> 面试题：vue组件之间有哪些通信方式?

父子组件通信

> 绝大部分 vue 本身提供的通信方式，都是父子组件通信

prop

最常见的组件通信方式之一，由父组件传递到子组件

event

最常见的组件通信方式之一，当子组件发生了某些事，可以通过 event 通知父组件

style 和 class

父组件可以向子组件传递 style 和 class ， 它们会合并到子组件的根元素中

示例

父组件

```
<template>
  <div id="app">
    <HelloWorld
      style="color:red"
      class="hello"
      msg="Welcome to Your Vue.js App"
    />
  </div>
</template>

<script>
import HelloWorld from "./components/HelloWorld.vue";

export default {
  components: {
    HelloWorld,
  },
};
</script>
```

子组件

```
<template>
  <div class="world" style="text-align:center">
    <h1>{{msg}}</h1>
  </div>
</template>

<script>
export default {
  name: "HelloWorld",
  props: {
    msg: String,
  },
};
</script>
```

渲染结果：

```
<div id="app">
  <div class="hello world" style="color:red; text-aling:center">
    <h1>Welcome to Your Vue.js App</h1>
  </div>
</div>
```

attribute

如果父组件传递了一些属性到子组件，但子组件并没有声明这些属性，则它们称之为 `attribute` ， 这些属性会直接附着在子组件的根元素上

> 不包括 `style` 和 `class` ， 它们会被特殊处理

示例

父组件

```
<template>
  <div id="app">
    <!-- 除 msg 外，其他均为 attribute -->
    <HelloWorld
      data-a="1"
      data-b="2"
      msg="Welcome to Your Vue.js App"
    />
  </div>
</template>

<script>
import HelloWorld from "./components/HelloWorld.vue";

export default {
  components: {
    HelloWorld,
  },
};
</script>
```

子组件

```
<template>
  <div>
    <h1>{{msg}}</h1>
  </div>
</template>

<script>
export default {
  name: "HelloWorld",
  props: {
    msg: String,
  },
  created() {
    console.log(this.$attrs); // 得到:  { "data-a": "1", "data-b": "2" }
  },
};
</script>
```

渲染结果:

```
<div id="app">
  <div data-a="1" data-b="2">
    <h1>Welcome to Your Vue.js App</h1>
  </div>
</div>
```

子组件可以通过 `inheritAttrs: false` 配置，禁止将 `attribute` 附着在子组件的根元素上，但不影响通过 `$attrs` 获取

natvie 修饰符

在注册事件时，父组件可以使用 `native` 修饰符，将事件注册到子组件的根元素上

示例

父组件

```
<template>
  <div id="app">
    <HelloWorld @click.native="handleClick" />
  </div>
</template>

<script>
import HelloWorld from "./components/HelloWorld.vue";

export default {
  components: {
    HelloWorld,
  },
  methods: {
    handleClick() {
      console.log(1);
    },
  },
};
</script>
```

子组件

```
<template>
  <div>
    <h1>Hello World</h1>
  </div>
</template>
```

渲染结果

```
<div id="app">
  <!-- 点击该 div，会输出 1 -->
  <div>
    <h1>Hello World</h1>
  </div>
</div>
```

$listeners

子组件可以通过 `$listeners` 获取父组件传递过来的所有事件处理函数

v-model

后续章节讲解

sync 修饰符

和 v-model 的作用类似，用于双向绑定，不同点在于 v-model 只能针对一个数据进行双向绑定，而 sync 修饰符没有限制

示例

子组件

```
<template>
  <div>
    <p>
      <button @click="$emit(`update:num1`, num1 - 1)">-</button>
      {{ num1 }}
      <button @click="$emit(`update:num1`, num1 + 1)">+</button>
    </p>
    <p>
      <button @click="$emit(`update:num2`, num2 - 1)">-</button>
      {{ num2 }}
      <button @click="$emit(`update:num2`, num2 + 1)">+</button>
    </p>
  </div>
</template>

<script>
export default {
  props: ["num1", "num2"],
};
</script>
```

父组件

```
<template>
  <div id="app">
    <Numbers :num1.sync="n1" :num2.sync="n2" />
    <!-- 等同于 -->
    <Numbers
      :num1="n1"
      @update:num1="n1 = $event"
      :num2="n2"
      @update:num2="n2 = $event"
    />
  </div>
</template>

<script>
import Numbers from "./components/Numbers.vue";

export default {
  components: {
    Numbers,
  },
  data() {
    return {
      n1: 0,
      n2: 0,
    };
  },
};
</script>
```

$parent 和 $children

在组件内部，可以通过 $parent 和 $children 属性，分别得到当前组件的父组件和子组件实例

$slots 和 $scopedSlots

后续章节讲解

ref

父组件可以通过 ref 获取到子组件的实例

跨组件通信

Provide 和 Inject

示例

```
// 父级组件提供 'foo'
var Provider = {
  provide: {
    foo: 'bar'
  },
  // ...
}

// 组件注入 'foo'
var Child = {
  inject: ['foo'],
  created () {
    console.log(this.foo) // => "bar"
  }
  // ...
}
```

详见：https://cn.vuejs.org/v2/api/?#provide-inject

router

如果一个组件改变了地址栏，所有监听地址栏的组件都会做出相应反应

最常见的场景就是通过点击 `router-link` 组件改变了地址， `router-view` 组件就渲染其他内容

vuex

适用于大型项目的数据仓库

store 模式

适用于中小型项目的数据仓库

```
// store.js
const store = {
  loginUser: ...,
  setting: ...
}

// compA
const compA = {
  data(){
    return {
      loginUser: store.loginUser
    }
  }
}

// compB
const compB = {
  data(){
    return {
      setting: store.setting,
      loginUser: store.loginUser
    }
  }
}
```

eventbus

组件通知事件总线发生了某件事，事件总线通知其他监听该事件的所有组件运行某个函数