【前端面试】Vue组件通讯方式有哪些?

前言

组件化是 Vue 框架的重要思想之一，在前端框架还未出现的时候，通常一个网站页面就是一个文件，如果一个页面有什么数据需要大家使用，直接声明一个全局变量就好了。但是 Vue 框架出现后，将一个页面组件化了，意味着一个页面被分割为了很多个文件，那么组件之间数据的共享就成了一大问题，当然 Vue 为实现组件间的数据共享提供了很多种方法，今天我们就梳理一下到底有哪些方法?

因为项目中常用的就那么几种，所有经常有很多小伙伴在面试的时候说不全，所以还是建议好好理一理。

1.那些场景需要通讯?

由于 Vue 所有的组件呈现组件树的形态，所以组件间的通讯也有很多种情况，大致有以下几种：

- 父子组件间通讯
- 兄弟组件间通讯
- 隔代组件间通讯
- 无相关组件间通讯

每种场景下建议的通讯方式也不一样，需要根据不同的场景选择最合适的组件间通讯方式。

2.props 和$emit

这种方式通常用于父子组件之间的传值，父组件通过属性的方式将值传递给子组件，子组件通过 props 进行接收。子组件通过事件的方式向父组件传递数据。

初始化项目：

我们建立一个最简单的 Vue 项目，分别建了 3 个组件：parent、child1、child2，然后在 APP.vue 中引入 parent 组件，parent 组件中引入 child1 和 child2 两个子组件，初始运行界面如下：

父级组件

child1组件

child2组件

2.1 父组件传值给子组件

接下来我们利用属性方式从父组件传递值给子组件。

父组件示例代码：

```
// src/views/parent.vue
<template>
  <div class="parent-box">
    <p>父级组件</p>
    <div>
      <button @click="changeMsg">更改数据</button>
    </div>
    <child1 :msg="msg"></child1>
    <child2 :msg="msg"></child2>
  </div>
</template>
<script>
```

```
import child1 from "./child1.vue";
import child2 from "./child2.vue";
export default {
  data() {
    return {
      msg: "我是父组件的数据",
    };
  },
  components: {
    child1,
    child2,
  },
  methods: {
    // 点击按钮更改数据
    changeMsg() {
      this.msg = "变成小猪课堂";
    },
  },
};
</script>
```

我们将父组件中的 msg 通过:msg=”msg”的方式传递给子组件，并且点击按钮的时候会修改父组件中的 msg。

子组件示例代码：

```
// src/views/child1.vue
<template>
  <div class="child-1">
    <p>child1 组件</p>
    <div>
      <p>parent 组件数据：{{ msg }}</p>
    </div>
  </div>
</template>
<script>
export default {
  props: {
    msg: {
      type: String,
      default: "",
    },
  },
};
</script>
```

子组件通过 props 属性的方式接收父组件传来的数据。

输出结果：

当我们点击按钮的时候，父组件的数据发生变化，子组件接收的数据也跟着发生了变化。

注意：:msg=”msg”接收的 msg 是一个变量，可以参考 bind 的使用原理，不加：则接收的就是一个字符串。

2.2 子组件传值给父组件

子组件可以通过$emit 自定义事件的方式向父组件传递值，父组件需要监听该事件来进行接收子组件传来的值。

父组件示例代码：

```
// src/views/parent.vue
<template>
  <div class="parent-box">
    <p>父级组件</p>
    <div>
      <button @click="changeMsg">更改数据</button>
```

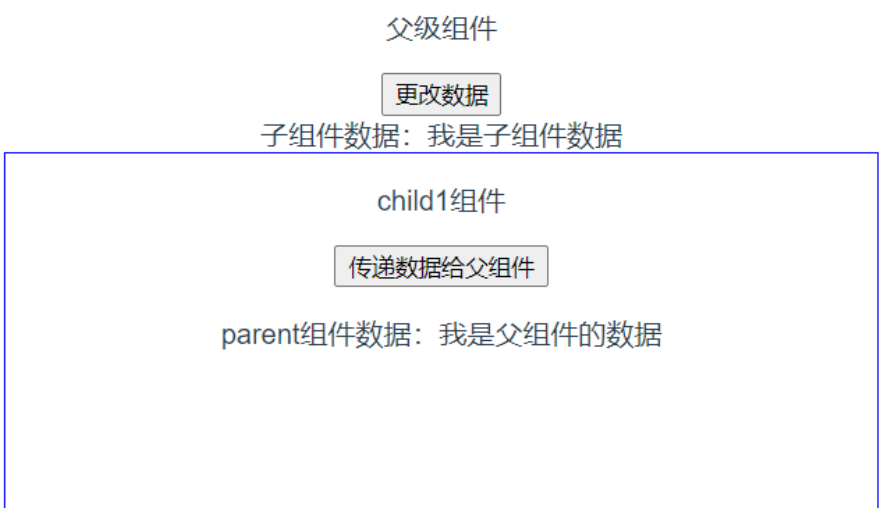
```
    </div>
    <div>子组件数据：{{ childData }}</div>
    <child1 :msg="msg" @childData="childData"></child1>
    <child2 :msg="msg"></child2>
  </div>
</template>
<script>
import child1 from "./child1.vue";
import child2 from "./child2.vue";
export default {
  data() {
    return {
      msg: "我是父组件的数据",
      childData: "",
    };
  },
  components: {
    child1,
    child2,
  },
  methods: {
    changeMsg() {
      this.msg = "变成小猪课堂";
    },
    // 监听子组件事件
    childData(data) {
      this.childData = data;
    },
  },
};
</script>
```

子组件示例代码：

```
// src/views/child1.vue
<template>
  <div class="child-1">
    <p>child1 组件</p>
    <div>
      <button @click="sendData">传递数据给父组件</button>
    </div>
    <div>
      <p>parent 组件数据：{{ msg }}</p>
    </div>
  </div>
</template>
<script>
export default {
  props: {
    msg: {
      type: String,
```

```
      default: "",
    },
  },
  methods: {
    // 点击按钮，使用$emit 向父组件传递数据
    sendData() {
      this.$emit("childData", "我是子组件数据");
    },
  },
};
</script>
```

输出结果：



我们在父组件中通过@childData=”getChildData”的方式来监听 childData 事件，从而获取子组件传递的数据，子组件中通过点击按钮触发$emit 事件向父组件传递数据。当我们点击按钮“传递数据给父组件”时，父组件便可以获取到数据。

3.$parent 获取父组件值

这种方式可以让子组件非常方便的获取父组件的值，不仅仅包括数据，还可以是方法。

子组件示例代码：

```
// src/views/child1.vue
<template>
  <div class="child-1">
    <p>child1 组件</p>
    <div>
      <button @click="sendData">传递数据给父组件</button>
    </div>
    <div>
      <button @click="getParentData">使用$parent</button>
    </div>
    <div>
      <p>parent 组件数据：{{ msg }}</p>
    </div>
```

```
  </div>
</template>
<script>
export default {
  props: {
    msg: {
      type: String,
      default: "",
    },
  },
  methods: {
    sendData() {
      this.$emit("childData", "我是子组件数据");
    },
    // 通过$parent 方式获取父组件值
    getParentData() {
      console.log("父组件", this.$parent);
    },
  },
};
</script>
```

点击“使用$parent”按钮时，通过$parent 获取父组件的属性或数据。

输出结果：

```
父组件                                                    child1.vue?9691:29
▼VueComponent {_uid: 4, _isVue: true, $options: {…}, _renderPro
xy: Proxy, _self: VueComponent, …}
  ▶$attrs: Object
  ▶$children: (2) [VueComponent, VueComponent]
  ▶$createElement: ƒ (a, b, c, d)
  ▶$el: div.parent-box
   $listeners: (…)
  ▶$options: {parent: VueComponent, _parentVnode: VNode, propsDa
  ▶$parent: VueComponent {_uid: 3, _isVue: true, $options: {…},
  ▶$refs: {}
  ▶$root: Vue {_uid: 2, _isVue: true, $options: {…}, _renderProx
  ▶$scopedSlots: {$stable: true, $key: undefined, $hasNormal: fa
  ▶$slots: {}
  ▶$store: Store {_committing: false, _actions: {…}, _actionSubs
  ▶$vnode: VNode {tag: 'vue-component-4-parent', data: {…}, chil
  ▶changeMsg: ƒ ()
   childData: ""
  ▶getChildData: ƒ ()
   msg: (…)
  ▶_c: ƒ (a, b, c, d)
  ▼_data:
     childData: (…)
     msg: "我是父组件的数据"
    ▶__ob__: Observer {value: {…}, dep: Dep, vmCount: 1}
```

我们可以看到控制台打印出了父组件的所有属性，不仅仅包含了 data 数据，还有里面定义的一些方法等。

4.$children 和$refs 获取子组件值

这两种方式和$parent 非常的类似，它们可以直接获取子组件的相关属性或方法，不仅限于数据。

父组件示例代码：

```
// src/views/parent.vue
<template>
  <div class="parent-box">
    <p>父级组件</p>
    <div>
      <button @click="changeMsg">更改数据</button>
    </div>
    <div>
      <button @click="getChildByRef">使用$children 和$refs</button>
    </div>
    <div>子组件数据：{{ childData }}</div>
    <child1 ref="child1" :msg="msg" @childData="getChildData"></child1>
    <child2 :msg="msg"></child2>
  </div>
</template>
<script>
import child1 from "./child1.vue";
import child2 from "./child2.vue";
export default {
  data() {
    return {
      msg: "我是父组件的数据",
      childData: "",
    };
  },
  components: {
    child1,
    child2,
  },
  methods: {
    changeMsg() {
      this.msg = "变成小猪课堂";
    },
    // 监听子组件的自定义事件
    getChildData(data) {
      this.childData = data;
    },
    // 使用$chilren 和$refs 获取子组件
    getChildByRef() {
      console.log("使用$children", this.$children);
      console.log("使用$refs", this.$refs.child1);
    },
  },
```

```
};
</script>
```

输出结果：

```
使用$children                                          parent.vue?fb70:39
▼(2) [VueComponent, VueComponent]
  ▶0: VueComponent {_uid: 5, _isVue: true, $options: {…}, _renderP
  ▶1: VueComponent {_uid: 6, _isVue: true, $options: {…}, _renderP
    length: 2
  ▶[[Prototype]]: Array(0)
使用$refs                                              parent.vue?fb70:40
▶VueComponent {_uid: 5, _isVue: true, $options: {…}, _renderProx
  y: Proxy, _self: VueComponent, …}
>
```

上段代码中，我们点击按钮，分别通过$children 和$refs 的方式获取到了子组件，从而拿到子组件数据。需要注意的是，$children 会返回当前组件所包含的所有子组件数组，使用$refs 时，需要在子组件上添加 ref 属性，有点类似于直接获取 DOM 节点的操作。

5.使用$attrs 和$listeners

$attrs 是在 Vue2.4.0 之后新提出的，通常在多层组件传递数据的时候使用。很多小伙伴如果遇到多层组件数据传递的场景，他可能会直接选用 Vuex 进行传递，但是如果我们需要传递的数据没有涉及到数据的更新和修改时，建议使用$arrts 的方式，毕竟 Vuex 还是比较重。

官网解释：

> 包含了父作用域中不作为 prop 被识别 (且获取) 的 attribute 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——在创建高级别的组件时非常有用。

官网的解释还是比较难理解的，我们可以用更加通俗一点的话在解释一遍。

通俗解释：

> 当父组件传递了很多数据给子组件时，子组件没有声明 props 来进行接收，那么子组件中的 $attrs 属性就包含了所有父组件传来的数据(除开已经 props 声明了的)，子组件还可以使用 v-bind="$attrs"的形式向它的子组件（孙子组件）传递数据，孙子组件使用$attrs 的方式和它的父组件原理类似。

说的再多可能还是没有代码来得简单易懂，我们新建一个孙子组件 child1-child.vue，编写之后界面如下：

5.1 $attrs 的使用

我们在 parent 父组件中多传一点数据给 child1 组件。

parent 组件示例代码：

```
// src/views/parent.vue
<template>
  <div class="parent-box">
    <p>父级组件</p>
    <child1
      ref="child1"
      :msg="msg"
      :msg1="msg1"
      :msg2="msg2"
      :msg3="msg3"
      :msg4="msg4"
      @childData="getChildData"
    ></child1>
  </div>
</template>
<script>
import child1 from "./child1.vue";
import child2 from "./child2.vue";
export default {
  data() {
    return {
```

```
      msg: "我是父组件的数据",
      msg1: "parent 数据 1",
      msg2: "parent 数据 2",
      msg3: "parent 数据 3",
      msg4: "parent 数据 4",
      childData: "",
    };
  },
  components: {
    child1,
    child2,
  }
};
</script>
```

这里我们删除了一些本节用不到的代码，大家需要注意一下。

child1 组件示例代码：

```
// src/views/child1.vue
<template>
  <div class="child-1">
    <p>child1 组件</p>
    <!-- 子组件 child1-child -->
    <child1-child v-bind="$attrs"></child1-child>
  </div>
</template>
<script>
import Child1Child from "./child1-child";
export default {
  components: {
    Child1Child,
  },
  props: {
    msg: {
      type: String,
      default: "",
    },
  },
  mounted() {
    console.log("child1 组件获取$attrs", this.$attrs);
  }
};
</script>
```

输出结果：

```
child1组件获取$attrs                                    child1.vue?9691:30
▼{msg1: 'parent数据1', msg2: 'parent数据2', msg3: 'parent数据3', m
 sg4: 'parent数据4'} ⓘ
    msg1: "parent数据1"
    msg2: "parent数据2"
    msg3: "parent数据3"
    msg4: "parent数据4"
  ▶ [[Prototype]]: Object
```

上段代码中我们的 parent 父组件传递了 5 个数据给子组件：msg、msg1、msg2、msg3、msg4。但是在子组件中的 props 属性里面，我们只接收了 msg。然后我们在子组件 mounted 中打印了$attrs，发现恰好少了 props 接收过的 msg 数据。

当我们在 child1 组件中使用$attrs 接收了组件后，可以使用 v-bind="$attrs"的形式在传递给它的子组件 child1-child，上段代码中我们已经加上了 v-bind。

child1-child 组件示例代码：

```
// src/views/child1-child.vue
<template>
  <div class="child1-child">
    <p>我是孙子组件 child1-child</p>
  </div>
</template>
<script>
export default {
  props: {
    msg1: {
      type: String,
      default: "",
    },
  },
  mounted() {
    console.log("child1-child 组件$attrs", this.$attrs);
  },
};
</script>
```

输出结果：

```
child1-child组件$attrs                                    child1-child.vue?79db:15
▼{msg2: 'parent数据2', msg3: 'parent数据3', msg4: 'parent数据4'}
    msg2: "parent数据2"
    msg3: "parent数据3"
    msg4: "parent数据4"
  ▶ [[Prototype]]: Object
child1组件获取$attrs                                      child1.vue?9691:30
▼{msg1: 'parent数据1', msg2: 'parent数据2', msg3: 'parent数据3', m
sg4: 'parent数据4'}
    msg1: "parent数据1"
    msg2: "parent数据2"
    msg3: "parent数据3"
    msg4: "parent数据4"
  ▶ [[Prototype]]: Object
```

我们发现 child1-child 组件中打印的$attrs 中少了 msg1，因为我们已经在 props 中接收了 msg1。

5.2 $listeners 的使用

$listeners 属性和$attrs 属性和类型，只是它们传递的东西不一样。

官网的解释：

> 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——在创建更高层次的组件时非常有用。

通俗的解释：

> **当父组件在子组件上定义了一些自定义的非原生事件时，在子组件内部可以通过$listeners 属性获取到这些自定义事件。**

它和$attrs 的区别很明显，$attrs 用来传递属性，$listeners 用来传递非原生事件，我们在 child1 组件中打印一下看看。

child1 组件示例代码：

```
// src/views/child1.vue
mounted() {
  console.log("child1 组件获取$attrs", this.$attrs);
  console.log("child1 组件获取$listeners", this.$listeners);
},
```

输出结果：

```
child1组件获取$attrs                                      child1.vue?9691:30
▶{msg1: 'parent数据1', msg2: 'parent数据2', msg3: 'parent数据3', m
sg4: 'parent数据4'}
child1组件获取$listeners                                  child1.vue?9691:31
▼{childData: ƒ}
  ▶ childData: ƒ invoker()
  ▶ [[Prototype]]: Object
```

可以发现输出了 childData 方法，这是我们在它的父组件自定义的监听事件。除次之外，$listeners 可以通过 v-on 的形式再次传递给下层组件。

child1 组件示例代码：

```
// src/views/child1.vue
<template>
  <div class="child-1">
    <p>child1 组件</p>
    <div>
      <button @click="sendData">传递数据给父组件</button>
    </div>
    <div>
      <button @click="getParentData">使用$parent</button>
    </div>
    <div>
      <p>parent 组件数据：{{ msg }}</p>
    </div>
    <!-- 子组件 child1-child -->
    <child1-child v-bind="$attrs" v-on="$listeners"></child1-child>
  </div>
</template>
```

child1-child 组件示例代码：

```
// src/views/child1-child.vue
mounted() {
  console.log("child1-child 组件$attrs", this.$attrs);
  console.log("child1-child 组件$listerners", this.$listeners);
},
```

输出结果：

可以看到在 child1-child 孙子组件中也获得了 parent 父组件中的 childData 自定义事件。使用 $listeners 的好处在于：如果存在多层级组件，无需使用$emit 的方式逐级向上触发事件，只需要使用$listerners 就可以得到父组件中的自定义事件，相当于偷懒了。

5.3 inheritAttrs

可能细心的小伙伴会发现，我们在使用$attrs 时，child1 子组件渲染的 DOM 节点上将我们传递的属性一起渲染了出来，如下图所示：

```
<div id="app">
  <div data-v-d03a4b78 class="parent-box"> flex
    <p data-v-d03a4b78>父级组件</p>
    <div data-v-d03a4b78>…</div>
    <div data-v-d03a4b78>…</div>
    <div data-v-d03a4b78>子组件数据：</div> == $0
    <div data-v-16b0daa2 data-v-d03a4b78 class="child-1" msg1="parent数据1" msg2=
    "parent数据2" msg3="parent数据3" msg4="parent数据4">…</div>
    <div data-v-1694aba0 data-v-d03a4b78 class="child-2">…</div>
```

这并不是我们想要的，为了解决这个问题，我们可以在子组件中设置 inheritAttrs 属性。

官网解释：

> 默认情况下父作用域的不被认作 props 的 attribute 绑定 (attribute bindings) 将会“回退”且作为普通的 HTML attribute 应用在子组件的根元素上。当撰写包裹一个目标元素或另一个组件的组件时，这可能不会总是符合预期行为。通过设置 inheritAttrs 到 false，这些默认行为将会被去掉。而通过 (同样是 2.4 新增的) 实例 property $attrs 可以让这些 attribute 生效，且可以通过 v-bind 显性的绑定到非根元素上。

官网说了非常多，但是不太通俗，我们可以简单的理解。

通俗解释：

> 父组件传递了很多数据给子组件，子组件的 props 没有完全接收，那么父组件传递的这些数据就会渲染到 HTML 上，我们可以给子组件设置 inheritAttrs 为 false，避免这样渲染。

child1 组件示例代码：

```
// src/views/child1.vue
props: {
  msg: {
    type: String,
    default: "",
  },
},
inheritAttrs: false,
```

输出结果：

```
<div data-v-d03a4b78 class="parent-box"> flex
  <p data-v-d03a4b78>父级组件</p>
  <div data-v-d03a4b78>…</div>
  <div data-v-d03a4b78>…</div>
  <div data-v-d03a4b78>子组件数据：</div> == $0
  <div data-v-16b0daa2 data-v-d03a4b78 class="child-1">…</div>
  <div data-v-1694aba0 data-v-d03a4b78 class="child-2">…</div>
</div>
```

此时我们节点上就没有那些无关的节点属性了。

5.4 总结

- $attrs：用来传递属性，出了 class、style 之类的，它是一个对象。

- $listeners：用来传递事件，出了原生事件，它也是一个对象。
- $attrs 和$listeners 这两个属性可以解决多层组件之间数据和事件传递的问题。
- inheritAttrs 解决未使用 props 接收的数据的属性渲染。

6.自定义事件：事件总线

在我们做项目的时候，会发现不相关的组件之间的数据传递是较为麻烦的，比如兄弟组件、跨级组件，在不使用 Vuex 情况下，我们可以使用自定义事件(也可以称作事件中心)的方式来实现数据传递。

事件中心的思想也比较简单：中间中心主要就两个作用：**触发事件和监听事件**。假如两个组件之间需要传递数据，组件 A 可以触发事件中心的事件，组件 B 监听事件中心的事件，从而让两个组件之间产生关联，实现数据传递。

实现步骤：

为了演示简单，我们在全局注册一个事件中心，修改 main.js。

main.js 代码如下：

```
// src/main.js
Vue.config.productionTip = false
Vue.prototype.$EventBus = new Vue()
new Vue({
  router,
  store,
  render: h => h(App)
}).$mount('#app')
```

child1 组件示例代码：

```
<template>
  <div class="child-1">
    <p>child1 组件</p>
    <div>
      <button @click="toChild2">向 child2 组件发送数据</button>
    </div>
  </div>
</template>
<script>
import Child1Child from "./child1-child";
export default {
  methods: {
    // 通过事件总线向 child2 组件发送数据
    toChild2() {
      this.$EventBus.$emit("sendMsg", "我是 child1 组件发来的数据");
    },
  },
};
</script>
```

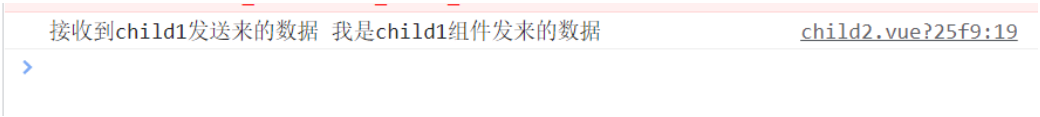
child1 组件中调用$EventBus.$emit 向事件中心添加 sendMsg 事件，这个用法有点类似与 props 和$emit 的关系。

child2 组件 2 示例代码：

```
// src/views/child1.vue
<template>
  <div class="child-2">
    <p>child2 组件</p>
    <div>
      <p>parent 组件数据：{{ msg }}</p>
    </div>
  </div>
</template>
<script>
export default {
  props: {
    msg: {
      type: String,
      default: "",
    },
  },
  mounted() {
    this.$EventBus.$on("sendMsg", (msg) => {
      console.log("接收到 child1 发送来的数据", msg);
    });
  },
};
</script>
```

当我们点击 child1 组件中的按钮时，就会触发 sendMsg 事件，在 child2 组件中我们监听了该事件，所以会接收到 child1 组件发来的数据。

输出结果：



事件中心实现数据传递的这种方式，其实就是一个发布者和订阅者的模式，这种方式可以实现任何组件之间的通信。

7.provide 和 inject

这两个是在 Vue2.2.0 新增的 API，provide 和 inject 需要在一起使用。它们也可以实现组件之间的数据通信，但是需要确保组件之间是父子关系。

官网的解释：

> 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，并在其上下游关系成立的时间里始终生效。

官网的解释就已经说得很明确了，所以这里我们就不需要通俗的解释了，简单一句话：父组件可以向子组件（无论层级）注入依赖，每个子组件都可以获得这个依赖，无论层级。

parent 示例代码：

```
// src/views/parent.vue
<script>
import child1 from "./child1.vue";
import child2 from "./child2.vue";
export default {
  provide() {
    return { parentData: this.msg };
  },
  data() {
    return {
      msg: "我是父组件的数据",
      msg1: "parent 数据 1",
      msg2: "parent 数据 2",
      msg3: "parent 数据 3",
      msg4: "parent 数据 4",
      childData: "",
    };
  },
  components: {
    child1,
    child2,
  },
};
</script>
```

child1-child 组件示例代码：

```
// src/views/child1-child.vue
<template>
  <div class="child1-child">
    <p>我是孙子组件 child1-child</p>
    <p>parent 组件数据：{{parentData}}</p>
  </div>
</template>
<script>
export default {
  inject: ["parentData"],
  props: {
    msg1: {
      type: String,
      default: "",
    },
  },
  mounted() {
    console.log("child1-child 组件$attrs", this.$attrs);
    console.log("child1-child 组件$listerners", this.$listeners);
```

```
    console.log("child1-child 组件获取 parent 组件数据", this.parentData)
  },
};
</script>
```

输出结果：

```
child1-child组件获取parent组件数据 我是父组件的数据          child1-child.vue?79db:19
```

通过 provide 和 inject 结合的方式，我们在 child1-child 组件中获取到了 parent 组件中的数据。如果你下来尝试过的话，可能会发现一个问题，此时数据不是响应式，也就是 parent 组件更改了数据，child1-child 组件中的数据不会更新。

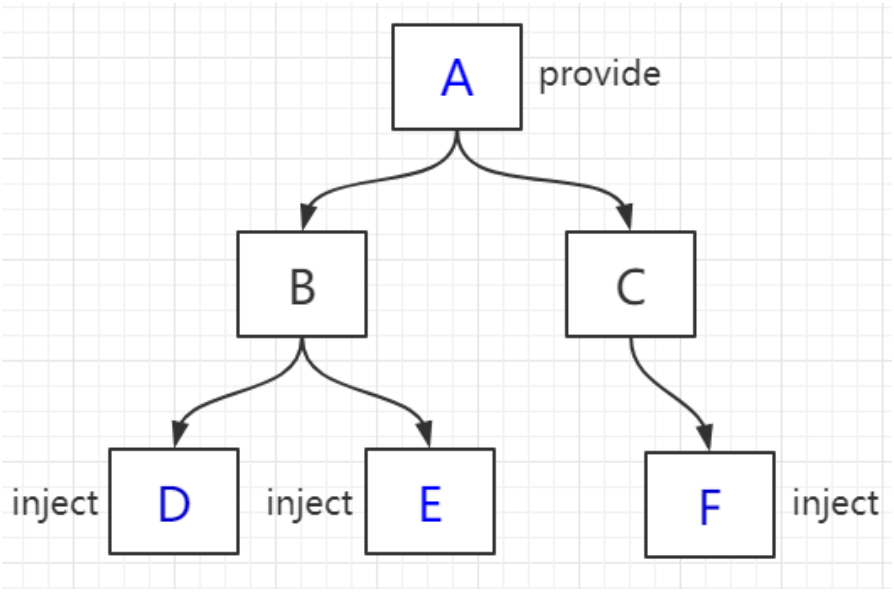
想要变为响应式的，我们需要修改一下 provide 传递的方式。

parent 代码如下：

```
// src/views/parent.vue
<script>
import child1 from "./child1.vue";
import child2 from "./child2.vue";
export default {
  provide() {
    return { parentData: this.getMsg };
  },
  data() {
    return {
      msg: "我是父组件的数据",
      msg1: "parent 数据 1",
      msg2: "parent 数据 2",
      msg3: "parent 数据 3",
      msg4: "parent 数据 4",
      childData: "",
    };
  },
  components: {
    child1,
    child2,
  },
  methods: {
    // 返回 data 数据
    getMsg() {
      return this.msg;
    },
  },
};
</script>
```

这个时候我们会发现数据变为响应式的了。

porvide 和 inject 的原理可以参考下图：



8.Vuex 和 localStorage

这两种方式应该是小伙伴们在实际项目中使用最多的了，所以这里就不但展开细说，只是提一下这两者的区别即可。

Vuex：

- Vuex 是状态管理器，它存储的数据不是持久化存储，一旦刷新页面或者关闭项目数据便不见了。
- Vuex 存储的数据是响应式的。

localstorage：

- loacalStorage 是 HTML5 中的一种数据存储方式，持久化存储，存储的数据不是响应式的。

9.v-model

v-model 是 vue 中的一个内置指令，它通常用在表单元素上以此来实现数据的双向绑定，它的本质是 v-on 和 v-bind 的语法糖。在这里我们也可以借助它来实现某些场景下的数据传递。注意，这儿的场景必须是父子组件。

parent 组件示例代码：

```
<template>
  <div class="parent-box">
    <p>父级组件</p>
    <div>modelData： {{modelData}}</div>
    <child2 :msg="msg" v-model="modelData"></child2>
    <!-- 实际等同于 -->
    <!-- <child2 v-bind:value="modelData" v-on:input="modelData=$event"></ch
  </div>
</template>
<script>
```

```
import child2 from "./child2.vue";
export default {
  provide() {
    return { parentData: this.getMsg };
  },
  data() {
    return {
      modelData: "parent 组件的 model 数据"
    };
  },
  components: {
    child1,
  },
};
</script>
```

child2 组件示例代码：

```
<template>
  <div class="child-2">
    <p>child2 组件</p>
    <div>
      <button @click="confirm">修改 v-model 数据</button>
    </div>
  </div>
</template>
<script>
export default {
  props: {
    value: {
      type: String,
      default: "",
    },
  },
  mounted() {
    console.log("child2 组件接收附件见 v-model 传递的数据", this.value);
  },
  methods: {
    // 通过$emit 触发父组件的 input 事件，并将第二个参数作为值传递给父组件
    confirm() {
      this.$emit("input", "修改 parent 传递的 v-model 数据");
    },
  },
};
</script>
```

我们在父组件中使用 v-model 向 child2 子组件传递数据，子组件的 props 中使用默认的 value 属性接收，在子组件中利用$emit 触发父组件中默认 input 事件，此时传递的数据便会在子组件和父组件中发生变化，这就是数据双向绑定。

如果想要更加详细的学习 v-model 的使用，可以参考官网。

总结

Vue 中组件通讯的方式有很多种，每一种应用的场景可能都有一些不一样，我们需要在合适的场景下选择合适的通讯方式。

- 父子组件间通讯：props 和$emit、$parent、$refs 和$children、v-model
- 兄弟组件间通讯：事件总线、Vuex、localStorage
- 隔代组件间通讯：provide 和 inject
- 无相关组件间通讯：事件总线、Vuex、localStorage