Notification > Push > Android SDK 가이드

Push SDK를 적용하면 모바일 애플리케이션과 Push를 쉽게 연동할 수 있습니다.

주요 기능

- OS에 알림 토큰 등록
- 알림 메시지 수신 및 표시
- 메시지 수신 및 수신된 메시지를 통한 애플리케이션 실행 지표 수집

다운로드

TOAST Document에서 **Notification > Push** 아래의 **Android SDK (AAR)**을 클릭해 파일을 다운로드합니다.

지원 환경

버전

- API 레벨 15(4.0.3) 이상

지원 플랫폼

- Google Cloud Messaging(이하 GCM)
- Tencent Mobile Push(이하 Tencent)
- Amazon Device Messaging(이하 ADM)

프로젝트 설정

공통 설정 (JCenter)

의존성 추가

- 아래처럼 SDK의 의존을 추가합니다.

```
dependencies {
    implementation 'com.toast.android:pushsdk:1.7.0'
    // compile 'com.toast.android:pushsdk:1.7.0' // (Gradle < 3.4)
}
```

Android Support 라이브러리 중복이 발생할 경우

- 아래와 같이 SDK에서 Support 라이브러리를 제외합니다.

```
dependencies {
    implementation('com.toast.android:pushsdk:1.7.0') {
        exclude group: 'com.android.support', module: 'support-v4'
    }
}
```

공통 설정 (Manual)

- JCenter를 이용하지 않거나 이용할 수 없는 경우, SDK를 다운로드해서 수동으로 설정해줍니다.

다운로드 및 의존성 추가

- SDK(AAR) 다운로드 및 추가
  - 프로젝트 폴더 하위에 libs 폴더가 없으면 생성합니다.
  - 다운로드한 AAR 파일을 프로젝트의 libs 폴더에 추가합니다.
- build.gradle에 SDK 추가
  - dependencies에 다음과 같이 추가합니다.

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.aar'])
    compile 'com.android.support:support-v4:26.1.0'
}
```

GCM 설정

프로젝트 설정

- build.gradle에 GCM SDK 추가
  - dependencies에 다음과 같이 추가합니다.

```
dependencies {
    compile 'com.google.android.gms:play-services-gcm:11.0.0'
}
```

AndroidManifest.xml 수정

- 아래에 '[YOUR_PACKAGE_NAME]'으로 되어 있는 모든 부분을 애플리케이션 기본 페키지 네임으로 변경합니다.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="
[YOUR_PACKAGE_NAME]">
<application android:icon="@drawable/app_icon">
        <receiver android:name="com.google.android.gms.gcm.GcmReceiver"
android:exported="true"
                android:permission="com.google.android.c2dm.permission.SEND">
            <intent-filter>
```

```
                <action android:name="com.google.android.c2dm.intent.RECEIVE"
/>
                <category android:name="[YOUR_PACKAGE_NAME]" />
            </intent-filter>
        </receiver>
    <service android:name="com.toast.android.pushsdk.PushSdk$GcmListener"
android:exported="false">
    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.RECEIVE" />
    </intent-filter>
    </service>
        <service
android:name="com.toast.android.pushsdk.PushService$IdListener"
android:exported="false">
            <intent-filter>
                <action android:name="com.google.android.gms.iid.InstanceID"/>
            </intent-filter>
        </service>
</application>
<permission android:name="[YOUR_PACKAGE_NAME].permission.C2D_MESSAGE"
android:protectionLevel="signature"/>
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
<uses-permission android:name="[YOUR_PACKAGE_NAME].permission.C2D_MESSAGE" />
</manifest>
```

Tencent 설정

프로젝트 설정

가이드는 Tencent SDK 3.2.3을 기준으로 작성되었습니다.

- Tencent SDK 다운로드 페이지인 [腾讯移动推送 | 信鸽](#)에서 Tencent SDK를 다운로드합니다.
- 다운로드한 SDK의 압축을 해제하고 아래 파일을 프로젝트 하위의 libs 폴더에 추가합니다.
    - 필수
        - libs/jg_filter_sdk_1.1.jar
        - libs/mid-core-sdk-4.0.6.jar
        - libs/wup-1.0.0.E-SNAPSHOT.jar
        - libs/Xg_sdk_v3.2.3_20180403_1839.jar
        - libs/armeabi 폴더 전체
    - 선택 : 다른 아키텍처 사용을 원하는 경우
        - Other-Platform-SO/{아키텍처 이름의 폴더}
- build.gradle 수정

```
android {
    sourceSets {
        main {
            jniLibs.srcDirs = ['libs']
        }
    }
}
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
}
```

AndroidManifest.xml 수정

- 아래에 '[YOUR_PACKAGE_NAME]'으로 되어 있는 모든 부분을 애플리케이션 기본 페키지 네임으로 변경합니다.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="
[YOUR_PACKAGE_NAME]">
  <application android:icon="@drawable/app_icon">
    <receiver android:name="com.tencent.android.tpush.XGPushReceiver"
android:process=":xg_service_v3" >
      <intent-filter android:priority="0x7fffffff" >
        <action android:name="com.tencent.android.tpush.action.SDK" />
        <action
android:name="com.tencent.android.tpush.action.INTERNAL_PUSH_MESSAGE" />
        <action android:name="android.intent.action.USER_PRESENT" />
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
      </intent-filter>
    </receiver>
    <service android:name="com.tencent.android.tpush.service.XGPushServiceV3"
android:exported="true" android:persistent="true"
android:process=":xg_service_v3">
    </service>
    <receiver android:name="com.toast.android.pushsdk.PushSdk$XgListener">
      <intent-filter>
        <action android:name="com.tencent.android.tpush.action.PUSH_MESSAGE"
/>
        <action android:name="com.tencent.android.tpush.action.FEEDBACK" />
      </intent-filter>
    </receiver>
    <service android:name="com.tencent.android.tpush.rpc.XGRemoteService"
android:exported="true" >
        <intent-filter>
            <action android:name="[YOUR_PACKAGE_NAME].PUSH_ACTION" />
        </intent-filter>
    </service>
    <provider
    android:name="com.tencent.android.tpush.XGPushProvider"
```

```
    android:authorities="[YOUR_PACKAGE_NAME].AUTH_XGPUSH"
    android:exported="true"/>
    <provider
        android:name="com.tencent.android.tpush.SettingsContentProvider"
        android:authorities="[YOUR_PACKAGE_NAME].TPUSH_PROVIDER"
        android:exported="false" />
    <provider
        android:name="com.tencent.mid.api.MidProvider"
        android:authorities="[YOUR_PACKAGE_NAME].TENCENT.MID.V3"
        android:exported="true" >
    </provider>
  </application>
  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
  <uses-permission android:name="android.permission.BROADCAST_STICKY" />
  <uses-permission android:name="android.permission.KILL_BACKGROUND_PROCESSES"
/>
  <uses-permission android:name="android.permission.GET_TASKS" />
  <!--permissions requiring user alerts-->
  <uses-permission android:name="android.permission.READ_PHONE_STATE" />
  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
  <uses-permission android:name="android.permission.RECEIVE_USER_PRESENT" />
  <uses-permission android:name="android.permission.RESTART_PACKAGES" />
  <uses-permission android:name="android.permission.WRITE_SETTINGS" />
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
  <uses-permission android:name="android.permission.READ_LOGS" />

  <uses-permission android:name="android.permission.VIBRATE" />
</manifest>
```

ADM 설정

프로젝트 설정

- Amazon Device Messaging(이하 ADM)은 Fire OS 2세대 이상의 기기에서 사용할 수 있습니다.
- Amazon Developer SDKs 다운로드 페이지에서 Amazon Device Messaging SDK를 다운로드합니다.
- 다운로드한 SDK의 압축을 해제하고 amazon-device-messaging-*.jar 파일을 프로젝트 하위의 libs 폴더에 추가합니다.
- build.gradle을 수정합니다.

```
dependencies {
    compileOnly files('libs/amazon-device-messaging-1.0.1.jar')
}
```

AndroidManifest.xml 수정

- 아래에 '[YOUR_PACKAGE_NAME]'으로 되어 있는 모든 부분을 애플리케이션 기본 페키지 네임으로 변경합니다.

Namespace 추가

```
<manifest xmlns:amazon="http://schemas.amazon.com/apk/res/android">
```

권한 추가

```
<permission
    android:name="[YOUR_PACKAGE_NAME].permission.RECEIVE_ADM_MESSAGE"
    android:protectionLevel="signature" />
<uses-permission android:name="
[YOUR_PACKAGE_NAME].permission.RECEIVE_ADM_MESSAGE" />
<uses-permission android:name="com.amazon.device.messaging.permission.RECEIVE"
/>
```

Handler 및 Receiver 추가

- [YOUR_HANDLER_CLASS]에는 사용자가 작성한 Handler의 클래스를 입력합니다.
- [YOUR_RECEIVER_CLASS]에는 사용자가 작성한 Receiver의 클래스를 입력합니다.
    - Handler 및 Receiver 구현은 아래를 참고합니다.

```
<amazon:enable-feature android:name="com.amazon.device.messaging"
android:required="true"/>
<service
    android:name="[YOUR_HANDLER_CLASS]"
    android:exported="false" />

<receiver
    android:name="[YOUR_RECEIVER_CLASS]"
    android:permission="com.amazon.device.messaging.permission.SEND" >
    <intent-filter>
        <action android:name="com.amazon.device.messaging.intent.REGISTRATION"
/>
        <action android:name="com.amazon.device.messaging.intent.RECEIVE" />

        <category android:name="[YOUR_PACKAGE_NAME]" />
    </intent-filter>
</receiver>
```

Handler 및 Receiver 구현

- 수신한 Push 데이터를 이용해서 추가 작업이 필요없을 경우, 기본 Handler와 Receiver를 사용하셔도 됩니다.
    - 기본 Handler : com.toast.android.pushsdk.listener.DefaultPushSdkADMHandler
    - 기본 Receiver : com.toast.android.pushsdk.listener.DefaultPushSdkADMReceiver
- 사용자 Handler는 com.toast.android.pushsdk.listener.AbstractPushSdkADMHandler 클래스를 상속해야 합니다.

```
public class CustomADMHandler extends AbstractPushSdkADMHandler {
    public CustomADMHandler() {
        super(CustomADMHandler.class);
    }

    @Override
    protected void onMessage(Intent intent) {
        Bundle extras = intent.getExtras();
        String value = extras.getString("key");
        // 수신한 데이터를 이용해서 알림 표시
    }
}
```

- 사용자 Receiver는 com.amazon.device.messaging.ADMMessageReceiver 클래스를 상속해야 합니다.

```
public class CustomADMReceiver extends ADMMessageReceiver {
    public CustomADMReceiver() {
        super(CustomADMHandler.class); // 반드시 사용자 Handler의 클래스를 입력해야 합니다.
    }
}
```

SDK 사용 가이드

PushParams 정의

- PushParams 객체는 토큰 등록 및 조회를 위해서 필요한 객체입니다.
- PushParams.Builder 클래스를 통해서 객체를 생성할 수 있습니다.
- PushParams에 포함된 정보는 아래와 같습니다.

프로퍼티	설명	필수 여부	기본값
appKey	Push 서비스 키	필수	없음
userId	사용자 식별자	필수	없음
context	컨텍스트	필수	없음
pushType	푸시 유형(GCM, Tencent)	필수	없음
channel	채널	선택	default
country	국가 코드	선택	시스템 국가 코드
language	언어 코드	선택	시스템 언어 코드
isNotificationAgreement	알림 표시 동의 여부	선택	false
isAdAgreement	광고성 정보 알림 표시 동의 여부	선택	false
isNightAdAgreement	야간 광고성 정보 알림 표시 동의 여부	선택	false

PushParams 객체 생성

- PushParams 객체는 PushParams.Builder 클래스를 통해서 생성할 수 있습니다.
- 예제 코드

```
PushType pushType = null;
if (isGCM) {
    pushType = PushType.gcm("[YOUR_SENDER_ID]");
} else if (isTencent) {
    long yourAccessId = 1234567890L; // [YOUR_ACCESS_ID]
    pushType = PushType.tencent(yourAccessId, "[YOUR_ACCESS_KEY]");
} else if (isADM) {
    pushType = PushType.adm();
}

PushParams.Builder builder = new PushParams.Builder(this, "[YOUR_APP_KEY]", "
[YOUR_USER_ID]", pushType);

builder.setChannel("default"); // 선택값
builder.setNotificationAgreement(true); // 선택값
builder.setAdAgreement(true); // 선택값
builder.setNightAdAgreement(true); // 선택값
builder.setCountry("KR"); // 선택값
builder.setLanguage("ko"); // 선택값

PushParams pushParams = builder.build();
```

토큰 등록

- 푸시 타입에 따라 토큰을 생성해서 서버에 토큰을 등록합니다.
- 예제 코드

```
PushSdk.register(pushParams, new PushRegisterCallback() {
    @Override
    public void onRegister(PushRegisterResult result) {
        if (result.isSuccessful()) {
            // 토큰 등록 성공
        } else {
            // 토큰 등록 실패
            Log.e(TAG, "error,code=" + result.getCode() + ",message=" +
result.getMessage());
        }
    }
});
```

> **Tencent를 사용하는 경우 예외 사항**
>
> Tencent는 WRITE_SETTINGS 권한이 필요하며, API 레벨 23(6.0) 이상에서는 별도의 권한 설정창이 노출됩니다. 권한 설정창 노출과 콜백은 비동기이므로, 권한 설정창 노출과 함께 콜백으로 ERROR_PERMISSION_REQUIRED 오류가 반환됩니다. 이 경우, 다시 토큰 등록을 호출해야합니다. 사용자가 WRITE_SETTINGS 권한을 허용했다면 정상적으로 토큰이 등록됩니다. 사용자가 WRITE_SETTINGS 권한을 허용하지 않았다면 계속해서 ERROR_PERMISSION_REQUIRED 오류가 반환됩니다.

토큰 정보 조회

- 현재 서버에 저장된 토큰 정보가 PushQueryResult 객체에 담겨 콜백으로 반환됩니다.
- 예제 코드

```
PushSdk.query(pushParams, new PushQueryCallback() {
    @Override
    public void onQuery(PushQueryResult result) {
        if (result.isSuccessful()) {
            // 토큰 정보 조회 성공
            TokenInfo tokenInfo = result.getTokenInfo();
        } else {
            // 토큰 정보 조회 실패
            Log.e(TAG, "error,code=" + result.getCode() + ",message=" +
result.getMessage());
        }
    }
});
```

디버그 로그 활성화

- Push SDK는 SDK의 디버그 로그를 활성화하는 메서드를 제공합니다.
- 반드시 개발 중에만 디버그 로그를 활성화해야 합니다. 릴리스할 때는 제거하거나 false로 설정해야 합니다.

```
PushSdk.setDebug(true);
```

지표 수집 (GCM Only)

수신(Received) 지표

- SDK에서 제공하는 기본 리시버를 사용하는 경우, 수신 지표가 자동으로 수집됩니다.
- 사용자가 직접 리시버를 구현하는 경우, 수신 지표 수집을 위해서 아래 메서드를 리시버에 추가해야 합니다.
- 예제 코드

```
public static class CustomPushReceiver extends GcmListenerService {
    @Override
    public void onMessageReceived(String from, Bundle data) {
        PushAnalytics.onReceived(this, data);
        // 알림 등록 로직 수행
    }
}
```

실행(Opened) 지표

- 알림 영역의 알림을 통해서 실행했을 경우를 실행 지표라고 합니다.
- 실행 지표 수집을 위해서는 액티비티와 푸시 리시버를 수정해야 합니다.
- MainActivity 혹은 알림 클릭 시 실행되는 액티비티에 다음과 같은 코드를 추가해야 합니다.

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        PushAnalytics.onOpened(this, getIntent());
    }

    @Override
    protected void onNewIntent(Intent intent) {
        PushAnalytics.onOpened(this, intent);
    }
}
```

- 푸시 리시버에 다음과 같은 코드를 추가해야 합니다.
    - SDK에서 제공하는 기본 리시버를 사용하는 경우, 아래 코드를 추가할 필요가 없습니다.

```
public static class CustomPushReceiver extends GcmListenerService {
    @Override
    public void onMessageReceived(String from, Bundle data) {
        Intent launchIntent = new Intent(context, MainActivity.class);
        Intent intent = PushAnalytics.newIntentForOpenedEvent(launchIntent,
bundle);

        // 혹은 Intent intent = PushAnalytics.newIntentForOpenedEvent(context,
MainActivity.class, bundle);
    }
}
```

오류 코드

- 오류 코드는 com.toast.android.pushsdk.annotations.PushResultCode 어노테이션에 @IntDef 로 정의되어있습니다.

오류 코드	설명
ERROR_SYSTEM_FAIL	시스템 문제로 토큰 획득에 실패한 경우
ERROR_NETWORK_FAIL	네트워크 문제로 요청에 실패한 경우
ERROR_SERVER_FAIL	서버에서 실패 응답을 반환한 경우
ERROR_ALREADY_IN_PROGRESS	토큰 등록/조회가 이미 실행 중인 경우
ERROR_INVALID_PARAMETERS	파라미터가 잘못된 경우
ERROR_PERMISSION_REQUIRED	권한이 필요한 경우(Tencent만 해당)
ERROR_PARSE_JSON_FAIL	서버 응답을 파싱하지 못한 경우

리치 메시지 사용 가이드

- 리치 메시지 기능을 위한 SDK 사용 가이드입니다.

리치 메시지란?

- 제목, 본문과 함께 다양하고 풍부한 메시지를 수신할 수 있습니다.

지원하는 리치 메시지 기능 및 스펙

버튼

- 알림 제거 : 현재 알림을 제거합니다.
- 앱 열기 : 앱을 실행합니다.
- URL 열기 : 특정 URL로 이동합니다.

    - Custom scheme을 이용한 Activity/BroadcastReceiver 이동도 가능
- 답장 전송 : 알림에서 바로 답장을 보냅니다.
    - 안드로이드 7.0(API 레벨 24) 이상에서만 사용 가능합니다.
    - NotificationConverter를 통해 변환된 알림일 경우, 안드로이드 7.0 미만 기기에서는 답장 버튼이 제거된채로 알림이 노출됩니다.
    - 답장 처리를 위해서는 리스너 등록이 필요합니다. 자세한 내용은 *답장 리스너 구현 및 등록* 을 참고해주세요.

> 버튼은 최대 3개까지 지원합니다.

> 버튼 기능을 사용하기 위해서는 *NotificationConverter 클래스* 를 이용해서 알림을 생성해야 합니다.

미디어

- 이미지 : 알림에 이미지를 추가합니다. (내부, 외부 이미지 모두 지원)
    - 이미지는 가로와 세로 비율이 2:1인 이미지를 권장합니다.
    - 다른 비율의 이미지는 잘린채로 노출될 수 있습니다.
- 그 외 : 그 외의 미디어(동영상, 소리 등)는 지원하지 않습니다.

큰 아이콘

- 알림에 큰 아이콘을 추가합니다. (내부, 외부 이미지 모두 지원)
    - 큰 아이콘의 이미지는 1:1 비율을 권장합니다.
    - 다른 비율의 이미지는 강제로 비율이 1:1로 변경되기 때문에 기대와 다른 이미지가 노출될 수 있습니다.

그룹

- 같은 키의 알림들을 하나로 묶습니다.
    - 안드로이드 7.0(API 레벨 24) 이상에서만 사용가능합니다.
    - NotificationConverter를 통해 변환된 알림일 경우, 안드로이드 7.0 미만 기기에서는 그룹이 무시된채로 알림이 노출됩니다.

리치 메시지 수신 및 알림 등록

- Push SDK에서 제공하는 기본 리시버들을 그대로 사용 중이라면 별도의 처리없이 리치 메시지를 수신 및 알림으로 등록이 가능합니다.
- 별도의 리시버를 구현해서 사용 중이라면 *ToastPushMessage 클래스와 NotificationConverter 클래스* 를 확인해주세요.

ToastPushMessage 정의

- Push 제공자(GCM, Tencent, ADM 등)별로 서로 다른 데이터 타입을 통합한 데이터 구조입니다.
- TOAST Push 서버에서 발송한 알림의 제목, 내용 외에 **리치 메시지**가 담겨 있습니다.
- NotificationConverter 객체를 이용해서 쉽게 알림(Notification)으로 변환이 가능합니다.
    - 사용자는 ToastPushMessage의 데이터를 이용해서 직접 Notification 객체를 만들 수 있습니다.
- 아래 입력 타입을 지원합니다.
    - GCM(Google Cloud Messaging) : Bundle
    - ADM(Amazon Device Messaging) : Bundle

    - Tencent : XGPushTextMessage
- 데이터 구조는 아래와 같습니다.

```
class ToastPushMessage {
    CharSequence title;
    CharSequence body;
    RichMessage richMessage;
    Map<String, String> extras;
}

class RichMessage {
    MediaInfo media;
    LargeIconInfo largeIcon;
    GroupInfo group;
    List<ButtonInfo> buttons;
}
```

NotificationConverter 사용하기

- ToastPushMessage 객체를 좀 더 편하게 알림(Notification) 객체로 변환할 수 있도록 NotificationConverter를 제공합니다.
- NotificationConverter는 *리치 메시지가 모두 적용된 알림(Notification) 객체* 를 반환합니다.
- NotificationConverter에 Extender를 추가해서 Notification을 사용자 정의할 수 있습니다.
    - 추가된 Extender는 NotificationConverter의 내부 처리가 완료된 이후에 추가된 순서대로 처리됩니다.
- 예제)

```
// Example for GCM
final ToastPushMessage message = ToastPushMessage.fromBundle(bundle);
final NotificationManagerCompat manager =
NotificationManagerCompat.from(context);
final NotificationConverter converter = new
NotificationConverter("YOUR_NOTIFICATION_CHANNEL", message);
// Extends NotificationCompat.Builder when you need
converter.addExtender(new NotificationCompat.Extender() {
    @Override
    public NotificationCompat.Builder extend(NotificationCompat.Builder builder)
{
        // Extends builder
        // ex) return builder.setAutoCancel(false);
        return builder;
    }
});
converter.convert(context, new NotificationConverter.ConvertCallback() {
    @Override
    public void onConvert(int notificationId, @NonNull Notification
notification) {
```

```
        // Notify notification to notification bar
        manager.notify(notificationId, notification);
    }
});
```

답장 리스너 구현 및 등록

- 답장 버튼을 통해서 사용자의 입력을 받으면 앱은 사용자 입력에 대한 처리를 해야합니다.
    - 메신저로 예를 들면, 메신저 서버로 사용자 입력을 전송해야 합니다.

답장 리스너 구현

- 답장 처리를 위해서 ReplyActionListener 인터페이스를 상속받아서 리스너를 구현해야합니다.

> (주의) 사용자가 입력을 완료하고 전송 버튼을 누르면 알림(Notification)은 로딩바가 노출되며 알림이 제거되지 않습니다. 따라서 답장 처리가 완료되면 알림(Notification)을 제거하거나 업데이트하는 코드를 추가해야합니다. 아래 예제 코드를 참고해주세요.

- 예제)

```
public class ReplyHandler implements ReplyActionListener {
    @Override
    public void onReceive(@NonNull Context context, @NonNull ReplyActionResult
result) {
        // Do Something (ex. Send message contents to server)

        // Choice 1. Remove previous reply notification with notification id

NotificationManagerCompat.from(context).cancel(result.getNotificationId());

        // Choice 2. Update previous reply notification with notification id

NotificationManagerCompat.from(context).notify(result.getNotificationId(),
                new NotificationCompat.Builder(context,
result.getNotificationChannel())
                        .setSmallIcon(/* Resource ID of your icon */
getDefaultIcon(context))
                        .setContentTitle("Send")
                        .setContentText("Success to send message")
                        .build());
    }
}
```

답장 리스너 등록

- 구현된 리스너의 클래스를 **PushSdk.setReplyActionListenerClass** 메소드를 통해서 등록해야 합니다.
- 예제)

```
PushSdk.setReplyActionListenerClass(/* Context */ this, ReplyHandler.class);
```