Web Studio 2019
12.router

김상근 / sisobus1@gmail.com

Contents

API와 연동하기

연동하기 전에 client code 작성하기

1. 임의의 데이터를 javascript단에서 만들어 놓음
2. 해당 데이터가 있을 때 component를 mount하도록 작성

```
render() {
  return (
    <div className="article-container">
    {this.state.articles &&
      this.state.articles.map((article, idx) => (
        <Article
          title={article.title}
          author={article.author}
          content={article.content}
          like={article.like}
          key={idx}
          idx={idx}
          onClick={this.handleLike}
        />
      ))
    }
    </div>
  )
}
```

```
class Articles extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      articles: [{
        title: 'asdfasdf',
        content: 'content!1!',
        author: 'Sangkeun Kim',
        key: 1,
        idx: 1,
      }, {
        title: 'asdfasdf',
        content: 'content!2!',
        author: 'Sangkeun Kim',
        key: 2,
        idx: 2,
      }, {
        title: 'asdfasdf',
        content: 'content!3!',
        author: 'Sangkeun Kim',
        key: 3,
        idx: 3,
      }, {
        title: 'asdfasdf',
        content: 'content!4!',
        author: 'Sangkeun Kim',
        key: 4,
        idx: 4,
      }]
    }
    this.handleLike = this.handleLike.bind(this)
  }
```

API와 연동하기

fetch로 API에 요청하기

1. 임의의 데이터로 component가 잘 붙는 것을 확인한 후
2. 임의의 데이터를 지우고 fetch로 가져옴
3. (당연한 얘기지만 API server는 켜져있어야함)

```
class Articles extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      articles: []
    }
    this.handleLike = this.handleLike.bind(this)

    fetch('http://localhost:5000/api/articles')
      .then(response => {
        response.json().then(rsp => {
          this.setState({ articles: rsp })
        })
      })
  }
```

Client-side routing

React-router-dom

1. SPA(Single Page Application)가 아닌 MPA에서는 a tag를 이용해 페이지 전환을 함
   (e.g. ⟨a href="https://sisobus.com"⟩Go to sisobus!⟨/a⟩)
2. A tag를 이용하면 페이지 전환시 화면이 refresh 됨
3. 이를 해결하기 위해 데이터를 동적으로 가져오는 방법이 있었음
   (XHTTPRequest, jQuery, …)
4. 하지만 DOM manipulation이 일어나게 되고, 이는 곧 성능의 저하로 이어짐

Client-side routing

React-router-dom

1. React는 Virtual DOM을 이용해 이 부분의 성능을 높임
2. 거기에 Component만 상황에 맞게(Page 단위) unmount, mount시켜주면 MPA처럼 사용 가능
3. 이 말은 즉 Client-side routing은 사실 Component를 router에 맞게 unmount, mount를 시켜준다는 말과 같음
4. react-router-dom은 이를 쉽게 사용할 수 있도록 만들어진 구현체
   〉〉 갖다 쓰면 됨!

Client-side routing

React-router-dom

1. npm install react-router-dom으로 설치
2. App.js에 BrowserRouter를 import
3. path와 해당 component를 연결해줌

```
import React from 'react';
import './App.css';
import MainPage from './_components/Main'
import BlankPage from './_components/BlankPage'
import { BrowserRouter as Router, Route } from "react-router-dom"

function App() {
  return (
    <div className="App">
      <Router>
        <Route path="/" exact component={MainPage} />
        <Route path="/blank" component={BlankPage} />
        <Route path="/about" component={MainPage} />
        <Route path="/hi" component={BlankPage} />
        <Route path="/secret" component={MainPage} />
      </Router>
    </div>
  );
}

export default App;
```

Client-side routing

React-router-dom

1. npm install react-router-dom으로 설치
2. App.js에 BrowserRouter를 import
3. path와 해당 component를 연결해줌

```
import React from 'react';
import './App.css';
import MainPage from './_components/Main'
import BlankPage from './_components/BlankPage'
import { BrowserRouter as Router, Route } from "react-router-dom"

function App() {
  return (
    <div className="App">
      <Router>
        <Route path="/" exact component={MainPage} />
        <Route path="/blank" component={BlankPage} />
        <Route path="/about" component={MainPage} />
        <Route path="/hi" component={BlankPage} />
        <Route path="/secret" component={MainPage} />
      </Router>
    </div>
  );
}

export default App;
```

Client-side routing

React-router-dom

1. Main 페이지에서 다른 페이지로의 전환은
   MPA에선 a tag였다면, react-router-dom에선
   〈Link to="/〈next-path〉"〉Go to page〈/Link〉
2. Go to Blank 버튼을 누르면 /blank로 이동

```jsx
import React from 'react'
import './Main.css'
import Nav from './Nav'
import Articles from './Articles'
import { Button } from 'antd'
import { Link } from "react-router-dom"

class MainPage extends React.Component {
  render() {
    return (
      <React.Fragment>
        <Nav />
        <Link to="/blank">
          <Button type="primary">Go to Blank</Button>
        </Link>
        <div className="main-wrapper">
          <div className="main-container">
            <Articles />
          </div>
        </div>
      </React.Fragment>
    )
  }
}

export default MainPage
```

Client-side routing

React-router-dom

1. Link를 이용하지 않고, 동적으로 페이지를 전환하고 싶다면 history가 필요함
   (e.g. 버튼을 클릭하면 다른페이지로 이동,
   e.g. fetch가 끝나면 다른페이지로 이동)
2. 우선 history를 만들어 주고,
3. App.js의 router에 history를 추가해줌
4. history.push('/경로')를 이용해 이동할 수 있음

.jsx JS history.js ×

```
import { createBrowserHistory } from 'history'

export const history = createBrowserHistory()
```

```
import React from 'react';
import './App.css';
import MainPage from './_components/Main'
import BlankPage from './_components/BlankPage'
import { Router, Route } from "react-router-dom"
import { history } from './_components/history'

function App() {
  return (
    <div className="App">
      <Router history={history}>
        <Route path="/" exact component={MainPage} />
        <Route path="/blank" exact component={BlankPage} />
        <Route path="/about" exact component={MainPage} />
        <Route path="/hi" exact component={BlankPage} />
        <Route path="/secret" exact component={MainPage} />
      </Router>
    </div>
  );
}

export default App;
```

Client-side routing

React-router-dom

1. Link를 이용하지 않고, 동적으로 페이지를 전환하고 싶
   history가 필요함
   (e.g. 버튼을 클릭하면 다른페이지로 이동,
   e.g. fetch가 끝나면 다른페이지로 이동)
2. 우선 history를 만들어 주고,
3. App.js의 router에 history를 추가해줌
4. history.push('/경로')를 이용해 이동할 수 있음

```
import React from 'react'
import { Button } from 'antd'
import { Link } from "react-router-dom"
import {history } from './history'
import Nav from './Nav'

class BlankPage extends React.Component {
  render() {
    return (
      <React.Fragment>
        <Nav />
        <h1>Blank Page!!</h1>
        <Link to="/">
          <Button type="error">Go to MainPage</Button>
        </Link>
        <Button type="error">
          <Link to="/">Go to MainPage</Link>
        </Button>
        <Button type="error" onClick={() => history.push('/')}>
          Go to MainPage
        </Button>
      </React.Fragment>
    )
  }
}

export default BlankPage
```

React Life Cycle

Component의 Life Cycle

```
mount: constructor => componentWillMount => render => componentDidMount
unmount: componentWillUnmount
update: componentWillReceiveProps => shouldComponentUpdate => componentWillUpdate =>
        render => componentDidUpdate
(if state is change then it will be start shouldComponentUpdate)
*-------------------------------------------------------------------------*
| constructor ===> componentWillMount ===> render ===========> componentDidMount |
| componentWillReceiveProps => ShouldComponentUpdate ====*                |
|                                    ^                    |                |
|                                    |                    |                |
| (state changed) ---------------------                   |                |
|                                                         |                |
|  componentDidUpdate <= render <= componentWillUpdate <=*                 |
|                                                                          |
| componentWillUnmount                                                     |
*-------------------------------------------------------------------------*
```

https://github.com/sisobus/dev-document/blob/master/react.md

Q & A