Analytics > Log & Crash Search > Android SDK 사용 가이드

> [Deprecated] Log & Crash Android SDK 버전은 더 이상 지원되지 않습니다. TOAST SDK를 이용해 주시기 바랍니다.

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

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

지원 환경

- Android 2.3.3, API Level 10 이상

다운로드

TOAST Document에서 Android SDK를 받을 수 있습니다.

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

설치

구성

Android SDK는 다음과 같이 구성되어 있습니다.

```
docs/       ; Android SDK 문서
libs/       ; Android SDK 라이브러리
sample/     ; Android SDK 샘플
```

SDK 샘플

같이 제공되는 sample/에 대해 설명합니다.

1. libs/를 sample/libs/로 복사합니다.
2. Eclipse를 열어서 File - New - Android - Android Project from Existing Code 로 sample/을 열어 줍니다.
    - AndroidStudio에서는 File - New - Import Project...를 사용합니다.

3. ToastLogSample.java를 열어서 onCreate()에서 앱키, 수집서버 주소를 수정해 줍니다. 버전, 로그 소스, 로그 타입 등을 수정하면 검색에 도움이 됩니다.
4. 실행합니다.
5. Initialize 버튼을 눌러서 시작합니다.
6. debug, info, warn, error, fatal 버튼을 눌러 로그를 전송합니다.
7. send crash, crash 버튼을 눌러 크래시 로그를 전송합니다. send crash 버튼은 크래시 로그만 보내는 기능입니다. crash 버튼은 강제로 crash를 발생시켜 앱 종료와 동시에 크래시 로그를 전송합니다.

사용 예

1.Android SDK의 libs/를 해당 프로젝트 libs/에 복사합니다. 2.AndroidManifest.xml 파일에 권한을 추가합니다.

- 네트워크 상태, 디바이스정보, 통신사 등의 정보를 가져오기 위하여 권한이 필요합니다.

```
<!-- 로그 전송을 위한 인터넷 접근 권한 (필수) -->
<uses-permission android:name="android.permission.INTERNET" />

<!-- Platform, Carrier등 폰의 정보에 접근하기 위한 권한 (필수) -->
<uses-permission android:name="android.permission.READ_PHONE_STATE" />

<!-- 네트워크 상태에 접근하기 위한 권한 (필수) -->
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

<!-- 네트워크 접속이 안되는 경우 파일로 로그를 저장하기 위한 권한 (옵션) -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />

<application
      ......
```

3.제공되는 ToastLog class를 사용하여 로그를 전송합니다.

- ToastLog.initialize()를 실행하여 초기화를 해줍니다.
- debug()/info()/warn()/error()/fatal() 함수를 사용하여 해당 logLevel 로그를 수집 서버로 보냅니다.
- 앱 크래시가 발생하면 크래시 로그가 수집 서버로 전송됩니다.

```
.....
public class MainActivity extends Activity {
  .....
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    .....
    if (ToastLog.initialize(getApplication(), "__수집서버_주소__", 0, "__앱키__",
"__버전__") {
      // 초기화 성공
      ToastLog.info("Init Success")
    } else {
```

```
      // 초기화 실패
    }
    .....
  }
}
.....
```

API List

com.toast.android.logncrash.ToastLog class에서 제공하는 기능들을 설명합니다.

초기화

```
public static final String DEFAULT_APP_KEY = "__app_key__";
public static final String DEFAULT_VERSION = "1.0.0";
public static final String DEFAULT_COLLECTOR_ADDR = "https://api-
logncrash.cloud.toast.com";
public static final int DEFAULT_COLLECTOR_PORT = 0;
public static final String DEFAULT_LOG_SOURCE = "logncrash-logSource";
public static final String DEFAULT_LOG_TYPE = "logncrash-logType";

public static boolean initialize(Application application, String
collectorAddr, int collectorPort, String appKey, String version, String
userId);

public static boolean initialize(Application application, String
collectorAddr, int collectorPort, String appKey, String version);

public static boolean initialize(Application application, String
collectorAddr, int collectorPort, String appKey, String version, boolean
syncStart);
```

- ToastLog를 초기화합니다.
- ToastLog 기능이 제대로 동작하기 위해서는 반드시 호출되어야 합니다.
- 파라미터
  - application: Android Application 정보. getApplication() 반환값을 넣어 줍니다.
  - collectorAddr: 수집서버 주소
    - HTTP 수집 서버: https://api-logncrash.cloud.toast.com
  - collectorPort: 수집서버의 포트 정보, 0으로 지정하면 각 protocol 기본 포트가 사용됩니다.
    - HTTP: 80
  - appKey: 앱키
  - version: 앱 버전
  - userId; 사용자 아이디

    - syncStart: true인 경우 발생한 로그들은 startSendThread가 호출되기 전까지 서버에 전송하지 않고, 큐에 저장합니다. 단 Crash가 발생한 경우 ThreadLock을 해제하고 로그를 전송합니다.
- 반환값
    - 초기화 성공시 true
    - 실패시 false

SendThread 잠금 해제

```
    (void) startSendThread;
```

- SendThread의 잠금 상태를 해제합니다.

초기화 주의사항

- Application onCreate에서 초기화를 실행하는 경우
- Application onCreate는 리시버, 서비스, 액티비티가 생성될 때 호출됨으로 의도하지 않은 호출이 발생할 수 있습니다.

```
public static boolean isInitialized()
```

- ToastLog 초기화 여부를 반환합니다.
- 반환값
    - 초기화 되었으면 true
    - 아니면 false

로그 보내기

```
public static void fatal(String message, Throwable t)
public static void error(String message, Throwable t)
public static void warn(String message, Throwable t)
public static void info(String message, Throwable t)
public static void debug(String message, Throwable t)

public static void fatal(String message)
public static void error(String message)
public static void warn(String message)
public static void info(String message)
public static void debug(String message)
```

- 지정된 로그 레벨로 수집서버에 로그를 보냅니다.
- 파라미터
    - message: 로그 메시지. null이나 ""를 쓰면 기본 메시지가 전송됩니다.
    - Throwable t: 수집된 에러를 같이 전송합니다. null을 사용할 수 있습니다.
        - 에러를 같이 전송하면 핸들드 로그로 분류됩니다.

```
public static void crash(String message, Throwable t)
```

- 실행시 발생한 Exception을 수집서버로 보냅니다.
- 파라미터
  - message: 로그 메시지. null이나 ""을 쓰면 기본 메시지가 전송됩니다.
  - Throwable t: 수집된 에러를 같이 전송합니다.
    - 에러를 같이 전송하면 크래시 로그로 분류됩니다.

errorCode, location 등의 정보를 Throwable에서 조사하도록 향상되면서 다음 Method들이 **Deprecated** 되었습니다.

```
public static void fatal(String errorCode, String message, String location)
public static void error(String errorCode, String message, String location)
public static void warn(String errorCode, String message, String location)
public static void info(String errorCode, String message, String location)
public static void debug(String errorCode, String message, String location)

public static void fatal(String errorCode, String message)
public static void error(String errorCode, String message)
public static void warn(String errorCode, String message)
public static void info(String errorCode, String message)
public static void debug(String errorCode, String message)

public static void crash(Throwable throwable, String errorCode, String
message, String location)
public static void crash(Throwable throwable, String errorCode, String
message)
```

커스텀 키 지정하기

```
public static void addCustomField(String key, String value)

public static void removeCustomField(String key)

public static void clearCustomFields()
```

- 커스텀 키를 추가, 삭제, 전부 삭제 기능을 제공합니다.
- 커스텀 키는 대소문자로 시작하고 대소문자, 숫자, '-', '*'만 사용하실수 있습니다. ([A-Za-z][A-Za-z0-9-]*)*
- 커스텀 키는 대소문자 관계없이 다음 이름은 사용하실 수 없습니다.
  - "projectName", "projectVersion", "body"
  - "logSource", "logType", "host", "sendTime", "logLevel"
  - "DmpData", "Platform", "NeloSDK", "Exception", "Location", "Cause"
  - "SessionID", "UserID"
  - "Carrier", "CountyCode", "DeviceModel", "Locale", "NetworkType", "Rooted"

기본 설정 관리

```
public static String getAppKey()
public static String getVersion()
public static String getCollectorAddr()
public static int getCollectorPort()
```

- 초기화에 사용된 앱키, 버전, 수집서버 주소, 수집서버 포트를 구합니다.

```
public static String getLogSource()
public static void setLogSource(String logSource)
```

- 로그소스를 구하거나 새로 지정합니다.

```
public static String getLogType()
public static void setLogType(String logType)
```

- 로그 타입을 구하거나 새로 지정합니다.

중복 제거 모드 설정

2.4.0 이상 SDK 부터 일반 로그에 중복 제거 로직이 적용되었습니다.

중복 로그 기능이 켜져있는 경우 body와 logLevel의 내용이 같은 로그가 발생하면 전송하지 않습니다.

```
public static void setDuplicate(bool enable)
```

true:(Default값) 중복 제거 로직 활성화
false: 중복 제거 로직 비활성화

SDK 샘플을 이용한 Proguard 테스트

Android에서 제공하는 Proguard를 통해서 코드 난독화를 테스트하는 방법을 설명합니다. Proguard를 적용하기 위해서는 Release로 프로젝트를 생성해야 합니다. 이에 필요한 키스토어, Proguard 설정 등이 sample/에 포함되어 있습니다.

Eclipse를 사용하여 Proguard 테스트

1. libs/를 sample/libs/로 복사합니다.
2. Eclipse를 구동해서 해당 프로젝트를 선택하고 메뉴에서 File - Export... 를 선택합니다.
3. Export 창이 나타나면 Android - Export Android Application 을 선택합니다.
4. 프로젝트 이름을 확인하고 Next를 클릭하고, Keystore 선택에서 mykey.keystore를 선택해 줍니다. Keystore 암호 'abcdefg', Alias 이름 'mykey', Alias 암호 'abcdefg'로 지정되어 있습니다. 자세한 내용은 여기 을 참고해 주세요.
5. 저장할 경로를 지정해 줍니다.
6. 저장된 .apk 파일을 'adb install <파일이름>'으로 장치에 설치합니다.
7. 실행하고 Initialize 버튼을 누른 후, crash 버튼을 눌러 크래시 로그를 발생시킵니다. sample 에서는

ToastLogSample.clickCrash() 멤버 펑션이 난독화 되어 있어서 정상적으로 나오지 않습니다.

정상적으로 Proguard가 구동된 상태에서는 proguard/mapping.txt가 생성이 됩니다.

1. Console - Analytics - Log & Crash - Settings - 심볼 파일 에서 mapping.txt를 올립니다. 이때 프로젝트 버전은 반드시 동일하게 입력해야 합니다.
2. 앱에서 크래시 로그를 보냅니다. Proguard가 풀린 로그는 "DmpData" 필드에 "보기"를 클릭하시면 확인할 수 있습니다.
3. ToastLogSample.clickCrash() 이름이 제대로 나오는지 확인합니다.

Ant 빌드를 사용하여 Proguard 테스트

1. libs/를 sample/libs/로 복사합니다.
2. build.xml이 없는 경우 android update project -p . -n AndroidSDKSample 명령어로 생성해 줍니다.
3. ant clean release로 빌드해 줍니다. 결과물은 bin/ 안에 AndroidSDKSampe-release.apk로 저장됩니다.

Ant를 이용하여 Release 빌드를 하는 경우 Eclipse Release 빌드와는 달리 mapping.txt 위치가 bin/proguard/mapping.txt 입니다.

AndroidStudio를 사용하여 Proguard 테스트

1. libs/를 sample/libs/로 복사합니다.
2. AndroidStudio를 구동하여 메뉴에서 File - New - Import Project... 를 실행하여 새로운 위치에 AndroidStudio 프로젝트를 생성합니다.
3. 기존 소스에서 mykey.keystore 를 새로 생성된 프로젝트로 복사합니다.
4. Build - Generate Signed APK... 를 실행합니다. 첫번째 페이지에서 Module을 확인한 후 Next를 클릭합니다. 두번째 페이지에서 Key store path를 mykey.keystore로 지정해 주고, Keystore 암호 'abcdefg', Alias 이름 'mykey', Alias 암호 'abcdefg'로 지정한 후 Next를 클릭합니다. 세번째 페이지에서 APK Destination Folder를 확인하고 Finish를 클릭합니다.
5. 생성된 .apk를 설치합니다.

AndroidStudio를 이용하여 Release 빌드를 하면 mapping.txt 위치가 app/build/outputs/mapping/release/mapping.txt입니다.

JNI 적용 가이드

Android NDK를 이용하여 JNI를 사용시 Android SDK를 활용하는 방법에 대하여 설명합니다. Log & Crash Android SDK에서는 Java상에서 Exception을 Catch 할 수 있는 상황에서만 정상적으로 동작합니다. 그러기 위해서 작성한 Native Code에서 에러 발생시 에러를 전달하는 클래스를 생성하여야 합니다. 예를 들어 아래와 같이 getString 함수가 실행되는 과정에서 에러가 발생한다면, try/catch 구문을 이용하여 Error을 Catch한 후 Java 코드로 예외를 생성하여 Throw해주면, Java에서 해당 에러를 잡아 로그로 전송합니다.

JNI code

```
.....
// Native Code
void throwArithmeticError(JNIEnv *env, char *message) {
  jclass exClass;
  char *className = "java/lang/ArithmeticException: / by zero";
  exClass = (*env)->FindClass(env, className);
```

```
  if (exClass == NULL) {
    return throwNoClassDefError(env, className);
  }
  (*env)->ThrowNew(env, exClass, message);
}

JNIEXPORT JNICALL jstring Java_com_example_jnitest_HelloJNI_getString(JNIEnv
*env, jobject object) {
  if (...) {
    return str;
  } else {
    //On Error
    throwArithmeticError(env, "THIS IS JNI EXCEPTION");
  }
}
.....

**Java Code**

.....
// Java Code
try{
  mTextView.setText( new HelloJNI().getString() );
} catch(Throwable e){
  ToastLog.crash(e, "JNI ERROR", "JNI ERROR MESSAGE - Throw Error");
}
.....
```

Segment falut와 같은 System Crash를 잡기 위해서는 Native Code에서 Signal을 잡는 구문을 추가하면 됩니다. 아래의 예제는 여기를 참고하여 작성되었습니다.

Native Code

```
.....
//Native Code
JNIEXPORT jint android_sigaction(int signal, siginfo_t *info, void *reserved) {
  old_sa[signal].sa_handler(signal);
  return throwNoClassDefError(env, "THIS android_sigaction ACTION");
}

JNIEXPORT jint JNICALL JNI_OnLoad(JavaVM *jvm, void *reserved) {
  jclass cls, vcls;
  if ((*jvm)->GetEnv(jvm, (void **) &env, JNI_VERSION_1_2))
    return JNI_ERR;

  // Try to catch crashes...
  struct sigaction handler;
  memset(&handler, 0, sizeof(sigaction));
  handler.sa_sigaction = android_sigaction;
```

```
  handler.sa_flags = SA_RESETHAND;

  #define CATCHSIG(X) sigaction(X, &handler, &old_sa[X])
  CATCHSIG(SIGILL);
  CATCHSIG(SIGABRT);
  CATCHSIG(SIGBUS);
  CATCHSIG(SIGFPE);
  CATCHSIG(SIGSEGV);
  CATCHSIG(SIGSTKFLT);
  CATCHSIG(SIGPIPE);

  return JNI_VERSION_1_2;
}

jint throwNoClassDefError(JNIEnv *env, char *message) {
  jclass exClass;
  char *className = "java/lang/NoClassDefFoundError";

  exClass = (*env)->FindClass(env, className);
  if (exClass == NULL) {
    return throwNoClassDefError(env, className);
  }
  (*env)->ThrowNew(env, exClass, message);
}
.....
```