# 总结Vue3的13种传参通信方式，非常齐全

## 前言：

vue3出来很久了，也非常成熟了，平时项目只管用也没多想，直至今天想写一编关于vue3传参，然后我总结一下竟然总结出来13种方式那么多，13种我分别列了出来，由于vue3有两种setup写法，下面我将用**最简洁**的代码例子针对主流的`<script setup>`写法对每一种用法进行细说。如果有那里不对或者有补充欢迎大佬指导。

1. 父传子
2. 子传父
3. 兄弟组件传参（mitt）
4. `$attrs`
5. refs
6. v-model
7. provide/inject
8. 路由传参
9. vuex传参
10. pinia传参
11. 浏览器缓存
12. window
13. app.config.globalProperties

## 一、父传子

> **思路**：父组件通过冒号：绑定变量，然后子组件用const props = defineProps({})进行接收参数。

**父组件代码：**在第二行那里 :name="name" 把那么传给子组件

TypeScript

```
<template>
  <child :name="name"></child>
</template>

<script setup>
import { ref } from 'vue'
import child from './child.vue'

const name = ref('天天鸭')
</script>
```

**子组件代码：**const props = defineProps({})接收后直接在标签使用

TypeScript

```
<template>
  <div>{{ props.name }}</div>
</template>

<script setup>
import { defineProps } from 'vue'
const props = defineProps({
  name: {
    type: String,
    default: '',
  },
})
</script>
```

# 二、子传父

> 思路：子组件用 const emits = defineEmits(['触发的方法']) 注册某个在父组件的事件，然后通过 emits('触发的事件', 参数) 触发父组件事件并且带上参数。

**子组件代码：**注册 addEvent 事件后， 用 emits('addEvent', name.value) 触发父组件的 addEvent 事件

TypeScript

```
<template>
  <div ></div>
</template>

<script setup>
import { ref, defineEmits } from 'vue'

const name = ref('天天鸭')
const emits = defineEmits(['addEvent'])
const handleSubmit = () => {
  emits('addEvent', name.value)
}
</script>
```

**父组件代码：**触发 addEvent 事件后，在对应的方法里面直接能拿到传过来的参数

TypeScript

```
<template>
  <child @addEvent="handle"></child>
</template>

<script setup>
import { ref } from 'vue'
import child from './child.vue'

const handle = value => {
  console.log(value); // '天天鸭'
}
</script>
```

## 三、兄弟组件传参（mitt）

> 以前 vue2 是用 EventBus 事件总线跨组件实现兄弟组件通信的。但 vue3 中没有，所以 vue3 目前主流使用 mitt.js 插件来进行替代实现兄弟通信。

### 1、npm 包引入

TypeScript

```
npm install --save mitt
```

### 2、在 main.js 文件进行全局挂载，$bus 是自定义属性名

TypeScript

```
import mitt from "mitt"

const app = createApp(App)

app.config.globalProperties.$bus = new mitt()
```

### 3、传参出去的兄弟组件代码

TypeScript

```
<script setup>
    import mitt from 'mitt'
    const emitter = mitt()
    emitter.emit('自定义的事件名称','参数')
</script>
```

### 4、接收参数的兄弟组件代码

TypeScript

```
<script setup>
     import mitt from 'mitt'
     const emitter = mitt()
     emitter.on('自定义的事件名称', '参数' )
</script>
```

## 四、`$attrs`

> 注意： 以前在在 vue2 里面中除了 `$attrs`，还有 `$listeners`； 但 `vue3` 直接把 `$listeners` 合并到 `$attrs` 里面了。

> 简要说明： `$attrs` 主要作用是接收没在 `props` 里面定义，但父组件又传了过来的属性。看下面代码例子就好懂多了

**父组件代码：**传两个属性过去，一个在子组件 props 中，一个不在

TypeScript

```
<template>
  <child :name="天天鸭" data="PC9527" />
</template>

<script setup>
import child from './child.vue'
</script>
```

**子组件代码：** `$attrs` 接收到 props 以外的内容，所以用 `useAttrs()` 打印出来没有 name 只有 data

TypeScript

```
<template>
  <div>
    {{ props.name }}   // '天天鸭'
  </div>
</template>

<script setup>
import { defineProps, useAttrs } from 'vue'
const props = defineProps({
  name: {
    type: String
  }
})

const myattrs = useAttrs()
console.log(myattrs)   //  { "data": "PC9527" }
</script>
```

## 五、refs 传参

> 简单说明：父组件通过在子组件上定义 ref='ref 名称'，然后 const ref 名称 = ref(null)，就能通过 ref 名称操控子组件的属性和方法（子组件用 defineExpose 对外暴露才能被操控），具体看下面例子。

**父组件代码：**

TypeScript

```
<template>
  <child ref="myref"></child>
  <button @click="myClick">点击</button>
</template>

<script setup>
  import child from "./child.vue"
  import { ref } from "vue"
  const myref = ref(null)
  const myClick = () => {
      console.log(myref.value.name) // 直接获取到子组件的属性
      myref.value.chileMethod()     // 直接调用子组件的方法
  }
</script>
```

**子组件代码：**用 `defineExpose` 对外暴露才能被操控

TypeScript

```
<template>
  <div></div>
</template>

<script setup>
    import { defineExpose } from "vue"

    const chileMethod = () =>{
      console.log("我是方法")
    }
    const name = ref('天天鸭')

    defineExpose({     // 对外暴露
        name,
        chileMethod
    })
</script>
```

## 六、v-model

> 简单讲解： v-model其实语法糖，如下两行代码作用是一样，上面是下面的简写。

TypeScript

```
<chile v-model:title="title" />

<chile :title="title" @update:title="title = $event" />
```

**父组件代码：**直接使用v-model传参

TypeScript

```
<template>
  <child v-model:name="name" v-model:num="num"></child>
</template>

<script setup>
    import child from "./child.vue"
    import { ref, reactive } from "vue"
    const name = ref("天天鸭")
    const num = ref("2222")
</script>
```

**子组件代码：**通过`defineEmits`获取到然后用 emit("update:修改的属性", 修改的内容)进行修改父组件的内容，，注意：`update:`是固定写法。

TypeScript

```
<template>
  <button @click="myClick">点击</button>
</template>

<script setup>
  import { defineEmits } from "vue"
  const emit = defineEmits(["name","num"])

  // 子组件触发使用
  const myClick = () => {
      emit("update:name", "改个新名字")
      emit("update:num", "换个新号码")
  }
</script>
```

## v-model扩展：`defineModel()`：

> `defineModel()` 宏的简单说明：父子组件的数据双向绑定，不用 emit 和 props 的繁重代码
> 版本要求：必须要 `3.4+`

> 示例场景：父组件引入一个子组件弹窗，点击就 `父传子（props）` 弹出子组件弹窗，子组件里面有个按钮点击就 `子传父（emit）` 关闭

**父组件代码：**用 v-model 在子组件身上绑定 showDevice 属性，该属性用于通知子组件是否打开弹窗。

TypeScript

```
<template>
  <child v-if="showDevice" v-model="showDevice"></child>
</template>

<script setup>
    import child from "./child.vue"
    import { ref } from "vue"

    const showDevice = ref(false) // 控制子组件的显示和隐藏
</script>
```

**子组件代码：**如下的 `handleClickCancel` 方法，通过 `defineModel` 宏声明一个 model，点击按钮能直接通知父组件修改属性。

TypeScript

```
<template>
  <button @click="handleClickCancel">点击取消子组件弹窗</button>
</template>

<script setup>
  import { defineModel } from 'vue'
  const model = defineModel()                          // 写法一
  // const model = defineModel({ type: Boolean })    // 写法二 也可以用声明类型的方法

  const handleClickCancel = () => {
    model.value = false
  }
</script>
```

上面例子通过 `defineModel` 宏，直接不需要 props 和 emit 就实现了父子通信效果，非常简洁好用。

## 七、provide/inject

> 简单讲解：provide 和 inject 叫依赖注入，是 vue 官方提供的 API，它们可以实现多层组件传递数据，无论层级有多深，都可以通过这 API 实现。

**假设这是太老爷组件：**provide('名称', 传递的参数)向后代组件提供数据，只要是后代都能接收

TypeScript

```
<template>
  <div></div>
</template>

<script setup>
import { ref, provide } from 'vue'
const name = ref('天天鸭')
// 向后代组件提供数据，只要是后代都能接收
provide('name', name.value)
</script>
```

**最深层的孙组件：**无论层级多深，用 inject(接收什么参数) 进行接收即可

TypeScript

```
<template>
  <div>{{ name }}</div>
</template>

<script setup>
import { inject } from 'vue'
// 接收顶层组件的通信
const name = inject('name')
</script>
```

# 八、路由传参

> 简单讲解： 路由跳转事上参数也是传参的一种，而且传参方式还不止一种呢，下面细说。

### 1、query 传参

TypeScript

```typescript
// 传递方
const query = { id: 9527, name: '天天鸭' }
router.push({ path: '/user', query })

// 接收方
import { useRoute} from 'vue-router'
const route = useRoute()
console.log(route.query)
```

## 2、params传参

> 注意：4.1.4 (2022-08-22) 删除了 param 这种方式

TypeScript

```typescript
// 发送方
router.push({
    name: 'test',
    params: {
        name: '天天鸭'
    }
})

// 接收方
import { useRoute} from 'vue-router'
const route = useRoute()
console.log(route.params) // { name: '天天鸭' }
```

## 3、state传参

TypeScript

```typescript
// 发送方
const state= { name: '天天鸭' }
router.push({ path: '/user', state })

// 接收方直接使用
console.log(history?.state?.name)
```

## 九、vuex 传参

写过对应的文章，可以直接细看：对比 vuex 和 pinia 用法

## 十、pinia

写过对应的文章，可以直接细看：对比 vuex 和 pinia 用法

## 十一、浏览器缓存

> localStorage 和 sessionStorage： 这算是用的不多，但也是必用的一种通信方式了，下面看看区别：
> `sessionStorage`（临时存储）：为每一个数据源维持一个存储区域，在浏览器打开期间存在，包括页面重新加载
> `localStorage`（长期存储）：与 sessionStorage 一样，但是浏览器关闭后，数据依然会一直存在

下面直接上使用的语法。

TypeScript

```typescript
// 存储数据
localStorage.setItem('key', 'value');
sessionStorage.setItem('key', 'value');

// 获取数据
const valueFromLocalStorage = localStorage.getItem('key');
const valueFromSessionStorage = sessionStorage.getItem('key');

// 删除数据
localStorage.removeItem('key');
sessionStorage.removeItem('key');

// 清空所有数据
localStorage.clear();
sessionStorage.clear();
```

## 十二、通过 window 对象全局挂载全局对象或者属性

> 简单说明：直接用语法 window.name = '天天鸭' 定义然后 `全局` 通用即可。 `存放在内存刷新会清空` 。

**注意**：在 Vue 3 应用程序中，虽然可以直接将属性挂载到 window 对象上实现全局访问，但这并不是推荐的做法，因为 `直接修改全局对象可能会导致命名冲突、难以进行模块化管理以及不利于应用的封装与维护` 。

用法：直接定义，可以是属性也可以是对象

TypeScript

```typescript
window.duname = '天天鸭'
window.duObj = { test: '看看对象' }
```

引用：

```typescript
console.log( window.duname);    // 天天鸭
console.log( window.duObj);    // {test: '看看对象'}
```

## 十三、app.config.globalProperties

> 简单讲解：app.config.globalProperties 是 Vue 官方的一个 api，这是对 Vue 2 中 Vue.prototype 使用方式的一种替代，Vue.prototype 法在 Vue3 已经不存在了。与任何全局的东西一样，应该谨慎使用。

用法示例：在 main.js 文件挂载一个全局的 msg 属性

```typescript
import { createApp } from 'vue'

const app = createApp(App)

app.config.globalProperties.msg = 'hello'
```

在其它页面使用 `getCurrentInstance()` 获取

TypeScript

```typescript
import { getCurrentInstance } from "vue";

const { proxy } = getCurrentInstance() // 使用proxy，类似于vue2的this

console.log(proxy.msg);     // hello
```

## 总结：

空闲想总结一下，我尽量用最简洁易懂的方式写了，如果有什么写错了或者不够明细都欢迎大家指出。这也当笔记用了，以后忘记用法回来看一眼。