哈喽

亲爱的小伙伴们，我想，看到这里，相信你已经收起爱玩的小心心，目不转睛的盯着电脑屏幕四个月了吧，那么现在到了我们最后的冲刺阶段。

你还在为买不起房苦恼吗？

你还在为买不起车担心吗？

你还在为没钱陪女朋友逛街痛心疾首吗？

那么、现在、此刻、葵花宝典来了！

熟读、谨记、可以带你飞的一份资料，就展现在你的眼前，还在等什么，现在就行动。拿起电话，告诉就业老师，我要！

此时不搏何时搏，人生能有机会博！

（内部资料，严禁外传）

广州前端就业小组（三剑客）

2018/7/5

一、VUE 面试题

1、Vue. js 是什么

Vue.js（是一套构建用户界面的 渐进式框架。与其他重量级框架不同的是，Vue 采用自底向上增量开发的设计。Vue 的核心库只关注视图层，并且非常容易学习，非常容易与其它库或已有项目整合。另一方面，Vue 完全有能力驱动采用单文件组件和 Vue 生态系统支持的库开发的复杂单页应用。

Vue.js 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件

2、VueJS 特性：

I: MVVM 模式（数据变量（model）发生改变 视图（view）也改变， 视图（view）改变，数据变量（model）也发生改变）

使用 MVVM 模式有几大好处：

1. 低耦合。View 可以独立于 Model 变化和修改，一个 ViewModel 可以绑定到不同的 View 上，当 View 变化的时候 Model 可以不变，当 Model 变化的时候 View 也可以不变。

2. 可重用性。可以把一些视图的逻辑放在 ViewModel 里面，让很多 View 重用这段视图逻辑。

3. 独立开发。开发人员可以专注与业务逻辑和数据的开发(ViewModel)。设计人员可以专注于界面(View)的设计。

4. 可测试性。可以针对 ViewModel 来对界面(View)进行测试

II: 组件化

III 指令系统

IIII: vue2.0 开始支持虚拟 dom

vue1.0 是操作的是真的 dom 元素而不是虚拟的

虚拟 dom:可以提升页面的刷新速度

虚拟 DOM 有利也有弊。

A :大小 - 其中之一就是更多的功能意味着代码包中更多行的代码。幸运的是，Vue.js 2.0 依旧比 较小（当前版本 21.4kb），并

且也正在删除很多东西。

B: 内存 -同样，虚拟 DOM 需要将现有的 DOM 拷贝后保存在内存中，这是一个在 DOM 更新速度和内存使用中的权衡。

C: 并不适用所有情况 -如果虚拟 DOM 可以一次性进行批量的修改是非常好的。但是如果是单独的、稀少的更新呢？这样的任何

DOM 更新都将会使虚拟 DOM 带来无意义的预计算

3. Vue. js 特点

简洁：页面由 HTML 模板+Json 数据+Vue 实例组成

数据驱动：自动计算属性和追踪依赖的模板表达式

组件化：用可复用、解耦的组件来构造页面

轻量：代码量小，不依赖其他库

快速：精确有效批量 DOM 更新

模板友好：可通过 npm，bower 等多种方式安装，很容易融入

4. vue 中的 MVVM 模式

即 Model-View-ViewModel。

Vue 是以数据为驱动的，Vue 自身将 DOM 和数据进行绑定，一旦创建绑定，DOM 和数据将保持同步，每当数据发生变化，DOM 会跟着变化。

ViewModel 是 Vue 的核心，它是 Vue 的一个实例。Vue 实例时作用域某个 HTML 元素上的，这个 HTML 元素可以是 body，也可以是某个 id 所指代的元素。

DOM Listeners 和 Data Bindings 是实现双向绑定的关键。DOM Listeners 监听页面所有 View 层 DOM 元素的变化，当发生变化，Model 层的数据随之变化；Data Bindings 监听 Model 层的数据，当数据发生变化，View 层的 DOM 元素随之变化。

5. v-show 指令，v-if 的区别

条件渲染指令，与 v-if 不同的是，无论 v-show 的值为 true 或 false，元素都会存在于 HTML 代码中；而只有当 v-if 的值为 true，元素才会存在于 HTML 代码中。v-show 指令只是设置了元素 CSS 的 style 值

6. 如何让 css 只在当前组件中起作用

在每一个 vue 组件中都可以定义各自的 css，js，如果希望组件内写的 css 只对当前组件起作用，只需要在 style 中写入 scoped，即：

<style scoped></style>

7. 指令 keep-alive

在 vue-router 写着 keep-alive，keep-alive 的含义：

如果把切换出去的组件保留在内存中，可以保留它的状态或避免重新渲染。为此可以添加一个 keep-alive 指令

<component :is='curremtView' keep-alive></component>

8. 路由嵌套

路由嵌套会将其他组件渲染到该组件内，而不是进行整个页面跳转 router-view 本身就是将组件渲染到该位置，想要进行页面跳转，就要将页面渲染到根组件，在起始配置路由时候写到：

var App = Vue.extend({ root });

router.start(App,'#app');

这里首先将根组件注册进来，用于将路由中配置好的各个页面渲染出来，然后将根组件挂载到与#app 匹配的元素上。

9. vuejs 中使用事件名

在 vuejs 中，我们经常要绑定一些事件，有时候给 DOM 元素绑定，有时候给组件绑定。绑定事件在 HTML 中用 v-on:click-"event",这时 evet 的名字不要出现大写，因为在 1.x 中不区分大小写，所以如果我们在 HTML 写 v-on:click="myEvent"而在 js 中写 myEvent 就出错误，所以在 vuejs 的 1.x 绑定事件时候，要尽量避免使用大写字母。在 2.0 中没有该限制！

10. Vue.js 和 AngularJS 之间的区别是什么？

1.vue 仅仅是 mvvm 中的 view 层，只是一个如 jquery 般的工具库，而不是框架，而 angular 而是 mvvm 框架。

2.vue 的双向邦定是基于 ES5 中的 3.getter/setter 来实现的，而 angular 而是由自己实现一套模版编译规则，需要进行所谓的“脏”检查，vue 则不需要。因此，vue 在性能上更高效，但是代价是对于 ie9 以下的浏览器无法支持。

4.vue 需要提供一个 el 对象进行实例化，后续的所有作用范围也是在 el 对象之下，而 angular 而是整个 html 页面。一个页面，可以有多个 vue 实例，而 angular 好像不是这么玩的。

5.vue 真的很容易上手，学习成本相对低，不过可以参考的资料不是很丰富，官方文档比较简单，缺少全面的使用案例。高级的用法，需要自己去研究源码，至少目前是这样。

12、嵌套路由怎么定义？

在实际项目中我们会碰到多层嵌套的组件组合而成，但是我们如何实现嵌套路由呢？因此我们需要在 VueRouter 的参数中使用 children 配置，这样就可以很好的实现路由嵌套。

index.html，只有一个路由出口

```
<div id="app">
    <!-- router-view 路由出口，路由匹配到的组件将渲染在这里 -->
    <router-view></router-view>
</div>
```

14、active-class 是哪个组件的属性?嵌套路由怎么定义?

答:vue-router 模块的 router-link 组件。

15、怎么定义 vue-router 的动态路由?怎么获取传过来的动态参数?

答:在 router 目录下的 index.js 文件中，对 path 属性加上/:id。 的 params.id

使用 router 对象

16、vue-router 有哪几种导航钩子？

三种，

第一种：是全局导航钩子：router.beforeEach(to,from,next)，作用：跳转前进行判断拦截。

第二种：组件内的钩子

第三种：单独路由独享组件

17、scss 是什么？在 vue.cli 中的安装使用步骤是？有哪几大特性？

css 的预编译。

使用步骤：

第一步：用 npm 下三个 loader（sass-loader、css-loader、node-sass）

第二步：在 build 目录找到 webpack.base.config.js，在那个 extends 属性中加一个拓展.scss

第三步：还是在同一个文件，配置一个 module 属性

第四步：然后在组件的 style 标签加上 lang 属性 ，例如：lang=" scss"

有哪几大特性:

可以用变量，例如（$变量名称=值）；

2、可以用混合器，例如（）

3、可以嵌套

18、scss 是什么?在 vue.cli 中的安装使用步骤是?有哪几大特性?

答:css 的预编译。

使用步骤:

第一步:用 npm 下三个 loader(sass-loader、css-loader、node-sass)

第二步:在 build 目录找到 webpack.base.config.js，在那个 extends 属性中加一个 拓展.scss

第三步:还是在同一个文件，配置一个 module 属性 第四步:然后在组件的 style 标签加上 lang 属性 ，例如:lang="scss" 有哪几大特性：1、可以用变量，例如($变量名称=值);

2、可以用混合器，例如() 3、可以嵌套

19、mint-ui 是什么？怎么使用？说出至少三个组件使用方法？

基于 vue 的前端组件库。npm 安装，然后 import 样式和 js，vue.use（mintUi）全局引入。在单个组件局部引入：import {Toast} from ‘mint-ui’ 。

组件一：Toast(‘登录成功’)；

组件二：mint-header；

组件三：mint-swiper

20、v-model 是什么?怎么使用? vue 中标签怎么绑定事件?

答:可以实现双向绑定，指令(v-class、v-for、v-if、v-show、v-on)。vue 的

model 层的 data 属性。绑定事件: `<input @click=doLog()/>`

23、axios 是什么?怎么使用?描述使用它实现登录功能的流程?

答:请求后台资源的模块。npm install axios -S 装好，然后发送的是跨域，需在配置文 件中 config/index.js 进行设置。后台如果是 Tp5 则定义一个资源路由。js 中使用 import 进来，然后.get 或.post。返回在.then 函数中如果成功，失败则是在.catch 函 数中

24、axios+tp5 进阶中，调用 axios.post(‘api/user’)是进行的什么操作? axios.put(‘api/user/8′)呢?

答:跨域，添加用户操作，更新操作。

25、什么是 RESTful API?怎么使用?

答:是一个 api 的标准，无状态请求。请求的路由地址是固定的，如果是 tp5 则先路由配置

中把资源路由配置好。标准有:.post .put .delete

26、vuex 是什么？怎么使用？哪种功能场景使用它？

vue 框架中状态管理。在 main.js 引入 store，注入。新建了一个目录 store，….. export 。场景有：单页应用中，组件之间的状态。音乐播放、登录状态、加入购物车

27、mvvm 框架是什么？它和其它框架（jquery）的区别是什么？哪些场景适合？

一个 model+view+viewModel 框架，数据模型 model，viewModel 连接两个

区别：vue 数据驱动，通过数据来显示视图层而不是节点操作。

场景：数据操作比较多的场景，更加便捷

28、mvvm 框架是什么?它和其它框架(jquery)的区别是什么?哪些场景适合?

答:一个 model+view+viewModel 框架，数据模型 model，viewModel 连接两个

区别:vue 数据驱动，通过数据来显示视图层而不是节点操作。

场景:数据操作比较多的场景，更加便捷

29、自定义指令(v-check、v-focus)的方法有哪些?它有哪些钩子函数?还有哪些钩子 函数参数?

答:全局定义指令:在 vue 对象的 directive 方法里面有两个参数，一个是指令名称，另外 一个是函数。组件内定义指令:directives

钩子函数:bind(绑定事件触发)、inserted(节点插入的时候触发)、update(组件内相 关更新)

钩子函数参数:el、binding

30、说出至少 4 种 vue 当中的指令和它的用法?

答:v-if:判断是否隐藏;v-for:数据循环出来;v-bind:class:绑定一个属性;v- model:实现双向绑定

31、vue-router 是什么?它有哪些组件?

答:vue 用来写路由一个插件。router-link、router-view

32、Vue 的双向数据绑定原理是什么?

答:vue.js 是采用数据劫持结合发布者-订阅者模式的方式，通过 Object.defineProperty()来劫持各个属性的 setter，getter，在数据变动时发布消息 给订阅者，触发相应的监听回调。

具体步骤: 第一步:需要 observe 的数据对象进行递归遍历，包括子属性对象的属性，都加上 setter

和 getter 这样的话，给这个对象的某个值赋值，就会触发 setter，那么就能监听到了数据变化

第二步:compile 解析模板指令，将模板中的变量替换成数据，然后初始化渲染页面视图，并 将每个指令对应的节点绑定更新函数，添加监听数据的订阅者，一旦数据有变动，收到通知， 更新视图

第三步:Watcher 订阅者是 Observer 和 Compile 之间通信的桥梁，主要做的事情是:

1、在自身实例化时往属性订阅器(dep)里面添加自己

2、自身必须有一个 update()方法

3、待属性变动 dep.notice()通知时，能调用自身的 update()方法，并触发 Compile 中 绑定的回调，则功成身退。

第四步:MVVM 作为数据绑定的入口，整合 Observer、Compile 和 Watcher 三者，通过 Observer 来监听自己的 model 数据变化，通过 Compile 来解析编译模板指令，最终利用 Watcher 搭起 Observer 和 Compile 之间的通信桥梁，达到数据变化 -> 视图更新;视图 交互变化(input) -> 数据 model 变更的双向绑定效果。

33、请详细说下你对 vue 生命周期的理解?

答:总共分为 8 个阶段创建前/后，载入前/后，更新前/后，销毁前/后。

创建前/后:在 beforeCreated 阶段，vue 实例的挂载元素$el 和数据对象 data 都为 undefined，还未初始化。在 created 阶段，vue 实例的数据对象 data 有了，$el 还没 有。

载入前/后:在 beforeMount 阶段，vue 实例的$el 和 data 都初始化了，但还是挂载之前 为虚拟的 dom 节点，data.message 还未替换。在 mounted 阶段，vue 实例挂载完成， data.message 成功渲染。

更新前/后:当 data 变化时，会触发 beforeUpdate 和 updated 方法。 **销毁前/后:**在执行 destroy 方法后，对 data 的改变不会再触发周期函数，说明此时 vue

实例已经解除了事件监听以及和 dom 的绑定，但是 dom 结构依然存在

34、你是怎么认识 vuex 的？

vuex 可以理解为一种开发模式或框架。比如 PHP 有 thinkphp，java 有 spring 等。
通过状态（数据源）集中管理驱动组件的变化（好比 spring 的 IOC 容器对 bean 进行集中管理）。

应用级的状态集中放在 store 中； 改变状态的方式是提交 mutations，这是个同步的事物； 异步逻辑应该封装在 action 中。

35、请说下封装 vue 组件的过程?

答:首先，组件可以提升整个项目的开发效率。能够把页面抽象成多个相对独立的模块，解决 了我们传统项目开发:**效率低**、**难维护**、**复用性**等问题。 然后，使用 Vue.extend 方法创建一个组件，然后使用 Vue.component 方法注册组件。子 组件需要数据，可以在 props 中接受定义。而子组件修改好数据后，想把数据传递给父组 件。可以采用 emit 方法。

36、你是怎么认识 vuex 的?

答:vuex 可以理解为一种开发模式或框架。比如 PHP 有 thinkphp，java 有 spring 等。

通过状态(数据源)集中管理驱动组件的变化(好比 spring 的 IOC 容器对 bean 进行集中管 理)。

应用级的状态集中放在 store 中;改变状态的方式是提交 mutations，这是个同步的事 物;异步逻辑应该封装在 action 中。

37、vue-loader 是什么?使用它的用途有哪些?

答:解析.vue 文件的一个加载器,跟 template/js/style 转换成 js 模块。用途:js 可以写 es6、style 样式可以 scss 或 less、template 可以加 jade 等

38、请说出 vue.cli 项目中 src 目录每个文件夹和文件的用法?

答:assets 文件夹是放静态资源;components 是放组件;router 是定义路由相关的配置;view 视图;app.vue 是一个应用主组件;main.js 是入口文件

39、聊聊你对 Vue.js 的 template 编译的理解？

简而言之，就是先转化成 AST 树，再得到的 render 函数返回 VNode（Vue 的虚拟 DOM 节点）

详情步骤：

首先，通过 compile 编译器把 template 编译成 AST 语法树（abstract syntax tree 即 源代码的抽象语法结构的树状表现形式），compile 是 createCompiler 的返回值，createCompiler 是用以创建编译器的。另外 compile 还负责合并 option。

然后，AST 会经过 generate（将 AST 语法树转化成 render funtion 字符串的过程）得到 render 函数，render 的返回值是 VNode，VNode 是 Vue 的虚拟 DOM 节点，里面有（标签名、子节点、文本等等）

40、vue 的历史记录

history 记录中向前或者后退多少步

41、出 vue.cli 项目中 src 目录每个文件夹和文件的用法？

assets 文件夹是放静态资源；components 是放组件；router 是定义路由相关的配置;view 视图；app.vue 是一个应用主组件；main.js 是入口文件

42、vuejs 与 angularjs 以及 react 的区别？

1.与 AngularJS 的区别

相同点：

都支持指令：内置指令和自定义指令。

都支持过滤器：内置过滤器和自定义过滤器。

都支持双向数据绑定。

都不支持低端浏览器。

不同点：

1.AngularJS 的学习成本高，比如增加了 Dependency Injection 特性，而 Vue.js 本身提供的 API 都比较简单、直观。

2.在性能上，AngularJS 依赖对数据做脏检查，所以 Watcher 越多越慢。

Vue.js 使用基于依赖追踪的观察并且使用异步队列更新。所有的数据都是独立触发的。

对于庞大的应用来说，这个优化差异还是比较明显的。

2.与 React 的区别

相同点：

React 采用特殊的 JSX 语法，Vue.js 在组件开发中也推崇编写.vue 特殊文件格式，对文件内容都有一些约定，两者都需要编译后使用。

中心思想相同：一切都是组件，组件实例之间可以嵌套。

都提供合理的钩子函数，可以让开发者定制化地去处理需求。

都不内置列数 AJAX，Route 等功能到核心包，而是以插件的方式加载。

在组件开发中都支持 mixins 的特性。

不同点：

React 依赖 Virtual DOM,而 Vue.js 使用的是 DOM 模板。React 采用的 Virtual DOM 会对渲染出来的结果做脏检查。

Vue.js 在模板中提供了指令，过滤器等，可以非常方便，快捷地操作 DOM。

43、vue.cli 中怎样使用自定义的组件？有遇到过哪些问题吗？

第一步：在 components 目录新建你的组件文件（smithButton.vue），script 一定要 export default {

第二步：在需要用的页面（组件）中导入：import smithButton from ‘../components/smithButton.vue’

第三步：注入到 vue 的子组件的 components 属性上面,components:{smithButton}

第四步：在 template 视图 view 中使用，<smith-button> </smith-button>

问题有：smithButton 命名，使用的时候则 smith-button。

44、vue.cli 中怎样使用自定义的组件?有遇到过哪些问题吗?

答:第一步:在 components 目录新建你的组件文件(smithButton.vue)，script 一定 要 export default {

第二步:在需要用的页面(组件)中导入:import smithButton from '../components/smithButton.vue'

第三步:注入到 vue 的子组件的 components 属性上面,components:{smithButton} 第四步:在 template 视图 view 中使用， 问题有:smithButton 命名，使用的时候则 smith-button。

45、聊聊你对 Vue.js 的 template 编译的理解?

答:简而言之，就是先转化成 AST 树，再得到的 render 函数返回 VNode(Vue 的虚拟 DOM 节点)

详情步骤:

首先，通过 compile 编译器把 template 编译成 AST 语法树(abstract syntax tree 即 源代码的抽象语法结构的树状表现形式)，compile 是 createCompiler 的返回值， createCompiler 是用以创建编译器的。另外 compile 还负责合并 option。

然后，AST 会经过 generate(将 AST 语法树转化成 render funtion 字符串的过程)得到 render 函数，render 的返回值是 VNode，VNode 是 Vue 的虚拟 DOM 节点，里面有(标签 名、子节点、文本等等)

46、vue 组件封装的过程

1、 首先，使用 Vue.extend()创建一个组件

2、然后，使用 Vue.component()方法注册组件

3、接着，如果子组件需要数据，可以在 props 中接受定义

4、最后，子组件修改好数据之后，想把数据传递给父组件，可以使用 emit()方法

47、什么是 vue 的虚拟 dom

虚拟的DOM的核心思想是：对复杂的文档DOM结构，提供一种方便的工具，进行最小化地DOM操作。

48、vue 怎么添加自定义事件？

自定义事件一般用于自定义组件的，img 标签很显然不具备你需要的这些事件，解决方案是封装为一个自定义组件，实现并开放这几个自定义事件。

```
Vue.component('custom-img', {
  template: '<img :src="src" @touchmove="ontouchmove" />',
  props: {
    src: String
  },
  methods: {
    ontouchmove: function (e) {
      // if (...) 实现判断 slideLeft 的逻辑
      {
        this.$emit('slideLeft');
      }
    }
  }
});
```

调用之处：

```
<custom-img :src="list.src" @slide-left="onSlideLeft"></custom-img>
```

49、数组和对象的什么操作不会在 vue 反映

由于 JavaScript 的限制， Vue 不能检测以下变动的数组：

1 当你利用索引直接设置一个项时，例如： vm.items[indexOfItem] = newValue

2 当你修改数组的长度时，例如： vm.items.length = newLength

50、异步组件

vue 作为一个轻量级前端框架，其核心就是组件化开发。我们一般常用的是用脚手架 vue-cli 来进行开发和管理，一个个组件即为一个个 vue 页面，这种叫单文件组件。我们在引用组件之时只需将组件页面引入，再注册即可使用。

当项目比较大型，结构比较复杂时，我们一般选用 vue-cli 脚手架去构建项目。因为 vue-cli 集成了 webpack 环境，使用单文件组件，开发更简单，易上手，尤其是在对组件的处理上。对于原生 vue.js，我们就得将组件构建在同一个 html 的 script 标签下或者 html 的外部 js 中，所有组件集中在一块，不容易管理，这也是原生 vue.js 的一点不便之处

VUE 及 EELEMENT 使用过程中遇到的一些问题

在做项目的过程中，目前主要遇到了以下几个问题：

一.样式问题

1. 样式中使用 scoped 的问题：

主要表现在从一个页面跳到另一个页面时，第二个页面的样式不能正确显示，通过刷新才能恢复页面的预定样式。
究其原因，是因为如果浏览器在加载了上一个页面的样式时，在跳到第二个页面时，css 样式文件是懒加载的，上一
个页面的样式还是会作用到当前的页面，尤其是使用了组件框架时，当大量的以标签作为选择器的样式存在时，就会
使页面产生混乱。

2. 解决方案

<1>所有页面统一在 style 标签内使用 scoped 属性，避免页面间的相互影响。
添加 scoped 之后，实际上 vue 在背后做的工作是将当前组件的节点添加一个像 data-v-1233 这样唯一属性的标识，当然也会

给当前 style 的所有样式添加[data-v-1233]。这样的话，就可以使得当前样式只作用于当前组件的节点。

优点：可以有限避免页面样式错乱的问题

缺点：代码复用率低

<2>使用统一的公共样式，在公共样式的基础上给页面添加单独样式。

<3>适当使用行内标签

二. 过滤器的问题

在使用 Vue 过滤器的过程中出现了这样一个问题，需要将电话号码格式化为 344 的格式，在过滤器中这样写

```
formatTelNum (value){
    let temp = value.split("").....
    ……
}
```

一运行就会报 undefined 的错误，解决的方法是在 method 中定义格式化电话号码的方法。

三. vue 父子组件通讯中传值得一些问题

在 Vue 父子组件通讯的过程中可能遇到下面的问题：

当父组件中想要传递给子组件的值为异步获取到的时，即使在父组件的 created（mounted）中就调用这个异步函数，在子

组件 beforeUpdated 钩子前都是拿不到这个传递过来的值得，如果想要在子组件页面渲染之前就要使用到这个值，必须在子组件的

标签中使用 v-if 或者 v-show。在父组件想要传递的值还没有获取到之前，使子组件的 v-if 为 false，当异步函数完成时设置为 true，即可在

子组件的 created 钩子中拿到需要传递的值。

v-if 和 v-show 的区别：

`v-if` 是“真正”的条件渲染，因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建。

`v-if` 也是惰性的：如果在初始渲染时条件为假，则什么也不做——直到条件第一次变为真时，才会开始渲染条件块。

相比之下，v-show 就简单得多——不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 进行切换。

一般来说，v-if 有更高的切换开销，而 v-show 有更高的初始渲染开销。因此，如果需要非常频繁地切换，则使用 v-show 较好；如果在运行时条件很少改变，则使用 v-if 较好。

四. vue 钩子

beforeCreate

类型：Function

详细：

在实例初始化之后，数据观测 (data observer) 和 event/watcher 事件配置之前被调用。

created

类型：Function

详细：

在实例创建完成后被立即调用。在这一步，实例已完成以下的配置：数据观测 (data observer)，属性和方法的运算，watch/event 事件回调。然而，挂载阶段还没开始，$el 属性目前不可见。

beforeMount

类型：Function

详细：

在挂载开始之前被调用：相关的 render 函数首次被调用。

该钩子在服务器端渲染期间不被调用。

mounted

类型：Function

详细：

el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子。如果 root 实例挂载了一个文档内元素，当 mounted 被调用时 vm.$el 也在文档内。

注意 mounted 不会承诺所有的子组件也都一起被挂载。如果你希望等到整个视图都渲染完毕，可以用 vm.$nextTick 替换掉 mounted：

```
mounted: function () {
  this.$nextTick(function () {
  // Code that will run only after the
  // entire view has been rendered
  })
}
```

该钩子在服务器端渲染期间不被调用。

beforeUpdate

类型：`Function`

详细：

数据更新时调用，发生在虚拟 DOM 重新渲染和打补丁之前。

你可以在这个钩子中进一步地更改状态，这不会触发附加的重渲染过程。

该钩子在服务器端渲染期间不被调用。

updated

类型：`Function`

详细：

由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子。

当这个钩子被调用时，组件 DOM 已经更新，所以你现在可以执行依赖于 DOM 的操作。然而在大多数情况下，你应该避免在此期间更改状态。如果要相应状态改变，通常最好使用计算属性或 watcher 取而代之。

注意 `updated` 不会承诺所有的子组件也都一起被重绘。如果你希望等到整个视图都重绘完毕，可以用 vm.$nextTick 替换掉 `updated`：

```
updated: function () {
  this.$nextTick(function () {
  // Code that will run only after the
  // entire view has been re-rendered
  })
}
```

该钩子在服务器端渲染期间不被调用。

activated

类型：`Function`

详细：

keep-alive 组件激活时调用。

该钩子在服务器端渲染期间不被调用。

deactivated

类型：`Function`

详细：

keep-alive 组件停用时调用。

该钩子在服务器端渲染期间不被调用。

beforeDestroy

类型：`Function`

详细：

实例销毁之前调用。在这一步，实例仍然完全可用。

该钩子在服务器端渲染期间不被调用。

destroyed

类型：`Function`

详细：

Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。

该钩子在服务器端渲染期间不被调用。

errorCaptured

2.5.0+ 新增

类型：`(err: Error, vm: Component, info: string) => ?boolean`

详细：

当捕获一个来自子孙组件的错误时被调用。此钩子会收到三个参数：错误对象、发生错误的组件实例以及一个包含错误来源信息的字符串。此钩子可以返回 `false` 以阻止该错误继续向上传播。

你可以在此钩子中修改组件的状态。因此在模板或渲染函数中设置其它内容的短路条件非常重要，它可以防止当一个错误被捕获时该组件进入一个无限的渲染循环。

错误传播规则

默认情况下，如果全局的 `config.errorHandler` 被定义，所有的错误仍会发送它，因此这些错误仍让会向单一的分析服务的地方进行汇报。

如果一个组件的继承或父级从属链路中存在多个 `errorCaptured` 钩子，则它们将会被相同的错误逐个唤起。

如果此 errorCaptured 钩子自身抛出了一个错误，则这个新错误和原本被捕获的错误都会发送给全局的 config.errorHandler。

一个 errorCaptured 钩子能够返回 false 以阻止错误继续向上传播。本质上是说“这个错误已经被搞定了且应该被忽略”。它会阻止其它任何会被这个错误唤起的 errorCaptured 钩子和全局的 config.errorHandler。

五.、$nextTick

这个问题可以说是最容易被忽略有最容易令人焦头烂额的了，先上案例：

```
<el-table
  stripe
  ref="multipleTable"
  :data="vendorList"
  tooltip-effect="dark"
  style="width: 100%"
  @select-all="selectAll"
  @selection-change="handleSelectionChange">
      <el-table-column type="selection">
      </el-table-column>
      <el-table-column prop="vendorNum" label="设备编号">
      </el-table-column>
      <el-table-column prop="vendorModelName" label="设备型号">
      </el-table-column>
      <el-table-column prop="nickName" label="点位昵称" show-overflow-tooltip>
      </el-table-column>
      <el-table-column label="当前版本">


      </el-table-column>
</el-table>
```

现有以上这段代码，表格第一列是 CheckBox，场景是在表格数据载入之前，和现有数据（vendorNum）比对，如果
vendorNum 存在，在相应的 CheckBox 打上对号。实现的方法很简单，就是在请求回来数据之后调用这个函数：

```
//通过和请求到数据比较 vendorId,设置相应状态为选中
setSelectedStatus(){
    let temp = {};
    this.selectedVendorList.vendorNumIds.forEach(item=>{
        temp[item.num] = item;
        this.multipleSelection.push({vendorNum:item.num,vendorId:item.id});
    });
    for(let i = 0, l = this.vendorList.length; i < l; i++){
        let res = this.vendorList[i];
        if(temp[res.vendorNum]){
            this.$refs.multipleTable.toggleRowSelection(this.vendorList[i]);
        }
    }
},
```

这样做的结果是，相应的选项 CheckBox 是选中的状态，但渲染的结果是未选中。这就要用到 VUE 里的 nextTick 了。

Vue 实现响应式并不是数据发生变化之后 DOM 立即变化，而是按一定的策略进行 DOM 的更新。

$nextTick 是在下次 DOM 更新循环结束之后执行延迟回调，在修改数据之后使用 $nextTick，则可以在回调中获取更新后的 DOM。

于是解决的方法如家下：

```
const list = await getVendorList(data);
if(list.status){
    this.vendorList = list.data.items;
    this.totalPage = list.data.totalCount;
}else{
    this.$message({
        type:"error",
        message:"数据获取失败"
 })
```

```
}
//保证在 DOM 更新之后进行数据比对
this.$nextTick(function(){
    this.setSelectedStatus();
});
```

运行，一切 OK！

六.Popover 弹出框

官方文档使用方法如下：

```
<el-popover ref="popover1" placement="top-start" title="标题" width="200" trigger="hover" content="这是一段内容,这是一段内容,这是一段内容,这是一段内容。"> </el-popover>

<el-button v-popover:popover1>hover 激活</el-button>
```

通常情况狂下这种用法中规中矩，没有任何问题，但是在表格中，尤其是需要渲染一个数组的数据时用这种方法就会产生一大堆报错，因为和 ref 和 id 一样，都是独一无二的。此时

需要用如下方法：

```
<el-table-column label="商品名称">
    <template slot-scope="scope">
        <el-popover
 placement="right"
 width="400"
 trigger="hover">
            <el-table :data="scope.row.groups">
                <el-table-column width="150" label="商品">
```

```
                <template slot-scope="scope1">
                    <span                    style="cursor:                    pointer;"
@click="linkDetails(scope1.row)">{{scope1.row.goodsGroupName}}</span>
                </template>
            </el-table-column>
            <el-table-column  width="100"  property="amount"  label="数量"></el-table-
column>
            <el-table-column  width="300"  property="productPrice"  label="价格"></el-
table-column>
          </el-table>
          <span           slot="reference"           style="cursor:           pointer;"
@click="linkDetails(scope.row)">{{scope.row.goodsGroupName}}</span>
       </el-popover>
   </template>
</el-table-column>
```

将触发 el-popover 显示的内容放在 el-popover 组件中的插槽中，用一个 reference 的具名 slot ，就可以解决这个问题。

VUE 生命周期

1、什么是 vue 生命周期？

Vue 实例从创建到销毁的过程，就是生命周期。也就是从开始创建、初始化数据、编译模板、挂载 Dom→渲染、更新→渲染、卸载等一系列过程，我们称这是 Vue 的生命周期。

2、vue 生命周期的作用是什么？

它的生命周期中有多个事件钩子，让我们在控制整个 Vue 实例的过程时更容易形成好的逻辑。

3、vue 生命周期总共有几个阶段？

它可以总共分为 8 个阶段：创建前/后，载入前/后,更新前/后,销毁前/销毁后

4、第一次页面加载会触发哪几个钩子？

第一次页面加载时会触发 beforeCreate, created, beforeMount, mounted 这几个钩子

5、DOM 渲染在 哪个周期中就已经完成？

DOM 渲染在 mounted 中就已经完成了

6、简单描述每个周期具体适合哪些场景？

生命周期钩子的一些使用方法： beforecreate：可以在这加个 loading 事件，在加载实例时触发 created：初始化完成时的事件写在这里，如在这结束 loading 事件，异步请求也适宜在这里调用 mounted：挂载元素，获取到 DOM 节点 updated：如果对数据统一处理，在这里写上相应函数 beforeDestroy：可以做一个确认停止事件的确认框 nextTick：更新数据后立即操作 dom

arguments 是一个伪数组，没有遍历接口，不能遍历

7、cancas 和 SVG 的是什么以及区别

SVG

SVG 是一种使用 XML 描述 2D 图形的语言。

SVG 基于 XML，这意味着 SVG DOM 中的每个元素都是可用的。您可以为某个元素附加 JavaScript 事件处理器。

在 SVG 中，每个被绘制的图形均被视为对象。如果 SVG 对象的属性发生变化，那么浏览器能够自动重现图形。

Canvas

Canvas 通过 JavaScript 来绘制 2D 图形。

Canvas 是逐像素进行渲染的。

在 canvas 中，一旦图形被绘制完成，它就不会继续得到浏览器的关注。如果其位

置发生变化，那么整个场景也需要重新绘制，包括任何或许已被图形覆盖的对象。

8、Canvas 与 SVG 的比较

Canvas

依赖分辨率

不支持事件处理器

弱的文本渲染能力

能够以 .png 或 .jpg 格式保存结果图像

最适合图像密集型的游戏，其中的许多对象会被频繁重绘复制代码

SVG

不依赖分辨率

支持事件处理器

最适合带有大型渲染区域的应用程序（比如谷歌地图）

复杂度高会减慢渲染速度（任何过度使用 DOM 的应用都不快）

不适合游戏应用

二、小程序面试题

1、简单描述下微信小程序的相关文件类型?

答:微信小程序项目结构主要有四个文件类型,如下

一、WXML (WeiXin Markup Language)是框架设计的一套标签语言，结合 基础组 件、事件系统，可以构建出页面的结构。内部主要是微信自己定 义的一套组件。

二、WXSS (WeiXin Style Sheets)是一套样式语言，用于描述 WXML 的 组件样式，

二、js 逻辑处理，网络请求

三、json 小程序设置，如页面 注册，页面标题及 tabBar。

四、app.json

必须要有这个文件，如果没有这个文件，项目无法运行，因为微信框架 把这个作为配置文 件入口，整个小程序的全局配置。包括页面注册，网 络设置，以及小程序的 window 背景 色，配置导航条样式，配置默认标 题。

五、app.js 必须要有这个文件，没有也是会报错!但是这个文件创建一下 就行 什么都不需要写以后我 们可以在这个文件中监听并处理小程序的 生命周期函数、声明全局变量。

六、app.wxss

2、你是怎么封装微信小程序的数据请求的?

答: 一、将所有的接口放在统一的 js 文件中并导出

二、在 app.js 中创建封装请求数据的方法 三、在子页面中调用封装的方 法请求数据

3、有哪些参数传值的方法?

答: 一、给 HTML 元素添加 data-*属性来传递我们需要的值，然后通过 e.currentTarget.dataset 或 onload 的 param 参数获取。但 data-名称不能 有大写字母 和不可以存放对象 二、设置 id 的方法标识来传值通过

e.currentTarget.id 获取设置的 id 的值,然后通过设置 全局对象的方式来 传递数值 三、在 navigator 中添加参数传值

4、你使用过哪些方法，来提高微信小程序的应用速度?

答: 一、提高页面加载速度

二、用户行为预测

三、减少默认 data 的大小

四、组件化方案

5、小程序与原生 App 哪个好?

答: 小程序除了拥有公众号的低开发成本、低获客成本低以及无需下载等 优势，在服务请 求延时与用户使用体验是都得到了较大幅度 的提升， 使得其能够承载跟复杂的服务功能 以及使用户获得更好的用户体验。

6、简述微信小程序原理?

答:微信小程序采用 JavaScript、WXML、WXSS 三种技术进行开发，从

技术讲和现有的 前端开发差不多，但深入挖掘的话却又有所不同。

JavaScript:首先 JavaScript 的代码是运行在微信 App 中的，并不是运行 在浏览器中，因 此一些 H5 技术的应用，需要微信 App 提供对应的 API 支持，而这限制住了 H5 技术的应 用，且其不能称为严格的 H5， 可以称其为伪 H5，同理，微信提供的独有的某些 API，H5 也不支持或 支持的不是特别好。

WXML:WXML 微信自己基于 XML 语法开发的，因此开发时，只能使 用微信提供的现有标签，HTML 的标签是无法使用的。

WXSS:WXSS 具有 CSS 的大部分特性，但并不是所有的都支持，而且支持哪些，不支持 哪些并没有详细的文档。

微信的架构，是数据驱动的架构模式，它的 UI 和数据是分离的，所有 的页面更新，都需 要通过对数据的更改来实现。

小程序分为两个部分 webview 和 appService。其中 webview 主要用来 展现 UI， appService 有来处理业务逻辑、数据及接口调用。它们在两个进程中运行，通过系统层 JSBridge 实现通信，实现 UI 的渲染、事件 的处理

7、分析下微信小程序的优劣势？

答: 优势:

1、无需下载，通过搜索和扫一扫就可以打开。

2、良好的用户体验:打开速度快。

3、开发成本要比 App 要低。

4、安卓上可以添加到桌面，与原生 App 差不多。

5、为用户提供良好的安全保障。小程序的发布，微信拥有一套严格的审 查流程， 不能通 过审查的小程序是无法发布到线上的。

劣势: 1、限制较多。页面大小不能超过 1M。不能打开超过 5 个层级的 页面。

2、样式单一。小程序的部分组件已经是成型的了，样式不可以修改。例 如:幻灯片、导 航。

3、推广面窄，不能分享朋友圈，只能通过分享给朋友，附近小程序推 广。其中附近小程 序也受到微信的限制。

4、依托于微信，无法开发后台管理功能。

8、微信小程序与 H5 的区别?

答:

第一条是运行环境的不同

传统的 HTML5 的运行环境是浏览器，包括 webview，而微信小程序的 运行环境并非完整的浏览器，是微信开发团队基于浏览器内核完全重构 的一个内置解析器，针对小程序专门做了优化，配合自己定义的开发语 言标准，提升了小程序的性能。

第二条是开发成本的不同

只在微信中运行，所以不用再去顾虑浏览器兼容性，不用担心生产环境

中出现不可预料的 奇妙 BUG

第三条是获取系统级权限的不同 系统级权限都可以和微信小程序无缝衔

接

第四条便是应用在生产环境的运行流畅度

长久以来，当 HTML5 应用面对复杂的业务逻辑或者丰富的页面交互 时，它的体验总是不尽人意，需要不断的对项目优化来提升用户体验。 但是由于微信小程序运行环境独立

9、怎么解决小程序的异步请求问题？

答: 在回调函数中调用下一个组件的函数: app.js

```
success: function (info) { that.apirtnCallback(info) }
```

index.js

```
onLoad: function () { app.apirtnCallback = res => { console.log(res) }
}
```

10、小程序的双向绑定和 vue 哪里不一样？

答: 小程序直接 this.data 的属性是不可以同步到视图的，必须调用:

this.setData({ noBind:true })

11、小程序的 wxss 和 css 有哪些不一样的地方？

答:一、wxss 的图片引入需使用外链地址;

二、没有 Body;样式可直接使用 import 导

12、webview 中的页面怎么跳回小程序中？

答:首先要引入最新版的 jweixin-1.3.2.js，然后

wx.miniProgram.navigateTo({

url: '/pages/login/login'+'$params' })

13、小程序关联微信公众号如何确定用户的唯一性?

答:使用 wx.getUserInfo 方法 withCredentials 为 true 时 可获取 encryptedData，里对称解密

14、如何实现下拉刷新?

答:用 view 代替 scroll-view,,设置 onPullDownRefresh 函数实现

15、小程序调用后台接口遇到哪些问题？

答：一、数据的大小有限制，超过范围会直接导致整个小程序崩溃，除非重启小程序；2、小程序不可以直接渲染文章内容页这类型的 html 文本内容，若需显示要借住插件，但插件渲染会导致页面加载变慢，所以最好在后台对文章内容的 html 进行过滤，后台直接处理批量替换 p 标签 div 标签为 view 标签，然后其它的标签让插件来做，减轻前端的时间。

16、webview 的页面怎么跳转到小程序导航的页面？

答：小程序导航的页面可以通过 switchTab，但默认情况是不会重新加载数据的。
若需加载新数据，则在 success 属性中加入以下代码即可：

```
success: function (e) {
var page = getCurrentPages().pop();
if (page == undefined || page == null) return;
page.onLoad();
}
```

webview 的页面，则通过

```
wx.miniProgram.switchTab({
url: '/pages/index/index'
})
```

17、小程序和 Vue 写法的区别？

答：

一、循环遍历的时候：小程序是 wx:for="list"，而 Vue 是 v-for="info in list"

二、调用 data 模型的时候：小程序是 this.data.uinfo，而 Vue 是 this.uinfo；给模型赋值也不一样，小程序是 this.setData({uinfo:1})，而 Vue 是直接 this.uinfo=1

18、使用 webview 直接加载要注意哪些事项？

答：一、必须要在小程序后台使用管理员添加业务域名；二、h5 页面跳转至小程序的脚本必须是 1.3.1 以上；三、微信分享只可以都是小程序的主名称了，如果要自定义分享的内容，需小程序版本在 1.7.1 以上；四、h5 的支付不可以是微信公众号的 appid，必须是小程序的 appid，而且用户的 openid 也必须是用户和小程序的。

三、REACT 常用面试题目

1、调用 setState 之后发生了什么？

在代码中调用 `setState` 函数之后，React 会将传入的参数对象与组件当前的状态合并，然后触发所谓的调和过程（Reconciliation）。经过调和过程，React 会以相对高效的方式根据新的状态构建 React 元素树并且着手重新渲染整个 UI 界面。在 React 得到元素树之后，React 会自动计算出新的树与老树的节点差异，然后根据差异对界面进行最小化重渲染。在差异计算算法中，React 能够相对精确地知道哪些位置发生了改变以及应该如何改变，这就保证了按需更新，而不是全部重新渲染。

2、React 中 Element 与 Component 的区别是？

简单而言，React Element 是描述屏幕上所见内容的数据结构，是对于 UI 的对象表述。典型的 React Element 就是利用 JSX 构建的声明式代码片然后被转化为 `createElement` 的调用组合。而 React Component 则是可以接收参数输入并且返回某个 React Element 的函数或者类。更多介绍可以参考 React Elements vs React Components。

3、在什么情况下你会优先选择使用 Class Component 而不是 Functional Component？

在组件需要包含内部状态或者使用到生命周期函数的时候使用 Class Component，否则使用函数式组件。

4、React 中 refs 的作用是什么？

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：

```
class CustomForm extends Component {
  handleSubmit = () => {
    console.log("Input Value: ", this.input.value)
  }
  render () {
    return (
      <form onSubmit={this.handleSubmit}>
        <input
          type='text'
          ref={(input) => this.input = input} />
        <button type='submit'>Submit</button>
      </form>
    )
  }
}
```

上述代码中的 `input` 域包含了一个 `ref` 属性，该属性声明的回调函数会接收 `input` 对应的 DOM 元素，我们将其绑定到 `this` 指针以便在其他的类函数中使用。另外值得一提的是，refs 并不是类组件的专属，函数式组件同样能够利用闭包暂存其值：

```
function CustomForm ({handleSubmit}) {
  let inputElement
  return (
    <form onSubmit={() => handleSubmit(inputElement.value)}>
      <input
        type='text'
        ref={(input) => inputElement = input} />
      <button type='submit'>Submit</button>
    </form>
  )
```

```
}
```

5、React 中 keys 的作用是什么？

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。

```
render () {
  return (
    <ul>
      {this.state.todoItems.map(({task, uid}) => {
        return <li key={uid}>{task}</li>
      })}
    </ul>
  )
}
```

在开发过程中，我们需要保证某个元素的 key 在其同级元素中具有唯一性。在 React 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。此外，React 还需要借助 Key 值来判断元素与本地状态的关联关系，因此我们绝不可忽视转换函数中 Key 的重要性。

6、如果你创建了类似于下面的 `Twitter` 元素，那么它相关的类定义是啥样子的？

```
<Twitter username='tylermcginnis33'>
  {(user) => user === null
    ? <Loading />
    : <Badge info={user} />}
</Twitter>
import React, { Component, PropTypes } from 'react'
import fetchUser from 'twitter'
// fetchUser take in a username returns a promise
// which will resolve with that username's data.
class Twitter extends Component {
  // finish this
```

```
}
```

如果你还不熟悉回调渲染模式（Render Callback Pattern），这个代码可能看起来有点怪。这种模式中，组件会接收某个函数作为其子组件，然后在渲染函数中以 `props.children` 进行调用：

```
import React, { Component, PropTypes } from 'react'
import fetchUser from 'twitter'
class Twitter extends Component {
  state = {
    user: null,
  }
  static propTypes = {
    username: PropTypes.string.isRequired,
  }
  componentDidMount () {
    fetchUser(this.props.username)
      .then((user) => this.setState({user}))
  }
  render () {
    return this.props.children(this.state.user)
  }
}
```

这种模式的优势在于将父组件与子组件解耦和，父组件可以直接访问子组件的内部状态而不需要再通过 Props 传递，这样父组件能够更为方便地控制子组件展示的 UI 界面。譬如产品经理让我们将原本展示的 `Badge` 替换为 `Profile`，我们可以轻易地修改下回调函数即可：

```
<Twitter username='tylermcginnis33'>
  {(user) => user === null
    ? <Loading />
    : <Profile info={user} />}
</Twitter>
```

7、Controlled Component 与 Uncontrolled Component 之间的区别是什么？

React 的核心组成之一就是能够维持内部状态的自治组件，不过当我们引入原生的 HTML 表单元素时（input,select,textarea 等），我们是否应该将所有的数据托管到 React 组件中还是将其仍然保留在 DOM 元素中呢？这个问题的答案就是受控组件与非受控组件的定义分割。受控组件（Controlled Component）代指那些交由 React 控制并且所有的表单数据统一存放的组件。譬如下面这段代码中 `username` 变量值并没有存放到 DOM 元素中，而是存放在组件状态数据中。任何时候我们需要改变 `username` 变量值时，我们应当调用 `setState` 函数进行修改。

```
class ControlledForm extends Component {
  state = {
    username: ''
  }
  updateUsername = (e) => {
    this.setState({
      username: e.target.value,
    })
  }
  handleSubmit = () => {}
  render () {
    return (
      <form onSubmit={this.handleSubmit}>
        <input
          type='text'
          value={this.state.username}
          onChange={this.updateUsername} />
        <button type='submit'>Submit</button>
      </form>
    )
  }
}
```

而非受控组件（Uncontrolled Component）则是由 DOM 存放表单数据，并非存放在 React 组件中。我们可以使用 refs 来操控 DOM 元素：

```
class UnControlledForm extends Component {
```

```
  handleSubmit = () => {
    console.log("Input Value: ", this.input.value)
  }
  render () {
    return (
      <form onSubmit={this.handleSubmit}>
        <input
          type='text'
          ref={(input) => this.input = input} />
        <button type='submit'>Submit</button>
      </form>
    )
  }
}
```

竟然非受控组件看上去更好实现，我们可以直接从 DOM 中抓取数据，而不需要添加额外的代码。不过实际开发中我们并不提倡使用非受控组件，因为实际情况下我们需要更多的考虑表单验证、选择性的开启或者关闭按钮点击、强制输入格式等功能支持，而此时我们将数据托管到 React 中有助于我们更好地以声明式的方式完成这些功能。引入 React 或者其他 MVVM 框架最初的原因就是为了将我们从繁重的直接操作 DOM 中解放出来。

8、在生命周期中的哪一步你应该发起 AJAX 请求？

我们应当将 AJAX 请求放到 componentDidMount 函数中执行，主要原因有下：

React 下一代调和算法 Fiber 会通过开始或停止渲染的方式优化应用性能，其会影响到 componentWillMount 的触发次数。对于 componentWillMount 这个生命周期函数的调用次数会变得不确定，React 可能会多次频繁调用 componentWillMount。如果我们将 AJAX 请求放到 componentWillMount 函数中，那么显而易见其会被触发多次，自然也就不是好的选择。

如果我们将 AJAX 请求放置在生命周期的其他函数中，我们并不能保证请求仅在组件挂载完毕后才会要求响应。如果我们的数据请求在组件挂载之前就完成，并且调用了 `setState` 函数将数据添加到组件状态中，对于未挂载的组件则会报错。而在 componentDidMount 函数中进行 AJAX 请求则能有效避免这个问题。

9、shouldComponentUpdate 的作用是啥以及为何它这么重要？

shouldComponentUpdate 允许我们手动地判断是否要进行组件更新，根据组件的应用场景设置函数的合理返回值能够帮我们避免不必要的更新。

10、如何告诉 React 它应该编译生产环境版本？

通常情况下我们会使用 Webpack 的 DefinePlugin 方法来将 NODE_ENV 变量值设置为 production。编译版本中 React 会忽略 propType 验证以及其他的告警信息，同时还会降低代码库的大小，React 使用了 Uglify 插件来移除生产环境下不必要的注释等信息。

11、为什么我们需要使用 React 提供的 Children API 而不是 JavaScript 的 map？

`props.children` 并不一定是数组类型，譬如下面这个元素：

```
<Parent>
  <h1>Welcome.</h1>
</Parent>
```

如果我们使用 `props.children.map` 函数来遍历时会受到异常提示，因为在这种情况下 `props.children` 是对象（object）而不是数组（array）。React 当且仅当超过一个子元素的情况下会将 `props.children` 设置为数组，就像下面这个代码片：

```
<Parent>
  <h1>Welcome.</h1>
  <h2>props.children will now be an array</h2>
</Parent>
```

这也就是我们优先选择使用 `React.Children.map` 函数的原因，其已经将 `props.children` 不同类型的情况考虑在内了。

12、概述下 React 中的事件处理逻辑

为了解决跨浏览器兼容性问题，React 会将浏览器原生事件（Browser Native Event）封装为合成事件（SyntheticEvent）传入设置的事件处理器中。这里的合成事件提供了与原生事件相同的接口，不过它们屏蔽了底层浏览器的细节差异，保证了行为的一致性。另外有意思的是，React 并没有直接将事件附着到子元素上，而是以单一事件监听器的方式将所有的事件发送到顶层进行处理。这样 React 在更新 DOM 的时候就不需要考虑如何去处理附着在 DOM 上的事件监听器，最终达到优化性能的目的。

13、createElement 与 cloneElement 的区别是什么？

createElement 函数是 JSX 编译之后使用的创建 React Element 的函数，而 cloneElement 则是用于复制某个元素并传入新的 Props。

14、传入 setState 函数的第二个参数的作用是什么？

该函数会在 `setState` 函数调用完成并且组件开始重渲染的时候被调用，我们可以用该函数来监听渲染是否完成：

```
this.setState(
  { username: 'tylermcginnis33' },
  () => console.log('setState has finished and the component has re-rendered.')
)
```

15、下述代码有错吗？

```
this.setState((prevState, props) => {
  return {
    streak: prevState.streak + props.count
  }
})
```

16、当你调用 setState 的时候，发生了什么事？

当调用 **`setState`** 时，React 会做的第一件事情是将传递给 **`setState`** 的对象合并到组件的当前状态。这将启动一个称为和解（reconciliation）的过程。和解（reconciliation）的最终目标是以最有效的方式，根据这个新的状态来更新 UI。 为此，React 将构建一个新的 **`React`** 元素树（您可以将其视为 UI 的对象表示）。

一旦有了这个树，为了弄清 UI 如何响应新的状态而改变，React 会将这个新树与上一个元素树相比较（diff）。

通过这样做， React 将会知道发生的确切变化，并且通过了解发生什么变化，只需在绝对必要的情况下进行更新即可最小化 UI 的占用空间。

17、在 React 当中 Element 和 Component 有何区别？

简单地说，一个 React element 描述了你想在屏幕上看到什么。换个说法就是，一个 React element 是一些 UI 的对象表示。

一个 React Component 是一个函数或一个类，它可以接受输入并返回一个 React element t（通常是通过 JSX ，它被转化成一个 createElement 调用）

18、什么时候在功能组件（ Class Component ）上使用类组件（ Functional Component ）？

如果您的组件具有状态(state)或生命周期方法，请使用 Class 组件。否则，使用功能组件

19、什么是 React 的 refs ，为什么它们很重要？

refs 就像是一个逃生舱口，允许您直接访问 DOM 元素或组件实例。为了使用它们，您可以向组件添加一个 ref 属性，该属性的值是一个回调函数，它将接收底层的 DOM 元素或组件的已挂接实例，作为其第一个参数。

```
class UnControlledForm extends Component {
  handleSubmit = () => {
    console.log("Input Value: ", this.input.value)
  }
  render () {
    return (
      <form onSubmit={this.handleSubmit}>
        <input
          type='text'
          ref={(input) => this.input = input} />
        <button type='submit'>Submit</button>
      </form>
    )
  }
}
```

以上注意到我们的输入字段有一个 ref 属性，其值是一个函数。该函数接收我们然后放在实例上的实际的 DOM 元素，以便在 ***handleSubmit*** 函数内部访问它。经常误解的是，您需要使用类组件才能使用 ref ，但 ref 也可以通过利用 **JavaScript** 中的 闭包 与 功能组件(functional components)一起使用。

```
function CustomForm ({handleSubmit}) {
  let inputElement
```

```
  return (
    <form onSubmit={() => handleSubmit(inputElement.value)}>
      <input
        type='text'
        ref={(input) => inputElement = input} />
      <button type='submit'>Submit</button>
    </form>
  )
}
```

20、React 中的 keys 是什么，为什么它们很重要？

keys 是什么帮助 React 跟踪哪些项目已更改、添加或从列表中删除。

```
  return (
    <ul>
      {this.state.todoItems.map(({task, uid}) => {
        return <li key={uid}>{task}</li>
      })}
    </ul>
  )
}
```

每个 keys 在兄弟元素之间是独一无二的。我们已经谈过几次关于和解（reconciliation）的过程，而且这个和解过程（reconciliation）中的一部分正在执行一个新的元素树与最前一个的差异。keys 使处理列表时更加高效，因为 React 可以使用子元素上的 keys 快速知道元素是新的还是在比较树时才被移动。

而且 keys 不仅使这个过程更有效率，而且没有 keys ， React 不知道哪个本地状态对应于移动中的哪个项目。所以当你 map 的时候，不要忽略了 keys 。

21、看下面的代码：如果您在 下创建了一个 React 元素， 的组件定义将如何？

```
<Twitter username='tylermcginnis33'>
  {(user) => user === null
    ? <Loading />
```

```
    : <Badge info={user} />}
</Twitter>
import React, { Component, PropTypes } from 'react'
import fetchUser from 'twitter'
// fetchUser 接收用户名返回 promise
// 当得到 用户的数据的时候 ， 返回 resolve 状态

class Twitter extends Component {
  // 在这里写下你的代码
}
```

如果你不熟悉渲染回调模式（render callback pattern），这将看起来有点奇怪。在这种模式中，一个组件接收一个函数作为它的 child。注意上面包含在 标签内的内容。 ***Twitter*** 组件的 child 是一个函数，而不是你曾经习以为常的一个组件。 这意味着在实现 ***Twitter*** 组件时，我们需要将 ***props.children*** 作为一个函数来处理。

以下是答案。

```
import React, { Component, PropTypes } from 'react'
import fetchUser from 'twitter'

class Twitter extends Component {
  state = {
    user: null,
  }
  static propTypes = {
    username: PropTypes.string.isRequired,
  }
  componentDidMount () {
    fetchUser(this.props.username)
      .then((user) => this.setState({user}))
  }
  render () {
    return this.props.children(this.state.user)
```

```
  }
}
```

值得注意的是，正如我上面提到的，我通过调用它并传递给 user 来把 props.children 处理为为一个函数。

这种模式的好处是我们已经将我们的父组件与我们的子组件分离了。父组件管理状态，父组件的消费者可以决定以何种方式将从父级接收的参数应用于他们的 UI。

为了演示这一点，我们假设在另一个文件中，我们要渲染一个 ***Profile*** 而不是一个 ***Badge,*** ，因为我们使用渲染回调模式，所以我们可以轻松地交换 UI ，而不用改变我们对父（Twitter）组件的实现。

```
<Twitter username='tylermcginnis33'>
  {(user) => user === null
    ? <Loading />
    : <Profile info={user} />}
</Twitter>
```

22、受控组件(controlled component)与不受控制的组件(uncontrolled component)有什么区别？

React 的很大一部分是这样的想法，即组件负责控制和管理自己的状态。

当我们将 native HTML 表单元素（ input, select, textarea 等）投入到组合中时会发生什么？我们是否应该使用 React 作为“单一的真理来源”，就像我们习惯使用 React 一样？ 或者我们是否允许表单数据存在 DOM 中，就像我们习惯使用 HTML 表单元素一样？ 这两个问题是受控（controlled）VS 不受控制（uncontrolled）组件的核心。

受控组件是 React 控制的组件，也是表单数据的唯一真理来源。

如下所示， ***username*** 不存在于 DOM 中，而是以我们的组件状态存在。每当我们想要更新 ***username*** 时，我们就像以前一样调用 setState。

```
class ControlledForm extends Component {
  state = {
    username: ''
  }
  updateUsername = (e) => {
    this.setState({
      username: e.target.value,
```

```
    })
  }
  handleSubmit = () => {}
  render () {
    return (
      <form onSubmit={this.handleSubmit}>
        <input
          type='text'
          value={this.state.username}
          onChange={this.updateUsername} />
        <button type='submit'>Submit</button>
      </form>
    )
  }
}
```

不受控制(uncontrolled component)的组件是您的表单数据由 DOM 处理，而不是您的 React 组件。我们使用 refs 来完成这个。

```
class UnControlledForm extends Component {
  handleSubmit = () => {
    console.log("Input Value: ", this.input.value)
  }
  render () {
    return (
      <form onSubmit={this.handleSubmit}>
        <input
          type='text'
          ref={(input) => this.input = input} />
        <button type='submit'>Submit</button>
      </form>
    )
  }
```

```
}
```

虽然不受控制的组件通常更容易实现，因为您只需使用引用从 DOM 获取值，但是通常建议您通过不受控制的组件来支持受控组件。

主要原因是受控组件 支持即时字段验证，允许您有条件地禁用/启用按钮，强制输入格式，并且更多的是 『the React way』。

23、您如何告诉 React 构建（build）生产模式，该做什么？

通常，您将使用 Webpack 的 ***DefinePlugin*** 方法将 NODE_ENV 设置为 production。这将剥离像 propType 验证和额外的警告。除此之外，还有一个好主意，可以减少你的代码，因为 React 使用 Uglify 的 dead-code 来消除开发代码和注释，这将大大减少你的包的大小。

为什么要使用 React.Children.map（props.children，（）=>） 而不是 props.children.map（（）=>）因为不能保证 props.children 将是一个数组。

以此代码为例，

```
<Parent>
  <h1>Welcome.</h1>
</Parent>
```

在父组件内部，如果我们尝试使用 props.children.map 映射孩子，则会抛出错误，因为 props.children 是一个对象，而不是一个数组。

如果有多个子元素，React 只会使 props.children 成为一个数组。就像下面这样：

```
<Parent>
  <h1>Welcome.</h1>
  <h2>props.children will now be an array</h2>
</Parent>
```

这就是为什么你喜欢 `React.Children.map`，因为它的实现考虑到 `props.children` 可能是一个数组或一个对象。

24、描述事件在 React 中的处理方式。

为了解决跨浏览器兼容性问题，您的 React 中的事件处理程序将传递 `SyntheticEvent` 的实例，它是 React 的浏览器本机事件的跨浏览器包装器。

这些 `SyntheticEvent` 与您习惯的原生事件具有相同的接口，除了它们在所有浏览器中都兼容。有趣的是，React 实际上并没有将事件附加到子节点本身。React 将使用单个事件监听器监听顶层的所

有事件。这对于性能是有好处的，这也意味着在更新 DOM 时，React 不需要担心跟踪事件监听器。

25、redux 中间件

中间件提供第三方插件的模式，自定义拦截 action -> reducer 的过程。变为 action -> middlewares -> reducer 。这种机制可以让我们改变数据流，实现如异步 action ，action 过滤，日志输出，异常报告等功能。

常见的中间件：

redux-logger：提供日志输出

redux-thunk：处理异步操作

redux-promise：处理异步操作，actionCreator 的返回值是 promise

26、redux 有什么缺点

1.一个组件所需要的数据，必须由父组件传过来，而不能像 flux 中直接从 store 取。

2.当一个组件相关数据更新时，即使父组件不需要用到这个组件，父组件还是会重新 render，可能会有效率影响，或者需要写复杂的 shouldComponentUpdate 进行判断。

27、react 组件的划分业务组件技术组件？

根据组件的职责通常把组件分为 UI 组件和容器组件。

UI 组件负责 UI 的呈现，容器组件负责管理数据和逻辑。

两者通过 React-Redux 提供 connect 方法联系起来。

28、react 生命周期函数

这个问题要考察的是组件的生命周期

一、初始化阶段：

getDefaultProps:获取实例的默认属性

getInitialState:获取每个实例的初始化状态

componentWillMount：组件即将被装载、渲染到页面上

render:组件在这里生成虚拟的 DOM 节点

componentDidMount:组件真正在被装载之后

二、运行中状态：

componentWillReceiveProps:组件将要接收到属性的时候调用

shouldComponentUpdate:组件接受到新属性或者新状态的时候（可以返回 false，接收数据后不更新，阻止 render 调用，后面的函数不会被继续执行了）

componentWillUpdate:组件即将更新不能修改属性和状态

render:组件重新描绘

componentDidUpdate:组件已经更新

三、销毁阶段：

componentWillUnmount:组件即将销毁

29、react 性能优化是哪个周期函数？

shouldComponentUpdate 这个方法用来判断是否需要调用 render 方法重新描绘 dom。因为 dom 的描绘非常消耗性能，如果我们能在 shouldComponentUpdate 方法中能够写出更优化的 dom diff 算法，可以极大的提高性能。

30、为什么虚拟 dom 会提高性能?

虚拟 dom 相当于在 js 和真实 dom 中间加了一个缓存，利用 dom diff 算法避免了没有必要的 dom 操作，从而提高性能。

具体实现步骤如下：

用 JavaScript 对象结构表示 DOM 树的结构；然后用这个树构建一个真正的 DOM 树，插到文档当中

当状态变更的时候，重新构造一棵新的对象树。然后用新的树和旧的树进行比较，记录两棵树差异

把 2 所记录的差异应用到步骤 1 所构建的真正的 DOM 树上，视图就更新了。

31、diff 算法?

把树形结构按照层级分解，只比较同级元素。

给列表结构的每个单元添加唯一的 key 属性，方便比较。

React 只会匹配相同 class 的 component（这里面的 class 指的是组件的名字）

合并操作，调用 component 的 setState 方法的时候, React 将其标记为 dirty.到每一个事件循环结束, React 检查所有标记 dirty 的 component 重新绘制.

选择性子树渲染。开发人员可以重写 shouldComponentUpdate 提高 diff 的性能。

32、react 性能优化方案

（1）重写 shouldComponentUpdate 来避免不必要的 dom 操作。

（2）使用 production 版本的 react.js。

（3）使用 key 来帮助 React 识别列表中所有子组件的最小变化。

33、简述 flux 思想

Flux 的最大特点，就是数据的"单向流动"。

1.用户访问 View

2.View 发出用户的 Action

3.Dispatcher 收到 Action，要求 Store 进行相应的更新

4.Store 更新后，发出一个"change"事件

5.View 收到"change"事件后，更新页面

34、React 项目用过什么脚手架？Mern？Yeoman？

Mern：MERN 是脚手架的工具，它可以很容易地使用 Mongo, Express, React and NodeJS 生成同构 JS 应用。它最大限度地减少安装时间，并得到您使用的成熟技术来加速开发。

四、React Native

1、如何利用 Android Studio 打包 React Native APK

ok！百度出来的东西很杂，所以，这里介绍一种最简单，最合适我们（新手，应该是吧）的 APK 的打包方式！

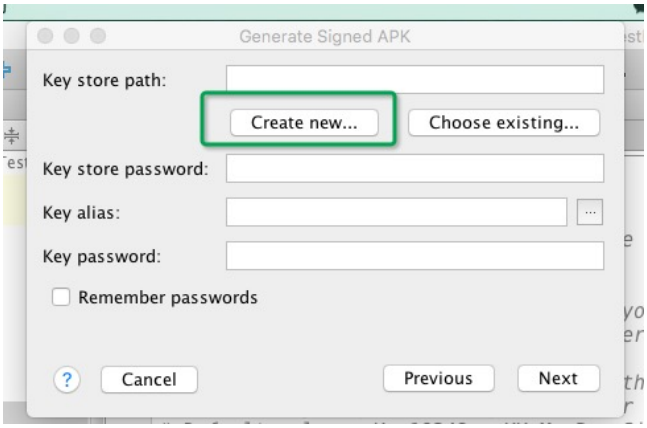
当然！这种打包是基于 Android Studio 的，所以，注意喽！！！！

废话不多说开始吧！

首先，我们要整理我们的思路，第一步是给 APK 签名~第二步是完成打包

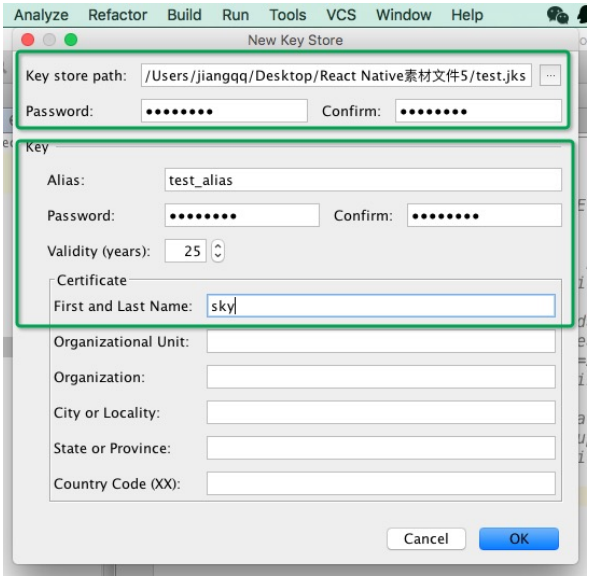
第一步：签名

打开 Android Studio 然后在菜单栏的 Build——>Generate Signed APK 在打开的界面点击 Next，会弹出下面的界面

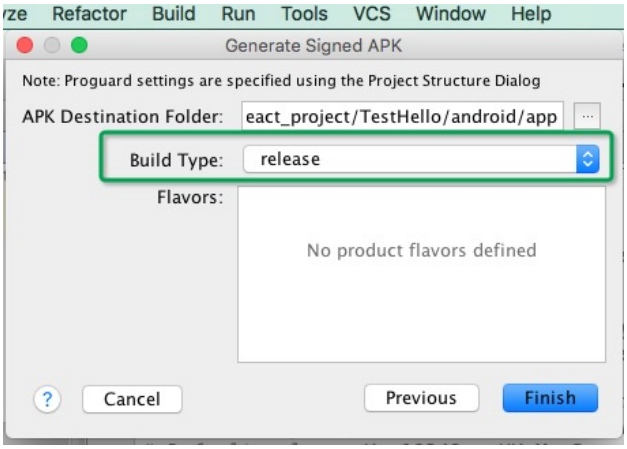


然后点击 create new 在弹出的界面中选择填写秘钥存放的位置,名称,密码。同样还要写别名的名字,证书的所有者，国家，组织以及城市相关信息。

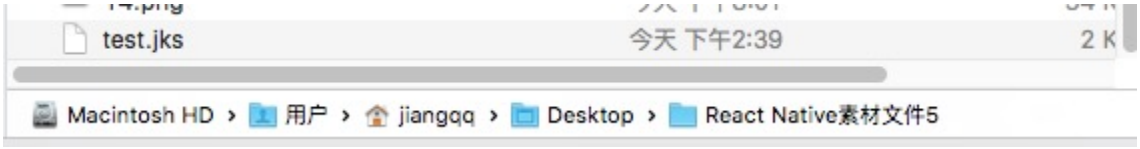
注释：我们这里 app 的名称是 jd，别名是 jd_alias，名称的密码和别名的密码都一样



点击 OK，会默认填写上创建好的签名的信息,



最后点击 finish 会生成签名秘钥

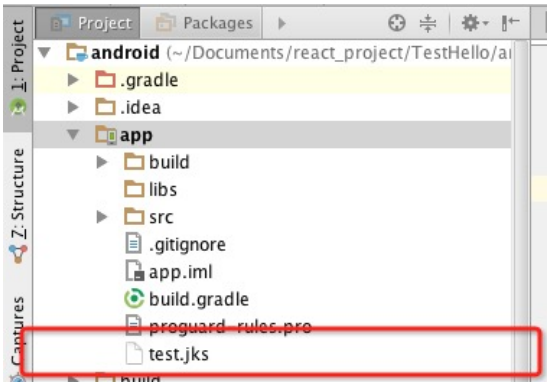


第二部：

通过以上几步就已经将 APP 的签名完成了！下面开始真正的打包了哦！

1.Gradle 配置：

1.首先我们要把刚刚生成的签名文件复制到项目 android/app 文件夹下面（你的 xxx.jks 文件路径可能和我不同，但是没关系复制到这个 andrid/app 路径下就可以）



然后进行修改项目中 gradle.properties 文件，进行添加如下的代码(注意下面的签名和别名的名称和上一步放入的 test.jks 要一样，下面两项分别填写签名和别名的密码)-我取的密码为 ztt12345

```
MYAPP_RELEASE_STORE_FILE=test.jks
MYAPP_RELEASE_KEY_ALIAS=test_alias
MYAPP_RELEASE_STORE_PASSWORD=ztt12345
MYAPP_RELEASE_KEY_PASSWORD=ztt12345
```

这一步我们是进行全局的 gradlde 进行变量化的配置，后边我们会在后边的步骤中给相应的应用进行签名。

[注意].以上的签名秘钥请大家一定要妥善保管，因为在应用发布的时候需要的。

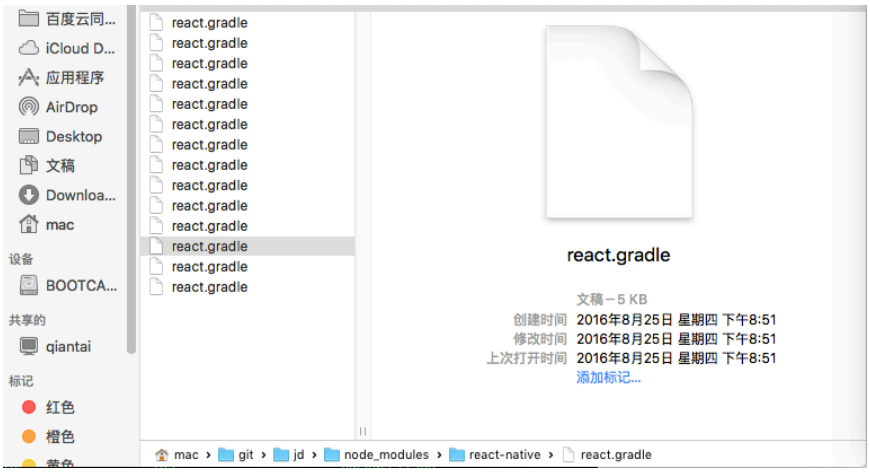
2。给应用添加签名-配置局部应用 Gradle 文件

直接在工程目录下得 android/app/build.gradle 中以下节点添加如下内容:

```
...
android {
    ...
    defaultConfig { ... }
    signingConfigs {
        release {
            storeFile file(MYAPP_RELEASE_STORE_FILE)
            storePassword MYAPP_RELEASE_STORE_PASSWORD
            keyAlias MYAPP_RELEASE_KEY_ALIAS
            keyPassword MYAPP_RELEASE_KEY_PASSWORD
        }
    }
    buildTypes {
        release {
            ...
            signingConfig signingConfigs.release
        }
    }
}
```

注释：这里面的内容就不需要改了哈！位置放对即可

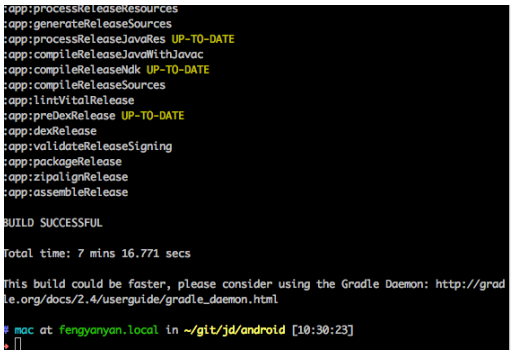
最后：查看有没有 react.gradle 文件（一般 init 化都会有，具体路径如图所示的底部）



ok！只要有了这个文件~我们就可以开始打包了~进入项目根目录！（是根目录哦）执行以下命令：

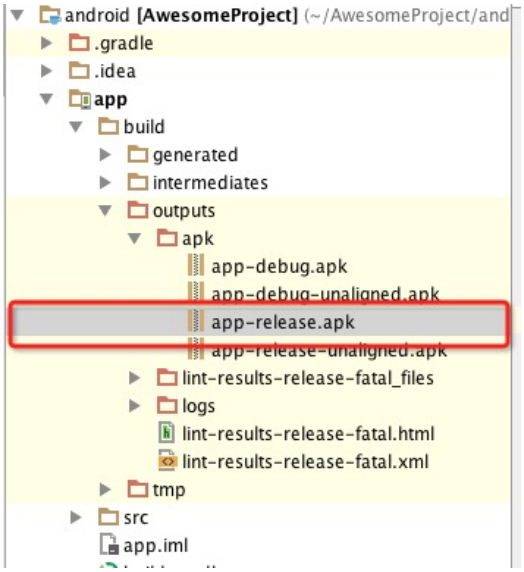
cd android && ./gradlew assembleRelease

然后会开始打包~~（打包的时间会很长~十分钟）然后会出现以下的图！



ok~打包完成~~

然后我们就进入，会在 android/app/build/outputs/apk 目录下面生成 app-release.apk 该文件



好了！你就可以发布啦！还有一点的是！若你修改了文件，重新打包就可以直接去运行打包命令（cd android && ./gradlew assembleRelease ）而无需去配置其他东东了！！！！

2、基础组件（一）

1. TextInput

允许用户输入文本的基础组件.

属性

onChangeText 接受一个函数，而此函数会在文本变化时被调用。

onSubmitEditing 在文本被提交后（用户按下软键盘上的提交键）调用

实例：

```
<TextInput
      style={{height: 40}}
      placeholder="Type here to translate!"
      onChangeText={(text) => this.setState({text})}
    />
    <Text style={{padding: 10, fontSize: 42}}>

```

```
        {this.state.text.split(' ').map((word) => word && '🍕').join(' ')}
    </Text>
```

2.ScrollView

通用的可滚动的容器.可以垂直滚动，也可以水平滚动。（通过 horizontal 属性来设置）

ScrollView 适合用来显示数量不多的滚动元素。放置在 ScollView 中的所有组件都会被渲染.

实例：

```
<ScrollView>
          <Text style={{fontSize:96}}>Scroll me plz</Text>
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Text style={{fontSize:96}}>If you like</Text>
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
          <Text style={{fontSize:96}}>Scrolling down</Text>
          <Image source={require('./img/favicon.png')} />
          <Image source={require('./img/favicon.png')} />
        </ScrollView>
```

3.ListView

显示一个垂直的滚动列表，其中的元素之间结构近似而仅数据不同。

ListView 并不立即渲染所有元素，而是优先渲染屏幕上可见的元素。

ListView 的一个常用场景就是从服务器端取回列表数据然后显示

属性

dataSource 列表的数据源 **必须**

renderRow 逐个解析数据源中的数据，然后返回一个设定好格式的组件来渲染 **必须**

rowHasChanged 函数也是 ListView 的必需属性。 **必须**

实例

```
import React, { Component } from 'react';
import { AppRegistry, ListView, Text, View } from 'react-native';

class ListViewBasics extends Component {
  // 初始化模拟数据
  constructor(props) {
    super(props);
    const ds = new ListView.DataSource({rowHasChanged: (r1, r2) => r1 !== r2});
    this.state = {
      dataSource: ds.cloneWithRows([
        'John', 'Joel', 'James', 'Jimmy', 'Jackson', 'Jillian', 'Julie', 'Devin'
      ])
    };
  }
  render() {
    return (
      <View style={{paddingTop: 22}}>
        <ListView
          dataSource={this.state.dataSource}
          renderRow={(rowData) => <Text>{rowData}</Text>}
        />
      </View>
    );
```

```
  }
}
```

3. Image

用于显示多种不同类型图片的 React 组件，包括网络图片、静态资源、临时的本地图片、以及本地磁盘上的图片（如相册）等.

常用属性

onLayout function

当元素挂载或者布局改变的时候调用，参数为：`{nativeEvent: {layout: {x, y, width, height}}}`.

onLoad function

加载成功完成时调用此回调函数。

onLoadEnd function

加载结束后，不论成功还是失败，调用此回调函数。

onLoadStart function

加载开始时调用。

resizeMode enum('cover', 'contain', 'stretch')

决定当组件尺寸和图片尺寸不成比例的时候如何调整图片的大小。

`cover`: 在保持图片宽高比的前提下缩放图片，直到宽度和高度都大于等于容器视图的尺寸（如果容器有 padding 内衬的话，则相应减去）。译注：这样图片完全覆盖甚至超出容器，容器中不留任何空白。

`contain`: 在保持图片宽高比的前提下缩放图片，直到宽度和高度都小于等于容器视图的尺寸（如果容器有 padding 内衬的话，则相应减去）。译注：这样图片完全被包裹在容器中，容器中可能留有空白

`stretch`: 拉伸图片且不维持宽高比，直到宽高都刚好填满容器。

source {uri: string}, number

`uri` 是一个表示图片的资源标识的字符串，它可以是一个 http 地址或是一个本地文件路径（使用 `require(相对路径)` 来引用）

此处需要注意如果是网络图片或者原生图片，必须指定图片大大小，否则不会渲染出结果！！

style

tintColor color

为所有非透明的像素指定一个颜色；

其它和 css 类似。

(ios)accessibilityLabel string

当用户与图片交互时，读屏器（无障碍功能）会朗读的文字。

(ios)accessible bool

当此属性为真的时候，表示这个图片是一个启用了无障碍功能的元素。

(ios)defaultSource {uri: string}

一个静态图片，当最终的图片正在下载的过程中时显示（loading 背景图）。

(ios)onProgress function

在加载过程中不断调用，参数为`{nativeEvent: {loaded, total}}`

方法

static getSize(uri: string, success: (width: number, height: number) => void, failure: (error: any) => void)

在显示图片前获取图片的宽高(以像素为单位)。如果图片地址不正确或下载失败,此方法也会失败。要获取图片的尺寸,首先需要加载或下载图片(同时会被缓存起来)。这意味着理论上你可以用这个方法来预加载图片,虽然此方法并没有针对这一用法进行优化,而且将来可能会换一些实现方案使得并不需要完整下载图片即可获取尺寸。所以更好的预加载方案是使用下面那个专门的预加载方法。

static prefetch(url: string)

预加载一个远程图片(将其下载到本地磁盘缓存)。

4. TouchableWithoutFeedback

由于这个组件在被按下时没有任何视觉反馈，一般情况下不建议使用。

注意：TouchableWithoutFeedback 只支持一个子节点

如果你希望包含多个子组件，用一个 View 来包装它们。

属性

这一部分比较重要，因为以 Touchable 开头的按钮都是在此组件的基础上实现的

delayLongPress number

单位是毫秒，从 onPressIn 开始，到 onLongPress 被调用的延迟。

delayPressIn number

单位是毫秒，从触摸操作开始到 onPressIn 被调用的延迟。

delayPressOut number

单位是毫秒，从触摸操作结束开始到 onPressOut 被调用的延迟。

disabled bool

如果设为 true，则禁止此组件的一切交互。

hitSlop {top: number, left: number, bottom: number, right: number}

这一属性定义了按钮的外延范围。这一范围也会使 `pressRetentionOffset`（见下文）变得更大。 注意：触摸范围不会超过父视图的边界，也不会影响原先和本组件层叠的视图（保留原先的触摸优先级）。

onLayout function

当加载或者布局改变的时候被调用，参数为：

```
{nativeEvent: {layout: {x, y, width, height}}}
```

onLongPress function

长按触发

onPress function

当触摸操作结束时调用，但如果被取消了则不调用（譬如响应者被一个滚动操作取代）

onPressIn function

onPressOut function

pressRetentionOffset {top: number, left: number, bottom: number, right: number}

在当前视图不能滚动的前提下指定这个属性，可以决定当手指移开多远距离之后，会不再激活按钮。

但如果手指再次移回范围内，按钮会被再次激活。只要视图不能滚动，你可以来回多次这样的操作。确保你传入一个常量来减少内存分配。

5.TouchableHighlight

本组件用于封装视图，使其可以正确响应触摸操作。当按下的时候，封装的视图的不透明度会降低，同时会有一个底层的颜色透过而被用户看到，使得视图变暗或变亮。

注意：TouchableHighlight 只支持一个子节点
如果你需要包含多个子组件，可以用一个 View 来包装。

属性

TouchableWithoutFeedback props...

本组件继承了所有 TouchableWithoutFeedback 的属性。

activeOpacity number

指定封装的视图在被触摸操作激活时以多少不透明度显示（通常在 0 到 1 之间）。

onHideUnderlay function

当底层的颜色被隐藏的时候调用。

onShowUnderlay function

当底层的颜色被显示的时候调用。

style

css

underlayColor string

有触摸操作时显示出来的底层的颜色。

6.TouchableOpacity

本组件用于封装视图，使其可以正确响应触摸操作。当按下的时候，封装的视图的不透明度会降低。

属性

TouchableWithoutFeedback props...

本组件继承了所有 TouchableWithoutFeedback 的属性。

activeOpacity number

指定封装的视图在被触摸操作激活时以多少不透明度显示（通常在 0 到 1 之间）。

7.TouchableNativeFeedback

本组件用于封装视图，使其可以正确响应触摸操作（仅限 Android 平台）。在 Android 设备上，这个组件利用原生状态来渲染触摸的反馈。

按下时显示水波纹的效果。

目前它只支持一个单独的 View 实例作为子节点。在底层实现上，实际会创建一个新的 RCTView 结点替换当前的子 View，并附带一些额外的属性。

原生触摸操作反馈的背景可以使用 `background` 属性来自定义。

属性

TouchableWithoutFeedback props...

本组件继承了所有 TouchableWithoutFeedback 的属性。

background backgroundPropType

决定在触摸反馈的时候显示什么类型的背景。它接受一个有着 `type` 属性和一些基于 `type` 属性的额外数据的对象。推荐使用以下的静态方法之一来创建这个对象：

1) TouchableNativeFeedback.SelectableBackground() - 会创建一个对象，表示安卓主题默认的对于被选中对象的背景。(?android:attr/selectableItemBackground)

2) TouchableNativeFeedback.SelectableBackgroundBorderless() - 会创建一个对象，表示安卓主题默认的对于被选中的无边框对象的背景。(?android:attr/selectableItemBackgroundBorderless)。只在 Android API level 21+适用。

3) TouchableNativeFeedback.Ripple(color, borderless) - 会创建一个对象，当按钮被按下时产生一个涟漪状的背景，你可以通过 color 参数来指定颜色，如果参数 `borderless` 是 true，那么涟漪还会渲染到视图的范围之外。（参见原生的 actionbar buttons 作为该效果的一个例子）。这个背景类型只在 Android API level 21+适用。

3、基础组件(二)

1.MapView

显示地图的组件

属性

ios annotations [{latitude: number, longitude: number, animateDrop: bool, title: string, subtitle: string, hasLeftCallout: bool, hasRightCallout: bool, onLeftCalloutPress: function, onRightCalloutPress: function, id: string}]

地图上的标注点，可以带有标题及副标题。

ios mapType enum('standard', 'satellite', 'hybrid')

要显示的地图类型。

standard: 标准道路地图（默认）。

satellite: 卫星视图。

hybrid: 卫星视图并附带道路和感兴趣的点标记。

ios maxDelta number

可以被显示的最大区域尺寸。

ios minDelta number

可以被显示的最小区域尺寸。

onAnnotationPress function

当用户点击地图上的标注之后会调用此回调函数一次。

region {latitude: number, longitude: number, latitudeDelta: number, longitudeDelta: number}

地图显示的区域。

区域使用中心的坐标和要显示的范围来定义。

zoomEnabled bool

如果此属性为 `false`，用户则不能旋转/缩放地图。默认值为 `true`。

2. Modal

Modal 组件可以用来覆盖包含 React Native 根视图的原生视图（如 UIViewController，Activity）。

在嵌入 React Native 的混合应用中可以使用 Modal。Modal 可以使你应用中 RN 编写的那部分内容覆盖在原生视图上显示。

在从根视图开始就使用 RN 编写的应用中，你应该使用 Navigator 来代替 Modal。通过一个最顶层的 Navigator，你可以通过 configureScene 属性更加方便的控制如何将模态场景覆盖显示在你 App 其余的部分上。

目前这个组件还只能在 iOS 上使用。

3. Navigator

导航器通过路由对象来分辨不同的场景。利用 `renderScene` 方法，导航栏可以根据指定的路由来渲染场景。

可以通过 `configureScene` 属性获取指定路由对象的配置信息，从而改变场景的动画或者手势。查看 `Navigator.SceneConfigs` 来获取默认的动画和更多的场景配置选项。

基本用法

Navigator 折叠源码

```
<Navigator
    initialRoute={{name: 'My First Scene', index: 0}}
    renderScene={(route, navigator) =>
      <MySceneComponent
        name={route.name}
        onForward={() => {
          var nextIndex = route.index + 1;
          navigator.push({
            name: 'Scene ' + nextIndex,
            index: nextIndex,
          });
        }}
        onBack={() => {
          if (route.index > 0) {
            navigator.pop();
          }
        }}
      />
    }
  />
```

导航方法

getCurrentRoutes() - 获取当前栈里的路由，也就是 push 进来，没有 pop 掉的那些。

jumpBack() - 跳回之前的路由，当然前提是保留现在的，还可以再跳回来，会给你保留原样。

jumpForward() - 上一个方法不是调到之前的路由了么，用这个跳回来就好了。

jumpTo(route) - 跳转到已有的场景并且不卸载。

push(route) - 跳转到新的场景，并且将场景入栈，你可以稍后跳转过去

pop() - 跳转回去并且卸载掉当前场景

replace(route) - 用一个新的路由替换掉当前场景

replaceAtIndex(route, index) - 替换掉指定序列的路由场景

replacePrevious(route) - 替换掉之前的场景

immediatelyResetRouteStack(routeStack) - 用新的路由数组来重置路由栈

popToRoute(route) - pop 到路由指定的场景，其他的场景将会卸载。

popToTop() - pop 到栈中的第一个场景，卸载掉所有的其他场景。

属性

configureScene function

可选的函数，用来配置场景动画和手势。会带有两个参数调用，一个是当前的路由，一个是当前的路由栈。然后它应当返回一个场景配置对象

```
(route, routeStack) => Navigator.SceneConfigs.FloatFromRight
```

initialRoute object

定义启动时加载的路由。路由是导航栏用来识别渲染场景的一个对象。`initialRoute` 必须是 `initialRouteStack` 中的一个路由。`initialRoute` 默认为 `initialRouteStack` 中最后一项。

initialRouteStack [object]

提供一个路由集合用来初始化。如果没有设置初始路由的话则必须设置该属性。如果没有提供该属性，它将被默认设置成一个只含有 `initialRoute` 的数组。

navigationBar node

可选参数，提供一个在场景切换的时候保持的导航栏。

navigator object

可选参数，提供从父导航器获得的导航器对象。

renderScene function

必要参数。用来渲染指定路由的场景。调用的参数是路由和导航器。

```
(route, navigator) =>
  <MySceneComponent title={route.title} navigator={navigator} />
```

sceneStyle View#style

将会应用在每个场景的容器上的样式。

4.Picker

本组件可以在 iOS 和 Android 上渲染原生的选择器（Picker）。用例：

Picker 折叠源码

```
1 <Picker
2   selectedValue={this.state.language}
3   onValueChange={(lang) => this.setState({language: lang})}>
4   <Picker.Item label="Java" value="java" />
```

```
  <Picker.Item label="JavaScript" value="js" />
</Picker>
```

属性

onValueChange function

某一项被选中时执行此回调。调用时带有如下参数：

itemValue: 被选中项的 value 属性

itemPosition: 被选中项在 picker 中的索引位置

selectedValue any

默认选中的值。可以是字符串或整数。

style pickerStyleType

testID string

用于在端对端测试中定位此视图。

android enabled bool

如果设为 false，则会禁用此选择器。

android mode enum('dialog', 'dropdown')

在 Android 上，可以指定在用户点击选择器时，以怎样的形式呈现选项：

dialog（对话框形式）：显示一个模态对话框。默认选项。

dropdown（下拉框形式）：以选择器所在位置为锚点展开一个下拉框。

android prompt string

设置选择器的提示字符串。在 Android 的对话框模式中用作对话框的标题。

ios itemStyle itemStylePropType

指定应用在每项标签上的样式。

5. Switch

跨平台通用的可以在两个状态中切换的组件。

属性

disabled bool

如果为 true，这个组件不能进行交互。

onValueChange function

当值改变的时候调用此回调函数，参数为新的值。

testID string

用来在端到端测试中定位此视图。

value bool

表示此开关是否打开。默认为 false（关闭状态）。

ios onTintColor ColorPropType

开启状态时的背景颜色。

ios thumbTintColor ColorPropType

开关上圆形按钮的背景颜色。

ios tintColor ColorPropType

关闭状态时的背景颜色。

6. RefreshControl

这一组件用在 ScrollView 内部，为其添加下拉刷新的功能。当 ScrollView 处于竖直方向的起点位置（scrollY: 0），此时下拉会触发一个 `onRefresh` 事件。

属性

onRefresh function

在视图开始刷新时调用。

refreshing bool

视图是否应该在刷新时显示指示器。

android colors [ColorPropType]

指定至少一种颜色用来绘制刷新指示器。

android enabled bool

指定是否要开启刷新指示器。

android progressBackgroundColor ColorPropType

指定刷新指示器的背景色。

android size RefreshLayoutConsts.SIZE.DEFAULT

指定刷新指示器的大小，具体数值可参阅 RefreshControl.SIZE.

ios tintColor ColorPropType

指定刷新指示器的颜色。

ios title string

指定刷新指示器下显示的文字。

4、组件的生命周期

概述

就像 Android 开发中的 View 一样，React Native（RN） 中的组件也有生命周期（Lifecycle）。所谓生命周期，就是一个对象从开始生成到最后消亡所经历的状态，理解生命周期，是合理开发的关键。RN 组件的生命周期整理如下图：

如图，可以把组件生命周期大致分为三个阶段：

- 第一阶段：是组件第一次绘制阶段，如图中的上面虚线框内，在这里完成了组件的加载和初始化；
- 第二阶段：是组件在运行和交互阶段，如图中左下角虚线框，这个阶段组件可以处理用户交互，或者接收事件更新界面；

- 第三阶段：是组件卸载消亡的阶段，如图中右下角的虚线框中，这里做一些组件的清理工作。

生命周期回调函数

下面来详细介绍生命周期中的各回调函数。

GETDEFAULTPROPS

在组件创建之前，会先调用 `getDefaultProps()`，这是全局调用一次，严格地来说，这不是组件的生命周期的一部分。在组件被创建并加载候，首先调用 `getInitialState()`，来初始化组件的状态。

COMPONENTWILLMOUNT

然后，准备加载组件，会调用 `componentWillMount()`，其原型如下：

```
void componentWillMount()
```

这个函数调用时机是在组件创建，并初始化了状态之后，在第一次绘制 `render()` 之前。可以在这里做一些业务初始化操作，也可以设置组件状态。这个函数在整个生命周期中只被调用一次。

COMPONENTDIDMOUNT

在组件第一次绘制之后，会调用 `componentDidMount()`，通知组件已经加载完成。函数原型如下：

```
void componentDidMount()
```

这个函数调用的时候，其虚拟 DOM 已经构建完成，你可以在这个函数开始获取其中的元素或者子组件了。需要注意的是，RN 框架是先调用子组件的 `componentDidMount()`，然后调用父组件的函数。从这个函数开始，就可以和 JS 其他框架交互了，例如设置计时 `setTimeout` 或者 `setInterval`，或者发起网络请求。这个函数也是只被调用一次。这个函数之后，就进入了稳定运行状态，等待事件触发。

COMPONENTWILLRECEIVEPROPS

如果组件收到新的属性（props），就会调用 `componentWillReceiveProps()`，其原型如下：

```
void componentWillReceiveProps(
  object nextProps
)
```

输入参数 nextProps 是即将被设置的属性，旧的属性还是可以通过 this.props 来获取。在这个回调函数里面，你可以根据属性的变化，通过调用 this.setState() 来更新你的组件状态，这里调用更新状态是安全的，并不会触发额外的 render() 调用。如下：

```
componentWillReceiveProps: function(nextProps) {
  this.setState({
    likesIncreasing: nextProps.likeCount > this.props.likeCount
  });
}
```

SHOULDCOMPONENTUPDATE

当组件接收到新的属性和状态改变的话，都会触发调用 shouldComponentUpdate(...)，函数原型如下：

```
boolean shouldComponentUpdate(
  object nextProps, object nextState
)
```

输入参数 nextProps 和上面的 componentWillReceiveProps 函数一样，nextState 表示组件即将更新的状态值。这个函数的返回值决定是否需要更新组件，如果 true 表示需要更新，继续走后面的更新流程。否者，则不更新，直接进入等待状态。

默认情况下，这个函数永远返回 true 用来保证数据变化的时候 UI 能够同步更新。在大型项目中，你可以自己重载这个函数，通过检查变化前后属性和状态，来决定 UI 是否需要更新，能有效提高应用性能。

componentWillUpdate

如果组件状态或者属性改变，并且上面的 shouldComponentUpdate(...) 返回为 true，就会开始准更新组件，并调用 componentWillUpdate()，其函数原型如下：

```
void componentWillUpdate(
  object nextProps, object nextState
)
```

输入参数与 shouldComponentUpdate 一样，在这个回调中，可以做一些在更新界面之前要做的事情。需要特别注意的是，在这个函数里面，你就不能使用 this.setState 来修改状态。这个函数调用之后，就会把 nextProps 和 nextState 分别设置到 this.props 和 this.state 中。紧接着这个函数，就会调用 render() 来更新界面了。

componentDidUpdate

调用了 render() 更新完成界面之后，会调用 componentDidUpdate() 来得到通知，其函数原型如下：

```
void componentDidUpdate(
  object prevProps, object prevState
)
```

因为到这里已经完成了属性和状态的更新了，此函数的输入参数变成了 prevProps 和 prevState。

componentWillUnmount

当组件要被从界面上移除的时候，就会调用 componentWillUnmount()，其函数原型如下：

```
void componentWillUnmount()
```

在这个函数中，可以做一些组件相关的清理工作，例如取消计时器、网络请求等。

总结

到这里，RN 的组件的完整的生命都介绍完了，在回头来看一下前面的图，就比较清晰了，把生命周期的回调函数总结成如下表格：

生命周期	调用次数	能否使用 setSate()
getDefaultProps	1(全局调用一次)	否
getInitialState	1	否
componentWillMount	1	是
render	>=1	否
componentDidMount	1	是
componentWillReceiveProps	>=0	是
shouldComponentUpdate	>=0	否
componentWillUpdate	>=0	否
componentDidUpdate	>=0	否
componentWillUnmount	1	否