

**КАЛУЖСКИЙ ФИЛИАЛ  
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО  
ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ  
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
ИМЕНИ Н.Э. БАУМАНА (национальный исследовательский университет)»**



**Факультет** «Информатика и управление»

**Кафедра** "Программное обеспечение ЭВМ, информационные технологии"

# **Источники данных**

**Калуга**

По умолчанию отдельные базы данных приложений изолированы друг от друга, то есть их содержимое может быть использовано только приложением, которое создало ту или иную базу.

Однако Источники данных (**Content Providers**) обеспечивают возможность совместного использования баз данных приложений. Т.е. вы можете сконфигурировать собственный Content Provider, чтобы разрешить доступ к своим данным из других приложений, а также использовать Content Provider другого приложения для обращения к его хранилищу данных.

Этот механизм позволяет отделить логику приложения от данных, делая программы нечувствительными к источникам, из которых поступает информация, скрывая базовый источник данных.

Любое приложение с соответствующими полномочиями может добавлять, удалять или изменять данные, принадлежащие другому приложению.

Унифицированные идентификаторы содержимого (Content URI) в Android напоминают HTTP URI, но начинаются с **content** и строятся по следующему образцу:

***content://authority-name/path-segment1/path-segment2/etc..***

Пути URI должны быть уникальными и могут представляться двумя способами:

- `content://com.google.provider.NotePad/notes` - это запрос ко всем значениям определенного типа
- `content://com.google.provider.NotePad/notes/23` - запрос к определенной строке

# Создание нового источника данных

- Необходимо наследовать абстрактный класс **ContentProvider**. Переопределить метод **onCreate**, чтобы создать (и инициализировать) базовый источник, который вы хотите опубликовать
- Предусмотрите публичное статическое свойство, которое возвращает полный URI для данного источника. URI Источника данных должен быть уникальным, поэтому лучше привязать его к имени вашего пакета.
- Создайте и настройте **UriMatcher**, чтобы анализировать пути URI и распознавать их вид. В методе **addUri UriMatcher** подается комбинация: authority, path и константа. Причем, можно использовать спецсимволы: \* - строка любых символов любой длины, # - строка цифр любой длины.
- Назначьте имена каждому столбцу, доступному в вашем Источнике данных, чтобы упростить извлечение данных из результирующего Курсора.
- Для упрощения доступа к запросам и транзакциям в Источнике данных, можно реализовать методы **delete**, **insert**, **update** и **query**.
- После создания источника данных, его нужно добавить в манифест приложения.

```

public class MyContactsProvider extends ContentProvider {
    final String LOG_TAG = "myLogs";
    static final String DB_NAME = "mydb";
    static final int DB_VERSION = 1;
    static final String CONTACT_TABLE = "contacts";
    static final String CONTACT_ID = "_id";
    static final String CONTACT_NAME = "name";
    static final String CONTACT_EMAIL = "email";
    static final String DB_CREATE = "create table " + CONTACT_TABLE + "(" + CONTACT_ID +
" integer primary key autoincrement, " + CONTACT_NAME + " text, " + CONTACT_EMAIL + "
text" + ");";
    static final String AUTHORITY = "ru.startandroid.providers.AdressBook";
    static final String CONTACT_PATH = "contacts";
    public static final Uri CONTACT_CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/"
+ CONTACT_PATH);
    static final String CONTACT_CONTENT_TYPE = "vnd.android.cursor.dir/vnd." +
AUTHORITY + "." + CONTACT_PATH;
    static final String CONTACT_CONTENT_ITEM_TYPE = "vnd.android.cursor.item/vnd." +
AUTHORITY + "." + CONTACT_PATH;
    static final int URI_CONTACTS = 1;
    static final int URI_CONTACTS_ID = 2;
    private static final UriMatcher uriMatcher;

    static {
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI(AUTHORITY, CONTACT_PATH, URI_CONTACTS);
        uriMatcher.addURI(AUTHORITY, CONTACT_PATH + "/#", URI_CONTACTS_ID);
    }

    DBHelper dbHelper;
    SQLiteDatabase db;

```

```

public boolean onCreate() {
    Log.d(LOG_TAG, "onCreate");
    dbHelper = new DBHelper(getContext());
    return true;
}

public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {
    Log.d(LOG_TAG, "query, " + uri.toString());
    switch (uriMatcher.match(uri)) {
        case URI_CONTACTS:
            Log.d(LOG_TAG, "URI_CONTACTS");
            if (TextUtils.isEmpty(sortOrder)) {
                sortOrder = CONTACT_NAME + " ASC";
            }
            break;
        case URI_CONTACTS_ID:
            String id = uri.getLastPathSegment();
            Log.d(LOG_TAG, "URI_CONTACTS_ID, " + id);
            if (TextUtils.isEmpty(selection)) {
                selection = CONTACT_ID + " = " + id;
            } else {
                selection = selection + " AND " + CONTACT_ID + " = " + id;
            }
            break;
        default:
            throw new IllegalArgumentException("Wrong URI: " + uri);
    }
    db = dbHelper.getWritableDatabase();
    Cursor cursor = db.query(CONTACT_TABLE, projection, selection, selectionArgs, null, null, sortOrder);
    cursor.setNotificationUri(getContext().getContentResolver(), CONTACT_CONTENT_URI);
    return cursor;
}

```

```

public Uri insert(Uri uri, ContentValues values) {
    Log.d(LOG_TAG, "insert, " + uri.toString());
    if (uriMatcher.match(uri) != URI_CONTACTS)
        throw new IllegalArgumentException("Wrong URI: " + uri);
    db = dbHelper.getWritableDatabase();
    long rowID = db.insert(CONTACT_TABLE, null, values);
    Uri resultUri = ContentUris.withAppendedId(CONTACT_CONTENT_URI, rowID);
    getContext().getContentResolver().notifyChange(resultUri, null);
    return resultUri;
}

```

```

public int delete(Uri uri, String selection, String[] selectionArgs) {
    Log.d(LOG_TAG, "delete, " + uri.toString());
    switch (uriMatcher.match(uri)) {
        case URI_CONTACTS:
            Log.d(LOG_TAG, "URI_CONTACTS"); break;
        case URI_CONTACTS_ID:
            String id = uri.getLastPathSegment();
            Log.d(LOG_TAG, "URI_CONTACTS_ID, " + id);
            if (TextUtils.isEmpty(selection)) {
                selection = CONTACT_ID + " = " + id;
            } else {
                selection = selection + " AND " + CONTACT_ID + " = " + id;
            } break;
        default:
            throw new IllegalArgumentException("Wrong URI: " + uri);
    }
    db = dbHelper.getWritableDatabase();
    int cnt = db.delete(CONTACT_TABLE, selection, selectionArgs);
    getContext().getContentResolver().notifyChange(uri, null);
    return cnt;
}

```

```

public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs) {
    Log.d(LOG_TAG, "update, " + uri.toString());
    switch (uriMatcher.match(uri)) {
        case URI_CONTACTS:
            Log.d(LOG_TAG, "URI_CONTACTS"); break;
        case URI_CONTACTS_ID:
            String id = uri.getLastPathSegment();
            Log.d(LOG_TAG, "URI_CONTACTS_ID, " + id);
            if (TextUtils.isEmpty(selection)) {
                selection = CONTACT_ID + " = " + id;
            } else {
                selection = selection + " AND " + CONTACT_ID + " = " + id;
            } break;
        default:
            throw new IllegalArgumentException("Wrong URI: " + uri);
    }
    db = dbHelper.getWritableDatabase();
    int cnt = db.update(CONTACT_TABLE, values, selection, selectionArgs);
    getContext().getContentResolver().notifyChange(uri, null);
    return cnt;
}

```

```

public String getType(Uri uri) {
    Log.d(LOG_TAG, "getType, " + uri.toString());
    switch (uriMatcher.match(uri)) {
        case URI_CONTACTS:
            return CONTACT_CONTENT_TYPE;
        case URI_CONTACTS_ID:
            return CONTACT_CONTENT_ITEM_TYPE;
    }
    return null;
}

```

# Использование источника данных

```
public class MainActivity extends Activity {
```

```
    final String LOG_TAG = "myLogs";
```

```
    final Uri CONTACT_URI = Uri.parse("content://ru.startandroid.providers.AdressBook/contacts");
```

```
    final String CONTACT_NAME = "name";
```

```
    final String CONTACT_EMAIL = "email";
```

```
@Override
```

```
public void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.main);
```

```
    Cursor cursor = getContentResolver().query(CONTACT_URI, null, null, null, null);
```

```
    startManagingCursor(cursor);
```

```
    String from[] = {"name", "email"};
```

```
    int to[] = {android.R.id.text1, android.R.id.text2};
```

```
    SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,  
android.R.layout.simple_list_item_2, cursor, from, to);
```

```
    ListView lvContact = (ListView) findViewById(R.id.lvContact);
```

```
    lvContact.setAdapter(adapter);
```

```
}
```

```
public void onClickInsert(View v) {
```

```
    ContentValues cv = new ContentValues();
```

```
    cv.put(CONTACT_NAME, "name 4");
```

```
    cv.put(CONTACT_EMAIL, "email 4");
```

```
    Uri newUri = getContentResolver().insert(CONTACT_URI, cv);
```

```
    Log.d(LOG_TAG, "insert, result Uri : " + newUri.toString());
```

```
}
```



```

public void onClickUpdate(View v) {
    ContentValues cv = new ContentValues();
    cv.put(CONTACT_NAME, "name 5");
    cv.put(CONTACT_EMAIL, "email 5");
    Uri uri = ContentUris.withAppendedId(CONTACT_URI, 2);
    int cnt = getContentResolver().update(uri, cv, null, null);
    Log.d(LOG_TAG, "update, count = " + cnt);
}

```

```

public void onClickDelete(View v) {
    Uri uri = ContentUris.withAppendedId(CONTACT_URI, 3);
    int cnt = getContentResolver().delete(uri, null, null);
    Log.d(LOG_TAG, "delete, count = " + cnt);
}

```

```

public void onClickError(View v) {
    Uri uri = Uri.parse("content://ru.startandroid.providers.AdressBook/phones");
    try {
        Cursor cursor = getContentResolver().query(uri, null, null, null, null);
    } catch (Exception ex) {
        Log.d(LOG_TAG, "Error: " + ex.getClass() + ", " + ex.getMessage());
    }
}
}
}

```

```

<provider android:authorities="com.example.mycontentprovider.providers.AdressBook"
          android:name="MyContent">
</provider>

```

12:55   



# ContProvClient

INSERT

UPDATE

DELETE

ERROR

Name1

Email1

Name2

Email2

Name3

Email3

# Встроенные источники данных

- **Browser.** Используется для чтения или изменения закладок, истории посещений или использования поиска в обозревателе.
- **CallLog.** Выводит или обновляет историю звонков (входящие, исходящие, пропущенные), открывает доступ к детальной информации, такой как номер звонившего и продолжительность разговора.
- **ContactsContract.** Используется для получения, изменения или хранения информации о контактах.
- **MediaStore.** Предоставляет централизованный, управляемый доступ к файлам мультимедиа на вашем устройстве, включая аудио, видео и изображения. Вы можете хранить в нем собственные файлы мультимедиа и делать их общедоступными.
- **Settings.** Позволяет получить доступ к настройкам устройства с помощью этого Источника данных. Предоставляется возможность просматривать большинство системных настроек, а некоторые из них даже изменять. Класс `android.provider.Settings` содержит коллекцию действий для Намерений, которые могут использоваться для открытия соответствующего экрана с настройками, чтобы пользователь мог их поменять.
- **UserDictionary.** Обеспечивает доступ к пользовательским наборам слов, добавленных в словарь для интеллектуального ввода текста.