# Analytics > Log & Crash Search > Log4J SDK 사용 가이드

Log & Crash Log4J SDK는 Log & Crash Search 수집 서버에 로그를 보내는 기능을 제공합니다. Log & Crash Log4J SDK 특·장점은 다음과 같습니다.

- 로그를 수집 서버로 보냅니다.
- Log & Crash Search 에서 전송된 로그를 조회 및 검색이 가능합니다.
- 멀티 쓰레딩 환경에서 동작합니다.

## 지원 환경

- Log4J 1.2.x (1.2.14, 1.2.16, 1.2.17)

## 다운로드

TOAST Document에서 Log4J SDK를 받을 수 있습니다.

```
[DOCUMENTS] > [Download] > [Analytics > Log & Crash Search] > [Log4J SDK] 클릭
```

## 설치

### 구성

Log4J SDK는 다음과 같이 구성되어 있습니다.

```
docs/       ; Log4J SDK 문서
lib/        ; Log4J 라이브러리
sample/     ; Log4J 샘플
```

### SDK 샘플

같이 제공되는 sample/log4j/에 대해 설명합니다.

1.Eclipse를 실행하고 메뉴에서 File - Import - Maven - Existing Maven Projects 를 실행하여 sample/log4j/를 불러옵니다. 2.src/test/resources/log4j.xml 파일을 열어 발급받은 앱키와 버전을 수정하고, 필요하면 수집 서버 주소를 변경합니다.

```
<param name="collectorUrl" value="https://api-logncrash.cloud.toast.com" />
<param name="appKey" value="__app_key__" />
<param name="version" value="1.0.0" />
```

3.Eclipse 메뉴에서 Project - Properties - Java Build Path - Libraries 를 선택하여 toast-logncrash-log4j-sdk-.jar 를 추가합니다. 4.Eclipse 메뉴에서 Run - Run As - JUnit Test를 선택하여 실행합니다.

# 사용 예

1.Log4J SDK 라이브러리를 Project에 추가합니다.

- 예를 들어 Eclipse 메뉴 Project - Properties - Java Build Path - Libraries 를 선택하여 toast-logncrash-log4j-sdk-.jar 를 추가합니다.

2.Maven을 사용하는 경우, pom.xml에 dependency를 추가합니다.

```
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.17</version>
</dependency>
<dependency>
  <groupId>commons-lang</groupId>
  <artifactId>commons-lang</artifactId>
  <version>2.5</version>
</dependency>
<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpclient</artifactId>
  <version>4.2.6</version>
</dependency>
<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpcore</artifactId>
  <version>4.2.4</version>
</dependency>
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>servlet-api</artifactId>
  <version>2.4</version>
</dependency>
<dependency>
  <groupId>org.json</groupId>
  <artifactId>json</artifactId>
  <version>20090211</version>
</dependency>
```

- SLF4J를 사용하시는 경우 다음 dependency를 추가합니다.

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>1.7.2</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.7.2</version>
</dependency>
```

3.Maven을 사용하지 않는 경우 다음 라이브러리들을 별도로 다운로드를 받아 class path에 추가합니다.

```
log4j, 1.2.17
commons-lang, 2.5
httpclient, 4.2.6
httpcore, 4.2.4
servlet-api, 2.4
json, 20090211
```

4.Appender 설정과 구성을 위해서 log4j.xml을 작성합니다.

- 전체 구성은 sample/log4j/src/test/resources/log4j.xml 를 참고해 주세요.
- collectorUrl, appKey에는 반드시 **수집서버 주소**, **발급받은 앱키** 를 사용해야 합니다.

```
<appender name="logncrash-http"
class="com.toast.java.logncrash.log4j.LogNCrashHttpAppender">
     <param name="collectorUrl" value="https://api-logncrash.cloud.toast.com"
/>
     <!-- v2 -->
     <!--     -->
     <param name="appKey" value="__app_key__" />
     <param name="version" value="1.0.0" />
     <param name="logSource" value="http-log4j" />
     <param name="logType" value="log" />
     <param name="Threshold" value="ALL" />
     <param name="errorCodeType" value="default" />
     <param name="enable" value="true" />
     <param name="debug" value="false" />
</appender>
...
<root>
 <appender-ref ref="logncrash-http" />
</root>
```

5.properties로 설정하기 위해서는 log4j.properties을 작성합니다.

```
log4j.rootLogger=ALL, STDOUT, logncrash-http
```

```
log4j.appender.STDOUT=org.apache.log4j.ConsoleAppender
log4j.appender.STDOUT.Threshold=DEBUG
log4j.appender.STDOUT.layout=org.apache.log4j.PatternLayout
log4j.appender.STDOUT.layout.ConversionPattern=%m%n
log4j.appender.logncrash-
http=com.toast.java.logncrash.log4j.LogNCrashHttpAppender
log4j.appender.logncrash-http.collectorUrl=https://api-
logncrash.cloud.toast.com
log4j.appender.logncrash-http.appKey=__appkey__
log4j.appender.logncrash-http.version=1.0.0
log4j.appender.logncrash-http.logSource=http-log4j
log4j.appender.logncrash-http.logType=log
log4j.appender.logncrash-http.Threshold=ALL
log4j.appender.logncrash-http.errorCodeType=default
log4j.appender.logncrash-http.enable=true
log4j.appender.logncrash-http.debug=false
```

6.Java에서 다음과 같이 사용합니다.

```
        ...
private static Logger logger = Logger.getLogger(LogNCrashLog4jSample.class);
...
// Custom Message
MDC.put("custommessage", "custom message");
logger.debug("Log4j SDK Debug Message");
try {
        String npe = null;
        npe.toString();
} catch(Exception e) {
        logger.error("Log4J SDK Exception", e);
}
```

# API List

## log4j.xml 설정 항목

- collectorUrl: 수집 서버 주소 HTTP: https://api-logncrash.cloud.toast.com
- appKey: 프로젝트 앱키, 필수
- version: 프로젝트 버전, 기본값 "1.0.0"
- logSource: 로그 소스, 기본값 "http-log4j"
- logType: 로그 타입, 기본값 "log"
- Threshold: 전송할 로그 레벨 지정, 기본값 "ALL"
- enable: Appender 사용 여부 설정, 기본값 "true"
- debug: 디버그 사용 여부 설정, 기본값 "false"
- errorCodeType: 에러 코드 타입 설정, 기본값 "default" default: Exception 정보를 사용 mdc: Log4j MDC의 errorCode 항목값을 설정해서 사용한다.

# 제약 사항

- 현재 **log4j 2.0** 버전에서는 동작하지 않습니다. log4j 1.3은 alpha8만 작동하지만 log4j 1.2로 마이그레이션을 권장합니다. 권장 버전은 log4j 1.2.14, 1.2.16, 1.2.17입니다.
- 오류 데이터가 한꺼번에 많이 발생하는 경우 logncrash-async appender의 bufferSize가 작으면 log4j 자체에서 처리시 지연이 발생할 수 있으므로, bufferSize 조절이 필요합니다.

# FAQ

## blocking을 false로 사용하려면?

log4j.xml에서 다음과 같이 logncrash-async의 class명을 변경한다.

```
<!-- define logncrash-async appender -->
<appender name="logncrash-async"
class="com.toast.java.logncrash.log4j.LogNCrashAsyncAppender">
    <param name="Threshold" value="ALL" />
    <param name="blocking" value="false" />
    <param name="locationInfo" value="false" />
    <param name="bufferSize" value="2048" />
    <appender-ref ref="logncrash-http" />
</appender>
```

## batch program(project)에서 logncrash client를 사용하려면?

Quartz 등을 사용해서 데몬 형태로 구동하는 batch project에는 적용되지 않습니다. batch 프로그램의 마지막에 몇초간 대기하는 코드를 추가합니다.

```
try {
    Thread.sleep(3000L);
} catch (InterruptedException ignore){}
```

logncrash-async appender의 경우 org.apache.log4j.AsyncAppender를 사용하고 있습니다. AsyncAppender 안에서 로그를 기록하는 별도의 데몬 스레드가 생성되어 비동기로 로그를 전달하게 되어 있습니다. Java batch program에서는 main thread가 바로 종료되기 때문에 AsyncAppender 데몬 스레드가 생성되어 로그를 보내기 전에 batch 애플리케이션이 종료됩니다. 데몬 스레드와 상관없이 살아 있는 일반 스레드가 없을 경우에 JVM은 바로 종료됩니다. 다른 방법은 아래처럼 batch용 log4j.xml을 별도로 사용하는 것입니다. logger에서 appender logncrash를 바로 사용하도록 log4j.xml을 수정합니다. 이 경우 logging이 동기모드로 작동되기 때문에 에러 발생시 에러 수집 서버 호출을 위해 delay가 발생됩니다. web project에서는 이 방법을 사용하지 않도록 합니다.

```
<!-- // define loggers // -->
<logger name="com" additivity="false">
    <level value="INFO"/>
    <appender-ref ref="STDOUT"/>
    <appender-ref ref="logncrash"/>
</logger>
<!-- // define root // -->
<root>
    <level value="WARN"/>
    <appender-ref ref="STDOUT"/>
    <appender-ref ref="logncrash"/>
</root>
```

## Java stack trace를 log4j(Log & Crash Search 포함)에 로깅하려면?

log4j를 이용하여 stack trace를 출력하려면 log.error(e.getMessage(), e); 형태를 사용합니다. log.error(e);의 경우는 stack trace가 출력되지 않습니다.

```
String[] aa = null;
try {
    aa[0] = "111";
} catch (NullPointerException e) {
    log.error(e); //stacktrace 출력 안됨.
    log.error(e.getMessage(), e); ///stacktrace 출력
}
```

## log4j(Log & Crash Search 포함) logging으로 인한 성능 저하를 최소화 하려면?

log4j.xml의 logger 설정에서 name과 level을 사용하여 filtering을 최대화합니다. 아래처럼 logger 설정에서 com이나 org를 DEBUG level로 설정하게 되면 logger에서 많은 LoggingEvent(log4j)가 불필요하게 생성됩니다. Appender에서 Threshold가 ERROR로 설정되어 있어 실제 로그 전송은 되지 않지만 일단 logger에서 LoggingEvent가 생성이 되어 appender에 전달이 됩니다.

[성능이 저하되는 설정(개발용으로만 사용)]

```
 <!-- // define loggers // -->
 <logger name="com" additivity="false">
     <level value="DEBUG"/>
     <appender-ref ref="STDOUT"/>
     <appender-ref ref="logncrash-async"/>
 </logger>

 <!-- // define loggers // -->
 <logger name="org" additivity="false">
     <level value="DEBUG"/>
     <appender-ref ref="STDOUT"/>
```

```
    <appender-ref ref="logncrash-async"/>
</logger>

<!-- // define root // -->
<root>
    <level value="WARN"/>
    <appender-ref ref="STDOUT"/>
    <appender-ref ref="logncrash-async"/>
</root>
```

[성능이 고려된 설정(운영용으로 사용)]

```
<!-- // define loggers // -->
<logger name="com" additivity="false">
    <level value="ERROR"/>
    <appender-ref ref="STDOUT"/>
    <appender-ref ref="logncrash-async"/>
</logger>

<!-- // define root // -->
<root>
    <level value="WARN"/>
    <appender-ref ref="STDOUT"/>
    <appender-ref ref="logncrash-async"/>
</root>
```

## WAS 에서 사용시 안정적인 종료를 하려면?

에러로그가 전송중인 상황에서 WAS(Tomcat 등)가 종료되는 경우에는, 다음과 같은 Exception이 발생하며 WAS가 정상적으로 종료되지 않을 때가 있습니다.

```
Exception in thread "pool-12-thread-1" java.lang.NullPointerException
at_external.org.apache.mina.common.AbstractPollingIoProcessor$Worker.run(AbstractPollingIoProcessor.java:740)
at_external.org.apache.mina.util.NamePreservingRunnable.run(NamePreservingRunnable.java:51)
at_java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886)
at_java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908)
at_java.lang.Thread.run(Thread.java:619)
```

로그가 전송중에 WAS가 종료되는 경우에 해당 Exception이 발생합니다. 이러한 현상을 방지하기 위해서는 WAS 종료시에 LogManager.shutdown() 메소드를 호출하여 logncrash appender를 close하면 안정적으로 종료가 가능합니다. org.springframework.web.util.Log4jConfigListener를 사용하는 경우에는 WAS 종료시에 Log4jConfigListener가 LogManager.shutdown() 메소드를 호출해주기 때문에 추가적인 설정없이 안정적으로 종료가 가능합니다. Log4jConfigListener를 사용하지 않는 경우를 위해서 logncrash-appender에서는 com.toast.java.logncrash.log4j.Log4jShutdownListener 를 제공하고 있습니다. web.xml에 다음과 같은 설

정을 추가하면 WAS 종료시에 에러로그 전송이 일어나도 안정적인 종료가 가능합니다.

```
<listener>
    <listener-class>
        com.toast.java.logncrash.log4j.Log4jShutdownListener
    </listener-class>
</listener>
```