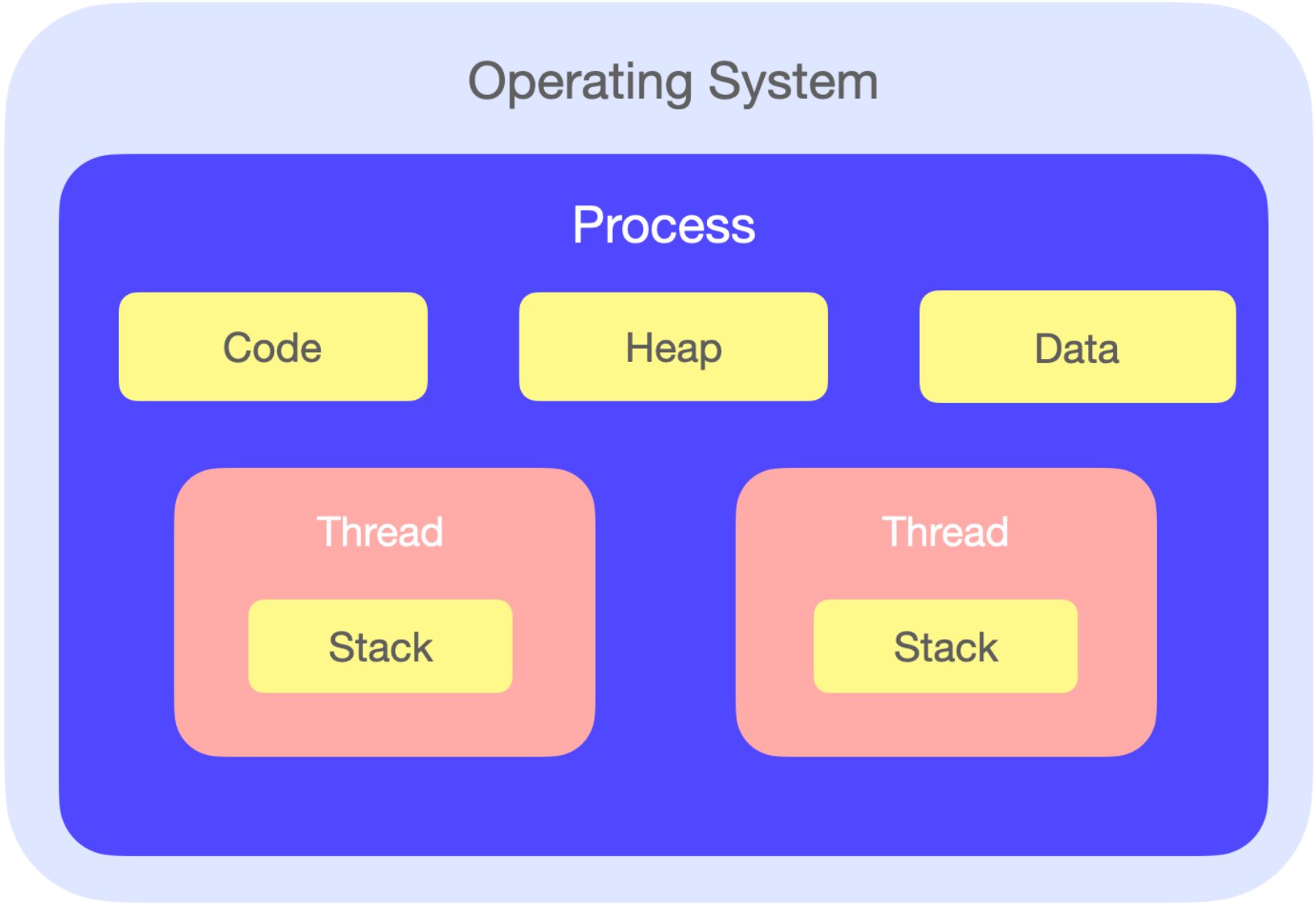
비동기 처리 방식의 종류와 특징

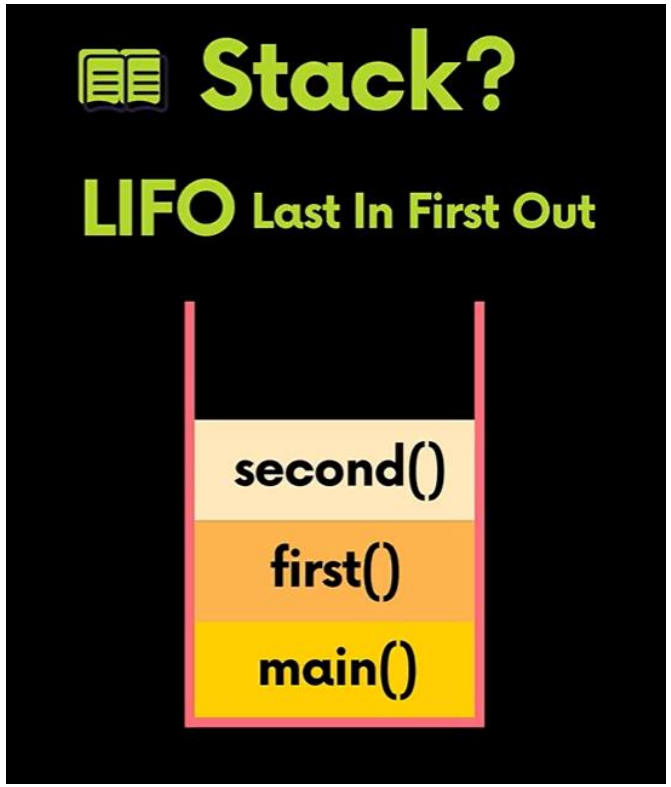
항해99 14기 이준영

동기와 비동기의 이해

자바스크립트는 동기적 언어이며, Single Thread 언어다.

자바스크립트는 hoisting이 된 이후부터, 코드를 한줄씩 스크립팅하여 순차적으로 실행된다. 여기서 자바스크립트는 Single Thread 언어이기 때문에 하나의 Thread만 존재하며, 하나의 stack이 존재한다.


Operating System
Process
Code
Heap
Data
Thread
Stack
Thread
Stack



이 말의 뜻은, 하나의 Call stack에 쌓인 환경정보를 기반으로 컨텍스트를 생성하고 코드를 실행시키기 때문에 LIFO에 의해 순차적으로 순서가 보장된다는 것이며, 즉 자바스크립트 엔진 자체만으로는 멀티쓰레딩이 불가능 하다는 것이다. 현재 task가 종료되기 전까지 다른 task가 실행될 수 없다는 것을 의미하기도 한다.

자바스크립트는 Single thread 언어인데,

어떻게 비동기 처리가 가능할까?

자바스크립트 런타임 환경 !

자바스크립트 엔진 + 웹 브라우저 API + Task Queue + Event Loop

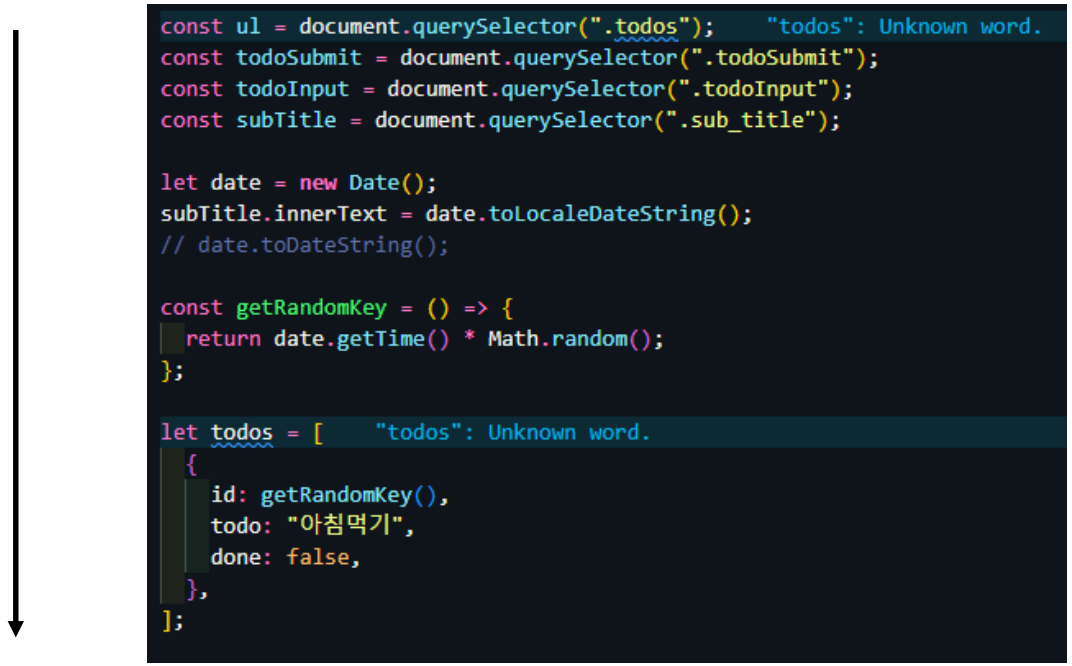
💡 자바스크립트는 Single Thread 언어인데, 어떻게 비동기 처리가 가능할까?

이 부분에 대해서는 자바스크립트 런타임 환경에 대한 이해가 필요하다.
자바스크립트는 런타임 환경에서 동작하게 되는데 이 때 자바스크립트는
자바스크립트 엔진과 웹 브라우저의 웹 API와 함께 동작하게 된다.

웹 브라우저는 멀티쓰레딩을 지원하며, 다양한 비동기 처리 웹 API들과 함께
자바스크립트 엔진에 탑재된 태스크 큐와 이벤트 루프에 비동기 작업에 대한 정보를
전달하여 자바스크립트에서도 비동기적인 작업이 가능하게 한다.

자세한 내용이 궁금하다면 자바스크립트 엔진, 태스크 큐와 이벤트 루프 등을 공부해 보도록 하자.

동기적 처리 방식



```
const ul = document.querySelector(".todos");    "todos": Unknown word.
const todoSubmit = document.querySelector(".todoSubmit");
const todoInput = document.querySelector(".todoInput");
const subTitle = document.querySelector(".sub_title");

let date = new Date();
subTitle.innerText = date.toLocaleDateString();
// date.toDateString();

const getRandomKey = () => {
  return date.getTime() * Math.random();
};

let todos = [    "todos": Unknown word.
  {
    id: getRandomKey(),
    todo: "아침먹기",
    done: false,
  },
];
```

장점 : 현재 task가 종료되어야 다음 task가 실행된다. 동기적, 순차적으로 즉시 실행되어 순서를 보장할 수 있다.

단점 : 현재 task가 종료될 때 까지 다음 task가 실행이 되지 않는 *blocking이 발생한다.

***blocking** : task가 완료될 때 까지 작업이 중단되고 이후 코드들이 실행되지 않는다.

YouTube KR

검색

로그

홈
Shorts
구독

보관함
시청 기록

로그인하면 동영상에 좋아요를 표시하고 댓글을 달거나 구독할 수 있습니다.

로그인

탐색
인기 급상승
쇼핑
음악
영화
실시간
게임
스포츠
학습

채널 탐색

YouTube 더보기

전체 | 음악 | 뉴스 | 실시간 | 게임 | 랩 | 요리 | 축구 | 최근에 업로드된 동영상 | 감상한 동영상

조회수 188만회 • 3일 전

조회수 1398만회 • 7일 전

조회수 591회 • 1일 전

안유진 시켜줘, 놀토 분량 지킴이😎 다른 팀들한테 힌트 될만한 단어는 공개❌ ...

tvN
조회수 45만회 • 3일 전

[리무진서비스] EP.56 만우절 특집 with 방탄소년단 지민 | April Fools' Day...

KBS Kpop
조회수 239만회 • 13일 전

뉴진스, 코카콜라의 초고퀄리티 광고음악, 제로 (Zero) 해외반응 | 해외리액션 |...

옆집 원어민
조회수 26만회 • 9일 전

[SUB] 술 냄새만 남기고 갔단다🍸 뽀에버 YOUNG지수🌸 [차린건 쥐뿔도 없지...

차린건 쥐뿔도 없지만
조회수 1198만회 • 6일 전

Shorts

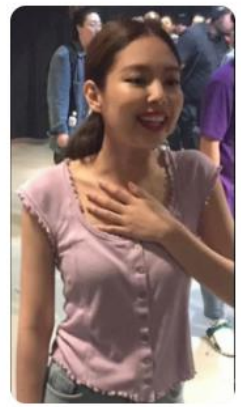
NewJeans (뉴진스) -

솔로지옥 아쉽지만

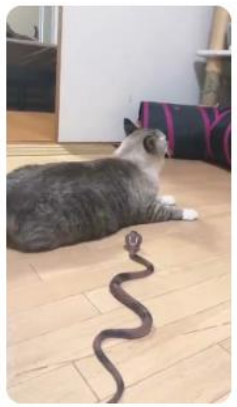
승리 출소 한달 후 근

갈수록 심한 제이홉

[솔로지옥] 슬기 여우

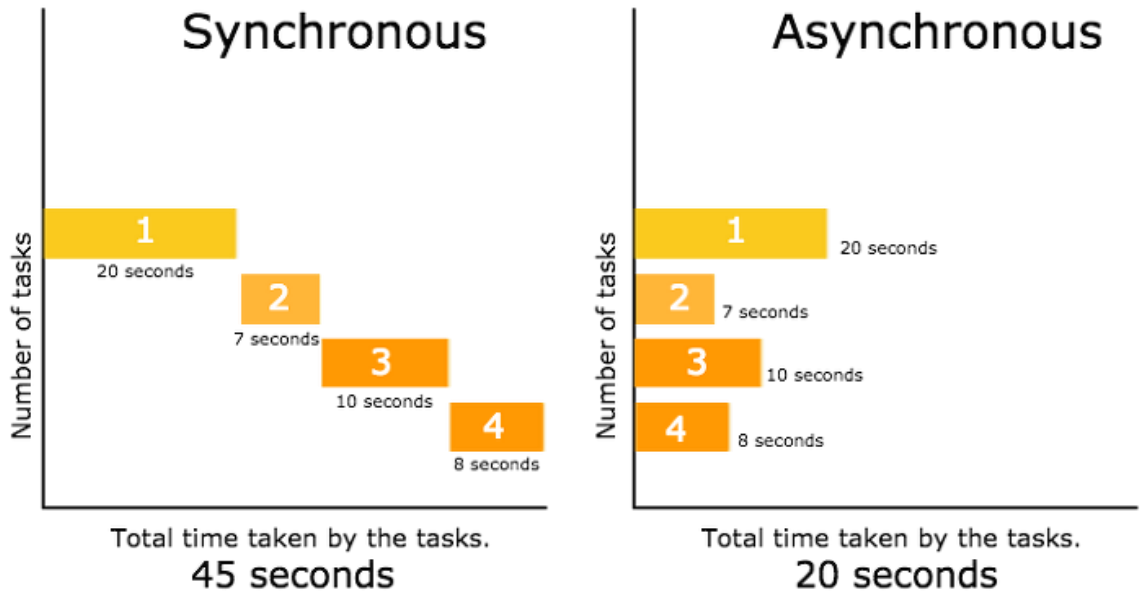


비율 오진다는 블핑



놀랄만한 고양이

윈터 골반 유연성

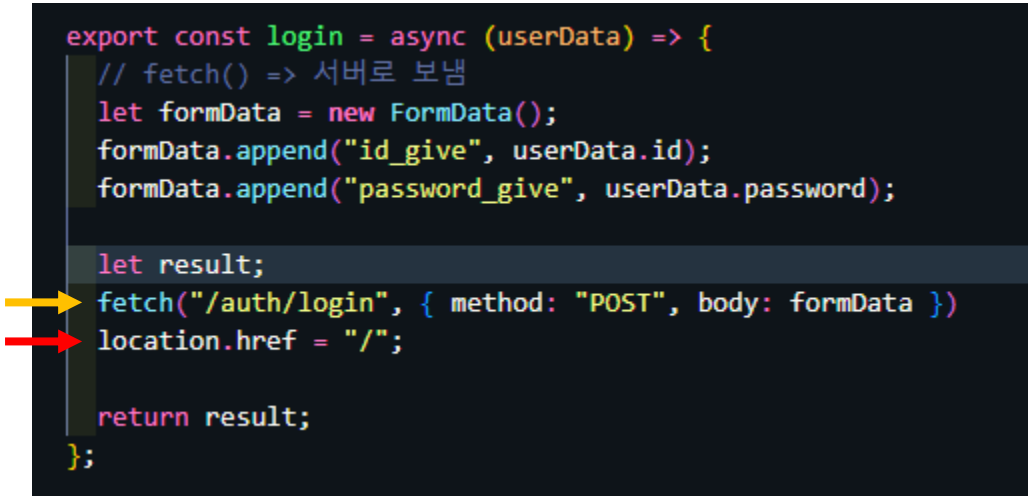


💡 자바스크립트에서 비동기 처리 방식이 필요한 이유

자바스크립트는 Single Thread언어로, 한 번에 하나의 task만 실행할 수 있다.
동기적 방식은 현재 task가 완료될 때 까지 blocking(작업중단)되어 다른 작업들이 일시정지 되고
다음 코드가 읽히지 않는다. 따라서 코드의 복잡도가 높거나 네트워크 통신과 같이 처리시간이 긴 로직을
동기적으로 수행할 경우 blocking되는 시간이 늘어나고, 다른 자바스크립트 코드를 중단시키기 때문에
사용자 불편성을 초래할 수 있다.

이와 같은 이유로 주로 네트워크 통신이나 요청,대기,보류 등 작업이 오래걸리는 코드를 수행할 때에는
비동기적으로 처리하여 해당 task가 진행되는 동안에도 다른 코드들이 실행될 수 있도록 해야 한다.

비동기적 처리 방식



```
export const login = async (userData) => {
  // fetch() => 서버로 보냄
  let formData = new FormData();
  formData.append("id_give", userData.id);
  formData.append("password_give", userData.password);

  let result;
  fetch("/auth/login", { method: "POST", body: formData })
  location.href = "/";

  return result;
};
```

장점 : 현재 실행중인 task가 완료되지 않아도, 다음 task를 실행하여 blocking이 발생하지 않는다.

단점 : task의 실행 순서를 보장할 수 없다. 비동기 처리 방식의 순서를 동기적으로 제어하지 않으면, 순서에 의존적인 작업을 수행할 경우 이전 task가 완료되기 전에 다음 task가 실행된다.

ex) 로그인 후 => 메인페이지로 이동 해야하는데, 로그인 요청 후 로그인 처리가 되기도 전에 메인페이지로 이동해 버린다.

비동기적 처리 방식

💡 비동기적 처리 방식의 순서를 제어하는 방법
이와 같이 비동기적 처리 방식에서 순서를 보장하지 못한다는 단점을 보완하기 위해
비동기를 동기적으로 순서를 제어할 수 있는 문법 등을 사용해야 하며
callback함수, Promise, async-await 등과 같은 문법들이바로 비동기를 동기적인 것처럼 순서를 제어하는 방법

비동기를 동기적으로 제어하는 방법

콜백 함수(Call back)

비동기를 동기적으로 제어하는 방법 - 콜백함수

1. Callback 함수

Callback 함수가 모두 비동기적인 것은 아니다. 동기와 비동기로 나뉘어 진다.

> 장점 : 비동기 콜백 내부에서 다음 작업을 호출하여 비동기 처리의 실행 순서를 제어할 수 있다.
>
> 단점 : 웹의 복잡도가 증가할 수록 비동기 코드가 증가하여 콜백 지옥에 빠질 가능성이 크다.

비동기를 동기적으로 제어하는 방법 - 콜백함수

2) 비동기적 콜백 : 실행 중인 코드의 완료 여부와 무관하게 바로 다음 코드로 넘어가는 방식

=> addEventListener(), setTimeOut() 등

```
// setTimeout
setTimeout(function() {
  console.log("Hello, world!");
}, 1000);
```

비동기를 동기적으로 제어하는 방법 - 콜백함수

3) **콜백 지옥 :** 비동기 콜백들의 중첩 사용으로 인한 콜백 지옥

웹의 복잡도가 증가할 수록 비동기 코드가 늘어나 콜백 지옥에 빠질 가능성이 높다.

> 단점
> 1. 가독성이 굉장히 많이 떨어진다. 연결관계나 로직을 한눈에 이해하기가 굉장히 어렵다.
> 2. error 발생시 디버깅도 굉장히 어렵고 유지보수도 힘들다.

비동기를 동기적으로 제어하는 방법 - 콜백지옥

```
// 콜백 지옥
class UserStorage {
  loginUser(id, password, onSuccess, onError) {
    setTimeout(() => {
      // 실제 백엔드가 없기 때문에 login시 걸리는 시간처럼 가정해 본 것
      if (
        (id === "junyoung" && password === "junyoung123") ||
        (id === "lee" && password === "lee123")
      ) {
        onSuccess(id); // id 전달
      } else {
        onError(new Error("not found")); // 콜백 > new Error object에 not found 전달
      }
    }, 2000);
  }

  // role : 역할 > 사용자 개인마다의 AD,guest역할 등의 정보를 서버에 요청해서 받아오는 함수
  getRoles(user, onSuccess, onError) {
    setTimeout(() => {
      if (user === "junyoung") {
        onSuccess({ name: "junyoung", role: "admin" }); //object를 전달
      } else {
        onError(new Error("no access")); // 콜백 > new Error object에 not found 전달
      }
    }, 1000);
  }
}
```

비동기를 동기적으로 제어하는 방법 - 콜백지옥

```
userStorage.loginUser(
  id,
  password,
  (user) => {
    // login성공 onSuccess시
    userStorage.getRoles(
      user,
      (userWithRole) => {
        alert(
          `Hello ${userWithRole.name}, you have a ${userWithRole.role} role`
        );
      },
      (error) => {
        console.log(error);
      }
    );
  },
  (error) => {
    // login실패 onError시
    console.log(error);
  }
);
```

비동기를 동기적으로 제어하는 방법 - 콜백지옥

이 역시도 근본적인 해결책은 아닌 것 같다. 위와 같은 콜백 지옥을 벗어나 좀 더 동기적으로, 가독성 좋게 보일 수 있도록 자바스크립트에서는 비동기적인 작업을 동기적으로(동기적인 것 처럼 보이도록) 처리해주는 문법을 계속해서 마련해주고 있다.

Promise, async/await(ES7)같은 것들이다. 위 코드를 Promise를 사용하여 좀 더 동기적으로, 가독성 좋게 개선시켜 보도록 하자.

비동기를 동기적으로 제어하는 방법

프로미스(Promise)

비동기를 동기적으로 제어하는 방법 - Promise

2. Promise 객체

Promise는 ES6에 등장한 비동기를 처리하는 Object이고, **비동기 처리가 끝나면 알려달라는 '약속'**을 의미한다.

프로미스는 비동기 처리 기능을 수행하고 나서, **정상적으로 기능을 수행하였다면 성공의 메세지와 함께 처리된 결과값을 전달하고, 기능 수행 중 예상치 못한 문제 발생 시 에러를 전달**해 준다.

> **장점 : 콜백함수에 비해 가독성도 좋고 비동기 처리를 동기적으로 보이게 하여 순서를 파악하기 쉽다.**
>
> **단점 : 콜백지옥과 같은 맥락으로 .then()을 연쇄적으로 호출하다 보면 코드가 조금 난잡해지거나 error가 발생했을 때, 몇번째 .then()에서 발생한 에러인지 가독성 등에서 직관적으로 파악하기 어려워질 수 있다.**

비동기를 동기적으로 제어하는 방법 - Promise

프로미스를 이해하기 위해서는 2가지 포인트를 중점으로 봐야 한다.

1. 상태(state) : 수행중(pending)상태, 성공적으로 완료 상태(fulfilled), 파일을 찾을 수 없거나 네트워크 통신 문제 상태 (rejected) => 프로미스가 무거운 기능을 수행중인지, 성공하였는지, 실패하였는지 상태를 이해한다.

2. 데이터를 제공(producer)하는 역할과, 제공된 데이터를 쓰는(consumer) 역할의 차이점에 따른 견해를 이해해야 한다.

비동기를 동기적으로 제어하는 방법 - Promise

1) Producer와 Consumer 관점에서의 Promise

데이터 제공

```
// 1.Producer // 우리가 원하는 기능을 비동기적으로 실행하는 promise 생성
// resolve(성공시 최종 데이터를 전달하는)와 reject(문제가 생기면 호출하는) 두가지를 받는 콜백함수
const promise = new Promise((resolve, reject) => {
  console.log("doing something...");
  // 비동기 처리, 데이터 통신 로직
  setTimeout(() => {
    resolve("junyoung");
    // reject(new Error("no network"));
  }, 2000);
});
```

데이터 사용

```
// 2. consumer // producer의 promise 데이터를 사용 > then, catch, finally > 를 통해 값을 받아올 수 있다.
promise
  .then((value) => {
    console.log(value);
  }) // then을 호출하게 되면, 결국 같은 promise를 다시 return하기 때문에, .catch()를 다시 호출이 가능하다
  .catch((error) => {
    console.log(error);
  })
  // finally > 최근에 추가됨 > 성공과 실패 여부와 상관없이 마지막에 무조건 호출되어지는 함수
  .finally(() => {
    console.log("finally");
  });
```

비동기를 동기적으로 제어하는 방법 - Promise

2) 프로미스 체이닝

```
// 서버에서 숫자를 받아오는 promise 생성
const fetchNumber = new Promise((resolve, reject) => {
  setTimeout(() => resolve(1), 1000);
});

fetchNumber
  .then((num) => num * 2) // then 은 값을 바로 전달할 수도 있고, 다른 비동기인 promise를 전달할 수 도 있
  .then((num) => num * 3)
  .then((num) => {
    // 6
    return new Promise((resolve, reject) => {
      setTimeout(() => resolve(num - 1), 1000); // 5
    });
  })
  .then((num) => console.log(num)); // 5
```

콜백지옥 Promise로 개선하기

콜백 지옥

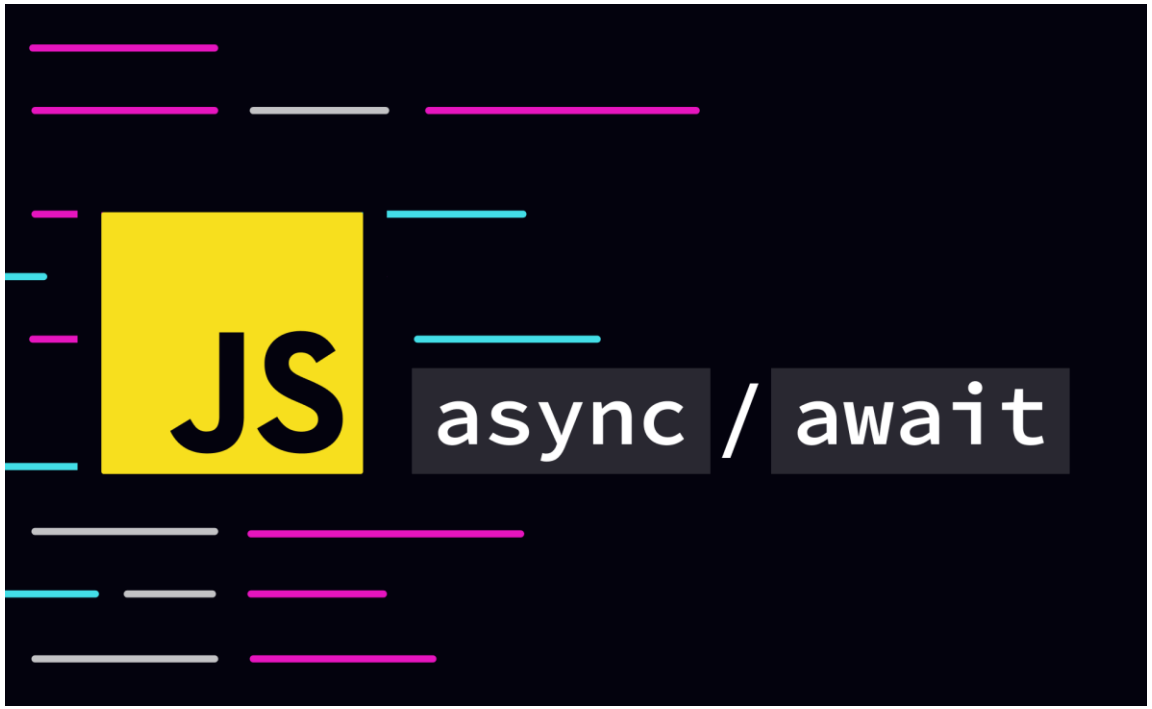
```
userStorage.loginUser(
  id,
  password,
  (user) => {
    // login성공 onSuccess시
    userStorage.getRoles(
      user,
      (userWithRole) => {
        alert(
          `Hello ${userWithRole.name}, you have a ${userWithRole.role} role`
        );
      },
      (error) => {
        console.log(error);
      }
    );
  },
  (error) => {
    // login실패 onError시
    console.log(error);
  }
);
```

콜백지옥 Promise로 개선하기

Promise로 개선된 코드

```
userStorage
  .loginUser(id, password) // id, password 정보 가져오기 성공하면,
  .then(userStorage.getRoles) // userStorage.getRoles호출
  .then((user) => alert(`Hello ${user.name}, you have a ${user.role} role`)) // us
  .catch(console.log); // 에러시 콘솔출력
```

비동기를 동기적으로 제어하는 방법



async - await

비동기를 동기적으로 제어하는 방법 – 'async-await'

3. async - await

async - await은 ES7에서 추가된 syntatic sugar(기존 프로토타입 위에 문법적으로 추가된)이다.

Promise 체이닝에서 **.then()을 연쇄적으로 호출하다 보면 코드가 난잡해져 가독성이 떨어질 수도 있고, error가 발생했을 때, 몇번째 .then()에서 발생한 에러인지 직관적으로 파악하기 어려워질 수 있다.** 이런 불편 사항을 보완하기 위해 async-await키워드가 추가됐다.

비동기를 동기적으로 제어하는 방법 – 'async-await'

> **사용 방법**
> 비동기 작업을 할 함수 앞에 **async**를 붙여주고, 실질적으로 비동기 처리를 하는 위치마다 **await**을 붙여 준다.
>
> **장점**
> 1. 비동기 코드를 동기 코드처럼 보이게 작성해 **가독성을 높일 수 있고 사용 방법이 굉장히 간단**하다.
> 2. async-await을 이용하면 **await이 대기를 처리해주기 때문에 .then이 거의 필요하지 않다.** 또 Promise에서 사용하던 **.then과 .catch 대신 일반적으로 성공과 실패를 처리하는 try/catch 사용할 수 있다**는 장점도 생긴다.

=> 반드시 프로미스를 대체하여 async-await을 사용해야 하는 것은 아니다. 프로미스를 사용해야 할 때와 async-await을 사용해야 좀 더 코드가 깔끔해지는 경우를 잘 구분하여 사용하도록 한다. 프로젝트를 하면서 감을 찾는 것이 좋다.

비동기를 동기적으로 제어하는 방법 – 'async-await'

```
// 실제 현업에서 많이 사용되는 코드
export const login = async (payload: { email: string; password: string }) => {
  try {
    const { data } = await axiosAuth.post('login', payload);
    return data;
  } catch (error) {
    if (error instanceof AxiosError) {
      console.error(error.response?.data.detail);
    }
    return false;
  }
};
```

수고하셨습니다!