|  |  |
| --- | --- |
| **案卷号** |  |
| **日期** |  |

**Front end Developer Interview Questions**

作 者： bovin.phang@gmail.com

完成日期： 2017/09/18

签 收 人：

签收日期：

**版本历史**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 版本号 | 修改批准人 | 修改人 | 起止日期 | 备注 | 签收人 |
| 1.0 |  | bovin | 09/18/2017 | 创建当前文档 |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

目录

[**Front end Developer Interview Questions** 1](#_Toc497738788)

[1. 序言 7](#_Toc497738789)

[1.1. 文档说明 7](#_Toc497738790)

[2. 常见问题（一些开放性题目） 7](#_Toc497738791)

[2.1. 自我介绍 7](#_Toc497738792)

[2.2. 你觉得前端工程师的价值体现在哪？ 7](#_Toc497738793)

[2.3. 说说你对前端架构师的理解 7](#_Toc497738794)

[2.4. 对前端界面工程师这个职位是怎么样理解的？（如何看待前端开发？）它的前景会怎么样？ 7](#_Toc497738795)

[2.5. 未来的职业规划是怎样的？ 8](#_Toc497738796)

[2.6. 你希望加入一个什么样的团队？ 8](#_Toc497738797)

[2.7. 最后你有什么要问我的吗？ 8](#_Toc497738798)

[2.8. 谈谈你对重构的理解 8](#_Toc497738799)

[2.9. 什么样的前端代码是好的？ 9](#_Toc497738800)

[3. HTML相关问题 9](#_Toc497738801)

[3.1. 标签title与h1的区别、b与strong的区别、i与em的区别？ 9](#_Toc497738802)

[3.2. Doctype作用? 严格模式与混杂模式如何区分？（即标准模式与兼容模式各有什么区别？）它们有何意义? 9](#_Toc497738803)

[3.3. 你知道多少种Doctype文档类型？ 9](#_Toc497738804)

[3.4. HTML5为什么只需要写 <!DOCTYPE HTML>？ 10](#_Toc497738805)

[3.5. HTML与XHTML——二者有什么区别 10](#_Toc497738806)

[3.6. 行内元素有哪些？块级元素有哪些？ 空(void)元素有那些？ 10](#_Toc497738807)

[3.7. html5有哪些新特性、移除了那些元素？如何处理HTML5新标签的浏览器兼容问题？如何区分 HTML 和 HTML5？ 11](#_Toc497738808)

[3.8. HTML5的离线储存怎么使用，工作原理能不能解释一下？ 11](#_Toc497738809)

[3.9. 浏览器是怎么对HTML5的离线储存资源进行管理和加载的呢？ 12](#_Toc497738810)

[3.10. iframe有那些缺点？ 12](#_Toc497738811)

[3.11. Label的作用是什么？是怎么用的？ 12](#_Toc497738812)

[3.12. HTML5的form如何关闭自动完成功能？ 12](#_Toc497738813)

[3.13. 页面可见性（Page Visibility API）可以有哪些用途？ 12](#_Toc497738814)

[3.14. 网页验证码是干嘛的，是为了解决什么安全问题。 12](#_Toc497738815)

[4. CSS相关问题 12](#_Toc497738816)

[4.1. CSS中link 和@import的区别是？ 12](#_Toc497738817)

[4.2. 介绍一下box-sizing属性？ 13](#_Toc497738818)

[4.3. 介绍一下标准的CSS的盒子模型？低版本IE的盒子模型有什么不同的？ 13](#_Toc497738819)

[4.4. CSS 选择符有哪些？哪些属性可以继承？ 13](#_Toc497738820)

[4.5. CSS优先级算法如何计算？ 13](#_Toc497738821)

[4.6. CSS定义的权重 14](#_Toc497738822)

[4.7. CSS3新增伪类有那些？ 14](#_Toc497738823)

[4.8. 如何居中div？如何居中一个浮动元素？如何让绝对定位的div居中？ 14](#_Toc497738824)

[4.9. display有哪些值？说明他们的作用。 15](#_Toc497738825)

[4.10. display:none和visibility:hidden的区别？ 15](#_Toc497738826)

[4.11. display:inline-block 什么时候会显示间隙？ 15](#_Toc497738827)

[4.12. position:absolute和float属性的异同 15](#_Toc497738828)

[4.13. position的值relative和absolute分别是相对于谁进行定位的（即定位原点是什么）？ 16](#_Toc497738829)

[4.14. absolute的containing block(容器块)计算方式跟正常流有什么不同？ 16](#_Toc497738830)

[4.15. CSS3有哪些新特性？ 16](#_Toc497738831)

[4.16. 用纯CSS创建一个三角形的原理是什么？ 17](#_Toc497738832)

[4.17. 如果需要手动写动画，你认为最小时间间隔是多久，为什么？ 17](#_Toc497738833)

[4.18. 经常遇到的浏览器的兼容性有哪些？原因，解决方法是什么，常用hack的技巧？ 17](#_Toc497738834)

[4.19. li与li之间有看不见的空白间隔是什么原因引起的？有什么解决办法？ 18](#_Toc497738835)

[4.20. 为什么要初始化CSS样式。 18](#_Toc497738836)

[4.21. 什么是CSS预处理器/后处理器？ 18](#_Toc497738837)

[4.22. 对BFC规范(块级格式化上下文：block formatting context)的理解？ 18](#_Toc497738838)

[4.23. 解释下浮动和它的工作原理？清除浮动的技巧 18](#_Toc497738839)

[4.24. 浮动元素引起的问题和解决办法？ 19](#_Toc497738840)

[4.25. 简述一下你对HTML语义化的理解？ 19](#_Toc497738841)

[4.26. 什么是 FOUC（无样式内容闪烁）？你如何来避免 FOUC？ 20](#_Toc497738842)

[4.27. 介绍一下你对浏览器内核的理解？常见的浏览器内核有哪些？ 20](#_Toc497738843)

[4.28. 浏览器渲染原理解析 20](#_Toc497738844)

[4.29. 如何在页面上实现一个圆形的可点击区域？ 21](#_Toc497738845)

[4.30. 如何实现不使用 border 画出1px高的线，在不同浏览器的标准模式与怪异模式下都能保持一致的效果。 21](#_Toc497738846)

[5. JS相关问题 21](#_Toc497738847)

[5.1. 介绍js的基本数据类型。 21](#_Toc497738848)

[5.2. 介绍js有哪些内置对象？ 21](#_Toc497738849)

[5.3. JavaScript有几种类型的值？有什么区别？ 21](#_Toc497738850)

[5.4. 写几条JavaScript的基本规范？ 21](#_Toc497738851)

[5.5. javascript对象的几种创建方式 22](#_Toc497738852)

[5.6. javascript继承的6种方法及其优缺点 23](#_Toc497738853)

[5.7. 说说你对闭包的理解，有什么特性？为什么要使用闭包？ 24](#_Toc497738854)

[5.8. 什么是原型？什么是原型链？有什么特点？ 24](#_Toc497738855)

[5.9. instanceof 确定原型和实例之间的关系 25](#_Toc497738856)

[5.10. isPrototypeOf 25](#_Toc497738857)

[5.11. 说说你对作用域链的理解 25](#_Toc497738858)

[5.12. null和undefined的区别？ 25](#_Toc497738859)

[5.13. new操作符具体干了什么呢? 25](#_Toc497738860)

[5.14. call()和apply()的区别和作用？ 26](#_Toc497738861)

[5.15. 如何解决跨域问题 26](#_Toc497738862)

[5.16. DOM操作——怎样添加、移除、移动、复制、创建和查找节点。 27](#_Toc497738863)

[5.17. document.write()的用法 27](#_Toc497738864)

[5.18. 请解释什么是事件代理 27](#_Toc497738865)

[5.19. 渐进增强和优雅降级 27](#_Toc497738866)

[5.20. XML和JSON的区别？ 27](#_Toc497738867)

[5.21. Javascript垃圾回收方法 28](#_Toc497738868)

[5.22. 栈和队列的区别? 28](#_Toc497738869)

[5.23. 栈和堆的区别？ 28](#_Toc497738870)

[5.24. 快速排序的思想并实现一个快排？ 28](#_Toc497738871)

[5.25. 对前端模块化的认识 29](#_Toc497738872)

[5.26. 说说你对AMD和Commonjs的理解 29](#_Toc497738873)

[5.27. 谈谈你对webpack的看法 29](#_Toc497738874)

[5.28. 你觉得jQuery或zepto源码有哪些写的好的地方 30](#_Toc497738875)

[5.29. 说说你对MVC和MVVM的理解，并列举一些MVC和MVVM架构的js框架。 30](#_Toc497738876)

[5.30. 双向绑定和单向数据绑定的优缺点 31](#_Toc497738877)

[5.31. 如何评价AngularJS和BackboneJS 31](#_Toc497738878)

[5.32. react和vue的区别 31](#_Toc497738879)

[5.33. react的优缺点 32](#_Toc497738880)

[5.34. jsx的优缺点 32](#_Toc497738881)

[5.35. 用过哪些设计模式？ 32](#_Toc497738882)

[5.36. ES6的了解 33](#_Toc497738883)

[5.37. js中的 "use strict" 做了什么？说说严格模式的限制？使用严格模式有什么优点和缺点？ 33](#_Toc497738884)

[5.38. 谈一谈let与var和const的区别？ 34](#_Toc497738885)

[5.39. 箭头函数 34](#_Toc497738886)

[5.40. 浏览器本地存储 34](#_Toc497738887)

[5.41. 如何实现浏览器内多个标签页之间的通信? 34](#_Toc497738888)

[5.42. web storage和cookie的区别 35](#_Toc497738889)

[5.43. 请描述一下 cookies，sessionStorage 和 localStorage 的区别 35](#_Toc497738890)

[5.44. 请你谈谈Cookie的弊端 35](#_Toc497738891)

[5.45. 如何删除一个cookie 36](#_Toc497738892)

[5.46. cookie 和session 的区别？ 36](#_Toc497738893)

[5.47. 什么是Cookie 隔离？（或者说：请求资源的时候不要让它带cookie怎么做） 36](#_Toc497738894)

[5.48. Web Worker 和webSocket 36](#_Toc497738895)

[5.49. WEB应用从服务器主动推送Data到客户端有那些方式？区别是什么？ 37](#_Toc497738896)

[5.50. webSocket如何兼容低浏览器？ 37](#_Toc497738897)

[5.51. 一个页面从输入 URL 到页面加载显示完成，这个过程中都发生了什么？ 37](#_Toc497738898)

[5.52. 创建ajax的过程 37](#_Toc497738899)

[5.53. 异步加载和延迟加载 38](#_Toc497738900)

[5.54. js延迟加载的方式有哪些？ 38](#_Toc497738901)

[5.55. defer和async区别 38](#_Toc497738902)

[5.56. ajax的缺点和在IE下的问题？ 38](#_Toc497738903)

[5.57. Flash、Ajax各自的优缺点，在使用中如何取舍？ 38](#_Toc497738904)

[5.58. 说说你对Promise的理解 39](#_Toc497738905)

[5.59. 请解释一下 JavaScript 的同源策略。 39](#_Toc497738906)

[5.60. 为什么要有同源限制？ 39](#_Toc497738907)

[5.61. 常见兼容性问题？ 39](#_Toc497738908)

[5.62. 说说你知道的移动端web的兼容性bug 40](#_Toc497738909)

[5.63. 列举IE 与其他浏览器不一样的特性？ 40](#_Toc497738910)

[5.64. 事件、IE与火狐的事件机制有什么区别？ 如何阻止冒泡？ 41](#_Toc497738911)

[5.65. ie各版本和chrome可以并行下载多少个资源 41](#_Toc497738912)

[5.66. git fetch和git pull的区别 41](#_Toc497738913)

[5.67. attribute和property的区别是什么？ 41](#_Toc497738914)

[5.68. exports和module.exports区别 41](#_Toc497738915)

[5.69. 单线程优点 41](#_Toc497738916)

[5.70. Node.js是如何做到I/O的异步和非阻塞的？ 41](#_Toc497738917)

[5.71. 并行与并发，进程与线程 41](#_Toc497738918)

[5.72. 谈谈Nodejs优缺点 42](#_Toc497738919)

[6. 测试相关问题 42](#_Toc497738920)

[6.1. 单元测试/集成测试/系统测试的区别是什么？ 42](#_Toc497738921)

[7. 效能相关问题 42](#_Toc497738922)

[7.1. 前端性能优化 42](#_Toc497738923)

[7.2. 移动端性能优化 43](#_Toc497738924)

[7.3. 前端长列表的性能优化 43](#_Toc497738925)

[8. 安全相关问题 44](#_Toc497738926)

[8.1. 哪些操作会造成内存泄漏？ 44](#_Toc497738927)

[8.2. sql注入原理 44](#_Toc497738928)

[8.3. 什么是Token？Token有什么作用？作用实现的思路与原理是什么？ 44](#_Toc497738929)

[8.4. 如何生成Token？ 45](#_Toc497738930)

[8.5. XSS 原理是什么？怎么防范呢？请写一个攻击和防御的例子。CSRF 和 hash 碰撞的原理是什么？怎么防范？XSS与CSRF有什么区别吗？ 46](#_Toc497738931)

[9. 网络相关问题 47](#_Toc497738932)

[9.1. HTTP请求报文和HTTP响应报文？ 47](#_Toc497738933)

[9.2. 什么是Etag？ 48](#_Toc497738934)

[9.3. ETag应用 49](#_Toc497738935)

[9.4. Expires和Cache-Control 49](#_Toc497738936)

[9.5. 说说304缓存的原理 49](#_Toc497738937)

[9.6. 说说网络分层里七层模型是哪七层，每一层的作用是什么？ 50](#_Toc497738938)

[9.7. 说说TCP传输的三次握手四次挥手策略 50](#_Toc497738939)

[9.8. TCP和UDP的区别 51](#_Toc497738940)

[9.9. HTTP和HTTPS 51](#_Toc497738941)

[9.10. 为什么HTTPS安全 51](#_Toc497738942)

[9.11. 关于Http 2.0 你知道多少？ 51](#_Toc497738943)

[9.12. HTTP method 及Restful 51](#_Toc497738944)

[9.13. GET和POST的区别，何时使用POST？ 52](#_Toc497738945)

[9.14. 正向代理和反向代理的定义，区别是什么？ 52](#_Toc497738946)

[9.15. HTTP状态码 52](#_Toc497738947)

[10. 数据库相关问题 53](#_Toc497738948)

[10.1. 说说mongoDB和MySQL的区别 53](#_Toc497738949)

[10.2. MongoDB与Redis的比较 54](#_Toc497738950)

[11. 代码相关问题 55](#_Toc497738951)

[11.1. 实现一个函数clone，可以对JavaScript中的5种主要的数据类型（包括Number、String、Object、Array、Boolean）进行值复制 55](#_Toc497738952)

[11.2. 编写一个方法，求一个字符串的字节长度 55](#_Toc497738953)

[12. 趣味问题 55](#_Toc497738954)

# 序言

## 文档说明

本文档包含了一些用于考查候选者的前端面试问题。不建议对单个候选者问及每个问题 (那需要好几个小时)。只要从列表里挑选一些，就能帮助你考查候选者是否具备所需要的技能。

备注： 这些问题中很多都是开放性的，可以引发有趣的讨论。这比直接的答案更能体现此人的能力。

# 常见问题（一些开放性题目）

## 自我介绍

（1）自我介绍：除了基本个人信息以外，面试官更想听的是你与众不同的地方和你的优势。

（2）项目介绍

## 你觉得前端工程师的价值体现在哪？

* 为简化用户使用提供技术支持（交互部分）
* 为多个浏览器兼容性提供支持
* 为提高用户浏览速度（浏览器性能）提供支持
* 为跨平台或者其他基于webkit或其他渲染引擎的应用提供支持
* 为展示数据提供支持（数据接口）

## 说说你对前端架构师的理解

* 负责前端团队的管理及与其他团队的协调工作，提升团队成员能力和整体效率；
* 带领团队完成研发工具及平台前端部分的设计、研发和维护；
* 带领团队进行前端领域前沿技术研究及新技术调研，保证团队的技术领先；
* 负责前端开发规范制定、功能模块化设计、公共组件搭建等工作，并组织培训。

## 对前端界面工程师这个职位是怎么样理解的？（如何看待前端开发？）它的前景会怎么样？

前端是最贴近用户的程序员，比后端、数据库、产品经理、运营、安全都近。

（1）实现界面交互

（2）提升用户体验

（3）有了Node.js，前端可以实现服务端的一些事情

前端的能力就是能让产品从 90分进化到 100 分，甚至更好。

参与项目，快速高质量完成实现效果图，精确到1px；

与团队成员，UI设计，产品经理的沟通；

做好的页面结构，页面重构和用户体验；

处理hack，兼容、写出优美的代码格式；

针对服务器的优化、拥抱最新前端技术。

## 未来的职业规划是怎样的？

（1）首先应该是一个优秀的程序员

（2）其次是努力使自己成为某一领域的技术专家

（3）通过技术更好的服务于团队和业务

（4）提高沟通能力，团队协作，发现问题，解决问题，总结问题能力

（5）写写博客，输出就是最好的学习

（6）提升个人前端的工作效率和工作质量

（7）关注前端前言技术和发展方向，通过新技术服务团队和业务

（8）一专多长

想成为优秀的前端工程师，首先在专业技能领域必不可少，其次在团队贡献、业务思索、价值判断上也有要求。这三方面能决定你的专业技能能够为公司产出多大的价值。

我觉得程序员最核心的竞争力是学习力和责任。 学习能力的源泉就是好奇心，也就是对新知识的渴求，以及对探索未知的冲动。

## 你希望加入一个什么样的团队？

（1）对前端开发有激情

（2）能够持之以恒的学习

（3）团队做事方式是否规范（代码规范，安全规范，流程规范）

（4）团队有足够的成长空间，对自己有个清晰的定位。

（5）团队认可我的价值

## 最后你有什么要问我的吗？

（1）可以问一下公司具体的情况，比如我即将加入的部门的主要业务

（2）问一下具体工作情况，比如需要做哪些内容

（3）公司的氛围和公司的文化

（4）贵司对这项职务的工作内容和期望目标

## 谈谈你对重构的理解

网站重构：在不改变外部行为的前提下，简化结构、添加可读性，而在网站前端保持一致的行为。也就是说是在不改变UI的情况下，对网站进行优化，在扩展的同时保持一致的UI。

对于传统的网站来说重构通常是：

* 表格(table)布局改为DIV+CSS
* 使网站前端兼容于现代浏览器(针对于不合规范的CSS、如对IE6有效的)

对于移动平台的优化

针对于SEO进行优化

深层次的网站重构应该考虑的方面：

* 减少代码间的耦合
* 让代码保持弹性
* 严格按规范编写代码
* 设计可扩展的API
* 代替旧有的框架、语言(如VB)
* 增强用户体验

通常来说对于速度的优化也包含在重构中

* 压缩JS、CSS、image等前端资源(通常是由服务器来解决)
* 程序的性能优化(如数据读写)
* 采用CDN来加速资源加载
* 对于JS DOM的优化
* HTTP服务器的文件缓存

## 什么样的前端代码是好的？

高复用低耦合，这样文件小，好维护，而且好扩展。

# HTML相关问题

## 标签title与h1的区别、b与strong的区别、i与em的区别？

title属性没有明确意义只表示是个标题，H1则表示层次明确的标题，对页面信息的抓取也有很大的影响；

strong是标明重点内容，有语气加强的含义，使用阅读设备阅读网络时：<strong>会重读，而<B>是展示强调内容。

i内容展示为斜体，em表示强调的文本；

<strong>标签和<em>标签一样，用于强调文本，但它强调的程度更强一些。

<em>是斜体强调标签，更强烈强调，表示内容的强调点。相当于html元素中的 <i>...</i>;

<b>、<i>是视觉要素，分别表示无意义的加粗，无意义的斜体。

<em>和<strong>是表达要素(phrase elements)。

Physical Style Elements -- 自然样式标签

b, i, u, s, pre

Semantic Style Elements -- 语义样式标签

strong, em, ins, del, code

应该准确使用语义样式标签, 但不能滥用, 如果不能确定时首选使用自然样式标签。

## Doctype作用? 严格模式与混杂模式如何区分？（即标准模式与兼容模式各有什么区别？）它们有何意义?

1）<!DOCTYPE>声明位于位于HTML文档中的第一行，处于<html>标签之前。告知浏览器的解析器用什么文档标准解析这个文档。DOCTYPE不存在或格式不正确会导致文档以兼容模式呈现。

2）严格模式（标准模式）的排版和JS运作模式是以该浏览器支持的最高标准运行。

3）在混杂模式（兼容模式）中，页面以宽松的向后兼容的方式显示。模拟老式浏览器的行为以防止站点无法工作。

4）DOCTYPE不存在或格式不正确会导致文档以混杂模式（兼容模式）呈现。

## 你知道多少种Doctype文档类型？

该标签可声明三种 DTD 类型，分别表示严格版本、过渡版本以及基于框架的 HTML 文档。

HTML 4.01 规定了三种文档类型：Strict、Transitional 以及 Frameset。

XHTML 1.0 规定了三种 XML 文档类型：Strict、Transitional 以及 Frameset。

Standards （标准）模式（也就是严格呈现模式）用于呈现遵循最新标准的网页，而 Quirks

（包容）模式（也就是松散呈现模式或者兼容模式）用于呈现为传统浏览器而设计的网页。

## HTML5为什么只需要写 <!DOCTYPE HTML>？

（1）HTML5 不基于 SGML，因此不需要对DTD进行引用，但是需要doctype来规范浏览器的行为（让浏览器按照它们应该的方式来运行）；

（2）而HTML4.01基于SGML,所以需要对DTD进行引用，才能告知浏览器文档所使用的文档类型。

## HTML与XHTML——二者有什么区别

XHTML，即可扩展超文本标记语言，是一种置标语言，表现方式与超文本标记语言（HTML）类似，不过语法上更加严格。

从继承关系上讲，HTML是一种基于标准通用置标语言的应用，是一种非常灵活的置标语言，而XHTML则基于可扩展标记语言，可扩展标记语言是标准通用置标语言的一个子集。

**区别：**

（XHTML是当前HTML版的继承者。HTML语法要求比较松散，这样对网页编写者来说，比较方便，但对于机器来说，语言的语法越松散，处理起来就越困难，对于传统的计算机来说，还有能力兼容松散语法，但对于许多其他设备，比如手机，难度就比较大。因此产生了由DTD定义规则，语法要求更加严格的XHTML。

大部分常见的浏览器都可以正确地解析XHTML，即使早期的浏览器，XHTML作为HTML的一个子集，许多也可以解析。也就是说，几乎所有的网页浏览器在正确解析HTML的同时，也可兼容XHTML。当然，从HTML完全转移到XHTML，还需要一个过程。

跟层叠式样式表（外语缩写：CSS）结合后，XHTML能发挥真正的威力；这使实现样式跟内容的分离的同时，又能有机地组合网页代码，在另外的单独文件中，还可以混合各种XML应用，比如MathML、SVG。

从HTML到XHTML过渡的变化比较小，主要是为了适应XML。最大的变化在于文档必须是良构的，所有标签必须闭合，也就是说开始标签要有相应的结束标签。另外，XHTML中所有的标签必须小写。而按照HTML 2.0以来的传统，很多人都是将标签大写，这点两者的差异显著。在XHTML中，所有的参数值，包括数字，必须用双引号括起来（而在SGML和HTML中，引号不是必须的，当内容只是数字、字母及其它允许的特殊字符时，可以不用引号）。所有元素，包括空元素，比如img、br等，也都必须闭合，实现的方式是在开始标签末尾加入斜扛，比如<img … /> 、<br />。省略参数，比如<option selected>，也不允许，必须用<option selected="selected"/>。两者的详细差别，可通过W3C XHTML说明来查阅。

## 行内元素有哪些？块级元素有哪些？ 空(void)元素有那些？

首先：CSS规范规定，每个元素都有display属性，确定该元素的类型，每个元素都有默认的display值，如div的display默认值为“block”，则为“块级”元素；span默认display属性值为“inline”，是“行内”元素。

（1）行内元素有：a b span img input select strong（强调的语气）

（2）块级元素有：div ul ol li dl dt dd h1 h2 h3 h4…p

（3）常见的空元素：

<br> <hr> <img> <input> <link> <meta>

鲜为人知的是：

<area> <base> <col> <command> <embed> <keygen> <param> <source> <track> <wbr>

## html5有哪些新特性、移除了那些元素？如何处理HTML5新标签的浏览器兼容问题？如何区分 HTML 和 HTML5？

**新特性：**

HTML5 现在已经不是 SGML 的子集，主要是关于图像，位置，存储，多任务等功能的增加。

拖拽释放(Drag and drop) API

语义化更好的内容标签（header,nav,footer,aside,article,section）

音频、视频API(audio,video)

画布(Canvas) API

地理(Geolocation) API

本地离线存储 localStorage 长期存储数据，浏览器关闭后数据不丢失；

sessionStorage 的数据在浏览器关闭后自动删除

表单控件，calendar、date、time、email、url、search

新的技术webworker, websocket, Geolocation

**移除的元素**

纯表现的元素：basefont，big，center，font, s，strike，tt，u；

对可用性产生负面影响的元素：frame，frameset，noframes；

**支持HTML5新标签：**

IE8/IE7/IE6支持通过document.createElement方法产生的标签，

可以利用这一特性让这些浏览器支持HTML5新标签，

当然最好的方式是直接使用成熟的框架、使用最多的是html5shim框架

<!--[if lt IE 9]>

<script> src="http://html5shim.googlecode.com/svn/trunk/html5.js"</script>

<![endif]-->

**如何区分：** DOCTYPE声明\新增的结构元素\功能元素

## HTML5的离线储存怎么使用，工作原理能不能解释一下？

在用户没有与因特网连接时，可以正常访问站点或应用，在用户与因特网连接时，更新用户机器上的缓存文件。

**原理：**HTML5的离线存储是基于一个新建的.appcache文件的缓存机制(不是存储技术)，通过这个文件上的解析清单离线存储资源，这些资源就会像cookie一样被存储了下来。之后当网络在处于离线状态下时，浏览器会通过被离线存储的数据进行页面展示。

**如何使用：**

1、页面头部像下面一样加入一个manifest的属性；

2、在cache.manifest文件的编写离线存储的资源；

CACHE MANIFEST

#v0.11

CACHE:

js/app.js

css/style.css

NETWORK:

resourse/logo.png

FALLBACK:

/ /offline.html

3、在离线状态时，操作window.applicationCache进行需求实现。

## 浏览器是怎么对HTML5的离线储存资源进行管理和加载的呢？

在线的情况下，浏览器发现html头部有manifest属性，它会请求manifest文件，如果是第一次访问app，那么浏览器就会根据manifest文件的内容下载相应的资源并且进行离线存储。如果已经访问过app并且资源已经离线存储了，那么浏览器就会使用离线的资源加载页面，然后浏览器会对比新的manifest文件与旧的manifest文件，如果文件没有发生改变，就不做任何操作，如果文件改变了，那么就会重新下载文件中的资源并进行离线存储。

离线的情况下，浏览器就直接使用离线存储的资源。

## iframe有那些缺点？

（1）iframe会阻塞主页面的Onload事件；

（2）搜索引擎的检索程序无法解读这种页面，不利于SEO；

（3）iframe和主页面共享连接池，而浏览器对相同域的连接有限制，所以会影响页面的并行加载。

使用iframe之前需要考虑以上这些缺点。如果需要使用iframe，最好是通过javascript动态给iframe添加src属性值，这样可以绕开以上两个问题。

## Label的作用是什么？是怎么用的？

label标签来定义表单控制间的关系,当用户选择该标签时，浏览器会自动将焦点转到和标签相关的表单控件上。

<label for="Name">Number:</label>

<input type=“text“name="Name" id="Name"/>

<label>Date:<input type="text" name="B"/></label>

## HTML5的form如何关闭自动完成功能？

给不想要提示的 form 或某个 input 设置为 autocomplete=off。

## 页面可见性（Page Visibility API）可以有哪些用途？

通过visibilityState 的值检测页面当前是否可见，以及打开网页的时间等；

监控用户行为，当用户的视角不在当前页面时，暂停加载广告，幻灯片、停止加载视频、开始加载小动画等；

减少对用户宽带的占用，减少服务器压力，节省用户内存，以及到达更好的播放效果。

## 网页验证码是干嘛的，是为了解决什么安全问题。

区分用户是计算机还是人的公共全自动程序。可以防止恶意破解密码、刷票、论坛灌水；

有效防止黑客对某一个特定注册用户用特定程序暴力破解方式进行不断的登陆尝试。

# CSS相关问题

## CSS中link 和@import的区别是？

(1) link属于XHTML标签，除了加载CSS外，还能用于定义RSS, 定义rel连接属性等作用；而@import是CSS提供的，只能用于加载CSS；

(2) 页面被加载的时，link会同时被加载，而@import被引用的CSS会等到引用它的CSS文件被加载完再加载；

(3) import是CSS2.1 提出的，只在IE5以上才能被识别，而link是XHTML标签，无兼容问题；

(4) link方式的样式的权重 高于@import的权重。

## 介绍一下box-sizing属性？

box-sizing属性主要用来控制元素的盒模型的解析模式。默认值是content-box。

content-box：让元素维持W3C的标准盒模型。元素的宽度/高度由border + padding + content的宽度/高度决定，设置width/height属性指的是content部分的宽/高

border-box：让元素维持IE传统盒模型（IE6以下版本和IE6~7的怪异模式）。设置width/height属性指的是border + padding + content

标准浏览器下，按照W3C规范对盒模型解析，一旦修改了元素的边框或内距，就会影响元素的盒子尺寸，就不得不重新计算元素的盒子尺寸，从而影响整个页面的布局。

## 介绍一下标准的CSS的盒子模型？低版本IE的盒子模型有什么不同的？

（1）有两种，IE 盒子模型、W3C 盒子模型；

（2）盒模型：内容(content)、填充(padding)、边界(margin)、 边框(border)；

（3）区别：IE的content部分把 border 和 padding计算了进去。

## CSS 选择符有哪些？哪些属性可以继承？

1.id选择器（ # myid）

2.类选择器（.myclassname）

3.标签选择器（div, h1, p）

4.相邻选择器（h1 + p）

5.子选择器（ul > li）

6.后代选择器（li a）

7.通配符选择器（ \* ）

8.属性选择器（a[rel = "external"]）

9.伪类选择器（a: hover, li:nth-child）

**可继承的样式：** font-size font-family color, UL LI DL DD DT

**不可继承的样式：**border padding margin width height

## CSS优先级[算法](http://lib.csdn.net/base/datastructure)如何计算？

优先级就近原则，同权重情况下样式定义最近者为准;

载入样式以最后载入的定位为准;

**优先级为:**

!important > id > class > tag

important 比内联优先级高，但内联比id要高

## CSS定义的权重

以下是权重的规则：标签的权重为1，class的权重为10，id的权重为100，以下例子是演示各种定义的权重值：

/\*权重为1\*/

div{

}

/\*权重为10\*/

.class1{

}

/\*权重为100\*/

#id1{

}

/\*权重为100+1=101\*/

#id1 div{

}

/\*权重为10+1=11\*/

.class1 div{

}

/\*权重为10+10+1=21\*/

.class1 .class2 div{

}

如果权重相同，则最后定义的样式会起作用，但是应该避免这种情况出现。

## [CSS3](http://lib.csdn.net/base/css3)新增伪类有那些？

**举例：**

p:first-of-type 选择属于其父元素的首个 <p> 元素的每个 <p> 元素。

p:last-of-type 选择属于其父元素的最后 <p> 元素的每个 <p> 元素。

p:only-of-type 选择属于其父元素唯一的 <p> 元素的每个 <p> 元素。

p:only-child 选择属于其父元素的唯一子元素的每个 <p> 元素。

p:nth-child(2) 选择属于其父元素的第二个子元素的每个 <p> 元素。

:after 在元素之前添加内容,也可以用来做清除浮动。

:before 在元素之后添加内容

:enabled :disabled 控制表单控件的禁用状态。

:checked 单选框或复选框被选中。

## 如何居中div？如何居中一个浮动元素？如何让绝对定位的div居中？

**居中div：**

给div设置一个宽度，然后添加margin:0 auto属性：

div{

width:200px;

margin:0 auto;

}

**居中一个浮动元素：**

确定容器的宽高 宽500 高 300 的层

设置层的外边距

.div {

width:500px ;

height:300px;//高度可以不设

margin: -150px 0 0 -250px;

position:relative; //相对定位

background-color:pink; //方便看效果

left:50%;

top:50%;

}

**让绝对定位的div居中：**

position: absolute;

width: 1200px;

background: none;

margin: 0 auto;

top: 0;

left: 0;

bottom: 0;

right: 0;

## display有哪些值？说明他们的作用。

block 块类型。默认宽度为父元素宽度，可设置宽高，换行显示。

none 缺省值。像行内元素类型一样显示。

inline 行内元素类型。默认宽度为内容宽度，不可设置宽高，同行显示。

inline-block 默认宽度为内容宽度，可以设置宽高，同行显示。

list-item 象块类型元素一样显示，并添加样式列表标记。

table 此元素会作为块级表格来显示。

inherit 规定应该从父元素继承 display 属性的值。

## display:none和visibility:hidden的区别？

display:none 隐藏对应的元素，在文档布局中不再给它分配空间，它各边的元素会合拢，就当他从来不存在。

visibility:hidden 隐藏对应的元素，但是在文档布局中仍保留原来的空间。

## display:inline-block 什么时候会显示间隙？

移除空格、使用margin负值、使用font-size:0、letter-spacing、word-spacing

## position:absolute和float属性的异同

* 共同点：对内联元素设置float和absolute属性，可以让元素脱离文档流，并且可以设置其宽高。
* 不同点：float仍会占据位置，absolute会覆盖文档流中的其他元素。

## position的值relative和absolute分别是相对于谁进行定位的（即定位原点是什么）？

**absolute：**生成绝对定位的元素，相对于被设置成相对定位（relative）的父级元素的定位。在被设置成相对元素的父级元素没有relative定位的情况下，它是相对于浏览器窗口TL定位的。

**fixed：**（老IE不支持）生成绝对定位的元素，通常相对于浏览器窗口或 frame 进行定位。

**relative：**生成相对定位的元素，相对于其在普通流中的位置（正常位置）进行定位。

**static：**默认值。没有定位，元素出现在正常的流中（忽略 top, bottom, left, right z-index 声明）。

**inherit：**规定从父元素继承 position 属性的值。

**sticky：**生成粘性定位的元素，容器的位置根据正常文档流计算得出

## absolute的containing block(容器块)计算方式跟正常流有什么不同？

无论属于哪种，都要先找到其祖先元素中最近的 position 值不为 static 的元素，然后再判断：

1、若此元素为 inline 元素，则 containing block 为能够包含这个元素生成的第一个和最后一个 inline box 的 padding box (除 margin, border 外的区域) 的最小矩形；

2、否则,则由这个祖先元素的 padding box 构成。

如果都找不到，则为 initial containing block。

补充：

1. static(默认的)/relative：简单说就是它的父元素的内容框（即去掉padding的部分）

2. absolute: 向上找最近的定位为absolute/relative的元素

3. fixed: 它的containing block一律为根元素(html/body)，根元素也是initial containing block

## CSS3有哪些新特性？

新增各种CSS选择器（: not(.input)：所有 class 不是“input”的节点）

圆角（border-radius），多列布局（multi-column layout），阴影（box-shadow），文字特效（text-shadow），文字渲染（Text-decoration），线性渐变（gradient），旋转（transform）

transform:rotate(9deg) scale(0.85,0.90) translate(0px,-30px) skew(-9deg,0deg) animation;//旋转,缩放,定位,倾斜, 动画

增加了更多的CSS选择器 多背景 rgba

在CSS3中唯一引入的伪元素是::selection.

媒体查询，多栏布局

border-image

CSS3中新增了一种盒模型计算方式：box-sizing。盒模型默认的值是content-box, 新增的值是padding-box和border-box，

几种盒模型计算元素宽高的区别如下：

**content-box（默认）**

布局所占宽度Width：

Width = width + padding-left + padding-right + border-left + border-right

布局所占高度Height:

Height = height + padding-top + padding-bottom + border-top + border-bottom

**padding-box**

布局所占宽度Width：

Width = width(包含padding-left + padding-right) + border-top + border-bottom

布局所占高度Height:

Height = height(包含padding-top + padding-bottom) + border-top + border-bottom

**border-box**

布局所占宽度Width：

Width = width(包含padding-left + padding-right + border-left + border-right)

布局所占高度Height:

Height = height(包含padding-top + padding-bottom + border-top + border-bottom)

## 用纯CSS创建一个三角形的原理是什么？

把上、左、右三条边隐藏掉（颜色设为 transparent）

#demo {

width: 0;

height: 0;

border-width: 20px;

border-style: solid;

border-color: transparent transparent red transparent;

}

## 如果需要手动写动画，你认为最小时间间隔是多久，为什么？

多数显示器默认频率是60Hz，即1秒刷新60次，所以理论上最小间隔为**1/60＊1000ms ＝ 16.7ms**

## 经常遇到的浏览器的兼容性有哪些？原因，解决方法是什么，常用hack的技巧？

* png24位的图片在IE6浏览器上出现背景，解决方案是做成PNG8.
* 浏览器默认的margin和padding不同。解决方案是加一个全局的\*{margin:0;padding:0;}来统一。
* IE6双边距bug:块属性标签float后，又有横行的margin情况下，在IE6显示margin比设置的大。

浮动IE产生的双倍距离 #box{ float:left; width:10px; margin:0 0 0 100px;}

这种情况之下IE会产生20px的距离，解决方案是在float的标签样式控制中加入 ——\_display:inline;将其转化为行内属性。(\_这个符号只有IE6会识别)

渐进识别的方式，从总体中逐渐排除局部。

首先，巧妙的使用“\9”这一标记，将IE游览器从所有情况中分离出来。

接着，再次使用“+”将IE8和IE7、IE6分离开来，这样IE8已经独立识别。

css

.bb{

background-color:#f1ee18;/\*所有识别\*/

.background-color:#00deff\9; /\*IE6、7、8识别\*/

+background-color:#a200ff;/\*IE6、7识别\*/

\_background-color:#1e0bd1;/\*IE6识别\*/

}

* IE下,可以使用获取常规属性的方法来获取自定义属性,

也可以使用getAttribute()获取自定义属性;

Firefox下,只能使用getAttribute()获取自定义属性。

解决方法:统一通过getAttribute()获取自定义属性。

* IE下,even对象有x,y属性,但是没有pageX,pageY属性;

Firefox下,event对象有pageX,pageY属性,但是没有x,y属性。

解决方法：（条件注释）缺点是在IE浏览器下可能会增加额外的HTTP请求数。

* Chrome 中文界面下默认会将小于 12px 的文本强制按照 12px 显示,

可通过加入 CSS 属性 -webkit-text-size-adjust: none; 解决。

超链接访问过后hover样式就不出现了 被点击访问过的超链接样式不在具有hover和active了解决方法是改变CSS属性的排列顺序:

L-V-H-A : a:link {} a:visited {} a:hover {} a:active {}

## li与li之间有看不见的空白间隔是什么原因引起的？有什么解决办法？

行框的排列会受到中间空白（回车\空格）等的影响，因为空格也属于字符,这些空白也会被应用样式，占据空间，所以会有间隔，把字符大小设为0，就没有空格了。

## 为什么要初始化CSS样式。

因为浏览器的兼容问题，不同浏览器对有些标签的默认值是不同的，如果没对CSS初始化往往会出现浏览器之间的页面显示差异。

当然，初始化样式会对SEO有一定的影响，但鱼和熊掌不可兼得，但力求影响最小的情况下初始化。

## 什么是CSS预处理器/后处理器？

**预处理器** 例如：LESS、Sass、Stylus，用来预编译Sass或less，增强了css代码的复用性，还有层级、mixin、变量、循环、函数等，具有很方便的UI组件模块化开发能力，极大的提高工作效率。

**后处理器** 例如：PostCSS，通常被视为在完成的样式表中根据CSS规范处理CSS，让其更有效；目前最常做的是给CSS属性添加浏览器私有前缀，实现跨浏览器兼容性的问题。

## 对BFC规范(块级格式化上下文：block formatting context)的理解？

BFC，块级格式化上下文，一个创建了新的BFC的盒子是独立布局的，盒子里面的子元素的样式不会影响到外面的元素。在同一个BFC中的两个毗邻的块级盒在垂直方向（和布局方向有关系）的margin会发生折叠。

（W3C CSS 2.1 规范中的一个概念，它是一个独立容器，它决定了元素如何对其内容进行布局，以及与其他元素的关系和相互作用。）

一个页面是由很多个 Box 组成的,元素的类型和 display 属性,决定了这个 Box 的类型。

不同类型的 Box,会参与不同的 Formatting Context（决定如何渲染文档的容器）,因此Box内的元素会以不同的方式渲染,也就是说BFC内部的元素和外部的元素不会互相影响。

## 解释下浮动和它的工作原理？清除浮动的技巧

浮动的框可以向左或向右移动，直到他的外边缘碰到包含框或另一个浮动框的边框为止。由于浮动框不在文档的普通流中，所以文档的普通流的块框表现得就像浮动框不存在一样。浮动的块框会漂浮在文档普通流的块框上。

浮动元素脱离文档流，不占据空间。浮动元素碰到包含它的边框或者浮动元素的边框停留。

（1）使用空标签清除浮动。

这种方法是在所有浮动标签后面添加一个空标签 定义css clear:both. 弊端就是增加了无意义标签。

（2）使用overflow。

给包含浮动元素的父标签添加css属性 overflow:auto; zoom:1; zoom:1用于兼容IE6。

（3）使用after伪对象清除浮动。

该方法只适用于非IE浏览器。具体写法可参照以下示例。使用中需注意以下几点。一、该方法中必须为需要清除浮动元素的伪对象中设置 height:0，否则该元素会比实际高出若干像素；

## 浮动元素引起的问题和解决办法？

浮动元素引起的问题：

（1）父元素的高度无法被撑开，影响与父元素同级的元素

（2）与浮动元素同级的非浮动元素（内联元素）会跟随其后

（3）若非第一个元素浮动，则该元素之前的元素也需要浮动，否则会影响页面显示的结构

解决方法：

使用CSS中的clear:both;属性来清除元素的浮动可解决2、3问题，对于问题1，添加如下样式，给父元素添加clearfix样式：

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

.clearfix{display: inline-block;} /\* for IE/Mac \*/

清除浮动的几种方法：

（1）额外标签法，<div style="clear:both;"></div>（缺点：不过这个办法会增加额外的标签使HTML结构看起来不够简洁。）

（2）使用after伪类

#parent:after{

content:".";

height:0;

visibility:hidden;

display:block;

clear:both;

}

（3）浮动外部元素

（4）设置overflow为hidden或者auto

## 简述一下你对HTML语义化的理解？

（1）用正确的标签做正确的事情。

（2）去掉或者丢失样式的时候能够让页面呈现出清晰的结构，即使在没有样式CSS情况下也以一种文档格式显示，并且是容易阅读的。

（3）有利于SEO：html语义化让页面的内容结构化，结构更清晰，便于对浏览器、搜索引擎解析，和搜索引擎建立良好沟通，有助于爬虫抓取更多的有效信息：搜索引擎的爬虫依赖于标签来确定上下文和各个关键字的权重；

（4）方便其他设备解析（如屏幕阅读器、盲人阅读器、移动设备）以意义的方式来渲染网页；

（5）便于团队开发和维护，语义化更具可读性，是下一步吧网页的重要动向，遵循W3C标准的团队都遵循这个标准，可以减少差异化。使阅读源代码的人对网站更容易将网站分块，便于阅读维护理解。

## 什么是 FOUC（无样式内容闪烁）？你如何来避免 FOUC？

FOUC - Flash Of Unstyled Content 文档样式闪烁，用户定义样式表加载之前浏览器使用默认样式显示文档，用户样式加载渲染之后再从新显示文档，造成页面闪烁。

解决方法：把样式表放到文档的head 。<style type="text/css" media="all">@import "../fouc.css";</style>

而引用CSS文件的@import就是造成这个问题的罪魁祸首。IE会先加载整个HTML文档的DOM，然后再去导入外部的CSS文件，因此，在页面DOM加载完成到CSS导入完成中间会有一段时间页面上的内容是没有样式的，这段时间的长短跟网速，电脑速度都有关系。

解决方法简单的出奇，只要在<head>之间加入一个<link>或者<script>元素就可以了。

## 介绍一下你对浏览器内核的理解？常见的浏览器内核有哪些？

浏览器内核主要分成两部分：渲染引擎(layout engineer或Rendering Engine)和JS引擎。

**渲染引擎：**负责取得网页的内容（HTML、XML、图像等等）、整理讯息（例如加入CSS等），以及计算网页的显示方式，然后会输出至显示器或打印机。浏览器的内核的不同对于网页的语法解释会有不同，所以渲染的效果也不相同。所有网页浏览器、电子邮件客户端以及其它需要编辑、显示网络内容的应用程序都需要内核。

**JS引擎：**解析和执行javascript来实现网页的动态效果。

最开始渲染引擎和JS引擎并没有区分的很明确，后来JS引擎越来越独立，内核就倾向于只指渲染引擎。

**常见的浏览器内核：**

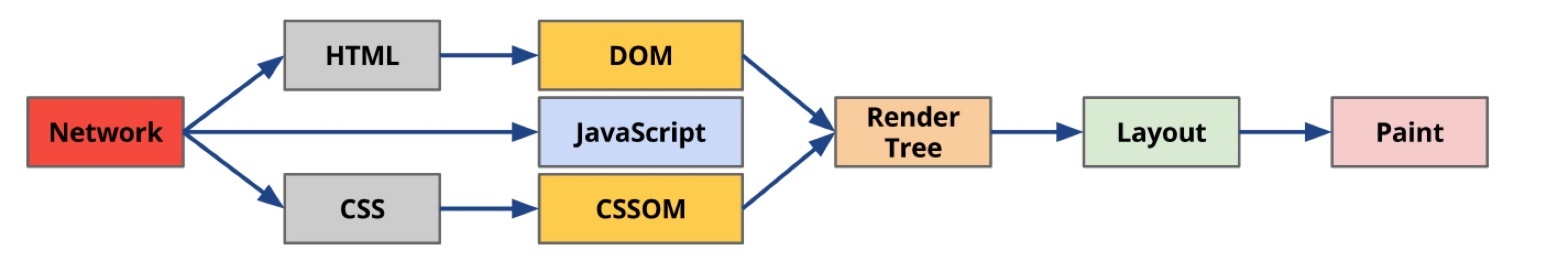
Trident内核：IE,MaxThon,TT,The World,360,搜狗浏览器等。[又称MSHTML]

Gecko内核：Netscape6及以上版本，FF,MozillaSuite/SeaMonkey等

Presto内核：Opera7及以上。 [Opera内核原为：Presto，现为：Blink;]

Webkit内核：Safari,Chrome等。 [ Chrome的：Blink（WebKit的分支）]

## 浏览器渲染原理解析



（1）首先渲染引擎下载HTML，解析生成DOM Tree

（2）遇到css标签或JS脚本标签就新起线程去下载他们，并继续构建DOM。（其中css是异步下载同步执行）浏览器引擎通过 DOM Tree 和 CSS Rule Tree 构建 Rendering Tree

（3）通过 CSS Rule Tree 匹配 DOM Tree 进行定位坐标和大小，这个过程称为 Flow 或 Layout 。

（4）最终通过调用Native GUI 的 API 绘制网页画面的过程称为 Paint 。

当用户在浏览网页时进行交互或通过 js 脚本改变页面结构时，以上的部分操作有可能重复运行，此过程称为 Repaint 或 Reflow。 重排是指dom树发生结构变化后，需要重新构建dom结构。 重绘是指dom节点样式改变，重新绘制。 重排一定会带来重绘，重绘不一定有重排。

如何减少浏览器重排：将需要多次重排的元素，position属性设为absolute或fixed，这样此元素就脱离了文档流，它的变化不会影响到其他元素。

## 如何在页面上实现一个圆形的可点击区域？

（1）map+area或者svg

（2）border-radius

（3）纯js实现 需要求一个点在不在圆上简单算法、获取鼠标坐标等等

## 如何实现不使用 border 画出1px高的线，在不同浏览器的标准模式与怪异模式下都能保持一致的效果。

<div style="height:1px;overflow:hidden;background:red"></div>

# JS相关问题

## 介绍js的基本数据类型。

ECMAScript中有6种简单数据类型（也称为基本数据类型）： Undefined、Null、Boolean、Number、String和Symbol(new in ECMAScript 2015)。Object、Array和Function则属于引用类型。

## 介绍js有哪些内置对象？

数据封装类对象：Object、Array、Boolean、Number 和 String

其他对象：Function、Arguments、Math、Date、RegExp、Error

Object 是 JavaScript 中所有对象的父对象。

## JavaScript有几种类型的值？有什么区别？

栈：原始数据类型（Undefined，Null，Boolean，Number、String）

堆：引用数据类型（对象、数组和函数）

两种类型的区别是：存储位置不同；

原始数据类型直接存储在栈(stack)中的简单数据段，占据空间小、大小固定，属于被频繁使用数据，所以放入栈中存储；

引用数据类型存储在堆(heap)中的对象,占据空间大、大小不固定,如果存储在栈中，将会影响程序运行的性能；引用数据类型在栈中存储了指针，该指针指向堆中该实体的起始地址。当解释器寻找引用值时，会首先检索其在栈中的地址，取得地址后从堆中获得实体。

## 写几条JavaScript的基本规范？

1、不要在同一行声明多个变量。

2、请使用 ===/!==来比较true/false或者数值

3、使用对象字面量替代new Array这种形式

4、不要使用全局函数。

5、Switch语句必须带有default分支

6、函数不应该有时候有返回值，有时候没有返回值。

7、For循环必须使用大括号

8、If语句必须使用大括号

9、for-in循环中的变量 应该使用var关键字明确限定作用域，从而避免作用域污染。

## javascript对象的几种创建方式

javascript创建对象简单的说,无非就是使用内置对象或各种自定义对象，当然还可以用JSON；但写法有很多种，也能混合使用。

1. 使用Object构造函数来创建一个对象

下面代码创建了一个person对象，并用两种方式打印出了Name的属性值。

|  |
| --- |
| var person = new Object();  person.name="kevin";  person.age=31;  alert(person.name);  alert(person["name"]) |

上述写法的另外一种表现形式是使用对象字面量创建一个对象，不要奇怪person[“5”],这里是合法的；另外使用这种加括号的方式字段之间是可以有空格的如person[“my age”].

|  |
| --- |
| var person =  {  name:"Kevin",  age:31,  5:"Test"  };  alert(person.name);  alert(person["5"]); |

虽然Object 构造函数或对象字面量都可以用来创建单个对象，但这些方式有个明显的缺点：使用同一个接口创建很多对象，会产生大量的重复代码。为解决这个问题，人们开始使用工厂模式的一种变体。

1. 工厂模式

工厂模式是软件工程领域一种广为人知的设计模式，这种模式抽象了创建具体对象的过程，考虑到在ECMAScript 中无法创建类，开发人员就发明了一种函数，用函数来封装以特定接口创建对象的细节，如下面的例子所示。

|  |
| --- |
| function createPerson(name, age, job){  var o = new Object();  o.name = name;  o.age = age;  o.job = job;  o.sayName = function(){  alert(this.name);  };  return o;  }  var person1 = createPerson("Nicholas", 29, "Software Engineer");  var person2 = createPerson("Greg", 27, "Doctor"); |

工厂模式虽然解决了创建多个相似对象的问题，但却没有解决对象识别的问题（即怎样知道一个对象的类型）。随着JavaScript的发展，又一个新模式出现了。

1. 构造函数模式
2. 原型模式

|  |
| --- |
| function Dog(){  }  Dog.prototype.name="旺财";  Dog.prototype.eat=function(){  alert(this.name+"是个吃货");  }  var wangcai =new Dog();  wangcai.eat(); |

1. 混合构造函数和原型模式

|  |
| --- |
| function Car(name,price){  this.name=name;  this.price=price;  }  Car.prototype.sell=function(){  alert("我是"+this.name+"，我现在卖"+this.price+"万元");  }  var camry =new Car("凯美瑞",27);  camry.sell(); |

1. 动态原型模式
2. 寄生构造函数模式
3. 稳妥构造函数模式

## javascript继承的6种方法及其优缺点

（1）原型链继承

缺点：一是字面量重写原型会中断关系，使用引用类型的原型，并且子类型还无法给超类型传递参数。

（2）借用构造函数继承（类式继承）

缺点：借用构造函数虽然解决了刚才两种问题，但没有原型，则复用无从谈起。所以我们需要原型链+借用构造函数的模式，这种模式称为组合继承

（3）组合继承(原型+借用构造)

缺点：组合式继承是比较常用的一种继承方法，其背后的思路是 使用原型链实现对原型属性和方法的继承，而通过借用构造函数来实现对实例属性的继承。这样，既通过在原型上定义方法实现了函数复用，又保证每个实例都有它自己的属性。

（4）原型式继承

（5）寄生式继承

（6）寄生组合式继承

## 说说你对闭包的理解，有什么特性？为什么要使用闭包？

闭包（closure）是指有权访问另一个函数作用域中的变量的函数，创建闭包的最常见的方式就是在一个函数内创建另一个函数，通过另一个函数访问这个函数的局部变量。

闭包的优点是可以避免全局变量的污染，缺点是闭包会常驻内存，会增大内存使用量，使用不当很容易造成内存泄露。在js中，函数即闭包，只有函数才会产生作用域的概念

**闭包有三个特性：**

（1）函数嵌套函数；

（2）函数内部可以引用外部的参数和变量；

（3）参数和变量不会被垃圾回收机制回收。

**为什么要使用闭包：**

为了设计私有的方法和变量，避免全局变量污染，希望一个变量长期驻扎在内存中。一般函数执行完毕后，局部活动对象就被销毁，内存中仅仅保存全局作用域。但闭包的情况不同！

**嵌套函数的闭包：**

function aaa() {

var a = 1;

return function(){

alert(a++)

};

}

var fun = aaa();

fun();// 1 执行后 a++，，然后a还在~

fun();// 2

fun = null;//a被回收！！

其实你写的每一个js函数都是闭包，一个js函数的顶层作用域就是window对象，js的执行环境本身就是一个scope（浏览器的window/node的global），我们通常称之为全局作用域。每个函数，不论多深，都可以认为是全局scope的子作用域，可以理解为闭包。

## 什么是原型？什么是原型链？有什么特点？

每个对象都会在其内部初始化一个属性，就是**prototype(原型)**。

当我们访问一个对象的属性时，如果这个对象内部不存在这个属性，那么他就会去prototype里找这个属性，这个prototype又会有自己的prototype，于是就这样一直找下去，也就是我们平时所说的**原型链**的概念。

即：当从一个对象那里调取属性或方法时，如果该对象自身不存在这样的属性或方法，就会去自己关联的prototype对象那里寻找，如果prototype没有，就会去prototype关联的前辈prototype那里寻找，如果再没有则继续查找Prototype.Prototype引用的对象，依次类推，直到Prototype.….Prototype为undefined（Object的Prototype就是undefined）从而形成了所谓的“原型链”。

**特点：**JavaScript对象是通过引用来传递的，我们创建的每个新对象实体中并没有一份属于自己的原型副本。当我们修改原型时，与之相关的对象也会继承这一改变。

## instanceof 确定原型和实例之间的关系

instanceof用来判断某个构造函数的prototype属性是否存在另外一个要检测对象的原型链上。

对象的\_\_proto\_\_指向自己构造函数的prototype。obj.\_\_proto\_\_.\_\_proto\_\_...的原型链由此产生，包括我们的操作符instanceof正是通过探测obj.\_\_proto\_\_.\_\_proto\_\_... === Constructor.prototype来验证obj是否是Constructor的实例。

function C(){}

var o = new C(){}

//true 因为Object.getPrototypeOf(o) === C.prototype

o instanceof C

instanceof只能用来判断对象和函数，不能用来判断字符串和数字

## isPrototypeOf

用于测试一个对象是否存在于另一个对象的原型链上。

判断父级对象 可检查整个原型链。

## 说说你对作用域链的理解

作用域链的作用是保证执行环境里有权访问的变量和函数是有序的，作用域链的变量只能向上访问，变量访问到window对象即被终止，作用域链向下访问变量是不被允许的。

## null和undefined的区别？

null是一个表示”无”的对象，转为数值时为0；undefined是一个表示”无”的原始值，转为数值时为NaN。

当声明的变量还未被初始化时，变量的默认值为undefined。

null用来表示尚未存在的对象，常用来表示函数企图返回一个不存在的对象。

undefined表示”缺少值”，就是此处应该有一个值，但是还没有定义。典型用法是：

（1）变量被声明了，但没有赋值时，就等于undefined。

（2) 调用函数时，应该提供的参数没有提供，该参数等于undefined。

（3）对象没有赋值的属性，该属性的值为undefined。

（4）函数没有返回值时，默认返回undefined。

null表示”没有对象”，即该处不应该有值。典型用法是：

（1） 作为函数的参数，表示该函数的参数不是对象。

（2） 作为对象原型链的终点。

## new操作符具体干了什么呢?

（1）创建一个空对象，并且 this 变量引用该对象，同时还继承了该函数的原型（实例对象通过\_\_proto\_\_属性指向原型对象；obj.\_\_proto\_\_ = Base.prototype;）。

（2）属性和方法被加入到 this 引用的对象中。

（3）新创建的对象由 this 所引用，并且最后隐式的返回 this 。

var obj = {};

obj.\_\_proto\_\_ = Base.prototype;

Base.call(obj);

## call()和apply()的区别和作用？

**区别：**

（1）call，apply都属于Function.prototype的一个方法，它是JavaScript引擎内在实现的，因为属于Function.prototype，所以每个Function对象实例(就是每个方法)都有call，apply属性。既然作为方法的属性，那它们的使用就当然是针对方法的了，这两个方法是容易混淆的，因为它们的作用一样，只是使用方式不同。

（2）语法：foo.call(this, arg1,arg2,arg3) == foo.apply(this, arguments) == this.foo(arg1, arg2, arg3);

（3）相同点：两个方法产生的作用是完全一样的。

（4）不同点：方法传递的参数不同。

**作用：**动态改变某个类的某个方法的运行环境（执行上下文）。

## 如何解决跨域问题

**（1）JSONP：**

原理是：动态插入script标签，通过script标签引入一个js文件，这个js文件载入成功后会执行我们在url参数中指定的函数，并且会把我们需要的json数据作为参数传入。

由于同源策略的限制，XmlHttpRequest只允许请求当前源（域名、协议、端口）的资源，为了实现跨域请求，可以通过script标签实现跨域请求，然后在服务端输出JSON数据并执行回调函数，从而解决了跨域的数据请求。

优点是兼容性好，简单易用，支持浏览器与服务器双向通信。缺点是只支持GET请求。

JSONP：json+padding（内填充），顾名思义，就是把JSON填充到一个盒子里

<script>

function createJs(sUrl){

var oScript = document.createElement('script');

oScript.type = 'text/javascript';

oScript.src = sUrl;

document.getElementsByTagName('head')[0].appendChild(oScript);

}

createJs('jsonp.js');

box({

'name': 'test'

});

function box(json){

alert(json.name);

}

</script>

**（2）CORS**

服务器端对于CORS的支持，主要就是通过设置Access-Control-Allow-Origin来进行的。如果浏览器检测到相应的设置，就可以允许Ajax进行跨域的访问。

**（3）通过修改document.domain来跨子域**

将子域和主域的document.domain设为同一个主域.前提条件：这两个域名必须属于同一个基础域名!而且所用的协议，端口都要一致，否则无法利用document.domain进行跨域

主域相同的使用document.domain

**（4）使用window.name来进行跨域**

window对象有个name属性，该属性有个特征：即在一个窗口(window)的生命周期内,窗口载入的所有的页面都是共享一个window.name的，每个页面对window.name都有读写的权限，window.name是持久存在一个窗口载入过的所有页面中的

**（5）使用**[**HTML5**](http://lib.csdn.net/base/html5)**中新引进的window.postMessage方法来跨域传送数据**

还有flash、在服务器上设置代理页面等跨域方式。个人认为window.name的方法既不复杂，也能兼容到几乎所有浏览器，这真是极好的一种跨域方法。

## DOM操作——怎样添加、移除、移动、复制、创建和查找节点。

（1）创建新节点

createDocumentFragment() //创建一个DOM片段

createElement() //创建一个具体的元素

createTextNode() //创建一个文本节点

（2）添加、移除、替换、插入

appendChild()

removeChild()

replaceChild()

insertBefore() //并没有insertAfter()

（3）查找

getElementsByTagName() //通过标签名称

getElementsByName() //通过元素的Name属性的值(IE容错能力较强，

会得到一个数组，其中包括id等于name值的)

getElementById() //通过元素Id，唯一性

## document.write()的用法

document.write()方法可以用在两个方面：页面载入过程中用实时脚本创建页面内容，以及用延时脚本创建本窗口或新窗口的内容。

document.write只能重绘整个页面。innerHTML可以重绘页面的一部分

## 请解释什么是事件代理

事件代理（Event Delegation），又称之为事件委托。是JavaScript中常用绑定事件的常用技巧。顾名思义，“事件代理”即是把原本需要绑定的事件委托给父元素，让父元素担当事件监听的职务。事件代理的原理是DOM元素的事件冒泡。使用事件代理的好处是可以提高性能。

## 渐进增强和优雅降级

**渐进增强：**针对低版本浏览器进行构建页面，保证最基本的功能，然后再针对高级浏览器进行效果、交互等改进和追加功能达到更好的用户体验。

**优雅降级：**一开始就构建完整的功能，然后再针对低版本浏览器进行兼容。

## XML和JSON的区别？

（1）数据体积方面。

JSON相对于XML来讲，数据的体积小，传递的速度更快些。

（2）数据交互方面。

JSON与JavaScript的交互更加方便，更容易解析处理，更好的数据交互。

（3）数据描述方面。

JSON对数据的描述性比XML较差。

（4）传输速度方面。

JSON的速度要远远快于XML。

## Javascript垃圾回收方法

**标记清除（mark and sweep）**

这是JavaScript最常见的垃圾回收方式，当变量进入执行环境的时候，比如函数中声明一个变量，垃圾回收器将其标记为“进入环境”，当变量离开环境的时候（函数执行结束）将其标记为“离开环境”。

垃圾回收器会在运行的时候给存储在内存中的所有变量加上标记，然后去掉环境中的变量以及被环境中变量所引用的变量（闭包），在这些完成之后仍存在标记的就是要删除的变量了

**引用计数(reference counting)**

在低版本IE中经常会出现内存泄露，很多时候就是因为其采用引用计数方式进行垃圾回收。引用计数的策略是跟踪记录每个值被使用的次数，当声明了一个 变量并将一个引用类型赋值给该变量的时候这个值的引用次数就加1，如果该变量的值变成了另外一个，则这个值得引用次数减1，当这个值的引用次数变为0的时 候，说明没有变量在使用，这个值没法被访问了，因此可以将其占用的空间回收，这样垃圾回收器会在运行的时候清理掉引用次数为0的值占用的空间。

在IE中虽然JavaScript对象通过标记清除的方式进行垃圾回收，但BOM与DOM对象却是通过引用计数回收垃圾的，

也就是说只要涉及BOM及DOM就会出现循环引用问题。

## 栈和队列的区别?

**队列先进先出**，**栈先进后出**。

栈的插入和删除操作都是在一端进行的，而队列的操作却是在两端进行的。

栈只允许在表尾一端进行插入和删除，而队列只允许在表尾一端进行插入，在表头一端进行删除

## 栈和堆的区别？

栈区（stack）— 由编译器自动分配释放，存放函数的参数值，局部变量的值等。

堆区（heap）— 一般由程序员分配释放，若程序员不释放，程序结束时可能由OS回收。

堆（数据结构）：堆可以被看成是一棵树，如：堆排序；

栈（数据结构）：一种先进后出的数据结构。

## 快速排序的思想并实现一个快排？

“快速排序”的思想很简单，整个排序过程只需要三步：

　　（1）在数据集之中，找一个基准点

　　（2）建立两个数组，分别存储左边和右边的数组

　　（3）利用递归进行下次比较

<script type="text/javascript">

function quickSort(arr){

if(arr.length<=1){

return arr;//如果数组只有一个数，就直接返回；

}

var num = Math.floor(arr.length/2);//找到中间数的索引值，如果是浮点数，则向下取整

var numValue = arr.splice(num,1);//找到中间数的值

var left = [];

var right = [];

for(var i=0;i<arr.length;i++){

if(arr[i]<numValue){

left.push(arr[i]);//基准点的左边的数传到左边数组

}

else{

right.push(arr[i]);//基准点的右边的数传到右边数组

}

}

return quickSort(left).concat([numValue],quickSort(right));//递归不断重复比较

}

alert(quickSort([32,45,37,16,2,87]));//弹出“2,16,32,37,45,87”

</script>

## 对前端模块化的认识

**AMD 是 RequireJS 在推广过程中对模块定义的规范化产出。**

**CMD 是 SeaJS 在推广过程中对模块定义的规范化产出。**

AMD 是提前执行，CMD 是延迟执行。

AMD推荐的风格通过返回一个对象做为模块对象，CommonJS的风格通过对module.exports或exports的属性赋值来达到暴露模块对象的目的。

**CMD模块方式**

define(function(require, exports, module) {

// 模块代码

});

## 说说你对AMD和Commonjs的理解

CommonJS是服务器端模块的规范，[**Node.js**](http://lib.csdn.net/base/nodejs)采用了这个规范。CommonJS规范加载模块是同步的，也就是说，只有加载完成，才能执行后面的操作。AMD规范则是非同步加载模块，允许指定回调函数。

AMD推荐的风格通过返回一个对象做为模块对象，CommonJS的风格通过对module.exports或exports的属性赋值来达到暴露模块对象的目的。

## 谈谈你对webpack的看法

WebPack 是一个模块打包工具，你可以使用WebPack管理你的模块依赖，并编绎输出模块们所需的静态文件。它能够很好地管理、打包Web开发中所用到的HTML、[**JavaScript**](http://lib.csdn.net/base/javascript)、CSS以及各种静态文件（图片、字体等），让开发过程更加高效。对于不同类型的资源，webpack有对应的模块加载器。webpack模块打包器会分析模块间的依赖关系，最后 生成了优化且合并后的静态资源。

webpack的两大特色：

1.code splitting（可以自动完成）

2.loader 可以处理各种类型的静态文件，并且支持串联操作

webpack 是以commonJS的形式来书写脚本滴，但对 AMD/CMD 的支持也很全面，方便旧项目进行代码迁移。

webpack具有requireJs和browserify的功能，但仍有很多自己的新特性：

（1） 对 CommonJS 、 AMD 、ES6的语法做了兼容

（2）对js、css、图片等资源文件都支持打包

（3）串联式模块加载器以及插件机制，让其具有更好的灵活性和扩展性，例如提供对CoffeeScript、ES6的支持

（4）有独立的配置文件webpack.config.js

（5）可以将代码切割成不同的chunk，实现按需加载，降低了初始化时间

（6）支持 SourceUrls 和 SourceMaps，易于调试

（7）具有强大的Plugin接口，大多是内部插件，使用起来比较灵活

（8）webpack 使用异步 IO 并具有多级缓存。这使得 webpack 很快且在增量编译上更加快

## 你觉得jQuery或zepto源码有哪些写的好的地方

[**jQuery**](http://lib.csdn.net/base/jquery)源码封装在一个匿名函数的自执行环境中，有助于防止变量的全局污染，然后通过传入window对象参数，可以使window对象作为局部变量使用，好处是当jquery中访问window对象的时候，就不用将作用域链退回到顶层作用域了，从而可以更快的访问window对象。同样，传入undefined参数，可以缩短查找undefined时的作用域链。

(function( window, undefined ) {

//用一个函数域包起来，就是所谓的沙箱

//在这里边var定义的变量，属于这个函数域内的局部变量，避免污染全局

//把当前沙箱需要的外部变量通过函数参数引入进来

//只要保证参数对内提供的接口的一致性，你还可以随意替换传进来的这个参数

window.jQuery = window.$ = jQuery;

})( window );

jquery将一些原型属性和方法封装在了jquery.prototype中，为了缩短名称，又赋值给了jquery.fn，这是很形象的写法。

有一些数组或对象的方法经常能使用到，jQuery将其保存为局部变量以提高访问速度。

jquery实现的链式调用可以节约代码，所返回的都是同一个对象，可以提高代码效率。

## 说说你对MVC和MVVM的理解，并列举一些MVC和MVVM架构的js框架。

**（1）MVC**

MVC是Model—View—Controler的简称。即模型—视图—控制器。

MVC是一种设计模式，它强制性的把应用程序的输入、处理和输出分开。MVC中的模型、视图、控制器它们分别担负着不同的任务。

视图：视图是用户看到并与之交互的界面。视图向用户显示相关的数据，并接受用户的输入。视图不进行任何业务逻辑处理。**（View 传送指令到 Controller）**

模型：模型表示业务数据和业务处理。一个模型能为多个视图提供数据。这提高了应用程序的重用性。

**（Model 将新的数据发送到 View，用户得到反馈）**

控制器：当用户单击Web页面中的提交按钮时,控制器接受请求并调用相应的模型去处理请求。然后根据处理的结果调用相应的视图来显示处理的结果。**（Controller 完成业务逻辑后，要求 Model 改变状态）**

MVC的处理过程：首先控制器接受用户的请求，调用相应的模型来进行业务处理，并返回数据给控制器。控制器调用相应的视图来显示处理的结果。并通过视图呈现给用户。MVC所有通信都是单向的。

**（2）MVVM**

Model层代表数据模型，可以在Model中定义数据修改和操作业务逻辑； view 代表UI组件。负责将数据模型转换成UI展现出来 ViewModel 是一个同步View和Model的对象

用户操作view层，view数据变化会同步到Model，Model数据变化会立即反应到view中。viewModel通过双向数据绑定把view层和Model层连接了起来。

**MVC架构的js框架：**Backbone.js

**MVVM架构的js框架：**Ember.js、Angular.js、Knockout.js、React.js、Vue.js

## 双向绑定和单向数据绑定的优缺点

只有 UI控件 才存在双向，非 UI控件 只有单向。

**单向绑定的优点：**

可以带来单向数据流，这样的好处是流动方向可以跟踪，流动单一，没有状态, 这使得单向绑定能够避免状态管理在复杂度上升时产生的各种问题, 程序的调试会变得相对容易。单向数据流更利于状态的维护及优化，更利于组件之间的通信，更利于组件的复用

**双向数据流的优点：**

无需进行和单向数据绑定的那些CRUD（Create，Retrieve，Update，Delete）操作； 双向绑定在一些需要实时反应用户输入的场合会非常方便 用户在视图上的修改会自动同步到数据模型中去，数据模型中值的变化也会立刻同步到视图中去；

**缺点：**

双向数据流是自动管理状态的, 但是在实际应用中会有很多不得不手动处理状态变化的逻辑, 使得程序复杂度上升 无法追踪局部状态的变化 双向数据流，值和UI绑定，但由于各种数据相互依赖相互绑定，导致数据问题的源头难以被跟踪到

## 如何评价AngularJS和BackboneJS

Angular它采用双向绑定（data-binding）：View的变动，自动反映在 ViewModel，反之亦然。组成部分Model、View、ViewModel：

View：UI界面

ViewModel：它是View的抽象，负责View与Model之间信息转换，将View的Command传送到Model；

Model：数据访问层

backbone具有依赖性，依赖underscore.js。Backbone + Underscore + jQuery(or Zepto) 就比一个[**AngularJS**](http://lib.csdn.net/base/angularjs) 多出了2 次HTTP请求.

Backbone的Model没有与UI视图数据绑定，而是需要在View中自行操作DOM来更新或读取UI数据。AngularJS与此相反，Model直接与UI视图绑定，Model与UI视图的关系，通过directive封装，AngularJS内置的通用directive，就能实现大部分操作了，也就是说，基本不必关心Model与UI视图的关系，直接操作Model就行了，UI视图自动更新。

AngularJS的directive，你输入特定数据，他就能输出相应UI视图。是一个比较完善的前端MVW框架，包含模板，数据双向绑定，路由，模块化，服务，依赖注入等所有功能，模板功能强大丰富，并且是声明式的，自带了丰富的 Angular 指令。

## react和vue的区别

**相同点：**

* 都支持服务端渲染
* 都有Virtual DOM，组件化开发，通过props参数进行父子组件数据的传递，都实现webComponents规范
* 数据驱动视图
* 都有支持native的方案，React的React native，Vue的weex

**不同点：**

* React严格上只针对MVC的view层，Vue则是MVVM模式
* virtual DOM 不一样 vue会跟踪每一个组件的依赖关系，不需要重新渲染整个组件树。而对于React而言，每当应用的状态被改变时，全部子组件都会重新渲染。当然，这可以通过shouldComponentUpdate这个生命周期方法来进行控制，
* 组件写法不一样 React 推荐的做法是 JSX + inline style，也就是把 HTML 和 CSS 全都写进 JavaScript 了，即”all in js” Vue 推荐的是使用 webpack + vue-loader 的单文件组件格式，即html,css,js写在同一个文件；
* 数据绑定：Vue有实现了双向数据绑定，React数据流动是单向的
* state对象在react应用中是不可变的，需要使用setState方法更新状态；在Vue中，state对象并不是必须的，数据由data属性在Vue对象中进行管理。

## react的优缺点

**优点：**

* 可以通过函数式方法描述视图组件（好处：相同的输入会得到同样的渲染结果，不会有副作用；组件不会被实例化，整体渲染性能得到提升）
* 集成虚拟DOM（性能好）
* 单向数据流（好处是更容易追踪数据变化排查问题
* 一切都是component：代码更加模块化，重用代码更容易，可维护性高
* 大量拥抱 es6 新特性
* jsx

**缺点：**

* jsx的一个问题是，渲染函数常常包含大量逻辑，最终看着更像是程序片段，而不是视觉呈现。后期如果发生需求更改，维护起来工作量将是巨大的
* 大而全，上手有难度

## jsx的优缺点

允许使用熟悉的语法来定义HTML元素树 JSX 让小组件更加简单、明了、直观。 更加语义化且易懂的标签 JSX 本质是对JavaScript语法的一个扩展，看起来像是某种模板语言，但其实不是。但正因为形似HTML，描述UI就更直观了，也极大地方便了开发； 在React中babel会将JSX转换为React.createElement函数调用，然后将JSX转换为正确的JSON对象（VDOM 也是一个“树”形的结构） React/JSX乍看之下，觉得非常啰嗦，但使用JavaScript而不是模板语法来开发（模板语法比较有局限性），赋予了开发者许多编程能力。

## 用过哪些设计模式？

**工厂模式**

主要好处就是可以消除对象间的耦合，通过使用工程方法而不是new关键字。将所有实例化的代码集中在一个位置防止代码重复。

工厂模式解决了重复实例化的问题 ，但还有一个问题,那就是识别问题，因为根本无法 搞清楚他们到底是哪个对象的实例。

function createObject(name,age,profession){//集中实例化的函数var obj = new Object();

obj.name = name;

obj.age = age;

obj.profession = profession;

obj.move = function () {

return this.name + ' at ' + this.age + ' engaged in ' + this.profession;

};

return obj;

}

var test1 = createObject('trigkit4',22,'programmer');//第一个实例var test2 = createObject('mike',25,'engineer');//第二个实例

**构造函数模式**

使用构造函数的方法 ，即解决了重复实例化的问题 ，又解决了对象识别的问题，该模式与工厂模式的不同之处在于：

（1）构造函数方法没有显示的创建对象 (new Object());

（2）直接将属性和方法赋值给 this 对象;

（3）没有 renturn 语句。

## ES6的了解

新增模板字符串（为JavaScript提供了简单的字符串插值功能）、箭头函数（操作符左边为输入的参数，而右边则是进行的操作以及返回的值Inputs=>outputs。）、for-of（用来遍历数据—例如数组中的值。）arguments对象可被不定参数和默认参数完美代替。ES6将promise对象纳入规范，提供了原生的Promise对象。增加了let和const命令，用来声明变量。增加了块级作用域。let命令实际上就增加了块级作用域。ES6规定，var命令和function命令声明的全局变量，属于全局对象的属性；let命令、const命令、class命令声明的全局变量，不属于全局对象的属性。。还有就是引入module模块的概念。

## js中的 "use strict" 做了什么？说说严格模式的限制？使用严格模式有什么优点和缺点？

标记严格模式后：

* 如果在语法检测时发现语法问题，则整个代码块失效，并导致一个语法异常。
* 如果在运行期出现了违反严格模式的代码，则抛出执行异常。

严格模式主要有以下限制：

* 变量必须声明后再使用
* 函数的参数不能有同名属性，否则报错
* 不能使用with语句
* 不能对只读属性赋值，否则报错
* 不能使用前缀0表示八进制数，否则报错
* 不能删除不可删除的属性，否则报错
* 不能删除变量delete prop，会报错，只能删除属性delete global[prop]
* eval不会在它的外层作用域引入变量
* eval和arguments不能被重新赋值
* arguments不会自动反映函数参数的变化
* 不能使用arguments.callee
* 不能使用arguments.caller
* 禁止this指向全局对象
* 不能使用fn.caller和fn.arguments获取函数调用的堆栈
* 增加了保留字（比如protected、static和interface）

设立”严格模式”的目的（即严格模式的优点），主要有以下几个：

* 消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为;
* 消除代码运行的一些不安全之处，保证代码运行的安全；
* 提高编译器效率，增加运行速度；
* 为未来新版本的Javascript做好铺垫。

**缺点：**

现在网站的JS 都会进行压缩，一些文件用了严格模式，而另一些没有。这时这些本来是严格模式的文件，被 merge 后，这个串就到了文件的中间，不仅没有指示严格模式，反而在压缩后浪费了字节。

**注：经过**[**测试**](http://lib.csdn.net/base/softwaretest)**IE6、7、8、9均不支持严格模式，会将其当成普通字符串加以忽略。**

## 谈一谈let与var和const的区别？

* let为ES6新添加申明变量的命令，它类似于var，但是有以下不同：
* let命令不存在变量提升，如果在let前使用，会导致报错
* 暂时性死区的本质，其实还是块级作用域必须“先声明后使用”的性质。
* let，const和class声明的全局变量不是全局对象的属性。

const声明的变量与let声明的变量类似，它们的不同之处在于，const声明的变量只可以在声明时赋值，不可随意修改，否则会导致SyntaxError（语法错误）。

const只是保证变量名指向的地址不变，并不保证该地址的数据不变。const可以在多个模块间共享 let 暂时性死区的原因：var 会变量提升，let 不会。

## 箭头函数

箭头函数不属于普通的 function，所以没有独立的上下文。箭头函数体内的this对象，就是定义时所在的对象，而不是使用时所在的对象。 由于箭头函数没有自己的this，函数对象中的call、apply、bind三个方法，无法"覆盖"箭头函数中的this值。 箭头函数没有原本(传统)的函数有的隐藏arguments对象。 箭头函数不能当作generators使用，使用yield会产生错误。

在以下场景中不要使用箭头函数去定义：

* 定义对象方法、定义原型方法、定义构造函数、定义事件回调函数。
* 箭头函数里不但没有 this，也没有 arguments, super ……

## 浏览器本地存储

在较高版本的浏览器中，js提供了sessionStorage和globalStorage。在HTML5中提供了localStorage来取代globalStorage。

html5中的Web Storage包括了两种存储方式：sessionStorage和localStorage。

sessionStorage用于本地存储一个会话（session）中的数据，这些数据只有在同一个会话中的页面才能访问并且当会话结束后数据也随之销毁。因此sessionStorage不是一种持久化的本地存储，仅仅是会话级别的存储。

而localStorage用于持久化的本地存储，除非主动删除数据，否则数据是永远不会过期的。

## 如何实现浏览器内多个标签页之间的通信?

WebSocket、SharedWorker；

也可以调用localstorge、cookies等本地存储方式；

localstorge另一个浏览上下文里被添加、修改或删除时，它都会触发一个事件，

我们通过监听事件，控制它的值来进行页面信息通信；

注意quirks：Safari 在无痕模式下设置localstorge值时会抛出 QuotaExceededError 的异常；

## web storage和cookie的区别

Web Storage的概念和cookie相似，区别是它是为了更大容量存储设计的。Cookie的大小是受限的，并且每次你请求一个新的页面的时候Cookie都会被发送过去，这样无形中浪费了带宽，另外cookie还需要指定作用域，不可以跨域调用。

除此之外，Web Storage拥有setItem,getItem,removeItem,clear等方法，不像cookie需要[**前端开发**](http://lib.csdn.net/base/javascript)者自己封装setCookie，getCookie。

但是cookie也是不可以或缺的：cookie的作用是与服务器进行交互，作为HTTP规范的一部分而存在 ，而Web Storage仅仅是为了在本地“存储”数据而生

浏览器的支持除了IE７及以下不支持外，其他标准浏览器都完全支持(ie及FF需在web服务器里运行)，值得一提的是IE总是办好事，例如IE7、IE6中的userData其实就是javascript本地存储的解决方案。通过简单的代码封装可以统一到所有的浏览器都支持web storage。

localStorage和sessionStorage都具有相同的操作方法，例如setItem、getItem和removeItem等

## 请描述一下 cookies，sessionStorage 和 localStorage 的区别

cookie是网站为了标示用户身份而储存在用户本地终端（Client Side）上的数据（通常经过加密）。

cookie数据始终在同源的http请求中携带（即使不需要），记会在浏览器和服务器间来回传递。

sessionStorage和localStorage不会自动把数据发给服务器，仅在本地保存。

**存储大小：**

cookie数据大小不能超过**4k**。

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**。

**有期时间：**

localStorage 存储持久数据，浏览器关闭后数据不丢失除非主动删除数据；

sessionStorage 数据在当前浏览器窗口关闭后自动删除。

cookie 设置的cookie过期时间之前一直有效，即使窗口或浏览器关闭

## 请你谈谈Cookie的弊端

cookie虽然在持久保存客户端数据提供了方便，分担了服务器存储的负担，但还是有很多局限性的。

（1） IE6或更低版本最多20个cookie

（2） IE7和之后的版本最后可以有50个cookie。

（3） Firefox最多50个cookie

（4） chrome和Safari没有做硬性限制

IE和Opera会清理近期最少使用的cookie，Firefox会随机清理cookie。

cookie的最大大约为4096字节，为了兼容性，一般不能超过4095字节。

IE 提供了一种存储可以持久化用户数据，叫做userdata，从IE5.0就开始支持。每个数据最多128K，每个域名下最多1M。这个持久化数据放在缓存中，如果缓存没有清理，那么会一直存在。

**优点：**极高的扩展性和可用性

（1）通过良好的编程，控制保存在cookie中的session对象的大小。

（2）通过加密和安全传输技术（SSL），减少cookie被破解的可能性。

（3）只在cookie中存放不敏感数据，即使被盗也不会有重大损失。

（4）控制cookie的生命期，使之不会永远有效。偷盗者很可能拿到一个过期的cookie。

**缺点：**

（1） `Cookie`数量和长度的限制。每个domain最多只能有20条cookie，每个cookie长度不能超过4KB，否则会被截掉。

（2）安全性问题。如果cookie被人拦截了，那人就可以取得所有的session信息。即使加密也与事无补，因为拦截者并不需要知道cookie的意义，他只要原样转发cookie就可以达到目的了。

（3）有些状态不可能保存在客户端。例如，为了防止重复提交表单，我们需要在服务器端保存一个计数器。如果我们把这个计数器保存在客户端，那么它起不到任何作用。

## 如何删除一个cookie

（1）将时间设为当前时间往前一点。

var date = new Date();

date.setDate(date.getDate() - 1);//真正的删除

setDate()方法用于设置一个月的某一天。

（2）expires的设置

document.cookie = 'user='+ encodeURIComponent('name') + ';expires = ' + new Date(0)

## cookie 和session 的区别？

（1） cookie数据存放在客户的浏览器上，session数据放在服务器上。

（2） cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗，考虑到安全应当使用session。

（3） session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能，考虑到减轻服务器性能方面，应当使用COOKIE。

（4）单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

（5）所以个人建议：

将登陆信息等重要信息存放为SESSION

其他信息如果需要保留，可以放在COOKIE中

## 什么是Cookie 隔离？（或者说：请求资源的时候不要让它带cookie怎么做）

如果静态文件都放在主域名下，那静态文件请求的时候都带有的cookie的数据提交给server的，非常浪费流量，所以不如隔离开。

因为cookie有域的限制，因此不能跨域提交请求，故使用非主要域名的时候，请求头中就不会带有cookie数据，这样可以降低请求头的大小，降低请求时间，从而达到降低整体请求延时的目的。

同时这种方式不会将cookie传入Web Server，也减少了Web Server对cookie的处理分析环节，提高了webserver的http请求的解析速度。

## Web Worker 和webSocket

**worker主线程:**

1.通过 worker = new Worker( url ) 加载一个JS文件来创建一个worker，同时返回一个worker实例。

2.通过worker.postMessage( data ) 方法来向worker发送数据。

3.绑定worker.onmessage方法来接收worker发送过来的数据。

4.可以使用 worker.terminate() 来终止一个worker的执行。

**WebSocket**是Web应用程序的传输协议，它提供了双向的，按序到达的数据流。他是一个[**Html5**](http://lib.csdn.net/base/html5)协议，WebSocket的连接是持久的，他通过在客户端和服务器之间保持双工连接，服务器的更新可以被及时推送给客户端，而不需要客户端以一定时间间隔去轮询。

## WEB应用从服务器主动推送Data到客户端有那些方式？区别是什么？

* 传统Ajax短轮询
* Commet：基于HTTP长连接的服务器推送技术
* 基于WebSocket的推送方案
* SSE（Server-Send Event，服务端推送事件）：服务器推送数据新方式

区别：websocket是全双工的，本质上是一个额外的tcp连接，而sse是直接建立在当前http连接上的，本质上是保持一个http长连接，但是和comet不同的是：comet是每次服务端返回数据后，连接关闭然后客户端马上再次发起连接。而sse是保持长连接常驻。

## webSocket如何兼容低浏览器？

* Adobe Flash Socket
* ActiveX HTMLFile (IE)
* 基于 multipart 编码发送 XHR
* 基于长轮询的 XHR

## 一个页面从输入 URL 到页面加载显示完成，这个过程中都发生了什么？

分为4个步骤：

（1）当发送一个URL请求时，不管这个URL是Web页面的URL还是Web页面上每个资源的URL，浏览器都会开启一个线程来处理这个请求，同时在远程DNS服务器上启动一个DNS查询。这能使浏览器获得请求对应的IP地址。

（2）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。该握手包括一个同步报文，一个同步-应答报文和一个应答报文，这三个报文在 浏览器和服务器之间传递。该握手首先由客户端尝试建立起通信，而后服务器应答并接受客户端的请求，最后由客户端发出该请求已经被接受的报文。

（3）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。

（4）此时，`Web`服务器提供资源服务，客户端开始下载资源。

请求返回后，便进入了我们关注的前端模块，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`又可以根据`DOM API`操作`DOM`

## 创建ajax的过程

(1)创建`XMLHttpRequest`对象,也就是创建一个异步调用对象。

(2)创建一个新的`HTTP`请求,并指定该`HTTP`请求的方法、`URL`及验证信息。

(3)设置响应`HTTP`请求状态变化的函数。

(4)发送`HTTP`请求。

(5)获取异步调用返回的数据。

(6)使用JavaScript和DOM实现局部刷新。

var xmlHttp = new XMLHttpRequest();

xmlHttp.open('GET','demo.php','true');

xmlHttp.send()

xmlHttp.onreadystatechange = function(){

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

}

}

## 异步加载和延迟加载

（1）异步加载的方案： 动态插入script标签

（2）通过ajax去获取js代码，然后通过eval执行

（3）script标签上添加defer或者async属性

（4）创建并插入iframe，让它异步执行js

（5）延迟加载：有些 js 代码并不是页面初始化的时候就立刻需要的，而稍后的某些情况才需要的。

## js延迟加载的方式有哪些？

defer和async、动态创建DOM方式（创建script，插入到DOM中，加载完毕后callBack）、按需异步载入js。

## defer和async区别

defer并行加载js文件，会按照页面上script标签的顺序执行

async并行加载js文件，下载完成立即执行，不会按照页面上script标签的顺序执行

## ajax的缺点和在IE下的问题？

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。

IE缓存问题：

在IE浏览器下，如果请求的方法是GET，并且请求的URL不变，那么这个请求的结果就会被缓存。解决这个问题的办法可以通过实时改变请求的URL，只要URL改变，就不会被缓存，可以通过在URL末尾添加上随机的时间戳参数('t'= + new Date().getTime())

或者：

open('GET','demo.php?rand=+Math.random()',true);//

Ajax请求的页面历史记录状态问题

可以通过锚点来记录状态，location.hash。让浏览器记录Ajax请求时页面状态的变化。

还可以通过HTML5的history.pushState，来实现浏览器地址栏的无刷新改变

## Flash、Ajax各自的优缺点，在使用中如何取舍？

* Flash适合处理多媒体、矢量图形、访问机器；对CSS、处理文本上不足，不容易被搜索。Ajax对CSS、文本支持很好，支持搜索；多媒体、矢量图形、机器访问不足。
* 共同点：与服务器的无刷新传递消息、用户离线和在线状态、操作DOM

## 说说你对Promise的理解

依照Promise/A+ 的定义，Promise 有四种状态：

pending: 初始状态, 非 fulfilled 或 rejected.

fulfilled: 成功的操作.

rejected: 失败的操作.

settled: Promise已被fulfilled或rejected，且不是pending

另外，fulfilled与rejected一起合称settled。

Promise对象用来进行延迟(deferred) 和异步(asynchronous ) 计算。

**Promise 的构造函数**

构造一个 Promise，最基本的用法如下：

var promise = new Promise(function(resolve, reject) {

if (...) { // succeed

resolve(result);

} else { // fails

reject(Error(errMessage));

}

});

Promise实例拥有 then 方法（具有 then 方法的对象，通常被称为 thenable）。它的使用方法如下：promise.then(onFulfilled, onRejected)

接收两个函数作为参数，一个在 fulfilled 的时候被调用，一个在 rejected 的时候被调用，接收参数就是 future，onFulfilled对应 resolve, onRejected 对应 reject。

## 请解释一下 JavaScript 的同源策略。

概念：同源策略是客户端脚本（尤其是Javascript）的重要的安全度量标准。它最早出自Netscape Navigator2.0，其目的是防止某个文档或脚本从多个不同源装载。

这里的同源策略指的是：协议，域名，端口相同，同源策略是一种安全协议。

指一段脚本只能读取来自同一来源的窗口和文档的属性。

## 为什么要有同源限制？

我们举例说明：比如一个黑客程序，他利用Iframe把真正的银行登录页面嵌到他的页面上，当你使用真实的用户名，密码登录时，他的页面就可以通过Javascript读取到你的表单中input中的内容，这样用户名，密码就轻松到手了。

## 常见兼容性问题？

png24位的图片在iE6浏览器上出现背景，解决方案是做成PNG8.也可以引用一段脚本处理.

浏览器默认的margin和padding不同。解决方案是加一个全局的\*{margin:0;padding:0;}来统一。

IE6双边距bug:块属性标签float后，又有横行的margin情况下，在ie6显示margin比设置的大。

浮动ie产生的双倍距离（IE6双边距问题：在IE6下，如果对元素设置了浮动，同时又设置了margin-left或margin-right，margin值会加倍。）

#box{ float:left; width:10px; margin:0 0 0 100px;}

这种情况之下IE会产生20px的距离，解决方案是在float的标签样式控制中加入

\_display:inline;将其转化为行内属性。(\_这个符号只有ie6会识别)

渐进识别的方式，从总体中逐渐排除局部。

首先，巧妙的使用“\9”这一标记，将IE游览器从所有情况中分离出来。

接着，再次使用“+”将IE8和IE7、IE6分离开来，这样IE8已经独立识别。

css

.bb{

background-color:#f1ee18;/\*所有识别\*/

.background-color:#00deff\9; /\*IE6、7、8识别\*/

+background-color:#a200ff;/\*IE6、7识别\*/

\_background-color:#1e0bd1;/\*IE6识别\*/

}

怪异模式问题：漏写DTD声明，Firefox仍然会按照标准模式来解析网页，但在IE中会触发

怪异模式。为避免怪异模式给我们带来不必要的麻烦，最好养成书写DTD声明的好习惯。现在

可以使用[html5](http://www.w3.org/TR/html5/single-page.html)推荐的写法：`<doctype html>`

上下margin重合问题：

ie和ff都存在，相邻的两个div的margin-left和margin-right不会重合，但是margin-top和margin-bottom却会发生重合。

解决方法，养成良好的代码编写习惯，同时采用margin-top或者同时采用margin-bottom。

## 说说你知道的移动端web的兼容性bug

（1）一些情况下对非可点击元素如(label,span)监听click事件，ios下不会触发，css增加cursor:pointer就搞定了。

（2）position 在Safari下的两个定位需要都写，只写一个容易发生错乱

（3）Input 的placeholder会出现文本位置偏上的情况

input 的placeholder会出现文本位置偏上的情况：PC端设置line-height等于height能够对齐，而移动端仍然是偏上，解决是设置line-height：normal

（4）zepto点击穿透问题

引入fastclick解决；event.preventDefault

（5）当输入框在最底部的时候，弹起的虚拟键盘会把输入框挡住。

Element.scrollIntoViewIfNeeded(opt\_center)

## 列举IE 与其他浏览器不一样的特性？

* IE支持currentStyle，FIrefox使用getComputStyle
* IE 使用innerText，Firefox使用textContent
* 滤镜方面：IE:filter:alpha(opacity= num)；Firefox：-moz-opacity:num
* 事件方面：IE：attachEvent：火狐是addEventListener
* 鼠标位置：IE是event.clientX；火狐是event.pageX
* IE使用event.srcElement；Firefox使用event.target
* IE中消除list的原点仅需margin:0即可达到最终效果；FIrefox需要设置margin:0;padding:0以及list-style:none
* CSS圆角：ie7以下不支持圆角

## 事件、IE与火狐的事件机制有什么区别？ 如何阻止冒泡？

（1）我们在网页中的某个操作（有的操作对应多个事件）。例如：当我们点击一个按钮就会产生一个事件。是可以被 Javascript 侦测到的行为。

（2）事件处理机制：IE是事件冒泡、firefox同时支持两种事件模型，也就是：捕获型事件和冒泡型事件。；

（3）`ev.stopPropagation()`;注意旧ie的方法 `ev.cancelBubble = true`;

## ie各版本和chrome可以并行下载多少个资源

IE6 两个并发，iE7升级之后的6个并发，之后版本也是6个

Firefox，chrome也是6个

## git fetch和git pull的区别

git pull：相当于是从远程获取最新版本并merge到本地

git fetch：相当于是从远程获取最新版本到本地，不会自动merge

## attribute和property的区别是什么？

attribute是dom元素在文档中作为html标签拥有的属性；

property就是dom元素在js中作为对象拥有的属性。

所以：

对于html的标准属性来说，attribute和property是同步的，是会自动更新的，

但是对于自定义的属性来说，他们是不同步的。

## exports和module.exports区别

exports 是 module.exports 的一个引用 module.exports 初始值为一个空对象 {}，所以 exports 初始值也是 {} require 引用模块后，返回的是 module.exports 而不是 exports。

## 单线程优点

Node.js依托于v8引擎，都是以单线程为基础的。单线程资源占用小。单线程避免了传统PHP那样频繁创建、切换线程的开销，使执行速度更加迅速。

## Node.js是如何做到I/O的异步和非阻塞的？

其实Node在底层访问I/O还是多线程的。Node可以借助livuv来来实现多线程。

如果我们非要让Node.js支持多线程，还是提倡使用官方的做法，利用libuv库来实现。

cluster可以用来让Node.js充分利用多核cpu的性能。

## 并行与并发，进程与线程

并发 (Concurrent) = 2 队列对应 1 咖啡机。

并行 (Parallel) = 2 队列对应 2 咖啡机。

线程是进程下的执行者，一个进程至少会开启一个线程（主线程），也可以开启多个线程。

## 谈谈Nodejs优缺点

**优点：**

* + 事件驱动，异步编程，占用内存少
  + npm设计得好

**缺点：**

* + Debug 很困难。没有 stack trace，出了问题很难查找问题的原因；
  + 如果设计不好，很容易让代码充满 callback，代码不优雅；
  + 可靠性低；
  + 单进程，单线程，只支持单核CPU，不能充分的利用多核CPU服务器。

# 测试相关问题

## 单元测试/集成测试/系统测试的区别是什么？

（1）单元测试：单元测试是对软件基本组成单元（软件设计的最小单位）进行正确性检验的测试工作，如函数、过程(function,procedure)或一个类的方法(method)。

（2）集成测试：集成测试是在单元测试的基础上，将所有模块按照概要设计要求组装成为子系统或系统，验证组装后功能以及模块间接口是否正确的测试工作。集成测试也叫组装测试、联合测试、子系统测试或部件测试。

（3）系统测试：系统测试是将已经集成好的软件系统，作为整个基于计算机系统的一个元素，与计算机硬件、外设、某些支持软件、数据和人员等其他系统元素结合在一起，在实际使用环境下，对计算机系统进行一系列的组装测试和确认测试的工作。

**区别：**

|  |  |  |
| --- | --- | --- |
| 测试阶段 | 测试方法 | 评估基准对应阶段 |
| 单元测试 | 白盒测试，单元内部的数据结构，逻辑控制，异常处理等 | 逻辑覆盖率，详细设计说明书 |
| 集成测试 | 灰盒测试，模块间接口以及模块组合后的整体功能 | 接口覆盖率，概要设计说明书 |
| 系统测试 | 黑盒测试，整个系统对需求的符合度 | 测试用例对需求的覆盖率，需求说明书 |

# 效能相关问题

## 前端性能优化

参考：<https://csspod.com/frontend-performance-best-practices/>

**1、页面内容：**（1）减少 HTTP 请求数；（2）减少 DNS 查询；（3）避免重定向；（4）缓存 Ajax 请求；（5）延迟加载；（6）预先加载；（7）减少 DOM 元素数量；（8）划分内容到不同域名；（9）尽量减少 iframe 使用；（10）避免 404 错误。

**2、服务器：**（1）使用 CDN；（2）添加 Expires 或 Cache-Control 响应头；（3）启用 Gzip；（4）配置 Etag；（5）尽早输出缓冲；（6）Ajax 请求使用 GET 方法；（7）避免图片 src 为空。

**3、Cookie：**（1）减少 Cookie 大小；（2）静态资源使用无 Cookie 域名。

**4、CSS：**（1）把样式表放在 <head> 中；（2）不要使用 CSS 表达式；（3）使用 <link> 替代 @import；（4）不要使用 filter。

**5、JavaScript：**（1）把脚本放在页面底部；（2）使用外部 JavaScript 和 CSS；（3）压缩 JavaScript 和 CSS；（4）移除重复脚本；（5）减少 DOM 操作；（6）使用高效的事件处理。

**6、图片：**（1）优化图片；（2）优化 CSS Sprite；（3）不要在 HTML 中缩放图片；（4）使用体积小、可缓存的 favicon.ico。

**7、移动端：**（1）保持单个文件小于 25 KB；（2）打包内容为分段（multipart）文档。

前端性能的一个重要指标是页面加载时间，不仅事关用户体验，也是搜索引擎排名考虑的一个因素。

**或：**

代码层面：避免使用css表达式，避免使用高级选择器，通配选择器。

缓存利用：缓存Ajax，使用CDN，使用外部js和css文件以便缓存，添加Expires头，服务端配置Etag，减少DNS查找等

请求数量：合并样式和脚本，使用css图片精灵，初始首屏之外的图片资源按需加载，静态资源延迟加载。

请求带宽：压缩文件，开启GZIP，

代码层面的优化

* 用hash-table来优化查找
* 少用全局变量
* 用innerHTML代替DOM操作，减少DOM操作次数，优化javascript性能
* 用setTimeout来避免页面失去响应
* 缓存DOM节点查找的结果
* 避免使用CSS Expression
* 避免全局查询
* 避免使用with(with会创建自己的作用域，会增加作用域链长度)
* 多个变量声明合并
* 避免图片和iFrame等的空Src。空Src会重新加载当前页面，影响速度和效率
* 尽量避免写在HTML标签中写Style属性

## 移动端性能优化

* 尽量使用css3动画，开启硬件加速。
* 适当使用touch事件代替click事件。
* 避免使用css3渐变阴影效果。
* 可以用transform: translateZ(0)来开启硬件加速。
* 不滥用Float。Float在渲染时计算量比较大，尽量减少使用
* 不滥用Web字体。Web字体需要下载，解析，重绘当前页面，尽量减少使用。
* 合理使用requestAnimationFrame动画代替setTimeout
* CSS中的属性（CSS3 transitions、CSS3 3D transforms、Opacity、Canvas、WebGL、Video）会触发GPU渲染，请合理使用。过渡使用会引发手机过耗电增加
* PC端的在移动端同样适用

## 前端长列表的性能优化

只渲染页面用用户能看到的部分。并且在不断滚动的过程中去除不在屏幕中的元素，不再渲染，从而实现高性能的列表渲染。

借鉴着这个想法，我们思考一下。当列表不断往下拉时，web中的dom元素就越多，即使这些dom元素已经离开了这个屏幕，不被用户所看到了，这些dom元素依然存在在那里。导致浏览器在渲染时需要不断去考虑这些dom元素的存在，造成web浏览器的长列表渲染非常低效。因此，实现的做法就是捕捉scroll事件，当dom离开屏幕，用户不再看到时，就将其移出dom tree。

# 安全相关问题

## 哪些操作会造成内存泄漏？

内存泄漏指任何对象在您不再拥有或需要它之后仍然存在。

垃圾回收器定期扫描对象，并计算引用了每个对象的其他对象的数量。如果一个对象的引用数量为 0（没有其他对象引用过该对象），或对该对象的惟一引用是循环的，那么该对象的内存即可回收。

setTimeout 的第一个参数使用字符串而非函数的话，会引发内存泄漏。

闭包、控制台日志、循环（在两个对象彼此引用且彼此保留时，就会产生一个循环）

## sql注入原理

就是通过把SQL命令插入到Web表单递交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令。总的来说有以下几点：

（1）永远不要信任用户的输入，要对用户的输入进行校验，可以通过正则表达式，或限制长度，对单引号和双"-"进行转换等。

（2）永远不要使用动态拼装SQL，可以使用参数化的SQL或者直接使用存储过程进行数据查询存取。

（3）永远不要使用管理员权限的数据库连接，为每个应用使用单独的权限有限的数据库连接。

（4）不要把机密信息明文存放，请加密或者hash掉密码和敏感的信息。

## 什么是Token？Token有什么作用？作用实现的思路与原理是什么？

Token，即口令、令牌，最大的特点就是随机性，不可预测。一般黑客或软件无法猜测出来。

**Token的作用：**

* + 防止表单重复提交
  + 防止CSRF攻击
  + 用于签名验证

简单来讲，Token的主要作用就是验证，以上三个其实核心都是验证，相信接触到Token的人，对Token验证都不陌生了，不管是在支付的时候用的Token，还是微信用的，或者是网站请求时候用的，简单说Token就是用来验证的。

原理上都是通过session token来实现的。当客户端请求页面时，服务器会生成一个随机数Token，并且将Token放置到session当中，然后将Token发给客户端（一般通过构造hidden表单）。下次客户端提交请求时，Token会随着表单一起提交到服务器端。

然后，如果应用于"**anti csrf攻击**"，则服务器端会对Token值进行验证，判断是否和session中的Token值相等，若相等，则可以证明请求有效，不是伪造的。

不过，如果应用于"**防止表单重复提交**"，服务器端第一次验证相同过后，会将session中的Token值更新下，若用户重复提交，第二次的验证判断将失败，因为用户提交的表单中的Token没变，但服务器端session中Token已经改变了。

上面的session应用相对安全，但也叫繁琐，同时当多页面多请求时，必须采用多Token同时生成的方法，这样占用更多资源，执行效率会降低。因此，也可用cookie存储验证信息的方法来代替session Token。比如，应对"重复提交"时，当第一次提交后便把已经提交的信息写到cookie中，当第二次提交时，由于cookie已经有提交记录，因此第二次提交会失败。

不过，cookie存储有个致命弱点，如果cookie被劫持（xss攻击很容易得到用户cookie），那么又一次gameover。黑客将直接实现csrf攻击。

所以，安全和高效相对的。具体问题具体对待吧。

此外，要避免**"加token但不进行校验"**的情况，在session中增加了token，但服务端没有对token进行验证，根本起不到防范的作用。

还需注意的是，**对数据库有改动的增删改操作，需要加token验证，对于查询操作，一定不要加token**，防止攻击者通过查询操作获取token进行csrf攻击。但并不是这样攻击者就无法获得token，只是增大攻击成本而已。

## 如何生成Token？

**1、token的设计目的：**

因为APP端没有和PC端一样的session机制，所以无法判断用户是否登陆，以及无法保持用户状态，所以就需要一种机制来实现session，这就是token的作用，token是用户登陆的唯一票据，只要APP传来的token和服务器端一致，就能证明你已经登陆（就和你去看电影一样，需要买票，拿着票就能进了）

**2、token设计时的种类：**

（1）第三方登陆型：这种token形如微信的access\_token，设计原理是按照OAuth2.0来的，其特点是定时刷新（比如两小时刷新），目的是因为数据源将登陆权限赋予第三方服务器时必须要控制其有效期和权限，要不然第三方服务器可以不经过用户同意，无限期从数据源服务器获取用户任意数据

（2）APP自用登陆型：这种token就是一般的APP用的token，因为不经过第三方，而是用户直接取数据源服务器数据，所以设计比较随意，只需要保证其token的唯一性就行

**3、Token构成：**

从需求功能上来讲，为了防止CSRF工具，token需要具有不重复，另外，还含有特定的功能信息，比如过期时间戳。

下面的图描述了一个token的数据构成：

-----------------------------------------------------------------------------

| msg | separator | signature |

-----------------------------------------------------------------------------

| key | timestamp | . | Base64(sha256(msg)) |

----------------------------------------------------------------------------

token由三部分组成：a).msg b). separator c).signature。

a). msg部分：而msg本身也有两部分组成：一部分，随机字符的主体，另一部分是过期时间戳。

b). 分隔符号：用符号分隔msg部分，和加密后生成的signature签名部分，这里用的是”.“

c). 签名signature。signature签名，是对上面提到的msg，按照msg中提到的msg的信息部分，按照特定的秘锁进行加密。

token = base64(msg)格式化.base64(sha256("秘锁", msg))

**4、Token的加密。**首先，是按照合适得加密方法对数据进行加密。这里我们通用的就使用了sha256散列算法，然后进行BASE64的格式转换。然后，我们需要在token串中隐含过期时间的设定，从需求上讲，每条与服务器交互的token有是有过期时间的，超过这个时间范围，就无效了，需要重新从服务器中取得。

**5、.Token的验证。**

当用户从客户端，得到了token,再次提交给服务器的时候，服务器需要判断token的有效性，否则不加判断直接处理数据，token的生成就无意义了。

验证的过程是：

a). token解包。

先把接受到的token，进行分解。“.”为分隔符，分为msg部分+signature签名部分。

b). 比对签名。

对msg部分进行base64解码, decode\_base64(msg)然后在对解码后的msg明文，进行同样的encode\_base64(sha256(msg))加密。秘锁相同，然后，判断加密后的数据和客户端传过来的token.signature的部分是否一致。如果一致，说明这个token是有效的。

c). 判断时间过期。如果是有效的,取出msg.timestamp，和当前系统时间进行比较，如果过期时间小于当前时间，那这个token是过期的，需要重新的取得token。

## XSS 原理是什么？怎么防范呢？请写一个攻击和防御的例子。CSRF 和 hash 碰撞的原理是什么？怎么防范？XSS与CSRF有什么区别吗？

**XSS原理**

XSS(cross-site scripting)攻击指的是攻击者往Web页面里插入恶意 html标签或者javascript代码。比如：攻击者在论坛中放一个看似安全的链接，骗取用户点击后，窃取cookie中的用户私密信息；或者攻击者在论坛中加一个恶意表单，当用户提交表单的时候，却把信息传送到攻击者的服务器中，而不是用户原本以为的信任站点。

**XSS防范方法**

（1）代码里对用户输入的地方和变量都需要仔细检查长度和对”<”,”>”,”;”,”’”等字符做过滤；其次任何内容写到页面之前都必须加以encode，避免不小心把html tag 弄出来。这一个层面做好，至少可以堵住超过一半的XSS 攻击。

（2）避免直接在cookie 中泄露用户隐私，例如email、密码等等。

（3）通过使cookie 和系统ip 绑定来降低cookie 泄露后的危险。这样攻击者得到的cookie 没有实际价值，不可能拿来重放。如果网站不需要再浏览器端对cookie 进行操作，可以在Set-Cookie 末尾加上HttpOnly 来防止javascript 代码直接获取cookie 。

（4）尽量采用POST 而非GET 提交表单

**CSRF攻击原理及过程如下：**

（1）用户C打开浏览器，访问受信任网站A，输入用户名和密码请求登录网站A；

（2）在用户信息通过验证后，网站A产生Cookie信息并返回给浏览器，此时用户登录网站A成功，可以正常发送请求到网站A；

（3）用户未退出网站A之前，在同一浏览器中，打开一个TAB页访问网站B；

（4）网站B接收到用户请求后，返回一些攻击性代码，并发出一个请求要求访问第三方站点A；

（5）浏览器在接收到这些攻击性代码后，根据网站B的请求，在用户不知情的情况下携带Cookie信息，向网站A发出请求。网站A并不知道该请求其实是由B发起的，所以会根据用户C的Cookie信息以C的权限处理该请求，导致来自网站B的恶意代码被执行。

**hash 碰撞的原理**

由于Hash固定长度输出的特性，必然会存在多个不同输入产生相同输出的情况。如果两个输入串的hash函数的值一样，则称这两个串是一个碰撞(Collision)。在理论范围内，存在一个输出串对应无穷多个输入串，所以碰撞具有其必然性。如果找到碰撞，那么意味着我们可以破坏信息的一致性而不被接收方察觉，搜寻指定输入的Hash碰撞值的过程被称作“Hash破解”。

**CSRF的防御**

目前防御 CSRF 攻击主要有三种策略：

（1）验证 HTTP Referer 字段；

（2）在请求地址中添加 token 并验证；

（3）在 HTTP 头中自定义属性并验证。

**XSS与CSRF有什么区别吗？**

CSRF（Cross-site request forgery）跨站请求伪造，也被称为“One Click Attack”或者Session Riding，通常缩写为CSRF或者XSRF，是一种对网站的恶意利用。尽管听起来像跨站脚本（XSS），但它与XSS非常不同，XSS利用站点内的信任用户，而CSRF则通过伪装来自受信任用户的请求来利用受信任的网站。与XSS攻击相比，CSRF攻击往往不大流行（因此对其进行防范的资源也相当稀少）和难以防范，所以被认为比XSS更具危险性。

XSS是获取信息，不需要提前知道其他用户页面的代码和数据包。CSRF是代替用户完成指定的动作，需要知道其他用户页面的代码和数据包。

要完成一次CSRF攻击，受害者必须依次完成两个步骤：

（1）登录受信任网站A，并在本地生成Cookie。

（2）在不登出A的情况下，访问危险网站B。

# 网络相关问题

## HTTP请求报文和HTTP响应报文？

1. **请求报文**

**（1）请求行：**

　　　　请求行由请求方法字段、URL字段和HTTP协议版本字段3个字段组成，它们用空格分隔。

　　　　例如打开路径http://www.baidu.com/index.php，用火狐浏览器可以查看到请求报文为：

　　　　GET /index.php HTTP/1.1

　　　　因此用空格分隔之后得到的信息为

　　　　(1)请求方法: GET

　　　　(2)URL信息: /index.php

　　　　(3)HTTP协议版本: HTTP/1.1

　　　　请求方法的类型：

　　　　1、Get：Get方法GET方法要求服务器将URL定位的资源放在响应报文的数据部分，回送给客户端。使用GET方法时，请求参数和对应的值附加在URL后面，利用一个问号（“?”）代表URL的结尾与请求参数的开始，传递参数长度受限制，各个数据之间用”&”符号隔开，传送的数据直接在浏览器的地址条里就能够看到。显然，这种方式不适合传送私密数据。另外，由于不同的浏览器对地址的字符限制也有所不同，一般最多只能识别1024个字符，所以如果需要传送大量数据的时候，也不适合使用GET方式。

　　　　2、Post：POST方法将请求参数封装在HTTP请求数据中，以名称/值的形式出现，可以传输大量数据，POST方式对传送的数据大小没有限制，而且也不会显示在URL中。

　　　　Get和Post一个各有优缺点， Get方法请求的数据，都一样，用户可以随意复制路径分享给其它人，Post方法不限制请求数据量，常用于表单提交。Post方法比Get方法安全性强。

　　　　3、HEAD：就像GET，只不过服务端接受到HEAD请求后只返回响应头，而不会发送响应内容。当我们只需要查看某个页面的状态的时候，使用HEAD是非常高效的，因为在传输的过程中省去了页面内容。也就是说，该方法只用于测试一下当前服务器的状态。

**（2）请求头部：**

　　　　User-Agent：浏览器的具体类型　　如：User-Agent：Mozilla/5.0 (Windows NT 6.1; rv:17.0) Gecko/20100101 Firefox/17.0

　 Accept：浏览器支持哪些数据类型　　如：Accept: text/html, application/xml;q=0.9,\*/\*;q=0.8

　　　　Accept-Charset：浏览器采用的是哪种编码　　如：Accept-Charset: ISO-8859-1

　　　　Accept-Encoding：浏览器支持解码的数据压缩格式　　如：Accept-Encoding: gzip, deflate

　　　　Accept-Language：浏览器的语言环境　　如：Accept-Language zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3

　　　 Host：请求的主机名，允许多个域名同处一个IP地址，即虚拟主机。Host:www.baidu.com

　　　 Connection：表示是否需要持久连接。Keep-Alive/close，HTTP1.1默认是持久连接，它可以利用持久连接的优点，当页面包含多个元素时（例如Applet，图片），显著地减少下载所需要的时间。要实现这一点，Servlet需要在应答中发送一个Content-Length头，最简单的实现方法是：先把内容写入ByteArrayOutputStream，然后在正式写出内容之前计算它的大小。如：Connection: Keep-Alive

　　　　Content-Length：表示请求消息正文的长度。对于POST请求来说Content-Length必须出现。

　　　　Content-Type：WEB服务器告诉浏览器自己响应的对象的类型和字符集。例如：Content-Type: text/html; charset='gb2312'

　　　　Content-Encoding：WEB服务器表明自己使用了什么压缩方法（gzip，deflate）压缩响应中的对象。例如：Content-Encoding：gzip

　　　　Content-Language：WEB服务器告诉浏览器自己响应的对象的语言。

　　　　Cookie：最常用的请求头，浏览器每次都会将cookie发送到服务器上，允许服务器在客户端存储少量数据。

　　　 Referer：包含一个URL，用户从该URL代表的页面出发访问当前请求的页面。服务器能知道你是从哪个页面过来的。Referer: <http://www.baidu.com/>

1. **响应报文**

　HTTP响应报文与HTTP请求报文是对应的，也是分为三个部分。

　　1、响应行：主要是设置响应状态等信息。

　　2、响应头：而设置Cookie，缓存等信息就是在响应头属性设置的。

3、响应体：如果请求的是HTML页面，那么返回的就是HTML代码。如果是JS就是JS代码。

HTTP响应报文常用属性：

**Cache-Control**

响应输出到客户端后，服务端通过该报文头属告诉客户端如何控制响应内容的缓存。

下面，的设置让客户端对响应内容缓存3600秒，也即在3600秒内，如果客户再次访问该资源，直接从客户端的缓存中返回内容给客户，不要再从服务端获取（当然，这个功能是靠客户端实现的，服务端只是通过这个属性提示客户端“应该这么做”，做不做，还是决定于客户端，如果是自己宣称支持HTTP的客户端，则就应该这样实现）。

Cache-Control: max-age=3600

**ETag**

一个代表响应服务端资源（如页面）版本的报文头属性，如果某个服务端资源发生变化了，这个ETag就会相应发生变化。它是Cache-Control的有益补充，可以让客户端“更智能”地处理什么时候要从服务端取资源，什么时候可以直接从缓存中返回响应。

ETag: "737060cd8c284d8af7ad3082f209582d"

**Location**

我们在Asp.net中让页面Redirect到一个某个A页面中，其实是让客户端再发一个请求到A页面，这个需要Redirect到的A页面的URL，其实就是通过响应报文头的Location属性告知客户端的，如下的报文头属性，将使客户端redirect到iteye的首页中：

Location: http://www.google.com.hk

**Set-Cookie**

服务端可以设置客户端的Cookie，其原理就是通过这个响应报文头属性实现的。

Set-Cookie: UserID=JohnDoe; Max-Age=3600; Version=1

**Server**

Server响应报头域包含了服务器用来处理请求的软件信息

**Last-Modified**

Last-Modified实体报头域用于指示资源的最后修改日期和时间。

**Expires**

Expires实体报头域给出响应过期的日期和时间。为了让代理服务器或浏览器在一段时间以后更新缓存中(再次访问曾访问过的页面时，直接从缓存中加载，缩短响应时间和降低服务器负载)的页面，我们可以使用Expires实体报头域指定页面过期的时间。eg：Expires：Thu，15 Sep 2006 16:23:12 GMT

## 什么是Etag？

当发送一个服务器请求时，浏览器首先会进行缓存过期判断。浏览器根据缓存过期时间判断缓存文件是否过期。

情景一：若没有过期，则不向服务器发送请求，直接使用缓存中的结果，此时我们在浏览器控制台中可以看到 200 OK(from cache) ，此时的情况就是完全使用缓存，浏览器和服务器没有任何交互的。

情景二：若已过期，则向服务器发送请求，此时请求中会带上①中设置的文件修改时间，和Etag

然后，进行资源更新判断。服务器根据浏览器传过来的文件修改时间，判断自浏览器上一次请求之后，文件是不是没有被修改过；根据Etag，判断文件内容自上一次请求之后，有没有发生变化

情形一：若两种判断的结论都是文件没有被修改过，则服务器就不给浏览器发index.html的内容了，直接告诉它，文件没有被修改过，你用你那边的缓存吧—— 304 Not Modified，此时浏览器就会从本地缓存中获取index.html的内容。此时的情况叫协议缓存，浏览器和服务器之间有一次请求交互。

情形二：若修改时间和文件内容判断有任意一个没有通过，则服务器会受理此次请求，之后的操作同①

① 只有get请求会被缓存，post请求不会

## ETag应用

Etag由服务器端生成，客户端通过If-Match或者说If-None-Match这个条件判断请求来验证资源是否修改。常见的是使用If-None-Match。请求一个文件的流程可能如下：

====第一次请求===

1.客户端发起 HTTP GET 请求一个文件；

2.服务器处理请求，返回文件内容和一堆Header，当然包括Etag(例如"2e681a-6-5d044840")(假设服务器支持Etag生成和已经开启了Etag).状态码200

====第二次请求===

客户端发起 HTTP GET 请求一个文件，注意这个时候客户端同时发送一个If-None-Match头，这个头的内容就是第一次请求时服务器返回的Etag：2e681a-6-5d0448402.服务器判断发送过来的Etag和计算出来的Etag匹配，因此If-None-Match为False，不返回200，返回304，客户端继续使用本地缓存；流程很简单，问题是，如果服务器又设置了Cache-Control:max-age和Expires呢，怎么办

答案是同时使用，也就是说在完全匹配If-Modified-Since和If-None-Match即检查完修改时间和Etag之后，

服务器才能返回304.(不要陷入到底使用谁的问题怪圈)

为什么使用Etag请求头?

Etag 主要为了解决 Last-Modified 无法解决的一些问题。

## Expires和Cache-Control

Expires要求客户端和服务端的时钟严格同步。HTTP1.1引入Cache-Control来克服Expires头的限制。如果max-age和Expires同时出现，则max-age有更高的优先级。

Cache-Control: no-cache, private, max-age=0

ETag: abcde

Expires: Thu, 15 Apr 2014 20:00:00 GMT

Pragma: private

Last-Modified: $now // RFC1123 format

## 说说304缓存的原理

服务器首先产生ETag，服务器可在稍后使用它来判断页面是否已经被修改。本质上，客户端通过将该记号传回服务器要求服务器验证其（客户端）缓存。

304是HTTP状态码，服务器用来标识这个文件没修改，不返回内容，浏览器在接收到个状态码后，会使用浏览器已缓存的文件。

客户端请求一个页面（A）。 服务器返回页面A，并在给A加上一个ETag。 客户端展现该页面，并将页面连同ETag一起缓存。 客户再次请求页面A，并将上次请求时服务器返回的ETag一起传递给服务器。 服务器检查该ETag，并判断出该页面自上次客户端请求之后还未被修改，直接返回响应304（未修改——Not Modified）和一个空的响应体。

## 说说网络分层里七层模型是哪七层，每一层的作用是什么？

* 应用层：应用层、表示层、会话层（从上往下）（HTTP、FTP、SMTP、DNS）
* 传输层（TCP和UDP）
* 网络层（IP）
* 物理和数据链路层（以太网）

每一层的作用如下：

* 物理层：通过媒介传输比特,确定机械及电气规范（比特Bit）
* 数据链路层：将比特组装成帧和点到点的传递（帧Frame）
* 网络层：负责数据包从源到宿的传递和网际互连（包PackeT）
* 传输层：提供端到端的可靠报文传递和错误恢复（段Segment）
* 会话层：建立、管理和终止会话（会话协议数据单元SPDU）
* 表示层：对数据进行翻译、加密和压缩（表示协议数据单元PPDU）
* 应用层：允许访问OSI环境的手段（应用协议数据单元APDU）

各种协议：

* ICMP协议： 因特网控制报文协议。它是TCP/IP协议族的一个子协议，用于在IP主机、路由器之间传递控制消息。   
  TFTP协议： 是TCP/IP协议族中的一个用来在客户机与服务器之间进行简单文件传输的协议，提供不复杂、开销不大的文件传输服务。   
  HTTP协议： 超文本传输协议，是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。   
  DHCP协议： 动态主机配置协议，是一种让系统得以连接到网络上，并获取所需要的配置参数手段。

## 说说TCP传输的三次握手四次挥手策略

为了准确无误地把数据送达目标处，TCP协议采用了三次握手策略。用TCP协议把数据包送出去后，TCP不会对传送后的情况置之不理，它一定会向对方确认是否成功送达。握手过程中使用了TCP的标志：SYN和ACK。

发送端首先发送一个带SYN标志的数据包给对方。接收端收到后，回传一个带有SYN/ACK标志的数据包以示传达确认信息。

最后，发送端再回传一个带ACK标志的数据包，代表“握手”结束。

若在握手过程中某个阶段莫名中断，TCP协议会再次以相同的顺序发送相同的数据包。

**断开一个TCP连接则需要“四次挥手”：**

第一次挥手：主动关闭方发送一个FIN，用来关闭主动方到被动关闭方的数据传送，也就是主动关闭方告诉被动关闭方：我已经不 会再给你发数据了(当然，在fin包之前发送出去的数据，如果没有收到对应的ack确认报文，主动关闭方依然会重发这些数据)，但是，此时主动关闭方还可 以接受数据。

第二次挥手：被动关闭方收到FIN包后，发送一个ACK给对方，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号）。

第三次挥手：被动关闭方发送一个FIN，用来关闭被动关闭方到主动关闭方的数据传送，也就是告诉主动关闭方，我的数据也发送完了，不会再给你发数据了。

第四次挥手：主动关闭方收到FIN后，发送一个ACK给被动关闭方，确认序号为收到序号+1，至此，完成四次挥手。

## TCP和UDP的区别

TCP（Transmission Control Protocol，传输控制协议）是基于连接的协议，也就是说，在正式收发数据前，必须和对方建立可靠的连接。一个TCP连接必须要经过三次“对话”才能建立起来

UDP（User Data Protocol，用户数据报协议）是与TCP相对应的协议。它是面向非连接的协议，它不与对方建立连接，而是直接就把数据包发送过去！

UDP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。

## HTTP和HTTPS

HTTP协议通常承载于TCP协议之上，在HTTP和TCP之间添加一个安全协议层（SSL或TSL），这个时候，就成了我们常说的HTTPS。

默认HTTP的端口号为80，HTTPS的端口号为443。

## 为什么HTTPS安全

因为网络请求需要中间有很多的服务器路由器的转发。中间的节点都可能篡改信息，而如果使用HTTPS，密钥在你和终点站才有。https之所以比http安全，是因为他利用ssl/tls协议传输。它包含证书，卸载，流量转发，负载均衡，页面适配，浏览器适配，refer传递等。保障了传输过程的安全性。

## 关于Http 2.0 你知道多少？

HTTP/2引入了“服务端推（server push）”的概念，它允许服务端在客户端需要数据之前就主动地将数据发送到客户端缓存中，从而提高性能。

HTTP/2提供更多的加密支持

HTTP/2使用多路技术，允许多个消息在一个连接上同时交差。

它增加了头压缩（header compression），因此即使非常小的请求，其请求和响应的header都只会占用很小比例的带宽。

## HTTP method 及Restful

REST（Representational State Transfer） REST的意思是表征状态转移，可理解为"表现层状态转化"。是一种基于HTTP协议的网络应用接口风格，充分利用HTTP的方法实现统一风格接口的服务，如果一个架构符合REST原则，就称它为RESTful架构。HTTP定义了以下8种标准的方法(HTTP method)：

**GET（SELECT）：**请求从服务器获取指定资源（一项或多项）。

**POST（CREATE）：**在服务器新建一个资源。起初是用来向服务器输入数据的。实际上，通常会用它来支持HTML的表单。表单中填好的数据通常会被送给服务器，然后由服务器将其发送到要去的地方。

**PUT（UPDATE）：**请求服务器存储一个资源。即在服务器更新资源（客户端提供改变后的完整资源）。让服务器用请求的主体部分来创建一个由所请求的URL命名的新文档，或者，如果那个URL已经存在的话，就用干这个主体替代它。

**PATCH（UPDATE）：**在服务器更新资源（客户端提供改变的属性）。

**DELETE（DELETE）：**请求服务器删除请求URL指定的资源。

**HEAD：**请求指定资源的响应头。即获取资源的元数据。与GET类似，但服务器在响应中值返回首部，不返回实体的主体部分

**TRACE：**会在目的服务器端发起一个环回诊断，最后一站的服务器会弹回一个TRACE响应并在响应主体中携带它收到的原始请求报文。TRACE方法主要用于诊断，用于验证请求是否如愿穿过了请求/响应链。

**OPTIONS：**请求web服务器告知其支持的各种功能。可以查询服务器支持哪些方法或者对某些特殊资源支持哪些方法。

根据REST设计模式，通常实现以下功能： GET（获取），POST（新增），PUT（更新），DELETE（删除）

## GET和POST的区别，何时使用POST？

GET：一般用于信息获取，使用URL传递参数，对所发送信息的数量也有限制，一般在2000个字符

POST：一般用于修改服务器上的资源，对所发送的信息没有限制。

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，也就是说Get是通过地址栏来传值，而Post是通过提交表单来传值。

然而，在以下情况中，请使用 POST 请求：

（1）无法使用缓存文件（更新服务器上的文件或数据库）

（2）向服务器发送大量数据（POST 没有数据量限制）

（3）发送包含未知字符的用户输入时，POST 比 GET 更稳定也更可靠

## 正向代理和反向代理的定义，区别是什么？

**正向代理：**只用于代理内部网络对Internet的连接请求，客户机必须指定代理服务器,并将本来要直接发送到Web服务器上的http请求发送到代理服务器中。

**反向代理（Reverse Proxy）：**是指以代理服务器来接受Internet上的连接请求，然后将请求转发给内部网络上的服务器；并将从服务器上得到的结果返回给Internet上请求连接的客户端，此时代理服务器对外就表现为一个服务器。

正向代理：用浏览器访问时，被残忍的block，于是你可以在国外搭建一台代理服务器，让代理帮我去请求google.com，代理把请求返回的相应结构再返回给我。

反向代理：反向代理服务器会帮我们把请求转发到真实的服务器那里去。Nginx就是性能非常好的反向代理服务器，用来做负载均衡。 正向代理的对象是客户端，反向代理的对象是服务端。

**两者区别：**

**从用途 上来讲：**

正向代理的典型用途是为在防火墙内的局域网客户端提供访问Internet的途径。正向代理还可以使用缓冲特性减少网络使用率。反向代理的典型用途是将 防火墙后面的服务器提供给Internet用户访问。反向代理还可以为后端的多台服务器提供负载平衡，或为后端较慢的服务器提供缓冲服务。

另外，反向代理还可以启用高级URL策略和管理技术，从而使处于不同web服务器系统的web页面同时存在于同一个URL空间下。

**从安全性 来讲：**

正向代理允许客户端通过它访问任意网站并且隐藏客户端自身，因此你必须采取安全措施以确保仅为经过授权的客户端提供服务。反向代理对外都是透明的，访问者并不知道自己访问的是一个代理。

## HTTP状态码

**1\*\*(信息类)：表示接收到请求并且继续处理**

100 Continue 继续，一般在发送post请求时，已发送了http header之后服务端将返回此信息，表示确认，之后发送具体参数信息

101 客户要求服务器根据请求转换HTTP协议版本

**2\*\*(响应成功)：表示动作被成功接收、理解和接受**

200 OK 正常返回信息。表明该请求被成功地完成，所请求的资源发送回客户端

201 Created 请求成功并且服务器创建了新的资源

202 Accepted 服务器已接受请求，但尚未处理

203 返回信息不确定或不完整

204 请求收到，但返回信息为空

205 服务器完成了请求，用户代理必须复位当前已经浏览过的文件

206 服务器已经完成了部分用户的GET请求

**3\*\*(重定向类)：为了完成指定的动作，必须接受进一步处理**

301 Moved Permanently 永久重定向，请求的网页已永久移动到新位置。

302 Found 临时性重定向，资源已临时分配新URI

303 See Other 临时性重定向，且总是使用 GET 请求新的 URI。建议客户访问其他URL或访问方式。

304 Not Modified 自从上次请求后，请求的网页未修改过。服务器返回此响应时，不会返回网页内容，代表上次的文档已经被缓存了，还可以继续使用

305 请求的资源必须从服务器指定的地址得到

306 前一版本HTTP中使用的代码，现行版本中不再使用

307 申明请求的资源临时性删除

**4\*\*(客户端错误类)：请求包含错误语法或不能正确执行**

400 Bad Request (错误请求)服务器无法理解请求的格式，客户端不应当尝试再次使用相同的内容发起请求。

401 Unauthorized (未授权)请求未授权。请求要求身份验证。 对于需要登录的网页，服务器可能返回此响应。这个状态代码必须和WWW-Authenticate报头域一起使用

403 Forbidden (禁止)禁止访问。服务器拒绝请求。

404 Not Found (未找到)找不到如何与 URI 相匹配的资源。服务器找不到请求的网页。

405 (方法禁用) 禁用请求中指定的方法。用户在Request-Line字段定义的方法不允许

406 根据用户发送的Accept拖，请求资源不可访问

407 类似401，用户必须首先在代理服务器上得到授权

408 客户端没有在用户指定的饿时间内完成请求

409 对当前资源状态，请求不能完成

410 服务器上不再有此资源且无进一步的参考地址

411 服务器拒绝用户定义的Content-Length属性请求

412 一个或多个请求头字段在当前请求中错误

413 请求的资源大于服务器允许的大小

414 请求的资源URL长于服务器允许的长度

415 请求资源不支持请求项目格式

416 请求中包含Range请求头字段，在当前请求资源范围内没有range指示值，请求也不包含If-Range请求头字段

417 服务器不满足请求Expect头字段指定的期望值，如果是代理服务器，可能是下一级服务器不能满足请求长。

**5\*\*(服务端错误类)：服务器不能正确执行一个正确的请求**

500 Internal Server Error (服务器内部错误)最常见的服务器端错误。服务器遇到错误，无法完成请求。

501 (尚未实施) 服务器不具备完成请求的功能。 例如，服务器无法识别请求方法时可能会返回此代码。

502 (错误网关) 服务器作为网关或代理，从上游服务器收到无效响应。

503 Service Unavailable (服务不可用) 服务器端暂时无法处理请求（可能是过载或维护）。通常，这只是暂时状态。

504 (网关超时) 服务器作为网关或代理，但是没有及时从上游服务器收到请求。

# 数据库相关问题

## 说说mongoDB和MySQL的区别

[**MySQL**](http://lib.csdn.net/base/mysql)是传统的关系型[**数据库**](http://lib.csdn.net/base/mysql)，[**MongoDB**](http://lib.csdn.net/base/mongodb)则是非关系型数据库

mongodb以BSON结构（二进制）进行存储，对海量数据存储有着很明显的优势。

对比传统关系型数据库,NoSQL有着非常显著的性能和扩展性优势，与关系型数据库相比，MongoDB的优点有：   
①弱一致性（最终一致），更能保证用户的访问速度：   
②文档结构的存储方式，能够更便捷的获取数据。

## MongoDB与Redis的比较

MongoDB和Redis都是NoSQL，采用结构型数据存储。二者在使用场景中，存在一定的区别，这也主要由于二者在内存映射的处理过程，持久化的处理方法不同。

MongoDB建议集群部署，更多的考虑到集群方案，Redis更偏重于进程顺序写入，虽然支持集群，也仅限于主-从模式。

|  |  |  |  |
| --- | --- | --- | --- |
| 比较指标 | MongoDB(v2.4.9) | Redis(v2.4.17) | 比较说明 |
| 实现语言 | c++ | c/c++ | - |
| 协议 | BSON,自定义二进制 | 类telnet | - |
| 性能 | 依赖内存,TPS较高 | 依赖内存,TPS非常高 | Redis优于MongoDB |
| 可操作性 | 丰富的数据表达,索引;最类似于关系型数据库,支持丰富的查询语句 | 数据丰富,较少的IO | MongoDB优于Redis |
| 内存及存储 | 适合大数据量存储,依赖系统虚拟内存,采用镜像文件存储;内存占用率比较高,官方建议独立部署在64位系统 | Redis2.0后支持虚拟内存特性(VM) 突破物理内存限制;数据可以设置时效性,类似于memcache | 不同的应用场景,各有千秋 |
| 可用性 | 支持master-slave,replicatset(内部采用paxos选举算法,自动故障恢复),auto sharding机制,对客户端屏蔽了故障转移和切片机制 | 依赖客户端来实现分布式读写;主从复制时,每次从节点重新连接主节点都要依赖整个快照,无增量复制;不支持auto sharding,需要依赖程序设定一致性hash机制 | MongoDB优于Redis；单点问题上,MongoDB应用简单,相对用户透明,Redis比较复杂,需要客户端主动解决.(MongoDB一般使用replicasets和sharding相结合,replicasets侧重高可用性以及高可靠,sharding侧重性能,水平扩展) |
| 可靠性 | 从1.8版本后,采用binlog方式(类似Mysql) 支持持久化 | 依赖快照进行持久化;AOF增强可靠性;增强性的同时,影响访问性能 |  |
| 一致性 | 不支持事务,靠客户端保证 | 支持事务,比较脆,仅能保证事务中的操作按顺序执行 | Redis优于MongoDB |
| 数据分析 | 内置数据分析功能(mapreduce) | 不支持 | MongoDB优于Redis |
| 应用场景 | 海量数据的访问效率提升 | 较小数据量的性能和运算 | MongoDB优于Redis |

# 代码相关问题

## 实现一个函数clone，可以对JavaScript中的5种主要的数据类型（包括Number、String、Object、Array、Boolean）进行值复制

Object.prototype.clone = function(){

var o = this.constructor === Array ? [] : {};

for(var e in this){

o[e] = typeof this[e] === "object" ? this[e].clone() : this[e];

}

return o;

}

## 编写一个方法，求一个字符串的字节长度

假设：一个英文字符占用一个字节，一个中文字符占用两个字节

function GetBytes(str){

var len = str.length;

var bytes = len;

for(var i=0; i<len; i++){

if (str.charCodeAt(i) > 255) bytes++;

}

return bytes;

}

alert(GetBytes("你好,as"));

# 趣味问题