HTML

1、Doctype作用？标准模式与兼容模式各有什么区别?

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

2.标准模式的排版 和JS运作模式都是以该浏览器支持的最高标准运行。在兼容模式中，页面以宽松的向后兼容的方式显示,模拟老式浏览器的行为以防止站点无法工作。

2、HTML5 为什么只需要写 `<!DOCTYPE HTML>`？

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

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

SGML (Standard Generalized Markup Language) 标准通用标记语言

3、行内元素有哪些？块级元素有哪些？ 空(void)元素有那些?

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

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

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

（3）常见的空元素：

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

鲜为人知的是：

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

不同浏览器（版本）、HTML4（5）、CSS2等实际略有差异

4、页面导入样式时，使用link和@import有什么区别?

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

2）页面被加载的时，link会同时被加载，而@import引用的CSS会等到页面被加载完再加载;

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

4. link支持使用js控制DOM去改变样式，而@import不支持;

5、介绍一下你对浏览器内核的理解?

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

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

JS引擎则：解析和执行javascript来实现网页的动态效果。最开始渲染引擎和JS引擎并没有区分的很明确，后来JS

引擎越来越独立，内核就倾向于只指渲染引擎。

6、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]

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

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

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

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

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, websocket, Geolocation;

移除的元素：

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

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

- 支持HTML5新标签：

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

  可以利用这一特性让这些浏览器支持HTML5新标签，

  浏览器支持新标签后，还需要添加标签默认的样式。

- 当然也可以直接使用成熟的框架、比如html5shim;

```
<!--[if lt IE 9]>
```

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

```
<![endif]-->
```

- 如何区分HTML5： DOCTYPE声明\新增的结构元素\功能元素

8、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。
```

9、请描述一下 cookies，sessionStorage 和 localStorage 的区别?

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

```
cookie数据始终在同源的http请求中携带（即使不需要），记会在浏览器和服务器间来回传递。
sessionStorage和localStorage不会自动把数据发给服务器，仅在本地保存。
```

存储大小：

cookie数据大小不能超过4k。

sessionStorage和localStorage 虽然也有存储大小的限制，但比cookie大得多，可以达到5M或更大。

有期时间：

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

sessionStorage 数据在当前浏览器窗口关闭后自动删除。

cookie 设置的cookie过期时间之前一直有效，即使窗口或浏览器关闭。

10、iframe有那些缺点?

*iframe会阻塞主页面的Onload事件；

*搜索引擎的检索程序无法解读这种页面，不利于SEO;

*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，这样可以绕开以上两个问题。

11、Label的作用是什么？是怎么用的?

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

```
<label for="Name">Number:</label>
<input type= “text “name="Name" id="Name"/>
<label>Date:<input type="text" name="B"/></label>
```

12、HTML5的form如何关闭自动完成功能?

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

13、如何实现浏览器内多个标签页之间的通信?(阿里)

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；

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

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

15、网页验证码是干嘛的，是为了解决什么安全问题。

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

有效防止黑客对某一个特定注册用户用特定程序暴力破解方式进行不断的登陆尝试。

16、title与h1的区别、b与strong的区别、i与em的区别?

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

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

i内容展示为斜体，em表示强调的文本；

Physical Style Elements -- 自然样式标签

b, i, u, s, pre

Semantic Style Elements -- 语义样式标签

strong, em, ins, del, code

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

CSS 相关

17、介绍一下标准的CSS的盒子模型？低版本IE的盒子模型有什么不同的?

1）有两种， IE 盒子模型、W3C 盒子模型；

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

3）区 别： IE的content部分把 border 和 padding计算了进去;

18、CSS选择符有哪些？哪些属性可以继承?

- **1.id选择器（ # myid）**

  2.类选择器（.myclassname）

  3.标签选择器（div, h1, p）

  4.相邻选择器（h1 + p）

  5.子选择器（ul > li）

  6.后代选择器（li a）

  7.通配符选择器（ * ）

  8.属性选择器（a[rel = "external"]）

  9.伪类选择器（a:hover, li:nth-child）

- 可继承的样式： **font-size font-family color, UL LI DL DD DT;**
- 不可继承的样式：**border padding margin width height** ;

19、CSS优先级算法如何计算?

- 优先级就近原则，同权重情况下样式定义最近者为准;
- 载入样式以最后载入的定位为准;

优先级为:

同权重: 内联样式表（标签内部）> 嵌入样式表（当前文件中）> 外部样式表（外部文件中）。

!important > id > class > tag

important 比 内联优先级高

20、CSS3新增伪类有那些?

举例:

p:first-of-type 选择属于其父元素的首个

元素的每个

元素。

p:last-of-type 选择属于其父元素的最后

元素的每个

元素。

p:only-of-type 选择属于其父元素唯一的

元素的每个

元素。

p:only-child 选择属于其父元素的唯一子元素的每个

元素。

p:nth-child(2) 选择属于其父元素的第二个子元素的每个

元素。

::after 在元素之前添加内容,也可以用来做清除浮动。

::before 在元素之后添加内容

:enabled

:disabled 控制表单控件的禁用状态。

:checked 单选框或复选框被选中。

21、如何居中div?

水平居中：给div设置一个宽度，然后添加margin:0 auto属性

```
 div{
    width:200px;
    margin:0 auto;
 }
```

让绝对定位的div居中

```
div {
    position: absolute;
    width: 300px;
    height: 300px;
    margin: auto;
    top: 0;
    left: 0;
    bottom: 0;
    right: 0;
    background-color: pink; /* 方便看效果 */
 }
```

水平垂直居中一（确定容器的宽高 宽500 高 300 的层 设置层的外边距）

```
div {
    position: relative;     /* 相对定位或绝对定位均可 */
    width:500px;
    height:300px;
    top: 50%;
    left: 50%;
    margin: -150px 0 0 -250px;      /* 外边距为自身宽高的一半 */
    background-color: pink;     /* 方便看效果 */
}
```

水平垂直居中二(未知容器的宽高，利用 `transform` 属性)

```
div {
    position: absolute;     /* 相对定位或绝对定位均可 */
    width:500px;
    height:300px;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    background-color: pink;     /* 方便看效果 */
}
```

水平垂直居中三(利用 flex 布局)

```
.container {
    display: flex;
    align-items: center;        /* 垂直居中 */
    justify-content: center;    /* 水平居中 */

 }
 .container div {
    width: 100px;
    height: 100px;
    background-color: pink;     /* 方便看效果 */
 }
```

22、display有哪些值？说明他们的作用。

```
block         块类型。默认宽度为父元素宽度，可设置宽高，换行显示。
none          缺省值。象行内元素类型一样显示。
inline        行内元素类型。默认宽度为内容宽度，不可设置宽高，同行显示。
inline-block  默认宽度为内容宽度，可以设置宽高，同行显示。
list-item     象块类型元素一样显示，并添加样式列表标记。
table         此元素会作为块级表格来显示。
inherit       规定应该从父元素继承 display 属性的值。
```

23、position的值relative和absolute定位原点是?

absolute：生成绝对定位的元素，相对于值不为 static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（忽略 top，bottom，left，right z-index 声明）。

inherit：规定从父元素继承 position 属性的值。

24、CSS3有哪些新特性?

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

圆角 （border-radius:8px）

多列布局 （multi-column layout）

阴影和反射 （Shadow\Reflect）

文字特效 （text-shadow、）

文字渲染 （Text-decoration）

线性渐变 （gradient）

旋转 （transform）

缩放,定位,倾斜,动画,多背景

例如:transform:\scale(0.85,0.90)\ translate(0px,-30px)\ skew(-9deg,0deg)\Animation:

25、请解释一下CSS3的Flexbox（弹性盒布局模型）,以及适用场景?

一个用于页面布局的全新CSS3功能，Flexbox可以把列表放在同一个方向（从上到下排列，从左到右），并让列表能延伸到占用可用的空间。

较为复杂的布局还可以通过嵌套一个伸缩容器（flex container）来实现。

采用Flex布局的元素，称为Flex容器（flex container），简称"容器"。

它的所有子元素自动成为容器成员，称为Flex项目（flex item），简称"项目"。

常规布局是基于块和内联流方向，而Flex布局是基于flex-flow流可以很方便的用来做局中，能对不同屏幕大小自适应。

在布局上有了比以前更加灵活的空间。

26、用纯CSS创建一个三角形的原理是什么?

把上、左、右三条边隐藏掉（颜色设为 transparent）

```
demo {
  width: 0;
  height: 0;
  border-width: 20px;
  border-style: solid;
  border-color: transparent transparent red transparent;
}
```

27、一个满屏 品 字布局 如何设计?

简单的方式：

上面的div宽100%,

下面的两个div分别宽50%,

然后用float或者inline使其不换行即可

28、CSS多列等高如何实现?

利用padding-bottom|margin-bottom正负值相抵；

设置父容器设置超出隐藏（overflow:hidden），这样子父容器的高度就还是它里面的列没有设定padding-bottom时的高度，

当它里面的任 一列高度增加了，则父容器的高度被撑到里面最高那列的高度，

其他比这列矮的列会用它们的padding-bottom补偿这部分高度差。

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

- png24位的图片在iE6浏览器上出现背景，解决方案是做成PNG8.
    - 浏览器默认的margin和padding不同。解决方案是加一个全局的*{margin:0;padding:0;}来统一。
    - IE6双边距bug:块属性标签float后，又有横行的margin情况下，在ie6显示margin比设置的大。

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

      这种情况之下IE会产生20px的距离，解决方案是在float的标签样式控制中加入 ——_display:inline;将其转化为行内属性。(_这个符号只有ie6会识别)

      渐进识别的方式，从总体中逐渐排除局部。

      首先，巧妙的使用“\9”这一标记，将IE游览器从所有情况中分离出来。

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

eg：

```
.bb{
  background-color:red;/*所有识别*/
  background-color:#00deff\9; /*IE6、7、8识别*/
  +background-color:#a200ff;/*IE6、7识别*/
  _background-color:#1e0bd1;/*IE6识别*/
}
```

- **IE下,可以使用获取常规属性的方法来获取自定义属性,**

  也可以使用getAttribute()获取自定义属性;

  Firefox下,只能使用getAttribute()获取自定义属性。

  解决方法:统一通过getAttribute()获取自定义属性。

  - **IE下,even对象有x,y属性,但是没有pageX,pageY属性;**

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

  - 解决方法：（条件注释）缺点是在IE浏览器下可能会增加额外的HTTP请求数。

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

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

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

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

30、li与li之间有看不见的空白间隔是什么原因引起的？有什么解决办法?

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

31、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： *** {padding: 0; margin: 0;}** （强烈不建议）

淘宝的样式初始化代码：

```
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; }
```

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

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

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

2、否则,则由这个祖先元素的 **padding box** 构成。

如果都找不到，则为 initial containing block。

补充：

```
1. static(默认的)/relative：简单说就是它的父元素的内容框（即去掉padding的部分）
2. absolute：向上找最近的定位为absolute/relative的元素
3. fixed：它的containing block一律为根元素(html/body)，根元素也是initial containing block
```

33、CSS里的visibility属性有个collapse属性值是干嘛用的？在不同浏览器下以后什么区别？

对于普通元素visibility:collapse;会将元素完全隐藏,不占据页面布局空间,与display:none;表现相同. 如果目标元素为table,visibility:collapse;将table隐藏,但是会占据页面布局空间. 仅在Firefox下起作用,IE会显示元素,Chrome会将元素隐藏,但是占据空间.

34、position跟display、margin collapse、overflow、float这些特性相互叠加后会怎么样？

如果元素的display为none,那么元素不被渲染,position,float不起作用,如果元素拥有position:absolute;或者position:fixed;属性那么元素将为绝对定位,float不起作用.如果元素float属性不是none,元素会脱离文档流,根据float属性值来显示.有浮动,绝对定位,inline-block属性的元素,margin不会和垂直方向上的其他元素margin折叠.

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

（W3C CSS 2.1 规范中的一个概念,它是一个独立容器，决定了元素如何对其内容进行定位,以及与其他元素的关系和相互作用。） 一个页面是由很多个 Box

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

36、CSS定义的权重

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

```
/*权重为1*/
div{
}
/*权重为10*/
.class1{
}
/*权重为100*/
#id1{
}
/*权重为100+1=101*/
#id1 div{
}
/*权重为10+1=11*/
.class1 div{
}
/*权重为10+10+1=21*/
.class1 .class2 div{
}
```

如果权重相同，则最后定义的样式会起作用，但是应该避免这种情况出现

37、请解释一下为什么需要清除浮动？清除浮动的方式

清除浮动是为了清除使用浮动元素产生的影响。浮动的元素，高度会塌陷，而高度的塌陷使我们页面后面的布局不能正常显示。

1、父级div定义height；

2、父级div 也一起浮动；

3、常规的使用一个class；

```
.clearfix::before, .clearfix::after {
  content: " ";
  display: table;
}
.clearfix::after {
  clear: both;
}
.clearfix {
  *zoom: 1;
}
```

4、SASS编译的时候，浮动元素的父级div定义伪类:after

```
&::after,&::before{
  content: " ";
  visibility: hidden;
  display: block;
  height: 0;
  clear: both;
}
```

解析原理：

i. display:block 使生成的元素以块级元素显示,占满剩余空间;

ii. height:0 避免生成内容破坏原有布局的高度。

iii. visibility:hidden 使生成的内容不可见，并允许可能被生成内容盖住的内容可以进行点击和交互;

4）通过 content:"."生成内容作为最后一个元素，至于content里面是点还是其他都是可以的，例如oocss里面就有经典的 content:".",有些版本可能 content 里面内容为空,一丝冰凉是不推荐这样做的,firefox直到7.0 content:”" 仍然会产生额外的空隙；

5）zoom：1 触发IE hasLayout。

通过分析发现，除了clear：both用来闭合浮动的，其他代码无非都是为了隐藏

掉content生成的内容，这也就是其他版本的闭合浮动为什么会有font-size：0，line-height：0。

38、什么是外边距合并

外边距合并指的是，当两个垂直外边距相遇时，它们将形成一个外边距。

合并后的外边距的高度等于两个发生合并的外边距的高度中的较大者。

39、zoom:1的清除浮动原理?

清除浮动，触发hasLayout；

Zoom属性是IE浏览器的专有属性，它可以设置或检索对象的缩放比例。解决ie下比较奇葩的bug。

譬如外边距（margin）的重叠，浮动清除，触发ie的haslayout属性等。

来龙去脉大概如下：

当设置了zoom的值之后，所设置的元素就会就会扩大或者缩小，高度宽度就会重新计算了，这里一旦改变zoom值时其实也会发生重新渲染，运用这个原理，也就解决了ie下子元素浮动时候父元素不随着自动扩大的问题。

Zoom属是IE浏览器的专有属性，火狐和老版本的webkit核心的浏览器都不支持这个属性。然而，zoom现在已经被逐步标准化，出现在 CSS 3.0 规范草案中。

目前非ie由于不支持这个属性，它们又是通过什么属性来实现元素的缩放呢?

可以通过css3里面的动画属性scale进行缩放。

40、移动端的布局用过媒体查询吗?

假设你现在正用一台显示设备来阅读这篇文章，同时你也想把它投影到屏幕上，或者打印出来， 而显示设备、屏幕投影和打印等这些媒介都有自己的特点，CSS就是为文档提供在不同媒介上展示的适配方法

当媒体查询为真时，相关的样式表或样式规则会按照正常的级联规被应用。 当

媒体查询返回假， 标签上带有媒体查询的样式表 仍将被下载 （只不过不会被应用）。

包含了一个媒体类型和至少一个使用 宽度、高度和颜色等媒体属性来限制样式表范围的表达式。 CSS3加入的媒体查询使得无需修改内容便可以使样式应用于某些特定的设备范围。

41、使用 CSS 预处理器吗？喜欢那个？

SASS（SASS、LESS没有本质区别，只因为团队前端都是用的SASS）

Less更趋向于js式的处理，所以说less处理逻辑性高的可能更好一点

42、CSS优化、提高性能的方法有哪些？

关键选择器（key selector）。选择器的最后面的部分为关键选择器（即用来匹配目标元素的部分）；

如果规则拥有 ID 选择器作为其关键选择器，则不要为规则增加标签。过滤掉无关的规则（这样样式系统就不会浪费时间去匹配它们了）；

提取项目的通用公有样式，增强可复用性，按模块编写组件；增强项目的协同开发性、可维护性和可扩展性;

使用预处理工具或构建工具（gulp对css进行语法检查、自动补前缀、打包压缩、自动优雅降级）；

43、浏览器是怎样解析CSS选择器的？

样式系统从关键选择器开始匹配，然后左移查找规则选择器的祖先元素。

只要选择器的子树一直在工作，样式系统就会持续左移，直到和规则匹配，或者是因为不匹配而放弃该规则。

44、在网页中的应该使用奇数还是偶数的字体？为什么呢？

为何偶数居多?

1.比例关系

相对来说偶数字号比较容易和页面中其他部分的字号构成一个比例关系。如我使用14px的字体作为正文字号，那么其他部分的字体（如标题）就可以使用14×1.5 =21px的字体，或者在一些地方使用到了14×0.5=7px的padding或者margin，如果你是在用sass或者less编写css，这时候用处就凸显出来了。

2.UI设计师的缘故

大多数设计师用的软件如ps提供的字号是偶数，自然到了 前端那边也是用的是偶数。

3.浏览器缘故

其一是低版本的浏览器ie6会把奇数字体强制转化为偶数，即13px渲染为14px。

其二是为了平分字体。偶数宽的汉字，如12px的汉子，去掉1像素的字体间距，填充了的字体像素宽度其实就是11px，这样的汉字中竖线左右是平分的，如“中”子，左右就是5px了。

4.系统差别

Windows 自带的点阵宋体（中易宋体）从 Vista 开始只提供 12、14、16 px 这三个大小的点阵，而 13、15、17 px 时用的是小一号的点阵（即每个字占的空间大了 1 px，但点阵没变），于是略显稀疏。

而在Linux和其他手持设备上，奇数偶数的渲染效果其实相差不大。

也可以使用奇数！！！

目前来说12，13，14，15，16px都是比较常用而且不错的字号，通过审查元素可以看到知乎的正文字号和豆瓣部分栏目（豆瓣电影）也是用了13px字号，效果都很不错。使用奇数号字体不好的地方是，文本段落无法对齐。

说到这里，有人会想起11px这个字号。那么关于这个这号，在谷歌中默认最小是12px字体，如果设置11px字体会渲染为12px字号，有人说通过设置css属性-webkit-text-size-adjust: none;来解决。本人实测失败，原因如下：

1.只对chrome27.0 版本以下有效，27.0以上版本无效；

2.只对英文才有效，对中文无效。

在新版的chrome中，已经禁止了改属性,建议使用CSS3中的方法:

transform:scale(0.7);

当使用transform:scale(0.7)时; 不仅是文字变小了，整个文字所在的容器也同时会变小，那么这时候可能就涉及到布局问题了，需要重新调整位置。

关于浏览器的支持的最小字体：

1.iphone6-plus、iphone5： 微信、QQ直接打开、safari中字体可以从最1px字体开始；

2.小米2s：微信、QQ浏览器最小显示字体8px；自带浏览器最小字体从1px开始；

3.小米4s：firefox可以从最小字体1px开始；chrome中最小显示字体为12px；

4.pc： 360安全浏览器7：最小最示12px；firefox与ie10最小显示字体为1px。

45、margin和padding分别适合什么场景使用?

margin是用来隔开元素与元素的间距；padding是用来隔开元素与内容的间隔。

margin用于布局分开元素使元素与元素互不相干；

padding用于元素与内容之间的间隔，让内容（文字）与（包裹）元素之间有一段；

46、什么是响应式设计？响应式设计的基本原理是什么？如何兼容低版本的IE?

47、::before 和 :after中双冒号和单冒号 有什

么区别？解释一下这2个伪元素的作用。

::before 和 :after中双冒号和单冒号 有什么区别？解释一下这2个伪元素的作用。

单冒号(:)用于CSS3伪类，双冒号(::)用于CSS3伪元素。（伪元素由双冒号和伪元素名称组成）

双冒号是在当前规范中引入的，用于区分伪类和伪元素。不过浏览器需要同时支持旧的已经存在的伪元素写法，

比如:first-line、:first-letter、:before、:after等，

而新的在CSS3中引入的伪元素则不允许再支持旧的单冒号的写法。

想让插入的内容出现在其它内容前，使用::before，否者，使用::after；

在代码顺序上，::after生成的内容也比::before生成的内容靠后。

如果按堆栈视角，::after生成的内容会在::before生成的内容之上

48、如何修改chrome记住密码后自动填充表单的黄色背景？

```
input:-webkit-autofill, textarea:-webkit-autofill, select:-webkit-
autofill {
  background-color: rgb(250, 255, 189); /* #FAFFBD; */
  background-image: none;
  color: rgb(0, 0, 0);
}
```

49、让页面里的字体变清晰，变细用CSS怎么做？

```
-webkit-font-smoothing: antialiased;
```

50、font-style属性可以让它赋值为“oblique” oblique是什么意思?

倾斜的字体样式

51、position:fixed;在android下无效怎么处理?

fixed的元素是相对整个页面固定位置的，你在屏幕上滑动只是在移动这个所谓的**viewport**，

原来的网页还好好的在那，**fixed**的内容也没有变过位置，

所以说并不是**iOS**不支持**fixed**，只是**fixed**的元素不是相对手机屏幕固定的。

52、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**

53、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**

54、overflow: scroll时不能平滑滚动的问题怎么处理?

```
1、阻止所有能导致页面滚动的事件。//scroll不能阻止，只能阻止mousewheel，鼠标拽滚动条就悲剧了;
2、bodyoverflow:hidden//win下右侧滚动条会消失导致页面横移，移动端阻止不了;
3、把滚动部分单独放在一个div里，和弹出部分同级，body和window同高。//所有涉及offset/scrollTop的方法都要修改。fix并且width100%的元素(比如微博顶栏)会压在内容区滚动条上;
4、弹出时算scrollTop，给内容区fix然后top移动到目前位置，同时body给一个overflow-y:scroll强撑出滚动条
```

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

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

因为cookie有域的限制，因此不能跨域提交请求，故使用非主要域名的时候，请求头中就不会带有cookie数据，这样可以降低请求头的大小，降低请求时间，从而达到降低整体请求延时的目的。同时这种方式不会将cookie传入Web Server，也减少了Web Server对cookie的处理分析环节，提高了webserver的http请求的解析速度。

56、什么是CSS 预处理器 / 后处理器?

- 预处理器例如：LESS、Sass、Stylus，用来预编译Sass或less，增强了css代码的复用性，

  还有层级、mixin、变量、循环、函数等，具有很方便的UI组件模块化开发能力，极大的提高工作效率。

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

JavaScript

57、介绍js的基本数据类型。

Undefined、Null、Boolean、Number、String、

ECMAScript 2015 新增:Symbol(创建后独一无二且不可变的数据类型)

58、介绍js有哪些内置对象?

Object 是 JavaScript 中所有对象的父对象

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

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

59、说几条写JavaScript的基本规范?

```
1.不要在同一行声明多个变量。
2.请使用 ===/!==来比较true/false或者数值
3.使用对象字面量替代new Array这种形式
4.不要使用全局函数。
5.Switch语句必须带有default分支
6.函数不应该有时候有返回值，有时候没有返回值。
7.For循环必须使用大括号
8.If语句必须使用大括号
9.for-in循环中的变量 应该使用var关键字明确限定作用域，从而避免作用域污染。
```

60、JavaScript原型，原型链？有什么特点?

每个对象都会在其内部初始化一个属性，就是prototype(原型)，当我们访问一个对象的属性时，

如果这个对象内部不存在这个属性，那么他就会去prototype里找这个属性，这个prototype又会有自己的prototype，

于是就这样一直找下去，也就是我们平时所说的原型链的概念。

关系：**instance.constructor.prototype = instance.proto**

特点：

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

当我们需要一个属性的时，Javascript引擎会先看当前对象中是否有这个属性， 如果没有的话，

就会查找他的Prototype对象是否有这个属性，如此递推下去，一直检索到 Object 内建对象。

```
function Func(){ }
Func.prototype.name = "Sean";
Func.prototype.getInfo = function() {
  return this.name;
}
var person = new Func();//现在可以参考var person =
Object.create(oldObject);
console.log(person.getInfo());//它拥有了Func的属性和方法
//"Sean"
console.log(Func.prototype);
// Func { name="Sean", getInfo=function()}
```

61、JavaScript有几种类型的值？，你能画一下他们的内存图吗?

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

堆：引用数据类型（对象、数组和函数）

两种类型的区别是：存储位置不同；

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

引用数据类型存储在堆(heap)中的对象,占据空间大、大小不固定。如果存储在栈中，将会影响程序运行的性能；引用数据类型在栈中存储了指针，该指针指

向堆中该实体的起始地址。当解释器寻找引用值时，会首先检索其在栈中的地址，取得地址后从堆中获得实体

62、如何将字符串转化为数字，例如'12.3b'?

- parseFloat('12.3b');
- 正则表达式，'12.3b'.match(/(\d)+(.)?(\d)+/g)[0] * 1, 但是这个不太靠谱，提供一种思路而已。

63、如何将浮点数点左边的数每三位添加一个逗号，如12000000.11转化为『12,000,000.11』？

```
function commafy(num){
  return num && num
    .toString()
    .replace(/(\d)(?=(\d{3})+\.)/g, function($1, $2){
    return $2 + ',';
  });
}
```

64、如何实现数组的随机排序?

方法一：

```
var arr = [1,2,3,4,5,6,7,8,9,10];
function randSort1(arr){
  for(var i = 0,len = arr.length;i < len; i++ ){
    var rand = parseInt(Math.random()*len);
    var temp = arr[rand];
    arr[rand] = arr[i];
    arr[i] = temp;
  }
  return arr;
}
console.log(randSort1(arr));
```

方法二：

```
var arr = [1,2,3,4,5,6,7,8,9,10];
function randSort2(arr){
  var mixedArray = [];
  while(arr.length > 0){
    var randomIndex = parseInt(Math.random()*arr.length);
    mixedArray.push(arr[randomIndex]);
    arr.splice(randomIndex, 1);
  }
  return mixedArray;
}
console.log(randSort2(arr));
```

方法三：

```
var arr = [1,2,3,4,5,6,7,8,9,10];
arr.sort(function(){
  return Math.random() - 0.5;
})
console.log(arr);
```

65、JavaScript如何实现继承?

1、构造继承

2、原型继承

3、实例继承

4、拷贝继承

原型prototype机制或apply和call方法去实现较简单，建议使用构造函数与原型混合方式。

```
function Parent(){
  this.name = 'wang';
}

function Child(){
  this.age = 28;
}
Child.prototype = new Parent();//继承了Parent，通过原型

var demo = new Child();
alert(demo.age);
alert(demo.name);//得到被继承的属性
```

66、JavaScript创建对象的几种方式?

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

1、对象字面量的方式

```
var person={firstname:"Mark",lastname:"Yun",age:25,eyecolor:"black"};
```

2、用**function**来模拟无参的构造函数

```
function Person(){}
var person=new Person();//定义一个function，如果使用new"实例化",该function可以看作是一个Class
person.name="Mark";
person.age="25";
person.work=function(){
  alert(person.name+" hello...");
}
person.work();
```

3、用**function**来模拟参构造函数来实现（用**this**关键字定义构造的上下文属性）

```
function Pet(name,age,hobby){
  this.name=name;//this作用域：当前对象
  this.age=age;
  this.hobby=hobby;
  this.eat=function(){
    alert("我叫"+this.name+",我喜欢"+this.hobby+",是个程序员");
  }
}
var maidou =new Pet("麦兜",25,"coding");//实例化、创建对象
maidou.eat();//调用eat方法
```

4、用工厂方式来创建（内置对象）

```
var wcDog =new Object();
wcDog.name="旺财";
wcDog.age=3;
wcDog.work=function(){
  alert("我是"+wcDog.name+",汪汪汪......");
}
wcDog.work();
```

5、用原型方式来创建

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

6、用混合方式来创建

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

67、Javascript作用链域?

全局函数无法查看局部函数的内部细节，但局部函数可以查看其上层的函数细节，直至全局细节。

当需要从局部函数查找某一属性或方法时，如果当前作用域没有找到，就会上溯到上层作用域查找，

直至全局函数，这种组织形式就是作用域链。

68、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；

ES6 箭头函数中的This指向（自己百度）

69、eval是做什么的?

它的功能是把对应的字符串解析成JS代码并运行；

应该避免使用eval，不安全，非常耗性能（2次，一次解析成js语句，一次执行）。

由JSON字符串转换为JSON对象的时候可以用eval，var obj =eval('('+ str +')');

70、什么是window对象? 什么是document对象?

window对象是指浏览器打开的窗口。

document对象是Documentd对象（HTML 文档对象）的一个只读引用，window对象的一个属性。（Document Object Model，简称DOM）

71、null，undefined 的区别?

null 表示一个对象是“没有值”的值，也就是值为“空”；

undefined 表示一个变量声明了没有初始化(赋值)；

undefined不是一个有效的JSON，而null是；

undefined的类型(typeof)是undefined；

null的类型(typeof)是object；

Javascript将未赋值的变量默认值设为undefined；

Javascript从来不会将变量设为null。它是用来让程序员表明某个用var声明的变量时没有值的。

```
typeof undefined
 //"undefined"
```

undefined :是一个表示"无"的原始值或者说表示"缺少值"，就是此处应该有一个值，但是还没有定义。当尝试读取时会返回 undefined；

例如变量被声明了，但没有赋值时，就等于undefined

```
typeof null
 //"object"
```

null ： 是一个对象(空对象，没有任何属性和方法)；

例如作为函数的参数，表示该函数的参数不是对象；

注意：

在验证null时，一定要使用 === ，因为 == 无法分别 null 和 undefined

```
null == undefined // true
null === undefined // false
```

再来一个例子：

```
 null

Q：有张三这个人么？

A：有！

Q：张三有房子么？

A：没有！

undefined

Q：有张三这个人么？

A：有！

Q: 张三有多少岁？

A: 不知道（没有被告诉）
```

72、写一个通用的事件侦听器函数。

```
// event(事件)工具集，来源：github.com/markyun
markyun.Event = {
  // 页面加载完成后
  readyEvent : function(fn) {
    if (fn==null) {
      fn=document;
    }
    var oldonload = window.onload;
    if (typeof window.onload != 'function') {
      window.onload = fn;
    } else {
      window.onload = function() {
        oldonload();
        fn();
      };
    }
  },
  // 视能力分别使用dom0||dom2||IE方式 来绑定事件
  // 参数： 操作的元素,事件名称 ,事件处理程序
  addEvent : function(element, type, handler) {
```

```
    if (element.addEventListener) {
      //事件类型、需要执行的函数、是否捕捉
      element.addEventListener(type, handler, false);
    } else if (element.attachEvent) {
      element.attachEvent('on' + type, function() {
        handler.call(element);
      });
    } else {
      element['on' + type] = handler;
    }
  },
  // 移除事件
  removeEvent : function(element, type, handler) {
    if (element.removeEventListener) {
      element.removeEventListener(type, handler, false);
    } else if (element.datachEvent) {
      element.detachEvent('on' + type, handler);
    } else {
      element['on' + type] = null;
    }
  },
  // 阻止事件（主要是事件冒泡，因为IE不支持事件捕获）
  stopPropagation : function(ev) {
    if (ev.stopPropagation) {
      ev.stopPropagation();
    } else {
      ev.cancelBubble = true;
    }
  },
  // 取消事件的默认行为
  preventDefault : function(event) {
    if (event.preventDefault) {
      event.preventDefault();
    } else {
      event.returnValue = false;
    }
  },
  // 获取事件目标
  getTarget : function(event) {
    return event.target || event.srcElement;
  },
  // 获取event对象的引用，取到事件的所有信息，确保随时能使用event；
  getEvent : function(e) {
    var ev = e || window.event;
    if (!ev) {
      var c = this.getEvent.caller;
      while (c) {
```

```
        ev = c.arguments[0];
        if (ev && Event == ev.constructor) {
          break;
        }
        c = c.caller;
      }
    }
    return ev;
  }
};
```

73、["1", "2", "3"].map(parseInt) 答案是多少?

parseInt() 函数能解析一个字符串，并返回一个整数，需要两个参数 (val, radix),

其中 radix 表示要解析的数字的基数。【该值介于 2 ~ 36 之间，并且字符串中的数字不能大于radix才能正确返回数字结果值】；

但此处 map 传了 3 个 (element, index, array),我们重写parseInt函数测试一下是否符合上面的规则。

```
function parseInt(str, radix) {
 return str+'-'+radix;
};
var a=["1", "2", "3"];
a.map(parseInt);  // ["1-0", "2-1", "3-2"] 不能大于radix
```

因为二进制里面，没有数字3,导致出现超范围的radix赋值和不合法的进制解析，才会返回NaN

所以["1", "2", "3"].map(parseInt) 答案也就是：[1, NaN, NaN]

74、事件是？IE与火狐的事件机制有什么区别？ 如何阻止冒泡?

i. 我们在网页中的某个操作（有的操作对应多个事件）。例如：当我们点击

一个按钮就会产生一个事件。是可以被 JavaScript 侦测到的行为。

ii. 事件处理机制：IE是事件冒泡、Firefox同时支持两种事件模型，也就是：捕获型事件和冒泡型事件；

iii. ev.stopPropagation();（旧ie的方法 ev.cancelBubble = true;）

75、什么是闭包（closure），为什么要用它?

闭包是指有权访问另一个函数作用域中变量的函数，创建闭包的最常见的方式就是在一个函数内创建另一个函数，通过另一个函数访问这个函数的局部变量，利用闭包可以突破作用链域，将函数内部的变量和方法传递到外部。

闭包的特性：

1.函数内再嵌套函数

2.内部函数可以引用外层的参数和变量

3.参数和变量不会被垃圾回收机制回收

```
//li节点的onclick事件都能正确的弹出当前被点击的li索引
  <ul id="testUL">
     <li> index = 0</li>
     <li> index = 1</li>
     <li> index = 2</li>
     <li> index = 3</li>
 </ul>
 <script type="text/javascript">
    var nodes = document.getElementsByTagName("li");
    for(i = 0;i<nodes.length;i+= 1){
        nodes[i].onclick = (function(i){
                  return function() {
                     console.log(i);
                  } //不用闭包的话，值每次都是4
                })(i);
    }
 </script>
```

执行say667()后,say667()闭包内部变量会存在,而闭包内部函数的内部变量不会存在

使得Javascript的垃圾回收机制GC不会收回say667()所占用的资源

因为say667()的内部函数的执行需要依赖say667()中的变量

这是对闭包作用的非常直白的描述

```
function say667() {
    // Local variable that ends up within closure
    var num = 666;
    var sayAlert = function() {
        alert(num);
    }
    num++;
    return sayAlert;
 }

  var sayAlert = say667();
  sayAlert()//执行结果应该弹出的667
```

76、javascript 代码中的"use strict";是什么意思？使用它区别是什么?

use strict是一种ECMAscript 5 添加的（严格）运行模式,这种模式使得 Javascript 在更严格的条件下运行,

使JS编码更加规范化的模式,消除Javascript语法的一些不合理、不严谨之处,减少一些怪异行为。

默认支持的糟糕特性都会被禁用，比如不能用with，也不能在意外的情况下给全局变量赋值;

全局变量的显示声明,函数必须声明在顶层，不允许在非函数代码块内声明函数,arguments.callee也不允许使用；

消除代码运行的一些不安全之处，保证代码运行的安全,限制函数中的arguments修改，严格模式下的eval函数的行为和非严格模式的也不相同;

提高编译器效率，增加运行速度；

为未来新版本的Javascript标准化做铺垫。

77、如何判断一个对象是否属于某个类?

使用instanceof （待完善）

```
if(a instanceof Person){
  alert('yes');
}
```

78、new操作符具体干了什么呢?

1、创建一个空对象，并且 this 变量引用该对象，同时还继承了该函数的原型。

2、属性和方法被加入到 this 引用的对象中。

3、新创建的对象由 this 所引用，并且最后隐式的返回 this 。

```
var obj  = {};
obj.__proto__ = Base.prototype;
Base.call(obj);
```

79、Javascript中，有一个函数，执行时对象查找时，永远不会去查找原型，这个函数是?

hasOwnProperty

javaScript中hasOwnProperty函数方法是返回一个布尔值，指出一个对象是否具有指定名称的属性。此方法无法检查该对象的原型链中是否具有该属性；该属性必须是对象本身的一个成员。

使用方法：

object.hasOwnProperty(proName)

其中参数object是必选项。一个对象的实例。

proName是必选项。一个属性名称的字符串值。

如果 object 具有指定名称的属性，那么JavaScript中hasOwnProperty函数方法返回 true，反之则返回 false。

80、JSON 的了解?

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

如：{"age":"12", "name":"back"}

```
 //JSON字符串转换为JSON对象:
var obj =eval('('+ str +')');
var obj = str.parseJSON();
var obj = JSON.parse(str);
//JSON对象转换为JSON字符串：
var last=obj.toJSONString();
var last=JSON.stringify(obj);
```

81、Ajax 是什么? 如何创建一个Ajax?

ajax的全称：Asynchronous Javascript And XML。

异步传输+js+xml。

所谓异步，在这里简单地解释就是：向服务器发送请求的时候，我们不必等待结果，而是可以同时做其他的事情，等到有了结果它自己会根据设定进行后续操作，与此同时，页面是不会发生整页刷新的，提高了用户体验。

(1)创建XMLHttpRequest对象,也就是创建一个异步调用对象

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

(3)设置响应HTTP请求状态变化的函数

(4)发送HTTP请求

(5)获取异步调用返回的数据

(6)使用JavaScript和DOM实现局部刷新

82、Ajax 解决浏览器缓存问题?

1、在ajax发送请求前加上 anyAjaxObj.setRequestHeader("If-Modified-Since","0")。

2、在ajax发送请求前加上 anyAjaxObj.setRequestHeader("Cache-Control","no-cache")。

3、在URL后面加上一个随机数： "fresh=" + Math.random();。

4、在URL后面加上时间戳："nowtime=" + new Date().getTime();。

5、如果是使用jQuery，直接这样就可以了 $.ajaxSetup({cache:false})。这样页面的所有ajax都会执行这条语句就是不需要保存缓存记录。

83、同步和异步的区别?

同步的概念应该是来自于OS中关于同步的概念:不同进程为协同完成某项工作而在先后次序上调整(通过阻塞,唤醒等方式).同步强调的是顺序性.谁先谁后.异步则不存在这种顺序性.

同步：浏览器访问服务器请求，用户看得到页面刷新，重新发请求,等请求完，页面刷新，新内容出现，用户看到新内容,进行下一步操作。

异步：浏览器访问服务器请求，用户正常操作，浏览器后端进行请求。等请求完，页面不刷新，新内容也会出现，用户看到新内容。

主任务与任务队列 js并没有真正的异步 只是调整执行时间，具体可研究libuv工作原理

84、如何解决跨域问题?

85、页面编码和被请求的资源编码如果不一致如何处理?

```
<script src="http://www.xxx.com/test.js" charset="utf-8"></script>
```

86、服务器代理转发时，该如何处理cookie?

Nginx

87、模块化开发怎么做?

立即执行函数,不暴露私有成员

```
var module1 = (function(){
  var _count = 0;
  var m1 = function(){
    //...
  };
  var m2 = function(){
    //...
  };
  return {
    m1 : m1,
    m2 : m2
  };
})();
```

还可以结合现有的三大框架

AMD（Modules/Asynchronous-Definition）、CMD（Common Module Definition）规范区别?

AMD 规范在这里：https://github.com/amdjs/amdjs-api/wiki/AMD

CMD 规范在这里：https://github.com/seajs/seajs/issues/242

Asynchronous Module Definition，异步模块定义，所有的模块将被异步加载，模块加载不影响后面语句运行。所有依赖某些模块的语句均放置在回调函数中。

区别：

1. 对于依赖的模块，AMD 是提前执行，CMD 是延迟执行。不过 RequireJS 从 2.0 开始，也改成可以延迟执行（根据写法不同，处理方式不同）。CMD 推崇 as lazy as possible.

2. CMD 推崇依赖就近，AMD 推崇依赖前置。看代码：

// CMD

```
define(function(require, exports, module) {
  var a = require('./a')
  a.doSomething()
  // 此处略去 100 行
  var b = require('./b') // 依赖可以就近书写
  b.doSomething()
  // ...
})
```

// AMD 默认推荐

```
define(['./a', './b'], function(a, b) { // 依赖必须一开始就写好
  a.doSomething()
  // 此处略去 100 行
  b.doSomething()
  // ...
})
```

88、requireJS的核心原理是什么？（如何动态加载的？如何避免多次加载的？如何 缓存的？）

89、documen.write和 innerHTML的区别?

document.write只能重绘整个页面

innerHTML可以重绘页面的一部分

90、DOM操作——怎样添加、移除、移动、复制、创建和查找节点?

（1）创建新节点

```
createDocumentFragment()    //创建一个DOM片段
createElement()   //创建一个具体的元素
createTextNode()   //创建一个文本节点
```

（2）添加、移除、替换、插入

```
appendChild()
removeChild()
replaceChild()
insertBefore() //在已有的子节点前插入一个新的子节点
```

（3）查找

```
getElementsByTagName()    //通过标签名称
getElementsByName()    //通过元素的Name属性的值(IE容错能力较强，会得到一个数组，其中包括id等于name值的)
getElementById()    //通过元素Id，唯一性
```

91、.call() 和 .apply() 的区别?

例子中用 add 来替换 sub，add.call(sub,3,1) == add(3,1) ，所以运行结果为：alert(4);

注意：js 中的函数其实是对象，函数名是对 Function 对象的引用。

```
function add(a,b)
{
  alert(a+b);
}

function sub(a,b)
{
  alert(a-b);
}
add.call(sub,3,1);
```

92、数组和对象有哪些原生方法，列举一下?

创建js数组两种方式： var arr = []; 或 var arr = new Array(); ()里可以指定长度，也可以不指定，指不指定都无所谓，因为js里的数组长度是可变的。

1.concat(arr1,arr2,arr3.....arrx):js数组合并,返回合并后的新数组,至少需要传递一个数组，也可以传递多个数组

```
var arr1 = new Array(3);
arr1[0] = "George";
arr1[1] = "John";
arr1[2] = "Thomas";
arr1[3] = "Jeery";
var arr2 = new Array(3);
arr2[0] = "James";
arr2[1] = "Adrew";
arr2[2] = "Martin";
var arr3 = new Array(3);
arr3[0] = "Java";
arr3[1] = "C#";
arr3[2] = "PHP";
var arr4 = arr1.concat(arr2,arr3);
alert(arr4);
```

2.join():将数组元素按指定的分隔符拼接成一字符串返回，默认分隔符为英文逗号，也可以指定分隔符。

```
var arr = new Array(3)
arr[0] = "George";
arr[1] = "John";
arr[2] = "Thomas";
arr[3] = "Jeery";
arr.join(".");
```

3.sort(fn):数组排序,默认是按英文字母的ASC码升序排列。

比如排在orange前面，其实sort还可以接收一个参数，该参数function类型，有点类似于java里的比较器的意思，就是说如果不想按默认的比较规则来排序，就必须提供一比较函数，该函数有两个参数a、b,

如果返回值小于0，则a排在b前面

如果返回值大于0，则b排在a前面 如果返回值等于0，则a、b位置不变

```
var arr = new Array(6);
arr[0] = 5;
arr[1] = 23;
arr[2] = 4;
arr[3] = 18;
arr[4] = 88;
arr[5] = 10;
arr.sort(sortNumber);
function sortNumber(a, b) {
  return a - b;
}
```

4.pop()：删除数组的最后一个元素，把数组长度减 1，并且返回它删除的元素的值。 如果数组已经为空，则 pop() 不改变数组，并返回 undefined 值。

```
var arr = new Array(6);
arr[0] = 5;
arr[1] = 23;
arr[2] = 4;
arr[3] = 18;
arr[4] = 88;
arr[5] = 10;

var a = arr.pop();
alert(a);
for(var x in arr){
  alert(arr[x]);
}
```

5.push(n1, n2, n3, ...nx):向数组末尾添加一个或多个元素，并返回添加后数组的长度。 注意*，此方法操作的是原数组对象，不会创建副本。此方法可以接收多个参数，至少要传递一个参数.

```
var arr = new Array(6);
arr[0] = 5;
arr[1] = 23;
arr[2] = 4;
arr[3] = 18;
arr[4] = 88;
arr[5] = 10;

var len = arr.push(44,80);
alert(len);
for(var x in arr){
  alert(arr[x]);
}
```

6.reverse():颠倒数组中元素的顺序。

即如果原来数组元素是1,2,3,4,5,调用reverse()后，元素顺序为5,4,3,2,1,注意，此方法直接操作的是原数组对象，不会创建副本。

```
var arr = [3,5,11,6,90,0];
arr.reverse();
for(var x in arr){
  alert(arr[x]);
}
```

7.shift():删除数组的第一个元素，并返回它删除的那个元素。 如果数组已经为空，则 **shift()** 不改变数组，并返回 **undefined** 值。 注意*，此方法直接操作的是原数组，不会创建副本对象

```
var arr = [3,5,11,6,90,0];
var a = arr.shift();
alert(a);
for(var x in arr){
  alert(arr[x]);
}
```

8.slice(start,end):用于截取**start**和**end**之间的数组元素并存入新数组返回. 注意*，此方法不会修改原数组，会创建数组副本对象。如果**end**未指定，则表示直接从**start**直到数组末尾，如果**start**或**end**为负数，表示从后面开始算起，比如**-1**，表示从倒数第一个元素算起，以此类推。截取的区间范围是[**start**,**end**),前闭后开区间,且**start**必须小于**end**，如果没找到一个元素，则返回一个空数组即数组的长度为**0**。

```
var arr = [3,5,11,6,90,0];
var a = arr.slice(2,4);
alert(a.join());
```

9. splice(index,howmany,element1,.....,elementX): 用于删除从 **index** 处开始的零个或多个元素，并且用参数列表中声明的一个或多个值来替换那些被删除的元素，并返回刚刚删除的元素组成的新数组。

注意*：此方法是直接操作的原数组对象，不会创建对象副本。
第一个参数：表示从index位置开始删除，index从零开始计算
第二个参数：表示从index位置开始，往后连续删除几个元素，前两个参数是必需的，后面的参数是可选的。
后面的参数是用添加的元素，添加的元素从index处开始添加，如果后面添加的元素个数大于
实际删除的元素个数，多出几个，后面的元素就往后移动几位。比如你实际删除了4个元素，
而实际你后面添加了6个元素，那最终会从index处开始添加6个元素，由于前面只删除4个元素，
位置不够，所以后面的元素会自动往后移动2位。

```
var arr = [3,5,11,6,90,0,15,57,70,20];
var a = arr.splice(0,4,1,2,3,4,5,6);
alert(a);
for(var x in arr){
  alert(arr[x]);
}
```

10. unshift(element1,.....,element): 向数组开头添加一个或多个元素，并返回添加后的数组长度。至少要传一个参数。 注意*：此方法是直接操作原数组，最后添加的元素index=0,以此类推。

```
var arr = [3,5,11,6,90,0,15,57,70,20];
arr.unshift(22,23,24);
alert(arr.toString());
alert(arr.length);
```

11.扩展Array的函数：

```
Array.prototype.indexOf = function(o){
  for(var i = 0,len=this.length; i<len;i++){
    if(this[i] == o){
        return i;
        }
    }
    return -1;
}
Array.prototype.remove = function(o){
    var index = this.indexOf(o);
  if(index != -1){
    this.splice(index,1);
    }
    return this;
}
var arr = [3,5,11,6,90,0,15,57,70,20];
arr.remove(90);
alert(arr.toString());
alert(arr.length);
```

93、JS 怎么实现一个类。怎么实例化这个类

```
function Person() {
  this.name=" 张三 "; // 定义一个属性 name
  this.sex=" 男 "; // 定义一个属性 sex
  this.say=function(){ // 定义一个方法 say()
    document.write("嗨！大家好，我的名字是 " + this.name + " ，性别是
" + this.sex + "。");
  }
}
var zhangsan=new Person();
zhangsan.say();
```

94、JavaScript中的作用域与变量声明提升?

96、内存泄漏：指一块被分配的内存既不能

使用，又不能回收，直到浏览器进程结束。

如何避免内存泄露?

1、JS的回收机制

JavaScript垃圾回收的机制很简单：找出不再使用的变量，然后释放掉其占用的内存，但是这个过程不是实时的，因为其开销比较大，所以垃圾回收系统（GC）会按照固定的时间间隔,周期性的执行。

到底哪个变量是没有用的？所以垃圾收集器必须跟踪到底哪个变量没用，对于不再有用的变量打上标记，以备将来收回其内存。用于标记的无用变量的策略可能因实现而有所区别，通常情况下有两种实现方式：标记清除和引用计数。引用计数不太常用，标记清除较为常用。

2、标记清除

js中最常用的垃圾回收方式就是标记清除。当变量进入环境时，例如，在函数中声明一个变量，就将这个变量标记为“进入环境”。从逻辑上讲，永远不能释放进入环境的变量所占用的内存，因为只要执行流进入相应的环境，就可能会用到它们。而当变量离开环境时，则将其标记为“离开环境”。

function test(){

```
var a=10;//被标记，进入环境

var b=20;//被标记，进入环境
```

}

test();//执行完毕之后a、b又被标记离开环境，被回收 3、引用此时

引用计数的含义是跟踪记录每个值被引用的次数。当声明了一个变量并将一个引用类型值（function object array）赋给该变量时，则这个值的引用次数就是1。如果同一个值又被赋给另一个变量，则该值的引用次数加1。相反，如果包含对这个值引用的变量又取得了另外一个值，则这个值的引用次数减1。当这个值的引用次数变成0时，则说明没有办法再访问这个值了，因而就可以将其占用的内存空间回收回来。这样，当垃圾回收器下次再运行时，它就会释放那些引用次数为0的值所占用的内存。

```
function test(){
  var a={};//a的引用次数为0
  var b=a;//a的引用次数加1，为1
  var c=a;//a的引用次数加1，为2
  var b={};//a的引用次数减1，为1
}
```

4、哪些操作会造成内存泄露

1）意外的全局变量引起的内存泄露

```
function leak(){
  leak="xxx";//leak成为一个全局变量，不会被回收
}
```

2）闭包引起的内存泄露

```
function bindEvent(){
  var obj=document.createElement("XXX");
  obj.onclick=function(){
    //Even if it's a empty function
  }
}
```

闭包可以维持函数内局部变量，使其得不到释放。 上例定义事件回调时，由于是函数内定义函数，并且内部函数--事件回调的引用外暴了，形成了闭包。

解决之道，将事件处理函数定义在外部，解除闭包,或者在定义事件处理函数的外部函数中，删除对dom的引用。

```
//将事件处理函数定义在外部
function onclickHandler(){
  //do something
}
function bindEvent(){
  var obj=document.createElement("XXX");
  obj.onclick=onclickHandler;
}
//在定义事件处理函数的外部函数中，删除对dom的引用
function bindEvent(){
  var obj=document.createElement("XXX");
  obj.onclick=function(){
    //Even if it's a empty function
  }
  obj=null;
}
```

3）没有清理的DOM元素引用

```
var elements={
    button: document.getElementById("button"),
    image: document.getElementById("image"),
text: document.getElementById("text")
};
function doStuff(){
    image.src="http://some.url/image";
    button.click():
console.log(text.innerHTML)
}
function removeButton(){
    document.body.removeChild(document.getElementById('button'))
}
```

4）被遗忘的定时器或者回调

```
var someResouce=getData();
setInterval(function(){
    var node=document.getElementById('Node');
    if(node){
        node.innerHTML=JSON.stringify(someResouce)
    }
},1000)
```

这样的代码很常见, 如果 id 为 Node 的元素从 DOM 中移除, 该定时器仍会存在, 同时, 因为回调函数中包含对 someResource 的引用, 定时器外面的 someResource 也不会被释放。 5）子元素存在引起的内存泄露

黄色是指直接被 js变量所引用，在内存里，红色是指间接被 js变量所引用，如上图，refB 被 refA 间接引用，导致即使 refB 变量被清空，也是不会被回收的子元素 refB 由于 parentNode 的间接引用，只要它不被删除，它所有的父元素（图中红色部分）都不会被删除。

6）IE7/8引用计数使用循环引用产生的问题

```
function fn(){
  var a={};
  var b={};
  a.pro=b;
  b.pro=a;
}
fn();
```

fn()执行完毕后，两个对象都已经离开环境，在标记清除方式下是没有问题的，但是在引用计数策略下，因为a和b的引用次数不为0，所以不会被垃圾回收器回收内存，如果fn函数被大量调用，就会造成内存泄漏。在IE7与IE8上，内存直线上升。

IE中有一部分对象并不是原生js对象。例如，其内存泄漏DOM和BOM中的对象就是使用C++以COM对象的形式实现的，而COM对象的垃圾回收机制采用的就是引用计数策略。因此，即使IE的js引擎采用标记清除策略来实现，但js访问的COM对象依然是基于引用计数策略的。换句话说，只要在IE中涉及COM对象，就会存在循环引用的问题。

```
var element=document.getElementById("some_element");
var myObject=new Object();
myObject.e=element;
element.o=myObject;
```

上面的例子在一个DOM元素（element)与一个原生js对象（myObject)之间创建了循环引用。其中，变量myObject有一个名为e的属性指向element对象；而变量element也有一个属性名为o回指myObject。由于存在这个循环引用，即使例子中的DOM从页面中移除，它也永远不会被回收。

看上面的例子，有人会觉得太弱了，谁会做这样无聊的事情，但是其实我们经常会这样做

```
window.onload=function outerFunction(){
  var obj=document.getElementById("element"):
  obj.onclick=function innerFunction(){};
};
```

这段代码看起来没什么问题，但是**obj**引用了**document.getElementById(“element”)**，而**document.getElementById(“element”)**的**onclick**方法会引用外部环境中的变量，自然也包括**obj**，是不是很隐蔽啊。

最简单的解决方式就是自己手工解除循环引用，比如刚才的函数可以这样

```
myObject.element=null;
element.o=null;
window.onload=function outerFunction(){
  var obj=document.getElementById("element"):
  obj.onclick=function innerFunction(){};
  obj=null;
};
```

将变量设置为**null**意味着切断变量与它此前引用的值之间的连接。当垃圾回收器下次运行时，就会删除这些值并回收它们占用的内存。 要注意的是，**IE9+**并不存在循环引用导致**Dom**内存泄漏问题，可能是微软做了优化，或者**Dom**的回收方式已经改变

5、如何分析内存的使用情况

Google Chrome浏览器提供了非常强大的**JS**调试工具，**Memory** 视图 **profiles** 视图让你可以对 **JavaScript** 代码运行时的内存进行快照，并且可以比较这些内存快照。它还让你可以记录一段时间内的内存分配情况。在每一个结果视图中都可以展示不同类型的列表，但是对我们最有用的是 **summary** 列表和 **comparison** 列表。**summary** 视图提供了不同类型的分配对象以及它们的合计大小：**shallow size**（一个特定类型的所有对象的总和）和 **retained size** （**shallow size** 加上保留此对象的其它对象的大小）。**distance** 显示了对象到达 **GC** 根（校者注：最初引用的那块内存，具体内容可自行搜索该术语）的最短距离。 **comparison** 视图提供了同样的信息但是允许对比不同的快照。这对于找到泄漏很有帮助。

6、怎样避免内存泄露

1）减少不必要的全局变量，或者生命周期较长的对象，及时对无用的数据进行垃圾回收；

2）注意程序逻辑，避免“死循环”之类的 ；

3）避免创建过多的对象 原则：不用了的东西要及时归还。

97、jQuery的源码看过吗？能不能简单概况一下它的实现原理？

参考资料

98、谈一下jQuery中的bind(),live(),delegate(),on()的区别？

bind(type,[data],fn) 为每个匹配元素的特定事件绑定事件处理函数

$("a").bind("click",function(){alert("ok");});

live(type,[data],fn) 给所有匹配的元素附加一个事件处理函数，即使这个元素是以后再添加进来的

$("a").live("click",function(){alert("ok");});

delegate(selector,[type],[data],fn) 指定的元素（属于被选元素的子元素）添加一个或多个事件处理程序，并规定当这些事件发生时运行的函数

$("#container").delegate("a","click",function(){alert("ok");})

on(events,[selector],[data],fn) 在选择元素上绑定一个或多个事件的事件处理函数

差别：

.bind()是直接绑定在元素上

.live()则是通过冒泡的方式来绑定到元素上的。更适合列表类型的，绑定到document DOM节点上。和.bind()的优势是支持动态数据。

.delegate()则是更精确的小范围使用事件代理，性能优于.live()

.on()则是最新的1.9版本整合了之前的三种方式的新事件绑定机制

99、针对 jQuery性能的优化方法?

1、总是使用#id去寻找element.

在jQuery中最快的选择器是ID选择器 ($('#someid')). 这是因为它直接映射为JavaScript的getElementById()方法。 选择单个元素

```
<div id="content">
 <form method="post" action="/">
  <h2>Traffic Light</h2>
  <ul id="traffic_light">
   <li><input type="radio" class="on" name="light" value="red" /> Red</li>
   <li><input type="radio" class="off" name="light" value="yellow" />
Yellow</li>
   <li><input type="radio" class="off" name="light" value="green" />
Green</li>
  </ul>
  <input class="button" id="traffic_button" type="submit" value="Go" />
 </form>
</div>
```

选择button的性能不好的一种方式：

var traffic_button = $('#content .button');

取而代之的是直接选择button:

var traffic_button = $('#traffic_button');

选择多个元素

在我们讨论选择多个元素的时候，我们真正需要知道的是DOM的遍历和循环才是性能低下的原因。为了尽量减少性能损失， 总是使用最近的父ID去寻找。

var traffic_lights = $('#traffic_light input');

在Classes前面使用Tags

在jQuery中第二快的选择器就是Tag选择器 ($('head')). 而这是因为它直接映射到JavaScript的getElementsByTagName()方法。

```
<div id="content">
 <form method="post" action="/">
  <h2>Traffic Light</h2>
  <ul id="traffic_light">
   <li><input type="radio" class="on" name="light" value="red" /> Red</li>
   <li><input type="radio" class="off" name="light" value="yellow" />
Yellow</li>
   <li><input type="radio" class="off" name="light" value="green" />
Green</li>
  </ul>
  <input class="button" id="traffic_button" type="submit" value="Go" />

 </form>
</div>
```

总是在一个Class前面加上一个tag名字（记得从一个ID传下来）

var active_light = $('#traffic_light input.on');

注意：在jQuery里Class选择器是最慢的一个选择器;在IE中它循环整个DOM。可能的话尽量避免使用它。不要在ID前面 加Tags。例如，它会因为去循环所有的

元素去寻找ID为content的
，而导致很慢。

var content = $('div#content');

按照同样的思路，从多个ID传下来是冗余的。

var traffic_light = $('#content #traffic_light');

3、缓存jQuery对象

养成保存jQuery对象到一个变量上（就像上面的例子）的习惯。例如，不要这样做：

```
$('#traffic_light input.on).bind('click', function(){...});
$('#traffic_light input.on).css('border', '3px dashed yellow');
$('#traffic_light input.on).css('background-color', 'orange');
$('#traffic_light input.on).fadeIn('slow');
```

取而代之，首现保存jQuery变量到一个本地变量后，再继续你的操作。

```
var $active_light = $('#traffic_light input.on');
$active_light.bind('click', function(){...});
$active_light.css('border', '3px dashed yellow');
$active_light.css('background-color', 'orange');
$active_light.fadeIn('slow');
```

提示：使用$前辍表示我们的本地变量是一个jQuery包集。记住，不要在你的应该程序里出现一次以上的jQuery重复的选择操作。 额外提示：延迟存储jQuery对象结果。

如果你想在你的程序的其它地方使用jQuery结果对象（result object(s)），或者你的函数要执行多次，要把它缓存在一个全局范围的对象里。通过定义一个全局容器保存jQuery结果对象，就可以在其它的函数里引用它。

```
// Define an object in the global scope (i.e. the window object)
window.$my ={
 // Initialize all the queries you want to use more than once
 head : $('head'),
 traffic_light : $('#traffic_light'),

 traffic_button : $('#traffic_button')};
function do_something(){
 // Now you can reference the stored results and manipulate them
 var script = document.createElement('script');
 $my.head.append(script);
 // When working inside functions, continue to save jQuery results
 // to your global container.
 $my.cool_results = $('#some_ul li');
 $my.other_results = $('#some_table td');
 // Use the global functions as you would a normal jQuery result
 $my.other_results.css('border-color', 'red');
 $my.traffic_light.css('border-color', 'green');
}
```

4、更好的利用链

前面的例子也可以这样写：

```
var $active_light = $('#traffic_light input.on');
$active_light.bind('click', function(){...})
 .css('border', '3px dashed yellow')
 .css('background-color', 'orange')
 .fadeIn('slow');
```

这样可以让我们写更少的代码，使JavaScript更轻量。

5、使用子查询

jQuery允许我们在一个包集上附加其它的选择器。因为我们已经在本地变量里保存了父对象这样会减少以后在选择器上的性能开销。

```
<div id="content">
 <form method="post" action="/">
  <h2>Traffic Light</h2>

  <ul id="traffic_light">
   <li><input type="radio" class="on" name="light" value="red" /> Red</li>
   <li><input type="radio" class="off" name="light" value="yellow" />
Yellow</li>
   <li><input type="radio" class="off" name="light" value="green" />
Green</li>

  </ul>
  <input class="button" id="traffic_button" type="submit" value="Go" />
 </form>
</div>
```

例如，我们可以利用子查询缓存active和inactive lights以便后面的操作

```
var $traffic_light = $('#traffic_light'),
$active_light = $traffic_light.find('input.on'),
$inactive_lights = $traffic_light.find('input.off');
```

提示：可以用逗号隔开一次定义多个本地变量，这样可以节省一些字节。

6、限制直接对DOM操作

DOM操作的基本做法是在内存中创建DOM结构，然后再更新DOM结构。这不是jQuery最好的做法，但对JavaScript来讲是高效的。直接操作DOM结构性能是低下的。 例如，如果你需要动态创建一列元素，不要这样做：

```
var top_100_list = [...], // assume this has 100 unique strings
$mylist = $('#mylist'); // jQuery selects our <ul> element
for (var i=0, l=top_100_list.length; i<l; i++){
 $mylist.append('<li>' + top_100_list[i] + '</li>');
}
```

取而代之，我们希望在插入DOM结构之前先在一个字符串里创建一套元素。 代码

```
var top_100_list = [...], // assume this has 100 unique strings
$mylist = $('#mylist'), // jQuery selects our <ul> element
top_100_li = ""; // This will store our list items
for (var i=0, l=top_100_list.length; i<l; i++){
 top_100_li += '<li>' + top_100_list[i] + '</li>';
}
$mylist.html(top_100_li);
```

更快的做法，在插入DOM结构之前我们应该总是在一个父节点里包含许多元素

```
var top_100_list = [...], // assume this has 100 unique strings
$mylist = $('#mylist'), // jQuery selects our <ul> element
top_100_ul = '<ul id="#mylist">'; // This will store our entire unordered
list
for (var i=0, l=top_100_list.length; i<l; i++){
 top_100_ul += '<li>' + top_100_list[i] + '</li>';
}
top_100_ul += '</ul>'; // Close our unordered list
$mylist.replaceWith(top_100_ul);
```

如是你照着上面的做了还是对性能有些迷惑的话，可以参考以下内容：

- 试一下jQuery提供的Clone()方法。Clone()方法创建节点数的拷贝，随后你可以在这个副本中进行操作。
- 使用DOM DocumentFragments. As the creator of jQuery points out, 比直接操作DOM性能上更好. 先创建你需要的结构(就像我们上面用一个字符串做的一样), 然后使用jQuery的 insert or replace methods.

100、针对 jQuery 的优化方法?

*基于Class的选择性的性能相对于Id选择器开销很大，因为需遍历所有DOM元素。

*频繁操作的DOM，先缓存起来再操作。用Jquery的链式调用更好。

```
比如: var str=$("a").attr("href");
```

*for (var i = size; i < arr.length; i++) {}

```
for 循环每一次循环都查找了数组 (arr) 的.length 属性，在开始循环的时候设置一个变量来存储这个数字，可以让循环跑得更快:   for (var i = size, length = arr.length; i < length; i++) {}
```

101、什么是Zepto，Zepto的点透问题如何解决?

方案一：来得很直接github上有个fastclick可以完美解决
https://github.com/ftlabs/fastclick

引入fastclick.js，因为fastclick源码不依赖其他库所以你可以在原生的js前直接加上

```
window.addEventListener( "load", function() {
    FastClick.attach( document.body );3 }, false );
```

或者有zepto或者jqm的js里面加上

$(function() {FastClick.attach(document.body);3 });

当然require的话就这样:

var FastClick = require('fastclick'); FastClick.attach(document.body, options);

方案二：用touchend代替tap事件并阻止掉touchend的默认行为preventDefault()

```
 $("#cbFinish").on("touchend", function (event) {
 //很多处理比如隐藏什么的
   event.preventDefault();
});
```

方案三：延迟一定的时间(300ms+)来处理事件

```
 $("#cbFinish").on("tap", function (event) {
  setTimeout(function(){
//很多处理比如隐藏什么的
},320);
 });
```

这种方法其实很好，可以和fadeIn/fadeOut等动画结合使用，可以做出过度效果

理论上上面的方法可以完美的解决tap的点透问题，如果真的倔强到不行，用click

102、jQuery一个对象可以同时绑定多个事件，这是如何实现的?

- 多个事件同一个函数：

```
$("div").on("click mouseover", function(){});
```

- 多个事件不同函数

```
$("div").on({

  click: function(){},

  mouseover: function(){}

});
```