Day 03 (2019-06-12)
1. 자바스크립트 동작

자바스크립트 엔진 종류

자바스크립트 코드를 실행하는 인터프리터이다. 대부분이 성능 개선을 위해 JIT 컴파일을 지원한다.

아래는 자바스크립트 엔진의 종류이다.

엔진	설명
라이노	모지라 재단이 운영하며 자바로 개발 됨.
스파이더몽키	최초의 자바스크립트 엔진으로 모질라 파이어폭스에서 사용.
V8	크롬 용 자바스크립트 엔진.
웹킷	사파리 용 자바스크립트 엔진.
차크라	마이크로소프트 엣지 용 자바스크립트 엔진.
Nashorn	OpenJDK의 일부인 오픈소스 자바스크립트 엔진. Oracle Java Languages and Tool Group이 개발.
제리스크립트	IoT를 위한 매우 가벼운 자바스크립트 엔진.

자바스크립트 엔진은 내부적으로 여러 스레드를 사용

- Main thread를 통해 코드를 실행
- 최적화 코드를 위한 JIT 컴파일러(크랭크 샤프트) 실행
- 가비지 컬렉션 및 메모리 스윕

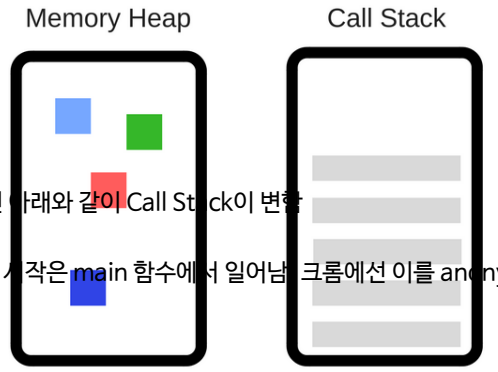
하지만 **자바스크립트 언어**는 단일 스레드로 동작함
→ 하나의 호출 스택만을 가지기 때문

자바스크립트 엔진의 구성

자바스크립트 호출 스택 (Javascript Call Stack)

호출 스택이 하나라는 것은 한 번에 한 작업만 처리 된다는 것
호출 스택에는 현재 프로그램이 어느 부분을 실행하는 지에 대해 기록 함.

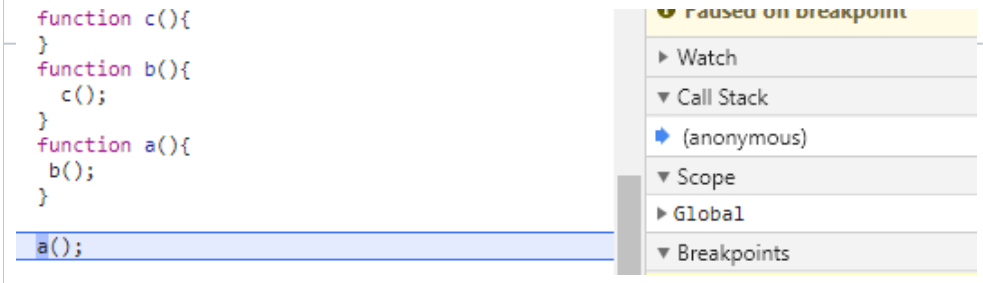
```
function c(){

}
function b(){
  c();
}
function a(){
  b();
}

a();
```

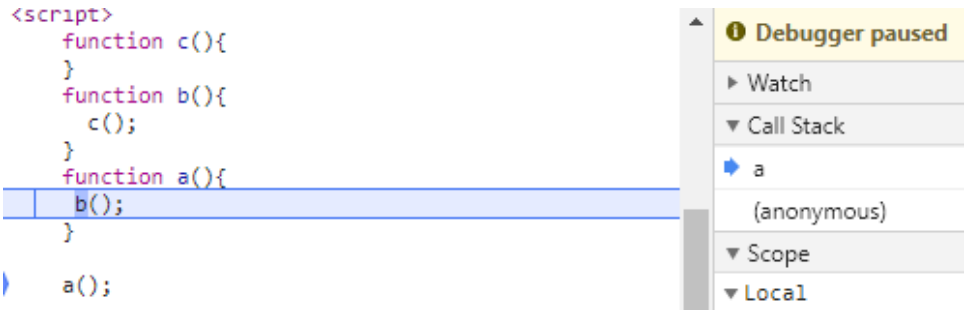


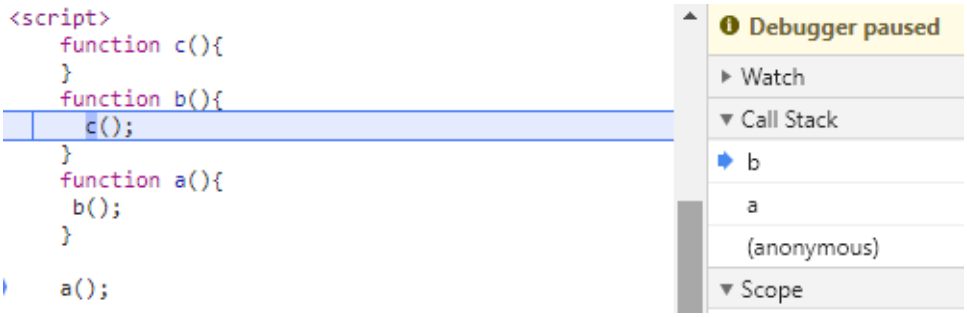
위의 예제 코드를 실행하면 아래와 같이 Call Stack이 변함

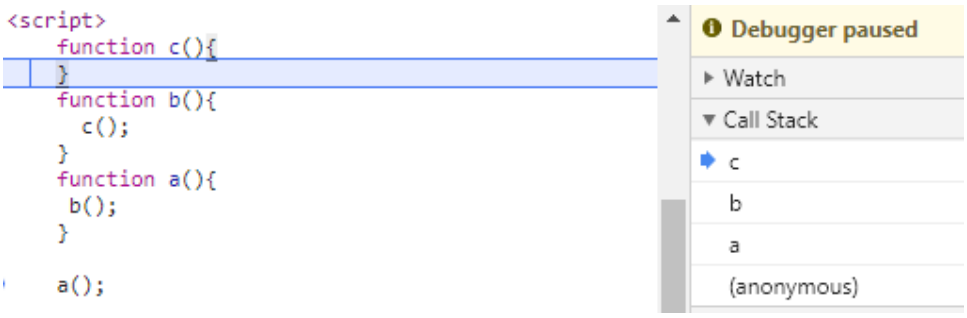
※ 자바스크립트에서 항상 시작은 main 함수에서 일어남. 크롬에선 이를 anonymous로 표시함

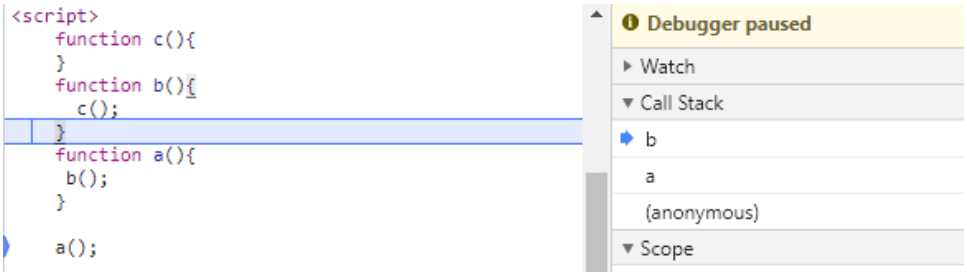
Debugger

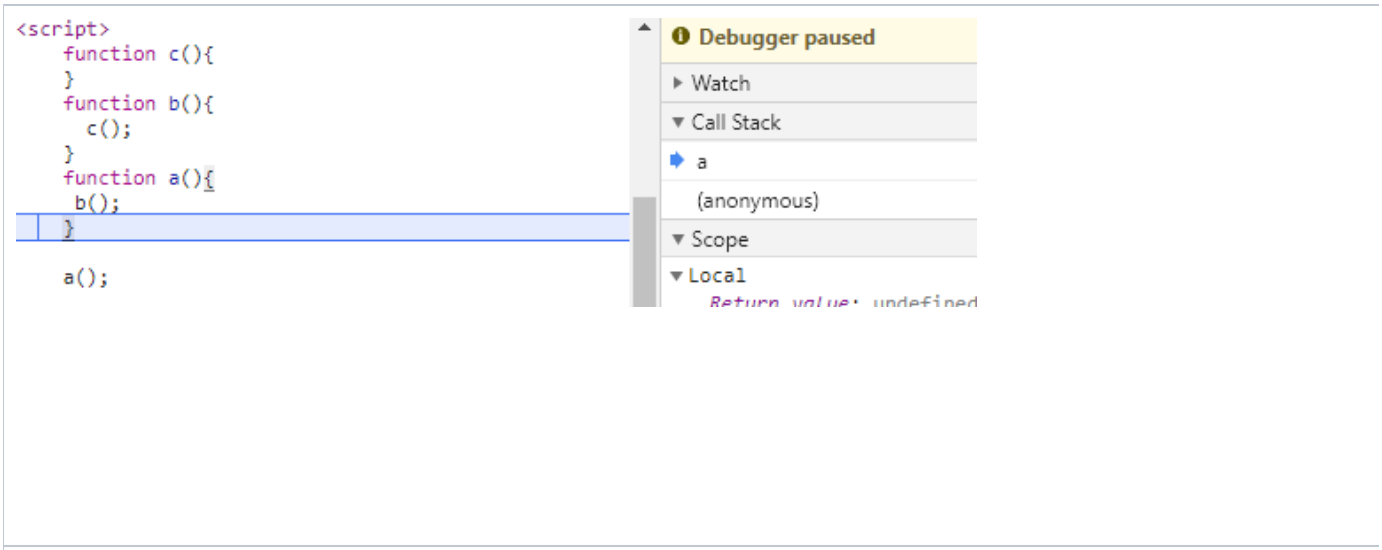


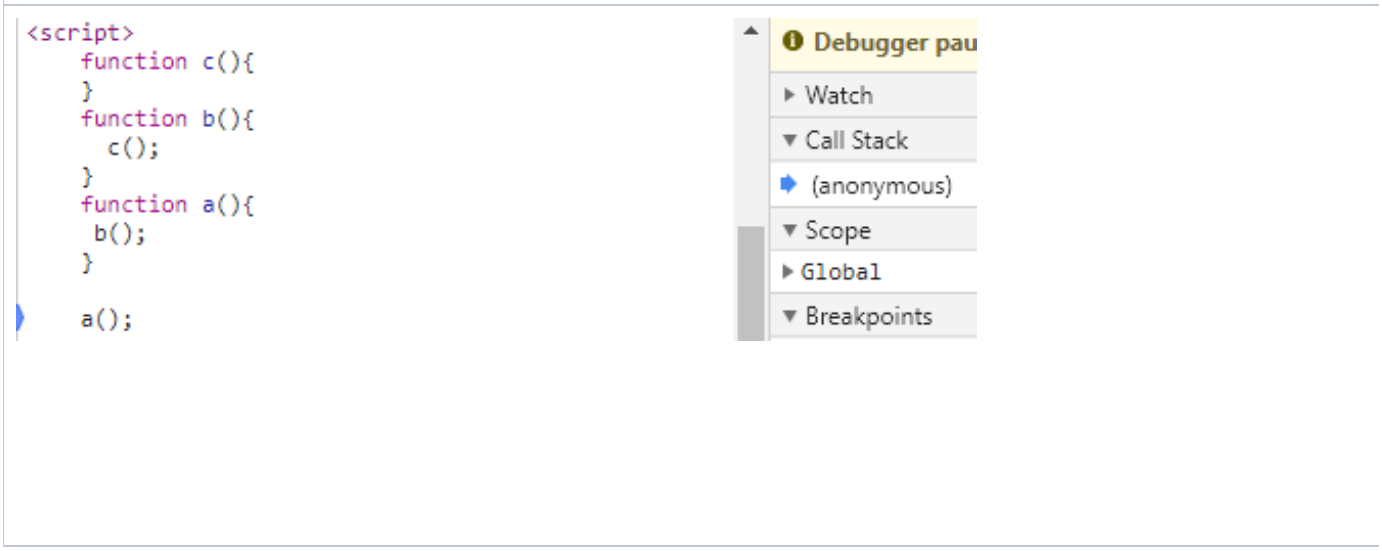



<script>
function c(){
}
function b(){
c();
}
function a(){
b();
}
a();
Debugger paused
Watch
Call Stack
b
a
(anonymous)
Scope


<script>
function c(){
}
function b(){
c();
}
function a(){
b();
}
a();
Debugger paused
Watch
Call Stack
c
b
a
(anonymous)

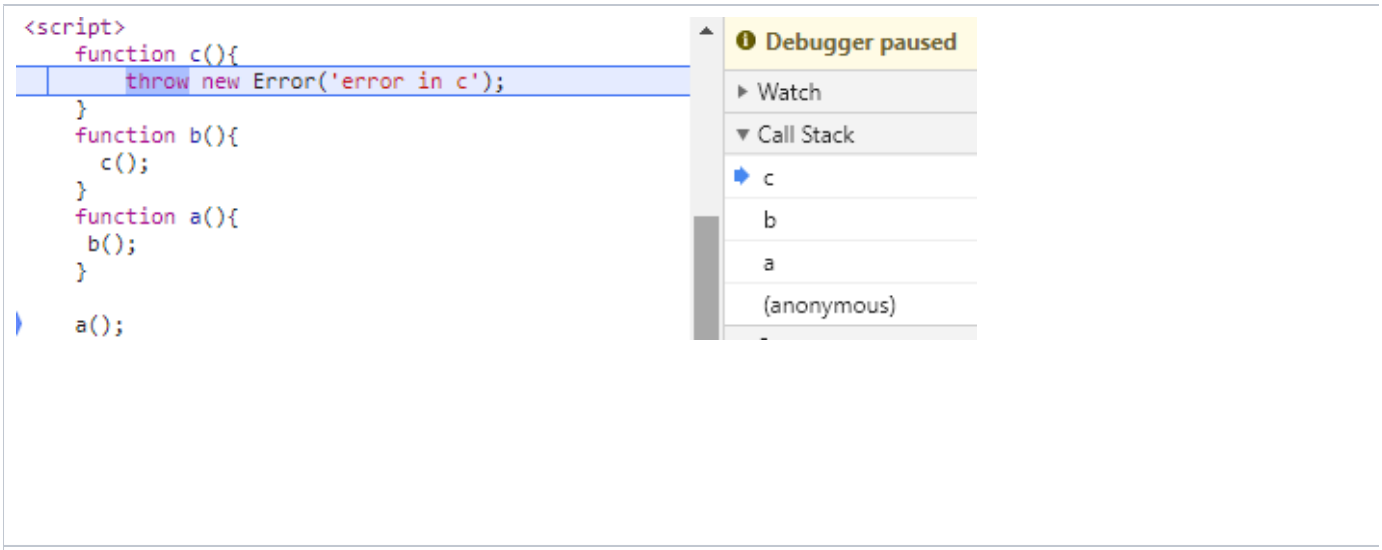

<script>
function c(){
}
function b(){
c();
}
function a(){
b();
}
a();
Debugger paused
Watch
Call Stack
b
a
(anonymous)
Scope

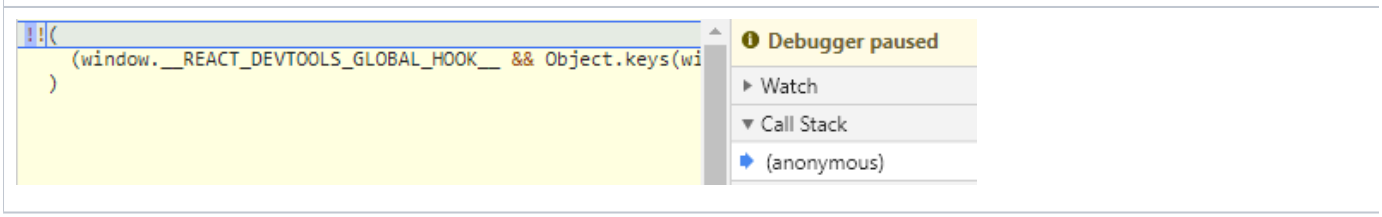




만약 C를 실행하는 도중 예외가 발생한다면?

Debugger



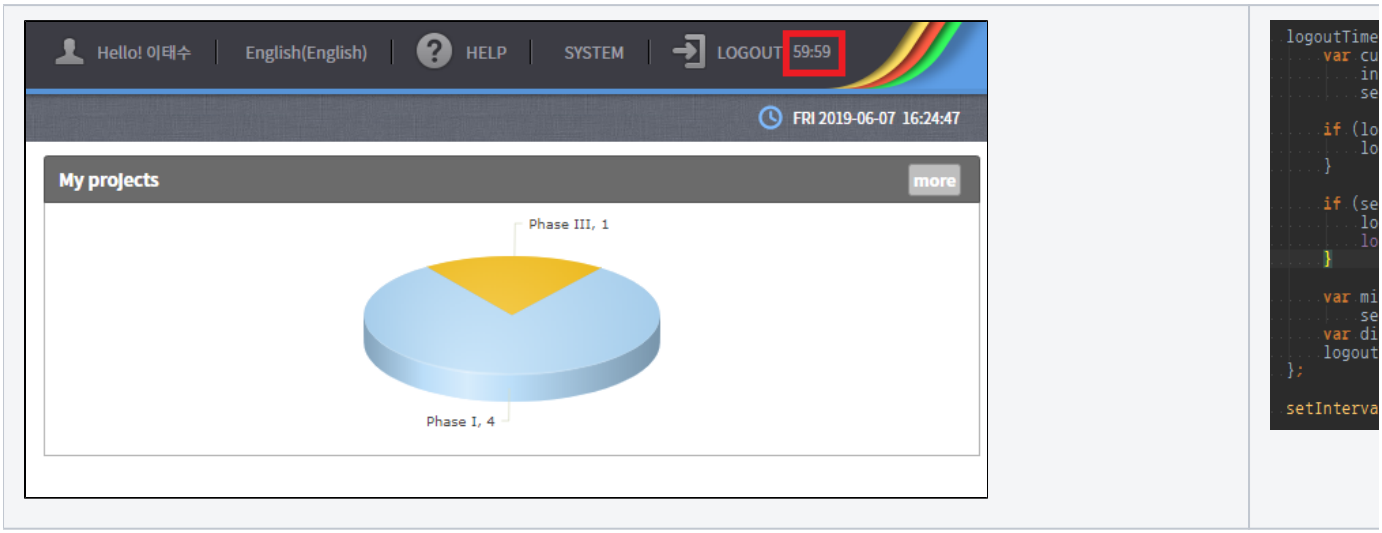


비동기에 대한 처리

앞선 호출 스택에 대한 설명은 동기적으로 처리되는 로직에 대한 호출 스택

자바스크립트 기반 어플리케이션에서 필연적으로 사용되는 비동기 로직은 어떻게 처리될까?

예를들어 세션 타임아웃을 알리기 위한 타이머는 매 초마다 갱신되어야 한다.



위의 타이머를 갱신하기 위해 logoutTimer.timeoutHandler가 1000ms를 간격으로 호출된다.

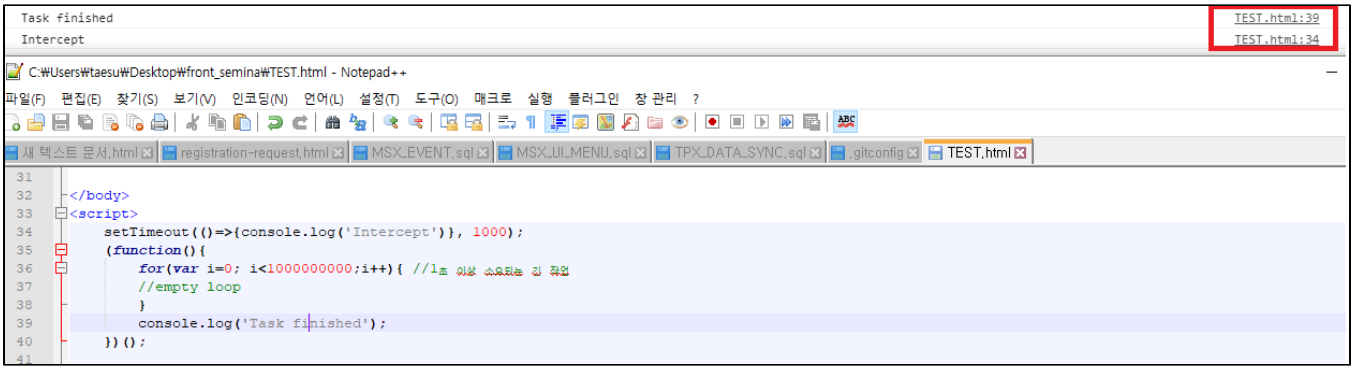
그렇다면 1000ms마다 logoutTimer.timeoutHandler 함수는 현재 호출 스택에 상관 없이 올라가는 것인가?

아래의 에제 코드를 확인해보자

```
setTimeout(()=>{console.log('Intercept')}, 1000);
(function(){
        for(var i=0; i<1000000000;i++){ //1
        //empty loop
        }
        console.log('Task finished');
})();
```

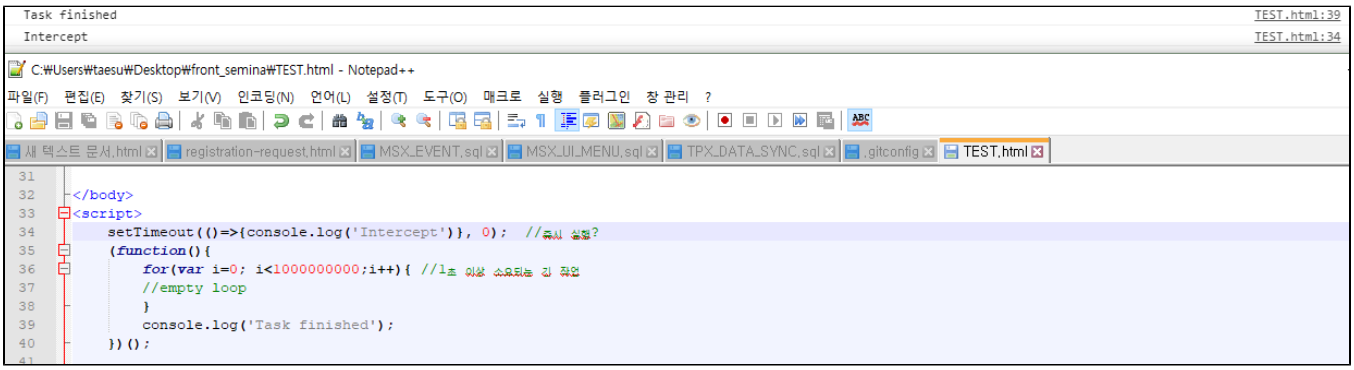
setTimeout callback에서 출력하는 'Intercept' 메시지는 대부분 'Task finished' 메시지보다 늦게 출력된다.

분명 개발자는 1초 이상 소요되는 긴 작업이 끝나기 전에 'Intercept' 메시지가 출력 되도록 의도했을 것인데 말이다.



혹시나 1초 이상 소요되는 작업이 사실은 1초보다 짧은 것 아니냐느 의심을 가질 수 있기에 아래와 같이도 실행해본다.

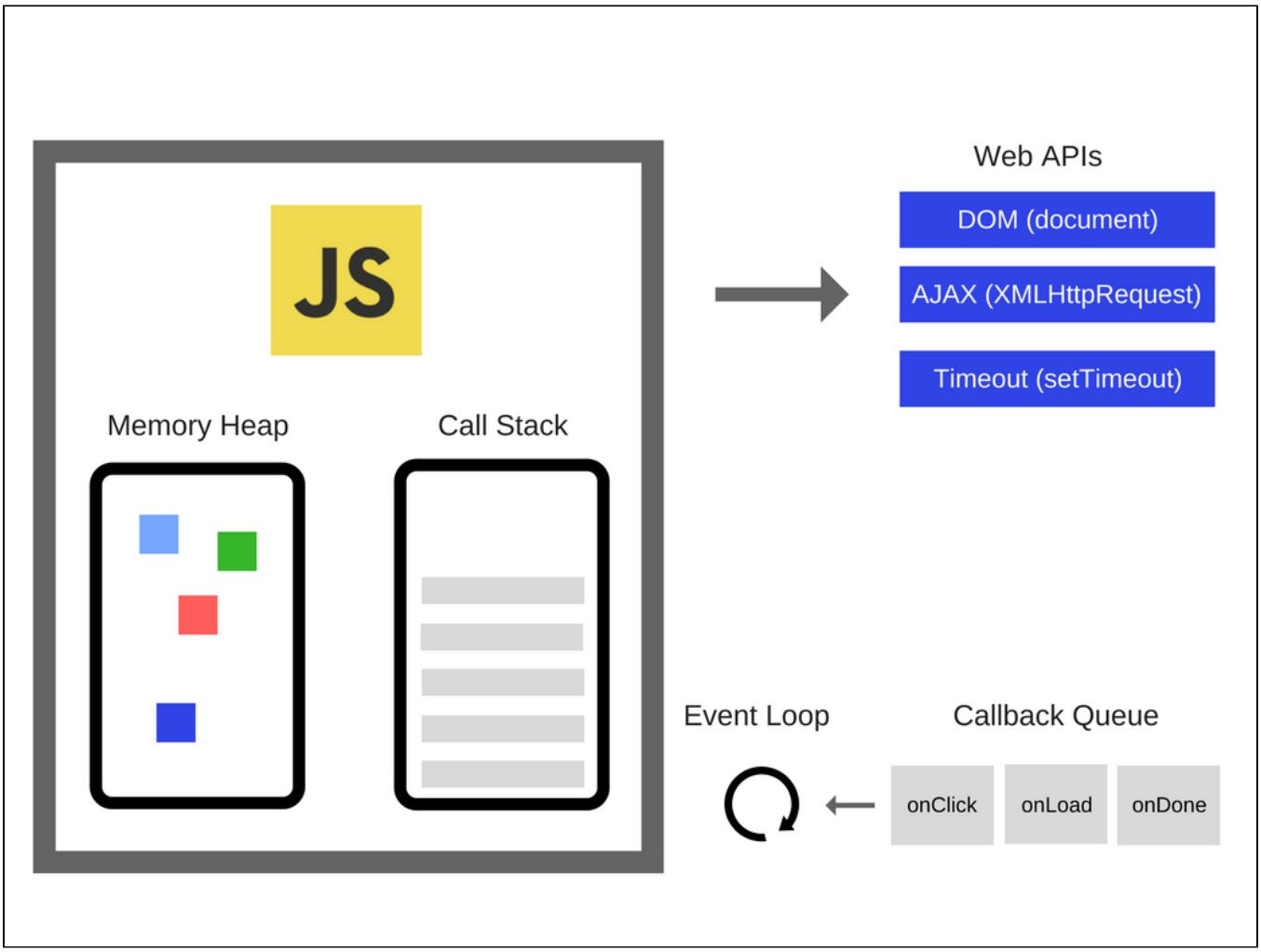
```
setTimeout(()=>{console.log('Intercept')}, 0); //
(function(){
        for(var i=0; i<1000000000;i++){ //1
        //empty loop
        }
        console.log('Task finished');
})();
```



자바스크립트 런타임과 이벤트 루프

자바스크립트에선 DOM Events(사용자 마우스 클릭, 키보드 입력 등), Ajax 요청과 같은 동시 다발적으로 일어나는 작업의 동시성 처리를 위해

이벤트 루프를 사용한다. 아래는 자바스크립트 엔진을 포함하는 자바스크립트 런타임(자바스크립트 실행환경)의 모습이다.



Event Queue라고도 불리는 Callback Queue는 특정 이벤트가 발생 했을 때 수행해야 할 콜백 함수를 대기시키는 곳이다.

예를들어 사용자가 마우스 클릭 이벤트를 발생시키면 Callback Queue에 onClick이 쌓이고
setTimeout은 파라미터로 지정된 시간이 지나면 콜백 메소드가 Callback Queue에 쌓인다. **(setTimeout에 지정된 시간이 경과했는지는 누가 할까?)**

Event Loop는 **호출 스택이 빌 때마다** Callback Queue에 있는 Callback 들을 하나씩 꺼내어 실행하는데 아래의 과정을 거친다.

- 이벤트 큐에 작업이 있는가?
- 큐에 있는 작업을 호출 스택에 쌓는다.

예제 코드 동작 해석 및 중점 정리

(1) 동작 해석

이제 조금 전 예제 코드를 해석 할 수 있다.

1. setTimeout이 호출되는 순간 이것은 Web API이므로 해당 담당자에게 위임된다
2. 그후 1초 이상 소용되는 긴 작업이 바로 시작에 들어간다
   → 호출 스택에 해당 function이 적재된다
3. Web API 담당자가 시간을 재어보니 0초이므로 바로 Callback Queue에 쌓인다.
4. 1초 이상 소요되는 긴 작업이 끝난다 (Task finished 메시지 출력)
5. Callback Queue에 있는 작업이 호출 스택에 쌓인다 (Intercept 메시지 출력)

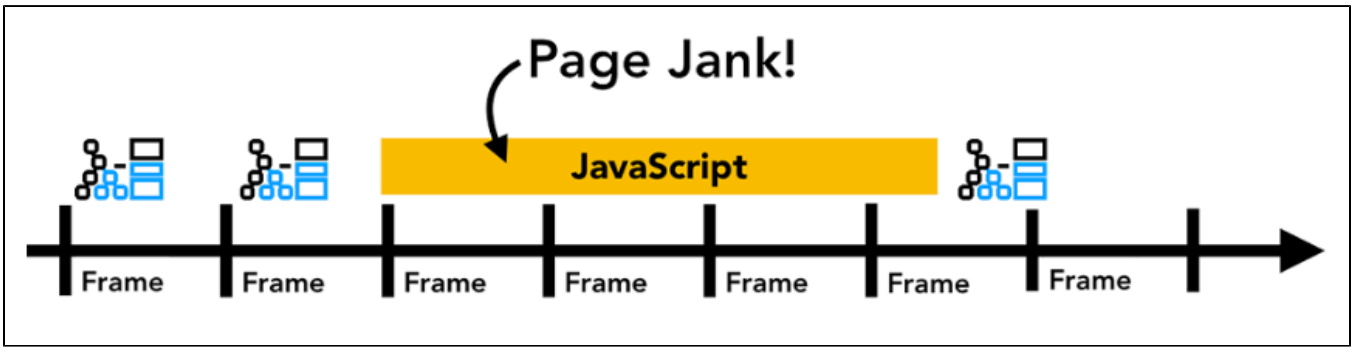
```
setTimeout(()=>{console.log('Intercept')}, 0); //
(function(){
        for(var i=0; i<1000000000;i++){ //1
        //empty loop
        }
        console.log('Task finished');
})();
```

(2) 중점 정리

결과적으로 자바스크립트의 동작 특성으로 인해 긴 시간이 소요되는 작업을 직접적으로 처리하지 않아야 한다.

위의 예제 코드에선 for문의 body에서 아무런 일도 하지 않아 1000000000번의 루프가 긴 작업으로 처리되었지만
루프를 돌면서 정규표현식을 통해 매칭하고 replace 하는 등의 무거운 작업들이라고 가정하면 데이터가 1000건 정도이어도
심각한 문제를 초래할 수 있다.

특히나 렌더러 프로세스가 화면 주사율에 맞추어 렌더링 작업을 수행하고 있는 상태에서 애플리케이션이 긴 시간 Javascript를
실행하면 Page Jank (버벅 거리거나 화면이 잠시 멈췄다가 어느순간 딱! 하고 나타나는 현상)를 야기할 수 있다.



다음에선 Page jank의 발생 원인 재현과 해결 방안을 정리한다.

참고자료

https://blog.sessionstack.com/how-javascript-works-inside-the-v8-engine-5-tips-on-how-to-write-optimized-code-ac089e62b12e

http://blog.naver.com/PostView.nhn?blogId=tmondev&logNo=220937034820&categoryNo=0&parentCategoryNo=1&viewDate=&currentPage=1&postListTopCurrentPage=1&from=postView

https://blog.sessionstack.com/how-javascript-works-inside-the-v8-engine-5-tips-on-how-to-write-optimized-code-ac089e62b12e