# Ajax入门和应用

第3天课堂笔记（本课程共5天）

# 目录

[Ajax入门和应用 1](#_Toc493454507)

[目录 1](#_Toc493454508)

[一、复习 2](#_Toc493454509)

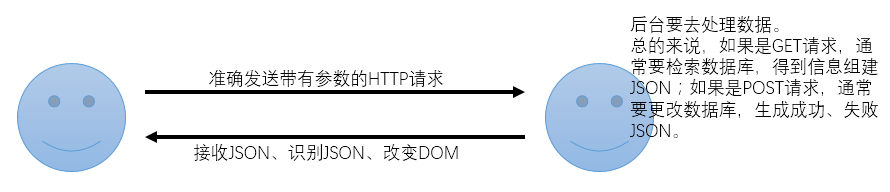
[二、瀑布流 4](#_Toc493454510)

[三、JSONP跨域 11](#_Toc493454511)

[3.1 JSONP概念 11](#_Toc493454512)

[3.2 JSONP实用函数封装 14](#_Toc493454513)

# 一、复习



发送请求：

|  |
| --- |
| j.get("check.php",{"xuehao":199991},function(err,data){  }); |

|  |
| --- |
| j.post("check.php",{"xuehao":199991},function(err,data){  }); |

data返回的可能是JSON样子的字符串，所以我们要把这个字符串变为真正的对象JSON：

|  |
| --- |
| var jsonObj = eval("(" + data + ")");  var jsonObj = (new Function("return " + data))();  var jsonObj = JSON.parse(data); |

我们要会识别、解析JSON。枚举属性用.，枚举数组用[]。

|  |
| --- |
| {  "name" : "小明",  "sex" : "男",  "age" : 12,  "award" : [  {  "title" : "三好学生",  "time" : "2016年"  },  {  "title" : "三好学生",  "time" : "2015年"  },  {  "title" : "三好学生",  "time" : "2014年"  }  ]  } |

比如拿到2015年：

|  |
| --- |
| dataObj.award[1].time |

接下来就要通过JSON来对DOM进行处理，我们发现处理DOM（尤其是增加节点），不管是字符串方法、DOM方法都复杂，都需要连字符。而我们不喜欢连字符，提出模板技术。

|  |
| --- |
| // 模板字符串  var str = "我早上吃了<%=fan%>，花了<%=qian%>元，我很<%=xinqing%>啊！";  // 字典  var dictionary = {  "fan" : "煎饼",  "qian" : 4.5,  "xinqing" : "高兴"  }  // 数据绑定，（或者叫做数据填充，数据注入）  str = str.replace(/\<\%\=(\w+)\%\>/g,function(match,$1){  return dictionary[$1];  });  alert(str); |

所有的模板都写在

|  |
| --- |
| <script type="text/template" id="temp1"></script> |

里面，这是一个放模板的容器，可以通过

|  |
| --- |
| $("#temp1").html(); |

得到模板字符串。

前端Ajax技术，特别是模板技术，更改了企业里面的习惯。

2011年之前，后台处理模板

|  |
| --- |
| <?php  while($row = mysql\_fetch\_array($result)){  ?>  <div class="news">  <a href="**<?php echo $row["link"]; ?>**">  <div class="pic">  <img src="" alt="" />  </div>  <div class="content">  <h3>**<?php echo $row["title"]; ?>**</h3>  <div class="summary">**<?php echo $row["summary"]; ?>**</div>  <div class="info">  <span class="zuozhe">**<?php echo $row["link"]; ?>**</span>  <span class="shijian">**<?php echo $row["link"]; ?>**</span>  <span class="yuedu">阅读(**<?php echo $row["link"]; ?>**)</span>  </div>  </div>  </a>  </div>  <?php  }  ?> |

2011年之后，开始流行前后端分离。后台代码不要出现在前端！！后台只负责输入JSON，然后前端调用。

|  |
| --- |
| <script type="text/template" id="temp">  <div class="news">  <a href="<%=m\_display\_url%>">  <div class="pic">  <img src="<%=m\_image\_url%>" alt="" />  </div>  <div class="content">  <h3><%=m\_title%></h3>  <div class="summary"><%=m\_summary%></div>  <div class="info">  <span class="zuozhe"><%=m\_writer\_name%></span>  <span class="shijian"><%=m\_create\_time%></span>  <span class="yuedu"><%=hotcount%></span>  </div>  </div>  </a>  </div>  </script> |

革命式的里程碑：

table →→→→ div + css

后台模板 →→→→ 前端模板，前后端分离

前端开发工程师更重要的不是轮播图、浮动定位，在今天，必须要碰到数据！！

# 

# 二、瀑布流

复习一下：

发：HTTP请求，带参数。GET、POST，参数在哪儿。缓存的防止、404。jQuery发送请求。$.get(); $.post();

收：JSON → DOM。 模板引擎，JSON看做是字典，jQuery创建新DOM。

后台：生成JSON，数据库中的内容怎么变为JSON，后台哥哥通常怎么设置get请求。

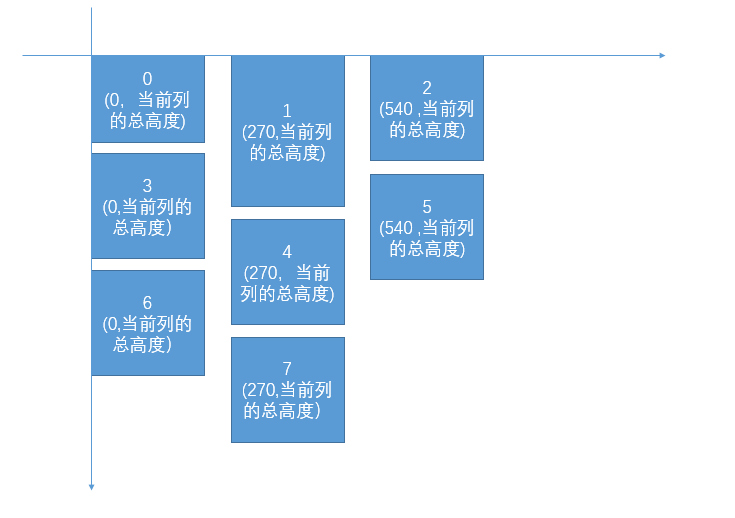
模型：Ajax提交表单、Ajax滚动到底部加载更多、Ajax分页

瀑布流，今天学习的是等列宽瀑布流。页面上有5列（会根据页面宽度来调整），列的宽度是一样的，每列都有小格grid，每个grid内容尺寸不一样（图片尺寸、内容多少不一样），所以每个grid的高度不一致，整体的看，就是错落有致的感觉。



实现瀑布流的策略：绝对定位法、浮动法。

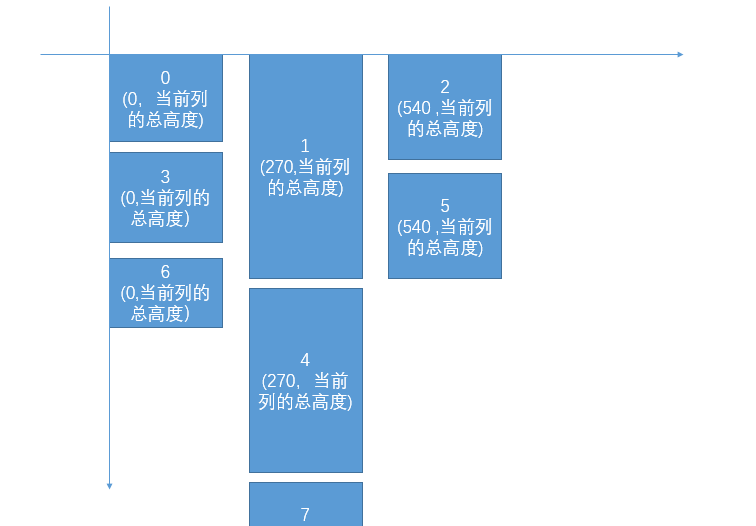
算法：



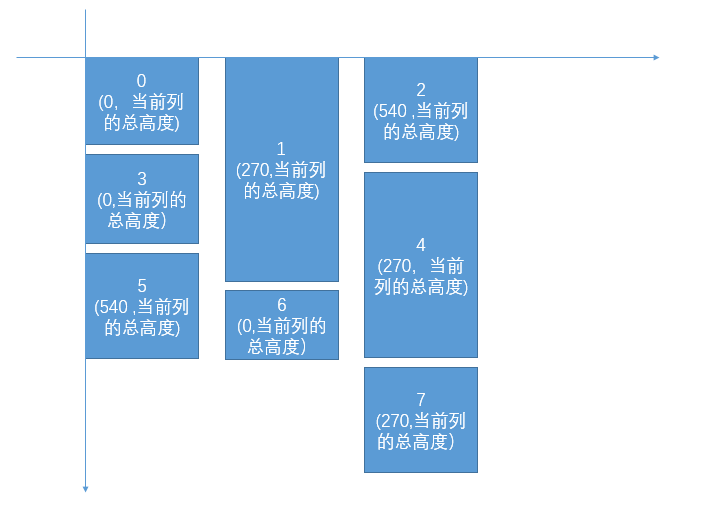
每个grid都绝对定位，jQuery来控制他们的left值和top值。left值根据自己的编号与3的余数来定，比如列宽250，间隔是20，所以left值就是(index % 3 \* 270)。top值就直接看当前列总高度就行了。

**上面的算法看似正确，但实际上有两个非常大的BUG：**

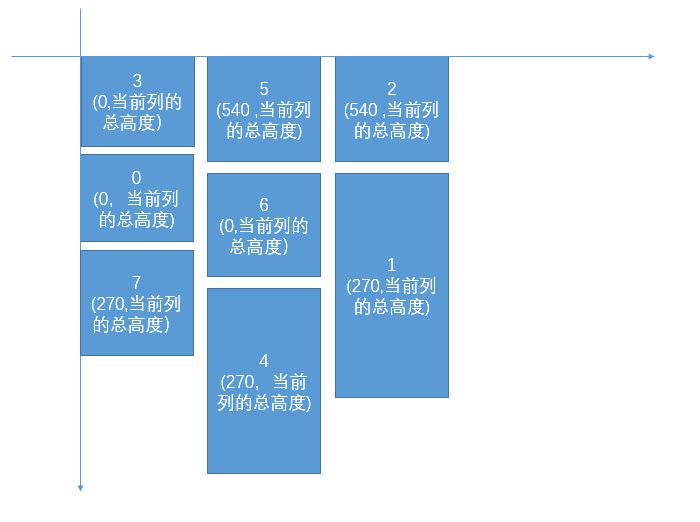
1. 底部难以平齐，当grid的高度在某一列有明显的高度差别的时候，比如某一列都是高个儿，一下子底边不平齐了。



解决方法就是不看每个grid的编号来决定在第几列，而是看当前哪个列总height最小，就上哪个列：



第二，图片不一定是按自然顺序上树！而应该是谁先加载完，谁先上树！这样上树的元素一定是立即可见的，并且能够保证网页性能是最优的。HTTP是多线程、并发的，一次性能够发出HTTP请求，HTTP请求彼此异步。



总结：

1. 哪列最短，上哪列；
2. 谁先回来，谁先上。

我们现在要知道一个图片的HTTP请求什么时候发出。img一旦上DOM了，就肯定能发出HTTP请求。但是，我们现在不上DOM，也要把请求发出去，看谁先load了，谁先上DOM。

此时使用一个办法轻松解决：

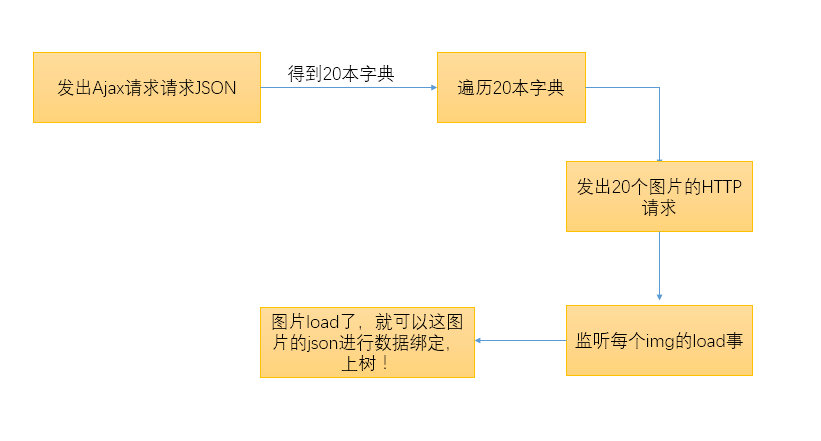
|  |
| --- |
| //创建一个img孤儿节点，等价于：  //var img = document.createElement("img");  var img = new Image();  //一旦给孤儿节点设置src属性，HTTP上行请求将发出  img.src = dictionary.imgurl; |

此时就能够监听每个img的load事件：

|  |
| --- |
| //监听图片的load事件  $(img).load(function(){  console.log($(this).attr("src") + "我加载成功了");  }); |

也就是说，用(new Image()).src来发出请求，用load事件来监听请求回来。

时间线：



我们下面利用一个闭包的特性，让每个img自动记住自己的dictionary。

|  |
| --- |
| // 发出Ajax请求  $.get("data/json1.txt",function(data){  var dataobj = eval("(" + data + ")");    //遍历json，得到20本字典  \_.each(dataobj.news,function(dictionary){  var domstr = compiled(dictionary);  // 创建一个img孤儿节点，等价于：  // var img = document.createElement("img");  var img = new Image();  // 一旦给孤儿节点设置src属性，HTTP上行请求将发出  img.src = dictionary.imgurl;  //监听图片的load事件  $(img).load(function(){  // 此图片已经加载完毕了，此时就可以数据绑定、上树了。  // 由于有闭包特性，所以dictionary就是这个图片的字典。不会串。  var domstr = compiled(dictionary);  //上树  $(domstr).appendTo($waterfall);  });  });  }); |

闭包特性小题目，

|  |
| --- |
| for(var i = 0 ; i < 10 ; i++){  setTimeout(function(**i**){  fun(**i**);  },2000,**i**);  }  function fun(a){  console.log(a);  } |

输出0~9

|  |
| --- |
| for(var i = 0 ; i < 10 ; i++){  setTimeout(function(){  fun(i);  },2000);  }  function fun(a){  console.log(a);  } |

输出10个10

|  |
| --- |
| for(var i = 0 ; i < 10 ; i++){  fun(i);  }  function fun(a){  setTimeout(function(){  console.log(a);  }，2000);  } |

输出10个10

|  |
| --- |
| for(var i = 0 ; i < 10 ; i++){  setTimeout(function(){  (function(i){  fun(i);  })(i);  }，2000);  }  function fun(a){  console.log(a);    } |
|  |

输出10个10

上面解决的事情就是第一个大事情，就是发出HTTP请求，把控谁先load谁先上树！

下面就是第二个大事情，瀑布流grid的left、top设置。

此时，我们用的算法就是哪列最短，进哪列。用一个数组，维护每列的总高度。

|  |
| --- |
| var colHeight = [0,0,0,0,0] |

此时3号盒子最先load了，发现数组中最小项是第0项，此时插入到第0列去，left值为0，top为0。并且把自己的高度写入数组，自己的高度是288。

|  |
| --- |
| [288,0,0,0,0] |

此时1号盒子load了，发现数组中最小项是第1项，此时插入到第1列去，left值为270，top为0。并且把自己的高度写入数组，自己的高度是388。

|  |
| --- |
| [288,388,0,0,0] |

……此时0号、2号、4号也load了，同理：

|  |
| --- |
| [288,388,**166**,666,466] |

此时7号盒子load了，上第2列！（因为第2列166最小），此时把自己的高度300加上去

|  |
| --- |
| [288,388,**466**,666,466] |

…………

今后要敏感一些！什么找到数组中的最小值、最大值、平均数、综合、排序、计数等等，还有一些关于对象的操作，**想都别想，直接用underscore！你有时间去写，还不如抽抽烟。**

看清楚这两个API的区别：

|  |
| --- |
| var colHeight = [11,33,4,22,99];  var min = **\_.min(**colHeight**)**; //得到数组中最小值，返回4  var mincolnumber = **\_.indexOf(**colHeight,min**)**; //得到某个值的下标，返回2 |

每个grid的left值、top值都是根据当前最小列编号来定的，比如最小列编号是m：

left值就是m \* 270

top值就是colHeight[m]

http://loading.io/

说一下响应式的实现，5列的时候，和4列的grid的变化是动画实现。动画用

|  |
| --- |
| animate(终点,1000); |

终点是什么？需要我们一个一个grid算，算每个人新终点！

所以就要清空我们colHeightArray数组，变为0。重新进行“找最短插入，自己的高加上去”这一系列算法。得到了新的位置，animate就行了。

# 三、JSONP跨域

## 3.1 JSONP概念

Ajax不能跨域，比如你的页面在127.0.0.1里，不能请求127.0.0.2里面的JSON文件。比如，你在www.163.com里，不能请求www.sohu.com里面的JSON文件。

域名不同（实际上ip不同也不行），Ajax不能跨域名的。比如news.sina.com.cn能不能请求ent.sina.com.cn不一定！如果news这个机器和ent是一个ip，就能，如果不是一个机器，就不能。

比如我们现在的页面运行在127.0.0.1，试图请求127.0.33.88的文件，会报错：

|  |
| --- |
| $.get("**http://127.0.33.88/**test.txt",function(data){  alert(data);  }); |



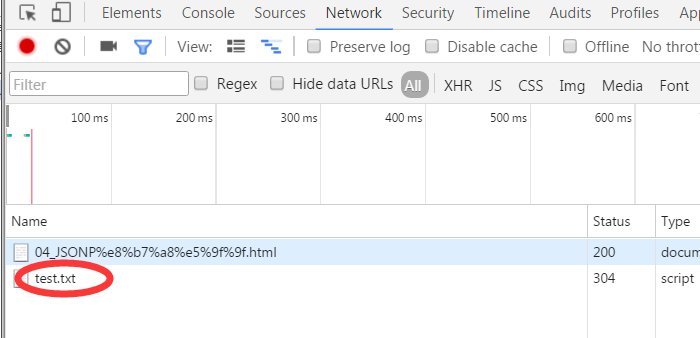
你觉得不能跨域这个事情，是浏览器的限制，还是服务器的限制？是服务器的限制，服务器有机制阻止来自其他网址的XHR请求。

比如A服务器上运行着页面，B服务器上有数据库（有JSON），现在A必须去请求B，怎么办？用JSONP。

JSONP不是Ajax！是一种奇淫技巧，发明人不可考，近些年变得流行。

首先，我们写一个script标签，这个标签现在有一个src属性，指向一个txt文件。此时将会发出HTTP请求，请求这个文件。

|  |
| --- |
| <script type="text/javascript" src="test.txt"></script> |



我们现在请求不变：

|  |
| --- |
| <script type="text/javascript" src="test.txt"></script> |

但是改变文件

|  |
| --- |
| alert("你好"); |

此时你好就能弹出，这是因为script标签会引包，同时也执行了js文件。

问题是，**HTML页面没有变量能够持有这个你好两个字**，所以我们现在就必须用奇淫技巧：

|  |
| --- |
| fun("你好"); |

我们之前都是在js文件中定义一个函数，在主页面中调用；而今天我们给反过来了！！在主页面定义函数，在js文件中调用函数！！

主页面中定义函数：

|  |
| --- |
| function fun(obj){  alert(obj.result[1].nianling);  } |

在外部文件中调用函数：

|  |
| --- |
| fun("你好"); |

此时外部文件的数据，被当做了实参传入了HTML页面，被形参接收。

如果参数是JSON呢？那JSON不就进页面了么！！我们起了一个名字，JSON with Padding。简称JSONP。

红色的部分（调用函数语句）就是所谓的padding。

|  |
| --- |
| **fun(**  {  "result" : [  {  "xingming":"小明",  "nianling" : 12,  "sex" : "男"  },  {  "xingming":"小红",  "nianling" : 13,  "sex" : "女"  },  {  "xingming":"小绿",  "nianling" : 9,  "sex" : "女"  }  ]  }  **);** |

JSONP是可以跨域的，垫付了Ajax的写法。甚至不用出现XMLHttpRequest对象就能实现Ajax。

JSONP刚刚诞生的时候，整个世界觉得被戏弄了。

JSONP是函数的调用，通过实参把数据传递到HTML页面中。

|  |
| --- |
| <script type="text/javascript">  function fun(obj){  **→ 定义函数**    }  </script>  <script type="text/javascript" src="test.txt"></script>  **→ 调用函数** |

JSONP不安全。因为如果你的JSONP文件，被人篡改了，你的页面就等于被黑了~~。

所以使用JSONP的时候，如果JSONP文件是自己公司的，放心大胆用，一点毛病没有！

## 3.2 JSONP实用函数封装

我们现在能不能封装一个类似：

|  |
| --- |
| myajax.get(URL,{},function(){  }); |

这样的函数。

主页面：

|  |
| --- |
| myjsonp("http://www.jd.com/sadfadsf/32rwe/wer",function(data){  //data就是JSONP请求回来的值  },"fetchJSON\_comment98vv21142"); |

JSONP外面的文件在调用函数，

|  |
| --- |
| fetchJSON\_comment98vv21142 ("你好"); |

这是我们已经写好的函数：

|  |
| --- |
| function myjsonp(URL,callback,callbackname){  //给系统中创建一个全局变量，叫做callbackname，指向callback函数  //定义  window[callbackname] = callback;  //创建一个script节点  var oscript = document.createElement("script");  //和image不一样，设置src并不会发出HTTP请求  oscript.src = URL;  oscript.type = "text/javascript";  //script标签的请求是在上树的时候发出，请求的是一个函数的执行语句  document.head.appendChild(oscript);  //为了不污染页面，我们瞬间把script拿掉  document.head.removeChild(oscript);  } |