모던 자바스크립트
Deep Dive

42장_비동기 프로그래밍

비동기 프로그래밍

1. 함수 실행과 실행 컨텍스트
함수 호출 → 함수 평가 → 실행 컨텍스트 생성 → 실행 컨텍스트 스택 푸시 → 함수 실행 → 실행 컨텍스트 스택 팝

2. 자바스크립트 엔진은 싱글 스레드
- 자바스크립트 엔진은 하나의 실행 컨텍스트 스택(싱글 스레드)을 가짐
- 함수를 동시에 실행할 수 없음
- 실행 컨텍스트 스택에서 최상위 실행 컨텍스트를 제외한 실행 컨텍스트들은 '실행 대기 중인 태스크'
  → 처리에 시간이 걸리는 태스크를 실행하면, 블로킹(작업 중단)이 발생

※ 싱글 스레드의 장, 단점
- 멀티 스레드 환경에서 나오는 복잡한 시나리오를 고려하지 않아도 됨
- 블로킹

3. 동기 처리
- 현재 실행 중인 태스크가 종료될 때까지 다음에 실행될 태스크는 대기하는 방식
- 코드 실행 순서 보장되는 장점, 블로킹이 발생하는 단점

4. 비동기 처리
- 현재 실행 중인 태스크가 종료되지 않았어도 다음에 실행될 태스크는 바로 실행하는 방식
- 블로킹이 발생하지 않는다는 장점, 코드 실행 순서가 보장되지 않는다는 단점

비동기 프로그래밍

콜백 패턴

- 비동기 함수는 일반적으로 콜백 패턴 사용

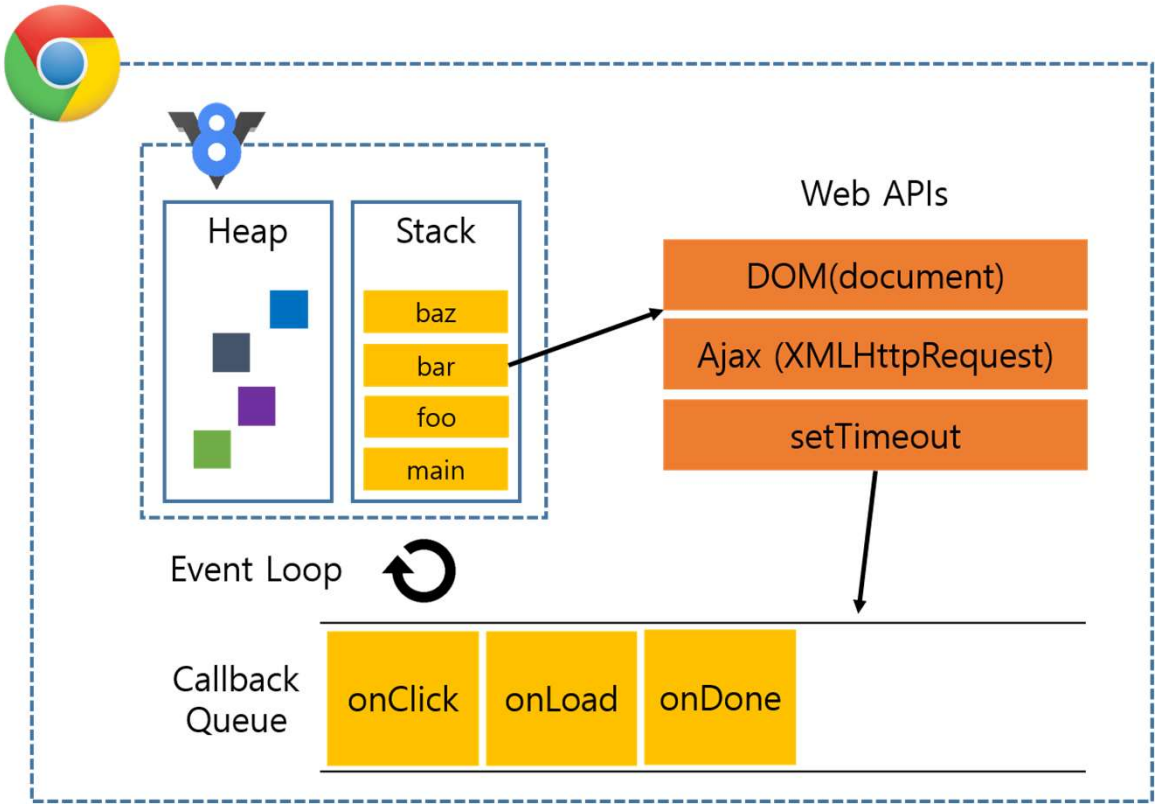
- 콜백 패턴은
  (1) 콜백 헬 발생하여 가독성 ↓
  (2) 비동기 처리 중 발생한 에러의 예외 처리 어려움
  (3) 여러 개의 비동기 처리를 한 번에 처리 불가

- 타이머 함수인 setTimeout, setInterval, HTTP 요청, 이벤트 헨들러는 비동기 처리 방식으로 동작

이벤트 루프와 태스크 큐

자바스크립트는 싱글 스레드로 동작. 하지만 브라우저가 동작하는 것을 보면, 많은 태스크가 동시에 처리되는 것처럼 보임

자바스크립트는 브라우저에 내장되어 있는 **이벤트 루프**를 통해서 동시성 지원



자바스크립트 엔진의 영역

자바스크립트 엔진은 크게 두 가지 영역으로 구분 가능

1. 콜 스택
- 실행 컨텍스트 스택
- 최상위 실행 컨텍스트를 제외한 실행 컨텍스트들은 '실행 대기 중인 태스크'

2. 힙
- 객체가 저장되는 메모리 공간
- 객체는 크기가 정해져 있지 않음.
- 할당해야 할 메모리 공간의 크기를 런타임에 결정(동적 할당)하므로 힙은 구조화 되어 있지 않음

자바스크립트 엔진의 역할과 자바스크립트 엔진 구동 환경의 역할
비동기 처리에서 소스코드의 평가와 실행을 제외한 모든 처리는 자바스크립트 엔진 구동 환경에서 처리
예를 들면, 비동기 방식으로 동작하는 setTimeout 콜백 함수의 평가와 실행은 자바스크립트 엔진 담당
호출 스케줄링을 위한 타이머 설정과 콜백 함수의 등록은 브라우저 또는 Node.js 담당
이를 위해서 브라우저는 **태스크 큐**와 **이벤트 루프**를 제공

태스크 큐와 이벤트 루프

1. 태스크 큐
- 콜백 함수 또는 이벤트 핸들러가 일시적으로 보관되는 영역
- 프로미스의 후속 처리 메서드 내의 콜백 함수가 일시적으로 보관되는 **Microtask Queue**도 존재

브라우저에 존재하는 다양한 Queue
브라우저에는 다양한 큐들이 존재
- Macrotask Queue는 비동기 함수(setTimeout, setInterval, HTTP 요청, 이벤트 핸들러)의 콜백 함수 보관
- Microtask Queue는 프로미스의 후속 처리 메서드의 콜백 함수 보관
- Animation Frame은 rAF의 콜백 함수 보관
- 이벤트 루프가 어떤 우선순위로 각 Queue의 Task를 가져오는가?
  Micro → Animation → Macro

※ 코드 참고

2. 이벤트 루프
- 콜 스택이 비어있는지, 태스크 큐에 대기중인 함수가 있는지 반복해서 확인
- 콜 스택이 비어있고, 태스크 큐에 대기중인 함수가 있다면, FIFO에 의해 대기 중인 함수를 콜 스택으로 이동 및 실행

예제

```
function foo() {
  console.log('foo');
}

function bar() {
  console.log('bar');
}

setTimeout(foo, 0);
bar();
```

1. 전역 코드 평가 → 전역 실행 컨텍스트 생성 → 콜 스택 푸시

2. 전역 코드 실행 → setTimeout 함수 호출 → setTimeout 함수 실행 컨스트 생성 → 콜 스택 푸시
브라우저 Web API인 타이머 함수도 함수이므로, 함수 실행 컨텍스트 생성

3. setTimeout 함수 실행 → 콜백 함수 호출 스케줄링(타이머 설정 및 타이머 만료) → 콜 스택 팝

4. 4-1은 **브라우저**에 의해, 4-2는 **자바스크립트 엔진**에 의해 병렬 처리

4-1. 타이머 설정 및 타이머 만료 기다림 → 타이머가 만료되면 콜백 함수 태스크 큐에 푸시. 위 예제의 경우, 지연 시간이 0ms 이지만 실제로는 4ms 가량 존재. 즉 4ms 이후 foo가 태스크 큐에 푸시되어 대기.

※ setTimeout의 지연 시간과 관련해서
setTimeout 함수의 콜백 함수가 정확히 지연 시간 후에 호출된다는 보장은 없다. **지연 시간은 콜백 함수가 태스크 큐에 푸시되기까지 기다려야 하는 시간**이다. 이후 콜 스택이 비어야 호출되므로, 약간의 시간차가 발생할 수 있다.

4-2. bar 함수 호출 및 평가 → bar 실행 컨텍스트 생성 → 콜 스택 푸시 → bar 함수 실행 → 콜 스택 팝

예제

```
function foo() {
  console.log('foo');
}

function bar() {
  console.log('bar');
}

setTimeout(foo, 0);
bar();
```

5. 전역 코드 실행 종료 → 전역 실행 컨텍스트 팝. 콜 스택은 비어있게 됨

6. 태스크 큐에서 대기 중인 콜백 함수 foo가 이벤트 루프에 의해 콜 스택에 푸시 및 실행

정리
- 브라우저가 아닌 자바스크립트 엔진은 싱글 스레드 방식으로 동작
- 브라우저는 멀티 스레드로 동작
- 비동기 함수의 콜백 함수는 태스크 큐에서 대기
- 콜 스택이 비게 되면, 즉 전역 코드 및 명시적으로 실행한 함수가 모두 종료되면 콜 스택에 푸시 및 실행
- 브라우저는 (1) 타이머 설정, (2) 타이머 만료 후 태스크 큐 푸시, (3) 콜 스택 비어있는지 확인 후 태스크 이동
- 브라우저와 자바스크립트 엔진이 협력하여 setTimeout 함수를 실행한다.

참고

https://dev.to/lydiahallie/javascript-visualized-promises-async-await-5gke#syntax