useAsyncData

Within your pages, components, and plugins you can use useAsyncData to get access to data that resolves asynchronously.

`useAsyncData` is a composable meant to be called directly in a setup function, plugin, or route middleware. It returns reactive composables and handles adding responses to the Nuxt payload so they can be passed from server to client without re-fetching the data on client side when the page hydrates.

Type

Signature

```ts
function useAsyncData<DataT, DataE>(
  handler: (nuxtApp?: NuxtApp) => Promise<DataT>,
  options?: AsyncDataOptions<DataT>
): AsyncData<DataT, DataE>
function useAsyncData<DataT, DataE>(
  key: string,
  handler: (nuxtApp?: NuxtApp) => Promise<DataT>,
  options?: AsyncDataOptions<DataT>
): Promise<AsyncData<DataT, DataE>

type AsyncDataOptions<DataT> = {
  server?: boolean
  lazy?: boolean
  immediate?: boolean
  default?: () => DataT | Ref<DataT> | null
  transform?: (input: DataT) => DataT
  pick?: string[]
  watch?: WatchSource[]
```

```ts
}

type AsyncData<DataT, ErrorT> = {
  data: Ref<DataT | null>
  pending: Ref<boolean>
  refresh: (opts?: AsyncDataExecuteOptions) => Promise<void>
  execute: (opts?: AsyncDataExecuteOptions) => Promise<void>
  error: Ref<ErrorT | null>
  status: Ref<AsyncDataRequestStatus>
};

interface AsyncDataExecuteOptions {
  dedupe?: boolean
}

type AsyncDataRequestStatus = 'idle' | 'pending' | 'success' | 'error'
```

Params

- **key**: a unique key to ensure that data fetching can be properly de-duplicated across requests. If you do not provide a key, then a key that is unique to the file name and line number of the instance of `useAsyncData` will be generated for you.
- **handler**: an asynchronous function that must return a truthy value (for example, it should not be `undefined` or `null`) or the request may be duplicated on the client side
- **options**:
  - *server*: whether to fetch the data on the server (defaults to `true`)
  - *lazy*: whether to resolve the async function after loading the route, instead of blocking client-side navigation (defaults to `false`)
  - *immediate*: when set to `false`, will prevent the request from firing immediately. (defaults to `true`)
  - *default*: a factory function to set the default value of the `data`, before the async function resolves - useful with the `lazy: true` or `immediate: false` option
  - *transform*: a function that can be used to alter `handler` function result after resolving
  - *pick*: only pick specified keys in this array from the `handler` function result
  - *watch*: watch reactive sources to auto-refresh

Under the hood, `lazy: false` uses `<Suspense>` to block the loading of the route before the data has been fetched. Consider using `lazy: true` and implementing a loading state instead for a snappier user experience.

Return Values

- **data**: the result of the asynchronous function that is passed in.
- **pending**: a boolean indicating whether the data is still being fetched.
- **refresh**/**execute**: a function that can be used to refresh the data returned by the `handler` function.
- **error**: an error object if the data fetching failed.
- **status**: a string indicating the status of the data request (`"idle"` , `"pending"` , `"success"` , `"error"`).

By default, Nuxt waits until a `refresh` is finished before it can be executed again.

If you have not fetched data on the server (for example, with `server: false`), then the data *will not* be fetched until hydration completes. This means even if you await `useAsyncData` on the client side, `data` will remain `null` within `<script setup>` .

Example

```
const { data, pending, error, refresh } = await useAsyncData(
  'mountains',
  () => $fetch('https://api.nuxtjs.dev/mountains')
)
```

Example with watching params change

The built-in `watch` option allows automatically rerunning the fetcher function when any changes are detected.

```
const page = ref(1)
const { data: posts } = await useAsyncData(
```

```
  'posts',
  () => $fetch('https://fakeApi.com/posts', {
    params: {
      page: page.value
    }
  }), {
    watch: [page]
  }
)
```

`useAsyncData` is a reserved function name transformed by the compiler, so you should not name your own function `useAsyncData`.

👉 Read more in Docs > Getting Started > Data Fetching.

Edit on Github