20220427 - 동기/비동기, 스프링 프레임워크

Created	@2022년 4월 27일 오후 9:56
Tags	Personal
Property	

💡 동기(Synchronous) vs 비동기(Asynchronous)

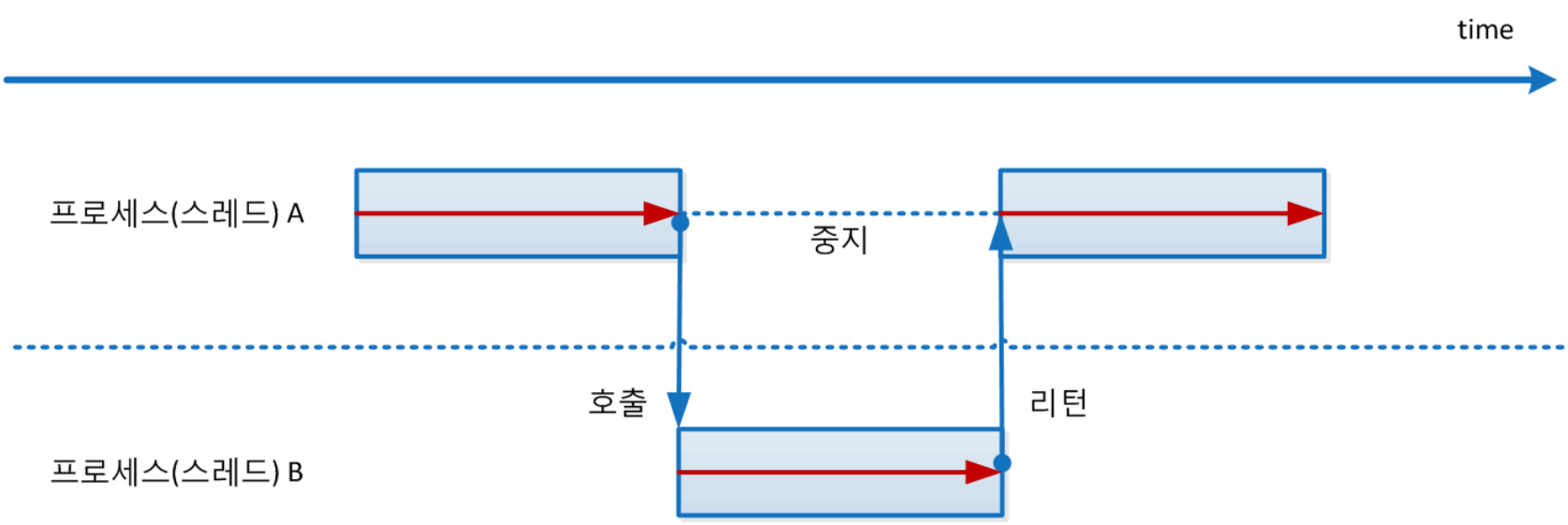
- 동기와 비동기로 작성한 코드의 가장 큰 차이 중 하나는 **런타임 시 발생하는 지연시간**
- 모든 코드가 동기적으로 실행될 경우 → 카톡을 보내면 카톡이 오기 전까지 아무것도 할 수 없음
- 하지만 이러한 방식이 사용되어야 하는 경우도 있음 → ATM, 키오스크 등

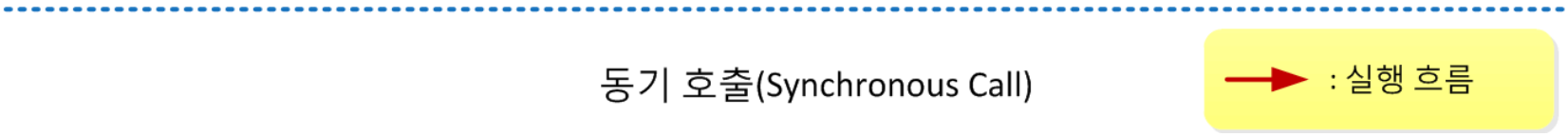
동기 방식의 장점

- 코드 파악이 쉬움
- 유지보수와 디버깅이 쉬움
- 코드가 순서대로 실행되기 때문에 Breakpoint를 옮겨가며 디버깅하면 에러 찾기가 용이하다.
- 클라이언트와 서버의 작업 처리 단위(transaction)를 동시에 맞춤

동기 방식의 단점

- 싱글 스레드를 사용하는 자바스크립트에서는 문제가 되는 경우가 많음
- 데이터베이스 작업이 완료되어 데이터가 오기까지 기다린다면, 애플리케이션이 유휴 상태가 되고 다른 작업을 할 수 있는 시간이 낭비됨

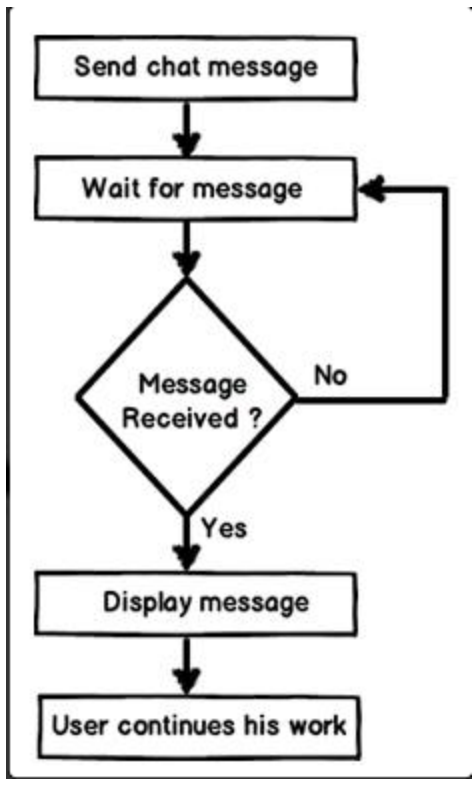




- 웹에서는 동기로 구현할 경우 최악의 UX를 제공할 수 있다.
- 서버로부터 데이터를 요청하는 것 외에도 클릭, 입력 이벤트 등을 모두 동기적으로 처리한다면 대기시간에는 차이가 있겠지만 결국 멈추게 될 것이다.

비동기 방식

- 자바스크립트에서는 동기 방식의 문제를 해결하기 위해 콜백 함수를 사용한다.
- 콜백 함수를 사용하여 데이터가 준비 되었을 때 런타임에 호출할 함수를 제공하여 블록킹 문제를 해결한다.
- 코드의 특정 부분이 런타임 시스템에 제어권이 있으므로 **“제어의 역전(IoC)”**이 일어난다.



- 사용자가 채팅을 보냄 → 보내고 나서 자신의 작업을 계속 진행 → 응용 프로그램이 데이터가 준비되면 이벤트를 수신함 → 메시지가 도착하면 이벤트가 활성화되고 콜백 함수가 호출됨(제어의 역전)
- 이벤트가 프로그램 순서를 주도한다. 이런 이벤트 흐름 접근 방식은 직접 호출을 하지 않으므로 프로그램의 유연성은 높아지고 비동기적 호출을 할 수 있다. 이벤트 호출 후 애플리케이션이 제어권을 반환하여 다른 작업을 계속할 수 있다.

비동기가 답인가?

- 비동기 함수를 구현하면 지속적으로 응답할 수 있게 해주지만, 동기적 방식에 비해 많은 비용이 발생
- 동기 함수의 경우 각 단계가 순서대로 실행되는 구문들의 절차라 표현할 수 있다. 그리고 각 단계의 결과는 이전 단계의 결과에 의존하게 된다. 다음에 발생할 상황 파악이 쉽고, 코드 작성 및 디버깅이 쉽다.
- 비동기 함수의 경우 어떤 순서로 작업이 실행되더라도 상관이 없다. 즉, 실행 순서를 보장할 수 없다.
- 이에 비동기 함수라도 각 작업을 순서대로 실행되게 보장하도록 하기 위한 노력이 있었으며, 콜백 함수를 중첩시키는 방법을 통해 가능하게 되었다. 그러나 이는 콜백 지옥을 발생하게 되었다.

🤔 Spring Framework

- 사전적 정의: 자바 플랫폼을 위한 오픈 소스 애플리케이션 프레임워크 - 간단히 스프링
  - **프레임워크:** 뼈대나 기반 구조라는 뜻으로 IoC 개념이 적용된 대표적 기술. 소프트웨어에서는 **'소프트웨어의 특정 문제를 해결하기 위해서 상호 협력하는 클래스와 인터페이스 집합'** → 프로그래머가 완성해야됨
  - **라이브러리:** 단순 활용 가능한 도구들의 집합. 개발자가 만든 클래스에서 호출하여 사용. 클래스들의 나열로 필요한 클래스를 불러서 사용
  - 프레임워크는 전체적인 흐름을 스스로가 쥐고 있으며 사용자가 그 안에 코드를 작성. 라이브러리는 사용자가 전체적인 흐름을 만들며 라이브러리를 가져다 쓰는 것
- 엔터프라이즈급 애플리케이션을 만들기 위한 모든 기능을 종합적으로 제공하는 경량화 솔루션
  - **애플리케이션 프레임워크:** 일반적인 라이브러리나 프레임워크는 특정 업무 분야나 한 가지 기술에 특화된 목표를 가지고 만든다. 그러나 애플리케이션 프레임워크는 특정 계층이나 기술, 업무 분야에 국한되지 않고 애플리케이션의 전 영역을 포괄하는 범용적 프레임워크이다. 즉, 애플리케이션 개발의 전 과정을 빠르고 편리하며 효율적으로 진행하는데 일차적인 목표를 두는 프레임워크
- JEE가 제공하는 다수의 기능을 지원 → JEE 대체 프레임워크
- 자바로 엔터프라이즈 애플리케이션을 만들 때 포괄적으로 사용하는 프로그래밍 및 설정 모델을 제공해주는 프레임워크로 애플리케이션 수준의 인프라 스트럭처 제공
- 스프링 삼각형 → IoC/DI, AOP, PSA, 가운데 POJO
- 현실에서의 반영이 어려운 EJB 대체

특징

- 경량 컨테이너 → 자바를 직접 관리(객체의 라이프 사이클 관리, 스프링으로부터 필요한 객체 얻어오기)

- POJO 방식의 프레임워크 → 특정 인터페이스 구현이나 상속 필요가 없어 기존에 존재하는 라이브러리 등을 지원하기에 용이하고 객체가 가벼움
- 제어의 역전(IoC)를 지원 → 컨트롤의 제어권이 사용자가 아니라 프레임워크(스프링 컨테이너)에 있어서 필요에 따라 스프링에서 사용자의 코드를 호출
- 의존성 주입(DI) 지원 → 각각의 계층이나 서비스 간 의존성이 존재할 경우 프레임워크가 서로 연결시켜줌
- 관점 지향 프로그래밍(AOP) 지원 → 트랜잭션, 로깅, 보안과 같이 여러 모듈에서 공통적으로 사용하는 기능을 분리해서 관리 할 수 잇다.
- 영속성(데이터 영구 저장)과 관련된 다양한 서비스를 지원 → MyBatis 등과 같은 완성도 높은 데이터베이스 처리 라이브러리와 연결할 수 있는 인터페이스 제공
- 높은 확장성 → 수많은 라이브러리 지원
- WAS에 독립적인 개발 환경 → 과거 EJB가 동작하려면 고가의 느리고 무거운 서버가 필요했으나 스프링은 가장 단순한 서버 환경인 톰캣이나 제티(Jetty)에서도 완벽하게 동작함. 단순한 개발 툴과 기본적인 개발 환경으로도 엔터프라이즈 개발에서 필요로 하는 주요한 기능을 갖춘 애플리케이션을 개발하기에 충분

주요 모듈

- IoC 컨테이너: Reflection(런타임 시점에 자신의 구조와 행위 관리)을 이용해서 객체의 생명주기를 관리하고 의존성 주입을 통해 각 계층이나 서비스들간의 의존성을 맞춰줌. XML 파일에 의해 설정되고 수행됨
- 관점 지향 프로그래밍 프레임워크: 강력한 관점 지향 프로그래밍 프레임워크인 AspectJ 지원, 스프링 자체적으로 지원하는 런타임 조합 방식도 존재
- 데이터 액세스 프레임워크: JDBC, MyBatis 등에 대한 지원 제공, 쉬운 데이터베이스 프로그래밍
- 트랜잭션 관리 프레임워크: XML 설정파일 등을 이용한 선언적 방식 및 프로그래밍을 통한 트랜잭션 관리 제공
- MVC 패턴: Spring MVC라 불리는 MVC 패턴을 사용. DispatcherServlet이 Controller 역할을 담당하여 각종 요청을 적절한 서비스에 분산시켜주며, 이를 각 서비스들이 처리하여 결과를 생성하고 그 결과는 다양한 형식의 View 서비스들로 화면에 표시될 수 있음
- 배치 프레임워크: 특정 시간대에 실행하거나 대용량의 자료를 처리하는데 쓰이는 일괄 처리를 지원(Quartz)