Component Options

`asyncData` and `fetch` component options

Nuxt 3 provides new options for fetching data from an API.

Isomorphic Fetch

In Nuxt 2 you might use `@nuxtjs/axios` or `@nuxt/http` to fetch your data - or just the polyfilled global `fetch` .

In Nuxt 3 you can use a globally available `fetch` method that has the same API as the Fetch API or `$fetch` method which is using unjs/ofetch. It has a number of benefits, including:

1. It will handle 'smartly' making direct API calls if it's running on the server, or making a client-side call to your API if it's running on the client. (It can also handle calling third-party APIs.)
2. Plus, it comes with convenience features including automatically parsing responses and stringifying data.

You can read more about direct API calls or fetching data.

Using Composables

Nuxt 3 provides new composables for fetching data: `useAsyncData` and `useFetch` . They each have 'lazy' variants (`useLazyAsyncData` and `useLazyFetch`), which do not block client-side navigation.

In Nuxt 2, you'd fetch your data in your component using a syntax similar to:

```
export default {
  async asyncData({ params, $http }) {
    const post = await $http.$get(`https://api.nuxtjs.dev/posts/${params.id}`)
    return { post }
  },
  // or alternatively
  fetch () {
    this.post = await $http.$get(`https://api.nuxtjs.dev/posts/${params.id}`)
  }
}
```

Within your methods and templates, you could use the `post` variable similar how you'd use any other piece of data provided by your component.

With Nuxt 3, you can perform this data fetching using composables in your `setup()` method or `<script setup>` tag:

```
<script setup lang="ts">
// Define params wherever, through `defineProps()`, `useRoute()`, etc.
const { data: post, refresh } = await useAsyncData('post', () => $fetch(`https://api.nuxtjs.dev/po
// Or instead - useFetch is a convenience wrapper around useAsyncData when you're just performing
const { data: post, refresh } = await useFetch(`https://api.nuxtjs.dev/posts/${params.id}`)
</script>
```

You can now use `post` inside of your Nuxt 3 template, or call `refresh` to update the data.

Despite the names, `useFetch` is not a direct replacement of the `fetch()` hook. Rather, `useAsyncData` replaces both hooks and is more customizable; it can do more than simply fetching data from an endpoint. `useFetch` is a convenience wrapper around `useAsyncData` for simply fetching data from an endpoint.

Migration

1. Replace the `asyncData` hook with `useAsyncData` or `useFetch` in your page/component.
2. Replace the `fetch` hook with `useAsyncData` or `useFetch` in your component.

head

See meta tag migration.

`key`

You can now define a key within the `definePageMeta` compiler macro.

```diff
pages/index.vue
- <script>
- export default {
-   key: 'index'
-   // or a method
-   // key: route => route.fullPath
- }
+ <script setup>
+ definePageMeta({
+   key: 'index'
+   // or a method
+   // key: route => route.fullPath
+ })
  </script>
```

Migration

1. Migrate `key` from component options to `definePageMeta`.

`layout`

See layout migration.

`loading`

This feature is not yet supported in Nuxt 3.

middleware

See middleware migration.

scrollToTop

This feature is not yet supported in Nuxt 3. If you want to overwrite the default scroll behavior of `vue-router`, you can do so in `~/app/router.options.ts` (see docs) for more info.

transition

See layout migration.

validate

The validate hook in Nuxt 3 only accepts a single argument, the `route`. Just as in Nuxt 2, you can return a boolean value. If you return false and another match can't be found, this will mean a 404. You can also directly return an object with `statusCode` / `statusMessage` to respond immediately with an error (other matches will not be checked).

pages/users/[id].vue

```diff
- <script>
- export default {
-   async validate({ params }) {
-     return /^\d+$/.test(params.id)
-   }
- }
+ <script setup>
+ definePageMeta({
+   validate: async (route) => {
+     const nuxtApp = useNuxtApp()
+     return /^\d+$/.test(route.params.id)
+   }
+ })
  </script>
```

watchQuery

This is not supported in Nuxt 3. Instead, you can directly use a watcher to trigger refetching data.

pages/users/[id].vue

```
<script setup lang="ts">
const route = useRoute()
const { data, refresh } = await useFetch('/api/user')
watch(() => route.query, () => refresh())
</script>
```

Edit on Github