# Ajax入门和应用

第4天课堂笔记（本课程共4天）

# 目录

[Ajax入门和应用 1](#_Toc493854002)

[目录 1](#_Toc493854003)

[一、常见排序算法 2](#_Toc493854004)

[二、Ajax分页 8](#_Toc493854005)

[三、Ajax分页讲解 9](#_Toc493854006)

[二、卷动到页面底部加载更多 12](#_Toc493854007)

[三、动态绑定 16](#_Toc493854008)

[四、数据预读取 18](#_Toc493854009)

[五、懒加载 20](#_Toc493854010)

[六、浮动瀑布流 23](#_Toc493854011)

[七、后台跨域 25](#_Toc493854012)

# 一、常见排序算法

underscrore中的常见函数，尽量都要自己写一遍。

自己写count：

|  |
| --- |
| function count(array,fn){  var o = {}; // 返回的结果  for(var i = 0 ; i < array.length ; i++){  var r = fn(array[i]); //r是某一个词，就是用户return自行设置的词  //如果o中有这个属性，让这个属性值加1  if(o.hasOwnProperty(r)){  o[r]++;  }else{  //如果o中没有这个属性，让它成为属性，并且初值为1  o[r] = 1;  }  }  return o;  } |

API：

|  |
| --- |
| var arr = [{"name":"小明","age":16},{"name":"小红","age":20},{"name":"小绿","age":13},{"name":"大明","age":27}];  var result = count(arr,function(item){  if(item.age >= 18 && item.age <= 20){  return "少年";  }else if(item.age < 18){  return "孩子";  }else if(item.age > 20){  return "老年";  }  }); |



现在说说数组排序。有三种比较常见的方法：

方法1：JS内置了一个sort函数，需要你传入一个比较函数。

|  |
| --- |
| var arr = [32,45,32,2,32,4,3,65,78,4,5];  var result = arr.sort(function(a,b){  if(a > b){  return -1;  }else{  return 1;  }  });  console.log(result); |

说一下传入的函数什么意思，你的参数比如叫做a，b。假设a比b大，如果你想让a在b前面，返回-1；如果你想让a在b后面，返回1；假设a比b小，如果你想让a在b前面，返回1，如果想让a在b后面，返回-1。所以系统会让里面的数字两两比较，就得出总序列。



方法2：冒泡排序

|  |
| --- |
|  |
|  |
|  |

var arr = [35,23,47,36,12,27];

数组是6项，比较5趟，每趟比较5、4、3、2、1次，一共15次。

冒泡排序程序，6个数字，需要15字比较：

|  |
| --- |
| // 冒泡排序  var arr = [35,47,23,36,12,49];  // i表示趟数  for(var i = 0 ; i < 5 ; i++){  // j表示每趟比较的数组的下标  for(var j = 5; j > i ; j--){  if(arr[j] < arr[j - 1]){  var temp = arr[j - 1];  arr[j - 1] = arr[j];  arr[j] = temp;  }  }  console.log("第" + i + "趟");  console.log(arr);  } |



方法3：快速排序

有三个坐标，分别是i、j、key。i和j分别和key数字比较，满足某种情况，会进行交换数字。即i和j交换。

|  |
| --- |
|  |
|  |
|  |

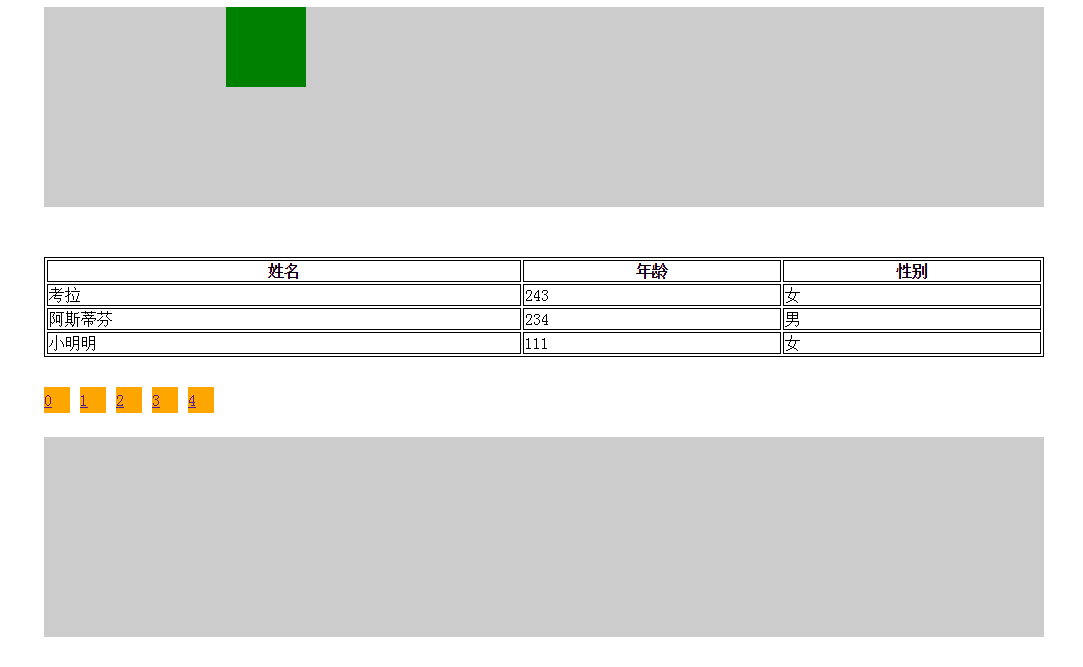
有两个标兵，分别叫做i、j。i往右边寻找，找到第一个比它大的数字，i和j交换数字；j往左边寻找，找到第一个比它小的数字，i和j交换数字。如果i和j移动到了一起。这就是一趟，每一趟，都能让标杆到达最终的位置。

|  |
| --- |
| // 快速排序  var arr = [35,47,23,36,12,49];    function quicksort(arr){  if(arr.length <= 1){  return arr;  }    // 标杆的位置  var pivot = 0;  // 从数组中删除标杆这一项  var p = arr.splice(pivot,1);  // 准备两个数组  var left = []; //放比标杆小的数字  var right = []; //放比标杆大的数字    for(var i = 0 ; i < arr.length ; i++){  if(arr[i] < p){  left.push(arr[i]);  }else{  right.push(arr[i]);  }  }    return quicksort(left).concat(p,quicksort(right));  }  var result = quicksort(arr); |

分治，数组划分为两部分，大数、小数。然后再划分……。

# 二、Ajax分页

很多年都是后台工程师来实现分页，看今天的案例03.php。注意这个案例需要你有和我一样的数据库。



后台制作分页，每个页面的头部、尾部都重新刷新了。而用户仅仅关心的是中间内容的改变！

所以Ajax分页就越来越火爆，就是让页面的URL根本不变化，等于说就是在同一个页面中，通过Ajax请求不同的JSON，实现局部更新。优点就是局部更新，缺点就是不易于分享。

后台哥哥仅仅负责生成这个页面的JSON就行了，不需要把页面做出来。现在PHP就负责生成数据就行了！不需要负责数据可视化，数据可视化，让开！让专业的来！

分页条：

|  |  |
| --- | --- |
| 较前页面 |  |
|  |
|  |
|  |
| 中间阶段 |  |
|  |
|  |
|  |
| 末尾阶段 |  |
|  |

# 三、Ajax分页讲解

Ajax分页，分页自古以来都是后台的事儿，后台做分页有很大的不方便：

1） 页面的结构和数据严重耦合，Ajax的哲学就是把结构和数据分开，而分页条更加让数据、结构耦合更严重。2） 传统分页，页面是全刷新，而不是局部刷新

3） 安全性的问题，就是页面的所有的URL都向用户暴露了，给了黑客更多的攻击机会。

传统分页也有好处：

页面地址可以根据分页的变化来变化，

|  |
| --- |
| 1. http://www.163.com/news.php?page=1; |

Ajax局部刷新，所以URL不变。不过这个问题可以用hash来轻松解决，hash就是URL最后的#。还记得页面锚点这个事儿么？

|  |
| --- |
| 1. http://www.163.com/news.php**#1**; |

JS中用非常简单的API可以读取这个hash：

|  |
| --- |
| 1. var hash = **window.location.hash**; 2. alert(hash);  **//#1** |



这个hash也可以非常方便的设置：

|  |
| --- |
| 1. window.location.hash = a; |

所以Ajax是局部刷新，既然是局部刷新，那么不会产生URL的变化，就给Ajax项目带来了不方便：分享不方便。如何解决分享不方便，就是使用hash #来表示当前读取的内容。

要学会组件开发，页面上如果有大量复现的相同结构、相同业务的东西，体现出了“自治”的感觉，想都不要想，用组件开发！！把相同的结构new出来。

分页导航条，最最经典的就是7格法，所有总页面数量超过7页的，都能用这个方法。

这个当行条做出来，用户永远能快速跳转到第1页、最后一页。也能够快速的跳转到前一页、后一页。

比如页面总数是9：

1：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | …… | 8 | 9 |

2：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | …… | 8 | 9 |

3：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | …… | 8 | 9 |

4：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | …… | 3 | 4 | 5 | …… | 9 |

5：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | …… | 4 | 5 | 6 | …… | 9 |

6：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | …… | 5 | 6 | 7 | …… | 9 |

7：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | …… | 6 | 7 | 8 | 9 |

8：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | …… | 6 | 7 | 8 | 9 |

9：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | …… | 6 | 7 | 8 | 9 |

有兴趣的同学自己可以琢磨9格法：：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | …… | 76 | 77 | 78 | …… | 164 | 165 |

页面总数1~7页：

比如，一共只有4页：

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |

# 二、卷动到页面底部加载更多

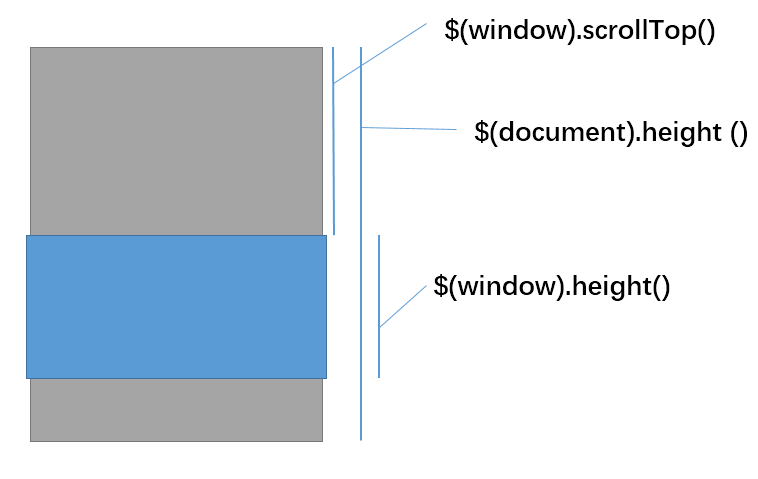
真实工作的时候，不可能请求一个静态json文件。因为所有json都是从数据库中得来的，所以你一定请求的是后台接口。

|  |
| --- |
| read.php?page=1  read.do?page=1  read/1 |

这个页面给你一个JSON，JSON是后台哥哥从数据库中读取出来的。

但是上课的时候，我们不可能做一个PHP页面，所以只能请求静态JSON文件。

如何判定滚动到底部？



用比值，0~1：

|  |
| --- |
| ($(window).scrollTop() + $(window).height()) / $(document).height(); |

一般来说，我们现在判定大于0.7，就可以加载了。

|  |
| --- |
| $(window).scroll(function(){  //比例  **var rate = ($(this).height() + $(this).scrollTop()) / $(document).height();**  if(rate > 0.7){  //加载更多  }  }); |

结束标记？结束标记是后台给的！有两种策略：

第一种，每个JSON里面，都写上全部有几页

|  |
| --- |
| {  "pageAmount" : 80,  "thisPage" : 49,  "data" : [  {},  {},  {},  {},  ]  } |

此时就要在Ajax的回调函数里面，选择性的开锁：

|  |
| --- |
| if(pageNumber < data.pageAmount){  lock = true;  } |

第二种，就是page一旦超过数值，就返回结束标记。

比如，你请求

|  |
| --- |
| read.php?page=1  read.php?page=2  read.php?page=3 |

合法页面的时候会返回JSON

|  |
| --- |
| read.php?page=7  read.php?page=8  read.php?page=9  read.php?page=10 |

会返回

|  |
| --- |
| {"end":"end"} |

此时编程的时候，就是Ajax回调函数中选择性return：

|  |
| --- |
| $.get("data/page" + pageNumber + ".json",{"z":Math.random()},function(data){  if(data.hasOwnProperty("end") && data.end == "end"){  $(".loading").html("没有更多了");  return;  }  ……  ); |

# 三、动态绑定

给刚刚创建的元素添加监听，就叫做动态事件绑定，简称动态绑定。

DOM方法来创建元素，绑定监听，没有问题。

|  |
| --- |
| //我们用DOM方法添加元素  var op = document.createElement("p");  //上树  box.appendChild(op);  //直接用op来添加监听  op.onclick = function(){  alert("你好");  } |

文本创建元素，绑定监听也没有问题。

|  |
| --- |
| // 我们用文本的方式添加  box.innerHTML = "<p></p>";  var thep = box.getElementsByTagName("p")[0];  thep.onclick = function(){  alert("你好~");  } |

|  |
| --- |
| // 批量添加监听  for(var i = 0 ; i < 10 ; i++){  var op = document.createElement("p");  //上树  box.appendChild(op);  //编号  op.index = i;  //绑定监听  op.onclick = function(){  alert("你好" + this.index);  }  } |

小题目：

|  |
| --- |
| for(var i = 0 ; i < 4 ; i++){  setTimeout(function(){  console.log(i);  },0);  } |



因为setTimeout是异步语句，也是一条需要大量系统开销的语句， for(){}不会等他的。

如果用文本的方式创建节点，创建一个绑定一个，此时就有问题：

|  |
| --- |
| for(var i = 0 ; i < 5 ; i++){  box.innerHTML += "<p></p>";  **var thep = box.getElementsByTagName("p")[i];**  thep.onclick = function(){  alert("你好~~");  }  } |

只有最后一个节点监听添加成功了。

jQuery也不能创建一个，绑定一个（文本方法）：

|  |
| --- |
| for(var i = 0 ; i < 10 ; i++){  $("#box").html($("#box").html() + "<p><span></span></p>");  $("span").eq(i).click(function(){  alert($(this).index());  });  } |

甚至给每个span添加ip，通过id绑定监听都有问题：

|  |
| --- |
| for(var i = 0 ; i < 10 ; i++){  box.innerHTML += '<p><span id="s' + i + '"></span></p>';  document.getElementById("s" + i).style.backgroundColor = "green";  document.getElementById("s" + i).onclick = function(){  alert("你好");  }  } |

# 四、数据预读取

Ajax刚刚普及的时候，大家的认识就是“悄悄的做一些事情”，优化用户体验。但是随着认识的深入，Ajax的深层哲学就是控制请求的发送和接收。

一个优秀的用户体验，不是图片慢慢刷出来，而是没有加载完毕的时候显示小菊花。要么不显示，要么一次性显示全。原理：

|  |
| --- |
| <script type="text/javascript">  var image = new Image();  image.src = "images/smallpic/0.jpg";  $(image).load(function(){  $(this).appendTo("li");  });  </script> |

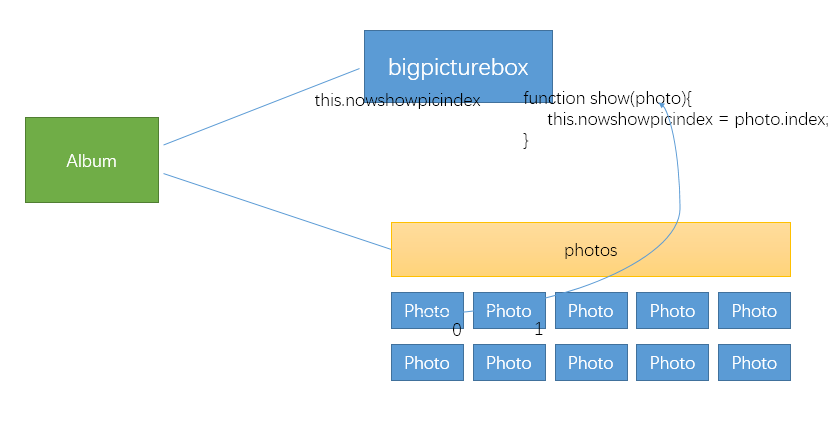
所有的图片列表都要有loading图，一次性显示。

我们一般在中介者中，都会有一个属性是一个数组，存放自己的所有其他类的实例。

比如Album类里面有一个photos属性，存放Photo类的实例。按理说就Album维持了Photos，而每个Photos不知道自己在中介者中的名次。为了让每个实例都知道自己在中介者数组中的名次，可以在new的时候，传入序号参数。

|  |
| --- |
| for(var i = 0 ; i < data.data.length ; i++){  **self.photos.push(**new Photo(data.data[i].smallurl , data.data[i].bigurl,**i**)**)**;  } |

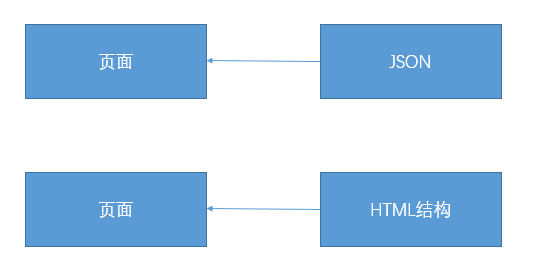
小元素知道中介者中自己的名次， 中介者也知道每个小元素的名次。双向维持序号。



当用户看1的时候预读取2，看2的时候预读取3……

# 五、懒加载

Ajax被人开发的非常充分，我们不仅仅能够请求JSON，还能够请求HTML结构。



我们的外部html文件：

|  |
| --- |
| <div>  <ul>  <li><img src="images/smallpic/0.jpg" alt=""></li>  <li><img src="images/smallpic/1.jpg" alt=""></li>  <li><img src="images/smallpic/2.jpg" alt=""></li>  <li><img src="images/smallpic/3.jpg" alt=""></li>  <li><img src="images/smallpic/4.jpg" alt=""></li>  </ul>  </div> |

我们的主页面去读取这个文件：

|  |
| --- |
| $.get("data/html1.html",function(data){  $("#box").html(data);  }); |

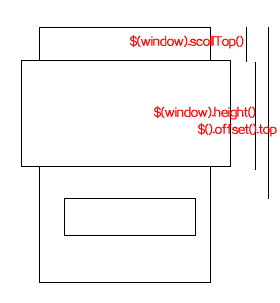
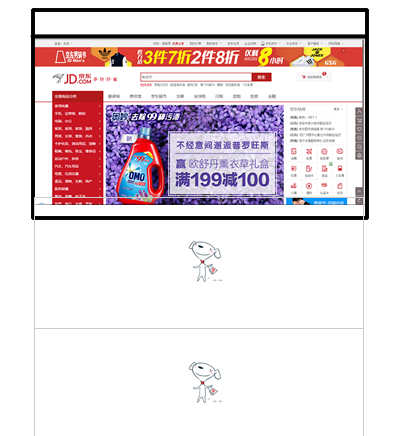
jQuery给我们提供了一个非常简单的方法：

|  |
| --- |
| $("#box").load("data/html1.html"); |

等价于

|  |
| --- |
| $.get("data/html1.html",function(data){  $("#box").html(data);  }); |

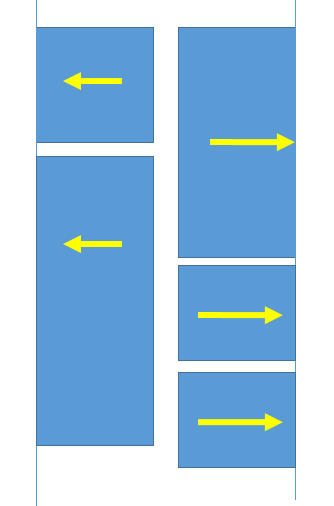
这样的技术，就被升华为“懒加载”。 **当用户的视口中还没有呈递某一部分的时候，那一部分的HTML不加载。当视口看见的时候，才加载。**



|  |
| --- |
| //把所有带有lazy-load-section类名的元素放入数组  //数组中存放的是对象，每个对象有3个属性，分别是DOM元素、这个元素的offsettop、是否已经加载完毕的标记。  var lazyloadarray = (function(){  var arr = [];  $(".lazy-load-section").each(function(){  arr.push({  "$dom" : $(this),  "offsettop" : $(this).offset().top,  "alreadyload" : false  });  });  return arr;  })();    //窗口卷动事件  $(window).scroll(function(){  //A是窗口的已经卷动的高度 ； B是窗口的高度  var A = $(this).scrollTop();  var B = $(this).height();  //遍历数组，看看此时能显示出谁，并且谁刚好already是false，防止你重复load  for(var i = 0 ; i < lazyloadarray.length ; i++){  if(lazyloadarray[i].alreadyload == false && A + B >= lazyloadarray[i].offsettop){  //jQuery新API，详见06号案例  lazyloadarray[i].$dom.load("data/" + lazyloadarray[i].$dom.attr("data-lazy-load"));  //更改标记，把alreadyload标记设置为true，防止重新加载一次  lazyloadarray[i].alreadyload = true;  break;  }  }  }); |

# 六、浮动瀑布流

两列瀑布流就能用浮动来做。策略就是每个元素要么左浮动，要么右浮动，浮动的方向取决于当前两列中谁短。往短的那边浮动。



核心算法（完整算法请见今日案例01）：

|  |
| --- |
| //左边列高，右边列高  var leftHeight = 0;  var rightHeight = 0;  //判定怎么浮动  if(leftHeight <= rightHeight){  $dom.addClass("fll");  leftHeight += $dom.height();  }else{  $dom.addClass("flr");  rightHeight += $dom.height();  } |

# 七、后台跨域

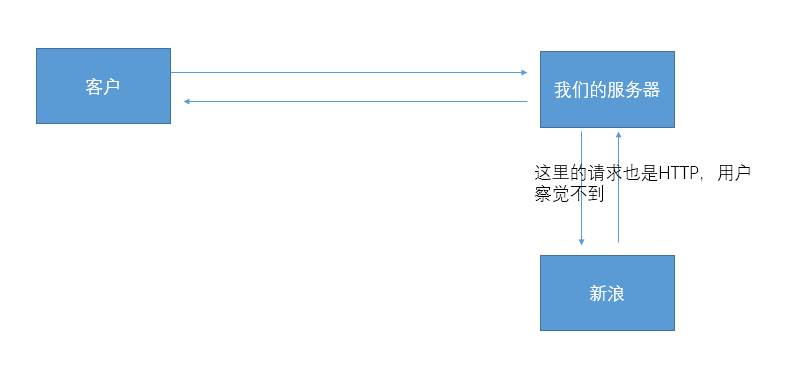
面试的时候如果问你，Ajax如何跨域，就回答①JSONP ② 后台跨域。JSONP跨域实际上不是Ajax，是奇淫技巧，用的是script请求一个函数的执行，把数据通过实参传输给我们页面中的形参。

后端跨域设置response的header头：Access-Control-Allow-Origin: \* 表示所有，但是不安全，最好是白名单

比如面试让你说说JSONP，你就大大方方的，说说你的理解就行：

JSONP不是Ajax，没有利用XMLHttpRequest对象，而是利用script标签请求一个外部的网址，那个网址的内容是一个**函数的执行**，比如fun(JSON)。所以叫做JSONP，就是JSON + padding。此时页面中要定义一个函数，就相当于数据通过实参传给形参进入页面。我们可以瞬间创建一个script标签，然后立即删除它来发出JSONP请求，jQuery中的JSONP原理就是这样。

我们现在说的是“后台跨域”，后台跨域实际上属于偷数据：



偷数据很简单，用file\_get\_contents函数即可。

|  |
| --- |
| <?php  $content = file\_get\_contents("http://www.sina.com.cn");  echo $content;  ?> |

面试的时候要尽可能的展示出自己知识的宽度和深度。