



useFetch

This composable provides a convenient wrapper around `useAsyncData` and `$fetch`. It automatically generates a key based on URL and fetch options, provides type hints for request url based on server routes, and infers API response type.

`useFetch` 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

```
function useFetch<DataT, ErrorT>(  
  url: string | Request | Ref<string | Request> | () => string | Request,  
  options?: UseFetchOptions<DataT>  
) : Promise<AsyncData<DataT, ErrorT>>  
  
type UseFetchOptions<DataT> = {  
  key?: string  
  method?: string  
  query?: SearchParams  
  params?: SearchParams  
  body?: RequestInit['body'] | Record<string, any>  
  headers?: Record<string, string> | [key: string, value: string][] | Headers  
  baseURL?: string  
  server?: boolean  
  lazy?: boolean  
  immediate?: boolean  
  default?: () => DataT
```

Signature

```

transform?: (input: DataT) => DataT
pick?: string[]
watch?: WatchSource[] | false
}

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

- **URL:** The URL to fetch.
- **Options (extends unjs/ofetch options & AsyncDataOptions):**
 - `method` : Request method.
 - `query` : Adds query search params to URL using ufo
 - `params` : Alias for `query`
 - `body` : Request body – automatically stringified (if an object is passed).
 - `headers` : Request headers.
 - `baseUrl` : Base URL for the request.

All fetch options can be given a `computed` or `ref` value. These will be watched and new requests made automatically with any new values if they are updated.

- **Options (from `useAsyncData`):**

- **key** : a unique key to ensure that data fetching can be properly de-duplicated across requests, if not provided, it will be generated based on the static code location where `useAsyncData` is used.
- **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 an array of reactive sources and auto-refresh the fetch result when they change. Fetch options and URL are watched by default. You can completely ignore reactive sources by using `watch: false` . Together with `immediate: false` , this allows for a fully-manual `useFetch` .

If you provide a function or ref as the `url` parameter, or if you provide functions as arguments to the `options` parameter, then the `useFetch` call will not match other `useFetch` calls elsewhere in your codebase, even if the options seem to be identical. If you wish to force a match, you may provide your own key in `options` .

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 `useFetch` on client-side, `data` will remain null within `<script setup>` .

Example

```
const route = useRoute()

const { data, pending, error, refresh } = await useFetch(`https://api.nuxtjs.dev/mountains/${route.params.id}`, {
  pick: ['title']
})
```

Adding Query Search Params:

Using the `query` option, you can add search parameters to your query. This option is extended from `unjs/ofetch` and is using `unjs/ufo` to create the URL. Objects are automatically stringified.

```
const param1 = ref('value1')
const { data, pending, error, refresh } = await useFetch('https://api.nuxtjs.dev/mountains', {
  query: { param1, param2: 'value2' }
})
```

Results in `https://api.nuxtjs.dev/mountains?param1=value1¶m2=value2`

Using interceptors:


```
const { data, pending, error, refresh } = await useFetch('/api/auth/login', {
  onRequest({ request, options }) {
    // Set the request headers
    options.headers = options.headers || {}
    options.headers.authorization = '...'
  },
  onRequestError({ request, options, error }) {
    // Handle the request errors
  },
  onResponse({ request, response, options }) {
    // Process the response data
    localStorage.setItem('token', response._data.token)
  },
  onResponseError({ request, response, options }) {
    // Handle the response errors
  }
})
```

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

 Read and edit a live example in [Docs > Examples > Advanced > Use Custom Fetch Composable](#).

 Read more in [Docs > Getting Started > Data Fetching](#).

 Read and edit a live example in [Docs > Examples > Features > Data Fetching](#).

 [Edit on Github](#)



© 2016-2023 Nuxt - MIT
License

Enterprise

Design Kit
Nuxt Studio

NuxtLabs

