ElementUI,iviewUI,antdesign,mitiUI,

**你对MVVM的理解是什么?**

MVVM分为Model、View、ViewModel三者。

Model 代表数据模型，数据和业务逻辑都在Model层中定义；

View 代表UI视图，负责数据的展示；

ViewModel 负责监听 Model 中数据的改变并且控制视图的更新，处理用户交互操作；

Model 和 View 并无直接关联，而是通过 ViewModel 来进行联系的，Model 和 ViewModel 之间有着双向数据绑定的联系。因此当 Model 中的数据改变时会触发 View 层的刷新，View 中由于用户交互操作而改变的数据也会在 Model 中同步。

我们可以不关心DOM的结构，而是关心数据如何存储。

我们把数据看作Model，把DOM节点看作View，并假定它们之间通过ViewModel关联起来了。我们并不操作DOM，而是直接修改JavaScript对象,通过改变JavaScript对象的状态，会导致DOM结构作出对应的变化, 这就是MVVM的设计思想：关注Model的变化，让MVVM框架去自动更新DOM的状态，从而把开发者从操作DOM的繁琐步骤中解脱出来.

**MVC,MVP,MVVM的区别?**

MVC: MVC之间的数据通信都是单向的。View(视图层)发送指令到controller(控制层)，完成业务逻辑后，要求Model(模型层)改变状态，匠心的数据发送到(view)视图层，用户得到反馈。

MVP:在MVP中，view和model之间没有任何通信关系，所有的通信和业务逻辑都放在presenter层中。View层发送指令到presenter层，presenter层处理业务逻辑，要求model层改变状态，完成状态修改之后，发送指令到presenter层，之后再通知view层做出改变。

MVVM: Model专门用来处理数据模型。View专门用来处理用户视图，ViewModel用来使view和model双向绑定，view的任何变化都会通知ViewModel,而model的任何变化也会通知ViewModel，无论哪一项发生改变，都会使对应的视图/数据模型同时发生改变。

**数据双向绑定实现的原理(Vue的响应式原理)?**

vue数据双向绑定是通过数据劫持结合发布者-订阅者模式的方式来实现的.

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，触发相应监听回调。当把一个普通 Javascript 对象传给 Vue 实例来作为它的 data 选项时，Vue 将遍历它的属性，用 Object.defineProperty 将它们转为 getter/setter。用户看不到 getter/setter，但是在内部它们让 Vue 追踪依赖，在属性被访问和修改时通知变化。

将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的数据变化，通过Compile来解析编译模板指令（用来解析双大括号{{}}），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变更双向绑定效果。

**如何理解Vue的渐进式框架?**

使用vue，你可以在原有大系统的上面，把一两个组件改用它实现，当jQuery用；也可以整个用它全家桶开发，当Angular用；

还可以用它的视图，搭配你自己设计的整个下层用它只是个轻量视图而已，只做了自己该做的事，没有做不该做的事，仅此而已。你不必一开始就用Vue所有的全家桶，根据场景，官方提供了方便的框架供你使用。比如:

你只想用Vue的声明式渲染,组件系统和数据双向绑定来管理你整个页面的DOM;当你觉得不够的时候,可以继续引入Vue客户端路由控制路由,引入axios来对接后台数据,或者继续引入vuex来实现大规模状态管理.

根据场景，官方都提供了方便的框架供我们使用.

**你是怎么使用Vue的?**

通过vuecli脚手架工具搭建 vue（整体架构）;

使用vue-router路由模块来管理页面路由调转;

使用vuex进行大型数据状态管理.

提取各个页面的公共组件,编写各个组件.

使用mint-UI(移动端UI框架库) 或antd-vue(PC端UI框架库)来构建UI组件.

使用axios（ajax请求）发送请求,对接后台数据,实现数据拦截.

后期优化显示等.

测试.

上线.

**Vue的特性有哪些?(简单介绍一下Vue)**

(1)数据双向绑定

vue.js最大的优点，通过MVVM思想实现数据的双向绑定，让开发者不用再操作dom对象，有更多的时间去思考业务逻辑。

(2) 组件化开发

提高开发效率,方便重复使用.

(3) Virtual DOM

虚拟DOM属于预处理操作，并没有真实的操作DOM，所以叫做虚拟DOM。最后在计算完毕才真正将DOM操作提交，将DOM操作变化反映到DOM树上。

(4) 动画系统

**Vue 有哪些指令?指令有哪些好处?封装过自己的指令吗?**

1. v-text: 更新元素文本

2. v-html: 等同于JS的innerHtml属性。

3. v-pre: 已知页面内部有大段内容无需编译，使用v-pre指令阻止编译可以提高性能.

4. v-cloak:这防止页面加载时出现 vuejs 的变量名而设计的.

5. v-once: Vue性能优化:vue-once关联的实例，只会渲染一次。之后的重新渲染，所有的子节点将被视为静态内容跳过，这可以用于优化更新性能。

6. v-if:实现条件渲染，Vue会根据表达式的值的真假条件来渲染元素。

7. v-else:v-else是搭配v-if使用的，它必须紧跟在v-if或者v-else-if后面，否则不起作用。

8. v-show: 根据条件展示元素。和v-if不同的是，如果v-if的值是false，则这个元素被销毁，不在dom中。但是v-show的元素会始终被渲染并保存在dom中，它只是简单的切换css的dispaly为none。

Ps: v-if有更高的切换开销,v-show有更高的初始渲染开销。

因此，如果要非常频繁的切换，则使用v-show较好.

9. v-for: 用v-for指令根据遍历数组来进行渲染.

10. v-bind:属性绑定, 常用于动态绑定class和style.

11. v-model:用于表单元素的双向绑定, v-model会忽略所有表单元素的value、checked、selected特性的初始值。因为它选择Vue实例数据做为具体的值。相当于react的受控表单.

Ps: v-model修饰符:

(1): .lazy;默认情况下，v-model同步输入框的值和数据。可以通过这个修饰符，转变为在change事件再同步。

(2): .number自动将用户的输入值转化为数值类型.

(3) .trim自动过滤用户输入的首尾空格

12. v-on:主要用来监听dom事件.

Ps: v-on修饰符:

(1) .stop 阻止事件继续传播

(2) .prevent 事件不再重载页面

(3) .capture 使用事件捕获模式,即元素自身触发的事件先在此处处理，然后才交由内部元素进行处理

(4) .self 只当在 event.target 是当前元素自身时触发处理函数

(5) .once 事件将只会触发一次

(6) .passive 告诉浏览器你不想阻止事件的默认行为.

(7) .native,父组件给子组件绑定事件,将子组件变成了普通的html标签,此时不加.native修饰符是无法触发绑定在组件上事件的.

自定义指令好就好在：原先的那些通用组件，无论是纯js的也好，基于jQuery的也好，都可以拿来主义直接吸收，而不需要改造或重构。

第三方插件如何与Vue.js集成的问题，都可以尝试用自定义指令实现。

我写文档通常会用到highlight.js，我们可以直接将其封装为一个自定义指令，这样highlight.js就变成了Vue的一个新功能。

我用自定义指令实现过一个常见的 tip 提示弹框的功能，鼠标移入元素，可以在元素的右方显示 tip，鼠标移出则隐藏 tip。

文本框限制只能输入指定位数的小数，写正则每次去匹配很麻烦，所以我写了一个自定义指令,使用起来更方便.

**什么是vue生命周期？**

答： Vue 实例从创建到销毁的过程，就是生命周期。从开始创建、初始化数据、编译模板、挂载Dom→渲染、更新→渲染、销毁等一系列过程，称之为 Vue 的生命周期。

(1): beforeCreate（创建前） 在数据观测和初始化事件还未开始

(2): created（创建后） 完成数据观测，属性和方法的运算，初始化事件，$el属性还没有显示出来

(3): beforeMount（载入前） 在挂载开始之前被调用，相关的render函数首次被调用。实例已完成以下的配置：编译模板，把data里面的数据和模板生成html。注意此时还没有挂载html到页面上。

(4) mounted（载入后） 在el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用。实例已完成以下的配置：用上面编译好的html内容替换el属性指向的DOM对象。完成模板中的html渲染到html页面中。此过程中进行ajax交互。

(5): beforeUpdate（更新前） 在数据更新之前调用，发生在虚拟DOM重新渲染和打补丁之前。可以在该钩子中进一步地更改状态，不会触发附加的重渲染过程。

(6): updated（更新后） 在由于数据更改导致的虚拟DOM重新渲染和打补丁之后调用。调用时，组件DOM已经更新，所以可以执行依赖于DOM的操作。然而在大多数情况下，应该避免在此期间更改状态，因为这可能会导致更新无限循环。该钩子在服务器端渲染期间不被调用。

(7): beforeDestroy（销毁前） 在实例销毁之前调用。实例仍然完全可用。

(8): destroyed（销毁后） 在实例销毁之后调用。调用后，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。

**vue生命周期的作用是什么？**

答：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。

**第一次页面加载会触发哪几个钩子？**

答：会触发 下面这几个beforeCreate, created, beforeMount, mounted 。

**DOM 渲染在 哪个周期中就已经完成？**

答：DOM 渲染在 mounted 中就已经完成了。

**在Vue的各个生命周期里的业务逻辑是怎样的?**

1、 beforeCreate （使用频率低）

\* 没有实例化，data,methods数据访问不到, 此时加一些loading效果，在created时进行移除.

2、 created （使用频率高）

\* 这个阶段已经完成了数据观测,属性和方法的运算,及watch/event回调.但是$el属性不可见,数据没有在DOM元素里显示.

\* 修改数据不会触发updated beforeUpdate钩子函数

\* 可以在这个钩子函数里发ajax请求，初始化数据.

3、 beforeMounted（）

\* 相关的render函数首次被调用.

4、mounted（使用频率高）

\* el选项的DOM节点被新创建的vm.$el替换,并挂载到实例上去,此时数据在DOM节点上进行渲染.可以在这个生命周期获取dom节点.

\*可以用来操作dom；也可以在此发ajax

5、 beforeUpdate（）

\* 数据更新时调用,但不进行DOM重新渲染,在数据更新时DOM渲染前可以在这个生命函数里进行状态处理.

6、 updated（）

\*数据更新要做统一业务处理的时候使用.

\*这个状态下数据更新并且DOM重新渲染完毕,可以执行依赖于DOM的操作.

7.beforeDestory:实例被销毁前调用.

8.destoryed:Vue实例被销毁后调用,在这里解绑事件监听器.

**Vue.nextTick**

由于Vue中DOM更新是异步的，Vue.nextTick用于在DOM渲染更新完毕后执行回调函数.它接受2个参数（回调函数和执行回调函数的上下文环境），如果没有提供回调函数，那么将返回promise对象。

所以在Vue生命周期进行的DOM操作一定要放在Vue.nextTick()的回调函数中.

created(){

this.$nextTick(() => {

this.num= this.$refs.myDOM.innerText

})

}

**Vue的路由实现：hash模式 和 history模式**

hash模式：在浏览器中符号“#”，#以及#后面的字符称之为hash，用window.location.hash读取；

特点：hash虽然在URL中，但不被包括在HTTP请求中；用来指导浏览器动作，对服务端安全无用，hash不会重加载页面。

history模式：history采用HTML5的新特性；且提供了两个新方法：pushState（），replaceState（）可以对浏览器历史记录栈进行修改，以及popState事件的监听到状态变更。

**$route和$router的区别?**

$route是“路由信息对象”，包括path，params，hash，query，fullPath，matched，name等路由信息参数。而$router是“路由实例”对象包括了路由的跳转方法，钩子函数等。

**对keep-alive 的了解？**

keep-alive是 Vue 内置的一个组件，可以使被包含的组件保留状态，或避免重新渲染。

在vue 2.1.0 版本之后，keep-alive新加入了两个属性: include(包含的组件缓存) 与 exclude(排除的组件不缓存，优先级大于include) 。

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

include 和 exclude 的属性允许组件有条件地缓存。二者都可以用“，”分隔字符串、正则表达式、数组。当使用正则或者是数组时，要记得使用v-bind 。

**路由传参的方法？**

1. 字符串拼接 ： 路由后面直接拼接要传递的参数，用this.$rote.params 接收。

2. path和query：path后面跟要跳转的路由，query后边跟要传递参数的对象用this.$route.query接收。

3. name和params：name后面跟要跳转路由的名称，params后面跟传递参数的对象，用this.$route.params接收。

**Vue实现前进刷新,后退不刷新.**

场景一: 在一个列表页中，第一次进入的时候，请求获取数据。

点击某个列表项，跳到详情页，再从详情页后退回到列表页时，不刷新。

实现: keep-alive组件的include属性的动态修改可以动态设置需要缓存的组件，来控制组件数据的刷新

在App.vue设置:

<keep-alive>

<router-view include="list"></router-view>

</keep-alive>

列表页为 list.vue，详情页为 detail.vue，这两个都是子组件。

我们在 keep-alive 添加列表页的名字，缓存列表页。

然后在列表页的 created 函数里添加 ajax请求，这样只有第一次进入到列表页的时候才会请求数据，当从列表页跳到详情页，再从详情页回来的时候，列表页就不会刷新。

场景二:在需求一的基础上，再加一个要求：可以在详情页中删除对应的列表项，这时返回到列表页时需要刷新重新获取数据。

实现:在路由配置文件上对 详情页detail.vue 增加一个 meta 属性

1. {
2. path: '/detail',
3. name: 'detail',
4. component: () => import('../view/detail.vue'),
5. meta: {isRefresh: true}
6. }

在详情页中通过 this.$route.meta.isRefresh 来读取和设置。

在列表页里，通过beforeEach路由钩子函数判断from.meta值的变化，再请求一遍数据.

**怎么配置子路由?**

1.使用子路由,即在配置路由的时候,使用children来配置父级路由下的子路由.

2.children可以配置多个路由路径,使用数组和对象的形式.

3.子路由的路径不需要再加'/'了,只需要写父级路由下的路径那部分即可

4.如果访问不存在的子路由,没有配置子路由,它什么都不会渲染,如果想要渲染一些东西的话,在children数组里面添加path:"",component:"xxx",即表示访问未配置的子路径时渲染的组件.

var router = new VueRouter({

routes:[{

path:"/main",

component:lay\_main,

children:[{path:"news",component:lay\_news},{path:"msg",component:lay\_msg}]

}]

})

**路由守护钩子函数有哪些?**

路由导航(守护)就是监听路由发生变化,比如从一个路由进入另一个路由之前,之后;或者进入某一个路由之前,由函数来控制路由导航.

全局前置守卫: router.beforeEach((to, from, next) => {..}):

应用场景:在进入某些路由组件时判断用户是否登录,登录过的就next()放行,否则要求用户登录.

全局后置守卫: router.afterEach((to, from):

不接受 next 函数,在进入某些路由组件时使用逻辑.

组件独享的守卫:

(1) 路由独享的守卫beforeRouteEnter:(to,from,next)=>{}:

beforeRouteEnter:(to,from,next)=>{

next(vm=>{

alert("hello" + vm.name);

})

}

//当守卫执行前，组件实例还没被创建，不能获取this,data数据还未渲染。所以这里，next()会给一个对应的回调，使用vm.data来获取数据data。

(2) 路由独享的守卫beforeRouteUpdate (to, from, next)=>{}:

在路由改变且组件被复用时调用,即路由动态参数跳转的是当前组件,当使用动态路由之间跳转时可以使用. 可以访问组件实例 `this`

(3) beforeRouteLeave(to, from, next)=>{}:

通常用来禁止用户在还`未保存修改前`突然离开。该导航可以通过 next(false) 来取消.

路由独享的守卫:beforeEnter:(to,from,next)=>{} 用法与全局守卫一致。只是，将其写进其中一个路由对象中，只在这个路由下起作用.

**watch和computed的区别是什么？**

computed看是计算属性，它会根据你所依赖的数据动态显示新的计算结果。计算结果会被缓存，只有在它依赖的属性值改变之后，下一次获取computed的值时才会重新调用对应的getter方法来计算.

如果一个数据依赖于大量其他数据，那么应该用computed.

watcher 是一个 data 的数据监听回调，当依赖的 data 的数据变化，执行回调.

watch比较适合对状态的监控，比如监控页面一个变量的值改变，需要进行什么操作。而computed适合简单计算并返回结果，结果随着内部变量改变而改变。

watch适合比较耗时的操作，比如网络异步请求，一个变量改变触发网络请求。

**例举Vue组件通信的几种方式.**

1. 父组件传子组件:props:

父组件使用子组件,在组件身上使用v-bind属性绑定绑定数据;(注:组件中的数据有三种形式:data,props,computed)

子组件声明props属性,可以在template模版内直接使用数据,或者在生命周期内this.props取得数据.有必要的话开启props参数验证.例如:

props:{

users:{

type:Array,

required:true

}

}

2.子组件传递父组件:$emit + 通过events事件传参

在子组件里面,通过this.$emit("eventname","我是子组件的参数")来自定义事件名称和传递子组件要传参数.

在父组件里,使用子组件,在子组件身上通过v-on绑定自定义事件,触发methods的方法,在方法里面通过打印e即取得组件参数.

<child v-on:eventname="test"></child>

methods:{test(e){console.log(e)}} // 打印我是子组件的参数

3.$emit和$on + $parent实现父子通信

在子组件里面,通过this.$parent.$emit("eventname","我是子组件的参数")来自定义事件名称和传递子组件要传参数.

在父组件里通过this.$on("eventname",data=>{console.log(data)}),通过事件名称和回调函数取得参数

注意:因为$ on是在父组件中,所以要从当前子组件出发来找到父组件,即使用this.$parent,再使用$emit().

3.$emit和$on + 空的vue实例实现任意平级,跨级,父子组件通信:

通过一个空的vue实例来作为事件中心,用它来触发($emit)和监听($on)事件,巧妙地实现了任意组件间的通信.

新建eventBus.js文件,暴露eventBus=new Vue();import导入eventBus,使用eventBus.$emit("eventname",data)触发事件,在另一个组件也导入eventBus,使用eventBus.$on("eventname",data=>{console.log(data)})注册事件.

4.Vuex:

Vuex实现了一个单向数据流，在全局拥有一个state存放数据，当组件要更改state中的数据时，必须通过mutation进行，mutation同时提供了订阅者模式供外部插件调用获取state数据的更新。而当所有异步操作(常见于调用后端接口异步获取更新数据)或批量的同步操作需要走action，但action也是无法直接修改state的，还是需要通过mutation来修改state的数据。最后，根据state的变化，渲染到视图上。

(1) dispatch：操作行为触发方法，是唯一能执行action的方法。

(2) actions：操作行为处理模块,由组件中的 $store.dispatch('action 名称',data)来触发。然后由commit()来触发mutation的调用 , 间接更新 state。负责处理Vue Components接收到的所有交互行为。包含同步/异步操作，

(3) commit：状态改变提交操作方法。对mutation进行提交，是唯一能执行mutation的方法。

(4) mutations：状态改变操作方法，由actions中的 commit('mutation 名称')来触发。是Vuex修改state的唯一推荐方法。该方法只能进行同步操作，且方法名只能全局唯一。

(5) state：页面状态管理容器对象。集中存储Vue components中data对象的零散数据，全局唯一，以进行统一的状态管理。

(6) getters：state对象读取方法。Vue Components通过该方法读取全局state对象。

5. provide/inject:

祖先组件中通过provider来提供变量，然后在子孙组件中通过inject来注入变量。provide / inject API 主要解决了跨级组件间的通信问题，不过它的使用场景，主要是子组件获取上级组件的状态，跨级组件间建立了一种主动提供与依赖注入的关系。provide 和 inject 绑定并不是可响应的。这是vue设计时候刻意为之的。

// A.vue

export default

{

provide: {

name: '浪里个浪'

}

}

// B.vue

Export default

{

inject: [

'name'

],

mounted () {

console.log(this.name); // 浪里个浪

}

}

实现provide 和 inject 绑定实现数据响应式,使用2.6最新API Vue.observable 优化响应式 provide(推荐):

//使用vue2.6最新API Vue.observable 优化响应式 provide

//父组件提供provide数据,使用observable实现响应式.

provide() {

this.theme = Vue.observable({

color: "blue"

});

return {

theme: this.theme

};

}

//子组件:

{{injections.theme.color}}

export default{

inject: {

theme: {

//函数式组件取值不一样

default: () => ({})

}

}

};

6. $parents/$child & ref:

ref：如果在普通的 DOM 元素上使用，引用指向的就是 DOM 元素；如果用在子组件上，引用就指向组件实例.

$parent / $children：访问父 / 子实例.

ref和$parent / $children这两种都是直接得到组件实例，使用后可以直接调用组件的方法或访问数据。

// 子组件:

export default{

data () {

return {

title: 'lalala'

}

}

}

// 父组件取值

<child ref="comA"></child>

export default {

mounted() {

const comA = this.$refs.comA;

console.log(comA.title); //取到子组件的值

comA.sayHello(); //取到子组件的方法

}

};

这两种方法均无法在跨级或兄弟间通信。

7.$attr,$listeners

// 父组件:

<child :arr="arr"></child>

//子组件

<sun v-bind ="$attrs"

console.log(this.$attrs.arr)

//孙组件

console.log(this.$attrs.arr)

$attrs 里存放的是父组件中绑定的非 Props 属性,$listeners里存放的是父组件中绑定的非原生事件,支持组件一级一级往下传递参数.

**vuex是什么？怎么使用？哪种功能场景使用它？**

vuex其实是集中的数据管理仓库，任何组件都可以存取仓库中的数据。vuex中有四个部分：state 、 mutations 、 actions 、getters:

state:

Vuex 使用单一状态树,即每个应用将仅仅包含一个store 实例，但单一状态树和模块化并不冲突。存放的数据状态，不可以直接修改里面的数据。

mutations:

mutations定义的方法动态修改Vuex 的 store 中的状态或数据。

getters:

类似vue的computed计算属性，主要用来过滤一些数据。

action:

actions可以理解为通过将mutations里面处理数据的方法变成可异步的处理数据的方法，简单的说就是异步操作数据。

modules:

项目特别复杂的时候，可以让每一个模块拥有自己的state、mutation、action、getters,使得结构非常清晰，方便管理。

**Vuex中actions和mutations的区别？**

mutations的更改是同步更改，用于用户执行直接数据更改，this.$store.commit(‘名’)触发。

actions的更改是异步操作，用于需要与后端交互的数据更改,this.$store.dispath(“名”)触发。

**vue.js的两个核心是什么？**

答：数据驱动、组件系统

**vue常用的修饰符？**

答：.prevent: 提交事件不再重载页面；

.stop: 阻止单击事件冒泡；

.self: 当事件发生在该元素本身而不是子元素的时候会触发；

.capture: 事件侦听，事件发生的时候会调用.

**v-on 可以绑定多个方法吗？**

答：可以,对象,事件作为键,执行函数作为值.例如:

v-on='{click:DoSomething,mouseleave:MouseLeave}'

**vue中 key 值的作用？**

答：当 Vue.js 用 v-for 正在更新已渲染过的元素列表时，它默认用“就地复用”策略。如果数据项的顺序被改变，Vue 将不会移动 DOM 元素来匹配数据项的顺序， 而是简单复用此处每个元素，并且确保它在特定索引下显示已被渲染过的每个元素。key的作用主要是为了高效的更新虚拟DOM。

**vue等单页面应用及其优缺点**

答：优点：Vue核心是一个响应的数据绑定系统。MVVM、数据驱动、组件化、轻量、简洁、高效。

缺点：不支持低版本的浏览器，最低只支持到IE9；不利于SEO的优化（如果要支持SEO，建议通过服务端来进行渲染组件）；第一次加载首页耗时相对长一些(首屏加载耗时).

**怎么实现首屏加载速度优化?**

将各模块拆分为组件粒度.

将组件依赖的资源全部封装在组件内部进行调用.

组件化的拆分，确保模块之间不会互相影响和产生耦合之后，我们可以方面地调整加载策略。加载的策略是根据可见性来处理优先级问题。

优先加载首屏可见模块.

其余不可见模块懒加载，待可见或即将可见时加载

将公用的JS库通过script标签外部引入，减小app.bundel的大小，让浏览器并行下载资源文件，提高下载速度；

在配置 路由时，页面和组件使用懒加载的方式引入，进一步缩小 app.bundel 的体积，在调用某个组件时再加载对应的js文件；

使用骨架屏. 为真实的组件做一个在尺寸、样式上非常接近真实组件的组件，叫做骨架屏。

加一个首屏 loading 图，提升用户体验；