# 第2章 Javascript优化

## 课程介绍

- 事件流
- 事件委托
- 懒加载
- 预加载

## 2.1 事件流

### 2.1.1 事件流有几种？

事件流指从页面中接收事件的顺序，有冒泡流和捕获流。

当页面中发生某种事件（比如鼠标点击，鼠标滑过等）时，毫无疑问子元素和父元素都会接收到该事件，可具体顺序是怎样的呢？冒泡和捕获则描述了两种不同的顺序。

DOM二级事件规定事件流包括三个阶段：

1. 事件捕获阶段
2. 处于目标阶段
3. 事件冒泡阶段

如图1：

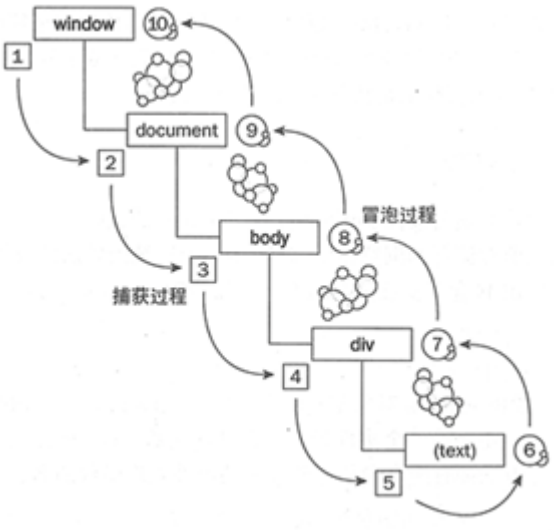



图1

假如我们点击一个div, 实际上是先点击document，然后点击事件传递到div,而且并不会在这个div就停下，div有子元素就还会向下传递，最后又会冒泡传递回document。

为了兼容更多的浏览器，非特殊情况一般我们都是把事件添加到在事件冒泡阶段。

### 2.1.1.1 事件冒泡

事件冒泡即事件开始时，由最具体的元素接收（也就是事件发生所在的节点），然后逐级传播到较为不具体的节点。

**实例：**

```
<body>
    <div id="parent">
        <button id="child" >click me</button>
    </div>
</body>

<script type="text/javascript">
    var parent = document.getElementById('parent');
    var child = document.getElementById('child');

    child.addEventListener("click", function(e){
        console.log('1. You click child');
    }, false);

    parent.addEventListener("click", function(e){
        console.log('2. You click parent');
    }, false);

    document.body.addEventListener("click", function(e){
        console.log('3. You click body');
    }, false);

    document.addEventListener("click", function(e){
        console.log('4. You click document');
    }, false);

    window.addEventListener("click", function(e){
        console.log('5. You click window');
    }, false);
</script>
```

效果如图2所示：

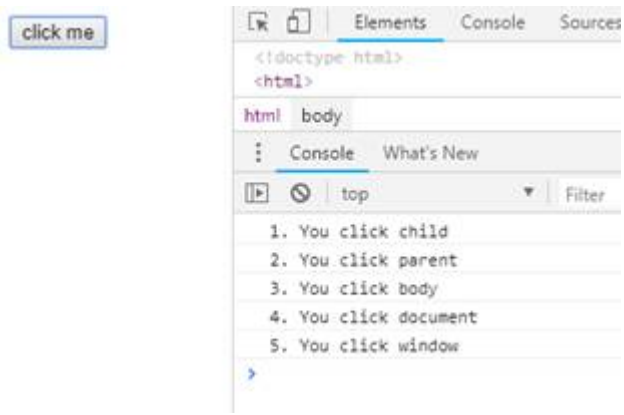



图2

在代码所示的页面中，如果点击了button，那么这个点击事件会按如下的顺序传播（Chrome浏览器）：

1. Button
2. Button的父级div
3. body
4. document
5. window

也就是说，click事件首先在 `<button>` 元素上发生，然后逐级向上传播。这就是事件冒泡。

### 2.1.1.2 事件捕获

事件捕获的概念，与事件冒泡正好相反。它认为当某个事件发生时，父元素应该更早接收到事件，具体元素则最后接收到事件。比如说刚才的实例（01.事件冒泡.html），将false参数改为true，变成事件捕获。(addEventListener最后一个参数，*为true则代表使用事件捕获模式*，false则表示使用事件冒泡模式。)

**实例：**

```
<body>
    <div
         id="parent">
        <button id="child" >click me</button>
    </div>
</body>

<script type="text/javascript">
    var parent = document.getElementById('parent');
    var child = document.getElementById('child');

    child.addEventListener("click", function(e){
        console.log('1. You click child');
    }, true);

    parent.addEventListener("click", function(e){
        console.log('2. You click parent');
    }, true);

    document.body.addEventListener("click", function(e){
        console.log('3. You click body');
    }, true);

    document.addEventListener("click", function(e){
        console.log('4. You click document');
    }, true);

    window.addEventListener("click", function(e){
        console.log('5. You click window');
    }, true);
</script>
```

事件发生顺序会是这样的：

1. window

2. document
3. body
4. Button的父级div
5. Button

效果如图3所示：

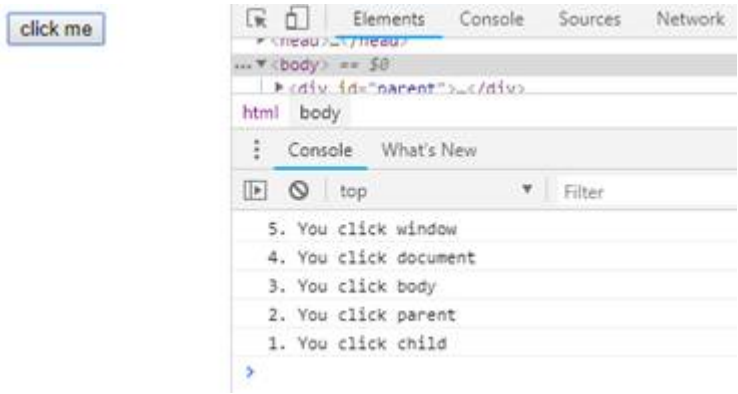



图3

## 2.1.2 阻止事件冒泡

事件冒泡过程，是可以被阻止的。防止事件冒泡而带来不必要的错误和困扰。

这个方法就是: stopPropagation() ，我们对 button 的click事件做一些改造。

**实例：**

```
<body>
    <div id="parent">
        <button id="child" >click me</button>
    </div>
</body>

<script type="text/javascript">
    var parent = document.getElementById('parent');
    var child = document.getElementById('child');

    child.addEventListener("click", function(e){
        console.log('1. You click child');
        e.stopPropagation();
    }, false);

    parent.addEventListener("click", function(e){
        console.log('2. You click parent');
    }, false);

    document.body.addEventListener("click", function(e){
        console.log('3. You click body');
    }, false);

    document.addEventListener("click", function(e){
        console.log('4. You click document');
    }, false);
```

```
    window.addEventListener("click", function(e){
        console.log('5. You click window');
    }, true);
</script>
```

效果如图5所示：

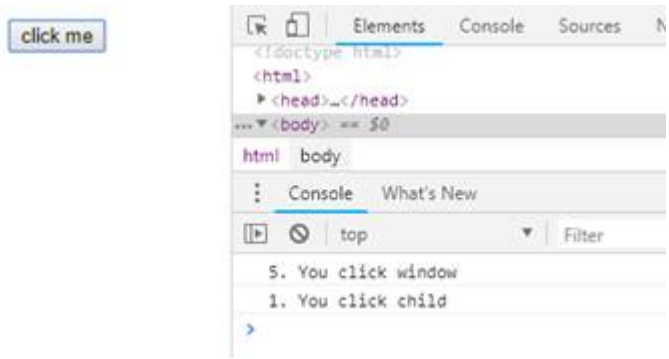



图5

不难看出，事件在到达具体元素后，停止了冒泡。但不影响父元素的事件捕获。

## 2.2 事件委托

每当将事件处理程序制定给元素时，运行中的浏览器代码与支持页面交互的JS代码之间就会建立一个连接，而这种连接越多，页面执行起来就越慢。考虑内存和性能问题，为了解决事件处理程序过多的问题，采用事件委托变得很有必要。（考虑到内存，也应该尽量减少不必要的事件处理程序，对于内存中过时不用的'空事件处理程序'，也是很有必要将其移除的；）

因为冒泡机制，比如既然点击子元素，也会触发父元素的点击事件，那我们完全可以将子元素的事件要做的事写到父元素的事件里，也就是将子元素的事件处理程序写到父元素的事件处理程序中，这就是事件委托；利用事件委托，只指定一个事件处理程序，就可以管理某一个类型的所有事件；

**实例：**

```
<body>
    <ul>
        <li>aaa</li>
        <li>bbb</li>
        <li>ccc</li>
        <li>ddd</li>
    </ul>
    <script src="js/eventUtil.js" type="text/javascript" charset="utf-8"></script>
    <script type="text/javascript">
        var ul=document.getElementsByTagName('ul')[0];
        var myHandlers=function(event){
            event=EventUtil.getEvent(event);
            var target=EventUtil.getTarget(event);
            console.log(target.innerHTML);
        };
        EventUtil.addHandler(ul,'click',myHandlers);
    </script>
</body>
```

这样相当就可以点击每个li的时候去获取到li里面的内容。

# 2.3 懒加载和预加载

## 2.3.1 懒加载

### 2.3.1.1 什么是懒加载？

懒加载也就是延迟加载。当访问一个页面的时候，先把img元素或是其他元素的背景图片路径替换成一张大小为1*1px图片的路径（这样就只需请求一次，俗称占位图），只有当图片出现在浏览器的可视区域内时，才设置图片正真的路径，让图片显示出来。这就是图片懒加载。

### 2.3.1.2 为什么要有懒加载？

很多页面，内容很丰富，页面很长，图片较多。比如说各种商城页面。这些页面图片数量多，而且比较大，少说百来K，多则上兆。要是页面载入就一次性加载完毕，估计用户已经失去耐心关闭网页了。

### 2.3.1.3 原理是什么？

页面中的img元素，如果没有src属性，浏览器就不会发出请求去下载图片，只有通过javascript设置了图片路径，浏览器才会发送请求。懒加载的原理就是先在页面中把所有的图片统一使用一张占位图进行占位，把正真的路径存在元素的“data-url”（一般这个属性我们自己定义）属性里，要用的时候就取出来，再设置。

### 2.3.1.4 懒加载的实现步骤？

1)首先，不要将图片地址放到src属性中，而是放到其它属性(data-original)中。

2)页面加载完成后，根据scrollTop判断图片是否在用户的视野内，如果在，则将data-original属性中的值取出存放到src属性中。

3)在滚动事件中重复判断图片是否进入视野，如果进入，则将data-original属性中的值取出存放到src属性中。

### 2.3.1.5 懒加载的优点？

页面加载速度快、可以减轻服务器的压力、节约了流量,用户体验好

### 2.3.1.6 代码实现

**实例：**

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>页面懒加载</title>
        <style type="text/css">
            img {
                display: block;
                width: 500px;
                height: 400px;
            }
        </style>
```

```
    </head>
    <body>
        <img src="" class="lazyload" data-
src="http://img.taopic.com/uploads/allimg/130104/240392-13010415433920.jpg">
        <img src="" class="lazyload" data-
src="http://old.bz55.com/uploads/allimg/150210/139-150210134411-50.jpg">
        <img src="" class="lazyload" data-
src="http://imgsrc.baidu.com/imgad/pic/item/7c1ed21b0ef41bd5221ae1d15bda81cb39db3d4d.jp
g">
        <img src="" class="lazyload" data-
src="http://imgsrc.baidu.com/imgad/pic/item/faf2b2119313b07e8c62703307d7912397dd8c29.jp
g">
        <img src="" class="lazyload" data-
src="http://imgsrc.baidu.com/imgad/pic/item/7e3e6709c93d70cf48ebdd13f2dcd100baa12b9e.jp
g">
        <script type="text/javascript">
            var imgs = document.querySelectorAll('img.lazyload');
            var n = 0;//存储图片加载到的位置，避免每次都从第一张图片开始遍历
            window.onscroll = function() {
                showImg();
            };

            function showImg(){
                var seeHeight = document.documentElement.clientHeight;
                var scrollTop = document.body.scrollTop ||
document.documentElement.scrollTop;
                for (var i = n; i < imgs.length; i++) {
                    if (imgs[i].offsetTop < seeHeight + scrollTop) {
                        if (imgs[i].getAttribute('src') == '') {
                            imgs[i].src = imgs[i].getAttribute('data-src');
                        }
                        n = i + 1;
                    }
                }
            }
            showImg()  // 第一次进入页面就要调用一次
        </script>
    </body>
</html>
```

## 2.3.2 预加载

### 2.3.2.1 什么是预加载？

提前加载图片，当用户需要查看时可直接从本地缓存中渲染。

### 2.3.2.2 为什么要使用预加载？

图片预先加载到浏览器中，访问者便可顺利地在你的网站上冲浪，并享受到极快的加载速度。这对图片画廊及图片占据很大比例的网站来说十分有利，它保证了图片快速、无缝地发布，也可帮助用户在浏览你网站内容时获得更好的用户体验。

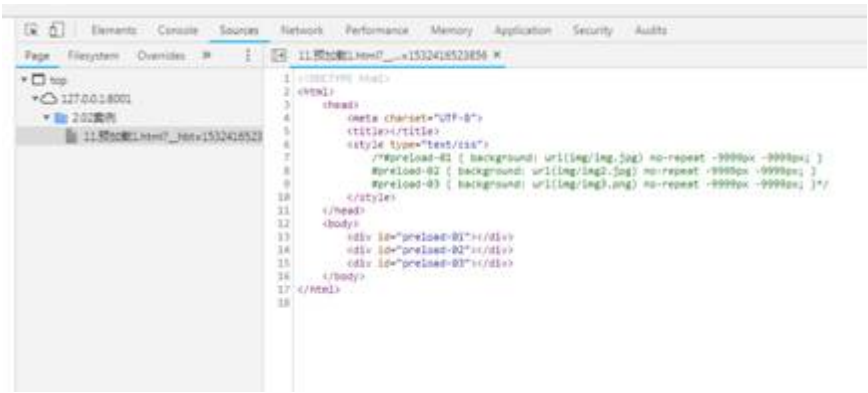
### 2.3.2.3 实现预加载的方法有哪些？

**方法一：** 用CSS和JavaScript实现预加载

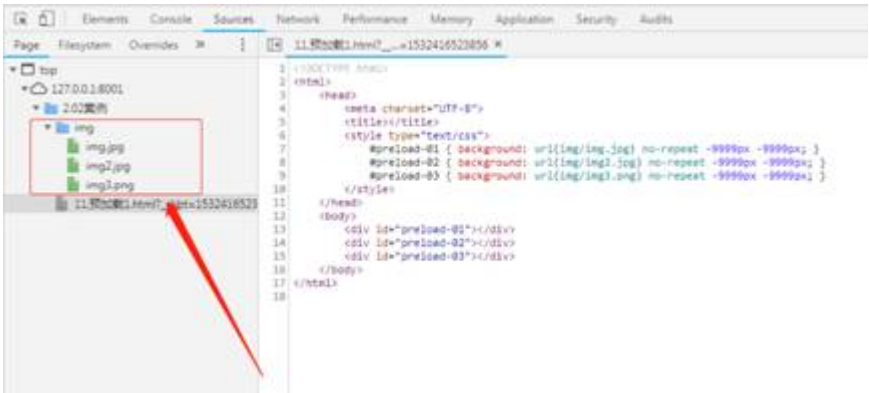
单纯使用CSS，可容易、高效地预加载图片。

**实例：**

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
        <style type="text/css">
            #preload-01 { background: url(img/img.jpg) no-repeat -9999px -9999px; }
            #preload-02 { background: url(img/img2.jpg) no-repeat -9999px -9999px; }
            #preload-03 { background: url(img/img3.png) no-repeat -9999px -9999px; }
        </style>
    </head>
    <body>
        <div id="preload-01"></div>
        <div id="preload-02"></div>
        <div id="preload-03"></div>
    </body>
</html>
```



如上图，在将样式代码注释的时候，我们打开浏览器的调试功能，查看Sources资源，里面的图片是未加载的状态，我们是无论如何也是无法查找到那3张图片的。



如上图，在将样式代码放开的情况下，我们可以看到在Sources资源中，已经将3张图片加载完毕了。

简单的思路，就是我们通过css去加载所需要的图片，但是通过背景位置的调整到页面之外的部分，可是图片还是被浏览器缓存了。之后再页面的其他地方我们还是可以直接使用，从缓存中读取。

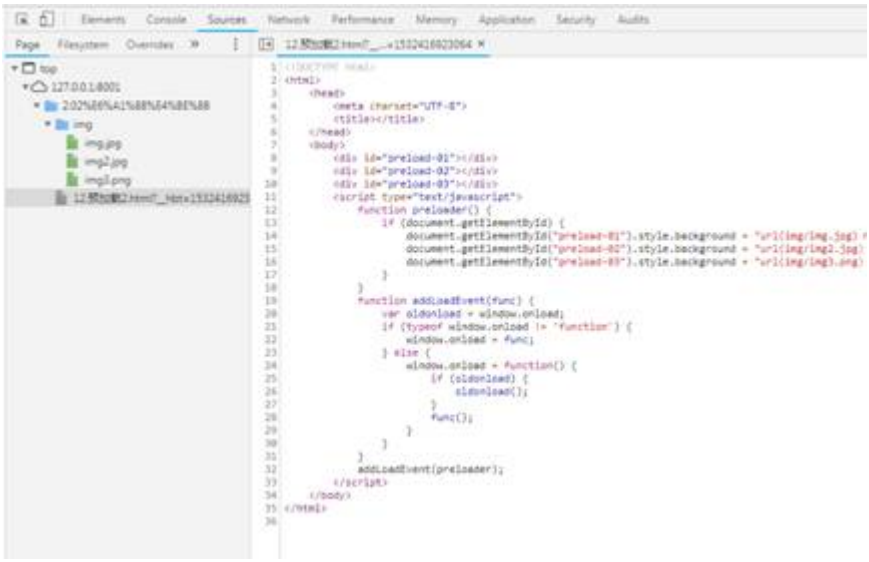
上面的这个方法虽然比较好，但是出现一种情况,如果这个CSS代码在加载的时候读取到需要加载背景图，正好背景图又比较大，这样整个页面的耗时比较厉害。我们最好是等到页面加载完成之后再去加载需要预加载的图。那这个时候，我们使用JS来进行加载还是比较友好的。

**实例：**

```
<!DOCTYPE html>
<html>
    <head>
       <meta charset="UTF-8">
       <title></title>
    </head>
    <body>
       <div id="preload-01"></div>
       <div id="preload-02"></div>
       <div id="preload-03"></div>
       <script type="text/javascript">
           function preloader() {
               if (document.getElementById) {
                   document.getElementById("preload-01").style.background =
"url(img/img.jpg) no-repeat -9999px -9999px";
                   document.getElementById("preload-02").style.background =
"url(img/img2.jpg) no-repeat -9999px -9999px";
                   document.getElementById("preload-03").style.background =
"url(img/img3.png) no-repeat -9999px -9999px";
               }
           }
           function addLoadEvent(func) {
               var oldonload = window.onload;
               if (typeof window.onload != 'function') {
                   window.onload = func;
               } else {
                   window.onload = function() {
                       if (oldonload) {
                           oldonload();
                       }
                       func();
                   }
               }
           }
           addLoadEvent(preloader);
       </script>
    </body>
</html>
```

我们这段代码里面，第一段内容写了一个preloader方法用于将css样式写入节点里去。第二段代码我们去监听是否页面加载完成，只有页面加载完成之后我们才会去执行预加载。

同样的，我们可以去查看浏览器中的Sources资源是否已经将我们想要预加载的图片缓存了。



显而易见的，图片已经被浏览器缓存了。

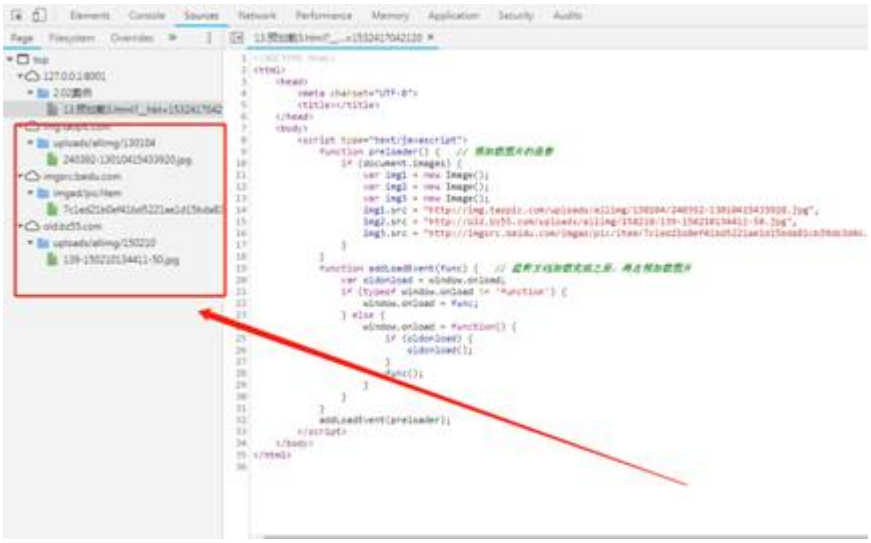
**方法二：** 仅使用JavaScript实现预加载

**实例：**

```
<script type="text/javascript">
    function preloader() {   // 预加载图片的函数
        if (document.images) {
            var img1 = new Image();
            var img2 = new Image();
            var img3 = new Image();
            img1.src = "http://img.taopic.com/uploads/allimg/130104/240392-
13010415433920.jpg",
                img2.src = "http://old.bz55.com/uploads/allimg/150210/139-150210134411-
50.jpg",
                img3.src =
"http://imgsrc.baidu.com/imgad/pic/item/7c1ed21b0ef41bd5221ae1d15bda81cb39db3d4d.jpg"
        }
    }
    function addLoadEvent(func) {   // 监听文档加载完成之后，再去预加载图片
        var oldonload = window.onload;
        if (typeof window.onload != 'function') {
            window.onload = func;
        } else {
            window.onload = function() {
                if (oldonload) {
                    oldonload();
                }
                func();
            }
        }
    }
    addLoadEvent(preloader);
</script>
```

该方法尤其适用预加载大量的图片。一般画廊网站使用该技术，将该脚本应用到其他页面，当跳转到这个页面之后所有的内容图片已经被浏览器缓存。

同样的，我们可以去查看浏览器中的Sources资源是否已经将我们想要预加载的图片缓存了。



这次使用的是网络图片，但还是同样的加载在浏览器缓存中了。

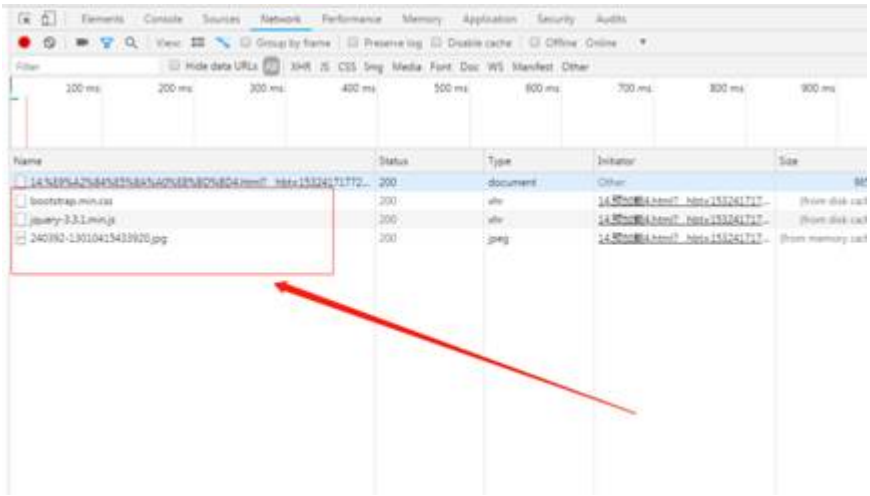
**方法三：** 使用Ajax实现预加载

我们使用Ajax实现图片预加载的方法。该方法利用DOM，不仅仅预加载图片，还可以预加载CSS、JavaScript等相关的东西。使用Ajax，比直接使用JavaScript，优越之处在于JavaScript和CSS的加载不会影响到当前页面。该方法简洁、高效。

**实例：**

```
<script type="text/javascript">
    window.onload = function() {
        setTimeout(function() {
            // 发送请求加载 JS ，CSS文件
            var xhr = new XMLHttpRequest();
            xhr.open('GET', 'https://code.jquery.com/jquery-3.3.1.min.js');
            xhr.send('');
            xhr = new XMLHttpRequest();
            xhr.open('GET',
'https://cdn.bootcss.com/bootstrap/3.3.7/css/bootstrap.min.css');
            xhr.send('');
            // 加载图片
            new Image().src = "http://img.taopic.com/uploads/allimg/130104/240392-
13010415433920.jpg";
        }, 1000);    // 定时器防止请求事件过长
    };
</script>
```

而这一次，因为是使用了ajax的请求，我们打开浏览器中Network窗口，刷新试试看结果。



我们可以明显的看到样式代码，插件代码，以及图片都已经加载到浏览器缓存中了。

## 2.3.3 懒加载和预加载的对比

### 2.3.3.1 概念

**懒加载也叫延迟加载：**JS图片延迟加载,延迟加载图片或符合某些条件时才加载某些图片。

**预加载：**提前加载图片，当用户需要查看时可直接从本地缓存中渲染。

### 2.3.3.2 区别

**两种技术的本质：**两者的行为是相反的，一个是提前加载，一个是迟缓甚至不加载。懒加载对服务器前端有一定的缓解压力作用，预加载则会增加服务器前端压力。

### 2.3.3.3 懒加载的意义及实现方式

**意义：**

懒加载的主要目的是作为服务器前端的优化，减少请求数或延迟请求数。

**实现方式：**

1. 第一种是纯粹的延迟加载，使用setTimeOut或setInterval进行加载延迟.
2. 第二种是条件加载，符合某些条件，或触发了某些事件才开始异步下载。
3. 第三种是可视区加载，即仅加载用户可以看到的区域，这个主要由监控滚动条来实现，一般会在距用户看到某图片前一定距离遍开始加载，这样能保证用户拉下时正好能看到图片。

### 2.3.3.4 预加载的意义及实现方式

**意义:**预加载可以说是牺牲服务器前端性能，换取更好的用户体验，这样可以使用户的操作得到最快的反映。

**实现方式：**实现预载的方法非常多，比如：用CSS和JavaScript实现预加载；仅使用JavaScript实现预加载；使用Ajax实现预加载。

常用的是new Image();设置其src来实现预载，再使用onload方法回调预载完成事件。只要浏览器把图片下载到本地，同样的src就会使用缓存，这是最基本也是最实用的预载方法。当Image下载完图片头后，会得到宽和高，因此可以在预载前得到图片的大小(方法是用记时器轮循宽高变化)。

# 2.4 课程总结

- 事件流
  - 事件冒泡
  - 事件捕获
  - 事件捕获阶段
  - 处于目标与事件冒泡阶段
  - 阻止事件冒泡
- 事件委托
  - 利用事件委托，只指定一个事件处理程序，就可以管理某一个类型的所有事件
- 懒加载和预加载
  1. 懒加载原理

     懒加载的实现步骤

     懒加载的优点
  2. 预加载原理

     实现预加载的方法

     懒加载和预加载的对比

     懒加载的意义及实现方式

     预加载的意义及实现方式

## 2.4 实训

下载jquery.lazyload.js，并且使用该插件，实现一组图片的懒加载功能。