# 深入理解CSS的定位属性：

前言：

一个页面有多个box组成，元素的类型和display属性决定了box的类型。

normal flow：正常流（文档流）：块级元素（参与BFC）独占一行，垂直分布；行内元素（参与IFC）：水平方向上一个接一个排列。

BFC：block formatting contexts块级格式化上下文：

特点：

元素垂直排列；每个盒子的左边界margin-left紧靠容器的左边界border-left（因为垂直分布）

垂直边距由margin决定；同一个BFC的垂直margin会重叠（取最大值）；解决：每个盒子设为BFC

不与浮动元素重叠；计算高度时浮动元素也会参与计算

触发条件：body根元素、display：flex|inline-flex|inline-block|table-cell等、overflow：hidden|auto|scroll不设为visible、position：absolute|fixed

应用：清除浮动、左侧固定右侧自适应、防止垂直margin合并

IFC:inline formatting block

box水平排列，水平方向的margin border和padding保留，垂直无效

垂直方向的对齐方式不唯一：可以顶部、底部或者文字基线对齐（由vertical-align设置）

line box的高度由包含元素的高度(最高元素)决定

不可能有块级元素。当插入块级元素时（如p中插入div）会产生两个匿名块与div分隔开，即产生两个IFC，每个IFC对外表现为块级元素，与div垂直排列。

应用：(line-height设置行高，行间距)

**文本元素的垂直水平居中**：父元素：line-hight=height且text-align:center子元素inline-block外层形成IFC 同时vertical-middle：middle；line-height:normal

水平居中：子元素设置其为inline-block则会在外层产生IFC，(内层是BFC)通过父元素text-align：center则可以使其水平居中。

垂直居中：创建一个父元素：设置一个子元素撑开其高度然后设置其vertical-align:middle，父元素的其他**行内**元素则可以在此父元素下垂直居中。

# display属性

inline：将设置该元素的标签以内联元素即像普通文本一样插入文档流，不会自动换行或者设置宽度跟高度；

block: 将绑定该属性的标签以块的形式插入到文档流中，即会自动设置宽跟高并且会换行，说俗一点也就是我们口中的块，他会单独的占一行，可以设定margin 跟padding。

inline-block: 将绑定该元素的标签以inline内联样式插入到文档流中,然后它本身拥有block块的属性，可以设置宽度跟高度

none: 将该元素隐藏，不会再文档流中出现

table 此元素会作为块级表格来显示（类似table），表格前后带有换行符。   
inline-table 此元素会作为内联表格来显示（类似table），表格前后没有换行符。

# position属性

static：默认属性，即没有定位，TRBL不起作用

absolute：绝对定位，脱离文档流，不占空间。不配合TRBL时候相对于父级元素的左上角进行定位；使用TRBL时相对于**已经定位的父级元素**（除static属性）进行定位如果没有则相对于body进行定位，所以父级元素不推荐使用它。

fixed：固定定位。脱离文档流，不占空间。相对于浏览器窗口定位，位置固定，滚动条对其也无影响。

relative：相对定位。使用TRBL时相当于static，使用TRBL时则相对于自身位置进行偏移。

inherit：继承父级元素的定位属性。

# 原生轮播图：

实现一个轮播图还是不难的，大体思路： （为什么一定要过渡图片呢？）

先创建一个div，限定其宽度和高度，overflow：hidden，且设置其position为relative。

然后创建一个装图片的div，宽度为所有图片的总宽度，且设置其position为absolute，并且使其中的内容浮动，这样所有的图片就处于一行中。

然后为了实现无缝滚动，所以需要在首尾分别添加一张过渡图片。 先添加两个按钮， 使其可以手动轮播，然后只需要添加一个定时器使其朝一个方向自动轮播即可，因为用户有时需要查看详情，所以当鼠标进入时就clear定时器，滑出再定时播放。为了更好地用户体验，我们再下面添加了一排小圆点，用户可以清楚地知道现在所处的位置， 最后， 利用闭包使得用户可以直接通过点击小圆点切换图片。

# Bootstrap常见的面试题：

1.为什么使用bootstrap

bootstrap具有移动设备优先，浏览器支持良好，容易上手和响应式设计的特点。

2.为什么bootstrap栅格系统采用12列？  
原先是16列，但是设计者发现人们更喜欢奇数分割屏幕。12是2.3.4的倍数

3.对于各类尺寸的设备：bootstrap内置了一套响应式、移动设备优先的流式栅格系统随着屏幕设备或者视口viewport尺寸增加，系统自动分为最多12列；

|  |  |  |  |
| --- | --- | --- | --- |
| 手机<768px | 平板≥768px | 台式电脑≥992px | 大型台式电脑≥1200px |
| col-xs- | col-sm- | col-md- | col-lg- |

4.

|  |  |
| --- | --- |
| 设置响应式表格 | class="table-responsive" |
| 创建垂直表单 | 父form添加role="form"  标签和控件放在一个带有class="form-group"的div中  所有表单元素<input><textarea><select>添加class="form-control" |
| 创建水平表单 | 父form添加role="form-horizontal"  标签和控件放在一个带有class="form-group"的div中  所有表单元素<input><textarea><select>添加class="control-label" |
| 元素浮动 | class="pull left" class="pull right" |
| 清除浮动 | class="clearfix" |
| 除了屏幕阅读器外，其他设备隐藏 | class="sr-only" |
| 分页 | 分页：class="pagination"  翻页：class="pager" |
| 超大屏幕 | class="jumbotron".该组件可以增加标题的大小并增加更多的外边距 |

5.为了让bootstrap开发的网站对移动设备友好，确保适当的绘制和触屏缩放，需要在网页的head中添加viewport meta标签（<meta> 标签永远位于 head 元素内部。提供了有关页面的元信息）

<meta name="viewport" content="width=device-width,initial-scale=1">

width控制设备宽度。设置为device-width确保网页正确呈现在不同设备上

initial-scale=1确保网页加载时以1:1的比例呈现不会有任何缩放

通常情况下，maximum-scale=1.0 与 user-scalable=no 一起使用。这样禁用缩放功能后，用户只能滚动屏幕，就能让您的网站看上去更像原生应用的感觉，所以要加上以下代码

# 圣杯和双飞翼布局

圣杯布局和双飞翼布局基本上是一致的，都是两边固定宽度，中间自适应的三栏布局，其中，中间栏放到文档流前面，**保证先行渲染。**

|  |  |
| --- | --- |
| 圣杯 | 双飞翼 |
| 三栏float  中间宽度：width:100%  左右加上margin让其和中间栏并排  *不同在于解决”中间栏div内容不被遮挡“问题的思路不一样：* | |
| 三栏外层包裹层左右padding-left padding-right根据左右栏宽度设置内边距;  左右栏配合position:relative 以及RL移回左右位置（相对于自身移动） | 将中间栏里新建一个div 内容写在这个内部div中利用margin-left margin-right设置左右外边距把左右栏移走。（注：而外层那个middle其实宽度仍然是100%（重叠了left和right），但是middle-inner是实现了压缩的） |
| 利用相对定位 | 简洁 |

浏览器渲染HTML文件的过程？渲染过程总是先生成DOM树再等待css下载完成后生成CSS Tree结合生成Render Tree

1. 用户输入网址（假设是个html页面，并且是第一次访问），浏览器向服务器发出请求，服务器返回html文件；
2. 浏览器开始载入html代码，发现<head>标签内有一个<link>标签引用外部CSS文件；
3. 浏览器又发出CSS文件的请求，服务器返回这个CSS文件；
4. 浏览器继续载入html中<body>部分的代码，并且CSS文件已经拿到手了，可以开始渲染页面了；
5. 浏览器在代码中发现一个<img>标签引用了一张图片，向服务器发出请求。此时浏览器不会等到图片下载完，而是继续渲染后面 的代码；
6. 服务器返回图片文件，由于图片占用了一定面积，影响了后面段落的排布，因此浏览器需要回过头来重新渲染这部分代码；
7. 浏览器发现了一个包含一行Javascript代码的<script>标签，赶快运行它；
8. Javascript脚本执行了这条语句，它命令浏览器隐藏掉代码中的某个<div> （style.display=”none”）。杯具啊，突然就少了这么一个元素，浏览器不得不重新渲染这部分代码；
9. 终于等到了</html>的到来，浏览器泪流满面……
10. 等等，还没完，用户点了一下界面中的“换肤”按钮，Javascript让浏览器换了一下<link>标签的CSS路径；
11. 浏览器召集了在座的各位<div><span><ul><li>们，“大伙儿收拾收拾 行李，咱得重新来过……”，浏览器向服务器请求了新的CSS文件，重新渲染页面。

CSS不会阻塞DOM解析，但会阻塞页面渲染。JS阻塞DOM解析，（defer或者async属性，此时脚本下载的过程中是不会阻塞DOM解析的。）

# vue生命周期

注：所有的生命周期钩子自动绑定this上下文到实例中。所以不能使用箭头函数定义生命周期方法。

 Vue 实例从创建到销毁的过程，就是生命周期。从开始创建、初始化数据、编译模板、挂载Dom→渲染、更新→渲染、销毁等一系列过程，称之为 Vue 的生命周期。

beforeCreate:实例创建前：$el和data均未初始化

created：data已经初始化 但是$el属性尚不能用。可以发送http请求

beforeMount：实例挂载前，开始编译模板data中的数据和模板生成HTML

mounted：$el被挂载到实例上之后，~~用上面编译好的html内容替换el属性指向的DOM对象。~~完成模板中的html渲染到html页面中。此过程中进行ajax交互。

beforeUpdate:数据更新前。发生在virtual dom 打补丁之前。~~在服务器端渲染期间不会调用~~

updated：数据更新后。发生在virtual dom 打补丁之后。

beforeDestroy:销毁前，实例仍然可以用。可以移除事件监听器以及清除计时器等。~~在服务器端渲染期间不会调用~~

destroyed：vue实例销毁后，移除所有子组件还有事件监听器。~~在服务器端渲染期间不会调用~~

# ES5为数组定义的迭代方法(数组项的值，索引，数组对象本身)

|  |  |
| --- | --- |
| arr.filter() | 给数组每一项运行给定函数。返回**该函数会返回true的项**组成的数组（一般小于等于原数组长度） |
| arr.forEach() | 给数组每一项运行给定函数。没返回值 |
| arr.map() | 给数组每一项运行给定函数。返回**每次函数调用的结果**组成的数组（一般和原数组长度一样） |
| arr.every() arr.some() | 给数组每一项运行给定函数。如果该函数对**任**一项都返回true则返回true  给数组每一项运行给定函数。如果该函数对**某**一项都返回true则返回true |

**缩小数组方法：**

arr.reduce()从数组第一项开始逐个遍历到最后 arr.reduceRight()从数组最后一项开始逐个遍历到第一项

**arr.sort()**

默认情况下，sort()按升序排列数组项，比较的永远是字符串，因为其会调用每个数组项的toString ()方法比较得到的字符串。因而有时得到的结果和数字排序并不一样。因此它接受一个函数作为参数：每次从数组中（倒序）挑选两个数，如果返回0，1则两个数不交换位置 返回小于0的数则交换位置。

**HTTP请求方法合集**

|  |  |  |  |
| --- | --- | --- | --- |
| http1.0 | get | 获取资源 | 请求数据放在url中，限制在2k左右 |
| head | 获取报文首部 | 和get相同，只是返回时不返回主体只返回响应的头 |
| post | 传输实体文本 | 传输数据放在请求体中，数量不限 |
| http1.1 | put | 传输文件，然后保存在服务器指定位置 |  |
| delete | 让服务器删除某个资源 |  |
| trace | 追踪路径，回显服务器收到的请求 | 主要用于测试判断 |
| options | 询问服务器可以提交哪些请求方法 |  |
| connection | 要求用隧道协议进行TCP通信 | 主要使用SSL和TLS（传输层安全）协议把通信协议加密后经网络隧道传输 |

# promise

将异步操作以同步操作的流程表达出来，避免了层层嵌套的回调函数。

含义：简单说就是一个容器，里面包含着一个未来可能结束的事件的结果

特点：

1.对象状态不受外界影响：pending进行中 fulfilled成功 rejected失败。只有异步操作的结果可以决定当前是哪一种状态其他操作不可以---promise的由来

2.状态一经改变就无法再变：只有pending-->fulfilled 和pending-->rejected两种可能，只要状态发生改变就不会再变，任何时候都可以得到这个结果。

Promise对象是一个构造函数，接收一个函数作为参数...新建后立即执行

Promise实例生成后可以用then方法分别指定resolved和rejected状态的回调函数

注：

1.**立即resolved的Promise总是在本轮事件循环的末尾执行，总是晚于本轮事件循环的同步任务**

2.then方法返回的是一个全新的Promise实例所以可以进行链式调用，指定一组按次序调用的回调函数。

方法介绍

|  |  |
| --- | --- |
| Promise.catch() | 就是.then(null,function(error){})建议使用这种方法的原因是：catch还可以捕捉前面的then方法执行中的错误；如果都写在一个then中显然是不会捕获 |
| Promise.all() | 将多个Promise实例包装成一个新的Promise实例，只有这些Promise都resolved才会resolved只要有一个rejected就rejected。但是实例里面如果有catch 那么最后整个实例的catch不会捕获到错误所以是resolved状态 |
| Promise.race() | 和all方法类似但是只要有一个实例状态改变，最终状态就用它 |
| Promise.resolve()可以将一个对象变成Promise对象（**立即resolved的Promise总是在本轮事件循环的末尾执行，总是晚于本轮事件循环的同步任务**） | 参数是一个Promise实例，不做修改原封不动的返回实例 |
| 参数是一个具有then方法的对象，会将该对象转变成Promise对象，然后立即执行then方法 |
| 不是对象或者不含then方法的对象。返回一个resolved状态的新Promise对象 |
| 不带参数返回一个resolved状态的Promise对象 |
| Promise.reject() | 返回状态为rejected状态的Promise实例 |

promise缺点：

* 无法取消Promise，一旦新建它就会执行，无法中途取消
* 如果不设置回调函数，Promise内部抛出的错误，不会反应到外部
* 当处于Pending状态时，无法得知目前进展到哪一个阶段（刚刚开始还是即将完成）

# Iterator

概念：Iterator是一种机制，为不同的数据结构提供统一的简单的访问接口；Iterator主要是供for..of使用

注：es6之后我们有了四种表示集合的数据结构：Array,Object,Set（类似于数组）,Map（类似于对象）

使用：每次调用指针对象的next方法都返回数据结构的当前成员的值value和一个表示是否遍历结束的布尔值done。

ES6默认的Iterator接口部署在数据结构的Symbol.iterator属性上，只要一个数据结构具有这个属性就认为是可遍历的。（Array Map Set String 函数的arguments对象 NodeList对象 TypedArray对象）。调用Iterator接口即调用Symbol.iterator方法：解构赋值、扩展运算符、yield\*。由于数组的遍历会调用遍历器接口所以任何接收数组作为参数的场合都调用了Iterator接口 像 for ...of Array.from Map() WeakMap() Set() WeakSet() Promise.all() Promise.race()

# Genenrator函数

是Iterator接口（或者说实现Symbol.iterator方法）的一种方式。G函数返回一个部署了Iterator接口的遍历器对象。每次调用遍历器对象的next方法，就会返回一个有着value和done两个属性的对象。value属性表示当前的内部状态的值，是yield语句后面那个表达式的值；done属性是一个布尔值，表示是否遍历结束。

（1）遇到yield语句，就暂停执行后面的操作，并将紧跟在yield后面的那个表达式的值，作为返回的对象的value属性值；

（2）下一次调用next方法时，再继续往下执行，直到遇到下一个yield语句；

（3）如果没有再遇到新的yield语句，就一直运行到函数结束，直到return语句为止，并将return语句后面的表达式的值，作为返回的对象的value属性值；

（4）如果该函数没有return语句，则返回的对象的value属性值为undefined。

需要注意的是，yield语句后面的表达式，只有当调用next方法、内部指针指向该语句时才会执行，因此等于为JavaScript提供了手动的“惰性求值”（Lazy Evaluation）的语法功能。

* yield句本身没有返回值（返回undefined）。next方法可以带一个参数，该参数就会被当作上一个yield语句的返回值。
* for of循环可以自动遍历G函数yield后面的那个表达式的值，不需要调用next方法。
* yield\*一个可迭代对象，就相当于把这个可迭代对象的所有迭代值分次 yield 出去。

# Async函数

generator函数的语法糖。\*-->async yield-->await 实现原理即：将G函数和自动执行器包在一个函数里。

优点:

1.内置执行器不用调用next方法

2.更好的语义说明，async表示异步操作 await表示后面的表达式需要等待结果

3.更广的适用性：await后面可以是原始类型值或promise对象。await命令后是一个Promise对象。如果不是，会被转成一个立即resolve的Promise对象。

4.返回promise对象，可以用then方法添加回调函数。

# JavaScript设计模式

原型模式：

是创建对象的一种模式，也就是说通过Object.create对对象进行复制得到新的对象。

单例模式

发布者-订阅者模式

# websocket

由短轮询polling（耗流量消耗CPU利用率，大部分请求是无用功）-->长轮询（假设服务端数据更新速度快，服务器发给客户端后必须等待下一个get请求的到来，这个等待时间对于用户来说其实是不能接受的）-->websocket

1. 服务器可以主动向客户端推送信息，客户端也可以主动向服务器发送信息，是真正的双向平等对话，属于服务器推送技术的一种。
2. 建立在 TCP 协议之上，服务器端的实现比较容易。通过单个TCP连接提供全双工（双向通信）通信信道的计算机通信协议。
3. 与 HTTP 协议有着良好的兼容性。默认端口也是 80 和 443 ，并且握手阶段采用 HTTP 协议，因此握手时不容易屏蔽，能通过各种 HTTP 代理服务器。复用了HTTP握手通道：具体指的是客户端通过http请求与websocket服务端协商升级协议。协议升级完成后后续的数据的交换遵照websocket协议
4. 数据格式比较轻量，性能开销小，通信高效。
5. 可以发送文本，也可以发送二进制数据。
6. 没有同源限制，客户端可以与任意服务器通信。
7. 协议标识符是ws（如果加密，则为wss），服务器网址就是 URL。

Ajax轮询：在特定时间内由浏览器向服务器发送HTTP请求，服务器将数据返回给客户端。缺点：http带有较长的头部，浪费带宽消耗流量，消耗CPU利用率

websocket：单个TCP连接

微信二维码扫描登录：

实际上是将URL转换成二维码。

通过客户端扫码后打开了URL，同时URL中包含本次对话唯一的ID类似于sessionID，通过这个ID就能定向的将确认结果反馈到网页上。

过程：扫码--微信提取当前微信APP的登录信息并将上面的ID发给服务器--服务器确认两件事：一确认登录到微信客户端的用户是验证过的二是通过ID服务器知道把一中的结果反馈推送到哪个网页--假设验证用户合法那么就可以用websocket推送到指定网页实现登录。然后可能是推送给客户端登录成功之类的客户端可以进行跳转到商户页面。

# vue音乐播放器

1.recommend.vue今日推荐页面实现的布局：左侧固定（小图片），右侧自适应（专辑信息）

2.获取数据，使用axios反向代理：在config的index.js配置proxyTable实现跨域获取数据。可以讲讲反向代理

3.使用swiper实现首页轮播图利用了它的自动播放autoplay以及分页pagination，直接阅读官方文档。遇到问题：一开始因为没有接口，想自己存三张图但是这样其实很死板，后来参考了别人的代码因为QQ音乐是通过jsonp跨域拿数据的所以我也在项目里引入了jsonp：

第二个参数：与后端约定请求的字段名称，默认是callback对其进行了一个修改，自己定义了一个函数返回一个promise实例。因为qq音乐的数据请求时需要在携带很多参数所以做了拼接。

<swiper :options="swiperOption" >

<!-- slides -->

<swiper-slide>

4.图片懒加载：vue-lazyload 使用<v-lazy>即可

5.开发歌手列表的特色：左右联动实现类似通讯录的效果：首先把乱序的数据聚合起来然后对首字母属性进行排序出ABCD 利用a.title.charCodeAt(0) sort排序。左右联动的实现就是通过watch监听滚动的位置，同时根据位置计算出此位置在哪个group空间计算出currentIndex和右侧的快速入口中：动态给当前index的li标签添加class类名，实现同步高亮。

6.歌手详情页的动画效果：利用vue内置的transition组件实现歌手列表向歌手详情页平滑过渡进入离开。translate3d(100%, 0, 0)开启GPU加速横向撑开

7.vuex解决路由参数传递：使用vuex的插件plugins: debug ? [createLogger()] : []观察每次修改state时生成的日志

8.异步加载歌词组件：明天看

模块化：

# webpack

什么是模块化？就是将一个复杂的系统分解成多个模块方便编码

为什么出现模块化：命名空间组织代码：命名冲突；无法合理管理依赖和版本；无法方便的控制依赖的加载顺序。

是什么？

webpack其实就是一个JavaScript应用程序的静态模块打包器。

优点：

* 专注于处理模块化的项目，能做到开箱即用一步到位；
* 通过 Plugin 扩展，完整好用又不失灵活；
* 使用场景不仅限于 Web 开发；
* 社区庞大活跃，经常引入紧跟时代发展的新特性，能为大多数场景找到已有的开源扩展；
* 良好的开发体验。

作用：

webpack会将项目的资源文件当成一个一个模块，模块之间会有依赖关系，webpack将会对这些有依赖关系的文件进行处理，让浏览器能够识别，最后将应用程序需要的每个模块打包成一个或者多个bundle。

简介：webpack一个打包模块化JavaScript的工具，从入口文件出发，识别出源码中的模块化导入语句，递归的找出入口文件的所有依赖，将入口和所有依赖打包到output到一个单独的文件或者多个文件中。

**webpack-cli**:命令行工具,用于在命令行中运行webpack

打包模式：  
Package.json文件下添加scripts：生产模式（代码压缩，量较少）和开发模式；(webpack4

webpack.config.js（以下简称config）配置（均写在module.exports）：

* 1.入口：一个字符串、数组，或者一个对象。如果entry是一个object，就可能会出现多个 Chunk，这时 Chunk 的名称是object键值对里键的名称。

2.出口：一个对象：包含一系列配置项。output.chunkFilename配置无入口的 Chunk 在输出时的文件名称。主要用于使用import('path/to/module')动态加载的时候

均可以多文件：入口：在你想要多个依赖文件一起注入，并且将它们的依赖导向到一个“chunk”时，传入数组的方式就很有用。

3.loader： loader 让 webpack 能够去处理那些非 JavaScript 文件（webpack 自身只理解 JavaScript）。**loader 可以将所有类型的文件转换为 webpack 能够处理的有效模块**，然后你就可以利用 webpack 的打包能力，对它们进行处理。

在webpack.config.js文件里配置module中的rules：

test 、include、exclude属性，用于标识出应该被对应的 loader 进行转换的某个或某些文件。

use 属性，表示进行转换时，应该使用哪个 loader。是一个由 Loader 名称组成的数组。Loader 的执行顺序是由后到前的；

**如何编译less文件：**

安装less-loader和less

{

test:/\.less$/,

use:['style-loader','css-loader','less-loader']

},

**编译sass文件**

安装sass-loader和node-sass

{

test:/\.scss$/,

use:['style-loader','css-loader','sass-loader']

},

**PostCSS处理浏览器前缀：**

安装postcss-loader和autoprefixer

{

test:/\.css$/,

use:['style-loader','css-loader',{

loader:'postcss-loader',

options:{

plugins:[require("autoprefixer")()],

sourceMap:true

}

}]

},

**文件处理：** {

test:/\.(png|jpg|gif|jpeg)$/,

use:[{

loader:'file-loader',

options:{

name:'[hash]wangou.jpg',

outputPath:'./img'

}

}]

},

{

test:/\.(ttf|woff|woff2|eot|svg)$/,

use:[{

loader:'file-loader',

options:{

outputPath:'font/'

}

}]

},

配置options:

name：为你的文件配置自定义文件名模板（默认值[hash].[ext]）

context：配置自定义文件的上下文，默认为 webpack.config.js

publicPath：为你的文件配置自定义 public 发布目录

outputPath：为你的文件配置自定义 output 输出目录

4.plugins: webpack是通过plugins属性来配置需要使用的插件列表的。plugins属性是一个数组，里面的每一项都是插件的一个实例，在实例化一个组件时可以通过构造函数传入这个组件支持的配置属性。

5.resolve：配置webpack如何寻找模块对应的文件。resolve:{ extensions: ['.js', '.json'],

alias:"common":resolve('src/common')}

resolve.alias配置项通过别名来把原导入路径映射成一个新的导入路径。

resolve.extensions用于配置在尝试过程中用到的后缀列表

**HtmlWebpackPlugin**会自动为你生成一个HTML文件，根据指定的index.html模板生成对应的 html 文件。

const HtmlWebpackPlugin = require('html-webpack-plugin');

new HtmlWebpackPlugin({

template:"./public/index.html",

filename:"webpack.html",

minify:{

minimize:true,

removeComments:true,

removeAttributeQuotes:true,

collapseWhitespace:true,

minifyCSS:true,

minifyJS:true,

removeEmptyElements:true

},

hash:true

}),

**ExtractTextPlugin:**将所有的入口 chunk(entry chunks)中引用的 css，移动到独立分离的 CSS 文件

增加了一步：rules配置fallback:编译后用什么loader来提取css文件

{

test:/\.css$/,

use:ExtractTextPlugin.extract({

fallback:"style-loader",

use:['css-loader',{

loader:'postcss-loader',

options:{

plugins:[require("autoprefixer")()]

}

}]

})

},

**optimize-css-assets-webpack-plugin：**

引入，配置

assetNameRegExp: 正则表达式，用于匹配需要优化或者压缩的资源名。默认值是 /\.css$/g

cssProcessor: 用于压缩和优化CSS 的处理器，默认是 cssnano.

cssProcessorPluginOptions:传递给cssProcessor的插件选项，默认为{}

canPrint:表示插件能够在console中打印信息，默认值是true

discardComments:去除注释

webpack-dev-server：  
webpack-dev-server是webpack官方提供的一个小型Express服务器。用于服务网页请求，同时会帮助启动 Webpack ，并接收 Webpack 发出的文件更变信号，通过 WebSocket 协议自动刷新网页做到实时预览。

主要功能：为静态文件提供服务；自动刷新和热替换(HMR)

实时预览：

通过 DevServer 启动的 Webpack 会**开启监听模式**，当发生变化时重新执行完构建后通知 DevServer。 DevServer 会让 Webpack 在构建出的 JavaScript 代码里注入一个代理客户端用于控制网页，网页和 DevServer 之间**通过 WebSocket 协议通信**， 以方便 DevServer 主动向客户端发送命令。 DevServer 在收到来自\*\* Webpack 的文件变化\*\*通知时通过 **注入的客户端控制网页刷新**。

HMR：

模块热替换(HMR - Hot Module Replacement)功能会在应用程序运行过程中替换、添加或删除模块，而无需重新加载整个页面。

配置config中的module.exports devServer: hot:true

配置Plugins:

new webpack.NamedModulesPlugin(),

new webpack.HotModuleReplacementPlugin()

NamedModulesPlugin：当开启 [HMR](https://github.com/docschina/webpack.js.org/blob/cn/guides/hot-module-replacement) 的时候使用该插件会显示模块的相对路径

**第三方JS库引入**：

webpack.ProvidePlugin参数是键值对形式，键就是我们项目中使用的变量名，值就是键所指向的库。webpack.ProvidePlugin会先从npm安装的包中查找是否有符合的库。

如果webpack配置了resolve.alias选项（理解成“别名”），那么webpack.ProvidePlugin就会顺着设置的路径一直找下去

编译ES6：

目前，ES6（ES2015）这样的语法已经得到很大规模的应用，它具有更加简洁、功能更加强大的特点，实际项目中很可能会使用采用了ES6语法的模块，但浏览器对于ES6语法的支持并不完善。为了实现兼容，就需要使用转换工具对ES6语法转换为ES5语法，babel就是最常用的一个工具

babel转化语法所需依赖项：

babel-loader： 负责 es6 语法转化

babel-core： babel核心包

babel-preset-env：告诉babel使用哪种转码规则进行文件处理

配置config文件：

{

test:/\.js$/,

exclude:/node\_modules/,

use:'babel-loader'

},

根目录新建.babelrc文件配置转换规则

{

"presets":["@babel/preset-env"]

}

sourcemap

是为了解决实际运行代码（打包后的）出现问题时无法定位到开发环境中的源代码的问题。

devtool配置 Webpack 如何生成 Source Map，默认值是false即不生成 Source Map，

module.export = {

devtool: 'source-map'

}

区分开发环境和生产环境：

在这里我们可以将webpack.config.js拆分为三个文件，分别是webpack.common.conf.js、webpack.dev.conf.js和webpack.prod.conf.js。

webpack.common.conf.js是放一些我们公用的配置，比如入口entry、出口output、常用loader以及插件等。

webpack.dev.conf.js是在开发环境上的配置，比如devServer配置、模块热替换等方便开发的配置

webpack.prod.conf.js是在生产环境上的配置，比如提取分离css、压缩css和js等

需要修改package.json文件中的script

"scripts": {

"dev": "webpack --mode development --config ./config/webpack.dev.conf.js",

"build": "webpack --mode production --config ./config/webpack.prod.conf.js",

"start": "webpack-dev-server --mode development --config ./config/webpack.dev.conf.js"

}, --config可以指定使用的配置文件