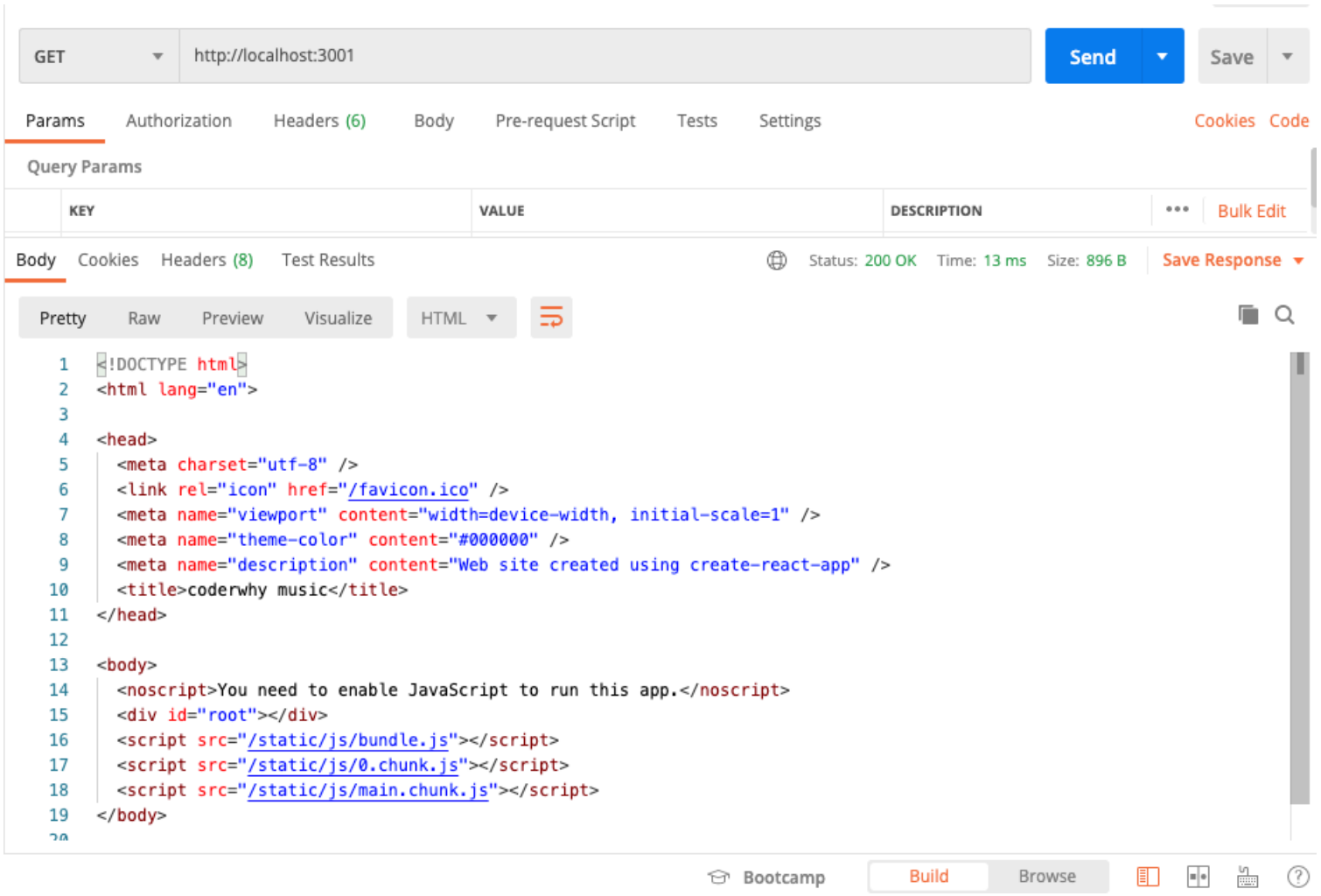
React SSR

王红元 coderwhy

实力IT教育

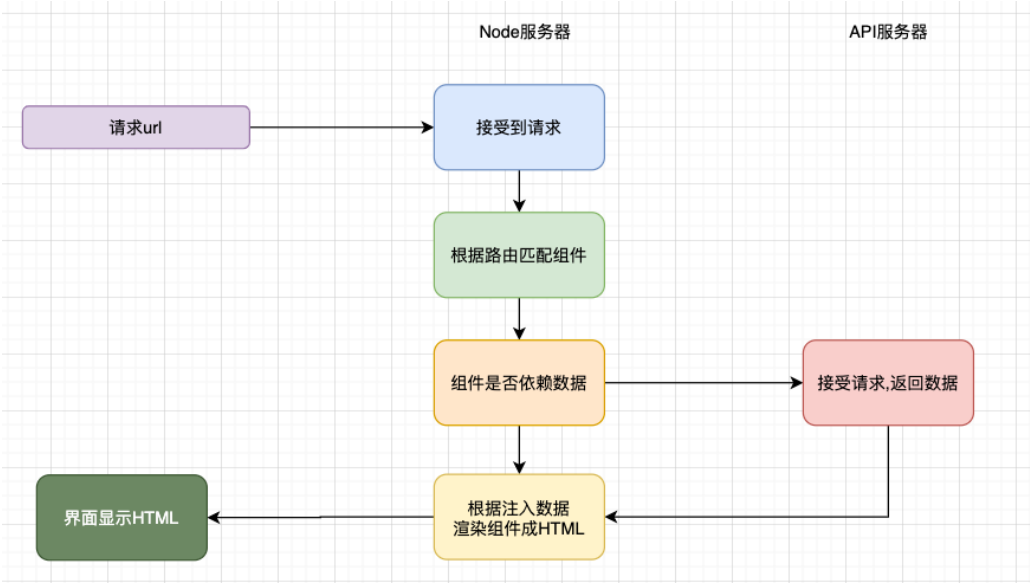
为什么需要SSR呢？

- 单页面富应用的局限：
  - 之前我们开发的应用程序，如果直接请求可以看到上面几乎没有什么内容。
  - 但是为什么我们页面可以看到大量的内容呢？
  - 因为当我们请求下来静态资源之后会执行JS，JS会去请求数据，并且渲染我们想要看到的。
- 但是这个过程存在另外两个问题：
  - 问题一：首屏显示的速度较慢；
  - 问题二：不利于SEO的优化；
- 如何解决这个问题呢？
  - 采用服务端渲染；



认识SSR和同构？

- 认识SSR
- SSR（**Server Side Rendering，服务端渲染**），指的是页面在服务器端已经生成了完成的HTML页面结构，不需要浏览器解析；
- 对应的是CSR（**Client Side Rendering，客户端渲染**），我们开发的SPA页面通常依赖的就是客户端渲染；
- 早期的服务端渲染包括PHP、JSP、ASP等方式，但是在目前前后端分离的开发模式下，前端开发人员不太可能再去学习PHP、JSP等技术来开发网页；
- 不过我们可以借助于Node来帮助我们执行JavaScript代码，提前完成页面的渲染；
- 什么是同构？
- **一套代码既可以在服务端运行又可以在客户端运行，这就是同构应用。**
- **同构是一种SSR的形态，是现代SSR的一种表现形式。**
- **当用户发出请求时，先在服务器通过SSR渲染出首页的内容。**
- **但是对应的代码同样可以在客户端被执行。**
- **执行的目的包括事件绑定等以及其他页面切换时也可以在客户端被渲染；**



使用React SSR

- 使用React SSR主要有两种方式：
  - 方式一：手动搭建一个SSR框架；
  - 方式二：使用已经成熟的SSR框架：Next.js
- 安装Next.js框架的脚手架：

```
npm install -g create-next-app
```

- 创建Next.js项目

```
create-next-app next-demo
```

- package.json文件

```
"scripts": {
  "dev": "next dev",
  "build": "next build",
  "start": "next start"
},
```

```
"dependencies": {
  "next": "9.5.2",
  "react": "16.13.1",
  "react-dom": "16.13.1"
}
```

首页的展示

- Next.js默认已经给我们配置好了路由映射关系：
  - 路径和组件的映射关系；
  - 这个映射关系就是在pages中配置相关的组件都会自动生成对应的路径；
- 默认page/index.js是页面的默认路径：

```
export default function Home() {
  return (
    <div>
      <h2>轮播图展示</h2>
      <h2>推荐商品</h2>
      <ul>
        {
          [0, 1, 2].map(item => {
            return <li key={item}>推荐商品{item}</li>
          })
        }
      </ul>
    </div>
  )
}
```

定义关于页面和页面跳转

- 定义About页面

```
export default memo(function About() {
  return (
    <div>
      <Head>
        <title>网易云音乐</title>
      </Head>
      <h2>About页面</h2>
    </div>
  )
})
```

- 从Home页面跳转到About页面

```
{/* 链接跳转 */}
<a href="/about">关于</a>

{/* 路由跳转 */}
<Link href="/about">
  <a>关于</a>
</Link>
```

Layout组件

- 我们发现home和about是两个相互独立的组件：
  - 如果它们有一些公共的内容：比如头部、尾部是一样的，是否每个地方都需要写一遍呢？
- 有两种解决方案：
  - 方案一：自定义一个Layout的组件，将公共的内容放到Layout中；
  - 方案二：在_app中编写公共部分的内容；

```
export default memo(function Layout(props) {
  return (
    <div>
      <Head>
        <title>网易云音乐</title>
      </Head>
      <Header />
      {props.children}
      <Footer />
    </div>
  )
})
```

```
function MyApp({ Component, pageProps }) {
  return (
    <div>
      <Head>
        <title>网易云音乐</title>
      </Head>
      <Header />
      <Component {...pageProps} />
      <Footer />
    </div>
  )
}
```

Next.js支持各种样式

- 方式一：全局样式引入
- 方式二：module.css
- 方式三：默认集成styled-jsx

```
<style>{`
  h2 {
    color: red;
  }
`}</style>
```

- 方式四：其他css in js方案，比如styled-components
  - 引入相关的依赖；
  - 创建和编辑 .babelrc文件

```
yarn add styled-components
yarn add -D babel-plugin-styled-components
```

```
{
  "presets": [
    "next/babel"
  ],
  "plugins": [
    ["styled-components"]
  ]
}
```

路由的补充

- 路由的嵌套（子路由）：
  - 文件夹的嵌套，最后就可以形成子路由；
- 路由的传参：
  - Next.js中无法通过 /user/:id的方式传递参数；
  - 只能通过 /user?id=123的方式来传递参数；

- 传递参数有两种办法：
  - Link中的路径；
  - Router.push(pathname, query)

```
<Link key={item} href={"/detail?id="+item}>
  <li>商品信息{item}</li>
</Link>
```

```
const productClick = (id) => {
  Router.push({
    pathname: "/detail",
    query: {
      id
    }
  })
}
```

```
const router = useRouter();
console.log(router.query.id);
```

请求数据 - getInitialProps

```
Home.getInitialProps = (props) => {
  return axios({
    url: "http://123.207.32.32:8000/home/multidata"
  }).then(res => {
    console.log(res);
    return {
      banner: res.data.data.banner,
      recommend: res.data.data.recommend
    };
  })
}
```