服务器端渲染（2）：理解 Next.js

我们已经知道了服务器端渲染的原理，你只需要搭建一个 Express 服务器，在服务器端手工打造『脱水』，在浏览器端做『注水』，完成某个页面的服务器端渲染并不难。

不过，服务器端渲染的问题并不这么简单，一个最直接的问题，就是怎么处理多个页面的『单页应用』（Single-Page-Application）？

所以单页应用，就是虽然用户感觉有多个页面，但是实现上只有一个页面，用户感觉到页面可以来回切换，但其实只是一个页面并没有完全刷新，只是局部界面更新而已。

假设一个单页应用有三个页面 Home、Prodcut 和 About，分别对应的的路径是 `/home` 、 `/product` 和 `/about` ，而且三个页面都依赖于 API 调用来获取外部数据。

现在我们要做服务器端渲染，如果只考虑用户直接在地址栏输入 `/home` 、 `/product` 和 `/about` 的场景，很容易满足，按照上面说的套路做就是了。但是，这是一个单页应用，用户可以在 Home 页面点击链接无缝切换到 Product，这时候 Product 要做完全的浏览器端渲染。换句话说，每个页面都需要既支持服务器端渲染，又支持完全的浏览器端渲染，更重要的是，对于开发者来说，肯定不希望为了这个页面实现两套程序，所以必须有同时满足服务器端渲染和浏览器端渲染的代码表示方式。

读者可以思考一下什么样的代码表示合适，也可以直接往下，看看业界公认最科学的实现方式 Next.js 是如何做的。

快速创建 Next.js 项目

在说明 Next.js 的工作原理之前，我们先看怎么快速创建 Next.js 项目，这个问题用代码来说明会更顺畅。

我们也可以手工创建 Next.js 项目，不过更简单的方式是用自动化工具 create-next-app，这个 create-next-app 类似于 create-react-app，一个命令就创建一个可以运行的应用。

首先安装 create-next-app。

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

然后，就可以在你专门存放项目的目录下执行 create-next-app，产生一个使用 Next.js 的 React 应用，下面的命令创建一个叫 next_demo 的应用：

```
create-next-app next_demo
```

进入新生成的项目目录 next_demo 里检查一下，可以看到文件结构非常简洁， `pages` 目录下是页面文件， `package.json` 中差不是下面这样，没有繁冗的 webpack 和 babel 依赖包，因为一切都被 Next.js 封装起来了。

```
{
  "name": "create-next-example-app",
  "scripts": {
    "dev": "next",
    "build": "next build",
    "start": "next start"
  },
  "dependencies": {
    "next": "^6.0.3",
    "react": "^16.5.2",
    "react-dom": "^16.5.2"
  }
}
```

虽然有不少框架都表示自己的功能很强大，但其中有很多框架的设计并不中立，用这些框架去开发某些特定应用或许还行，如果放到一个更大范围的应用类型中，就会发现无法满足要求，这样的框架通用性不足，开发者一定要谨慎使用。

讲良心话，Next.js 真的是一个通用性非常高的框架，因为 Next.js 完全遵从了 React 的技术哲学：一切皆为组件。

在 Next.js 中，创造一个页面，其实就是创造一个 React 组件，接下来我们看看如何创建一个页面。

编写页面

使用下面的命令启动 Next.js 应用，进入的是开发者模式，这时候对代码的改变，会立刻体现在网页上。

```
npm run dev
```

请注意，这一点上 Next.js 的习惯用法和 create-react-app 产生的应用不一样。在 create-react-app 产生的应用中， npm run start 启动是开发者模式，但在 Next.js 应用中，习惯上 npm run start 以产品模式启动，所以要先运行 npm run build 然后才能运行 npm run start。

Next.js 遵从『协定优于配置』（convention over configuration）的设计原则，根据『协定』，在 `pages` 中每个文件对应一个网页文件，文件名对应的就是网页的路径名，比如 `pages/home.js` 文件对应的就是 `/home` 路径的页面，当然 `pages/index.js` 比较特殊，对应的是默认根路径 `/` 的页面。

我们修改 `pages/index.js` ，让它更简单一些，如下：

```
import React from 'react'

const Home = (props) => (
  <h1>
    Hello World
  </h1>
)

export default Home
```

这样会在页面上显示出一个 `Hello World` ，而这个页面代码就是一个普通的 React 组件而已。

页面都是 React 组件，这就是 Next.js 的哲学。

getInitialProps

我们还是要回到本来的话题，如何优雅地实现服务器端渲染，上面的 `Home` 页面虽然能够渲染出完整包含 `Hello World` 的 HTML，但是并没有调用任何外部 API 资源，所以也没有异步操作，并不能体现服务器端渲染的难度。

我们用一个函数来实现异步操作，以此模拟调用 API 的延迟效果，如下：

```
const timeout = (ms, result) => {
  return new Promise(resolve => setTimeout(() => resolve(result), ms));
};
```

然后，我们利用这个 `timeout` 来获得展示网页所需的数据。比如说，获取用户名，那么我们的 Home 组件就要换一个写法，像下面那样，增加 `getInitialProps` 的定义：

```
const Home = (props) => (
  <h1>
    Hello {props.userName}
  </h1>
)

Home.getInitialProps = async () => {
  return await timeout(200, {userName: 'Morgan'});
};
```

这个 getiInitialProps 是 Next.js 最伟大的发明，它确定了一个规范，一个页面组件只要把访问 API 外部资源的代码放在 getInitialProps 中就足够，其余的不用管，Next.js 自然会在服务器端或者浏览器端调用 getInitialProps 来获取外部资源，并把外部资源以 props 的方式传递给页面组件。

注意 getInitialProps 是页面组件的静态成员函数，可以用下面的方法定义：

```
Home.getInitialProps = async () = {...};
```

也可以在组件类中加上 `static` 关键字定义：

```
class Home extends React.Component {
  static async getInitialProps() {
     ...
  }
}
```

通过上面的代码，我么也可以注意到，getInitialProps 是一个 async 函数，所以，在 getInitialProps 函数中可以使用 `await` 关键字，用同步的方式编写异步逻辑。

我们可以这样来看待 getInitialProps，它就是 Next.js 对代表页面的 React 组件生命周期的扩充。React 组件的生命周期函数缺乏对异步操作的支持，所以 Next.js 干脆定义出一个新的生命周期函数 getInitialProps，在调用 React 原生的所有生命周期函数之前，Next.js 会调用 getInitialProps 来获取数据，然后把获得数据作为 props 来启动 React 组件的原生生命周期过程。

这个生命周期函数的扩充十分巧妙，因为：

1. 没有侵入 React 原生生命周期函数，以前的 React 组件该怎么写还是怎么写；
2. getInitialProps 只负责获取数据的过程，开发者不用操心什么时候调用 getInitialProps，依然是 React 哲学的声明式编程方式；
3. getInitialProps 是 async 函数，可以利用 JavaScript 语言的新特性，用同步的方式实现异步功能。

Next.js 的“脱水”和“注水”

我们说过服务器端渲染的关键是如何“脱水”和“注水”，如果你对 Next.js 如何实现这两个关键点好奇（实际上你确实应该感到好奇），那么在浏览器中使用“显示网页源代码”就可以让你一目了然。

在网页的 HTML 中，可以看到类似下面的内容：

```
<script>
  __NEXT_DATA__ = {
    "props":{
      "pageProps": {"userName":"Morgan"}},
      "page":"/","pathname":"/","query":{},"buildId":"-","assetPrefix":"","nextExport":false,"err":nu
</script>
```

Next.js 在做服务器端渲染的时候，页面对应的 React 组件的 getInitialProps 函数被调用，异步结果就是“脱水”数据的重要部分，除了传给页面 React 组件完成渲染，还放在内嵌 script 的 `__NEXT_DATA__` 中，这样，在浏览器端渲染的时候，是不会去调用 getInitialProps 的，直接通过 `__NEXT_DATA__` 中的“脱水”数据来启动页面 React 组件的渲染。

这样一来，如果 getInitialProps 中有调用 API 的异步操作，只在服务器端做一次，浏览器端就不用做了。

那么，getInitialProps 什么时候会在浏览器端调用呢?

当在单页应用中做页面切换的时候，比如从 Home 页切换到 Product 页，这时候完全和服务器端没关系，只能靠浏览器端自己了，Product页面的 getInitialProps 函数就会在浏览器端被调用，得到的数据用来开启页面的 React 原生生命周期过程。

关键点是，浏览器可能会直接访问 `/home` 或者 `/product` ，也可能通过网页切换访问这两个页面，也就是说 Home 或者 Product 都可能被服务器端渲染，也可能完全只有浏览器端渲染，不过，这对应用开发者来说无所谓，应用开发者只要写好 getInitialProps，至于调用 getInitialProps 的时机，交给 Next.js 处理就好了。

你可以发明自己的服务器端框架，但很可能最后你发现，如果要做得通用性好，最后都会做到和 Next.js 一样的模式上来。

值得一提的是，getInitialProps 返回的应该是“纯数据”，也就是不要返回一个定制类的实例。比如，有一个类 Foo 有一个成员函数 bar，不要在 getInitialProps 返回一个 Foo 实例。不然，经过“脱水”和“注水”过程，网页组件获得的那个“Foo 实例”不再是你想的那个 Foo 实例了，它变成了一个纯粹的数据，不会包含成员函数 bar的。

小结

这一小节我们介绍了 React 服务器端渲染最优秀的框架 Next.js。Next.js 的内容很多，读者可以在官网深入了解，这一小节主要讲了以下内容：

1. getInitialProps 是 Next.js 的核心，也是 Next.js 最伟大的发明；
2. 在 getInitialProps 中实现 AJAX 等异步操作，其他 React 组件就无需关心自己是在服务器端还是浏览器端被渲染；
3. Next.js 如何实现“脱水”和“注水”。

留言

评论将在后台进行审核，审核通过后对所有人可见

Oliveryoung 前端开发
反正无论如何就是 只有第一个页面是服务端渲染的 其他页面都是 脱水之后 客户端 js 运作的
▲ 0 收起评论 1月前

Oliveryoung 前端开发
注水
1月前

程墨 Hulu
是的
1月前

评论审核通过后显示 评论

zhangyanling77 前端开发 @ 成都
上个月研究了一下next.js，当时没太搞清楚他怎么区分什么时候服务端渲染，什么时候浏览器端渲染。看了您的文章清楚了些，感谢！
▲ 0 评论 1月前

snowLu 前端小洋葱 @ lg
像知乎和简书好像都是服务端渲染的页面，但是我好像没在他们html中找到类似存放后台数据的data变量
▲ 0 评论 1月前

snowLu 前端小洋葱 @ lg
想问一下墨哥，如果把当前页面所需要的所有后台的数据都放到网页里的__NEXT_DATA__中，这样好吗?
▲ 0 收起评论 1月前

程墨 Hulu
只需要放前端渲染必需的就够，要知道__NEXT_DATA__太大也影响性能啊。
1月前

评论审核通过后显示 评论

blacker
扩展的React组件的生命周期函数是怎么做到的？decortor吗？
▲ 0 收起评论 1月前

程墨 Hulu
所谓扩展React组件的生命周期函数，指的是定义新的函数规则，然后自己写框架去调用这些函数，比如next.js的getIntialProps。
1月前

评论审核通过后显示 评论