面试

- 自我介绍 您好，我叫陈雨寒，今年22岁， 是江西财经大学软件工程专业2020届毕业的一名学生，我比较擅长前端的技术对后端技术也有些了解，比较熟悉js，html，css对react也比较熟悉， 我做过小程序的项目和react项目

- 有什么问题问我 你们公司现在用的技术大概是什么 如果我能够去的话，是在你们公司的话能够做什么工作

- 为什么做前端 自己在学校主要学的后端，平常做一些项目发现前端可以很好的展示项目，发现前端还是挺有趣的可以做出很多很好的界面，做出来的东西自己能够看得见，并不想后端那么枯燥

- 对前端的一些看法 前端 并不像后端那么枯燥，能够做出产品来，而且比较接近用户，能够学习很多的新东西 自己以后在前端的发展 我既然决定了在前端方面发展，就会努力往前端方面专研，有时间我也会学习后端东西，毕竟前端也会涉及后端的东西 先积累工作经验，和学习更多知识，并且在公司能够做出一个好的成绩 太远的也没有多想，既然选择了前端行业，希望能够在前端闯出一片天地

    - 碰到的问题 对于一些问题 自己在做项目的时候没有一个完整的规划，有时候在项目写到一定程度的时候可以需要修改比较打的范围 对前端界面工程师这个职位是怎么样理解的？它的前景会怎么样？ 实现页面交互 提升用户体验 node 也可以实现服务器端的一些事情 前端是最贴近用户的

- js

    - 一个页面从输入 URL 到页面加载完的过程中都发生了什么事情？越详细越好

        1. 浏览器中地址栏中输入URL并回车。 URL由三部分组成， 协议名， 域名， 端口
        2. 浏览器查找当前的URL是否存在缓存，并且比较缓存是否过期
        3. DNS解析URL的对应的IP地址 DNS解析就是根据域名来获取IP地址
        4. 根据IP地址建立TCP连接（三次握手） 第一次握手 客户端发送SYN（同步序列编号）包到服务器，，并进入SYN_SENT状态，等待服务器确认 第二次握手 服务器接受SYN包，想客户端发送SYN+ACK包，进入SYN_RECV状态 第三次握手 客户端接受到SYN+ACK包，想服务器发送确认包ACK，发送完毕TCP连接成功
        5. http发起请求 建立连接后浏览器向服务器发送http请求 包括，请求起始行，请求头，请求体
        6. 服务器处理请求， 浏览器接受响应 服务器接受浏览器的http请求后，将接受到的http报文封装成http的Request对象 并通过不用的web服务器处理， 处理完的结果以HTTP的Response对象返回 response主要包括 状态码 100-199: 表示请求已经接收，继续处理 200-299：表示请求成功 200 请求成功，并且请求的数据已经返回 300-399：重定向，请求转到另外的地址 400-499：客户端错误-请求出错，语法错误或者地址不存在 500-599：服务器错误- 响应头 响应头主要由Cache-Control、 Connection、Date、Pragma等组成。 响应报文 就是服务器响应请求返回的数据，主要由HTML，css，js，图片文件组成。
        7. 渲染页面，构建DOM树 浏览器解析渲染呈现给用户， 解析和渲染，渲染前构建DOM数和CSSDOM 回流 当内容结构内容位置发生尺寸发生改变，需要重新计算样式和渲染树 重绘 只是改变一些样式（背景颜色边框颜色等）只需要重新绘制样式就可以
        8. 关闭TCP连接（四次挥手）
           第一次挥手 浏览器发送完数据后，发送FIN请求断开连接 第二次挥手 服务器发送ACK表示

同意 第三次挥手 服务器端应用程序通知TCP关闭连接， 服务器端TCP发送报文 第四次挥手 客户端接受报文再发送报文 服务器接受报文 断开连接

- 基本数据类型 string number boolean null null===null 在判断的时候null 和undefined都是true undefined undefined===undefined null == undefined null和undefined在使用==的时候将会进行类型转换都为false symbol 具有代表性的，独一无二的值 可以作为标识符
    - 这个里有独特地方NaN不是基本数据类型， NaN（not a number） 不等于自身
- 获取对象的所有属性（不包括原型上的属性） 使用getOwnPropertyNames获取属性 还有hasOwnProperty判断是否拥有该属性
- 创建一个不能被修改的对象的属性 Object.defineProperty 这是writeable为false可以创建一个不能被修改的属性
- 跨域， qm_lession node cross_domain 细看 跨域的产生： 为了安全问题浏览器有同源策略，当（协议，域名，端口不同时）脚本发送请求会被浏览器拦截 两个域名之间不能跨过域名来发送请求或者请求数据，否则就是不安全的，这种不安全也就是CSRF（Cross-site request forgery），中文名称：跨站请求伪造，也被称为：one click attack/session riding，缩写为：CSRF/XSRF 但是在开发中需要一些绕过同源策略获取数据，这就是跨域 跨域最常用的两种方式， jsonp 只能使用GET方式 通过script标签实现跨域请求，然后通过传递过去的回调函数获取数据 首先在客户端注册一个callback，然后把callback的名字传给服务器。此时，服务器先生成json数据，然后以javascript语法的方式，生成function，function名字就是传递上来l带参数jsonp。最后将json数据直接以入参的方式，放置function中，这样就生成js语法的文档，返回给客户端。客户端浏览器，解析script变迁，并执行返回javascript文档，此时数据作为参数，传入了客户端预先定义好的callback函数里。简单的说，就是利用script标签没有跨域限制的“漏洞”来达到与第三方通讯的目的。 具体实现见 qm_lession/node/cross_domain cors (Cross-Origin-Resource-Sharing) 跨源资源分享 还支持其他Http请求， post，delete等 使用自定义的http头部允许浏览器和服务器相互了解对方， 从而决定请求或者响应成功与否 给服务器设置如下 Access-Control-Allow-Origin: 指定授权访问的域。（只能是所有地址*, 或者是某个地址....）; Access-Control-Allow-Methods: 授权请求的方法(POST, GET, DELETE,PUT, OPTIONS) 当浏览器发送请求后收到允许Access-Control-Allow-Origin就会将数据返回， 否则浏览器将会拦截数据 Access-Control-Allow-Origin: 跨域服务器允许的来源地址（跟请求的Origin进行匹配），可以是*或者某个确切的地址，不允许多个地址 Access-Control-Allow-Methods: 允许的方法GET、HEAD、POST Access-Control-Allow-Headers: 允许的Content-Type text/plain multipart/form-data application/x-www-form-urlencoded Access-Control-Max-Age: 预请求的返回结果(Access-Control-Allow-Methods和Access-Control-Allow-Headers)可以被缓存的时间，单位秒 请求头限制 XMLHttpRequestUpload对象均没有注册任何事件监听器 请求中没有使用ReadableStream对象
- xss和csrf
    - xss (跨站脚本攻击) 当用户提交数据的使用被嵌入了恶意代码被保存在数据库中，在你正常保存到数据库中再拿出来的时候代码就会被浏览器作为代码解析， 比如当你在保存信息的时候被嵌入一段script标签，这个标签会引入恶意的js代码，当下次取出信息的时候，就会执行代码。受到攻击 防御 对用户保存的信息作处理，将一些符号进行转义，例如大于号小于号等>< 在用户输入时候对一些不合法的东西都过滤掉，比如script， style或者一些dom属性 对cookie设置httponly
    - csrf (跨站请求伪造) 当你在登录一些网站的时候，在同时又访问一些危险的网站。 这时当点击链接时候会利用在其他网站处于登录状态来提交一些请求 例如当你登录银行的时候，不

小心点击了一些危险的链接，由于你处于银行网站登录状态，这时它就可以获取你的cookie伪装成你向银行发送请求盗取你钱信息等。 防御： 通过refer， token或者验证码来检测用户提交 尽量不要在页面的链接中暴露用户的隐私 对于用户修改删除等操作最好都使用post操作 避免全站通用的cookie， 严格设置cookie的域

- 数组的一些方法 arr.join() 将数据中的每一项，以某一种分隔符连接, 默认用(,)连接 返回连接后的 arr.push(),数组添加可以同时传入多个参数，返回长度， arr.pop() 删除最后一项元素 返回移除的项 shift() 删除第一项 返回删除的元素 unshift() 将参数添加在数组开头 返回长度 sort() 排序，可以定义排序规则传递一个函数 reverse() 将数组倒叙 concat() 将传递的参数添加都数组副本的后面 slice() 返回从指定起始下标到结束下标的数字，左闭右开， 没写任意一项默认以第一项，最后一项 splice() 可以删除，插入，修改 参数， 原数组不会改变 arr.splice(1, 2) 从第二项开始删除两项 arr.splice(1, 0, 3, 4) 在第一项后面就是插入， 3， 4 arr.splice(1, 1, 3, 4) 删除arr[1], 然后插入，3，4 indexOf() 接受两个参数， 第一个参数 需要查找的值， 第二个参数起始位置 返回下标 没有则返回 -1 lastIndexOf() 接受两个参数， 第一个参数需要查找的值， 第二参数起始位置但是是从后面向前查找, 返回下标 没有则返回 -1

  forEach(function(item, index){}) 遍历数组的每一项， 参数为自定义function(item,index,arr)， 没有返回的值 map() 跟forEach差不多， 但是经过function后返回新的数组 filter() 参数跟前两项一样，根据自定义函数对每一项进行筛选， return false则去除 every() 参数只需要一个函数当每一项返回的都为true 最后才返回true some() 只要一项满足要求就返回 ture reduce(function(prev, item, index, arr){}, 初始值) 从数组的第一项开始迭代 将上一次迭代的值作为函数的第一项参数， 可以给初始值， 返回迭代的结果

- 递归和迭代的区别， 递归就自己调用自己， 遇到满足的条件则终止迭代，使用return来结束 而迭代跟数组的reduce差不多 迭代跟不同循环差不多， 只是会将每次循环后的值作为下次计算的初始值

- 小程序和react的区别

- 深拷贝 由于复杂数据类型中保存是，堆中的地址并不是值， 使用JSON.stringify(), JSON.parse() 或者使用递归进行深拷贝

- 内存空间，垃圾回收

  1. 为变量对象分配需要的内存
  2. 在分配到的内存中读写数据
  3. 不再使用时就会将其销毁。 javaScript 中有自动垃圾回收机制，会销毁不再使用的变量等。手动销毁变量将值设置为null失去作用， 下次就自动销毁 函数局部变量在使用后一般会被垃圾回收销毁， 但是闭包会阻止这一过程

- 栈， 队列， 堆 内存就是栈和堆， 栈就是有序 栈 一种先进后出的一种线性表， 有操作系统分配释放，存放函数的参数值，局部变量等 都是在栈顶操作 堆 一种key value 的无序表 一般由程序员释放， 如果程序员不释放， 程序结束是会有操作系统回收， 分配方式类似于链表 队列是 先进先出的 在队头插入， 队尾删除

- 继承

  1. 原型链继承 将父类的实例作为子类的原型 特点： 非常纯粹的继承关系，实例是子类的实例，也是父类的实例 父类新增原型方法/原型属性，子类都能访问到 简单，易于实现 缺点： 要想为子类新增属性和方法，必须要在new Animal()这样的语句之后执行，不能放到

构造器中 无法实现多继承 来自原型对象的所有属性被所有实例共享（来自原型对象的引用属性是所有实例共享的）（详细请看附录代码： 示例1） 创建子类实例时，无法向父类构造函数传参 function Animal(){} function Cat(){} Cat.prototype = new Animal();

2. 构造继承 将父类的构造函数的内容复制给子类的构造函数 特点： 解决了1中，子类实例共享父类引用属性的问题 创建子类实例时，可以向父类传递参数 可以实现多继承（call多个父类对象） 缺点： 实例并不是父类的实例，只是子类的实例 只能继承父类的实例属性和方法，不能继承原型属性/方法 无法实现函数复用，每个子类都有父类实例函数的副本，影响性能 function Animal(){} function Cat(name){ Animal.call(this); this.name = name; }
3. 组合继承 function Super(name){ this.name = name; this.friends = ['gay1', 'gay2'] } Super.prototype.say = function(){ console.log(this.name); } function Sub(name, age){ Super.call(this, name); this.age = age; } Sub.prototype = new Super(); Sub.prototype.sayAge = function () { console.log(this.age); } 原型继承和构造继承的组合 先将父类的实例作为子类的原型， 子类再使用call复制父类的属性 优点： 父类的方法可以被复用， 父类的引用属性不会被共享，子类构建实例是可以父类传递参数 缺点： 调用了两次父类的构造函数，第一次给子类的原型添加了父类的name, arr属性，第二次又给子类的构造函数添加了父类的name, arr属性，从而覆盖了子类原型中的同名参数。这种被覆盖的情况造成了性能上的浪费。
4. 寄生组合继承 避免组合继承的缺点 实现： inheritPrototype(Sub, Super) { var prototype = Object.create(Super.prototype);创建一个父类的副本 prototype.constructor = Sub;为副本添加 constructor属性， 弥补重写原型失去的constructor属性 Sub.prototype = prototype } function Super(name) { this.name = name; } Super.prototype.sayName = function(){ console.log(this.name) } function Sub(name, age){ this.age = age; Super.call(this, name) } inheritPrototype(Sub, Super) Sub.prototype.sayAge = function() { console.log(this.age); }
5. 实例继承 function Animal(){}; function Cat(name) { const instance = new Animal(); instance.name = name; return instance; }
6. 拷贝继承 function Cat(name) { const animal = new Animal(); for (let p in animal) { Cat.prototype[p] = animal[p]; } Cat.prototype.name = name; }
7. es6 class extends 语法糖 本质与寄生组合相似 class Animal(){ } class Cat extends Animal() { construct (name) { this.name = name; } }

- 异步
    - promise
    - async await

- 实现对象的遍历 使用obj.hasOwnProperty(key) 来判断是否是对象的属性 Object.keys() 获取对象所有属性的key 是一个数组 使用for in 遍历key for of 遍历每一项 Object.getOwnPropertyNames(obj) 对象的方法获取对象所有的属性，不包括prototype的属性

- js查找元素 出了getElementById要加 s document.getElementById() document.getElementsByClass(); document.getElementsByTagName(); document.getElementsByName()

- queryselector 参数为选择器 querySelector() 当有多个时， 返回第一个匹配的 querySelectorAll() 以node list节点列表的形式返回所有的 接受选择符，

- js怎么实现进制转换 利用栈的特性，和递归 parseInt(10, 2) 将10转为2进制

- parseInt 类型转换， 怎么将字符串转换为number parseInt， parseFloat， number *（乘法）- / 都可以将字符转为数组
- 闭包 和 应用场景 由于函数的作用域问题需要用到，闭包 闭包指的是能够访问自由变量的函数 自由变量是可以在函数中使用，但是既不是函数的变量也不是参数。
- 作用域 变量作用域两种：全局变量、局部变量。js中函数内部可以读取全局变量，函数外部不能读取函数内部的局部变量 这是就应该到闭包来获取函数内部的值
- 使得对象的属性无法改变
- cookie localStore 和 sessionStore cookie: 由服务器生成 默认是关闭浏览器后失效， 可以设置失效时间 用来保存用户信息，用来给服务器辨别身份，只能保存4kb的数据 localStore: 用来本地保存数据， 可以永久保存数据， 可以保存5M的数据。 sessionStore: 跟localStore差不多也是用来保存数据的，但是只是将数据在一次会话中保存。session结束后将会清除sessionStore中的数据。
- null等于null吗 undefined等于undefined吗
- symbol 的作用 有象征性的独一无二的 作为标签
- 返回对象的可枚举属性和方法的名称数组。 Object.getOwnPropertyNames
- 怎么看待微信小程序
- typescript和原生js
- == 和 ===
- es6 const let 模板字面量`${}` 解构[x, y, z] = [1, 2, 3] 展开运算符[...arr] 可变参数 展开运算符作为函数参数用，就相当于把所有的参数都打包起来 对象字面量简写 for of 不能迭代对象 箭头函数 类 class 语法糖 super extends 语法糖
- 实现轮询 宏任务， 微任务
- 箭头函数
- this
- 异步
- call 和 apply有什么区别 apply 第一个参数是以一个对象， 第二个参数是数组 call 第一个是对象，后面则是一串参数
- 原型链 对象和原型都有原型，对象的原型就是原型对象， 原型链对象的原型也是一个对象，原型也有原型 prototype函数才有的属性 __proto__对象具有的属性，js万物皆对象，所以会形成一条原型链到Object而Object.**proto**=null __proto__可以理解为构造器的原型 __proto__指向prototype
  https://www.cnblogs.com/wyaocn/p/5815761.html
  https://www.cnblogs.com/shuiyi/p/5305435.html
- 事件机制
- 模板引擎
- SEO 搜索引擎优化

    - 变量提升
    - 如何判断数据类型 typeof 和 instanceof
    - 并发 JavaScript 用不阻塞， 处理IO通常由事件和回调函数来进行实现
- html 重绘 回流 对语义化的理解
    - 有利于团队的开发和维护，方便其他设备解析
- 浏览器 UDP TCP的区别 同源策略 跨域
- http https://www.cnblogs.com/ranyonsue/p/5984001.html
- 七层协议四层协议
- react
    - 为什么使用react 简单：仅仅表达应用在任何一个时间点该呈现的样子，当数据改变时React会自动处理用户界面的更新 组件在传递属性为函数的时候需要接收参数不能直接加括号this.handle() 需要使用函数来包裹 如 (param) => { this.handle(param) } 工作原理 React创建一个虚拟DOM。当组件中的状态改变时，通过diff算法来标记虚拟DOM中的改变， 第二步是调节(reconciliation)，会用diff的结果来更新真实DOM state 一个状态机，根据数据改变更新视图 state是组件自己管理的数据，控制自己的状态，相对组件自己来说是可变； props 从组件外部传入组件内部的数据，一般就是父子组件的传递，可读和不可变，
    - redux的实现原理 只是同步状态，可以使用redux-thunk或者saga redux将整个应用状态储存到store里面，其实就是一颗状态树 store中每一个state对应一个View需要修改状态的时候只能通过dispatch派发一个action，然后reducer通过action来修改store里面的数据，并且使用subscribe发布订阅者模式将监听函数放进数组，当通过dispatch派发action获得新的state也就是状态后将监听函数再执行一次，就是重新渲染获取新的状态 组件可以dispatch派发action行为给store，当reducer接收到action根据执行响应的操作，修改数据，数据修改后react组件将会重新渲染 组件来通过订阅store中的状态state来刷新视图 redux3三大原则 唯一数据源 保持只读状态 组件修改数据需要通过dispatch派发action，然后执行对应的reducer来改变数据 store中一个state对应一个View
    - react声命周期 https://images2015.cnblogs.com/blog/588767/201612/588767-20161205190022429-1074951616.jpg constructor getDefaultProps getInitialState componentWillMount 多用于根组件中的应用程序配置 render 生成虚拟的DOM节点，然后将节点渲染到页面上 不应该使用setState componentDidMount 在这可以完成所有没有 DOM 就不能做的所有配置，事件监听，数据获取 React16更换了渲染框架，使用的是异步渲染，导致render之前的生命周期函数可能会被执行多次，这样就会请求多次服务器资源，产生性能问题，所以才要求把请求放到componentDidMount中 发生改变 componentWillReceiveProps 只改变state没有这一过程 shouldComponentUpdate 这里可以改善性能，通过重写改函数阻止不必要的渲染。 ture 进行更新 componentWillUpdate 它可以用于代替组件的componentWillReceiveProps和shouldComponentUpdate但不能访问之前的props render componentDidUpdate 常用于更新DOM，响应 prop 或 state 的改变 false 不更新 组件卸载时触发 UnMount componentWillMount 在这你可以取消网络请求，或者移除所有与组件相关的事件监听器 结束
    - router 路由其实就是保住视图和URL的同步，用户通过手动输入或者页面交互来改变URL，然后通过同步或者异步来向服务器发送请求 通过对应的URL来渲染对应的视图 其实就是react的组件

history 对象是整个路由系统的核心 hash hashChange withRouter 高阶组件通过context来给没有经过Route渲染的组件提供 history， match， location

- 性能优化
  - 使用 production 版本的react.js
  - 使用key来帮助React识别列表中所有子组件的最小变化。
  - 在传递属性的时候，函数在组件的构造函数中使用bind绑定this 因为在构造函数中绑定只会渲染一次不会每次都渲染 其他两种方法每次执行render()的时候都会绑定一次
  - 在组件传递属性的时候应该先定义在使用，否则每次使用子组件时都会生成的对象 例如 <component style={ {color: 'red'} }> 应该为 const style={ color: 'red' } <component style={ style }>
  - pureComponent 只是浅比较 没有内部状态时使用 通过重写shouldComponentUpdate来优化 当props/state 和 nextProps和nextState 一致则返回false 组件不更新 否则组件更新 immutableJS 进行深层比较 然后使用is()函数来比较 两个immutable数据是否相同 在遍历的时候 key 不要使用index 索引 因为当遍历的时候顺序不一样会导致key的变化，会造成性能浪费
- setState 不能保证同步 在合成事件和钩子函数中是异步的，在原生事件和setTimeout中是同步的
  - 合成事件 onChange onClick 为了避免DOM事件的滥用导致性能受影响，屏蔽不同浏览器之间底层的差异 就是将DOM事件进行了一个封装 在document处监听所有支持的事件，当事件发生并冒泡至document处时，React将事件内容封装交给中间层SyntheticEvent（负责所有事件合成）当事件触发的时候，对使用统一的分发函数dispatchEvent将指定函数执行。
  - 钩子函数 生命周期函数 hooks 本质上就是一类特殊的函数

  合成事件和钩子函数的调用顺序在更新之前，导致合成事件和钩子函数没办法拿到更新后的值 父组件传递给子组件state的时候，props是不能同步刷新的re-render不能同步刷新 不能保证同步执行 是性能优化 调用setState的时候并不会立马修改state，而是把需要修改的状态放在一个队列中， React会优化真正的执行机制，并且出于性能原因会将多次setState合并成一次修改，setState会在最后批量执行 保证数据统一
- redux-thunk中间件 处理异步操作 创建的action函数返回可以是一个函数参数为dispatch，getState，然后在函数里面dispatch一个action function add() { return { type: 'ADD', } } function addIfOdd() { return (dispatch, getState) => { const currentValue = getState(); if (currentValue % 2 == 0) { return false; } //分发一个任务 dispatch(add()) } }
- 可控组件 维护自身状态 组件的状态都是有组件自己来控制，也就是状态都是state中，而不是由DOM来控制
- react element 和component 有什么区别 element描述了UI，是一些对UI对象表示，而component是一个函数或者类，接受输入和返回一个react element
- refs Refs 是 React 提供给我们的安全访问 DOM 元素或者某个组件实例的句柄。我们可以为元素添加ref属性然后在回调函数中接受该元素在 DOM 树中的句柄，该值会作为回调函数的第一个参数返回
- keys Keys 是 React 用于追踪哪些列表中元素被修改、被添加或者被移除的辅助标识。 Diff 算法中 React 会借助元素的 Key 值来判断该元素是新近创建的还是被移动而来的元素，从而减少不必要的元素重渲染，有助于提高性能， 每个key在兄弟元素间是独一无二的，并且在map的过程中最好不

要使用index作为key，因为当顺序改变时key发生了改变，React会认为数据发生了改变，就会重新渲染，造成性能浪费。

- 虚拟dom 使用js对象来模拟真实的DOM树，数据更新的时候创建新的虚拟DOM通过新旧对比来获取差异，然后通过特定的render将差异的虚拟DOM渲染成真实的DOM
- diff算法 通过对比虚拟DOM和新的虚拟DOM 传统算法 复杂度为n的3次方 只对同级的节点进行比较 比较组件类型 组件名字 同一层级的节点通过key来区分 合并操作，调用component的setState的时候，将其标记为dirty，到每一个事件循环结束，react检查所有的dirty的component重新绘制选择性渲染子树，可以通过重写shouldComponentUpdate来提高性能
- 类组件和函数组件 类组件有更多额外的功能，如组件自身状态和生命周期函数等，也能使组件访问store并维持状态 当组件仅仅是接受的props，并将组件自身渲染到页面，该组件就是一个无状态组件，这时可以使用函数来创建这样的组件
- react 新特性
  - hooks 就是在 react 函数组件中，也可以使用类组件（classes components）的 state 和 组件生命周期，而不需要在 mixin、 函数组件、HOC组件和 render props 之间来回切换，使得函数组件的功能更加实在，更加方便我们在业务中实现业务逻辑代码的分离和组件的复用。 useState 可以为函数组件提供state useEffect 提供类似componentDidMount的功能 useContext useReducer useCallback 为函数组件提供一些特殊的功能
  - 新的生命周期 将会删除 componentWillMount componentWillReceiveProps componentWillUpdate 添加getDerivedStateFromProps getSnapshotBeforeUpdate https://juejin.im/post/5b6f1800f265da282d45a79a#heading-12
- 阻止渲染 setState return null shouldComponentUpdate 返回false render return null
- 如何告诉React应该编译生成版本 通常使用Webpack中的DefinPlugin方法将NODE_EVN设置为production
- 性特性 错误处理 默认情况下某个组件出错，这个组件就会从组件树中卸载，而不是整个应用都需要刷新 render 返回类型增加， string boolean number null

- create-react-app postCss 自动添加前缀
- webpack
  - plugin和loader 由于webpack 只能 打包commonjs的js文件，对于其他资源无法加载，所以需要loder loader 主要用来的资源加载处理不同的文件， 作用于一种文件， 也是对webpack的扩展，但是只是转化文件 plugin 直接作用于webpack， 是对webpack的一些扩展，相当给webpack添加功能