链滴

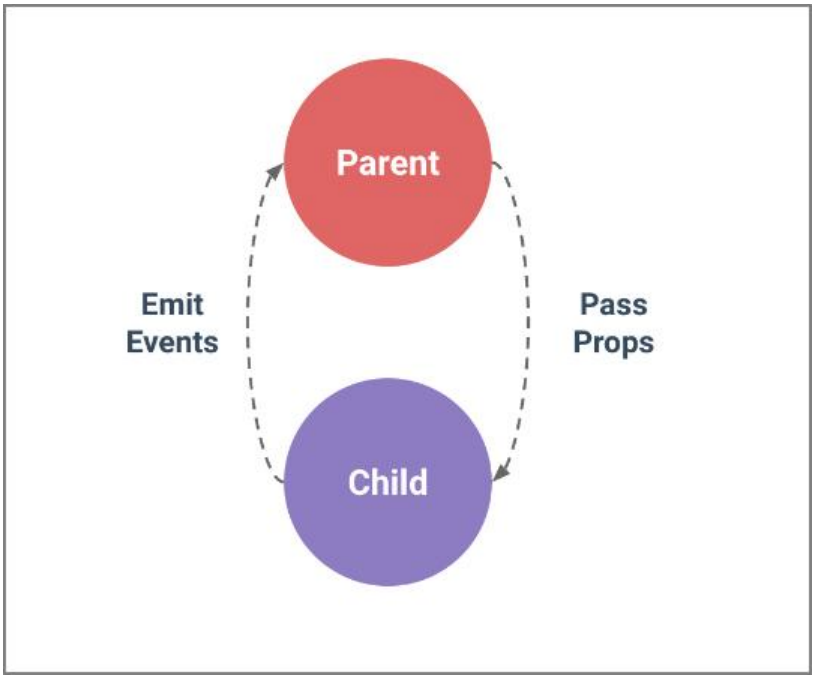
Vue 事件总线（EventBus）

作者：qycx
原文链接：https://ld246.com/article/1683877167399
来源网站：链滴

许多现代JavaScript框架和库的核心概念是能够将数据和UI封装在模块化、可重用的组件中。这对于发人员可以在开发整个应用程序时避免使用编写大量重复的代码。虽然这样做非常有用，但也涉及到件之间的数据通讯。在Vue中同样有这样的概念存在。通过前面一段时间的学习，Vue组件数据通讯常会有父子组件，兄弟组件之间的数据通讯。也就是说在Vue中组件通讯有一定的原则。

父子组件通讯原则

为了提高组件的独立性与重用性，父组件会通过props向下传数据给子组件，当子组件有事情要告诉组件时会通过$emit事件告诉父组件。如此确保每个组件都是独立在相对隔离的环境中运行，可以大提高组件的维护性。



在《Vue组件通讯》一文中有详细介绍过这部分。但这套通讯原则对于兄弟组件之间的数据通讯就有定的诟病。当然，在Vue中有其他的方式来处理兄弟组件之间的数据通讯，比如Vuex这样的库。但在多情况之下，咱们的应用程序不需要类似Vuex这样的库来处理组件之间的数据通讯，而可以考虑Vue的 **事件总线** ，即 **EventBus** 。

接下来的内容，就是来一起学习Vue中的EventBus相关的知识点。

EventBus的简介

EventBus又称为事件总线。在Vue中可以使用EventBus来作为沟通桥梁的概念，就像是所有组件共相同的事件中心，可以向该中心注册发送事件或接收事件，所以组件都可以上下平行地通知其他组件但也就是太方便所以若使用不慎，就会造成难以维护的灾难，因此才需要更完善的Vuex作为状态管中心，将通知的概念上升到共享状态层次。

如何使用EventBus

在Vue的项目中怎么使用EventBus来实现组件之间的数据通讯呢？具体可以通过下面几个步骤来完成。

初始化

首先你需要做的是创建事件总线并将其导出，以便其它模块可以使用或者监听它。我们可以通过两种式来处理。先来看第一种，新创建一个.js文件，比如event-bus.js：

```
// event-bus.js

import Vue from 'vue'
export const EventBus = new Vue()
```

你需要做的只是引入 Vue 并导出它的一个实例（在这种情况下，我称它为 EventBus ）。实质上它一个不具备 DOM 的组件，它具有的仅仅只是它实例方法而已，因此它非常的轻便。

另外一种方式，可以直接在项目中的main.js初始化EventBus：

```
// main.js
Vue.prototype.$EventBus = new Vue()
```

注意，这种方式初始化的EventBus是一个 **全局的事件总线** 。稍后我们会花点时间专门聊一聊全局的件总线。

现在我们已经创建了 EventBus，接下来你需要做到的就是在你的组件中加载它，并且调用同一个方，就如你在父子组件中互相传递消息一样。

发送事件

假设你有两个子组件：DecreaseCount和IncrementCount，分别在按钮中绑定了decrease()和increment()方法。这两个方法做的事情很简单，就是数值递减（增）1，以及角度值递减（增）180。在这个方法中，通过EventBus.$emit(channel: string, callback(payload1,...))监听decreased（和incremented）频道。

```
<!-- DecreaseCount.vue -->
<template>
  <button @click="decrease()">-</button>
</template>

<script>
  import { EventBus } from "../event-bus.js";
  export default {
    name: "DecreaseCount",
    data() {
      return {
        num: 1,
        deg:180
      };
    },
    methods: {
      decrease() {
        EventBus.$emit("decreased", {
          num:this.num,
          deg:this.deg
        });
      }
    }
  };
</script>
```

```
<!-- IncrementCount.vue -->
<template>
  <button @click="increment()">+</button>
</template>

<script>
  import { EventBus } from "../event-bus.js";
  export default {
    name: "IncrementCount",
    data() {
      return {
        num: 1,
        deg:180
      };
    },
    methods: {
      increment() {
        EventBus.$emit("incremented", {
          num:this.num,
          deg:this.deg
        });
      }
    }
  };
</script>
```

上面的示例，在DecreaseCount和IncrementCount分别发送出了decreased和incremented频道。下来，我们需要在另一个组件中接收这两个事件，保持数据在各组件之间的通讯。

接收事件

现在我们可以在组件App.vue中使用EventBus.$on(channel: string, callback(payload1,...)) 监听DecreaseCount和IncrementCount分别发送出了decreased和incremented频道。

```
<!-- App.vue -->
<template>
  <div id="app">
    <div class="container" :style="{transform: 'rotateY(' + degValue + 'deg)'}">
      <div class="front">
        <div class="increment">
          <IncrementCount />
        </div>
        <div class="show-front">
          {{fontCount}}
        </div>
        <div class="decrement">
          <DecreaseCount />
        </div>
      </div>

      <div class="back">
        <div class="increment">
          <IncrementCount />
```

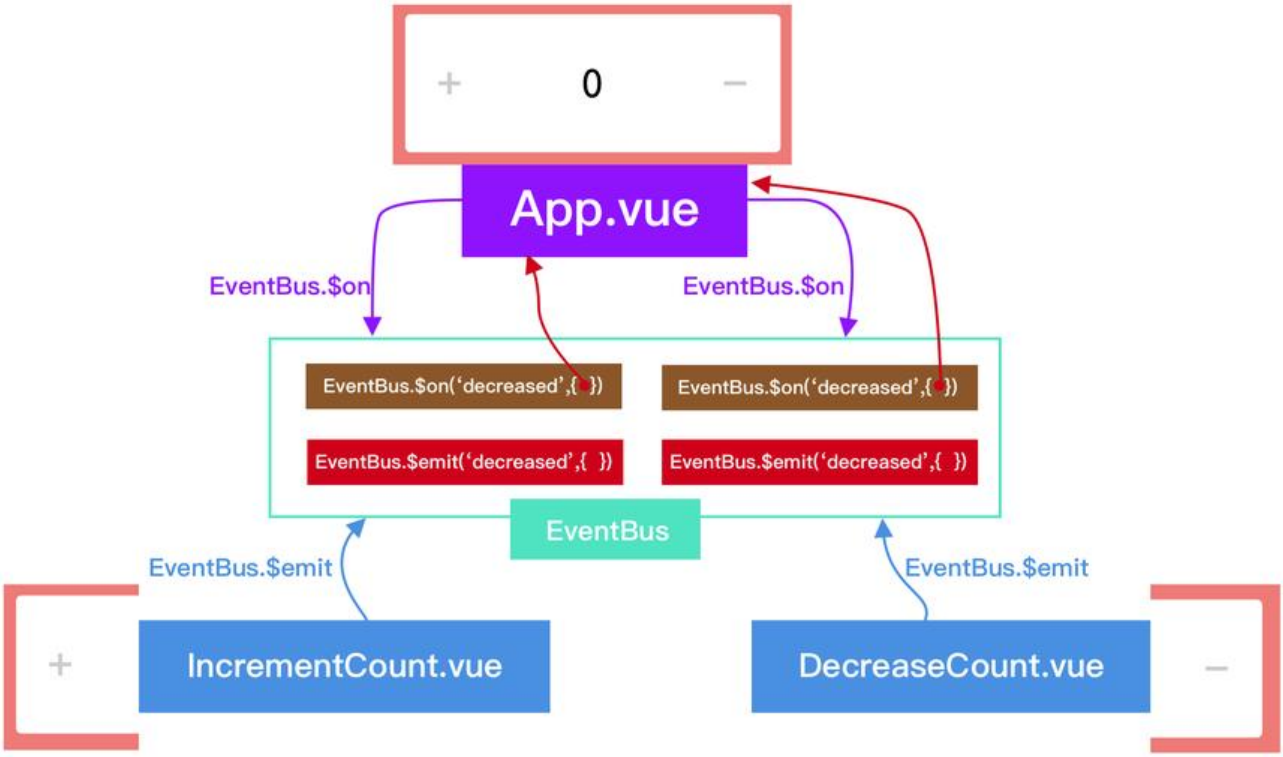
```
            </div>
            <div class="show-back">
                {{backCount}}
            </div>
            <div class="decrement">
                <DecreaseCount />
            </div>
        </div>
    </div>
  </div>
</template>

<script>
  import IncrementCount from "./components/IncrementCount";
  import DecreaseCount from "./components/DecreaseCount";
  import { EventBus } from "./event-bus.js";
  export default {
    name: "App",
    components: {
      IncrementCount,
      DecreaseCount
    },
    data() {
      return {
        degValue:0,
        fontCount:0,
        backCount:0
      };
    },
    mounted() {
      EventBus.$on("incremented", ({num,deg}) => {
        this.fontCount += num
        this.$nextTick(()=>{
          this.backCount += num
          this.degValue += deg;
        })
      });
      EventBus.$on("decreased", ({num,deg}) => {
        this.fontCount -= num
        this.$nextTick(()=>{
          this.backCount -= num
          this.degValue -= deg;
        })
      });
    }
  };
</script>
```

最终得到的效果如下：

<iframe src="https://codesandbox.io/embed/qkrk20n2wj?view=preview" data-src="https://codesandbox.io/embed/qkrk20n2wj?view=preview" sandbox="allow-modals allow-forms allow popups allow-scripts allow-same-origin"></iframe>

最后用一张图来描述示例中用到的EventBus之间的关系：



如果你只想监听一次事件的发生，可以使用 EventBus.$once(channel: string, callback(payload1,...))。

移除事件监听者

如果想移除事件的监听，可以像下面这样操作：

```
import { eventBus } from './event-bus.js'
EventBus.$off('decreased', {})
```

你也可以使用 EventBus.$off('decreased') 来移除应用内所有对此事件的监听。或者直接调用 EventBus.$off() 来移除所有事件频道， **注意不需要添加任何参数** 。

上面就是EventBus的使用方式，是不是很简单。上面的示例中我们也看到了，每次使用EventBus时需要在各组件中引入event-bus.js。事实上，我们还可以通过别的方式，让事情变得简单一些。那就创建一个全局的EventBus。接下来的示例向大家演示如何在Vue项目中创建一个全局的EventBus。

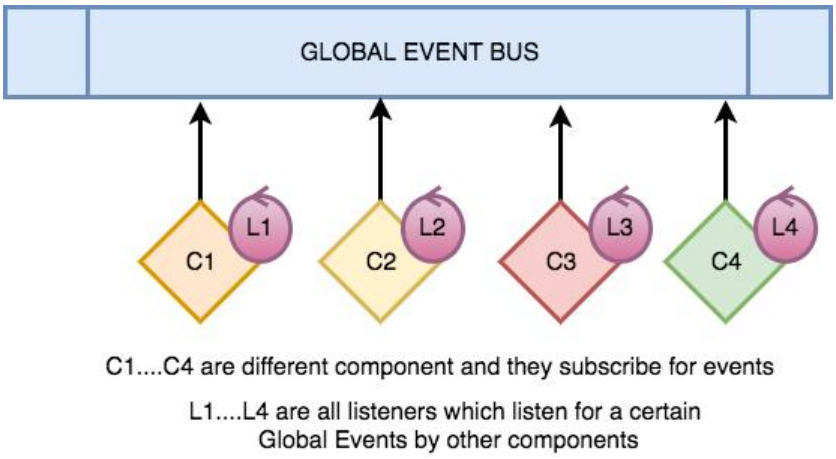
全局EventBus

全局EventBus，虽然在某些示例中不提倡使用，但它是一种非常漂亮且简单的方法，可以跨组件之间享数据。

它的工作原理是发布/订阅方法，通常称为Pub/Sub。

这整个方法可以看作是一种设计模式，因为如果你查看它周围的东西，你会发现它更像是一种体系结解决方案。我们将使用普通的JavaScript，并创建两个组件，并演示EventBus的工作方式。

让我们看看下图，并试着了解在这种情况下究竟发生了什么。



我们从上图中可以得出以下几点：

- 有一个全局EventBus
- 所有事件都订阅它
- 所有组件也发布到它，订阅组件获得更新
- 总结一下。所有组件都能够将事件发布到总线，然后总线由另一个组件订阅，然后订阅它的组件将到更新

在代码中，我们将保持它非常小巧和简洁。我们将它分为两部分，将展示两个组件以及生成事件总线代码。

创建全局EventBus

全局事件总线只不过是一个简单的vue组件。代码如下：

```
var EventBus = new Vue();

Object.defineProperties(Vue.prototype, {
    $bus: {
        get: function () {
            return EventBus
        }
    }
})
```

现在，这个特定的总线使用两个方法$on和$emit。一个用于创建发出的事件，它就是$emit；另一个于订阅$on：

```
var EventBus = new Vue();

this.$bus.$emit('nameOfEvent',{ ... pass some event data ...});

this.$bus.$on('nameOfEvent',($event) => {
    // ...
})
```

现在，我们创建两个简单的组件，以便最终得出结论。

接下来的这个示例中，我们创建了一个ShowMessage的组件用来显示信息，另外创建一个UpdateMessage的组件，用来更新信息。

在UpdateMessage组件中触发需要的事件。在这个示例中，将触发一个updateMessage事件，这个件发送了updateMessage的频道：

```
<!-- UpdateMessage.vue -->
<template>
  <div class="form">
    <div class="form-control">
      <input v-model="message" >
      <button @click="updateMessage()">更新消息</button>
    </div>
  </div>
</template>
<script>
  export default {
    name: "UpdateMessage",
    data() {
      return {
        message: "这是一条消息"
      };
    },
    methods: {
      updateMessage() {
        this.$bus.$emit("updateMessage", this.message);
      }
    },
    beforeDestroy () {
      $this.$bus.$off('updateMessage')
    }
  };
</script>
```

同时在ShowMessage组件中监听该事件：

```
<!-- ShowMessage.vue -->
<template>
  <div class="message">
    <h1>{{ message }}</h1>
  </div>
</template>

<script>
  export default {
    name: "ShowMessage",
    data() {
      return {
        message: "我是一条消息"
      };
    },
    created() {
      var self = this
      this.$bus.$on('updateMessage', function(value) {
```

```
            self.updateMessage(value);
        })
    },
    methods: {
        updateMessage(value) {
            this.message = value
        }
    }
};
</script>
```

最终的效果如下：

<iframe src="https://codesandbox.io/embed/2wvv773r7p?view=preview" data-src="https://codesandbox.io/embed/2wvv773r7p?view=preview" sandbox="allow-modals allow-forms allow popups allow-scripts allow-same-origin"></iframe>

从上面的代码中，我们可以看到ShowMessage组件侦听一个名为updateMessage的特定事件，这事件在组件实例化时被触发，或者你可以在创建组件时触发。另一方面，我们有另一个组件UpdateMssage，它有一个按钮，当有人点击它时会发出一个事件。这导致订阅组件侦听发出的事件。这产生了ub/Sub模型，该模型在兄弟姐妹之间持续存在并且非常容易实现。

总结

本文主要通过两个实例学习了Vue中有关于EventBus相关的知识点。主要涉及了EventBus如何实例，又是怎么通过$emit发送频道信号，又是如何通过$on来接收频道信号。最后简单介绍了怎么创建局的EventBus。从实例中我们可以了解到，EventBus可以较好的实现兄弟组件之间的数据通讯。

出处：https://www.w3cplus.com/vue/event-bus.html