CONTENTPROVIDER 앱 간 데이터 공유 기본

- DB로 직접 접근하기 vs ContentProvider 사용하기?

  안드로이드에서는 기본적으로 SQLite를 사용할 수 있도록 API를 다양하게 편리하게 제공해주고 있다. 하지만, 이러한 SQLite는 기본적으로 해당 DB를 생성한 앱에서만 접근이 가능한 제약 사항이 있기 때문에, 앱간의 공유가 어려웠었다.
  하지만, 이제는 안드로이드에서 기본적으로 주소록이나 콜로그 등에 대해서 ContentProvider를 통해서 데이터를 제공해주고 있고, Android SDK에서도 해당하는 내용을 사용할 것을 권하고 있다. 만약 앱에서 데이터를 내부적으로만 사용한다면, 굳이 ContentProvider를 사용하지 않아도 되고, 사실 ContentProvider를 처음에 접하게 되면 익숙하지 않기도 하고 제대로 된 매뉴얼도 없어서 상당히 어려움을 많이 받기도 한다.
  하지만 점점 많은 개발자들이 data 레이어 위에 보안을 위해서 하나의 추가적인 레이어를 두는 것을 추천하기도 하고, 최초에는 데이터를 공유하지 않으나 혹 나중에 앱이 활성화된 이후에 데이터 공유할 일이 있다면 처음에 ContentProvider에 익숙해지는 비용보다 그 때 가서 수정에 드는 비용이 너무나 크기 때문에 내부적으로 사용하더라도 데이터 수집을 ContenteProvider를 통해서 하나의 API로 하는 것을 권하기도 한다.
  사실 SQLite를 통해서 직접 DB를 접근하는 것과 ContentProvider를 통해서 접근하는 것은 크게 다른 점은 없으나, 향후 확장성을 고려한다면 ContentProvider를 사용하는 것이 효율적일 수 있다는 의견이 많다. 따라서, 한번 ContentProvider를 이해하는데 드는 시간과 비용, 그리고 나중에 앱이 활성화 되었을 때 데이터를 공유해야할 가능성을 고려하여 앱에 적용 여부를 결정하면 좋을 것이다.
  만약 대형 앱 프로젝트를 진행하고 있다면 적용하면 좋을 것이고, 소형 프로젝트에서는 이미 알고 있는 내용이 아니라면 고민을 해봐도 괜찮을 것이다. 이제부터 ContenteProvider를 사용하는 안드로이드 매뉴얼을 살펴보자.

- ContentProvider에 대하여

  ContentProvider는 구조화된 데이터들을 접근하는데 관리하는 개념이다. 데이터를 한 단계 더 캡슐화 시키고, 데이터를 제공하는데 있어서 기본적인 보안 메카니즘들을 제공한다. ContentProvider는 특정 프로세스에 있는 데이터를 다른 프로세스로 연동시키는 표준 인터페이스를 제공해준다.

  ContentProvider를 통해서 데이터를 접근할 때에는 앱의 Context에서 ContentResolver 객체를 사용하여 데이터 제공자에 접근하게 된다. ContentResolver 객체는 데이터 제공자와 통신하여 ContentProvider를 구현하는 객체의 인스턴스를 가져오게 되고, 해당 provider 객체를 통해서 데이터를 요청하고 결과를 수신하게 된다.

  다른 앱과 데이터 통신을 할 예정이 아니라면, 굳이 ContentProvider를 이용하여 개발할 필요는 없지만, 임의의 자동 검색 등과 같은 기능을 쉽게 구현하기 위해서는 ContentProvider를 구현하면 좋다. 또한, 복잡한 데이터나 파일들을 복사하여 다른 앱에서 사용하고 싶다면 필요할 것이다.

  안드로이드는 자체적으로 오디오/비디오/이미지와 주소록 등의 정보를 제공받을 수 있는 ContentProvider를 포함하고 있고, 이것은 android.provider 패키지에 대한 레퍼런스 문서를 보면 목록을 볼 수 있을 것이다. 이러한 ContentProvider는 몇 가지 제약점을 가지지만, 모든 안드로이드 앱

에서 접근이 가능한 것을 참고하면 ContentProvider의 활용도를 이해할 수 있을 것이다.

- ## ContentProvider 기본

ContentProvider는 중앙 집중형 저장소에서 데이터 접근에 대한 관리를 담당하고 있다. 데이터 제공자는 안드로이드 앱의 일부이며, 때때로는 제공자 자체가 별도의 UI를 가지고 있기도 하다. 어쨌든 ContentProvider는 기본적으로 다른 앱에서 데이터 제공자가 제공하는 데이터 객체에 접근하고자 할 때 사용된다.
데이터의 제공자와 제공자에 접근하는 클라이언트는 표준화된 인터페이스를 통해서 데이터를 주고 받으며, 안전한 보안 메카니즘까지 제공해주게 된다.

- ## 개요

ContentProvider는 RDB에 저장되어있는 테이블들을 외부에 있는 앱에게 제공을 해준다. 테이블에 있는 하나의 행은 Collection에 있는 하나의 데이터 인스턴스와 같이 제공되며, 각 칼럼은 인스턴스에 저장되어있는 데이터처럼 취급 한다.

예를 들면,안드로이드에서 기본적으로 제공해주고 있는 ContentProvider로는 user dictionary이다. 이것은 사용자가 문법적으로 맞지 않은 단어들을 지속적으로 사용하여 문법 체크를 하지 않을 수 있도록 저장하는 테이블이다. 아래 테이블은 제공자의 테이블에 저장되어있는 예이다.

word	app id	frequency	locale	_ID
mapreduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
applet	user2	225	fr_CA	3
const	user1	255	pt_BR	4
int	user5	100	en_UK	6

위의 테이블에서 각 행은 표준 사전에 등록되어 있지 않을 단어에 대한 인스턴스를 나타내고, 각 열은 해당 단어에 대한 정보를 나타낸다. word, app id 등과 같은 열의 이름을 provider에 저장되고, 각 행의 locale을 접근하기 위해서는 locale 칼럼에 접근을 하게 될 것이다. 이 제공자는 ContentProvider가 자동으로 보유하게 되는 _ID 칼럼을 primary key로 활용하고 있다.

* 참고: provider는 primary key나 _ID로 명명된 칼럼이 반드시 필요하지는 않지만, 만약 ListView와 데이터를 바인딩해서 보여주고 싶다면, 하나의 칼럼명은 반드시 _ID의 이름을 가지고 있어야 한다.

- ## Provider 접근하기

앱이 ContentProvider를 통해서 데이터에 접근할 때에는 ContentResolver를 통해서 접근하게 된다. 이 객체는 ContentProvider를 상속받아 특정 함수들을 보유하고 있는 객체이다. ContentResolver는 기존적인 CRUD(Create Retreive Update Delete) 함수들을 보유하고 있다.

ContenteResovler 객체는 클라이언트의 앱에서 보유하고 있고, ContentProvider 객체는 데이터를 제공하는 앱에서 가지고 있어서 자동으로 IPC를 통해 통신하게 된다. ContentProvider는 또한 데이터 저장소 위에 하나의 추상적인 계층으로 형성되어, 외부에 데이터 테이블과 저장소에 대한 정보를 캡슐

화할 수 있다.

* 참고: 제공자에 접근하기 위해서는 앱에서는 manifest 파일에 permission을 설정해야 한다.

예를 들면, 위의 사용자 단어와 해당 locale에 대한 정보를 얻어오기 위해서 UserDictionaryProvider를 사용한다고 하면, 처음에 ContenteResolver.query()를 호출하게 된다. 그러면 query() 함수는 내부적으로 UserDictionaryProvider로 제공되는 ContentProvider.query() 함수를 호출하게 된다. 아래는 ContentResolver.query()에 대한 예이다.

```
// Queries the user dictionary and returns results
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,   // The content URI of the words table
    mProjection,                        // The columns to return for each row
    mSelectionClause                    // Selection criteria
    mSelectionArgs,                     // Selection criteria
    mSortOrder);                        // The sort order for the returned rows
```

아래는 SQL 중에서 SELECT에 해당하는 query 함수에 대한 파라미터를 나타낸다.

query() 파라미터	SELECT 구문 변환	비고
Uri	FROM table_name	Uri는 provider의 이름을 포함하여 table_name과 매핑된다
projection	col, col, col, ...	projection은 칼럼들의 배열을 나타낸다.
selection	WHERE col = value	selection은 쿼리하는 row에 대한 where 절을 설정한다
selectionArgs	WHERE절에 있는 ?를 대체함	
sortOrder	ORDER BY col, col, ...	sortOrder는 Cursor에 리턴되는 order를 정의한다

- **컨텐트 URi**

  컨텐트 Uri는 데이터 provider 안에 있는 데이터를 정의한다. 컨텐트 Uri는 provider에 해당하는 고유한 이름을 가지고 있는데, 이것을 authority라고 하고, 테이블을 나타내는 path로 이루어져 있다. provider에 있는 테이블에 접근하고자 할 때 컨텐트 Uri의 파라미터 중 하나로 테이블명을 넣는다.

  위에 있는 소스 코드에서 CONTENT_URI는 컨텐트 URI를 나타내며, 사용자 사전에 있는 words 테이블에 접근하게 된다. ContentResolver 객체는 URI의 authority를 파싱하여 시스템에 공유되고 있는 provider들의 authority와 비교하여 테이블에 접근하게 된다. ContentResolver는 이후에 해당 provider를 대상으로 쿼리를 실행하게 된다.

  ContentProvider는 path를 컨텐트 URI에 포함시켜서 어떠한 테이블에 접근할지 결정하게 된다. Provider들은 일반적으로 각 테이블에 접근할 수 있는 path를 각각 가지고 있다.

위의 코드에서 CONTENT_URI의 URI는 아래와 같이 정의되어있다.

```
content://user_dictionary/words
```

여기서 user_dictionary 스트링이 provider에 대한 authority를 나타내고, words는 테이블을 나타내는 경로이다. 그리고 문자열은 content://로 시작하는 scheme을 가지게 되어 이것이 컨텐트URI라는 것을 나타낸다.

많은 provider는 URI의 뒤에 테이블에 검색할 ID를 첨부하여 하나의 row를 검색할 수 있도록 제공해준다. 예를 들면, _ID 칼럼이 4인 row를 user_dictionary에서 조회할 때에는 아래와 같이 컨텐트URI를 생성할 수 있다.

```
Uri singleUri = ContentUris.withAppendedId(UserDictionary.Words.CONTENT_URI,4);
```

이후에 row를 조회하거나 update 또는 delete를 할 때 ID값을 자주 사용하게 될 것이다.

* 참고: Uri와 Uri.Builder 클래스는 Uri 객체를 문자열에서부터 생성하는데 다양한 편의 함수들을 제공해준다. ContentUris는 id 값을 URI에 첨부할 수 있는 다양한 편의 함수들을 제공해준다. 위의 소스에서도 withAppendedId() 함수를 사용하여 Uri의 뒤에 id를 첨부하였다.

- **Provider를 통해 데이터 가져오기**

이번에는 사용자 사전의 Provider에서 데이터를 가져오는 것을 예로 활용해서 살펴보자. 먼저 한가지 주의할 점이 있다면, ContentResolver.query()는 UI 쓰레드에서 호출하고 있지만, 이것은 사실 별도의 쓰레드에서 비동기적으로 가져오는 것이 바람직하다. 이것을 구현하는 하나의 방법은 바로 CursorLoader 클래스를 사용하는 것으로 Loaders와 관련된 매뉴얼을 참조하면 된다. 또한, 아래의 소스 코드는 소스의 일부만 보여주고 있다.

우선 provider를 통해서 데이터를 가져오려면 아래의 단계들을 기본적으로 밟아야 한다.

1. **읽기 permission을 provider에 요청**
2. **Provider로 데이터 요청을 위한 쿼리 소스 작성**

- **읽기 permission 요청하기**

Provider로부터 데이터를 수집하기 위하여 앱에서는 "읽기 접근 허용 권한"을 요청해야 한다. 런타임에는 권한을 요청할 수는 없고, manifest에서 <uses-permission>에서 정의를 해야 한다. provider에 맞는 정확한 이름으로 정의하여 사용해야 한다. 이렇게 manifest에 권한을 명시하게 되면, 앱을 위해서 권한 요청을 진행하게 된다. 그리고 사용자들이 앱을 설치할 때에 이 권한을 부여에 동의하게 되는 것이다.

사용하고자 하는 읽기 권한을 위한 정확한 이름을 알기 위해서는 다른 접근 권한 이름을 아는 것과 마찬가지로 provider에서 제공하는 문서를 참조하면 된다.
권한의 역할에 대해서는 컨태트 provider 권한 섹션에 더 상세하게 설명할 것이다. 사용자 사전

provider의 읽기 권한은 android.permision.READ_USER_DICTIONARY라고 manifest 파일에서 정의하고 있다. 따라서 다른 앱에서 이 provider를 사용하기 위해서는 이 권한을 요청해야 한다.

- **쿼리 하기**

다음은 provider를 통해서 쿼리를 해야 한다. 우선 먼저 사용자 사전 provider에 접근하기 위한 변수들을 선언해야 한다.

```
// "projection" 변수는 결과로 수집할 칼럼들을 정의한다.
String[] mProjection =
{
    UserDictionary.Words._ID,     // _ID 칼럼명을 나타내는 상수
    UserDictionary.Words.WORD,    // word 칼럼명을 나타내는 상수
    UserDictionary.Words.LOCALE   // locale 칼럼명을 나타내는 상수
};

// selection을 정의하는 변수
String mSelectionClause = null;

// selection arguments를 정의하는 변수
String[] mSelectionArgs = {""};
```

다음은 사용자 사전 provider에 ContentResolver.query()를 사용하는 예가 나타나있다. Provider에 접근하는 클라이언트는 SQL 쿼리와 다소 유사하고, 리턴될 칼럼의 목록과 where 절의 조건들과 정렬되는 순서도 함께 포함된다. 먼저데이터를 조회하는 경우 리턴되는 칼럼의 목록은 projection이라고 불리운다. 위에서는 mProjection 변수로 정의되어있다.

어떠한 행들을 가져올지는 selection과 selection arguments를 통해서 정의된다. 이 selection 구문은 논리 표현과 boolean 표현을 기반으로 칼럼 명과 값들을 비교하며, 위의 변수들 중에서는 mSelectionClause 이다. 만약 동적인 파라미터로 ?를 넣게 된다면 selection arguments를 통해서 동적으로 선택이 가능하며, 위의 변수 중에서 mSelectionArgs를 나타낸다.

이제 다음 소스에서는 만약 사용자가 단어를 입력하지 않는다면, selection 변수는 null로 설정되어 쿼리는 provider에서 제공되는 모든 단어들을 반환하게 된다. 만약 사용자가 단어를 입력한다면, selection은 "UserDictionary.Words.WORD + " = ?" 로 설정되고, 사용자가 입력한 단어는 selection arguments를 통해서 입력하게 된다.

```
/*
 * selection argument를 정의하는 1개짜리 배열 변수
 */
String[] mSelectionArgs = {""};

// UI로부터 사용자가 단어를 검색했는지 확인
mSearchString = mSearchWord.getText().toString();
```

```
// 문자열을 가져온 뒤, 위의 문자열의 검증하는 단계를 여기에 구현 필요

// 문자열이 공백이라면 모든 것을 가져온다.
if (TextUtils.isEmpty(mSearchString)) {
    // selection을 null로 입력해서 전부다 가져오도록 설정
    mSelectionClause = null;
    mSelectionArgs[0] = "";
} else {
    // selection을 설정해서 사용자가 입력하는 문자열과 비교할 수 있도록 정의
    mSelectionClause = UserDictionary.Words.WORD + " = ?";

    // 사용자의 입력을 selection arguments에 넣기
    mSelectionArgs[0] = mSearchString;
}

// CONTENT_URI에 해당하는 테이블에 쿼리하고 cursor를 리턴한다.
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,   // 단어 테이블을 나타내는 URI
    mProjection,                        // 결과로 수집할 칼럼의 목록
    mSelectionClause                    // null 또는 사용자가 입력한 단어를 수용할 수 있게 입력
    mSelectionArgs,                     // 빈 배열 또는 사용자가 입력한 배열
    mSortOrder);                        // 결과를 수집하는 정렬 순서

// 어떠한 provider들은 cursor로 null을 리턴하기도 하고, exception이 일어나는 provider도 있다.
if (null == mCursor) {
    /*
     * 에러를 처리하는 소스를 여기에 구현 필요
     * android.util.Log.e() 를 호출하여 에러로 로그를 남기는 것이 좋을 것이다.
     *
     */
// 만약 cursor가 비어있다면, 찾은 결과가 없는 것이다.
} else if (mCursor.getCount() < 1) {

    /*
     * 여기에 사용자의 검색 결과가 없었다는 것을 알려주는 소스를 넣으면 된다.
     * 이것은 에러가 아니기 때문에 사용자에게 다른 검색을 요청하거나 새로운 행을 insert할 수 있도록 권하면 된다.
     */

} else {
    // 여기서는 결과를 처리하는 소스를 넣으면 된다.
```

```
}
```

위의 소스는 아래의 쿼리와 동일하게 생각할 수 있다.

```
SELECT _ID, word, locale FROM words WHERE word =  ORDER BY word ASC;
```

이 쿼리에서는 연동 클래스의 상수가 아닌 실제 칼럼들의 이름을 사용하고 있다.

- **침입을 시도하는 입력을 방어하는 방법**

  만약 컨텐트 provider를 통해서 외부의 신뢰할 수 없는 데이터를 그대로 사용하는 경우 SQL injection으로 침입될 수 있다. 예를 들면 아래와 같은 selection이 있다고 하면,

  ```
  // selection을 사용자의 입력을 통해서 설정한다.
  String mSelectionClause =  "var = " + mUserInput;
  ```

  만약 위와 같이 바로 구현을 한다면, SQL injection 침입을 당할 수 있는 위험성이 포함되어있다. 예를 들면, 사용자가 "nothing; DROP TABLE *;"로 쿼리를 입력하게 된다면, selection은 "var = nothing; DROP TABLE *;"이 되어 추가적인 쿼리 구문으로 인식하게 된다. 이 쿼리를 실행하게 되면 DB에 있는 모든 테이블들을 다 삭제되어버릴 것이다.

  이것을 방지하기 위해서는 selection arguments를 통해서 selection에 있는 ?를 대체하는 방법이 있다. 이렇게 이용하게 된다면, SQL로 취급되지 않고 쿼리의 일부인 문자열로 변환되어지고, SQL로 인식을 하지 않기 때문에 SQL injection이 일어나지 않을 것이다. 따라서, 사용자의 입력을 바로 문자열에 첨부하여 사용하기 보다는 아래와 같이 사용하면 좋다.

  ```
  // selection에는 ?로 대체 문자열을 넣는다.
  String mSelectionClause =  "var = ?";
  ```

  그리고 selection arguments 배열을 아래와 같이 설정하면 된다.

  ```
  // selection arguments를 정의하는 배열
  String[] selectionArgs = {""};
  ```

  마지막으로 검색하고자하는 값은 아래와 같이 삽입하면 된다.

  ```
  // selection argument에 사용자 입력을 넣는다.
  selectionArgs[0] = mUserInput;
  ```

  selection에 포함되어있는 ?들은 selection arguments 배열에 있는 값들로 대처되며, provider가 SQL DB 기반으로 데이터 수집을 하지 않더라도 이러한 방법으로 검색하게 되면 전부 수용이 가능하다.

- **쿼리 결과 보여주기**

  ContentResolver.query() 를 실행하고나면, 검색결과를 Cursor 객체에 포함되어 반환하게 된다. Cursor 객체는 각 행과 열에 랜덤 액세스를 지원해주기도 하고, Cursor의 함수들을 사용하여 검색된

순서대로 결과들을 조회할수도 있다. 어떠한 Cursor들은 provider의 데이터가 변경되면 자동으로 함께 변경되어 Cursor를 observe하는 객체의 함수를 트리거 시키기도 한다.

* 참고: Provider에서는 몇몇 중요한 칼럼들에 대하여 접근을 제한할 수 있다. 예를 들면, 주소록 provider는 몇몇 칼럼들에 접근을 제한하고 다른 activity나 service에 반환하지 않는다.

만약 selection 기준에 맞는 데이터가 없는 경우에는 Cursor 객체에서는 Cursor.getCount()가 0을 리턴한다. 만약 내부적으로 에러가 일어나는 경우에는 처리하는 방법은 해당 provider에 따라 달려있다. 어떠한 provider는 Cursor를 null로 리턴 할수도 있고, 어떠한 Cursor는 Exception이 일어날 수도 있다.

Cursor는 행의 목록으로 Cursor의 내용을 보여주기 적합한 방법은 바로 SimpleCursorAdapter를 통해서 ListView와 연동하는 것이다. 이어지는 아래의 소스는 위의 소스에 이어서 SimpleCursorAdapter를 생성하여 Cursor로 수집된 데이터를 ListView에 설정하는 소스이다.

```
// Cursor에서 수집하여 보여줄 칼럼들의 목록
String[] mWordListColumns =
{
    UserDictionary.Words.WORD,    // word칼럼명을 포함하는 상수
    UserDictionary.Words.LOCALE   // locale칼럼명을 포함하는 상수
};

// 각 칼럼들을 설정한 View의 ID들을 정의
int[] mWordListItems = { R.id.dictWord, R.id.locale};

// SimpleCursorAdapter를 생성
mCursorAdapter = new SimpleCursorAdapter(
    getApplicationContext(),                // 앱의 Context객체
    R.layout.wordlistrow,                   // ListView에서 하나의 행을 나타낼 레이아웃의 XML
    mCursor,                                  // query의 결과
    mWordListColumns,                         // cursor의 칼럼을 나타내는 칼럼 배열
    mWordListItems,                           // 행의 레이아웃에 있는 View의 ID 목록
    0);                                       // 플래그 (보통 필요하지 않음)

// ListView에 대한 adapter를 설정함
mWordList.setAdapter(mCursorAdapter);
```

* 참고: ListView와 Cursor가 연동하기 위해서는 Cursor는 _ID의 칼럼이 필요하다. 이것 때문에, 이전에 "words" 테이블에서 데이터를 수집할 때 요청했던 쿼리에는 _ID가 포함된다. 이에 따라 대부분의 ListView들은 대부분 _ID 칼럼을 활용하고 있는 이유이다.

- **쿼리 결과에서 데이터 가져오기**

  쿼리의 결과를 단순히 ListView에 보여주는 것 이외에, 데이터를 다른 곳에서 활용할수도 있을 것이

다. 예를 들면, 사용자 사전에서 단어의 스펠링은 가져온 다음, 또 다른 provider에서 활용할 수 있을 수도 있다. 이를 위해서 Cursor 안에 있는 행들을 순차적으로 접근하면 된다.

```
// word 칼럼의 인덱스가 몇 번째인지 확인
int index = mCursor.getColumnIndex(UserDictionary.Words.WORD);

/*
 * Cursor가 유효할 때에만 실행한다.사용자 사전 provider는 내부 오류가 일어나면 null을 리턴한다.
 * 다른 provider는 null을 리턴하는 대신에 Exception이 일어날수도 있다.
 */

if (mCursor != null) {
    /*
     * Cursor안에 있는 다음 행으로 이동. 최초로 이동하기 전에는 Cursor는 인덱스가 -1이고,
     * -1일 때 해당 위치에서 접근하고자 한다면 Exception이 일어날 것이다.
     */
    while (mCursor.moveToNext()) {

        // 칼럼에서 값을 가져오기
        newWord = mCursor.getString(index);

        // 수집된 단어에 대한 추가적인 처리 수행

        ...

        // while루프 끝
    }
} else {

    // 여기에 cursor가 널이거나 Exception이 일어났을 때 처리할 수 있는 소스 구현
}
```

Cursor는 서로 다른 변수형의 데이터를 가져오는데 "get"함수들을 지원고 있다. 예를 들면, 위의 소스에 있는 getString() 함수를 사용하고 있고, getType() 함수는 해당 칼럼의 데이터형대로 값을 리턴한다.

- **컨텐트 provider 권한**

  Provider 앱에서는 다른 앱에서 권한을 가질 수 있도록 설정할 수 있도록 permission을 명시해야 한다. 이 권한들은 사용자가 앱이 어떠한 데이터에 접근하고자 하는 알수 있도록 해주기 위함이다. Provider의 요구사항에 기반해서 다른 앱들은 provider에 접근할 수 있게 권한을 요청해야 한다. 사용자들은 앱을 설치할 때 이렇게 provider 에 대한 권한 요청을 하는 것을 확인할 수 있다.

  Provider 앱에서 아무런 권한을 명시하지 않았다면, 다른 앱들은 provider의 데이터에 접근할 수 있는

권한이 주어지지 않는다. 어쨋든, provider앱 안에 있는 컴포넌트들은 권한 정의와 상관 없이 항상 데이터에 읽기와 쓰기의 권한을 전부다 가지고 있다.

위에서 썼듯이, 사용자 사전 provider는 android.permission.READ_USER_DICTIONARY 권한이 필요하다. 해당 provider는 android.permission.WRITE_USER_DICTIONARY로 삽입, 갱신, 삭제 등의 쓰기 권한이 별ㄷ로 있다.

provider에 접근할 수 있는 권한을 얻기 위해서는 manifest 파일에 <uses-permission>에 해당 권한에 대하여 정의해야 한다. 그리고 안드로이드 패키지 매니저가 앱을 인스톨할 때에 사용자가 해당 권한을 앱에서 사용하게 될 것을 승인해야 한다. 만약 사용자가 전부다 승인하게 되면, 패키지 매니저는 인스톨을 이어서 하게 되고, 사용자가 승인을 하지 않으면 인스톨을 취소한다. 아래는 사용자 사전 provider를 읽는 요청을 정의한 것이다.

```
<uses-permission android:name="android.permission.READ_USER_DICTIONARY">
```

- **데이터 추가, 갱신, 삭제하기**

  데이터를 provider로부터 쿼리할 때와 마찬가지로 provider앱의 ContentProvider를 통해서 데이터를 수정할 수 있다. ContentResolver의 함수를 호출하면서 파라미터로 ContentProvider의 대응하는 함수에서 사용할 정보들을 넘겨주게 된다. Provider앱과 이를 사용하는 앱의 IPC간 보안은 자동으로 적용된다.

- **데이터 추가하기**

  데이터를 추가하기 위해서는 ContentReslover.insert() 함수를 호출한다. 이 함수는 Provider에 새로운 데이터를 추가하고 해당 데이터를 나타내는 URI를 리턴한다. 아래 소스는 사용자 사전 provider에 새로운 단어를 추가하는 것을 나타낸다.

```
// 삽입의 결과로 받을 URI를 저장할 변수
Uri mNewUri;

...

// 새로 추가할 데이터 객체의 값들을 저장하는 객체
ContentValues mNewValues = new ContentValues();

/*
 * 각 칼럼에 해당하는 값들을 삽입한다. "put" 함수로 "칼럼명"과 "값"을 파라미터로 넣는다.
 */
mNewValues.put(UserDictionary.Words.APP_ID, "example.user");
mNewValues.put(UserDictionary.Words.LOCALE, "en_US");
mNewValues.put(UserDictionary.Words.WORD, "insert");
mNewValues.put(UserDictionary.Words.FREQUENCY, "100");

mNewUri = getContentResolver().insert(
```

```
    UserDictionary.Word.CONTENT_URI,    // 사용자 사전의 URI
    mNewValues                          // 추가할 새로운 값들
);
```

새로운 행으로 추가할 데이터들은 ContentValues 객체 하나에 넣게 된다. 이 객체에 설정하는 칼럼들은 같은 꼭 실제 데이터형과 일치하지 않아도 되고, 전부다 설정하고 싶지 않고 null로 설정하고 싶을 경우, ContentValues.putNull()을 실행하면 된다.

위의 소스는 _ID 칼럼을 추가하지 않는데, 이 칼럼의 값은 자동으로 생성된다. Provider는 각 행마다 _ID에 대하여 고유한 값을 가지도록 선언한다. 이 칼럼은 일반적으로 테이블에서 PK로 활용된다.

리턴되는 컨텐트 URI는 newUri로 새로 추가된 행을 나타내도록 정의되고, 아래와 같이 나타낸다.
content://user_dictionary/words/<id_value>

<id_value>는 새로운 행의 _ID 칼럼의 값을 가진다. 대부분의 provider들은 이러한 URI를 자동으로 인식하여 해당 행을 다시 가져오는 요청으로 활용할 수 있다.

리턴된 Uri로부터 _ID값을 가져오기 위해서는 ContentUris.parseId() 함수를 호출하면 된다.

- **데이터 수정하기**

  행을 수정하기 위해서는 새로운 행을 추가했던 것과 똑같이 ContentValues 객체를 사용하여 수정하고, 수정 기준은 쿼리에서 사용했던 것과 동일하게 사용하면 된다. Provider를 사용하기 위해서는 ContentResolver.update() 함수를 사용하면 되고, 갱신하고 싶은 값들만 ContentValues 객체에 추가하면 된다. 만약 특정 칼럼의 값을 삭제하고 싶다면 null 로 설정하면 된다.

  아래는 locale이 "en"인 행들에 대하여 locale을 null로 설정하는 예이다. 리턴 값은 전체 수정된 행들의 수이다.

```
// 수정할 값들을 정의할 객체 생성
ContentValues mUpdateValues = new ContentValues();

// 수정할 행을 선택하는 기준을 정의하는 selection과 selection arguments
String mSelectionClause = UserDictionary.Words.LOCALE +  "LIKE ?";
String[] mSelectionArgs = {"en_%"};

// 몇 개의 행이 수정 되었는지 저장할 변수
int mRowsUpdated = 0;

...

/*
 * 수정할 칼럼에 값을 설정한다. (locale을 널로 수정)
 */
```

```
mUpdateValues.putNull(UserDictionary.Words.LOCALE);

mRowsUpdated = getContentResolver().update(
    UserDictionary.Words.CONTENT_URI,    // 사용자 사전을 나타내는 URI
    mUpdateValues                         // 수정할 칼럼의 값들
    mSelectionClause                     // selection 기준
    mSelectionArgs                        // selection 기준으로 비교할 값
);
```

위의 침입하는 사용자 입력 부분에서 설명했듯이, 사용자 입력을 selection arguments로 방지하는 것을 똑같이 구현하면 좋다.

- **데이터 삭제하기**

데이터를 삭제하는 것은 데이터를 가져오는 것과 유사하다. 지우고 싶은 행들을 기준들을 selectio으로 명시하고 실행하면 삭제된 행의 수를 리턴한다. 아래의 소스는 appid가 "user"와 동일한 행들을 지우는 예이다.

```
// 지우고자 하는 행들의 selection 변수를 정의
String mSelectionClause = UserDictionary.Words.APP_ID + " LIKE ?";
String[] mSelectionArgs = {"user"};

// 삭제된 행의 수를 저장할 변수 정의
int mRowsDeleted = 0;

...

// selection 기준에 맞는 행들을 삭제하는 함수
mRowsDeleted = getContentResolver().delete(
    UserDictionary.Words.CONTENT_URI,    // 사용자 사전을 나타내는 URI
    mSelectionClause                      // selection기준
    mSelectionArgs                        // selection기준에서 사용할 값
);
```

이번에는 유사하게 ContentResolver.delete()를 호출 할 때 침입 가능한 SQL injection을 막기 위하여 위와 같이 selection을 설정해주고 있다.

- **Provider의 데이터형**

컨텐트provider는 서로 다른 다양한 데이터 형을 지원해준다. 사용자 사전 provider는 문자열만 지원해주는데, provider들은 추가적인 데이터형을 더 지원해준다.

- integer
- long integer(long)
- floating point

- long floating point(double)

또 추가적인 다른 데이터형은 바로 BLOB(Binary Large OBject)이다. BLOB 형은 64KB 바이트 배열로 이루어져 있고, Cursor 크래스에서 get 함수를 통해서 가져올 수 있다.

provider에서 제공해주는 데이터형의 목록은 문서를 통해서 확인하면 되는데, 사용자 사전 provider에 대한 각각의 데이터형은 UserDictionary.Words에 대한 문서에서 참고할 수 있다. 또한 데이터형을 확인하기 위하여 Cursor.getType()을 사용할수도 있다.

Provider들은 각각의 URI에 대한 MIME 데이터형 정보를 가지고 있다. 따라서, MIME 정보를 보고 미리 앱에서 해당 데이터를 사용할 수 있는지 판별을 하거나, MIME형을 보고 어떠한 데이터를 처리할지 선택할 수도 있다. 일반적으로 복잡한 데이터형을 제공하는 provider에 대하여 MIME 정보가 필요할 것이다. 예를 들면, ContactsContract.Data 테이블은 각 행에 저장되어있는 주소록 정보를 MIME 정보로 구분시켜준다. URI에 해당하는 MIME 정보는 ContentResolver.getType()을 통해서 얻어올 수 있다.

- **Provider 접근 방법**

  Provider에 접근할 때 3가지의 방법을 통해서 데이터에 접근할 수 있다. 이는 앱 개발에 있어서 아주 중요할 것이다.

  - **배치실행** : ContentProviderOperation 클래스를 통해서 접근하는 쿼리를 배치로 생성한 다음, ContentResolver.applyBatch()를 통해서 배치 실행을 하면 된다.
  - **비동기 쿼리** : 쿼리를 별도의 쓰레드에 비동기적으로 실행하면 된다. 이것을 하는 방법은 CursorLoader객체를 사용하는 것이다.
  - **인텐트를 통한 데이터 접근** : 일반적으로는 provider에 바로 인텐트를 보내지는 못하지만, provider의 앱으로 보내는 것은 가능하다. 이것은 provider의 데이터를 수정할 때에 유용하게 활용 가능할 것이다.

- **배치실행**

  배치실행은 provider에 대량의 데이터를 삽입하거나 다수의 테이블에 많은 행들을 한번에 삽입할 때, 그리고 다수의 쿼리를 하나의 트랜젝션으로 관리하고자 할 때 유용하다. Provider를 배치모드로 실행하려고 한다면, ContentProviderOperation 객체의 배열을 만들어서, 이것을 ContentResolver.applyBatch()를 통해서 실행하면 된다. 이 때에는 특정 URI를 전달하는 것이 아니라, 컨텐트 provider의 authority를 넘겨주게 된다. 이것은 ContentProviderOperation객체의 배열이 서로 다른 테이블에 작업을 할 수 있도록 해주는 것이다. ContentResolver.applyBatch()는 처리 결과의 배열을 리턴해준다.

  ContactsContract.RawContacts 클래스는 배치로 삽입을 실행하는 것을 보여주고 있다.

- **인텐트를 통한 데이터 접근**

  인텐드는 컨텐트 provider에 접근을 제공해줄 수 있다. 앱이 접근 권한이 없더라도, 사용자가 데이터에 접근하여 인텐트로 결과를 받을 수 있도록 제공해준다. 또는 접근 권한이 있는 다른 앱을 작동 시켜서

사용자들이 해당 앱에서 작업을 하도록 할수도 있다.

- **임시 권한을 통한 접근**

  비록 앱에는 특정 컨텐트 provider에 접근 권한이 없더라도, 접근 권한이 있는 다른 앱에 인텐트를 보낸 다음, 인텐트에 대한 리턴 결과로 접근 권한이 있는 컨텐트 URI를 받을 수 있다. 이 URI는 인텐트를 받은 activity가 남아있을 때까지 활용 가능하다. 이 때에 접근 권한이 있는 앱에서는 인텐트에 flag를 설정해서 결과 인텐트를 전송해야 한다.

  - 읽기 권한: FLAG_GRANT_READ_URI_PERMISSION
  - 쓰기 권한: FLAG_GRANT_READ_URI_PERMISSION

  * 참고: 이 때에 전체 authority에 해당하는 접근 권한이 주어지는 것이 아니라, 해당 컨텐트 URI에 해당하는 URI 자체에 대한 접근 권한이 주어진다.

  Provider가 URI 권한을 설정하기 위하여 manifest 파일에서 <provider> 태그에 android:grantUriPermission 속성이나 <grant-uri-permission> 태그를 자식으로 가지고 있으면 된다. 예를 들면, 주소록 provider에서 READ_CONTACTS 권한이 없어도 데이터를 가져올 수 있다. 만약 누군가의 생일일 때 축하 메세지를 보내고 싶다면, READ_CONTACTS 권한을 요청하지 않는 대신, 사용자가 직접 어떠한 정보들을 사용할지 선택할 수 있도록 할 수 있다. 이를 위해서는 아래와 같은 순서를 거치면 된다.

  1. 앱에서 액션으로 ACTION_PICK와 "contacts"의 MIME 형인 CONTENT_ITEM_TYPE을 startActivityForResult()로 호출한다.
  2. 인텐트가 People 앱의 "selection" 액티비티와 일치하기 때문에, 이 액티비티가 전면으로 나오게 된다.
  3. Selection 액티비티에서 사용자가 contact를 선택하게 된다.
     이 때에, selection 액티비티는 setResult(resultCode, intent) 함수를 호출하여 앱으로 리턴할 인텐트를 설정한다.
     이 인텐트는 사용자가 선택한 contact들에 대한 접근 권한을 가지는 컨텐트 URI를 포함하고 있고, 추가적으로 "extras" 플래그에 FLAG_GRANT_READ_URI_PERMISSION이 설정된다. 이 플래그가 설정됨에 따라 컨텐트 URI를 수신하는 앱에서는 해당 URI를 통해서 데이터에 접근이 가능하다. 그리고 selection 액티비티에서는 finish() 함수를 호출하여 다시 원래의 앱으로 돌아간다.
  4. 앱이 다시 전면으로 나오게 되면, 시스템에서는 onActivityResult() 콜백 함수를 호출하게 되고, 이 함수에서는 결과 인텐트를 받게 된다.
  5. 결과 인텐트에 있는 컨텐트 URI에서는 contact 데이터를 주소록 provider로부터 읽어올 수 있으며, 이것은 앱의 manifest 파일에 읽기 권한을 요청하지 않아도 수행이 가능하다. 이렇게 함으로써 선택된 사람들에게 생일 정보를 확인하고 축하 메시지를 보낼 수 있게 될 것이다.

- **다른 앱 사용하기**

  Provider에 접근할 수 있는 권한이 없는데도 사용자가 데이터를 수정할 수 있도록 하는 방법은, 수정 권한이 있는 앱을 활성화 시켜서 사용자가 거기에서 작업을 할 수 있도록 하면 된다.

  예를 들면, 달력 앱은 ACTION_INSERT 인텐트를 수용하는데, 이는 달력 앱에서 insert에 해당하는 UI

를 활성화 시키는 것을 허용하는 것이다. 이 때에 "extras"를 통해서 이 인텐트에 추가적인 데이터를 넘겨줄 수 있고, UI에 미리 특정 데이터들을 적용시킬 수 있다. 일반적으로 되풀이되는 이벤트들은 복잡한 구문을 사용해야 하므로, 이벤트를 생성할 때에는 달력 앱을 활성화 시켜서 거기에서 이벤트를 생성하도록 하는 것이 효율적이다.

- Contract 클래스

Contract 클래스는 앱이 컨텐트 URI를 가지고 작업하는데, 칼럼 명이나 인텐트 action, 그리고 또 다른 도움이 되는 정보들을 주는 클래스이다. Contract 클래스는 자동적으로 provider에 포함되지 않고, provider의 개발자가 정의하고 이것이 다른 개발자들에게 사용 가능하도록 제공해줘야 한다. 안드로이드에 포함된 많은 provider들을 android.provider 패키지 안에 contract 클래스를 두고 있다.

예를 들면, 사용자 사전 provider는 UserDictionary라는 contract 클래스를 제공하여 컨텐트 URI와 칼럼명에 대한 정보를 알려주고 있다. 만약 "words" 테이블에 대한 컨텐트 URI를 알고 싶으면, UserDictionary.Words.CONTENT_URI를 참조하면 된다. UserDisctionary.Words 클래스 또한 칼럼명들을 상수로 가지고 있고, 이들은 위의 예제 들에서 사용되기도 했다. 예를 들면, 위에서 쿼리의 projection을 설정할 때 아래와 같은 배열을 정의했었다.

```
String[] mProjection =
{
    UserDictionary.Words._ID,
    UserDictionary.Words.WORD,
    UserDictionary.Words.LOCALE
};
```

다른 contract 클래스로는 ContactsContract 클래스가 주소록 provider를 위해서 있다. 이러한 클래스들은 레퍼런스 문서에서 사용하는 예가 나타나있고, ContactsContract.Intents.Insert는 인텐트를 위한 상수들과 인텐트 정보들을 포함하고 있다.

- MIME 타입 레퍼런스

컨텐트 provider들은 표준 MIME 미디어 타입이나 임의의 MIME 타입 문자열을 리턴할 수 있다. 기본적으로 MIME 타입은 아래와 같은 형식을 가지게 된다.

```
type/subtype
```

예를 들면 가장 잘 알려진 MIME 타입은 text/html 로 text가 type이면 html이 subtype이다. Provider가 이러한 MIME 타입을 URI에 대해 리턴하게 되면 이 URI는 html이 포함된 텍스트를 리턴한다는 것을 알 수 있다. 임의의 MIME 타입 문자열은, 다른 말로 "vendor-specific" MIME 타입이라고도 하는데, 이들은 더 복잡한 type과 subtype을 가지게 된다. 이 때에 type에는 항상 아래의 정보로 여러개의 행을 리턴할 때에는,

```
vnd.android.cursor.dir
```

또는 하나의 행을 나타내는 아래의 type을 가지게 된다.

vnd.android.cursor.item

subtype은 provider에 따라 다르게 나타난다. 안드로이드에 탑재되는 provider 들은 일반적으로 간단한 subtype을 가지게 된다. 예를 들면, 주소록 앱이 전화번호에 대한 행을 생성할 때 아래와 같은 MIME 타입을 가지게 된다.

vnd.android.cursor.item/phone_v2

이 때에 subtype의 값을 바로 phone_v2이다. 다른 provider 개발자들은 자기만의 subtype들을 authority나 테이블명에 따라서 생성해도 된다. 예를 들면, 기차의 시간표에 대한 provider가 있다고 한다면, 이 provider의 authority는 com.example.trains가 될 것이고, Line1, Line2, 그리고 Line3에 대한 테이블을 가지게 될 것이다. 이를 컨텐트 URI로 나타내면 아래와 같다.
content://com.example.trains/Line1

테이블 Line1의 provider는 아래와 같은 MIME 타입을 리턴한다.
vnd.android.cursor.dir/vnd.example.line1

아래의 컨텐트 URI에 대해서
content://com.example.trains/Line2/5

아래와 같은 MIME 타입을 가진다.
vnd.android.cursor.item/vnd.example.line2
대부분의 컨텐트 provider 들은 contract 클래스의 상수로 사용하는 MIME 타입을 가지고 있다.
주소록 provider에서는 contract 클래스로 ContactsContract.RawContacts를 보유한다. 여기에 CONTENT_ITEM_TYPE의 MIME 타입을 하나의 행을 가져올 때 쓰는 MIME 타입이다.