1. 비동기처리

동기 처리

- 동기적 제어 흐름은 현재 실행중인 코드가 종료되기 전까지 다음 줄의 코드를 실행하지 않는 것을 의미한다.
- 분기문, 반복문, 함수 호출 등이 동기적으로 실행된다.
- 코드의 흐름과 실제 제어 흐름이 동일하다.
- 싱글 스레드 환경에서 메인 스레드를 긴 시간 점유하면, 프로그램을 멈추게 한다.

```
// 동기적 코드에는 어떤 것들이 있을지 !

// 선언문
let a = 10

// 동기적 실행
console.log("a : ", a)

// 반복문
function foo(num) {
  for (let i = 0; i < 10; ++i) {
    console.log(num)
  }
}

// 함수 호출
foo(num)

//등이 있다 !
```

비동기 처리

- 비동기적 제어 흐름은 현재 실행중인 코드가 종료되기 전에 다음 라인의 코드를 실행하는 것을 의미한다.
- 프로미스, 콜백 함수를 호출하는 함수 등은 비동기적으로 실행된다.

- 코드 흐름과 실제 제어 흐름이 다르다.
- 비동기 작업을 기다리는 동안 메인 스레드는 다른 작업을 처리한다.

```
let a = 10

setTimeout(function callback() {
  console.log('a : ', a)
}, 3000)

console.log('Finished')

//Finished
//a: 10
```

이벤트루프

- 자바스크립트 엔진은 비동기 처리를 제공하지 않는다.
- 대신, 비동기 코드는 정해진 '함수'를 제공하여 활용할 수 있다.
- 이 함수들을 API(Application Programming Interface)라 한다.
- 비동기 API의 예시로, setTimeout, XMLHttpRequest, fetch 등의 Web API가 있다.

비동기 처리의 필요성

웹 애플리케이션에서는 종종 서버에서 데이터를 받아와야 하는 경우가 있다. 이때 데이터를 요청하고, 서버가 응답을 줄 때까지 기다리는 동안 다른 작업이 멈추면 안 되기 때문에 비동기 처리가 필요하다.

웹사이트에 접속했는데 비동기 처리가 안된다면?

페이지 로딩이 지연되는 문제가 발생할 수 있다. 웹 페이지에서 네트워크 요청(예: API 호출)이 동기적으로 처리되면, 그 요청이 완료될 때까지 다른 작업이 실행되지 않기 때문에 화면에 페이지 데이터가 로딩되는 데 시간이 걸리거나, 웹 애플리케이션이 멈추는 것처럼 보일 수 있다.

비동기 처리 예시

1. API 요청 (웹에서 서버로 데이터 요청)

웹 애플리케이션에서는 종종 서버에서 데이터를 받아와야 하는 경우가 있습니다. 이때 데이터를 요청하고, 서버가 응답을 줄 때까지 기다리는 동안 다른 작업이 멈추면 안 되기 때문에 비동기 처리가 필요합니다

```
function fetchDataFromAPI() {
    console.log("데이터 요청 중...");
    // 3초 후에 서버 응답을 받는 것처럼 시뮬레이션
    setTimeout(() => {
        console.log("서버 응답 완료!");
        // 데이터 처리
    }, 3000);

    console.log("다른 작업 실행 중...");
}

fetchDataFromAPI();
```

<출력결과>

```
데이터 요청 중...
다른 작업 실행 중...
(3초 후)
서버 응답 완료!
```

그래서 데이터를 받아오는 동안 다른 작업을 계속 실행할 수 있도록 비동기 처리가 필요하다.

2. 파일 읽기 (서버 사이드 예시)

서버에서 파일을 읽는 작업도 시간이 걸리기 때문에 비동기적으로 처리하여, 읽는 동안 다른 작업을 처리할 수 있도록 해야 한다.

<예시> (Node.js 환경)

```
const fs = require('fs');

console.log("파일 읽기 시작");

fs.readFile('example.txt', 'utf8', (err, data) => {
    if (err) {
        console.log("파일 읽기 실패:", err);
    } else {
        console.log("파일 내용:", data);
    }
});
```

```
console.log("파일 읽는 동안 다른 작업 수행");
```

<실행결과>

```
파일 읽기 시작
파일 읽는 동안 다른 작업 수행
(파일이 읽히고 나서)
파일 내용: (파일 내용이 출력됩니다)
```

파일을 읽는 동안 서버는 다른 작업을 처리할 수 있어야 한다. 만약 동기적으로 처리된다면, 파일을 읽는 동안 서버가 멈추고 다른 요청을 처리하지 못할 것이다.

3. 사용자 입력 처리

사용자가 버튼을 클릭하거나 폼을 제출할 때 시간이 오래 걸리는 작업이 있을 수 있다. 예를 들어, 입력을 받은 후 서버로 데이터를 보내고 응답을 기다리는 동안 UI가 멈추지 않도록 비동기 처리가 필요하다.

<예시>

```
document.getElementById("submitButton").addEventListener("click", function() {
    console.log("폼 제출 시작...");

    // 비동기적으로 데이터를 서버로 보내는 예시
    setTimeout(() => {
        console.log("서버에 데이터 전송 완료!");
    }, 2000);

    console.log("폼 제출 대기 중...");
});
```

<실행결과>

```
폼 제출 시작...
폼 제출 대기 중...
(2초 후)
서버에 데이터 전송 완료!
```

사용자가 폼을 제출하는 동안 다른 작업을 할 수 있어야 한다. 만약 동기적으로 처리된다면 서버 응답을 기다리는 동안 화면이 멈추게 되기 때문이다.

비동기 처리 모델

=> 메인 스레드가 콜스택을 읽고 코드를 실행함, 콜스택은 아래부터 차례로 실행 컨텍스트가 쌓이게 된다,
예를들어 setTimeout은 WEB API에 등록되었을 때 몇 초 뒤 종료되면 task queue로 들어가도록 설정
되어있다.

- 비동기 코드를 처리하는 모듈은 자바스크립트 엔진 외부에 있다.
- 이벤트 루프(event loop), 태스크 큐(task queue), 잡 큐(job queue) 등으로 구성된다.
- API모듈은 비동기 요청을 처리 후 태스크큐에 콜백함수를 넣는다.
- 모든 코드가 실행 된 후 콜 스택이 비어있으면 자바스크립트 엔진(API 모듈 등도 마찬가지)은 이벤트
  루프를 통해 태스크 큐의 콜백 함수를 콜스택에 넣고 실행한다.

image.png

```
request("user-data", (userData) => {
  console.log("userData로드 ")
  saveUsers(userData)
});

console.log("DOM 변경 ")
console.log("유저 입력 ")
```

request는 비동기적으로 동작해서 비동기 처리가 끝났을 때, userData라는 콜백함수 task queue에
넣어주는 역할을 한다.

Revision #7
Created 3 January 2025 20:48:02 by DainChoi
Updated 3 January 2025 21:31:53 by DainChoi