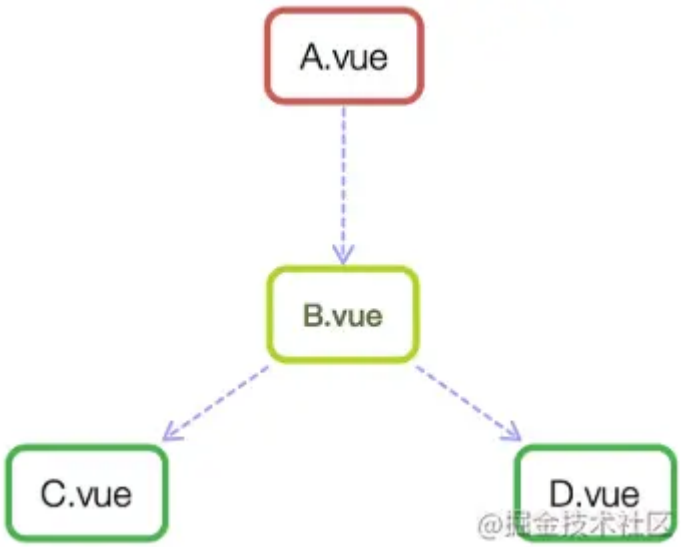
Vue 组件间通信六种方式（完整版）

> 链接：https://juejin.cn/post/6844903845642911752#comment。这个博主叫浪里行舟，写的文章是真好，我看了好多篇他的文章了，通俗易懂，非常棒！！

前言

组件是 vue.js最强大的功能之一，而组件实例的作用域是相互独立的，这就意味着不同组件之间的数据无法相互引用。一般来说，组件可以有以下几种关系：



如上图所示，A 和 B、B 和 C、B 和 D 都是父子关系，C 和 D 是兄弟关系，A 和 C 是隔代关系（可能隔多代）。

针对不同的使用场景，如何选择行之有效的通信方式？这是我们所要探讨的主题。本文总结了vue组件间通信的几种方式，如props、`$emit`/`$on`、vuex、`$parent`/`$children`、`$attrs`/`$listeners`和provide/inject，以通俗易懂的实例讲述这其中的差别及使用场景，希望对小伙伴有些许帮助。

本文的代码请猛戳github博客，纸上得来终觉浅，大家动手多敲敲代码！

方法一、`props`/`$emit`

父组件A通过props的方式向子组件B传递，B to A 通过在 B 组件中 $emit, A 组件中 v-on 的方式实现。

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
  }
}
复制代码
//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>
复制代码
```

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.$emit("titleChanged","子向父组件传值");//自定义事件   传递值“子向父组件传值”
    }
  }
}
</script>
```
复制代码
```
// 父组件
<template>
  <div id="app">
    <app-header v-on:titleChanged="updateTitle" ></app-header>//与子组件titleChanged自定义事件保持一致
   // updateTitle($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>
```
复制代码

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。

方法二、`$emit/$on`

这种方法通过一个空的Vue实例作为中央事件总线（事件中心），用它来触发事件和监听事件,巧妙而轻量地实现了任何组件间的通信，包括父子、兄弟、跨级。当我们的项目比较大时，可以选择更好的状态管理解决方案vuex。

1.具体实现方式：

```
    var Event=new Vue();
    Event.$emit(事件名,数据);
    Event.$on(事件名,data => {});
复制代码
```

2.举个例子

假设兄弟组件有三个，分别是A、B、C组件，C组件如何获取A或者B组件的数据

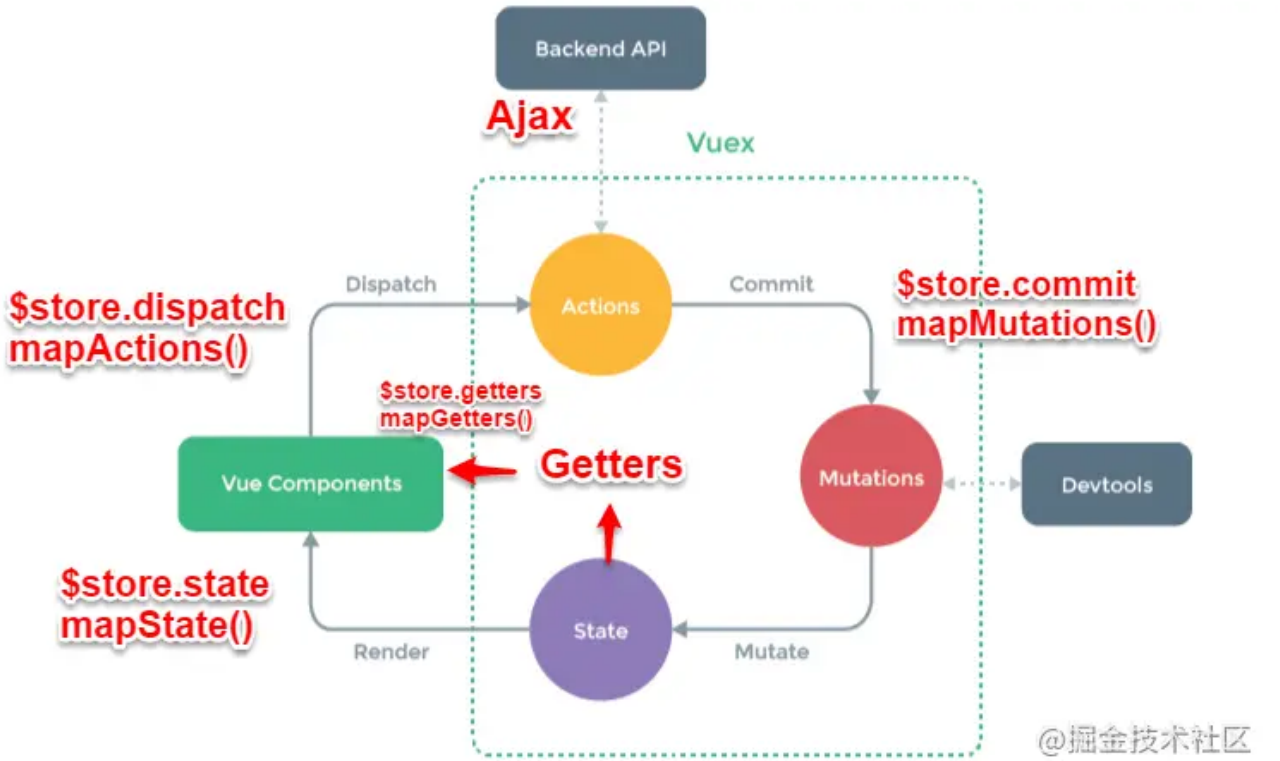
```
<div id="itany">
    <my-a></my-a>
    <my-b></my-b>
    <my-c></my-c>
</div>
<template id="a">
  <div>
    <h3>A组件: {{name}}</h3>
    <button @click="send">将数据发送给C组件</button>
  </div>
</template>
<template id="b">
  <div>
    <h3>B组件: {{age}}</h3>
    <button @click="send">将数组发送给C组件</button>
  </div>
</template>
<template id="c">
  <div>
    <h3>C组件: {{name}}, {{age}}</h3>
  </div>
</template>
<script>
var Event = new Vue();//定义一个空的Vue实例
var A = {
    template: '#a',
    data() {
      return {
        name: 'tom'
      }
    },
    methods: {
      send() {
        Event.$emit('data-a', this.name);
      }
    }
}
var B = {
    template: '#b',
    data() {
      return {
```

```
          age: 20
        }
      },
      methods: {
        send() {
          Event.$emit('data-b', this.age);
        }
      }
  }
  var C = {
      template: '#c',
      data() {
        return {
          name: '',
          age: ""
        }
      },
      mounted() {//在模板编译完成后执行
       Event.$on('data-a',name => {
            this.name = name;//箭头函数内部不会产生新的this，这边如果不用=>,this指代Event
       })
       Event.$on('data-b',age => {
            this.age = age;
       })
      }
  }
  var vm = new Vue({
      el: '#itany',
      components: {
        'my-a': A,
        'my-b': B,
        'my-c': C
      }
  });
  </script>
```

复制代码

`$on` 监听了自定义事件 data-a和data-b，因为有时不确定何时会触发事件，一般会在 mounted 或 created 钩子中来监听。

方法三、vuex



1.简要介绍Vuex原理

Vuex实现了一个单向数据流，在全局拥有一个State存放数据，当组件要更改State中的数据时，必须通过Mutation进行，Mutation同时提供了订阅者模式供外部插件调用获取State数据的更新。而当所有异步操作(常见于调用后端接口异步获取更新数据)或批量的同步操作需要走Action，但Action也是无法直接修改State的，还是需要通过Mutation来修改State的数据。最后，根据State的变化，渲染到视图上。

2.简要介绍各模块在流程中的功能：

- Vue Components：Vue组件。HTML页面上，负责接收用户操作等交互行为，执行dispatch方法触发对应action进行回应。
- dispatch：操作行为触发方法，是唯一能执行action的方法。
- actions：**操作行为处理模块,由组件中的$store.dispatch('action 名称', data1)来触发。然后由commit()来触发mutation的调用 , 间接更新 state。**负责处理Vue Components接收到的所有交互行为。包含同步/异步操作，支持多个同名方法，按照注册的顺序依次触发。向后台API请求的操作就在这个模块中进行，包括触发其他action以及提交mutation的操作。该模块提供了Promise的封装，以支持action的链式触发。
- commit：状态改变提交操作方法。对mutation进行提交，是唯一能执行mutation的方法。
- mutations：**状态改变操作方法，由actions中的commit('mutation 名称')来触发。**是Vuex修改state的唯一推荐方法。该方法只能进行同步操作，且方法名只能全局唯一。操作之中会有一些hook暴露出来，以进行state的监控等。
- state：页面状态管理容器对象。集中存储Vue components中data对象的零散数据，全局唯一，以进行统一的状态管理。页面显示所需的数据从该对象中进行读取，利用Vue的细粒度数据响应机制来进行高效的状态更新。
- getters：state对象读取方法。图中没有单独列出该模块，应该被包含在了render中，Vue Components通过该方法读取全局state对象。

3.Vuex与localStorage

vuex 是 vue 的状态管理器，存储的数据是响应式的。但是并不会保存起来，刷新之后就回到了初始状态，**具体做法应该在vuex里数据改变的时候把数据拷贝一份保存到localStorage里面，刷新之后，如果localStorage里有保存的数据，取出来再替换store里的state。**

```
let defaultCity = "上海"
try {   // 用户关闭了本地存储功能，此时在外层加个try...catch
  if (!defaultCity){
    defaultCity = JSON.parse(window.localStorage.getItem('defaultCity'))
  }
}catch(e){}
export default new Vuex.Store({
  state: {
    city: defaultCity
  },
  mutations: {
    changeCity(state, city) {
      state.city = city
      try {
      window.localStorage.setItem('defaultCity', JSON.stringify(state.city));
      // 数据改变的时候把数据拷贝一份保存到localStorage里面
      } catch (e) {}
    }
  }
})
复制代码
```

这里需要注意的是：由于vuex里，我们保存的状态，都是数组，而localStorage只支持字符串，所以需要用JSON转换：

```
JSON.stringify(state.subscribeList);   // array -> string
JSON.parse(window.localStorage.getItem("subscribeList"));    // string -> array
复制代码
```

方法四、`$attrs`/`$listeners`

1.简介

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 inheritAttrs 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：

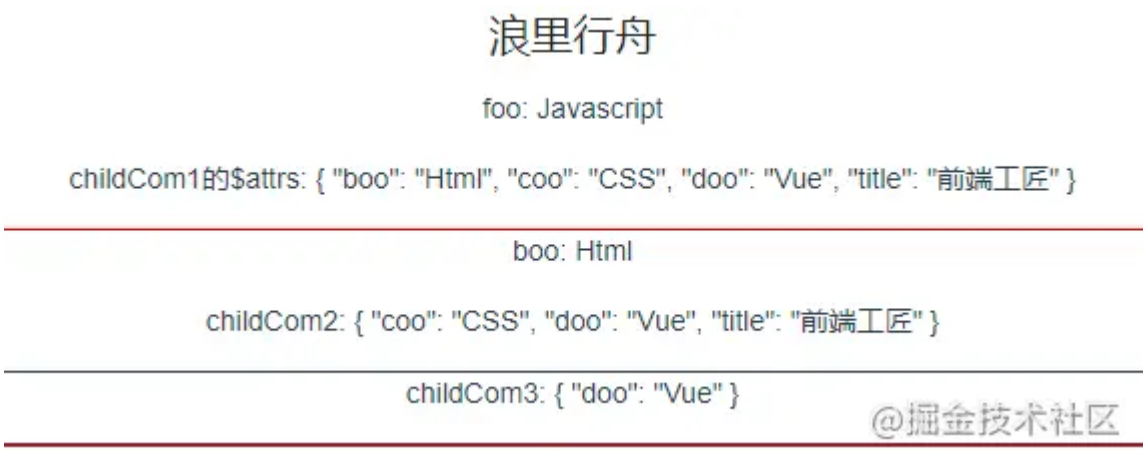
```
// index.vue
<template>
  <div>
    <h2>浪里行舟</h2>
    <child-com1
      :foo="foo"
      :boo="boo"
      :coo="coo"
      :doo="doo"
      title="前端工匠"
    ></child-com1>
  </div>
</template>
<script>
const childCom1 = () => import("./childCom1.vue");
export default {
  components: { childCom1 },
  data() {
    return {
      foo: "Javascript",
      boo: "Html",
      coo: "CSS",
      doo: "Vue"
    };
  }
};
</script>
```
复制代码
```
// 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>
```
复制代码

```
// 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>
```

复制代码

```
// childCom3.vue
<template>
  <div class="border">
    <p>childCom3: {{ $attrs }}</p>
  </div>
</template>
<script>
export default {
  props: {
    coo: String,
    title: String
  }
};
</script>
```

复制代码



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。

方法五、provide/inject

1.简介

Vue2.2.0新增API,这对选项需要一起使用，**以允许一个祖先组件向其所有子孙后代注入一个依赖，不论组件层次有多深，并在起上下游关系成立的时间里始终生效。**一言而蔽之：祖先组件中通过provider来提供变量，然后在子孙组件中通过inject来注入变量。 **provide / inject API 主要解决了跨级组件间的通信问题，不过它的使用场景，主要是子组件获取上级组件的状态，跨级组件间建立了一种主动提供与依赖注入的关系。**

2.举个例子

假设有两个组件： A.vue 和 B.vue，B 是 A 的子组件

```
// A.vue
export default {
  provide: {
    name: '浪里行舟'
  }
}
复制代码
// B.vue
export default {
  inject: ['name'],
  mounted () {
    console.log(this.name);  // 浪里行舟
  }
}
复制代码
```

可以看到，在 A.vue 里，我们设置了一个 **provide: name**，值为 浪里行舟，它的作用就是将 **name** 这个变量提供给它的所有子组件。而在 B.vue 中，通过 `inject` 注入了从 A 组件中提供的 **name** 变量，那么在组件 B 中，就可以直接通过 **this.name** 访问这个变量了，它的值也是 浪里行舟。这就是 provide / inject API 最核心的用法。

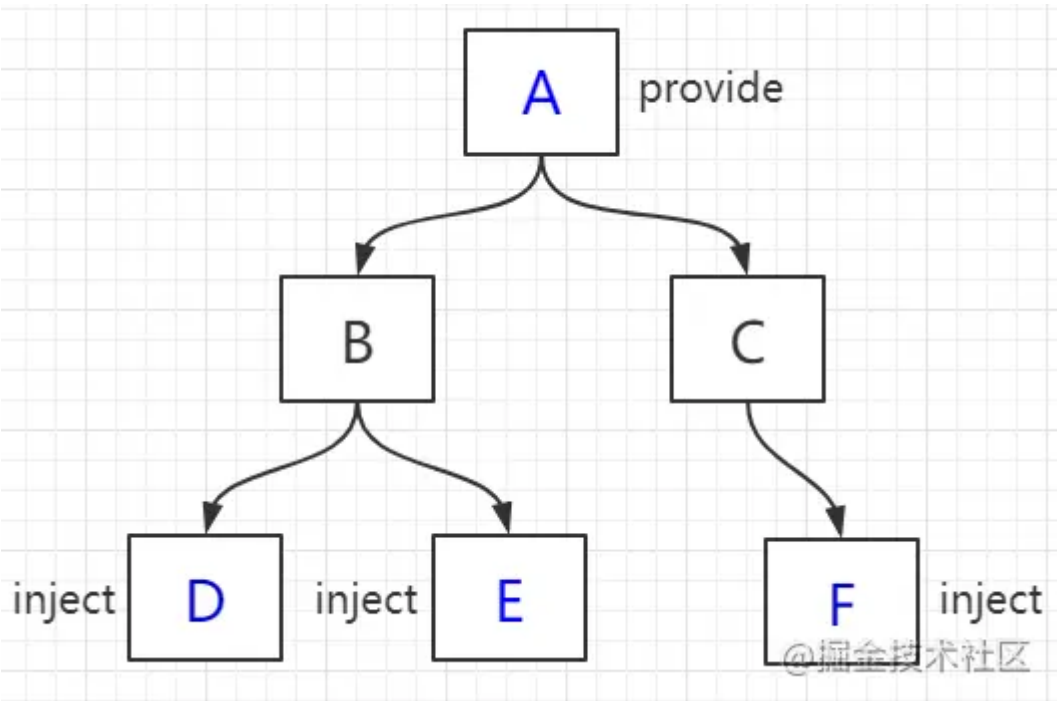
需要注意的是：**provide 和 inject 绑定并不是可响应的。这是刻意为之的。然而，如果你传入了一个可监听的对象，那么其对象的属性还是可响应的**----vue官方文档 所以，上面 A.vue 的 name 如果改变了，B.vue 的 this.name 是不会改变的，仍然是 浪里行舟。

3.provide与inject 怎么实现数据响应式

一般来说，有两种办法：

- provide祖先组件的实例，然后在子孙组件中注入依赖，这样就可以在子孙组件中直接修改祖先组件的实例的属性，不过这种方法有个缺点就是这个实例上挂载很多没有必要的东西比如props，methods
- 使用2.6最新API Vue.observable 优化响应式 provide(推荐)

我们来看个例子：孙组件D、E和F获取A组件传递过来的color值，并能实现数据响应式变化，即A组件的color变化后，组件D、E、F会跟着变（核心代码如下：）




```
// A 组件
<div>
      <h1>A 组件</h1>
      <button @click="() => changeColor()">改变color</button>
      <ChildrenB />
      <ChildrenC />
</div>
......
  data() {
    return {
      color: "blue"
    };
  },
  // provide() {
  //   return {
  //     theme: {
  //       color: this.color //这种方式绑定的数据并不是可响应的
  //     } // 即A组件的color变化后，组件D、E、F不会跟着变
```

```
  //   };
  // },
  provide() {
    return {
      theme: this//方法一：提供祖先组件的实例
    };
  },
  methods: {
    changeColor(color) {
      if (color) {
        this.color = color;
      } else {
        this.color = this.color === "blue" ? "red" : "blue";
      }
    }
  }
  // 方法二:使用2.6最新API Vue.observable 优化响应式 provide
  // provide() {
  //   this.theme = Vue.observable({
  //     color: "blue"
  //   });
  //   return {
  //     theme: this.theme
  //   };
  // },
  // methods: {
  //   changeColor(color) {
  //     if (color) {
  //       this.theme.color = color;
  //     } else {
  //       this.theme.color = this.theme.color === "blue" ? "red" : "blue";
  //     }
  //   }
  // }
```

复制代码

```
// F 组件
<template functional>
  <div class="border2">
    <h3 :style="{ color: injections.theme.color }">F 组件</h3>
  </div>
</template>
<script>
export default {
  inject: {
    theme: {
      //函数式组件取值不一样
      default: () => ({})
    }
  }
};
</script>
```

复制代码

虽说provide 和 inject 主要为高阶插件/组件库提供用例，但如果你能在业务中熟练运用，可以达到事半功倍的效果！

方法六、`$parent` / `$children` 与 `ref`

- `ref`：如果在普通的 DOM 元素上使用，引用指向的就是 DOM 元素；如果用在子组件上，引用就指向组件实例
- `$parent` / `$children`：访问父 / 子实例

需要注意的是：这两种都是直接得到组件实例，使用后可以直接调用组件的方法或访问数据。我们先来看个用 `ref` 来访问组件的例子：

```
// component-a 子组件
export default {
  data () {
    return {
      title: 'Vue.js'
    }
  },
  methods: {
    sayHello () {
      window.alert('Hello');
    }
  }
}
复制代码
// 父组件
<template>
  <component-a ref="comA"></component-a>
</template>
<script>
  export default {
    mounted () {
      const comA = this.$refs.comA;
      console.log(comA.title);  // Vue.js
      comA.sayHello();  // 弹窗
    }
  }
</script>
复制代码
```

不过，**这两种方法的弊端是，无法在跨级或兄弟间通信。**

```
// parent.vue
<component-a></component-a>
<component-b></component-b>
<component-b></component-b>
复制代码
```

我们想在 component-a 中，访问到引用它的页面中（这里就是 parent.vue）的两个 component-b 组件，那这种情况下，就得配置额外的插件或工具了，比如 Vuex 和 Bus 的解决方案。

总结

常见使用场景可以分为三类：

- 父子通信：

父向子传递数据是通过 props，子向父是通过 events（`$emit`）；通过父链 / 子链也可以通信（`$parent` / `$children`）；ref 也可以访问组件实例；provide / inject API；`$attrs/$listeners`

- 兄弟通信：

Bus；Vuex

- 跨级通信：

Bus；Vuex；provide / inject API、`$attrs/$listeners`

参考文章

- 珠峰架构课(强烈推荐)
- Vue.js 组件精讲
- Vue.js 官方文档
- Vue开发实战
- Vuex数据本地储存
- Vuex框架原理与源码分析
- Vue 组件通信方式全面详解

我自己补充

使用props用于子传父

- `props`也可以用于子组件向父组件传递数组，不过父组件要先像子组件传递一个函数，然后子组件利用传递过来的函数再将数据传递给父组件。

`App.vue`——父组件(部分代码)

```
<MyHeader :addTodo="addTodo" />
data() {
    return {
      todos: [
        { id: "001", title: "吃饭", isCompleted: true },
        { id: "002", title: "睡觉", isCompleted: false },
        { id: "003", title: "打豆豆", isCompleted: true },
        //这id要用字符串，因为你用number的话，在JS里面number是有尽头的
        //我是爹，我要把我里面的数据传给儿子(微信给儿子钱，儿子要写props确认)

        // 数据在App里面，那么所有对数据的增删改操作也应该在App里进行才是
      ],
    };
  },
```

```
    methods: {
    //添加一个todo
    addTodo(todoObj) {
      console.log("我是App组件，我收到了数据：", todoObj);
      this.todos.unshift(todoObj);
      //unshift是数组头部添加
    },
    },
  },
```

`MyHeader.vue`——子组件(部分代码)

```
  props: ["addTodo"],
  //父亲传的很高级，传的是一个函数

    data() {
    return {
      title: "",
    };
  },
  methods: {
    //方法1来得到用户的输入
    add(event) {
      //   别人什么也没输入按回车，不让他添加
      //进行数据的校验
      if (!this.title.trim()) return alert("输入不能为空");

      //包装成一个对象
      //   console.log(event);
      //   console.log(event.target.value);
      // 使用事件对象来获取用户的输入
      //   console.log(this.title);

      console.log(todoObj);
      //   有一个问题：保存一堆要做的事情的todos数据是在MyList组件中的
      //   而你输出的todoObj是位于MyHeader组件中的
      //   以目前我们的知识量，无法将MyHeader组件中的todoObj传给MyList组件
      //   以目前我们的知识量，想从组件外部给组件里面携带一些数据进去，只能有一个写法：props
      //   但是MyHeader和MyList不是包裹关系，不是父子，它们是兄弟关系
      //   APP组件是宰相，给谁传东西都能传。(直接在App组件里面写<MyList a="1"></MyList>即可)
      //通知App组件去添加一个对象
      this.addTodo(todoObj);
      //这步关键：父组件传过来一个函数，子组件再调用这个函数，顺便把数据传给父组件
    },
  },
```

消息订阅与发布

下载一个库，比如`pubsub-js`,它可以做到和全局事件总线相同的功能。学生向学校发送消息：

`Student.vue`部分代码(CSS没写):

```
<template>
  <div class="student">
    <h2>学生姓名: {{ name }}</h2>
    <h2>学生性别: {{ sex }}</h2>
    <button @click="sendStudentName">把学生名给School组件</button>
  </div>
</template>

<script>
import pubsub from "pubsub-js";
export default {
  name: "Student",
  data() {
    return {
      name: "张三",
      sex: "男",
    };
  },
  methods: {
    //   Student是提供消息的一方，是发布消息的
    sendStudentName() {
      //   this.$bus.$emit("hello", this.name);
      //   点击按钮后就会发布消息
      pubsub.publish("hello", 666);
      //发送消息名为hello的消息，并且携带的数据为666
    },
  },
};
</script>
```

`School.vue` 部分代码(CSS没写)

```
<template>
  <div class="school">
    <h2>学校名称: {{ name }}</h2>
    <h2>学校地址: {{ address }}</h2>
  </div>
</template>

<script>
import pubsub from "pubsub-js";
// pubsub它是一个对象，它身上有很多有用的方法
export default {
  name: "School",
  data() {
    return {
      name: "尚硅谷",
      address: "北京",
    };
  },
  mounted() {
    //School是需要数据，需要消息的一方，订阅消息，消息名随便写，这里是hello
```

```
    // pubsub.subscribe("hello", (a, b) => {
    this.pubId = pubsub.subscribe("hello", (msg, data) => {
      console.log(this);
      //如果你写普通函数的话，这个this是undefined，如果是箭头函数的话，才是vue实例对象
      // 他这个库的设计有点像定时器，你每次订阅的时候都能够收到订阅的id,而且每次订阅id都不一样
      //当你想取消订阅的时候，作者让你通过id来取消
      console.log("有人发布了hello消息，hello消息的订阅成功", msg, data);
      //   msg是hello，是消息名，data才是真正的数据
      //   这个库的作者就是这么设计的，有2个参数，都要写
    });
  },
  beforeDestroy() {
    //this.$bus.$off('hello')
    pubsub.unsubscribe(this.pubId);
  },
};
</script>
```

作用域插槽

`Category.vue` 部分代码(CSS没写):

```
<template>
  <div class="category">
    <h3>{{ title }}分类</h3>
    <!-- <ul>
      <li v-for="(item, index) in games" :key="index">{{ item }}</li>
    </ul> -->

    <!-- <slot>我是一些默认值，当使用者没有传递具体结构时，我会出现</slot> -->
    <!-- Category组件数据在这，根据数据生成的代码也在这，我们可以不使用插槽了 -->
    <!-- 接下来我提一个需求：数据都是这4个游戏，但是根据这4个游戏生成的结构不一样(由使用者来定)，分别是：无序列表、有序列表、h4-->
    <!-- 一个比较笨的方法，App组件传一个type数据，根据type数据里面内容不同进行条件渲染，这是可行的，但是弊端：你要写很多判断,不灵活-->
    <!-- 此时你就可以借助作用域插槽了 -->
    <slot :youxi="games" msg="hello">我是默认的一些内容</slot>
    <!-- <slot name="haha" :youxi="games" msg="hello">我是默认的一些内容</slot> -->
    <!-- 作用域插槽也是可以有名字的！！！ -->
    <!-- 它就是把这个youxi传给了插槽的使用者 -->
    <!--这是一个插槽，你写插槽的目的？等着往里面塞结构。那你注意：谁往你里面塞结构，那么这个youxi就传给谁 -->
    <!-- 这个是默认插槽 -->
    <!-- 插槽的作用就是：插槽的使用者往插槽里面塞东西，但是作用域插槽，有种感觉让数据流逆着过去了-->
    <!-- App是插槽的使用者！！！ -->
  </div>
</template>

<script>
export default {
  name: "Category",
  props: ["title"],
  data() {
```

```
    return {
      // 数据在插槽定义的那个组件(Category)里面
      //    但是用这个数据生成的结构是由插槽的使用者(App)而决定的，你说ul就ul,你说ol就ol,你说h4就h4

      //    插槽的作用：让父组件可以向子组件指定位置插入html结构，这也是一种组件间通信方式，适用于父组件=》子组件
      //    您能在一个组件结构中写另外一个组件的标签，这就是父子
      games: ["红色警戒", "穿越火线", "劲舞团", "超级玛丽"],
    };
  },
};
</script>
```

`App.vue` 部分代码(CSS没写)

```
<template>
  <div class="container">
    <!-- App组件是Category组件的使用者，并且数据也是存放在App里面的 -->
    <!-- 现在，我把数据不放在App里面了，放到Category里面了--此时插槽就没有必要使用了 -->
    <!-- <Category title="游戏" type="ul"> </Category>
    <Category title="游戏" type="ol"> </Category>
    <Category title="游戏" type="h4"> </Category> -->

    <!-- App是使用者，它可以决定我可以把什么样的结构给你 -->
    <!-- 数据在Category里，我们通过作用域插槽来拿到Category里的作用域，并且解析后返回给它 -->
    <Category title="游戏">
      <!-- App组件里面没有games，数据都在Category里面，这就像JS里面的作用域问题了 -->
      <!-- 我有点疑惑，我把数据放到App里面就可以了啊。。。。。。是不是实际中不行啊。。。 -->
      <!-- 有些场景：数据不在你(App)这里，而且数据也不允许给你，你就只能通过作用域插槽了 -->
      <!-- 要求：【必须】包裹一层template -->
      <template scope="atguigu">
        <!-- 这个只能叫scope,中文叫作用，注意：不是scoped!!! -->
        <!-- {{ atguigu }} -->
        <!-- {{ atguigu.youxi }} -->
        <ul>
          <li v-for="(item, index) in atguigu.youxi" :key="index">{{ item }}</li>
        </ul>
      </template>
    </Category>
    <Category title="游戏">
      <!-- <template scope="atguigu"> -->
      <!-- 可以使用解构赋值，简化代码 -->
      <template scope="{youxi}">
        <ol>
          <li style="color: red" v-for="(item, index) in youxi" :key="index">
            {{ item }}
          </li>
        </ol>
        <!-- <h4>{{ atguigu.msg }}</h4> -->
        <!-- <h4>{{ msg }}</h4> -->
      </template>
    </Category>
```

```vue
    <Category title="游戏">
      <!-- <template scope="{youxi}"> -->
      <!-- 另外一种写法：slot-scope，效果一样，新旧API的问题，slot-scope是新的API -->
      <template slot-scope="{ youxi }">
        <h4 v-for="(item, index) in youxi" :key="index">{{ item }}</h4>
      </template>
    </Category>
  </div>
</template>
<script>
import Category from "./components/Category.vue";
export default {
  name: "App",
  components: { Category },
  //   data() {
  //     return {
  //       games: ["红色警戒", "穿越火线", "劲舞团", "超级玛丽"],
  //     };
  //   },
};
</script>
```