# HTML部分

### 简述一下你对HTML语义化的理解？

* 用正确的标签做正确的事情。
* 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;
* 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;
* 搜索引擎的爬虫也依赖于HTML标记来确定上下文和各个关键字的权重，利于SEO;
* 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。

### 你能描述一下当你制作一个网页的工作流程吗？

* 内容分析：分清展现在网络中内容的层次和逻辑关系
* 结构设计：写出合理的html结构代码
* 布局设计：使用html+css进行布局
* 样式设计：首先要使用reset.css
* 交互设计：鼠标特效。行为设计：js代码，ajax页面行为和从服务器获取数据。最后测试兼容性。优化性能

### 你如何对网站的文件和资源进行优化？

* 期待的解决方案包括：
  + 文件合并
  + 文件最小化/文件压缩
  + 使用 CDN 托管
  + 缓存的使用
  + 其他

### 请说出三种减少页面加载时间的方法。（加载时间指感知的时间或者实际加载时间）

* 1). 尽量减少页面中重复的HTTP请求数量
* 2). 服务器开启gzip压缩
* 3). css样式的定义放置在文件头部
* 4). Javascript脚本放在文件末尾
* 5). 压缩Javascript、CSS代码
* 7). 尽可能减少DCOM元素
* 8). 使用多域名负载网页内的多个文件、图片
* 9). 使用CDN
* 10). 在服务器端配置control-cache last-modify-date
* 11). 在服务器配置Entity-Tag if-none-match

### 23条Web性能优化最佳实践和规则

* 1.尽可能减少HTTP请求次数
* 2.使用CDN
* 3.避免使用src和href标签
* 4.加入Expires或Cache-Control Header
* 5.使用Gzip压缩
* 6.在html文件顶部放置样式表
* 7.在html文件底部放置Javascript脚本
* 8.避免使用CSS表达式
* 9.使用外部Javascript和CSS外部文件
* 10.减少使用DNS查找次数
* 11.精简Javascript和CSS
* 12.避免重定向
* 13.移除重复的脚本
* 14.配置ETag
* 15.缓存AJAX
* 16.使用GET完成AJAX请求
* 17.减少DOM元素数量
* 18.避免404
* 19.减少Cookie大小
* 20.使用无Cookie的域
* 21.避免使用滤镜
* 22.不要在HTML中缩放图片
* 23.使用小favicon.ico文件，并让其可缓存

### 如何进行网站性能优化

content方面

* 1. 减少HTTP请求：合并文件、CSS精灵、inline Image
  2. 减少DNS查询：DNS查询完成之前浏览器不能从这个主机下载任何任何文件。方法：DNS缓存、将资源分布到恰当数量的主机名，平衡并行下载和DNS查询
  3. 避免重定向：多余的中间访问
  4. 使Ajax可缓存
  5. 非必须组件延迟加载
  6. 未来所需组件预加载
  7. 减少DOM元素数量
  8. 将资源放到不同的域下：浏览器同时从一个域下载资源的数目有限，增加域可以提高并行下载量
  9. 减少iframe数量
  10. 不要404

Server方面

* 1. 使用CDN
  2. 添加Expires或者Cache-Control响应头
  3. 对组件使用Gzip压缩
  4. 配置ETag
  5. Flush Buffer Early
  6. Ajax使用GET进行请求
  7. 避免空src的img标签
* Cookie方面
  1. 减小cookie大小
  2. 引入资源的域名不要包含cookie
* css方面
  1. 将样式表放到页面顶部
  2. 不使用CSS表达式
  3. 使用不使用@import
  4. 不使用IE的Filter
* Javascript方面
  1. 将脚本放到页面底部
  2. 将javascript和css从外部引入
  3. 压缩javascript和css
  4. 删除不需要的脚本
  5. 减少DOM访问
  6. 合理设计事件监听器
* 图片方面
  1. 优化图片：根据实际颜色需要选择色深、压缩
  2. 优化css精灵
  3. 不要在HTML中拉伸图片
  4. 保证favicon.ico小并且可缓存
* 移动方面
  1. 保证组件小于25k
  2. Pack Components into a Multipart Document

### 在制作一个Web应用或Web站点的过程中，你是如何考虑他的UI、安全性、高性能、SEO、可维护性以及技术因素的？

* UI：界面美观，要有个性，考虑用户使用的逻辑要简单，用起来舒适自由。使用习惯要符合大部分用户的习惯，比如少让用户输入，采用选择的方式，提供搜索和提示功能。
* 安全性：对输入进行有效性验证（非法字符，特殊字符）如PHP中的方法htmlspecialchars()将特殊字符（>）转化为html实体，trim()去掉用户输入的不必要字符，stripslashes()去掉用户输入的反斜杠等等。
* 对交互操作进行身份验证和授权（api-key,authtoken）,异常错误处理（向用户反馈单额错误提示不要让攻击者分析出一些网络环境和配置），内存溢出，注入攻击等。
* 高性能：
  + DNS负载均衡
  + HTTP重定向（通过使客户端重定向，来分散和转移请求压力，比如一些下载服务通常都有几个镜像服务器）
  + 分布式缓存
  + 负载均衡：反向代理负载均衡,
  + 数据库扩展：读写分离，垂直分区，水平分区。
  + SEO:选好关键字，描述语言，修饰性图片换成文本，合理使用h1-h6，对图片添加alt属性，链接添加target属性。
  + 可维护性：代码是否容易被理解，是否容易被修改和增加新的功能，当出现问题时是否能快速定位到问题代码。

### 请尽可能完整得描述下从输入URL到整个网页加载完毕及显示在屏幕上的整个流程

* 1）把URL分割成几个部分：协议、网络地址、资源路径。其中网络地址指示该连接网络上哪一台计算机，可以是域名或者IP地址，可以包括端口号；协议是从该计算机获取资源的方式，常见的是HTTP、FTP，不同协议有不同的通讯内容格式；资源路径指示从服务器上获取哪一项资源。 例如：<http://www.guokr.com/question/554991/> 协议部分：http 网络地址：[www.guokr.com](http://www.guokr.com/) 资源路径：/question/554991/
* 2）如果地址不是一个IP地址，通过DNS（域名系统）将该地址解析成IP地址。IP地址对应着网络上一台计算机，DNS服务器本身也有IP，你的网络设置包含DNS服务器的IP。 例如：[www.guokr.com](http://www.guokr.com/) 不是一个IP，向DNS询问请求www.guokr.com 对应的IP，获得IP： 111.13.57.142。这个过程里，你的电脑直接询问的DNS服务器可能没有www.guokr.com 对应的IP，就会向它的上级服务器询问，上级服务器同样可能没有，就依此一层层向上找，最高可达根节点，找到或者全部找不到为止。
* 3）如果地址不包含端口号，根据协议的默认端口号确定一个。端口号之于计算机就像窗口号之于银行，一家银行有多个窗口，每个窗口都有个号码，不同窗口可以负责不同的服务。端口只是一个逻辑概念，和计算机硬件没有关系。 例如：[www.guokr.com](http://www.guokr.com/) 不包含端口号，http协议默认端口号是80。如果你输入的url是<http://www.guokr.com:8080/> ，那表示不使用默认的端口号，而使用指定的端口号8080。
* 4）向2和3确定的IP和端口号发起网络连接。 例如：向111.13.57.142的80端口发起连接
* 5）根据http协议要求，组织一个请求的数据包，里面包含大量请求信息，包括请求的资源路径、你的身份 例如：用自然语言来表达这个数据包，大概就是：请求 /question/554991/ ，我的身份是xxxxxxx。
* 6）服务器响应请求，将数据返回给浏览器。数据可能是根据HTML协议组织的网页，里面包含页面的布局、文字。数据也可能是图片、脚本程序等。现在你可以用浏览器的“查看源代码”功能，感受一下服务器返回的是什么东东。如果资源路径指示的资源不存在，服务器就会返回著名的404错误。
* 7）如果（6）返回的是一个页面，根据页面里一些外链的URL，例如图片的地址，按照（1）－（6）再次获取。
* 8）开始根据资源的类型，将资源组织成屏幕上显示的图像，这个过程叫渲染，网页渲染是浏览器最复杂、最核心的功能。
* 9）将渲染好的页面图像显示出来，并开始响应用户的操作。 以上只是最基本的步骤，实际不可能就这么简单，一些可选的步骤例如网页缓存、连接池、加载策略、加密解密、代理中转等等都没有提及。即使基本步骤本身也有很复杂的子步骤，TCP/IP、DNS、HTTP、HTML：每一个都可以展开成庞大的课题，而浏览器的基础——操作系统、编译器、硬件等更是一个比一个复杂。不是计算机专业的同学看了上面的解释完全不明白是很正常的，可能会问为什么要搞得那么复杂，但我保证这每一个步骤都经过深思熟虑和时间的考验。你输入URL即可浏览互联网，而计算机系统在背后做了无数你看不到的工作，计算机各个子领域无数工程师为此付出你难以想象的努力。

### 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？

\* 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。

\* 绘画 canvas

用于媒介回放的 video 和 audio 元素

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

语意化更好的内容元素，比如 article、footer、header、nav、section

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

新的技术webworker, websockt, 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，

浏览器支持新标签后，还需要添加标签默认的样式：

\* 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

请描述一下cookies,sessionStorage和localStorage的区别？  
1，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）。  
2，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。  
3，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。

有效时间：

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

### iframe有哪些缺点？

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之前需要考虑这两个缺点。如果需要使用iframe，最好是通过javascript

动态给iframe添加src属性值，这样可以可以绕开以上两个问题。

### 如何实现浏览器内多个标签页之间的通信? (阿里)

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；

### webSocket如何兼容低浏览器？(阿里)

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

### 常见的浏览器内核有哪些？

Trident内核：IE,MaxThon,TT,The World,360,搜狗浏览器等。[又称MSHTML]。bug多，对w3c标准的支持不是很好。

Gecko内核：Netscape6及以上版本，FF,MozillaSuite/SeaMonkey等。可以支持很多复杂网页效果，但是能耗高，占内存。

Presto内核：Opera7及以上。 [Opera内核原为：Presto，现为：Blink;]。公认的浏览网页速度最快的内核，处理js时比其他内核快3倍左右。但是网页兼容性不太好。

Webkit内核：Safari,Chrome等。 [ Chrome的：Blink（WebKit的分支）]。速度仅次于presto，兼容性较好。

### HTTP request报文结构是怎样的

[rfc2616](http://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html)中进行了定义：

首行是Request-Line包括：请求方法，请求URI，协议版本，CRLF

首行之后是若干行请求头，包括general-header，request-header或者entity-header，每个一行以CRLF结束

请求头和消息实体之间有一个CRLF分隔

根据实际请求需要可能包含一个消息实体 一个请求报文例子如下：

GET /Protocols/rfc2616/rfc2616-sec5.html HTTP/1.1

Host: www.w3.org

Connection: keep-alive

Cache-Control: max-age=0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/537.36

Referer: https://www.google.com.hk/

Accept-Encoding: gzip,deflate,sdch

Accept-Language: zh-CN,zh;q=0.8,en;q=0.6

Cookie: authorstyle=yes

If-None-Match: "2cc8-3e3073913b100"

If-Modified-Since: Wed, 01 Sep 2004 13:24:52 GMT

name=qiu&age=25

### HTTP response报文结构是怎样的

[rfc2616](http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html)中进行了定义：

首行是状态行包括：HTTP版本，状态码，状态描述，后面跟一个CRLF

首行之后是若干行响应头，包括：通用头部，响应头部，实体头部

响应头部和响应实体之间用一个CRLF空行分隔

最后是一个可能的消息实体 响应报文例子如下：

HTTP/1.1 200 OK

Date: Tue, 08 Jul 2014 05:28:43 GMT

Server: Apache/2

Last-Modified: Wed, 01 Sep 2004 13:24:52 GMT

ETag: "40d7-3e3073913b100"

Accept-Ranges: bytes

Content-Length: 16599

Cache-Control: max-age=21600

Expires: Tue, 08 Jul 2014 11:28:43 GMT

P3P: policyref="http://www.w3.org/2001/05/P3P/p3p.xml"

Content-Type: text/html; charset=iso-8859-1

{"name": "qiu", "age": 25}

### HTTP状态码

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

200 OK 正常返回信息

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

301 Moved Permanently 请求的网页已永久移动到新位置。

302 Found 临时性重定向。

303 See Other 临时性重定向，且总是使用 GET 请求新的 URI。

304 Not Modified 自从上次请求后，请求的网页未修改过。

400 Bad Request 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 请求未授权。

403 Forbidden 禁止访问。

404 Not Found 找不到如何与 URI 相匹配的资源。

500 Internal Server Error 最常见的服务器端错误。

503 Service Unavailable 服务器端暂时无法处理请求（可能是过载或维护）。

Doctype作用？严格模式与混杂模式如何区分？它们有何意义？有多少种Doctype文档类型？  
1，<!DOCTYPE> 声明位于文档中的最前面，处于 标签之前。告知浏览器以何种模式来渲染文档。  
2，严格模式的排版和 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呈现。  
5，可声明三种文档类型，分别表示严格版本、过渡版本以及基于框架的类型。

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，

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。

### 前端需要注意哪些SEO

* 合理的title、description、keywords：搜索对着三项的权重逐个减小，title值强调重点即可，重要关键词出现不要超过2次，而且要靠前，不同页面title要有所不同；description把页面内容高度概括，长度合适，不可过分堆砌关键词，不同页面description有所不同；keywords列举出重要关键词即可
* 语义化的HTML代码，符合W3C规范：语义化代码让搜索引擎容易理解网页
* 重要内容HTML代码放在最前：搜索引擎抓取HTML顺序是从上到下，有的搜索引擎对抓取长度有限制，保证重要内容一定会被抓取
* 重要内容不要用js输出：爬虫不会执行js获取内容
* 少用iframe：搜索引擎不会抓取iframe中的内容
* 非装饰性图片必须加alt
* 提高网站速度：网站速度是搜索引擎排序的一个重要指标

### HTTP 请求方式

* 一台服务器要与HTTP1.1兼容，只要为资源实现GET和HEAD方法即可
* GET是最常用的方法，通常用于请求服务器发送某个资源。
* HEAD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
* PUT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
* POST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。
* 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方法请求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。
* DELETE请求服务器删除请求URL指定的资源

### 请描述一下 GET 和 POST 的区别?

1、 get是从服务器获取数据 -----"取"; post是向服务器提交数据 -----“发”

2、 form表单默认的method为"GET"

3、 get将数据按照variable = value 的形式，加上URL的后面，中间用"?"连接，各个变量之间用"&"连接; post将数据不像get方式那样

4、 参数上面3的数据传输方式，可以得出：post安全性比get方式要高

5、

URL不存在参数上限的问题，HTTP协议没有对URL长度进行限制，限制的是部分浏览器和服务器的限制。

IE对URL长度的限制为2083KB

get方式是通过URL传输的数据的，数据量一般在2KB左右，但是执行效率比post高

理论上post方式没有大小限制，HTTP协议规范也没进行大小限制。post数据没有限制，限制的是服务器处理程序的能力

### 浏览器本地存储与服务器端存储之间的区别

* 其实数据既可以在浏览器本地存储，也可以在服务器端存储。
* 浏览器端可以保存一些数据，需要的时候直接从本地获取，sessionStorage、localStorage和cookie都由浏览器存储在本地的数据。
* 服务器端也可以保存所有用户的所有数据，但需要的时候浏览器要向服务器请求数据。
* 1.服务器端可以保存用户的持久数据，如数据库和云存储将用户的大量数据保存在服务器端。
* 2.服务器端也可以保存用户的临时会话数据。服务器端的session机制，如jsp的 session 对象，数据保存在服务器上。实现上，服务器和浏览器之间仅需传递session id即可，服务器根据session id找到对应用户的session对象。会话数据仅在一段时间内有效，这个时间就是server端设置的session有效期。
* 服务器端保存所有的用户的数据，所以服务器端的开销较大，而浏览器端保存则把不同用户需要的数据分布保存在用户各自的浏览器中。
* 浏览器端一般只用来存储小数据，而服务器可以存储大数据或小数据。
* 服务器存储数据安全一些，浏览器只适合存储一般数据。

### socket和http有什么区别？

socket是网络传输层的一种技术，跟http有本质的区别，http是应用层的一个网络协议。使用socket技术理论上来讲，按照http的规范，完全可以使用socket来达到发送http请求的目的，只要发送的数据包按照http协议来即可

Socket和http的区别：

Socket是长连接，http是短连接

Socket是双向通信，http是单向的，只能客户端向服务器发送数据

Socket的数据完全由自己组织，http必须按照http协议来发送

Socket的使用场景:

1. 客户端频繁请求服务器，如股票应用，需要一直向服务器请求最新的数据，如果使用http，那么第一，频繁请求，就会频繁连接，造成服务器压力巨大，如果使用socket，一次连接，不会消耗服务器太多资源。第二，频繁发送，返回数据，如果使用http，因为http协议的限制，发送的数据包中包含了很多的请求头，请求行等http协议必须带的数据，发送的数据量比socket大很多，socket只需要请求和返回需要的数据即可，如股票应用中，只需要返回股票的最新价格即可，及时性更高
2. 客户端和服务器互相发送数据，如聊天应用，需要客户端上传聊天内容，同时，别人给你发消息，服务器也能主动把别人发送的消息发送给你

需要使用socket技术的场景：网络游戏、即时通信、股票软件、自己实现推送机制.

# CSS部分

### CSS3有哪些新特性（包含哪些模块）？

1、圆角border-radius、阴影box-shadow,text-shadow、渐变gradients、过渡transitions、

动画animations、布局multi-columns, flex box, grid layout, Opacity,color(rgb,rgba,hsl,hsla )

2、子串匹配的属性选择器:E[attribute^="value"], E[attribute$="value"], E[attribute\*="value"]

3、新的伪类：

:target, :enabled 和 :disabled,:checked,:indeterminate,:root,:nth-child 和 :nth-last-child,

:nth-of-type和 :nth-last-of-type,:last-child,:first-of-type 和 :last-of-type,

:only-child 和 :only-of-type,:empty,和 :not

4、伪元素使用两个冒号而不是一个来表示：

:after 变为 ::after, :before 变为 ::before, :first-letter 变为 ::first-letter,

还有 :first-line 变为 ::first-line。

### 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。

\*最简单的初始化方法就是： \* {padding: 0; margin: 0;} （不建议）

淘宝的样式初始化：

body, h1, h2, h3, h4, h5, h6, hr, p, blockquote, dl, dt, dd, ul, ol, li, pre, form, fieldset, legend, button, input, textarea, th, td { margin:0; padding:0; }

body, button, input, select, textarea { font:12px/1.5tahoma, arial, \5b8b\4f53; }

h1, h2, h3, h4, h5, h6{ font-size:100%; }

address, cite, dfn, em, var { font-style:normal; }

code, kbd, pre, samp { font-family:couriernew, courier, monospace; }

small{ font-size:12px; }

ul, ol { list-style:none; }

a { text-decoration:none; }

a:hover { text-decoration:underline; }

sup { vertical-align:text-top; }

sub{ vertical-align:text-bottom; }

legend { color:#000; }

fieldset, img { border:0; }

button, input, select, textarea { font-size:100%; }

table { border-collapse:collapse; border-spacing:0; }

### CSS 选择符有哪些？哪些属性可以继承？优先级算法如何计算？ CSS3新增伪类有那些？

\* 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：

1、关于文字排版的属性如：font,word-break,letter-spacing,text-align,text-rendering,

word-spacing,white-space,text-indent,text-transform,text-shadow

2、line-height

3、color

4、visibility

5、cursor

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

\* 优先级就近原则,同权重情况下样式定义最近者为准;

!important > id > class > tag

important 比 内联优先级高

\* CSS3新增伪类举例：

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。

:enabled :disabled 控制表单控件的禁用状态。

:checked 单选框或复选框被选中。

### 解释下浮动和它的工作原理

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

### 列举不同的清除浮动的技巧，并指出它们各自适用的使用场景

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；二、content属性是必须的，但其值可以为空，蓝色理想讨论该方法的时候content属性的值 设为”.”，但我发现为空亦是可以的。

**介绍一下标准的CSS的盒子模型？与IE的盒子模型有什么不同的？**

1、有两种, IE 盒子模型、标准 W3C 盒子模型；IE的content部分包含了 border 和 pading;  
2、盒模型： 内容(content)、填充(padding)、边界(margin)、 边框(border). 文档中的每个元素被描绘为矩形盒子。确定其大小,属性——比如颜色、背景、边框,及其位置是渲染引擎的目标。CSS下这些矩形盒子由标准盒模型描述。这个模型描述元素内容占用空间。

盒子有四个边界：外边距边界margin edge, 边框边界border edge, 内边距边界padding edge 与 内容边界content edge。

a.内容区域content area 是真正包含元素内容的区域。位于内容边界的内部,

它的大小为内容宽度 或 content-box宽及内容高度或content-box高。

如果 box-sizing 为默认值,

width, min-width, max-width, height, min-height 与 max-height 控制内容大小。

b.内边距区域padding area 用内容及可能的边框之间的空白区域扩展内容区域。

它位于内边边界内部,通常有背景——颜色或图片（不透明图片盖住背景颜色）.

它的大小为 padding-box 宽与 padding-box 高。

内边距与内容边界之间的空间由padding-top,padding-right,padding-bottom,padding-left和简写padding控制。

c.边框区域border area 是包含边框的区域,扩展了内边距区域.

它位于边框边界内部,大小为 border-box 宽和 border-box 高。

由 border-width 及简写属性 border控制。

d.外边距区域margin area用空白区域扩展边框区域,以分开相邻的元素。

它的大小为margin-box,margin-top,margin-right,margin-bottom,margin-left及简写属性margin控制。

在外边距合并的情况下,由于盒之间共享外边距,外边距不容易弄清楚。

最后注意,对于行内非替换元素,其占用空间（行高）由 line-height 决定,即使有内边距与边框。

**display有哪些值？说明他们的作用。position的值relative和absolute定位原点是？absolute与fixed共同点与不同点？**

1、display

block 象块类型元素一样显示。

none 缺省值。象行内元素类型一样显示。

inline-block 象行内元素一样显示,但其内容象块类型元素一样显示。

list-item 象块类型元素一样显示,并添加样式列表标记。

2、position

absolute 生成绝对定位的元素,相对于 static 定位以外的第一个父元素进行定位。

fixed （老IE不支持） 生成绝对定位的元素,相对于浏览器窗口进行定位。

relative 生成相对定位的元素,相对于其正常位置进行定位。

static 默认值。没有定位,元素出现在正常的流中

inherit 规定从父元素继承 position 属性的值。

\*（忽略 top, bottom, left, right z-index 声明）

absolute与fixed共同点与不同点

A、共同点：

1.改变行内元素的呈现方式,display被置为block;

2.让元素脱离普通流,不占据空间;

3.默认会覆盖到非定位元素上;

B、不同点：

absolute的”根元素“是可以设置的,而fixed的”根元素“固定为浏览器窗口。

当你滚动网页,fixed元素与浏览器窗口之间的距离是不变的。

**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。

**解释下 CSS sprites,以及你要如何在页面或网站中使用它？**

CSS Sprites其实就是把网页中一些背景图片整合到一张图片文件中;

再利用CSS的“background-image”,“background- repeat,background-position”的组合进行背景定位;

background-position可以用数字能精确的定位出背景图片的位置;

这样可以减少很多图片请求的开销,因为请求耗时比较长；请求虽然可以并发,但是也有限制,一般浏览器都是6个;

对于未来而言,就不需要这样做了,因为有了`http2`;

\*优点：

减少HTTP请求数,极大地提高页面加载速度;

增加图片信息重复度,提高压缩比,减少图片大小;

更换风格方便,只需在一张或几张图片上修改颜色或样式即可实现;

\*缺点：

图片合并麻烦;

维护麻烦,修改一个图片可能需要从新布局整个图片,样式;

**什么叫外边距折叠(collapsing margins)？**

毗邻的两个或多个margin会合并成一个margin,叫做外边距折叠。规则如下：

1、两个或多个毗邻的普通流中的块元素垂直方向上的margin会折叠

2、浮动元素/inline-block元素/绝对定位元素的margin不会和垂直方向上的其他元素的margin折叠

3、创建了块级格式化上下文的元素,不会和它的子元素发生margin折叠

4、元素自身的margin-bottom和margin-top相邻时也会折叠

### 如果需要手动写动画，你认为最小时间间隔是多久，为什么？（阿里）

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

### display:inline-block 什么时候会显示间隙？(携程)

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

### 行内元素有哪些？块级元素有哪些？ 空(void)元素有那些？

（1）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，

比如div默认display属性值为“block”，成为“块级”元素；

span默认display属性值为“inline”，是“行内”元素。

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

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

（3）知名的空元素：

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

鲜为人知的是：

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

# JS部分

### javascript有哪几种数据类型

六种基本数据类型

* undefined
* null
* string
* boolean
* Number
* Object

### 说说js中的闭包和变量作用域？

一、变量的作用域

要理解闭包,首先必须理解Javascript特殊的变量作用域。

变量的作用域无非就是两种：全局变量和局部变量。

Javascript语言的特殊之处,就在于函数内部可以直接读取全局变量。

　　var n=999;

　　function f1(){

　　　　alert(n);

　　}

　　f1(); // 999

另一方面,在函数外部自然无法读取函数内的局部变量。

　　function f1(){

　　　　var n=999;

　　}

　　alert(n); // error

需要注意:函数内部声明变量的时候,一定要使用var命令。

如果不用的话,你实际上声明了一个全局变量！

　　function f1(){

　　　　n=999;

　　}

　　f1();

　　alert(n); // 999

---------------------------------------分割线---------------------------------------

二、如何从外部读取局部变量？

出于种种原因,我们有时候需要得到函数内的局部变量。

但是正常情况下,这是办不到的；只有通过变通方法才能实现。

那就是在函数的内部,再定义一个函数。

　　function f1(){

　　　　var n=999;

　　　　function f2(){

　　　　　　alert(n); // 999

　　　　}

　　}

在上面的代码中,函数f2就被包括在函数f1内部;

这时f1内部的所有局部变量,对f2都是可见的;

但是反过来就不行,f2内部的局部变量,对f1就是不可见的;

这就是Javascript语言特有的"链式作用域"结构（chain scope）;

子对象会一级一级地向上寻找所有父对象的变量。

所以,父对象的所有变量,对子对象都是可见的,反之则不成立。

既然f2可以读取f1中的局部变量,那么只要把f2作为返回值

,我们不就可以在f1外部读取它的内部变量了吗！

　　function f1(){

　　　　var n=999;

　　　　function f2(){

　　　　　　alert(n);

　　　　}

　　　　return f2;

　　}

　　var result=f1();

　　result(); // 999

---------------------------------------分割线---------------------------------------

三、闭包的概念

上一节代码中的f2函数,就是闭包。

各种专业文献上的"闭包"（closure）定义非常抽象,很难看懂。

我的理解是,闭包就是能够读取其他函数内部变量的函数。

由于在Javascript语言中,只有函数内部的子函数才能读取局部变量,

因此可以把闭包简单理解成"定义在一个函数内部的函数"。

所以,在本质上,闭包就是将函数内部和函数外部连接起来的一座桥梁。

---------------------------------------分割线---------------------------------------

四、闭包的用途

闭包可以用在许多地方。它的最大用处有两个:

一个是前面提到的可以读取函数内部的变量;

另一个就是让这些变量的值始终保持在内存中。

怎么来理解这句话呢？请看下面的代码。

　　function f1(){

　　　　var n=999;

　　　　nAdd=function(){n+=1}

　　　　function f2(){

　　　　　　alert(n);

　　　　}

　　　　return f2;

　　}

　　var result=f1();

　　result(); // 999

　　nAdd();

　　result(); // 1000

在这段代码中,result实际上就是闭包f2函数。

它一共运行了两次,第一次的值是999,第二次的值是1000。

这证明了,函数f1中的局部变量n一直保存在内存中,并没有在f1调用后被自动清除。

为什么会这样呢？原因就在于f1是f2的父函数,而f2被赋给了一个全局变量,

这导致f2始终在内存中,而f2的存在依赖于f1,因此f1也始终在内存中,

不会在调用结束后,被垃圾回收机制（garbage collection）回收。

这段代码中另一个值得注意的地方,就是"nAdd=function(){n+=1}"这一行,

1、首先在nAdd前面没有使用var关键字,因此nAdd是一个全局变量,而不是局部变量;

2、其次,nAdd的值是一个匿名函数（anonymous function）,而这个匿名函数本身也是一个闭包,

所以nAdd相当于是一个setter,可以在函数外部对函数内部的局部变量进行操作。

---------------------------------------分割线---------------------------------------

五、使用闭包的注意点

1、由于闭包会使得函数中的变量都被保存在内存中,内存消耗很大,所以不能滥用闭包,

否则会造成网页的性能问题,在IE中可能导致内存泄露。解决方法是,在退出函数之前,

将不使用的局部变量全部删除。

2、闭包会在父函数外部,改变父函数内部变量的值。

所以,如果你把父函数当作对象（object）使用,把闭包当作它的公用方法（Public Method）,

把内部变量当作它的私有属性（private value）,这时一定要小心,不要随便改变父函数内部变量的值。

---------------------------------------分割线---------------------------------------

六、思考题

如果你能理解下面两段代码的运行结果,应该就算理解闭包的运行机制了。（首先考虑作用域this的指向）

代码片段一。

　　var name = "The Window";

　　var object = {

　　　　name : "My Object",

　　　　getNameFunc : function(){

　　　　　　return function(){

　　　　　　　　return this.name;

　　　　　　};

　　　　}

　　};

　　alert(object.getNameFunc()());

代码片段二。

　　var name = "The Window";

　　var object = {

　　　　name : "My Object",

　　　　getNameFunc : function(){

　　　　　　var that = this;

　　　　　　return function(){

　　　　　　　　return that.name;

　　　　　　};

　　　　}

　　};

alert(object.getNameFunc()());

**说说js中的事件机制？**

一、事件机制(事件流)

浏览器中的事件流意味着页面上可有不仅一个,甚至多个元素响应同一个事件。

而这一个或多个元素响应事件发生的先后顺序在各个浏览器（主要针对IE和Netscape）上是不同的。

冒泡型事件（Dubbed Bubbling）

IE上的解决方案就是冒泡型事件（Dubbed Bubbling）,冒泡型事件的基本思想是,

事件按照从最特定的事件目标到最不特定的事件目标（document对象）的顺序触发。

捕获型事件（Event Capturing）

相对IE4.0,Netscape4.0则使用的是捕获型事件的解决方案。

这个事件触发的过程则正好和冒泡相反——在捕获型事件中,事件从最不精确的对象（document对象）开始触发,

然后到最精确的对象。

DOM 事件流

这个事件流则是W3C制定一个标准规范,它同时支持两种事件流模式,

不过是先发生捕获型事件流,再发生冒泡型事件流。

DOM事件流最独特的是,它支持文本节点也触发事件（IE中这不支持）,

不过说实话,我现在还看不出来让文本节点支持事件有什么作用。

最后要说的是,根据最近大家在开发的实践过程中的运用,

我们一般都采取冒泡型的事件流触发方式,这点我们的IE做的比较成功。

至于原因,我想你可以通过上面的解释可以看出,毕竟我们给元素触发事件,

肯定是希望从我们最希望先触发（从最精确的）的那个开始。

---------------------------------------分割线---------------------------------------

二、事件绑定

事件处理函数/监听函数

1、在DOM元素中直接绑定

2、在JavaScript代码中绑定

3、绑定事件监听函数

具体实践：

IE中attachEvent(”NAME\_OF\_EVENT\_HANDLER”, fnHandler)给元素绑定事件；

而在支持DOM事件流的浏览器里,则使用addEventListener(”NAME\_OF\_EVENT\_HANDLER”, fnHandler, isCapture);

前面我控制FIREFOX中触发捕获事件流,就是通过设置isCapture（ture：捕获；false：冒泡）做到的;

function addEvent(obj,type,handle){

try{ // Chrome、FireFox、Opera、Safari、IE9.0及其以上版本

obj.addEventListener(type,handle,false);

}catch(e){

try{ // IE8.0及其以下版本

obj.attachEvent('on' + type,handle);

}catch(e){ // 现代浏览器

obj['on' + type] = handle;

}

}

}

哪些事件是支持冒泡,那些不支持？

基本上只有onload,unload,focus,blur,submit和change事件是不支持冒泡的；

自然像keydownkeypress,keyup,click,dbclick,mousedown,mouseout,mouseover,mouseup,mousemove支持冒泡,

那就是“Mouse and Key Events”支持冒泡;

---------------------------------------分割线---------------------------------------

三、事件委托

document.onclick,这个示例把事件委托放到了document上,即点击document就直接触发我们相应的事件。

document.onclick = function(event){

var event = event || window.event; //IE doesn't pass in the event object

var target = event.target || event.srcElement; //IE uses srcElement as the target

switch(target.id){

case "help-btn":

openHelp();

break;

case "save-btn":

saveDocument();

break;

case "undo-btn":

undoChanges();

break;

//如果有其元素需要处理点击事件, 只需要在这里添加不同的case分支就行。

}

};

优点：

从“处理速度”、“新增元素事件处理”和“内存消耗”三方面比较了“事件委托”和“事件绑定”的对比,

可以很容易看出,“事件委托”在“处理速度”和“内存消耗”上,有得天独厚的优势。

所以,在Web编程的时候,尤其在构建大型系统的时候,应该尽量考虑使用“事件委托”。

但是,“事件委托”并不是万能的；它也有一些弊端。下面我们在论述一下它的弊端。

缺点：

使用“事件委托”时,并不是说把事件委托给的元素越靠近顶层就越好。

事件冒泡的过程也需要耗时,越靠近顶层,事件的”事件传播链”越长,也就越耗时。

---------------------------------------分割线---------------------------------------

四、阻止事件冒泡和阻止事件默认行为

a.阻止事件冒泡

function stopBubble(e) {

//如果提供了事件对象,则这是一个非IE浏览器

if ( e && e.stopPropagation ){

e.stopPropagation(); //因此它支持W3C的stopPropagation()方法

}else{

window.event.cancelBubble = true; //否则,我们需要使用IE的方式来取消事件冒泡

}

}

b.当按键后,不希望按键继续传递给如HTML文本框对象时,可以取消返回值.即停止事件默认行为;

//阻止浏览器的默认行为

function stopDefault(e) {

//如果提供了事件对象,则这是一个非IE浏览器

if (e && e.preventDefault ){

e.preventDefault(); //阻止默认浏览器动作(W3C)

}else{

window.event.returnValue = false; //IE中阻止函数器默认动作的方式

}

}

**说说http请求和ajax 以及ajax跨域？**

1、HTTP请求

超文本传输协议（HTTP）的设计目的是保证客户机与服务器之间的通信。

HTTP 的工作方式是客户机与服务器之间的请求-应答协议。

两种 HTTP 请求方法：GET 和 POST:

a.GET 方法

请注意,查询字符串（名称/值对）是在 GET 请求的 URL 中发送的：

/test/demo\_form.asp？name1=value1&name2=value2

有关 GET 请求的其他一些注释：

GET 请求可被缓存

GET 请求保留在浏览器历史记录中

GET 请求可被收藏为书签

GET 请求不应在处理敏感数据时使用

GET 请求有长度限制

GET 请求只应当用于取回数据

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

b.POST 方法

请注意,查询字符串（名称/值对）是在 POST 请求的 HTTP 消息主体中发送的：

POST /test/demo\_form.asp HTTP/1.1

Host: w3schools.com

name1=value1&name2=value2

有关 POST 请求的其他一些注释：

POST 请求不会被缓存

POST 请求不会保留在浏览器历史记录中

POST 不能被收藏为书签

POST 请求对数据长度没有要求

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

c、http状态码有那些？分别代表是什么意思？

100-199 用于指定客户端应相应的某些动作。

200-299 用于表示请求成功。

300-399 用于已经移动的文件并且常被包含在定位头信息中指定新的地址信息。

400-499 用于指出客户端的错误。

400 语义有误,当前请求无法被服务器理解。

401 当前请求需要用户验证

403 服务器已经理解请求,但是拒绝执行它。

500-599 用于支持服务器错误。

503 服务不可用

---------------------------------------分割线---------------------------------------

2、AJAX

AJAX (异步 JavaScript 和 XML) 是个新产生的术语,专为描述JavaScript的两项强大性能.

这两项性能在多年来一直被网络开发者所忽略,

直到最近Gmail, Google Suggest和Google Maps的横空出世才使人们开始意识到其重要性.

这两项被忽视的性能是:

无需重新装载整个页面便能向服务器发送请求.

对XML文档的解析和处理

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

1、步骤1 浏览器发送一个HTTP请求

为了用JavaScript向服务器发送一个HTTP请求, 需要一个具备这种功能的类实例.

这样的类首先由Internet Explorer以ActiveX对象引入, 被称为XMLHTTP.

后来Mozilla, Safari 和其他浏览器纷纷仿效, 提供了XMLHttpRequest类,它支持微软的ActiveX对象所提供的方法和属性.

A、创建一个跨浏览器的这样的类实例(对象), 可以应用如下代码:

if (window.XMLHttpRequest) {

http\_request = new XMLHttpRequest(); // Mozilla, Safari, ...

}

else if (window.ActiveXObject)

{

http\_request = new ActiveXObject("Microsoft.XMLHTTP"); // IE

}

(上例对代码做了一定简化,这是为了解释如何创建XMLHTTP类实例. 实际的代码实例可参阅本篇步骤3.)

如果服务器的响应没有XML mime-type header,某些Mozilla浏览器可能无法正常工作.

为了解决这个问题, 如果服务器响应的header不是text/xml,可以调用其它方法修改该header.

http\_request = new XMLHttpRequest();

http\_request.overrideMimeType('text/xml');

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

B、决定当收到服务器的响应后,需要做什么.这需要告诉HTTP请求对象用哪一个JavaScript函数处理这个响应.

可以将对象的onreadystatechange属性设置为要使用的JavaScript的函数名,如下所示:

http\_request.onreadystatechange = nameOfTheFunction;

注意:在函数名后没有括号,也无需传递参数.另外还有一种方法,

你可以使用一个匿名函数来描述那些要对服务器返回的响应内容所进行的操作,如下所示:

http\_request.onreadystatechange = function(){

// do the thing

};

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

C、在定义了如何处理响应后,就要发送请求了.可以调用HTTP请求类的open()和send()方法, 如下所示:

http\_request.open('GET', 'http://www.example.org/some.file', true);

http\_request.send(null);

1、open()的第一个参数是HTTP请求方式 – GET, POST, HEAD 或任何服务器所支持的您想调用的方式；

按照HTTP规范,该参数要大写;否则,某些浏览器(如Firefox)可能无法处理请求；

有关HTTP请求方法的详细信息可参考http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

2、第二个参数是请求页面的URL.由于自身安全特性的限制,该页面不能为第三方域名的页面.

同时一定要保证在所有的页面中都使用准确的域名,否则调用open()会得到"permission denied"的错误提示.

一个常见的错误是访问站点时使用domain.tld,而当请求页面时,却使用www.domain.tld.

3、第三个参数设置请求是否为异步模式.如果是TRUE, JavaScript函数将继续执行,而不等待服务器响应.

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

这就是"AJAX"中的"A"：

1、如果第一个参数是"GET",通过open()第二个参数,传递参数,一般用于查询,jsonp的跨域就是这个原理,也只能解决get方式的请求；

2、如果第一个参数是"POST",send()方法的参数可以是任何想送给服务器的数据. 这时数据要以字符串的形式送给服务器,如下所示:

name=value&anothername=othervalue&so=on

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

2、步骤2 服务器收到请求后的响应

当发送请求时,要提供指定处理响应的JavaScript函数名.

http\_request.onreadystatechange = nameOfTheFunction;

我们来看看这个函数的功能是什么.首先函数会检查请求的状态.

如果状态值是4,就意味着一个完整的服务器响应已经收到了,您将可以处理该响应.

if (http\_request.readyState == 4) {

// everything is good, the response is received

} else {

// still not ready

}

readyState的取值如下:

0 (未初始化)

1 (正在装载)

2 (装载完毕)

3 (交互中)

4 (完成)

(Source)

接着,函数会检查HTTP服务器响应的状态值. 完整的状态取值可参见 W3C site. 我们着重看值为200 OK的响应:

if (http\_request.status == 200) {

// perfect!

} else {

// there was a problem with the request,

// for example the response may be a 404 (Not Found)

// or 500 (Internal Server Error)

response codes

}

在检查完请求的状态值和响应的HTTP状态值后, 您就可以处理从服务器得到的数据了.有两种方式可以得到这些数据:

http\_request.responseText – 以文本字符串的方式返回服务器的响应

http\_request.responseXML – 以XMLDocument对象方式返回响应.处理XMLDocument对象可以用JavaScript DOM函数

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

3、简单实例

我们现在将整个过程完整地做一次,发送一个简单的HTTP请求.

我们用JavaScript请求一个HTML文件, test.html, 文件的文本内容为"I'm a test.",

然后我们"alert()"test.html文件的内容.

<script type="text/javascript" language="javascript">

var http\_request = false;

function makeRequest(url) {

http\_request = false;

if (window.XMLHttpRequest) { // Mozilla, Safari,...

http\_request = new XMLHttpRequest();

if (http\_request.overrideMimeType) {

http\_request.overrideMimeType('text/xml');

}

} else if (window.ActiveXObject) { // IE

try {

http\_request = new ActiveXObject("Msxml2.XMLHTTP");

}

catch (e) {

try {

http\_request = new ActiveXObject("Microsoft.XMLHTTP");

}catch(e){}

}

}

if (!http\_request) {

alert('Giving up :( Cannot create an XMLHTTP instance');

return false;

}

http\_request.onreadystatechange = alertContents;

http\_request.open('GET', url, true);

http\_request.send(null);

}

function alertContents() {

if (http\_request.readyState == 4) {

if (http\_request.status == 200) {

alert(http\_request.responseText);

}else{

alert('There was a problem with the request.');

}

}

}

</script>

<style>

.testbtn{ cursor: pointer; text-decoration: underline;}

</style>

<span class='testbtn' onclick="makeRequest('test.html')">Make a request</span>

本例中:

用户点击浏览器上的"请求"链接;

接着函数makeRequest()将被调用.其参数为HTML文件test.html在同一目录下;

这样就发起了一个请求.onreadystatechange的执行结果会被传送给alertContents();

alertContents()将检查服务器的响应是否成功地收到,如果是,就会"alert()"test.html文件的内容.

---------------------------------------分割线---------------------------------------

3、Ajax的跨域

概念：只要协议、域名、端口有任何一个不同,都被当作是不同的域。

URL 说明 是否允许通信

http://www.a.com/a.js http://www.a.com/b.js 同一域名下 允许

http://www.a.com/lab/a.js http://www.a.com/script/b.js 同一域名下不同文件夹 允许

http://www.a.com:8000/a.js http://www.a.com/b.js 同一域名,不同端口 不允许

http://www.a.com/a.js https://www.a.com/b.js 同一域名,不同协议 不允许

http://www.a.com/a.js http://70.32.92.74/b.js 域名和域名对应ip 不允许

http://www.a.com/a.js http://script.a.com/b.js 主域相同,子域不同 不允许

http://www.a.com/a.js http://a.com/b.js 同一域名,不同二级域名（同上） 不允许

http://www.cnblogs.com/a.js http://www.a.com/b.js 不同域名 不允许

对于端口和协议的不同,只能通过后台来解决。

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

A、跨域资源共享（CORS）

1、CORS（Cross-Origin Resource Sharing）跨域资源共享,定义了必须在访问跨域资源时,

浏览器与服务器应该如何沟通。

2、CORS背后的基本思想就是使用自定义的HTTP头部让浏览器与服务器进行沟通,

从而决定请求或响应是应该成功还是失败。

<script type="text/javascript">

var xhr = new XMLHttpRequest();

xhr.open("￼GET", "/trigkit4",true);

xhr.send();

</script>

以上的trigkit4是相对路径,如果我们要使用CORS,相关Ajax代码可能如下所示：

<script type="text/javascript">

var xhr = new XMLHttpRequest();

xhr.open("￼GET", "http://segmentfault.com/u/trigkit4/",true);

xhr.send();

</script>

代码与之前的区别就在于相对路径换成了其他域的绝对路径,也就是你要跨域访问的接口地址。

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。

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

要解决跨域的问题,我们可以使用以下几种方法：

B、通过jsonp跨域

现在问题来了？什么是jsonp？

1、维基百科的定义是：JSONP（JSON with Padding）是资料格式 JSON 的一种“使用模式”,

可以让网页从别的网域要资料。

2、JSONP也叫填充式JSON,是应用JSON的一种新方法,只不过是被包含在函数调用中的JSON,

例如: callback({"name","trigkit4"});

JSONP由两部分组成：回调函数和数据：

回调函数是当响应到来时应该在页面中调用的函数,

而数据就是传入回调函数中的JSON数据。

在js中,我们直接用XMLHttpRequest请求不同域上的数据时,是不可以的。

但是,在页面上引入不同域上的js脚本文件却是可以的,jsonp正是利用这个特性来实现的。例如：

<script type="text/javascript">

function dosomething(jsondata){

//处理获得的json数据

}

</script>

<script src="http://example.com/data.php？callback=dosomething"></script>

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。

所以jsonp是需要服务器端的页面进行相应的配合的。

<？php

$callback = $\_GET['callback'];//得到回调函数名

$data = array('a','b','c');//要返回的数据

echo $callback.'('.json\_encode($data).')';//输出

？>

最终,输出结果为：dosomething(['a','b','c']);

如果你的页面使用jquery,那么通过它封装的方法就能很方便的来进行jsonp操作了。

<script type="text/javascript">

$.getJSON('http://example.com/data.php？callback=？,function(jsondata)'){

//处理获得的json数据

});

</script>

jquery会自动生成一个全局函数来替换callback=？中的问号,之后获取到数据后又会自动销毁,

实际上就是起一个临时代理函数的作用。

$.getJSON方法会自动判断是否跨域：

1、不跨域的话,就调用普通的ajax方法；

2、跨域的话,则会以异步加载js文件的形式来调用jsonp的回调函数。

JSONP的优点是：

1、它不像XMLHttpRequest对象实现的Ajax请求那样受到同源策略的限制；

2、它的兼容性更好,在更加古老的浏览器中都可以运行,不需要XMLHttpRequest或ActiveX的支持；

3、并且在请求完毕后可以通过调用callback的方式回传结果。

JSONP的缺点则是：

1、它只支持GET请求而不支持POST等其它类型的HTTP请求；

2、它只支持跨域HTTP请求这种情况,不能解决不同域的两个页面之间如何进行JavaScript调用的问题。

CORS和JSONP对比：

CORS与JSONP相比,无疑更为先进、方便和可靠。

1、JSONP只能实现GET请求,而CORS支持所有类型的HTTP请求。

2、使用CORS,开发者可以使用普通的XMLHttpRequest发起请求和获得数据,比起JSONP有更好的错误处理。

3、JSONP主要被老的浏览器支持,它们往往不支持CORS,而绝大多数现代浏览器都已经支持了CORS）。

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

C、通过修改document.domain来跨子域

浏览器都有一个同源策略：

1、第一个限制就是第一种方法中我们说的不能通过ajax的方法去请求不同源中的文档。

2、第二个限制是浏览器中不同域的框架之间是不能进行js的交互操作的。

不同的框架之间是可以获取window对象的,但却无法获取相应的属性和方法。

有一个页面,它的地址是http://www.example.com/a.html ,

在这个页面里面有一个iframe,它的src是http://example.com/b.html,

这个页面与它里面的iframe框架是不同域的,所以我们是无法通过在页面中书写js代码来获取iframe中的东西的：

<script type="text/javascript">

function test(){

var iframe = document.getElementById('￼ifame');

var win = document.contentWindow;

//可以获取到iframe里的window对象,但该window对象的属性和方法几乎是不可用的

var doc = win.document;//这里获取不到iframe里的document对象

var name = win.name;//这里同样获取不到window对象的name属性

}

</script>

<iframe id = "iframe" src="http://example.com/b.html" onload = "test()"></iframe>

这个时候,document.domain就可以派上用场了:

1、只要设置http://www.example.com/a.html和http://example.com/b.html

这两个页面的document.domain为相同的域名就可以了。

2、但要注意的是：document.domain的设置是有限制的,

我们只能把document.domain设置成自身或更高一级的父域,且主域必须相同。

1.在页面 http://www.example.com/a.html 中设置document.domain:

<iframe id = "iframe" src="http://example.com/b.html" onload = "test()"></iframe>

<script type="text/javascript">

document.domain = 'example.com';//设置成主域

function test(){

//contentWindow可取得子窗口的window对象

alert(document.getElementById('￼iframe').contentWindow);

}

</script>

2.在页面 http://example.com/b.html 中也设置document.domain:

<script type="text/javascript">

//在iframe载入这个页面也设置document.domain,使之与主页面的document.domain相同

document.domain = 'example.com';

</script>

\*\*修改document.domain的方法只适用于不同子域的框架间的交互\*\*

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

D、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

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

E、使用HTML5的window.postMessage方法跨域

window.postMessage(message,targetOrigin) 方法是html5新引进的特性；

可以使用它来向其它的window对象发送消息,无论这个window对象是属于同源或不同源,

目前IE8+、FireFox、Chrome、Opera等浏览器都已经支持window.postMessage方法。

---------------------------------------分割线---------------------------------------

4、优缺点

\* 优点

1. 通过异步模式，提升了用户体验

2. 优化了浏览器和服务器之间的传输，减少不必要的数据往返，减少了带宽占用

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

\* 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。

**如何给js数组去重？**

以下是数组去重的三种方法：

Array.prototype.unique1 = function () {

var n = []; //一个新的临时数组

for (var i = 0; i < this.length; i++) //遍历当前数组

{

//如果当前数组的第i已经保存进了临时数组，那么跳过，

//否则把当前项push到临时数组里面

if (n.indexOf(this[i]) == -1) n.push(this[i]);

}

return n;

}

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

Array.prototype.unique2 = function()

{

var n = {},r=[]; //n为hash表，r为临时数组

for(var i = 0; i < this.length; i++) //遍历当前数组

{

if (!n[this[i]]) //如果hash表中没有当前项

{

n[this[i]] = true; //存入hash表

r.push(this[i]); //把当前数组的当前项push到临时数组里面

}

}

return r;

}

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

Array.prototype.unique3 = function()

{

var n = [this[0]]; //结果数组

for(var i = 1; i < this.length; i++) //从第二项开始遍历

{

//如果当前数组的第i项在当前数组中第一次出现的位置不是i，

//那么表示第i项是重复的，忽略掉。否则存入结果数组

if (this.indexOf(this[i]) == i) n.push(this[i]);

}

return n;

}

### 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。

XSS原理及防范

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。   
4.尽量采用POST 而非GET 提交表单

XSS与CSRF有什么区别吗？

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。

CSRF的防御

1.服务端的CSRF方式方法很多样，但总的思想都是一致的，就是在客户端页面增加伪随机数。 2.使用验证码

**说说JS的对象,类和对象的继承？**

一、对象的定义,及构造函数 ：Javascript 面向对象编程（一）：封装

a、Javascript是一种基于对象（object-based）的语言,

遇到的所有东西几乎都是对象;

b、但是,它又不是一种真正的面向对象编程（OOP）语言,

因为它的语法中没有class（类）;

1、生成对象的原始模式

假定我们把猫看成一个对象,它有"名字"和"颜色"两个属性。

　　var Cat = {

　　　　name : '',

　　　　color : ''

　　}

现在,我们需要根据这个原型对象的规格（schema）,生成两个实例对象。

　　var cat1 = {}; // 创建一个空对象

　　　　cat1.name = "大毛"; // 按照原型对象的属性赋值

　　　　cat1.color = "黄色";

　　var cat2 = {};

　　　　cat2.name = "二毛";

　　　　cat2.color = "黑色";

好了,这就是最简单的封装了,把两个属性封装在一个对象里面。

但是有两个缺点:

一是如果多生成几个实例,写起来就非常麻烦；

二是实例与原型之间,没有任何办法,可以看出有什么联系。

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

2、原始模式的改进

我们可以写一个函数,解决代码重复的问题。

　　function Cat(name,color){

　　　　return {

　　　　　　name:name,

　　　　　　color:color

　　　　}

　　}

然后生成实例对象,就等于是在调用函数：

　　var cat1 = Cat("大毛","黄色");

　　var cat2 = Cat("二毛","黑色");

这种方法的问题依然是,cat1和cat2之间没有内在的联系,不能反映出它们是同一个原型对象的实例。

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

3、构造函数模式

为了解决从原型对象生成实例的问题,Javascript提供了一个构造函数（Constructor）模式。

所谓"构造函数",其实就是一个普通函数,但是内部使用了this变量。

对构造函数使用new运算符,就能生成实例,并且this变量会绑定在实例对象上。

比如,猫的原型对象现在可以这样写,

　　function Cat(name,color){

　　　　this.name=name;

　　　　this.color=color;

　　}

我们现在就可以生成实例对象了。

　　var cat1 = new Cat("大毛","黄色");

　　var cat2 = new Cat("二毛","黑色");

　　alert(cat1.name); // 大毛

　　alert(cat1.color); // 黄色

这时cat1和cat2会自动含有一个constructor属性,指向它们的构造函数。

　　alert(cat1.constructor == Cat); //true

　　alert(cat2.constructor == Cat); //true

Javascript还提供了一个instanceof运算符,验证原型对象与实例对象之间的关系。

　　alert(cat1 instanceof Cat); //true

　　alert(cat2 instanceof Cat); //true

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

4、构造函数模式的问题

构造函数方法很好用,但是存在一个浪费内存的问题。

请看,我们现在为Cat对象添加一个不变的属性"type"（种类）,再添加一个方法eat（吃老鼠）,原型对象Cat就变成了下面这样：

　　function Cat(name,color){

　　　　this.name = name;

　　　　this.color = color;

　　　　this.type = "猫科动物";

　　　　this.eat = function(){alert("吃老鼠");};

　　}

还是采用同样的方法,生成实例：

　　var cat1 = new Cat("大毛","黄色");

　　var cat2 = new Cat ("二毛","黑色");

　　alert(cat1.type); // 猫科动物

　　cat1.eat(); // 吃老鼠

表面上好像没什么问题,但是实际上这样做,有一个很大的弊端。

那就是对于每一个实例对象,type属性和eat()方法都是一模一样的内容,每一次生成一个实例,

都必须为重复的内容,多占用一些内存。这样既不环保,也缺乏效率。

　　 alert(cat1.eat == cat2.eat); //false

能不能让type属性和eat()方法在内存中只生成一次,然后所有实例都指向那个内存地址呢？回答是可以的。

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

5、Prototype模式

Javascript规定,每一个构造函数都有一个prototype属性,指向另一个对象。

这个对象的所有属性和方法,都会被构造函数的实例继承。

这意味着,我们可以把那些不变的属性和方法,直接定义在prototype对象上。

　　function Cat(name,color){

　　　　this.name = name;

　　　　this.color = color;

　　}

　　Cat.prototype.type = "猫科动物";

　　Cat.prototype.eat = function(){alert("吃老鼠")};

然后,生成实例。

　　var cat1 = new Cat("大毛","黄色");

　　var cat2 = new Cat("二毛","黑色");

　　alert(cat1.type); // 猫科动物

　　cat1.eat(); // 吃老鼠

这时所有实例的type属性和eat()方法,其实都是同一个内存地址,指向prototype对象,因此就提高了运行效率。

　　alert(cat1.eat == cat2.eat); //true

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

6、Prototype模式的验证方法

为了配合prototype属性,Javascript定义了一些辅助方法,帮助我们使用它;

6.1 isPrototypeOf()

这个方法用来判断,某个proptotype对象和某个实例之间的关系。

　　alert(Cat.prototype.isPrototypeOf(cat1)); //true

　　alert(Cat.prototype.isPrototypeOf(cat2)); //true

6.2 hasOwnProperty()

每个实例对象都有一个hasOwnProperty()方法,

用来判断某一个属性到底是本地属性,还是继承自prototype对象的属性。

　　alert(cat1.hasOwnProperty("name")); // true

　　alert(cat1.hasOwnProperty("type")); // false

6.3 in运算符

in运算符可以用来判断,某个实例是否含有某个属性,不管是不是本地属性。

　　alert("name" in cat1); // true

　　alert("type" in cat1); // true

in运算符还可以用来遍历某个"对象"的所有属性。

　　for(var prop in cat1) {

alert("cat1["+prop+"]="+cat1[prop]);

}

---------------------------------------分割线---------------------------------------

二、对象构造函数的继承 Javascript面向对象编程（二）：构造函数的继承

今天要介绍的是,对象之间的"继承"的五种方法:

现在有一个"动物"对象的构造函数:

　　function Animal(){

　　　　this.species = "动物";

　　}

还有一个"猫"对象的构造函数:

　　function Cat(name,color){

　　　　this.name = name;

　　　　this.color = color;

　　}

怎样才能使"猫"继承"动物"呢？

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

1、构造函数绑定

第一种方法也是最简单的方法,使用call（子对象,单个参数）或apply（子对象,数组参数）方法,

将父对象的构造函数绑定在子对象上,即在子对象构造函数中加一行：

　　function Cat(name,color){

　　　　Animal.apply(this, arguments);

　　　　this.name = name;

　　　　this.color = color;

　　}

　　var cat1 = new Cat("大毛","黄色");

　　alert(cat1.species); // 动物

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

2、prototype模式

第二种方法更常见,使用prototype属性,

如果"猫"的prototype对象,指向一个Animal的实例,那么所有"猫"的实例,就能继承Animal了。

　　Cat.prototype = new Animal();

　　Cat.prototype.constructor = Cat;

　　var cat1 = new Cat("大毛","黄色");

　　alert(cat1.species); // 动物

代码的第一行,我们将Cat的prototype对象指向一个Animal的实例。

　　Cat.prototype = new Animal();

它相当于完全删除了prototype 对象原先的值,然后赋予一个新值。但是,第二行又是什么意思呢？

　　Cat.prototype.constructor = Cat;

原来,任何一个prototype对象都有一个constructor属性,指向它的构造函数;

如果没有"Cat.prototype = new Animal();"这一行,Cat.prototype.constructor是指向Cat的;

加了这一行以后,Cat.prototype.constructor指向Animal;

　　alert(Cat.prototype.constructor == Animal); //true

更重要的是,每一个实例也有一个constructor属性,默认调用prototype对象的constructor属性。

　　alert(cat1.constructor == Cat.prototype.constructor); // true

因此,在运行"Cat.prototype = new Animal();"这一行之后,cat1.constructor也指向Animal！

　　alert(cat1.constructor == Animal); // true

这显然会导致继承链的紊乱（cat1明明是用构造函数Cat生成的）,因此我们必须手动纠正,

将Cat.prototype对象的constructor值改为Cat。这就是第二行的意思;

很重要的一点,编程时务必要遵守。下文都遵循这一点,即如果替换了prototype对象,

　　o.prototype = {};

那么,下一步必然是为新的prototype对象加上constructor属性,并将这个属性指回原来的构造函数。

　　o.prototype.constructor = o;

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

3、直接继承prototype

第三种方法是对第二种方法的改进。

由于Animal对象中,不变的属性都可以直接写入Animal.prototype。

所以,我们也可以让Cat()跳过 Animal(),直接继承Animal.prototype。

现在,我们先将Animal对象改写：

　　function Animal(){}

　　Animal.prototype.species = "动物";

然后,将Cat的prototype对象,然后指向Animal的prototype对象,这样就完成了继承。

　　Cat.prototype = Animal.prototype;

　　Cat.prototype.constructor = Cat;

　　var cat1 = new Cat("大毛","黄色");

　　alert(cat1.species); // 动物

与前一种方法相比,这样做的优点是效率比较高（不用执行和建立Animal的实例了）,比较省内存。

缺点是 Cat.prototype和Animal.prototype现在指向了同一个对象,

那么任何对Cat.prototype的修改,都会反映到Animal.prototype。

所以,上面这一段代码其实是有问题的。请看第二行

　　Cat.prototype.constructor = Cat;

这一句实际上把Animal.prototype对象的constructor属性也改掉了,所有请看第四种方法！！！！

　　alert(Animal.prototype.constructor); // Cat

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

4、利用空对象作为中介

由于"直接继承prototype"存在上述的缺点,所以就有第四种方法,利用一个空对象作为中介。

　　var F = function(){};

　　F.prototype = Animal.prototype;

　　Cat.prototype = new F();

　　Cat.prototype.constructor = Cat;

F是空对象,所以几乎不占内存。这时,修改Cat的prototype对象,就不会影响到Animal的prototype对象。

　　alert(Animal.prototype.constructor); // Animal

我们将上面的方法,封装成一个函数,便于使用。

　　function extend(Child, Parent) {

　　　　var F = function(){};

　　　　F.prototype = Parent.prototype;

　　　　Child.prototype = new F();

　　　　Child.prototype.constructor = Child;

　　　　Child.uber = Parent.prototype;

　　}

使用的时候,方法如下

　　extend(Cat,Animal);

　　var cat1 = new Cat("大毛","黄色");

　　alert(cat1.species); // 动物

这个extend函数,就是YUI库如何实现继承的方法。

另外,说明一点,函数体最后一行：

　　Child.uber = Parent.prototype;

意思是为子对象设一个uber属性,这个属性直接指向父对象的prototype属性,uber是一个德语词,意思是"向上"、"上一层"。

这等于在子对象上打开一条通道,可以直接调用父对象的方法。这一行放在这里,只是为了实现继承的完备性,纯属备用性质。

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

5、拷贝继承

上面是采用prototype对象,实现继承。我们也可以换一种思路,纯粹采用"拷贝"方法实现继承。

简单说,如果把父对象的所有属性和方法,拷贝进子对象,不也能够实现继承吗？这样我们就有了第五种方法。

首先,还是把Animal的所有不变属性,都放到它的prototype对象上。

　　function Animal(){}

　　Animal.prototype.species = "动物";

然后,再写一个函数,实现属性拷贝的目的。

　　function extend2(Child, Parent) {

　　　　var p = Parent.prototype;

　　　　var c = Child.prototype;

　　　　for (var i in p) {

　　　　　　c[i] = p[i];

}

　　　　return c;

　　}

这个函数的作用,就是将父对象的prototype对象中的属性,一一拷贝给Child对象的prototype对象

（但这里for in 属于浅拷贝,因为for in 会遍历原型链上的属性） 。

使用的时候,这样写：

　　extend2(Cat, Animal);

　　var cat1 = new Cat("大毛","黄色");

alert(cat1.species); // 动物

---------------------------------------分割线---------------------------------------

三、非构造函数的继承 Javascript面向对象编程（三）：非构造函数的继承

第一部分介绍了"封装"；

第二部分介绍了使用构造函数实现"继承"。

第三部分介绍不使用构造函数实现"继承"。

1、什么是"非构造函数"的继承？

比如,现在有一个对象,叫做"中国人"。

　　var Chinese = {

　　　　nation:'中国'

　　};

还有一个对象,叫做"医生"。

　　var Doctor ={

　　　　career:'医生'

　　}

请问怎样才能让"医生"去继承"中国人",也就是说,我怎样才能生成一个"中国医生"的对象？

这里要注意,这两个对象都是普通对象,不是构造函数,无法使用构造函数方法实现"继承"。

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

2、object()方法 返回new原型对象的方法 （创建jquery对象就是用这个原理,但多了一步判断）

json格式的发明人Douglas Crockford,提出了一个object()函数,可以做到这一点。

　　function object(o) {

　　　　function F() {}

　　　　F.prototype = o;

　　　　return new F();

　　}

这个object()函数,其实只做一件事,就是把子对象的prototype属性,指向父对象,从而使得子对象与父对象连在一起。

使用的时候,第一步先在父对象的基础上,生成子对象：

　　var Doctor = object(Chinese);

然后,再加上子对象本身的属性：

　　Doctor.career = '医生';

这时,子对象已经继承了父对象的属性了。

　　alert(Doctor.nation); //中国

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

3、object.create()方法

a.在父对象：

var Chinese={"nation":"中国"};

b.建立子对象：

var Doctor= Object.create(Chinese);

c.再加上子对象本身的属性：

Doctor.career = '医生';

这时,子对象已经继承了父对象的属性了。

console.log(Doctor.nation + Doctor.career);//中国医生

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

4、浅拷贝

除了使用"prototype链"以外,还有另一种思路：把父对象的属性,全部拷贝给子对象,也能实现继承。

下面这个函数,就是在做拷贝：

　　function extendCopy(p) {

　　　　var c = {};

　　　　for (var i in p) {

　　　　　　c[i] = p[i];

　　　　}

　　　　return c;

　　}

使用的时候,这样写：

　　var Doctor = extendCopy(Chinese);

　　Doctor.career = '医生';

　　alert(Doctor.nation); // 中国

但是,这样的拷贝有一个问题。那就是,如果父对象的属性等于数组或另一个对象,

那么实际上,子对象获得的只是一个内存地址,而不是真正拷贝,因此存在父对象被篡改的可能。

请看,现在给Chinese添加一个"出生地"属性,它的值是一个数组。

　　Chinese.birthPlaces = ['北京','上海','香港'];

通过extendCopy()函数,Doctor继承了Chinese。

　　var Doctor = extendCopy(Chinese);

然后,我们为Doctor的"出生地"添加一个城市：

　　Doctor.birthPlaces.push('厦门');

发生了什么事？Chinese的"出生地"也被改掉了！

　　alert(Doctor.birthPlaces); //北京, 上海, 香港, 厦门

　　alert(Chinese.birthPlaces); //北京, 上海, 香港, 厦门

所以,extendCopy()只是拷贝基本类型的数据,我们把这种拷贝叫做"浅拷贝"。这是早期jQuery实现继承的方式。

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

5、深拷贝

所谓"深拷贝",就是能够实现真正意义上的数组和对象的拷贝。它的实现并不难,只要递归调用"浅拷贝"就行了。

　　function deepCopy(p, c) {

　　　　var c = c || {};

for (var i in p) {

if(p.hasOwnProperty(i)){ //不去继承p原型链上的属性

if (typeof p[i] === 'object') {

　　　　　　　　c[i] = (p[i].constructor === Array) ？ [] : {};

　　　　　　　　deepCopy(p[i], c[i]);

　　　　　　} else {

　　　　　　　　　c[i] = p[i];

　　　　　　}

}

　　　　}

　　　　return c;

　　}

使用的时候这样写：

　　var Doctor = deepCopy(Chinese);

现在,给父对象加一个属性,值为数组。然后,在子对象上修改这个属性：

　　Chinese.birthPlaces = ['北京','上海','香港'];

　　Doctor.birthPlaces.push('厦门');

这时,父对象就不会受到影响了。

　　alert(Doctor.birthPlaces); //北京, 上海, 香港, 厦门

　　alert(Chinese.birthPlaces); //北京, 上海, 香港

目前,jQuery库使用的就是这种继承方法。

### 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。

### 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。

### 谈谈This对象的理解

* this总是指向函数的直接调用者（而非间接调用者）；
* 如果有new关键字，this指向new出来的那个对象；
* 在事件中，this指向触发这个事件的对象，特殊的是，IE中的attachEvent中的this总是指向全局对象Window；

### 函数内部arguments变量有哪些特性,有哪些属性,如何将它转换为数组

* arguments所有函数中都包含的一个局部变量，是一个类数组对象，对应函数调用时的实参。如果函数定义同名参数会在调用时覆盖默认对象
* arguments[index]分别对应函数调用时的实参，并且通过arguments修改实参时会同时修改实参
* arguments.length为实参的个数（Function.length表示形参长度）
* arguments.callee为当前正在执行的函数本身，使用这个属性进行递归调用时需注意this的变化
* arguments.caller为调用当前函数的函数（已被遗弃）
* 转换为数组：var args = Array.prototype.slice.call(arguments, 0);

### offsetWidth/offsetHeight,clientWidth/clientHeight与scrollWidth/scrollHeight的区别

* offsetWidth/offsetHeight返回值包含**content + padding + border**，效果与e.getBoundingClientRect()相同
* clientWidth/clientHeight返回值只包含**content + padding**，如果有滚动条，也**不包含滚动条**
* scrollWidth/scrollHeight返回值包含**content + padding + 溢出内容的尺寸**

**JSON的了解？XML和JSON的区别？**

了解： JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。 它是基于JavaScript的一个子集。数据格式简单, 易于读写, 占用带宽小； {'age':'12', 'name':'back'} JSON.parse('str') //转换string为JSON格式 JSON.stringfy('JSON') //转换JSON为string格式

\*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。

**哪些地方会出现css阻塞,哪些地方会出现js阻塞？**

js的阻塞特性：

所有浏览器在下载JS的时候,会阻止一切其他活动,比如其他资源的下载,内容的呈现等等。

直到JS下载、解析、执行完毕后才开始继续并行下载其他资源并呈现内容。

为了提高用户体验,新一代浏览器都支持并行下载JS,但是JS下载仍然会阻塞其它资源的下载（例如.图片,css文件等）。

\*由于浏览器为了防止出现JS修改DOM树,需要重新构建DOM树的情况,所以就会阻塞其他的下载和呈现。

\*嵌入JS会阻塞所有内容的呈现,而外部JS只会阻塞其后内容的显示,2种方式都会阻塞其后资源的下载。

也就是说外部样式不会阻塞外部脚本的加载,但会阻塞外部脚本的执行。

CSS怎么会阻塞加载了？

CSS本来是可以并行下载的,在什么情况下会出现阻塞加载了(在测试观察中,IE6下CSS都是阻塞加载）

\*当CSS后面跟着嵌入的JS的时候,该CSS就会出现阻塞后面资源下载的情况;

\*而当把嵌入JS放到CSS前面,就不会出现阻塞的情况了。

根本原因：

\*因为浏览器会维持html中css和js的顺序,样式表必须在嵌入的JS执行前先加载、解析完。

\*而嵌入的JS会阻塞后面的资源加载,所以就会出现上面CSS阻塞下载的情况。

嵌入JS应该放在什么位置？

1、放在底部,虽然放在底部照样会阻塞所有呈现,但不会阻塞资源下载。

2、如果嵌入JS放在head中,请把嵌入JS放在CSS头部。

3、使用defer（只支持IE）

4、不要在嵌入的JS中调用运行时间较长的函数,如果一定要用,可以用`setTimeout`来调用

# 其他问题

### 部分地区用户反应网站很卡，请问有哪些可能性的原因，以及解决方法？

### 除了前端以外还了解什么其它技术么？你最最厉害的技能是什么？

### 对前端工程师这个职位是怎么样理解的？它的前景会怎么样？

### 你移动端前端开发的理解？（和 Web 前端开发的主要区别是什么？）

### 你对加班的看法？

### 平时如何管理你的项目？

### 介绍一个你最得意的作品吧？

### 项目中遇到国哪些印象深刻的技术难题，具体是什么问题，怎么解决？

### 最近在学什么？能谈谈你未来3，5年给自己的规划吗？