

# NEXTJS

React 服务端渲染框架

# Next.js 介绍

Next.js 是 React 服务端渲染应用框架. 用于构建 SEO 友好的 SPA 应用.

1. 支持两种预渲染方式, 静态生成和服务端渲染.
2. 基于页面的路由系统, 路由零配置
3. 自动代码拆分. 优化页面加载速度.
4. 支持静态导出, 可将应用导出为静态网站.
5. 内置 CSS-in-JS 库 styled-jsx
6. 方案成熟, 可用于生产环境, 世界许多公司都在使用
7. 应用部署简单, 拥有专属部署环境 Vercel, 也可以部署在其他环境.

# 创建 Next.js 项目

创建: `npm init next-app next-guide`

运行: `npm run dev`

访问: `localhost:3000`

临时安装 `create-next-app` 用于创建 Next.js 项目.

# 基于页面的路由系统

## 创建页面

在 Next.js 中, 页面是被放置在 pages 文件夹中的 React 组件.

组件需要被默认导出.

组件文件中不需要引入 React.

页面地址与文件地址是对应的关系.

# 基于页面的路由系统

## 创建页面

```
// pages/list.js
export default function List () {
  return <div>List page works</div>;
}
```

pages/index.js     /

pages/list.js     /list

pages/post/first.js   /post/first

# 基于页面的路由系统

## 页面跳转

页面与页面之间通过 Link 组件进行跳转.

```
import Link from 'next/link';  
  
<Link href="/list"><a>jump to list page</a></Link>
```

# 基于页面的路由系统

## 页面跳转

Link 组件默认使用 JavaScript 进行页面跳转. 即 SPA 形式的跳转.

如果浏览器中 JavaScript 被禁用. 则使用链接跳转.

Link 组件中不应添加除 href 属性以外的属性, 其余属性添加到a标签上.

Link 组件通过预取(在生产中)功能自动优化应用程序以获得最佳性能.

```
import Link from 'next/link';  
  
<Link href="/list"><a title="list page">list page</a></Link>
```

# 静态资源、元数据和 CSS

## 静态资源

应用程序根目录中的 public 文件夹用于提供静态资源.

通过以下形式进行访问.

public/images/1.jpg -> /images/1.jpg

public/css/base.css -> /css/base.css



# 静态资源、元数据和 CSS

## 修改页面元数据

通过 Head 组件修改元数据.

```
import Head from 'next/head';  
  
<  
  <Head>  
    <title>Index Page</title>  
  </Head>  
</>
```

# 静态资源、元数据和 CSS

## CSS 样式

### 内置 styled-jsx

在 Next.js 中内置了 styled-jsx, 它是一个 CSS-in-JS 库, 允许在 React 组件中编写 CSS, CSS 仅作用于组件内部.

```
<Link href="/list">
  <a className="demo">jump to list page</a>
</Link>
<style jsx>{`
  .demo {
    color: red;
  }
`}</style>
```

# 静态资源、元数据和 CSS

## CSS 样式

### CSS 模块

通过使用 CSS 模块功能, 允许将组件的 CSS 样式编写在单独的 CSS 文件中.

CSS 模块约定样式文件的名称必须为"组件文件名称.module.css"

```
// index.module.css
.p { color: green }

// index.js
import styles from './index.module.css';

<div className={styles.p}></div>
```

# 静态资源、元数据和 CSS

## CSS 样式

### 全局样式文件

1. 在 pages 文件夹中新建 \_app.js 文件并加入如下代码
2. 在项目根目录下创建 styles 文件夹, 并在其中创建 global.css
3. 在 \_app.js 中通过 import 引入 global.css.
4. 重新启动开发服务器

```
export default function App({ Component, pageProps }) {  
  return <Component { ... pageProps} />  
}
```

# 预渲染

## 预渲染概述

预渲染是指数据和HTML的拼接在服务器端提前完成.

预渲染可以使 SEO 更加友好.

预渲染会带来更好的用户体验, 可以无需运行 JavaScript 即可查看应用程序UI.

# 预渲染

## 预渲染的两种形式

在 Next.js 中支持两种形式的预渲染: 静态生成和服务器端渲染.

静态生成和服务器端渲染是生成 HTML 的时机不同.

静态生成: 静态生成是在构建时生成 HTML. 以后的每个请求都共用构建时生成好的 HTML.

服务器端渲染: 服务器端渲染是在请求时生成 HTML. 每个请求都会重新生成 HTML.

# 预渲染

## 两种预渲染方式的选择

Next.js 允许开发者为每个页面选择不同的预渲染方式. 不同的预渲染方式拥有不同的特点. 应根据场景进行渲染. 但建议大多数页面建议使用静态生成.

静态生成一次构建, 反复使用, 访问速度快. 因为页面都是事先生成好的.

适用场景: 营销页面、博客文章、电子商务产品列表、帮助和文档

服务器端渲染访问速度不如静态生成快, 但是由于每次请求都会重新渲染, 所以适用数据频繁更新的页面或页面内容随请求变化而变化的页面.

# 预渲染

## 无数据和有数据的静态生成

如果组件不需要在其他地方获取数据, 直接进行静态生成.

如果组件需要在其他地方获取数据, 在构建时 Next.js 会预先获取组件需要的数据, 然后再对组件进行静态生成.



# 预渲染

## 静态生成 `getStaticProps`

`getStaticProps` 方法的作用是获取组件静态生成需要的数据. 并通过 `props` 的方式将数据传递给组件.

该方法是一个异步函数, 需要在组件内部进行导出.

在开发模式下, `getStaticProps` 改为在每个请求上运行.

```
export async function getStaticProps() {  
  // 从文件系统, API, 数据库中获取的数据  
  const data = ...  
  // props 属性的值将会传递给组件  
  return {  
    props: ...  
  }  
}
```

# 预渲染

## 服务器端渲染 getServerSideProps

如果采用服务器端渲染, 需要在组件中导出 `getServerSideProps` 方法.

```
export async function getServerSideProps(context) {  
  // context 中会包含特定的请求参数  
  return {  
    props: {  
      // props for your component  
    }  
  }  
}
```

# 预渲染

## 基于动态路由的静态生成

基于参数为页面组件生成HTML页面，有多少参数就生成多少HTML页面

在构建应用时, 先获取用户可以访问的所有路由参数, 再根据路由参数获取具体数据, 然后根据数据生成静态 HTML.

# 预渲染

## 实现基于动态路由的静态生成

1. 创建基于动态路由的页面组件文件, 命名时在文件名称外面加上[], 比如[id].js
2. 导出异步函数 `getStaticPaths`, 用于获取所有用户可以访问的路由参数

```
export async function getStaticPaths () {  
  // 此处获取所有用户可以访问的路由参数  
  return {  
    // 返回固定格式的路由参数  
    paths: [{params: {id: 1}}, {params: {id: 2}}],  
    // 当用户访问的路由参数没有在当前函数中返回时, 是否显示404页面 false: 显示 true 不显示  
    fallback: false  
  }  
}
```

# 预渲染

## 实现基于动态路由的静态生成

3. 导出异步函数 `getStaticProps`, 用于根据路由参数获取具体的数据

```
export async function getStaticProps ({params}) {  
  // params → {id: 1}  
  // 此处根据路由参数获取具体数据  
  return {  
    // 将数据传递到组件中进行静态页面的生成  
    props: { }  
  }  
}
```

注: `getStaticPaths` 和 `getStaticProps` 只运行在服务器端, 永远不会运行在客户端, 甚至不会被打包到客户端 JavaScript 中, 意味着这里可以随意写服务器端代码, 比如查询数据库.

# 预渲染

## 自定义 404 页面

要创建自定义 404 页面, 需要在 pages 文件夹中创建 404.js 文件.

```
export default function Custom404() {  
  return <h1>404 - Page Not Found</h1>  
}
```

# API Routes

## 什么是 API Routes

API Routes 可以理解为接口, 客户端向服务器端发送请求获取数据的接口.

Next.js 应用允许 React 开发者编写服务器端代码创建数据接口.

# API Routes

## 如何实现 API Routes

1. 在 pages/api 文件夹中创建 API Routes 文件. 比如 user.js
2. 在文件中默认导出请求处理函数, 函数有两个参数, req 为请求对象, res 为响应对象.

```
export default function (req, res) {  
  res.status(200).send({id: 1, name: 'Tom'})  
}
```

注: 当前 API Routes 可以接收任何 Http 请求方法.



# API Routes

## 如何实现 API Routes

3. 访问 API Routes: localhost:3000/api/user

不要在 `getStaticPaths` 或 `getStaticProps` 函数中访问 API Routes, 因为这两个函数就是在服务器端运行的, 可以直接写服务器端代码.