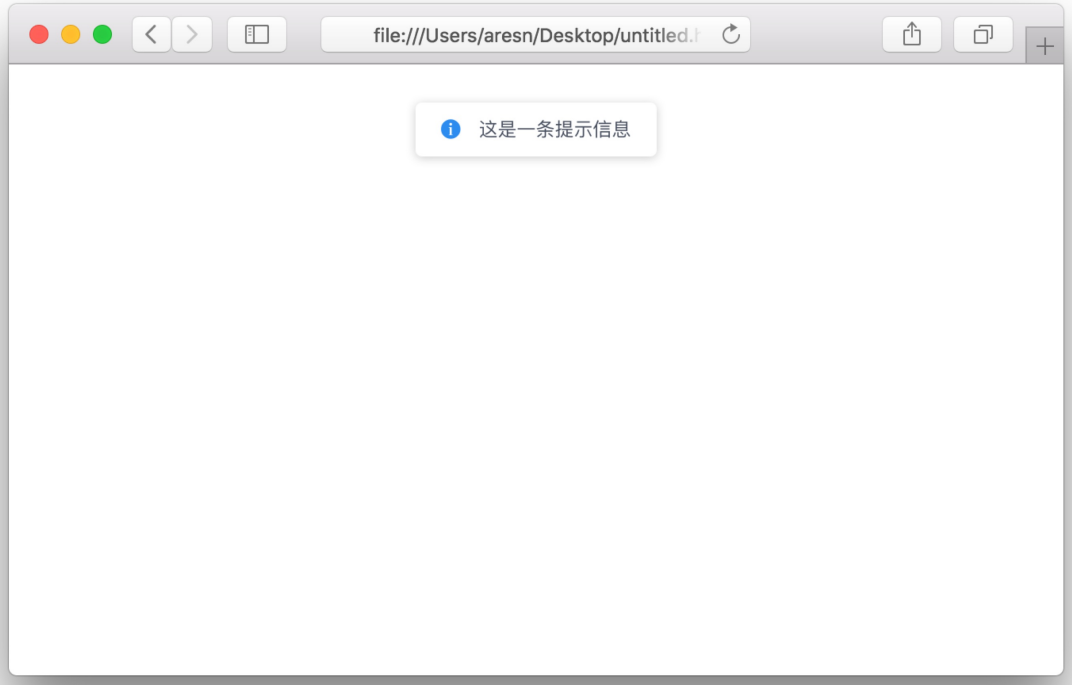
实战 4：全局提示组件——$Alert

有一种 Vue.js 组件，它不同于常规的组件，但组件结构本身很简单，比如下面的全局提示组件：



实现这样一个组件并不难，只需要简单的几行 div 和 css，但使用者可能要这样来显示组件：

```
<template>
  <div>
    <Alert v-if="show">这是一条提示信息</Alert>
    <button @click="show = true">显示</button>
  </div>
</template>
<script>
  import Alert from '../component/alert.vue';

  export default {
    components: { Alert },
    data () {
      return {
        show: false
      }
    }
  }
</script>
```

这样的用法，有以下缺点：

- 每个使用的地方，都得注册组件；
- 需要预先将 <Alert> 放置在模板中；
- 需要额外的 data 来控制 Alert 的显示状态；
- Alert 的位置，是在当前组件位置，并非在 body 下，有可能会被其它组件遮挡。

总之对使用者来说是很不友好的，那怎样才能优雅地实现这样一个组件呢？事实上，原生的 JavaScript 早已给出了答案：

```
// 全局提示
window.alert('这是一条提示信息');
// 二次确认
const confirm = window.confirm('确认删除吗？');
if (confirm) {
  // ok
} else {
  // cancel
}
```

所以，结论是：我们需要一个能用 JavaScript 调用组件的 API。

如果你使用过 iView 之类的组件库，一定对它内置的 $Message、$Notice、$Modal 等组件很熟悉，本节就来开发一个全局通知组件——$Alert。

1/3 先把组件写好

我们期望最终的 API 是这样的：

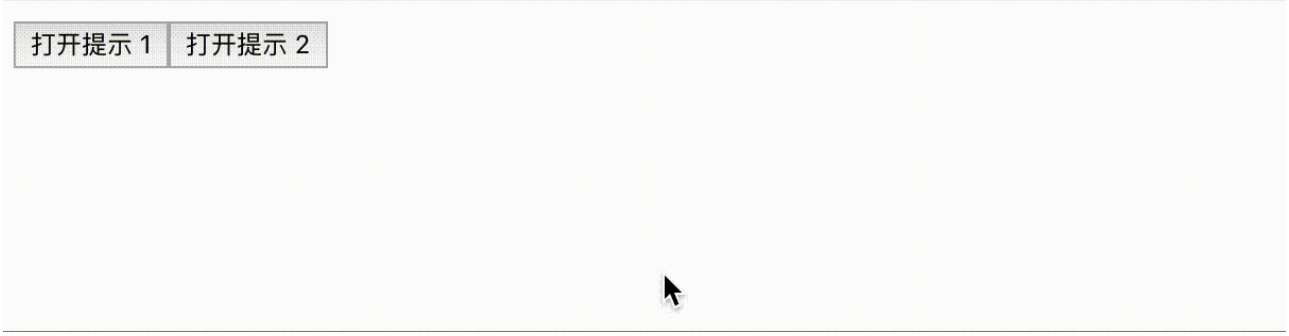
```
methods: {
  handleShow () {
    this.$Alert({
      content: '这是一条提示信息',
      duration: 3
    })
  }
}
```

this.$Alert 可以在任何位置调用，无需单独引入。该方法接收两个参数：

- content：提示内容；

- duration：持续时间，单位秒，默认 1.5 秒，到时间自动消失。

最终效果如下：



我们从最简单的入手，不考虑其它，先写一个基本的 Alert 组件。

在 src/component 下新建 alert 目录，并创建文件 alert.vue：

通知可以是多个，我们用一个数组 notices 来管理每条通知：

```
<!-- alert.vue -->
<template>
  <div class="alert">
    <div class="alert-main" v-for="item in
notices" :key="item.name">
      <div class="alert-content">{{ item.content
}}</div>
    </div>
  </div>
</template>
<script>
  export default {
    data () {
      return {
        notices: []
```

```
      }
    }
  }
</script>
<style>
  .alert{
    position: fixed;
    width: 100%;
    top: 16px;
    left: 0;
    text-align: center;
    pointer-events: none;
  }
  .alert-content{
    display: inline-block;
    padding: 8px 16px;
    background: #fff;
    border-radius: 3px;
    box-shadow: 0 1px 6px rgba(0, 0, 0, .2);
    margin-bottom: 8px;
  }
</style>
```

Alert 组件不同于常规的组件使用方式，它最终是通过 JS 来调用的，因此组件不用预留 props 和 events 接口。

接下来，只要给数组 notices 增加数据，这个提示组件就能显示内容了，我们先假设，最终会通过 JS 调用 Alert 的一个方法 add，并将 content 和 duration 传入进来：

```
<!-- alert.vue, 部分代码省略 -->
<script>
  let seed = 0;
```

```
  function getUuid() {
    return 'alert_' + (seed++);
  }

  export default {
    data () {
      return {
        notices: []
      }
    },
    methods: {
      add (notice) {
        const name = getUuid();

        let _notice = Object.assign({
          name: name
        }, notice);

        this.notices.push(_notice);

        // 定时移除，单位：秒
        const duration = notice.duration;
        setTimeout(() => {
          this.remove(name);
        }, duration * 1000);
      },
      remove (name) {
        const notices = this.notices;

        for (let i = 0; i < notices.length; i++)
{
          if (notices[i].name === name) {
```

```
            this.notices.splice(i, 1);
            break;
          }
        }
      }
    }
  }
</script>
```

在 add 方法中，给每一条传进来的提示数据，加了一个不重复的 name 字段来标识，并通过 setTimeout 创建了一个计时器，当到达指定的 duration 持续时间后，调用 remove 方法，将对应 name 的那条提示信息找到，并从数组中移除。

由这个思路，Alert 组件就可以无限扩展，只要在 add 方法中传递更多的参数，就能支持更复杂的组件，比如是否显示手动关闭按钮、确定 / 取消按钮，甚至传入一个 Render 函数都可以，完成本例后，读者可以尝试”改造“。

2/3 实例化封装

这一步，我们对 Alert 组件进一步封装，让它能够实例化，而不是常规的组件使用方法。实例化组件我们在第 8 节中介绍过，可以使用 Vue.extend 或 new Vue，然后用 $mount 挂载到 body 节点下。

在 src/components/alert 目录下新建 notification.js 文件：

```
// notification.js
import Alert from './alert.vue';
import Vue from 'vue';

Alert.newInstance = properties => {
  const props = properties || {};

  const Instance = new Vue({
    data: props,
    render (h) {
      return h(Alert, {
        props: props
      });
    }
  });

  const component = Instance.$mount();
  document.body.appendChild(component.$el);

  const alert = Instance.$children[0];

  return {
    add (noticeProps) {
      alert.add(noticeProps);
    },
    remove (name) {
      alert.remove(name);
    }
  }
};

export default Alert;
```

notification.js 并不是最终的文件，它只是对 alert.vue 添加了一个方法 newInstance。虽然 alert.vue 包含了 template、script、style 三个标签，并不是一个 JS 对象，那怎么能够给它扩展一个方法 newInstance 呢？事实上，alert.vue 会被 Webpack 的 vue-loader 编译，把 template 编译为 Render 函数，最终就会成为一个 JS 对象，自然可以对它进行扩展。

Alert 组件没有任何 props，这里在 Render Alert 组件时，还是给它加了 props，当然，这里的 props 是空对象 {}，而且即使传了内容，也不起作用。这样做的目的还是为了扩展性，如果要在 Alert 上添加 props 来支持更多特性，是要在这里传入的。不过话说回来，因为能拿到 Alert 实例，用 data 或 props 都是可以的。

在第 8 节已经解释过，`const alert = Instance.$children[0];`，这里的 alert 就是 Render 的 Alert 组件实例。在 newInstance 里，使用闭包暴露了两个方法 add 和 remove。这里的 add 和 remove 可不是 alert.vue 里的 add 和 remove，它们只是名字一样。

3/3 入口

最后要做的，就是调用 notification.js 创建实例，并通过 add 把数据传递过去，这是组件开发的最后一步，也是最终的入口。在 src/component/alert 下创建文件 alert.js：

```
// alert.js
import Notification from './notification.js';

let messageInstance;

function getMessageInstance () {
  messageInstance = messageInstance ||
Notification.newInstance();
  return messageInstance;
}

function notice({ duration = 1.5, content = '' })
{
  let instance = getMessageInstance();

  instance.add({
    content: content,
    duration: duration
  });
}

export default {
  info (options) {
    return notice(options);
  }
}
```

getMessageInstance 函数用来获取实例，它不会重复创建，如果 messageInstance 已经存在，就直接返回了，只在第一次调用 Notification 的 newInstance 时来创建实例。

alert.js 对外提供了一个方法 info，如果需要各种显示效果，比如成功的、失败的、警告的，可以在 info 下面提供更多的方法，比如 success、fail、warning 等，并传递不同参数让 Alert.vue 知道显示哪种状态的图标。本例因为只有一个 info，事实上也可以省略掉，直接导出一个默认的函数，这样在调用时，就不用 this.$Alert.info() 了，直接 this.$Alert()。

来看一下显示一个信息提示组件的流程：

最后把 alert.js 作为插件注册到 Vue 里就行，在入口文件 src/main.js中，通过 prototype 给 Vue 添加一个实例方法：

```
// src/main.js
import Vue from 'vue'
import App from './App.vue'
import router from './router'
import Alert from
'../src/components/alert/alert.js'

Vue.config.productionTip = false

Vue.prototype.$Alert = Alert

new Vue({
  router,
  render: h => h(App)
}).$mount('#app')
```

这样在项目任何地方，都可以通过 this.$Alert 来调用 Alert 组件了，我们创建一个 alert 的路由，并在 src/views 下创建页面 alert.vue：

```
<!-- src/views/alert.vue -->
<template>
  <div>
    <button @click="handleOpen1">打开提示1</button>
    <button @click="handleOpen2">打开提示2</button>
  </div>
</template>
<script>
  export default {
    methods: {
      handleOpen1 () {
        this.$Alert.info({
          content: '我是提示信息 1'
        });
      },
      handleOpen2 () {
        this.$Alert.info({
          content: '我是提示信息 2',
          duration: 3
        });
      }
    }
  }
</script>
```

duration 如果不传入，默认是 1.5 秒。

以上就是全局通知组件的全部内容。

友情提示

本示例算是一个 MVP（最小化可行方案），要开发一个完善的全局通知组件，还需要更多可维护性和功能性的设计，但离不开本例的设计思路。以下几点是同类组件中值得注意的：

1. Alert.vue 的最外层是有一个 .alert 节点的，它会在第一次调用 `$Alert` 时，在 body 下创建，因为不在 `<router-view>` 内，它不受路由的影响，也就是说一经创建，除非刷新页面，这个节点是不会消失的，所以在 alert.vue 的设计中，并没有主动销毁这个组件，而是维护了一个子节点数组 `notices`。
2. .alert 节点是 `position: fixed` 固定的，因此要合理设计它的 `z-index`，否则可能被其它节点遮挡。
3. notification.js 和 alert.vue 是可以复用的，如果还要开发其它同类的组件，比如二次确认组件 `$Confirm`, 只需要再写一个入口 `confirm.js`，并将 `alert.vue` 进一步封装，将 `notices` 数组的循环体写为一个新的组件，通过配置来决定是渲染 Alert 还是 Confirm，这在可维护性上是友好的。
4. 在 notification.js 的 new Vue 时，使用了 Render 函数来渲染 alert.vue，这是因为使用 template 在 runtime 的 Vue.js 版本下是会报错的。
5. 本例的 content 只能是字符串，如果要显示自定义的内容，除了用 `v-html` 指令，也能用 Functional Render（之后章节会介绍）。

结语

Vue.js 的精髓是组件，组件的精髓是 JavaScript。将 JavaScript 开发中的技巧结合 Vue.js 组件，就能玩出不一样的东西。

注：本节部分代码参考 iView (https://github.com/iview/iview/tree/2.0/src/components/ba