SAMSUNG SW ACADEMY FOR YOUTH

對外秘

프로젝트 명세서

유지보수를 위한 로깅(Logging)

목차

1. 프로젝트 개요

본 과제는 로깅(Logging)에 대해 학습하고 관통 프로젝트에 적용하며 실습하는 과제입니다. 과제를 통해 유지보수에 있어 로그의 중요성을 이해하고 이를 프로젝트에 적절히 적용할 수 있게 될 것입니다.

로깅이란 시스템에서 발생한 사건(event)을 기록하는 것을 의미합니다. 로깅을 통해 남겨진 기록을 로그(log)라고 부릅니다. 로그는 유지보수 활동에 있어 매우 중요한 역할을 합니다. 대표적으로는 장애의 발생을 확인하고 어떤 사건이 장애를 유발했는지 추적하는 것이 있습니다. 적합한 곳에 작성해둔 로그 출력 코드는 시스템 운영과 유지보수에 있어 개발자에게 동아줄이 되어 주기도 합니다. 유지보수 뿐만 아니라 개발을 하며 부딪히는 많은 문제들을 디버깅하기 위해서도 로깅을 많이 사용합니다. 또한 시스템의 보안 점검과 감사(auditing)에도 로그는 아주 중요한 역할을 합니다.

로컬 환경만 다뤄본 개발자는 흔히 System.out.print(파이썬이라면 print, JS라면 console.log)로 필요한 정보를 출력합니다. 하지만 이것은 운영 관점에서 좋지 않은 방식입니다. 본 과제를 통해 로깅에 대해 학습하고 실습하며 향후 프로젝트에 올바른 로깅을 적용할 수 있도록 준비하시기 바랍니다.

2. 로깅 개요

앞서 설명했듯이 로깅(logging)이란 시스템에서 발생한 사건을 기록하는 것을 의미하며, 그 기록을 로그(log)라고 부릅니다. 로그에는 다음과 같은 특징이 있습니다.

- 시계열(timeseries) : 대부분의 로그는 시계열 데이터의 특성을 가지고 있습니다. 이벤트가 발생한 시간 순서대로 기록됩니다.
- 이벤트(event)와 메시지(message) : 어떠한 사건이 발생했는지를, 시스템이 어떤 상태였는지 등을 포함합니다. 이 정보는 진단에 있어 필수적인 역할을 합니다. 예를 들어 예외(Exception)가 발생한 상황을 기록하는 로그에는 스택 트레이스(Stack trace) 정보, 예외의 종류, 예외가 발생한 코드에서 참조하는 변수의 값 등을 출력하게 합니다.
- 로깅 레벨(level) : 대부분의 로그는 이벤트의 **수준 정보**를 포함합니다. 일상적인 수준의 정보는 INFO 레벨로 출력합니다. 반면 큰 장애가 발생하는 상황이라면 FATAL(또는 SEVERE, CRITICAL, 치명, 심각) 레벨로 로그를 남깁니다. 개발자의 상세한 디버깅을 위한 로그라면 TRACE 나 DEBUG 레벨로 출력합니다. 일반적으로 로그 레벨을 5~6 개로 나눕니다.

  ■ 로그 레벨 순서(심각도 순) : FATAL > ERROR > WARN > INFO > DEBUG > TRACE

로그는 운영체제를 비롯하여 컴퓨터 시스템의 대부분에서 사용됩니다. 생산된 로그는 충돌 리포트(Crash Report)에 첨부되어 진단을 위해 사용되기도 합니다. 이 정보는 비행기의 블랙박스와 비슷한 역할을 합니다. 개발자는 로그를 보며 문제 상황을 이해하고 이를 재현하며 문제의 원인을 찾아내고 결함을 수정해 시스템을 개선하는데 사용합니다.

애플리케이션 개발을 할 때는 기존에 만들어진 로그 라이브러리(또는 프레임워크)를 활용합니다. 이런 라이브러리들은 언어 자체에 내장된 것들도 있고 별도 라이브러리로 만들어진 것도 있습니다. 로깅 라이브러리들은 보통 다음과 같은 기능을 제공합니다.

- 포맷 설정 : 시간, 로그 레벨, 프로세스 또는 스레드 ID, 출처(프로그램과 라인), 메시지 등의 정보를 어떤 형태로 출력할지 제어할 수 있습니다.

- 레벨 설정 : 앞서 설명한 로그 레벨에 따라 API 를 별도로 갖추고 있습니다. 예를 들어 debug 레벨로 출력하려면 logger.debug 함수로, error 레벨로 출력하려면 logger.error 함수로 출력합니다.

- 로그 출력 설정 : 콘솔(표준 출력)이나 파일로 또는 양쪽 모두에 출력되게 할 수 있습니다. 일반적으로 로컬 환경에서는 콘솔 출력으로 충분합니다. 하지만 여러 사람이 환경을 공유하는 개발, 테스트 환경에서는 파일 출력을 함께 합니다. 운영 환경에서는 콘솔을 보는 일이 드물기 때문에 파일 출력을 중심으로 합니다.

- 로그 출력 레벨 설정 : 로그가 출력되는 레벨을 설정할 수 있게 합니다. 일반적으로 운영 환경에서는 성능과 보안 등을 이유로 가급적 로그가 적게 출력되도록 설정합니다. 환경에 따라 로그 설정을 다르게 하기도 합니다. 예를 들어 로컬과 개발 환경에서는 DEBUG 레벨로, 테스트 환경에서는 INFO 레벨로, 운영 환경에서는 WARN 또는 ERROR 레벨로 설정하기도 합니다. 설정된 레벨 이하의 로그는 출력되지 않습니다.

- 로그 저장 관리 : 로그 파일은 쉽게 커질 수 있습니다. 따라서 로그 로테이션(rotation)이나 보관 주기 설정, 자동 압축 등을 지원합니다.

애플리케이션 개발자는 본인이 작성한 코드에서 추적이 필요한 부분, 문제가 생길 수 있는 부분에 로그 출력 코드를 추가합니다. 또한 횡단 관심사(cross cutting concern)를 핸들링 하는 코드(AOP 와 같은)에서 공통적으로 필요한 로그를 출력하게 할 수 있습니다. 예를 들어 인증 AOP 에서는 인증에 실패한 요청에 대해 별도 로그를 남길 수 있습니다. 이 정보는 보안 위협을 탐지하는데 사용될 수 있습니다.

로깅 자체는 개발에 필수적인 요소라고 보기 어렵습니다. 오히려 성능 측면에서는 I/O 를 발생시키므로 악영향을 준다고 볼 수 있습니다. 추가로 별도의 저장공간도 필요로 합니다. 하지만 디버깅과 유지보수, 그리고 운영 관점에서는 코드와 데이터 다음으로 중요한 요소로 볼 수 있습니다. 앞서 강조했지만 적절한 위치에 작성한 로그 코드는 많은 시간을 아끼고 큰 장애를 예방할 수도 있습니다.

3. 참고 자료

- Spring Boot Logging
  https://docs.spring.io/spring-boot/docs/3.2.x/reference/html/features.html#features.logging

- Sentry 로 우아하게 프론트엔드 에러 추적하기
  https://tech.kakaopay.com/post/frontend-sentry-monitoring/

- Django Logging
  https://docs.djangoproject.com/en/5.1/topics/logging/#topic-logging-parts-loggers

- 모바일(안드로이드)의 로그 작성 가이드
  https://developer.android.com/studio/debug/am-logcat?hl=ko

4. 과제

본 과제는 로깅에 대해 조사하고 1 학기에 진행한 관통 프로젝트에 적용해 보는 방식으로 진행합니다. 앞의 참고 자료 부분에 몇 가지 자료를 기재했으니 참고해 보시기 바랍니다.

1. 로깅 프레임워크 조사 및 선정

◆ 백엔드 개발에 주로 사용하는 스프링 부트에서는 Logback 을 기본으로 채택하고 있습니다. 이외에도 Log4j2 를 사용하도록 설정할 수도 있습니다.

◆ 파이썬 백엔드 중 하나인 장고에서는 파이썬의 내장 로깅 라이브러리를 사용할 수 있습니다.

◆ 프론트엔드는 사용자의 환경에서 로깅이 발생하므로 집중화된 추적이 어렵습니다(사용자의 PC 를 들여다보지 않는다면요). 특히 SPA 로 개발된 경우 사용자 추적이 더 어려울 수 있습니다. 따라서 백엔드에 별도의 로그 저장용 API 를 만들고 이를 호출하는 방식으로 많이 구현합니다.

◆ 모바일(안드로이드) : 모바일 개발 환경에서는 대부분 SDK 에 로깅과 에러 리포팅 기능이 내장되어 있습니다.

2. 관통 프로젝트에 적용

◆ System.out.print 와 같은 표준 입출력(STDIO)으로 로그를 출력하고 있는 부분을 찾아 로깅 프레임워크를 사용하도록 적용해 봅니다.

◆ 다양한 레벨의 로그 메시지를 남겨보고 로그 출력 설정에 따라 달라지는 동작을 확인해 봅니다.

◆ 프론트엔드에도 사용자의 중요한 액션이나 장애 상황에 대해 이를 백엔드로 전송할 수 있도록 구현해 봅니다.

3. 패치 파일 생성과 제출

◆ 변경한 내용을 커밋하고, 변경 전과 후 내용을 비교해 patch 파일로 만들어 이를 산출물로 제출합니다.

4. 더 해보기

앞의 방식은 최소한의/전통적인 로깅 방식을 따르고 있습니다. 다음의 항목을 학습해 조금 더 발전시킬 수 있습니다.

◆ 환경 분리 : 앞서 설명했듯이 로컬과 개발, 테스트, 운영 환경에 따라 로그 정책을 다르게 설정할 필요가 있습니다. 이를 어떻게 효율적으로 적용할 수 있는지 조사해 봅시다. #키워드 : 환경 분리

◆ 분산 추적(중앙화 로깅) : 클러스터링, HA(High Availability, 고가용성)이나 컨테이너(도커 등)와 MSA(Micro Service Architecture) 등 분산/휘발성 아키텍처로 인해 하나의 사용자 이벤트를 연속적으로 추적하기 어려워지고 있습니다. 이를 해결하기 위해 중앙화된 로깅(Centralized Logging)을 적용하기도 합니다. 이에 대해 학습하고 향후 프로젝트에 적용할 수 있는지 시험해 봅시다.

5. 산출물 제출

https://lab.ssafy.com/s12-study/seasonal_fesw 의 “산출물 제출 가이드.docx” 참조