目录

[1 Vue基础 1](#_Toc89607946)

[1.1Vue组件之间的通信方式 1](#_Toc89607947)

[1.2 vuex 4](#_Toc89607948)

[1.3 $attrs/$listeners 4](#_Toc89607949)

# Vue基础

## 1.1Vue组件之间的通信方式

·props 和emit 父组件向子组件传递数据是通过 prop 传递的，子组件传递数据给父组件是通过emit 触发事件来做到的。

·parent,children 获取当前组件的父组件和当前组件的子组件

·vuex 状态管理

·attrs 和listeners A->B->C。Vue 2.4 开始提供了attrs 和listeners 来解决这个问题。

方法一：通过使用props和emit的方式进行通信

**1.父组件向子组件传值**

接下来我们通过一个例子，说明父组件如何向子组件传递值：在子组件Users.vue中如何获取父组件App.vue中的数据 users:["Henry","Bucky","Emily"]

//App.vue父组件

<template>

<div id="app">

<users v-bind:users="users"></users>//前者自定义名称便于子组件调用，后者要传递数据名

</div>

</template>

<script>

import Users from "./components/Users"

export default {

name: 'App',

data(){

return{

users:["Henry","Bucky","Emily"]

}

},

components:{

"users":Users

}

}

//users子组件

<template>

<div class="hello">

<ul>

<li v-for="user in users">{{user}}</li>//遍历传递过来的值，然后呈现到页面

</ul>

</div>

</template>

<script>

export default {

name: 'HelloWorld',

props:{

users:{ //这个就是父组件中子标签自定义名字

type:Array,

required:true

}

}

}

</script>

**总结：父组件通过props向下传递数据给子组件。注：组件中的数据共有三种形式：data、props、computed**

**2.子组件向父组件传值（通过事件形式）**

接下来我们通过一个例子，说明子组件如何向父组件传递值：当我们点击“Vue.js Demo”后，子组件向父组件传递值，文字由原来的“传递的是一个值”变成“子向父组件传值”，实现子组件向父组件值的传递。



// 子组件

<template>

<header>

<h1 @click="changeTitle">{{title}}</h1>//绑定一个点击事件

</header>

</template>

<script>

export default {

name: 'app-header',

data() {

return {

title:"Vue.js Demo"

}

},

methods:{

changeTitle() {

this.$emit("titleChanged","子向父组件传值");//自定义事件 传递值“子向父组件传值”

}

}

}

</script>

// 父组件

<template>

<div id="app">

<app-header v-on:titleChanged="updateTitle" ></app-header>//与子组件titleChanged自定义事件保持一致

// updateTitle($event)接受传递过来的文字

<h2>{{title}}</h2>

</div>

</template>

<script>

import Header from "./components/Header"

export default {

name: 'App',

data(){

return{

title:"传递的是一个值"

}

},

methods:{

updateTitle(e){ //声明这个函数

this.title = e;

}

},

components:{

"app-header":Header,

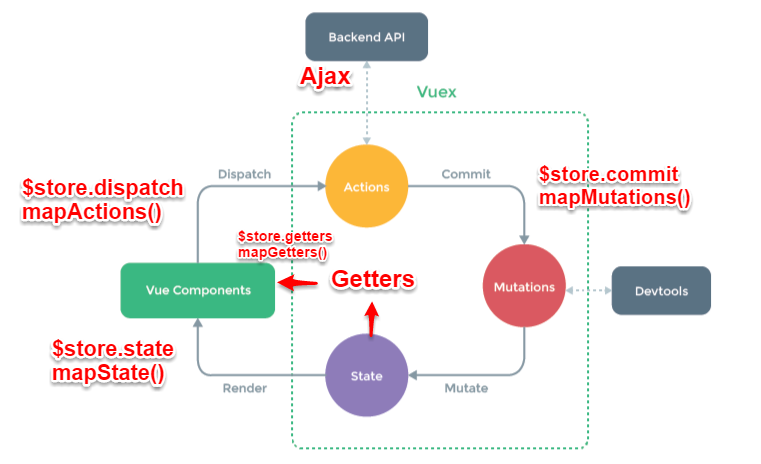
}

}

</script>

**总结：子组件通过events给父组件发送消息，实际上就是子组件把自己的数据发送到父组件。**

## 1.2 Vuex



vuex 是 vue 的状态管理器，存储的数据是响应式的。

## 1.3 $attrs/$listeners

多级组件嵌套需要传递数据时，通常使用的方法是通过vuex。但如果仅仅是传递数据，而不做中间处理，使用 vuex 处理，未免有点大材小用。为此Vue2.4 版本提供了另一种方法----$attrs/$listeners

$attrs：包含了父作用域中不被 prop 所识别 (且获取) 的特性绑定 (class 和 style 除外)。当一个组件没有声明任何 prop 时，这里会包含所有父作用域的绑定 (class 和 style 除外)，并且可以通过 v-bind="$attrs" 传入内部组件。通常配合 interitAttrs 选项一起使用。

$listeners：包含了父作用域中的 (不含 .native 修饰器的) v-on 事件监听器。它可以通过 v-on="$listeners" 传入内部组件

接下来我们看个跨级通信的例子：

// index.vue

<template>

<div>

<h2>浪里行舟</h2>

<child-com1

:foo="foo"

:boo="boo"

:coo="coo"

:doo="doo"

title="前端工匠"

></child-com1>

</div>

</template>

<script>

const childCom1 = () => import("./childCom1.vue");

export default {

components: { childCom1 },

data() {

return {

foo: "Javascript",

boo: "Html",

coo: "CSS",

doo: "Vue"

};

}

};

</script>

// childCom1.vue

<template class="border">

<div>

<p>foo: {{ foo }}</p>

<p>childCom1的$attrs: {{ $attrs }}</p>

<child-com2 v-bind="$attrs"></child-com2>

</div>

</template>

<script>

const childCom2 = () => import("./childCom2.vue");

export default {

components: {

childCom2

},

inheritAttrs: false, // 可以关闭自动挂载到组件根元素上的没有在props声明的属性

props: {

foo: String // foo作为props属性绑定

},

created() {

console.log(this.$attrs); // { "boo": "Html", "coo": "CSS", "doo": "Vue", "title": "前端工匠" }

}

};

</script>

// childCom2.vue

<template>

<div class="border">

<p>boo: {{ boo }}</p>

<p>childCom2: {{ $attrs }}</p>

<child-com3 v-bind="$attrs"></child-com3>

</div>

</template>

<script>

const childCom3 = () => import("./childCom3.vue");

export default {

components: {

childCom3

},

inheritAttrs: false,

props: {

boo: String

},

created() {

console.log(this.$attrs); // {"coo": "CSS", "doo": "Vue", "title": "前端工匠" }

}

};

</script>

// childCom3.vue

<template>

<div class="border">

<p>childCom3: {{ $attrs }}</p>

</div>

</template>

<script>

export default {

props: {

coo: String,

title: String

}

};

</script>

  
如上图所示$attrs表示没有继承数据的对象，格式为{属性名：属性值}。Vue2.4提供了$attrs , $listeners 来传递数据与事件，跨级组件之间的通讯变得更简单。

简单来说：$attrs与$listeners 是两个对象，$attrs 里存放的是父组件中绑定的非 Props 属性，$listeners里存放的是父组件中绑定的非原生事件。

## 1.4 Vue的响应式原理

**响应式原理的定义**：在改变数据的时候，视图会跟着更新。这意味着你只需要进行数据的管理，给我们搬砖提供了很大的便利。

VUE则是利用了Object.defineProperty的方法里面的setter 与getter方法的观察者模式来实现。

### 1.4.1 [Object.defineProperty](https://link.zhihu.com/?target=https%3A//developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperty)

这个方法就是在一个对象上定义一个新的属性，或者改变一个对象现有的属性，并且返回这个对象。里面有两个字段 set,get。顾名思义，set都是取设置属性的值，而get就是获取属性的值。

举个栗子：

*// 在对象中添加一个属性与存取描述符的示例*

**var** bValue;

**var** o **=** {};

Object.defineProperty(o, "b", {

get **:** **function**(){

console.log('监听正在获取b')

**return** bValue;

},

set **:** **function**(newValue){

console.log('监听正在设置b')

bValue **=** newValue;

},

enumerable **:** **true**,

configurable **:** **true**

});

o.b **=** 38;

console.log(o.b)

最终打印

监听正在设置b

监听正在获取b

38

从在上述栗子中，可以看到当我们对 o.b 赋值38的时候，就会调用[set函数](https://www.zhihu.com/search?q=set%E5%87%BD%E6%95%B0&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A88648401%7D)，这时候给bValue赋值，之后我们就可以通过o.b来获取这个值，这时候，[get函数](https://www.zhihu.com/search?q=get%E5%87%BD%E6%95%B0&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A88648401%7D)被调用。

掌握到这一步，我们已经可以实现一个极简的VUE双向绑定了。

**<**input type**=**"text" id**=**"txt" **/>**

**<**span id**=**"sp"**><**/span>

**<**script**>**

**var** txt **=** document.getElementById('txt'),

sp **=** document.getElementById('sp'),

obj **=** {}

*// 给对象obj添加msg属性，并设置setter访问器*

Object.defineProperty(obj, 'msg', {

*// 设置* [*obj.msg*](https://www.zhihu.com/search?q=obj.msg&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A88648401%7D) *当obj.msg反生改变时set方法将会被调用*

set**:** **function** (newVal) {

*// 当obj.msg被赋值时 同时设置给 input/span*

txt.value **=** newVal

sp.innerText **=** newVal

}

})

*// 监听文本框的改变 当文本框输入内容时 改变obj.msg*

txt.addEventListener('keyup', **function** (event) {

obj.msg **=** event.target.value

})

**<**/script>

VUE给data里所有的属性加上set,get这个过程就叫做**Reactive化**。

### 1.4.2 观察者模式

什么是观察者模式？它分为**注册环节跟发布环节**。

比如我去买芝士蛋糕，但是店家还没有做出来。这时候我又不想在店外面傻傻等，我就需要隔一段时间来回来问问蛋糕做好没，对于我来说是很麻烦的事情，说不定我就懒得买了。

店家肯定想要做生意，不想流失我这个吃货客户。于是，在蛋糕没有做好的这段时间，有客户来，他们就让客户把自己的电话留下，这就是观察者模式中的**注册环节**。然后蛋糕做好之后，一次性通知所有记录了的客户，这就是观察者的**发布环节**。

这里来简单实现一个观察者模式的类

**function** Observer() {

**this**.dep **=** [];

register(fn) {

**this**.dep.push(fn)

}

notify() {

**this**.dep.forEach(item => item())

}

}

**const** wantCake **=** **new** Oberver();

*// 每来一个顾客就注册一个想执行的函数*

wantCake.register(() => {'console.log("call daisy")'})

wantCake.register(() => {'console.log("call anny")'})

wantCake.register(() => {'console.log("call sunny")'})

*// 最后蛋糕做好之后，通知所有的客户*

wantCake.notify()

1.4.3 响应式原理

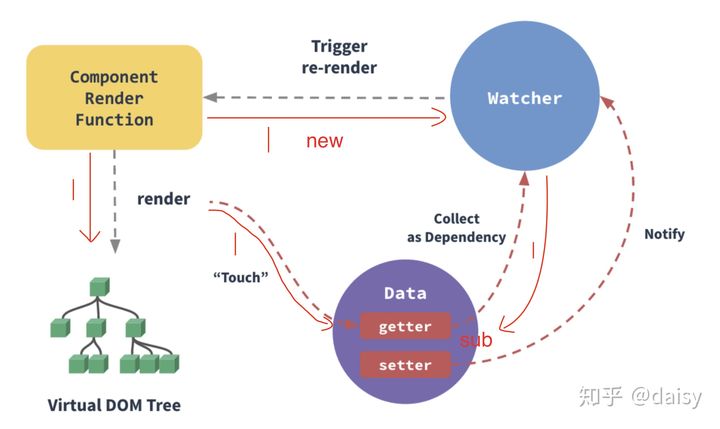
**1、init 阶段：** VUE 的 data的属性都会被reactive化，也就是加上 setter/[getter函数](https://www.zhihu.com/search?q=getter%E5%87%BD%E6%95%B0&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A88648401%7D)。

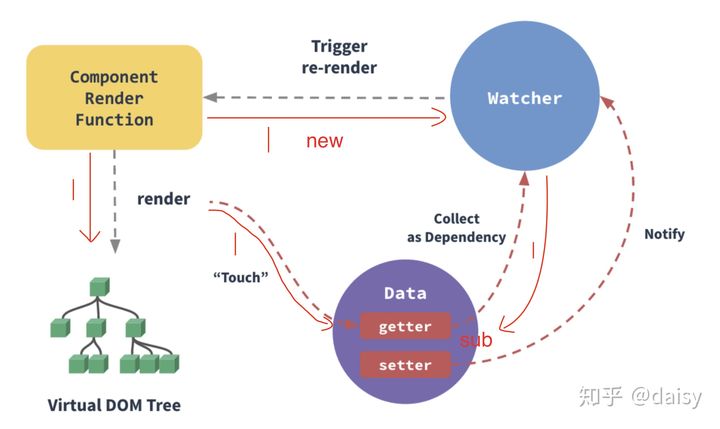
**2、mount 阶段：**mount 阶段的时候，会创建一个Watcher类的对象。这个Watcher实际上是连接Vue组件与Dep的桥梁。  
每一个Watcher对应一个vue component。

**3、更新阶段：**当blogTitle 发生改变的时候，就去调用Dep的[notify函数](https://www.zhihu.com/search?q=notify%E5%87%BD%E6%95%B0&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A88648401%7D),然后通知所有的Watcher调用update函数更新。

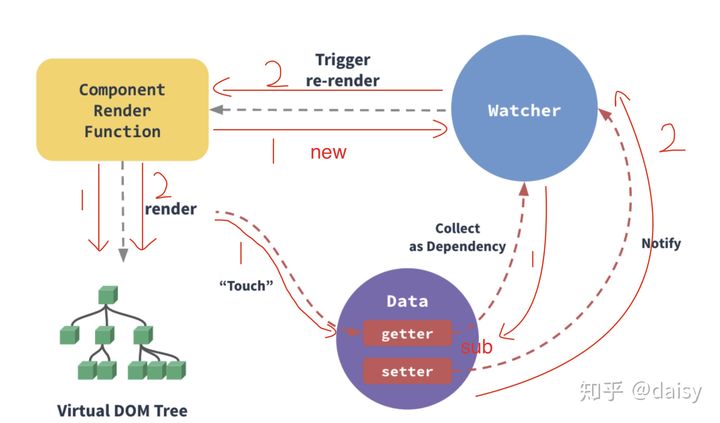
总结

最后，我们通过解释官方的图来做个总结。





**1、第一步：**组件初始化的时候，先给每一个Data属性都注册getter，setter，也就是reactive化。然后再new 一个自己的Watcher对象，此时watcher会立即调用组件的render函数去生成虚拟DOM。在调用render的时候，就会需要用到data的属性值，此时会触发getter函数，将当前的Watcher函数注册进sub里。



**2、第二步：**当data属性发生改变之后，就会遍历sub里所有的watcher对象，通知它们去重新渲染组件。

## 1.5 MVVM

