





7 Promise, Fetch API, Axios

GitHub - yhyem/mju-likelion-session7

Contribute to yhyem/mju-likelion-session7 development by creating an account on GitHub.

 <https://github.com/yhyem/mju-likelion-session7>

yhyem/mju-likelion-session7



Ax 0 Contributors Issues Stars Forks

JavaScript는 동기식 언어입니다.

자바스크립트는 한 번에 하나의 작업을 수행해요.

각 스레드는 한 번에 하나의 작업만 순차적으로 수행하는데 자바스크립트는 싱글스레드를 사용하기 때문에 기본적으로 동기적으로 코드를 처리합니다.

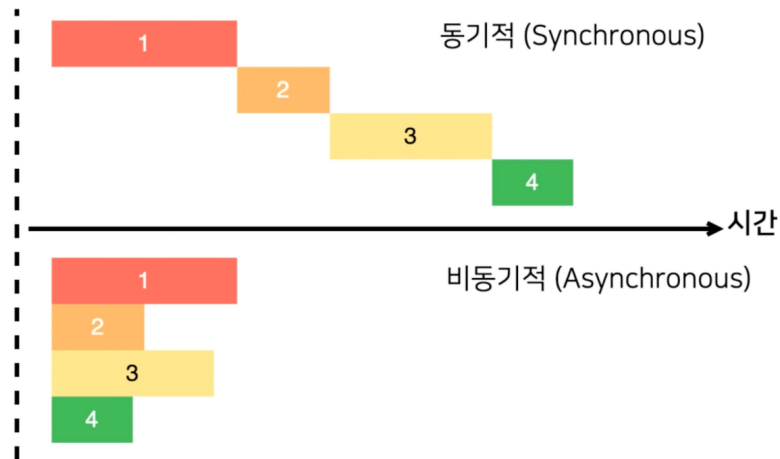
기본적으로 한줄한줄 **순차적**으로 실행합니다.

```
function a() {  
  console.log("a called!");  
}  
  
console.log(1);  
console.log(2);  
a();  
console.log(3);  
console.log(4);
```

▼ 정답

1
2
a called!

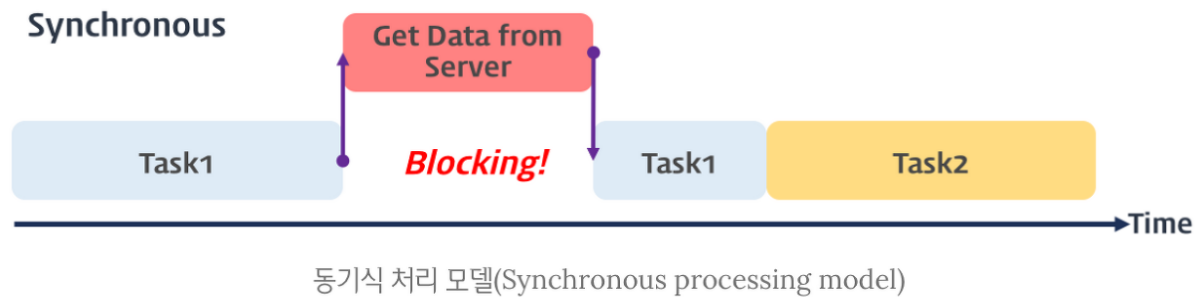
3
4



우리가 서버에 데이터를 요청하면, 데이터 로드가 다 된 후에 우리가 보는 화면에 데이터 표시가 되어야 합니다.

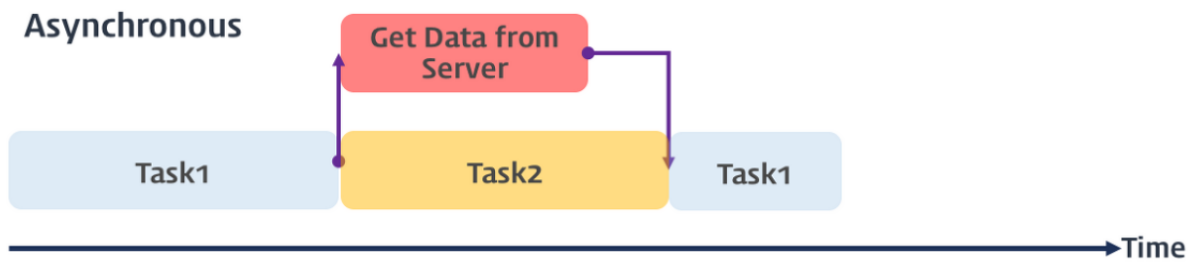
만약 로드가 다 되지 않은 상태로 화면에 데이터를 표시하려고 하려고 한다면 어떻게 될까요?

오류가 발생하게 되겠죠?



그래서 서버로 부터 데이터를 불러오는 무거운 작업 같은 경우에는 비동기적으로 요청해두고(어딘가에서 알아서 무거운 작업을 실행하도록 하고), 바로 다음 작업을 수행할 수 있다면 아주 좋

을 것 같아요!



비동기식 처리 모델(Asynchronous processing model)

Authentication

Use the world's best entertainment API to get information about what movies and TV shows are streaming where, as well as all the metadata you need to build an amazing experience yourself.

 <https://developer.themoviedb.org/reference/intro/authentication>

*API KEY 설정 규칙

.env 파일 생성

.gitignore 파일에 .env 추가

REACT_APP_ 으로 시작해야 하고, quote("")는 필요없다. =으로 블랭크 없이 적어준다

*REST API?

자원을 이름(자원의 표현)으로 구분해 해당 자원의 상태(정보)를 주고 받는 모든 것을 의미합니다.

즉, **자원(resource)의 표현(representation)에 의한 상태 전달**

REST는 기본적으로 웹의 기존 기술과 HTTP 프로토콜을 그대로 활용하기 때문에, **웹의 장점을 최대한 활용할 수 있는 아키텍처 스타일**입니다.

첫 번째,

URI는 정보의 자원을 표현해야 한다.

두 번째,

자원에 대한 행위는 HTTP Method(GET, POST, PUT, DELETE)로 표현한다.

```
GET /members/show/1    (x)
GET /members/1         (o)
```

POST, GET, PUT, DELETE 이 4가지의 Method를 가지고 CRUD를 할 수 있습니다.

METHOD	역할
POST	POST를 통해 해당 URI를 요청하면 리소스를 생성합니다.
GET	GET를 통해 해당 리소스를 조회합니다. 리소스를 조회하고 해당 도큐먼트에 대한 자세한 정보를 가져온다.
PUT	PUT를 통해 해당 리소스를 수정합니다.
DELETE	DELETE를 통해 리소스를 삭제합니다.

다음과 같은 식으로 URI는 자원을 표현하는 데에 집중하고 행위에 대한 정의는 HTTP METHOD를 통해 하는 것이 REST한 API를 설계하는 중심 규칙입니다.

Promise

Promise는 비동기 작업을 조금 더 편하게 처리 할 수 있도록 ES6 에 도입된 기능입니다.

ES6 이전에는 비동기 작업을 처리 할 때에는 콜백 함수로 처리했었습니다.

(비동기로 주어진 일을 모두 마친 뒤에는 이 함수를 실행하도록 추후 업무를 맡겨둔다 라고 생각하시면 됩니다!)

콜백함수 예제

```
//흔히들 말하는 콜백지옥 함수 입니다 :)
function increaseAndPrint(n, callback) {
```

```

    setTimeout(() => {
      const increased = n + 1;
      console.log(increased);
      if (callback) {
        callback(increased);
      }
    }, 1000);
  }

  increaseAndPrint(0, n => {
    increaseAndPrint(n, n => {
      increaseAndPrint(n, n => {
        increaseAndPrint(n, n => {
          increaseAndPrint(n, () => {
            console.log('끝!');
          });
        });
      });
    });
  });
});
});

```

Promise 예제

```

const condition = true;
const promise = new Promise((resolve, reject) => {
  if (condition) {
    resolve('resolved');
  } else {
    reject('rejected');
  }
});

promise
  .then((res) => {
    console.log(res);
  })
  .catch((error) => {
    console.error(error);
  });

```

값이 참이면 `resolve` 를 호출하고, 아닐시에는 `reject` 를 호출합니다.

(우리가 변수명을 정하는 것 처럼, 우리 맘대로 해도 사실 상관은 없지만, 국룰을 따르도록 합시다.)

`resolve` 한 반환 값에 대해서는 `then()` 을 통해 결과 값을 반환 받을 수 있고 `reject` 의 반환 값에 대해서는 `catch()` 를 통해 반환 받습니다.

`then()` 과 `catch()` 문의 체이닝을 통해 비동기 로직의 성공 여부에 따른 분기 처리가 가능합니다

`.then`은 promise 작업이 **성공**했을 때를 가져옵니다.

`.catch`는 promise 작업이 **실패**했을 때를 가져옵니다.

Fetch

정의



fetch란?

fetch는 비동기 적으로 api를 불러오고, 정보를 내보내 주기도 하는 함수
fetch 함수에 쓰여지는 method 설정을 따로 안해주면 기본적으로 get으로 설정이 되어있습니다.

- promise 패턴

```
import { useState, useEffect } from "react";

const Main = () => {
  const [data, setData] = useState([]);

  const options = {
    method: "GET",
    headers: {
      accept: "application/json",
      Authorization: "Bearer " + process.env.REACT_APP_API_KEY,
    },
  };

  //   useEffect(() => {
  fetch(
    "https://api.themoviedb.org/3/movie/popular?language=en-US&page=1",
    options
  )
  )
```

```

    .then((response) => response.json())
    .then((response) => {
      console.log(response);
      // setData(response.results);
    })
    .catch((err) => console.error(err));
// }, []);

return (
  <>
    {data.map((data, index) => (
      <div key={index}>{data.title}</div>
    ))}
  </>
);
};

export default Main;

```

useEffect를 사용하지 않는다면?



스테이트 변경 -> 리렌더 -> 데이터 페칭 -> (state 값 변경으로 인한) 리렌더 -> 리렌더 되니 다시 데이터 페칭 반복으로 무한루프에 빠지게 될거예요!

이때 우리는 useEffect를 이용해서 제한시켜줄수 있습니다. 😊

Promise 는 **내용은 실행 되었지만 결과를 아직 반환하지 않은 객체**입니다.

fetch를 한 후에 날아오는 response 객체는 모든 header가 도착하자마자 우리에게 주어집니다.

그렇기 때문에 현재 자바스크립트는 해당 body 값을 '기다리고 있는 상태'인 Promise 객체를 반환하고

중간에 기다려서 Promise 객체가 다 끝난 상태를 받고 싶지만 그건 불가능합니다.

애초에 Promise를 바로 리턴해버리기 때문에 .then으로 chaining하여 실행해야합니다.

Promise의 작동 방식

1. 우리가 받은 Response 객체는 완전한게 아님. 아직 데이터를 받는 중인 것이기에 Promise를 반환
2. 다 받고 난 뒤에 온전한 Response 객체상태에서 작업
3. 이 방법이 비효율적이라고 느낀다면 await으로 기다린 이후에 .json()을 출력할수 있어요!

- async/await 패턴

```
async function request() {
  const response = await fetch(
    "https://api.themoviedb.org/3/movie/popular?language=en-US&page=1",
    {
      method: "GET", //method 지정을 하지 않을 경우에 default 값은 GET이다. 사실상 필요 없는 코드
      headers: {
        accept: "application/json",
        Authorization: "Bearer " + process.env.REACT_APP_API_KEY,
      },
    }
  );
  const data = await response.json();
  console.log(data); // data를 응답 받은 후의 로직
}

request();
```

장점

- 자바스크립트의 내장 라이브러리 이므로 별도로 import 할 필요가 없음
- Promise 기반으로 만들어졌기 때문에 데이터 다루기 편리
- 내장 라이브러리가기 때문에 업데이트에 따른 에러 방지가 가능

단점

- JSON으로 변환해주는 과정 필요
- 상대적으로 axios에 비해 기능이 부족 (간단한 비동기 작업 Fetch 그렇지 않다면 axios사용 하는 것이 좋음)

Axios

정의



axios란?

Node.js와 브라우저를 위한 Promise API를 활용하는 HTTP 통신 라이브러리입니다.

비동기로 HTTP 통신을 할 수 있으며 return을 promise 객체로 해주기 때문에 response 데이터를 다루기 쉬워요.

설치 방법

```
yarn add axios
```

사용 방법

```
import axios from 'axios';
```

```
axios.get(url)
  .then(function (response) { // 성공했을 때
    console.log(response);
  })
  .catch(function (error) { // 실패했을 때
    console.log(error);
  });
```

가장 큰 차이점은 response 받을때 json 메소드를 사용할 필요가 없습니다.

Axios가 data object를 간단하게 반환 시켜주기 때문입니다.

그리고 request와 관련된 모든 error가 발생한 경우 즉시 catch 구문을 실행하여 찾기 용이합니다.

```
{
  // `data`는 서버가 제공한 응답(데이터)입니다.
  data: {},

  // `status`는 서버 응답의 HTTP 상태 코드입니다.
  status: 200,

  // `statusText`는 서버 응답으로 부터의 HTTP 상태 메시지입니다.
  statusText: 'OK',

  // `headers` 서버가 응답 한 헤더는 모든 헤더 이름이 소문자로 제공됩니다.
  headers: {},

  // `config`는 요청에 대해 `axios`에 설정된 구성(config)입니다.
  config: {},

  // `request`는 요청입니다.
  request: {}
}
```

POST 요청은 다음과 같이 사용할 수 있습니다.

```
axios.post(url, {
  id: '',
  password: '' // 보낼 데이터
})
.then(function (response) { // 성공했을 때
  console.log(response);
})
.catch(function (error) { // 실패했을 때
  console.log(error);
});
```

실습

Authentication

Use the world's best entertainment API to get information about what movies and TV shows are streaming where, as well as all the metadata you need to build an amazing experience yourself.

 <https://developer.themoviedb.org/reference/intro/authentication>

6주차 과제 코드에 week7 브랜치를 판뒤 Axios를 활용해서 데이터를 가져와주세요!

- Data.js 파일 삭제
- 메인 페이지 영화 인기순 api 불러오기
- 영화 세부 페이지 api 불러오기