Vue 高级指南

Vue 插件 plugin

插件通常用来为 Vue 添加全局功能。插件的功能范围没有严格的限制——一般有下面几种：

1. 添加全局方法或者 property。如：vue-custom-element
2. 添加全局资源：指令/过滤器/过渡等。如 vue-touch
3. 通过全局混入来添加一些组件选项。如 vue-router
4. 添加 Vue 实例方法，通过把它们添加到 `Vue.prototype` 上实现。
5. 一个库，提供自己的 API，同时提供上面提到的一个或多个功能。如 vue-router

对于每个我们需要使用的插件来说，我们需要在具体的业务逻辑前，也就是实例化 Vue 前，使用 Vue.use 来使用插件。

```
Vue.use(MyPlugin)

new Vue({
  // ...组件选项
})

事实上等同于 `Myplugin.install(Vue)`
```

对于我们开发插件对象来说，我们需要给这个对象下暴露一个 install 方法。也就是说只要一个「对象」有 install 方法，同时它的第一个参数为 Vue 构造函数，第二个参数为一个 options，那么他就是一个合法的 Vue 插件。

```
// myPlugin 只需要是一个对象即可
let MyPlugin = {}
let myPlugin = function() {}
MyPlugin.install = function (Vue, options) {}
```

我们可以使用插件做很多自动化的事情，某些情况下我们可以比「组件化」能够做更多细粒度的封装。

```
// 1. 添加全局静态方法 myGlobalMethod
Vue.myGlobalMethod = function () {}

// 2. 为组件增加 created 生命周期
Vue.mixin({created: function () {}})

// 3. 添加实例方法，this.$myMethod
Vue.prototype.$myMethod = function (methodOptions) {}
```

我们可以使用插件的形式，更加方便的封装组件和通用的业务逻辑，甚至实现我们自己的一些生命周期。

Vue 混合 mixin

混入 (mixin) 提供了一种非常灵活的方式，来分发 Vue 组件中的可复用功能。一个混入对象可以包含任意组件选项。当组件使用混入对象时，所有混入对象的选项将被“混合”进入该组件本身的选项。

我们的 mixin 可以在组件级别和全局两种方式进行混入

```
// 定义一个混入对象
var myMixin = {
  created: function () {
    this.hello()
  },
  methods: {
    hello: function () {
      console.log('hello from mixin!')
    }
  }
}

// 定义一个使用混入对象的组件
var Component = Vue.extend({
  mixins: [myMixin]
})

var component = new Component() // => "hello from mixin!"
```

全局对象：

```
Vue.mixin({
  created: function () {
    var myOption = this.$options.myOption
    if (myOption) {
      console.log(myOption)
    }
  }
})
```

对于 merge 的策略，主要分为以下几类：

- data 部分会在内部进行递归的处理，也就是 deepMerge，遇到同名 key 以组件内部的为准
- 生命周期函数会进行一个数组处理，所有 mixin 进来的生命周期都会执行，并且 mixin 传入的生命周期会在组件内部生命周期之前执行。

自定义选项将使用默认策略，即简单地覆盖已有值。如果想让自定义选项以自定义逻辑合并，可以向 `Vue.config.optionMergeStrategies` 添加一个函数：

```
Vue.config.optionMergeStrategies.myOption = function (toVal, fromVal) {
  // 返回合并后的值
}
```

Vue 插槽 slot

我们可以通过 slot 也就是插槽系统，来承载父组件的子元素。熟悉 react 的同学肯定知道，这一步其实就是 react 当中的 `this.props.children`

对于插槽元素来说，他会将 children 的内容在插槽占位符的地方渲染出来。

```
// parent.vue
<child>
  this.props.children
</child>

// child.vue
<div><slot /></div>
```

插槽是一个比较重要的特性，使用插槽能让我们实现更多通用功能的封装。

- 对于插槽来说，父级模板里的所有内容都是在父级作用域中编译的；子模板里的所有内容都是在子作用域中编译的。
- 对于插槽来说，我们可以使用模版中默认提供模版的形式，来提供一个渲染的默认值

同时我们也可以给插槽起一个名字，来应对一个模版有多个插槽的情况，默认情况下，只有一个插槽的组件，他的插槽名称默认为 default。

比如在下面的例子中，我们希望当前 container 在调用时，能够在调用方渲染出来 header、main 和 footer，我们通过传入 name 的形式，就能够控制不同插槽的渲染。

```
<div class="container">
  <header>
    <slot name="header"></slot>
  </header>
  <main>
    <slot></slot>
  </main>
  <footer>
    <slot name="footer"></slot>
  </footer>
</div>
```

对于调用组件来渲染插槽的部分，我们需要使用 template 的 v-slot 标签来指定具体某个模版的名称。

```
<base-layout>
  <template v-slot:header>
    <h1>Here might be a page title</h1>
  </template>

  <p>A paragraph for the main content.</p>
  <p>And another one.</p>

  <template v-slot:footer>
    <p>Here's some contact info</p>
  </template>
</base-layout>
```

在调用组件的内部，如果我们不提供 template 的名字时，默认就会将它作为 default 的 template 名，那么最终这部分内容还是会渲染到默认的没有提供名称的组件中。

对于解析的流程，我们可以分为以下的几个步骤：

1. 解析父组件的 vDOM，以配置的形式解析出来每个数组元素的内容
2. 将父组件内部的内容转移至子组件 options 的 renderChildren 上（initInternalComponent）
3. 子组件转移 renderChildren 到 vm.$slots 上 （initRender）
4. 解析子组件 vDOM，将模版中的 slot 占位符替换为 _t 函数，最终 _t 执行，从 vm.$slots 中替换渲染内容（renderMixin）