21 로깅과 사용추적

HTTP 완벽가이드 - V. 콘텐츠 발행 및 배포

목차

거의 모든 서비스는 시스템의 상태나 오류를 “**기록(로깅)**” 한다.

▼ 1. 로그란 무엇인가?

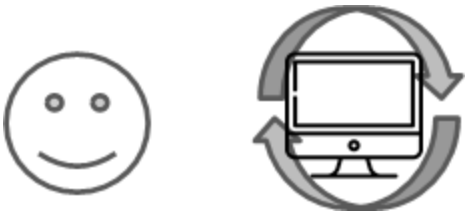
> 프로그램의 동작, 상태를 관찰할 수 있도록 제공하는 정보

> 프로그램 동작시 발생하는 일의 기록

▼ 누가 - Who

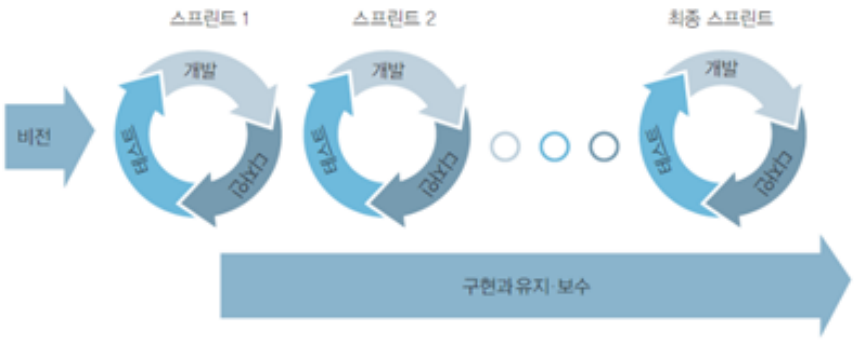
- 다양한 사람들이 활용 가능
  - **>> 개발 <<**
  - 마케팅
  - 기획/디자인
  - 기타

하지만 우리는 SW 시스템을 개발하는 개발자 중점으로..!



▼ 언제 - When

- SW 생명주기 동안



▼ 어디서 - Where

▼ 책에서는 ...

- 서버/프락시 문제 찾기
- 웹사이트 접근 통계

▼ 추가적으로...

- 개발영역
  - 오류 원인 파악 및 대응에 대한 의사결정 판단의 근거
  - 비정상적인 동작 감지
  - 특정 기능에 대한 사용성 진단
  - 버그 상시 트래킹
- 마케팅 영역
  - 마케팅 채널별 ROI 진단 및 비용 최적화
  - 배너/프로모션/이벤트 효과 측정

        - 유저 Segmentation, Targeting
    - 기획/디자인 영역
        - 시나리오/기능/디자인에 대한 성과 측정 및 개선 (A/B 테스트)
        - 유저 Journey 경로 분석 및 이탈 구간 개선 (UX/UI 최적화)
        - 유저 Persona 구축 (with 리서치) 및 신규 기능 Ideation
    - 기타 영역
        - 영업 및 CS 관련 대응
        - 사업 및 투자 성과 진단

▼ 무엇을 (What)

▼ 책에서는....

- HTTP 메서드
- 클라이언트와 서버의 HTTP 버전
- 요청받은 리소스의 URL
- 응답의 HTTP 상태코드
- 요청과 응답 메시지의 크기 (모든 엔터티 본문을 포함)
- 트랜잭션이 일어난 시간
- Refer와 User-Agent의 헤더 값

▼ 추가적으로...

- **서비스 동작상태**
    - 시스템 로딩
    - HTTP 통신
    - 트랜잭션
    - DB요청
    - 의도를 가진 Exception
- **서비스 장애**

- NPE
- I/O Exception
- 의도하지 않은 Exception

이러한 로그를 읽고 문제 분석 및 해결에 사용!

▼ 어떻게 (How)

- **컨텍스트**를 담아서 **로깅레벨**에 맞게 **저장**한다

▼ 컨텍스트

best 레이아웃 : 실패한 동작 + 실패한 이유 + 후속작업 !!

- Transaction failed (X) → 무슨 트랜잭션? 왜 실패?
  - Transaction 23453 failed : cc number checksum incorrect (O)
  - Transaction 23453번이 실패했구나 : 이유는 이렇구나!
- User operation succeeds (X) → 유저가 어떤 operation을 성공한거??
  - User 54543 successfully **registered e-mail** user@domain.com (O)
  - User 54543이 이메일 등록을 성공적으로 수행했구나!
- java.lang.IndexOutOfBoundsException (X) → 어떤 Index가 범위가 얼마였길래?
  - java.lang.IndexOutOfBoundsException: **index 12** is greater than **collection size 10** (O)
  - collection size가 10 인데 Index 12 를 사용했구나!

▼ 로깅레벨

의도하지 않은 Exception

▼ Fatal

- 네트워크 데몬이 네트워크 소켓과 바인딩을 하지 못했을 때 등..
- 프로그램이 정상적으로 종료되지 않았기 때문에 로그가 남는다고 보장할 수 없음 (따라서 이 레벨은 최대한 사용하지 않는게 b)

▼ Error

- 에러가 발생은 했지만 프로그램이 종료되지 않은 경우
- 외부 api 호출 응답이 error, 시스템 내부에 다른 error 발생시

명확한 의도를 가진 로그들

▼ Warn

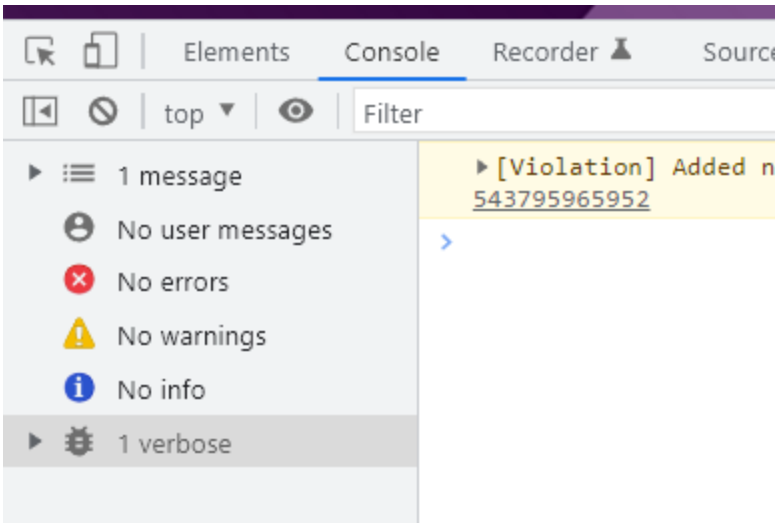
- 메모리용량이 거의 다 차갈때
- db커넥션이 예상보다 오래걸릴때
- 에러가 나기전에 상황을 미리 알 수 있고 대처 가능

▼ Info

- 시나리오 대로 잘 동작하고 있는지
- app상태를 간결하게 보여줌

- Debug
- Trace

▼ 크롬기준 → info/ warn/ error



▼ 저장

- 파일로 저장
- DB에 저장

▼ ⚠️ **로깅시 주의해야 할 점**

- DB 생명주기 & 저장소 용량
- 로그파일
- 개인정보
- 시스템 주요 정보(계정정보/ 시스템보안)

  → 개인정보법 위배

  → 보안적으로 취약

▼ 왜 (Why)

- 많은 이유로 → 어디서(Where) 참고

▼ 2. 로그 포맷

다양한 환경에서 우리는 로그를 찍을 수 있겠으나,

▼ 책에서의 다양한 로그 포맷

- 일반 로그 포맷

  ```
  127.0.0.1 user-identifier frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326
  ```

- 혼합 로그 포맷

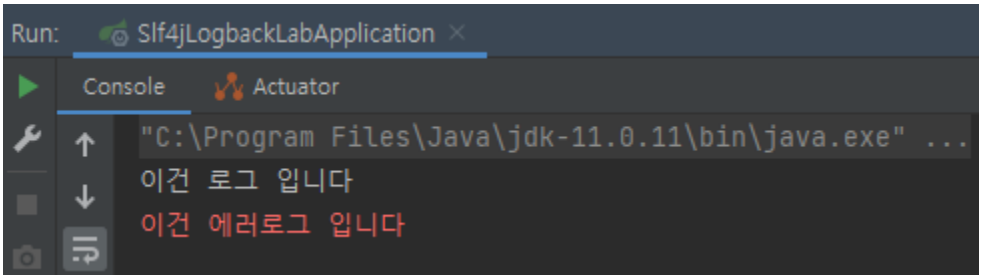
  ```
  127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326 "http://www.example.com/start.html" "Mozilla/4.08 [en] (Win98; I ;Nav)"
  ```

- 넷스케이프 확장1 로그포맷
- 넷스케이프 확장2 로그포맷
- 스퀴드(squid)프락시 로그포맷

▼ **우리가 로그를 찍어야 할 환경에서의 로그를 살펴보자**

- 자바에서의 로그포맷
  - **System 로그**

```
public static void main(String[] args) {
    System.out.println("이건 로그 입니다");
    System.err.println("이건 에러로그 입니다");
```



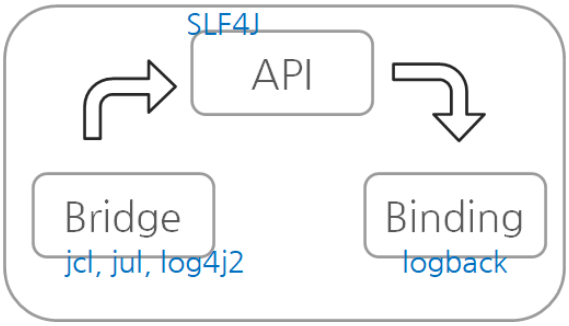
        - but → 실제 운영에서는 권장하지 않는다.
            - 에러별, 상황별로 남기기 어렵고, 무조건 출력되기 때문에
            - 성능적으로도 그냥 시스템 출력보단 로깅 라이브러리가 좋다고 함
    - **여러 로깅 라이브러리**
        - 그럼 우리는 어떤 로깅 라이브러리를 사용해야 할까?

- 로깅 라이브러리 선택권은 애플리케이션 개발자의 것
    - ▼ **로깅 추상화 라이브러리**
        - ▼ **JCL (Jakarta Commons Logging)**
            - 로깅 라이브러리가 아니라 로깅 **추상화** 라이브러리
            - 태초에 스프링은 JCL을 사용해서 로깅을 구현했다.
            - JCL이 로깅 구현체를 찾는 법
                - **클래스패스**에서 구현체 찾아보기
                    - JCL이 구현체를 선택하는 시점이 **런타임**이라 **클래스로더에 의존적**이다
                - 아무것도 못찾으면 기본 구현체 사용
            - JCL의 문제를 한마디로 퉁치면 "가비지 컬렉션"이 제대로 작동하지 않는 치명적인 문제 가 발생

- 문제 해결을 위해서 클래스로더 대신에 컴파일 시점에 구현체를 선택하도록 변경된다. → Slf4j

▼ **Slf4j (Simple Logging Facade for Java)**

- 모듈을 제공하여 컴파일 시점에 로깅 구현체를 결정
- 세가지 모듈로 구성되어 있음



▼ `Bridge`

- 레거시를 위해 사용할 수 있는 모듈
- 레거시 코드에서 로거를 호출하면 로거 호출 slf4j 인터페이스로 연결
- 여러개를 사용해도 ㄱㅊ
- Binding과 같은 로거면 사용 의미가 없음

▼ `API`

- 로깅 인터페이스

▼ `Binding`

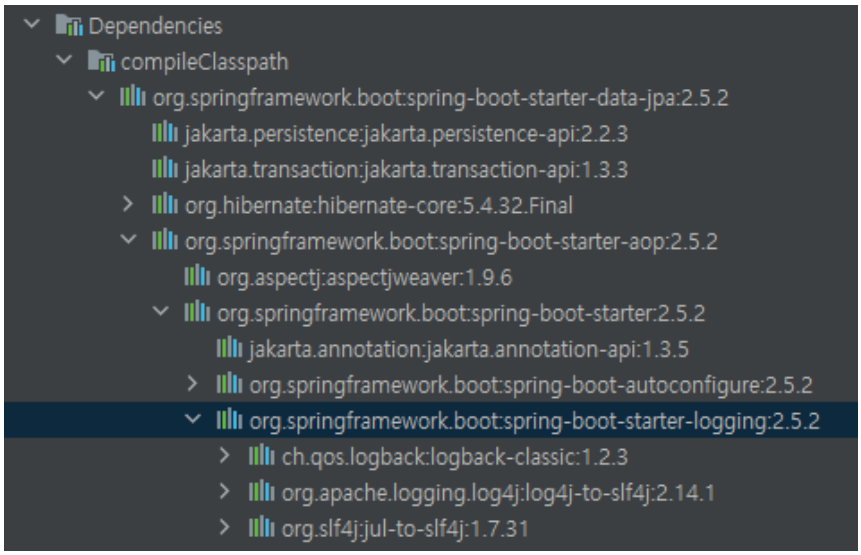
- 실제 구현되어있는 로거로 연결해주는 일을 함
- Binding은 한가지만 추가되어야 함
- 이 예제에서, 사용하는건 slf4j지만 실제 로거는 logback

▼ 사용

- Slf4j & 구현체 라이브러리 의존성 별도 추가
- spring boot start 의존성 추가

- spring-boot-start-web/spring-boot-starter-logging에 구현체

```
dependencies{
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-web'
}
```



- lombok 의존성 추가 후 어노테이션 사용
  - @Slf4j

  ▼ 코드 예시

```java
@RestControllerAdvice
public class Slf4jRestControllerAdvice {

    private Logger logger = LoggerFactory.getLogger(Slf4jRestControllerAdvice.class);

    @ExceptionHandler(Exception.class)
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    public String exception(Exception exception) {
        logger.error("Internal Server Exception: {}", exception.getMessage());
        return exception.getMessage();
    }

    @ExceptionHandler(IllegalArgumentException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public String illegalArgumentException(IllegalArgumentException exception) {
        logger.error("IllegalArgumentException: {}", exception.getMessage());
        return exception.getMessage();
    }

    @ExceptionHandler(CommonException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public String notFoundException(CommonException exception) {
        logger.error("CommonException: {}", exception.getMessage());
        return exception.getMessage();
    }
}
```

```java
@RestControllerAdvice
@Slf4j
public class Slf4jRestControllerAdvice {

    @ExceptionHandler(Exception.class)
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    public String exception(Exception exception) {
        log.error("Internal Server Exception : {}", exception.getMessage());
        return exception.getMessage();
    }

    @ExceptionHandler(IllegalArgumentException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public String illegalArgumentException(IllegalArgumentException exception) {
        log.error("IllegalArgumentException: {}", exception.getMessage());
        return exception.getMessage();
    }

    @ExceptionHandler(CommonException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public String notFoundException(CommonException exception) {
        log.error("CommonException: {}", exception.getMessage());
        return exception.getMessage();
    }
}
```

▼ 구현체

- ~~Log4j~~

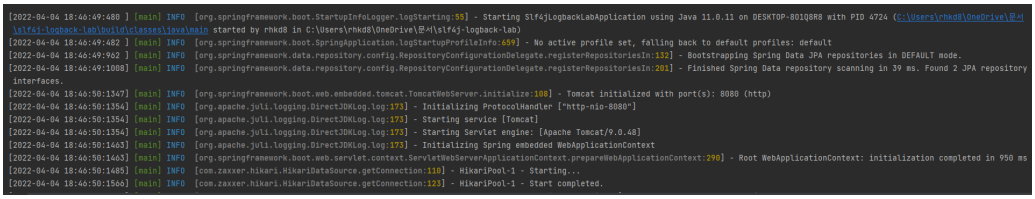
▼ Logback

- 개발 환경에서 콘솔에서 출력되는 내용을 조절하는 방법으로 적당하다
    - `application.yml` 에서 로깅 수준을 설정 가능
        - 실제 제품에 사용하기엔 한계가 있고 세부적인 설정이 불편
    - `logback-spring.xml` 을 별도로 만들어서 로그를 관리

- **스프링 부트에서의 로그 포맷**

  기본적으로 스프링 부트에서는 Slf4j와 Logback을 채택하고 있음

  ▼ 예제




```
<?xml version="1.0" encoding="UTF-8"?>

<configuration> -- 설정 시작
    <timestamp key="BY_DATE" datePattern="yyyy-MM-dd"/>

    <property name="LOG_PATTERN"
              value="[%d{yyyy-MM-dd HH:mm:ss}:%-4relative] %green([%threa
d]) %highlight(%-5level) %boldWhite([%C.%M:%yellow(%L)]) - %msg%n"/>

    <springProfile name="!prod">
        <include resource="console-appender.xml"/>

        <root level="INFO"> -- Info 레벨의 로그를
            <appender-ref ref="CONSOLE"/> -- Console에 추가한다
        </root>
    </springProfile>

    <springProfile name="prod">
        <include resource="file-info-appender.xml"/>
        <include resource="file-warn-appender.xml"/>
        <include resource="file-error-appender.xml"/>

        <root level="INFO">
            <appender-ref ref="FILE-INFO"/>
            <appender-ref ref="FILE-WARN"/>
            <appender-ref ref="FILE-ERROR"/>
```

```
        </root>
    </springProfile>
</configuration>
```

로깅 패턴

형식	설명
%-5level	로그레벨, -5(출력 고정폭 5글자)
%d{날짜 형식 포맷}	로그 기록시간 %d{yyyy-MM-dd HH:mm:ss}
%relative	밀리초까지 보여주는 옵션
%thread	현재 Thread 명
%C	로그를 발생시킨 클래스 명
%L	로그를 발생시킨 라인
%M	로깅을 발생시킨 메소드 명
%F	로깅을 발생시킨 파일 명
%msg	로그 메시지
%n	줄바꿈 (new line)

이 외에도 여러 가지가 있음 (http://logback.qos.ch/manual/layouts.html)

▼ 3. 적중 계량하기

클라이언트ㅡ서버 사이에는 캐시가 있어서,

많은 요청이 서버까지 오지않고 캐시되어 있는 리소스로 처리되고 끝난다.

결국 클라이언트가 콘텐츠에 접근했다는 기록을 남기지 못하고 **로그파일에 누락이 발생**된다.

▼ 로그 파일 누락 문제 해결 방안

- 중요 페이지의 캐시를 파기시킨다
  - 요청이 원서버로 가게되어 리소스에 대한 접근을 로깅할 수 있다.
  - 요청에 대한 응답속도가 느려지고, 원서버와 네트워크 부하가 가중된다 ⚠️
- **적중계량(Hit Metering) 규약**

> 캐시가 정기적으로 캐시 접근 통계를 원 서버에 보고한다

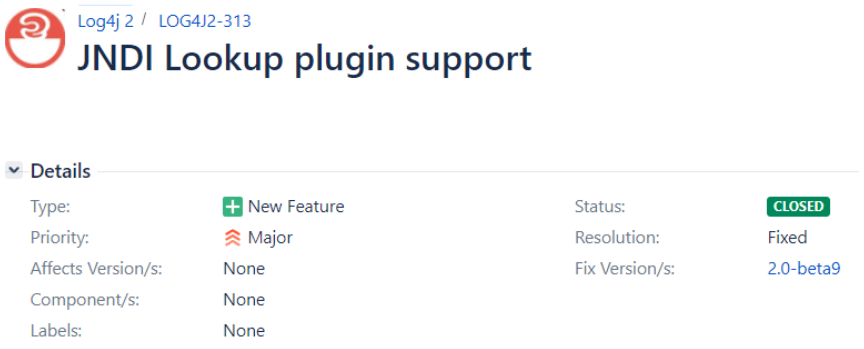
- HTTP의 확장
- 위 문제에 대한 해결책으로 제시

▼ 4. 개인정보 보호에 대해

- 로깅은 관리자와 개발자에게 매우 유용한 도구
- 사용자들의 인지나 허가가 없으면 로깅이 사생활 침해가 된다는 것을 유념해야 함
  - 최종 사용자의 개인 정보를 보호하는데 신경을 많이 쓰자!

▼ Log4j

- 취약점이 만들어진 경위



  - JNDI 조회 플러그인 지원 이슈
  - Lookup 플러그인에 jndi을 추가하는게 이 이슈의 목적
    - 추가된 jndi를 통해 취약점이 발생

JNDI(Java Naming and Directory Interface)

  - Java 프로그램이 디렉토리를 통해 데이터(Java 객체 형태)를 찾을 수 있도록 하는 디렉토리 서비스
  - Java 프로그램들은 앞서 말한 JNDI와 LDAP를 통해 Java 객체를 찾을 수 있다.
    - ex) URL `ldap://localhost:389/o=JNDITutorial`을 접속한다면

- LDAP 서버에서 JNDITutorial 객체를 찾을 수 있는 것

이런 문법은 로그가 기록될 때도 사용이 가능 했고, 결국 해커가 로그에 기록되는 곳을 찾아 ${jndi:sndi:snd://example.com/a}과 같은 값을 추가하기만 하면 취약점을 이용할 수 있는 것이다. 이 값을 넣는 방법은 User-Agent와 같은 일반적인 HTTP 헤더일 수도 있고 여러가지 방법이 있다.

- 해당 버그는 2021년 11월 24일, 알리바바 클라우드 보안팀의 Chen Zhaojun가 발견했다.
- 2021년 11월 30일, 해당 문제를 수정하는 PR이 log4j 깃허브에 올라왔다

- 12/10 04:15 KST,[10] PaperMC가 자사 Discord 서버를 이용하여 긴급 공지를 전송함으로써 이슈가 크게 번지기 시작했다. 내용의 요지는 **이 글을 보는 즉시 긴급 패치된 파일로 업데이트하라**는 것.[11] 이후 이 사건에 대한 내용이 밝혀지기 시작했다.
- 12/10 09:40 KST,[12] 오픈소스 코드 저장소로 알려진 GitHub 에서 운영하는 GitHub Advisory Database[13]에 CVE-2021-44228 취약점이 게재되었다.
- 이후 12:44 KST,[14] 마인크래프트의 기술 책임자가 본인 트위터를 통해 **"마인크래프트에 영향을 미치는 중요한 보안 문제가 발견되어 수정하였다."**고 발표하였다.

- ArsTechnica는 특집 기사를 통해 이 사건을 보도하였으며, **마인크래프트는 시작일 뿐이다** 라는 점을 분명히 했다.

문제가 된 것은 Log4j 라이브러리의 취약점(명령어를 실행시킬 수 있다는) 때문인데,

이것은 현재 대부분의 자바 웹프로그래밍 서버에서 사용되고 있는 라이브러리이기 때문에 Apple이나 Twitter 등등에도 이 버그가 영향을 미칠 수 있다고 AP통신이 보도했다.

또한 이 이슈가 문제가 되는 것은 **"서버에 로그인한 것 만으로 해커가 사용자의 컴퓨터를 사실상 원격 조종할 수 있는 것[16]이기 때문이다."** 라고 경고했다.

일본 의 정보보도매체인 ITMedia는 iCloud 와 Steam 에서도 이 취약점이 이용 가능하다고 보도했다.

- 가장 안전한 방법은 log4j 를 2.17.1 이상으로 올리는 것이다. (Java 8 필요)
- Java 7이라면 2.12.4 버전을 사용하면 된다.
- Java 6이라면 2.3.2 버전을 사용하면 된다.
- logback 등 다른 로깅 모듈로의 교체를 검토한다.
- 다른 패키지가 log4j-core에 의존하고 있을 가능성이 있으므로 실제 포함 여부를 의존성 트리 구조를 통해 확인해야 한다

▼ 출처

https://ideveloper2.dev/blog/2020-02-23--로깅과-사용-추적/

https://techblog.woowahan.com/2536/

https://livenow14.tistory.com/63

https://velog.io/@stella6767/logback-spring.xml-설정방법

[무늬의 로깅] https://youtu.be/MxxeKXydn4A

[검프의 로깅1] https://youtu.be/1MD5xbwznlI

[검프의 로깅2] https://youtu.be/JqZzy7RyudI

[최악의 보안사태 log4j] https://youtu.be/kwS3twdVsko

[log4j 나무위키] https://namu.wiki/w/Log4j 보안 취약점 사태

[스프링부트와 로깅] https://www.slideshare.net/whiteship/ss-47273947