학습결과서

이름 : 이승현

날짜 : 2020.07.03(금)

**오늘 학습 한 내용**

JavaScript for Web Browser (생활코딩)

**- HTML Element / HTML Collection**

호출한 HTML이 단수일경우, 해당 HTML을 추상화 한 객체의 이름은 HTML Element 이고, 복수일 경우에는 해당 HTML을 추상화 한 객체의 이름은 HTML Collection입니다.

모든 엘리먼트는 HTML Element의 자식이다. 따라서 HTMLElement의 프로퍼티를 똑같이 가지고 있다. 동시에 엘리먼트의 성격에 따라서 자신만의 프로퍼티를 가지고 있다.

HTMLCollection은 리턴 결과가 복수인 경우에 사용하게 되는 객체다. 유사배열로 배열과 비슷한 사용방법을 가지고 있지만 배열은 아니다. HTMLCollection 개체의 목록은 따로 변수에 담아 재 조회 하지 않아도 결과가 실시간으로 반영된다

Do it! 리액트 프로그래밍 정석

**- (리액트 ES6문법 액기스) 06. 객체 확장 표현식과 구조 분해 할당**

ES6에서는 이전 자바스크립트 문법에서 객체를 선언할 때 불편했던 점을 개선하고, 객체나 배열의 특정값을 손쉽게 추출할 수 있는 표현식을 추가했습니다.

ex) 객체 선언

var obj = { x : x , y : y }

* var obj = { x, y }

ex) 객체나 배열의 구조 분해

var key1 = obj.key1;

var key2 = obj.key2;

var newKey1 = obj.key1;

* var {

key 1 : newKey1,

key 2

}

**- (리액트 ES6문법 액기스) 07. 라이브러리 의존성 관리**

의존성( Dependency )란?

어떤 코드나 파일에서 다른 코드나 파일을 필요로 하는 것입니다. 주로 객체지향언어에서는 두 클래스 간의 관계라고 말하기도 합니다.

기존 자바스크립트는 라이브러리나 모듈의 의존성을 script 엘리먼트를 이용하여 관리했습니다. 이때 정의되지 않은 함수를 참조하는 의존성 문제가 발생 할 수 있었는데, ES6는 이러한 문제를 import구문을 사용하여 script 엘리먼트 없이 연결된 파일 및 의존파일을 모두 먼저 내려받고 코드를 구동하도록 변경하여 해결하였습니다.

**- (리액트 ES6문법 액기스) 08. 배열 함수**

**keys()와 entries()**

keys()는 배열의 각 인덱스에 대한 key들을 가지는 Array Iterator객체를 반환합니다. entries는 배열 형태로 전부다 반환합니다..

const 큰돌몬 = {

"이름":"큰돌",

"나이":"25"

};

console.log(Object.keys(큰돌몬));

console.log(Object.entries(큰돌몬));

//["이름", "나이"]

//["이름", "큰돌"]["나이", "25"]

**find와 findIndex**

find는 배열의 첫번째요소부터 원하는 "요소"를 찾아서 그 요소를 찾으면 바로.. break를 걸며 그 요소를 반환하며, 못찾을 경우, undefined를 반환합니다. findIndex 는 그 인덱스를 반환합니다. 못찾을 경우, -1을 반환합니다.

const numbers = [1, 3, 4, 5, 6];

const findEven = numbers.find(element => {

return element % 2 == 0;

})

console.log(findEven) //4

const findEvenIndex = numbers.findIndex(element => {

return element % 2 == 0;

})

console.log(findEvenIndex) //2

**forEach, map, filter, reduce**

forEach 는 배열을 이용하여 "깔끔하게" 돌릴 때. (가장 성능이 좋습니다)  
map 은 배열을 이용하여 "새로운" 배열을 만들 때.   
filter 은 배열을 이용하여 "조건"에 맞는 원소로 배열을 만들 때   
reduce는 배열을 통해 "하나의" 값을 추출해 낼때 사용됩니다.

//forEach는 배열을 다시 만들지는 못하지만 요소 안에서 로직을 짤 때 가장 빠릅니다..!

const func1 = (element, index)=>{

console.log(`${index}번째 요소는 ${element} 입니다.`);

}

//map은 새로운 배열..!

const func2 = (element, index)=>{

return element \* 2;

}

//reduce는 배열을 통해 하나의 값을..!

const func3 = (prev, curr, index)=>{

return prev + curr;

}

const sf = [1, 2, 3, 4].forEach(func1);

const sm = [1, 2, 3, 4].map(func2);

const sr = [1, 2, 3, 4].reduce(func3);

console.log(sf, sm, sr)

//sf는 리턴값이 없으므로 undefined를 반환합니다.

//sm은 배의 원소들을 반환하고 sr은 다 합친 10을 반환합니다.

//filter 함수를 돌려서 true값이 나오면 배열에 push를 합니다. 이 때 원소에 어떠한 행위를 할 수는 없습니다.

const s = [1, 2, 3, 4].filter(e=> e % 2);

console.log(s); // [1, 3]

const s = [1, 2, 3, 4].filter(e=> {

if(e % 2){

return true;

}else{

return false;

}

});

console.log(s); // [1, 3]

const s = [1, 2, 3, 4].filter(e=> {

if(e % 2){

return e \* 2;

}else{

return false;

}

});

console.log(s); // 이 때 [1, 3] 요소의 변환은 되지 않습니다.

**every와 some**

every는 배열의 모든 요소가 특정조건을 만족해야 true, some은 하나라도 만족한다면 true, 이것들이 아니라면 false를 반환합니다.

const love = ['나','는','너','를','사랑','해'];

const findLoveSome = love.some(element=>{

return element == '사랑';

})

console.log(findLoveSome) //true

const findLoveEvery = love.every(element=>{

return element == '사랑';

})

console.log(findLoveEvery) //false

**출처 :** <https://m.blog.naver.com/PostView.nhn?blogId=jhc9639&logNo=221147307852&proxyReferer=https:%2F%2Fwww.google.com%2F> **- (리액트 ES6문법 액기스) 09. 비동기 함수**

**비동기 처리란?**

작업시간이 많이 필요한 작업A를 처리하느라 다른 작업들이 대기 하고 있는 상태라면 일단 다른 작업들을 먼저 진행하고 작업A와 작업A와 연관된 작업을 이후에 처리하는 방식을 말합니다.   
비동기는 준비가 완료되는 것 부터 실행하기때문에 동시에 실행될 수 있습니다. 비동기의 가치는 처리중인 지연된 정보를 기다리는동안 다른 코드를 또 실행할 수 있다는 점입니다. API응답을 기다리는동안 사용자는 다른 요청을 할 수 있습니다.

**프로미스(Promise)**

promise로 구현된 비동기 처리 함수는 콜백을 예측 가능한 패턴으로 사용하도록 도와주며, 콜백함수 안에서 생성된 프로미스 객체를 활용해 콜백함수가 성공,실패,오류 각각의 경우에 따라 후속 처리를 할 수 있습니다. 순차적이지 않는 비동기함수의 실행순서를 제어할 수 있게 도와줍니다. 콜백패턴에 비해 코드 가독성이 좋고 반환된 결과물을 사용하기 편합니다.

프로미스를 만드는 방법은 다음과 같습니다.

구현하려는 비동기 함수안에 Promise 객체를 만드는(new) 로직을 넣습니다.

프로미스 생성자 함수에 비동기 처리를 담당할 함수를 인자로 넣고,(=new Promise 생성자 함수 안에 비동기 처리함수를 인자로 넣고) 다시 그 처리 함수는 resolve와 reject 라는 인자를 전달 받습니다. 이 생성자 함수를 new를 통해 인스턴스화 된 객체가 바로 프로미스 입니다.

비동기 처리함수가 어떤 결과값을 갖는냐에 따라, 생성된 프로미스 객체는 아래와 같은 3가지 상태값을 갖습니다.

1. **pending(비동기 처리 로직이 아직 완료되지 않는 상태)**

new Promise로 Promise가 생성된 직후부터 resolve나 reject가 호출되기 전까지의 순간을 pending 이라고 합니다.

1. **fullfilled(처리가 완료되어 프로미스가 결과값을 반환한 상태)**

만약 비동기 처리함수 값이 성공이라면 resolve 메소드가 호출되고,resolve 메소드의 인자로 설정한, 비동기 함수의 결과값이 전달된다. 이 결과값은 then을 통해 후속 처리된다.

1. **Rejected(처리가 실패하거나 오류가 발생한 상태)**

[성공했을 경우]

비동기 처리함수가 성공이라면 ->resolve 메소드 실행 -> 이때 resolve 메소드 인자에 이미 설정한 처리함수의 결과값이 (resolve 메소드를 통해) 이동 -> then메서드를 통해 성공의 후속처리 실행



출처 : <https://moonsupport.tistory.com/m/159?category=797572>

**- (리액트 ES6문법 액기스) 10. 모듈**

일반적으로 모듈은 파일 단위로 분리되어 있으며 애플리케이션은 필요에 따라 명시적으로 모듈을 로드하여 재사용한다. 즉, 모듈은 애플리케이션에 분리되어 개별적으로 존재하다가 애플리케이션의 로드에 의해 비로소 애플리케이션의 일원이 된다. 모듈은 기능별로 분리되어 작성되므로 코드의 단위를 명확히 분리하여 애플리케이션을 구성할 수 있으며 재사용성이 좋아서 개발 효율성과 유지보수성을 높일 수 있다. ES6부터는 모듈 시스템을 사용할 수 있게 만들었지만 브라우저에서 지원이 되지 않고있기때문에 webpack같은 모듈 번들러를 사용해야만 모듈시스템(import, export)을 사용할 수 있다.

**export, import**

export const pi = Math.PI;

 export function square(*x*) {

   return *x* \* *x*;

 }

 export class Person {

   constructor(*name*) {

     this.name = *name*;

   }

 }

export 키워드로 변수, 함수, 클래스를 외부의 스크립트로 모듈화 시킬 수 있다.

또한 export default pi; 이렇게 default키워드를 붙일 수도 있는데 default키워드를 사용하면 이 모듈을 사용하는 스크립트에서 따로 모듈 변수명을 입력하지 않으면 자동으로 default키워드가 있는 모듈을 가져온다.

*// main.js*

import { pi, square, Person } from './lib';

console.log(pi);         *// 3.141592653589793*

console.log(square(10)); *// 100*

console.log(new Person('Lee')); *// Person { name: 'Lee' }*

반대로 가져오는 방법은 위와 같이 "import" 키워드를 사용하고 from 으로 해당 모듈의 js파일을 가리키면 된다.

**오늘 알게된 사실**

- HTML element와 collection에 대해 공부했습니다.

- ES6 배열메소드 여러가지를 알게 되었고, 모듈에 대해 공부하였습니다.

**내가 부족하다고 느낀 부분**

- 아직 react공부할게 산더미입니다.

- ES6 중 비동기에서 Promise관련내용이 조금 난해했습니다. 후에 더 공부해야 할 것 같습니다.

**수업에 바라는 점**

-