WebStorage / AJAX

WebStorage

> 쿠키와는 별개로 클라이언트(웹 브라우저)에 저장되는 키/밸류 형태의 데이터

- 키/밸류는 문자열 데이터만 가능하다

Session Storage

- 브라우저 탭을 닫으면 자동으로 지워지는 데이터

Local Storage

- 영구적으로 저장할 수 있는 데이터

```
💡쿠키와의 차이

- 쿠키는 HTTP 헤더에 담겨 서버와 통신하는데 사용되는 데이터
- WebStorage는 클라이언트 측에서만 저장하는 데이터
- 쿠키보다 큰 데이터를 저장할 수 있다
```

사용방법

```
const localStorage= window.localStorage
const sessionStorage = window.sessionStorage
```

- 저장

```
storage.setItem("key", "value");
```

- 조회

```
storage.getItem("Key");
```

- 삭제

```
storage.removeItem("key");
```

- 전체 삭제

```
storage.clear();
```

Asynchronous Javascript And XML (AJAX)

> 현재 페이지 내에서 JavaScript를 이용하여 비동기적으로 요청을 보내는 방법

- 브라우저를 통해서 요청을 보내는 것이 아니므로 새로고침을 할 필요가 없다

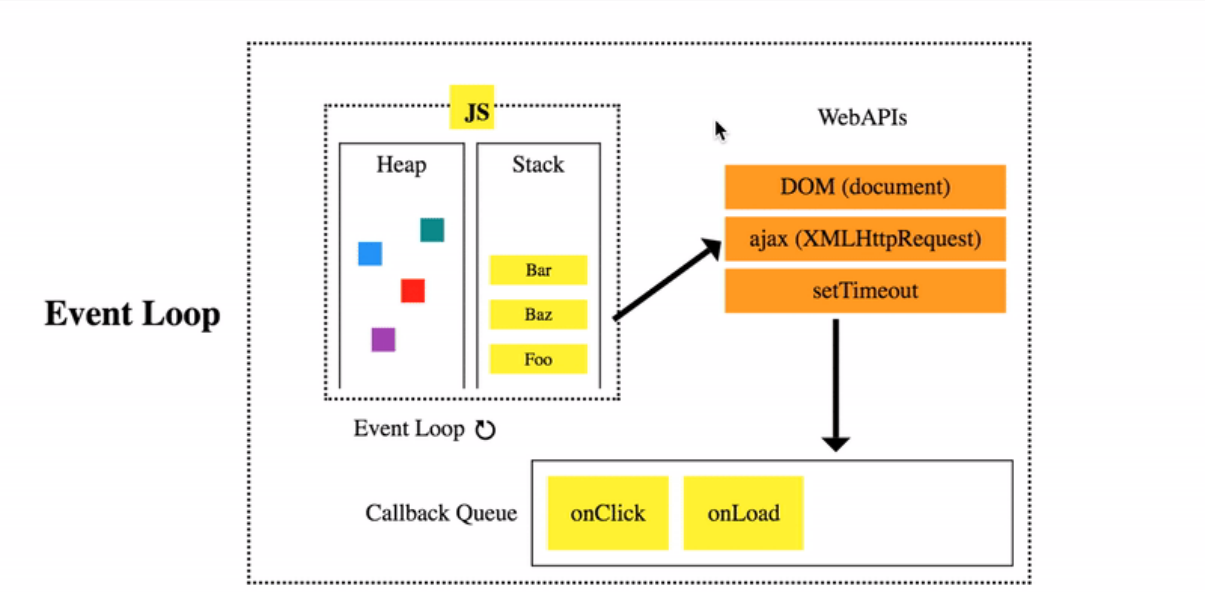
```
💡비동기 요청이란?

Front-end와 Back-end는 별도의 프로그램이며, 다른 컴퓨터 위에서 각각 실행되고 있을 수 있다.

그렇기 때문에 요청과 응답을 주고받는데 오랜 시간이 걸릴 수도 있다.

요청을 보낸 후 Front-end는 다른 작업을 수행하다가 요청에 대한 응답을 받으면 그때 응답을 처리한다.
```

- (참고) 비동기 작업은 Javascript Event Loop를 통해 처리함



- 비동기 작업은 Event Loop의 Callback Queue에 담기며 작업이 완료되는 순서대로 Queue에서 빠져나가 처리됨

XMLHttpRequest

```
// 비동기 통신을 위한 XML Http Requset 객체 생성
const xhr = new XMLHttpRequest();

// 객체 설정값 초기화
xhr.open(HTTP메서드, URL);

// HTTP 헤더 설정
xhr.setRequestHeader("Content-Type", "application/text");

// xhr객체에 ready state change라는 Event가 발생했을때 처리할 Event Handler 등록
xhr.onreadystatechange(fucntion() {

});

// 요청 보내기 (POST 방식의 경우 파라미터 데이터를 인자로 넣어줌)
xhr.send();
```

```
xhr.onreadystatechange(fucntion() {
  if (xhr.readyState === xhr.DONE) { // 응답을 성공적으로 받았다
    if (xhr.status === 200) { // 서버측에서 보낸 응답코드가 200(OK)이다
        ...xhr.responseText...
    }
```

```
  }
});
```

XHR readyState

Value	State	Description
`0`	`UNSENT`	Client has been created. `open()` not called yet.
`1`	`OPENED`	`open()` has been called.
`2`	`HEADERS_RECEIVED`	`send()` has been called, and headers and status are available.
`3`	`LOADING`	Downloading; `responseText` holds partial data.
`4`	`DONE`	The operation is complete

Fetch (ES6 추가)

- 비동기 작업이 가능한 Promise 객체를 반환하는 메소드

```
fetch(URL, 설정값 객체) // 요청을 보냄
.then(() => {
}) //응답을 받았을 때 수행될 콜백함수 정의
```

Promise 객체 (ES6 추가)

- 비동기 작업을 위한 객체
- 성공적인 비동기 응답을 처리할 때는 .then() 메소드 사용
- 비동기 작업 실패를 처리할 때는 .catch() 메소드 사용

```
💡정리하면

Fetch 메소드는 Http 비동기 요청을 처리하는 Promise 객체를 반환해준다

fetch(......).then((response)=>{})

와 같이 HTTP 요청에 대한 응답을 비동기적으로 처리할 수 있게 된다
```

```
fetch("data/car.json", {method: "get"})
.then((response) => {
```

```
  let responseJson = response.json();
  return responseJson;
})
.then((data) => {
  console.log(data);
});
```