**1. 什么是ajax，为什么要使用Ajax（请谈一下你对Ajax的认识）**

什么是ajax：

AJAX是“Asynchronous JavaScript and XML”的缩写。他是指一种创建交互式网页应用的网页开发技术。

Ajax包含下列技术：

基于web标准（standards-based presentation）XHTML+CSS的表示；

使用 DOM（Document Object Model）进行动态显示及交互；

使用 XML 和 XSLT 进行数据交换及相关操作；

使用 XMLHttpRequest 进行异步数据查询、检索；

使用 JavaScript 将所有的东西绑定在一起。

为什么要用ajax：Ajax应用程序的优势在于：

1. 通过异步模式，提升了用户体验

2. 优化了[浏览器](http://baike.baidu.com/view/7718.htm)和服务器之间的传输，减少不必要的数据往返，减少了带宽占用

3. Ajax引擎在客户端运行，承担了一部分本来由服务器承担的工作，从而减少了大用户量下的服务器负载。

**2. Ajax的最大的特点是什么。**

Ajax可以实现动态不刷新（局部刷新）

就是能在不更新整个页面的前提下维护数据。这使得Web应用程序更为迅捷地回应用户动作，并避免了在网络上发送那些没有改变过的信息。

**3. 请介绍一下XMLHTTPREQUEST对象？**

Ajax的核心是JavaScript对象XmlHttpRequest。该对象在Internet Explorer 5中首次引入，它是一种支持异步请求的技术。简而言之，XmlHttpRequest使您可以使用JavaScript向服务器提出请求并处理响应，而不阻塞用户。通过XMLHttpRequest对象，Web开发人员可以在页面加载以后进行页面的局部更新。

**4. Ajax技术体系的组成部分有哪些？**

HTML，css，dom，xml，xmlHttpRequest，javascript

**5. AJAX应用和传统Web应用有什么不同？**

在传统的Javascript编程中，如果想得到服务器端数据库或文件上的信息，或者发送客户端信息到服务器，需要建立一个HTML form然后GET或者POST数据到服务器端。用户需要点击”Submit”按钮来发送或者接受数据信息，然后等待服务器响应请求，页面重新加载。

因为服务器每次都会返回一个新的页面， 所以传统的web应用有可能很慢而且用户交互不友好。

使用AJAX技术， 就可以使Javascript通过XMLHttpRequest对象直接与服务器进行交互。

通过HTTP Request， 一个web页面可以发送一个请求到web服务器并且接受web服务器返回的信息(不用重新加载页面)，展示给用户的还是通一个页面，用户感觉页面刷新，也看不到到Javascript后台进行的发送请求和接受响应。

**6. AJAX请求总共有多少种CALLBACK**

Ajax请求总共有八种Callback

onSuccess

onFailure

onUninitialized

onLoading

onLoaded

onInteractive

onComplete

onException

**7.Ajax和javascript的区别？**

javascript是一种在浏览器端执行的脚本语言，Ajax是一种创建交互式网页应用的开发技术 ，它是利用了一系列相关的技术其中就包括javascript。

Javascript是由网景公司开发的一种脚本语言，它和sun公司的java语言是没有任何关系的，它们相似的名称只是一种行销策略。

在一般的web开发中，javascript是在浏览器端执行的，我们可以用javascript控制浏览器的行为和内容。

在 Ajax应用中信息是如何在浏览器和服务器之间传递的

     通过XML数据或者字符串

**8，在浏览器端如何得到服务器端响应的XML数据**

      XMLHttpRequest对象的responseXMl属性

**9， XMLHttpRequest对象在IE和Firefox中创建方式有没有不同？**

有，IE中通过new ActiveXObject()得到，Firefox中通过new XMLHttpRequest()得到

**10，介绍一下XMLHttpRequest对象的常用方法和属性（回答的越多越好）**

            open(“method”,”URL”) 建立对服务器的调用，第一个参数是HTTP请求    方式可以为GET，POST或任何服务器所支持的您想调用的方式。  
第二个参数是请求页面的URL。

send()方法，发送具体请求

abort()方法，停止当前请求

readyState属性   请求的状态 有5个可取值 0=未初始化 ，1=正在加载

            2=以加载，3=交互中，4=完成

responseText 属性  服务器的响应，表示为一个串

reponseXML 属性 服务器的响应，表示为XML

    status    服务器的HTTP状态码，200对应ok  400对应not found

**11，Ajax的优点和缺点**

使用Ajax的最大优点，就是能在不更新整个页面的前提下维护数据。这使得Web应用程序更为迅捷地回应用户动作，并避免了在网络上发送那些没有改变过的信息。

对应用Ajax最主要的缺点就是，它可能破坏浏览器后退按钮的正常行为

因为Ajax中采用了xml技术，所以在Ajax中也可能问到XML的问题

**12，什么是XML**

    XML是扩展标记语言，能够用一系列简单的标记描述数据

**13，xml的解析方式**

    常用的用dom解析和sax解析。dom解析是一次性读取xml文件并将其构造为DOM对象供程序使用，优点是操作方便，但是比较耗内存。Sax是按事件驱动的方式解析的，占用内存少，但是编程复杂

**14，你采用的是什么框架（架包）？**

      这题是必问的，一般也是最开始就会问到。

      在java中比较流行的有 dojo, Prototype , JQuery, Dwr, extjs  等等

**15，如果熟悉某种ajax框架，他可能会问到怎样在程序中使用这种框架**

DWR框架介绍

DWR(Direct Web Remoting)是一个WEB远程调用框架.利用这个框架可以让AJAX开发变得很简单.利用DWR可以在客户端利用JavaScript直接调用服务端的Java方法并返回值给JavaScript就好像直接本地客户端调用一样(DWR根据Java类来动态生成JavaScrip代码).

DWR的实现原理是通过反射，将java翻译成javascript，然后利用回调机制，从而实现了javascript调用Java代码

**16，介绍一下Prototype的$()函数，$F()函数，$A()函数都是什么作用？**

$() 方法是在DOM中使用过于频繁的 document.getElementById() 方法的一个便利的简写，就像这个DOM方法一样，这个方法返回参数传入的id的那个元素。

$F()函数是另一个大收欢迎的“快捷键”，它能用于返回任何表单输入控件的值，比如text box,drop-down list。这个方法也能用元素id或元素本身做为参数。

$A()函数能把它接收到的单个的参数转换成一个Array对象。

**17、介绍一下XMLHttpRequest对象**

通过XMLHttpRequest对象，Web开发人员可以在页面加载以后进行页面的局部更新。

AJAX开始流行始于Google在2005年使用的”Google Suggest”。

“Google Suggest”就是使用XMLHttpRequest对象来创建动态的Web接口：

当用户开始输入google的搜索框，Javascript发送用户输入的字符到服务器，然后服务器返回一个建议列表。

XMLHttpRequest对象在IE5.0+, Safari 1.2, Mozilla 1.0/Firefox, Opera 8+ 和NetScapt7 开始被支持。

**18、AJAX应用和传统Web应用有什么不同？**

在传统的Javascript编程中，如果想得到服务器端数据库或文件上的信息，或者发送客户端信息到服务器，需要建立一个HTML form然后GET或者POST数据到服务器端。用户需要点击”Submit”按钮来发送或者接受数据信息，然后等待服务器响应请求，页面重新加载。

因为服务器每次都会返回一个新的页面， 所以传统的web应用有可能很慢而且用户交互不友好。

使用AJAX技术， 就可以使Javascript通过XMLHttpRequest对象直接与服务器进行交互。

通过HTTP Request， 一个web页面可以发送一个请求到web服务器并且接受web服务器返回的信息(不用重新加载页面)，展示给用户的还是通一个页面，用户感觉页面刷新，也看不到到Javascript后台进行的发送请求和接受响应。

**19、AJAX的全称是什么？介绍一下AJAX？**

AJAX的全称是Asynchronous JavaScript And XML.

AJAX是2005年由Google发起并流行起来的编程方法， AJAX不是一个新的编程语言，但是它是一个使用已有标准的新的编程技术。

使用AJAX可以创建更好，更快，更用户界面友好的Web应用。

AJAX技术基于Javascript和HTTP Request.

**20，介绍一下XMLHttpRequest对象的常用方法和属性？**

open(“method”,”URL”) 建立对服务器的调用，第一个参数是HTTP请求    方式可以为GET，POST或任何服务器所支持的您想调用的方式。

第二个参数是请求页面的URL。

send()方法，发送具体请求

abort()方法，停止当前请求

readyState属性   请求的状态 有5个可取值 0=未初始化 ，1=正在加载  2=以加载，3=交互中，4=完成

responseText 属性 服务器的响应，表示为一个串

reponseXML 属性 服务器的响应，表示为XML

status    服务器的HTTP状态码，200对应ok 400对应not found

**21，.Ajax主要包含了哪些技术？**

Ajax（Asynchronous JavaScript + XML）的定义

基于web标准（standards-based presentation）XHTML+CSS的表示；

使用 DOM（Document Object Model）进行动态显示及交互；

使用 XML 和 XSLT 进行数据交换及相关操作；

使用 XMLHttpRequest 进行异步数据查询、检索；

使用 JavaScript 将所有的东西绑定在一起。英文参见Ajax的提出者Jesse James Garrett的原文,原文题目(Ajax: A New Approach to Web Applications)。

类似于DHTML或LAMP，AJAX不是指一种单一的技术，而是有机地利用了一系列相关的技术。事实上，一些基于AJAX的“派生/合成”式（derivative/composite）的技术正在出现，如“AFLAX”。

AJAX的应用使用支持以上技术的web浏览器作为运行平台。这些浏览器目前包括：Mozilla、Firefox、Internet Explorer、Opera、Konqueror及Safari。但是Opera不支持XSL格式对象，也不支持XSLT。

**22、AJAX都有哪些有点和缺点？**

1、最大的一点是页面无刷新，用户的体验非常好。

2、使用异步方式与服务器通信，具有更加迅速的响应能力。

3、可以把以前一些服务器负担的工作转嫁到客户端，利用客户端闲置的能力来处理，减轻服务器和带宽的负担，节约空间和宽带租用成本。并且减轻服务器的负担，ajax的原则是“按需取数据”，可以最大程度的减少冗余请求，和响应对服务器造成的负担。

4、基于标准化的并被广泛支持的技术，不需要下载插件或者小程序。

ajax的缺点

1、ajax不支持浏览器back按钮。

2、安全问题 AJAX暴露了与服务器交互的细节。

3、对搜索引擎的支持比较弱。

4、破坏了程序的异常机制。

5、不容易调试。

原生JS

// 设置用户名框的失去焦点事件

$('input[name=username]').blur(function(){

// 获取当前用户输入的值

var username = $(this).val();

// console.log(username);

// 设置用户名的正则

var reg = /^[a-z]\w{3,11}$/i;

// 检测是否符合正则

var res = reg.test(username);

if(!res){

// 提示错误信息

$('.msg').html('<span class="error">用户名不合法</span>')

// 终止执行

return false;

}else{

// 符合格式

// 将用户名传递到服务器进行数据的验证，验证该用户名是否存在

// 创建ajax

var ajax = new XMLHttpRequest();

// 设置onreadystatechange事件，检测与服务器的请求和响应是否结束

ajax.onreadystatechange = function(){

// readyState为4的时候是请求和响应都结束了

// 网页的http状态码 200(响应成功) 404(未找到)

// console.log(ajax.readyState);

// 获取网页的http状态码

// console.log(ajax.status);

if(ajax.readyState==4 && ajax.status==200){

console.log('我正常的请求到了服务器，服务器也成功的响应给了我信息');

// 获取服务器响应过来的数据

console.log(ajax.responseText);

if(ajax.responseText=='ok'){

// 用户名可用

$('.msg').html('<span class="success">用户名可用</span>');

// 设置用户名可用的标志

var isUser = 'ok';

}else{

$('.msg').html('<span class="error">用户名已占用</span>');

}

}

}

// get方式如何传递参数

/\*

ajax在请求服务器的文件时，会将服务器文件输出的所有的字符串响应给客户端

\*/

// 打开与服务器的链接

ajax.open('get','1.php?username='+username);

// 发送ajax

ajax.send();

}

})

JQ

// 添加单击事件

$('button').click(function(){

// console.log(1111);

var id = $(this).attr('id');

// console.log(id);

// 发送ajax请求

$.ajax({

// 第一个设置传递数据的方式

type:'post',

// 请求的url地址

url:'./data.php',

// 请求的参数 ?id=10&user=1110

data:'id='+id,

// {id:id}

// 设置返回的数据格式 --

// 表示从服务器返回的就是一个json格式的对象

// 自动将从服务器返回字符串自动转为json格式的对象

dataType:'json',

// 当请求成功时

success:function(data){

// console.log(1111);

// data变量接收的就是服务器端传回来的数据

// eval('var obj='+data);

// console.log(obj);

// console.log(data);

var str = '';

// 遍历数据

// console.log($(data));

$(data).each(function(index,value){

// console.log(value);

str += '<li>'+value+'</li>';

});

// 设置html的内容

$('#content ul').html(str);

}

});

});

$.post('./data.php',params,function(data){

// console.log(data);

var str = '';

// 遍历数据

// console.log($(data));

$(data).each(function(index,value){

// console.log(value);

str += '<li>'+value+'</li>';

});

// 设置html的内容

$('#content ul').html(str);

},'json')

var params = {id:id}

$.get('./data.php',params,function(data){

// console.log(data);

var str = '';

// 遍历数据

// console.log($(data));

$(data).each(function(index,value){

// console.log(value);

str += '<li>'+value+'</li>';

});

// 设置html的内容

$('#content ul').html(str);

},'json')

<?php

// 定义包含了各种各样新闻的数组

$news = array(

'junshi' => array(

'最近特朗普当选美国总统',

'韩国的闺蜜',

'泰国的总统',

'日本的首相',

'英国的首相',

'德国的元首'

),

'yule'=> array(

'杨幂上了真正男子汉，刘恺威最近在出轨',

'王宝强上了真正男子汉,马蓉，宋喆最近在出轨',

'张翰和古力娜扎和郑爽和胡彦斌',

),

'tianqi' => array(

'昨天夜间发生地震',

'小震不用跑',

'大震跑不了',

)

);

// 接收用户传递的参数

//$id = $\_GET['id'];

// 使用post方式接收数据

$id = $\_POST['id'];

// echo $id;

// 获取军事新闻

// var\_dump($news['junshi']);

// 将数组按照json格式的字符串返回到客户端浏览器

echo json\_encode($news[$id]);

// '{"name":"zhandsan"}'

HTTP

下面的图表试图显示不同的TCP/IP和其他的协议在最初[OSI模型](http://zh.wikipedia.org/zh-cn/OSI%E6%A8%A1%E5%9E%8B)中的位置：

|  |  |  |
| --- | --- | --- |
| 7 | **应用层** | 例如[HTTP](http://zh.wikipedia.org/zh-cn/HTTP)、[SMTP](http://zh.wikipedia.org/w/index.php?title=Simple_Mail_Transfer_Protocol&action=edit&redlink=1)、[SNMP](http://zh.wikipedia.org/w/index.php?title=Simple_Network_Management_Protocol&action=edit&redlink=1)、[FTP](http://zh.wikipedia.org/w/index.php?title=File_transfer_protocol&action=edit&redlink=1)、[Telnet](http://zh.wikipedia.org/zh-cn/Telnet)、[SIP](http://zh.wikipedia.org/zh-cn/Session_Initiation_Protocol)、[SSH](http://zh.wikipedia.org/w/index.php?title=Secure_Shell&action=edit&redlink=1)、[NFS](http://zh.wikipedia.org/w/index.php?title=Network_File_System&action=edit&redlink=1)、[RTSP](http://zh.wikipedia.org/zh-cn/RTSP)、[XMPP](http://zh.wikipedia.org/zh-cn/XMPP)、[Whois](http://zh.wikipedia.org/zh-cn/Whois)、[ENRP](http://zh.wikipedia.org/w/index.php?title=Endpoint_Handlespace_Redundancy_Protocol&action=edit&redlink=1) |
| 6 | **表示层** | 例如[XDR](http://zh.wikipedia.org/w/index.php?title=External_Data_Representation&action=edit&redlink=1)、[ASN.1](http://zh.wikipedia.org/w/index.php?title=Abstract_Syntax_Notation_1&action=edit&redlink=1)、[SMB](http://zh.wikipedia.org/w/index.php?title=Server_message_block&action=edit&redlink=1)、[AFP](http://zh.wikipedia.org/w/index.php?title=Apple_Filing_Protocol&action=edit&redlink=1)、[NCP](http://zh.wikipedia.org/w/index.php?title=NetWare_Core_Protocol&action=edit&redlink=1) |
| 5 | **会话层** | 例如[ASAP](http://zh.wikipedia.org/w/index.php?title=Aggregate_Server_Access_Protocol&action=edit&redlink=1)、[TLS](http://zh.wikipedia.org/zh-cn/Transport_Layer_Security)、[SSH](http://zh.wikipedia.org/zh-cn/SSH)、ISO 8327 / CCITT X.225、[RPC](http://zh.wikipedia.org/w/index.php?title=Remote_procedure_call&action=edit&redlink=1)、[NetBIOS](http://zh.wikipedia.org/w/index.php?title=NetBIOS&action=edit&redlink=1)、[ASP](http://zh.wikipedia.org/w/index.php?title=AppleTalk&action=edit&redlink=1)、[Winsock](http://zh.wikipedia.org/w/index.php?title=Winsock&action=edit&redlink=1)、[BSD sockets](http://zh.wikipedia.org/zh-cn/Berkeley_sockets) |
| 4 | **传输层** | 例如[TCP](http://zh.wikipedia.org/zh-cn/Transmission_Control_Protocol)、[UDP](http://zh.wikipedia.org/w/index.php?title=User_Datagram_Protocol&action=edit&redlink=1)、[RTP](http://zh.wikipedia.org/w/index.php?title=Real-time_Transport_Protocol&action=edit&redlink=1)、[SCTP](http://zh.wikipedia.org/w/index.php?title=Stream_Control_Transmission_Protocol&action=edit&redlink=1)、[SPX](http://zh.wikipedia.org/w/index.php?title=Sequenced_packet_exchange&action=edit&redlink=1)、[ATP](http://zh.wikipedia.org/w/index.php?title=AppleTalk&action=edit&redlink=1)、[IL](http://zh.wikipedia.org/w/index.php?title=IL_Protocol&action=edit&redlink=1) |
| 3 | **网络层** | 例如[IP](http://zh.wikipedia.org/zh-cn/Internet_Protocol)、[ICMP](http://zh.wikipedia.org/w/index.php?title=Internet_control_message_protocol&action=edit&redlink=1)、[IGMP](http://zh.wikipedia.org/w/index.php?title=Internet_group_management_protocol&action=edit&redlink=1)、[IPX](http://zh.wikipedia.org/zh-cn/IPX)、[BGP](http://zh.wikipedia.org/zh-cn/BGP)、[OSPF](http://zh.wikipedia.org/zh-cn/OSPF)、[RIP](http://zh.wikipedia.org/zh-cn/RIP)、[IGRP](http://zh.wikipedia.org/zh-cn/IGRP)、[EIGRP](http://zh.wikipedia.org/w/index.php?title=EIGRP&action=edit&redlink=1)、[ARP](http://zh.wikipedia.org/zh-cn/Address_Resolution_Protocol)、[RARP](http://zh.wikipedia.org/zh-cn/RARP)、 [X.25](http://zh.wikipedia.org/zh-cn/X.25) |
| 2 | **数据链路层** | 例如[Ethernet](http://zh.wikipedia.org/zh-cn/Ethernet)、[Token ring](http://zh.wikipedia.org/w/index.php?title=Token_ring&action=edit&redlink=1)、[HDLC](http://zh.wikipedia.org/zh-cn/HDLC)、[Frame relay](http://zh.wikipedia.org/w/index.php?title=Frame_relay&action=edit&redlink=1)、[ISDN](http://zh.wikipedia.org/w/index.php?title=Integrated_Services_Digital_Network&action=edit&redlink=1)、[ATM](http://zh.wikipedia.org/w/index.php?title=Asynchronous_Transfer_Mode&action=edit&redlink=1)、[802.11 WiFi](http://zh.wikipedia.org/zh-cn/IEEE_802.11)、[FDDI](http://zh.wikipedia.org/w/index.php?title=Fiber_distributed_data_interface&action=edit&redlink=1)、[PPP](http://zh.wikipedia.org/zh-cn/Point-to-Point_Protocol) |
| 1 | **物理层** | 例如[wire](http://zh.wikipedia.org/w/index.php?title=Wire&action=edit&redlink=1)、[radio](http://zh.wikipedia.org/w/index.php?title=Radio&action=edit&redlink=1)、[fiber optic](http://zh.wikipedia.org/w/index.php?title=Fiber_optic&action=edit&redlink=1)、[Carrier pigeon](http://zh.wikipedia.org/w/index.php?title=IP_over_Avian_Carriers&action=edit&redlink=1) |

通常人们认为OSI模型的最上面三层（应用层、表示层和会话层）在TCP/IP组中是一个应用层。由于TCP/IP有一个相对较弱的会话层，由TCP和RTP下的打开和关闭连接组成，并且在TCP和UDP下的各种应用提供不同的端口号，这些功能能够被单个的应用程序（或者那些应用程序所使用的库）增加。与此相似的是，IP是按照将它下面的网络当作一个黑盒子的思想设计的，这样在讨论TCP/IP的时候就可以把它当作一个独立的层。

|  |  |  |
| --- | --- | --- |
| 4 | **应用层** （OSI5 到 7层） | 例如[HTTP](http://zh.wikipedia.org/zh-cn/HTTP)、[FTP](http://zh.wikipedia.org/w/index.php?title=File_transfer_protocol&action=edit&redlink=1)、[DNS](http://zh.wikipedia.org/zh-cn/DNS) （如[*BGP*](http://zh.wikipedia.org/w/index.php?title=Border_Gateway_Protocol&action=edit&redlink=1)和[*RIP*](http://zh.wikipedia.org/w/index.php?title=Routing_information_protocol&action=edit&redlink=1)这样的路由协议，尽管由于各种各样的原因它们分别运行在TCP和UDP上，仍然可以将它们看作网络层的一部分） |
| 3 | **传输层** （OSI4 和 5层） | 例如[TCP](http://zh.wikipedia.org/zh-cn/Transmission_Control_Protocol)、[UDP](http://zh.wikipedia.org/w/index.php?title=User_Datagram_Protocol&action=edit&redlink=1)、[RTP](http://zh.wikipedia.org/w/index.php?title=Real-time_Transport_Protocol&action=edit&redlink=1)、[SCTP](http://zh.wikipedia.org/w/index.php?title=Stream_Control_Transmission_Protocol&action=edit&redlink=1) （如[*OSPF*](http://zh.wikipedia.org/zh-cn/Open_shortest_path_first)这样的路由协议，尽管运行在IP上也可以看作是网络层的一部分） |
| 2 | **网络互连层** （OSI3层） | 对于TCP/IP来说这是[因特网协议](http://zh.wikipedia.org/zh-cn/Internet_Protocol)（IP） （如[*ICMP*](http://zh.wikipedia.org/w/index.php?title=Internet_control_message_protocol&action=edit&redlink=1)和[*IGMP*](http://zh.wikipedia.org/zh-cn/IGMP)这样的必须协议尽管运行在IP上，也仍然可以看作是网络互连层的一部分；[*ARP*](http://zh.wikipedia.org/zh-cn/ARP)不运行在IP上） |
| 1 | **网络接口层** （OSI1和2层） | 例如[Ethernet](http://zh.wikipedia.org/zh-cn/Ethernet)、[Wi-Fi](http://zh.wikipedia.org/zh-cn/Wi-Fi)、[MPLS](http://zh.wikipedia.org/w/index.php?title=Multiprotocol_Label_Switching&action=edit&redlink=1)等。 |

**应用层**

该层包括所有和应用程序协同工作，利用基础网络交换应用程序专用的数据的协议。 [应用层](http://zh.wikipedia.org/zh-cn/%E5%BA%94%E7%94%A8%E5%B1%82)是大多数普通与网络相关的程序为了通过网络与其他程序通信所使用的层。这个层的处理过程是应用特有的；数据从网络相关的程序以这种应用内部使用的格式进行传送，然后被编码成标准协议的格式。

一些特定的程序被认为运行在这个层上。它们提供服务直接支持用户应用。这些程序和它们对应的协议包括[HTTP](http://zh.wikipedia.org/zh-cn/HTTP)（The World Wide Web）、[FTP](http://zh.wikipedia.org/w/index.php?title=File_Transfer_Protocol&action=edit&redlink=1)（文件传输）、[SMTP](http://zh.wikipedia.org/w/index.php?title=Simple_Mail_Transfer_Protocol&action=edit&redlink=1)（电子邮件）、[SSH](http://zh.wikipedia.org/w/index.php?title=Secure_shell&action=edit&redlink=1)（安全远程登陆）、[DNS](http://zh.wikipedia.org/zh-cn/DNS)（名称<-> IP 地址寻找）以及许多其他协议。

一旦从应用程序来的数据被编码成一个标准的应用层协议，它将被传送到IP栈的下一层。

在传输层，应用程序最常用的是TCP或者UDP，并且服务器应用程序经常与一个[公开的端口号](http://zh.wikipedia.org/w/index.php?title=TCP_and_UDP_port_numbers&action=edit&redlink=1)相联系。服务器应用程序的端口由[Internet Assigned Numbers Authority](http://zh.wikipedia.org/w/index.php?title=Internet_Assigned_Numbers_Authority&action=edit&redlink=1)（IANA）正式地分配，但是现今一些新协议的开发者经常选择它们自己的端口号。由于在同一个系统上很少超过少数几个的服务器应用，端口冲突引起的问题很少。应用软件通常也允许用户强制性地指定端口号作为[运行](http://zh.wikipedia.org/w/index.php?title=%E8%BF%90%E8%A1%8C&action=edit&redlink=1)[参数](http://zh.wikipedia.org/w/index.php?title=%E5%8F%82%E6%95%B0&action=edit&redlink=1)。

连结外部的客户端程序通常使用系统分配的一个随机端口号。监听一个端口并且然后通过服务器将那个端口发送到应用的另外一个副本以建立对等连结（如[IRC](http://zh.wikipedia.org/w/index.php?title=Internet_Relay_Chat&action=edit&redlink=1)上的[dcc](http://zh.wikipedia.org/w/index.php?title=Direct_Client-to-Client&action=edit&redlink=1)文件传输）的应用也可以使用一个随机端口，但是应用程序通常允许定义一个特定的端口范围的规范以允许端口能够通过实现[网络地址转换](http://zh.wikipedia.org/zh-cn/%E7%BD%91%E7%BB%9C%E5%9C%B0%E5%9D%80%E8%BD%AC%E6%8D%A2)（NAT）的路由器映射到内部。

每一个应用层（[TCP/IP参考模型](http://zh.wikipedia.org/zh-cn/TCP/IP%E5%8F%82%E8%80%83%E6%A8%A1%E5%9E%8B) 的最高层）协议一般都会使用到两个传输层协议之一： 面向连接的[TCP](http://zh.wikipedia.org/zh-cn/TCP)[传输控制协议](http://zh.wikipedia.org/zh-cn/%E4%BC%A0%E8%BE%93%E6%8E%A7%E5%88%B6%E5%8D%8F%E8%AE%AE)和无连接的包传输的[UDP](http://zh.wikipedia.org/zh-cn/UDP)[用户数据报文协议](http://zh.wikipedia.org/zh-cn/%E7%94%A8%E6%88%B7%E6%95%B0%E6%8D%AE%E6%8A%A5%E6%96%87%E5%8D%8F%E8%AE%AE) 。

常用的应用层协议有：

运行在[TCP](http://zh.wikipedia.org/zh-cn/TCP)协议上的协议：

* [HTTP](http://zh.wikipedia.org/zh-cn/HTTP)（Hypertext Transfer Protocol，超文本传输协议），主要用于普通浏览。
* [HTTPS](http://zh.wikipedia.org/zh-cn/HTTPS)（Hypertext Transfer Protocol over Secure Socket Layer, or HTTP over SSL，安全超文本传输协议）,HTTP协议的安全版本。
* [FTP](http://zh.wikipedia.org/zh-cn/FTP)（File Transfer Protocol，文件传输协议），由名知义，用于文件传输。
* [POP3](http://zh.wikipedia.org/zh-cn/POP3)（Post Office Protocol, version 3，邮局协议），收邮件用。
* [SMTP](http://zh.wikipedia.org/zh-cn/SMTP)（Simple Mail Transfer Protocol，简单邮件传输协议），用来发送电子邮件 。
* [TELNET](http://zh.wikipedia.org/zh-cn/TELNET)（Teletype over the Network，网络电传），通过一个终端（terminal）登陆到网络。
* [SSH](http://zh.wikipedia.org/zh-cn/SSH)（Secure Shell，用于替代安全性差的[TELNET](http://zh.wikipedia.org/zh-cn/TELNET)），用于加密安全登陆用。

运行在[UDP](http://zh.wikipedia.org/zh-cn/UDP)协议上的协议：

* [BOOTP](http://zh.wikipedia.org/zh-cn/BOOTP)（Boot Protocol，启动协议），应用于无盘设备。
* [NTP](http://zh.wikipedia.org/zh-cn/NTP)（Network Time Protocol，网络时间协议），用于网络同步。

其他：

* [DNS](http://zh.wikipedia.org/zh-cn/DNS)（Domain Name Service，域名服务），用于完成地址查找，邮件转发等工作（运行在[TCP](http://zh.wikipedia.org/zh-cn/TCP)和[UDP](http://zh.wikipedia.org/zh-cn/UDP)协议上）。
* [ECHO](http://zh.wikipedia.org/w/index.php?title=ECHO&action=edit&redlink=1)（Echo Protocol，回绕协议），用于查错及测量应答时间（运行在[TCP](http://zh.wikipedia.org/zh-cn/TCP)和[UDP](http://zh.wikipedia.org/zh-cn/UDP)协议上）。
* [SNMP](http://zh.wikipedia.org/zh-cn/SNMP)（Simple Network Management Protocol，简单网络管理协议），用于网络信息的收集和网络管理。
* [DHCP](http://zh.wikipedia.org/zh-cn/DHCP)（Dynamic Host Configuration Protocol，动态主机配置协议），动态配置IP地址。
* [ARP](http://zh.wikipedia.org/zh-cn/ARP)（Address Resolution Protocol，地址解析协议），用于动态解析以太网硬件的地址。

#### 传输层

[传输层](http://zh.wikipedia.org/zh-cn/%E4%BC%A0%E8%BE%93%E5%B1%82)的协议，能够解决诸如可靠性（“数据是否已经到达目的地？”）和保证数据按照正确的顺序到达这样的问题。在TCP/IP协议组中，传输协议也包括所给数据应该送给哪个应用程序。

在TCP/IP协议组中技术上位于这个层的动态路由协议通常被认为是网络层的一部分；一个例子就是[OSPF](http://zh.wikipedia.org/zh-cn/OSPF)（IP协议89）。

[TCP](http://zh.wikipedia.org/zh-cn/Transmission_control_protocol)（IP协议6）是一个“可靠的”、[面向连结的](http://zh.wikipedia.org/w/index.php?title=%E9%9D%A2%E5%90%91%E8%BF%9E%E7%BB%93%E7%9A%84&action=edit&redlink=1)传输机制，它提供一种[可靠的字节流](http://zh.wikipedia.org/w/index.php?title=%E5%8F%AF%E9%9D%A0%E7%9A%84%E5%AD%97%E8%8A%82%E6%B5%81&action=edit&redlink=1)保证数据完整、无损并且按顺序到达。TCP尽量连续不断地测试网络的负载并且控制发送数据的速度以避免网络过载。另外，TCP试图将数据按照规定的顺序发送。这是它与UDP不同之处，这在实时数据流或者路由高[网络层](http://zh.wikipedia.org/zh-cn/%E7%BD%91%E7%BB%9C%E5%B1%82)丢失率应用的时候可能成为一个缺陷。

较新的[SCTP](http://zh.wikipedia.org/w/index.php?title=Stream_Control_Transmission_Protocol&action=edit&redlink=1)也是一个“可靠的”、[面向连结的](http://zh.wikipedia.org/w/index.php?title=%E9%9D%A2%E5%90%91%E8%BF%9E%E7%BB%93%E7%9A%84&action=edit&redlink=1)传输机制。它是面向纪录而不是面向字节的，它在一个单独的连结上提供了通过多路复用提供的多个子流。它也提供了多路自寻址支持，其中连结终端能够被多个IP地址表示（代表多个物理接口），这样的话即使其中一个连接失败了也不中断。它最初是为电话应用开发的（在[IP](http://zh.wikipedia.org/zh-cn/Internet_Protocol)上传输[SS7](http://zh.wikipedia.org/zh-cn/SS7)），但是也可以用于其他的应用。

[UDP](http://zh.wikipedia.org/w/index.php?title=User_datagram_protocol&action=edit&redlink=1)（IP协议号17）是一个[无连结的](http://zh.wikipedia.org/w/index.php?title=%E6%97%A0%E8%BF%9E%E7%BB%93%E7%9A%84&action=edit&redlink=1)数据报协议。它是一个“best effort”或者“不可靠”协议——不是因为它特别不可靠，而是因为它不检查数据包是否已经到达目的地，并且不保证它们按顺序到达。如果一个应用程序需要这些特点，它必须自己提供或者使用[TCP](http://zh.wikipedia.org/zh-cn/Transmission_Control_Protocol)。

UDP的典型性应用是如流媒体（音频和视频等）这样按时到达比可靠性更重要的应用，或者如[DNS](http://zh.wikipedia.org/zh-cn/DNS)查找这样的简单查询／响应应用，如果建立可靠的连结所作的额外工作将是不成比例地大。

[DCCP](http://zh.wikipedia.org/zh-cn/DCCP)目前正由IEFT开发。它提供TCP流动控制语义，但对于用户来说保留了UDP的数据报服务模型。

TCP和UDP都用来支持一些高层的应用。任何给定网络地址的应用通过它们的TCP或者UDP[*端口号*](http://zh.wikipedia.org/w/index.php?title=%E7%AB%AF%E5%8F%A3%E5%8F%B7&action=edit&redlink=1)区分。根据惯例使一些大众所知的端口与特定的应用相联系。

[RTP](http://zh.wikipedia.org/w/index.php?title=Real-time_Transport_Protocol&action=edit&redlink=1)是为如音频和视频流这样的实时数据设计的数据报协议。RTP是使用UDP包格式作为基础的会话层，然而据说它位于因特网协议栈的传输层。

#### 网络互连层

正如最初所定义的，[网络层](http://zh.wikipedia.org/zh-cn/%E7%BD%91%E7%BB%9C%E5%B1%82)解决在一个单一网络上传输数据包的问题。类似的协议有[X.25](http://zh.wikipedia.org/zh-cn/X.25)和[ARPANET](http://zh.wikipedia.org/zh-cn/ARPANET)的[Host/IMP Protocol](http://zh.wikipedia.org/w/index.php?title=Host/IMP_Protocol&action=edit&redlink=1)。

随着[因特网](http://zh.wikipedia.org/zh-cn/Internet)思想的出现，在这个层上添加了附加的功能，也就是将数据从源[网络](http://zh.wikipedia.org/w/index.php?title=Computer_network&action=edit&redlink=1)传输到目的网络。这就牵涉到在网络组成的网上选择路径将数据包传输，也就是[因特网](http://zh.wikipedia.org/zh-cn/%E5%9B%A0%E7%89%B9%E7%BD%91)。

在因特网协议组中，[IP](http://zh.wikipedia.org/zh-cn/Internet_Protocol)完成数据从源发送到目的基本任务。IP能够承载多种不同的高层协议的数据；这些协议使用一个唯一的IP协议号进行标识。ICMP和IGMP分别是1和2。

一些IP承载的协议，如[ICMP](http://zh.wikipedia.org/w/index.php?title=Internet_Control_Message_Protocol&action=edit&redlink=1)（用来发送关于IP发送的诊断信息）和[IGMP](http://zh.wikipedia.org/w/index.php?title=Internet_Group_Management_Protocol&action=edit&redlink=1)（用来管理[多播](http://zh.wikipedia.org/zh-cn/%E5%A4%9A%E6%92%AD)数据），它们位于IP层之上但是完成网络层的功能，这表明了因特网和OSI模型之间的不兼容性。所有的路由协议，如[BGP](http://zh.wikipedia.org/w/index.php?title=Border_Gateway_Protocol&action=edit&redlink=1)、 [OSPF](http://zh.wikipedia.org/zh-cn/OSPF)、和[RIP](http://zh.wikipedia.org/w/index.php?title=Routing_information_protocol&action=edit&redlink=1)实际上也是网络层的一部分，尽管似乎它们应该属于更高的协议栈。

#### 网络接口层

网络接口层实际上并不是因特网协议组中的一部分，但是它是数据包从一个设备的网络层传输到另外一个设备的网络层的方法。这个过程能够在[网卡](http://zh.wikipedia.org/zh-cn/%E7%BD%91%E5%8D%A1)的[软件](http://zh.wikipedia.org/zh-cn/%E8%BD%AF%E4%BB%B6)[驱动程序](http://zh.wikipedia.org/zh-cn/%E9%A9%B1%E5%8A%A8%E7%A8%8B%E5%BA%8F)中控制，也可以在[韧体](http://zh.wikipedia.org/zh-cn/%E9%9F%A7%E4%BD%93)或者专用[芯片](http://zh.wikipedia.org/zh-cn/%E8%8A%AF%E7%89%87)中控制。这将完成如添加[报头](http://zh.wikipedia.org/w/index.php?title=%E6%8A%A5%E5%A4%B4&action=edit&redlink=1)准备发送、通过[物理](http://zh.wikipedia.org/zh-cn/Physical_layer)[媒介](http://zh.wikipedia.org/w/index.php?title=Transmission_medium&action=edit&redlink=1)实际发送这样一些[数据链路](http://zh.wikipedia.org/zh-cn/Data_link_layer)功能。另一端，链路层将完成数据帧接收、去除报头并且将接收到的包传到网络层。

然而，链路层并不经常这样简单。它也可能是一个[虚拟专有网络](http://zh.wikipedia.org/zh-cn/%E8%99%9A%E6%8B%9F%E4%B8%93%E6%9C%89%E7%BD%91%E7%BB%9C)（VPN）或者隧道，在这里从网络层来的包使用[隧道协议](http://zh.wikipedia.org/w/index.php?title=%E9%9A%A7%E9%81%93%E5%8D%8F%E8%AE%AE&action=edit&redlink=1)和其他（或者同样的）协议组发送而不是发送到物理的接口上。VPN和隧道通常预先建好，并且它们有一些直接发送到物理接口所没有的特殊特点（例如，它可以加密经过它的数据）。由于现在链路“层”是一个完整的网络，这种协议组的[递归](http://zh.wikipedia.org/zh-cn/Recursion)使用可能引起混淆。但是它是一个实现常见复杂功能的一个优秀方法。（尽管需要注意预防一个已经封装并且经隧道发送下去的数据包进行再次地封装和发送）。

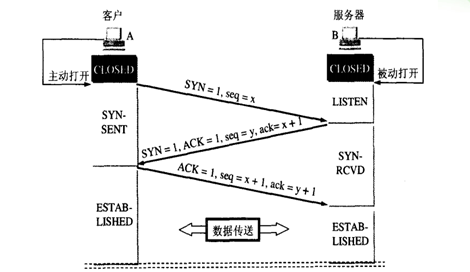
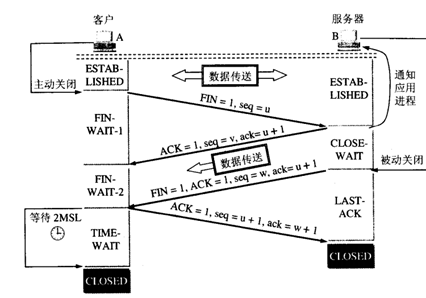
[**TCP三次握手四次挥手详解**](http://www.cnblogs.com/xuhj001/p/3409104.html)

1、建立连接协议（三次握手）

（1）客户端发送一个带SYN标志的TCP报文到服务器。这是三次握手过程中的报文1。  
（2） 服务器端回应客户端的，这是三次握手中的第2个报文，这个报文同时带ACK标志和SYN标志。因此它表示对刚才客户端SYN报文的回应；同时又标志SYN给客户端，询问客户端是否准备好进行数据通讯。  
（3） 客户必须再次回应服务段一个ACK报文，这是报文段3。  
2、连接终止协议（四次挥手）  
　 　由于TCP连接是全双工的，因此每个方向都必须单独进行关闭。这原则是当一方完成它的数据发送任务后就能发送一个FIN来终止这个方向的连接。收到一个 FIN只意味着这一方向上没有数据流动，一个TCP连接在收到一个FIN后仍能发送数据。首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。  
　（1） TCP客户端发送一个FIN，用来关闭客户到服务器的数据传送（报文段4）。  
　（2） 服务器收到这个FIN，它发回一个ACK，确认序号为收到的序号加1（报文段5）。和SYN一样，一个FIN将占用一个序号。  
　（3） 服务器关闭客户端的连接，发送一个FIN给客户端（报文段6）。  
　（4） 客户段发回ACK报文确认，并将确认序号设置为收到序号加1（报文段7）。  
CLOSED: 这个没什么好说的了，表示初始状态。  
LISTEN: 这个也是非常容易理解的一个状态，表示服务器端的某个SOCKET处于监听状态，可以接受连接了。  
SYN\_RCVD: 这个状态表示接受到了SYN报文，在正常情况下，这个状态是服务器端的SOCKET在建立TCP连接时的三次握手会话过程中的一个中间状态，很短暂，基本上用netstat你是很难看到这种状态的，除非你特意写了一个客户端测试程序，故意将三次TCP握手过程中最后一个ACK报文不予发送。因此这种状态时，当收到客户端的ACK报文后，它会进入到ESTABLISHED状态。  
SYN\_SENT: 这个状态与SYN\_RCVD遥想呼应，当客户端SOCKET执行CONNECT连接时，它首先发送SYN报文，因此也随即它会进入到了SYN\_SENT状态，并等待服务端的发送三次握手中的第2个报文。SYN\_SENT状态表示客户端已发送SYN报文。  
ESTABLISHED：这个容易理解了，表示连接已经建立了。  
FIN\_WAIT\_1: 这个状态要好好解释一下，其实FIN\_WAIT\_1和FIN\_WAIT\_2状态的真正含义都是表示等待对方的FIN报文。而这两种状态的区别是：FIN\_WAIT\_1状态实际上是当SOCKET在ESTABLISHED状态时，它想主动关闭连接，向对方发送了FIN报文，此时该SOCKET即进入到FIN\_WAIT\_1状态。而当对方回应ACK报文后，则进入到FIN\_WAIT\_2状态，当然在实际的正常情况下，无论对方何种情况下，都应该马上回应ACK报文，所以FIN\_WAIT\_1状态一般是比较难见到的，而FIN\_WAIT\_2状态还有时常常可以用netstat看到。  
FIN\_WAIT\_2：上面已经详细解释了这种状态，实际上FIN\_WAIT\_2状态下的SOCKET，表示半连接，也即有一方要求close连接，但另外还告诉对方，我暂时还有点数据需要传送给你，稍后再关闭连接。  
TIME\_WAIT: 表示收到了对方的FIN报文，并发送出了ACK报文，就等2MSL后即可回到CLOSED可用状态了。如果FIN\_WAIT\_1状态下，收到了对方同时带FIN标志和ACK标志的报文时，可以直接进入到TIME\_WAIT状态，而无须经过FIN\_WAIT\_2状态。  
CLOSING: 这种状态比较特殊，实际情况中应该是很少见，属于一种比较罕见的例外状态。正常情况下，当你发送FIN报文后，按理来说是应该先收到（或同时收到）对方的ACK报文，再收到对方的FIN报文。但是CLOSING状态表示你发送FIN报文后，并没有收到对方的ACK报文，反而却也收到了对方的FIN报文。什么情况下会出现此种情况呢？其实细想一下，也不难得出结论：那就是如果双方几乎在同时close一个SOCKET的话，那么就出现了双方同时发送FIN报文的情况，也即会出现CLOSING状态，表示双方都正在关闭SOCKET连接。  
CLOSE\_WAIT: 这种状态的含义其实是表示在等待关闭。怎么理解呢？当对方close一个SOCKET后发送FIN报文给自己，你系统毫无疑问地会回应一个ACK报文给对方，此时则进入到CLOSE\_WAIT状态。接下来呢，实际上你真正需要考虑的事情是察看你是否还有数据发送给对方，如果没有的话，那么你也就可以close这个SOCKET，发送FIN报文给对方，也即关闭连接。所以你在CLOSE\_WAIT状态下，需要完成的事情是等待你去关闭连接。  
LAST\_ACK: 这个状态还是比较容易好理解的，它是被动关闭一方在发送FIN报文后，最后等待对方的ACK报文。当收到ACK报文后，也即可以进入到CLOSED可用状态了。  
最后有2个问题的回答，我自己分析后的结论（不一定保证100%正确）  
1、 为什么建立连接协议是三次握手，而关闭连接却是四次握手呢？  
这是因为服务端的LISTEN状态下的SOCKET当收到SYN报文的建连请求后，它可以把ACK和SYN（ACK起应答作用，而SYN起同步作用）放在一个报文里来发送。但关闭连接时，当收到对方的FIN报文通知时，它仅仅表示对方没有数据发送给你了；但未必你所有的数据都全部发送给对方了，所以你可以未必会马上会关闭SOCKET,也即你可能还需要发送一些数据给对方之后，再发送FIN报文给对方来表示你同意现在可以关闭连接了，所以它这里的ACK报文和FIN报文多数情况下都是分开发送的。  
2、 为什么TIME\_WAIT状态还需要等2MSL后才能返回到CLOSED状态？  
这是因为：虽然双方都同意关闭连接了，而且握手的4个报文也都协调和发送完毕，按理可以直接回到CLOSED状态（就好比从SYN\_SEND状态到ESTABLISH状态那样）；但是因为我们必须要假想网络是不可靠的，你无法保证你最后发送的ACK报文会一定被对方收到，因此对方处于LAST\_ACK状态下的SOCKET可能会因为超时未收到ACK报文，而重发FIN报文，所以这个TIME\_WAIT状态的作用就是用来重发可能丢失的ACK报文。

新：

1. TCP的三次握手最主要是防止已过期的连接再次传到被连接的主机。(废弃连接问题）  
    如果采用两次的话，会出现下面这种情况。比如是A机要连到B机，结果发送的连接信息由于某种原因没有到达B机；  
于是，A机又发了一次，结果这次B收到了，于是就发信息回来，两机就连接。传完东西后，断开。结果这时候，原先没有到达的连接信息突然又传到了B机，于是B机发信息给A，然后B机就以为和A连上了，这个时候B机就在等待A传东西过去。  
  
2. 三次握手改成仅需要两次握手，死锁是可能发生  
    考虑计算机A和B之间的通信，假定B给A发送一个连接请求分组，A收到了这个分组，并发送了确认应答分组。按照两次握手的协定，A认为连接已经成功地建立了，可以开始发送数据分组。可是，B在A的应答分组在传输中被丢失的情况下，将不知道A是否已准备好，不知道A建议什么样的序列号，B甚至怀疑A是否收到自己的连接请求分组。在这种情况下，B认为连接还未建立成功，将忽略A发来的任何数据分组，只等待连接确认应答分组。而A在发出的分组超时后，重复发送同样的分组。这样就形成了死锁  
  
3. TCP通讯中，select到读事件，但是读到的数据量是0，为什么，如何解决????  
    select 返回0代表超时。select出错返回-1。  
    select到读事件，但是读到的数据量为0，说明对方已经关闭了socket的读端。本端关闭读即可。  
    当select出错时，会将接口置为可读又可写。这时就要通过判断select的返回值为-1来区分。  
  
4. 2MSL(maximum segment lifetime)   
    (1).等待一段时间，防止最后的FIN的ACK包丢失，对方未收到ACK会重发FIN  
    (2).TCP连接在2MSL时间内 ip,port不能重新被bind  
  
5.复位报文(RST)  
(1).接收到不存在端口的连接请求，回复RST包(但是udp是响应ICMP端口不可达的error)  
(2).异常终止一个连接，发送RST包，收到RST的一方终止该连接。  
(3).收到一个半开连接的数据包后，回复RST,收到RST的一方终止该连接。  
  
6.几种情况  
(1).服务器未开启服务，回复RST  
(2).服务器连接正常关闭，回复FIN  
(3).服务器进程异常终止，回复RST。  
(4).服务器直接掉电，如果客户端没有"发送数据"或者"设置keepalive选项"，客户端将一直保持此半开连接。  
    如果客户端重新连接，将新建立一个连接。  
(5).服务器重启时，如果收到一个半开连接的数据包，回复RST.

* 常见面试题
  + TCP协议和UDP协议的区别是什么
    - TCP协议是有连接的，有连接的意思是开始传输实际数据之前TCP的客户端和服务器端必须通过三次握手建立连接，会话结束之后也要结束连接。而UDP是无连接的
    - TCP协议保证数据按序发送，按序到达，提供超时重传来保证可靠性，但是UDP不保证按序到达，甚至不保证到达，只是努力交付，即便是按序发送的序列，也不保证按序送到。
    - TCP协议所需资源多，TCP首部需20个字节（不算可选项），UDP首部字段只需8个字节。
    - TCP有流量控制和拥塞控制，UDP没有，网络拥堵不会影响发送端的发送速率
    - TCP是一对一的连接，而UDP则可以支持一对一，多对多，一对多的通信。
    - TCP面向的是字节流的服务，UDP面向的是报文的服务。
    - [TCP介绍](http://www.cnblogs.com/newwy/archive/2013/08/02/3232503.html)和UDP介绍
  + 请详细介绍一下TCP协议建立连接和终止连接的过程？
    - 助于理解的[一段话](http://www.cppblog.com/MemoryGarden/archive/2011/07/02/150007.html)：
    - 建立一个TCP连接时，会发生下述情形：
    - 1、服务器端必须做好准备接受外来的连接。这通常通过 socket(), bind(), listen() 三个函数来完成的。我们称之为 被动打开(passive open).  
        
        
      2、客户端通过调用connect发起主动打开(active open)。这导致客户端TCP发送SYN同步分节。它告诉服务器客户端在(待建立的)连接中发送的数据的初始化序列号。通用SYN分节不携带数据，  
      3、服务器必须确认(ACK) 客户端的SYN，同时自己也得发送一个SYN分节，它含有服务器将在统一连接中发送的数据的初始化序号。服务器在单个分节中发送SYN和对客户端SYN的ACK确认。  
      4、客户端必须确认服务器的SYN。  
        
      TCP连接的终止：  
      1、某个应用程序首先调用close，主动关闭(active close) 该端的TCP于是发送一个FIN分节，表示数据发送完毕。  
      2、接收到这个FIN的对端执行被动关闭(passive close)。这个FIN是TCP确认。它的接收也作为一个文件结束符(end of file) 传递给接收端的应用程序(放在排队等候应用进程接收的任何其他数据之后)，因为FIN的接收意味着接收端应用程序在相应连接上再无额外数据可以接收。  
      3、一段时间以后，接收到这个文件结束符的应用进程将调用close关闭它的套接字。这导致它的TCP也发送一个FIN。  
      4、接收这个最终FINd额原发送端TCP(即执行主动关闭的一端)确认这个FIN
    - 两幅图（[来源](http://www.cnblogs.com/newwy/archive/2013/08/03/3234536.html)）：
      * 建立连接：三次握手
      * [](http://images.cnitblog.com/blog/385532/201308/30193701-72f027c0ea0f4c69834076466fa5383c.png)
      * 关闭连接：四次挥手
      * [](http://images.cnitblog.com/blog/385532/201308/30193703-63640062b79a4fc8b8ed31e95fd87bd8.png)
  + 三次握手建立连接时，发送方再次发送确认的必要性？
    - 主要是为了防止已失效的连接请求报文段突然又传到了B,因而产生错误。假定出现一种异常情况，即A发出的第一个连接请求报文段并没有丢失，而是在某些网络结点长时间滞留了，一直延迟到连接释放以后的某个时间才到达B，本来这是一个早已失效的报文段。但B收到此失效的连接请求报文段后，就误认为是A又发出一次新的连接请求，于是就向A发出确认报文段，同意建立连接。假定不采用三次握手，那么只要B发出确认，新的连接就建立了，这样一直等待A发来数据，B的许多资源就这样白白浪费了。
  + 四次挥手释放连接时，等待2MSL的意义？
    - 第一，为了保证A发送的最有一个ACK报文段能够到达B。这个ACK报文段有可能丢失，因而使处在LAST-ACK状态的B收不到对已发送的FIN和ACK报文段的确认。B会超时重传这个FIN和ACK报文段，而A就能在2MSL时间内收到这个重传的ACK+FIN报文段。接着A重传一次确认。
    - 第二，就是防止上面提到的已失效的连接请求报文段出现在本连接中，A在发送完最有一个ACK报文段后，再经过2MSL，就可以使本连接持续的时间内所产生的所有报文段都从网络中消失。

* + 常见的应用中有哪些是应用TCP协议的，哪些又是应用UDP协议的，为什么它们被如此设计？
    - 以下应用一般或必须用udp实现？
      * 多播的信息一定要用udp实现，因为tcp只支持一对一通信。
      * 如果一个应用场景中大多是简短的信息，适合用udp实现，因为udp是基于报文段的，它直接对上层应用的数据封装成报文段，然后丢在网络中，如果信息量太大，会在链路层中被分片，影响传输效率。
      * 如果一个应用场景重性能甚于重完整性和安全性，那么适合于udp，比如多媒体应用，缺一两帧不影响用户体验，但是需要流媒体到达的速度快，因此比较适合用udp
      * 如果要求快速响应，那么udp听起来比较合适
      * 如果又要利用udp的快速响应优点，又想可靠传输，那么只能考上层应用自己制定规则了。
      * 常见的使用udp的例子：ICQ,QQ的聊天模块。
    - 以qq为例的一个说明（转载自[知乎](http://www.zhihu.com/question/20292749)）

登陆采用TCP协议和HTTP协议，你和好友之间发送消息，主要采用UDP协议，内网传文件采用了P2P技术。总来的说：   
1.登陆过程，客户端client 采用TCP协议向服务器server发送信息，HTTP协议下载信息。登陆之后，会有一个TCP连接来保持在线状态。   
2.和好友发消息，客户端client采用UDP协议，但是需要通过服务器转发。腾讯为了确保传输消息的可靠，采用上层协议来保证可靠传输。如果消息发送失败，客户端会提示消息发送失败，并可重新发送。   
3.如果是在内网里面的两个客户端传文件，QQ采用的是P2P技术，不需要服务器中转。

# HTTP请求方法

根据HTTP标准，HTTP请求可以使用多种请求方法。  
HTTP1.0定义了三种请求方法： GET, POST 和 HEAD方法。  
HTTP1.1新增了五种请求方法：OPTIONS, PUT, DELETE, TRACE 和 CONNECT 方法。

GET 请求指定的页面信息，并返回实体主体。

HEAD 类似于get请求，只不过返回的响应中没有具体的内容，用于获取报头

POST 向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改。

PUT 从客户端向服务器传送的数据取代指定的文档的内容。

DELETE 请求服务器删除指定的页面。

CONNECT HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。

OPTIONS 允许客户端查看服务器的性能。

TRACE 回显服务器收到的请求，主要用于测试或诊断。

# HTTP工作原理

HTTP协议定义Web客户端如何从Web服务器请求Web页面，以及服务器如何把Web页面传送给客户端。HTTP协议采用了请求/响应模型。客户端向服务器发送一个请求报文，请求报文包含请求的方法、URL、协议版本、请求头部和请求数据。服务器以一个状态行作为响应，响应的内容包括协议的版本、成功或者错误代码、服务器信息、响应头部和响应数据。

以下是 HTTP 请求/响应的步骤：

###### 1、客户端连接到Web服务器

一个HTTP客户端，通常是浏览器，与Web服务器的HTTP端口（默认为80）建立一个TCP套接字连接。例如，[http://www.oakcms.cn。](http://www.oakcms.cn./)

###### 2、发送HTTP请求

通过TCP套接字，客户端向Web服务器发送一个文本的请求报文，一个请求报文由请求行、请求头部、空行和请求数据4部分组成。

###### 3、服务器接受请求并返回HTTP响应

Web服务器解析请求，定位请求资源。服务器将资源复本写到TCP套接字，由客户端读取。一个响应由状态行、响应头部、空行和响应数据4部分组成。

###### 4、释放连接[TCP连接](http://www.jianshu.com/p/ef892323e68f)

若connection 模式为close，则服务器主动关闭[TCP连接](http://www.jianshu.com/p/ef892323e68f)，客户端被动关闭连接，释放[TCP连接](http://www.jianshu.com/p/ef892323e68f);若connection 模式为keepalive，则该连接会保持一段时间，在该时间内可以继续接收请求;

###### 5、客户端浏览器解析HTML内容

客户端浏览器首先解析状态行，查看表明请求是否成功的状态代码。然后解析每一个响应头，响应头告知以下为若干字节的HTML文档和文档的字符集。客户端浏览器读取响应数据HTML，根据HTML的语法对其进行格式化，并在浏览器窗口中显示。

例如：在浏览器地址栏键入URL，按下回车之后会经历以下流程：

1、浏览器向 DNS 服务器请求解析该 URL 中的域名所对应的 IP 地址;

2、解析出 IP 地址后，根据该 IP 地址和默认端口 80，和服务器建立[TCP连接](http://www.jianshu.com/p/ef892323e68f);

3、浏览器发出读取文件(URL 中域名后面部分对应的文件)的HTTP 请求，该请求报文作为 [TCP 三次握手](http://www.jianshu.com/p/ef892323e68f)的第三个报文的数据发送给服务器;

4、服务器对浏览器请求作出响应，并把对应的 html 文本发送给浏览器;

5、释放 [TCP连接](http://www.jianshu.com/p/ef892323e68f);

6、浏览器将该 html 文本并显示内容;

# GET和POST请求的区别

###### GET请求

GET /books/?sex=man&name=Professional HTTP/1.1

Host: www.wrox.com

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)

Gecko/20050225 Firefox/1.0.1

Connection: Keep-Alive

注意最后一行是空行

###### POST请求

POST / HTTP/1.1

Host: www.wrox.com

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)

Gecko/20050225 Firefox/1.0.1

Content-Type: application/x-www-form-urlencoded

Content-Length: 40

Connection: Keep-Alive

name=Professional%20Ajax&publisher=Wiley

1、GET提交，请求的数据会附在URL之后（就是把数据放置在HTTP协议头中），以?分割URL和传输数据，多个参数用&连接；例 如：login.action?name=hyddd&password=idontknow&verify=%E4%BD%A0 %E5%A5%BD。如果数据是英文字母/数字，原样发送，如果是空格，转换为+，如果是中文/其他字符，则直接把字符串用BASE64加密，得出如： %E4%BD%A0%E5%A5%BD，其中％XX中的XX为该符号以16进制表示的ASCII。

POST提交：把提交的数据放置在是HTTP包的包体中。上文示例中红色字体标明的就是实际的传输数据

###### 因此，GET提交的数据会在地址栏中显示出来，而POST提交，地址栏不会改变

2、传输数据的大小：首先声明：HTTP协议没有对传输的数据大小进行限制，HTTP协议规范也没有对URL长度进行限制。

而在实际开发中存在的限制主要有：

**GET**:特定浏览器和服务器对URL长度有限制，例如 IE对URL长度的限制是2083字节(2K+35)。对于其他浏览器，如Netscape、FireFox等，理论上没有长度限制，其限制取决于操作系 统的支持。

因此对于GET提交时，传输数据就会受到URL长度的 限制。

**POST**:由于不是通过URL传值，理论上数据不受 限。但实际各个WEB服务器会规定对post提交数据大小进行限制，Apache、IIS6都有各自的配置。

3、安全性

POST的安全性要比GET的安全性高。比如：通过GET提交数据，用户名和密码将明文出现在URL上，因为(1)登录页面有可能被浏览器缓存；(2)其他人查看浏览器的历史纪录，那么别人就可以拿到你的账号和密码了，除此之外，使用GET提交数据还可能会造成Cross-site request forgery攻击

4、Http get,post,soap协议都是在http上运行的

（1）get：请求参数是作为一个key/value对的序列（查询字符串）附加到URL上的  
查询字符串的长度受到web浏览器和web服务器的限制（如IE最多支持2048个字符），不适合传输大型数据集同时，它很不安全

（2）post：请求参数是在http标题的一个不同部分（名为entity body）传输的，这一部分用来传输表单信息，因此必须将Content-type设置为:application/x-www-form- urlencoded。post设计用来支持web窗体上的用户字段，其参数也是作为key/value对传输。  
但是：它不支持复杂数据类型，因为post没有定义传输数据结构的语义和规则。

（3）soap：是http post的一个专用版本，遵循一种特殊的xml消息格式  
Content-type设置为: text/xml 任何数据都可以xml化。

Http协议定义了很多与服务器交互的方法，最基本的有4种，分别是GET,POST,PUT,DELETE. 一个URL地址用于描述一个网络上的资源，而HTTP中的GET, POST, PUT, DELETE就对应着对这个资源的查，改，增，删4个操作。 我们最常见的就是GET和POST了。GET一般用于获取/查询资源信息，而POST一般用于更新资源信息.

我们看看GET和POST的区别

* 1. GET提交的数据会放在URL之后，以?分割URL和传输数据，参数之间以&相连，如EditPosts.aspx?name=test1&id=123456. POST方法是把提交的数据放在HTTP包的Body中.
  2. GET提交的数据大小有限制（因为浏览器对URL的长度有限制），而POST方法提交的数据没有限制.
  3. GET方式需要使用Request.QueryString来取得变量的值，而POST方式通过Request.Form来获取变量的值。
  4. GET方式提交数据，会带来安全问题，比如一个登录页面，通过GET方式提交数据时，用户名和密码将出现在URL上，如果页面可以被缓存或者其他人可以访问这台机器，就可以从历史记录获得该用户的账号和密码.

### 18. 三次握手

答: 建立TCP连接需要三次握手, 而断开连接需要执行四次挥手.

* 三次握手: 首先Client端发送连接请求报文，Server段接受连接后回复ACK报文，并为这次连接分配资源。Client端接收到ACK报文后也向Server段发生ACK报文，并分配资源，这样TCP连接就建立了。
  + 第一步: 客户机的TCP先向服务器的TCP**发送一个连接请求报文**. 这个特殊的报文中不含应用层数据, 其首部中的**SYN标志位被置1**. 另外, 客户机会随机选择一个起始序号**seq=x**(连接请求报文不携带数据,但要消耗掉一个序号)
  + 第二步: 服务器端的TCP收到连接请求报文后, 若同意建立连接, 就向客户机发送请求, 并为该TCP连接**分配TCP缓存和变量**. 在确认报文中,**SYN和ACK位都被置为1**, 确认号字段的值为**x+1**, 并且服务器随机产生起始序号**seq=y**(确认报文不携带数据, 但也要消耗掉一个序号). 确认报文同样不包含应用层数据.
  + 第三步: 当客户机收到确认报文后, 还要向服务器给出确认, 并且也要给**该连接分配缓存和变量**. 这个报文的**ACK标志位被置为1**, 序号字段为**x+1**, 确认号字段为**y+1**
* 四次挥手
  + 第一步: 客户机打算关闭连接,就向其TCP发送一个**连接释放报文,**并停止再发送数据,**主动关闭TCP连接**, 该报文的**FIN标志位被置1, seq=u**, 它等于前面已经传送过的数据的最后一个字节的序号加1(FIN报文即使不携带数据,也要消耗掉一个序号)
  + 第二步: 服务器接收连接释放报文后即发出确认, 确认号是**ack=u+1**, 这个**报文自己的序号是v**, 等于它前面已传送过的数据的最后一个自己的序号加1. 此时, 从客户机到服务器这个方向的连接就释放了, TCP连接处于**半关闭状态**. 但服务器若发送数据, 客户机仍要接收, 即从服务器到客户机的连接仍未关闭.
  + 第三步: 若服务器已经没有了要向客户机发送的数据, 就**通知TCP释放连接**, 此时其发出**FIN=1**的连接释放报文
  + 第四步: 客户机收到连接释放报文后, 必须发出确认. 在确认报文中, **ACK字段被置为1**, 确认号**ack=w+1, 序号seq=u+1.** 此时, TCP连接还没有释放掉, 必须经过等待计时器设置的时间2MSL后, A才进入到连接关闭状态.

### 19. tcp/ip／http对应哪一层 七层模型

* TCP/IP 四层协议: 应用层、传输层、网络互连层和主机到网络层. http对应应用层
* ISO 七层模型: 物理层, 数据链路层, 网络层, 传输层, 会话层, 表示层, 应用层.  http对应应用层

### 20. 浏览器中输入网址后到页面展现的过程

**1）用户输入网址**

**2）浏览器通过DNS获取网站的IP地址。**客户端先检查本地是否有对应的IP地址，若找到则返回响应的IP地址。若没找到则请求上级DNS服务器，直至找到或到根节点。

　　　　DNS查找IP地址的顺序: 浏览器缓存、系统缓存、互联网服务提供商（ISP）的DNS缓存、递归搜索（从浏览器缓存开始，如果没找到就继续往下一个找。）找到后，浏览器会获得一个IP地址。

**3）浏览器客户端发送http请求**

　　　　HTTP请求包括**请求报头**和**请求主体**两个部分，其中请求报头包含了至关重要的信息，包括请求的方法（GET / POST）、目标url、遵循的协议（http / https / ftp…），返回的信息是否需要缓存，以及客户端是否发送cookie等。

**4）传输层TCP传输报文**。TCP协议通过“三次握手”等方法保证传输的安全可靠。

**5）网络层IP协议查询MAC地址**

　　　　IP协议的作用是把TCP分割好的各种数据包传送给接收方。而要保证确实能传到接收方还需要接收方的MAC地址，也就是物理地址。IP地址和MAC地址是一一对应的关系，一个网络设备的IP地址可以更换，但是MAC地址一般是固定不变的。ARP协议可以将IP地址解析成对应的MAC地址。当通信的双方不在同一个局域网时，需要多次中转才能到达最终的目标，在中转的过程中需要通过下一个中转站的MAC地址来搜索下一个中转目标。

**7）数据到达数据链路层**

　　　　在找到对方的MAC地址后，就将数据发送到数据链路层传输。这时，客户端发送请求的阶段结束

**8）服务器接收数据**

　　　　 接收端的服务器在链路层接收到数据包，再层层向上直到应用层。这过程中包括在运输层通过TCP协议将分段的数据包重新组成原来的HTTP请求报文。

**9）服务器响应请求**

　　　　服务接收到客户端发送的HTTP请求后，查找客户端请求的资源，并返回响应报文，响应报文中包括一个重要的信息——状态码。状态码由三位数字组成，其中比较常见的是200 OK表示请求成功。301表示永久重定向，即请求的资源已经永久转移到新的位置。在返回301状态码的同时，响应报文也会附带重定向的url，客户端接收到后将http请求的url做相应的改变再重新发送。404 not found 表示客户端请求的资源找不到。

**10）服务器返回响应文件**

　　　　请求成功后，服务器会返回相应的HTML文件。接下来就到了页面的渲染阶段了。

**11） 页面渲染：** 解析HTML以构建DOM树 –> 构建渲染树 –> 布局渲染树 –> 绘制渲染树。

　　　　关于页面渲染过程：

　　　　1）解析HTML代码，生成一棵DOM树

　　　　2）解析CSS文件

　　　　3）生成渲染树（受样式影响，包含不可见元素）

　　　　4）渲染树中的节点

　　　　DOM树是由HTML文件中的标签排列组成，渲染树是在DOM树中加入CSS或HTML中的style样式而形成。渲染树只包含需要显示在页面中的DOM元素，像<head>元素或display属性值为none的元素都不在渲染树中。

   　　　在浏览器还没接收到完整的HTML文件时，它就开始渲染页面了，在遇到外部链入的脚本标签或样式标签或图片时，会再次发送HTTP请求重复上述的步骤。在收到CSS文件后会对已经渲染的页面重新渲染，加入它们应有的样式，图片文件加载完立刻显示在相应位置。在这一过程中可能会触发页面的重绘或重排。

　　参考： [从输入URL到浏览器显示页面发生了什么](http://www.cnblogs.com/kongxy/p/4615226.html)

　　其他： [浏览器中输入网址后到页面展现的过程](http://www.cnblogs.com/haoyijing/p/5789348.html#z5_4)

### 21. 浏览器是如何进行加载, 解析, 渲染的呢? 重点说一下浏览器渲染页面的过程?

答:

1. 用户访问网页, DNS服务器(域名解析系统)会根据用户提供的域名查找对应的IP地址, 找到后, 系统会向对应IP地址的网络服务器发送一个http请求
2. 网络服务器解析请求, 并发送数据给数据库服务器
3. 数据库服务器将请求的资源返回给网络服务器, 网络服务器解析数据, 并生成html文件, 放入http response中, 返回给服务器.
4. 浏览器解析http response
5. 浏览器解析http response后, 需要下载html文件, 以及html文件内包含的外部引用文件, 及文件内涉及的图片或者多媒体文件

关于加载顺序:

* 当浏览器获得一个html文件, 会"自上而下"加载, 并在加载过程中进行解析渲染. 下载和渲染是同时进行的
* 在渲染到页面的某一部分时, 其上面到所有部分都已经下载完成(并不是说所有关联元素都已经下载完)
* 如果加载过程中遇到外部css文件, 浏览器会发出一个请求, 来获取css文件.
* 样式表在下载完成后, 将和以前下载的所有样式表一起进行解析, 解析完成后, 将对此前所有元素(含以前已经渲染的)**重新进行渲染**
* 遇到**图片资源**, 浏览器会发出请求获取图片资源. 这是**异步请求**, 并不会影响html文档进行加载,
* 当文档加载过程中遇到js文件, html文档会**挂起渲染**(加载解析渲染同步)的线程, 不仅要等待文档中js文件加载完毕, 还要等待解析执行完毕, 才可以恢复html文档的渲染线程. 即js的加载**不能并行下载和解析**
  + 原因: js有可能会修改DOM, 比如document.write. 这意味着, 在js执行完成前, 后续所有资源的下载可能是没有必要的, 这是js阻塞后续资源下载的根本原因.
  + 所以一般将外部引用的js文件放在</body>前
* 虽然css文件的加载不影响js文件的加载,但是却影响js文件的执行, 即使js文件内只有一行代码, 也会造成阻塞
  + 原因: 可能会有: var width = $('#id').width(). 这意味着, 在js代码执行前, 浏览器必须保证css文件已下载和解析完成。这也是css阻塞后续js的根本原因。
  + 办法：当js文件不需要依赖css文件时，可以将js文件放在头部css的前面。
  + 当然除了，<link href="" />这种形式，内部<style></style>这种样式定义，在考虑阻塞时也要考虑
* js,css中如果有重定义, 后定义函数将覆盖前定义函数

主要解析过程:

1. 浏览器解析html源码, **创建一棵DOM树**
2. 浏览器解析CSS代码, **计算出最终的样式数据**
3. **js解析**因为文件在加载的同时也进行解析
4. 构建DOM树, 并且计算出样式数据后, 下一步就是**构建一棵渲染树**(rendering tree)
   1. 渲染树和DOM树有区别, DOM树完全与html标签一一对应, 但是渲染树会忽略掉不需要渲染的元素, 比如head, display: none的元素等
   2. 一大段文本中的每一行在渲染树中都是一个独立的节点
   3. 渲染树的每一个节点都存储有对应的css属性
5. 渲染树创建好, 浏览器就可以根据渲染树直接把页面绘制到屏幕上

* 1.用户输入网址（假设是个html页面，并且是第一次访问），浏览器向服务器发出请求，服务器返回html文件；
* 2.浏览器开始载入html代码，发现＜head＞标签内有一个＜link＞标签引用外部CSS文件；
* 3.浏览器又发出CSS文件的请求，服务器返回这个CSS文件；
* 4.浏览器继续载入html中＜body＞部分的代码，并且CSS文件已经拿到手了，可以开始渲染页面了；
* 5.浏览器在代码中发现一个＜img＞标签引用了一张图片，向服务器发出请求。此时浏览器不会等到图片下载完，而是继续渲染后面的代码；
* 6.服务器返回图片文件，由于图片占用了一定面积，影响了后面段落的排布，因此浏览器需要回过头来重新渲染这部分代码；
* 7.浏览器发现了一个包含一行Javascript代码的＜script＞标签，赶快运行它；
* 8.Javascript脚本执行了这条语句，它命令浏览器隐藏掉代码中的某个＜div＞ （style.display=”none”）。杯具啊，突然就少了这么一个元素，浏览器不得不重新渲染这部分代码；
* 9.终于等到了＜/html＞的到来，浏览器泪流满面……
* 10.等等，还没完，用户点了一下界面中的“换肤”按钮，Javascript让浏览器换了一下＜link＞标签的CSS路径；
* 11.浏览器召集了在座的各位＜div＞＜span＞＜ul＞＜li＞们，“大伙儿收拾收拾行李，咱得重新来过……”，浏览器向服务器请求了新的CSS文件，重新渲染页面。

### 1. js实现数组去重怎么实现?

**方法1**. 创建一个新的临时数组来保存数组中已有的元素

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | var a = new Array(1,2,2,2,2,5,3,2,9,5,6,3);  Array.prototype.unique1 = function(){      var n = [];     //一个新的临时数组      for(var i=0; i<this.length; i++){          //如果把当前数组的第i已经保存进了临时数组, 那么跳过          if(n.indexOf(this[i]) == -1){              n.push(this[i]);          }      }      return n;  }  console.log(a.unique1()); |

**方法2.** 使用哈希表存储已有的元素

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | Array.prototype.unique2 = function(){      var hash = {},          n = [];     //hash 作为哈希表, n为临时数组      for(var i=0; i<this.length; i++){          if(!hash[this[i]]){         //如果hash表中没有当前项              hash[this[i]] = true;   //存入hash表              n.push(this[i]);        //当前元素push到临时数组中          }      }      return n;  } |

**方法3**. 使用indexOf判断数组元素第一次出现的位置是否为当前位置

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | Array.prototype.unique3 = function(){      var n = [this[0]];      for(var i=1; i<this.length; i++)    //从第二项开始遍历      {          //如果当前数组元素在数组中出现的第一次的位置不是i          //说明是重复元素          if(this.indexOf(this[i]) == i){              n.push(this[i]);          }      }      return n;  } |

**方法4.** 先排序再去重

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | Array.prototype.unique4 = function(){      this.sort(function(a, b){ return a - b;});      var n = [this[0]];      for(var i=1; i<this.length; i++){          if(this[i] != this[i-1]){              n.push(this[i]);          }      }      return n;  } |