4. event-loop 관련 포스팅 읽고 정리하기

JS Engine	· Memory Heap & Call Stack으로 구성 - Memory Heap : 메모리 할당이 일어나는 곳 - Call Stack : 코드가 실행될 때 쌓이는 곳 JS는 단일 스레드(single thread) 프로그래밍 언어로, Call Stack이 하나.
Web API	· JS Engine과 별도 브라우저에서 제공되는 API ex) DOM, AJax, Timeout · Call Stack에서 실행된 비동기 함수는 Web API 호출 → Web API는 콜백함수를 Callback Queue에 삽입
Callback Queue ↳ Queue가 아님?? 실행가능 상태일 때 하나씩 가져가는.. ?	· 비동기적으로 실행된 콜백함수가 보관되는 영역
Event Loop	· Call Stack과 Callback Queue의 상태를 체크하여 Call Stack이 빈 상태가 되면 Callback Queue의 첫번째 콜백을 Call Stack으로 밀어넣는 것. → 틱(tick)
결론	· JS는 단일 스레드 프로그래밍 언어이기 때문에 한번에 하나씩만 실행가능하다. 하지만 Web API, Callback Queue, Event Loop를 통해 멀티 스레드 같이 보이도록 할 수 있다!
주제	

· node.js → 나중에...

Callback Queue 세분화(?)

- 호출되는 Web API에 따라 콜백함수는 우선도가 다른 Queue에 삽입된다.
- Queue의 종류
  - Microtask Queue : microtask가 담기는 Queue
    = Job Queue
    microtask는 비동기 작업이 현재 실행되는 스크립트 바로 다음에 일어나는 작업.
  - Task Queue · task가 담기는 Queue.
    = Event Queue
    예약된 순서를 보장한다는 의미!
    ↗ 꼭 task 다음 task 수행보장은 X
    task는 비동기 작업이 순차적으로 수행될 수 있도록 보장하는 형태의 작업.
  - Animation Frames : requestAnimationFrame API의 콜백이 담기는 Queue인듯하다.
    애니메이션이 있는 경우 60fps정도에 한번씩 랜더링하는데
    이를 구현하기 위해 사용되는 것 같다.
- Queue의 우선순위
  - Microtask Queue > Animation Frames > Task Queue
    모호?
    ↳ 막상 실행해보면 항상 이에 지켜지지 않는 경우도 있다고 한다.
    브라우저마다 실행결과가 상이하게 나올 수 있기 때문!
    아무튼 이론상으로는 그렇다.

Render Queue

브라우저에서 사용자에게 렌더링 이미지를 보여주기 위해 HTML, CSS, Javascript 코드를 변환하는 일을 수행

- DOM Tree → CSS Tree → CSSOM → Render Tree → Layouting
  → Layer Tree → Paint(Skia) → GPU Sync → Composition