* **如何做RN在安卓和IOS端的适配**
* **RN为什么能在原生中绘制成原生组件（bundle.js）**
* **native提供了什么能力给RN**
* **RN和原生通信**
* **RN的原理，为什么可以同时在安卓和IOS端运行**
* **RN如何调用原生的一些功能**
* **介绍RN的缺点**
* **RN有没有做热加载**
* **RN遇到的兼容性问题**
* **RN如何实现一个原生的组件**
* **RN混原生和原生混RN有什么不同**
* **为什么3大框架出现以后就出现很多native（RN）框架（虚拟DOM）**
* **介绍koa2**

Koa 是一个新的 web 框架，由 Express 幕后的原班人马打造， 致力于成为 web 应用和 API 开发领域中的一个更小、更富有表现力、更健壮的基石。 通过利用 async 函数，Koa 帮你丢弃回调函数，并有力地增强错误处理。 Koa 并没有捆绑任何中间件， 而是提供了一套优雅的方法，帮助您快速而愉快地编写服务端应用程序。

* **koa原理，为什么要用koa(express和koa对比)**

koa采用async+await异步 解决了callback地狱。koa中间件模型更像洋葱形。当当前中间件await next（）将处理逻辑交给下个中间件后，当下个中间件的代码执行完毕，再回到当前中间件执行后续逻辑。koa更将小巧简洁，内部未内置任何中间件，也未自带路由功能。

* **使用过的koa2中间件**

koa-router：类似express的路由功能

koa-body：解析请求体

koa-session：session回话存储

koa-static：静态资源管理，类似于espress自带的static中间件

* **koa-body原理**

koa-body返回一个ansyc函数，监听request的data和end事件 异步获取请求体的数据，添加到ctx.request.body上。成功后调用await next()执行下个中间件。

* **有没有涉及到Cluster**

首先，Cluster会创建一个master，然后根据你指定的数量复制出多个server app（也被称之为工作线程）。它通过[IPC](https://en.wikipedia.org/wiki/Inter-process_communication)（Inter-Process Communication，进程间通信）通道与工作线程之间进行通信，并使用内置的负载均衡来更好地处理线程之间的压力，该负载均衡使用了[Round-robin](https://en.wikipedia.org/wiki/Round-robin_scheduling)算法（也被称之为循环算法）。

* **介绍pm2**

PM2是node进程管理工具，可以利用它来简化很多node应用管理的繁琐任务，如性能监控、自动重启、负载均衡等，而且使用非常简单

* **master挂了的话pm2怎么处理**
* **服务端怎么做统一的状态处理**

数据库，redis ?

* **npm2和npm3+有什么区别**

npm2所有项目依赖是嵌套关系，而npm3为了改进嵌套过多、套路过深的情况，会将所有依赖放在第二层依赖中（所有依赖只嵌套一次，彼此平行，也就是平铺的结构）,都平行的放在node\_module目录下，除非两个包引入不同版本的第三方包，则后一个版本的第三方包放在包路径的node\_module下

* **如何和MySQL进行通信**

安装mysql模块，创建mysql实例，spl操作。也可使用knexjs或sequelize这样的

**ORM（**对象关系映射**）**操作数据库

* **介绍下Promise，内部实现**

每个promisre实例内部有三种状态：pending（进行中）、fulfilled（已成功）和rejected（已失败）。状态一旦从pending->fulfilled或pending->rejected后将不可改变。Promise构造函数接受函数参数，函数可获得Promise实例的resolve和reject方法用于改变状态。promise.then()方法，接受函数参数并存储，待promise实例状态改变之后执行，promise的then每次会返回新的Promise实例，这为promise的链式调用提供了基础，新得到的promise实例是对then方法接受的函数参数执行结果的包装。

* **介绍二叉搜索树的特点**
* **react介绍Suspense**

Suspense:解决代码分割和异步数据获取，在之前中可能需要didMount中请求，setState修改，然后重新render，Suspense利用了componentDidCatch新生命周期(可获取err)，不用复杂的获取数据流程。

* **项目中树的使用场景以及了解**

遍历dom树，

node获取文件名称，判断文件名是否重复

这两处设涉及到了树的遍历（深度遍历和广度遍历）

* **前端怎么做单元测试**
* **小程序里面开页面最多多少**
* **Emit事件怎么发，需要引入什么**
* **如何配置把js、css、html单独打包成一个文**
* **对React看法，有没有遇到一些坑**
* **介绍下React的Filber架构**

react的更新策略可以分为两个阶段 1：计算Virtual DOM，和之间的虚拟dom树对比，得到需要修改的dom。2：执行修改，并依次冒泡触发各组件render之后的生命周期。react之前的调度是stack reconciler（栈调用？），从父组件开始，执行父组件render之前的生命周期，执行子组件render之前的生命周期，以此类推，然后递归到最底层的子组件，获得新的虚拟dom，对比虚拟dom找到差异，更新dom，然后再由深往浅调用各组件render之后的生命周期。这整个过程就像是一个潜泳的游泳选手，这个调用占据了浏览器的主要内存，其他的操作都被阻塞无法执行（例如输入框的输入，用户的点击等交互）。所以在react 16.x中，facebook采用filber架构，异步渲染来优化这个问题。fiber将原来Virtual DOM的更新任务拆分成一个个小的任务，在各个组件中render之前的生命阶段是可以暂停的，react遇到更高优先级的任务会暂停组件的Virtual DOM计算，去执行更高优先级的任务，等任务执行完毕再继续执行组件的更新。

* **用户体验做过什么优化**
* **什么是单页项目**
* **随机值存在一样的情况，如何避免**
* **垃圾回收时栈和堆的区别**
* **服务端渲染SSR**

解决两个问题1:首页白屏2：SEO困难

基本思路：服务端一套代码，客户端一套代码，首屏请求服务端获取首屏渲染的html返回客户端，客户端代码复用首屏html，之后有客户端代码控制用户交互和路由跳转。

服务端和客户端不同：

1:服务端代码无需异步加载模块，因为没有网络请求，而且异步加载的原理是动态添加script标签在node环境不适用。

2:服务端指定构建环境为node(target)，webpack会忽略fs等node内置模块

3：指定模块采用commonjs规范

4：忽略css、图片、svg等资源，让其勿打包进入工程

5：服务端采用的router模式不同 不可采用hashRouter和browserRouter 因为他们都是根据浏览器的路径渲染。react采用MemeryRouter 该路由模式根据传入的path渲染。

6：服务端的数据预获取，http模块获取预获取数据后可插入html模板一起返回客户端，这要求了在编写代码时需要手动配置各个路径需要获取的预获取数据，对团队配合要求较高。

* **如何解决props层级过深的问题**

redux全局状态管理

Context，const {Provider, Consumer} = React.createContext(defaultValue);

Provider：件可传递value修改defaultValue，Consumer：内部组件根据value渲染。

* **介绍ATS**

ATS是抽象语法树，是将代码逐字解析成树状对象的形式。是语言转化、代码检查、代码自动补全的基础。babel编译es6到es5的核心也是利用ATS。

以es6转es5为例 babel先将es6代码转化为ATS，然后根据相应规则将es6规则的ATS转化为es5的ATS，最后将es5的ATS转化为es5语法的代码

* **React15/16.x的区别**

生命周期的不同

增加了hooks 函数组件增加了state状态

引入了fiber架构

* **如何配置React-Router**

在v4之前 react-router是集中式管理的 页面中所有的router都在一个文件重定义好了 不允许出现在组件中嵌套的行为。在V4中react-router实现了拆分管理，可以在组件中定于子路由，使得路由的定义更灵活化动态化。比如在V4之前如果两个子路由有公用的模块，只能通过高阶组件和父路由的component 调用props.children来实现。

其他细节：

v4中路由默认不是精确匹配。比如‘/add’可以命中‘/’和‘/add’两个路由。

V4实现精确匹配可以添加exact属性

V4中可以通过添加<Switch>组件只渲染匹配到的第一个路由

V4去除了IndexRouter组件

* **路由的动态加载模块**

路由的动态加载首先就要提到webpack自身的动态加载功能 在webpack中通过import(/\*webpackChunkName:name\*/’filepath’)来实现按需加载，它返回一个promise异步加载模块加载完毕之后变为resolve状态。实际使用中根据不同的框架有不同的调整，在react可以用高阶组件接受异步动态组件，等待组件加载完毕返回动态组件。在Vue中 vue可接受异步组件（一个函数返回import后的promise）

* **介绍路由的history**

react-router的history:history分为browser、hash、memory三种，分别通过h5的history、url hash值、缓存来处理路由跳转，其中browser利用H5 history特性（可以不刷新的修改页面路径）history.pushState(),history.replaceState()。但pushState和replaceState没有监听事件，无法像hash的onhashchange 无法捕捉事件，更新页面、组件；所以引入了Link这样的内置组件，点击时捕获路径更新对应页面（这也是a标签和Link的不同）；在浏览器的前进后退操作中监听浏览器事件onpopstate，获取history.state，更新页面。

* **介绍Redux数据流的流程**

redux：是一个应用状态管理方案，一般在引用中只会创建一个store对象实例，由createStore()函数创建，该方法返回一个store对象 该对象包含主要的常用方法为：getStore()---获取当前store状态state、dispath()--接受action调用reducer触发store状态更新、subscribe()--设置监听函数，repalaceReducer()--更换reducer;createStore()接受三个参数 ：reducer，初始化状态，和中间件处理函数，也就是常用的applyMiddleware()处理后的中间件，每个中间件会过滤处理一遍action将自己处理完毕后的中间件交给下一个中间件处理，直到成为符合规范的reducer能处理的action。redux的数据流可以整理为 action->中间件->dispatch->reducer

* **Redux如何实现多个组件之间的通信，多个组件使用相同状态如何进行管理**

组件A触发store状态更新 组件B监听store对应的状态变化，具体是由react-redux提供的connect函数实现，它接受需要监听的属性以及绑定触发更新的action到props，装饰需要引入redux的组件类。

* **使用过的Redux中间件**

redux-chunk:处理函数类型的中间件

redux-logger:打印日志

redux-promise:处理promise类型的action

* **如何解决跨域的问题**

1:jsonp 利用浏览器js脚本没有跨域限制的原理 将后端接口作为js外链地址插入页面，传递后端接受数据的回调callback函数 操作得到的数据

2：cros跨域 需要后端设置Access-Control-Allow-Origin

3：反向代理

4：postMessage跨域iframe.contentWindow.postMessage(JSON.stringify(data)，'http://www.domain2.com')向iframe发送消息；window.addEventListener(‘message’,()=>{})监听消息；

window.parent.postMessage(JSON.stringify(data), 'http://www.domain1.com');iframe页面发送消息

* **常见Http请求头**

Host:请求的主机地址和端口 47.97.5.188

User-Agent:客户端信息

Accept:客户端接受的资源类型 “/”

Accept-Language：客户端希望接受的资源语言 “zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK”

Referer：告诉服务器请求来自哪里 “localhost:8080/a”

Date：客户端请求时间 Tue, 18 Jul 12:15:02 GMT

Connection：和客户端建立tcp连接后是否关闭 Keep-Alive/Close

If-Modified-Since：资源上次修改的时间 用于和服务器对比判断缓存是否失效 Thu, 14 Mar 2019 07:22:27 GMT

If-None-Match：和If-Modified-Since相似 返回服务器传来的Etag标志

常见响应头：

协议 状态码：http/1.1 200

Content-Type：服务端返回的资源类型和编码格式 text/html; charset=GB2312

Cache-Control：强缓存

Connection：同请求头

Date: 服务端响应客户端的时间 Tue, 18 Jul 2017 12:18:03 GMT

Server:web服务器名称 nginx/1.13.6

Set-Cookie：设置cookie

Last-Modified：资源最后修改时间

Etag：和Last-Modified功能相似 不过是个标签

Access-Control-Allow-Origin：指定哪些网站可以跨域

Content-Disposition：指定文件下载名和弹出下载文件

Location：重定向并 常配合状态码302一起

* **移动端适配1px的问题**

媒体查询获取设备像素比，@media (-webkit-device-pixel-ratio: 3){}，设定“border”为 ：after伪元素，transform转换 “border”高度。

flexible.js布局但只解决了ios手机的问题 安卓因其设备像素比各个手机难以统一 故未做适配。

* **介绍flex布局**

flex布局也叫弹性布局，在解决了传统css一些基于display、position、float属性的痛点。它分别父元素和子元素。

父元素的属性：flex-direction:定义主轴的方向（默认是横向）

justify-content:子元素在横轴上如何分布（靠右，靠左，居中，两端，等间距）

align-item:定义单排的元素在纵轴上的对其方式（顶端，居中，下端）

align-content:定义多排子元素在纵轴的对其方式（）

flex-wrap:定义一排子元素放不下后是否换行

flex-flow:flex-direction和flex-wrap的组合

子元素的属性：order:排序权重越小越靠前

flex-grow:在排列还有剩余空间时（一行有空隙）设定放大的倍数 且会撑满整个行空间。比如父宽1000，子宽100\*4个，如果设置其中一个的flex-grow:1，那么他将撑满所有剩余空间 宽达到700，其他不变，如果另一个设置了flex-grow为0.25那么子元素宽为160 640 100 100

flex-shrink:如果项目一行放不下 缩小比例，默认为1，放不下都缩小，为0则不会缩小

align-self:设置单个子元素和其他元素不一样的纵轴对其方式 覆盖父元素的align-item属性

* **其他css方式设置垂直居中**

1：设置position:absolute, top,left,right,bottom全都设置为0

2：设置margin 50%，设置子元素的transform:translate(-50%,-50%)

3：父元素设置伪元素 ::after{content:’’,vertical-align:middle,height:100%}

子元素设置vertical-align 中间对其。

* **居中为什么要使用transform（**为什么不使用marginLeft/Top）

1：在未知居中元素的宽高情况下 marginLeft/Top做不到居中

2：marginLeft/Top的修改会触发浏览器的重绘和回流 而transform不会

* **使用过webpack里面哪些plugin和loader**

loader:

style-loader:将计算好的样式插入页面

css-loader:允许css模块化

less-loader/sass-loader

postcss-loader:css兼容性前缀

babel-loader:bable编译

url-loader:处理图片

ignore-loader:不处理，忽略所引入的文件 比如在node环境中忽略css 忽略img资源

plugins:

html-webpack-plugin:根据模板生成html入口

webpack.HotModuleReplacementPlugin：热模块替换

clean-webpack-plugin：清空文件夹

webpack.optimize.CommonsChunkPlugin：提取公共代码

webpack.DefinePlugin:定义环境变量

extract-text-webpack-plugin：抽取css到单独文件中

webpack.DllPlugin:构建动态链接库文件，包含大量复用模块的动态链接库只需要编译一次，在之后的构建过程中被动态链接库包含的模块将不会在重新编译，而是直接使用动态链接库中的代码。

* **webpack里面的插件是怎么实现的**

一个插件是一个类 或者构造函数，实例化时接受用户的参数，然后调用实例的apply方法，apply接受Compiler（可以简单地把它理解为 Webpack 实例）

然后给Compiler添加监听事件compiler.plugin('event-name',function(params) {

…

});

event-name包括webpack各个生命周期的事件 比如文件将要输出时-‘emit’ ，function可以拿到Compilation 并通过compilation.assets获取到输出的文件列表，并修改输出文件；也可以通过监听webpack文件输出完成是end事件，上传打包好的文件至cdn等。

（Compiler 代表了整个 Webpack 从启动到关闭的生命周期，而 Compilation 只是代表了一次新的编译。）

* **dev-server是怎么跑起来**

webpack-dev-server 是express+webpack-dev-middleware 两者区别是webpack-dev-server是封装好的 除了webpack.config.js配置 很难做定制化的设置，但webpack-dev-middleware 并未实现webpack-dev-server的模块热替换功能 所以需要额外引入webpack-hot-middleware。这两个中间件加上express就可以定制更灵活化的dev-server

* **项目优化**

在开发环境： 1：webpack.DllPlugin:构建动态链接库文件提高编译速度，

2：使用HappyPack插件，用多线程处理loader处理文件的过程

3：热模块替换

在生产环境：抽取公共代码：webpack.optimize.commonsChunkPlugin

代码按需加载

设置缓存runtime

chunk上传cdn 并设置缓存

雪碧图、小图尽量使用base64或svg格式、减少请求量、

* **抽取公共文件是怎么配置的**

抽取公共代码，在webpack4之前webpack自带了抽取公共代码的插件 webpack.optimize.commonsChunkPlugin，配置公共库名称，抽取缓存runtime。在webpack4中commonChunkPlugin已经被舍弃，有单独的optimize配置项（splitChunk,runtimeChunk,minChunk-最小被共享数，）

* **介绍redux，主要解决什么问题**

一个应用全局状态管理解决方案，主要用来解决组件间公共状态的管理

* **文件上传如何做断点续传**

文件断线上传的核心是分割文件 和生成文件唯一标识。核心思想是：用户选取大文件，前端获取到file，根据文件名称、文件大小、文件最后修改时间、浏览器唯一标志生成一个唯一标志（md5），利用File对象的slice函数分割大文件，循环向后端传输（包含唯一标志）；传输打断之后重新开始传输时，先和后端确认文件是否已经上传及长传进度（根据文件唯一标志），然后由上次进度开始继续上传

* **表单可以跨域吗**
* **promise、async有什么区别**

promise是es6定义的异步解决方案，async是es7提出的异步解决方案 他是generator加promise的语法糖。 async比较promise使用起来更加简洁，代码可读性更佳好，特别是在promise需要链式调用的问题上。

* **搜索请求如何处理（防抖）**

防抖是高频延时事件只会触发一次 且最后一次会覆盖前面的(在上一次事件还未处理完毕时高频事件触发了则取消上一次事件执行，取最后一次执行)。此处可以给搜索输入框绑定输入事件，延时200ms请求，如果输入事件频发则取消上次的定时器执行 取最后一次的输入框内容发起请求

* **搜索请求中文如何请求**
* **介绍观察者模式**

观察者模式有两个角色分类 一个是管理者其内部维护者一个观察列表 ，另一个是观察者，观察者定义了一个方法用来接收管理者的更新调用，当管理者触发更新后 会循环观察列表中观察者的接收函数 也就是管理者和观察者是两者直接联系的。

* **介绍中介者模式**
* **观察者和订阅-发布的区别，各自用在哪里**
* **介绍react优化**

使用pureComponent,

正确使用shouldComponentUpdate

相同结构重复性的子组件加上唯一key值

在render函数中尽量少的新建变量和执行函数

尽量创建浅层级的结构不复杂的state

合理利用无状态组件

服务端渲染

* **介绍http2.0**

1：多路复用，HTTP2.0使得tcp连接口可以并发的发送请求，

2：HTTP2.0采用二进制格式而非文本格式传输数据

3：HTTP2.0 报头压缩，传输更快

4：服务器推送：服务器可以对一个客户端请求发送多个响应，服务端根据客户端的请求，提前返回多个响应，推送额外的资源给客户端。

* **通过什么做到并发请求**
* **http1.1时如何复用tcp连接**

设置请求头和响应头 Connection:keep-alive

* **介绍service worker**

sever worker是离线缓存的一种解决方案 它也属于PWA使用的主要技术之一，它是独立于页面主进程的另一个进程，可以全权的掌控我们的主进程页面，因为它可以缓存资源清单，过滤页面请求。在应用时主页面需要注册server-worker服务，server-worker install 在此阶段我们创建缓存版本 缓存需要缓存的资源（js，css,，图片）。在server-worker 拦截页面的请求 这样可以做一些个性化的定制，server-worker开始运行并接管我们的页面。（在server-worker服务需要修改更新时只需创建新的版本号，server-worker比对版本不同会重新缓存新的资源，在此我们需要手动激活server-worker 去删除旧版本，执行新版本，并重启客户端）

* **介绍css3中position:sticky**

在屏幕范围（viewport）时该元素的位置并不受到定位影响（设置是top、left等属性无效），当该元素的位置将要移出偏移范围时，定位又会变成fixed，根据设置的left、top等属性成固定位置的效果。css3新引入的属性，但浏览器支持度不好（ie不支持，chrom56+，FireFox和Safari支持度较好）

* **redux请求中间件如何处理并发**
* **介绍Promise，异常捕获**

Promise内部的错误虽然浏览器会提示错误，但不会阻止外部代码的执行的，且Promise的错误具有冒泡性质会一直向后传递。虽然浏览器会提示错误。在Promise中可以在then中定义reject状态的函数捕获Promise实例化时函数内执行的错误，但是却无法捕获resolve状态函数中抛出的错误，而catch可以同时捕获，

* **介绍position属性包括CSS3新增**

static：默认，不定位元素在文档流中

fixd：根据浏览器窗口固定定位

absolute：绝对定位，相对于第一个非static定位的父元素定位，如果没有就相对于body定位。  
relative：相对定位，相对于文档里中的位置定位

* **浏览器事件流向**

事件流最早要从IE和网景公司的浏览器大战说起，IE提出的是冒泡流，而网景提出的是捕获流，后来在W3C组织的统一之下，JS支持了冒泡流和捕获流，但是目前低版本的IE浏览器还是只能支持冒泡流(IE6,IE7,IE8均只支持冒泡流)。

现代浏览器先捕获后冒泡，老的IE（6,7,8）仅支持冒泡流

dom 0级事件仅支持冒泡流，但元素同一事件只能绑定一次。

dom2级事件可以绑定同一事件多个处理函数，addEventListener IE的兼容性写法attachEvent(仅支持冒泡)

阻止冒泡：stopPropagation（）(IE: event.cancelBubble=true)

阻止默认行为：preventDefault（IE:returnValue）

target：事件的目标元素

currentTarget：当前处理事件的元素

* **介绍事件代理以及优缺点**

优点：减少内存消耗，不用为每个子元素绑定事件，

可以为动态增加的元素绑定事件

缺点：对于部分不冒泡的事件不支持： focus、blur

如果绑定的元素为根级元素会频繁触发事件，执行绑定函数，消耗性能

* **React组件中怎么做事件代理**

react中可以为每一个组件绑定点击事件

* **React组件事件代理的原理**

React并不会真正的绑定事件到每一个具体的元素上，而是采用事件代理的模式：在根节点document上为每种事件添加唯一的Listener，然后通过事件的target找到真实的触发元素。这样从触发元素到顶层节点之间的所有节点如果有绑定这个事件，React都会触发对应的事件处理函数。这就是所谓的React模拟事件系统。

* **介绍this各种情况**

js this默认指向全局window，在严格模式下指向undefined。js的this绑定有以下几种情况。隐式绑定，函数作为对象的方法调用时 函数内部的this指向调用对象本身。限时绑定：bind、call、apply。new 对象绑定：构造函数中的this指向创建的新对象。

* **前端怎么控制管理路由**

react-router的history:history分为browser、hash、memory三种，分别通过h5的history、url hash值、缓存来处理路由跳转，其中browser利用H5 history特性（可以不刷新的修改页面路径）history.pushState(),history.replaceState()。但pushState和replaceState没有监听事件，无法像hash的onhashchange 无法捕捉事件，更新页面、组件；所以引入了Link这样的内置组件，点击时捕获路径更新对应页面（这也是a标签和Link的不同）；在浏览器的前进后退操作中监听浏览器事件onpopstate，获取history.state，更新页面。

* **使用路由时出现问题如何解决**
* **React怎么做数据的检查和变化**

•react15中react自带PropTypes库，

在16中：

prop-types第三方库可以定义类组件接受的props数据类型 Component.Proptypes={}

* **清除浮动**

**//**核心思想，在元素内插入有个块级元素，让其高宽为0（不影响布局）不可见，设置**clear:both**，为兼容ie加上zoom:1

.clearfix:after{

　　　　　　content:"";//设置内容为空

height:0;//高度为0

line-height:0;//行高为0

display:block;//将文本转为块级元素

visibility:hidden;//将元素隐藏

clear:both//清除浮动

　　　　　}

　　　　.clearfix{

zoom:1;为了兼容IE

}

* **从输入URL到页面加载全过程**

输入网址，浏览器在缓存中寻找，向DNS服务器获取网址ip地址，开始tcp连接，连接完成发起请求，收到服务器的响应

* **tcp3次握手**

客户端发送一个syn包（j）进入SYN\_SENT状态； 服务端收到后返回syn（k）和ack包（j+1）进入SYN\_RECV状态；客户端收到返回一个ack包（k+1）进入ESTABLISHED（TCP连接成功）

第一次握手：主机A发送位码为syn＝1，随机产生seq number=1234567的数据包到服务器，主机B由SYN=1知道，A要求建立联机；

第二次握手：主机B收到请求后要确认联机信息，向A发送ack number=(主机A的seq+1)，syn=1，ack=1，随机产生seq=7654321的包；

第三次握手：主机A收到后检查ack number是否正确，即第一次发送的seq number+1，以及位码ack是否为1，若正确，主机A会再发送ack number=(主机B的seq+1)，ack=1，主机B收到后确认seq值与ack=1则连接建立成功

* **tcp属于哪一层（1 物理层 -> 2 数据链路层 -> 3 网络层(ip)-> 4 传输层(tcp) -> 5 应用层(http)）**
* **redux的设计思想**

1：单一数据源

2：单向数据流，state只读，只能通过action触发修改

3：数据不可变，每次都是返回一个新的数据

* **接入redux的过程**

创建store实例（通过createStore），需传入reducers，initState(可选)，middleware。在react中需要react实例上挂载store（通过react-redux自带的Provider组件）；组件中要调用store的state数据需要connect。

* **绑定connect的过程**

connect是一个函数，返回一个高阶组件修饰原组件，它接受需要调用的redux store实例中state的状态和action 作为props传递给原组件，且设置store的subscribe()函数，添加到store的监听列表中，当store对应的状态修改后，触发绑定的监听函数，更新组件。

* **webpack介绍**

webapck是一个前端自动化构建工具。主要包含 entry、output、modules(loader配置)、plugin、resolve(extensions—后缀缩写，alias-文件夹路径别名)、devtool(调试maps生成方法)、target(运行环境)等主要配置项。做到代码和资源编译，压缩，分割，按需加载等多种强大功能

* **== 和 ===的区别，什么情况下用相等==**

**==**自带自动类型转换，比如string和number数据 1==‘1’成立，0==false成立，1==true成立，null==undefined，而===更准确它不会执行类型自动转换，但还是会有一些不足的点 比如NaN===NaN返回错误，+0===-0返回正确 ，es6中Object.is可以解决这两点

* **bind、call、apply的区别**

从使用场景上call绑定this但不会立即执行函数，当再次调用返回的函数时再执行，且其可以分部传递实参，类似于函数柯里化。

从语言规范上来说 bind是es5才引入的方法，call和apply是es3

call和apply接收的参数不同 apply第二个参数是数组

* **动画的了解**

css3动画，通过@keyframes设定动画，用过animation使用动画，

* **介绍下原型链（解决的是继承问题吗）**

js原型链，每个实例对象都有一个指向原型对象的指针，指向构造函数的原型对象（prototype）。在现代游览器中这个指针是\_\_proto\_\_，在es6中可以通过getPrototypeOf获取。在js继承中，通过改写函数的原型对象实现，将函数原型对象指向父构造函数的实例，那么子类实例就获得了父类的方法和属性。如果父类也继承另一个父类则他们的原型对象就形成了一条链。

* **什么是纯函数**

对于相同的输入，永远都只会有相同的输出，不会影响外部的变量，也不会被外部变量影响。

* **介绍冒泡排序，选择排序，冒泡排序如何优化**

冒牌排序：从左至右依次对比相邻元素大小，前比后大则交换位置，直至上一次最后一次对比的位置，每次都会找到剩余未排序的最大值，冒泡到右边，它的时间复杂度最大为O（n^2）最小为O(n)。冒泡的优化：原始冒泡排序 要循环冒泡n-1次，但可以设置一个标记，记录上次冒泡遍历最后一次换序的位置index，此位置之后的元素排序已经是正序，所以可避免重复遍历比较，下次就接着前index个元素中执行冒泡算法，跳过中间已顺序的元素。

选择排序：遍历n-1次数组，每次找出剩余未排元素的极值和已排序列的后一位交换位置（放已排序列后面），时间复杂度永远为O(n^2)。

插入排序：从左到右，从第二的元素开始，与前一个元素比较，如果小于前一个元素则交换位置，直到遇到比自己小的元素终止本轮插入。最大和最小复杂度和冒泡一样

* **transform动画和直接使用left、top改变位置有什么优缺点**

transform只会触发重绘不会触发回流

* **介绍暂时性死区**

在es6中，在声明之前调用let,const定义的变量 会产生暂时性死区。而var声明的变量会声明提前。这是因为在进入当前执行上下，创建当前上下文词法环境，和变量环境时 var声明的变量已初始化为undefined，let、const定义的变量并未初始化，所以产生暂时性死区。

* **ES6中的map和原生的对象有什么区别**

map可以使用非字符串为键值，例如键值就是一个对象，或者一个symbol。

* **观察者和发布-订阅的区别**
* **pureComponent和FunctionComponent区别**

pureComponent为我们做了state和props的浅比较一定程度避免了组件的无用重新rander。FunctionComponent多用来创建无状态的纯组件，但在16.7版的react中为FunctionComponent引入了hooks，在这里FunctionComponent有了和媲美Component的强大功能

* **介绍JSX**

JSX是react 官方为React.createElement()创建的语法糖，它让dom in js，方便了我们将逻辑和dom结合起来。

* **介绍虚拟DOM**

传统js在数据更新后直接操作dom更新的方式，react和vue等前端框架与传统js不同，开发者专注于业务处理，更新dom的任务交给框架内部。为了避免每次都需要重新渲染整个dom树，引入了虚拟dom，用一个js对象记录整个视图的dom结构，每次更新数据，获取新的虚拟dom与上次对比（获取虚拟dom书并比较要比整个重新渲染dom开销小），通过优化算法找出两次的不同，只修改不同的地方，这样达到了以最小的代价更新视图的目的。

* **Vue和React的比较**

区别：

1：在 React 应用中，当某个组件的状态发生变化时，它会以该组件为根，重新渲染整个组件子树。如要避免不必要的子组件的重渲染，你需要手动实现；在 Vue 应用中，组件的依赖是在渲染过程中自动追踪的，所以系统能精确知晓哪个组件确实需要被重渲染，开发者不需要考虑组件是否需要重新渲染之类的优化。

2：在React中，一切都是JavaScript，所有的组件的渲染功能都依靠 JSX。JSX 是使用 XML 语法编写 JavaScript 的一种语法糖。你可以使用完整的编程语言 JavaScript 功能来构建你的视图页面；在Vue中有自带的渲染函数，Vue也支持JSX，Vue官方推荐使用模板渲染视图。组件分为逻辑类组件和表现类组件。

3：组件作用域内的CSS。CSS 作用域在 React 中是通过 CSS-in-JS 的方案实现的；在Vue中是通过给style标签加scoped标记实现的。

4：Vue 的路由库和状态管理库都是由官方维护支持且与核心库同步更新的。React 则是选择把这些问题交给社区维护，因此创建了一个更分散的生态系统。

* **如何设计Promise.all()**

将多个promise实例包装成一个promise实例 ，只有所有实例的状态都为resolve后，合并包装后的实例才为resolve。

* **介绍高阶组件**

高阶组件是一种修饰原组件的方式，接受一个组件返回另一个修饰后的组件，典型的应用比如react-redux的connect函数返回一个高阶组件，接受原组件返回绑定了redux数据和action的组件。应用中可用来实现 代码复用，权限控制，状态提取等。

* **sum(2, 3)实现sum(2)(3)的效果**

函数返回函数，延迟计算，类似函数柯里化的应用。

* **两个对象如何比较**

如果单纯的比较是否指向同一内存地址，用Object.is或===，如果是深比较，则需要递归遍历，对于简单数据类型的属性===比较，对于引用数据类型的属性递归调用比较。

* **JS的原型**

每个函数都有一个原型对象用prototype表示，每一个实例对象都有一个指针，指向它的构造函数的原型在现代浏览器中这个是“\_\_proto\_\_”，在es6中可以用Object.getPrototypeOf()获取。默认函数的prototype对象的\_\_proto\_\_指向Object的原型对象，Object的原型对象的\_\_proto\_\_指向null

* **变量作用域链**

在一个执行上下文中记录着可引入的外部变量，一个变量如果在当前执行上下文中未找到就会在父执行上下文找，依次类推，一直到最外部作用域，这就是作用域链。如果一直为找到则抛出ReferenceError错误

* **防抖和节流的区别**

他们都是使一段时间内高频事件只会触发一次操作

防抖：前面的触发的操作会被取消，最后一次的操作被执行 。

节流：第一次操作会被执行，后面的操作会被忽略。

* **react生命周期**

一:组件实例化时

1：getDefaultProps---只会调用一次 用于获取初始的props

2：getInitialState---只会调用一次 用于设置初始化的state

3：componentWillMount---在首次渲染之前调用 最后一次修改state的机会 但修改state并不会触发render

4: render ---创建虚拟dom

5: componentDidMount ---虚拟dom创建完毕 并且所有子组件render执行完毕 可以在此获取数据

componentWillMount不能用来获取数据 因为数据的获取是异步的 并不能触发render

二：组件存在期

componentWillReceiveProps ---在组件props更新时执行 在此处可以根据props的修改 更新组件的state并且此时的setState并不会触发组件的render

shouldComponentUpdate---在组件props和state更新时触发 返回false则组件不会更新 返回true则会更新

componentWillUpdate---组件即将重新渲染

componentDidUpdate---组件重新渲染完成 此处可以访问和修改dom

三:组件销毁期

componentWillUnmount ---组件从dom中卸载移除时 在此处应完成清理和销毁工作 例如定时器和dom事件监听

* **介绍Fiber**

https://www.jianshu.com/p/bf824722b496

Fiber在react16中引入，借此提供异步渲染的解决方案。react中的组件渲染是同步的，从父到子一层层渲染，如果组件嵌套很深，那么react可能一直占用浏览器主线程处理渲染计算，无法处理用户的交互。Fiber允许我们暂停组件在render及之前生命周期的操作，跳出渲染，处理其他任务，然后在继续上次的断点处继续执行渲染任务。

* **介绍DOM树对比**

react的diff :react在组件状态修改后 会重新生成virtual-dom与旧的树进行比较，找出差别，只修改需要更新的dom。react的diff比较算法有三个策略：分别是树节点计较（tree diff）、组件比较（component diff）、元素节点比较（element diff）。tree diff：react忽略dom节点跨节点操作，只比较同一级节点，跨节点会删除dom再重新创建一个新的；component diff:不同组件类创建的组件节点不同 就算其结构相似也会删除它及它包含的子节点；3：element diff:同一节点下的同一类子节点，根据key值判断，已存在的不会删除 只做移动位置 ，未存在的插入，已卸载的删除。：element diff中的算法：已知新的子节点各自的顺序，如何操作现有dom，如果当前子节点的原索引小于其左侧子节点原索引的最大值则它移动位置，否则它保持不动，举个现实的例子，有10~99岁的90人，他们原本按年龄排序，你希望他们按一定顺序排序（新的子节点排列顺序），尊老爱幼，年纪大的人不动，年纪轻的人移动，以新顺序的第一个为开始基准，第二个和第一个相比，如果< 第二个原本在第一个前面 那么他现在需要移动到后面来，如果>那么他不用动，第三个和前面两个的最大者比，如果在他最大那么他不动，他小于任何一个那么他动起来插入到它应该的位置。

* **介绍xss，xsrf**

XSRF：跨站请求伪造-用户登录A网址，然后访问B网址，B网址利用A的登录态，模拟用户发起请求，达到攻击目的。预防方法是重要请求采用token+用户登录态验证

xss：跨站脚本攻击，攻击者通过注入脚本（输入框提交，url拼接等），让第三方用户访问时执行脚本，达到攻击目的。比如用户提交评论，评论为一段js脚本，为过滤的脚本保存在数据库，第三方查看攻击者的评论时，js脚本执行，造成攻击。预防方法是过滤输入框，url.

* **http缓存控制**

浏览器的缓存机制:从缓存位置来分 分为server-worker ,memery cache(内存缓存)，disk cache(硬盘缓存)。从缓存机制分为:硬缓存和协商缓存；如果硬缓存时效未过期直接使用缓存，若过期判断协商缓存。硬缓存：在时效内不向服务器发送请求，直接从浏览器缓存中拿取数据，通过设置响应头的cache-control和expires(http1.0产物已过时)。cache-control指令项：max-age（缓存时效）、s-maxage(代理服务器时效)、public(客户端和代理服务器都可缓存)、private（默认开启，服务端缓存）、no-cache(缓存资源。但使不使用取决于协商缓存)、no-store:（不缓存 强制和协商都不使用）；协商缓存：如果强缓存时效，浏览器会携带第一次请求资源得到的响应头中的缓存标识，再次向服务器发起请求，如果服务器根据缓存标识发现文件未修改则返回304状态码和空的响应体 否则重新返回资源。

* **shouldComponentUpdate是为了解决什么问题**

解决不必要的重复render

* **webpack生命周期**

主要分为：

初始化：启动构建，读取与合并配置参数，加载 Plugin，实例化 Compiler。

编译：从 Entry 发出，针对每个 Module 串行调用对应的 Loader 去翻译文件内容，再找到该 Module 依赖的 Module，递归地进行编译处理。

输出：对编译后的 Module 组合成 Chunk，把 Chunk 转换成文件，输出到文件系统。

各个周期内有许多事件，

编译: watch-run, watch模式下每次从新编译时触发。

run:启动编译后执行，只会执行一次

after-compile：一次Compilation 执行完成

invalid：编译时遇到错误

输出：

emit: 确定好要输出哪些文件后，执行文件输出，可以在这里获取和修改输出内容。

done: 成功完成一次完成的编译和输出流程。

* **介绍下浏览器跨域**

同源策略是指：同域名，同端口

1. DOM 同源策略：禁止对不同源页面 DOM 进行操作。这里主要场景是 iframe 跨域的情况，不同域名的 iframe 是限制互相访问的。

2. XMLHttpRequest 同源策略：禁止使用 XHR 对象向不同源的服务器地址发起 HTTP 请求。

跨域是指绕过浏览器同源策略，进行跨域名、端口请求。

* **xsrf（csrf）跨域攻击的安全性问题怎么防范**

重要的接口增加token+登录session验证，xsrf是跨站请求伪造，第三方网站利用用户的登录态，伪造用户发送请求。

* **使用Async会注意哪些东西、**

是否有多个异步操作，前后是否有先后依赖关系，如何处理异常。

* **Async里面有多个await请求，可以怎么优化（请求是否有依赖）**
* **Promise和Async处理失败的时候有什么区别**

promise在then方法的第二个参数中处理失败的结果，或则在链式的结尾的catch中处理失败结果。Async用try包裹await语句，catch中处理失败结果

* **Redux在状态管理方面解决了React本身不能解决的问题**

解决了react多组件深层级公共状态的维护和管理问题。

* **Redux有没有做过封装**

封装过业务中用到的中间件，请求数据的action ，项目中处理类型为promise的action的中间件是自己封装的。

* **前端开发中用到哪些设计模式**

单例模式：一个类只会创建一个实例对象，每次实例化都会返回第一次实例化的对象

工厂模式：把创建实例对象的逻辑封装起来，成为一个方法，不关心对象是如何创建的，根据执行方法传入的实参返回不同的实例对象

观察者模式：观察者模式有两个角色分类 一个是管理者其内部维护者一个观察列表 ，另一个是观察者，观察者定义了一个方法用来接收管理者的更新调用，当管理者触发更新后 会循环观察列表中观察者的接收函数 也就是管理者和观察者是两者直接联系的。

订阅发布模式：订阅-发布模式和观察者的思想很相似 区别在于发布者和订阅者之间还存在一个调度中心 发布者不直接管理订阅者是否收到更新消息 由调度中心去管理。

装饰器模式：模式创建了一个装饰类，用来包装原有的类，并在保持类方法签名完整性的前提下，提供了额外的功能。

* **React/Redux中哪些功能用到了哪些设计模式**

订阅发布模式，装饰器模式

* **JS变量类型分为几种，区别是什么**

基本数据类型：String Number undefined null boolean symbol(es6)

引用数据类型：Object Array Function

* **JS里垃圾回收机制是什么，常用的是哪种，怎么处理的**

引用计数和标记清除。现在浏览器垃圾回收一般以标记清除为主。

引用计数：记录数据被引用的次数，如果没有被引入则在下次垃圾回收中被回收。缺点如果两个对象存在互相引用，那么引用会一直存在不会被回收。

标记清除：把需要清除的内存中的数据看作无法到达的数据，。即从根部（在JS中就是全局对象）出发定时扫描内存中的对象，凡是能从根部到达的对象，**保留**。那些从根部出发无法触及到的对象被标记为**不再使用**，稍后进行回收。

* **一般怎么组织CSS（Webpack**

loader: postcss-loader(处理css兼容性前缀)，less/sass-loader(编译css预处理语言)，css-loader(支持css模块化)，style-loader(计算css加入页面)

plugin：extract-text-webpack-plugin：抽取css到单独文件中

* **取数组的最大值（ES5、ES6）**

Math.max.apply(null,[1,2,3,4,5]);

Math.max(…[1,2,3,4,5])

* **some、every、find、filter、map、forEach有什么区别**

#### 携程

* **对闭包的看法，为什么要用闭包**

闭包定义是函数内部可以访问到函数外部定义的变量。常见的应用比如模拟私有属性，解决循环绑定的index失真，柯里化延迟计算等。我认为闭包实际上是延长了数据的存在期，原本应该被垃圾回收机制回收的变量，因为闭包而被长效的留在缓存中，所以在使用闭包时就需要格外注意，特别是定时器，数据结构较大时使用不慎会造成严重的内存泄露，造成浏览器崩溃。

* **手写数组去重函数**

es6:[ …new Set([1,2,3,4])];

es5：循环遍历，indexof查找，判断插入。

* **手写数组扁平化函数**

特殊清情况如果已知数组的数据类型是number或string:

arr.toString().split(‘,’)

通用情况：

var arr=[1,2,[1,2,3],{a:1},4,5,6];

function extentArr(a,b){

let arr= Array.isArray(b)?a.concat(b.reduce(extentArr,[])):a.concat(b);

return arr;

}

let a=arr.reduce(extentArr,[]);

* **介绍下Promise的用途和性质**

promise是一种异步解决方案，它的出现解决了回调地狱这种不优雅的异步写法。promise实例有三个状态，padding进行中，已成功，和以失败，且状态一旦修改就不能改变，一旦执行promise也不能中途取消。在异步操作有了结果，那么promise的状态也就完成了修改，通过than方法传入的参数，处理成功后的逻辑。一般用在ajax，读取文件能异步操作中。

* **Promise和Callback有什么区别**

promise链式调用写法，代码更佳优雅，逻辑直观，方便阅读，可维护性更高。

#### 喜马拉雅

* **componentWillReceiveProps的触发条件是什么**

组件接受新的props，或则父组件重新render，造成子组件也重新render

* **React16.3对生命周期的改变**

移除：componentWillMount、componentWillReceiveProps、componentWillUpdate

新增getDerivedStateFromProps（替换componentWillReceiveProps）、getSnapshotBeforeUpdate（替换componentWillUpdate）

* **介绍React高阶组件**

接受一个组件，返回修饰后的新组件

* **React-Router版本号**

V2将路由集中管理 通常放到一个router.js文件中定义

V4是将路由才拆分开来的动态路由 可以在组件中定义路由 route变成完全的组件化了 可以再组件中使用

对比之前如果需要在两个子路由组件使用公共组件和功能 是通过设定父路由的component 然后通过this.props.children克隆子路由组件现在可以直接定义父路由父组件 在父组件中使用route定义子路由

V4路径默认非精准匹配 比如'/add' 可以匹配 '/'和'/add'两个route

V4增加 Switch 组件 只渲染一个匹配到的route

V4正价exact属性 实现精准匹配

V4 去除了IndexRoute组件 可以由<Route exact>实现

* **网站SEO怎么处理**

html标签语义化 em strong header nav footer

meta设置关键字，网址描述

spa 服务端渲染

* **介绍下HTTP状态码**

2XX:请求成功

200：成功服务器已经处理了请求。

201：处理了请求并创建了一个新资源（生成了exel或上传文件接受等）

202：服务器已接受请求，但还未处理完毕。（比如异步任务，文件转码等）

3XX:重定向

301：永久重定向，请求链接一永久重定向到了新链接，下次直接跳往新链接

302：临时重定向，请求链接临时重定向到新链接，下次还是从原链接请求

304：缓存未过期从浏览器缓存获取数据，响应体为空。

4XX：请求错误

401 - 未授权。请求要求身份验证。 对于需要登录的网页，服务器可能返回此响应。

403：服务器拒绝了浏览器的请求

404：资源不存在

405：请求方法不允许

413：请求体携带的资源过大

408：请求超时

5XX:服务器错误

500：服务器遇到错误，无法完成请求

502 ：代理服务器未收到原服务器的有效响应（nginx）。

503:服务器过载或临时维护，当前无法完成请求，可能在一段时间后恢复。

505：当前http协议不支持

* **介绍HTTPS**

https：https和http区别 http传输是明文的/https是经过ssl协议加密的安全传输协议，http端口是80/https端口是443。https客户端和服务端消息经过对称加密，对称加密的算法经过非对称加密（即公钥和私钥）每个连接的对称加密算法都不一样。服务器保存着私钥，客户端保存着公钥，公钥和私钥可以相互解密；公钥由服务器发送给客户端。数字证书：客户端将公钥放在第三方证书颁发机构(CA)私钥加密过的数字证书中，客户端收到数字证书后由本地CA公钥解密验证公钥的正确性（公钥与访问的网址匹配），拿到公钥和服务端确定对称加密的算法，

* **对PWA有什么了解**

PWA：全名叫渐进式web应用， 主要分为离线缓存、消息推送、mainfest.json应用清单三大功能。桌面图片、应用名称主要通过manifest.json配置。离线缓存通过Service Worker实现，主页面进程注册Service Worker；Service Worker服务install(如果Service Worker文件未修改已创建过则服务已存在不会再次install)在此时保存需缓存的文件清单；当文件保存完毕手动使Service Worker进入activating激活阶段，在激活阶段删除之前的Service Worker服务 ，然后更新客户端让新版本服务控制客户端；在Service Worker激活后 Service Worker可以全面接管页面的请求，做到过滤缓存资源的目的。消息推送需要客户端-Service Worker，服务端共同配合，需要在各自浏览器提供的消息推送服务注册，客户端注册消息推送服务提交自己得到的唯一身份信息，后台每次发送消息都向浏览器提供的消息推送服务器 发送消息内容和客户端身份信息，浏览器服务器会将消息转发给各个客户端。目前PWA在安卓上支持度较好 但在ios11.3才等到manifest和离线缓存支持 但消息推送并未支持 ，且目前浏览器消息推送服务器chrome在国内被墙无法使用，只有firefox支持尚好

* **对安全有什么了解**

sql注入

xss：跨站脚本攻击

crfs：跨站请求伪造

DDOS攻击

* **介绍下数字签名的原理**

数字签名：服务端对信息使用hash算法得到摘要，再对摘要私钥加密生成数字签名。客户端用公钥解密数字签名生成摘要，再用hash算法对信息计算生成自己的摘要，如果两份摘要一致，则证明信息没有被修改过。在https中这个数字签名的私钥不是服务端自己的私钥生成的，而是CA的私钥加密后的数字签名，而客户端有CA的公钥。https数字证书即包含数字签名也包含公钥。

* **前后端通信使用什么方案**

RESTful

RESTful介绍：

（1）每一个URI代表一种资源；

　　（2）客户端和服务器之间，传递这种资源的某种表现层；

　　（3）客户端通过四个HTTP动词，对服务器端资源进行操作，实现"表现层状态转化"。

* **RESTful常用的Method**

GET POST PUSH PATCH DELETE

#### 兑吧

* **localStorage和cookie有什么区别**

cookie大小4k

localStorage最大可存储5M数据

cookie在http请求时请求头携带

cookie可设置过期时间，是否可访问

* **CSS选择器有哪些**

类选择器 .box

id选择器 #box

元素选择器 div

后代选择器 div p

子选择器 div>p

伪选择器 (a:hover, a:visited)

群组选择器（a,div,p{}）

结构性伪类选择器(:nth-child,:first-child)

通用选择器 \*

!important > 行内样式 > id选择器>类选择器>元素选择器>通用选择器>

* **什么是BFC**

BFC叫块格式化上下文

任何有如下css属性的元素都会创建一个BFC:

根元素

position: absolute/fixed

display: inline-block / table

float 元素

ovevflow !== visible

BFC有如下特点：

BFC内的元素margin可能发生重合

float元素不会和BFC元素重合 （）

BFC就是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面的元素；

BFC元素计算高度时浮动元素也计算高度。所以如果有浮动塌陷，可以给父元素设置BFC

* **盒子模型，以及标准情况和IE下的区别**

标准盒模型元素宽高等于content border-content

IE盒模型是元素宽高等于content+padding+border border-sizing

* **new是怎么实现的**

创建一个新对象，将对象的原型指正指向构造函数的原型对象，将构造函数的this绑定为对象，执行函数，返回对象.

* **如何实现H5手机端的适配**

rem布局

* **Redux状态管理器和变量挂载到window中有什么区别**

window触发变更不利于追踪数据修改的记录，也不好绑定观察事件

* **webpack和gulp的优缺点**

gulp配置更直观,灵活，通过使用一个一个的task一步步构建项目

webpack使用更自动化，视所有资源都为模块。

* **前端性能优化（1js css；2 图片；3 缓存预加载； 4 SSR； 5 多域名加载；6 负载均衡）**
* **并发请求资源数上限（6个）**

http1.x可以服用tcp连接 但浏览器最多同时建立6个tcp连接

* **介绍webp这个图片文件格式**

google2010年推出的一种图片编码格式，相同质量情况下 wbbp比jpeg小30%，缺点是浏览器支持度不好，IE，saferi不支持，firefox较高版本支持

* **异步请求，低版本fetch如何低版本适配**

fetch-polyfill

* **介绍Immuable**

Immuable.js数据不可变，每次修改都会生成新的数据，在react state中我们推荐使用不可变的数据，即对于引用类型的数据每次应该创建新的数据替换原本的而不是在原数据上修改（这样前后两次还是指向同一内存地址 即===）；

* **如何继承**

原型链继承：绑定构造函数为子类实例；缺点：无法向父构造函数传参，对于引用数据类型的属性被一个实例修改后（push，splice等非从新赋值操作）会影响所有子类实例对象。

构造继承：子构造函数复用父构造函数，Father.apply(this)；优点：可给父类传参，引用类型属性不会被篡改影响所有子类对象，缺点：无法继承父类原型上的属性。

组合继承：组合前两种方式（子构造函数复用父构造函数，子构造函数复用父构造函数）;几乎完美的继承方式，但是父类构造函数中的属性会在子类实例和子类原型上都有，有少许新能浪费，且子类实例的constructor指向的是父类非子类。

原型式继承：Object.create(父类实例)类似，这种继承没有什么实际作用，因为父类引用数据也会被共享，而且每次获取新对象还要自定义自己的属性和方法 没有复用子类自己的属性和方法。

寄生式继承：基于原型式，不过把逻辑封装在函数里了，把原型式获取新对象和修饰新对象方法函数里

寄生组合式：和组合式的不同在于，将子类的原型指向寄生式包装过的父类原型，且将包装后的对象的constructor指向子类。

#### 微医

* **栈和堆的区别**

堆：堆是一种无序的树状结构 我们只需要知道数据叫什么就可以取出来用 比如josn 我们知道key就能取得value

栈：栈是一种先进后出的结构 类似于兵乓球盒

队列：队列是一种先进先出的结**构**

* **闭包的使用场景**

动态生成dom 循环绑定事件

模拟私有私有属性

柯里化延迟计算

* **JS怎么实现异步**

callback回调函数

定时器

promise

generator

Async

* **Promise和setTimeout执行先后的区别**

promise then中的代码优先于Timeout中的代码执行，因为promise的异步是微任务，setTimeout的异步是宏任务。js的事件循环机制中一次宏任务结束后立即执行可执行的微任务，然后检查渲染，GUI线程接管浏览器渲染渲染树，然后执行下一个宏任务。

* **JS为什么要区分微任务和宏任务**

js语言设定js引擎与GUI引擎是互斥的,也就是说GUI引擎在渲染时会阻塞js引擎计算.原因很简单,如果在GUI渲染的时候,js改变了dom,那么就会造成渲染不同步，宏任务和微任务为了解决js线程的处理和GUI引擎渲染的冲突

* **JS执行过程中分为哪些阶段**

js的执行过程分为两个阶段：1：预编译(我认为也可以叫做创建执行上下文)，2：执行阶段。预编译阶段设定的当前执行上下文的this，创建了当前执行上下文的词法环境和变量环境，初始化了var声明的变量（undefined）,定义了声明的函数（已赋值），记录了let const定义了变量（但未初始化，所以会有暂时性死区现象），保存了可使用的外部执行上下文的变量。执行阶段即执行代码。

* **词法作用域和this的区别**

词法作用域（scope）表示当前函数的作用域，在函数定义时就被保存下来并成为函数的一部分。

* **loadsh深拷贝实现原理**

循环遍历拷贝对象和数组的属性，遇见引用数据类型的属性则递归调用函数自身。

* **React中setState后发生了什么**

shouldComponentUpdate;

componentWillUpdate;

render;

conponentDidUpdate;

* **setState为什么默认是异步**

异步也不是真正意义的异步 而是在一个react自定义事件或生命周期中将state的更新批量执行。这样批量执行可以防止多次更新state带来的不必要的计算和视图更新

* **setState什么时候是同步的**

在除了react自定义的事件和react的生命周期中都是同步的 ，比如定时器，原生dom绑定的事件中

* **虚拟DOM主要做了什么**

将数据层和数据层中间加入一层虚拟dom，当数据层的变动引起试图层的更新时，先生成一份虚拟dom，对比前后虚拟dom的差异，操作真实dom只改变差异部分的结构。

* **打包时Hash码是怎么生成的**

在webpack配置文件 entry 配置filename hash ，chunkFilename异步懒加载文件的hash ，如果需要生成缓存hash需要引入webpack自带的CommonsChunkPlugin插件，生成项目的runtime，避免不必要的hash修改。

* **使用webpack构建时有无做一些自定义操作**

webpack-dev-server集成化太高，不利于根据项目拓展。将其抽离出来，用express+webpack-dev-middleware+webpack-hot-middleware手动搭建了开发环境，基于此引入了node代理，解决了开发环境的跨域问题。

还尝试过编写webpack的插件，将打包好的文件上传cdn，原理是webpack的插件是一个类，该类的实例拥有apply方法，能得到webpack执行实例，webpack实例监听打包完毕的end事件 在end事件中调用上传cdn的api

* **遇到的复杂业务场景**

后台数据统计页，根据用户角色（7~8级），统计卡统计指标不同，计算方式不同，形容词不同，下方有下级部门翻页列表，下级列表有各下级部门的统计指标（各个级别下级部门的统计指标不同）。上一级有一级级往下观看下级统计指标的权利。该应用场景角色层级较多，统计指标多样，且呈现并不单一，上级部门可浏览各个层级的各种统计页，且可拓展（角色的可能增减，统计指标项可能修改，统计的计算方式也可修改）。我在编写该组件时将交互层（翻页，上级往下查看，根据数据渲染表格，表格的首行数据名，数据的计算方法）与角色解耦，将各个层级的获取数据的api，数据统计名称，该获取的json属性名，数据描述，数据计算，统一在一个配置文件中配置，这样在日后维护和拓展中只用修改配置文件就能完成。

* **Promise.all实现原理**

#### 寺库

* **介绍排序算法和快排原理**

快排的原理，取一个数组元素（中间元素或最左边元素），以该元素为界将数组分为两组，一组>元素 一组<=元素；然后两个分组继续执行以上操作，以此类推，直到分组只有一个元素，不能再分。

* **介绍DNS解析**

dns解析也叫域名解析，将网站域名解析为ip地址，通过ip地址找到服务器，与服务器建立连接。域名解析的服务（域名解析服务器）一般由域名注册商提供

* **cookie的引用为了解决什么问题**

解决了在客户端存储状态和数据的需求。比如用户的登录态（和session配合），

#### 宝宝树

* **rudux和全局管理有什么区别（数据可控、数据响应）**

#### 海康威视

* **如何判断一个变量是不是数组**

es5:Array.isArray()

通用：Object.prototype.toString.call(arr)

* **变量a和b，如何交换**

用变量c保存 a

b赋值给a

c赋值给b

* **类数组和数组的区别**

类数组对象有数组的索引属性结构和length属性 可以通过索引下标[index]获取类数组中的元素。但是类数组没有数组的方法 比如push pop forEach indexOf，常见的类数组对象有函数的arguments，元素节点数组。

* **dom的类数组如何转成数组**

es6可以通过Array.from()

* **介绍单页面应用和多页面应用**

单页面首页加载慢 但页面转化快，seo不好，公共资源一次加载多次复用，用户体验更好（交互页面跳转快，页面跳转可以设置过度跳转动画），利于维护和协作开发，整个项目为一个整体，方便做整体的协调的全局配置。

多页面加载快，seo良好，开发较轻便入门门槛较低，但用户体验逊于单页面应用。

* **介绍localstorage的API**

localStorage.setItem('myCat', 'Tom'); //设置localstorage 属性值

localStorage.getItem(‘myCat’);//获取localStorage 属性值

localStorage.removeItem('myCat');//删除localStorage 属性值

localStorage.clear();//移除所有localStorage保存的数据

#### 蘑菇街

* **html语义化的理解**

语义化是值在页面中，用恰当的标签去表现不同的结果，比如导航有nav标签，底部有footer，标题有h标签，加粗强调使用strong，列表ul li等。合理使用标签能有助于seo和阅读代码。

* **<b>和<strong>的区别**

strong:加粗强调

b:斜体强调

* **使用原型最大的好处**

复用属性和方法

* **Redux数据回溯设计思路**

redux的数据流可以整理为 action->中间件->dispatch->reducer->state更新，state的更新使我们根据state沿着链路找到触发state某个状态更新的起因在哪里。

#### 酷家乐

* **添加原生事件不移除为什么会内存泄露**
* **还有哪些地方会内存泄露**

1：意外的全局变量

在函数为使用var或let、const声明变量，变量挂载在全局对象中。或则在函数中错误的使用this，给this设定属性，但此处this非预期，而是指向全局对象。

2：在ie老版本中 dom绑定事件，事件中引用可dom 所以造成了循环引用，造成内存泄露

3：闭包

* **setInterval需要注意的点**

定时器的时间不可靠

* **定时器为什么是不精确的**

因为定时器作为异步任务，被加入到事件队列，当定时时间到了之后还需要等待主线程处理完毕当前任务然后执行定时器宏任务，这就造成了等待，时间的不准确性。

* **介绍pureComponet**

在组件重新渲染时（setState,父组件重新render）pureComponet会浅比较state和props是否修改，相当于帮我们定义了shouldComponentUpdate，避免组件无意义的重复render

* **介绍class和ES5的类以及区别**

语法 classs声明 extends继承

es6的继承如果用es5实现 我认为最接近组合寄生式继承，但也有不同。1：子类的\_\_proto\_\_指向父类，2：在继承中子类class构造函数的this在父类中生成并修饰，然后由子类修饰this，所以es6中子类的构造函数必须调用super()

* **介绍箭头函数和普通函数的区别**

箭头函数默认返回

箭头函数中的this指向创建时的对象

普通函数中的this永远指向调用它的对象

* **介绍defineProperty方法，什么时候需要用到**

Object. defineProperty(obj,{…})，用于定义对象的属性，可配置该属性的set和get ，value、是否可枚举（遍历）、是否可修改。在定于不能被篡改的属性和方法是会用到

* **for..in 和 object.keys的区别**

for in可遍历对象原型对象上可遍历的属性 object.keys只能获取对象本身的属性

* **使用闭包特权函数的使用场景**

有权访问和修改函数内部私有变量的方法叫特权函数，比如redux，redux的实例redux，并不能通过访问store的属性直接获取state,但是可以通过store的getState方法获取store里保存的state，也可以通过dispatch方法修改state

* **get和post有什么区别**

GET参数通过URL查询字符串传递放在请求头中，POST放在Request body中。

GET请求在URL中传送的参数是有长度限制的，而POST没有。

GET请求的参数只能是ASCII码，所以中文需要URL编码，而POST请求传参没有这个限制

GET请求会被浏览器主动缓存，而POST不会，除非手动设置。

GET产生一个TCP数据包；POST产生两个TCP数据包。对于GET方式的请求，浏览器会把http header和data一并发送出去，服务器响应200（返回数据）；而对于POST，浏览器先发送一个options类型请求，服务器响应100 continue，浏览器再发送data，服务器响应200 ok（返回数据）。

#### 百分点

* **重新渲染render会做些什么**

render会生成当前组件的vdom，当遇到子组件时会迭代触发子组件的render(如果子组件为阻止重新渲染)，知道所有的子组件render完毕，则组件的虚拟dom生成完毕。之后会去对比前后vdom的变化，视图更新，之后进入componentDidUpdate声明周期。

* **哪些方法会触发react重新渲染**

setState（）除了componentWillReceiveProps中的setState()

组件的props修改，多来源于父组件的重新render

父组件重新render 即使子组件的props前后完全一样

* **state和props触发更新的生命周期分别有什么区别**

props触发componentWillReceiveProps生命周期；

之后的shouldComponentUpdate、componentWillUpdate 、componentDidUpdate一致。

在react16.3版本后他们触发的生命周期变化并无不同

不过shouldComponentUpdate替换成getDerivedStateFromProps

componentWillUpdate替换成getSnapshotBeforeUpdate

* **对无状态组件的理解**

内部无状态的纯展示性组件，对于这种无状态的组件，使用函数式的方式声明，会使得代码的可读性更好，并能大大减少代码量。无状态组件使大的组件拆分为小的组件，利于维护和复用。

* **let、const以及var的区别**

块级作用域，不可声明提前会存在暂时性死区

* **算法：前K个最大的元素**

1：取数组前k个元素为一个新数组，获取最小值，从第k+1个元素遍历，如果元素大于新数组最小值，替换最小值，重新获取新数组最小值。

2：可以参考加速排序的算法

分割数组，如果右边的分组length小于k 右边全部加入新数组，还剩下k-length个元素，继续对左边的分组进行如上操作；如果length>k 则丢掉左边的分组，从右边的分组中继续如上操作，如此递归。

#### 海风教育

* **react的理念是什么（拿函数式编程来做页面渲染）**

UI = render（data）

用户所看到的的UI界面是由data（数据）作为参数调用render函数（纯函数）作用的结果

* **进程和线程的区别（一个node实例就是一个进程，node是单线程，通过事件循环来实现异步**

**）**

一个程序至少一个进程，一个进程至少一个线程。线程是程序执行时的最小单位，它是进程的一个执行流，是CPU调度和分派的基本单位，一个进程可以由很多个线程组成，线程间共享进程的所有资源，每个线程有自己的堆栈和局部变量。线程由CPU独立调度执行，在多CPU环境下就允许多个线程同时运行。同样多线程也可以实现并发操作，每个请求分配一个线程来处理。

* **介绍下DFS深度优先**

深度优先： 从任选顶点v起 若当前顶点问的相邻顶点，访问与其中一个相邻的顶点，依次类推向下访问，若其相邻订点都访问过或没有则返回上一次访问的顶点，直到所有能与v顶点想通的顶点都被访问到。若图中还存在为访问的顶点则已该顶点起重复上述步骤。

例如s={

a:{

a1:{ a1\_1:1, a1\_2:1},

a2:{ a2\_1:1, a2\_2:1}

},

b:{

b1:{ b1\_1:1, b1\_2:1},

b2:{ b2\_1:1, b2\_2:1 }

},

}

从s顶点出发 a顶点与其相连且未被访问 访问a ,a顶点中有a1顶点与其相连 访问a1，再到a1\_1，a1-1未有与其相邻未访问过的顶点于是退回a1，a1还剩下a1\_2为访问的相邻顶点，然后返回a1 ，a1不存在未访问的相邻顶点返回a ,

s->a->a1->a1\_1-<a1->a1\_2-<a1-<a->a2->a2\_1-<a2->a2\_2-<a2-<a-<s->b….

广度优先：从选择的顶点出发 先横向访问所有与其相

邻的顶点，再依次按顺序访问这些相邻顶点的相邻顶点，直到所有顶点被访问

* **观察者模式里面使用的数据结构(不具备顺序 ，是一个list)**
* **CSS中link和@import的区别**

link属于HTML标签，而@import是CSS提供的

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

import只在IE5以上才能识别，而link是HTML标签，无兼容问题

link方式的样式的权重 高于@import的权重