**你为什么接触前端？**

 喜欢，喜欢敲代码然后看着自己敲的代码运行出结果的感觉。对工科的很多东西都很感兴趣，在学校也比较喜欢C语言的课程设计之类的可以自己动手敲代码的课。 。。 。 。 。。

**你对前端界面工程师这个职位是怎么样理解的？它的前景会怎么样**

前端界面工程师：1. 与交互设计师、视觉设计师协作，根据设计图完成页面制

做 2. 维护及优化网站前端性能。

前景： 长期以来，国内前端开发在整个软件开发行业内占得比重较小，发展较晚。网站也多倾向于赢利最大化而轻视用户体验。前后端比例悬殊大。

近几年来，随着 *以用户为中心* 的思想普及发展。前端开发呈现出强劲的发展态势，前端开发人员在项目中的重要性日益突出。前端开发人员呈现出严

重的人员短缺现象。

**有关**HTML

**1，**有关HTML的语义化，你的理解：html中自带了一些标签，这些标签有特定含义，比如H1-H6代表标题、p代表段落、img代表图片、html5里也推出了一些新的标签：header、footer、nav这些标签都有特定的含义，使用的时候应该尽量按照它的语义来使用

好处：1.别人阅读你的代码的时候能够根据标签才出来你的用意，有利于程序员阅读，便于团队开发和维护

2.浏览器读取方便

3.有利于搜索引擎优化（SEO：Search Engine Optimization）（比如说会根据标签来搜索，可能重点会搜索h1）

4、有一些标签默认会有一些样式，如果浏览器禁止了CSS样式还可以达到突出样式的效果，比如h1

**你如何理解HTML结构的**语义化

去掉或样式丢失的时候能让页面呈现清晰的结构：

1.html本身是没有表现的，我们看到例如<h1>是粗体，字体大小2em，加粗；<strong>是加粗的，不要认为这是html的表现，这些其实html默认的css样式在起作用，所以去掉或样式丢失的时候能让页面呈现清晰的结构不是语义化的HTML结构的优点，但是浏览器都有有默认样式，默认样式的目的也是为了更好的表达html的语义，可以说浏览器的默认样式和语义化的HTML结构是不可分割的。屏幕阅读器（如果访客有视障）会完全根据你的标记来“读”你的网页.

2.例如,如果你使用的含语义的标记,屏幕阅读器就会“逐个拼出”你的单词,而不是试着去对它完整发音.

PDA、手机等设备可能无法像普通电脑的浏览器一样来渲染网页（通常是因为这些设备对CSS的支持较弱）.

3.使用语义标记可以确保这些设备以一种有意义的方式来渲染网页.理想情况下,观看设备的任务是符合设备本身的条件来渲染网页.语义标记为设备提供了所需的相关信息,就省去了你自己去考虑所有可能的显示情况（包括现有的或者将来新的设备）.例如,一部手机可以选择使一段标记了标题的文字以粗体显示.而掌上电脑可能会以比较大的字体来显示.无论哪种方式一旦你对文本标记为标题,您就可以确信读取设备将根据其自身的条件来合适地显示页面.

搜索引擎的爬虫也依赖于标记来确定上下文和各个关键字的权重.

4.过去你可能还没有考虑搜索引擎的爬虫也是网站的“访客”,但现在它们他们实际上是极其宝贵的用户.没有他们的话,搜索引擎将无法索引你的网站,然后一般用户将很难过来访问.你的页面是否对爬虫容易理解非常重要,因为爬虫很大程度上会忽略用于表现的标记,而只注重语义标记.

5.因此,如果页面文件的标题被标记,而不是,那么这个页面在搜索结果的位置可能会比较靠后.除了提升易用性外,语义标记有利于正确使用CSS JavaScript,因为其本身提供了许多“钩钩”来应用页面的样式与行为.SEO主要还是靠你网站的内容和外部链接的。

便于团队开发和维护

6.W3C给我们定了一个很好的标准，在团队中大家都遵循这个标准，可以减少很多差异化的东西，方便开发和维护，提高开发效率，甚至实现模块化开发。

**什么是**SEO**？**

SEO（Search Engine Optimization），汉译：*搜索引擎优化*。搜索引擎优化是一种利用搜索引擎的搜索规则来提高目的网站在有关搜索引擎内的排名的方式。所谓“针对搜索引擎作最佳化的处理”，是指为了要让网站更容易被搜索引擎接受。深刻理解是：通过SEO这样一套基于搜索引擎的营销思路，为网站提供生态式的自我营销解决方案，让网站在行业内占据领先地位，从而获得品牌收益。SEO可分为站外SEO和站内SEO两种。

**谈谈以前端角度出发做好SEO需要考虑什么**

1、了解搜索引擎如何抓取网页（如何抓取：关键词的提取）和如何索引网页你需要知道一些搜索引擎的基本工作原理，各个搜索引擎之间的区别，搜索机器人（SE robot 或叫 web crawler）如何进行工作，搜索引擎如何对搜索结果进行排序等等。

2、Meta标签优化

主要包括主题（Title)，网站描述(Description)，和关键词（Keywords）。还有一些其它的隐藏文字比如Author（作者），Category（目录），Language（编码语种）等。

3、如何选取关键词并在网页中放置关键词

搜索就得用关键词。关键词分析和选择是SEO最重要的工作之一。首先要给网站确定主关键词（一般在5个上下），然后针对这些关键词进行优化，包括关键词密度（Density），相关度（Relavancy），突出性（Prominency）等等。

4、了解主要的搜索引擎

虽然搜索引擎有很多，但是对网站流量起决定作用的就那么几个。比如英文的主要有Google，Yahoo，Bing等；中文的有百度，搜狗，有道等。

不同的搜索引擎对页面的抓取和索引、排序的规则都不一样。还要了解各搜索门户和搜索引擎之间的关系，比如AOL网页搜索用的是Google的搜索技术，MSN用的是Bing的技术。

5、主要的互联网目录

Open Directory自身不是搜索引擎，而是一个大型的网站目录，他和搜索引擎的主要区别是网站内容的收集方式不同。目录是人工编辑的，主要收录网站主页；搜索引擎是自动收集的，除了主页外还抓取大量的内容页面。

6、按点击付费的搜索引擎

搜索引擎也需要生存，随着互联网商务的越来越成熟，收费的搜索引擎也开始大行其道。最典型的有Overture和百度，当然也包括Google的广告项目Google Adwords。越来越多的人通过搜索引擎的点击广告来定位商业网站，这里面也大有优化和排名的学问，你得学会用最少的广告投入获得最多的点击。

7、搜索引擎登录

网站做完了以后，别躺在那里等着客人从天而降。要让别人找到你，最简单的办法就是将网站提交（submit）到搜索引擎。如果你的是商业网站，主要的搜索引擎和目录都会要求你付费来获得收录（比如Yahoo要299美元），但是好消息是（至少到目前为止）最大的搜索引擎Google目前还是免费，而且它主宰着60％以上的搜索市场。

8、链接交换和链接广泛度（Link Popularity）

网页内容都是以超文本（Hypertext）的方式来互相链接的，网站之间也是如此。除了搜索引擎以外，人们也每天通过不同网站之间的链接来Surfing（“冲浪”）。其它网站到你的网站的链接越多，你也就会获得更多的访问量。更重要的是，你的网站的外部链接数越多，会被搜索引擎认为它的重要性越大，从而给你更高的排名。

9、标签的合理使用

css:

css的兼容性问题：（can i use ）

列举CSS兼容性：1. 透明度问题，标准浏览器(0-1)opacity:0.2，IE (0-100);filter:  alpha(opacity=20);

2、IE6不支持display：inline－block （**inline-block属性值，使其既具有block的可以设置宽高度特性，又具有inline的同行特性）**（\_display:inline;）**3、**（在css参考手册中有许多问题和解决方案）在IE6及更早的浏览器中定义小高度容器(因为默认有字体、行高)：overflow:hidden;font-size:0;height:1px;line-height:0;

4，IE6中浮动式产生一个双边距的问题：把元素的display:inline

5，IE6之间的浏览器不支持min-height，解决：加一个\_height:200px;

1. CSShack（利用了这些浏览器的漏洞价格，专门用这些漏洞来解决兼容性问题）:条件hack、属性级hack（\_color:IE6；\*color:IE7；color:#090\9:IE9）、选择符级hack3 （\*html是IE6；\*+html是IE7），

**7.**被点击过的超链接不再具有hover、active了

   解决：CSS属性排列顺序 Link-Visit-Hover-Active

把：hover ：visit之类的里面的样式设置成一样的

1. IE中有一个3像素BUG，当一个元素浮动，另一个元素和他并列是会有一个3像素的边距，给左边浮动的元素加一个\_margin-right :3px来解决。负值往右，正值往左

.**aa**{  
 **width**: 200**px**;  
 **height**: 200**px**;  
 **float**: **left**;  
 **background**: **#ff0000**;  
 **\_margin-right**:-3**px**;  
}  
.**bb**{  
 */\*float: left;\*/* **background**: **#0000ff**;  
 **width**: 200**px**;  
 **height**: 200**px**;  
}

**9**.  IE6只支持 a标签的：hover，不支持其他标签的：hover

**10.** IE6不支持png24的透明（png透明）

CSS3中很多东西在IE中都不支持，手册中有CSS中的常见问题。

**hack**

(引入条件hack，引入一个css文件）

<!—[if IE 6]>

<script src=”DD\_belatedPNG\_0.0.8a.js”></script>

<script>

DD\_belatedPNG.fix(‘\*’);

</script>

<![endif]-->条件级hack，解决IE6下面不支持png透明问题

如果想要页面支持IE6，在head里面写

<!—[if IE 6]>

<link rel=”stylesheet” href=”ie6\_sb.css”/>

<![endif]-->

**条件hack:**

<!--[if IE 6]>

<style>

#div1{

background: red;

}

</style>

<link rel="stylesheet" href="ie8.css">

<![endif]—>

单独为IE6写一个CSS样式做兼容，不需要就删

**属性级hack:**

p{

color: #00ff00;

color: #ff0000\9;    //IE8+

\*color: #ff0000;     //IE7及以下

\_color: #00ff00;     //IE6及以下

}

**选择符级hack:**

\* html .test{     //IE6及以下

color: red;

}

\* + html .test{  //IE7

color: yellow;

}

IE6以前浏览器不支持min-height:px;怎么解决？

min-height:100px;

\_height:100px;

**Css选择器**：标签、ID、类、后代（空格的）、子代（加<>的）、分组（加,的）、伪类（加：的）、属性（{name:lisi}）

Css3里：firstChild、lastChild、伪类选择器（当成一个对象）、伪对象选择器（不是真有这个对象，当成一个对象来使用）

**1，**CSS的选择器：标签选择器、ID选择器 #，Class选择器 . ，标签选择器 p ，后代选择器 h1 a ，伪类选择器 a：hover，伪对象选择器 a:after，属性选择器，有关

 伪类选择器，伪对象选择器(p标签的第一行，firstline)是什么，都有哪些？

伪类（）：link，visited，hover，active，focus，first-child

伪对象(伪元素)：before，after，first-line，first-letter

新的选择器，伪类选择器和伪对象选择器，（css3中对数字不换行），伪对象不是真有对象，而是当成一个对象来使用，伪对象用两个：：，伪类选择器是真的有这个对象

伪类选择器，是选择真实存在的元素；伪对象选择器是

a link链接，hover划过，focus触发，单击但是不抬起鼠标，visited访问过的

not选择器是排除后面的选择器；root选网页中根元素，html；nth-child（n）选择第几个孩子，n可以是数，也支持公式（2n+1这类的）还有odd(奇数)even（偶数），比如：#div1 p:nth-child(1){这里面样式操作的就是div下面的第一个元素，当元素是p的时候才会被选中}；nth-of-type（n）按类型选择，比如：#div1 p:nth-of-type(1){这里面样式操作的就是选中类型是p标签中的第一个}

**锚点**：在style里面写#div2:target{这里面的样式是当点击div2的时候会瞬间触发的样式，div2是一个a标签，上面的地址栏会加上#div，只要#div在，样式就不恢复}

**after:**在style里面写#div1:after{content:’hahaha’;background:#ff0000;width:200px;display:block;}会在div1的后面插入：after里面的内容，插入的内容是行级元素；after在使用的时候可以解决：如果一个元素的子元素浮动，父元素就会没有高，在已知高度的情况下可以直接设置高度解决，但是在高度不确定的情况下，比如：<div id=”div2” class=”clearfix”><div id=”left”>left</div><div id=”right”>right</div></div>.clearfix:after{content:’’; display:block;clear:both;这里面是指在div2的后面插入东西}

2，行级元素和块级元素：块级元素：div。P。Hi。Ul。Li，特点：垂直分布，宽度自适应，占满父元素的剩余空间

行级元素：span，a,img,input，style，特点：横着分布，不能设置宽高。宽度由内容撑开，但是img可以设置宽高

行内元素主要有：<span>、<a>、<b>、<img>、<br>、<button>、<strong>、<textarea>、<select>

块级元素主要有：<div>、<ul>、<li>、<p>、<fieldset>、<form>、<h1>、<h2>、<h3>、<h4>、<h5>、<h6>、<hr>、

<iframe>、<ol>、<pre>、<table>、<tr>、<td>

行内元素可以通过display:block转为块级元素。另外块级元素的margin和padding都正常，行内元素左右maring和左右padding正常，上下不识别，

也就是说不能通过margin-top和padding-top来改变行高。

2，权重值：ن

ن ن！Important（不推荐使用）的权重大于style（内链），优先级的积分为style为1000，ID为100，类选择器属性选择器为10，标签选择器、伪类选择器为1.

除了内链样式，即在标签里面跟着的样式之外，其他方式设置样式都是后面的把前面的覆盖。声明的时候谁在后面谁优先，除非加上!important

对于！important，IE不认识，其他浏览器优先解析。

如何**提升**权重值：增加选择器，在css里面#xx.xxx；或者！important，这个是最优先，但是不建议使用

3，定位

static  没有定位（默认值，正常它在文本流中应该出现的位置）

relative   相对定位（相对自己的初始位置），原来空间不释放

absolute  绝对定位（相对最近已定位的的元素），原来空间释放

fixed   固定定位（相对于浏览器），原来空间释放     (IE6不支持=怪异模式不支持)

IE6不支持固定定位，**解决方法**，用绝对定位absolute代替，

在window.onscroll事件中，设置固定元素的位置，但有可能由于滚动太快，但是计算慢，出现抖动现象。

var oNav = document.getElementById('nav');

window.onscroll = function(){

var scrollTop = document.documentElement.scrollTop || document.body.scrollTop;

oNav.style.top = scrollTop + 'px';

};

4，浮动：还有什么？

**两列布局：**左侧固定，右侧自适应，左侧宽高固定右侧不能浮动（块级元素），然后设置margin-left把左侧让出来。

**三列布局**：左右固定，中间自适应，dom结构，先左右后中间

**浮动高度塌陷：**元素浮动之后父元素的高度会塌陷，有两种方法解决：1、如果父元素的高度是固定的，那就把高度写死

2、可以在dom元素里加一个<div style="clear:both"></div>或者定义一个class=”clearfix”,在style中写上.clearfix:after{/\*after：伪类\*/

content: '';

display: block;

clear: both;

height:0;

visibility:hidden

}

css盒模型；标准盒模型。IE下的怪异盒模型（把文档类型去掉）：宽会把内边距和外边距减去而不是加上。

**清除浮动的方法：**

1.父级div手动定义height，就解决了父级div无法自动获取到高度的问题。

2.结尾处加空div标签  <div style="clear:both;"></div>

3.父级div定义 伪类:after 和 zoom

.clearfloat:after{ display:block;    clear:both;    content:"";   visibility:hidden;   height:0;   }

.clearfloat{zoom:1}

   IE8以上和非IE浏览器才支持:after，原理和方法2类似

   zoom(IE专有属性，它可以设置或检索对象的缩放比例)

6，CSS盒模型：W3C标准盒模型，怪异模式下的盒模型

1. **W3C 标准盒模型**

比如我定义了一个

#div1{

width: 200px;

height: 200px;

border: 5px solid #000;

padding: 10px;

background: #f00;

}

的div，视觉上呈现的宽度就是w+b+p；在标准浏览器下css3里有一个box-sizing:设置成border-box就可以使其和IE6下的怪异盒模型一致；还有一个content-box

*容器可用宽度：*

clientWidth = width + padding（如果有滚动条，减掉滚动条）

*实际内容的宽度：*

 scrollWidth = width + padding

 srcollHeight = height + padding(如果内容很多，是有滚动条的，scrollHeight的高度为内容的高度，大于容器的高度clientHeight)

*容器可视宽度：*

 offsetWidth = width + padding + border

**2.IE 传统下盒模型**（ IE6 以下，不包含 IE6 版本 或  ”QuirksMode 下 IE5.5+” ） 如果丢失了doctype这个文档类型，就会触发怪异模式

 clientWidth = width - border（如果有滚动条，减掉滚动条）

 offsetWidth = width

两块级元素，给子元素加上外边距，加不上，会因为上外边距合并，父子元素一起掉下来。

解决方法：给父元素加上内边距，或给子元素float

 float，定位的元素脱离了原来的文档流，就不会上内边距的问题

display: none;空间释放

visibility: hidden;空间不释放

*标准盒模型* 的宽度就是：width

*怪异模式下的盒模型*的宽度就是：width+border+padding

DOCTYPE是文档类型，完整定义DOCTYPE，会触发标准模式，如果DOCTYPE缺失在ie6,ie7,ie8下将会触发怪异模式

标准盒模型的宽度就是width，怪异模式下的盒模型是width+padding+border

弹性盒模型，  display：-webkit-flex：  flex：1；flex：2；flex：3则各占一份两份三份~新语法。若设置了父元素的宽度，则在**老的语法**中display：box；   box-flex：1,2,3则也是各占。若子元素有一个有固定宽度，则按比例分布剩余的部分。

弹性盒模型：

box-flex是分配剩余空间，针对块级元素有效

#div1{

display:-webkit-box;

}

#div1 div:nth-child(1){

-webkit-box-flex:1;

}\*n分成几份

1.W3C 标准盒模型

容器可用宽度：

 clientWidth = width + padding（如果有滚动条，减掉滚动条）

实际内容的宽度：

 scrollWidth = width + padding

 srcollHeight = height + padding(如果内容很多，是有滚动条的，scrollHeight的高度为内容的高度，大于容器的高度clientHeight)

容器可视宽度：

 offsetWidth = width + padding + border

2.IE 下的怪异盒模型（ IE6 以下，不包含 IE6 版本 或  ”QuirksMode 下 IE5.5+” ）

 clientWidth = width - border（如果有滚动条，减掉滚动条）

 offsetWidth = width

两块级元素，给子元素加上外边距，加不上，会因为上外边距合并，父子元素一起掉下来。

解决方法：给父元素加上内边距，或给子元素float

 float，定位的元素脱离了原来的文档流，就不会上内边距的问题

display: none;空间释放

visibility: hidden;空间不释放

**HTML5  的新的表单属性**

**新增了语义化的标签：header\nav\footer**

**新增了一些表单控件和表单属性**

**新增了音频、视频**

**新增了新的选择器：querySelector\querySelectorAll\getElementByClassName**

**新增加了一些json的新的方法：parse\stringify（对象拷贝，先把对象转化成字符串，再转化成json）**

**新增了历史管理、本地存储**

**新的 form 属性：**

autocomplete

novalidate

**新的 input 属性：**

autocomplete

autofocus

form

form overrides (formaction, formenctype, formmethod, formnovalidate, formtarget)

height 和 width

list

min, max 和 step

multiple

pattern (regexp)

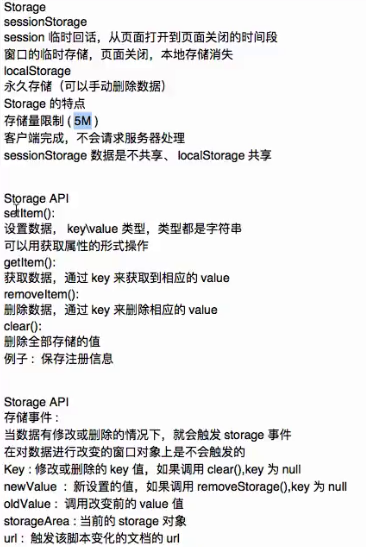
placeholder

required

HTML5 提供了两种在客户端存储数据的新方法：

localStorage - 没有时间限制的数据存储

sessionStorage - 针对一个 session 的数据存储



之前，这些都是由 cookie 完成的。但是 cookie 不适合大量数据的存储，因为它们由每个对服务器的请求来传递，这使得 cookie 速度很慢而且效率也不高。

在 HTML5 中，数据不是由每个服务器请求传递的，而是只有在请求时使用数据。它使在不影响网站性能的情况下存储大量数据成为可能。

对于不同的网站，数据存储于不同的区域，并且一个网站只能访问其自身的数据。

HTML5 使用 JavaScript 来存储和访问数据。

localStorage 方法存储的数据没有时间限制

sessionStorage 方法针对一个 session 进行数据存储。当用户关闭浏览器窗口后，数据会被删除。

sessionStorage 、localStorage 和 cookie 之间的区别

**共同点：**都是保存在浏览器端，且同源的。

**区别：**

**1、** cookie数据始终在同源的[http请求](https://www.baidu.com/s?wd=http%E8%AF%B7%E6%B1%82&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y4mW9BmvNWPW-hnHczPW-90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EP1R1njT3P1T4nHTkP1f1P1mY)中携带（即使不需要），即cookie在浏览器和服务器间来回传递。而sessionStorage和localStorage不会自动把数据发给服务器，仅在本地保存。

cookie数据还有路径（path）的概念，可以限制cookie只属于某个路径下。

2、存储大小限制也不同，cookie数据不能超过4k，同时因为每次[http请求](https://www.baidu.com/s?wd=http%E8%AF%B7%E6%B1%82&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y4mW9BmvNWPW-hnHczPW-90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EP1R1njT3P1T4nHTkP1f1P1mY)都会携带cookie，所以cookie只适合保存很小的数据，如会话标识。sessionStorage和localStorage 虽然也有存储大小的限制，但比cookie大得多，可以达到5M或更大。数据有效期不同， sessionStorage：仅在当前浏览器窗口关闭前有效，自然也就不可能持久保持；localStorage：始终有效，窗口或浏览器关闭也一直保存，因此用作持久数据；cookie只在设置的cookie过期时间之前一直有效，即使窗口或浏览器关闭。

3、作用域不同，sessionStorage不在不同的浏览器窗口中共享，即使是同一个页面；localStorage 在所有同源窗口中都是共享的；cookie也是在所有同源窗口中都是共享的。Web Storage 支持事件通知机制，可以将数据更新的通知发送给监听者。Web Storage 的 api 接口使用更方便。

 H5中的两种方式不会放在请求头里，不会被发送，减少了请求量

Cookie:

在客户端存储信息，问题是存储量小。

安全性不高，可以找到文件。

作用，免登陆；记录广告点击次数信息。。。

Document.cookie = ‘username=lisi(;age = 23)’;

Session会话，重启浏览器就会过期。

Now.setDate(//过期时间);

如果想删除，就把日期往前面设置，就是当前日期之前。

sessionstorage:关闭浏览器就删除

localstorage:一直有，除非手动删除

sessionstorage和localstorage存储量比cookie大

cookie用不用都从客户端发送到服务器端，会浪费传输的数据量，sessionstorage和localstorage不会。sessionstorage和localstorage在最新浏览器支持，HTML5里面的。

用之前先进行能力检测，看是否支持。

cookie，session的区别：

cookie客户端，session服务器端，session是服务器端的会话，登陆成功把用户信息存在session里面，然后在进行操作的时候先判断是否登录，服务器辨别不同的session，为不同的浏览器辨别不同的session，客户端与服务器端建立连接后，服务器端在cookie里面存一个sessionID的码。Session在底层原理会用到cookie

HTML5 <canvas> 标签用于绘制图像（通过脚本，通常是 JavaScript）。

不过，<canvas> 元素本身并没有绘制能力（它仅仅是图形的容器） - 必须使用脚本来完成实际的绘图任务。

getContext(2d) 方法可返回一个对象，该对象提供了用于在画布上绘图的方法和属性。

**Websocket：基于TCP/IP协议的，双向的、全双工的数据连接**

**双向的：客户端、服务器端**

**全双工：数据的发送与接收，两者同步进行**

[CSS3新增UI样式](http://www.cnblogs.com/BetterMyself/p/5660072.html)

**1、圆角：border-radius**

1个：都一样 border-radius: 一样

2个：对角 border-radius: 左上&右下 右上&左下

3个：斜对角 border-radius: 左上 右上&左下 右下

4个：全部，顺时针 border-radius: 左上 右上 右下 左下

border-radius: 1-4个数字 / 1-4个数字——前面是水平半径，后面是垂直半径border-radius: 10px/5px;

不给“/”则水平和垂直一样

**2、边框**

**（1）边框图片：border-image**

border-image-sourceg 引入图片

border-image-slice 切割图片

border-image-width 边框宽度

border-image-repeat 图片的排列方式：round 平铺,repeat 重复,stretch拉伸

**（2）边框颜色 border-colors（火狐下）**

**3、线性渐变**

**（1）格式**

线性渐变格式 linear-gradient([<起点> || <角度>] ?<点>, <点>…)

**只能用在背景上**

**（2）参数**

起点：从什么方向开始渐变

默认：top left（左上角）

角度：从什么角度开始渐变

xxx deg的形式（逆时针旋转）

点：渐变点的颜色和位置

black 50%，位置可选

**（3）实例：**background-image:-webit-linear-gradient(60deg,red 0,blue 40%,green 100%)

**（4）平铺：background：**-webit-repeating-linear-gradient(15deg,green ,green 10px,#fff 10px,#fff 20px,);

**（5）兼容IE9以下:（gradientType=1为从左往右，0为从上往下）**

filter:progid:DXImageTransform.Microsoft.gradient(startColorstr='#ffffff',endColorstr='#ff0000',GradientType='1');　

7，css3动画：

1.变形transform（只有变形没有动画）（-webkit-transform:;）：

*旋转*rotate(围绕x、y、z三个方向)、

*缩放*scale（水平，垂直）、

*位移*translate(比 position 效率高，translateX translateY translateZ,逗号分隔)、

*扭曲*skew（水平，垂直，单位是度数deg）

*矩阵*matrix

scale3d

2.过渡transition（只要改变值就动，过度比较平滑）：做动画的属性(width, all)，动画执行时间，动画形式，延迟时间

（给一个初值，初值一改变就执行动画）

-webkit-transition-property: all;

-webkit-transition-duration: 2s;

-webkit-transition-timing-function: ease;

-webkit-transition-delay:1s;

-webkit-

-moz-

-ms-

-o-

3.动画animation（调用了就动）：动画名称animation-name、执行时间animation-duration、动画函数animation-timing-function、延迟animation-delay、执行次数animation-iteration-count

 (infinite无限次  n次)、方向animation-direction(normal正常  alternate轮流反向) 、保持最终状态forwards

animation(适合复杂动画，就是为了分多阶段)

*.stress{*

*animation:   stress   1s   forwards;*

*-webkit-animation:   stress   1s   forwards;*

*}*

*@keyframes stress{*

*from{}*

*to{  background: #ee0;   }*

*}*

*@-webkit-keyframes stress（动画名字）{*

*from{}                      0%{}*

*to{background: #ee0;}               100%{}*

*}*

4.3d变换，围绕xyz轴方向

如果有多级元素进行3D变换，把perspective加在最外层身上，子元素加上transform-style : preserve－3d

保持3d：transform-style : preserve－3d

视角－webkit－perspective： 800； 不是伪3d，为了呈现3d效果，加在父元素上

视角原点－webkit－perspective-origin： 50% 50%；加在父元素上

**4、径向渐变**

**（1）格式**

线性渐变格式 radial-gradient([<起点>]? [<形状> || <大小>]?  <点>, <点>…)

**（2）参数**

起点：可以是关键字（left,top,right,bottom），具体数值或百分比

形状：ellipse（椭圆）、circle（圆形）

大小 :具体数值或百分比，也可以是关键字（最近端，最近角，最远端，最远角，包含或覆盖 (closest-side, closest-corner, farthest-side, farthest-corner, contain or cover)）;

点：渐变点的颜色和位置

black 50%，位置可选

**5、背景**

**（1）多背景：**逗号分开 background: url(a.jpg) 0 0, url(b.jpg) 0 100%;

**（2）背景尺寸：**background-size:x y

background-size:100% 100%

Cover 放大

Contain 缩小

**（3）背景原点：background-origin ： border | padding | content**

border-box： 从border区域开始显示背景。

padding-box： 从padding区域开始显示背景。

content-box： 从content区域开始显示背景。

**（4）背景裁剪：background-clip： border | padding | content|no-clip**

border-box： 从border区域向外裁剪背景。

padding-box： 从padding区域向外裁剪背景。

content-box： 从content区域向外裁剪背景。

text： 从文本区域向外裁剪背景。**（仅支持webkit内核浏览器）**

**6、遮罩**

Mask-image

Mask-position

Mask-repeat

**ajax是什么**：客户端与服务器端异步的通信的技术（不刷新页面的）

原理：

第一步：**创建XMLHttpRequest 对象**

xmlhttp=new XMLHttpRequest()（标准浏览器中用）;（它有兼容性问题，在ie中用的是new ActiveXObject(‘’)）

**if**(**window**.XMLHttpRequest){  
 **var** xmlhttp = **new** XMLHttpRequest();  
}**else**{  
 **var** xmlhttp = **new** ActiveXObject(**'Microsoft.XMLHTTP'**);  
}

第二步：**向服务器发送请求**

*get  
xmlhttp.open('GET','servers.php?name=xx',true);  
xmlhttp.send();  
// post*xmlhttp.open(**'POST'**,**'servers.php'**,**true**); xmlhttp.setRequestHeader(**"Content-type"**,**"application/x-www-form-urlencoded"**);  
xmlhttp.send(**'name=cc&age=20'**);

第三步：服务器端接受数据并处理，响应结果，在页面中通过**执行回调函数，在回调函数中进行相应的dom操作（**不阻塞用户，达到无刷新的效果**）**

xmlhttp.onreadystatechange=function()

  {

  if (xmlhttp.readyState==4 && xmlhttp.status==200)

    {请求状态5个，0 = 未初始化，1 = 正在加载，2 = 加载完成，3 = 交互中，4 = 完成

    //相应的dom操作

    document.getElementById("myDiv").innerHTML=xmlhttp.responseText;

    }

  }

readystate

  0 - (未初始化)   还没有调用send()方法

  1 - (正在加载)     已调用send()方法，正在发送请求

  2 - (加载完成)      send()方法执行完成

  3 - (交互)     正在解析响应内容

  4 - (完成)     响应内容解析完成，可以在客户端调用了

xmlhttp.status

**• 200** ＝ 服务器        成功返回网页

• 301 ＝ 客户端所请求的 永久重定向   URL已移走，需要客户端重定向到其他URL

• 302 ＝ 暂时性重定向

• 304  ＝ 客户端所请求的    URL未发生变化

• 400 =  服务器不理解请求的语法

• 403 ＝ 客户端的请求被服务端       禁止

404 = 客户端所请求的URL在服务器端      不存在

500 ＝ 服务端在处理客户端请求时      发生异常

502 = 错误的网关

AJAX的优点

<1>.无刷新更新数据。

<2>.异步与服务器通信

**ajax跨域：禁止一个网站上的js文件访问另一网站上的资源**

（完全是不同的域，主域和子域）解决方案：

**跨域**

**我知道有两种方式：**

**第一种： jsonp（json+padding）：实现原理是利用script标签的src属性可以跨域加载别的域中的文件来实现的**

**假设有一个a.com，在a.com中的一个网页中加入下面的代码：**

**<script src="http://www.b.com/test.php?name=lisi&callback=aaa"></script>**

**在b.com中的test.php中可以接收传递过来的参数name还有callback，然后执行：**

**b.com中执行回调函数的调用echo 'callback(要回传给a.com的数据)'**

**1.jsonP**

**原理：**

所谓同源是指，域名，协议，端口相同。 通过 script 标签 的 src 来实现跨域

这其实就是JSONP的简单实现模式：**创建一个回调函数，然后在远程服务上调用这个函数并且将JSON 数据形式作为参数传递，完成回调**。

**将JSON数据填充进回调函数**，这就是JSONP的JSON+Padding的含义吧。

**第二种： 服务器代理：因为服务器端程序是不存在跨域问题**

**发热吗也是利用src加载跨域的文件**

js继承机制

**我知道一种基于原型的继承机制，举个例子来说：**

**假设有一个ClassA和ClassB，ClassB想继承ClassA**

**首先要在ClassA的构造函数里定义属性，在ClassA的原型里定义方法：**

<!--

function ClassA() {

    this.color = sColor;

}

ClassA.prototype.sayColor = function () {

    alert(this.color);

};

-->

**然后在ClassB的构造函数中使用ClassA.call(this)来继承ClassA中的属性：**

<!--

function ClassB() {

    ClassA.call(this);

}

-->

**再用ClassB.prototype等于ClassA的一个实例对象来继承ClassA中的方法：**

<!--

ClassB.prototype = new ClassA();

ClassB.prototype.constructor = ClassB;

-->

**jquery底层原理：**$函数只是作为一个选择器，接收一些参数，函数里面返回jQuery自定义的对象,jQuery是一个类，下面有原型下的方法，真正的DOM中查找遍历的操作是在构造函数里面进行的，将查到的元素存到this.elements[i]数组里，对数组的元素进行遍历，进行绑定事件。

$(“.div1”).addClass()

$(“#div1”) = new Jquery();

function Jqurey(str){

this.arr = [];

}

Jquery.prototype.addClass = functionn(){

This.arr

}

**bootstarp:**就是一种前端框架，提供了很多样式布局，流式布局，固定布局，js插件，能够快速搭建前端的页面不用重头写

**jQuery有几大特点：**1.jquery的选择器非常强大，除了支持标准的CSS选择器之外，还支持一些自带的伪类选择器，2.提供了大量的方法（API函数），帮我们简化DOM操作，3.还可以支持链式操作（连点的操作，只有方法能链式操作）

jquery的所有操作都是基于$开始的，$在jquery中可以传好几种类型的参数，$支持字符串，代表选择器，返回值是数组；对象，代表选中的DOM对象，代表选择器；函数，代表文档就绪函数。

**$**本身是个函数，接受三种类型的参数，执行完返回三种自定义对象，自定义

$(function(){})文档就绪函数，相当于给window onload绑定事件（53:00）

**函数立即调用**减少全局变量的污染；**文档就绪函数**也是，在函数内部声明的变量不会污染到全局，还要等到body结构全部加载完毕再执行

对一个事件多次绑定addeventlistener（原生的）可以不覆盖

函数return对象是自定义的函数对象（还有DOM对象，内置对象）

自定义对象的创建方式：对象直接量{}，new object

构造函数中不能return

优势：不用太考虑兼容性问题，简化代码

**类**：具有相同的属性和方法的对象的集合

$函数返回值是一个自定义对象

**模块化开发**的加载器：解决代码重用，组件是单独的，没有关联，模块化是有关联的。

**require.js**解决的问题：1.模块化的问题，2.文件的依赖加载问题

**原理：**data-main属性加载入口文件，在底层动态创建script标签，用正则表达式匹配加载的入口文件名，动态的创建标签，动态的加载。

requirejs.org

<script src="require.js"  data-main="xxx.js"></script>

xxx.js是入口文件，页面级的JS，比如index.js，可以不写.js，直接写index

index.js:

require(['a'],function(add2){

alert(add2(4+5));

});

a.js:

define(function(){

function add(a,b){

return a + b;

}

return add

});

模块与函数的区别：模块是粒度更大的代码的抽象和封装

按照升序排序：

arr.sort(function(a,b){

return a-b;

});

a.js:

define(function(){

function sortArray(arr){

return arr.sort(function(a,b){

return a-b;

});

}

return sortArray;

});

function isArray(args){

return (args instanceof )

}

**sea.js**没有第一个参数，没有函数作为参数

define(function(require){

});

var sortArray = require('a');

define(function(require){

});

require.js在2.0以后也支持和sea.js一样的写法

AMD依赖前置require.js

CMD 依赖就近sea.js

**AMD（A:*async*hronous异步模块定义）与CMD（C：common通用模块定义）区别**

Require遵循AMD，seajs遵循CMD

在1.0里面，对于依赖的模块，AMD提前执行，CMD延迟执行，运行到了在加载；CMD依赖就近，AMD依赖前置

Require的原理：底层是动态创建script标签来加载js文件，异步加载，使用的是async（异步的）。底层通过正则解析字符串，找require。

require解决了依赖加载，模块化开发。

动态加载js的原理：var   document.createElement(oScript);

oScript.src=index;

Document.body.appendChild();

动态创建script标签来加载js文件

异步：不用等待。在script里面加上async

Defer和async看上去效果是一样的，defer是IE中支持的语法，在老的IE中支持defer。区别：async，等文件加载完就执行，但是不保证文件的执行顺序，比如123个Js文件同时异步加载，看谁快设先加载；defer加载时可以好几个文件都异步加载，dom结构加载完进行，理论上可以保证文件的加载顺序。

弹层

定义模块用define

想使用一个模块，而自己不是模块，用require

把类实例化对象 new

属性构造函数 方法原型

options = $\_extend({ },options)默认的样式

jquery.js解决的是DOM操作，兼容性的问题;require.js解决的是模块化开发的问题

alert就是弹出窗口，提示信息；confirm是与用户交互，让用户选择，做进一步的处理；prompt用来接收用户的输入。原生弹窗

**angular.js:**

四大核心特性：

1、MVC，module，view，controller

2、模块化和依赖注入，provider

3、双向数据绑定，

4、指令 AEMC（Attribute（属性） Element（元素,标签） Comment（注释） Class）

不写死的数据在controller里

**angular适合写什么样的程序？**

不刷新页面的，

单项数据绑定是模板和数据进行一次合并，然后显示在view里面

双向数据绑定：view中的数据改，model中的数据也改变，反过来也是，而不需要写dom操作(MVVM)。视图和数据是对应的，

<!DOCTYPE html>

<html lang="en" ng-app>

<head>

<meta charset="UTF-8">

<title>Document</title>

</head>

<body>

<div ng-controller="aaa">

<input type="text" ng-model="name">

<div id="div1">

**{{name}}**

</div>

</div>

<script src="angular-1.3.0.js"></script>

<script>

function aaa($scope){

$scope.name = 'wangwu123';//这是input的默认字节

localStorage.name = $scope.name;

setTimeout(function(){

alert(localStorage.name);

},2000);

}

</script>

</body>

</html>

MVC只是手段，**目标**是模块化和复用(很多逻辑是一样的)

**前端MVC的困难：**1、操作DOM的代码必须等待整个页面全部加载完成（js写在底部或者文档就绪函数）2、多个JS文件之间如果出现互相依赖程序员必须自己解决（只加载自己最关心的，类似于require.js）3、JS的原型继承给前端编程带来麻烦

在页面首页如果有数据绑定，用 ng-bind 避免快速刷新显示被匹配的项

$scope:往里面存东西，一个桥梁，建立C和V的链接，存的是数据M

service

先找ng-app,里边所有内容都归angular管，再找ng-

往scope里添加东西相当于往模型里添加数据

directive:

angular需要通过指令去实现

view的复用是通过指令来实现的

#：内部锚点

ng-bind:第一个页面用，剩下的用{{}}，页面不会很丑（突然冒出来双括号）

angular.js中所有的元素dom的操作都必须放在link下

run方法：定义在模块下，页面加载后只运行一次，适合定义全局的，初始化一次就够了

replace

transclude:让指令之间进行嵌套的使用

3，**事件**

事件捕获，子元素先触发，父元素后触发，和事件冒泡相反，从外往里来 true：

标准浏览器中

ele.addEventListener('click',function(){},false// true就是事件捕获)

IE中：不支持事件捕获ele.attachEvent ('onclick',function(){});

事件冒泡，子元素的事件会向上传播，传播到父元素身上。

ele.addEventListener('click',function(){},false//false就是事件冒泡)

**如何取消**事件冒泡：标准浏览器：e.stopPropagation;

IE:window.event.cancelBubble=true;（为了做兼容，如果在前面写明e=e||window.event;IE里面就是e.cancelBubble=true;）

从里往外来 false，浏览器的默认行为，直到document为止

事件冒泡：在一个对象上触发某类事件onclick，那么这个事件会向这个对象的父级对象传播，董兵：211444

从里到外，直至它被处理

利用事件冒泡，解决后生成元素的事件绑定问题，叫做事件委托或者事件代理。

不是所有的事件都能冒泡，例如：blur、focus、load、unload

好处：  节省性能，只在父元素上绑定一个就行了

事件e，事件源target，事件处理函数oDiv.onclick兼容性非常好，所有浏览器都支持，不能多次绑定，只执行最后一次绑定

**添加事件监听：**

标准浏览器：  3个参数

  addEventListener(事件类型click，事件处理function(  e  ) {

e.stopPropagation(); //阻止事件传播，点击子元素就不会冒泡到父元素上了

}, // 事件冒泡false／ 事件捕获true );

IE：2个参数，不支持第3个参数，只能是事件冒泡

   attachEvent(事件类型onclick ， 事件处理函数function(){

  window.event.cancelBubble = true;  //  阻止事件传播，点击子元素就不会冒泡到父元素上了

} ) ;

**移除事件监听：**

  标准浏览器：removeEventListener(事件类型 ,  事件处理函数 ,  false );

   IE：detachEvent(事件类型，事件处理函数);

**阻止默认行为**：  如让a标签失效

不跳转不刷新页面

<a href="javascript:void(0)">删除</a>

<a href="javascript:;" >删除</a>

标准浏览器：

oBtn.onclick = function(  e ){

e.preventDefault();

}

IE：

oBtn.onclick = function(  e ){

window.event.returnValue = false ;

}

e = e || window.event;

**事件源：**

事件源oBtn e.target||window.event.srcElement

事件e

事件处理函数function

1对象

**对象存在就是为了重复使用**， 具有相同属性或方法的对象的集合叫类

，单页面网站和游戏类开发常用面向对象

面向对象编程：

1.介绍JS中面向对象的语法：

JS不是纯面向对象语言，是基于对象的语言，JS中没有对象的概念，是通过函数模拟的。可以用 对象直接量 或new Object的方式创建对象

如果多个对象都有共享一些属性和方法，可以创建类，function 类名，首字母最好大写，

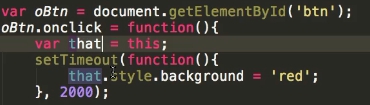
类中用this.属性的方式定义一些

2，闭包：

一个外部函数里面套一个内部函数，内部函数调用外部函数的局部变量，当外部函数执行完之后，变量不会释放；外部函数和内部函数结束的时间不一样，如果外部函数执行完之后，变量不想被释放，优点，变量不会被释放，缺点：会占据内存，手动释放变量。例如：函数立即调用，单击ul弹出li的索引。一个按钮，单击按钮让按钮运动，

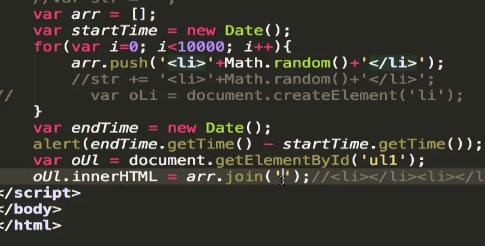
   闭包在表象上是：外部函数声明内部函数，内部函数引用外部函数中的局部变量，这样当外部函数执行完毕时，局部变量不被释放。

例子：有许多个li，单击弹出li的索引。



JS优化

字符串拼接问题，如果频繁的拼接字符串，会导致性能问题，在IE6中有这个问题，利用数组的join函数解决字符串拼接问题，可以解决在IE6中因字符串拼接而不断地开辟新的空间而产生的性能问题，在新的浏览器中已经被解决了。



有一串乱序的数组，排序

exports.sortNumber = function(arr){

return arr.sort(function(a,b){

return a-b;

});

有一串按顺序排列的数组，打乱顺序

exports.sortNumber = function(arr){

return arr.sort(function(a,b){

return 0.5 – Math.random();

});

JS书写规范，函数首字母小写，后面首字母大写，类首字母大写

8，响应式设计

响应式设计：媒体查询，设置页面样式随屏幕宽度的变化而变化（IE8和IE8之前都不支持）

viewport可视窗口和浏览器宽高不一样

<meta name=“viewport” content=“width=device-width//可视窗口浏览器的宽度=设备的宽度（如果不加这句话在所有的移动端设备的宽度都是980）, initial-scale＝1.0, user-scalable=no//不允许用户缩放,maximum-scale=1.0//大的缩放系数, minimum-scale=1.0小的缩放系数”>

媒体查询 media queries：

@media screen:屏幕设备(手机、IPAD之类的)

@media print:打印机

@media projector:投影仪

@media screen and (min-width: 480px) and (max-width: 760px){

body{

响应式设计原则

1.移动优先：在设计的初期就要考虑页面如何在多终端展示

2.渐进式增强，平稳退化：充分发挥硬件设备的最大功能

（对于支持CSS3 feature的浏览器要充分发挥它硬件设备的功能，给他更好的体验，对于低端浏览器使用JS方法hack）

实现响应式布局的方法：

1.CSS3 - Media Query（最简单的方式）

2.借助原生的JS（成本高，不推荐）

3.第三方开源框架 bootstrap（可以很好的浏览器响应式布局的设计）

background：green;

}

}

前端性能优化：

雅虎军规

1. 尽可能的减少http请求数
   1. 合并图片
   2. 合并css
   3. 合并js
2. 使用cdn(内容分发网络)
   1. 什么是cdn?
      1. 在离你最近的地方，放置一台性能好链接顺畅的副本服务器，让你能够以最近的距离，最快的速度获取内容。靠多台服务器找到离你最近的去获取数据，
3. css放在页面最上边，css放到页面顶部head中
   1. css不放在上边：ie中会出现空白 firefox低版本浏览器会出现闪烁
4. 将script放到页面最下边
   1. 页面加载时从上向下的，js写在上边会阻止下边内容的加载
5. 最小化js、css 减少文件体积 压缩js css文件
6. 减少重定向
   1. 用户想访问的页面a被重新指向了页面b,请求了2次，增加了一次请求。
      1. 301永久重定向：用户被请求的页面a被指向了b ,搜索引擎搜索后就会发现，就会将a删掉，直接记住b,使搜索引擎更智能了
      2. 302临时重定向：请求a访问不到的时候，会重新定向到b，
7. 适当使用图片预加载和懒加载去加载图片
8. 移除重复的脚本
9. 减少dom的重排和重绘
   1. 浏览器渲染加载过程 ?
      1. 先是创建一个dom树 ，
      2. 再根据css规则树构建一个渲染树（渲染树里的东西一定是能看到的）
10. 把JavaScript和CSS都放到外部文件

1.减少HTTP请求

基本原理：

在浏览器（客户端）和服务器发生通信时，就已经消耗了大量的时间，尤其是在网络情况比较糟糕的时候，这个问题尤其的突出。

一个正常HTTP请求的流程简述：如在浏览器中输入"<www.xxxxxx.com>"并按下回车，浏览器再与这个URL指向的服务器建立连接，然后浏览器才能向服务器发送请求信息，服务器在接受到请求的信息后再返回相应的信息，浏览器接收到来自服务器的应答信息后，对这些数据解释执行。

而当我们请求的网页文件中有很多图片、CSS、JS甚至音乐等信息时，将会频繁的与服务器建立连接，与释放连接，这必定会造成资源的浪费，且每个HTTP请求都会对服务器和浏览器产生性能负担。

网速相同的条件下，下载一个100KB的图片比下载两个50KB的图片要快。所以，请减少HTTP请求。

解决办法：

合并图片（css sprites精灵），合并CSS和JS文件；图片较多的页面也可以使用 lazyLoad 等技术进行优化。

9.压缩图片和使用图片Spirit技术

基本原理：

注：其实压缩图片和图片精灵是两个方面的技术，可是既然都是关于图片的优化还是放到一块吧。

现在由于工作的细分，专业的前端工程师已经少有机会去切图了，可是关于图片压缩还是得略微了解，一般图片压缩的方式有：

1.缩小图片分辨率；

2.改变图片格式；

3.降低图片保存质量。

关于图片精灵（Spirit）技术就和我们工作直接相关，不管是在CSS中的图片还是在HTML结构中的图片都会产生HTTP请求，前端优化的第一条就是减少请求数，最直接有效的方法是使用图片精灵（CSS Spirit）。图片精灵就是把许多图片放到一张大图片里面，通过CSS来显示图片的一部分。

至于图片精灵的操作细节就不多做介绍了，网上相关内容很多。

3.请减少对DOM的操作

基本原理：

对DOM操作的代价是高昂的，这在网页应用中的通常是一个性能瓶颈。

天生就慢。在《高性能JavaScript》中这么比喻："把DOM看成一个岛屿，把JavaScript（ECMAScript）看成另一个岛屿，两者之间以一座收费桥连接"。所以每次访问DOM都会教一个过桥费，而访问的次数越多，交的费用也就越多。所以一般建议尽量减少过桥次数。

解决办法：

修改和访问DOM元素会造成页面的Repaint和Reflow，循环对DOM操作更是罪恶的行为。所以请合理的使用JavaScript变量储存内容，考虑大量DOM元素中循环的性能开销，在循环结束时一次性写入。

减少对DOM元素的查询和修改，查询时可将其赋值给局部变量。

注：在IE中：hover会降低响应速度。

观可读性强的代码。如果你是纯粹的前台开发人员，一定会非常喜欢JSON。

7.将CSS和JS放到外部文件中引用，CSS放头，JS放尾

放在下面避免影响网页加载

基本原理：

注：这个是很基础且必须遵循的知识点，引入外部文件好处是显而易见的，而且是项目稍稍复杂一点的时候就有必要了这样做了。

易维护、易扩展，方便管理和重复利用。

正确的方式：

JavaScript是浏览器中的霸主，为什么这么说，因为在浏览器在执行JavaScript代码时，不能同时做其它事情，即《script》每次出现都会让页面等待脚本的解析和执行（不论JavaScript是内嵌的还是外链的），JavaScript代码执行完成后，才继续渲染页面。这个也就是JavaScript的阻塞特性。

因为这个阻塞的特点，建议把JavaScript代码放到《/body》标签以前，这样既能有效的防止JavaScript的阻塞，又能使得页面的HTML结构能更快的释放。

HTML规范清楚指出CSS要放包含在页面的《head》区域内，这里就不多解释了。

12.适当使用图片预加载和懒加载

懒加载：当页面被请求时，只加载可视区域的图片，其它部分的图片则不加载，只有这些图片出现在可视区域时才会动态加载这些图片

如何判断是否出现在可视区域内吗？ 那就是你可以获取当前img的相对于文档顶的偏移距离减去scrollTop的距离， 然后和浏览器窗口高度在进行比较，如果小于浏览器窗口则出现在了可视区域内了， 反之，则没有。

预加载：用不用都加载，轮播图的时候先把所有图片都加载出来，可以让用户有更好的体验。

2.请正确理解 Repaint 和 Reflow

注：Repaint 和 Reflow 也就是重绘和重排，

基本原理：

Repaint（重绘）就是在一个元素的外观被改变，但没有改变布局（宽高）的情况下发生，如改变visibility、outline、背景色等等。

Reflow（重排）就是DOM的变化影响到了元素的几何属性（宽和高），浏览器会重新计算元素的几何属性，会使渲染树中受到影响的部分失效，浏览器会验证DOM树上的所有其它结点的visibility属性，这也是Reflow低效的原因。如：改变窗囗大小、改变文字大小、内容的改变、浏览器窗口变化，style属性的改变等等。如果Reflow的过于频繁，CPU使用率就会上涨，所以前端也就有必要知道 Repaint 和 Reflow的知识。

减少性能影响的办法：

上面提到通过设置style属性改变结点样式的话，每设置一次都会导致一次reflow，所以最好通过设置class的方式；　有动画效果的元素，它的position属性应当设为fixed或absolute，这样不会影响其它元素的布局；如果功能需求上不能设置position为fixed或absolute，那么就权衡速度的平滑性。

重排一定会导致重绘，重绘不一定会重排，重排对性能影响更大。

总之，因为 Reflow 有时确实不可避免，所以只能尽可能限制Reflow的影响范围。

浏览器会先创建DOM树，根据CSS规则（解析CSS规则树）创建渲染树，display=none会出现在渲染树当中。

5.高效使用HTML标签和CSS样式

基本原理：

HTML是一门用来描述网页的一种语言，它使用标记标签来描述网页，作为一名合格的前端开发，你有必要去知道其常用标签代表的含义（SEO）和属性（表现形式）。

CSS指层叠样式表 （Cascading Style Sheets），如果说把页面想象成一个人，HTML就是人的骨架，CSS就是人的衣装，一个人的品味从他的衣装就能一目了然。

一名专业的前端开发也是一名优秀的重构，因为在页面中经常会有各种不合理的嵌套和重复定义的CSS样式，我不是要你重构页面，只是希望你在碰到这种情况的时候解决这些问题。如这样的HTML：

 <table><tr><td>

　　<table><tr><td>

　　　　...

　　</td></tr></table>

</td></tr></table>

或者这样的CSS：

body .box .border ul li p strong span{color：#000}

以上都是对HTML和CSS非常糟糕的使用方法。

正确理解：

HTML是一门标记语言，使用合理的HTML标签前你必须了解其属性，比如Flow Elements，Metadata Elements ，Phrasing Elements。比较基础的就是得知道块级元素和内联元素、盒模型、SEO方面的知识。

CSS是用来渲染页面的，也是存在渲染效率的问题。CSS选择符是从右向左进行匹配的，这里对css选择符按照开销从小到大的顺序梳理一下：

ID选择符 #box

类选择符 .box

标签 div

伪类和伪元素 a:hover

当页面被触发引起回流（reflow）的时候，低效的选择符依然会引发更高的开销，所以请避免低效。

6.使用CDN加速（内容分发网络）

基本原理：

CDN的全称是Content Delivery Network，即内容分发网络。

"其基本思路是尽可能避开互联网上有可能影响数据传输速度和稳定性的瓶颈和环节，使内容传输的更快、更稳定。通过在网络各处放置节点服务器所构成的在现有的互联网基础之上的一层智能虚拟网络，CDN系统能够实时地根据网络流量和各节点的连接、负载状况以及到用户的距离和响应时间等综合信息将用户的请求重新导向离用户最近的服务节点上。"

不足之处：

实时性不太好是CDN的致命缺陷。随着对CDN需求的逐渐升温，这一缺陷将得到改进，使来自于远程服务器的网络内容网页与复本服务器或缓存器中的网页保持同步。解决方法是在网络内容发生变化时将新的网络内容从服务器端直接传送到缓存器，或者当对网络内容的访问增加时将数据源服务器的网络内容尽可能实时地复制到缓存服务器。

8.精简CSS和JS文件

基本原理：

有一条非常重要的准则一直没有提到，就是CSS和JavaScript的压缩，直接减少下载的文件体积。我个人经常使用的方式是使用 YUI Compressor，它的特点是：移除注释；移除额外的空格；细微优化；标识符替换。

YUI Compressor是java程序，如果你对java很熟悉的话可快速的上手使用yuicompressor.jar；如果你对java很陌生也没关系，一样可以使用YUI Compressor，下面介绍其使用方式。

YUI Compressor的配置和使用：

先配置使用环境：

1.先确保电脑中是否安装了JDK

2.再配置必要的环境变量（细节不能三两句说清，所以不知道如何设置还是搜索吧）

3.在cmd界面，输入javac可测试是否安装成功

使用方法可从cmd到进入yuicompressor.jar所在磁盘，我以自己的yuicompressor-2.4.2.jar为例：

1.压缩JS

java -jar yuicompressor-2.4.2.jar api.js > api.min.js

2.压缩CSS

java -jar yuicompressor-2.4.2.jar style.css > style.min.css

当然，还有另一种更傻瓜式的使用方式，赶兴趣的朋友自己可去多尝试下。

10.注意控制Cookie大小和污染

基本原理和使用方法：

有关Cookie的基础和高级知识可以去看本人写过的一篇文章《JavaScript 操作 Cookie》。

因为Cookie是本地的磁盘文件，每次浏览器都会去读取相应的Cookie，所以建议去除不必要的Coockie，使Coockie体积尽量小以减少对用户响应的影响；

使用Cookie跨域操作时注意在适应级别的域名上设置coockie以便使子域名不受其影响

Cookie是有生命周期的，所以请注意设置合理的过期时间，合理地Expire时间和不要过早去清除coockie，都会改善用户的响应时间。

11．避免重定向

曾经有的资源，原来的地址被转移到了新的地址，如果不重定向，会404找不到，php里面header，location（具体的），重定向发送两次请求，

301永久重定向，302临时重定向，先去别的服务器下面进行配置，然后还能回到原来的地址。

13.适当的使用CSS3动画代替传统的定位

CSS3动画会开启GPU加速

利用事件委托为相同的子元素事件绑定，委托给父元素

**HTTP底层协议**

在浏览器地址栏里输入一个地址，按回车发生了什么？

将客户端浏览器的“请求request“发送给服务器端，服务器端处理请求，

将”响应response“再返回给客户端浏览器。

详细解释：

URL按回车之后，**浏览器会将你输入的域名发送到DNS服务器上**(进行动态域名解析,该服务器在你买域名的地方)，**DNS服务器将进行域名解析，解析成对应的IP地址，**浏览器与远程 Web 服务器通过 TCP 三次握手协商来建立一个 TCP/IP 连接。

一旦 TCP/IP 连接建立，浏览器会通过该连接向远程服务器发送 HTTP 的 GET 请求。远程服务器找到资源并使用 HTTP 响应返回该资源，值为 200 的 HTTP 响应状态表示一个正确的响应。

进入到一个大主机上，该主机上有一个apache web server，他的端口号为80

通过这个80端口号标识，由apache web server处理这个请求。

apache打开myblog/test.php文件，apache认识html语法，

但不认识php语法，**apache就加载php模块，解析php这段代码，**

**并将得到的结果放在原php位置**。然后apache将处理完的结果，

当成一个大字符串返回给客户端浏览器，浏览器将html语法解释执行，

简单来说，浏览器会解析 HTML 生成 DOM Tree，其次会根据 CSS 生成 CSS Rule Tree，

而 javascript脚本 又根据 DOM API 操作 DOM Tree 和 CSS Rule Tree

将结果显示在页面上。

php文件是在服务器端运行的。(mysql默认端口号3306，tomcat默认端口号8080，默认端口号可以改）

当你写80这个端口号时，该服务器直接指定apache处理，

当你写3306这个端口号时，该服务器直接指定mysql处理)

html语法和php语法写在一起叫混编方式。

HTTP   TCP／IP

TCP三次握手：建立连接

 (1)  Client端向Server端发送一个SYN包，请求建立连接。

(2)Server端收到SYN包后，会发送一个SYN/ACK确认包回去，表示对第一个SYN包的确认。

(3)Client端收到SYN/ACK包后，会发送一个ACK确认包，通知Server连接已建立。

TCP四次挥手：关闭连接

(1) Client端向Server端FIN报文，请求关闭连接

(2) Server端发送ACK，表示收到请求。但是Server端还有数据没有发送完成，则不必着急关闭Socket，可以继续发送数据。这时Client端进入FIN\_WAIT状态，等待Server端的FIN报文。

(3)当Server端确定数据已发送完成，则向Client端发送FIN报文，表明可以关闭连接了

(4)Client端收到FIN报文后，知道可以关闭连接了，再发送ACK，后进入TIME\_WAIT状态，Client端等待了2MSL(最大报文生存时间)后依然没有收到回复，则证明Server端已正常关闭，然后Client端也可以关闭连接了，进入Closed状态。

Server端收到ACK后，就关闭连接了。如果Server端没有收到ACK，那么将重发FIN。

1. TCP建立连接为什么需要三次握手，两次不行吗？

答：在谢希仁著《计算机网络》第四版中讲“三次握手”的目的是“为了防止已失效的连接请求报文段突然又传送到了服务端，因而产生错误”。

“已失效的连接请求报文段”的产生在这样一种情况下：client发出的第一个连接请求报文段并没有丢失，而是在某个网络结点长时间的滞留了，以致延误到连接释放以后的某个时间才到达server，本来这是一个早已失效的报文段。但server收到此失效的连接请求报文段后，就误认为是client再次发出的一个新的连接请求。于是就向client发出确认报文段，同意建立连接。

假设不采用“三次握手”，那么只要server发出确认，新的连接就建立了。由于现在client并没有发出建立连接的请求，因此不会理睬server的确认，也不会向server发送数据。但server却以为新的运输连接已经建立，并一直等待client发来数据。这样，server的很多资源就白白浪费掉了。采用“三次握手”的办法可以防止上述现象发生。例如刚才那种情况，client不会向server的确认发出确认。server由于收不到确认，就知道client并没有要求建立连接。”

为什么建立连接需要3次，而关闭连接需要4次？

关闭连接时，当Server端收到FIN报文时，很可能并不会立即关闭SOCKET，所以只能先回复一个ACK报文，。只有等到Server端所有的报文都发送完了，才能发送FIN报文，ACK和FIN不能一起发送，所以多一次。

SEQ数据包本身的序列号

SYN 同步序号，SYN报文是用来同步的

ACK 确认帧，ACK报文是用来应答的

握手阶段：

序号  方向    seq       ack

1　　A->B   10000    0

2       B->A    20000   10000+1=10001

3       A->B    10001   20000+1=20001

解释：

1：A向B发起连接请求，以一个随机数初始化A的seq,这里假设为10000，此时ACK＝0

2：B收到A的连接请求后，也以一个随机数初始化B的seq，这里假设为20000，意思是：你的请求我已收到，我这方的数据流就从这个数开始。B的ACK是A的seq加1，即10000＋1＝10001

3：A收到B的回复后，它的seq是它的上个请求的seq加1，即10000＋1＝10001，意思也是：你的回复我收到了，我这方的数据流就从这个数开始。A此时的ACK是B的seq加1，即20000+1=20001

TPC／UDP协议是**传输层**协议，有状态，主要解决数据如何在网络中传输。

HTTP是**应用层**协议，无状态，主要解决如何包装数据。

最显著的特点是客户端发送的每次请求都需要服务器回送响应，在请求结束后，会主动释放连接。从建立连接到关闭连接的过程称为“一次连接”。

Socket是对TCP的封装，它相当于一个调用接口，通过Socket，使用TCP协议。

UDP和TCP协议的主要区别:

TCP面向连接的，在正式收发数据前，必须和对方建立可靠的连接。

UDP面向非连接的，它不与对方建立连接，而是直接就把数据包发送过去。 适用于一次只传送少量数据、对可靠性要求不高的应用环境

TCP                    UDP

面向连接             面向非连接

可靠                    不可靠

传输大量数据      少量数据

慢                        快

TCP/IP协议栈分为四层:  应用层（http，［FTP、HTTP、TELNET］，［SNMP、TFTP、NTP］）

   传输层（TCP，UDP）

   互联层（IP，ICMPInternet控制报文协议Internet Control Message Protocol）

   网络接口层（ARP地址解析协议Address Resolution Protocol，RARP反向地址转换协议 Reverse ）

OSI参考模型：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层

post，get，put，delete，head，options

1.使用GET时，数据作为URL的一部分，在地址栏中显示，不太安全。

2.get传送的数据量较小，不能大于2KB。

   post传送的数据量较大，一般被默认为不受限制。

3.使用POST时，数据集会被编码到http协议的body中，构造成消息发送。

多次请求都返回同样的结果时使用GET，而当请求会改变服务器数据或状态时使用POST

scheme://host.domain:port/path/filename

scheme -   协议类型

host -       主机名     （http 的默认主机是 www）

domain -   域名

:port -       端口号       主机上的端口号  （http 的默认端口号是 80）

path -       路径           服务器上的路径 （如果省略，则文档必须位于网站的根目录中）

filename - 文件名

http 超文本传输协议           以http:// 开头的普通网页。不加密。

https 安全超文本传输协议       安全网页。加密所有信息交换。

ftp   文件传输协议           用于将文件下载或上传至网站。

file                             您计算机上的文件。

lidanni.com是顶级域名，也是一级域名

www.lidanni.com和host.lidanni.com

同为二级域名，两者是平等关系，只是人们习惯使用www为前缀的二级域名作为网站的主域名入口

Bom中的location.href等可以获得URL

Dom中的Global中有decodeURI(), encodeURI()

XSS 跨站脚本攻击： 通过插入恶意脚本，实现对用户浏览器的控制。

(1)非持久型(一次性)：用户访问一个被攻击者篡改后的链接，被植入的攻击脚本被用户浏览器执行，达到攻击目的。

(2)持久型：将混有恶意代码的数据存在服务器中，用户访问时触发。

**类，对象，继承机制**

**JS高级教程，继承机制**

**对象，new object()/{}/Object.creat({name:'lisi'})继承的方式创建对象**

**类：具有相同的属性和方法的对象的集合：**

**function Cat(){**

**this.name = name;**

**}**

**Cat.prototype.eat = function(){//特权方法,避免函数对象调用多次**

**构造函数 类里面定义属性，原型定义方法**

**<!-- 原型只是用来定义方法的，但是this是new出来的对象 -->**

**This是对当前这个对象的引用**

**alert(this.name +'is eating');**

**};**

**var cat1 = new Cat('mimi');**

**cat1.eat();**

**原型是用来继承，当在访问具体的实例下的方法时，先找有没有方法，没有向上找，一直找到object，再没有就undefinded,原型链**

**原型**其实是为了解决js里面没有类这个概念，使用函数模拟类，创建类的时候把一些属性方法放在这个类下面，下面基于这个类创建的实例化对象就有了这些属性方法。

**属性放在构造函数，方法在原型**

**function ClassA(color){**

**this.color = color;**

**}**

**ClassA.prototype.sayColor = function(){//公有方法**

**alert(this.color);**

**};**

**function ClassB(){**

**ClassA.call(this);//call函数前面必须是函数**

**}**

**ClassB.prototype = new ClassA();**

**ClassA.prototype.constructor = ClassB;**

**var o1 = new ClassB();**

**什么时候会用到类，多个属性和方法会用到类**

function  ClassA(sColor){

this.color = sColor;

this.sayColor = function(){

alert(this.color);

};

}

function ClassB(sColor){

}

var o = new ClassB();

怎样能使o.color和o.sayColor能调用color和sayColor

对象冒充(继承属性)：在ClassB(sColor)里面

this.fn = ClassA;

this.fn(sColor);

delete this.fn;

(this是new出来的对象)

或者可以在ClassB(sColor)里面写ClassA.call(this);

call函数允许this改变指针

aa.call({name:'lisi'},'haha','xixi');如果本来的aa有参数，则从第二个参数开始打印，如果没有把{}也打出来

aa.=apply({name:'lisi'},['haha',]);

call函数好多属性，apply两个属性，第一个参数改变this指针，第二个添加属性

为了避免内存的浪费，把函数的声明放在原型下面

call的方式不能继承原型下的方法，想要继承，ClassB.prototype = new ClassA();

原型的形式继承方法

任意对象下面都有一个默认的属性，constructor，返回创建这个对象的构造函数

new之后继承原型里的

在js中，基于原型做继承，首先要继承先有类，属性放在构造函数里面，方法放在原型里

封装继承的方法：

function Animal(name,age){

this.name = name;

this.age = age;

}

Animal.prototype.say = function(){

alert(this.name);

};

function extend(Cls){

function Fn(){

Cls.apply(this,arguments);

}

Fn.prototype = new Cls();

Fn.prototype.constructor = Fn;

return Fn;

}

var Cat = extend(Animal);

Cat.prototype.catchMouse = function(){

};

var cat1 = new Cat();

cat1.say();

函数名.length得到的是形参的个数

arguments.length得到的是实参的个数

形参是函数里定义的参数，实参是被赋值的形参，能参加运算

特权方法，自己特有的方法；公有方法，原型里面的方法，所有的对象都继承。私有方法（其实没有这个定义），在函数内部，用var关键字声明的函数。

\_\_proto\_\_:就是私有属性

function Animal(name){

this.name = name;

}

Animal.prototype.eat = function(){

};

var a1 = new Animal();

a1.\_\_proto\_\_ = Animal.prototype;

先继承属性

function Cat(name){

Animal.call(this,name);

}

再继承方法

Cat.prototype = new Animal();

Cat.prototype.constructor = Cat;

var cat1 = new Cat();

构造函数中的属性在方法中可以通过this访问

node.js+express

node.js是一个平台，不是语言(所用的语言是javascript) 和express是所应用的框架

node.js是一个服务器端的程序

node.js速度更快，可以解决多人同时请求问题，*是单线程异步非阻塞式的方式，就是*

基于异步回调的方式解决多请求问题

node.js启动：

命令行启动：用npm start启动（）

直接启动：在地址栏输入地址回车之后，向服务器发送了一个请求，（入口文件是app.js,根据路径在app.js里面查找）进到路由里面，通过响应res.render渲染view下面的文件，

并且可以传数据，将数据存到session中*在，view中，（可以利用<%= %>）接收数据*

script和<%= %>数据不可共享，<%= %>中的数据是通过映射了一个请求到服务器端，

服务器端解析<%= %>里面的代码，把解析的结果输出，再把结果返回到客户端，客户

端再进行渲染

如果在地址栏直接输入地址，（localhost:3000是根路径），从客户端发送一个请求

到服务器端，服务器端是通过routes下面的 index.js进行处理的

我在这个小例子当中用的存数据的方式是session，关闭浏览器数据就会没有了

路由就是一种做转发的功能

可以做服务器端的运行程序，还可以做命令行的环境，在命令行输入node，还可以在

软件上线之前进行文件压缩，文件合并，比如吧CSS JS合并成一个文件，把JS文件压

缩成.min文件，都可以基于 node.js进行编译

首先，模块化开发，模块：module

commonJS是一套标准而不是技术，使JS能实现浏览器以外的功能

exports.sortNumber = function(arr){

return arr.sort(function(a,b){

return a-b;

});

};//exports暴露，在exports下面添加属性方法，都会暴露给调用函数的对象

socket.io是一个web应用，为实时应用提供跨平台实时通信的库。

不是所有的浏览器都支持WebSocket，支持降级功能。

emit用来发射一个事件或者说触发一个事件，第一个参数是事件名，第二个参数是发送的数据

on用来监听emit发射的事件，第一个参数是监听的事件名，第二个参数是事件处理函数，就是接到这个数据要怎么处理

socket.io有三种默认的事件，connect/message/disconnect，客户端服务器端都有。

socket.emit()向建立该连接的客户端广播(发送)

socket.broadcast.emit()向除去建立该连接的客户端的所有客户

io.sockets.emit()向所有客户端

每登录一个用户，就把用户名当做对象中的属性，

**http协议**

**在浏览器地址栏里输入一个地址，按回车发生了什么？**

将客户端浏览器的“请求request“发送给服务器端，服务器端处理请求，

将”响应response“再返回给客户端浏览器。

详细解释：

URL按回车之后，**浏览器会将你输入的域名发送到DNS服务器上**(进行动态域名解析,该服务器在你买域名的地方)，**DNS服务器将进行域名解析，解析成对应的IP地址，**浏览器与远程 Web 服务器通过 TCP 三次握手协商来建立一个 TCP/IP 连接。

一旦 TCP/IP 连接建立，浏览器会通过该连接向远程服务器发送 HTTP 的 GET 请求。远程服务器找到资源并使用 HTTP 响应返回该资源，值为 200 的 HTTP 响应状态表示一个正确的响应。

进入到一个大主机上，该主机上有一个apache web server，他的端口号为80

通过这个80端口号标识，由apache web server处理这个请求。

apache打开myblog/test.php文件，apache认识html语法，

但不认识php语法，**apache就加载php模块，解析php这段代码，**

**并将得到的结果放在原php位置**。然后apache将处理完的结果，

当成一个大字符串返回给客户端浏览器，浏览器将html语法解释执行，

简单来说，浏览器会解析 HTML 生成 DOM Tree，其次会根据 CSS 生成 CSS Rule Tree，

而 javascript脚本 又根据 DOM API 操作 DOM Tree 和 CSS Rule Tree

将结果显示在页面上。

php文件是在服务器端运行的。(mysql默认端口号3306，tomcat默认端口号8080，默认端口号可以改）

当你写80这个端口号时，该服务器直接指定apache处理，

当你写3306这个端口号时，该服务器直接指定mysql处理)

html语法和php语法写在一起叫混编方式。

HTTP   TCP／IP

TCP三次握手：建立连接

 (1)  Client端向Server端发送一个SYN包，请求建立连接。

(2)Server端收到SYN包后，会发送一个SYN/ACK确认包回去，表示对第一个SYN包的确认。

(3)Client端收到SYN/ACK包后，会发送一个ACK确认包，通知Server连接已建立。

TCP四次挥手：关闭连接

(1) Client端向Server端FIN报文，请求关闭连接

(2) Server端发送ACK，表示收到请求。但是Server端还有数据没有发送完成，则不必着急关闭Socket，可以继续发送数据。这时Client端进入FIN\_WAIT状态，等待Server端的FIN报文。

(3)当Server端确定数据已发送完成，则向Client端发送FIN报文，表明可以关闭连接了

(4)Client端收到FIN报文后，知道可以关闭连接了，再发送ACK，后进入TIME\_WAIT状态，Client端等待了2MSL(最大报文生存时间)后依然没有收到回复，则证明Server端已正常关闭，然后Client端也可以关闭连接了，进入Closed状态。

Server端收到ACK后，就关闭连接了。如果Server端没有收到ACK，那么将重发FIN。

1. TCP建立连接为什么需要三次握手，两次不行吗？

答：在谢希仁著《计算机网络》第四版中讲“三次握手”的目的是“为了防止已失效的连接请求报文段突然又传送到了服务端，因而产生错误”。

“已失效的连接请求报文段”的产生在这样一种情况下：client发出的第一个连接请求报文段并没有丢失，而是在某个网络结点长时间的滞留了，以致延误到连接释放以后的某个时间才到达server，本来这是一个早已失效的报文段。但server收到此失效的连接请求报文段后，就误认为是client再次发出的一个新的连接请求。于是就向client发出确认报文段，同意建立连接。

假设不采用“三次握手”，那么只要server发出确认，新的连接就建立了。由于现在client并没有发出建立连接的请求，因此不会理睬server的确认，也不会向server发送数据。但server却以为新的运输连接已经建立，并一直等待client发来数据。这样，server的很多资源就白白浪费掉了。采用“三次握手”的办法可以防止上述现象发生。例如刚才那种情况，client不会向server的确认发出确认。server由于收不到确认，就知道client并没有要求建立连接。”

为什么建立连接需要3次，而关闭连接需要4次？

关闭连接时，当Server端收到FIN报文时，很可能并不会立即关闭SOCKET，所以只能先回复一个ACK报文，。只有等到Server端所有的报文都发送完了，才能发送FIN报文，ACK和FIN不能一起发送，所以多一次。

SEQ数据包本身的序列号

SYN 同步序号，SYN报文是用来同步的

ACK 确认帧，ACK报文是用来应答的

握手阶段：

序号  方向    seq       ack

1　　A->B   10000    0

2       B->A    20000   10000+1=10001

3       A->B    10001   20000+1=20001

解释：

1：A向B发起连接请求，以一个随机数初始化A的seq,这里假设为10000，此时ACK＝0

2：B收到A的连接请求后，也以一个随机数初始化B的seq，这里假设为20000，意思是：你的请求我已收到，我这方的数据流就从这个数开始。B的ACK是A的seq加1，即10000＋1＝10001

3：A收到B的回复后，它的seq是它的上个请求的seq加1，即10000＋1＝10001，意思也是：你的回复我收到了，我这方的数据流就从这个数开始。A此时的ACK是B的seq加1，即20000+1=20001

TPC／UDP协议是**传输层**协议，有状态，主要解决数据如何在网络中传输。

HTTP是**应用层**协议，无状态，主要解决如何包装数据。

最显著的特点是客户端发送的每次请求都需要服务器回送响应，在请求结束后，会主动释放连接。从建立连接到关闭连接的过程称为“一次连接”。

Socket是对TCP的封装，它相当于一个调用接口，通过Socket，使用TCP协议。

UDP和TCP协议的主要区别:

TCP面向连接的，在正式收发数据前，必须和对方建立可靠的连接。

UDP面向非连接的，它不与对方建立连接，而是直接就把数据包发送过去。 适用于一次只传送少量数据、对可靠性要求不高的应用环境

TCP                UDP

面向连接             面向非连接

可靠                不可靠

传输大量数据       少量数据

慢                  快

TCP/IP协议栈分为四层:  应用层（http，［FTP、HTTP、TELNET］，［SNMP、TFTP、NTP］）

    传输层（TCP，UDP）

    互联层（IP，ICMPInternet控制报文协议Internet Control Message Protocol）

    网络接口层（ARP地址解析协议Address Resolution Protocol，RARP反向地址转换协议 Reverse ）

OSI参考模型：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层

post，get，put，delete，head，options

1.使用GET时，数据作为URL的一部分，在地址栏中显示，不太安全。

2.get传送的数据量较小，不能大于2KB。

   post传送的数据量较大，一般被默认为不受限制。

3.使用POST时，数据集会被编码到http协议的header中，构造成消息发送。

多次请求都返回同样的结果时使用GET，而当请求会改变服务器数据或状态时使用POST

scheme://host.domain:port/path/filename

scheme -   协议类型

host -    主机名     （http 的默认主机是 www）

domain -   域名

:port -   端口号       主机上的端号  （http 的默认端口号是 80）

path -    路径           服务器上的径 （如果省略，则文档必须位于网站的根目录中）

filename - 文件名

http 超文本传输协议    以http:// 开头的普通网页。不加密。

https 安全超文本传输协议   安全网页。加密所有信息交换。

ftp   文件传输协议     用于将文件下载或上传至网站。

file   您计算机上的文件。

lidanni.com是顶级域名，也是一级域名

www.lidanni.com和host.lidanni.com

同为二级域名，两者是平等关系，只是人们习惯使用www为前缀的二级域名作为网站的主域名入口

Bom中的location.href等可以获得URL

Dom中的Global中有decodeURI(), encodeURI()

XSS 跨站脚本攻击： 通过插入恶意脚本，实现对用户浏览器的控制。

(1)非持久型(一次性)：用户访问一个被攻击者篡改后的链接，被植入的攻击脚本被用户浏览器执行，达到攻击目的。

(2)持久型：将混有恶意代码的数据存在服务器中，用户访问时触发。

call和apply区别

call和apply都可以用来改变函数里的this指向，**区别**是

call从第二个参数开始有多个参数，用来代表函数的实参

apply第二个参数是一个数组，数组中的每一个元素对应一个实参

<!--

function aa(name, age){

alert(this)

}

aa.call({name: 'lisi'}, 'lisi', 23);

aa.apply({name: 'lisi'}, ['lisi', 23]);

 -->

this指向

一个函数是直接调用的，this指向window。

一个普通函数里面的this指向的是window。

Call函数改变this指针。

call和apply的区别：aa.call({name:”lisi”},’lisi’,12);

aa.apply({name:”lisi”},[’lisi’,12]);

call和apply都执行函数，bind只是改变this的指针，而不执行。

aa.bind({name:’lisi’}).call();

Echarts