14 Next.js 开发 C 端（问卷 H5）

SSR

1. SSR 与 CSR
   a. SSR（Server side render， 服务端渲染），是指前端直接向后端请求一个渲染好的 HTML。
   b. CSR（Client side render，客户端渲染），是指为了得到一个真实的 HTML，前端需要①向后端请求 HTML、JS，②再使用 JS 请求若干次 JSON 数据。
   c. 由于 CSR 需要使用公共网络（不稳定）进行请求，因此性能较差；SSR 在服务器使用内网（速度快）进行请求，因此性能较好
2. SSR 的优缺点
   a. 优点：①性能好 ②易于 SEO 搜索引擎优化
   b. 缺点：开发成本高，需要前端框架 + Node.js 的支持
3. SSR 的适用场景：①对性能要求较高的系统（移动端、弱网环境等） ②操作交互较简单的系统
4. 常见的 SSR 框架
   a. React 技术栈：Next.js、Remix.js
   b. Vue 技术栈：Nuxt.js

Next.js

1. 创建项目

```
npx create-next-app@latest --typescript
```

> 创建配置项的选择

```
√ What is your project named? ... wenjuan-client
√ Would you like to use ESLint? ... No / Yes
√ Would you like to use Tailwind CSS? ... No / Yes
√ Would you like to use src/ directory? ... No / Yes
√ Would you like to use App Router? (recommended) ... No / Yes
√ Would you like to customize the default import alias (@/*)? ... No / Yes
```

2. 基本功能
    a. `src/pages` 下可以创建页面（可以通过路由访问，文件名就是路由地址）
        - `src/pages/index.tsx` 是主页，对应的路由为 /，可以通过 http://localhost:3000/ 的方式访问
        - eg. `src/pages/about.tsx` 对应的路由为 /about，可以通过 http://localhost:3000/about 的方式访问
    b. `src/pages/api` 下可以创建 api（可以请求获取相应的数据）
        - eg. `src/pages/api/hello.ts` 对应的请求 url 为 http://localhost:3000/api/hello
    c. `public` 下可以存放静态资源

Next.js pre-render

Static Generation

1. 定义：项目构建时，直接产出 HTML 文件（**静态**）
2. 实现：在 `src/pages` 下创建的页面中，通过 `export async function getStaticProps`，**异步请求数据**，返回的数据作为组件 `props` 传递，用于页面的渲染。

    > `getStaticProps` **仅在构建时执行**（即 npm run build），线上环境时，每次请求不会执行该函数

3. 示例

```tsx
// src/pages/welcome.tsx
import Head from "next/head";

interface IProps {
  info: string;
}

export default function Welcome(props: IProps) {
  const { info } = props;
  return (
    <>
      <Head>
        <title>Create Next App</title>
        <meta name="description" content="Generated by create next app" />
        <meta name="viewport" content="width=device-width, initial-scale=1" />
        <link rel="icon" href="/favicon.ico" />
      </Head>
```

```
      <div style={{ width: "100vw", textAlign: "center", marginTop: "90px" }}>
        <h2>Hello World</h2>
        <h5>Created By {info}</h5>
      </div>
    </>
  );
}

// Static Generation
export async function getStaticProps() {
  /* 这里可以 await 异步请求 */
  /* 该函数只有在 npm run build 构建时执行，上线后任何请求都不会触发该函数的执行 */
  return {
    props: {
      info: "yiTuChuan",
    },
  };
}
```

Server-side rendering

1. 定义：每次请求时，动态生成 HTML 文件（**动态**）
2. 实现：与 Static Generation 类似，通过 `export async function getServiceSideProps` **异步请求数据**，返回的数据作为组件 `props` 传递，用于页面的渲染

   `getServiceSideProps` 每次请求都会执行
3. 示例

```
// src/pages/welcome.tsx
import Head from "next/head";

interface IProps {
  info: string;
}

export default function Welcome(props: IProps) {
  const { info } = props;
  return (
    <>
      <Head>
        <title>Create Next App</title>
        <meta name="description" content="Generated by create next app" />
```

```
15          <meta name="viewport" content="width=device-width, initial-scale=1"
    />
16          <link rel="icon" href="/favicon.ico" />
17        </Head>
18        <div style={{ width: "100vw", textAlign: "center", marginTop: "90px"
    }}>
19          <h2>Hello World</h2>
20          <h5>Created By {info}</h5>
21        </div>
22      </>
23    );
24  }
25
26  // Server-side rendering
27  export async function getServerSideProps() {
28    /* 这里可以 await 异步请求 */
29    /* 该函数在每次发送请求时都会执行 */
30    return {
31      props: {
32        info: "Nasir",
33      },
34    };
35  }
36
```

动态获取 url 参数

Step1. 定义动态路由

- 我们知道，`src/pages` 下的文件名就是路由地址，如 `src/pages/state/success.tsx` 就可以通过 `/state/success` 这个路由地址访问到。
- 为了获取 url 中的参数，可以 `[参数名]` 的方式定义页面文件，其对应的也就是**动态路由**。如 `src/pages/question/[id].tsx` 就可以通过 `/question/xxxxxxxxxx` 这个路由地址访问到，这里的 `xxxxxxxxxx` 是任意内容，作为 id 的实际取值。

Step2. 通过 `getServerSideProps` 的 `context` 参数，以 `context.params` 的方式动态获取动态路由中定义的 url 参数。

- 动态路由中定义了 url 参数的 key，如 `src/pages/question/[id].tsx` 就定义了 `id` 这个 key。
- 当用户输入 `/question/xxxxxxxxxx` 路由，可以在 `getServerSideProps` 中通过 `const { id } = context.prarms` 的方式获取到 id，其值为 `xxxxxxxxxx`。

```tsx
import {
  GetServerSidePropsContext,
  GetServerSidePropsResult,
} from "next";
import Head from "next/head";

type PropsType = {
  id: string;
};

export default function Question(props: PropsType) {
  const { id } = props;
  return (
    <>
      <Head>
        <title>Create Next App</title>
        <meta name="description" content="Generated by create next app" />
        <meta name="viewport" content="width=device-width, initial-scale=1" />
        <link rel="icon" href="/favicon.ico" />
      </Head>
      <div style={{ width: "100vw", textAlign: "center", marginTop: "90px" }}>
        <h2>Question Page</h2>
        <h5>question id = {id}</h5>
      </div>
    </>
  );
}

// Server-side rendering
export async function getServerSideProps(
  context: GetServerSidePropsContext
): Promise<GetServerSidePropsResult<PropsType>> {
  /* 这里可以 await 异步请求 */
  /* 该函数在每次发送请求时都会执行 */
  const { id = "" } = context.params as PropsType;
  return {
    props: {
      id,
    },
  };
}
/* 关于动态路由参数
  - getServerSideProps 中通过 context.params 获取动态路由参数，其属性（对应参数）类型是 ParsedUrlQuery = string | string[] | undefined
  - 动态路由与参数类型
    - 一般动态路由：/post/[id] => id 的类型是 string; /post/123 => id = 123
```

```
46      - catch-all 动态路由：/post/[...id] => id 的类型是 string[]; /post/123/456 =>
    id = ["123", "456"]
47      - 动态路由的错误使用：当没有传递相应参数，或路由没有匹配到响应参数时 => 此时
    context.params 的某个属性的取值可能是 undefined
48  */
49  /* 关于 getServerSideProps 函数的类型
50    - 参数（context）类型：getServerSidePropContext
51    - 返回值类型：Promise<GetServerSidePropsResult<PropsType>>，这里的 PropsType 指
    的是函数返回的 props 的类型 */
```

答卷 H5 的设计

基本逻辑

Step1. 获取问卷数据

Step2. 根据组件列表，显示表单

Step3. 用户填写表单，提交数据

表单提交

1. 方式一：Ajax 提交
2. 方式二（推荐）：`<form>` 的 `action` 属性提交（简单、兼容性好、性能好）

> 提交的数据格式为 `{"组件的 fe_id": "用户填写的值"}`