，最终完成的 UI 逻辑如下图：



绑定 dom 事件：

在 dom 完成渲染之后，下面就需要绑定一些事件了，事件是图片查看器组件交互的基础，代码逻辑首先给大家总结一下，如下：

1. 通过监听 touchstart 事件，我们记录一些初始值，用来后续判断是否进入下一页，移动的基准值等等。
2. 通过监听 touchmove 事件，我们在手指移动时，动态的修改 img (li 元素) 的 transform 的 translate3d 的 x 值，来达到左右移动。
3. 当手指离开时，触发 touchend 事件，我们判断移动的距离来识别是进入下一张还是返回当前这张还是上一张。
4. 当手指离开时，触发 touchend 事件，我们给 img (li 元素) 添加一个 transition 过渡动画，让 img (li 元素) 缓慢的移动到指定位置。

然后我们在每个事件来细分讲解一下：

首先需要绑定 touchstart 事件，在手指刚接触到屏幕时需要记录一些初始值，代码如下：

```
 // 第三步 -- 绑定 DOM 事件
Slider.prototype.bindDOM = function () {
 var self = this
 var scaleW = self.scaleW
 var outer = self.outer


 // 手指按下的处理事件
 var startHandler = function (evt) {
  // 记录刚刚开始按下的时间
  self.startTime = new Date() * 1

  // 记录手指按下的坐标
  self.startX = evt.touches[0].pageX
  self.startY = evt.touches[0].pageY

  // 清除偏移量
  self.offsetX = 0

  if(evt.touches.length >= 2){  //判断是否有两个点在屏幕上
    self.joinPinchScale = true; //进入双指方法状态
    self.pinchStart = evt.touches;  //得到第一组两个点
    self.pinchScaleEnd = self.pinchScale || (self.joinDbClickScale ? self.scaleMax : 1);  //记录最后一次缩放的值
  }

  if (evt.touches.length === 1) {
    self.oneTouch = true;
  }
 }

 ...

}
```

上面代码中，我们主要做了这几件事：

1. 记录手指按下时的初始坐标，这个是为了后续位移时，可以根据初始坐标来计算。
2. 判断当前的手指个数，如果是 2 个手指触发的 `touchstart`，那就证明是在双指方法操作，否则就是翻页操作。

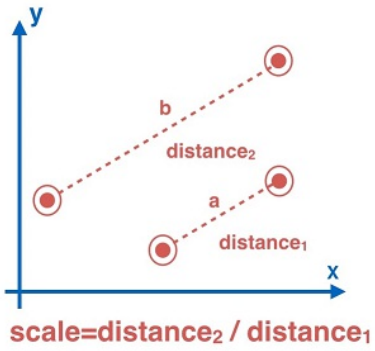
其次需要绑定 `touchmove` 事件，我们在 `touchmove` 时 会实现图片的位移，代码如下：

```
 // 第三步 -- 绑定 DOM 事件
Slider.prototype.bindDOM = function () {

 ...

 // 手指移动的处理事件
 var moveHandler = function (evt) {
  // 兼容chrome android，阻止浏览器默认行为
  evt.preventDefault()
  var target = evt.target;


  // 处理放大逻辑
  if (target.nodeName === 'IMG') {

    // 处理双指放大
    if (self.joinPinchScale && evt.touches.length >= 2) {
      var now = evt.touches;  //得到第二组两个点

      //得到缩放比例，getDistance是勾股定理的一个方法
      self.pinchScale = self.pinchScaleEnd*(getDistance(now[0],now[1])/getDistance(self.pinchStart[0],self.pinchStart[1]));

      // 首先将动画暂停
      target.style.webkitTransition = 'none';
```

```
      // 通过scale设置方法系数
      target.style.webkitTransform = 'scale3d('+self.pinchScale+', '+self.pinchScale+', 1)';

      return

    }


    //处理双击,双指放大状态时的拖动行为
    else if ((self.joinPinchScale || self.joinDbClickScale) && self.oneTouch) {
      // 计算手指的偏移量
      self._offsetX = (self._offsetEndX||0) + evt.targetTouches[0].pageX - self.startX;
      self._offsetY = (self._offsetEndY||0) + evt.targetTouches[0].pageY - self.startY;

      // 拖动时，保持图片缩放不变，只位移
      var _scale = self.joinPinchScale ? self.pinchScale : self.scaleMax;
      // 首先将动画暂停
      target.style.webkitTransition = 'none';
      target.style.webkitTransform = 'scale3d('+_scale+', '+_scale+', 1) translate3d('+(self._offsetX*0.5)+'px, '+(self._offsetY*0.5)+'px, 0)';
      return
    }

  }


  // 处理翻页逻辑
  if (self.oneTouch) {

    // 计算手指的偏移量
    self.offsetX = evt.targetTouches[0].pageX - self.startX

    var lis = outer.getElementsByTagName('li')
    // 起始索引
    var i = self.idx - 1
    // 结束索引
    var m = i + 3

    // 最小化改变DOM属性
    for (i; i < m; i++) {
      lis[i] && (lis[i].style.webkitTransition = '-webkit-transform 0s ease-out')
      lis[i] && (lis[i].style.webkitTransform = 'translate3d(' + ((i - self.idx) * self.scaleW + self.offsetX) + 'px, 0, 0)')
    }
  }

 }

 ....

}
```

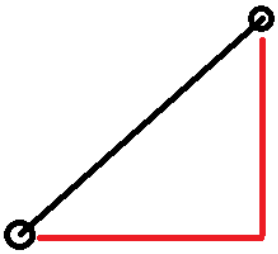
上面代码中，我们主要做了这几件事：

1. 在 touchmove 事件中，主要做两块逻辑 1）处理手指移动时的放大逻辑。2）处理手指移动时的翻页逻辑。
2. 放大逻辑的条件就是在 touchstart 记录的双指操作的标志位，有这个标志位就证明要进入双指放大逻辑。
3. 翻页逻辑的条件就是在 touchstart 记录的单指操作的标志位，有这个标志位就证明要进入翻页逻辑。
4. 双指放大，我们采用 css3 的 transform:scale3d 属性，这个属性通过 js 的 webkitTransform 来动态设置。
5. 在实现图片位移时，我们每次都要记录上次手指离开之后的位置坐标，然后计算出本次的位移起始量。
6. 双指放大，我们的两个手指的位移的不同方向，采用方向位移插值和勾股定理计算出放大缩小的位移量。原理如下图：



a 是两个手指开始时的位置，b 是两个手指位移后的位置，`distance1` 和 `distance2` 可以看成直角三角形的斜边，通过勾股定理计算出来，最终的缩放量采用 `distance1/distance2`。

勾股定理对应的 `getDistance` 原理和代码如下：



```
var getDistance = function (p1, p2) {
    // p1,p2代表两个手指的事件对象
    var x = p2.pageX - p1.pageX,
        y = p2.pageY - p1.pageY;
    return Math.sqrt((x * x) + (y * y)).toFixed(2);
};
```

最后需要在 `touchend` 时处理翻页的动画，标志位的重置等操作，代码如下:

```
...


// 手指抬起的处理事件
var endHandler = function (evt) {
  var target = evt.target;

  /****************下面开始处理标志位重置逻辑*************/

  //处理放大状态的拖动行为记录最后1次手指离开的坐标
  if (target.nodeName === 'IMG' && (self.joinDbClickScale || self.joinPinchScale)) {
    self._offsetEndX = self._offsetX;
    self._offsetEndY = self._offsetY;

    // 在双指缩放时，不允许缩放到原始尺寸小的值
    if (self.pinchScale < 1) {
      target.style.webkitTransition = '-webkit-transform .2s ease-in-out'
      target.style.webkitTransform = 'scale3d(1,1,1)'
      self.pinchScale = 1;
    }
  }

  // 重置标志位
  self.oneTouch = false;

  /****************下面开始处理翻页逻辑和动画*************/
  // 边界就翻页值
  var boundary = scaleW / 6

  // 手指抬起的时间值
  var endTime = new Date() * 1

  // 当手指移动时间超过300ms 的时候，说明是拖动(手指始终没有离开)操作，按设定临界值位移算
  if (endTime - self.startTime > 300) {
    // 如果超过临界值，就表示需要移动到下一页
    if (self.offsetX >= boundary) {
      // 上一页
      self.goIndex('-1')
    } else if (self.offsetX < 0 && self.offsetX < -boundary) {
      // 下一页
      self.goIndex('+1')
    } else {
      // 当前页
      self.goIndex('0')
    }
  } else {
    // 当手指移动时间不超过300ms 的时候，说明是swipe(手指很快离开)，按固定临界值算
    if (self.offsetX > 50) {
      self.goIndex('-1')
    } else if (self.offsetX < -50) {
      self.goIndex('+1')
    } else {
      self.goIndex('0')
    }
  }
}

...
```

上面代码中，我们主要做了这几件事：

1. 在 `touchend` 事件中，处理一些标志位的重置操作，即恢复到初始值。
2. 记录位移结束时的座标，就是我们上面提到的在实现图片位移时，我们每次都要记录上次手指离开之后的位置坐标，然后计算出在 `touchmove` 时的位移起始量。
3. 根据位移量来判断是否进入翻页逻辑即上一页，当前页，或者下一页。

双击放大事件：

在除了 `touchstart`，`touchmove`，`touchend` 这些事件之外另外需要我们完成的交互就是，双击放大效果，首先我们需要能够监听到双击事件，然后对图片进行缩放，代码如下：

```
...

// 双击放大事件
var dbHandler = function (evt) {

  var target = evt.target
  var d = evt

  // 确保点击的是图片
  if (target.nodeName === 'IMG') {

    // 如果在双击方法或者双指放大状态下有双击操作，就让图片恢复原样
    if (self.joinDbClickScale || self.joinPinchScale) {
      target.style.webkitTransition = '-webkit-transform .2s ease-in-out'
      target.style.webkitTransform = 'scale3d(1,1,1)'

      // 然后重置这2个标志位
      self.joinDbClickScale = false
      self.joinPinchScale = false

      self.pinchScale = 1;
    } else {

      // 否则就说明是要进行放大操作
      self.originX = d.offsetX;
      self.originY = d.offsetY;
      target.style.webkitTransition = '-webkit-transform .2s ease-in-out'
      target.style.webkitTransform = 'scale3d('+self.scaleMax+','+self.scaleMax+',1)'
      target.style.webkitTransformOriginX = self.originX + 'px'
      target.style.webkitTransformOriginY = self.originY + 'px'


      self.pinchScale = self.scaleMax;

      // 进入双击放大状态
      self.joinDbClickScale = true
    }
  }


}


var tapCloseHandler = function (evt) {
  self.wrap.style.opacity = 0
  setTimeout(function () {
    document.body.removeChild(self.wrap)
  }, 200)
}
var lastClickTime = 0
var clickTimer
var clkHandler = function (evt) {
  var nowTime = new Date().getTime()
  if (nowTime - lastClickTime < 230) {
    /* 双击 */
    lastClickTime = 0
    clickTimer && clearTimeout(clickTimer)
    dbHandler(evt)
  } else {
    /* 单击 */
    lastClickTime = nowTime
```

```
    clickTimer = setTimeout(function () {
      tapCloseHandler(evt)
    }, 230)
  }
}

...
```

上面代码中，我们主要做了这几件事：

1. 移动端的双击事件我们是采用的单击事件，在回调里判断第二次点击的时间间隔来模拟实现的，当然，各位也可以借助 hammer.js 来实现。
2. 如果已经在双击方法或者双指放大状态下有双击操作，就让图片恢复原样，反之就表明是进入双击放大的逻辑。
3. 借助 `transform` 的 `scale3d` 属性，可以将一个 dom 放大，而放大的原点，则利用 `transformOriginX` 和 `transformOriginY` 来设置。原点的座标就是手指双击时的座标 `offsetY` 和 `offsetX`。

注意：由于我们使用的 javascript 来设置元素的动画和 transform 样式，所以我们需要加上 `webkit` 的前缀，例如 `webkitTransform`， `webkitTransition` 等等。

将事件绑定到我们 outer 上：

`outer` 是在我们在开始的 `renderDOM` 方法中定义的外层父元素，我们要把事件绑定在 `outer` 上面，代码如下：

```
outer.addEventListener('touchstart', startHandler)
outer.addEventListener('touchmove', moveHandler)
outer.addEventListener('touchend', endHandler)
outer.addEventListener('click', clkHandler)
```

阻止 iOS 的原生双击放大功能

在 iOS 10 以后，Safari 已经不再识别我们在 `<meta>` 标签中设置的 `name="viewport" maximum-scale=1.0,minimum-scale=1.0,user-scalable=no"` 这个属性来阻止页面触发原生的双击放大功能了，所以需要使用另外的替代方法，代码如下：

```
window.onload = function() {

  // 阻止双指放大
  document.addEventListener('gesturestart', function(event) {
    event.preventDefault();
  });
}
```

小结：

本章节主要讲解了自研图片查看器 slider.js 的开发。
相关技术点：

1. 移动端 web 的图片查看器的基本功能，包括点击查看大图，前后拖动查看上一张和下一张，双击放大，双指放大查看功能。
2. 介绍了移动端 `touchstart`， `touchmove`， `touchend` 事件的用法结合图片查看器，实现拖动交互。难点在于放大之后的拖动行为，要记录每次拖动结束的值。
3. 双击事件的原理，以及利用 `transform` 的 `scale3d` 属性，可以将一个 dom 放大，利用 `transform` 的

translate3d，可以实现一个 dom 位移。

4. 最后想说的是，尽快现在有很多的开源图片查看器可以直接使用，但是大多数是基于 React 或者 Vue 版本的，这样的跨平台性并不是很好，并且如果直接采用开源的组件是可以实现我们的功能，但是其中的原理就不能很好的理解，所以本章节的意义就在于让学生明白其中的原理，授人以鱼不如授人以渔。

本章节完整源代码地址：Github

}

16 单条内容组件开发

18 下拉刷新组件开发