* **使用过的koa2中间件**
* **koa-body原理**
* **介绍自己写过的中间件**
* **有没有涉及到Cluster**
* **介绍pm2**
* **master挂了的话pm2怎么处理**
* **如何和MySQL进行通信**
* **React生命周期及自己的理解**
* **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）

* **服务端渲染SSR**
* **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**
* **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）
* **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，）

* **项目中如何处理安全问题**
* **怎么实现this对象的深拷贝**
* **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={}

* **对async、await的理解，内部原理**
* **介绍下Promise，内部实现**
* **清除浮动**

.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）

* **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**
* **如何判断链表是否有环**
* **介绍二叉搜索树的特点**
* **介绍暂时性死区**
* **ES6中的map和原生的对象有什么区别**
* **观察者和发布-订阅的区别**
* **react异步渲染的概念,介绍Time Slicing 和 Suspense**
* **16.X声明周期的改变**
* **16.X中props改变后在哪个生命周期中处理**
* **介绍纯函数**
* **前端性能优化**
* **pureComponent和FunctionComponent区别**
* **介绍JSX**
* **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）**
* **介绍虚拟DOM**
* **如何设计一个localStorage，保证数据的实效性**
* **如何设计Promise.all()**
* **介绍高阶组件**
* **sum(2, 3)实现sum(2)(3)的效果**
* **react性能优化**
* **两个对象如何比较**