Vue组件间通信6种方式

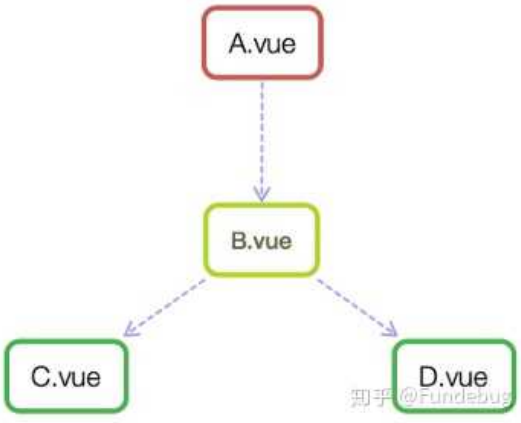
Fundebug
一行代码搞定BUG监控：www.fundebug.com

摘要： 总有一款合适的通信方式。

- 作者：浪里行舟

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



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
  }
}
//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>
```

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

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：

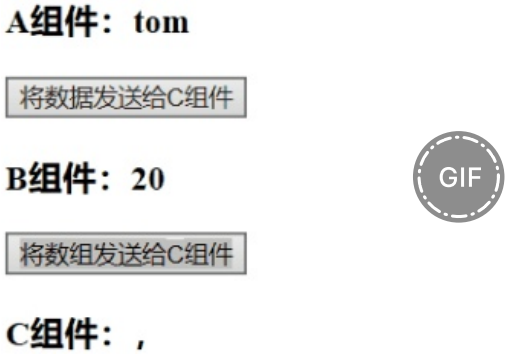
```
    var Event=new Vue();
    Event.$emit(事件名,数据);
    Event.$on(事件名,data => {});
```

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>
```

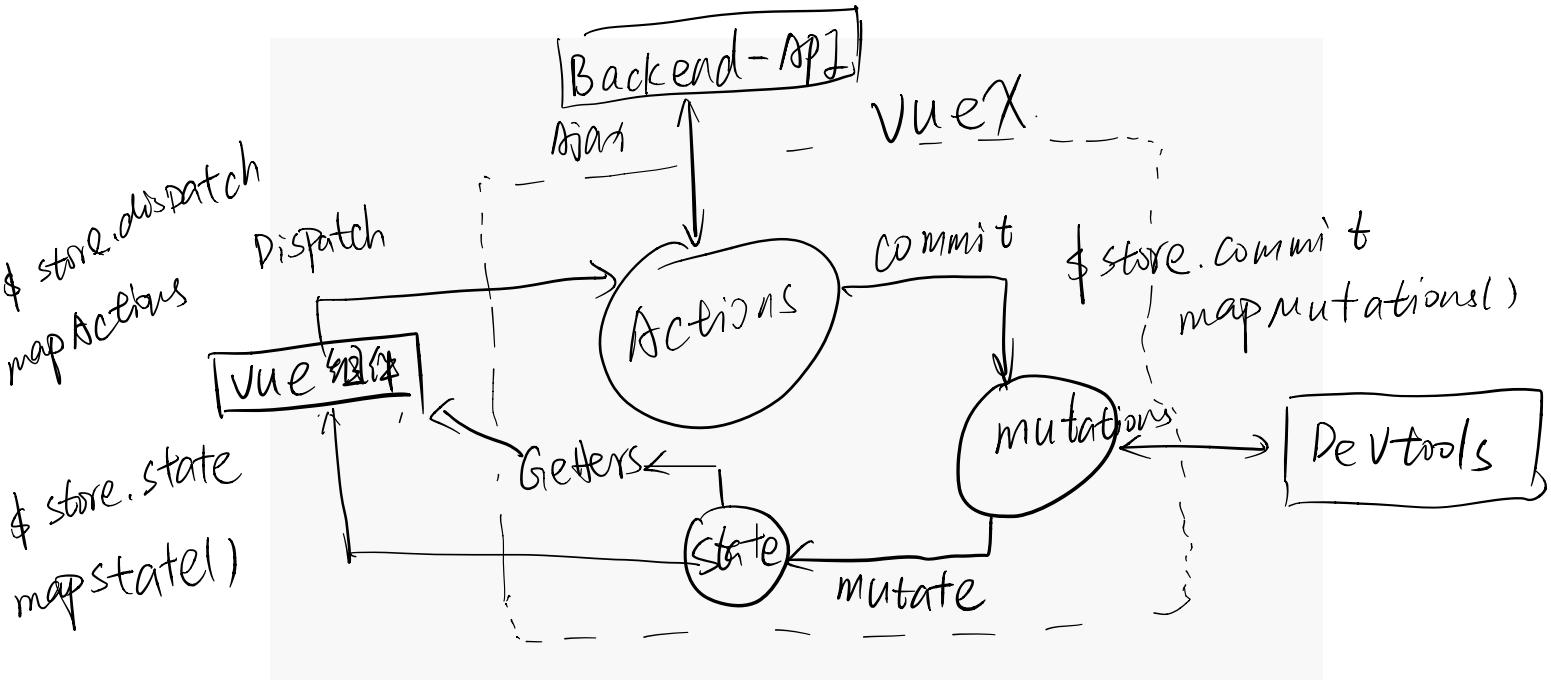


GIF

`$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) {}
    }
  }
})
```

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
```

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 interitAttrs 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>
// 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", "ti
  }
};
</script>
// 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); // { "boo": "Html", "coo": "CSS", "doo": "Vue", "ti
  }
};
</script>
// childCom3.vue
<template>
  <div class="border">
    <p>childCom3: {{ $attrs }}</p>
  </div>
</template>
<script>
export default {
  props: {
    coo: String,
    title: String
  }
};
</script>
```


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

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。

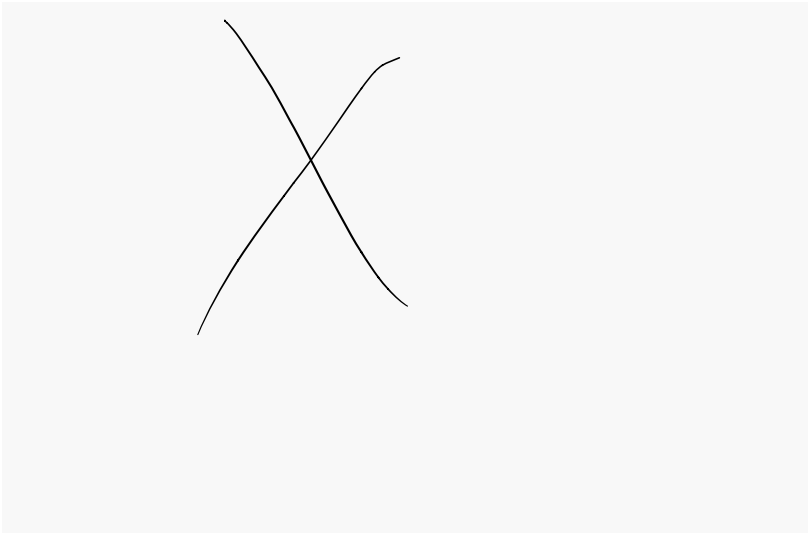
需要注意的是：**provide 和 inject 绑定并不是可响应的。这是刻意为之的。然而，如果你传入了一个可监听的对象，那么其对象的属性还是可响应的**----vue 官方文档 所以，上面 A.vue 的 name 如果改变了，B.vue 的 this.name 是不会改变的，仍然是 浪里行舟。

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 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>
```

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');
    }
  }
}
// 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>
```

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>
```

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

关于Fundebug

Fundebug专注于JavaScript、微信小程序、微信小游戏、支付宝小程序、React Native、Node.js和Java线上应用实时BUG监控。 自从2016年双十一正式上线，Fundebug累计处理了10亿+错误事件，付费客户有Google、360、金山软件、百姓网等众多品牌企业。欢迎大家免费试用！

发布于 2019-05-18

Vue.js　前端开发　前端工程师

文章被以下专栏收录

Fundebug

进入专栏

推荐阅读

前端·Vue2.x入门

呼神护卫

【2019 前端进阶之路】Vue 组件间通信方式完整版

江三疯

vue进阶系列——用typescript玩转vue和vuex

徐小夕 发表于趣谈前端

Vue的个人起步：编写组前端页面

张墨一

3 条评论

切换为时间排序

写下你的评论...

践知 2 个月前

长期坚持写文章，我辈楷模

赞

Passon 2 个月前

defaultCity 那块代码写错了吧

赞

TErRy 1 个月前

10种，别人写过了，你搜一下

赞