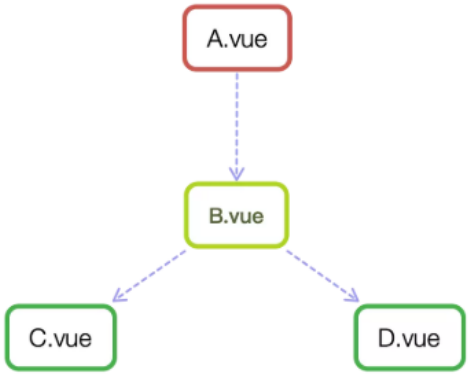
VUE组件间通信与组件编码规范

一、组件间通信

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：



组件关系：父子（A与B）、兄弟（C与D）、隔代（A与C、D）

通信方式：props、$emit、vuex、$parent / $children、provide/inject和中央事件总线EventBus

方法一、props/$emit

1.父传子：父组件A通过props的方式向子组件B传递，

```
//App.vue父组件
<template>
  <div id="app">
    <users v-bind:sub-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 subUsers">{{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:{
     subUsers:{           //这个就是父组件中子标签自定义名字
       type:Array,
       required:true
     }
   }
 }
 </script>

```

2.子传父：B to A 通过在 B 组件中 $emit, A 组件中 v-on 的方式实现。

```
// 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
   // 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){
      this.title = e;
    }
  },
  components:{
   "app-header":Header,
  }
}
</script>

```

方法二、$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

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。

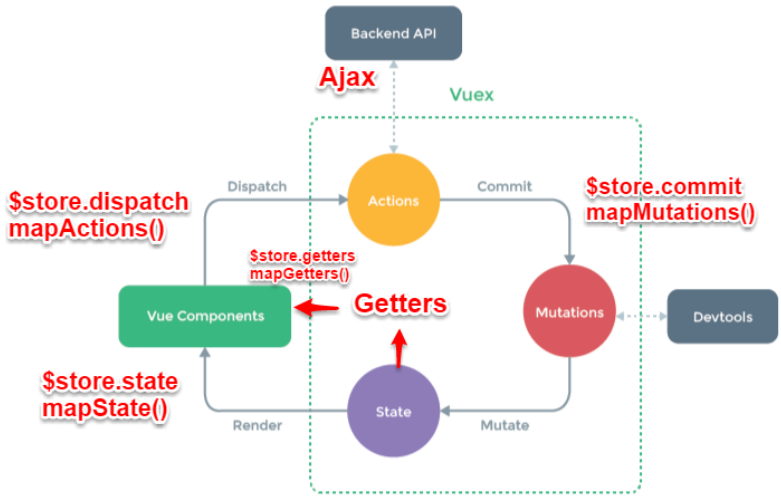
```
// component-a 子组件
export default {
  data () {
    return {
      title: 'Vue.js'
    }
  },
  methods: {
    sayHello () {
       window.alert('Hello');
    }
  }
}

```

```
// 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>
```

```
14
```

无法在跨级或兄弟间通信

方法三、vuex



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。

方法四、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。

简言之：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

A.vue 和 B.vue，B 是 A 的子组件

```
// A.vue
export default {
  provide: {
    name: '组件通信'
  }
}

```

```
// B.vue
export default {
  inject: ['name'],
  mounted () {
    console.log(this.name);  // 组件通信
  }
```

```
}
```

可以看到，在 A.vue 里，我们设置了一个 provide: name，值为 “组件通信”，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 “组件通信”。这就是 provide / inject API 最核心的用法。

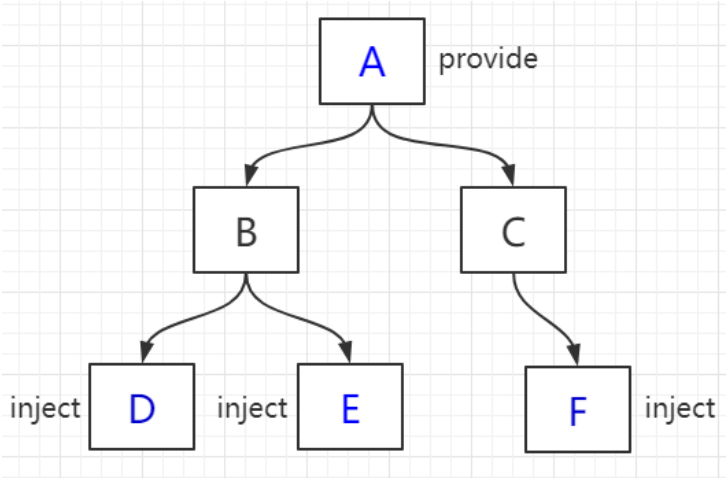
需要注意的是：provide 和 inject 绑定并不是可响应的。这是刻意为之的。然而，如果你传入了一个可监听的对象，那么其对象的属性还是可响应的----vue官方文档 所以，上面 A.vue 的 name 如果改变了，B.vue 的 this.name 是不会改变的，仍然是 浪里行舟。

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)

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不会跟着变（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";
//     }
//   }
// }

```

```
// 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>

```

这么做也是有明显的缺点的，在任意层级都能访问导致数据追踪比较困难，不知道是哪一个层级声明了这个或者不知道哪一层级或若干个层级使用了，因此这个属性通常并不建议使用,能用vuex的使用vuex，不能用的多传参几层，但是在做组件库开发时，不对vuex进行依赖，且不知道用户使用环境的情况下可以很好的使用

方法五、中央事件总线 EventBus（$emit/$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 => {});
```

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>

```

3.关闭监听

多次打开一个组件会重复触发$on监听事件，所以需要在组件注销之前关闭监听

```
beforeDestroy () {
  Event.$off('data-a', this.myhandle)
  Event.$off('data-b', this.myhandle)
```

```
4 }
```

二、组件编码规范

https://github.com/pablohpsilva/vuejs-component-style-guide/blob/master/README-CN.md#%E5%9F%BA%E4%BA%8E%E6%A8%A1%E5%9D%97%E5%BC%80%E5%8F%91