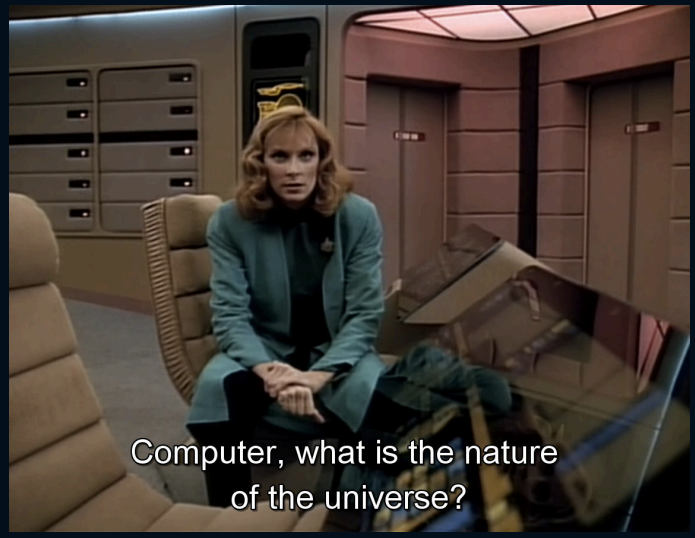
Dependency Injection in Vue

Ask ChatGPT:

What Vue features are most underutilized by Vue devs?

1. Teleport
2. Suspense
3. defineAsyncComponent
4. Provide / Inject 🌟
5. Custom Directives



What is Dependency Injection?

Where an object or function **receives** other objects or functions that it requires, as opposed to creating them itself

→ Let's say you're doing some math

→ "I'd like a calculator"

- With dependency injection, someone hands you a calculator
- Without dependency injection, you have to build your own calculator

Examples of Dependency Injection in Vue

- Props
- Slots
- Shared data stores (pinia, etc)
- Provide / Inject

`provide` and `inject`

Parent.vue

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

provide('message', 'hello')
</script>
```

Child.vue (or any descendant)

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

// Can set a default value, if nothing provided
const msg = inject('message', 'default message')
</script>
```

App-level provide

```vue
import { createApp } from 'vue'
const app = createApp({})
app.provide('message', 'hello!')
```

With TypeScript

```vue
<script setup lang="ts">
import { provide, inject } from 'vue'

provide<string>('message', 'hello')
provide<string>('message', 5)
```

> Argument of type 'number' is not assignable to parameter

```vue

const msg1 = inject<string>('message')
const msg2 = inject<string>('message', 'default')
</script>
```

Providing reactive data

Parent.vue

```
import { provide, ref, readonly } from 'vue'

const dynamicMsg = ref('hello')
provide('message', readonly(dynamicMsg))

// Providing an update function
// keeps the provider in control
const updateMsg = (newMsg) => {
  dynamicMsg.value = newMsg
}
provide('updateMsg', updateMsg)
```

Child.vue

```
import { inject } from 'vue'

const msg = inject('message')
const updateMsg = inject('updateMsg')

updateMsg('live long and prosper')
```

Injection Key Naming Collisions

```
import { provide, inject } from 'vue'

// Parent.vue
provide('message', 'hello')

// Grandparent.vue
provide('message', 'greetings')

// plugin
provide('message', 'nuq'neH')

// Child.vue ... what will we get?
const msg = inject('message')
```

Symbol Injection Keys

```
import { provide, inject } from 'vue'

// injectionKeys.ts
export const messageKey = Symbol()

// Parent.vue
provide(messageKey, 'hello')

// Child.vue
const msg = inject(messageKey)

// Symbols are great, but how do we get types?
const str: string | undefined = msg
```

Typed Symbol Injection Keys

```ts
import { provide, inject } from 'vue'
import type { InjectionKey } from 'vue'

// injectionKeys.ts
export const messageKey = Symbol() as InjectionKey<string>

// Parent.vue
provide(messageKey, 'hello')

// type-checking provide()
provide(messageKey, 5)
```

> Argument of type 'number' is not assignable to parameter of type 'string'.

```ts
// Child.vue
const msg = inject(messageKey)

const msgWithDefault = inject(messageKey, 'default value')
```

Putting it all together

Let's make a Vue plugin 🔧

```ts
// my-plugin
import type { App, InjectionKey, Ref } from 'vue'
import { inject, readonly, ref } from 'vue'

const messageKey = Symbol() as InjectionKey<Ref<string>>
const updateMsgKey = Symbol() as InjectionKey<(msg: strin

const message = ref('hello')
const updateMsg = (newMsg: string) => { message.value = n

export const myPlugin = {
    install (app: App) {
        app.provide(messageKey, readonly(message))
        app.provide(updateMsgKey, updateMsg)
    }
}
export const useMessage = () => {
  return inject(messageKey)
}
export const useUpdateMsg = () => {
  return inject(updateMsgKey)
}
```

```ts
// main.ts
import { createApp } from 'vue'
import App from './App.vue'
import { myPlugin } from 'my-plugin'

const app = createApp(App).use(myPlugin).mount('#app')
```

```vue
<!-- Component.vue -->
<script setup lang="ts">
import { useMessage, useUpdateMsg } from 'my-plugin'

const msg = useMessage()
const updateMsg = useUpdateMsg()
</script>

<template>
  <div>
    {{ msg }}
    <button @click="updateMsg('engage')">Engage</button>
  </div>
</template>
```

When you may want to use provide / inject

- Sharing data between a component and its descendants
- When you have a large number of props being passed through many components
- When you don't have access to a store (pinia, etc)
- Plugins
- Component libraries

When you may NOT want to use provide / inject

- When you can use simple props
- When you have access to a store (pinia, etc)
- When the relationship between provider and injector is too complex

Real world examples using provide / inject

- vue-router
- pinia
- vue-i18n
- vuetify
- quasar
- primevue
- vueuse

The End