**1.vue3的优势**

1. 打包大小减少了41%
2. 初次渲染快55%，更新速度快33%
3. 内存减少54%
4. 使用Proxy替代了defineProperty实现数据响应式
5. 重新虚拟dom的实现和Tree-shaking（是一种通过清除多余代码dead-code方式来优化项目打包体积的技术, 也就是找出使用的代码。）
6. 拥抱TypeScript
7. **对比vue2**
8. diff 算法的优化- vue2对虚拟DOM是进行一个全局对比；vue3新增了静态标记，对动态渲染的DOM数据进行标记，数据变化的时候只对标记的DOM进行对比更新
9. 静态提升 -vue2的DOM元素是否参与更新，都会被重新创建，进行渲染；vue3对不参与更新的DOM元素，会提出静态提升，只会创建一次，渲染时复用
10. 新增Composition (组合) API
11. **生命周期**

beforeCreate->setup

created->setup（组件创建之前，创建的是data和method）

beforeMount->onBeforeMount（组件挂载到节点之前）

mounted->onMounted（组件挂载完成后执行）

beforeUpdate->onBeforeUpdate（组件更新之前）

updated->onUpdated （组件更新完成之后执行）

beforeDestroy->onBeforeUnmount（组件卸载之前执行，实例和方法都可以放问到，清除定时器等）

destroyed->onUnmounted（组件卸载后执行）

errorCaptrued->onErrorCaptrued（当捕获一个来自子孙组件的异常时激活的函数）

vue3特有

onActivated(): 被包含在<keep-alive>中的组件，会多出两个生命周期钩子函数。被激活时执行。

onDeactivated(): 比如从 A 组件，切换到 B 组件，A 组件消失时执行。

onRenderTracked 状态跟踪（跟踪虚拟DOM重新渲染时调用，钩子函数接受debugger，event作为参数，此事件告诉你哪个操作跟踪了组件以及操作的目标对象和键）

onRenderTriggered 状态触发（当虚拟 DOM 重新渲染被触发时调用。和 renderTracked 类似，接收 debugger event 作为参数。此事件告诉你是什么操作触发了重新渲染，以及该操作的目标对象和键。）

1. **ref全家桶**

1、ref响应式（基本数据类型），自动添加 .value

2、isRef 判断是否是一个ref对象

3、shallowRef创建一个跟踪自身 .value 变化的 ref，但不会使其值也变成响应式的

4、triggerRef强制更新页面DOM

5、customRef customRef 是个工厂函数要求我们返回一个对象 并且实现 get 和 set 适合去做防抖之类的



1. **reactive 全家桶**
2. reacive 响应式复杂数据 例如对象和数据
3. readonly 只读数据
4. shallowReactive只能对浅层的数据 如果是深层的数据只会改变值 不会改变视图
5. **to系列全家桶**

1、toRef如果原始对象是非响应式的就不会更新视图 数据是会变的

2、toRefs将响应式复杂数据的单个值转化为响应式数据，可以帮我们批量创建ref对象主要是方便我们解构使用

3、toRaw将响应式对象转化为普通对象

**7.计算属性computed**：当依赖的属性的值发生变化的时候，才会触发他的更改，如果依赖的值，不发生变化的时候，使用的是缓存中的属性值。

1、let str=computed(()=>{ return ....})

2、对象形势let m=ref<string>=‘渣渣’ let num=computed({get:()=>{return m.value },set:(value)=>{m.value=‘小心心’}})

**8、watch侦听器**

1、let str=ref<number>=0; watch(str,(newValue,oldValue)=>{ console.log(newValue,oldValue)},

{immediate:true,deep:true}) immediate是否立即调用 deep深度监听

1. 监听对象属性

infers obj={name:string,age:number}

let obj=reactive<obj>({name:”渣渣喵”，age:6})

watch(()=>obj.name,(newValue,oldValue)=>{console.log(newValue,oldValue)})

1. 监听多个数据的变化

let xin=ref<number>=0;let pi=ref<string>=’小心心’

watch([xin,pi],(newValue,oldValue)=>{console.log(newValue,oldValue))

1. **watchEffect**（立即执行，同时响应式追踪其依赖，并在其依赖变更时重新运行该函数）
2. let mess1=ref<string>=’小胖心’let mess2=ref<Boolean>=true

watchEffect(() => {console.log(mess1, message.value);console.log(mess2, message2.value);})将要监听数据放入到watchEffect中就可以进行监听

1. 清除副作用

let mess1=ref<string>=’小胖心’let mess2=ref<Boolean>=true

watchEffect((oninvalidate)=>{

{oninvalidate(()=>{},

{flush:"post",onTrigger () {} //pre组件更新前执行sync强制效果始终同步触发 post组件更新后触发

})触发监听之前会调用一个函数可以处理你的逻辑例如防抖

1. 停止跟踪 watchEffect，let stop=watchEffect(()=>{}) stop()
2. **父子组件传值，**
3. defineProps接受父组件传的值，使用v-model绑定传的值，实现数据同步

TS写法defineProps<{name:string}>() 非TS写法defineProps({title:{default:""type:string }

1-2、withDefaults设置默认值

withDefaults(defineProps<Props>(), {

title: "张三",

data: () => [1, 2, 3]

}

1. 子父组件传值defineEmits 父组件自定义事件，子组件创建defineEmits实例，emit = defineEmits(['on-click']) on-click是父组件绑定的自定义事件名
2. 子组件暴露给父组件内部属性defineExpose 子组件 const count = ref(123456)

defineExpose({ count}) 父组件绑定ref=”child“ const child = ref(null) child.value.count（TS写法 menus = ref<InstanceType<typeof menus>>() //InstanceType是 运算类型）

1. **依赖注入Provide / Inject**

父组件let flag = ref<number>(1) provide('flag', flag)子组件 const flag = inject（'flag'，‘默认值设置’）

const flag = inject<Ref<number>>('flag', ref(1))

1. **兄弟之间传值**

使用mitt main.js中注册mitt组件 import mitt from "mitt"; 创建mitt实例 import type {Emitter} from 'mitt'; const Mit：Emitter = mitt();

通过/vue实例上挂载全局API

app.config.globalProperties.$Bus = Mit $Bus.emit("sayInfo","我是小弟，son2是大哥")发送

$Bus.on("sayInfo",(info:any)=>{info是传过来的信息})

**13、自定义hooks**  vue3 中的 hooks 就是函数的一种写法，就是将文件的一些单独功能的js代码进行抽离出来，放到单独的js文件中，或者说是一些可以复用的公共方法/功能。其实 hooks 和 vue2 中的 mixin 有点类似，但是相对 mixins 而言， hooks 更清楚复用功能代码的来源, 更清晰易懂。

**14、全局方法和变量**  app.config.globalProperties.进行绑定

**15、vue中的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。

**16，内置组件Teleport**

<Teleport to="body"><div>我是被 teleport 包裹的元素</div></Teleport>

1. **插槽的使用slot**

1、匿名插槽

子组件<div><slot></slot></div>

父组件 <Dialog><template v-slot><div>2132</div></template></Dialog> 使用v-slot

1. 具名插槽 子组件给slot设置name=‘header’ 父组件v-slot:header
2. 作用域插槽

子组件<slot :data="item"></slot>

父组件<template #default="{ data }"><div>{{ data }}</div></template>

1. 动态插槽
2. <Dialog><template #[name]><div>23</div></template></Dialog>

const name = ref('header')

1. **动态组件**

<component :is="A"></component>

1. **全局组件**

createApp(App).component('Card',Card).mount('#app')

1. **局部组件**在setup直接引入就可以使用
2. **异步组件**

defineAsyncComponent

defineAsyncComponent(() => import('../../components/Dialog/index.vue'))

1. **suspense**

<suspense> 组件有两个插槽。它们都只接收一个直接子节点。default 插槽里的节点会尽可能展示出来。如果不能，则展示 fallback 插槽里的节点。

<Suspense>

<template #default>

<Dialog>

<template #default>

<div>我在哪儿</div>

</template>

</Dialog>

</template>

<template #fallback>

<div>loading...</div>

</template>

</Suspense>

1. **keep-alive缓存组件**

keep-alive 生命周期的变化

初次进入时： onMounted> onActivated

退出后触发 deactivated

再次进入：

只会触发 onActivated

事件挂载的方法等，只执行一次的放在 onMounted中；组件每次进去执行的方法放在 onActivated中

1. **transition动画组件 通过绑定 name=”fade” 再通过css来设置过度动画效果**

**<button @click='flag = !flag'>切换</button>**

**<transition name='fade'><div v-if='flag' class="box"></div></transition>**

**.fade-enter-from：用来定义被fade组件包裹的元素进入开始时的样式**

**.fade-enter-to：用来定义被fade组件包裹的元素进入结束时的样式**

**.fade-leave-from：用来定义被fade组件包裹的元素离开开始时的样式**

**.fade-leave-to：用来定义被fade组件包裹的元素离开结束时的样式**

**.fade-enter-active：用来定义被fade组件包裹的元素进入过渡时的样式**

**.fade-leave-active：用来定义被fade组件包裹的元素离开过渡时的样式**