**\*\*Vue组件的使用步骤:\*\***

①.创建组件:按照大驼峰命名法命名组件(例如:HelloWorld.vue)

②.引入组件:在父组件中(例如:App.vue)引入,语法:import 组件名称 from '组件的路径'(写在script标签对内)

③.使用组件:在父组件中(例如:App.vue)的模板(template中)使用标签

语法:

<template>

 <组件名称></组件名称>

 (该地方可以有俩种,第一种大驼峰命名法<HelloWorld></HelloWorld>,第二种短横线命名法<hello-world></hello-world>)

</template>

数据的输出

①使用插值表达式显示数据,语法:{{数据的命名}}

定义数据写在script标签对内

②const 数据的名称='字符串'(或数字,{对象},[数组])

let 数据的名称=''

例如:const msg='你好'

     const age=18

     const user={

        name:'张三'，

        age：18

     }

     const=[1,2,3,4]

const:用来表示不打算修改引用的变量

let:用来表示可能会修改的变量

在vue3中,主要使用const或者let只是为了代码的清晰性和维护性

数据：静态数据和动态数据（数据可以在用户交互或者其他事件下自动更新）

使用ref（）函数创建响应式数据（字符串，数字）

注意：使用ref函数要在script中引入，语法为：import {ref} from 'vue'

在script中修改ref绑定的数据是需要通过.value进行修改，在template中修改绑定的数据不需要.value

例子：参考RefData.vue中的例子

在script中修改的例子

function changeText(){

    //里面写具体运行的代码

    msg.value='未来可期'

};

在template中修改的例子

   <button  @click="tel='12345678'">点击查看联系方式</button>

使用reactive（）函数创建响应式对象（或者数组）

注意：使用reactive函数要在script中引入，语法为：import {reactive} from 'vue'

在script和template中修改数据都不需要通过.value进行修改

总结:如果一个例子中同时用到ref和reactive,那么引用俩个函数的时候可以写在一起

语法:import {reactive,ref} from 'vue'

js函数:(ReactiveData.vue中)

1.join()将数组元素链接成字符串

可以指定分隔符,默认是逗号

join()数组里面的内容显示用逗号隔开,例如:阅读,旅行,编程

join('\*')表示分隔符为\*,阅读\*旅行\*编程

2.alert('要显示的内容'):弹窗函数

3.push()用于向数组的末尾添加一个或者多个元素

语法:要添加的数组.push(内容1,内容2,内容3)

4.unshift()用于向数组的开头添加一个或者多个元素

语法:要添加的数组.unshift(内容1,内容2,内容3)

5.pop()用于删除数组的最后一个元素

6.shift()用于删除数组的第一个元素

语法:要添加的数组.pop()

7.splice()函数:用于修改数组的内容,可以在数组中添加,删除或者替换元素

语法:splice(开始操作的位置,要删除元素的个数,要添加的元素1,要添加的元素2,要添加的元素3)

开始操作的位置:表示从第几个元素开始操作,顺序是按照索引进行的,索引为0表示第一个位置,索引为1表示第二个位置

语法结构:

要操作的数组.splice(开始操作的位置,要删除元素的个数,要添加的元素1,要添加的元素2,要添加的元素3)

\*\*\*

## Vue基础指令

## `1.内容渲染指令`

\*\*v-text指令\*\*:用于更新元素的文本内容,只能渲染纯文本

\*\*v-html指令\*\*:用于更新元素的文本内容,可以渲染标签

## `2.属性绑定指令`

用法:html属性后面跟的值是响应式的,不是固定的值,此时就会用到v-bind去进行绑定

\*\*v-bind指令\*\*:用于绑定html属性

注意：区分html属性和css属性，html属性是跟在html标签后面的,写在style里面的都是css属性

<input type='text'></input>

这里面的type就是一个html属性

<style>

      a{

         color:red;

      }

  </style>

  这里面的color就是一个css属性

<input type='text' style='color:red;'></input>

这里面的color也是一个css属性

\*\*\*语法:<标签 v-bind:属性='数据名'></标签>\*\*\*

v-bind:属性 可以简写成 :属性 (也就是v-bind可以不写,但是:号一定要有)

\*\*图片的引用\*\*

pubilc文件夹里面新建images文件夹,在images文件夹里面放图片

引用的路径:/images/图片的名称.图片的后缀

\*\*v-bind指令\*\*:用于绑定内联样式

```

语法:<标签名 v-bind:style='样式名称'></标签名>

```

例子:

```

<div style="width: 300px;height: 100px;background-color: red;color: yellow;padding: 10px;">未来可期</div>

```

本来标签后面的style属性里面是具体的值,那现在我们用v-bind绑定style之后,为了代码结构清晰,我们可以直接在style后面跟一个样式的名称,然后关于具体的样式在script标签对内定义

\*\*特别注意:用v-bind绑定style之后,写对应的属性的时候,原来是短横线形式的属性,要改写成小驼峰的形式(第二个字母要大写)\*\*

例如:background-color要写成backgroundColor

\*\*v-bind指令\*\*:用于绑定css类

```

语法:<标签名 v-bind:class='选择器的名称'></标签名>

```

选择器的样式是写在style标签里的

v-bind:class用来动态的控制哪些css类可以应用在相应的元素上

## `3.循环指令v-for`

用来遍历数组,对象,数字渲染出多个对象

key是一个必要的属性,用来帮助vue跟踪每个元素,提升渲染效率

\*\*数字渲染列表\*\*:

<标签名 v-for='(item,index) in 数字' :key='index'></标签名>

\*\*数组渲染列表\*\*

<标签名 v-for='(item,index) in 数组的名称' :key='index'></标签名>

\*\*对象渲染列表\*\*

<标签名 v-for='(item,index) in 对象的名称' :key='index'></标签名>

注意:item可以根据具体的案例换名称,item不是固定的,但是index表示索引,index是固定的

## `4.条件渲染指令`

v-if v-else v-else-if

条件渲染指令根据表达式的值决定是否渲染

## `5.双向数据绑定指令`

v-model 用于在表单控件中（如输入框，复选框，单选按钮等）和组件的属性之间创建双向数据绑定

注意：单选框，下拉列表中的value和属性要设置有效的值，这样v-model才能正确绑定数据

## `6.事件绑定指令`

v-on 用于监听DOM事件并在事件触发时执行js代码

语法：v-on：事件

v-on：事件可以简写成@事件

常见的几个事件

@click（单机） @dblclick（双击）@mouseover（鼠标悬停上去）@mouseout（移开）

事件修饰符：

@keydown.按键的名称

**## 组件的注册**

**### 局部注册**

和在app.vue中注册的方法是一样的

<script setup>import 组件名 from '路径'</script>

<template>

<组件名></组件名>

</template>

**### 全局注册**

全局注册组件需要再main.js中进行导入和注册

```

main.js 中需要添加代码如下：

import 组件名 from '路径'

app.compoment('组件名'，组件名)

```

如果一个组件在main.js 中注册为全局组件，其他组件或者是页面中需要使用该组件，只需要写<组件名></组件名>就可以

如果一个组件只是作为子组件，在父组件中进行使用，那就使用局部注册

如果一个组件要在多个组件或是页面中使用，那就使用全局注册

**## 组件之间的数据传递**

**### 子组件向父组件传递数据**

在子组件中使用defineEmits来申明事件

const emit=defineEmits(['事件函数'])

fuction 事件函数(参数){

   emit('事件函数'，参数)

}

父组件通过@事件函数='方法' 来接受事件并处理数据(相应的代码跟在父组件中相应的子组件后面)

**### 父组件向子组件传递数据**