组件的通信 1：provide / inject

上一节中我们说到，ref 和 $parent / $children 在**跨级**通信时是有弊端的。当组件 A 和组件 B 中间隔了数代（甚至不确定具体级别）时，以往会借助 Vuex 或 Bus 这样的解决方案，不得不引入三方库来支持。本小节则介绍一种无依赖的组件通信方法：Vue.js 内置的 provide / inject 接口。

什么是 provide / inject

provide / inject 是 Vue.js 2.2.0 版本后新增的 API，在文档中这样介绍：

> https://cn.vuejs.org/v2/api/#provide-inject (https://cn.vuejs.org/v2/api/#provide-inject)
>
> 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。如果你熟悉 React，这与 React 的上下文特性很相似。

并且文档中有如下提示：

> provide 和 inject 主要为高阶插件/组件库提供用例。并不推荐直接用于应用程序代码中。

看不懂上面的介绍没有关系，不过上面的这句提示应该明白，就是说 Vue.js 不建议在业务中使用这对 API，而是在插件 / 组件库（比如 iView，事实上 iView 的很多组件都在用）。**不过建议归建议，如果**

你用好了，这个 API 会非常有用。

我们先来看一下这个 API 怎么用，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: 'Aresn'
  }
}

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

可以看到，在 A.vue 里，我们设置了一个 **provide: name**，值为 Aresn，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 Aresn。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。

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 Aresn。

替代 Vuex

我们知道，在做 Vue 大型项目时，可以使用 Vuex 做状态管理，它是一个专为 Vue.js 开发的**状态管理模式**，用于集中式存储管理应用的所有组件的状态，并以相应的规则保证状态以一种可预测的方式发生变化。

那了解了 provide / inject 的用法，下面来看怎样替代 Vuex。当然，我们的目的并不是为了替代 Vuex，它还是有相当大的用处，这里只是介绍另一种可行性。

使用 Vuex，最主要的目的是跨组件通信、全局数据维护、多人协同开发。需求比如有：用户的登录信息维护、通知信息维护等全局的状态和数据。

一般在 webpack 中使用 Vue.js，都会有一个入口文件 **main.js**，里面通常导入了 Vue、VueRouter、iView 等库，通常也会导入一个入口组件 app.vue 作为根组件。一个简单的 app.vue 可能只有以下代码：

```
<template>
  <div>
    <router-view></router-view>
  </div>
</template>
<script>
  export default {

  }
</script>
```

使用 provide / inject 替代 Vuex，就是在这个 app.vue 文件上做文章。

我们把 app.vue 理解为一个最外层的根组件，用来存储所有需要的全局数据和状态，甚至是计算属性（computed）、方法（methods）等。因为你的项目中所有的组件（包含路由），它的父组件（或根组件）都是 app.vue，所以我们**把整个 app.vue 实例通过 provide 对外提供**。

app.vue：

```
<template>
  <div>
    <router-view></router-view>
  </div>
</template>
<script>
  export default {
    provide () {
      return {
        app: this
      }
    }
  }
</script>
```

上面，我们把整个 app.vue 的实例 this 对外提供，命名为 **app**（这个名字可以自定义，推荐使用 app，使用这个名字后，子组件不能再使用它作为局部属性）。接下来，任何组件（或路由）只要通过 inject 注入 app.vue 的 app 的话，都可以直接通过 **this.app.xxx** 来访问 app.vue 的 data、computed、methods 等内容。

app.vue 是整个项目第一个被渲染的组件，而且只会渲染一次（即使切换路由，app.vue 也不会被再次渲染），利用这个特性，很适合做一次性全局的状态数据管理，例如，我们将用户的登录信息保存起来：

app.vue，部分代码省略：

```
<script>
  export default {
    provide () {
      return {
        app: this
      }
    },
    data () {
      return {
        userInfo: null
      }
    },
    methods: {
      getUserInfo () {
        // 这里通过 ajax 获取用户信息后，赋值给
this.userInfo，以下为伪代码
        $.ajax('/user/info', (data) => {
          this.userInfo = data;
        });
      }
    },
    mounted () {
      this.getUserInfo();
    }
  }
</script>
```

这样，任何页面或组件，只要通过 inject 注入 app 后，就可以直接访问 userInfo 的数据了，比如：

```
<template>
  <div>
    {{ app.userInfo }}
  </div>
</template>
<script>
  export default {
    inject: ['app']
  }
</script>
```

是不是很简单呢。除了直接使用数据，还可以调用方法。比如在某个页面里，修改了个人资料，这时一开始在 app.vue 里获取的 userInfo 已经不是最新的了，需要重新获取。可以这样使用：

某个页面：

```
<template>
  <div>
    {{ app.userInfo }}
  </div>
</template>
<script>
  export default {
    inject: ['app'],
    methods: {
      changeUserInfo () {
        // 这里修改完用户数据后，通知 app.vue 更新，以下为伪代码
        $.ajax('/user/update', () => {
          // 直接通过 this.app 就可以调用 app.vue 里的方法
          this.app.getUserInfo();
        })
      }
    }
  }
</script>
```

同样非常简单。只要理解了 this.app 是直接获取整个 app.vue 的实例后，使用起来就得心应手了。想一想，配置复杂的 Vuex 的全部功能，现在是不是都可以通过 provide / inject 来实现了呢?

进阶技巧

如果你的项目足够复杂，或需要多人协同开发时，在 app.vue 里会写非常多的代码，多到结构复杂难以维护。这时可以使用 Vue.js 的混合 mixins，将不同的逻辑分开到不同的 js 文件里。

比如上面的用户信息，就可以放到混合里：

user.js：

```
export default {
  data () {
    return {
      userInfo: null
    }
  },
  methods: {
    getUserInfo () {
      // 这里通过 ajax 获取用户信息后，赋值给
this.userInfo，以下为伪代码
      $.ajax('/user/info', (data) => {
        this.userInfo = data;
      });
    }
  },
  mounted () {
    this.getUserInfo();
  }
}
```

然后在 app.vue 中混合：

app.vue：

```
<script>
  import mixins_user from '../mixins/user.js';

  export default {
    mixins: [mixins_user],
    data () {
      return {

      }
    }
  }
</script>
```

这样，跟用户信息相关的逻辑，都可以在 user.js 里维护，或者由某个人来维护，app.vue 也就很容易维护了。

独立组件中使用

如果你顾忌 Vue.js 文档中所说，provide / inject 不推荐直接在应用程序中使用，那没有关系，仍然使用你熟悉的 Vuex 或 Bus 来管理你的项目就好。我们介绍的这对 API，主要还是在独立组件中发挥作用的。

只要一个组件使用了 provide 向下提供数据，那其下所有的子组件都可以通过 inject 来注入，不管中间隔了多少代，而且可以注入多个来自不同父级提供的数据。需要注意的是，一旦注入了某个数据，比如上面示例中的 app，那这个组件中就不能再声明 app 这个数据了，因为它已经被父级占有。

独立组件使用 provide / inject 的场景，主要是具有联动关系的组件，比如接下来很快会介绍的第一个实战：具有数据校验功能的表单组件 Form。它其实是两个组件，一个是 Form，一个是 FormItem，FormItem 是 Form 的子组件，它会依赖 Form 组件上

的一些特性（props），所以就需要得到父组件 Form，这在 Vue.js 2.2.0 版本以前，是没有 provide / inject 这对 API 的，而 Form 和 FormItem 不一定是父子关系，中间很可能间隔了其它组件，所以不能单纯使用 $parent 来向上获取实例。在 Vue.js 2.2.0 之前，一种比较可行的方案是用计算属性动态获取：

```
computed: {
  form () {
    let parent = this.$parent;
    while (parent.$options.name !== 'Form') {
      parent = parent.$parent;
    }
    return parent;
  }
}
```

每个组件都可以设置 name 选项，作为组件名的标识，利用这个特点，通过向上遍历，直到找到需要的组件。这个方法可行，但相比一个 inject 来说，太费劲了，而且不那么优雅和 native。如果用 inject，可能只需要一行代码：

```
export default {
  inject: ['form']
}
```

不过，这一切的前提是你使用 Vue.js 2.2.0 以上版本。

结语

如果这是你第一次听说 provide / inject 这对 API，一定觉得它太神奇了，至少笔者第一时间知晓时是这样的。它解决了独立组件间通信的问题，用好了还有出乎意料的效果。笔者在开发 iView

Developer (https://dev.iviewui.com) 时，全站就是使用这种方法来做全局数据的管理的，如果你有机会一个人做一个项目时，不妨试试这种方法。

下一节，将介绍另一种组件间通信的方法，不同于 provide / inject 的是，它们不是 Vue.js 内置的 API。