비동기 처리? 그게 뭔가요?

자바스크립트의 비동기 처리란 특정 코드의 연산이 끝날 때까지 코드의 실행을 멈추지 않고 다음 코드를 먼저 실행하는 자바스크립트의 특성을 의미합니다.

비동기 처리의 첫 번째 사례

asynchronous javascript XML

비동기 처리의 가장 흔한 사례는 제이쿼리의 ajax입니다. 제이쿼리로 실제 웹 서비스를 개발할 때 ajax 통신을 빼놓을 수가 없습니다. 보통 화면에 표시할 이미지나 데이터를 서버에서 불러와 표시해야 하는데 이때 ajax 통신으로 해당 데이터를 서버로부터 가져올 수 있기 때문입니다.

그럼 ajax 코드를 잠깐 살펴보겠습니다.

```
function getData() {
    var tableData;
    $.get('https://domain.com/products/1', function(response) {
        tableData = response;
    });
    return tableData;
}

console.log(getData()); // undefined
```

여기서 `$.get()`이 ajax 통신을 하는 부분입니다. `https://domain.com` 에다가 HTTP GET 요청을 날려 1번 상품(product) 정보를 요청하는 코드죠. 좀 더 쉽게 말하면 지정된 URL에 '데이터를 하나 보내주세요' 라는 요청을 날리는 것과 같습니다.

그렇게 서버에서 받아온 데이터는 `response` 인자에 담깁니다. 그리고 `tableData = response;` 코드로 받아온 데이터를 `tableData`라는 변수에 저장합니다. 그럼 이제 이 getData()를 호출하면 어떻게 될까요? 받아온 데이터가 뭐든 일단 뭔가 찍혀야겠죠. 근데 결과는 맨 아래에서 보시는 것처럼 undefined입니다. 왜 그럴까요?

그 이유는 `$.get()`로 데이터를 요청하고 받아올 때까지 기다려주지 않고 다음 코드인 `return tableData;`를 실행했기 때문입니다. 따라서, getData()의 결과 값은 초기 값을 설정하지 않은 tableData의 값 undefined를 출력합니다.

이렇게 특정 로직의 실행이 끝날 때까지 기다려주지 않고 나머지 코드를 먼저 실행하는 것이 비동기 처리입니다. 자바스크립트에서 비동기 처리가 필요한 이유를 생각해보면, 화면에서 서버로 데이터를 요청했을 때 서버가 언제 그 요청에 대한 응답을 줄지도 모르는데 마냥 다른 코드를 실행 안 하고 기다릴 순 없기 때문입니다. 위에선 간단한 요청 1개만 보냈는데 만약 100개 보낸다고 생각해보세요. 비동기 처리가 아니고 동기 처리라면 코드 실행하고 기다리고, 실행하고 기다리고.. 아마 웹 애플리케이션을 실행하는데 수십 분은 걸릴 겁니다.

비동기 처리의 두 번째 사례

또 다른 비동기 처리 사례는 setTimeout()입니다. setTimeout()은 Web API의 한 종류입니다. 코드를 바로 실행하지 않고 지정한 시간만큼 기다렸다가 로직을 실행합니다. 아래 코드를 보겠습니다.

```
// #1
console.log('Hello');
// #2
setTimeout(function() {
    console.log('Bye');
}, 3000);
// #3
console.log('Hello Again');
```

비동기 처리에 대한 이해가 없는 상태에서 위 코드를 보면 아마 다음과 같은 결과값이 나올 거라고 생각할 겁니다.

- ‘Hello’ 출력
- 3초 있다가 ‘Bye’ 출력
- ‘Hello Again’ 출력

그런데 실제 결과 값은 아래와 같이 나오죠.

- ‘Hello’ 출력
- ‘Hello Again’ 출력
- 3초 있다가 ‘Bye’ 출력

setTimeout() 역시 비동기 방식으로 실행되기 때문에 3초를 기다렸다가 다음 코드를 수행하는 것이 아니라 일단 setTimeout()을 실행하고 나서 바로 다음 코드인 `console.log('Hello Again');`으로 넘어갔습니다. 따라서, ‘Hello’, ‘Hello Again’를 먼저 출력하고 3초가 지나면 ‘Bye’가 출력됩니다.

콜백 함수로 비동기 처리 방식의 문제점 해결하기

앞에서 자바스크립트 비동기 처리 방식에 의해 야기될 수 있는 문제들을 살펴봤습니다. 이러한 문제들은 어떻게 해결할 수 있을까요? 바로 콜백(callback) 함수를 이용하는 것입니다. 앞에서 살펴본 ajax 통신 코드를 콜백 함수로 개선해보겠습니다.

```Js Copy
function getData(callbackFunc) {
    $.get('https://domain.com/products/1', function(response) {
        callbackFunc(response); // 서버에서 받은 데이터 response를 callbackFunc() 함수에 넘겨
    });
}

getData(function(tableData) {
    console.log(tableData); // $.get()의 response 값이 tableData에 전달됨
});
```

이렇게 콜백 함수를 사용하면 특정 로직이 끝났을 때 원하는 동작을 실행시킬 수 있습니다.

Node.js: 비동기 프로그래밍 이해

6 JAN 2021

대부분의 기업형 애플리케이션은 중앙의 서버에서 동작합니다. 이러한 서버는 Web을 위한 HTTP 서버 또는 소켓 통신을 위한 네트워크 서버 등이 있습니다. 서버는 중앙집중형태로 클라이언트의 요청을 받으므로 병목현상이 발생하기 쉬우며 처리 성능에 항상 주목해야 합니다.

클라이언트의 요청이 많은 경우 서버는 병목 구간이 발생합니다. 이러한 병목구간을 분석해 보면 대부분 프로그램 로직보다는 입출력(IO)에서 발생합니다. IO에서 소요되는 비용은 생각보다 많이 나옵니다. 아래의 통계자료를 보면 주로 Disk 나 Network Access 시 비용이 가장 많이 나오는 것을 확인할 수 있습니다.

IO 종류	비 용
L1-cache	3 cycles
L2-cache	14 cycles
RAM	250 cycles
Disk	41,000,000 cycles
Network	240,000,000 cycles

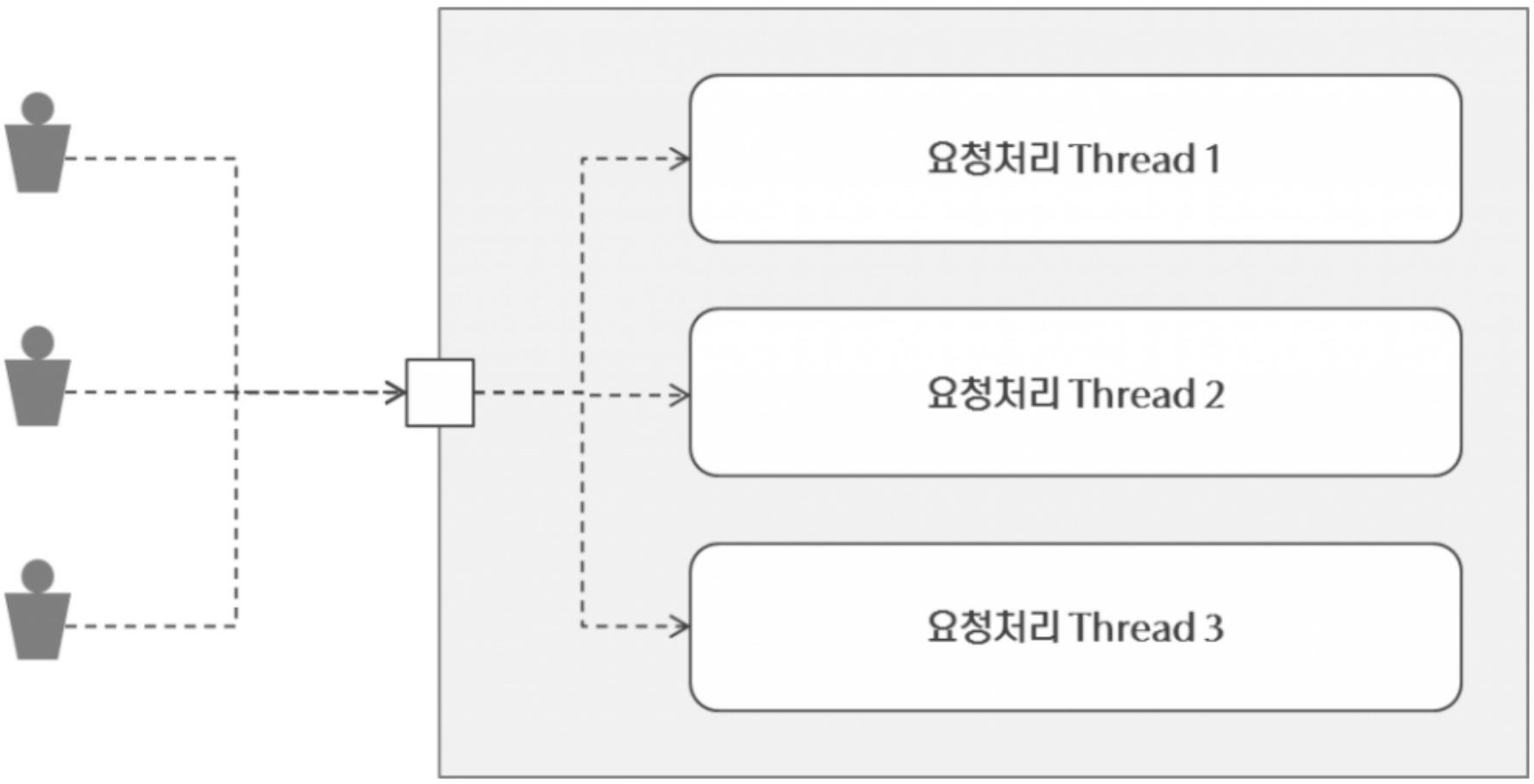
IO의 비용

이와같이 서버에서 IO를 처리하다가 지연이 발생하면 다른 요청들은 처리되지 못하고 계속 대기하는 현상이 발생합니다. 그래서 대부분의 기업형 서버 플랫폼들은 이 문제를 해결하기 위하여 사용자의 요청을 쓰레드로 처리하고 있습니다.

Multi-Thread 의 한계

Multi-Thread 방식은 서버의 요청 처리를 쓰레드에서 처리하도록 하여 병렬처리를 가능하도록 하는 방식입니다. 쓰레드는 서버 CPU 자원을 시분할 형태로 나누어 가짐으로써 독립실행이 가능하며 다른 요청을 동시에 받을 수 있게 합니다.

이러한 방식은 IO Blocking을 해결하는 이상적인 모델로 보일 수 있습니다. 그러나 몇가지 한계도 존재합니다.



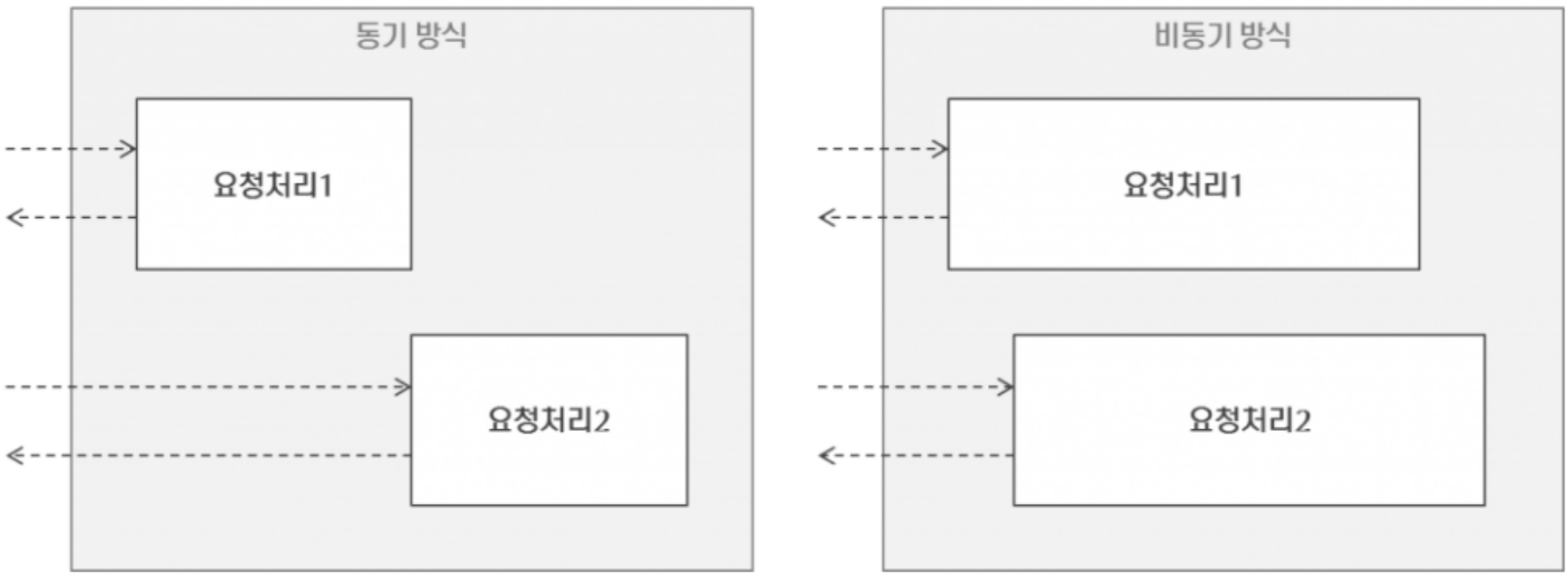
Multi-Thread 방식

Multi-Thread 기반의 서버는 일반적으로 클라이언트의 요청마다 Thread를 발생시킵니다. 이 말은 동시 접속자 수가 많을 수록 Thread가 많이 발생한다는 의미이며 그만큼 메모리 자원도 많이 소모한다는 의미입니다. 그러나 서버의 자원은 제한되어 있으므로 일정 수 이상의 Thread는 발생시킬 수 없습니다.

Multi-Thread 는 이런 근본적인 문제를 안고 있기 때문에 현장에서는 서버를 업그레이드 하거나 Load-Balancing 등으로 분산처리 하는 것입니다.

한편 Multi-Thread 방식은 각 Thread 간의 공유자원 접근시 신중해야 합니다. 각 Thread는 독립적인 시점에서 동작하므로 공유자원에 대한 동기화 없이 접근하면 예기치 않은 결과가 나올 수 있습니다.

병렬처리의 대안: 비동기 처리



동기 방식과 비동기 방식

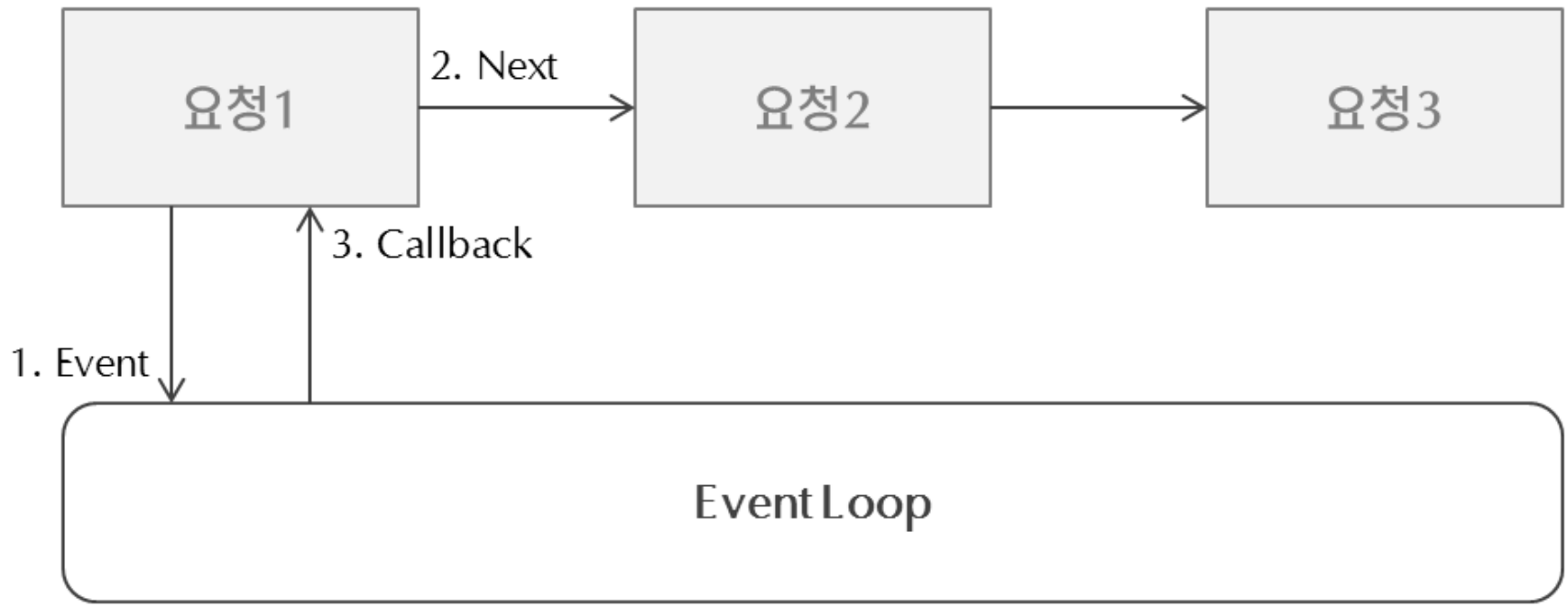
동기방식의 처리는 하나의 요청이 처리되는 동안 다른 요청이 처리되지 못하며 요청이 완료되어야만 다음 처리가 가능한 방식입니다. 동기 방식은 IO 처리를 Blocking 하는데 지금까지는 이 문제를 Thread로 처리했습니다.

이 문제를 비동기 방식으로 처리할 수도 있습니다. 비동기 방식은 하나의 요청 처리가 완료되기 전에 제어권을 다음 요청으로 넘깁니다. 따라서 IO 처리인 경우 Blocking 되지 않으며 다음 요청을 처리할 수 있는 것입니다.

Node.js 는 바로 이와같은 방식으로 병렬처리를 합니다.

Node.js 의 비동기 처리

Node.js 는 비동기 IO를 지원하며 Single-Thread 기반으로 동작하는 서버입니다. Node 서버는 비동기 방식으로 요청을 처리하므로 요청을 처리하면서 다음 요청을 받을 수 있습니다. 또한 병렬처리를 Thread 로 처리하지 않으므로 Multi-Thread가 갖는 근원적인 문제에서 자유롭습니다.



Node.js의 Event Loop

Node.js 의 비동기 처리는 이벤트 방식으로 풀어냅니다. 클라이언트의 요청을 비동기로 처리하기 위하여 이벤트가 발생하며 서버 내부에 메시지 형태로 전달됩니다. 서버 내부에서는 이 메시지를 Event Loop가 처리합니다. Event Loop가 처리하는 동안 제어권은 다음 요청으로 넘어가고 처리가 완료되면 Callback을 호출하여 처리완료를 호출측에 알려줍니다.

Event Loop는 요청을 처리하기 위하여 내부적으로 약간의 Thread와 프로세스를 사용합니다. 이는 Non-Blocking IO 또는 내부 처리를 위한 목적으로만 사용되지 요청 처리 자체를 Thread로 하지는 않습니다. 따라서 Node 서버는 Multi-Thread 방식의 서버에 비하여 Thread 수와 오버헤드가 훨씬 적습니다.

이벤트를 처리하는 Event Loop는 Single-Thread 로 이루어져 있습니다. 즉 요청 처리는 하나의 Thread 안에서 처리된다는 의미입니다. 그래서 이벤트 호출 측에는 비동기로 처리되지만 처리작업 자체가 오래 걸린다면 전체 서버 처리에 영향을 줍니다. 이는 Node.js 의 치명적인 약점이라고 볼 수 있습니다.