로깅 라이브러리 개선 사항 목록

1. JPA 수행 관련 로깅
   1. spring data jpa뿐 아니라 EntityManager를 직접 호출에 대한 로깅 지원
2. 비동기 기반 프로젝트
   1. mvc가 아닌 webflux 기반 비동기 기반 프로젝트에 대한 로깅 지원
   2. openfeign, webclient
3. 외부 API
   1. OpenFeign, WebClient를 통한 비동기 호출에 대한 로깅 지원
4. 로그 저장
   1. Rolling Appender 방식의 느린 성능 개선
   2. 로그들의 트랜잭션 보장
   3. 로그 저장 테이블 구조
      1. 로그 저장 테이블이 요청을 저장하는 테이블을 부모 테이블을 가지도록

**1. JPA 수행 관련 로깅**

현재 라이브러리는 JPA 수행 관련 로깅에 있어서 Spring Data JPA에 대한 로깅만을 지원함.

EntityManager를 통해 CRUD 수행, JPQL, QueryDSL등에 대해 JPA를 수행하는 경우에 있어서 로깅을 지원해야 함.

**2. 비동기 기반 어플리케이션**

현재 라이브러리는 MVC 기반의 스프링 어플리케이션에 관해서만 로깅을 지원함.

어플리케이션 내부적으로 비동기 기만으로 동작하는 Webflux 기반의 어플리케이션에 대해 로깅을 지원해야 함.

**방안**

1. 각각의 코드 수행을 하는 스레드에 있어서 MDC의 값을 전파하여 어플리케이션 내부의 동작을 로깅

**3. 외부 API**

현재 라이브러리는 외부 API 호출에 있어서 라이브러리에서 동기 방식인 제공하는 RestTemplate, 혹은 ClientHttpRequestInterceptor를 어플리케이션 측에서 체인을 구성하는 경우에 대해서만 로깅을 지원함. 비동기 기반 호출, OpenFeign(RestAPI 호출), WebClient등의 비동기 방식의 외부 API 호출에 대한 로깅도 지원해야 함.

**방안**

1. 라이브러리를 적용한 어플리케이션 A에서 요청 전처리(MDC 값 전파)
   1. Reactor Context를 통해 요청 전처리할 스레드에 MDC값 유지
2. 요청 헤더, 혹은 보낼 메시지에 MDC 값을 추가
3. 만약 요청을 보낼 어플리케이션 B가 MVC 기반의 스프링 어플리케이션이라면 현재 라이브러리에서 지원하는 필터를 사용하면 되지만, 비동기 방식이라면
   1. 어플리케이션 B 내부적으로 동작하는 스레드간에 MDC 값들을 전파하는 과정이 필요
4. 라이브러리를 적용한 어플리케이션B는 응답 헤더에 MDC 값을 담아 보냄
5. 어플리케이션A에서 응답에 대한 후처리를 통해 헤더에서 값 꺼내기
6. 응답을 후처리한 스레드에 꺼낸 값 다시 설정.
7. 응답을 후처리한 스레드로부터 Reactor Context를 통해 MDC에 값을 넣은 후 클라이언트에게 응답 반환환

**4. 로그 저장**

문제점1(성능)

현재 라이브러리의 성능을 많이 저하시키는 부분은 Rolling Appender를 통한 로그 백업 과정이다. ‘백업 파일을 생성 -> 파일에 쌓인 로그를 백업 파일에 반영 -> 파일에서 백업한 내용 지우기’의 과정이 동시성 처리를 기반으로 동작하기에 요청을 처리하는 어플리케이션에서 수행 시간에 많은 영향을 끼침.

문제점2(로그들의 트랜잭션 보장)

클라이언트의 한 요청에 대해 물리적으로 다른 서버에 있는 어플리케이션끼리 통신하는 과정을 통해 응답을 내릴 경우, 각각의 서버에서 생성되는 한 요청에 대한 로그들을 DB에 저장하는 과정에서 트랜잭션이 보장이 되지 않음. 서버1의 어플리케이션A, 서버2의 어플리케이션B중 한 곳의 로그들이 DB에 반영이 되지 않을 수 있음.

문제점3(로그 저장 테이블 구조)

현재 방식으로는 로그들을 요청 단위로 묶어내질 못함. 이렇게 되면 후에 수많은 요청에 대해 모니터링 시스템에서 로그를 검색하는 과정이 너무 느려짐. DB에 저장될 로그 데이터는 인덱싱도 하지 못하는 데이터들이기 때문에, 요청 단위로 묶어서 요청 목록을 보여주는 과정이 보다 유리할 것임.

**방안**

1. kafka, RabbitMQ 같은 비동기 메시지 큐를 사용하여 로그를 중앙 저장소에 전송하는 방법이 성능, 안정성에서 가장 좋아 보임
   1. Rolling File Appender 방식이 아닌, Kafka Appender를 통해 로그를 중앙화된 로그 시스템 (Elasticsearch, Splunk…)으로 로그 전송
   2. 장애 발생 시 로컬 파일에도 남기고, Kafka로 전송 실패 시 재시도 가능
   3. 중앙화된 로그 시스템에서는 로그들을 요청 단위로 묶을 수 있음
      1. 요청 목록 테이블을 부모로 가지는 로그 테이블 구조를 설계할 수 있음