10/4 강의자료

1교시

01. 동기 비동기 타이머 다르게 구현해보기 (월요일 자료와 비슷)

▼ Date.now()

console.log 함수의 console 객체 처럼 JS가 미리 가지고 있는 객체를 내장객체라고 합니다.

이중 대표적인 예로 Date객체가 있습니다.

먼저 new라는 키워드로 Date 객체를 만들 수 있습니다.

현재시간, 특정 시간 등으로 여러 시간대를 만들어 줄 수 있습니다.

```
// 현재 시간
let tmp = new Date();
console.log(tmp);

// 특정한 시간
tmp = new Date(1000); // UTC(1970년 1월 1일 00:00:00) 기준 + 인자값밀리초
console.log(tmp);

// 문자열
tmp = new Date("2022-10-04"); // 특정 날짜의 자정(new Date("YYYY-MM-DD"))
console.log(tmp);

tmp = new Date("2022-10-04T11:11:11"); // 특정 날짜 특정 시간(new Date("YYYY-MM-DDThh:mm:ss"))
console.log(tmp);

tmp = new Date("10/04/12 11:11:11");
console.log(tmp);

tmp = new Date("October 4, 2022 11:11:11");
console.log(tmp);

tmp = new Date("oct 4 2022 11:11:11");
console.log(tmp);

// 인자값
tmp = new Date(2022, 9, 04, 11, 11, 11); // 특정 날짜 특정 시간(new Date(YYYY,MM,DD,hh,mm,ss))
console.log(tmp);

// node.js 에서는 바로 인식 , Window에서는 Index로 인식
tmp = new Date(2022, 10); // 특정 날짜 특정 시간(new Date(YYYY,MM[, 1,0,0,0,0]))
console.log(tmp);

// Time stamp (기준값으로 부터 얼마나 지났는지)
```

```
console.log(tmp.getTime());

// 다양한 값들
console.log(tmp.getFullYear());
console.log(tmp.getMonth()); // window에서는 0 이 1월입니다.
console.log(tmp.getDate());
console.log(tmp.getDay()); // 요일은 0 이 일요일 입니다.
console.log(tmp.getHours());
console.log(tmp.getMinutes());
console.log(tmp.getSeconds());
console.log(tmp.getMilliseconds());

// Data의 여러가지 메서드

// 생성된 날짜 수정하기 set
tmp.setFullYear(2030);
console.log(tmp);

console.log(tmp.toLocaleDateString()); // 날짜 정보
console.log(tmp.toLocaleTimeString()); // 시간 정보
console.log(tmp.toLocaleString()); // 날짜 시간 정보

// 추가기능 (자동 수정)
tmp = new Date(2022, 9, 10);
console.log(tmp);

console.log(Date.now()); // 호출한 순간의 타임스탬프

// 형변환
console.log(typeof tmp); // object
console.log(String(tmp)); // 날자값 그대로 문자열
console.log(Number(tmp)); // getTime()을 통해 얻은 타임스탬프와 동일
console.log(Boolean(tmp)); // true

// Data끼리 연산
let tmp2 = new Date(2022, 10);

const timeDiff = tmp2 - tmp;
console.log(timeDiff); // ms
console.log(timeDiff / 1000); // sec
console.log(timeDiff / 1000 / 60); // min
console.log(timeDiff / 1000 / 60 / 60); // hour
console.log(timeDiff / 1000 / 60 / 60 / 24); // date
```

cf. 월 약자

1월 January (Jan.)

2월 February (Feb.)

3월 March (Mar.)

4월 April (Apr.)

5월 May (May)

6월 June (Jun.)

7월 July (Jul.)

8월 August (Aug.)

9월 September (Sep.)

10월 October (Oct.)

11월 November (Nov.)

12월 December (Dec.)

더 많은 내용 :
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Date

03. 콜백 대신 Promise 써보기

▼ Promise란?

Promise API는 비동기 API중 하나로 task queue가 아닌 Job queue를 사용합니다. (우선순위가 잡큐가 더 놓음 잡큐가 끝나면 테스크큐가 실행됩니다.)

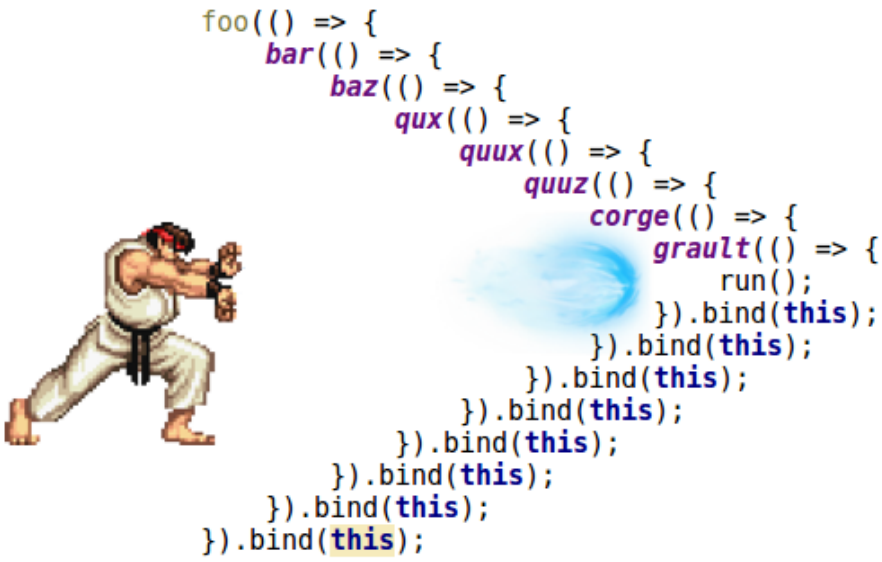
Promise는 비동기 작업을 표현하는 JS객체로 진행(pending), 성공(fulfilled, resolved), 실패(reject) 상태가 있습니다.

settled	비동기 처리가 수행된 상태 (성공 또는 실패)	resolve 또는 reject 함수가 호출된 상태

자세한 사용법은 4번 문제에서 배워보겠습니다!

▼ callback과 Promise 차이점

callback을 사용하면 비동기 로직의 결과값을 처리하기 위해 callback안에서만 처리를 해야하고 callback 밖에서는 비동기에서 온 값을 알 수가 없습니다. 이러다보니 callbak hell이 생기게 되죠.



하지만 Promise는 비동기에서 온 값이 Promise안에 저장되기 때문에 밖에서도 사용할 수 있어 코드 작성이 편리해집니다.

또한 error처리에서 callback 방식의 비동기 처리는 error를 핸들링할 수 없습니다. 비동기 처리 함수의 콜백 함수는 Task queue로 이동한 후 call stack이 비어졌을 때, call stack으로 이동되어 실행됩니다. 그러면서 비동기를 불러온 함수는 이미 call stack에서 사라졌기 때문에 error를 캐치하지 못합니다.

하지만 Promise를 사용하면 Promise 객체 안에 정보가 저장되기 때문에 catch를 통해 error를 핸들링 할 수 있습니다.

코드 예시를 보며 이해해보아요!

2교시

04.Promise 로 타이틀 바꾸기

▼ Promise 사용법

먼저 resolve, reject를 인자로 받는 callback 함수를 넣어 Promise객체를 만들어줍니다.

const promiseObj = new Promise((resolve, reject) ⇒ {})

promiseObj.then(() ⇒ {]).catch(() ⇒ {}).finally(() ⇒ {})

promiseObj.then(() ⇒ {] , () ⇒ {}).finally(() ⇒ {})

메서드 체이닝 : 동일한 객체에 메서드를 연결하여 사용

then, catch를 통해 Promise 객체를 리턴하여 then,catch를 추가적으로 사용할 수 있습니다.

Promise.resolve( ), Promise.reject( )

: static 메서드로 인자로 받은 값을 Promise로 래핑해줍니다.

생성자를 만들지 않고 바로 사용가능하여 동기 코드를 비동기로 변경할 때 유용합니다.

Promise.all([ ])

: static 메서드로 안에 들어가는 array 인자가 모두 성공하면 Promise의 resolve를 인자 순서대로 배열로 반환 / 하나라도 실패하면 실패함 가장 먼저 실패한 Promise의 reject를 반환 합니다.

Promise.race([ ])

: static 메서드로 안에 들어가는 array 인자중 가장 먼저 성공하는 Promise의 resolve를 반환 / 하나라도 실패하면 실패함 가장 먼저 실패한 Promise의 reject를 반환 합니다.

06. fetch 사용하여 json 데이터 가져오기 2

▼ fetch 알아보기

ES6에서 나온 메서드로 이전에는 HTTP통신을 하기 위해서는 **XMLHTTPRequest를 사용해야 했습니다.**

이를 사용하기 위해서는

```
var xmlHttp = new XMLHttpRequest(); // XMLHttpRequest 객체를 생성함.

xmlHttp.onreadystatechange = function () {
    // onreadystatechange 이벤트 핸들러를 작성함.

    // 서버상에 문서가 존재하고 요청한 데이터의 처리가 완료되어 응답할 준비가 완료되었을 때

    if (this.status == 200 && this.readyState == this.DONE) {
        // 요청한 데이터를 문자열로 반환함.

        document.getElementById("text").innerHTML = xmlHttp.responseText;
    }
};

xmlHttp.open("GET", "../1.06/employees.json", true);

xmlHttp.send();
```

위와 같이 매우 번거롭고 복잡했습니다.

또한 이벤트 기반이라 응답을 받을 때 State를 바꾸는 이벤트를 발생시키고, 이를 onreadystatechange를 통하 감지하는 방식으로 작동합니다. 하지만 초기 작품답게 브라우저 별로 불일치가 발생하며 Promise기반이 아니기 때문에 메서드 체이닝을 할 수 없어 콜백지옥이 발생하는 등 가독성, 작성력이 떨어집니다.

이를 fetch를 통해 편리하게 만들어 주었는데요. Promise기반이라 체이닝이 가능하며 나중에 배울 async, await 등을 통해 콜백지옥에서 쉽게 벗어날 수 있습니다. 또한 대부분 브라우저가 지원합니다.

cf. fetch로는 404, 500 에러를 잡아 낼 수 없다.

▼ JSON 이란?

• JavaScript Object Notation라는 의미의 축약어로 데이터를 저장하거나 전송할 때 많이 사용되는 경량의 DATA 교환 형식입니다. 사람과 기계 모두 이해하기 쉬우며 용량이 작아서, 최근에는 JSON이 XML을 대체해서 데이터 전송 등에 많이 사용합니다.

자주쓰는 메서드

- **JSON.parse(JSON으로 변환할 문자열)** : JSON 형식의 텍스트를 자바스크립트 객체로 변환한다.
- **JSON.stringify(JSON 문자열로 변환할 값)** : 자바스크립트 객체를 JSON 텍스트로 변환한다.