

# 一.Vue是什么

## 二.Vue核心语法

### 1.setup

#### 1.setup概述

#### 2.setup返回值

#### 3.setup与OptionsAPI

#### 4.setup的语法糖

#### 5.ref创建\_基本类型的响应式数据

#### 6.reation创建\_对象类型的数据 (只能定义对象类型的响应式数据)

#### 7.ref创建\_对象类型的响应式数据

#### 8.ref对比reative

宏观角度看：

1. `ref` 用来定义：基本类型数据、对象类型数据；
2. `reactive` 用来定义：对象类型数据。

• 区别：

1. `ref` 创建的变量必须使用 `.value` (可以使用 `volar` 插件自动添加 `.value`) 。
2. `reactive` 重新分配一个新对象，会失去响应式 (可以使用 `Object.assign` 去整体替换) 。

• 使用原则：

1. 若需要一个基本类型的响应式数据，必须使用 `ref` 。
2. 若需要一个响应式对象，层级不深，`ref`、`reactive` 都可以。
3. 若需要一个响应式对象，且层级较深，推荐使用 `reactive` 。

### 3.7. 【toRefs 与 toRef】

修改整个对象的方法,第一张图片不可以,第二张图片可以

```
// 数据
let car = reactive({brand:'奔驰',price:100})
let sum = ref(0)

// 方法
function changeBrand(){
  car.brand = '宝马'
}
function changePrice(){
  car.price += 10
}
function changeCar(){
  car = {brand:'奥拓',price:1}
}
function changeSum(){
  sum.value += 1
}

/script>
```

```
function changeCar(){
  // car = reactive({brand:'奥拓',price:1})
  Object.assign(car,{brand:'奥拓',price:1})
}
```

```
// 数据
let car = ref({brand:'奔驰',price:100})
let sum = ref(0)

// 方法
function changeBrand(){
  car.value.brand = '宝马'
}
function changePrice(){
  car.value.price += 10
}
function changeCar(){
  * // car = {brand:'奥拓',price:1} //这么写页面不更新的
  // car = reactive({brand:'奥拓',price:1}) //这么写页面不更新的

  // 下面这个写法页面可以更新
  // Object.assign(car,{brand:'奥拓',price:1})
  car.value = {brand:'奥拓',price:1}
}
```

## 9.toRefs和toRef

```
<script lang="ts" setup name="Person">
  import {reactive} from 'vue'

  // 数据
  let person = reactive({
    name:'张三',
    age:18
  })

  let {name,age} = person

  // 方法
  function changeName(){
    name += '~'
  }
  function changeAge(){
    age += 1
  }
</script>
```

Handwritten annotations:

- Red text: `person.name = 666`
- Red text: `person.age = 777`
- Red checkmark and arrow pointing to the `person` variable.
- Green text: `let name = person.name` (with `name` boxed in green)
- Green text: `let age = person.age`
- Green arrows pointing from `666` to `name` and `person.name`.
- Green text: `666` next to the `name` assignment.


Bottom right corner: 英语 高顿

```
<script lang="ts" setup name="Person">
  import {reactive,toRefs} from 'vue'

  // 数据
  let person = reactive({
    name:'张三',
    age:18
  })

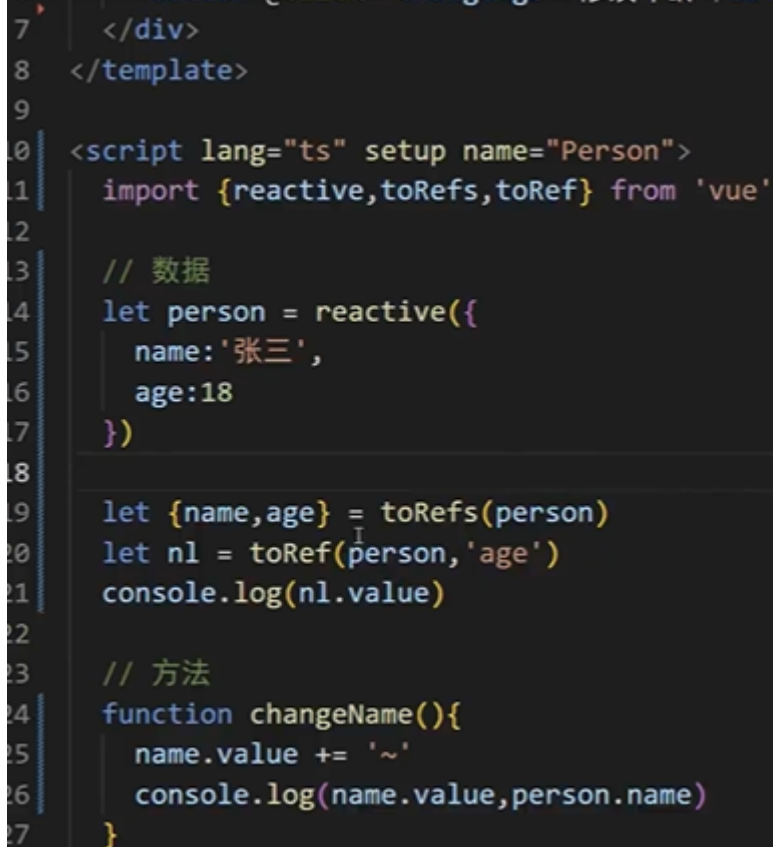
  let {name,age} = toRefs(person)

  // 方法
  function changeName(){
    name += '~'
    console.log(name,person.name)
  }
  function changeAge(){
    age += 1
  }
</script>
```



A diagram with two arrows pointing to the word 'ref'. A yellow arrow points from the 'toRefs' function call in the code to the 'ref' label. A green arrow points from the 'person' variable in the code to the 'ref' label.

```
7   </div>
8   </template>
9
10  <script lang="ts" setup name="Person">
11    import {reactive,toRefs,toRef} from 'vue'
12
13    // 数据
14    let person = reactive({
15      name:'张三',
16      age:18
17    })
18
19    let {name,age} = toRefs(person)
20    let n1 = toRef(person,'age')
21    console.log(n1.value)
22
23    // 方法
24    function changeName(){
25      name.value += '~'
26      console.log(name.value,person.name)
27    }
```



The screenshot shows a code editor with a dark theme. The code is in TypeScript and uses Vue.js. It defines a reactive object 'person' with 'name' and 'age' properties. It then uses 'toRefs' to create refs for 'name' and 'age', and 'toRef' to create a ref for 'age'. The 'changeName' function updates 'name.value' and logs it along with 'person.name'. The 'changeAge' function is also present but not shown in the screenshot. The editor has a line number on the left and a status bar at the bottom.

## 核心区别

场景	toRef	toRefs
处理范围	单个属性	所有属性
返回值	单个 ref 对象	包含多个 ref 的普通对象
用途	单独提取某个响应式属性	解构整个响应式对象（避免丢失响应性）

toRef 和 toRefs 的核心价值是保留响应式对象属性的响应性，尤其在解构或传递属性时非常有用，避免因普通解构导致响应性丢失。

## 10.computed计算属性

```
let firstName = ref('zhang');
let lastName = ref('san');
let fullName = computed(() => {
  return firstName.value.slice(0,1).toUpperCase() + firstName.value.slice(1) + '-' + la
});

function changeFullName(){
  firstName.value = 'li';
  lastName.value = 'si';
}
```

```
// 这么定义的fullName是一个计算属性，且是只读的
/* let fullName = computed(()=>{
  console.log(1)
  return firstName.value.slice(0,1).toUpperCase() + firstName.value.slice(1) + '-' + lastName.value
}) */
```

```
// 这么定义的fullName是一个计算属性，可读可写
let fullName = computed({
  get(){
    return firstName.value.slice(0,1).toUpperCase() + firstName.value.slice(1) + '-' + la
  },
  set(val){
    const [str1,str2] = val.split('-')
    firstName.value = str1
    lastName.value = str2
  }
})
```

```
> components > Person.vue > {} script setup
21 // 这么定义的fullName是一个计算属性，且是只读的
22 /* let fullName = computed(()=>{
23   console.log(1)
24   return firstName.value.slice(0,1).toUpperCase() + firstName.value.slice(1) + '-' + lastName.value
25 }) */
26
27 // 这么定义的fullName是一个计算属性，可读可写
28 let fullName = computed({
29   get(){
30     return firstName.value.slice(0,1).toUpperCase() + firstName.value.slice(1) + '-' + lastName.value
31   },
32   set(val){
33     console.log('set', val)
34   }
35 })
36
37 function changeFullName(){
38   fullName.value = 'li-si'
39 }
40 </script>
41
```

## 2.watch

### 3.9. 【watch】

- 作用：监视数据的变化（和 Vue2 中的 watch 作用一致）
- 特点：Vue3 中的 watch 只能监视以下四种数据：

1. ref 定义的数据。
2. reactive 定义的数据。
3. 函数返回一个值。
4. 一个包含上述内容的数组。

我们在 Vue3 中使用 watch 的时候，通常会遇到以下几种情况：

#### \* 情况一

监视 ref 定义的【基本类型】数据：直接写数据名即可，监视的是其 value 值的改变。

```
<template>
  <div class="person">
    <h2>watch情况一：监视【ref】定义的【基本类型】数据，默认监视的就是value值。</h2>
    <hr>
    <h3>当前求和：{{sum}}</h3>
    <button @click="changeSum">sum+1</button>
  </div>
</template>
```

#### 1.情况一

## \* 情况一

监视 `ref` 定义的【基本类型】数据：直接写数据名即可，监视的是其 `value` 值的改变。

```
<template>
  <div class="person">
    <h2>watch情况一：监视【ref】定义的【基本类型】数据，默认监视的就是value值。 </h2>
    <hr>
    <h3>当前求和：{{sum}}</h3>
    <button @click="changeSum">sum+1</button>
  </div>
</template>

<script setup lang="ts" name="Person">
  import {ref,watch} from 'vue'
  // 数据
  let sum = ref(1)
  // 方法
  function changeSum(){
    sum.value += 1
  }

  // 情况一：监视ref定义的【基本类型】数据，直接写数据名即可，不用写.value，但监视的是其value值的改变
  // watch调用的返回值stopwatch，是用于停止监听的函数
  const stopwatch = watch(sum, (newValue,oldValue)=>{
    console.log('sum变化了',newValue,oldValue)
    if(newValue >= 10){
      stopwatch()
    }
  })
```

```
运行(R) 终端(T) 帮助(H) Person.vue - hello_vue3 - Visual Studio Code
Person.vue M X
src > components > Person.vue > {} script setup
1 <template>
2   <div class="person">
3     <h2>当前求和为：{{sum}}</h2>
4     <button @click="changeSum">点我sum+1</button>
5   </div>
6 </template>
7
8 <script lang="ts" setup name="Person">
9   import {ref,watch} from 'vue'
10  // 数据
11  let sum = ref(0)
12  // 方法
13  function changeSum(){
14    sum.value += 1
15  }
16 </script>
17
18 <style scoped>...
31 </style>

export default {
  name:'Person',
  data(){
    return {sum:0}
  },
  watch:{ 0 }
}
```

```
7
8 <script lang="ts" setup name="Person">
9   import {ref,watch} from 'vue'
10  // 数据
11  let sum = ref(0)
12  // 方法
13  function changeSum(){
14    sum.value += 1
15  }
16  // 监视
17  watch(sum,(newValue,oldValue)=>{
18    console.log('sum变化了',newValue,oldValue)
19  })
20 </script>
21
22 > <style scoped> ...
35 </style>      I
```

```
}
// 监视, 情况一: 监视【ref】定义的【基本类型】数据
const stopWatch = watch(sum,(newValue,oldValue)=>{
  console.log('sum变化了',newValue,oldValue)
  if(newValue >= 10){
    stopWatch()
  }
})
</script>
```

## 2.情况二



## \* 情况二

监视 `ref` 定义的【对象类型】数据：直接写数据名，监视的是对象的【地址值】，若想监视对象内部的数据，要手动开启深度监视。

注意：

- 若修改的是 `ref` 定义的对象中的属性，`newValue` 和 `oldValue` 都是新值，因为它们是同一个对象。
- 若修改整个 `ref` 定义的对象，`newValue` 是新值，`oldValue` 是旧值，因为不是同一个对象了。

```
<template>
  <div class="person">
    <h2>watch情况二：监视【ref】定义的【对象类型】数据。</h2>
    <hr>
    <h3>人员信息：{{person.name}} . 今年{{person.age}}岁</h3>
    <button @click="changeName">修改名字</button>
    <button @click="changeAge">修改年龄</button>
    <button @click="changePerson">修改整个人</button>
  </div>
</template>

<script setup lang="ts" name="person">
  import {ref,watch} from 'vue'
  // 数据
  let person = ref({
    name:'张三',
```

ref



```
  }
  // 方法
  function changeName(){
    person.value.name += '~'
  }
  function changeAge(){
    person.value.age += 1
  }
  function changePerson(){
    person.value = {name:'李四',age:90}
  }
  // 监视，情况一：监视【ref】定义的【对象类型】数据，监视的是对象的地址值
  watch(person,(newValue,oldValue)=>{
    console.log('person变化了',newValue,oldValue)
  })
</script>
```

监视【ref】定义的【对象类型】数据，监视的是对象的地址值，若想监视对象内部属性的变化，需要手动开启深度监视

```
  // 监视，情况一：监视【ref】定义的【对象类型】数据，监视的是对象的地址值，若想监视对象内部属性的变化，需要手动开
  watch(person,(newValue,oldValue)=>{
    console.log('person变化了',newValue,oldValue)
  },{deep:true})
</script>
```

```

}
// 监视, 情况一: 监视【ref】定义的【对象类型】数据, 监视的是对象的地址值, 若想监视对象内部属性的变化, 需要手动开
// watch的第一个参数是: 被监视的数据
// watch的第二个参数是: 监视的回调
// watch的第三个参数是: 配置对象 (deep、immediate等等。。。)
watch(person, (newValue, oldValue) => {

```

### 3.情况三

左边是真正替换,右边只是改变值

```

import {ref, watch} from 'vue'
// 数据
let person = ref({
  name: '张三',
  age: 18
})
// 方法
function changeName(){
  person.value.name += '~'
}
function changeAge(){
  person.value.age += 1
}
function changePerson(){
  person.value = {name: '李四', age: 90}
}

import {reactive, watch} from 'vue'
// 数据
let person = reactive({
  name: '张三',
  age: 18
})
// 方法
function changeName(){
  person.name += '~'
}
function changeAge(){
  person.age += 1
}
function changePerson(){
  Object.assign(person, {name: '李四', age: 80})
}

```

默认开启深度监视,并且深度监视无法关闭

```

32 }
33 function changeAge(){
34   person.age += 1
35 }
36 function changePerson(){
37   Object.assign(person, {name: '李四', age: 80})
38 }
39 function test(){
40   obj.a.b.c = 888
41 }
42
43 // 监视, 情况三: 监视【reactive】定义的【对象类型】数据, 且默认是开启深度监视的
44 watch(person, (newValue, oldValue) => {
45   console.log('person变化了', newValue, oldValue)
46 })
47 watch(obj, (newValue, oldValue) => {
48   console.log('obj变化了', newValue, oldValue)
49 })
50
51 </script>
52

```

### 4.情况四

## \* 情况四

监视 `ref` 或 `reactive` 定义的【对象类型】数据中的某个属性，注意点如下：

1. 若该属性值不是【对象类型】，需要写成函数形式。
2. 若该属性值是依然是【对象类型】，可直接编，也可写成函数，不过建议写成函数。

```
<template>
  <div class="person">
    <h2>watch情况三：监视ref或reactive定义的【响应式对象】中的某个数据。</h2>
    <hr>
    <h2>当前人的信息：</h2>
    <h3>姓名：{{person.name}}</h3>
    <h3>年龄：</h3>
    <ul>
      <li>对外：{{person.age.foreignAge}}</li>
      <li>真实：{{person.age.realAge}}</li>
    </ul>
    <button @click="changeName">改姓名</button>
    <button @click="changeforeignAge">改对外年龄</button>
  </div>
</template>

<script setup lang="ts" name="Person">
  import {reactive,watch} from 'vue'

  let person = reactive({
```

## getter函数 能返回一个值的函数

不是对象类型

```
let person = reactive({
  name: '张三',
  age: 18,
  car: {
    c1: '奔驰',
    c2: '宝马'
  }
})

watch(() => person.name, (newValue, oldValue) => {
  console.log('person.name变化了', newValue, oldValue)
})
```

是对象类型

```
// 监视，情况四：监视响应式对象中的某个属性，且该属性是基本类型的，要写成函数式
/* watch(()=> person.name, (newValue,oldValue)=>{
  console.log('person.name变化了',newValue,oldValue)
}) */
```

```
watch(()=>person.car, (newValue,oldValue)=>{
  console.log('person.car变化了',newValue,oldValue)
},{deep:true})
```

新属性

```
// 监视，情况四：监视响应式对象中的某个属性，且该属性是基本类型的，要写成函数式
/* watch(()=> person.name, (newValue,oldValue)=>{
  console.log('person.name变化了',newValue,oldValue)
}) */
```

```
// 监视，情况四：监视响应式对象中的某个属性，且该属性是对象类型的，可以直接写，也能写函数，更推荐写函数
watch(()=>person.car, (newValue,oldValue)=>{
  console.log('person.car变化了',newValue,oldValue)
},{deep:true})
```

监视 `ref` 或 `reactive` 定义的【对象类型】数据中的某个属性，注意点如下：

1. 若该属性值不是【对象类型】，需要写成函数形式。
2. 若该属性值是依然是【对象类型】，可直接编，也可写成函数，建议写成函数。

结论：监视的要是对象里的属性，那么最好写函数式，注意点：若是对象监视的是地址值，需要关注对象内部，需要手动开启深度监视