高级阶段面试题

编写人: 全栈全体项目经理

一、Vue2.0面试题

1. vue生命周期? (必问)

1.1 什么是vue生命周期?

1.2 vue生命周期都有哪些钩子函数?这些钩子函数如何触发?

1.3 项目开发过程中,在生命周期里面都分别做过什么功能?

1.4 页面第一次加载时父子组件生命周期执行的顺序是什么?

> 面试官您好,我先介绍一下什么是vue的生命周期? 所谓的vue生命周期就是vue实例从创建到销毁的整个过程我们称之为vue的生命周期,通过vue的生命周期我们可以在不同的阶段进行不同的逻辑操作. vue生命周期常用的钩子函数一共有8个,分别是创建前后、挂载前后、更新前后以及销毁前后. 分别对应的钩子函数为beforeCreate 创建前、 created创建后、beforeMount 挂载前、mounted挂载后、beforeUpdate 更新前、updated更新后、beforeDestory 销毁前、 destoryed销毁后, 页面一开始加载的时候就会触发创建前后和挂载前后的钩子函数, 而更新的钩子函数需要当我们改变data的时候才能触发,比如 点击按钮,执行一个方法,在这个方式里面给data里面属性重新进行复制操作,这个时候就会更新的钩子函数, 销毁的钩子函数必须得当组件进行切换的时候就会进行销毁.
>
> 在项目开发过程中,我经常使用到的钩子函数有created,我们经常在created进行数据请求,或者获取本地存储的数据,还有一些其他的操作. 除了created还有mounted,我们经常在mounted里面获取dom元素 (有时 候也存在获取不到dom元素的情况,这个时候我们一般用$nextTick方法来解决). 以上就是我对生命周期的理解

2. vuex的理解? (必问) 讲解

2.1 什么是vuex?使用vuex能够解决什么问题?

2.2 vuex的五大核心是什么?

2.3 在组件里面如何调用五大核心的属性和方法?

2.4 vuex的执行机制是什么?

2.5 vuex的弊端是什么?怎么解决?

> 面试官您好,接下来我先给您介绍一下什么是vuex? 所谓的vuex其实就是vue官方给我们提供的一个状态管理工具,通过vuex我们可以组件之间数据共享的问题. vuex一共有5大核心,分别是state,里面保存的是状态,也可以理解为是数组, 接下来是getters, 他们用来获取state里面的状态,并且可以对state的数据进行处理之后在返回,有点类似于vue的计算属性, 接下来还有mutations, 他的作用主要是用来修改state里面的数据,不过在mutations里面只能进行同步的操作,还有actions,这个actions也可以去改变state的状态,不过在actions里面定义的方法可以进行异步操作,最后一个是modules,如果当我们的项目比较大的时候,那么保存的状态也会增加,如果都写到index.js文件里面,文件的内容就会变得特别臃肿,后期难以维护,所以我们可是使用modules进行模块化的处理,将多个状态抽离到对应js文件里面,最后在modules进行合并,这样后期就方便维护了.
>
> 接下来我在介绍一下在组件里面如何调用vuex里面的属性和方法,如果我们要获取state里面的状态,我们可以通过this.$store.state$来进行获取,如果要调用$getters$里面的方法,我们可以通过$"this.store.getter"$来进行调用,如果要调用$muations$里面的方法我们需要使用$this$,store.commit来触发,如果调用actions里面的方法,我们一般通过this.$store.dispacth来进行触发. 除了这种方式以外,我们还可以通过辅助函数的方式来进行调用和触发(mapState, mapMutation,mapAction, mapGetter)
>
> 我在项目当中如果要改变state的状态,我们一般是在组件里面调用this.$store.dispatch方式来触发actions里面的方法,在actions里面的方法通过commit来调用mutations里面定义的方法来改变state,同时这也是vuex的执行机制

不过vuex也有一些弊端,比如浏览器刷新的时候,vuex的数据会丢失,我们一般结合本地存储来解决,当我们在mutations里面改变state的时候在通过localStorage或者sessionStorage存储到本地,然后在state的状态的属性值那块写一个三元表达式,如果本地存在数据的话则读取本地存储的数据,否则就赋值为null

在项目当中我一般使用vuex会保存用户信息和token以及其他的一些状态. 以上就是我对vuex的理解.

3. vue路由有几种模式?有什么区别?原理是什么?(必问) 讲解

面试官您好,接下来我给您介绍一下vue的路由模式,vue的路由模式一共有两种,分别是哈希和history. 他们的区别是hash模式不会包含在http请求当中,并且hash不会重新加载页面,而使用history模式的话,如果前端的url和后端发起请求的url不一致的话,会报404错误,所以使用history模块的话我们需要和后端进行配合.

history的原理就是利用html5新增的两个特性方法,分别是psuhState和replaceState来完成的. 以上就是我对vue路由模式的理解.

4. vue路由守卫?(必问) 讲解页面权限

面试官您好,接下来我给您介绍一下vue路由守卫,首先呢,所谓的路由守卫就是当我们进行页面跳转的时候会触发的钩子函数,我们把它称之为vue路由守卫. vue一共给我们提供了三种路由守卫,第一种全局路由守卫,第二种是组件内路由守卫,第三种路由独享守卫,这个是写在路由里面. 不管是全局,还是组件以及独享,都会有beforeEach、beforeResolve 、afterEach 分别表示的意思路由跳转前会触发的钩子函数以及进入路由的时候,以及进入路由之后会触发的钩子函数. 这几个钩子函数里面都有一个回调函数,这个回调函数里面会有三个参数,分别是to,from,next,分别对应的是要进入的路由、离开之前的路由,以及进入写一个路由

在项目中我们经常使用路由守卫实现页面的鉴权. 比如:当用户登录之后,我们会把后台返回的token以及用户信息保存到vuex和本地,当页面进行跳转的时候,我们会在路由守卫里面获取vuex里面的token,如果token存在的话,我们则使用next让他进入要跳转的页面,如果token不存在的话我们使用next方法让他回到登录页

以上就是我对vue路由守卫的理解

5. v-if与v-show的区别?(90%)

面试官您好,接下来我给您介绍一下v-if和v-show的区别? 首先v-if和v-show都是控制元素的显示与隐藏, 不过v-if控制元素的显示和隐藏的时候会删除对用的dom元素,当每一个显示的时候,都会重新创建dom和渲染. 而v-show则是通过css的display:none和display:block来控制元素的显示与隐藏. v-if比较耗费性能,所以我们涉及到频繁的显示隐藏操作我们建议使用v-show,如果不是频繁操作的话,我们可以v-if

在项目中我会经常使用v-if和v-show,比如我们在搜索功能的时候,他有一个历史记录,这个时候我们根据是否有搜索的结果来判断历史记录的显示与隐藏,这块我就可以使用v-if ,当然用v-show也可以. 以上就是我对v-if和v-show的理解.

6. v-for与v-if的优先级那个高?如果同时使用v-for和v-if怎么解决?(90%) 讲解

v-for的优先级高. 因为v-for的时候我们才开始渲染dom元素,这个v-if还无法进行判断.

v-for和v-if不能同时使用,我们可以通过标签,比如div或者template标签来进行包裹,把v-if写到包裹的标签上面(写到v-for外面)

7. methods、computed和watch的区别?(90%)。讲解

首先呢,methods是用来定义方法的区域,methods定义的方法需要调用才能触发. 不具备缓存行

而computed是计算属性,他依赖于属性值的变化,当属性发生改变的时候,计算属性里面定义的方法就会触发,computed具有缓存性,依赖属性值的变化而变化.

而watch主要是用于监听,不具备被缓存性.依赖于数据变化而触发.

在项目中,比如我们获取state的状态的时候我们会把它放到computed里面,或者在写购物车数量计算的时候也会使用计算属性.

而watch也在项目经常使用,比如我们封装编辑 和 新增弹窗组件的时候会通过watch来进行id判断我们要显否要清空表单的数据.

以上就是我对computed和watch的理解.

8. vue组件通信?(必问)。eventBus 讲解

父传子 在子组件的标签上定义属性 子组件通过props来进行接受,可以通过数组的方式进行接受,也可以通过对象的方式来进行接收,如果父组件没有传递属性,子组件可以default来设置默认值

子传父 子组件通过this.$emit("自定义的事件",要传给父组件的数据), 父组件通过子组件的标签监听自定义的时间,通过方法来接收传递的数据

非父子组件通信

通过中央事件总线,我们也称之为eventBus,

我们需要创建一个空的js文件,在这个文件里面创建空的vue实例,然后导出这个空的vue实例,通过实例对象调用.on方法进行接收,通过emit方法来进行发送,以上就是非父子组件通信的方式

9. $nextTick方法有什么作用?(80%) 讲解

首先呢,$nextTick$也叫做异步更新队列方法,而$nextTick$方法的主要作用就是等待dom元素加载完毕之后才会执行的回调函数,我们经常会在nextTick方法里面获取dom元素

10. 说一下什么是mvvm模式?(70%)

MVVM 是把 MVC 的 Controller 和 MVP 的 Presenter 改成了 ViewModel 。

View 的变化会自动更新到 ViewModel， ViewModel 的变化也会自动同步到 View 上显示。这种自动

同步是因为 ViewModel 中的属性实现了 Observer， 当属性变更时都能触发对应的操作

11. vue双向数据绑定原理?(必问) 过一下

vue.js 则是采用 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。

这个时候就可以实现数据的双向绑定

12. vue常用的指令有哪些?(50%)

v-if

v-show

v-html

v-text

v-on

v-bind

v-model

v-for

13. vue常用的修饰符有哪些?(50%)

.trim 去除首尾多余的空格

.stop 阻止事件冒泡

.once 只渲染一次

.self 事件只作用在元素本身

.number 将值转化为number类型

.capter 组件之间捕获

.prevent 阻止元素的默认行为

.native 事件穿透,让我们可以在自定义组件上定义事件和方法

14. vue如何封装可复用的组件?以及请举例说明你封装过的组件?(50%) 讲解

1. 分析项目的所有页面结构和业务功能,抽离出相同的页面结构和业务功能
2. 在src目录下创建一个components这个的文件夹
3. 在这个文件夹内创建可复用的组件
4. 在需要的用的组件里面引入创建的这个可复用的组件,并进行注册,以标签的形式写在对应的地方
5. 接下来就需要对可复用的组件内容要进行封装,那么在封装的时候我们要注意组件的封闭性和开放性以及粗细粒度
6. 所谓的这个封闭性就是当我们在组件内部定义好之后外部是无法进行修改的,比如当前有一个关闭的符号,或者有一个箭头,我们需要不管任何人使用该组件时候都能够显示这个箭头,无法从外部进行修改
7. 所谓的开放性就是我们需要将动态的内容以及组件通信的方式进行数据传递,保证组件的可扩展性
8. 而所谓的粗细力度就是我们可以把一整个页面当作一个组件去进行封装,也可以一个页面包含了多个组件,至于到底选择哪种呢,这个是没有一个标准的,我们需要根据自己的业务需求去进行判断
9. 总结来说,所谓的如何封装可复用组件其实技术核心就是通过我们vue提供的组件通信在结合slot插槽来进行分装
10. 比如:封装一个搜索框组件:
    1. 在components里面创建search.vue
    2. 在search.vue里面实现搜索框的布局
    3. 在props里面接受 title, bgColor, size , icon,以及当点击搜索按钮或者点击回车键的时候,触发一个方法,通过this.$emit将输入框输入的值传递给父组件
    4. 接下来要使用这个搜索组件,我们需要通过import 在父组件内引入子组件,并在componetns属性里面进行注册,
    5. 在页面就可以使用,这个时候我们可以通过传递titile老控制子组件搜索框显示的内容,通过bgcolor可以控制搜索框的背景颜色,也可以通过size设置搜索框字体的大小,也可以通过icon来设置搜索框的图标, 通过监听$emit里面定义的方法来获取搜索框输入的值

    以上就是封装的过程

15. vue中key的作用是什么?(必问)。过一下

避免dom元素重复渲染. 我门一般在设置key的时候首先尽量会设置为id,或者index下表.

16. 说一下你对keep-alive的理解?以及在项目中如何使用?(90%) 过一下

keep-alive是vue内置的一个组件，而这个组件的作用就是能够缓存不活动的组件，我们能够知道，一般情况下，组件进行切换的时候，默认会进行销毁，如果有需求，某个组件切换后不进行销毁，而是保存之前的状态，比如说刚刚填好的表单数据。那么就可以利用keep-alive来实现

在搭建 vue 项目时，有某些路由组件没必要多次渲染，所以需要将组件在内存中进行‘持久化’，此时在router-view上使用keep-alive。 keep-alive可以使被包含的路由组件状态维持不变，即便是组件切换了，其内的状态依旧维持在内存之中。在下一次显示时，也不会重新渲染。

include - 字符串或正则表达式。只有名称匹配的组件会被缓存。 exclude - 字符串或正则表达式。任何名称匹配的组件都不会被缓存。 max-数字最多可以缓存多少组件。

以上就是我对keep-alive的理解

17. 说一下什么是vue过滤器? 有几种?项目中如何使用,请举例说明?(60%)

所谓的vue过滤器就是将数据进行二次处理,得到我们想要的结果数据

vue的过滤器分为两种,第一种是全局过滤器,通过vue.filet来进行定义,第二种是局部过滤器,需要定义在组件内部

项目中我们通过过滤器将后台返回的状态0 和1 转化为支付或者未支付

18. 说一下你对slot插槽的理解?(70%) 过一下

首先呢,所谓的插槽就是一个占位符,将自定义组件的内容展示出来.我们知道自定义的组件里面如果写内容的话,页面是不会显示出来的,如果我们想让自定义组件里面的内容显示出来,我们就需要使用slot的插槽.

而插槽分别具名插槽和匿名插槽、以及作用域插槽. 我们用的比较多的具名插槽和匿名插槽,具名插槽需要所有slot标签上指定name属性,而在对应标签上添加v-slot属性.

在项目中我们一般在进行组件封装的时候会使用插槽,以上就是我对插槽的理解.

19. 说一下vue中本地跨域如何解决?线上跨域如何解决?(必问) 线上在讲解一下

本地跨域是通过在vue.config.js文件里面的devServer属性里面的proxy属性里面配置,一共配置三个属性,分别是代理名称 代理地址 开启跨域 重写路径

线上跨域是在nginx.conf文件里面配置, 代理名称是通过location 代理名称。proxy_pass 代理地址

20. 说一下如何对axios进行二次封装?以及api如何封装?(30%)

1. 在src文件夹内创建utils文件夹
2. 在utils文件夹内创建request.js文件
3. 在request.js内引入axios
4. 使用axios.create方法创建axios的实例,在axios.create方法里面可以配置请求的公共地址和超时时间以及其他的一些配置
5. 在创建请求拦截器和响应拦截器
6. 在请求拦截器里面可以获取vuex的token,并通过config.header.token = vuex的token,将token发送给后台
7. 在请求拦截器里面我们配置loading加载
8. 在响应拦截器里面我们可以结束loading加载以及token的过期处理,以及错误响应信息的处理
9. 最后通过export default 导出axios的实例对象
10. 在src文件内创建api文件夹
11. 在api文件夹内创建对应模块的js文件
12. 在对应的文件里面引入request.js文件
13. 封装api方法
14. 最后通过export default 导出封装的api方法

21. 说一下axios的拦截器的作用?应用场景都有哪些?(80%)

> 首先呢,axios拦截器是axios给我们提供的两个方法,通过这两个方法我们可以对请求发送之前以及响应之后进行逻辑的再次处理(拦截). 这两个拦截器不需要手动触发,只要发送http请求的时候就会自动触发. 我在项目中经常通过拦截器发送token, 对token进行过期处理,以及进行其他的一些操作

22. 说一下vue和jquery的区别?(50%)

> 首先呢jquery他是用js封装的一个类库,主要是为了方便操作dom元素,而vue他是一个框架,并且呢,他会从真实dom构建出一个虚拟的dom树,通过diff算法渲染只发生改变的dom元素,其他的相同的dom元素不用在重新渲染. 而使用jquery去改变dom元素的时候,即使有相同的dom元素也会重新渲染,以上就是我对vue和jquery区别的理解.

23. vue中data发生变化,视图不更新如何解决?(必问) 过一下

> 面试官,您好,接下来我先给您介绍一下为什么data发生变化,视图不更新,**因为Vue实例中的数据是响应式的**而我们新增的属性并不是响应式的，由于受现在JavaScript的限制，Vue无法检测到属性的新增或删除。所以有时无法实时的更新到视图上。**
>
> 所以我在项目中遇到这类问题的时候一般是通过this.$set方法去解决. this.$set方法一共有三个参数,分别是目前属性,新增属性,新增的值.
>
> 以上就是我对视图不更新的理解.

24. 为什么vue中data必须是一个函数?(必问) 过一下

> 如果data是一个函数的话，这样每复用一次组件，就会返回一份新的data，类似于给每个组件实例创建一个私有的数据空间，让各个组件实例维护各自的数据。而单纯的写成对象形式，就使得所有组件实例共用了一份data，就会造成一个变了全都会变的结果。
>
> 所以说vue组件的data必须是函数。这都是因为js的特性带来的，跟vue本身设计无关。

25.MVVM模式的优点以及与MVC模式的区别?

MVVM模式的优点：

`1、低耦合：` 视图（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：` 你可以把一些视图逻辑放在一个ViewModel里面，让很多 view 重用这段视图逻辑。

`3、独立开发：` 开发人员可以专注于业务逻辑和数据的开发（ViewModel），设计人员可以专注于页面设计。

`4、可测试：` 界面素来是比较难于测试的，而现在测试可以针对ViewModel来写。

MVVM 和 MVC 的区别：

`mvc` 和 `mvvm` 其实区别并不大。都是一种设计思想。

> 主要区别
>
> mvc 中 Controller演变成 mvvm 中的 viewModel，
>
> mvvm 通过数据来显示视图层而不是节点操作。

> mvvm主要解决了:
>
> mvc中大量的DOM 操作使页面渲染性能降低，加载速度变慢，影响用户体验。

26. 怎样理解 Vue 的单向数据流?

数据总是从父组件传到子组件，子组件没有权利修改父组件传过来的数据，只能请求父组件对原始数据进行修改。这样会防止从子组件意外改变父级组件的状态，从而导致你的应用的数据流向难以理解。

> 注意：在子组件直接用 v-model 绑定父组件传过来的 prop 这样是不规范的写法 开发环境会报警告

如果实在要改变父组件的 prop 值 可以再 data 里面定义一个变量 并用 prop 的值初始化它 之后用$emit 通知父组件去修改

27. 虚拟 DOM 是什么? 有什么优缺点?

什么是虚拟dom?

由于在浏览器中操作 DOM 是很昂贵的。频繁的操作 DOM，会产生一定的性能问题. 所以在vue中将真实的DOM节点抽离成一个虚拟的DOM树,这个虚拟的DOM树就是虚拟DOM.

优点:

- 保证性能下限： 框架的虚拟 DOM 需要适配任何上层 API 可能产生的操作，它的一些 DOM 操作的实现必须是普适的，所以它的性能并不是最优的；但是比起粗暴的 DOM 操作性能要好很多，因此框架的虚拟 DOM 至少可以保证在你不需要手动优化的情况下，依然可以提供还不错的性能，即保证性能的下限；
- 无需手动操作 DOM： 我们不再需要手动去操作 DOM，只需要写好 View-Model 的代码逻辑，框架会根据虚拟 DOM 和 数据双向绑定，帮我们以可预期的方式更新视图，极大提高我们的开发效率；
- 跨平台： 虚拟 DOM 本质上是 JavaScript 对象,而 DOM 与平台强相关，相比之下虚拟 DOM 可以进行更方便地跨平台操作，例如服务器渲染、weex 开发等等。

缺点:

- 无法进行极致优化： 虽然虚拟 DOM + 合理的优化，足以应对绝大部分应用的性能需求，但在一些性能要求极高的应用中虚拟 DOM 无法进行针对性的极致优化。
- 首次渲染大量 DOM 时，由于多了一层虚拟 DOM 的计算，会比 innerHTML 插入慢

28. Vue的diff算法原理是什么?

Vue的diff算法是平级比较，不考虑跨级比较的情况。内部采用深度递归的方式+双指针方式比较

- 先比较两个节点是不是相同节点
- 相同节点比较属性，复用老节点
- 先比较儿子节点，考虑老节点和新节点儿子的情况
- 优化比较：头头、尾尾、头尾、尾头
- 比对查找，进行复用

29.组件写name有啥好处?

- 增加name属性，会在components属性中增加组件本身，实现组件的递归调用。
- 可以表示组件的具体名称，方便调试和查找对应的组件。

二、微信小程序面试题

1. 小程序有几个文件?

- `WXML` ： 微信自己定义的一套组件
- `WXSS` :用于描述 `WXML` 的组件样式
- `js` :逻辑处理
- `json` :小程序页面配置

2 小程序怎么跟随事件传值

> 在 页面标签上通过 绑定 `dataset-key = value` ， 然后绑定点击通过 `e.currentTarget.dataset.key` 来获取标签上绑定的值。

```
<button bindtap="get"  data-name="测试"> 拿到传值</button>

get(e){
    console.log(e.currentTarget.dataset.name)
  },
```

3 小程序WXSS与CSS 的区别

> **WXSS**

- `wxss` 背景图片只能引入外链，不能使用本地图片
- 小程序样式使用 `@import` 引入 外联样式文件，地址为相对路径。
- 尺寸单位为 `rpx` ，`rpx` 是响应式像素,可以根据屏幕宽度进行自适应。

4 小程序的双向绑定和Vue哪里不一样?

> 小程序 直接使用 `this.data.key = value` 是 不能更新到视图当中的。 必须使用 `this.setData({ key : value })` 来更新值。

5 小程序的生命周期函数

- `onLoad` :页面加载时触发。一个页面只会调用一次，可以在 `onLoad` 的参数中获取打开当前页面路径中的参数
- `onShow` :页面显示 / 切入前台时触发调用。
- `onReady` :页面初次渲染完成时触发,一个页面只会调用一次。
- `onHide` :页面隐藏 / 切入后台时触发，如 `navigateTo` 或底部 `tab` 切换到其他页面，小程序切入后台等
- `onUnload` :页面卸载时触发。如 `redirectTo` 或 `navigateBack` 到其他页面时.

6 小程序怎么实现下拉刷新

> 两种方案 方案 一 :

- 通过在 `app.json` 中， 将 `"enablePullDownRefresh": true,` 开启全局下拉刷新。
- 或者通过在 `组件 .json` ， 将 `"enablePullDownRefresh": true,` 单组件下拉刷新。

方案二:

- `scroll-view` ： 使用该滚动组件 自定义刷新，通过 `bindscrolltoupper` 属性， 当滚动到顶部/左边，会触发 `scrolltoupper` 事件，所以我们可以利用这个属性，来实现下拉刷新功能。

7. 小程序有哪些传递数据的方法

> 1. 使用全局变量

- 在 `app.js` 中的 `this.globalData = { }` 中放入要存储的数据。
- 在 `组件.js` 中， 头部 引入 `const app = getApp();` 获取到全局变量
- 直接使用 `app.globalData.key` 来进行赋值和获取值。

2. 使用 路由

- `wx.navigateTo` 和 `wx.redirectTo` 时，可以通过在 `url` 后 拼接 + 变量， 然后在 `目标页面` 通过在 `onLoad` 周期中，通过参数来获取传递过来的值。

3. 使用本地缓存

8. 简述下wx.navigateTo(),wx.redirectTo(),wx.switchTab(),wx.navigateBack(),wx.reLaunch() 区别

- `wx.navigateTo()` :保留当前页面，跳转到应用内的某个页面。但是不能跳到 `tabbar` 页面
- `wx.redirectTo()` :关闭当前页面，跳转到应用内的某个页面。但是不允许跳转到 `tabbar` 页面
- `wx.switchTab()` :跳转到 `TabBar` 页面，并关闭其他所有非 `tabBar` 页面
- `wx.navigateBack()` :关闭当前页面，返回上一页面或多级页面。可通过 `getCurrentPages()` 获取当前的页面栈，决定需要返回几层
- `wx.reLaunch()` :关闭所有页面，打开到应用的某个页面。

9. 小程序wx:if和 hidden`的区别

- `wx:if` :有更高的切换消耗。
- `hidden` :有更高的初始渲染消耗。

使用

- 频繁切换使用 `hidden` ,运行时条件变化使用 `wx: if`

10. app.json全局配置文件描述

- `pages` :用于存放当前小程序的所有页面路径
- `window` :小程序所有页面的顶部背景颜色，文字颜色配置。
- `tabBar` :小程序底部的 `Tab` ,最多5个，最少2个。

11. 如何封装小程序请求

- 封装 `wx.request` 请求传递需要的参数(`url` , `data` , `method` , `success 成功回调` , `fail 失败回调`),封装常用方法 `POST` , `GET` , `DELETE` , `PUT` .... 最后导出这些方法
- 然后新建一个 `api.js` 文件，导入封装好的方法，然后调取相应的方法，传递数据。

wx.request 封装

```
var app = getApp(); //获取小程序全局唯一app实例
var host = '******************'; //接口地址



//POST请求
function post(url, data, success,fail) {
  request(url, postData, "POST", doSuccess, doFail);
}

//GET请求
function get(url, data, success, fail) {
  request(url, postData, "GET", doSuccess, doFail);
}

function request(url, data, method, success, fail) {
```

```
    wx.showLoading({
      title: "正在加载中...",
    })
    wx.request({
      url: host + url, //请求地址
      method: method, //请求方法
      header: { //请求头
        "Content-Type": "application/json;charset=UTF-8"
      },
      data: data, //请求参数
      dataType: 'json', //返回数据格式
      responseType: 'text', //响应的数据类型
      success: function(res) {
        wx.hideLoading();
        //成功执行方法，参数值为res.data,直接将返回的数据传入
        success(res.data);
      },
      fail: function() {
        //失败执行方法
        fail();
      },
    })
  }
  module.exports = {
    postRequest: post,
    getRequest: get,
  }
```

组件使用 封装好的请求

```
  var http = require('../../utils/request.js'); //相对路径


  var params = {//请求参数
    id:this.data.userId
  }
  http.postRequest("user/delUser", params, function(res) {
    console.log("修改成功！");

  }, function(res) {
    console.log("修改失败！！！")
  })
```