웹 프로그래밍 - Frontend

중간고사 이후부터 정리

8주차

시멘틱 태그를 사용한 html의 기본 구조

HTML 코드

```
<!DOCTYPE html>
<html lang="ko">
  <head>
    <meta charset="UTF-8" />
    <title>제목</title>
  </head>
  <body>
    <header>
      HEADER
    </header>
    <aside>
      aside
    </aside>
    <section>
      Main Content
    </section>
    <footer>
      FOOTER
    </footer>
  </body>
</html>
```

CSS 코드 (일부)

```
header {
  position: fixed;
  background-color: beige;
  left: 0;
  right: 0;
  height: 120px;
}
```

`position:fixed`는 해당 div를 해당 문서에서 절댓값으로 위치를 주는 것이다.

`fixed`를 하면 `top, bottom, left, right`속성으로 해당 div를 문서위치에 위치한다.

```css
aside {
  position: fixed;
  background-color: black;
  color: white;
  left: 0;
  top: 100px;
  width: 100px;
  bottom: 100px;
}
```

해당 결과물은 오른쪽 사진과 같다.

만약 aside를 오른쪽에 두고 싶으면

```css
aside#right {
  right: 0;
  width: 100px;
}
```

로 해주면 된다. (**right는 0이어야 한다!**)

CSS calc()

css에서 `calc()`라는 함수가 있는데, 덧셈, 뺄셈, 곱셈, 나눗셈이 가능한 함수이다.

```css
.example {
  height: calc(100vh - 100px);
}
```

> **vh / vw (viewport) vs % (percent)**
>
> %는 부모의 높이를 뜻하고 vh는 부모가 브라우저에 보이는 높이를 뜻한다.
>
> 즉 부모의 높이가 2000px이고 브라우저는 1000px을 보여줄 때
>
> - 100vh : 1000px
> - 100% : 2000px
>
> 이 된다

스크롤 만들기

1. HTML 코드

```
<div class="parent">
  <div class="scroll-div">
    scroll area
  </div>
</div>
```

2. CSS 코드

```
.parent {
  height: 200px;
}
.scroll-div {
  height: 400px;
  overflow-y: auto;
}
```

자식의 height를 부모의 height보다 더 크기 준 다음, `overflow-y: auto`를 해주면 스크롤이 생긴다.

9주차-1

React 설치

1. Create-React-App 설치

```
npm install -g create-react-app # npm 버전
yarn global add create-react-app # yarn 버전
```

2. 프로젝트 생성

```
create-react-app web02react
```

3. Intellij에서 열기

React란

가상 DOM을 이용해 화면을 메모리상에 저장한 뒤, 실제 DOM과 비교하여 차이가 발생하면 다시 랜더링하는 방식이다.

React 사용

```
import React { Component } from "react"; // React를 불러옴

class Counter extends Component {
  // 1) state 사용
  state = {
    value: 0
  };

  // 2) render() 구현
  render() {
    return (
      <div className="counter-style">
        {/* 3) state 사용 */}
        <div>{this.state.value}</div>
        {/* 4) method 실행 */}
        <button onClick={this.handlePlus}>+</button>
        <button onClick={this.handleMinus}>-</button>
      </div>
    );
  }

  // 5) custom method 구현
  handlePlus = () => {
    // 6) setState 사용
    this.setState({
      value: this.state.value + 1
    });
  };
  handleMinus = () => {
    this.setState({
      value: this.state.value - 1
    });
  };
}
// export를 해줘야 외부에서 사용할 수 있다,
export default Counter;
```

1. state 사용

   state는 리액트에서 관리하는 변수이다.

   만약 `render()`에서 `state`를 사용할 경우 `state`의 값이 변경될 때 다시 `render()` 함수가 실행된다.

2. render() 구현

   리액트에서 class component를 만들때는 `render()` 함수를 구현해 줘야 한다.

   알아둬야 할 점은 `render()`은 컴포넌트가 생성될 때도 실행이 되지만 그 후, `state`가 변경되도 계속 실행되는 메소드기 때문에 초기화같은 것은 여기에 넣으면 안된다.

3. state 사용

state를 사용하기 위해서는 `this.state.value`와 같이 `this.state`를 통해 `state`에 접근해야 한다.

> 컴포넌트 (jsx라고 부른다)내에서 javascript변수나 메소드같은 코드를 실행하기 위해서는 `{}`를 앞과 뒤에 붙어야 한다. (php의 `<?= ?>`, jsp의 `<%= %>`를 생각하자.)

4. method 실행

   `onClick`과 같은 이벤트가 발생했을 때 특정 함수를 실행하고 싶으면 메소드를 넣으면 된다.

5. custom method 구현

   함수 생성을 `handlePlus = () => {}`형식으로 선언을 하였는데, 이부분은 중요하다.

   만약 `handlePlus() {}`로 선언을 한다면 함수 내에서 `this`는 `undefined`가 된다.

   그러므로 꼭 `handlePlus = () => {}` 이렇게 선언을 하자

   > 만약 4번에서 `onClick={() => this.handlePlus()}` 이렇게 사용을 한다면
   >
   > `handlePlus() {}`에서 this는 클래스 내로 정상 작동한다. 주의하자.

6. setState 사용

   리액트에서 `setState`는 state의 값을 변경시켜주는 함수이다.

   절대 this.state.value = "" 을 쓰면 안된다.

함수형 컴포넌트

```
import React from "react";

const Delta = props => {
  // props는 아래에서 자세히 설명한다. 인자값이 한개일 때는 괄호가 생각이 가능하다.
  return (
    <div>
      <input value={props.data} onChange={props.handleChange} />
    </div>
  );
};
// 마찬가지로 export를 해줘야 사용 가능
export default Delta;
```

함수형 컴포넌트는 함수의 인자값으로 props가 날라오고 리턴값을 클래스에서 `render()`의 리턴값과 같이 한다.

사용하는 이유는 성능이 좋고 간단하게 구현할 수 있어서라고 한다.

컴포넌트 사용

```
render() {
  return (
    <Delta data={3} handleChange={(e) => {console.log("changed")}}></Delta>
  )
}
```

여기서 attribute로 data와 onChange를 보내주었는데,

보낸 데이터가 함수형 컴포넌트에서는 `(props) => {}`로 오고 클래스에서는 `this.props`로 온다.

9주차-2

React-Router

React기본적으로 한 페이지(Single Page)위에서 동작한다. 즉 여러 페이지를 보여주기 위해서는

```
render() {
  return (
    <div>
      { location === 0 && <Home />}
      { location === 1 && <Page1 />}
      { location === 2 && <Page2 />}
    </div>
  )
}
```

> Tip: &&와 ||의 특별한 사용
>
> &&나 ||은 보통 if문 내에서 `if (a && b)` 이렇게 두가지를 표현하는데, 다른 사용 방법도 있다.
>
> 기본적으로 &&의 동작방식을 살피면 앞 식이 `false`이면 뒷 식을 검사하지 않고 넘어간다
>
> 그러기 때문에 뒷 값이 리턴이 안되고, 만약 앞 식이 `true`이면 뒷 식의 값을 리턴한다.
>
> ||는 반대로 앞 식의 값이 `true`이면 앞 식을 리턴하고 `false`이면 뒷 식의 값을 리턴한다.

```
const a = obj.a; // 만약 여기서 obj가 undefined면 오류가 발생한다.
// 오류를 발생시키지 않응 방법은 아래 방법이다.
const a = obj && obj.a; // obj가 존재하면 obj.a 리턴
// ||의 사용
const b = obj.b || "b"; // obj.b가 있으면 obj.b 사용 없으면 "b" 사용 (기본 값)
```

이렇게 하거나

React Router를 사용해야 한다.

- 설치

```
npm install --save react-router-dom
# or
yarn add react-router-dom
```

- 사용

```
const App = () => (
  <BrowserRouter>
    <header>
      {/* <a href=""></a>와 같은 링크 정의 */}
      <Link to="/">HOME</Link>
      <Link to="/Page1">Page1</Link>
      <Link to="/Page2">Page2</Link>
    </header>
    <section>
      {/* 화면에 보여질 페이지들 */}
      <Route path="/" exact component={Home} />
      <Route path="/page1" component={Page1} />
      <Route path="/page2" component={Page2} />
    </section>
  </BrowserRouter>
);
export default App;
```

이 두개는 세트로 이루어 져야 한다.

`exact` 는 해당 path가 정확히 같을 때에만 표시하게 해주는 것이다.

만약 `<Route path="/" compo... />` 여기서 exact가 없으면 `/page1` 에서 Home 컴포넌트가 보일 것이다 |

> 여기서는 `() => (<BrowserRouter ...)` 를 하였다. 만약 함수내에 리턴 값만 존재할 경우 줄여 쓸 수 있는데, 예시로는

```
const func = () => {
  return "hello";
};
// 위의 함수는 아래로 쓸 수 있다.
const func = () => "hello";
```

9주차

Mobx 어노테이션 정리

어노테이션	설명
`@observable`	변수를 MobX에서 관리하는 변수로 추가합니다.
`@action`	`observable` 값을 수정하는 메소드를 MobX에게 알립니다.
`@computed`	`observable` 값을 계산하여 리턴하는 메소드에 사용합니다.

MobX 사용 예

```
class UserStore {
  @observable name = "";
  @action setName(newName) {
    this.name = newName;
  }
  @computed get firstName() {
    return this.name[0];
  } // 사용할 때는 userStore.firstName; () 없이 사용
}
```

Mobx-React 어노테이션 정리

어노테이션	설명
`@inject('store1')`	React Component에 props에 해당 스토어들을 주입합니다.
`@observer`	React Component를 MobX관찰 컴포넌트로 추가합니다. (`@inject` 를 사용할 때 써야함.)

`@observer` 와 `@observable` 이랑 헷갈리지 않는 것이 중요하다

Mobx-React 사용 예

```
@inject("stores")
@observer
class Login extends Component {
  render() {
    return (
      <div>
        사용자 이름:
        {this.props.stores.userStore.name}
      </div>
    );
  }
}
```

React 프로젝트에서 MobX 초기화

```
// stores/index.js
import UserStore from "./UserStore";

export default {
  userStore: new UserStore()
};

// index.js
import { Provider } from "mobx-react";
import Stores from "./stores"; // index.js는 생략 가능

const App = () => (
  <Provider stores={Stores}>
    <div>{/* ...나머지들 */}</div>
  </Provider>
);
```

싱글턴 패턴

```
class PostStore {
  static __instance = null;
  static getInstance() {
    if (PostStore.__instance === null) PostStore.__instance = new PostStore();
    return PostStore.__instance;
  }

  constructor() {
    PostStore.__instance = this;
  }
  // 기타 내용...
}
export default PostStore.getInstance();
```

싱글턴 패턴의 특징은 모든 곳에서 같은 객체를 사용하며 공유하여 쓸 수 있다는 장점이 있다.

11주차

Axios 사용

axios는 http클라이언트로 timeout을 설정할 수 있어 서버로부터 아무런 응답이 없을 때 취소할 수 있는 장점을 가진다.

설치

```
npm install --save axios
```

Axios 사용 예

jQuery ajax 사용하는 것과 비슷하다

```
let response = await axios({
  url: "http://localhost:8080/api/board",
  method: "get",
  header: {
    "Content-type": "application/json; charset=UTF-8"
  },
  timeout: 3000
});
if (response.status === 200) {
  console.log(response.data);
}
```