Plugins

Introduction

Plugins are self-contained code that usually add app-level functionality to Vue. This is how we install a plugin:

```js
import { createApp } from 'vue'

const app = createApp({})

app.use(myPlugin, {
  /* optional options */
})
```

A plugin is defined as either an object that exposes an `install()` method, or simply a function that acts as the install function itself. The install function receives the app instance along with additional options passed to `app.use()`, if any:

```js
const myPlugin = {
  install(app, options) {
    // configure the app
  }
}
```

There is no strictly defined scope for a plugin, but common scenarios where plugins are useful include:

1. Register one or more global components or custom directives with `app.component()` and `app.directive()`.
2. Make a resource injectable throughout the app by calling `app.provide()`.
3. Add some global instance properties or methods by attaching them to `app.config.globalProperties`.

Writing a Plugin

In order to better understand how to create your own Vue.js plugins, we will create a very simplified version of a plugin that displays `i18n` (short for Internationalization) strings.

Let's begin by setting up the plugin object. It is recommended to create it in a separate file and export it, as shown below to keep the logic contained and separate.

JS plugins/i18n.js

```js
export default {
  install: (app, options) => {
    // Plugin code goes here
  }
}
```

We want to create a translation function. This function will receive a dot-delimited `key` string, which we will use to look up the translated string in the user-provided options. This is the intended usage in templates:

```template
<h1>{{ $translate('greetings.hello') }}</h1>
```

Since this function should be globally available in all templates, we will make it so by attaching it to `app.config.globalProperties` in our plugin:

JS plugins/i18n.js

```js
export default {
  install: (app, options) => {
    // inject a globally available $translate() method
    app.config.globalProperties.$translate = (key) => {
      // retrieve a nested property in `options`
      // using `key` as the path
      return key.split('.').reduce((o, i) => {
        if (o) return o[i]
      }, options)
    }
  }
}
```

Our `$translate` function will take a string such as `greetings.hello`, look inside the user provided configuration and return the translated value.

```js
import i18nPlugin from './plugins/i18n'

app.use(i18nPlugin, {
  greetings: {
    hello: 'Bonjour!'
  }
})
```

Now, our initial expression `$translate('greetings.hello')` will be replaced by `Bonjour!` at runtime.

See also: Augmenting Global Properties TS

> ⓘ TIP
>
> Use global properties scarcely, since it can quickly become confusing if too many global properties injected by different plugins are used throughout an app.

Provide / Inject with Plugins

Plugins also allow us to use `provide` to give plugin users access to a function or attribute. For example, we can allow the application to have access to the `options` parameter to be able to use the translations object.

JS plugins/i18n.js

```js
export default {
  install: (app, options) => {
    app.provide('i18n', options)
  }
}
```

Plugin users will now be able to inject the plugin options into their components using the `i18n` key:

```vue
<script setup>
import { inject } from 'vue'

const i18n = inject('i18n')

console.log(i18n.greetings.hello)
</script>
```

If you further want to build and publish your plugin for others to use, see Vite's section on Library Mode.

Edit this page on GitHub

Previous
Custom Directives

Next
Transition