2020.11.01. Self\_Seminar 2차

발표자 정대용

서버와 서버 언어

1. 서버의 개념과 작동 방식

1.1. 서버란 무엇인가?

- 네트워크를 통해 서비스를 제공하는 컴퓨터 혹은 장치를 서버라 한다.

- 서버로부터 서비스를 제공받는 컴퓨터를 클라이언트라 한다.

- 서버-클라이언트 시스템에서 가장 기본적인 작동 방식은, 클라이언트는 서버에게 요청(request)를 보내고, 서버는 클라이언트에게 응답(response)을 되돌려준다는 것이다.

- 서버는 상대적인 개념일 수 있다. ex) 서비스를 제공하는 맛집 어플이 지도 정보가 필요할 땐 클라이언트가 되어 네이버 지도로부터 서비스를 제공받는다.

- 실질적인 예시) 월드와이드 웹. 인터넷 주소를 입력할 때 www. 이 경우 웹 서버가 서버 역할을 하고, 웹 브라우저가 클라이언트가 된다. 여기서 웹 브라우저가 우리가 흔히 생각하는 인터넷 익스플로러, 파이어폭스, 사파리 등이다.

Q) 서버가 단순히 컴퓨터일 뿐이다?

A) 그렇다. 서버 소프트웨어를 설치한다면 당장 우리 랩탑도 서버가 될 수 있다. 그러나 보안 / 안정성 / 비용 등 모든 면에서 개인용 컴퓨터를 서버 컴퓨터로 쓰기엔 무리가 있다.

1.2. 서버와 인터넷은 무슨 관계인가?

- 인터넷은 모든 컴퓨터를 하나의 통신망 안에 연결하려던 시도에서 유래된 이름이며, (international network), 분산되어있는 컴퓨터를 하나의 네트워크로 묶는, 네트워크 시스템을 인터넷이라 한다.

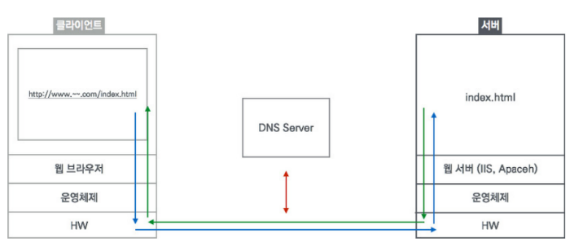
- 인터넷이 지원하는 서비스에 www, 텔넷, HTTP, FTP 등이 있다.

- 인터넷이 제공하는 서비스, 혹은 인터넷의 목표(분산 컴퓨터의 통신)를 달성하기 위한 것이 웹이다. 웹은 서버-클라이언트 시스템에 기반한다.

- (좋은 비유가 있어 가져옴) 인터넷이 도시라면, 웹은 집이고, 그 집을 짓는 도안 혹은 방식이 서버-클라이언트인 것이다.

Q) 프론트엔드, 백엔드?

A) 웹은 클라이언트와 서버의 통신이다. 여기서 서버 쪽 개발, 관리를 백엔드, 클라이언트 쪽 개발, 관리를 프론트엔드라 한다.



1.3. 서버 OS는 무엇인가?

- 서버는 주로 서버전용 os를 갖는다. ex) 리눅스, 솔라리스, 윈도우 서버 ..

- 서버 전용 OS의 특징은 다음과 같다.

GUI를 사용할 수 없거나 선택적으로 사용 가능,

중요한 데이터를 온라인 상에서 주기적으로 백업하는 고급 백업 기능,

볼륨 간, 장치 간 투명한 데이터 전송,

유연하고 고급적인 네트워킹 기능,

리소스, 데이터 등에 대한 강력한 보안

1.4. 구체적으로, 서버는 어떻게 작동되는가?

- 서버의 작동 과정 :

우리가 브라우저를 켜거나 특정 프로그램을 작동시키면, 이는 클라이언트 사이드 스크립트가 실행되는 것이다. 스크립트는 프로그램이라고 봐도 무방하다. 클라이언트 스크립트는 스스로 동작할 수도 있고, 서버에 요청을 보내 상호작용할 수도 있다. 클라이언트 스크립트가 보낸 요청을 서버가 받아서 해석하고, 실행시키는 프로그램을 서버 사이드 스크립트라고 한다. 클라이언트 스크립트와 서버 스크립트는 분리된 개념이기에, 각자의 역할은 서로가 볼 수 없다. 가령 서버가 내부 DB에 접근하고, 요청을 받아 무언가를 찾는 과정은 우리에게 (클라이언트 입장에서) 보여지지 않는다.

- (위키독스에 좋은 예시가 있어 인용하겠다)

메일 작성 웹사이트에 접속한다. 메일을 쓰고, 받을 이를 입력하고, 전송 버튼을 누르기까지는 모두 클라이언트 스크립트에 해당한다. 전송을 누름과 동시에 클라이언트 스크립트는 서버에 해당 메일 내용과 함께 요청을 보내고, 서버는 이를 받아 서버스크립트를 통해 이 메일을 다른 서버로 전송한다. 그러나 해당 서버스크립트의 코드는 이메일 주소, 서버 체계, 서버 주소, 개인 정보와 관련된 것들 등 보안이 필요한 내용을 포함한다. 따라서 이러한 코드는 클라이언트에게 주어지지 않는다. 메일을 성공적으로 보냈다면, 서버스크립트는 오로지 메일이 전송되었다는 응답만을 클라이언트에게 넘겨주는 것이다.

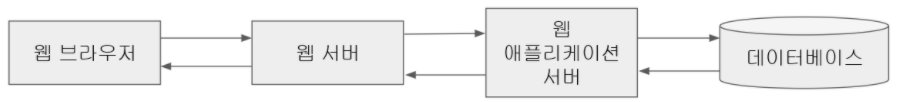
Q) 웹에서 마우스 우클릭으로 볼 수 있는 HTTP 소스는 무엇인가?

A) 클라이언트 스크립트의 코드이다. 현재 클라이언트 스크립트가 어떤 코드로 구성되어있는지, 어떤 명령/요청을 수행할 수 있는지를 보여주는 코드이다.

1.5. 웹 어플리케이션 서버란 무엇인가?

- 클라이언트의 요청 하나 당 서버가 하나의 프로세스를 생성하고, 웹 컨테이너를 통해 데이터베이스와 상호작용 후, 이를 다시 클라이언트에게 넘겨주는 방식을 취한다. 이때 동적인 컨텐츠는 CGI 라는 Common Gateway Interface를 따른다. 그러나 이 방식은 동적인 컨텐츠에 대한 수요가 높아짐에 따라 시스템에 더 큰 부하를 주게 된다.

- 이를 해결하기 위해 웹 서버와 웹 컨테이너를 합친 개념으로 웹 어플리케이션 서버(WAS)가 탄생한다. WAS가 서버와 데이터베이스 사이에서 동적인 요청을 전담하게 하고, 서버는 정적인 요청을 담당하는 방식을 택한다. 이러한 방식은 특히 JSP, ASP에서 사용된다. 대표적인 WAS로는 Tomcat, uWsgi, APACHE 등이 있다.



**Q) 프레임워크란?**

**A) 프레임워크는 개발의 편의를 위해 제공되는 뼈대이며, 도구의 모음집이라고 볼 수 있다. 가령, 파이썬이 서버 언어로 사용되지만, 파이썬의 빈 py파일부터 시작해서 서버를 만들기란 매우 어렵다. 이를 위해 Django라는 프레임워크가 view, serializer, model 등의 뼈대와 그에 걸맞는 장고 문법, 즉 도구를 제공하는 것이다.**

2. 서버 언어

- 1.4에서 살펴본 바, 서버, 클라이언트 스크립트 모두 스크립트 언어를 사용한다.

- 서버 사이드 스크립트 언어에는 PHP, Node.js, Python, JSP, ASP 등이 있다.

Q) 스크립트 언어란?

자바, C 등의 컴파일 언어는 코드 작성 후, 컴파일을 진행하고, 이때 컴파일러에 의해 기계어로 번역되는 것이다. 반대로 파이썬, 자바스크립트 등의 스크립트 언어는 컴파일 과정 없이 코드 작성 시에 저절로 기계어로 번역되어 작동된다. 컴파일 언어는 빠르고 디버깅이 쉽지만, 문법 난이도가 있는 편이며 운영체제마다 다른 방식으로 작업해야 한다. 반대로 스크립트 언어는 문법이 쉽고 한번 작성하면 끝이지만, 느리고, 컴파일 에러가 없기 때문에 디버깅이 어렵다. 스크립트 언어의 번역 과정을 인터프리터가 담당하여, 인터프리터 언어라고도 한다.

Q) 서버 컴파일 언어도 있나?

일반적으로 서버는 컴파일 언어를 쓰지 않는다. 공식적인 문서는 찾지 못하였으나 스택오버플로의 최다 추천수 글을 인용하면, 스크립트 언어가 느린 건 사실이다. 그러나 서버는 단지 클라이언트, 데이터베이스와 communicate할 뿐이라, 서버 자체의 실제 작업량은 많지 않다. 즉, 컴파일 언어를 씀으로써 얻는 응답 시간의 단축보다, 개발, 유지, 보수 측면에서 개발자 친화적인 환경이 더 중요시된다. 그리고 스크립트 언어도 오늘날에는 충분히 빠르다.

Q) 자바는 컴파일 언어인데, 왜 자바는 서버언어로 쓰일 수 있나?

A) 다른 컴파일 언어와 달리, 자바는 JavaVirtualMachine JVM을 갖는다. 바이트 코드 (컴파일러를 통해 자바 코드가 바이트 코드로 바뀐다.)가 JVM을 통해 각 운영체제, 플랫폼에 알맞게 클래스를 변환시켜준다. 따라서 컴파일 언어지만, 플랫폼 독립적이라는 인터프리터 언어의 장점을 공유하기 때문에 서버언어로 쓰일 수 있다.

2.1. JSP : 자바 서버 페이지, <자바면 다 해결!>

- 웹 상에서 자바만으로 서버를 개발하기 위해 만들어졌다.

- 서블릿을 기반으로 한다.

- 톰캣 WAS를 기반으로 한다.

- 대기업, 특히 금융권 내부 시스템 개발에 사용된다.

- [작동방식] 클라이언트로부터 요청을 받음 - 서블릿 컨테이너가 jsp 파일에 해당하는 서블릿이 있는지 확인하고, 없으면 jsp파일을 서블릿으로 컴파일함 - 이 서블릿은 다시 컨테이너에 담음 - 서블릿의 내용을 수행 / 정리하여 클라이언트에게 넘겨줌 - 클라이언트는 html태그를 분석하여 다시 사용자에게 보여줌.

- 장점) 플랫폼 독립성 (JVM과 비슷한 역할의 Java Servlet Container가 있기 때문) / 체계적이고 안정적. (자바와 OOP적인 성격 때문) / 기존의 자바 api 사용 가능 / 사용자가 많아지더라도 효율이 떨어지지 않음.

- 단점) 컴파일 과정이 있기에 느림

<https://sehun-kim.github.io/sehun/spring-short-story/>

2.2. ASP : Active Server Page, <MS를 등에 업은 무난한 언어>

- 동적으로 서버에서 작동하는 페이지를 의미한다. (<-> html)

- 윈도우 os를 기반으로 한다.

- 개인 서버의 구축에 용이하지만, 현재는 하향세이다.

- asp.dll이라는 문서 해독기를 사용한다.

- [작동방식] 클라이언트로부터 asp 요청을 받음 - asp소스를 asp.dll에 입력, 처리함 ()- 이를 다시 html로 정리하여 클라이언트에게 돌려줌.

- 장점) 유지보수 및 지원 (MS가 지원해줌) / 여러 언어를 동시 지원

- 단점) 라이센스 비용 높은 편 / 플렛폼 비독립적 / 확장성이 떨어짐

Q) DLL 이란?

A) 동적 링크. 특정 파일 실행 시 무조건 램에 같이 실리는 게 아니라, 프로그램에서 해당 기능 사용시에만 참조, 호출한다. 즉, 여러 프로그램이 한 dll을 동시에 돌려쓸 수 있고, 디스크 공간 절약, 메모리 절약 등의 효과가 있다.

2.3. PHP : Hypertext PreProcessor, <전통 강자, SQL사랑>

- C에 기반한다.

- 아직까지 대부분의 웹서버는 PHP로 개발된다.

- [작동방식] 클라이언트로부터 요청을 받음 - 서버가 PHP parser를 불러와 요청을 넘김 - PHP parser는 이를 수행하고 html 파일로 정리하여 서버에게 넘김 - 서버는 이를 클라이언트에 되돌려줌.

- 장점) 쉽고 직관적인 문법 / 오픈소스 / 가볍고 빠름 / 정보가 많음(이미 대다수의 서버들이 PHP로 이루어졌음) / SQL과의 연동성

- 단점) 복잡한 클래스 설계의 어려움(OOP적이지 않음) / 보안에 취약함

- 다만 php5 버전부터는 OOP 개념을 많이 추가하여 최대한 단점을 보완하려고 노력중.

2.4. Node.js <신흥 강호, 자바스크립트의 회심작>

- 자바스크립트를 기반으로 한다.

- 내장 HTTP 서버 라이브러리를 포함해 웹 서버에서 Apach같은 별도의 WAS가 필요없다.

- 비동기 I/O 처리

- 단일쓰레드 이벤트 루프 기반.

- [작동 방식] 클라이언트로부터 요청을 받음 - 단일 쓰레드가 이벤트루프를 돌며 필요한 경우 쓰레드 풀의 쓰레드와 상호작용함 - 리턴받은, 혹은 스스로 수행한 결과를 정리하여 클라이언트에 돌려줌. (아래 상세설명)

- 장점) 고도의 확장성 / 쉬운 유지보수 / 풀스택 개발이 용이 (자바스크립트를 기반으로 하기에) / Json과의 연동성 / 쉬운 비동기 프로그래밍(이후 설명) / 향후 성장 가능성 (Chrome V8엔진으로 빌드) / WAS가 없으므로 더 높은 통제력

- 단점) 단일 쓰레드 환경이라 메모리 효율이 좋지 않음.

2.5. Python <만능 파이썬>

- 굉장한 범용성을 지닌 파이썬은 서버에도 사용된다.

- 다만 한국에서는 아직까지 거의 쓰이지 않고 있다. (인프라 약함)

- [작동방식] 파이썬은 서버사이드스크립트 전용 언어가 아니기에 파이썬만의 획일화된 작동 방식이 없는 것으로 보임. 전체적은 흐름은 클라이언트의 요청을 받아 정적인 것은 Nginx, 동적인 것은 WSGI로 넘겨 작업 후 다시 서버가 이를 받아 정리해서 클라이언트에게 돌려주는 방식.

- 장점) 문법이 매우 쉬워 개발과 유지보수 측면에서 매우 생산적 / 라이브러리가 풍부함

- 단점) 느림(파이썬 자체가 속도보다 편의성에 초점이 맞춰졌으므로.)

3. 서버 언어와 관련된 개념들

3.1.

Q) 서블릿 및 서블릿 컨테이너란?

A) “클라이언트 요청을 처리하고 그 결과를 다시 클라이언트에게 전송하는 Servlet 클래스의 구현 규칙을 지킨 자바 프로그램”(직접인용 <https://jusungpark.tistory.com/15>).

- 멀티스레드 방식으로 기존 CGI의 한계를 극복한, 자바의 확장 CGI이다.

- 기존 CGI의 단점은 매번 프로세스를 생성해 메모리에 부하를 준다는 것이다. 이를 해결하기 위한 방안이 WAS였다.

- 요청이 정적이면 웹 서버 자체적으로 , 동적이면 WAS로 넘기는데, 이 판단을 서블릿이 한다.

이러한 서블릿들을 관리하고 스레드를 생성하여 서블릿을 작동하게 지휘하는 것이 서블릿 컨테이너이다. 위에서 WAS가 서버와 데이터베이스 사이에서 (혹은 서버를 대체하여) 작동하는 것이라고 했는데, WAS 톰캣이 바로 이 서블릿 컨테이너이다.

3.1. 단일쓰레드? 멀티쓰레드?

- 쓰레드는 메모리, cpu를 사용하는 하나의 작업 단위이다. Spring을 비롯한 웹 서버는 멀티쓰레드를 택하며, 쓰레드 풀 내의 쓰레드들이 각각 클라이언트의 요청을 담당하는 방식이다.

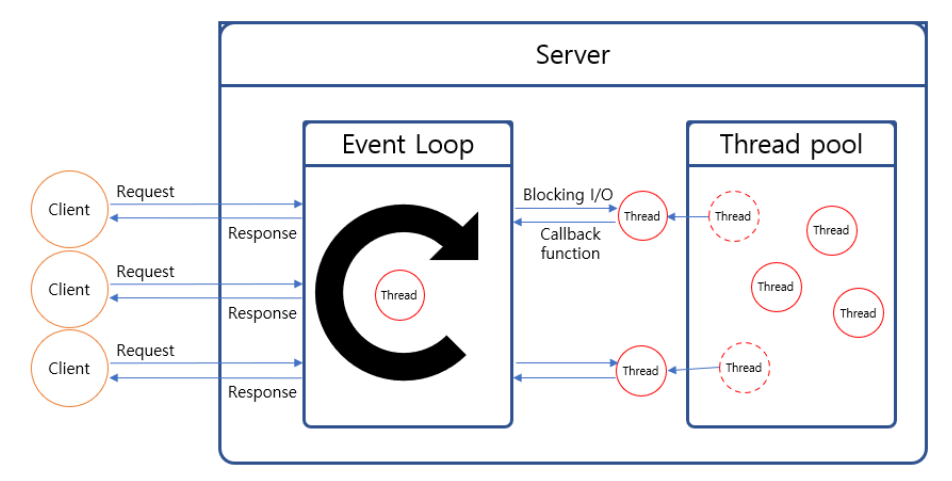
- Node.js는 단일쓰레드를 택하며, 하나의 쓰레드가 이벤트 루프를 돌며 쓰레드 풀의 쓰레드들과 상호작용하는 방식이다.

- 쓰레드를 물리적으로 확인할 수 있는 부분은 call stack이다. 단일쓰레드는 하나의 call stack만을 사용하고, 멀티쓰레드는 여러 개. 쉽게 말하자면, 멀티 쓰레드와 달리 하나의 쓰레드가 무한루프를 돌며 비동기 방식으로 클라이언트 요청들을 처리하고, 필요에 따라 쓰레드 풀의 다른 쓰레드들에게 처리를 지시하는 것이다.

3.2. 이벤트 루프?

- 단일 쓰레드의 효율을 극대화하기 위해 Call Stack과 Callback Queue를 관리하는 작업 방식이다. Node.js에서 사용된다.

- [작동 방식] 자바스크립트(+자바스크립트 기반의 Node.js)는 하나의 Call Stack에 차곡차곡 실행될 코드가 쌓는다. 이때 코드는 클라이언트의 요청에 관한 것이다. Call Stack에서 코드들이 실행되며 동기 함수는 그대로 실행되고, 비동기 함수는 Dom, Ajax 등과 같은 Web API에 처리를 요청한다. Web API는 이 요청을 받아 비동기함수의 콜백함수를 Callback Queue에 밀어넣는다. Event Loop는 Call Stack과 Callback Queue를 확인하고, 만약 Call Stack의 코드가 다 실행되었다면, Callback Queue의 콜백 함수들을 Call Stack에 쌓는다. 비동기가 다른 함수에게 처리를 위임하고, 동기 코드/함수들이 다 실행된 후에 값이 리턴되는 게 바로 이러한 이유이다. Call Stack의 동기 코드/함수들이 다 실행되고 나서야 Callback Queue에서 Call Stack으로 이동하기 때문이다.



3.3. Callback 함수란?

- 함수의 인자로 작용하는 함수이며, 전달된 함수 내에서 호출된다.

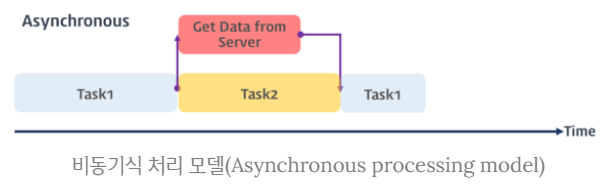
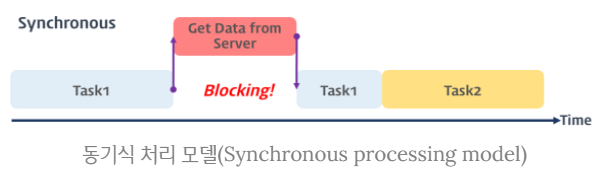
- Collections.sort(~ , 함수) 같은 예시에서 주로 볼 수 있다. 이는 서버에서 명령 처리 시 비동기적 메소드의 정보가 누락되는 경우가 발생하기 때문에 사용된다.

- Ex) ajax의 $.get 메소드는 호출되더라도 비동기적이기 때문에 비동기적이지 않은 코드가 먼저 실행된 후에서야 값을 리턴한다. 이 때문에 $.get 전에 Null 데이터가 반환될 가능성이 있는 것이다. 처음부터 함수에 인자로 콜백함수를 넣어주면 이를 해결할 수 있다.

3.4. 동기, 비동기란?

- 동기 : 직렬적으로 태스크를 수행하며, 하나의 태스크가 수행 중일 경우 다른 태스크들은 중단, 대기한다. 이때 중단, 대기하는 현상을 두고 blocking이라 한다.

- 비동기 : 병렬적으로 태스크를 수행하며, 하나의 태스크가 종료되지 않더라도 다음 태스크가 수행된다. 이때 수행은 이벤트 핸들러에 의해 진행되며, 각각의 태스크 상태는 non-blocking이라 표현된다.



- Ex) 편의점에서 물건 살 때 / 패스트푸드 점에서 음식 주문할 때

Q) 무조건 비동기가 좋다?

아니다. 작업이 꼭 순차적인 순서로 진행되어야 하는 경우, 혹은 부하가 적어 동기로도 충분히 커버할 수 있는 경우 동기가 더 선호된다. 비동기는 핸들러 및 외부적인 리소스를 호출해야 하기 때문에, 메모리 / 시간 상 한계가 있을 수 있다

참고 사이트)

<https://hpitos.tistory.com/7>

<https://www.fasthosts.co.uk/blog/client-side-vs-server-side-scripting/#:~:text=The%20most%20popular%20server%2Dside,customised%20content%20to%20the%20user>.

<https://conol.tistory.com/9>

<https://wikidocs.net/22322#:~:text=%EC%84%9C%EB%B2%84%20%EC%82%AC%EC%9D%B4%EB%93%9C%20%EC%8A%A4%ED%81%AC%EB%A6%BD%ED%8A%B8%EB%8A%94%20%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%A8,%EC%9D%B4%EA%B3%A0%2C%20%EC%8B%A4%EC%8B%9C%EA%B0%84%EC%9D%98%20%EC%9D%98%EB%AF%B8%EC%9D%B4%EB%8B%A4>.

<https://imraccoon-developer.tistory.com/12>

<https://pathas.tistory.com/176>

<https://m.blog.naver.com/PostView.nhn?blogId=blogpyh&logNo=220020363988&proxyReferer=https:%2F%2Fwww.google.com%2F>

<https://includestdio.tistory.com/25#:~:text=%EC%A3%BC%EB%A1%9C%20%EC%A0%95%EC%A0%81%20%EC%9E%90%EC%9B%90%EC%9D%B8%20HTML,%EC%84%9C%EB%B2%84%EC%97%90%20%EC%9A%94%EC%B2%AD%EC%9D%84%20%EB%84%98%EA%B9%81%EB%8B%88%EB%8B%A4.&text=%EC%9B%B9%20%EC%84%9C%EB%B2%84%EB%A1%9C%EB%B6%80%ED%84%B0%20%EC%9A%94%EC%B2%AD%EC%9D%84,%EC%9B%B9%20%EC%84%9C%EB%B2%84%EB%A1%9C%20%EB%B0%98%ED%99%98%ED%95%A9%EB%8B%88%EB%8B%A4>.

<https://medium.com/@chrisjune_13837/web-%EC%9B%B9%EC%84%9C%EB%B2%84-%EC%95%B1%EC%84%9C%EB%B2%84-was-app%EC%9D%B4%EB%9E%80-692909a0d363>

<https://jusungpark.tistory.com/15>

<https://logical-code.tistory.com/30>

<https://junshock5.tistory.com/118?category=875035>

<https://dheldh77.tistory.com/entry/%EB%84%A4%ED%8A%B8%EC%9B%8C%ED%81%AC-%EC%8B%B1%EA%B8%80%EC%8A%A4%EB%A0%88%EB%93%9C-%EC%84%9C%EB%B2%84%EC%99%80-%EB%A9%80%ED%8B%B0%EC%8A%A4%EB%A0%88%EB%93%9C-%EC%84%9C%EB%B2%84>

<https://pro-self-studier.tistory.com/89>

<https://velog.io/@thms200/Event-Loop-%EC%9D%B4%EB%B2%A4%ED%8A%B8-%EB%A3%A8%ED%94%84>

<https://poiemaweb.com/js-async>