**HTML**

**1. 标签的分类**

**1. 标签表示一个元素**

**2. 按性质划分: Block-Level和Inline-Level**

**3. 按语义划分:**

* **Headings**: h1, h2, h3, h4, h5, h6
* **Paragraphs**: p
* **Text Formatting**: em, strong, sub, del, ins, small
* **Lists**: ul, li, ol, dl, dt, dd
* **Tables**: table, thead, tbody, tr, th, td
* **Forms and Input**: form, input, select, textarea
* **Others**: div, span, a, img, <!---->
* **HTML5**: header, footer, article, section

**2. XHTML**

XHTML 于2000年的1月26日成为 W3C 标准。W3C 将 XHTML 定义为最新的HTML版本。XHTML 将逐渐取代 HTML。XHTML是通过把 HTML 和 XML 各自的长处加以结合形成的。XHTML 语法规则如下：

* 属性名和标签名称必须小写
* 属性值必须加引号
* 属性不能简写
* 用 Id 属性代替 name 属性
* XHTML 元素必须被正确地嵌套
* XHTML 元素必须被关闭

**3. 标签的语义化**

为表达语义而标记文档，而不是为了样式，结构良好的文档可以向浏览器传达尽可能多的语义，不论是浏览器位于掌上电脑还是时髦的桌面图形浏览器。结构良好的文档都能向用户传达可视化的语义，即使是在老的浏览器，或是在被用户关闭了 CSS 的现代浏览器中。同时结构良好的HTML代码也有助于搜索引擎索引你的网站。

* 不要使用table布局，table是用来表格显示的。
* 不要到处滥用div标签，div是用来分块用的。
* 不要使用样式标签，如font, center, big, small, b, i，样式可以用CSS来控制，b和i可以用strong和em来代替。
* 不要使用换行标签<br />和空格来控制样式，请用CSS。
* 尽量不要使用内联CSS

**CSS**

**1. CSS基础知识**

* 层叠和继承
* 优先级
* 盒模型
* 定位
* 浮动

**2. CSS进阶**

* CSS Sprite
* 浏览器兼容性
* IE HasLayout和Block Format Content
* CSS Frameworks
* CSS3
* CSS性能优化
* LESS and SASS

**CSS Sprite**

CSS Sprite主要用于前端性能优化的一种技术，原理是通过将多张背景图片合成在一张图片上从而减少HTTP请求，加快载入速度。

**浏览器兼容性**

绝大部分情况下我们需要考虑浏览器的兼容性，目前正在使用的浏览器版本非常多，IE6, IE7, IE8, IE9, IE10, Chrome, Firefox, Safari。

**IE HasLayout和Block Format Content**

IE HasLayout是一个 Internet Explorer for Windows的私有概念，它决定了一个元素如何显示以及约束其包含的内容、如何与其他元素交互和建立联系、如何响应和传递应用程序事件、用户事件等。这种渲染特性可以通过某些 CSS 属性被不可逆转地触发。而有些 HTML 元素则默认就具有”layout”。目前只有IE6和IE7有这个概率。BFC是 W3C CSS 2.1 规范中的一个概念，它决定了元素如何对其内容进行定位，以及与其他元素的关系和相互作用。这个其实和浏览器的兼容性有关，因为绝大部分的兼容性问题都是它们引起的。参考：[CSS BFC和IE Haslayout介绍](http://blog.csdn.net/borishuai/article/details/8127758)

**CSS Framework**

CSS框架是一系列CSS文件的集合体，包含了基本的元素重置，页面排版、网格布局、表单样式、通用规则等代码块,用于简化web前端开发的工作，提高工作效率。目前常见框架有：

* [960 Grid System](http://960.gs/)
* [Blueprint CSS](http://code.google.com/p/blueprintcss/)
* [Bluetrip](http://bluetrip.org/)
* [Minimum Page](http://www.minimumpage.com/)

还是一个比较出名和特殊的框架是Twitter的[Bootstrap](https://github.com/twitter)。Bootstrap是快速开发Web应用程序的前端工具包。它是一个CSS和HTML的集合，它使用了最新的浏览器技术，给你的Web开发提供了时尚的版式，表单，buttons，表格，网格系统等等。它是基于Less开发的。不支持IE6，在IE7和IE8里效果也不咋地。

**CSS3**

虽然CSS3还没有正式成为标准，但是包括IE9+, chrome, Firefox等现代浏览器都支持CSS3。CSS提供了好多以前需要用JavaScript和切图才能搞定的功能，目前主要功能有：

* 圆角
* 多背景
* @font-face
* 动画与渐变
* 渐变色
* Box阴影
* RGBa-加入透明色
* 文字阴影

**CSS性能优化**

CSS 代码是控制页面显示样式与效果的最直接“工具”，但是在性能调优时他们通常被 Web 开发工程师所忽略，而事实上不规范的 CSS 会对页面渲染的效率有严重影响，尤其是对于结构复杂的 Web 2.0 页面，这种影响更是不可磨灭。所以，写出规范的、高性能的 CSS 代码会极大的提高应用程序的效率。参考[CSS性能优化探讨](http://blog.csdn.net/borishuai/article/details/8671929)

**LESS和SASS**

[LESS](http://lesscss.org/)和[SASS](http://sass-lang.com/)都是 CSS 预处理器，用来为 CSS 增加一些编程的的特性，无需考虑浏览器的兼容性问题，例如你可以在 CSS 中使用变量、简单的程序逻辑、函数等等在编程语言中的一些基本技巧，可以让你的 CSS 更见简洁，适应性更强，代码更直观等诸多好处。

[SASS](http://sass-lang.com/)基于Ruby开发。[LESS](http://lesscss.org/)既可以在客户端运行，也可以借助Node.js或者Rhino在服务端运行。

**JavaScript**

**1. JavaScript基础知识**

* 数据类型
* 变量
* 表达式与运算符
* 控制语句
* 函数
* 异常
* OO
* 事件
* BOM
* 闭包

**2. JavaScript进阶**

* DOM
* JSON
* AJAX
* JavaScript Frameworks
* HTML5
* 前端模板
* 前端MVC
* 模块化开发
* JavaScript单元测试
* JavaScript设计模式
* NodeJS
* ES5

**DOM**

DOM即文档对象模型，HTML DOM 定义了访问和操作HTML文档的标准方法。几乎所有的现代浏览器都能很好的支持DOM了。

**JSON**

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式，易于人阅读和编写，同时也易于机器解析和生成。是目前事实上数据交换的标准格式，几乎所有语言都支持JSON，比XML强太多了。

**AJAX**

AJAX即“Asynchronous JavaScript and XML”（异步JavaScript和XML)，AJAX并非缩写词，而是由Jesse James Gaiiett创造的名词，由Google发扬光大。用于创建更好更快以及交互性更强的 Web 应用程序的技术。

**JavaScript Frameworks**

JavaScript Frameworks可以极大简化我们JavaScript编程的工作量，它主要提供了以下几个主要功能： DOM操作，跨浏览器兼容性，以及程序架构。当然像jQuery它本身其实并不是一个框架，它是一个库(lib)。目前主流的框架或库有如下几个：

* [jQuery](http://www.jquery.com/)
* [YUI](http://developer.yahoo.com/yui/)
* [DOJO](http://dojotoolkit.org/)
* [Mootoolos](http://mootools.net/)
* [ExtJS](http://blog.csdn.net/borishuai/article/details/www.sencha.com/)
* [Prototype](http://prototypejs.org/)

以上都是一些重量级的框架或者库，还是小巧的库也是强烈推荐的，比如[Underscore](http://documentcloud.github.com/underscore/)

**HTML5**

HTML5同CSS3类似，即虽然没有成为标准，但是主流的浏览器都支持了。HTML5不是HMTL，虽然也提供了一些新标签，但是它的主要用途还是JavaScript。HTML5主要提供以下功能：

* 本地音频视频播放
* Canvas/SVG
* 地理信息
* 硬件加速
* 本地运行
* 本地存储
* 从桌面拖放文件到浏览器上传
* 语义化标签，Form表单

**前端模板**

前端模板主要是为了解决复杂的数据拼接问题，可以将模板语言转换化为HTML结构，可以大大简化工作量，同时代码的可维护性得到很大的提高。目前比较主流前端模板有：

* [MustCache](http://mustache.github.com/)
* [JsRender](https://github.com/BorisMoore/jsrender)

**前端MVC**

Web应用的功能越来越强，Javascript代码也越来越多，大量的JS代码要以何种架构来组织就成了一个亟待解决的问题，于是就有人把传统的MVC架构移植到前端来解决这些问题。目前主流前端MVC框架主要有以下这些：

* [Backbone.js](http://backbonejs.org/)
* [Spine](http://spinejs.com/)
* [KnockoutJS](http://knockoutjs.com/)
* [YUI](http://yuilibrary.com/)
* [Agility.js](http://agilityjs.com/)
* [Ember.js](http://emberjs.com/)
* [Batman.js](http://batmanjs.org/)
* [AngularJS(Google)](http://angularjs.org/)
* [Dojo](http://dojotoolkit.org/)

[TodoMVC](http://addyosmani.github.com/todomvc/)用上面所有的MVC框架写了同一个示例代码“Todo List”，是个学习对比MVC框架的好地方。

**模块化开发**

其实现在JavaScript模块化是个很热门的东西了，主要特点是“模块化开发，按需加载“。这其中[CommonJS](http://commonjs.org/)组织定义了[AMD](https://github.com/amdjs/amdjs-api/wiki/AMD)的规范用来规范浏览器端的模块定义。[RequireJS](http://requirejs.org/)和[SeaJS](http://seajs.org/docs/)是实现了AMD的两个优秀的框架。详见：<http://www.weakweb.com/articles/341.html>

**JavaScript单元测试**

但是随着单元测试的普及，尤其是敏捷开发的推动，涌现了许多优秀的JavaScript单元测试框架，见详细列表。所有的这些框架基本上都能对Javascript代码进行很好的测试，当然UI部分的代码测试一样比较麻烦，但是我们可以通过精心构造我们的测试代码来测试部分UI代码。主流的测试框架如下：

* [QUnit](http://qunitjs.com/)
* [Jasmine](http://pivotal.github.com/jasmine/)
* [JsTestDriver](http://code.google.com/p/js-test-driver/)

目前jQuery的所有代码都是通过QUnit进行测试的，并且将[测试代码](https://github.com/jquery/jquery/tree/master/test/unit)放在Github上了，大家感兴趣可以参考一下。详见：[JavaScript单元测试框架介绍](http://www.weakweb.com/articles/255.html)

**JavaScript设计模式**

好多设计模式是可以应用于JavaScript的，比如经常用到的事件处理的观察者模式，因此设计模式是提升编码层次的必学技术。

**NodeJS**

[NodeJS](http://nodejs.org/)现在是比较火热的，其最大的贡献就是把JavaScript移植到服务器端了，这样前端和后端就可以使用同样的技术，方便统一开发。而且NodeJS是非阻塞调用的，在特定领域性能是非常强劲的。而且这是前端开发人员进军后台开发的好机会，进而前后端统一开发，但又不用去学习其它后台开发语言。

**ES5**

ES5就是ECMAScript 5，也就是最新的JavaScript规范，对之前的JavaScript作了很多改进，增加了好多新的特性，比如JSON，而且现代主流浏览器都开始支持ES5了，还是非常有必要学习一下的。

**Others**

下面是一些和HTML, CSS, JavaScript没有直接关系，但是对于前端开发同样非常重要的一些技术。

* 响应式设计
* Http1.1
* Web移动开发
* 前端安全
* 跨域处理
* 调试工具
* SEO
* A/B test
* 可用性/可访问性
* 前端流程/部署
* 正则表达式
* 编辑器
* 浏览器插件开发
* 浏览器原理

**响应式设计**

伴随着各种智能设备的流行，响应式设计现在是非常火热。以前做网页只要面向PC机的浏览器，页面直接固定宽度就行，比如960px，而现在通过手机的访问量已经超过PC机，并且设备的尺寸多种多样，未来会更多。在这种背景下，网页支持所有设备进行访问是基本要求了，而响应式设计能很好的解决这些问题。

**Http1.1**

HTTP对于前端开发来说还是很重要的，比如最简单的GET，POST方式，Request/Response 头部，状态码等。

**Web移动开发**

现在移动开发非常非常流行了，而开发方式一般是native的方式或者Web方式，作为前端开发人员来说自然是去学习Web移动开发了。PhoneGap是必学的，前端层面的框架如jQueryMobile, Sencha Touch, jQTouch等都是不错的选择。

**前端安全**

随着前端技术的发展,安全问题已经从服务器悄然来到了每一个用户的的面前，盗取用户数据, 制造恶意的可以自我复制的蠕虫代码,让病毒在用户间传播,使服务器当掉. 更有甚者可能会在用户不知觉得情况下,让用户成为攻击者,这绝对不是骇人听闻。富客户端的应用越来越广，前端的安全问题也随之增多。常见的攻击方法有：

* **XSS，跨站脚本攻击(Cross Site Script)**。它指的是恶意攻击者往Web页面里插入恶意html代码，当用户浏览该页之时，嵌入的恶意html代码会被执行，从而达到恶意用户的特殊目的。
* **CSRF(Cross Site Request Forgery)，跨站点伪造请求**。顾名思义就是 通过伪造连接请求在用户不知情的情况下，让用户以自己的身份来完成攻击者需要达到的一些目的。
* **cookie劫持**，通过获取页面的权限，在页面中写一个简单的到恶意站点的请求，并携带用户的cookie 获取cookie后通过cookie 就可以直以被盗用户的身份登录站点。

**跨域处理**

同源策略规定跨域之间的脚本是隔离的，一个域的脚本不能访问和操作另外一个域的绝大部分属性和方法。所谓的跨域处理就是处于不用域之间的脚步互相调用，目前有很多方法来处理它。

**调试工具**

前端的调试工具很多，比如Firebug，Webkit核心的web inspector, IE的iedeveloper。HTTP相关的fiddler, httpwatch等，还有格式化代码的[jsbeatutifier](http://www.jsbeautifier.org/)，它有助于阅读压缩处理过的JavaScript代码。IETester可以模拟所有的IE版本，是调试IE兼容性的好工具。

**SEO**

前端开发人员很多时候还是需要了解搜索引擎优化的。

**A/B test**

A / B测试的核心就是：确定两个元素或版本（A和B）哪个版本更好，你需要同时实验两个版本。最后，选择最好的版本使用。

**可用性/可访问性**

可用性指的是：产品是否容易上手，用户能否完成任务，效率如何，以及这过程中用户的主观感受可好，是从用户的角度来看产品的质量。可用性好意味着产品质量高，是企业的核心竞争力。

可访问性：上网用户中那些视力受损的人，通过屏幕阅读器使用键盘命令将网页的内容读给他们听。以语义化的HTML（结构和表现相分离的HTML）编写的网页文件，就可以让此类用户更容易导航，且网页文件中的重要信息也更有可能被这些用户找到。

可以通过逐步强化你的网站功能，同时对支持性进行测试。运用“渐进增强”和“平稳退化”原则开发网站。

**正则表达式**

估计绝大部分的编程语言都会用到它，当处理字符串时可以极大的简化你的工作。必学啊。

**编辑器**

人人都有自己喜欢的编辑器，从前端的角度看，Eclipse + Aptana, Notepad++, VIM都是不错的选择。我个人比较喜欢用Notepad++,简洁，快。

**前端流程/部署**

当前端项目比较复杂时，我们就应该考虑引入自动构建，自动化部署等技术了。可以使用JSLint来对JavaScript进行语法检查，用CSSLint或CSS Validator检查CSS语法，用JSMin或YUI Compressor对JavaScript代码进行压缩，可以使用JSDoc/YUIdoc进行文档自动化生成，使用Jasmine/JsTestDriver进行自动化单元测试，可以使用Ant/Maven/Make进行自动构建部署。不过伴随着NodeJS的流行，[Grunt](http://www.gruntjs.org/)，[Bower](http://bower.io/)和[Yeoman](http://yeoman.io/)现在几乎是前端最流行的自动化的项目构建工具了。详见[Web前端开发流程自动化](http://blog.csdn.net/borishuai/article/details/8477514)

**浏览器插件开发**

浏览器是我们的工作平台，在上面开发插件是很有趣的，很多时候也是很有用的。

**浏览器原理**

前端工作绝大部分都是运行在浏览器上面，所以了解浏览器原理有助于更深入的理解各种技术的原理，工作过程。

**沟通能力**

优秀的前端工程师需要具备良好的沟通能力，因为你的工作与很多人的工作息息相关。在任何情况下，前端工程师至少都要满足下列四类客户的需求。

* 1. 产品经理——这些是负责策划应用程序的一群人。他们能够想象出怎样通过应用程序来满足用户需求，以及怎样通过他们设计的模式赚到钱（但愿如此）。一般来说，这些人追求的是丰富的功能。
* 2. UI设计师——这些人负责应用程序的视觉设计和交互模拟。他们关心的是用户对什么敏感、交互的一贯性以及整体的好用性。他们热衷于流畅靓丽但并不容易实现的用户界面。
* 3. 项目经理——这些人负责实际地运行和维护应用程序。项目管理的主要关注点，无外乎正常运行时间（uptime）——应用程序始终正常可用的时间、性能和截止日期。项目经理追求的目标往往是尽量保持事情的简单化，以及不在升级更新时引入新问题。
* 4. 最终用户——当然是应用程序的主要消费者。尽管我们不会经常与最终用户打交道，但他们的反馈意见至关重要；没人想用的应用程序毫无价值。最终用户要求最多的就是对个人有用的功能，以及竞争性产品所具备的功能。

那么，前端工程师应该最关注哪些人的意见呢？答案是所有这四类人。优秀的前端工程师必须知道如何平衡这四类人的需求和预期，然后在此基础上拿出最佳解决方案。由于前端工程师处于与这四类人沟通的交汇点上，因此其沟通能力的重要性不言而喻。如果一个非常酷的新功能因为会影响前端性能，必须删繁就简，你怎么跟产品经理解释？再比如，假设某个设计如果不改回原方案可能会给应用程序造成负面影响，你怎么才能说服UI设计师？作为前端工程师，你必须了解每一类人的想法从何而来，必须能拿出所有各方都能接受的解决方案。从某种意义上说，优秀的前端工程师就像是一位大使，需要时刻抱着外交官的心态来应对每一天的工作。

**如何提高前端技术**

* Github是一个优秀的代码托管网站，我们可以在上创建我们个人的项目，同时也是学习的好地方，我们可以关注其它优秀的项目。JSFiddle是一个web开发人员的练习场，一个可以在很多方面应用的工具。我们可以用他来在线编辑一些HTML,CSS,javascript片段。你编辑的代码可以与其他人分享，或嵌入你的博客等
* 阅读优秀的开源代码
* 参加[前端聚会](http://blog.csdn.net/borishuai/article/details/8529507)
* 关注技术发展趋势，了解最新的行业技术，可以通过订阅知名博客，阅读技术新闻获取
* 写博客/记笔记，可以进行知识积累。博客可以自己买空间或者各大知名博客网站。笔记的话比如evernote，有道笔记等。

**按阶段划分**

如何划分仁者见仁，智者见智，要根据项目的情况作出调整，以下是我根据我自身的经验作出的划分，给大家作个参考。

**第一阶段**

入门，打基础同时能参与到项目中去。

* HMTL & XHTML
* CSS基础知识
* JavaScript基础知识
* DOM
* JSON
* AJAX
* JavaScript Frameworks
* 编辑器

**第二阶段**

掌握前端核心技术，可以独立干活。

* HTML5标签，TML标签语义化
* CSS Sprite
* 浏览器兼容性
* IE HasLayout和Block Format Content
* CSS3
* 精通JavaScript Frameworks
* HTML5
* 前端模板
* 前端MVC
* 模块化开发
* Http1.1
* 调试工具
* 正则表达式
* 响应式设计

**第三阶段**

把握整个前端项目，做整个前端项目的架构师。

* CSS性能优化
* LESS and SASS
* JavaScript单元测试
* JavaScript设计模式
* NodeJS
* ES5
* Web移动开发
* 浏览器插件开发
* 前端安全
* 跨域处理
* SEO
* A/B test
* 可用性/可访问性
* 前端流程/部署
* 浏览器原理

**第四阶段**

一代宗师。

* 不停的学习新的技术
* 交互设计能力，管理能力

**前端工程师未来的路在哪？**

第一条路是走技术流路线，即深入研究前端相关的各项技术，比如浏览器原理，JavaScript本身的研究，W3C各种标准等。前端技术发展很快，各种新技术层出不穷，这条路走下去是很累的。

第二路是往交互设计方向走，前端工程师做到一定程度后交互设计能力也会得到很大的提高，对整个信息架构的把握能力也会更强。可以很好的弥补视觉设计师在交换设计上的不足。这条路其实和第一条是不冲突的，甚至可以理解为同一条路。

第三条路是往后走，即去学习后台开发的技术，比如JAVA/PHP等，其实绝大部分传统的后台开发人员就是这个状态，即前后台都做。个人认为这样的话其实已经走回老路了，毕竟前端就是从原来的后台开发那里独立出来的。

最后一条路就是往管理方向，比如项目经理，或者干脆转行，比如公务员等。