回到顶部

vue和react区别

共同点：都是单向数据流，使用虚拟dom，支持跨平台。 不同点：

1. 指令解析不同，vue绑定使用v-on，react绑定事件使用合成事件;

2. 监听方式不同，vue使用object.defineProperty劫持数据，react使用render直接比较虚拟dom的变化。shouldComponentUpdate深比较，pureComponent浅比较。

3. vue使用v-module进行表单的双向绑定，react使用表单控件以及onchange事件进行双向绑定。

4. 组件之间的通讯方式不同。vue子向父$emit，父向子：props，跨组件使用发布订阅模式 react子向父：callback，父向子：props，跨组件：react.createContext/发布订阅模式/redux

5. vue文档比较友好，容易上手，很多封装的api可以直接使用，环境比较集中，vue vue-router vueX。react社区环境比较大，redux mobx react-router-dom 比较灵活，我们可以自己封装

- 回到顶部

组件内data为什么必须是函数

回到顶部

组件中的data写成一个函数，数据以函数返回值形式定义，这样每复用一次组件，就会返回一份新的data。如果单纯的写成对象形式，就使得所有组件实例共用了一份data，造成了数据污染。工厂模式，防止组件状态data指向同一地址，每一个组件保持自己的状态。

- 回到顶部

vue生命周期

beforeCreate 组件实例被创建之初，组件的属性生效之前

created 组件实例已经完全创建，属性也绑定，但真实dom还没有生成，$el还不可用

beforeMount 在挂载开始之前被调用：相关的 render 函数首次被调用

mounted 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

beforeUpdate 组件数据更新之前调用，发生在虚拟 DOM 打补丁之前

update 组件数据更新之后

activited keep-alive专属，组件被激活时调用

deadctivated keep-alive专属，组件被销毁时调用

beforeDestory 组件销毁前调用

destoryed 组件销毁后调用

- 回到顶部

vue数据驱动视图原理

当数据改变时，视图同步更新，首先，要监听数据的变化，object.defineProperty()里面的get和set来监听数据变化，模板内容解析，将数据和视图关联起来，来进行数据渲染。mvvm解析模板指令，劫持数据，解析之后，设置观察者。

- 回到顶部

mvvm和mvc的区别

mvvm是module view viewmodule(数据驱动视图)，只要关注数据变化就可以，数据改变会自动驱动视图。
mvc：module view controller(控制器)在控制器中要处理数据并且渲染视图，导致controller控制操作过于多，控制臃肿，不好维护。

- 回到顶部

vue的组件创建方式

1.使用 Vue.extend 来创建全局的Vue组件
2.直接使用 Vue.component 创建
3.在被控制的 #app 外面,使用 template 元素,定义组件的HTML模板结构

- 回到顶部

vue的组件通信

- props/$emit+v-on: 通过props将数据自上而下传递，而通过$emit和v-on来向上传递信息。
- EventBus: 通过EventBus进行信息的发布与订阅
- vuex: 是全局数据管理库，可以通过vuex管理全局的数据流

- $attrs/$listeners: Vue2.4中加入的$attrs/$listeners可以进行跨级的组件通信
- provide/inject：以允许一个祖先组件向其所有子孙后代注入一个依赖，不论组件层次有多深，并在起上下游关系成立的时间里始终生效，这成为了跨组件通信的基础
- slot插槽和ref实例
- 回到顶部

computed和watch的区别

computed:

computed是计算属性,也就是计算值,它更多用于计算值的场景 computed具有缓存性,computed的值在getter执行后是会缓存的，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来计算 computed适用于计算比较消耗性能的计算场景

watch:

更多的是「观察」的作用,类似于某些数据的监听回调,用于观察props $emit或者本组件的值,当数据变化时来执行回调进行后续操作 无缓存性，页面重新渲染时值不变化也会执行

小结:

当我们要进行数值计算,而且依赖于其他数据，那么把这个数据设计为computed 如果你需要在某个数据变化时做一些事情，使用watch来观察这个数据变化

- 回到顶部

vue给对象新增属性页面没有响应

由于Vue会在初始化实例时对属性执行getter/setter转化，所以属性必须在data对象上存在才能让Vue将它转换为响应式的。Vue提供了$set方法用来触发视图更新。

- 回到顶部

v-if和v-show区别

v-show 控制display：none显示，不管条件是否成立，都会显示，适用于频繁切换条件的场景； v-if 条件成立渲染。适用于在运行很少改变条件，不需要频繁切换条件的场景

- 回到顶部

vue路由有几种模式

hash模式

即地址栏URL中的#符号，它的特点在于：hash 虽然出现URL中，但不会被包含在HTTP请求中，对后端完全没有影响，不需要后台进行配置，因此改变hash不会重新加载页面。

history模式

利用了HTML5 History Interface 中新增的pushState() 和replaceState() 方法（需要特定浏览器支持）。history模式改变了路由地址，因为需要后台配置地址。

- 回到顶部

v-model原理

Vue的双向数据绑定是由数据劫持结合发布者订阅者实现的。 数据劫持是通过Object.defineProperty()来劫持对象数据的setter和getter操作。 在数据变动时作你想做的事

原理 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 在初始化vue实例时，遍历data这个对象，给每一个键值对利用Object.definedProperty对data的键值对新增get和set方法，利用了事件监听DOM的机制，让视图去改变数据
绑定value值，绑定input事件，可以作用在组件上，需要子组件接收value属性，触发input事件。

- 回到顶部

不可以出现同名的属性或者方法

属性和方法会全部作用在组件实例上。

- 回到顶部

watch怎么监听深层嵌套

- 当值第一次绑定时，不会执行监听函数，只有值发生改变时才会执行。如果我们需要在最初绑定值的时候也执行函数，则就需要用到immediate属性。
- 当需要监听一个对象的改变时，普通的watch方法无法监听到对象内部属性的改变，此时就需要deep属性对对象进行深度监听。

```
watch: {
    docData: {
        handler(newVal) {
            this.change_number++
        },
    deep: true
    }
}
```


- 回到顶部

vue操作dom的方法

1. 作用在组件上得到组件实例
2. ref在标签上得到dom节点
3. 自定义指令

- 回到顶部

自定义指令

为了操作dom
canvas（绘制饼图） video audio（自动播放） 按钮级权限（v-identity="admin" v-identity="user"）

两种方式：
1.注册全局指令的方式，通过 Vue.directive(id, [definition]) 方式注册全局指令，第一个参数为自定义指令名称（指令名称不需要加 v- 前缀，默认是自动加上前缀的，使用指令的时候一定要加上前缀），第二个参数可以是对象数据，也可以是一个指令函数。
2.局部自定义指令，通过在Vue实例中添加 directives 对象数据注册局部自定义指令。
钩子函数：

- bind：只调用一次，指令第一次绑定到元素时调用。在这里可以进行一次性的初始化设置。
- inserted：被绑定元素插入父节点时调用 (仅保证父节点存在，但不一定已被插入文档中)。
- update：所在组件的 VNode 更新时调用。
- componentUpdated：指令所在组件的 VNode 及其子 VNode 全部更新后调用。
- unbind：只调用一次，指令与元素解绑时调用。

参数：

1. el: 指令所绑定的元素，可以用来直接操作 DOM，就是放置指令的那个元素。
2. binding: 一个对象，里面包含了几个属性，这里不多展开说明，官方文档上都有很详细的描述。
3. vnode：Vue 编译生成的虚拟节点。
4. oldVnode：上一个虚拟节点，仅在 update 和 componentUpdated 钩子中可用。

- 回到顶部

vueX

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）。页面通过mapAction异步提交事件到action。action通过commit把对应参数同步提交到mutation。 mutation会修改state中对于的值。 最后通过getter把对应值跑出去，在页面的计算属性中 通过mapGetter来动态获取state中的值

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。

- state中保存着共有数据，数据是响应式的
- getter可以对state进行计算操作，主要用来过滤一些数据，可以在多组件之间复用
- mutations定义的方法动态修改state中的数据，通过commit提交方法，方法必须是同步的
- actions将mutations里面处理数据的方法变成异步的，就是异步操作数据，通store.dispatch来分发actions，把异步的方法写在actions中，通过commit提交mutations，进行修改数据。
- modules：模块化vuex
- 回到顶部

vue-loader是啥，用途?

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

- 回到顶部

vue常见的指令

v-on v-show v-if v-else v-else-if v-for v-module v-html v-cloak(提升用户体验) v-pre(提升性能) v-once v-bind v-text

- 回到顶部

路由守卫

- 全局守卫

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

2.回调函数中的参数，to：进入到哪个路由去，from：从哪个路由离开，next：函数，决定是否展示你要看到的路由页面。

3.全局后置守卫router.afterEach((to,from)=>{}) 只有两个参数，to：进入到哪个路由去，from：从哪个路由离。 如下，每次切换路由时，都会弹出alert，点击确定后，展示当前页面。

- 组件内守卫 到达这个组件时，beforeRouteEnter:(to,from,next)=>{}

```
beforeRouteEnter:(to,from,next)=>{
        next(vm=>{
            alert("hello" + vm.name);
        })
    }
```

离开这个组件时，beforeRouteLeave:(to,from,next)=>{}

```
beforeRouteLeave:(to,from,next)=>{
        if(confirm("确定离开此页面吗？") == true){
            next();
        }else{
            next(false);
        }
    }
```

- 路由独享守卫 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。
  直接写在路由表里面。
- 回到顶部

vue修饰符

- stop：阻止事件的冒泡
- prevent：阻止事件的默认行为
- once：只触发一次
- self：只触发自己的事件行为时，才会执行

- 回到顶部

vue项目中的性能优化

1.不要在模板里面写过多表达式

2.循环调用子组件时添加key

3.频繁切换的使用v-show，不频繁切换的使用v-if

4.尽量少用float，可以用flex

5.按需加载，可以用require或者import()按需加载需要的组件

6.路由懒加载

- 回到顶部

vue-router

Vue Router 是 Vue.js 官方的路由管理器。它和 Vue.js 的核心深度集成，让构建单页面应用变得易如反掌 和和

keep-alive

做组件缓存的,把组件缓存起来不会销毁,下次打开这个组件直接显示出来,把切换出去的组件保留在内存中,保留他的状态避免重新渲染.
include（包含） exclude（排出） name="a,b,c" activated 进入 deactivated 离开
组件实例期和挂载期的生命周期函数不会被执行。

- 回到顶部

你在工作中遇到那些问题，解决方法是什么

经常遇到的问题就是Cannot read property 'prototype' of undefined 解决办法通过浏览器报错提示代码定位问题，解决问题

Vue项目中遇到视图不更新，方法不执行，埋点不触发等问题 一般解决方案查看浏览器报错，查看代码运行到那个阶段未之行结束，阅读源码以及相关文档等 然后举出来最近开发的项目中遇到的算是两个比较大的问题。

form表单 有时候需要通过v-for动态循环加载el-form-item,并且绑定验证规则。

```
<el-form :model="form">
    <div v-for="(item,index) in form.list" :key="index">
       <el-form-item
          :prop="`list.${index}.name`"
          :rules="[{ required: true, message: '名字不能为空'}]">
          <el-input v-model="item.name" placeholder="请输入"></el-input>
      </el-form-item>
    </div>
</el-form>
```

```
data:{
    form: {
      title: '',
      list: [
        {
          name: ''
        }
      ]
  }
}
```

必须通过这样绑定，:prop="item.name",无法绑定，需切记。同时，如果v-for动态循环el-form的话，以上两种方法都无法绑定，目前没有找到解决办法。

- 回到顶部

$route和$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。
- $router是“路由实例”对象包括了路由的跳转方法，钩子函数等。
- 回到顶部

Vue中的diff算法:

diff算法的本质,就是他是找出两个对象之间的差异，目的是尽可能的复用节点。 这个对象就是对应vue中的virtual dom，他是使用js对象来表示页面中的dom结构。 virtual DOM是将真实的DOM数据抽离出来，以对象的形式 模拟 树形结构，diff比较比较的也是virtual DOM。

diff算法是对操作前后的dom树 同一层 的节点进行比较，一层一层的对比，然后再插入真实的dom中来渲染。 他会给循环的列表中添加唯一标识，因为vue组件高度复用，增加了key可以识别组件的唯一性。这样diff算法就可以正确的识别次节点，并且找到正确的位置插入新的节点。

- 回到顶部

父子组件传参生命周期的执行顺序

加载渲染过程

父beforeCreate->父created->父beforeMount->子beforeCreate->子created->子beforeMount->子mounted->父mounted

子组件更新过程

父beforeUpdate->子beforeUpdate->子updated->父updated

父组件更新过程

父beforeUpdate->父updated

销毁过程

父beforeDestroy->子beforeDestory->子destoroyed->父destoryed

- 回到顶部

vue进入页面之后的渲染顺序

1.new Vue，执行初始化
2.挂载$mount方法，通过自定义Render方法、template、el等生成Render函数
3.通过Watcher监听数据的变化
4.当数据发生变化时，Render函数执行生成VNode对象
5.通过patch方法，对比新旧VNode对象，通过DOM Diff算法，添加、修改、删除真正的DOM元素

- 回到顶部

vue给对象新增属性页面没有响应

由于Vue会在初始化实例时对属性执行getter/setter转化，所以属性必须在data对象上存在才能让Vue将它转换为响应式的。Vue提供了$set方法用来触发视图更新。

```
export default {
    data(){
        return {
            obj: {
                name: 'fei'
            }
        }
    },
    mounted(){
        this.$set(this.obj, 'sex', 'man')
    }

}
```


- 回到顶部

vue.extend和vue.component

- extend 是构造一个组件的语法器。 然后这个组件你可以作用到Vue.component这个全局注册方法里 还可以在任意vue模板里使用组件。 也可以作用到vue实例或者某个组件中的components属性中并在内部使用apple组件。
- Vue.component 你可以创建 ， 也可以取组件。

- 回到顶部