**Kendo UI for Vue使用教程：如何创建动态加载的组件（下）**

**动态加载**

就目前而言，已经在如何将组件放置到用户屏幕上增加了价值，但是仍然可以看到它的问题，仍需将组建手动添加到应用程序中，并且即使浏览站点的用户永远看不到它们，也将全部加载这些组件。

采取一些步骤，使这些组件根据浏览器的要求动态加载，这将使应用程序包更小，因为webpack会将它们放置在外部块中。

对于这种方法，我们将修改计算属性来返回一个函数，该函数将动态地导入组件。

<script>

const Guest = () => import("./components/Guest")

const User = () => import("./components/User")

const Member = () => import("./components/Member")

export default {

data() {

return {

user: null

};

},

computed: {

userComponent() {

if (!this.user) return Guest;

if (this.user && !this.user.subscription)

return User;

if (this.user && this.user.subscription)

return Member;

}

}

};

</script>

首先请注意，顶部的所有import语句现在都消失了。 这不再是必需的，因为组件将根据需要动态且异步地加载。 但是，我们选择在顶部创建const变量来导入它们。

如果直接在计算属性中返回import语句，则每次执行该语句都会返回一个新函数。如果以后您想使用<keep-alive>元素，则不会保留您的状态。

userComponent计算属性进行了重构，现在它分别返回带有import语句的函数，此函数本身返回一个将分解为组件的promise，即…drumroll…

如果您不关心没有返回语法的arrow函数，可以这样重写return语句：

const User = function() {

return import('./components/Guest')

}

关于<component>值得一提的小问题是<keep-alive>的特殊元素。

有时您会希望用户在应用程序内的不同元素之间切换。 想象一下，如果在我们的演示场景中，该成员还可以访问一个按钮，以切换到“用户”视图来使用某些其他功能。

您只需在计算的属性中添加一些额外的逻辑即可通过单击按钮在它们之间切换，但是当用户开始使用每个组件并来回跳转时，它们将遇到一个非常特殊的问题 – Vue是 当用户在组件之间切换时，销毁并重新安装组件，本地存储在组件中的任何状态都将完全丢失。

对于要使组件保持激活状态的这类场景，您可以在dev-belt上使用此工具。

<keep-alive>

<component :is="userComponent"/>

</keep-alive>

通过简单地将<component>包装在<keep-alive>中，Vue将开始缓存并保留这些组件在屏幕上交换时的状态。

如果直接在计算属性中返回导入函数，由于JavaScript比较的工作方式，该状态将不会被缓存。 简而言之，功能将不完全相同。

设置App.vue，以便可以轻松地在要测试的组件之间切换。

<template>

<div id="app">

<keep-alive>

<component :is="userComponent"/>

</keep-alive>

<br>

<button @click="user = null">Guest</button>

<button @click="user = {}">User</button>

<button @click="user = {subscription: true}">Member</button>

</div>

</template>

<script>

const Guest = () => import("./components/Guest");

const User = () => import("./components/User");

const Member = () => import("./components/Member");

export default {

data() {

return {

user: null

};

},

computed: {

userComponent() {

if (!this.user) return Guest;

if (this.user && !this.user.subscription) return User;

if (this.user && this.user.subscription) return Member;

}

}

};

</script>

<style>

#app {

font-family: "Avenir", Helvetica, Arial, sans-serif;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

text-align: center;

color: #2c3e50;

margin-top: 60px;

}

</style>

此外，对User.vue进行一些更改来添加基本的内部状态进行测试。

<template>

<div>

<div>User component {{ count }}</div>

<br>

<button @click="count++">MOAR!</button>

</div>

</template>

<script>

export default {

name:'User',

data() {

return {

count: 0

}

}

}

</script>

如果您现在单击MOAR按钮并增加计数器并在不同组件之间切换，则应该能够看到该状态已正确保存给用户。