Vue3开发与实战

作者：kerwin

版本：QF1.0

公众号: 大前端私房菜

> **千锋精品教程，好学得不像实力派！**

十一.TS与组合式API

1. 为组件的 props 标注类型

```
<script setup lang="ts">
const props = defineProps<{
  foo: string
  bar?: number
  book: Book
}>()
</script>
```

2. 为组件的 emits 标注类型

```
<script setup lang="ts">
// 运行时
const emit = defineEmits(['change', 'update'])

// 基于类型
const emit = defineEmits<{
  (e: 'change', id: number): void
  (e: 'update', value: string): void
}>()
</script>
```

3. 为 `ref()` 标注类型

```
const year: Ref<string | number> = ref('2020')
const year = ref<string | number>('2020')
```

4. 为 `reactive()` 标注类型

```
import { reactive } from 'vue'

interface Book {
  title: string
  year?: number
}

const book: Book = reactive({ title: 'Vue 3 指引' })
```

5. 为 `computed()` 标注类型

```
const double = computed<number>(() => {
  // 若返回值不是 number 类型则会报错
})
```

十二. Vue面试-内功心法

1. 解释单向数据流和双向数据绑定

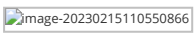
对于 Vue 来说，组件之间的数据传递具有单向数据流这样的特性称为单向数据流，单向数据流（Unidirectional data flow）方式使用一个上传数据流和一个下传数据流进行双向数据通信，两个数据流之间相互独立，单向数据流指只能从一个方向来修改状态。

而双向数据绑定即为当数据发生变化的时候，视图也就发生变化，当视图发生变化的时候，数据也会跟着同步变化，两个数据流之间互为影响。

2. Object.defineProperty有什么缺点

1、无法监听es6的Set、Map 变化;
2、无法监听Class类型的数据;
3、属性的新加或者删除也无法监听;
4、数组元素的增加和删除也无法监听。

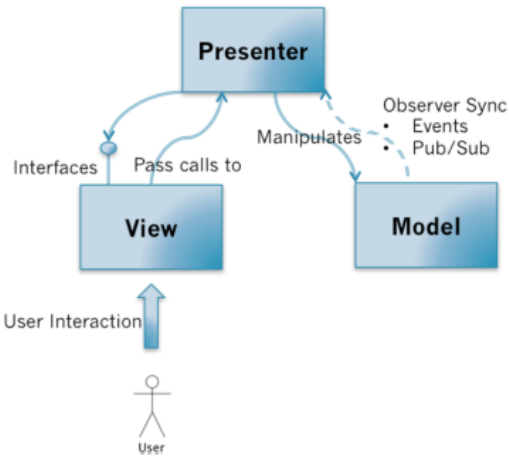
3. 对MVC，MVP，MVVM的理解

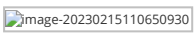


和MVC模式一样，用户对View的操作都会从View交移给Presenter。Presenter会执行相应的应用程序逻辑，并且对Model进行相应的操作；而这时候Model执行完业务逻辑以后，也是通过观察者模式把自己变更的消息传递出去，但是是传给Presenter而不是View。Presenter获取到Model变更的消息以后，**通过View提供的接口更新界面。**

View不依赖Model，View可以进行组件化。但Model->View的手动同步逻辑　麻烦，维护困难

MVP
ibm子公司提出





4. 生命周期

5. 你知道Vue响应式数据原理吗？Proxy 与 Object.defineProperty 优劣对比?

// 响应式原理

vue的响应式实现主要是利用了Object.defineProperty的方法里面的setter 与getter方法的观察者模式来实现。在组件初始化时会给每一个data属性注册getter和setter，然后再new 一个自己的Watcher对象，此时watcher会立即调用组件的render函数去生成虚拟DOM。在调用render的时候，就会需要用到data的属性值，此时会触发getter函数，将当前的Watcher函数注册进sub里。当data属性发生改变之后，就会遍历sub里所有的watcher对象，通知它们去重新渲染组件。

// proxy的优势如下:

Proxy 可以直接监听对象而非属性，可以直接监听数组的变化;
Proxy 有多达 13 种拦截方法,不限于 apply、ownKeys、deleteProperty、has 等等是 Object.defineProperty 不具备的;
Proxy 返回的是一个新对象,我们可以只操作新的对象达到目的,而 Object.defineProperty 只能遍历对象属性直接修改;

// Object.defineProperty 的优势如下:

兼容性好，支持 IE9，而 Proxy 的存在浏览器兼容性问题,而且无法用 polyfill(垫片)来弥补

6. Composition API 的出现带来哪些新的开发体验，为啥需要这个?

1：在Compostion API 中时根据逻辑相关组织代码的，提高可读性和可维护性，类似于react的hook写法。
2：更好的重用逻辑代码，在Options API中通过MIxins重用逻辑代码，容易发生命名冲突且关系不清。
3：解决在生命周期函数经常包含不相关的逻辑，但又不得不把相关逻辑分离到了几个不同方法中的问题，如在mounted中设置定时器，但需要在destroyed中来清除定时器，将同一功能的代码拆分到不同的位置，造成后期代码维护的困难。

7. 对比 jQuery，Vue 有什么不同

jQuery 专注视图层，通过直接操作 DOM 去实现页面的一些逻辑渲染；Vue 专注于数据层，通过数据的双向绑定，最终表现在 DOM 层面，减少了 DOM 操作。Vue 使用了组件化思想，使得项目子集职责清晰，提高了开发效率，方便重复利用，便于协同开发

8. 如何再Vue的单文件组件里的样式定义全局CSS?

在style标签上不加上scoped的属性，默认为全局css样式

9. 说一下$root$，parent，$refs

$root，和$parent都能访问父组件的属性和方法，区别在于如果存在多级子组件，通过parent 访问得到的是它最近一级的父组件，通过root 访问得到的是根父组件。通过在子组件标签定义 ref 属性，在父组件中可以使用$refs 访问子组件实例。

10. Vue 中怎么自定义指令

通过directive来自定义指令，自定义指令分为全局指令和局部指令，自定义指令也有几个的钩子函数，常用的有bind和

update，当 bind 和 update 时触发相同行为，而不关心其它的钩子时可以简写。

```
Vue.directive('focus', {
  // 当被绑定的元素插入到 DOM 中时……
  inserted: function (el) {
    // 聚焦元素
    el.focus()
  }
})


Vue.directive('color-swatch', function (el, binding) {
  el.style.backgroundColor = binding.value
})
```

11. Vue 中怎么自定义过滤器 (vue3不支持)

通过filter来定义过滤器，过滤器分为全局和局部过滤器，过滤器的主体为一个普通的函数，来对数据进行处理，可以传递参数。当有局部和全局两个名称相同的过滤器时候，会以就近原则进行调用，即：局部过滤器优先于全局过滤器被调用。

```
<!-- 在双花括号中 -->
{{ message | capitalize }}

<!-- 在 `v-bind` 中 -->
<div v-bind:id="rawId | formatId"></div>


filters: {
  capitalize: function (value) {
    if (!value) return ''
    value = value.toString()
    return value.charAt(0).toUpperCase() + value.slice(1)
  }
}

Vue.filter('capitalize', function (value) {
  if (!value) return ''
  value = value.toString()
  return value.charAt(0).toUpperCase() + value.slice(1)
})
```

12. Vue 等单页面应用的优缺点

```
// 优点
1单页应用的内容的改变不需要重新加载整个页面，web应用更具响应性和更令人着迷。

2、单页应用没有页面之间的切换，就不会出现“白屏现象”,也不会出现假死并有“闪烁”现象


3、单页应用相对服务器压力小，服务器只用出数据就可以，不用管展示逻辑和页面合成，吞吐能力会提高几倍。
```

4、良好的前后端分离。后端不再负责模板渲染、输出页面工作，后端API通用化，即同一套后端程序代码，不用修改就可以用于Web界面、手机、平板等多种客户端。

// 缺点
1、首次加载耗时比较多。

2、SEO问题，不利于百度，360等搜索引擎收录。

3、容易造成Css命名冲突。

4、前进、后退、地址栏、书签等，都需要程序进行管理，页面的复杂度很高，需要一定的技能水平和开发成本高。

13. Vue-router 使用params与query传参有什么区别

// 用法上
1：query要用path来引入，params要用name来引入，接收参数都是类似的，分别是this.$route.query和this.$route.params。

// 展示上

2：params是路由的一部分,必须要有。query是拼接在url后面的参数

```
// 命名的路由，并加上参数，让路由建立 url /users/eduardo
router.push({ name: 'user', params: { username: 'eduardo' } })

// 带查询参数，结果是 /register?plan=private
router.push({ path: '/register', query: { plan: 'private' } })

// 带 hash，结果是 /about#team
router.push({ path: '/about', hash: '#team' })
```

14. Vue中 keep-alive 的作用

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。一旦使用keepalive包裹组件，此时mouted，created等钩子函数只会在第一次进入组件时调用，当再次切换回来时将不会调用。此时如果我们还想在每次切换时做一些事情，就需要用到另外的周期函数，actived和deactived，这两个钩子函数只有被keepalive包裹后才会调用。

15. Vue如何实现单页面应用

通常的url 地址由以下内容构成：协议名 域名 端口号 路径 参数 哈希值，当哈希值改变，页面不会发生跳转，单页面应用就是利用了这一点，给window注册onhashchange事件，当哈希值改变时通过location.hash就能获得相应的哈希值，然后就能跳到相应的页面。

1.hash通过监听浏览器的onhashchange()事件变化，查找对应的路由规则

2.history原理： 利用H5的 history中新增的两个API pushState() 和 replaceState() 和一个事件onpopstate监听URL变化

16. 说出至少4种Vue当中的指令和它的用法?

v-if(判断是否隐藏，用来判断元素是否创建)
v-show(元素的显示隐藏，类似css中的display的block和hidden)
v-for(把数据遍历出来)
v-bind(绑定属性)
v-model(实现双向绑定)

17. 如何实现一个路径渲染多个组件?

可以通过命名视图(router-view)，它容许同一界面中拥有多个单独命名的视图，而不是只有一个单独的出口。如果 router-view 没有设置名字，那么默认为 default。通过设置components即可同时渲染多个组件。

```
<router-view class="view left-sidebar" name="LeftSidebar"></router-view>
<router-view class="view main-content"></router-view>
<router-view class="view right-sidebar" name="RightSidebar"></router-view>


const router = createRouter({
  history: createWebHashHistory(),
  routes: [
    {
      path: '/',
      components: {
        default: Home,
        // LeftSidebar: LeftSidebar 的缩写
        LeftSidebar,
        // 它们与 `<router-view>` 上的 `name` 属性匹配
        RightSidebar,
      },
    },
  ],
})
```

18. 如何实现多个路径共享一个组件?

只需将多个路径的component字段的值设置为同一个组件即可。

```
const routes = [
  { path: '/', component: Home },
  { path: '/home', component: Home },
]
```

19. 如何监测动态路由的变化

可以通过watch方法来对$route进行监听，或者通过导航守卫的钩子函数beforeRouteUpdate来监听它的变化。

20. vue-router 中的 router-link上 v-slot属性怎么用?

router-link 通过一个作用域插槽暴露底层的定制能力。这是一个更高阶的 API，主要面向库作者，但也可以为开发者提供便

利，多数情况用在一个类似 NavLink 这样的自定义组件里。

有时我们可能想把激活的 class 应用到一个外部元素而不是 <a> 标签本身，这时你可以在一个 router-link 中包裹该元素并使用 v-slot 属性来创建链接：

```
<router-link
  to="/foo"
  custom
  v-slot="{ href, route, navigate, isActive, isExactActive }"
>
  <li
    :class="[isActive && 'router-link-active', isExactActive && 'router-link-exact-active']"
  >
    <a :href="href" @click="navigate">{{ route.fullPath }}</a>
  </li>
</router-link>
```

21. Vue 如何去除url中的

将路由模式改为history

由于我们的应用是一个单页的客户端应用，如果没有适当的服务器配置，用户在浏览器中直接访问 https://example.com/user/id，就会得到一个 404 错误。这就尴尬了。

不用担心：要解决这个问题，你需要做的就是在你的服务器上添加一个简单的回退路由。如果 URL 不匹配任何静态资源，它应提供与你的应用程序中的 index.html 相同的页面。

```
var history = require('connect-history-api-fallback');
app.use(history({
    index: '/index.html'
})); //注意放在所有的接口后面
```

22. $route$和router 的区别

$route用来获取路由的信息的，它是路由信息的一个对象，里面包含路由的一些基本信息，包括name、meta、path、hash、query、params、fullPath、matched、redirectedFrom等。而$router主要是用来操作路由的，它是VueRouter的实例，包含了一些路由的跳转方法push，go, replace，钩子函数等

23. Vue 路由守卫

vue-router 提供的导航守卫主要用来对路由的跳转进行监控，控制它的跳转或取消，路由守卫有全局的, 单个路由独享的, 或者组件级的。导航钩子有3个参数：

1、to:即将要进入的目标路由对象；
2、from:当前导航即将要离开的路由对象；
3、next：调用该方法后，才能进入下一个钩子函数（afterEach）。

```
router.beforeEach(async (to, from) => {
  if (
    // 检查用户是否已登录
```

```
    !isAuthenticated &&
    // ❗ 避免无限重定向
    to.name !== 'Login'
  ) {
    // 将用户重定向到登录页面
    return { name: 'Login' }
  }
})
```

24. Vue路由实现的底层原理

在Vue中利用数据劫持defineProperty在原型prototype上初始化了一些getter,分别是router代表当前Router的实例 、route 代表当前Router的信息。在install中也全局注册了router-view,router-link,其中的Vue.util.defineReactive, 这是Vue里面观察者劫持数据的方法，劫持_route，当_route触发setter方法的时候，则会通知到依赖的组件。

接下来在init中，会挂载判断是路由的模式，是history或者是hash,点击行为按钮，调用hashchange或者popstate的同时更_route,_route的更新会触发route-view的重新渲染。

25. 路由懒加载

Vue Router 支持开箱即用的动态导入，这意味着你可以用动态导入代替静态导入：

```
// 将
// import UserDetails from './views/UserDetails.vue'
// 替换成
const UserDetails = () => import('./views/UserDetails.vue')

const router = createRouter({
  // ...
  routes: [{ path: '/users/:id', component: UserDetails }],
})
```

26. 用过插槽吗？用的是具名插槽还是匿名插槽

用过，都使用过。插槽相当于预留了一个位置，可以将我们书写在组件内的内容放入，写一个插槽就会将组件内的内容替换一次，两次则替换两次。为了自定义插槽的位置我们可以给插槽取名，它会根据插槽名来插入内容，一一对应。

举例来说，这里有一个 <FancyButton> 组件，可以像这样使用：
template
<FancyButton>
  Click me! <!-- 插槽内容 -->
</FancyButton>

而 <FancyButton> 的模板是这样的：

template

```
<button class="fancy-btn">
  <slot></slot> <!-- 插槽出口 -->
</button>
```

27. Vue-loader解释一下

解析和转换 .vue 文件，提取出其中的逻辑代码 script、样式代码 style、以及 HTML 模版 template，再分别把它们交给对应的 Loader 去处理。

28. Vue和React中diff算法区别

vue和react的diff算法，都是忽略跨级比较，只做同级比较。vue diff时调动patch函数，参数是vnode和oldVnode，分别代表新旧节点。

1.vue对比节点。当节点元素相同，但是classname不同，认为是不同类型的元素，删除重建，而react认为是同类型节点，只是修改节点属性。

2.vue的列表对比，采用的是两端到中间比对的方式，而react采用的是从左到右依次对比的方式。当一个集合只是把最后一个节点移到了第一个，react会把前面的节点依次移动，而vue只会把最后一个节点移到第一个。总体上，vue的方式比较高效。

29. 请你说一下 Vue 中 create 和 mount 的区别

create为组件初始化阶段，在此阶段主要完成数据观测(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，此时还未生成真实的DOM，也就无法获取和操作DOM元素。而mount主要完成从虚拟DOM到真实DOM的转换挂载，此时html已经渲染出来了，所以可以直接操作dom节点。

30. axios是什么？怎么使用？描述使用它实现登录功能的流程?

axios 是请求后台资源的模块。 通过npm install axios -S来安装，在大多数情况下我们需要封装拦截器，在实现登录的过程中我们一般在请求拦截器中来加入token，在响应请求器中通过判断后端返回的状态码来对返回的数据进行不同的处理。如果发送的是跨域请求，需在配置文件中 config/index.js 进行代理配置。

```
// Add a request interceptor
axios.interceptors.request.use(function (config) {
    // Do something before request is sent
    return config;
  }, function (error) {
    // Do something with request error
    return Promise.reject(error);
  });

// Add a response interceptor
axios.interceptors.response.use(function (response) {
    // Any status code that lie within the range of 2xx cause this function to trigger
    // Do something with response data
    return response;
  }, function (error) {
```

```
  // Any status codes that falls outside the range of 2xx cause this function to trigger
  // Do something with response error
  return Promise.reject(error);
 });
```

31. computed和watch的区别？watch实现原理？watch有几种写法？

计算属性computed :

1. 支持缓存，只有依赖数据发生改变，才会重新进行计算
2. 不支持异步，当computed内有异步操作时无效，无法监听数据的变化
3.computed 属性值会默认走缓存，计算属性是基于它们的响应式依赖进行缓存的，也就是基于data中声明过或者父组件传递的props中的数据通过计算得到的值
4. 如果一个属性是由其他属性计算而来的，这个属性依赖其他属性，是一个多对一或者一对一，一般用computed
5.如果computed属性属性值是函数，那么默认会走get方法；函数的返回值就是属性的属性值；在computed中的，属性都有一个get和一个set方法，当数据变化时，调用set方法。

```
computed: {
  // 一个计算属性的 getter
  publishedBooksMessage() {
   // `this` 指向当前组件实例
   return this.author.books.length > 0 ? 'Yes' : 'No'
  }
 }
```

侦听属性watch：

1. 不支持缓存，数据变，直接会触发相应的操作；
2.watch支持异步；
3.监听的函数接收两个参数，第一个参数是最新的值；第二个参数是输入之前的值；
4. 当一个属性发生变化时，需要执行对应的操作；一对多；
5. 监听数据必须是data中声明过或者父组件传递过来的props中的数据，当数据变化时，触发其他操作，函数有两个参数，
immediate：组件加载立即触发回调函数执行，
deep: 深度监听，为了发现对象内部值的变化，复杂类型的数据时使用，例如数组中的对象内容的改变，注意监听数组的变动不需要这么做。

```
watch: {
  // 每当 question 改变时，这个函数就会执行
  question(newQuestion, oldQuestion) {
   if (newQuestion.includes('?')) {
    this.getAnswer()
   }
  }
 },
watch: {
  someObject: {
   handler(newValue, oldValue) {
    // 注意：在嵌套的变更中，

    // 只要没有替换对象本身，
```

```
      // 那么这里的 `newValue` 和 `oldValue` 相同
    },
    deep: true
  }
}
```

32. Vue $forceUpdate的原理

1、作用：

迫使 `Vue` 实例重新渲染。注意它仅仅影响实例本身和插入插槽内容的子组件，而不是所有子组件。

2、内部原理：

```
Vue.prototype.$forceUpdate = function () {
  const vm: Component = this
  if (vm._watcher) {
    vm._watcher.update()
  }
}
```

实例需要重新渲染是在依赖发生变化的时候会通知watcher，然后通知watcher来调用update方法，就是这么简单。

33. v-for key

- key是为Vue中的vnode标记的唯一id,通过这个key,我们的diff操作可以更准确、更快速
- diff算法的过程中,先会进行新旧节点的首尾交叉对比,当无法匹配的时候会用新节点的key与旧节点进行比对,然后超出差异.

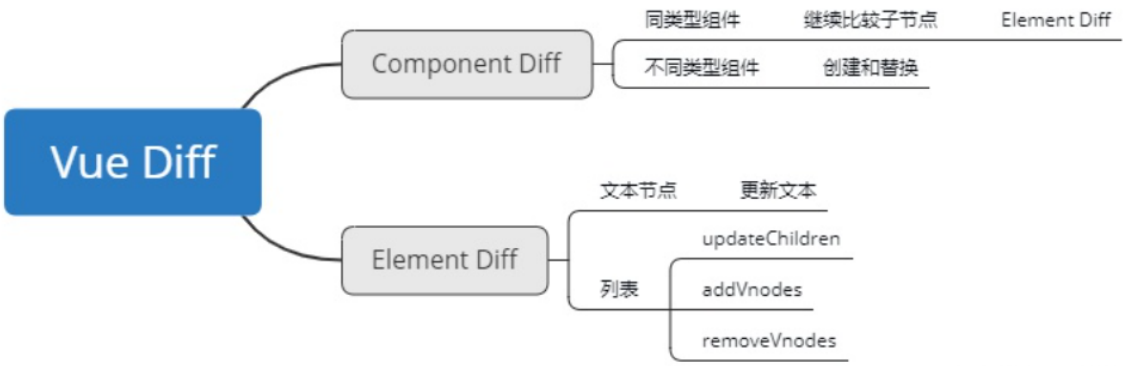
> diff程可以概括为：oldCh和newCh各有两个头尾的变量StartIdx和EndIdx，它们的2个变量相互比较，一共有4种比较方式。如果4种比较都没匹配，如果设置了key，就会用key进行比较，在比较的过程中，变量会往中间靠，一旦StartIdx>EndIdx表明oldCh和newCh至少有一个已经遍历完了，就会结束比较,这四种比较方式就是首、尾、旧尾新头、旧头新尾.
>
> 准确: 如果不加key,那么vue会选择复用节点(Vue的就地更新策略),导致之前节点的状态被保留下来,会产生一系列的bug. 快速: key的唯一性可以被Map数据结构充分利用,相比于遍历查找的时间复杂度 `O(n)` , `Map` 的时间复杂度仅仅为 `O(1)`

34. 为什么要设置key值，可以用index吗？为什么不能?

vue中列表循环需加:key="唯一标识" 唯一标识可以是item里面id index等，因为vue组件高度复用增加Key可以标识组件的唯一性，为了更好地区别各个组件 key的作用主要是为了高效的更新虚拟DOM

35. diff复杂度原理及具体过程画图

diff算法是一种通过同层的树节点进行比较的高效算法，避免了对树进行逐层搜索遍历，所以时间复杂度只有O(n)。



diff算法有两个比较显著的特点：

1、比较只会在同层级进行, 不会跨层级比较。

2、在diff比较的过程中，循环从两边向中间收拢。

diff流程： 首先定义 oldStartIdx、newStartIdx、oldEndIdx 以及 newEndIdx 分别是新老两个 VNode 的两边的索引。

接下来是一个 while 循环，在这过程中，oldStartIdx、newStartIdx、oldEndIdx 以及 newEndIdx 会逐渐向中间靠拢。while 循环的退出条件是直到老节点或者新节点的开始位置大于结束位置。

while 循环中会遇到四种情况：

情形一：当新老 VNode 节点的 start 是同一节点时，直接 patchVnode 即可，同时新老 VNode 节点的开始索引都加 1。

情形二：当新老 VNode 节点的 end 是同一节点时，直接 patchVnode 即可，同时新老 VNode 节点的结束索引都减 1。

情形三：当老 VNode 节点的 start 和新 VNode 节点的 end 是同一节点时，这说明这次数据更新后 oldStartVnode 已经跑到了 oldEndVnode 后面去了。这时候在 patchVnode 后，还需要将当前真实 dom 节点移动到 oldEndVnode 的后面，同时老 VNode 节点开始索引加 1，新 VNode 节点的结束索引减 1。

情形四：当老 VNode 节点的 end 和新 VNode 节点的 start 是同一节点时，这说明这次数据更新后 oldEndVnode 跑到了 oldStartVnode 的前面去了。这时候在 patchVnode 后，还需要将当前真实 dom 节点移动到 oldStartVnode 的前面，同时老 VNode 节点结束索引减 1，新 VNode 节点的开始索引加 1。

while 循环的退出条件是直到老节点或者新节点的开始位置大于结束位置。

情形一：如果在循环中，oldStartIdx大于oldEndIdx了，那就表示oldChildren比newChildren先循环完毕，那么newChildren里面剩余的节点都是需要新增的节点，把[newStartIdx, newEndIdx]之间的所有节点都插入到DOM中

情形二：如果在循环中，newStartIdx大于newEndIdx了，那就表示newChildren比oldChildren先循环完毕，那么oldChildren里面剩余的节点都是需要删除的节点，把[oldStartIdx, oldEndIdx]之间的所有节点都删除

36. Vue组件中的Data为什么是函数，根组件却是对象呢?

综上可知，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。

37. Vue的组件通信

1、props和$emit

父组件向子组件传递数据是通过prop传递的，子组件传递数据给父组件是通过$emit触发事件

2、$attrs$和 listeners

3、中央事件总线 bus

上面两种方式处理的都是父子组件之间的数据传递，而如果两个组件不是父子关系呢？这种情况下可以使用中央事件总线的方式。新建一个Vue事件bus对象，然后通过bus.$emit$触发事件，bus.on监听触发的事件。

4、provide和inject

父组件中通过provider来提供变量，然后在子组件中通过inject来注入变量。不论子组件有多深，只要调用了inject那么就可以注入provider中的数据。而不是局限于只能从当前父组件的prop属性来获取数据，只要在父组件的生命周期内，子组件都可以调用。

5、v-model

父组件通过v-model传递值给子组件时，会自动传递一个value的prop属性，在子组件中通过this.$emit('input',val)自动修改v-model绑定的值

6、$parent$和 children

7、boradcast和dispatch

8、vuex处理组件之间的数据交互 如果业务逻辑复杂，很多组件之间需要同时处理一些公共的数据，这个时候才有上面这一些方法可能不利于项目的维护，vuex的做法就是将这一些公共的数据抽离出来，然后其他组件就可以对这个公共数据进行读写操作，这样达到了解耦的目的。

38. 什么情况下使用 Vuex

> 如果应用够简单，最好不要使用 Vuex，一个简单的 store 模式即可，需要构建一个中大型单页应用时，使用Vuex能更好地在组件外部管理状态

39. Vuex可以直接修改state的值吗?

> 可以直接修改，但是极其不推荐，state的修改必须在mutation来修改，否则无法被devtool所监测，无法监测数据的来源，无法保存状态快照，也就无法实现时间漫游/回滚之类的操作。

40. 为什么Vuex的mutation不能做异步操作

> Vuex中所有的状态更新的唯一途径都是mutation，异步操作通过 Action 来提交 mutation实现，这样使得我们可以方便地跟踪每一个状态的变化，从而让我们能够实现一些工具帮助我们更好地了解我们的应用。每个mutation执行完成后都会对应到一个新的状态变更，这样devtools就可以打个快照存下来，否则无法被devtools所监测。如果mutation支持异步操作，就没有办法知道状态是何时更新的，无法很好的进行状态的追踪，给调试带来困难。

41. 怎么修改Vuex中的状态？Vuex中有哪些方法

- 通过**this.$store.state.属性** 的方法来访问状态
- 通过**this.$store.commit('mutation中的方法')** 来修改状态

42. Vuex的缺点

> 如果您不打算开发大型单页应用，使用 Vuex 可能是繁琐冗余的，并且state中的值会伴随着浏览器的刷新而初始化，无缓存。

43. 什么是 Vue.nextTick()?

1、$nextTick 是在下次DOM更新循环结束之后执行延迟回调。在修改数据之后立即使用这个方法，获取更新后的DOM，意思是 等你dom加载完毕以后再去调用nextTick()里面的数据内容

44. nextTick知道吗、实现的原理是什么？是宏任务还是微任务?

微任务

原理：

nextTick方法主要是使用了宏任务和微任务，定义了一个异步方法，多次调用nextTick会将方法存入队列中，通过这个异步方法清空队列。

作用： nextTick用于下次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用于下次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用于下次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。

45. 虚拟 dom 为什么会提高性能?

虚拟DOM其实就是一个JavaScript对象。通过这个JavaScript对象来描述真实DOM，真实DOM的操作，一般都会对某块元素的整体重新渲染，采用虚拟DOM的话，当数据变化的时候，只需要局部刷新变化的位置就好了，

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

46. 你做过哪些Vue的性能优化?

1、首屏加载优化

2、路由懒加载

```
{
  path: '/',
  name: 'home',
  component: () => import('./views/home/index.vue'),
  meta: { isShowHead: true }
}
```

3、开启服务器 Gzip

开启 Gzip 就是一种压缩技术，需要前端提供压缩包，然后在服务器开启压缩，文件在服务器压缩后传给浏览器，浏览器解压后进行再进行解析。首先安装 webpack 提供的`compression-webpack-plugin`进行压缩,然后在 vue.config.js：

```
const CompressionWebpackPlugin = require('compression-webpack-plugin')
const productionGzipExtensions = ['js', 'css'].....plugins: [
  new CompressionWebpackPlugin(
    {
      algorithm: 'gzip',
      test:     new RegExp('\\.(' + productionGzipExtensions.join('|') +                ')$'),
      threshold: 10240,
      minRatio: 0.8
       }
)]....
```

4、启动 CDN 加速

我们继续采用 cdn 的方式来引入一些第三方资源，就可以缓解我们服务器的压力，原理是将我们的压力分给其他服务器点。

5、代码层面优化

- computed 和 watch 区分使用场景

  computed： 是计算属性，依赖其它属性值，并且 computed 的值有缓存，只有它依赖的属性值发生改变，下一次获取 computed 的值时才会重新计算 computed 的值。当我们需要进行数值计算，并且依赖于其它数据时，应该使用 computed，因为可以利用 computed 的缓存特性，避免每次获取值时，都要重新计算；
  watch：类似于某些数据的监听回调 ，每当监听的数据变化时都会执行回调进行后续操作；当我们需要在数据变化时执行异步或开销较大的操作时，应该使用 watch，使用 watch 选项允许我们执行异步操作 (访问一个 API)，限制我们执行该操作的频率，并在我们得到最终结果前，设置中间状态。这些都是计算属性无法做到的。

- v-if 和 v-show 区分使用场景 v-if 适用于在运行时很少改变条件，不需要频繁切换条件的场景；v-show 则适用于需要非常频繁切换条件的场景。这里要说的优化点在于减少页面中 dom 总数，我比较倾向于使用 v-if，因为减少了 dom 数量。
- v-for 遍历必须为 item 添加 key，且避免同时使用 v-if v-for 遍历必须为 item 添加 key，循环调用子组件时添加 key，key 可以唯一标识一个循环个体，可以使用例如 item.id 作为 key 避免同时使用 v-if，v-for 比 v-if 优先级高，如果每一次都需要遍历整个数组，将会影响速度。

6、Webpack 对图片进行压缩

7、避免内存泄漏

8、减少 ES6 转为 ES5 的冗余代码

47. Vue的常用修饰符

一、v-model修饰符

1、.lazy：

输入框改变，这个数据就会改变，lazy这个修饰符会在光标离开input框才会更新数据：

```
<input type="text" v-model.lazy="value">
```

2、.trim：

输入框过滤首尾的空格：

```
<input type="text" v-model.trim="value">
```

3、.number：

先输入数字就会限制输入只能是数字，先字符串就相当于没有加number，注意，不是输入框不能输入字符串，是这个数据是数字：

```
<input type="text" v-model.number="value">
```

二、事件修饰符

4、.stop：

阻止事件冒泡，相当于调用了event.stopPropagation()方法：

```
<button @click.stop="test">test</button>
```

5、.prevent：

阻止默认行为，相当于调用了event.preventDefault()方法，比如表单的提交、a标签的跳转就是默认事件：

```
<a @click.prevent="test">test</a>
```

6、.self：

只有元素本身触发时才触发方法，就是只有点击元素本身才会触发。比如一个div里面有个按钮，div和按钮都有事件，我们点击按钮，div绑定的方法也会触发，如果div的click加上self，只有点击到div的时候才会触发，变相的算是阻止冒泡：

```
<div @click.self="test"></div>
```

7、.once：

事件只能用一次，无论点击几次，执行一次之后都不会再执行

```
<div @click.once="test"></div>
```

8、.capture：

事件的完整机制是捕获-目标-冒泡，事件触发是目标往外冒泡

9、.sync

对prop进行双向绑定

10、.keyCode：

监听按键的指令，具体可以查看vue的键码对应表

48. Vue 中 template 的编译过程

vue template模板编译的过程经过parse()生成ast(抽象语法树),optimize对静态节点优化，generate()生成render字符串 之后调用new Watcher()函数，用来监听数据的变化，render 函数就是数据监听的回调所调用的，其结果便是重新生成 vnode。 当这个 render 函数字符串在第一次 mount、或者绑定的数据更新的时候，都会被调用，生成 Vnode。 如果是数据的更新，那么 Vnode 会与数据改变之前的 Vnode 做 diff，对内容做改动之后，就会更新到 我们真正的 DOM

49. 谈谈你对Vue3.0有什么了解?

六大亮点

1. 性能比vue2.x快1.2~2倍
2. 支持tree-shaking，按需编译，体积比vue2.x更小
3. 支持组合API
4. 更好的支持TS
5. 更先进的组件

性能比vue2.x快1.2~2倍如何实现的呢

1.diff算法更快

vue2.0是需要全局去比较每个节点的，若发现有节点发生变化后，就去更新该节点

vue3.0是在创建虚拟dom中，会根据DOM的的内容会不会发生内容变化，添加静态标记， 谁有flag！比较谁。

2、静态提升

vue2中无论元素是否参与更新，每次都会重新创建，然后再渲染 vue3中对于不参与更新的元素，会做静态提升，只被创建一次，在渲染时直接复用即可

3、事件侦听缓存

默认情况下，onclick为动态绑定，所以每次都会追踪它的变化，但是因为是同一函数，没有必要追踪变化，直接缓存复用即可

在之前会添加静态标记8 会把点击事件当做动态属性 会进行diff算法比较， 但是在事件监听缓存之后就没有静态标记了，就会进行缓存复用

为什么vue3.0体积比vue2.x小

在vue3.0中创建vue项目 除了vue-cli，webpack外还有 一种创建方法是Vite Vite是作者开发的一款有意取代webpack的工具，其实现原理是利用ES6的import会发送请求去加载文件的特性，拦截这些请求，做一些预编译，省去webpack冗长的打包时间

50. vue3.0组合API

说一说vue3.0的组合API跟之前vue2.0在完成业务逻辑上的区别：

在vue2.0中： 主要是往data 和method里面添加内容，一个业务逻辑需要什么data和method就往里面添加，而组合API就是 有一个自己的方法，里面有自己专注的data 和method。

再说一下组合API的本质是什么： 首先composition API（组合API） 和 Option API（vue2.0中的data和method）可以共用 composition API（组合API）本质就是把内容添加到Option API中进行使用

51. ref和reactive的简单理解

1.ref和reactive都是vue3的监听数据的方法，本质是proxy 2.ref 基本类型复杂类型都可以监听(我们一般用ref监听基本类型)，reactive只能监听对象（arr，json） 3.ref底层还是reactive，ref是对reactive的二次包装， ref定义的数据访问的时候要多一个.value

52. Vuex和redux有什么区别？他们的共同思想。

Redux和Vuex区别

- Vuex改进了Redux中的Action和Reducer函数，以mutations变化函数取代Reducer，无需switch，只需在对应的mutation函数里改变state值就可以
- Vuex由于Vue自动重新渲染的特性，无需订阅重新渲染函数，只要生成新的state就可以
- Vuex数据流的顺序是:View调用store.commit提交对应的请求到Store中对应的mutation函数 -- store改变(vue检测到数据变化自动渲染)

共同思想

- 单一的数据源
- 变化可以预测
- `本质上:` Redux和Vuex都是对MVVM思想的服务，将数据从视图中抽离的一种方案
- `形式上:` Vuex借鉴了Redux，将store作为全局的数据中心，进行数据管理

53. 简单说一下 微信小程序 与 Vue 的区别

1、生命周期：

`小程序` 的钩子函数要简单得多 。 `vue` 的钩子函数在跳转新页面时，钩子函数都会触发，但是 `小程序` 的钩子函数，页面不同的跳转方式，触发的钩子并不一样。

在页面加载请求数据时，两者钩子的使用有些类似，vue一般会在created或者mounted中请求数据，而在小程序，会在onLoad或者onShow中请求数据。

2、数据绑定：

vue动态绑定一个变量的值为元素的某个属性的时候，会在变量前面加上冒号：

```
<img :src="imgSrc"/>
```

小程序 绑定某个变量的值为元素属性时，会用两个大括号括起来，如果不加括号，为被认为是字符串

```
<image src="{{imgSrc}}"></image>
```

3、列表循环

4、显示与隐藏元素

vue中，使用v-if 和 v-show控制元素的显示和隐藏

小程序中，使用wx-if和hidden控制元素的显示和隐藏

5、事件处理

vue：使用v-on:event绑定事件，或者使用@event绑定事件

小程序中，全用bindtap(bind+event)，或者catchtap(catch+event)绑定事件

6、数据的双向绑定

在vue中,只需要再表单元素上加上v-model,然后再绑定data中对应的一个值，当表单元素内容发生变化时，data中对应的值也会相应改变 。

当表单内容发生变化时，会触发表单元素上绑定的方法，然后在该方法中，通过this.setData({key:value})来将表单上的值赋值给data中的对应值 。

7、绑定事件传参

在vue中，绑定事件传参挺简单，只需要在触发事件的方法中，把需要传递的数据作为形参传入就可以了

在小程序中，不能直接在绑定事件的方法中传入参数，需要将参数作为属性值，绑定到元素上的data-属性上，然后在方法中，通过e.currentTarget.dataset.*的方式获取

8、父子组件通信

父组件向子组件传递数据，只需要在子组件通过v-bind传入一个值，在子组件中，通过props接收，即可完成数据的传递

父组件向子组件通信和vue类似，但是小程序没有通过v-bind，而是直接将值赋值给一个变量 在子组件properties中，接收传递的值