getStaticProps

Version History

Version	Changes

`v12.1.0` | On-demand Incremental Static Regeneration added (Beta). |
`v10.0.0` | `locale`, `locales`, `defaultLocale`, and `notFound` options added. |
`v10.0.0` | `fallback: 'blocking'` return option added. |
`v9.5.0` | Stable Incremental Static Regeneration |
`v9.3.0` | `getStaticProps` introduced. |

Exporting a function called `getStaticProps` will pre-render a page at build time using the props returned from the function:

```
export async function getStaticProps(context) {
  return {
    props: {}, // will be passed to the page component as props
  }
}
```

You can import modules in top-level scope for use in `getStaticProps`. Imports used will **not be bundled for the client-side**. This means you can write **server-side code directly in `getStaticProps`**, including fetching data from your database.

Context parameter

The `context` parameter is an object containing the following keys:

- `params` contains the route parameters for pages using dynamic routes. For example, if the page name is `[id].js` , then `params` will look like `{ id: ... }`. You should use this together with `getStaticPaths`, which we'll explain later.
- `preview` is `true` if the page is in the Preview Mode and `undefined` otherwise.
- `previewData` contains the preview data set by `setPreviewData`.
- `locale` contains the active locale (if enabled).
- `locales` contains all supported locales (if enabled).
- `defaultLocale` contains the configured default locale (if enabled).

getStaticProps return values

The `getStaticProps` function should return an object containing either `props`, `redirect`, or `notFound` followed by an **optional** `revalidate` property.

props

The `props` object is a key-value pair, where each value is received by the page component. It should be a serializable object so that any props passed, could be serialized with `JSON.stringify`.

```
export async function getStaticProps(context) {
  return {
    props: { message: `Next.js is awesome` }, // will be passed to the page component as pro
  }
}
```

revalidate

The `revalidate` property is the amount in seconds after which a page re-generation can occur (defaults to `false` or no revalidation).

```
// This function gets called at build time on server-side.
// It may be called again, on a serverless function, if
// revalidation is enabled and a new request comes in
export async function getStaticProps() {
  const res = await fetch('https://.../posts')
  const posts = await res.json()

  return {
    props: {
      posts,
    },
    // Next.js will attempt to re-generate the page:
    // - When a request comes in
    // - At most once every 10 seconds
    revalidate: 10, // In seconds
  }
}
```

Learn more about Incremental Static Regeneration

notFound

The `notFound` boolean allows the page to return a `404` status and 404 Page. With `notFound: true`, the page will return a `404` even if there was a successfully generated page before. This is meant to support use cases like user-generated content getting removed by its author. Note, `notFound` follows the same `revalidate` behavior described here

```
export async function getStaticProps(context) {
  const res = await fetch(`https://.../data`)
  const data = await res.json()
  if (!data) {
    return {
      notFound: true,
    }
  }

  return {
    props: { data }, // will be passed to the page component as props
  }
}
```

> **Note**: `notFound` is not needed for `fallback: false` mode as only paths returned from `getStaticPaths` will be pre-rendered.

`redirect`

The `redirect` object allows redirecting to internal or external resources. It should match the shape of `{ destination: string, permanent: boolean }`.

In some rare cases, you might need to assign a custom status code for older `HTTP` clients to properly redirect. In these cases, you can use the `statusCode` property instead of the `permanent` property, **but not both**. You can also set `basePath: false` similar to redirects in `next.config.js`.

```
export async function getStaticProps(context) {
  const res = await fetch(`https://...`)
  const data = await res.json()

  if (!data) {
    return {
      redirect: {
        destination: '/',
        permanent: false,
        // statusCode: 301
      },
    }
  }

  return {
    props: { data }, // will be passed to the page component as props
  }
}
```

If the redirects are known at build-time, they should be added in `next.config.js` instead.

Reading files: Use `process.cwd()`

Files can be read directly from the filesystem in `getStaticProps`.

In order to do so you have to get the full path to a file.

Since Next.js compiles your code into a separate directory you can't use `__dirname` as the path it will return will be different from the pages directory.

Instead you can use `process.cwd()` which gives you the directory where Next.js is being executed.

```
import { promises as fs } from 'fs'
import path from 'path'

// posts will be populated at build time by getStaticProps()
function Blog({ posts }) {
  return (
    <ul>
      {posts.map((post) => (
        <li>
          <h3>{post.filename}</h3>
          <p>{post.content}</p>
        </li>
      ))}
    </ul>
  )
}

// This function gets called at build time on server-side.
// It won't be called on client-side, so you can even do
// direct database queries.
export async function getStaticProps() {
  const postsDirectory = path.join(process.cwd(), 'posts')
  const filenames = await fs.readdir(postsDirectory)

  const posts = filenames.map(async (filename) => {
    const filePath = path.join(postsDirectory, filename)
    const fileContents = await fs.readFile(filePath, 'utf8')

    // Generally you would parse/transform the contents
    // For example you can transform markdown to HTML here

    return {
      filename,
      content: fileContents,
    }
  })
  // By returning { props: { posts } }, the Blog component
  // will receive `posts` as a prop at build time
  return {
    props: {
      posts: await Promise.all(posts),
    },
  }
}

export default Blog
```

getStaticProps with TypeScript

You can use the `GetStaticProps` type from `next` to type the function:

```
import { GetStaticProps } from 'next'

export const getStaticProps: GetStaticProps = async (context) => {
  // ...
}
```

If you want to get inferred typings for your props, you can use `InferGetStaticPropsType<typeof getStaticProps>`:

```
import { InferGetStaticPropsType } from 'next'

type Post = {
  author: string
  content: string
}

export const getStaticProps = async () => {
  const res = await fetch('https://.../posts')
  const posts: Post[] = await res.json()

  return {
    props: {
      posts,
    },
  }
}

function Blog({ posts }: InferGetStaticPropsType<typeof getStaticProps>) {
  // will resolve posts to type Post[]
}

export default Blog
```

Related

For more information on what to do next, we recommend the following sections:

Data Fetching: Learn more about data fetching in Next.js.