组件的通信 3：找到任意组件实例——findComponents 系列方法

概述

前面的小节我们已经介绍了两种组件间通信的方法：provide / inject 和 dispatch / broadcast。它们有各自的使用场景和局限，比如前者多用于子组件获取父组件的状态，后者常用于父子组件间通过自定义事件通信。

本节将介绍第 3 种组件通信方法，也就是 findComponents 系列方法，它并非 Vue.js 内置，而是需要自行实现，以工具函数的形式来使用，它是一系列的函数，可以说是组件通信的终极方案。findComponents 系列方法最终都是返回组件的实例，进而可以读取或调用该组件的数据和方法。

它适用于以下场景：

- 由一个组件，向上找到最近的指定组件；
- 由一个组件，向上找到所有的指定组件；
- 由一个组件，向下找到最近的指定组件；
- 由一个组件，向下找到所有指定的组件；
- 由一个组件，找到指定组件的兄弟组件。

5 个不同的场景，对应 5 个不同的函数，实现原理也大同小异。

实现

5 个函数的原理，都是通过递归、遍历，找到指定组件的 name 选项匹配的组件实例并返回。

> 本节以及后续章节，都是基于上一节的工程来完成，后续不再重复说明。
>
> 完整源码地址：https://github.com/icarusion/vue-component-book (https://github.com/icarusion/vue-component-book)

在目录 src 下新建文件夹 utils 用来放置工具函数，并新建文件 assist.js，本节所有函数都在这个文件里完成，每个函数都通过 export 对外提供（如果你不了解 export，请查看扩展阅读1）。

向上找到最近的指定组件——findComponentUpward

先看代码：

```
// assist.js
// 由一个组件，向上找到最近的指定组件
function findComponentUpward (context, componentName) {
  let parent = context.$parent;
  let name = parent.$options.name;

  while (parent && (!name || [componentName].indexOf(name) < 0)) {
    parent = parent.$parent;
    if (parent) name = parent.$options.name;
  }
  return parent;
}
export { findComponentUpward };
```

findComponentUpward 接收两个参数，第一个是当前上下文，比如你要基于哪个组件来向上寻找，一般都是基于当前的组件，也就是传入 this；第二个参数是要找的组件的 name 。

findComponentUpward 方法会在 while 语句里不断向上覆盖当前的 parent 对象，通过判断组件（即 parent）的 name 与传入的 componentName 是否一致，直到直到最近的一个组件为止。

与 dispatch 不同的是，findComponentUpward 是直接拿到组件的实例，而非通过事件通知组件。比如下面的示例，有组件 A 和组件 B，A 是 B 的父组件，在 B 中获取和调用 A 中的数据和方法：

```
<!-- component-a.vue -->
<template>
  <div>
    组件 A
    <component-b></component-b>
  </div>
</template>
<script>
  import componentB from './component-b.vue';

  export default {
    name: 'componentA',
    components: { componentB },
    data () {
      return {
        name: 'Aresn'
      }
    },
    methods: {
      sayHello () {
        console.log('Hello, Vue.js');
      }
    }
  }
</script>
```

```
<!-- component-b.vue -->
<template>
  <div>
    组件 B
  </div>
</template>
<script>
  import { findComponentUpward } from
'../utils/assist.js';

  export default {
    name: 'componentB',
    mounted () {
      const comA = findComponentUpward(this,
'componentA');

      if (comA) {
        console.log(comA.name);  // Aresn
        comA.sayHello();  // Hello, Vue.js
      }
    }
  }
</script>
```

使用起来很简单，只要在需要的地方调用 findComponentUpward 方法就行，第一个参数一般都是传入 this，即当前组件的上下文（实例）。

上例的 comA，保险起见，加了一层 `if (comA)` 来判断是否找到了组件 A，如果没有指定的组件而调用的话，是会报错的。

findComponentUpward 只会找到最近的一个组件实例，如果要找到全部符合要求的组件，就需要用到下面的这个方法。

向上找到所有的指定组件——findComponentsUpward

代码如下：

```
// assist.js
// 由一个组件，向上找到所有的指定组件
function findComponentsUpward (context,
componentName) {
  let parents = [];
  const parent = context.$parent;

  if (parent) {
    if (parent.$options.name === componentName)
parents.push(parent);
    return
parents.concat(findComponentsUpward(parent,
componentName));
  } else {
    return [];
  }
}
export { findComponentsUpward };
```

与 findComponentUpward 不同的是，findComponentsUpward 返回的是一个数组，包含了所有找到的组件实例（注意函数名称中多了一个“s”）。

findComponentsUpward 的使用场景较少，一般只用在递归组件里面（后面小节会介绍），因为这个函数是一直向上寻找父级（parent）的，只有递归组件的父级才是自身。事实上，iView 在使

用这个方法也都是用在递归组件的场景，比如菜单组件 Menu。由于递归组件在 Vue.js 组件里面并不常用，那自然 findComponentsUpward 也不常用了。

向下找到最近的指定组件——findComponentDownward

代码如下：

```
// assist.js
// 由一个组件，向下找到最近的指定组件
function findComponentDownward (context,
componentName) {
  const childrens = context.$children;
  let children = null;

  if (childrens.length) {
    for (const child of childrens) {
      const name = child.$options.name;

      if (name === componentName) {
        children = child;
        break;
      } else {
        children = findComponentDownward(child,
componentName);
        if (children) break;
      }
    }
  }
  return children;
}
export { findComponentDownward };
```

context.$children 得到的是当前组件的全部子组件，所以需要遍历一遍，找到有没有匹配到的组件 name，如果没找到，继续递归找每个 $children 的 $children，直到找到最近的一个为止。

来看个示例，仍然是 A、B 两个组件，A 是 B 的父组件，在 A 中找到 B：

```
<!-- component-b.vue -->
<template>
  <div>
    组件 B
  </div>
</template>
<script>
  export default {
    name: 'componentB',
    data () {
      return {
        name: 'Aresn'
      }
    },
    methods: {
      sayHello () {
        console.log('Hello, Vue.js');
      }
    }
  }
</script>
```

```
<!-- component-a.vue -->
<template>
  <div>
    组件 A
    <component-b></component-b>
  </div>
</template>
<script>
  import componentB from './component-b.vue';
  import { findComponentDownward } from
'../utils/assist.js';

  export default {
    name: 'componentA',
    components: { componentB },
    mounted () {
      const comB = findComponentDownward(this,
'componentB');
      if (comB) {
        console.log(comB.name);  // Aresn
        comB.sayHello();  // Hello, Vue.js
      }
    }
  }
</script>
```

示例中的 A 和 B 是父子关系，因此也可以直接用 ref 来访问，但如果不是父子关系，中间间隔多代，用它就很方便了。

向下找到所有指定的组件——findComponentsDownward

如果要向下找到所有的指定组件，要用到 findComponentsDownward 函数，代码如下：

```
// assist.js
// 由一个组件，向下找到所有指定的组件
function findComponentsDownward (context, componentName) {
  return context.$children.reduce((components, child) => {
    if (child.$options.name === componentName) components.push(child);
    const foundChilds = findComponentsDownward(child, componentName);
    return components.concat(foundChilds);
  }, []);
}
export { findComponentsDownward };
```

这个函数实现的方式有很多，这里巧妙使用 reduce 做累加器，并用递归将找到的组件合并为一个数组并返回，代码量较少，但理解起来稍困难。

用法与 findComponentDownward 大同小异，就不再写用例了。

找到指定组件的兄弟组件——findBrothersComponents

代码如下：

```
// assist.js
// 由一个组件，找到指定组件的兄弟组件
function findBrothersComponents (context,
componentName, exceptMe = true) {
  let res = context.$parent.$children.filter(item
=> {
    return item.$options.name === componentName;
  });
  let index = res.findIndex(item => item._uid ===
context._uid);
  if (exceptMe) res.splice(index, 1);
  return res;
}
export { findBrothersComponents };
```

相比其它 4 个函数，findBrothersComponents 多了一个参数 exceptMe，是否把本身除外，默认是 true。寻找兄弟组件的方法，是先获取 context.$parent.$children，也就是父组件的全部子组件，这里面当前包含了本身，所有也会有第三个参数 exceptMe。Vue.js 在渲染组件时，都会给每个组件加一个内置的属性 _uid，这个 _uid 是不会重复的，借此我们可以从一系列兄弟组件中把自己排除掉。

举个例子，组件 A 是组件 B 的父级，在 B 中找到所有在 A 中的兄弟组件（也就是所有在 A 中的 B 组件）：

```vue
<!-- component-a.vue -->
<template>
  <div>
    组件 A
    <component-b></component-b>
  </div>
</template>
<script>
  import componentB from './component-b.vue';

  export default {
    name: 'componentA',
    components: { componentB }
  }
</script>
```

```
<!-- component-b.vue -->
<template>
  <div>
    组件 B
  </div>
</template>
<script>
  import { findBrothersComponents } from
'../utils/assist.js';

  export default {
    name: 'componentB',
    mounted () {
      const comsB = findBrothersComponents(this,
'componentB');
      console.log(comsB);  // ① []，空数组
    }
  }
</script>
```

在 ① 的位置，打印出的内容为空数组，原因是当前 A 中只有一个 B，而 findBrothersComponents 的第三个参数默认是 true，也就是将自己除外。如果在 A 中再写一个 B：

```
<!-- component-a.vue -->
<template>
  <div>
    组件 A
    <component-b></component-b>
    <component-b></component-b>
  </div>
</template>
```

这时就会打印出 [VueComponent]，有一个组件了，但要注意在控制台会打印两遍，因为在 A 中写了两个 B，而 console.log 是在 B 中定义的，所以两个都会执行到。如果你看懂了这里，那应该明白打印的两遍 [VueComponent]，分别是另一个 <component-b>（如果没有搞懂，要仔细琢磨琢磨哦）。

如果将 B 中 findBrothersComponents 的第三个参数设置为 false：

```
// component-b.vue
export default {
  name: 'componentB',
  mounted () {
    const comsB = findBrothersComponents(this, 'componentB', false);
    console.log(comsB);
  }
}
```

此时就会打印出 [VueComponent, VueComponent]，也就是包含自身了。

以上就是 5 个函数的详细介绍，get 到这 5 个，以后就再也不用担心组件通信了。

结语

只有你认真开发过 Vue.js 独立组件，才会明白这 5 个函数的强大之处。

扩展阅读

- ES6 Module 的语法

(http://es6.ruanyifeng.com/#docs/module)

注：本节部分代码参考 iView (https://github.com/iview/iview/blob/2.0/src/utils/assist.js)。