

# 2019 年前端面试题宝典

## Beta 2.0

离线的面试题总会有过期的那一天，如想获取最新的面试题和答案，请登录跬步客学习平台, 查看最新面试题的网址: <https://www.kuibuke.com/exam>

免责声明：以下内容全部来源于互联网，如有侵权，请联系跬步客，我们将第一时间把线上的内容移除掉，谢谢理解。

## 目录

第一部分、HTML , HTTP , web 综合问题.....	5
前端需要注意哪些 SEO .....	5
web 开发中会话跟踪的方法有哪些 .....	5
<img>的 title 和 alt 有什么区别.....	6
doctype 是什么,举例常见 doctype 及特点 .....	6
HTML 全局属性(global attribute)有哪些.....	7
什么是 web 语义化,有什么好处 .....	7
HTTP method.....	8
从浏览器地址栏输入 url 到显示页面的步骤(以 HTTP 为例).....	8
HTTP request 报文结构是怎样的.....	11
HTTP response 报文结构是怎样的 .....	11
如何进行网站性能优化.....	12
什么是渐进增强 .....	13
HTTP 状态码及其含义 .....	14
第二部分、CSS 部分.....	15
CSS 选择器有哪些.....	15
css sprite 是什么,有什么优缺点 .....	17
display: none;与 visibility: hidden;的区别.....	17
css hack 原理及常用 hack .....	18
specified value,computed value,used value 计算方法.....	20
link 与@import 的区别 .....	20
display: block;和 display: inline;的区别 .....	20
PNG,GIF,JPG 的区别及如何选 .....	21
CSS 有哪些继承属性.....	22
IE6 浏览器有哪些常见的 bug,缺陷或者与标准不一致的地方,如何解决.....	22
容器包含若干浮动元素时如何清理(包含)浮动.....	24
什么是 FOUC?如何避免.....	25
如何创建块级格式化上下文(block formatting context),BFC 有什么用 .....	25
display,float,position 的关系.....	25

外边距折叠(collapsing margins).....	26
如何确定一个元素的包含块(containing block) .....	26
stacking context,布局规则 .....	27
如何水平居中一个元素.....	27
如何竖直居中一个元素.....	29
第三部分、javascript 概念部分 .....	30
DOM 元素 e 的 e.getAttribute(propName)和 e.propName 有什么区别和联系 .....	30
offsetWidth/offsetHeight,clientWidth/clientHeight 与 scrollWidth/scrollHeight 的区别.....	30
XMLHttpRequest 通用属性和方法 .....	31
focus/blur 与 focusin/focusout 的区别与联系 .....	31
mouseover/mouseout 与 mouseenter/mouseleave 的区别与联系.....	32
sessionStorage,localStorage,cookie 区别 .....	33
javascript 跨域通信.....	33
javascript 有哪几种数据类型 .....	34
什么闭包,闭包有什么用.....	34
javascript 有哪几种方法定义函数.....	35
应用程序存储和离线 web 应用.....	35
客户端存储 localStorage 和 sessionStorage.....	36
cookie 及其操作.....	36
javascript 有哪些方法定义对象.....	37
===运算符判断相等的流程是怎样的 .....	37
==运算符判断相等的流程是怎样的 .....	37
对象到字符串的转换步骤.....	38
对象到数字的转换步骤.....	38
<,>,<=,>=的比较规则.....	38
+运算符工作流程.....	38
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 .....	39
DOM 事件模型是如何的,编写一个 EventUtil 工具类实现事件管理兼容.....	39
评价一下三种方法实现继承的优缺点,并改进 .....	41

第四部分、javascript 编程部分 .....	42
请用原生 js 实现一个函数,给页面制定的任意一个元素添加一个透明遮罩(透明度可变,默认 0.2),使这个区域点击无效,要求兼容 IE8+ 及各主流浏览器,遮罩层效果如下图所示: .....	42
请用代码写出(今天是星期 x)其中 x 表示当天是星期几,如果当天是星期一,输出应该是"今天是星期一" .....	43
下面这段代码想要循环延时输出结果 0 1 2 3 4,请问输出结果是否正确,如果不正确,请说明为什么,并修改循环内的代码使其输出正确结果 .....	43
现有一个 Page 类,其原型对象上有许多以 post 开头的方法(如 postMsg);另有一拦截函数 chekc,只返回 ture 或 false.请设计一个函数,该函数应批量改造原 Page 的 postXXX 方法,在保留其原有功能的同时,为每个 postXXX 方法增加拦截验证功能,当 chekc 返回 true 时继续执行原 postXXX 方法,返回 false 时不再执行原 postXXX 方法.....	44
完成下面的 tool-tip .....	45
编写 javascript 深度克隆函数 deepClone .....	45
补充代码,鼠标单击 Button1 后将 Button1 移动到 Button2 的后面 .....	46
网页中实现一个计算当年还剩多少时间的倒计时程序,要求网页上实时动态显示"××年还剩××天××时××分××秒" .....	47
完成一个函数,接受数组作为参数,数组元素为整数或者数组,数组元素包含整数或数组,函数返回扁平化后的数组 .....	48
如何判断一个对象是否为数组 .....	49
请评价以下事件监听器代码并给出改进意见 .....	49
如何判断一个对象是否为函数 .....	50
编写一个函数接受 url 中 query string 为参数,返回解析后的 Object,query string 使用 application/x-www-form-urlencoded 编码.....	51
解析一个完整的 url,返回 Object 包含域与 window.location 相同 .....	52
完成函数 getViewPortSize 返回指定窗口的视口尺寸 .....	53
完成函数 getScrollOffset 返回窗口滚动条偏移量 .....	54
现有一个字符串 richText,是一段富文本,需要显示在页面上.有个要求,需要给其中只包含一个 img 元素的 p 标签增加一个叫 pic 的 class.请编写代码实现.可以使用 jQuery 或 KISSY.....	54

请实现一个 Event 类,继承自此类的对象都会拥有两个方法 on,off,once 和 trigger.....	55
编写一个函数将列表子元素顺序反转.....	56
以下函数的作用是?空白区域应该填写什么 .....	56
编写一个函数实现 form 的序列化(即将一个表单中的键值序列化为可提交的字符串) .....	57
使用原生 javascript 给下面列表中的 li 节点绑定点击事件,点击时创建一个 Object 对象,兼容 IE 和标准浏览器.....	59
有一个大数组,var a = ['1', '2', '3', ...];a 的长度是 100,内容填充随机整数的字符串.请先构造此数组 a,然后设计一个算法将其内容去重 .....	60

## 第一部分、HTML , HTTP , web 综合问题

### 前端需要注意哪些 SEO

1. 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
2. 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
3. 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
4. 重要内容不要用 js 输出：爬虫不会执行 js 获取内容
5. 少用 iframe：搜索引擎不会抓取 iframe 中的内容
6. 非装饰性图片必须加 alt
7. 提高网站速度：网站速度是搜索引擎排序的一个重要指标

### web 开发中会话跟踪的方法有哪些

1. cookie
2. session
3. url 重写
4. 隐藏 input

## 5. ip 地址

### <img>的 title 和 alt 有什么区别

1. title 是 [global attributes](#) 之一，用于为元素提供附加的 advisory information。通常当鼠标滑动到元素上的时候显示。
2. alt 是<img>的特有属性，是图片内容的等价描述，用于图片无法加载时显示、读屏器阅读图片。可提图片高可访问性，除了纯装饰图片外都必须设置有意义的值，搜索引擎会重点分析。

### doctype 是什么,举例常见 doctype 及特点

1. <!doctype>声明必须处于 HTML 文档的头部，在<html>标签之前，HTML5 中不区分大小写
2. <!doctype>声明不是一个 HTML 标签，是一个用于告诉浏览器当前 HTML 版本的指令
3. 现代浏览器的 html 布局引擎通过检查 doctype 决定使用兼容模式还是标准模式对文档进行渲染，一些浏览器有一个接近标准模型。
4. 在 HTML4.01 中<!doctype>声明指向一个 DTD，由于 HTML4.01 基于 SGML，所以 DTD 指定了标记规则以保证浏览器正确渲染内容
5. HTML5 不基于 SGML，所以不用指定 DTD

常见 doctype：

1. **HTML4.01 strict**：不允许使用表现性、废弃元素（如 font）以及 frameset。声明：  
`<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">`
2. **HTML4.01 Transitional**:允许使用表现性、废弃元素（如 font），不允许使用 frameset。声明：`<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">`
3. **HTML4.01 Frameset**:允许表现性元素，废弃元素以及 frameset。声明：`<!DOCTYPE  
HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"  
"http://www.w3.org/TR/html4/frameset.dtd">`
4. **XHTML1.0 Strict**:不使用允许表现性、废弃元素以及 frameset。文档必须是结构良好的 XML 文档。声明：`<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0  
Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">`
5. **XHTML1.0 Transitional**:允许使用表现性、废弃元素，不允许 frameset，文档必须是结构良好的 XML 文档。声明：`<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0  
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-  
transitional.dtd">`

6. **XHTML 1.0 Frameset**: 允许使用表现性、废弃元素以及 frameset，文档必须是结构良好的 XML 文档。声明：`<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">`
7. **HTML 5**: `<!doctype html>`

### HTML 全局属性(global attribute)有哪些

参考资料：[MDN: html global attribute](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Global_attributes) 或者 [W3C HTML global-attributes](https://www.w3.org/TR/html5/global-attributes)

- accesskey: 设置快捷键，提供快速访问元素如 [aaa](#) 在 windows 下的 firefox 中按 alt + shift + a 可激活元素
- class: 为元素设置类标识，多个类名用空格分开，CSS 和 javascript 可通过 class 属性获取元素
- contenteditable: 指定元素内容是否可编辑
- contextmenu: 自定义鼠标右键弹出菜单内容
- data-\*: 为元素增加自定义属性
- dir: 设置元素文本方向
- draggable: 设置元素是否可拖拽
- dropzone: 设置元素拖放类型：copy, move, link
- hidden: 表示一个元素是否与文档。样式上会导致元素不显示，但是不能用这个属性实现样式效果
- id: 元素 id，文档内唯一
- lang: 元素内容的语言
- spellcheck: 是否启动拼写和语法检查
- style: 行内 css 样式
- tabindex: 设置元素可以获得焦点，通过 tab 可以导航
- title: 元素相关的建议信息
- translate: 元素和子孙节点内容是否需要本地化

### 什么是 web 语义化,有什么好处

web 语义化是指通过 HTML 标记表示页面包含的信息，包含了 HTML 标签的语义化和 css 命名的语义化。HTML 标签的语义化是指：通过使用包含语义的标签（如 h1-h6）恰当地表示文档结构 css 命名的语义化是指：为 html 标签添加有意义的 class, id 补充未表达的语义，如 [Microformat](#) 通过添加符合规则的 class 描述信息 为什么需要语义化：

- 去掉样式后页面呈现清晰的结构

- 盲人使用读屏器更好地阅读
- 搜索引擎更好地理解页面，有利于收录
- 便团队项目的可持续运作及维护

## HTTP method

1. 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
2. **GET** 是最常用的方法，通常用于**请求服务器发送某个资源**。
3. **HEAD** 与 GET 类似，但**服务器在响应中只返回首部，不返回实体的主体部分**
4. **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，就用于这个主体替代它
5. **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。
6. **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。
7. **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。
8. **DELETE** 请求服务器删除请求 URL 指定的资源

## 从浏览器地址栏输入 url 到显示页面的步骤(以 HTTP 为例)

1. 在浏览器地址栏输入 URL
2. 浏览器查看**缓存**，如果请求资源在缓存中并且新鲜，跳转到转码步骤
  - i. 如果资源未缓存，发起新请求
  - ii. 如果已缓存，检验是否足够新鲜，足够新鲜直接提供给客户端，否则与服务器进行验证。
  - iii. 检验新鲜通常有两个 HTTP 头进行控制 Expires 和 Cache-Control :
    - HTTP1.0 提供 Expires，值为一个绝对时间表示缓存新鲜日期
    - HTTP1.1 增加了 Cache-Control: max-age=,值为以秒为单位的最大新鲜时间
3. 浏览器**解析 URL** 获取协议，主机，端口，path
4. 浏览器**组装一个 HTTP ( GET ) 请求报文**
5. 浏览器**获取主机 ip 地址**，过程如下：



- i. 浏览器缓存
  - ii. 本机缓存
  - iii. hosts 文件
  - iv. 路由器缓存
  - v. ISP DNS 缓存
  - vi. DNS 递归查询（可能存在负载均衡导致每次 IP 不一样）
6. 打开一个 socket 与目标 IP 地址，端口建立 TCP 链接，三次握手如下：
- i. 客户端发送一个 TCP 的 **SYN=1**，**Seq=X** 的包到服务器端口
  - ii. 服务器发回 **SYN=1**，**ACK=X+1**，**Seq=Y** 的响应包
  - iii. 客户端发送 **ACK=Y+1**，**Seq=Z**
7. TCP 链接建立后**发送 HTTP 请求**
8. 服务器接受请求并解析，将请求转发到服务程序，如虚拟主机使用 HTTP Host 头部判断请求的服务程序
9. 服务器检查 **HTTP 请求头是否包含缓存验证信息**如果验证缓存新鲜，返回 **304** 等对应状态码
10. 处理程序读取完整请求并准备 HTTP 响应，可能需要查询数据库等操作
11. 服务器将**响应报文通过 TCP 连接发送回浏览器**
12. 浏览器接收 HTTP 响应，然后根据情况选择**关闭 TCP 连接或者保留重用**，关闭 TCP 连接的四次握手如下：
- i. 主动方发送 **Fin=1**，**Ack=Z**，**Seq= X** 报文
  - ii. 被动方发送 **ACK=X+1**，**Seq=Z** 报文
  - iii. 被动方发送 **Fin=1**，**ACK=X**，**Seq=Y** 报文
  - iv. 主动方发送 **ACK=Y**，**Seq=X** 报文
13. 浏览器检查响应状态码：是否为 1XX，3XX，4XX，5XX，这些情况处理与 2XX 不同
14. 如果资源可缓存，**进行缓存**
15. 对响应进行**解码**（例如 gzip 压缩）
16. 根据资源类型决定如何处理（假设资源为 HTML 文档）
17. **解析 HTML 文档，构件 DOM 树，下载资源，构造 CSSOM 树，执行 js 脚本**，这些操作没有严格的先后顺序，以下分别解释
18. **构建 DOM 树**：
- i. **Tokenizing**：根据 HTML 规范将字符流解析为标记

- ii. **Lexing** : 词法分析将标记转换为对象并定义属性和规则
  - iii. **DOM construction** : 根据 HTML 标记关系将对象组成 DOM 树
19. 解析过程中遇到图片、样式表、js 文件, **启动下载**
20. 构建 **CSSOM 树** :
- i. **Tokenizing** : 字符流转换为标记流
  - ii. **Node** : 根据标记创建节点
  - iii. **CSSOM** : 节点创建 CSSOM 树
21. **根据 DOM 树和 CSSOM 树构建渲染树**:
- i. 从 DOM 树的根节点遍历所有**可见节点**, 不可见节点包括: 1) script,meta 这样本身不可见的标签。2)被 css 隐藏的节点, 如 display: none
  - ii. 对每一个可见节点, 找到恰当的 CSSOM 规则并应用
  - iii. 发布可视节点的内容和计算样式
22. **js 解析如下** :
- i. 浏览器创建 Document 对象并解析 HTML, 将解析到的元素和文本节点添加到文档中, 此时 **document.readyState 为 loading**
  - ii. HTML 解析器遇到**没有 async 和 defer 的 script 时**, 将他们添加到文档中, 然后执行行内或外部脚本。这些脚本会同步执行, 并且在脚本下载和执行时解析器会暂停。这样就可以用 document.write()把文本插入到输入流中。**同步脚本经常简单定义函数和注册事件处理程序, 他们可以遍历和操作 script 和他们之前的文档内容**
  - iii. 当解析器遇到设置了 **async** 属性的 script 时, 开始下载脚本并继续解析文档。脚本会在它**下载完成后尽快执行, 但是解析器不会停下来等它下载**。异步脚本**禁止使用 document.write()**, 它们可以访问自己 script 和之前的文档元素
  - iv. 当文档完成解析, document.readyState 变成 interactive
  - v. 所有 **defer** 脚本会**按照在文档出现的顺序执行**, 延迟脚本能访问完整文档树, 禁止使用 document.write()
  - vi. 浏览器在 **Document 对象上触发 DOMContentLoaded 事件**
  - vii. 此时文档完全解析完成, 浏览器可能还在等待如图片等内容加载, 等这些**内容完成载入并且所有异步脚本完成载入和执行**, document.readyState 变为 complete>window 触发 load 事件
23. **显示页面** ( HTML 解析过程中会逐步显示页面 )

## HTTP request 报文结构是怎样的

[rfc2616](#) 中进行了定义：

1. 首行是 **Request-Line** 包括：请求方法，请求 URI，协议版本，CRLF
2. 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
3. 请求头和消息实体之间有一个 **CRLF 分隔**
4. 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=qiue&age=25
```

## HTTP response 报文结构是怎样的

[rfc2616](#) 中进行了定义：

1. 首行是状态行包括：**HTTP 版本，状态码，状态描述**，后面跟一个 CRLF
2. 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**
3. 响应头部和响应实体之间用**一个 CRLF 空行分隔**
4. 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}
```

## 如何进行网站性能优化

[雅虎 Best Practices for Speeding Up Your Web Site](#) :

- content 方面
  - i. 减少 HTTP 请求：合并文件、CSS 精灵、inline Image
  - ii. 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
  - iii. 避免重定向：多余的中间访问
  - iv. 使 Ajax 可缓存
  - v. 非必须组件延迟加载
  - vi. 未来所需组件预加载
  - vii. 减少 DOM 元素数量
  - viii. 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
  - ix. 减少 iframe 数量
  - x. 不要 404
- Server 方面
  - i. 使用 CDN
  - ii. 添加 Expires 或者 Cache-Control 响应头
  - iii. 对组件使用 Gzip 压缩
  - iv. 配置 ETag
  - v. Flush Buffer Early
  - vi. Ajax 使用 GET 进行请求
  - vii. 避免空 src 的 img 标签
- Cookie 方面

- i. 减小 cookie 大小
  - ii. 引入资源的域名不要包含 cookie
- css 方面
  - i. 将样式表放到页面顶部
  - ii. 不使用 CSS 表达式
  - iii. 使用不使用@import
  - iv. 不使用 IE 的 Filter
- Javascript 方面
  - i. 将脚本放到页面底部
  - ii. 将 javascript 和 css 从外部引入
  - iii. 压缩 javascript 和 css
  - iv. 删除不需要的脚本
  - v. 减少 DOM 访问
  - vi. 合理设计事件监听器
- 图片方面
  - i. 优化图片：根据实际颜色需要选择色深、压缩
  - ii. 优化 css 精灵
  - iii. 不要在 HTML 中拉伸图片
  - iv. 保证 favicon.ico 小并且可缓存
- 移动方面
  - i. 保证组件小于 25k
  - ii. Pack Components into a Multipart Document

## 什么是渐进增强

渐进增强是指在 web 设计时强调可访问性、语义化 HTML 标签、外部样式表和脚本。保证所有人都能访问页面的基本内容和功能同时为高级浏览器和高带宽用户提供更好的用户体验。核心原则如下：

- 所有浏览器都必须能访问基本内容
- 所有浏览器都必须能使用基本功能
- 所有内容都包含在语义化标签中
- 通过外部 CSS 提供增强的布局
- 通过非侵入式、外部 javascript 提供增强功能
- end-user web browser preferences are respected

## HTTP 状态码及其含义

参考 [RFC 2616](#)

- 1XX：信息状态码
  - **100 Continue**：客户端应当继续发送请求。这个临时相应是用来通知客户端它的部分请求已经被服务器接收，且仍未被拒绝。客户端应当继续发送请求的剩余部分，或者如果请求已经完成，忽略这个响应。服务器必须在请求万仇向客户端发送一个最终响应
  - **101 Switching Protocols**：服务器已经理解客户端的请求，并将通过 Upgrade 消息头通知客户端采用不同的协议来完成这个请求。在发送完这个响应最后的空行后，服务器将会切换到 Upgrade 消息头中定义的那些协议。
- 2XX：成功状态码
  - **200 OK**：请求成功，请求所希望的响应头或数据体将随此响应返回
  - **201 Created**：
  - **202 Accepted**：
  - **203 Non-Authoritative Information**：
  - **204 No Content**：
  - **205 Reset Content**：
  - **206 Partial Content**：
- 3XX：重定向
  - **300 Multiple Choices**：
  - **301 Moved Permanently**：
  - **302 Found**：
  - **303 See Other**：
  - **304 Not Modified**：

- 305 Use Proxy :
- 306 ( unused ) :
- 307 Temporary Redirect :
- 4XX : 客户端错误
  - 400 Bad Request:
  - 401 Unauthorized:
  - 402 Payment Required:
  - 403 Forbidden:
  - 404 Not Found:
  - 405 Method Not Allowed:
  - 406 Not Acceptable:
  - 407 Proxy Authentication Required:
  - 408 Request Timeout:
  - 409 Conflict:
  - 410 Gone:
  - 411 Length Required:
  - 412 Precondition Failed:
  - 413 Request Entity Too Large:
  - 414 Request-URI Too Long:
  - 415 Unsupported Media Type:
  - 416 Requested Range Not Satisfiable:
  - 417 Expectation Failed:
- 5XX: 服务器错误
  - 500 Internal Server Error:
  - 501 Not Implemented:
  - 502 Bad Gateway:
  - 503 Service Unavailable:
  - 504 Gateway Timeout:
  - 505 HTTP Version Not Supported:

## 第二部分、CSS 部分

---

### CSS 选择器有哪些

1. **\*通用选择器**：选择所有元素，**不参与计算优先级**，兼容性 IE6+
2. **#X id 选择器**：选择 id 值为 X 的元素，兼容性：IE6+

3. **.X 类选择器**：选择 class 包含 X 的元素，兼容性：IE6+
4. **X Y 后代选择器**：选择满足 X 选择器的后代节点中满足 Y 选择器的元素，兼容性：IE6+
5. **X 元素选择器**：选择标所有签为 X 的元素，兼容性：IE6+
6. **:link, :visited, :focus, :hover, :active 链接状态**：选择特定状态的链接元素，顺序 LoVe HAte，兼容性：IE4+
7. **X + Y 直接兄弟选择器**：在 X 之后第一个兄弟节点中选择满足 Y 选择器的元素，兼容性：IE7+
8. **X > Y 子选择器**：选择 X 的子元素中满足 Y 选择器的元素，兼容性：IE7+
9. **X ~ Y 兄弟**：选择 X 之后所有兄弟节点中满足 Y 选择器的元素，兼容性：IE7+
10. **[attr]**：选择所有设置了 attr 属性的元素，兼容性 IE7+
11. **[attr=value]**：选择属性值刚好为 value 的元素
12. **[attr~=value]**：选择属性值为空白符分隔，其中一个的值刚好是 value 的元素
13. **[attr|=value]**：选择属性值刚好为 value 或者 value-开头的元素
14. **[attr^=value]**：选择属性值以 value 开头的元素
15. **[attr\$=value]**：选择属性值以 value 结尾的元素
16. **[attr=value]\***：选择属性值中包含 value 的元素
17. **[:checked]**：选择单选框，复选框，下拉框中选中状态下的元素，兼容性：IE9+
18. **X:after, X::after**：after 伪元素，选择元素虚拟子元素（元素的最后一个子元素），CSS3 中::表示伪元素。兼容性:after 为 IE8+，::after 为 IE9+
19. **:hover**：鼠标移入状态的元素，兼容性 a 标签 IE4+，所有元素 IE7+
20. **:not(selector)**：选择不符合 selector 的元素。**不参与计算优先级**，兼容性：IE9+
21. **::first-letter**：伪元素，选择块元素第一行的第一个字母，兼容性 IE5.5+
22. **::first-line**：伪元素，选择块元素的第一行，兼容性 IE5.5+
23. **:nth-child(an + b)**：伪类，选择前面有 an + b - 1 个兄弟节点的元素，其中 n >= 0，兼容性 IE9+
24. **:nth-last-child(an + b)**：伪类，选择后面有 an + b - 1 个兄弟节点的元素 其中 n >= 0，兼容性 IE9+
25. **X:nth-of-type(an+b)**：伪类，X 为选择器，**解析得到元素标签**，选择前面有 an + b - 1 个**相同标签**兄弟节点的元素。兼容性 IE9+
26. **X:nth-last-of-type(an+b)**：伪类，X 为选择器，解析得到元素标签，选择后面有 an+b-1 个**相同标签**兄弟节点的元素。兼容性 IE9+



27. **X:first-child** : 伪类, 选择满足 X 选择器的元素, 且这个元素是其父节点的第一个子元素。兼容性 IE7+
28. **X:last-child** : 伪类, 选择满足 X 选择器的元素, 且这个元素是其父节点的最后一个子元素。兼容性 IE9+
29. **X:only-child** : 伪类, 选择满足 X 选择器的元素, 且这个元素是其父元素的唯一子元素。兼容性 IE9+
30. **X:only-of-type** : 伪类, 选择 X 选择的元素, **解析得到元素标签**, 如果该元素没有相同类型的兄弟节点时选中它。兼容性 IE9+
31. **X:first-of-type** : 伪类, 选择 X 选择的元素, **解析得到元素标签**, 如果该元素 是此类型元素的第一个兄弟。选中它。兼容性 IE9+

### css sprite 是什么,有什么优缺点

概念: 将多个小图片拼接到一个图片中。通过 background-position 和元素尺寸调节需要显示的背景图案。

优点:

1. 减少 HTTP 请求数, 极大地提高页面加载速度
2. 增加图片信息重复度, 提高压缩比, 减少图片大小
3. 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:

1. 图片合并麻烦
2. 维护麻烦, 修改一个图片可能需要从新布局整个图片, 样式

### display: none;与 visibility: hidden;的区别

联系: 它们都能让元素不可见

区别:

1. display:none;会让元素完全从渲染树中消失, 渲染的时候不占据任何空间; visibility:hidden;不会让元素从渲染树消失, 渲染师元素继续占据空间, 只是内容不可见

2. display: none;是非继承属性，子孙节点消失由于元素从渲染树消失造成，通过修改子孙节点属性无法显示；visibility: hidden;是继承属性，子孙节点消失由于继承了hidden，通过设置 visibility: visible;可以让子孙节点显式
3. 修改常规流中元素的 display 通常会造成文档重排。修改 visibility 属性只会造成本元素的重绘。
4. 读屏器不会读取 display: none;元素内容；会读取 visibility: hidden;元素内容

### css hack 原理及常用 hack

原理：利用不同浏览器对 CSS 的支持和解析结果不一样编写针对特定浏览器样式。常见的 hack 有 1 ) 属性 hack。2 ) 选择器 hack。3 ) IE 条件注释

- IE 条件注释：适用于[IE5, IE9]常见格式如下

```
<!--[if IE 6]>
Special instructions for IE 6 here
<![endif]-->
```

- 选择器 hack：不同浏览器对选择器的支持不一样

```
/* ***** Selector Hacks ***** */

/* IE6 and below */
* html #uno { color: red }

/* IE7 */
*:first-child+html #dos { color: red }

/* IE7, FF, Saf, Opera */
html>body #tres { color: red }

/* IE8, FF, Saf, Opera (Everything but IE 6,7) */
html>/**/body #cuatro { color: red }

/* Opera 9.27 and below, safari 2 */
html:first-child #cinco { color: red }

/* Safari 2-3 */
html[xmlns=""] body:last-child #seis { color: red }

/* safari 3+, chrome 1+, opera9+, ff 3.5+ */
body:nth-of-type(1) #siete { color: red }

/* safari 3+, chrome 1+, opera9+, ff 3.5+ */
body:first-of-type #ocho { color: red }
```

```
/* saf3+, chrome1+ */
@media screen and (-webkit-min-device-pixel-ratio:0) {
  #diez { color: red }
}

/* iPhone / mobile webkit */
@media screen and (max-device-width: 480px) {
  #veintiseis { color: red }
}

/* Safari 2 - 3.1 */
html[xmlns*=""]:root #trece { color: red }

/* Safari 2 - 3.1, Opera 9.25 */
*|html[xmlns*=""] #catorce { color: red }

/* Everything but IE6-8 */
:root *> #quince { color: red }

/* IE7 */
*+html #dieciocho { color: red }

/* Firefox only. 1+ */
#veinticuatro, x:-moz-any-link { color: red }

/* Firefox 3.0+ */
#veinticinco, x:-moz-any-link, x:default { color: red }
```

- 属性 hack：不同浏览器解析 bug 或方法

```
/* IE6 */
#once { _color: blue }

/* IE6, IE7 */
#doce { *color: blue; /* or #color: blue */ }

/* Everything but IE6 */
#diecisiete { color/**/: blue }

/* IE6, IE7, IE8 */
#diecinueve { color: blue\9; }

/* IE7, IE8 */
#veinte { color/*\**/: blue\9; }

/* IE6, IE7 -- acts as an !important */
#veintesiete { color: blue !ie; } /* string after ! can be anything */
```

**specified value,computed value,used value 计算方法**

- specified value: 计算方法如下：
  - i. 如果样式表设置了一个值，使用这个值
  - ii. 如果没有设置值，这个属性是继承属性，从父元素继承
  - iii. 如果没设置，并且不是继承属性，使用 css 规范指定的初始值
- computed value: 以 specified value 根据规范定义的行为进行计算，通常将相对值计算为绝对值，例如 em 根据 font-size 进行计算。一些使用百分数并且需要布局来决定最终值的属性，如 width，margin。百分数就直接作为 computed value。line-height 的无单位值也直接作为 computed value。这些值将在计算 used value 时得到绝对值。

**computed value 的主要作用是用于继承**

- used value：属性计算后的最终值，对于大多数属性可以通过 window.getComputedStyle 获得，尺寸值单位为像素。以下属性依赖于布局，
  - background-position
  - bottom, left, right, top
  - height, width
  - margin-bottom, margin-left, margin-right, margin-top
  - min-height, min-width
  - padding-bottom, padding-left, padding-right, padding-top
  - text-indent

**link 与@import 的区别**

1. link 是 HTML 方式，@import 是 CSS 方式
2. link 最大限度支持并行下载，@import 过多嵌套导致串行下载，出现 [FOUC](#)
3. link 可以通过 rel="alternate stylesheet"指定候选样式
4. 浏览器对 link 支持早于@import，可以使用@import 对老浏览器隐藏样式
5. @import 必须在样式规则之前，可以在 css 文件中引用其他文件
6. 总体来说：[link 优于@import](#)

**display: block;和 display: inline;的区别**

block 元素特点：

1.处于常规流中时，如果 width 没有设置，会自动填充满父容器 2.可以应用 margin/padding 3.在没有设置高度的情况下会扩展高度以包含常规流中的子元素 4.处于常规流中时布局时在前后元素位置之间（独占一个水平空间） 5.忽略 vertical-align

inline 元素特点

1.水平方向上根据 direction 依次布局 2.不会在元素前后进行换行 3.受 white-space 控制 4.margin/padding 在竖直方向上无效，水平方向上有效 5.width/height 属性对非替换行内元素无效，宽度由元素内容决定 6.非替换行内元素的行框高由 line-height 确定，替换行内元素的行框高由 height,margin,padding,border 决定 6.浮动或绝对定位时会转换为 block 7.vertical-align 属性生效

### PNG,GIF,JPG 的区别及如何选

参考资料：[选择正确的图片格式](#) GIF:

1. 8 位像素，256 色
2. 无损压缩
3. 支持简单动画
4. 支持 boolean 透明
5. 适合简单动画

JPEG :

1. 颜色限于 256
2. 有损压缩
3. 可控制压缩质量
4. 不支持透明
5. 适合照片

PNG :

1. 有 PNG8 和 truecolor PNG
2. PNG8 类似 GIF 颜色上限为 256，文件小，支持 alpha 透明度，无动画
3. 适合图标、背景、按钮

## CSS 有哪些继承属性

- 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](#)
- [line-height](#)
- [color](#)
- [visibility](#)
- [cursor](#)

## IE6 浏览器有哪些常见的 bug,缺陷或者与标准不一致的地方,如何解决

- IE6 不支持 min-height，解决办法使用 css hack：

```
.target {
  min-height: 100px;
  height: auto !important;
  height: 100px;    // IE6 下内容高度超过会自动扩展高度
}
```

- ol 内 li 的序号全为 1，不递增。解决方法：为 li 设置样式 display: list-item;
- 未定位父元素 overflow: auto;，包含 position: relative;子元素，子元素高于父元素时会溢出。解决办法：1) 子元素去掉 position: relative;; 2) 不能为子元素去掉定位时，父元素 position: relative;

```
<style type="text/css">
.outer {
  width: 215px;
  height: 100px;
  border: 1px solid red;
  overflow: auto;
  position: relative; /* 修复 bug */
}
.inner {
  width: 100px;
  height: 200px;
```

```
background-color: purple;
position: relative;
}
</style>

<div class="outer">
  <div class="inner"></div>
</div>
```

- IE6 只支持 a 标签的: hover 伪类, 解决方法: 使用 js 为元素监听 mouseenter, mouseleave 事件, 添加类实现效果:

```
<style type="text/css">
.p: hover,
.hover {
background: purple;
}
</style>

<p class="p" id="target">aaaa bbbbbb<span>DDDDDDDDDDd</span> aaaa lkjlkjdfj</p>

<script type="text/javascript">
function addClass(elem, cls) {
  if (elem.className) {
    elem.className += ' ' + cls;
  } else {
    elem.className = cls;
  }
}
function removeClass(elem, cls) {
  var className = ' ' + elem.className + ' ';
  var reg = new RegExp(' + ' + cls + ' +', 'g');
  elem.className = className.replace(reg, ' ').replace(/^ +| +$/, '');
}

var target = document.getElementById('target');
if (target.attachEvent) {
  target.attachEvent('onmouseenter', function () {
    addClass(target, 'hover');
  });
  target.attachEvent('onmouseleave', function () {
    removeClass(target, 'hover');
  })
}
</script>
```

- IE5-8 不支持 opacity, 解决办法:

```
.opacity {
```

```
opacity: 0.4
filter: alpha(opacity=60); /* for IE5-7 */
-ms-filter: "progid:DXImageTransform.Microsoft.Alpha(Opacity=60)"; /*
for IE 8*/
}
```

- IE6 在设置 height 小于 font-size 时高度值为 font-size , 解决办法 : font-size: 0;
- IE6 不支持 PNG 透明背景 , 解决办法: **IE6 下使用 gif 图片**
- IE6-7 不支持 display: inline-block 解决办法 : 设置 inline 并触发 hasLayout

```
display: inline-block;
*display: inline;
*zoom: 1;
```

- IE6 下浮动元素在浮动方向上与父元素边界接触元素的外边距会加倍。解决办法 : 1 ) 使用 padding 控制间距。 2 ) 浮动元素 display: inline;这样解决问题且无任何副作用 : css 标准规定浮动元素 display:inline 会自动调整为 block
- 通过为块级元素设置宽度和左右 margin 为 auto 时 , IE6 不能实现水平居中 , 解决方法 : 为父元素设置 text-align: center;

### 容器包含若干浮动元素时如何清理(包含)浮动

1. 容器元素闭合标签前添加额外元素并设置 clear: both
2. 父元素触发块级格式化上下文(见块级可视化上下文部分)
3. 设置容器元素伪元素进行清理[推荐的清理浮动方法](#)

```
/**
 * 在标准浏览器下使用
 * 1 content 内容为空格用于修复 opera 下文档中出现
 *   contenteditable 属性时在清理浮动元素上下的空白
 * 2 使用 display 使用 table 而不是 block : 可以防止容器和
 *   子元素 top-margin 折叠,这样能使清理效果与 BFC , IE6/7
 *   zoom: 1;一致
 */

.clearfix:before,
.clearfix:after {
  content: " "; /* 1 */
  display: table; /* 2 */
}
```



```
.clearfix:after {
    clear: both;
}

/**
 * IE 6/7 下使用
 * 通过触发 hasLayout 实现包含浮动
 */
.clearfix {
    *zoom: 1;
}
```

### 什么是 FOUC?如何避免

Flash Of Unstyled Content : 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重新显示文档, 造成页面闪烁。解决方法: 把样式表放到文档的 head

### 如何创建块级格式化上下文(block formatting context),BFC 有什么用

创建规则:

1. 根元素
2. 浮动元素 ( float 不是 none )
3. 绝对定位元素 ( position 取值为 absolute 或 fixed )
4. display 取值为 inline-block, table-cell, table-caption, flex, inline-flex 之一的元素
5. overflow 不是 visible 的元素

作用:

1. 可以包含浮动元素
2. 不被浮动元素覆盖
3. 阻止父子元素的 margin 折叠

### display, float, position 的关系

1. 如果 display 为 none, 那么 position 和 float 都不起作用, 这种情况下元素不产生框
2. 否则, 如果 position 值为 absolute 或者 fixed, 框就是绝对定位的, float 的计算值为 none, display 根据下面的表格进行调整。

3. 否则，如果 float 不是 none，框是浮动的，display 根据下表进行调整
4. 否则，如果元素是根元素，display 根据下表进行调整
5. 其他情况下 display 的值为指定值 总结起来：**绝对定位、浮动、根元素都需要调整**

display

### 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

### 如何确定一个元素的包含块(containing block)

1. 根元素的包含块叫做初始包含块，在连续媒体中他的尺寸与 viewport 相同并且 anchored at the canvas origin；对于 paged media，它的尺寸等于 page area。初始包含块的 direction 属性与根元素相同。
2. position 为 relative 或者 static 的元素，它的包含块由最近的块级 ( display 为 block, list-item, table ) 祖先元素的**内容框**组成
3. 如果元素 position 为 fixed。对于连续媒体，它的包含块为 viewport；对于 paged media，包含块为 page area
4. 如果元素 position 为 absolute，它的包含块由祖先元素中最近一个 position 为 relative, absolute 或者 fixed 的元素产生，规则如下：
  - 如果祖先元素为行内元素，the containing block is the bounding box around the **padding boxes** of the first and the last inline boxes generated for that element.
  - 其他情况下包含块由祖先节点的 **padding edge** 组成

如果找不到定位的祖先元素，包含块为**初始包含块**

## stacking context, 布局规则

z 轴上的默认层叠顺序如下（从下到上）：

1. 根元素的边界和背景
2. 常规流中的元素按照 html 中顺序
3. 浮动块
4. positioned 元素按照 html 中出现顺序

如何创建 stacking context：

1. 根元素
2. z-index 不为 auto 的定位元素
3. a flex item with a z-index value other than 'auto'
4. opacity 小于 1 的元素
5. 在移动端 webkit 和 chrome22+ , z-index 为 auto , position: fixed 也将创建新的 stacking context

## 如何水平居中一个元素

- 如果需要居中的元素为**常规流中 inline 元素**，为父元素设置 text-align: center; 即可实现
- 如果需要居中的元素为**常规流中 block 元素**，1) 为元素设置宽度，2) 设置左右 margin 为 auto。3) IE6 下需在父元素上设置 text-align: center; 再给子元素恢复需要的值

```
<body>
  <div class="content">
    aaaaaa aaaaaa a a a a a a a
  </div>
</body>

<style>
  body {
    background: #DDD;
    text-align: center; /* 3 */
  }
  .content {
    width: 500px;      /* 1 */
    text-align: left;  /* 3 */
    margin: 0 auto;    /* 2 */
  }
</style>
```

```
        background: purple;
    }
</style>
```

- 如果需要居中的元素为**浮动元素**，1) 为元素设置宽度，2) position: relative;，3) 浮动方向偏移量 (left 或者 right) 设置为 50%，4) 浮动方向上的 margin 设置为元素宽度一半乘以-1

```
<body>
  <div class="content">
    aaaaaa aaaaaa a a a a a a a
  </div>
</body>

<style>
  body {
    background: #DDD;
  }
  .content {
    width: 500px;           /* 1 */
    float: left;

    position: relative;     /* 2 */
    left: 50%;              /* 3 */
    margin-left: -250px;    /* 4 */

    background-color: purple;
  }
</style>
```

- 如果需要居中的元素为**绝对定位元素**，1) 为元素设置宽度，2) 偏移量设置为 50%，3) 偏移方向外边距设置为元素宽度一半乘以-1

```
<body>
  <div class="content">
    aaaaaa aaaaaa a a a a a a a
  </div>
</body>

<style>
  body {
    background: #DDD;
    position: relative;
  }
  .content {
    width: 800px;

    position: absolute;
```

```
    left: 50%;  
    margin-left: -400px;  
  
    background-color: purple;  
  }  
</style>
```

- 如果需要居中的元素为**绝对定位元素**，1) 为元素设置宽度，2) 设置左右偏移量都为 0, 3) 设置左右外边距都为 auto

```
<body>  
  <div class="content">  
    aaaaaa aaaaaa a a a a a a a  
  </div>  
</body>  
  
<style>  
  body {  
    background: #DDD;  
    position: relative;  
  }  
  .content {  
    width: 800px;  
  
    position: absolute;  
    margin: 0 auto;  
    left: 0;  
    right: 0;  
  
    background-color: purple;  
  }  
</style>
```

## 如何竖直居中一个元素

参考资料：[6 Methods For Vertical Centering With CSS](#)。 [盘点 8 种 CSS 实现垂直居中](#)

- 需要居中元素为**单行文本**，为包含文本的元素设置大于 font-size 的 line-height：

```
<p class="text">center text</p>  
  
<style>  
.text {  
  line-height: 200px;  
}  
</style>
```

### 第三部分、javascript 概念部分

#### DOM 元素 e 的 e.getAttribute(propName)和 e.propName 有什么区别和联系

- e.getAttribute(), 是标准 DOM 操作文档元素属性的方法, 具有通用性可在任意文档上使用, 返回元素在源文件中**设置的属性**
- e.propName 通常是在 HTML 文档中访问特定元素的**特性**, 浏览器解析元素后生成对应对象 (如 a 标签生成 HTMLAnchorElement), 这些对象的特性会根据特定规则结合属性设置得到, 对于没有对应特性的属性, 只能使用 getAttribute 进行访问
- e.getAttribute()返回值是源文件中设置的值, 类型是字符串或者 null (有的实现返回 "")
- e.propName 返回值可能是字符串、布尔值、对象、undefined 等
- 大部分 attribute 与 property 是一一对应关系, 修改其中一个会影响另一个, 如 id, title 等属性
- 一些布尔属性<input hidden/>的检测设置需要 hasAttribute 和 removeAttribute 来完成, 或者设置对应 property
- 像<a href="../index.html">link</a>中 href 属性, 转换成 property 的时候需要通过转换得到完整 URL
- 一些 attribute 和 property 不是一一对应如: form 控件中<input value="hello"/>对应的是 defaultValue, 修改或设置 value property 修改的是控件当前值, setAttribute 修改 value 属性不会改变 value property

#### 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**

[Measuring Element Dimension and Location with CSSOM in Windows Internet Explorer 9](#)

## XMLHttpRequest 通用属性和方法

1. readyState:表示请求状态的整数，取值：
  - UNSENT ( 0 ) : 对象已创建
  - OPENED ( 1 ) : open()成功调用，在这个状态下，可以为 xhr 设置请求头，或者使用 send()发送请求
  - HEADERS\_RECEIVED(2) : 所有重定向已经自动完成访问，并且最终响应的 HTTP 头已经收到
  - LOADING(3) : 响应体正在接收
  - DONE(4) : 数据传输完成或者传输产生错误
3. onreadystatechange : readyState 改变时调用的函数
4. status : 服务器返回的 HTTP 状态码 ( 如 , 200 , 404 )
5. statusText:服务器返回的 HTTP 状态信息 ( 如 , OK , No Content )
6. responseText:作为字符串形式的来自服务器的完整响应
7. responseXML: Document 对象，表示服务器的响应解析成的 XML 文档
8. abort():取消异步 HTTP 请求
9. getAllResponseHeaders(): 返回一个字符串，包含响应中服务器发送的全部 HTTP 报头。每个报头都是一个用冒号分隔开的名/值对，并且使用一个回车/换行来分隔报头行
10. getResponseHeader(headerName):返回 headName 对应的报头值
11. open(method, url, asynchronous [, user, password]):初始化准备发送到服务器上的请求。method 是 HTTP 方法，不区分大小写；url 是请求发送的相对或绝对 URL；asynchronous 表示请求是否异步；user 和 password 提供身份验证
12. setRequestHeader(name, value):设置 HTTP 报头
13. send(body):对服务器请求进行初始化。参数 body 包含请求的主体部分，对于 POST 请求为键值对字符串；对于 GET 请求，为 null

## focus/blur 与 focusin/focusout 的区别与联系

1. focus/blur 不冒泡，focusin/focusout 冒泡
2. focus/blur 兼容性好，focusin/focusout 在除 FireFox 外的浏览器下都保持良好兼容性，如需使用事件托管，可考虑在 FireFox 下使用事件捕获  
elem.addEventListener('focus', handler, true)
3. 可获得焦点的元素：

- i. window
- ii. 链接被点击或键盘操作
- iii. 表单空间被点击或键盘操作
- iv. 设置 tabindex 属性的元素被点击或键盘操作

### mouseover/mouseout 与 mouseenter/mouseleave 的区别与联系

1. mouseover/mouseout 是标准事件，**所有浏览器都支持**；mouseenter/mouseleave 是 IE5.5 引入的特有事件后来被 DOM3 标准采纳，现代标准浏览器也支持
2. mouseover/mouseout 是**冒泡事件**；mouseenter/mouseleave **不冒泡**。需要为**多个元素**监听鼠标移入/出事件时，**推荐 mouseover/mouseout 托管，提高性能**
3. 标准事件模型中 event.target 表示发生移入/出的元素，**event.relatedTarget** 对应移出/如元素；在老 IE 中 event.srcElement 表示发生移入/出的元素，**event.toElement** 表示移出的目标元素，**event.fromElement** 表示移入时的来源元素

例子：鼠标从 div#target 元素移出时进行处理，判断逻辑如下：

```
<div id="target"><span>test</span></div>

<script type="text/javascript">
var target = document.getElementById('target');
if (target.addEventListener) {
    target.addEventListener('mouseout', mouseoutHandler, false);
} else if (target.attachEvent) {
    target.attachEvent('onmouseout', mouseoutHandler);
}

function mouseoutHandler(e) {
    e = e || window.event;
    var target = e.target || e.srcElement;

    // 判断移出鼠标的元素是否为目标元素
    if (target.id !== 'target') {
        return;
    }

    // 判断鼠标是移出元素还是移到子元素
    var relatedTarget = event.relatedTarget || e.toElement;
    while (relatedTarget !== target
        && relatedTarget.nodeName.toUpperCase() !== 'BODY') {
        relatedTarget = relatedTarget.parentNode;
    }
}
```



```
// 如果相等，说明鼠标在元素内部移动
if (relatedTarget === target) {
  return;
}

// 执行需要操作
//alert('鼠标移出');

}
</script>
```

### sessionStorage,localStorage,cookie 区别

1. 都会在浏览器端保存，有大小限制，同源限制
2. cookie 会在请求时发送到服务器，作为会话标识，服务器可修改 cookie；web storage 不会发送到服务器
3. cookie 有 path 概念，子路径可以访问父路径 cookie，父路径不能访问子路径 cookie
4. 有效期：cookie 在设置的有效期内有效，默认为浏览器关闭；sessionStorage 在窗口关闭前有效，localStorage 长期有效，直到用户删除
5. 共享：sessionStorage 不能共享，localStorage 在同源文档之间共享，cookie 在同源且符合 path 规则的文档之间共享
6. localStorage 的修改会促发其他文档窗口的 update 事件
7. cookie 有 secure 属性要求 HTTPS 传输
8. 浏览器不能保存超过 300 个 cookie，单个服务器不能超过 20 个，每个 cookie 不能超过 4k。web storage 大小支持能达到 5M

### javascript 跨域通信

同源：两个文档同源需满足

1. 协议相同
2. 域名相同
3. 端口相同

跨域通信：js 进行 DOM 操作、通信时如果目标与当前窗口不满足同源条件，浏览器为了安全会阻止跨域操作。跨域通信通常有以下方法

- 如果是 log 之类的简单**单项通信**，新建<img>,<script>,<link>,<iframe>元素，通过 src，href 属性设置为目标 url。实现跨域请求
- 如果请求 **json 数据**，使用<script>进行 jsonp 请求
- 现代浏览器中**多窗口通信**使用 HTML5 规范的 targetWindow.postMessage(data, origin);其中 data 是需要发送的对象，origin 是目标窗口的 origin。  
window.addEventListener('message', handler, false);handler 的 event.data 是 postMessage 发送来的数据，event.origin 是发送窗口的 origin，event.source 是发送消息的窗口引用
- 内部服务器代理请求跨域 url，然后返回数据
- 跨域请求数据，现代浏览器可使用 HTML5 规范的 CORS 功能，只要目标服务器返回 HTTP 头部\*\*Access-Control-Allow-Origin: \*\*\*即可像普通 ajax 一样访问跨域资源

## javascript 有哪几种数据类型

### 六种基本数据类型

- undefined
- null
- string
- boolean
- number
- [symbol](#)(ES6)

### 一种引用类型

- Object

## 什么闭包,闭包有什么用

**闭包是在某个作用域内定义的函数，它可以访问这个作用域内的所有变量。**闭包作用域链通常包括三个部分：

1. 函数本身作用域。
2. 闭包定义时的作用域。
3. 全局作用域。

闭包常见用途：

1. 创建特权方法用于访问控制
2. 事件处理程序及回调

### javascript 有哪几种方法定义函数

1. [函数声明表达式](#)
2. [function 操作符](#)
3. [Function 构造函数](#)
4. [ES6:arrow function](#)

重要参考资料：[MDN:Functions and function scope](#)

### 应用程序存储和离线 web 应用

HTML5 新增应用程序缓存，允许 web 应用将应用程序自身保存到用户浏览器中，用户离线状态也能访问。1.为 html 元素设置 manifest 属性:<html manifest="myapp.appcache">，其中后缀名只是一个约定，真正识别方式是通过 text/cache-manifest 作为 MIME 类型。所以需要配置服务器保证设置正确 2.manifest 文件首行为 CACHE MANIFEST，其余就是要缓存的 URL 列表，每个一行，相对路径都相对于 manifest 文件的 url。注释以#开头 3.url 分为三种类型：CACHE:为默认类型。NETWORK：表示资源从不缓存。 FALLBACK:每行包含两个 url，第二个 URL 是指需要加载和存储在缓存中的资源，第一个 URL 是一个前缀。任何匹配该前缀的 URL 都不会缓存，如果从网络中载入这样的 URL 失败的话，就会用第二个 URL 指定的缓存资源来替代。以下是一个文件例子：

```
CACHE MANIFEST

CACHE:
myapp.html
myapp.css
myapp.js

FALLBACK:
videos/ offline_help.html

NETWORK:
cgi/
```

## 客户端存储 localStorage 和 sessionStorage

- localStorage 有效期为永久，sessionStorage 有效期为顶层窗口关闭前
- 同源文档可以读取并修改 localStorage 数据，sessionStorage 只允许同一个窗口下的文档访问，如通过 iframe 引入的同源文档。
- Storage 对象通常被当做普通 javascript 对象使用：**通过设置属性来存取字符串值**，也可以通过 **setItem(key, value)** 设置，**getItem(key)** 读取，**removeItem(key)** 删除，**clear()** 删除所有数据，**length** 表示已存储的数据项数目，**key(index)** 返回对应索引的 key

```
localStorage.setItem('x', 1); // store x->1
localStorage.getItem('x'); // return value of x

// 枚举所有存储的键值对
for (var i = 0, len = localStorage.length; i < len; ++i) {
    var name = localStorage.key(i);
    var value = localStorage.getItem(name);
}

localStorage.removeItem('x'); // remove x
localStorage.clear(); // remove all data
```

## cookie 及其操作

- cookie 是 web 浏览器存储的少量数据，最早设计为服务器端使用，作为 HTTP 协议的扩展实现。cookie 数据会自动在浏览器和服务器之间传输。
- 通过读写 cookie 检测是否支持
- cookie 属性有**名，值，max-age，path，domain，secure**；
- cookie 默认有效期为浏览器会话，一旦用户关闭浏览器，数据就丢失，通过设置 **max-age=seconds** 属性告诉浏览器 cookie 有效期
- cookie 作用域通过**文档源**和**文档路径**来确定，通过 **path** 和 **domain** 进行配置，web 页面同目录或子目录文档都可访问
- 通过 cookie 保存数据的方法为：为 document.cookie 设置一个符合目标的字符串如下
- 读取 document.cookie 获得';'分隔的字符串，key=value,解析得到结果

```
document.cookie = 'name=qiu; max-age=9999; path=/; domain=domain; secure';

document.cookie = 'name=aaa; path=/; domain=domain; secure';
// 要改变 cookie 的值，需要使用相同的名字、路径和域，新的值
```

```
// 来设置 cookie，同样的方法可以用来改变有效期

// 设置 max-age 为 0 可以删除指定 cookie

//读取 cookie，访问 document.cookie 返回键值对组成的字符串，
//不同键值对之间用'; '分隔。通过解析获得需要的值
cookieUtil.js：自己写的 cookie 操作工具
```

### javascript 有哪些方法定义对象

1. 对象字面量：var obj = {};
2. 构造函数：var obj = new Object();
3. Object.create(): var obj = Object.create(Object.prototype);

### ===运算符判断相等的流程是怎样的

1. 如果两个值不是相同类型，它们不相等
2. 如果两个值都是 null 或者都是 undefined，它们相等
3. 如果两个值都是布尔类型 true 或者都是 false，它们相等
4. 如果其中有一个是 NaN，它们不相等
5. 如果都是数值型并且数值相等，他们相等，-0 等于 0
6. 如果他们都是字符串并且在相同位置包含相同的 16 位值，他它们相等；如果在长度或者内容上不等，它们不相等；两个字符串显示结果相同但是编码不同==和===都认为他们不相等
7. 如果他们指向相同对象、数组、函数，它们相等；如果指向不同对象，他们不相等

### ==运算符判断相等的流程是怎样的

1. 如果两个值类型相同，按照===比较方法进行比较
2. 如果类型不同，使用如下规则进行比较
3. 如果其中一个值是 null，另一个是 undefined，它们相等
4. 如果一个值是数字另一个是字符串，将字符串转换为数字进行比较
5. 如果有布尔类型，将 true 转换为 1，false 转换为 0，然后用==规则继续比较
6. 如果一个值是对象，另一个是数字或字符串，将对象转换为原始值然后用==规则继续比较

## 7. 其他所有情况都认为不相等

### 对象到字符串的转换步骤

1. 如果对象有 `toString()` 方法，javascript 调用它。如果返回一个原始值（primitive value 如：string number boolean），将这个值转换为字符串作为结果
2. 如果对象没有 `toString()` 方法或者返回值不是原始值，javascript 寻找对象的 `valueOf()` 方法，如果存在就调用它，返回结果是原始值则转为字符串作为结果
3. 否则，javascript 不能从 `toString()` 或者 `valueOf()` 获得一个原始值，此时 throws a `TypeError`

### 对象到数字的转换步骤

1. 如果对象有 `valueOf()` 方法并且返回元素值，javascript 将返回值转换为数字作为结果
2. 否则，如果对象有 `toString()` 并且返回原始值，javascript 将返回结果转换为数字作为结果
3. 否则，throws a `TypeError`

### <, >, <=, >= 的比较规则

所有比较运算符都支持任意类型，但是**比较只支持数字和字符串**，所以需要执行必要的转换然后进行比较，转换规则如下：

1. 如果操作数是对象，转换为原始值：如果 `valueOf` 方法返回原始值，则使用这个值，否则使用 `toString` 方法的结果，如果转换失败则报错
2. 经过必要的对象到原始值的转换后，如果两个操作数都是字符串，按照字母顺序进行比较（他们的 16 位 unicode 值的大小）
3. 否则，如果有一个操作数不是字符串，**将两个操作数转换为数字**进行比较

### + 运算符工作流程

1. 如果有操作数是对象，转换为原始值
2. 此时如果有一个**操作数是字符串**，其他的操作数都转换为字符串并执行连接
3. 否则：**所有操作数都转换为数字并执行加法**

**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);

**DOM 事件模型是如何的,编写一个 EventUtil 工具类实现事件管理兼容**

- DOM 事件包含捕获（capture）和冒泡（bubble）两个阶段：捕获阶段事件从 window 开始触发事件然后通过祖先节点一次传递到触发事件的 DOM 元素上；冒泡阶段事件从初始元素依次向祖先节点传递直到 window
- 标准事件监听 elem.addEventListener(type, handler, capture)/elem.removeEventListener(type, handler, capture)：handler 接收保存事件信息的 event 对象作为参数，event.target 为触发事件的对象，handler 调用上下文 this 为绑定监听器的对象，event.preventDefault()取消事件默认行为，event.stopPropagation()/event.stopImmediatePropagation()取消事件传递
- 老版本 IE 事件监听 elem.attachEvent('on'+type, handler)/elem.detachEvent('on'+type, handler)：handler 不接收 event 作为参数，事件信息保存在 window.event 中，触发事件的对象为 event.srcElement，handler 执行上下文 this 为 window 使用闭包中调用 handler.call(elem, event)可模仿标准模型，然后返回闭包，保证了监听器的移除。event.returnValue 为 false 时取消事件默认行为，event.cancelBubble 为 true 时取消时间传播
- 通常利用事件冒泡机制托管事件处理程序提高程序性能。

```
/**
 * 跨浏览器事件处理工具。只支持冒泡。不支持捕获
 * @author (qiu_deqing@126.com)
 */
var EventUtil = {
  getEvent: function (event) {
```

```
    return event || window.event;
  },
  getTarget: function (event) {
    return event.target || event.srcElement;
  },
  // 返回注册成功的监听器, IE 中需要使用返回值来移除监听器
  on: function (elem, type, handler) {
    if (elem.addEventListener) {
      elem.addEventListener(type, handler, false);
      return handler;
    } else if (elem.attachEvent) {
      var wrapper = function () {
        var event = window.event;
        event.target = event.srcElement;
        handler.call(elem, event);
      };
      elem.attachEvent('on' + type, wrapper);
      return wrapper;
    }
  },
  off: function (elem, type, handler) {
    if (elem.removeEventListener) {
      elem.removeEventListener(type, handler, false);
    } else if (elem.detachEvent) {
      elem.detachEvent('on' + type, handler);
    }
  },
  preventDefault: function (event) {
    if (event.preventDefault) {
      event.preventDefault();
    } else if ('returnValue' in event) {
      event.returnValue = false;
    }
  },
  stopPropagation: function (event) {
    if (event.stopPropagation) {
      event.stopPropagation();
    } else if ('cancelBubble' in event) {
      event.cancelBubble = true;
    }
  },
  /**
   * keypress 事件跨浏览器获取输入字符
   * 某些浏览器在一些特殊键上也触发 keypress, 此时返回 null
   */
  getChar: function (event) {
    if (event.which == null) {
      return String.fromCharCode(event.keyCode); // IE
    }
    else if (event.which != 0 && event.charCode != 0) {
      return String.fromCharCode(event.which); // the rest
    }
  }
}
```



```
    }  
    else {  
        return null;    // special key  
    }  
}  
};
```

### 评价一下三种方法实现继承的优缺点,并改进

```
function Shape() {}  
  
function Rect() {}  
  
// 方法 1  
Rect.prototype = new Shape();  
  
// 方法 2  
Rect.prototype = Shape.prototype;  
  
// 方法 3  
Rect.prototype = Object.create(Shape.prototype);  
  
Rect.prototype.area = function () {  
    // do something  
};
```

方法 1：

1. 优点：正确设置原型链实现继承
2. 优点：父类实例属性得到继承，原型链查找效率提高，也能对一些属性提供合理的默认值
3. 缺点：父类实例属性为引用类型时，不恰当地修改会导致所有子类被修改
4. 缺点：创建父类实例作为子类原型时，可能无法确定构造函数需要的合理参数，这样提供的参数继承给子类没有实际意义，当子类需要这些参数时应该在构造函数中进行初始化和设置
5. 总结：继承应该是继承方法而不是属性，为子类设置父类实例属性应该是通过在子类构造函数中调用父类构造函数进行初始化

方法 2：

1. 优点：正确设置原型链实现继承
2. 缺点：父类构造函数原型与子类相同。修改子类原型添加方法会修改父类

方法 3：

1. 优点：正确设置原型链且避免方法 1.2 中的缺点
2. 缺点：ES5 方法需要注意兼容性

改进：

1. 所有三种方法应该在子类构造函数中调用父类构造函数实现实例属性初始化

```
function Rect() {  
    Shape.call(this);  
}
```

2. 用新创建的对象替代子类默认原型，设置 `Rect.prototype.constructor = Rect`；  
保证一致性
3. 第三种方法的 polyfill：

```
function create(obj) {  
    if (Object.create) {  
        return Object.create(obj);  
    }  
  
    function f() {};  
    f.prototype = obj;  
    return new f();  
}
```

#### 第四部分、javascript 编程部分

请用原生 js 实现一个函数,给页面制定的任意一个元素添加一个透明遮罩(透明度可变,默认 0.2),使这个区域点击无效,要求兼容 IE8+ 及各主流浏览器,遮罩层效果如下图所示:

```
<style>  
#target {  
    width: 200px;  
    height: 300px;  
    margin: 40px;  
    background-color: tomato;  
}  
</style>  
  
<div id="target"></div>
```

```
<script>
function addMask(elem, opacity) {
  opacity = opacity || 0.2;

  var rect = elem.getBoundingClientRect();
  var style = getComputedStyle(elem, null);

  var mask = document.createElement('div');
  mask.style.position = 'absolute';
  var marginLeft = parseFloat(style.marginLeft);
  mask.style.left = (elem.offsetLeft - marginLeft) + 'px';
  var marginTop = parseFloat(style.marginTop);
  mask.style.top = (elem.offsetTop - marginTop) + 'px';
  mask.style.zIndex = 9999;
  mask.style.opacity = '' + opacity;
  mask.style.backgroundColor = '#000';

  mask.style.width = (parseFloat(style.marginLeft) +
    parseFloat(style.marginRight) + rect.width) + 'px';
  mask.style.height = (parseFloat(style.marginTop) +
    parseFloat(style.marginBottom) + rect.height) + 'px';

  elem.parentNode.appendChild(mask);
}

var target = document.getElementById('target');
addMask(target);

target.addEventListener('click', function () {
  console.log('click');
}, false);
</script>
```

请用代码写出(今天是星期 x)其中 x 表示当天是星期几,如果当天是星期一,输出应该是"今天是星期一"

```
var days = ['日', '一', '二', '三', '四', '五', '六'];
var date = new Date();

console.log('今天是星期' + days[date.getDay()]);
```

下面这段代码想要循环延时输出结果 0 1 2 3 4,请问输出结果是否正确,如果不正确,请说明为什么,并修改循环内的代码使其输出正确结果

```
for (var i = 0; i < 5; ++i) {
  setTimeout(function () {
    console.log(i + ' ');
  }, 100);
}
```

```
}
```

不能输出正确结果，因为循环中 `setTimeout` 接受的参数函数通过闭包访问变量 `i`。javascript 运行环境为单线程，`setTimeout` 注册的函数需要等待线程空闲才能执行，此时 `for` 循环已经结束，`i` 值为 5。五个定时输出都是 5。修改方法：将 `setTimeout` 放在函数立即调用表达式中，将 `i` 值作为参数传递给包裹函数，创建新闭包

```
for (var i = 0; i < 5; ++i) {  
  (function (i) {  
    setTimeout(function () {  
      console.log(i + ' ');  
    }, 100);  
  })(i);  
}
```

现有一个 `Page` 类,其原型对象上有许多以 `post` 开头的方法(如 `postMsg`);另有一拦截函数 `check`,只返回 `true` 或 `false`。请设计一个函数,该函数应批量改造原 `Page` 的 `postXXX` 方法,在保留其原有功能的同时,为每个 `postXXX` 方法增加拦截验证功能,当 `check` 返回 `true` 时继续执行原 `postXXX` 方法,返回 `false` 时不再执行原 `postXXX` 方法

```
function Page() {}  
  
Page.prototype = {  
  constructor: Page,  
  
  postA: function (a) {  
    console.log('a:' + a);  
  },  
  postB: function (b) {  
    console.log('b:' + b);  
  },  
  postC: function (c) {  
    console.log('c:' + c);  
  },  
  check: function () {  
    return Math.random() > 0.5;  
  }  
}  
  
function checkfy(obj) {  
  for (var key in obj) {  
    if (key.indexOf('post') === 0 && typeof obj[key] === 'function') {  
      (function (key) {  
        var fn = obj[key];  
        obj[key] = function () {  
          if (obj.check()) {  
            fn.apply(obj, arguments);  
          }  
        }  
      })(key);  
    }  
  }  
};
```

```
        }(key));
    }
}
} // end checkfy()

checkfy(Page.prototype);

var obj = new Page();

obj.postA('checkfy');
obj.postB('checkfy');
obj.postC('checkfy');
```

完成下面的 tool-tip

### 编写 javascript 深度克隆函数 deepClone

```
function deepClone(obj) {
    var _toString = Object.prototype.toString;

    // null, undefined, non-object, function
    if (!obj || typeof obj !== 'object') {
        return obj;
    }

    // DOM Node
    if (obj.nodeType && 'cloneNode' in obj) {
        return obj.cloneNode(true);
    }

    // Date
    if (_toString.call(obj) === '[object Date]') {
        return new Date(obj.getTime());
    }

    // RegExp
    if (_toString.call(obj) === '[object RegExp]') {
        var flags = [];
        if (obj.global) { flags.push('g'); }
        if (obj.multiline) { flags.push('m'); }
        if (obj.ignoreCase) { flags.push('i'); }

        return new RegExp(obj.source, flags.join(''));
    }

    var result = Array.isArray(obj) ? [] :
        obj.constructor ? new obj.constructor() : {};
}
```

```
    for (var key in obj ) {
        result[key] = deepClone(obj[key]);
    }

    return result;
}

function A() {
    this.a = a;
}

var a = {
    name: 'qiu',
    birth: new Date(),
    pattern: /qiu/gim,
    container: document.body,
    hobbys: ['book', new Date(), /aaa/gim, 111]
};

var c = new A();
var b = deepClone(c);
console.log(c.a === b.a);
console.log(c, b);
```

补充代码,鼠标单击 Button1 后将 Button1 移动到 Button2 的后面

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>TEst</title>
</head>
<body>

<div>
    <input type="button" id ="button1" value="1" />
    <input type="button" id ="button2" value="2" />
</div>

<script type="text/javascript">
    var btn1 = document.getElementById('button1');
    var btn2 = document.getElementById('button2');

    addListener(btn1, 'click', function (event) {
        btn1.parentNode.insertBefore(btn2, btn1);
    });

    function addListener(elem, type, handler) {
        if (elem.addEventListener) {
            elem.addEventListener(type, handler, false);
```

```
        return handler;
    } else if (elem.attachEvent) {
        function wrapper() {
            var event = window.event;
            event.target = event.srcElement;
            handler.call(elem, event);
        }
        elem.attachEvent('on' + type, wrapper);
        return wrapper;
    }
}
</script>
</body>
</html>
```

网页中实现一个计算当年还剩多少时间的倒数计时程序,要求网页上实时动态显示"××年还剩××天××时××分××秒"

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>TEst</title>
</head>
<body>

    <span id="target"></span>

<script type="text/javascript">
    // 为了简化。每月默认 30 天
    function getTimeString() {
        var start = new Date();
        var end = new Date(start.getFullYear() + 1, 0, 1);
        var elapse = Math.floor((end - start) / 1000);

        var seconds = elapse % 60 ;
        var minutes = Math.floor(elapse / 60) % 60;
        var hours = Math.floor(elapse / (60 * 60)) % 24;
        var days = Math.floor(elapse / (60 * 60 * 24)) % 30;
        var months = Math.floor(elapse / (60 * 60 * 24 * 30)) % 12;
        var years = Math.floor(elapse / (60 * 60 * 24 * 30 * 12));

        return start.getFullYear() + '年还剩' + years + '年' + months + '月'
            + days + '日'
            + hours + '小时' + minutes + '分' + seconds + '秒';
    }
}
```

```
function domText(elem, text) {
  if (text == undefined) {

    if (elem.textContent) {
      return elem.textContent;
    } else if (elem.innerText) {
      return elem.innerText;
    }
  } else {
    if (elem.textContent) {
      elem.textContent = text;
    } else if (elem.innerText) {
      elem.innerText = text;
    } else {
      elem.innerHTML = text;
    }
  }
}

var target = document.getElementById('target');

setInterval(function () {
  domText(target, getTimeString());
}, 1000)
</script>

</body>
</html>
```

完成一个函数,接受数组作为参数,数组元素为整数或者数组,数组元素包含整数或数组,函数返回扁平化后的数组

如：[1, [2, [ [3, 4], 5], 6]] => [1, 2, 3, 4, 5, 6]

```
var data = [1, [2, [ [3, 4], 5], 6]];

function flat(data, result) {
  var i, d, len;
  for (i = 0, len = data.length; i < len; ++i) {
    d = data[i];
    if (typeof d === 'number') {
      result.push(d);
    } else {
      flat(d, result);
    }
  }
}

var result = [];
flat(data, result);
```



```
console.log(result);
```

### 如何判断一个对象是否为数组

如果浏览器支持 `Array.isArray()` 可以直接判断否则需进行必要判断

```
/**
 * 判断一个对象是否是数组，参数不是对象或者不是数组，返回 false
 *
 * @param {Object} arg 需要测试是否为数组的对象
 * @return {Boolean} 传入参数是数组返回 true，否则返回 false
 */
function isArray(arg) {
    if (typeof arg === 'object') {
        return Object.prototype.toString.call(arg) === '[object Array]';
    }
    return false;
}
```

### 请评价以下事件监听器代码并给出改进意见

```
if (window.addEventListener) {
    var addListener = function (el, type, listener, useCapture) {
        el.addEventListener(type, listener, useCapture);
    };
}
else if (document.all) {
    addListener = function (el, type, listener) {
        el.attachEvent('on' + type, function () {
            listener.apply(el);
        });
    };
}
```

作用：浏览器功能检测实现跨浏览器 DOM 事件绑定

优点：

1. 测试代码只运行一次，根据浏览器确定绑定方法
2. 通过 `listener.apply(el)` 解决 IE 下监听器 `this` 与标准不一致的地方
3. 在浏览器不支持的情况下提供简单的功能，在标准浏览器中提供捕获功能

缺点：

1. `document.all` 作为 IE 检测不可靠，应该使用 `if(el.attachEvent)`

2. addListener 在不同浏览器下 API 不一样
3. listener.apply 使 this 与标准一致但监听器无法移除
4. 未解决 IE 下 listener 参数 event。target 问题

改进:

```
var addListener;

if (window.addEventListener) {
  addListener = function (el, type, listener, useCapture) {
    el.addEventListener(type, listener, useCapture);
    return listener;
  };
}
else if (window.attachEvent) {
  addListener = function (el, type, listener) {
    // 标准化 this, event, target
    var wrapper = function () {
      var event = window.event;
      event.target = event.srcElement;
      listener.call(el, event);
    };

    el.attachEvent('on' + type, wrapper);
    return wrapper;
    // 返回 wrapper。调用者可以保存，以后 remove
  };
}
```

### 如何判断一个对象是否为函数

```
/**
 * 判断对象是否为函数，如果当前运行环境对可调用对象（如正则表达式）
 * 的 typeof 返回 'function'，采用通用方法，否则采用优化方法
 *
 * @param {Any} arg 需要检测是否为函数的对象
 * @return {boolean} 如果参数是函数，返回 true，否则 false
 */
function isFunction(arg) {
  if (arg) {
    if (typeof (arg) !== 'function') {
      return typeof arg === 'function';
    } else {
      return Object.prototype.toString.call(arg) === '[object
Function]';
    }
  }
}
```

```
    } // end if  
    return false;  
}
```

编写一个函数接受 url 中 query string 为参数,返回解析后的 Object,query string 使用 application/x-www-form-urlencoded 编码

```
/**  
 * 解析 query string 转换为对象,一个 key 有多个值时生成数组  
 *  
 * @param {String} query 需要解析的 query 字符串,开头可以是?,  
 * 按照 application/x-www-form-urlencoded 编码  
 * @return {Object} 参数解析后的对象  
 */  
function parseQuery(query) {  
    var result = {};  
  
    // 如果不是字符串返回空对象  
    if (typeof query !== 'string') {  
        return result;  
    }  
  
    // 去掉字符串开头可能带的?  
    if (query.charAt(0) === '?') {  
        query = query.substring(1);  
    }  
  
    var pairs = query.split('&');  
    var pair;  
    var key, value;  
    var i, len;  
  
    for (i = 0, len = pairs.length; i < len; ++i) {  
        pair = pairs[i].split('=');  
        // application/x-www-form-urlencoded 编码会将' '转换为+  
        key = decodeURIComponent(pair[0]).replace(/\+/g, ' ');  
        value = decodeURIComponent(pair[1]).replace(/\+/g, ' ');  
  
        // 如果是新 key, 直接添加  
        if (!(key in result)) {  
            result[key] = value;  
        }  
        // 如果 key 已经出现一次以上, 直接向数组添加 value  
        else if (isArray(result[key])) {  
            result[key].push(value);  
        }  
    }  
}
```

```
        // key 第二次出现，将结果改为数组
        else {
            var arr = [result[key]];
            arr.push(value);
            result[key] = arr;
        } // end if-else
    } // end for

    return result;
}

function isArray(arg) {
    if (arg && typeof arg === 'object') {
        return Object.prototype.toString.call(arg) === '[object Array]';
    }
    return false;
}
/**
console.log(parseQuery('sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8'));
*/
```

### 解析一个完整的 url,返回 Object 包含域与 window.location 相同

```
/**
 * 解析一个 url 并生成 window.location 对象中包含的域
 * location:
 * {
 *     href: '包含完整的 url',
 *     origin: '包含协议到 pathname 之前的内容',
 *     protocol: 'url 使用的协议，包含末尾的:',
 *     username: '用户名', // 暂时不支持
 *     password: '密码', // 暂时不支持
 *     host: '完整主机名，包含:和端口',
 *     hostname: '主机名，不包含端口'
 *     port: '端口号',
 *     pathname: '服务器上访问资源的路径/开头',
 *     search: 'query string, ?开头',
 *     hash: '#开头的 fragment identifier'
 * }
 *
 * @param {string} url 需要解析的 url
 * @return {Object} 包含 url 信息的对象
 */
function parseUrl(url) {
```

```
var result = {};  
var keys = ['href', 'origin', 'protocol', 'host',  
            'hostname', 'port', 'pathname', 'search', 'hash'];  
var i, len;  
var regexp =  
/(((^[^:]+:)\:\/\/((([^\:\/\?#]+)(:\d+)?))(\.[^?#]*)?(\.[^#]*)?(\#.*?)?)/;  
  
var match = regexp.exec(url);  
  
if (match) {  
    for (i = keys.length - 1; i >= 0; --i) {  
        result[keys[i]] = match[i] ? match[i] : '';  
    }  
}  
  
return result;  
}
```

### 完成函数 getViewportSize 返回指定窗口的视口尺寸


```
/**  
 * 查询指定窗口的视口尺寸，如果不指定窗口，查询当前窗口尺寸  
 **/  
function getViewportSize(w) {  
    w = w || window;  
  
    // IE9 及标准浏览器中可使用此标准方法  
    if ('innerHeight' in w) {  
        return {  
            width: w.innerWidth,  
            height: w.innerHeight  
        };  
    }  
  
    var d = w.document;  
    // IE 8 及以下浏览器在标准模式下  
    if (document.compatMode === 'CSS1Compat') {  
        return {  
            width: d.documentElement.clientWidth,  
            height: d.documentElement.clientHeight  
        };  
    }  
  
    // IE8 及以下浏览器在怪异模式下  
    return {  
        width: d.body.clientWidth,  
        height: d.body.clientHeight  
    };  
}
```

**完成函数 getScrollOffset 返回窗口滚动条偏移量**

```
/**
 * 获取指定 window 中滚动条的偏移量，如未指定则获取当前 window
 * 滚动条偏移量
 *
 * @param {window} w 需要获取滚动条偏移量的窗口
 * @return {Object} obj.x 为水平滚动条偏移量,obj.y 为竖直滚动条偏移量
 */
function getScrollOffset(w) {
    w = w || window;
    // 如果是标准浏览器
    if (w.pageXOffset != null) {
        return {
            x: w.pageXOffset,
            y: w.pageYOffset
        };
    }

    // 老版本 IE，根据兼容性不同访问不同元素
    var d = w.document;
    if (d.compatMode === 'CSS1Compat') {
        return {
            x: d.documentElement.scrollLeft,
            y: d.documentElement.scrollTop
        };
    }

    return {
        x: d.body.scrollLeft,
        y: d.body.scrollTop
    };
}
```

现有一个字符串 richText,是一段富文本,需要显示在页面上.有个要求,需要给其中只包含一个 元素的 p 标签增加一个叫 pic 的 class.请编写代码实现.可以使用 jQuery 或 KISSY.

```
function richText(text) {
    var div = document.createElement('div');
    div.innerHTML = text;
    var p = div.getElementsByTagName('p');
    var i, len;

    for (i = 0, len = p.length; i < len; ++i) {
        if (p[i].getElementsByTagName('img').length === 1) {
            p[i].classList.add('pic');
        }
    }
}
```

```
    return div.innerHTML;
}
```

请实现一个 Event 类,继承自此类的对象都会拥有两个方法 on,off,once 和 trigger

```
function Event() {
    if (!(this instanceof Event)) {
        return new Event();
    }
    this._callbacks = {};
}
Event.prototype.on = function (type, handler) {
    this._callbacks = this._callbacks || {};
    this._callbacks[type] = this._callbacks[type] || [];
    this._callbacks[type].push(handler);

    return this;
};
Event.prototype.off = function (type, handler) {
    var list = this._callbacks[type];

    if (list) {
        for (var i = list.length; i >= 0; --i) {
            if (list[i] === handler) {
                list.splice(i, 1);
            }
        }
    }

    return this;
};
Event.prototype.trigger = function (type, data) {
    var list = this._callbacks[type];

    if (list) {
        for (var i = 0, len = list.length; i < len; ++i) {
            list[i].call(this, data);
        }
    }
};
Event.prototype.once = function (type, handler) {
    var self = this;

    function wrapper() {
        handler.apply(self, arguments);
        self.off(type, wrapper);
    }
}
```

```
    this.on(type, wrapper);  
    return this;  
};
```

### 编写一个函数将列表子元素顺序反转

```
<ul id="target">  
  <li>1</li>  
  <li>2</li>  
  <li>3</li>  
  <li>4</li>  
</ul>  
  
<script>  
  var target = document.getElementById('target');  
  var i;  
  var frag = document.createDocumentFragment();  
  
  for (i = target.children.length - 1; i >= 0; --i) {  
    frag.appendChild(target.children[i]);  
  }  
  target.appendChild(frag);  
</script>
```

### 以下函数的作用是?空白区域应该填写什么

```
// define  
(function (window) {  
  function fn(str) {  
    this.str = str;  
  }  
  
  fn.prototype.format = function () {  
    var arg = __1__;  
    return this.str.replace(__2__, function (a, b) {  
      return arg[b] || '';  
    });  
  };  
  
  window.fn = fn;  
})(window);  
  
// use  
(function () {  
  var t = new fn('<p><a href="{0}">{1}</a><span>{2}</span></p>');  
  console.log(t.format('http://www.alibaba.com', 'Alibaba', 'Welcome'));  
})();
```

define 部分定义一个简单的模板类，使用{}作为转义标记，中间的数字表示替换目标，format 实参用来替换模板内标记 横线处填：



1. `Array.prototype.slice.call(arguments, 0)`
2. `/\{s*(\d+)\s*\}/g`

编写一个函数实现 form 的序列化(即将一个表单中的键值序列化为可提交的字符串)

```
<form id="target">
  <select name="age">
    <option value="aaa">aaa</option>
    <option value="bbb" selected>bbb</option>
  </select>
  <select name="friends" multiple>
    <option value="qiu" selected>qiu</option>
    <option value="de">de</option>
    <option value="qing" selected>qing</option>
  </select>
  <input name="name" value="qiudeqing">
  <input type="password" name="password" value="11111">
  <input type="hidden" name="salary" value="3333">
  <textarea name="description">description</textarea>
  <input type="checkbox" name="hobby" checked value="football">Football
  <input type="checkbox" name="hobby" value="basketball">Basketball
  <input type="radio" name="sex" checked value="Female">Female
  <input type="radio" name="sex" value="Male">Male
</form>

<script>

/**
 * 将一个表单元素序列化为可提交的字符串
 *
 * @param {FormElement} form 需要序列化的表单元素
 * @return {string} 表单序列化后的字符串
 */
function serializeForm(form) {
  if (!form || form.nodeName.toUpperCase() !== 'FORM') {
    return;
  }

  var result = [];

  var i, len;
  var field, fieldName, fieldType;

  for (i = 0, len = form.length; i < len; ++i) {
    field = form.elements[i];
    fieldName = field.name;
    fieldType = field.type;

    if (field.disabled || !fieldName) {
```

```
        continue;
    } // enf if

    switch (fieldType) {
        case 'text':
        case 'password':
        case 'hidden':
        case 'textarea':
            result.push(encodeURIComponent(fieldName) + '=' +
                encodeURIComponent(field.value));
            break;

        case 'radio':
        case 'checkbox':
            if (field.checked) {
                result.push(encodeURIComponent(fieldName) + '=' +
                    encodeURIComponent(field.value));
            }
            break;

        case 'select-one':
        case 'select-multiple':
            for (var j = 0, jLen = field.options.length; j < jLen; ++j) {
                if (field.options[j].selected) {
                    result.push(encodeURIComponent(fieldName) + '=' +
                        encodeURIComponent(field.options[j].value ||
field.options[j].text));
                }
            } // end for
            break;

        case 'file':
        case 'submit':
            break; // 是否处理？

        default:
            break;
    } // end switch
} // end for

return result.join('&');
}

var form = document.getElementById('target');
console.log(serializeForm(form));
</script>
```

使用原生 javascript 给下面列表中的 li 节点绑定点击事件,点击时创建一个 Object 对象,兼容 IE 和标准浏览器

```
<ul id="nav">
  <li><a href="http://11111">111</a></li>
  <li><a href="http://2222">222</a></li>
  <li><a href="http://333">333</a></li>
  <li><a href="http://444">444</a></li>
</ul>
```

```
Object:
{
  "index": 1,
  "name": "111",
  "link": "http://1111"
}
```

script:

```
var EventUtil = {
  getEvent: function (event) {
    return event || window.event;
  },
  getTarget: function (event) {
    return event.target || event.srcElement;
  },
  // 返回注册成功的监听器, IE 中需要使用返回值来移除监听器
  on: function (elem, type, handler) {
    if (elem.addEventListener) {
      elem.addEventListener(type, handler, false);
      return handler;
    } else if (elem.attachEvent) {
      function wrapper(event) {
        return handler.call(elem, event);
      };
      elem.attachEvent('on' + type, wrapper);
      return wrapper;
    }
  },
  off: function (elem, type, handler) {
    if (elem.removeEventListener) {
      elem.removeEventListener(type, handler, false);
    } else if (elem.detachEvent) {
      elem.detachEvent('on' + type, handler);
    }
  },
  preventDefault: function (event) {
    if (event.preventDefault) {
      event.preventDefault();
    } else if ('returnValue' in event) {
      event.returnValue = false;
    }
  }
}
```

```
    },
    stopPropagation: function (event) {
        if (event.stopPropagation) {
            event.stopPropagation();
        } else if ('cancelBubble' in event) {
            event.cancelBubble = true;
        }
    }
};
var DOMUtil = {
    text: function (elem) {
        if ('textContent' in elem) {
            return elem.textContent;
        } else if ('innerText' in elem) {
            return elem.innerText;
        }
    },
    prop: function (elem, propName) {
        return elem.getAttribute(propName);
    }
};

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

EventUtil.on(nav, 'click', function (event) {
    var event = EventUtil.getEvent(event);
    var target = EventUtil.getTarget(event);

    var children = this.children;
    var i, len;
    var anchor;
    var obj = {};

    for (i = 0, len = children.length; i < len; ++i) {
        if (children[i] === target) {
            obj.index = i + 1;
            anchor = target.getElementsByTagName('a')[0];
            obj.name = DOMUtil.text(anchor);
            obj.link = DOMUtil.prop(anchor, 'href');
        }
    }

    alert('index: ' + obj.index + ' name: ' + obj.name +
        ' link: ' + obj.link);
});
```

有一个大数组, var a = ['1', '2', '3', ...]; a 的长度是 100, 内容填充随机整数的字符串. 请先构造此数组 a, 然后设计一个算法将其内容去重

```
/**
```

```
* 数组去重
**/
function normalize(arr) {
  if (arr && Array.isArray(arr)) {
    var i, len, map = {};
    for (i = arr.length; i >= 0; --i) {
      if (arr[i] in map) {
        arr.splice(i, 1);
      } else {
        map[arr[i]] = true;
      }
    }
  }
  return arr;
}

/**
 * 用 100 个随机整数对应的字符串填充数组。
 **/
function fillArray(arr, start, end) {
  start = start == undefined ? 1 : start;
  end = end == undefined ? 100 : end;

  if (end <= start) {
    end = start + 100;
  }

  var width = end - start;
  var i;
  for (i = 100; i >= 1; --i) {
    arr.push('' + (Math.floor(Math.random() * width) + start));
  }
  return arr;
}

var input = [];
fillArray(input, 1, 100);
input.sort(function (a, b) {
  return a - b;
});
console.log(input);

normalize(input);
console.log(input);
```

附：查看最新面试题的网址

<https://www.kuibuke.com/exam>

跬步客官网：[www.kuibuke.com](http://www.kuibuke.com)

跬步客客服：小冰

**注：加微信可免费获取最新面试资料**

公众号：跬步客

邮件：[kuibuke@qq.com](mailto:kuibuke@qq.com)



小冰 - 跬步客

加微信可免费获取最新面试资料



扫一扫上面的二维码图案，加我微信