ref('') .value [ ] ← reactive( ) 对象. 局限：重新分配 失响应

对象 ~~响应式对象数据~~ Proxy(Object) Object.assign(1,2,3)

类型 RefImpl

响应式数据. {:, :} 响应式对象 分配 { }

⇒ 基本类型. value：Dot value.

let. ref. 对象.层级不深：ref. reactive 表单

层级深：表单 reactive.

{{ }}

toRefs {{ }} 插值语法 解构 → 非响应. setup

Person对象 = reactive({:;:}). toRef( )

let {name, age} = person

{ let name = per.n → 对象属性

let age = per.a

reactive → ref toRef(Person, 'age')

一组 形成 所以的对象 let {nae, age} = toRefs(person) ↓ 响应式

<input v-bind:value> </ > <span> </>

:value=" " .slice(0,1).toUpcase() 大写

计算属性. v-model=" " 双向绑定. .slice(1)

lang="ts" <template> 简单

computed：{ } vue2.

ref.computed. let fullname = computed(()=>{ })

↓ 计算 function 没有缓存. 可读可写

且只可读. ①②→③→① let fullName = Computed({

ComputedRefImpl get( ){return }

const [str1, str2] = val.split('-') set( ){ } })

watch 监听（数据）变化

四种

ref() → 基本 / 对象. (let sum = ref(0) value？ 所需 / 不需

reactive

ref 定义的数据

函数返回一个值

1. 数组会包含. 监视的是对象的

{watch}

1. 第一种？ 回调函数

watch(sum, ()=>{ (newValue, oldValue) 地址值，

内部属性：开启深度监视

watch(, ,) if(>10) stopWatch()

3. 解除监视

watch(, , {deep:true} , immediate: true)

新 (value) 属性 的同一指

Object.assign( ) 修改 ~~地址值~~ 2. 对象：

监视 reactive（默认开启深度监视）

隐式创建深监听

obj = reactive({ a:{ b:{ c:666 } } })

import {reactive, watch} from 'vue'.

car: { c1:'', c2:'' }

getter 函数 → 能返回一个值的函数

watch(~~()=>{ return person. person.name }~~ , ()=>{ })

()=>person.name,

4，响应式对象中某属性 函数式.

→ 对象：

person.car, {deep:true} → { 地址 / 内部 }

建议写成函数，

(()=>person.car , {deep:true})

②

watch() 监视上述的多个数据 [()=>, ()=>]

watch([()=>person.name, ()=>person.car.c], ( )=>{ },
  {deep:true}
)

Effect 影响

{ref, watch}.

watchEffect

watch([temp, height], ( )=>{ }
) (降雨, ~~[illegible]~~) 带参)

watchEffect(()=>{ if(temp.value>=60 || h.value>=80) { log( );
})

全自动监视. 响应式

打标记.

JS → dom

con..log(document.getElementById('title2'))

<h2 ref="title2"></>

let title2 = ref().
容器

data-
局部样式 <style scoped>
局部样式

ref → html 标签 → DOM

<person/> 组件标签. 实例对象, 保护措施

import {defineExpose} from 'vue'

defineExpose({a,b,c}).

TS

types/index.ts

export interface PersonInter { [illegible]
id:string
接口限制 属性,
对象
}

import { } from '@/'

type PersonInter

let person : PersonInter = { }

Let personList : Array<personInter> = [{ }, { }, { }]
泛型.

// 自定义类型

export
typei persons = Array<personInters>
personInter[ ]

import {reactive } from 'vele'

Let personList = reactive<Persons>([{ }, { }, { }])

x : ?

import {defineProps} from 'vue'.

let x
defineProps(['a'])

['a', 'b']

<h2>{{a}}</h2>

<Person :list="personList"

:b="1+1"
js表达式

ref= ""
打标记,

<ul>

<li

>xx</li> :d="x"

表达式

</ul>

{{ }}={{ }}

xx.

xx.

v-for=" (??) in ?? "

数据源
里的每一项
形参,
xx

数据源

:key="p.id" 唯一标识,

v-for 遍历

④

接收list +限制类型.

defineProps(['list'])

defineProps<{list:Persons}>()

['list']

+限制必要性.

defineProps<{list?:Persons}>( )

[ ]

{id: ,}

[{ }]

withDefaults( )

{list : ()=>[{ ; ; ; }]}

difine 宏函数.

生命周期(组件) 钩子.

{生命周期函数

4 {
创建 beforeCreate created
挂载 beforMount mounted.
更新 beforUpdate updated debugger;
销毁. beforDestory destroyed.

v-if="条件"
isshow

vue.

v-show="ishow"

vue3 {
//创建: setup
//挂载: onBeforMount(()=>{ log(' ') })
onMounted. (()=>{ })

v-if="ishow"

let ishow = ref(true)

//更新 onBeforUpdate(()=>{})
onUpdated(()=>{ })

子-父.

App.vue 最后挂载

//卸载前
onBeforeUnmount
onUnmounted(()=>{ })

onMounted
onUpdated
onBeforeUnmount

---

npm i axios

<hr>

let doglist = reactive([
])

img{
width;
}

v-for="(dog,indx) in doglist"

:src="dog"

:key="indx"

]

```
fun getDog(){
let res=axios.get(' ').t
  await
    log(res.data.message.
    dogList.push()
        res.data.messg
}
```

img margin-right: 10px 右边距

```
try{

}catch(error){
    alert(
        error)
}
```

axios 拦截器

hooks
mixin

hooks.xx.js/xx.ts

use Xxx
功能.

hooks/useSum.ts
useDog.ts

```
                default
export ~~fun~~  export default func(){
  function(){                          }

   //向外部提供东西 return {  }
}                          doglist, getDog

 export default function(){
                         const [sum,add]=useSum()
 //
 return {    ,    }       const {doglist,dog}=useDog()
}
   computed.        OnMounted.( ()=>{ getDog()})
let bigSum= computed( ()=>{ return sum.value*10})
      return{ bigSum}
```

pages/views/ ~~nigx~~. nginx

try-files $uri//indx.html

history: createWebHistory()

拉 hash 模式, SEO模式.

to="/home".

<RouterLink></>

对象写法. :to="{path:'/about/'}"

路由起名: name:'' 1 跳转. ↳ to="/xx"

2 ⟶ 3 :to="{name:''}" 对象写法.

```
<ul>
  <li v-for="  " :key="id">
```

#list-sty: none;

. new li::marker{color: }

useRoute ← hooks

/news/deati ? xxx

route 传参,

router

?=

:to=" ?=${. }&title=${ , }"

:to="{ , }"

path:"//" / name:'',

query:{id:,
title:,
cont:,}

let {query}=toRefs(rote)

toRefs( )

params /:id/:title/:cont. params:{ :,
:,
:,}

占位符.

name:'', params / route props
规定

必 ? 可传
参数必要性.

props: true
definProps(['id', 'title', 'content'])

所有 params 参数.

query

2. props() { return { ' ', } }
route
route.query.

历史记录: replace.
push

<RouterLink . replace.
<a >

编程式导航

cosnt router= useRouter()

import {onMounted} from 'vue'

OnMoun(()=>{ SetTimeOut(()=>{
router.push('/news')
},3000
)
})

push({ name:'',
query:{ }
})

RouterLink 如

router.push( )

1, 符合条件跳转.

2, 鼠标划过.

{ path:'/',
redirect: '/new'
}

Pinia 集中式状态管理.
(数据)

①value=" "

v-model.number="n"

.unshift()

let obj = {id: nanoid(),

title = data. cotent}

{createpinia} fun pinia {data: {content : title}}.

//const pinia = createPinia()

app. use(pinia)

分类.

store/

count.ts.

import {defineStore} fro 'pinia'

const useCountStore = defineStore('count',

export { state(){ return{

sum:6

}

}

})

impo {} from '@/stor/cont'

use.

const ~~store~~ = useCou( )

Countsto

reactive( )

自动解包

{state.______

table:[{ }]

综合自变

.$patch({sum:

schu: '',

address: '',

})

=storeToRefs( )

只会把store数据

```
getters:{
    bigSum(){return 88}};
}
[]string{     }
$event
```

PS0

PS1

①开源对象

存储服务器

存储:图片、文件、视频

上传 → 压缩 → 存储

返回url地址链接

url

后台、数据库 开源对象存储无必要性.

本地 服务器文件夹 文件存储.

下载

云盘链接文件分享

设置公开文件(分链接.

云文件

文章内容

个人内容分享:文字、视频

免费云盘链接

文件分享