### html语义化，css模块化

就是尽可能的理解要表达的内容，选择适合的标签，将内容转换成浏览器认识的语言，通过浏览器传达给用户。从这个角度来说，我们更像是“翻译”。

除了专业的人外，没人会去看我们的标签是否使用得有语义

HTML的标签还不足以表达所有可能的语义

语义化会需要增加一些额外的代码

HTML5新增的几个标签，更多的补充了目前HTML标签语义上的不足，这点也可以看出语义化是以后发展的一个方向，当然更大的目标是统一的实现标准，语义化只是为了这个目标所使用的一个方法。这可能需要相当的一段时间，而我们正经历着这个过程。

### 模块化

模块化就是为了减少循环依赖，减少耦合，提高设计的效率。为了做到这一点，我们需要有一个设计规则，所有的模块都在这个规则下进 行设计。良好的设计规则，会把耦合密集的设计参数进行归类作为一个模块，并以此划分工作任务。而模块之间彼此通过一个固定的接口（所谓的可见参数）进行交互，除此之外的内部实现（所谓的隐参数）则由模块的开发团队进行自由发挥。

程序模块化的目的：

1.减少循环依赖

2.减少耦合

3.提高设计效率

css模块化原则

分离结构和表现

分离容器和内容

### Flex布局，position，垂直水平居中三种必问前端常识

Flex 是 Flexible Box 的缩写，意为"弹性布局"，用来为盒状模型提供最大的灵活性。

Webkit 内核的浏览器，必须加上-webkit前缀。

注意，设为 Flex 布局以后，子元素的float、clear和vertical-align属性将失效。

容器默认存在两根轴：水平的主轴（main axis）和垂直的交叉轴（cross axis）。主轴的开始位置（与边框的交叉点）叫做main start，结束位置叫做main end；交叉轴的开始位置叫做cross start，结束位置叫做cross end。

项目默认沿主轴排列。单个项目占据的主轴空间叫做main size，占据的交叉轴空间叫做cross size。

#### 容器的属性

以下6个属性设置在容器上。

flex-direction

flex-wrap

flex-flow

justify-content

align-items

align-content

align-items属性



#### 基本概念

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"。

### 任何一个容器都可以指定为 Flex 布局。

flex-grow属性

flex-grow属性定义项目的放大比例，默认为0，即如果存在剩余空间，也不放大。

flex-basis属性

flex-basis属性定义了在分配多余空间之前，项目占据的主轴空间（main size）。浏览器根据这个属性，计算主轴是否有多余空间。它的默认值为auto，即项目的本来大小

flex属性

flex属性是flex-grow, flex-shrink 和 flex-basis的简写，默认值为0 1 auto。后两个属性可选。

align-self属性

align-self属性允许单个项目有与其他项目不一样的对齐方式，可覆盖align-items属性。默认值为auto，表示继承父元素的align-items属性，如果没有父元素，则等同于stretch

### H5自适应方案

H5自适应方案大家在网速能找到很多，我个人推荐一种我非常喜欢的方式，就是rem. rem是一种相对单位，它基于html的font-size值来进行调整。

通常我们以750为基准，我们会在header中嵌套一段js脚本，获取手机网页分辨率尺寸除以375，为了方便计算，我们假设750像素下1rem = 100px；所以 我们除以375后需要乘以50.

### 清除浮动的几种方式，各自的优缺点？

①给父元素单独定义高度  
优点：简单快速、代码少。  
缺点：无法进行响应式布局。

②在标签结尾处加空div标签**<div style="clear: both"></div>**  
优点：简单快速、代码少，兼容性较高。  
缺点：增加空标签，不利于页面优化。

③父级定义overflow:hidden  
优点：简单快速、代码少，兼容性较高。  
缺点：超出部分被隐藏了，在布局的时候要注意。

④父级定义class="clearfix"，使用after伪类和zoom  
**.clearfix:after{content:"";display:block;clear:both;height:0;overflow:hidden;visibility:hidden;}**  
**.clearfix{zoom:1;}**  
优点：写法固定，没有多余结构，兼容性高。  
缺点：代码多。

### 用纯CSS创建一个三角形的原理是什么

.box{

width:0px;

height:0px;

border: 50px solid transparent;

border-left:50px solid #ef4848;

}

### 三栏布局的方法有哪些分别描述一下

圣杯布局

float布局

优势：简单

劣势：中间部分最后加载，内容较多时影响体验

<style> .left { float: left*;* width: 100px*;* height: 200px*;* background-color: red*;* } .right { float: right*;* width: 100px*;* height: 200px*;* background-color: yellow*;* } .main { background-color: green*;* height: 200px*;* margin-left: 120px*;* margin-right: 120px*;* } .container { border: 1px solid black*;* } <div class="container"> <div class="left"></div> <div class="right"></div> <div class="main"></div> </div>

Flex 布局

<style> .container{ display:flex; justify-content: center; height: 200px; background: #eee; } .left { width: 200px; background-color: red; height: 100%; } .main { background-color: yellow; flex: 1; } .right { width: 200px; background-color: green; } </style> <div class="container"> <div class="left">1</div> <div class="main">2</div> <div class="right">3</div> </div>

BFC规则

BFC(块格式化上下文)规则规定：BFC不会和浮动元素重叠。所以如果将main元素设定为BFC元素即可：

双飞翼布局

绝对定位

### css3实现0.5px的细线

<style> .line { position: relative; } .line:after { content: ""; position: absolute; left: 0; top: 0; width: 100%; height: 1px; background-color: #000000; -webkit-transform: scaleY(.5); transform: scaleY(.5); } </style> <div class="line"></div>

### CSS选择器

·id选择器（#myId）  
·类选择器（.myClassName）  
·标签选择器（div, h1, p）  
·后代选择器（h1 p）  
·相邻后代选择器（子）选择器（ul > li）  
·兄弟选择器（li~a）  
·相邻兄弟选择器（li+a）  
·属性选择器（a[rel="external"]）  
·伪类选择器（a:hover, li:nth-child）  
·伪元素选择器（::before, ::after）  
·通配符选择器（\*）

### ::before 和 :after 中双冒号和单冒号的区别？这2个伪元素的作用？

·在 CSS3 中 : 表示伪类， :: 表示伪元素

·想让插入的内容出现在其他内容前，使用::befroe。否则，使用::after

### CSS中哪些属性可以继承？

·每一个属性在定义中都给出了这个属性是否具有继承性，一个具有继承性的属性会在没有指定值的时候，会使用父元素的同属性的值

来作为自己的值。

·一般具有继承性的属性有，字体相关的属性，font-size和font-weight等。

·文本相关的属性，color和text-align等。

·表格的一些布局属性、列表属性如list-style等。

·还有光标属性cursor、元素可见性visibility。

·当一个属性不是继承属性的时候，我们也可以通过将它的值设置为inherit来使它从父元素那获取同名的属性值来继承

### src与href的区别？

1、href 是指向网络资源所在位置，建立和当前元素（锚点）或当前文档（链接）之间的链接，用于超链接。  
2、src是指向外部资源的位置，指向的内容将会嵌入到文档中当前标签所在位置；在请求src资源时会将其指向的资源下载并应用到文档内，例如js脚本，img图片和frame等元素。当浏览器解析到该元素时，会暂停其他资源的下载和处理，直到将该资源加载、编译、执行完毕，图片和框架等元素也如此，类似于将所指向资源嵌入当前标签内。这也是为什么将js脚本放在底部而不是头部。

### CSS优化，提高性能的方法有哪些？

·加载性能：  
① CSS 压缩：将写好的CSS 进行打包压缩，可以减少很多的体积。  
② CSS单一样式：当需要下边距和左边距的时候，很多时候选择：margin: top 0 bottom 0;  
但margin-top: top;margin-bottom: bottom;执行的效率更高。  
·选择器性能：  
① 关键选择器。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

### webpack的作用及核心配置

webpack配置中需要理解几个核心的概念Entry 、Output、Loaders 、Plugins、 Chunk

Entry：指定webpack开始构建的入口模块，从该模块开始构建并计算出直接或间接依赖的模块或者库  
Output：告诉webpack如何命名输出的文件以及输出的目录  
Loaders：由于webpack只能处理javascript，所以我们需要对一些非js文件处理成webpack能够处理的模块，比如sass文件

Plugins：Loaders将各类型的文件处理成webpack能够处理的模块，plugins有着很强的能力。插件的范围包括，从打包优化和压缩，一直到重新定义环境中的变量。但也是最复杂的一个。比如对js文件进行压缩优化的UglifyJsPlugin插件  
Chunk：coding split的产物，我们可以对一些代码打包成一个单独的chunk，比如某些公共模块，去重，更好的利用缓存。或者按需加载某些功能模块，优化加载时间。在webpack3及以前我们都利用CommonsChunkPlugin将一些公共代码分割成一个chunk，实现单独加载。在webpack4 中CommonsChunkPlugin被废弃，使用SplitChunksPlugin

### ==与===的比较，===的使用情况

===的使用情况我回答是，一般在比较字符串时来使用。

### 双向绑定的定义

* 一般只有UI表单控件才存在双向数据绑定，非UI表单控件只有单向数据绑定。
* 单向数据绑定是指：M的变化可以自动更新到ViewModel，但ViewModel的变化需要手动更新到M（通过给表单控件设置事件监听）
* 双向数据绑定是指念：M的变化可以自动更新到ViewModel，ViewModel的变化也可以自动更新到M
* 双向绑定 = 单向绑定 + UI事件监听。双向和单向只不过是框架封装程度上的差异，本质上两者是可以相互转换的。
* 优缺点：在表单交互较多的情况下，单向数据绑定的优点是数据更易于跟踪管理和维护，缺点是代码量较多比较啰嗦，双向数据绑定的优缺点和单向绑定正好相反

### 前端性能优化的常用手段

从最开始的CSS放到<head>里面、js放到</body>前面、使用雪碧图等，到后面的静态资源压缩、合并以及使用iconfont代替小图标，再到最近实践的gzip压缩、设置HTTP Header缓存字段...

gzip压缩、设置ETag等，早就在《高性能网站建设指南》那两本书中看过，但我一直认为这都是后端小伙伴干得事，就没有怎么留意过。直到最近，对前后端分离的理解越来越充分，对整个项目的部署越来越清晰，对项目里面的资源请求越来越明白，才恍然意识到：前后端分离了，这他妈就是前端自己干的事啊！！！

从以下几个方面来说一说自己实践过的优化方法

### 浏览器渲染页面的过程

所谓优化，第一个要弄明白的就是：优化什么、从哪里优化。前端做出来的页面是在浏览器里面呈现的，那浏览器是怎么渲染这个页面的呢？遇到CSS、js静态资源，浏览器是怎么处理的？具体的过程这里不再赘述，网上资源一大堆，我自己之前也写过一篇，[《网站性能优化—CRP》](https://segmentfault.com/a/1190000008550336" \t "_blank)，算是谷歌文档的翻译版吧。

理解了浏览器渲染页面的过程，也就明白了：CSS为什么要放到<head>里面、js放到</body>前面，以及js的异步加载（async、defer）等优化。

### 减少HTTP请求

· CSS/JS 合并打包

· 小图标等用iconfont代替：作为单个DOM节点使用，可以设置大小、颜色等，非常便利。个人建议前端来维护这个字体包，每次有新增的图标，让设计师给我们对应的svg文件即可，前端自己去 [icomoon.io/](https://icomoon.io/" \t "_blank) 这个网站，导入原来的selection.json文件，增量生成新的css，无比方便。之前，我一直以为iconfont只能是单色的呢，其实也可以是多色的，svg里面多一些path而已，设计师会搞定的。生成字体后，前端正常引用即可（引用的时候，多色字体会多一些标签）

· 使用base64格式的图片：有些小图片，可能色彩比较复杂，这个时候再用iconfont就有点不合适了，此时可以将其转化为base64格式（不能缓存），直接嵌在src中，比如webpack的url-loader设置limit参数即可

### 减少静态资源的体积

· 压缩静态资源：合并打包的js、css文件体积一般会比较大，一些图片也会比较大，这个时候必须要压缩处理。前后端分离项目，不论是gulp还是webpack，都有相应的工具包。针对个别图片，有时候也可以单独拿出来处理，我个人经常使用这个网站 [tinypng.com/](https://tinypng.com/" \t "_blank) 在线压缩

· 编写高效率的CSS：涉及到代码层面的优化比较多也比较细，不同水平的技术人员写出来的肯定不一样，这里不做进一步的分析。但是为什么要把CSS拿出来说一说呢？因为现在项目里面基本上都在使用CSS预处理器，Less、SaaS、Stylus等等，这导致了某些初级前端的滥用：嵌套5、6层，甚者能达到7、8层，吓死个人！嵌套这么深，影响浏览器查找选择器的速度不说，这也一定程度上产出了很多冗余的字节，这个要改、要提醒，一般建议嵌套3层即可。关于编写高效率的CSS，推荐一篇文章，[《Writing efficient CSS selectors》](https://csswizardry.com/2011/09/writing-efficient-css-selectors/" \t "_blank)

服务端开启gzip压缩：大招，最近刚知晓，真是太牛逼了，一般的css、js文件能压缩60、70%，当然，这个比率可以设定的。前后端分离，如果前端部署用node、express作服务器的话，使用中间件compression即可开启gzip压缩：

### 使用缓存

设置Http Header里面缓存相关的字段，做进一步的优化。可以设置etag、maxAge等，进一步会有200缓存和304缓存的区别：

### 内存溢出

这种优化因问题而异吧，最重要的是善于使用Google DevTools里面的Performance面板和Memory面板去分析、查找问题，进而找到优化的点。

内存溢出目前我只碰到过一次，同事用echarts画K线图，同事的js逻辑写的有问题，点击事件发生时canvas反复渲染，导致内存逐渐升高，在APP内，直接导致了APP闪退。我重写了一下js逻辑，针对canvas做了一些优化，修复了这个bug。

目前对这块分析经验还不是很多，后续碰到问题再实践。

### 。减少reflow的方法

不要一条一条地修改 DOM 的样式。与其这样，还不如预先定义好 css 的 class，然后修改 DOM 的 className：

// 不好的写法 var left = 10, top = 10; el.style.left = left + "px"; el.style.top = top + "px"; // 推荐写法 el.className += " theclassname";把 DOM 离线后修改。如：

使用 documentFragment 对象在内存里操作 DOM。

b> 先把 DOM 给 display:none (有一次 repaint)，然后你想怎么改就怎么改。比如修改 100 次，然后再把他显示出来。

c>clone 一个 DOM 节点到内存里，然后想怎么改就怎么改，改完后，和在线的那个的交换一下。

不要把 DOM 节点的属性值放在一个循环里当成循环里的变量。不然这会导致大量地读写这个结点的属性。

尽可能的修改层级比较低的 DOM节点。当然，改变层级比较底的 DOM节点有可能会造成大面积的 reflow，但是也可能影响范围很小。

为动画的 HTML 元件使用 fixed 或 absoult 的 position，那么修改他们的 CSS 是会大大减小 reflow 。

千万不要使用 table 布局。因为可能很小的一个小改动会造成整个 table 的重新布局。

### 中级前端所有手写内容

#### 防抖、节流

* 原理都是利用闭包保存变量。防抖是任务频繁触发的情况下，只有任务触发的间隔超过指定间隔的时候，任务才会执行，一般用于输入框实时搜索；节流是规定函数在指定的时间间隔内只执行一次，一般用于scroll事件。

#### 深浅拷贝

* 递归实现

#### 数组乱序、数组去重（各种时间空间复杂度）

#### 数组filter

#### 数组flat（平展一层到多层）

#### 手写call、bind、apply（及了解这三种的区别）

call、apply、bind是ES5中能改变this指向的方法。一般都会问一下这三个方法的区别。call和apply的传参不同，call接收逗号分隔的参数，apply接收数组（如何记不清这两个方法的区别的话，可以记apply接收array，都是a开头的，这样比较好记），调用都会立即执行。而bind调用完返回的是一个函数，需要再次调用才会执行。 接下来就会引申到能实现一个call/apply吗？或者能用apply实现一个bind吗？

#### ES6中export与export default差异总结

* export default在一个模块中只能有一个，但是export可以有多个。
* 通过export导出的（属性或者方法）可以修改，但是通过export default导出的是不可修改的。
* export var e1='...' 是合法语句，但是export default var e2='...'是不合法的（let和const也一样）。
* export default可以直接添加标识符导出，例如export default e2;export如果要导出已经声明的表示符,必须使用{},例如export {e1},注意：这里{}不是声明一个对象。
* ES6中模块通过export和export default暴露出来的属性或者方式并不是普通的赋值或者引用，它们是对模块内部定义的标志符类似指针的绑定。
* 对于一个导出的属性或者方法，在什么地方导出不重要，在什么时候导入也不重要，重要的是:访问这这个绑定的时候的当前值。

#### 简述Generator函数的基本语法

Generator （生成器）函数有多种理解角度。语法上，首先可以把它理解成，Generator 函数是一个状态机，封装了多个内部状态。执行 Generator 函数会返回一个遍历器对象。

特征一：function关键字与函数名之间有一个号（\*写在之间的任何位置都可以通过，不区分有几个空格）

特征二：函数体内有yield（产出）表达式，定义了不同的内部状态

**function\*** helloWorldGenerator() {

**yield** 'hello';

**yield** 'world';

**return** 'ending';}

**var** hw **=** helloWorldGenerator();hw.next()*// { value: 'hello', done: false }*hw.next()*// { value: 'world', done: false }*hw.next()*// { value: 'ending', done: true }*hw.next()*// { value: undefined, done: true }*

函数执行.next（）后返回的也不是函数运行结果，而是一个指向内部状态的指针对象。每次调用next方法，内部指针就从函数头部或上一次停下来的地方开始执行，直到遇到下一个yield表达式（或return语句）为止。换言之，Generator 函数是分段执行的，yield表达式是暂停执行的标记，而next方法可以恢复执行。

**function\*** gen() {

**yield** 123 **+** 456;}

上面代码中，yield后面的表达式123 + 456，不会立即求值，只会在next方法将指针移到这一句时，才会求值。

yield表达式与return语句既有相似之处，也有区别。相似之处在于，都能返回紧跟在语句后面的那个表达式的值。区别在于每次遇到yield，函数暂停执行，下一次再从该位置继续向后执行，而return语句不具备位置记忆的功能。一个函数里面，只能执行一次（或者说一个）return语句，但是可以执行多次（或者说多个）yield表达式。正常函数只能返回一个值，因为只能执行一次return；Generator 函数可以返回一系列的值，因为可以有任意多个yield。

for...of循环，扩展运算符（...）、解构赋值和Array.from方法内部调用的，都是遍历器接口。这意味着，它们都可以将 Generator 函数返回的 Iterator 对象，作为参数

**function\*** numbers () {

**yield** 1

**yield** 2

**return** 3

**yield** 4}

*// 扩展运算符*[...numbers()] *// [1, 2]// Array.from 方法*Array.from(numbers()) *// [1, 2]// 解构赋值***let** [x, y] **=** numbers();x *// 1*y *// 2// for...of 循环***for** (**let** n **of** numbers()) {

console.log(n)}*// 1// 2*

### 敏捷开发的全部流程

每次迭代都必须依次完成以下五个步骤。

需求分析（requirements analysis）

设计（design）

编码（coding）

测试（testing）

部署和评估（deployment / evaluation）

每个迭代大约持续2~6周。