ТЕХНОТРЕК

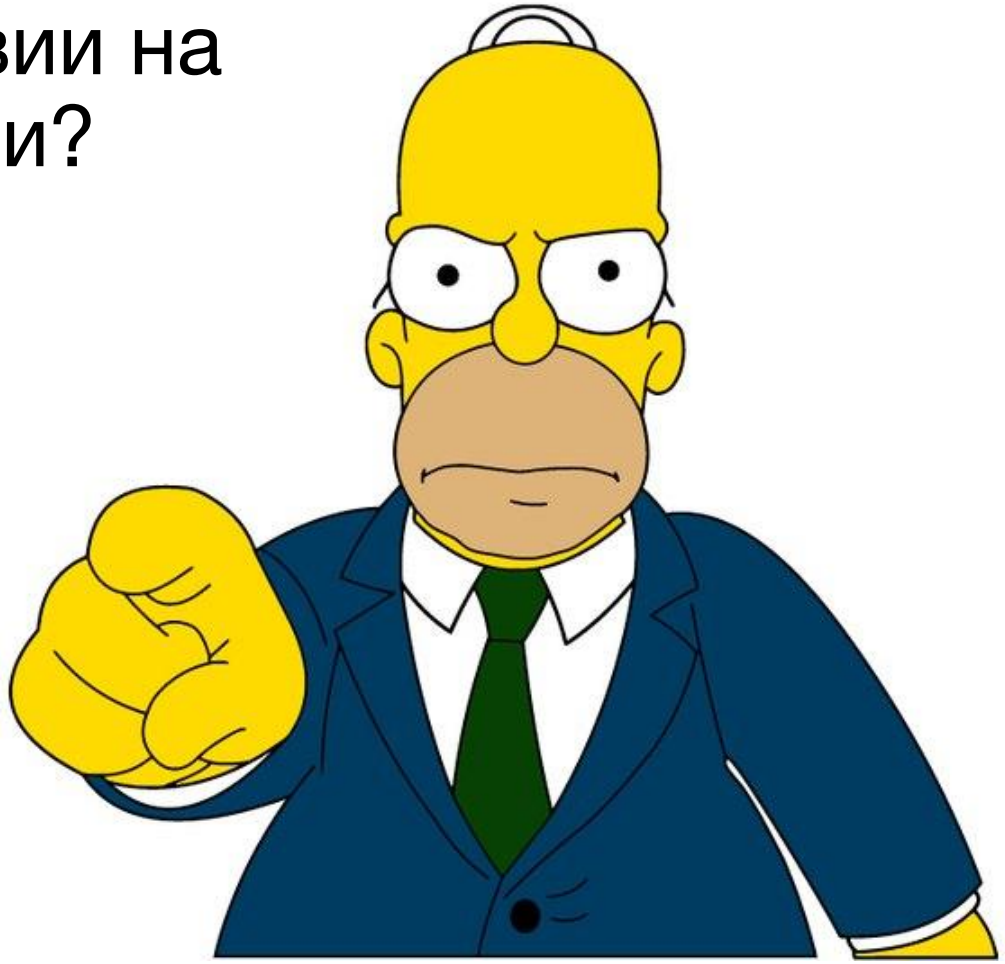
Занятие №8

Разработка приложений на Android

Кирилл Филимонов
Юрий Береза

Напоминание

А ты отметился о присутствии на занятии?



Agenda

1. Работа с файлами
2. LRU Cache
3. Shared Preferences
4. SQLite
5. Content Providers

Работаем с файлами

- Assets
- Internal Storage
- External Storage

Assets

Плюсы
- Сразу идут с приложением
- Всегда есть в наличии

Минусы
- Сразу идут с приложением
- Нельзя модифицировать

```
fun loadBitmapFromAssets(context : Context, filename : String) : Bitmap? {
    try {
        val inputStream = context.assets.open("images/" + filename)
        BitmapFactory.Options opt = BitmapFactory.Options()
        return BitmapFactory.decodeStream(inputStream , null, opt)
    }
    catch (e : IOException) {
        e.printStackTrace()
    }
    return null
}
```

Internal Storage

Плюсы
- Всегда доступны (почти)

Минусы
- Необходимо туда загрузить
- Занимают место

```
context.cacheDir context.filesDir contex.fileStreamPath
```

```
fun loadBitmapFromCacheDir(context: Context, filename: String): Bitmap? {
    val file = File(context.cacheDir, filename)
    if (file.exists()) {
        try {
            val inputStream = FileInputStream(file)
            val opt = BitmapFactory.Options()
            return BitmapFactory.decodeStream(inputStream, null, opt)
        } catch (e: FileNotFoundException) {
            e.printStackTrace()
        }
    }
    return null
}
```

External Storage

Плюсы
- Доступно больше места

Минусы
- Необходимо туда загрузить
- Не всегда доступны

```
context.externalCacheDir context.externalFileDir Environment.getExternalStorageDirectory()
```

```
fun loadBitmapFromCacheDir(context: Context, filename: String): Bitmap? {
    val state = Environment.getExternalStorageState()
    if (state == Environment.MEDIA_MOUNTED) {
        val cacheDir = Environment.getExternalStorageDirectory()
        val file = File(cacheDir, filename)
        if (file.exists()) {
            //TODO
        }
    }
    return null
}
```

LruCache

- **LRU** (least recently used) — это алгоритм, при котором вытесняются значения, которые дольше всего не запрашивались.
- Мы резервируем какой-то объем памяти для хранения картинок и отдает на откуп этому классу управление кешем.
- Прежде чем загружать всегда проверяем кэш.

```
private fun createLRUCache() : LruCache<String, BitmapDrawable>  {
    val maxMemory = (Runtime.getRuntime().maxMemory() / 1024).toInt()
    val cacheSize = maxMemory / 8
    return object : LruCache<String, BitmapDrawable>(cacheSize) {
        override fun sizeOf(key: String, bitmap: BitmapDrawable): Int {
            return bitmap.bitmap.rowBytes * bitmap.bitmap.height / 1024
        }
    }
}

fun getBitmapDrawable(key : String) : BitmapDrawable? = cache.get(key)

fun putBitmapIntoCache(key: String, bitmap: Bitmap) {
    cache.put(key, BitmapDrawable(resources, bitmap))
}
```

Shared Preferences

- Хранение данных внутри приложения
- Что хранить в Shared Preferences
- А что не хранить
- В чем удобство использования Shared Preferences

Shared Preferences

```
val prefs = requireContext().getSharedPreferences("settings", Context.MODE_PRIVATE)

val prefs = requireActivity().getPreferences(Context.MODE_PRIVATE)

val editor = prefs.edit()
editor.putString("greetings", "Hello, World!")
editor.apply()

val greetings = prefs.getString("greeting", "default value")
```

Storage

Demo!

SQLite

- SQLite – встраиваемая реляционная база данных
- Все хранит в одном файле
- Этот файл можно править различными инструментами
- Позволяет читать из множества потоков, но лочится на запись
- Есть некоторые проблемы с локализацией
- Надо помнить об эффективности запросов

Работа с базой в Android

- ContentValues – для вставки и обновления данных
- Cursor (SQLiteCursor) – для обработки выборки
- SQLiteOpenHelper – для создания, открытия и обновления версий базы
  - `fun onCreate(db : SQLiteDatabase)` – вызывается при создании базы когда ее еще нету
  - `fun onUpgrade(db : SQLiteDatabase, oldVersion : Int, newVersion : Int)` – вызывается если в конструкторе была передана версия больше чем текущая
- SQLiteDatabase – класс работы с базой
  - query– выбор из базы SELECT
  - delete – удаление DELETE
  - update – обновление UPDATE
  - insert – вставка нового значения INSERT
  - execSql – выполнить произвольный запрос
  - replace – аналог запроса INSERT OR REPLACE

SQL Injection

- Внедрение кода в ваш запрос
- Не доверяйте пользовательским данным
- Не вводите их напрямую в запросы
- Активно используйте значения аргументов фильтра в функциях query, delete, update, execSQL, rawQuery

Неправильно:

```
Db.execSQL("DELETE FROM some_table WHERE name = '" +
userInputString + "' ")
```

Правильно:

```
Db.execSQL("DELETE FROM some_table WHERE name = ?",
arrayOf(userInputString))
```

query

```
fun query(
    table : String,
    columns : Array<String>,
    selection : String,
    selectionArgs : Array<String>,
    groupBy : String,
    having : String,
    orderBy : String,
    limit : String
) : Cursor
```

- **table** — имя таблицы, к которой передается запрос;
- **columns** — список имен возвращаемых полей. При передаче null возвращаются все столбцы;
- **selection** — параметр, формирующий выражение WHERE
- **selectionArgs** — значения аргументов фильтра;
- **groupBy** - параметр, формирующий выражение GROUP BY
- **having** — параметр, формирующий выражение HAVING
- **sortOrder** — параметр, форматирующий выражение ORDER BY
- **limit** - параметр ограничивающий количество строк в выдаче

insert

```
fun insert(table : String, nullColumnHack : String,
values : ContentValues) : Long

fun insertOrThrow(table : String, nullColumnHack : String,
values : ContentValues) : Long

fun insertWithOnConflict(table : String, nullColumnHack : String,
initialValues : ContentValues, conflictAlgorithm : Int) : Long
```

- **table** — имя таблицы, в которую будет вставлена запись;
- **nullColumnHack** — в базе данных SQLite не разрешается вставлять полностью пустую строку, и если строка, полученная от клиента контент-провайдера, будет пустой, то только этому столбцу явно будет назначено значение null;
- **values** — карта отображений (класс Map и его наследники), которая содержит пары ключ-значение. Ключи в карте должны быть названиями столбцов таблицы, значения — вставляемыми данными.
- **conflictAlgorithm** – как обрабатывать конфликты (replace, rollback, ignore none)
- возвращает идентификатор _iD вставленной строки или -1в случае ошибки.

delete и update

```
fun update(
    table : String,
    values : ContentValues,
    whereClause : String,
    whereArgs : Array<String>
) : Int

fun delete(
    table : String,
    whereClause : String,
    whereArgs : Array<String>
) : Int
```

- **table** — имя таблицы, к которой передается запрос;
- **whereClause** — параметр, формирующий выражение WHERE
- **whereArgs**— значения аргументов фильтра;
- **values** — значения
- возвращают количество измененных или удаленных строк

execSQL и rawQuery

```
fun execSQL(sql : String)
fun execSQL(sql : String, bindArgs : Array<Any>)
fun rawQuery(sql : String, selectionArgs : Array<String>) : Cursor
```

- **sql** — запрос
- **bindArgs** — значения аргументов фильтра;
- execSQL ничего не возвращает
- execSQL не рекомендуется для SELECT, UPDATE, DELETE

Как заполнять базу

Из других данных

1. Создать xml или json с данными, положить в assets
2. Заполнять ее инсертами в onCreate

Из файла базы

3. Создать базу вне приложения заполнить ее
4. Положить файл базы в assets и затем скопировать ее по значению
5. Заархивировать и положить в raw, затем скопировать как файл на sdcard

19

Особенности работы

- **database is locked** – возникает при многопоточной записи в базу.
- **database is closed** – может возникнуть при работе с базой из разных частей программы, например, Activity и Service.
- **corrupted database** – возникает, если файл базы данных был испорчен либо пользователем, либо при неожиданном прерывании записи в базу
- **низкая производительность** при работе с базой данных

Database is locked

- блокировки в SQLite выполнены на уровне файла.
- читать базу может много потоков, а писать только один
- если вы пишете из двух потоков одного соединения, то один поток будет ждать, пока закончит писать другой.
- если вы пишете из двух потоков разных соединений, то произойдет ошибка – приложение вылетит с *SQLiteDatabaseLockedException*.
- приложение всегда должно иметь только один экземпляр *SQLiteOpenHelper*(именно открытого соединения), иначе в любой момент может возникнуть *SQLiteDatabaseLockedException*.
- *SQLiteOpenHelper* имеет 2 метода предоставляющих доступ *SQLiteOpenHelper.getReadableDatabase()==SQLiteOpenHelper.getWritableDatabase()*
- внутри класса *SQLiteDatabase* есть собственные блокировки – переменная mLock.
- поскольку на чтение и запись экземпляр *SQLiteDatabase* один, то чтение данных тоже блокируется.

Database is closed

- Возникает когда с базой работает и активити и сервисы
- При каждом обращении к базе проверять,- закрыта база или нет, и если закрыта, то переоткрывать её заново.
- Принудительно добавить фиктивную ссылку на базу и держать её пока база используется
- Использовать ContentProvider для доступа к базе. Причем желательно использовать именно один провайдер – это легко реализовать, поскольку ему можно добавить неограниченное количество Uri

Corrupted database

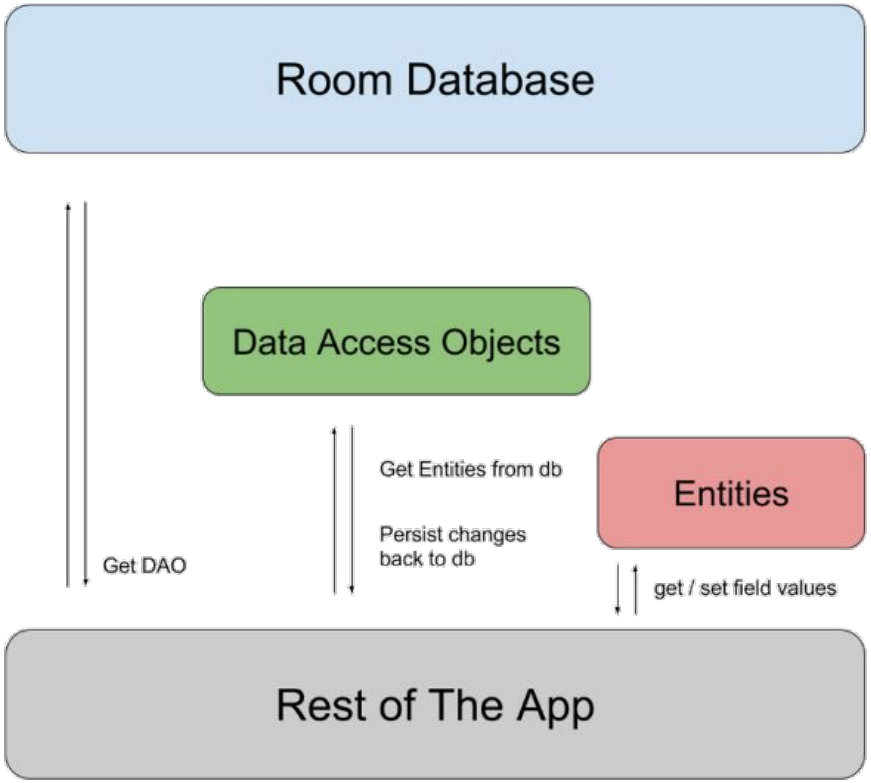
- Причина – испортился файл базы
  - Проблемы с питанием
  - Падение приложения
  - Глюки устройства
- Новую базу устройство создаст само на onCreate
- Можно вызвать VACUUM – база будет пересоздана

Оптимизация работы с базой

- Не пишите запросы которые возвращают больше 1000 строк или мегабайта данных
- Не используйте смещение в LIMIT
- Если нужно создать таблицу из существующей используйте INSERT AS SELECT
- Очищайте память после больших запросов вызовом *SQLiteDatabase.releaseMemory()*
- Используйте индексы
- Не используйте like
- В условиях ставьте легкие или индексированные запросы сначала
- Для интернациональных раскладок некорректно обрабатываются заглавные буквы в запросе.
- Правильно использовать JOIN
- Избегать фрагментации

Room

Room - это библиотека, предоставляющий абстрактный слой над SQLite, обеспечивающая удобный доступ к базе данных, прячущая сложность работы с SQLite



Room - компоненты

- Database - Объект базы данных, который держит соединение с базой данных SQLite и все операции осуществляются через него. Объявляется с помощью аннотации `@Database`
- Entity - Объект, представляющий собой запись в таблице базы данных. Объявляется с помощью аннотации `@Entity`
- DAO - Интерфейс, описывающий методы взаимодействия с базой данных. Объявляется с помощью аннотации `@Dao`

Room - добавляем в проект

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'
apply plugin: 'kotlin-kapt'

android {
    //..
}

dependencies {
    implementation "android.arch.persistence.room:runtime:1.1.1"
    kapt "android.arch.persistence.room:compiler:1.1.1"
}
```

Room - Entity

```
@Entity(tableName = "test")
data class Value(
        @PrimaryKey(autoGenerate = true) val id : Int,
        @ColumnInfo(name = "value") val text : String
)
```

Room - DAO

```
@Dao
interface ValuesDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertValue(value: Value)

    @Update
    fun updateValue(value: Value)

    @Delete
    fun deleteValue(value: Value)

    @Query("SELECT * FROM test WHERE value == :text")
    fun getValueByName(text: String): List<Value>

    @Query("SELECT * FROM test")
    fun getValues(): List<Value>

    @Query("SELECT Count(*) FROM test")
    fun size() : Long
}
```

Room - Database

```
@Database(entities = [Value::class], version = 1)
abstract class ValuesDatabase : RoomDatabase() {
    abstract fun valuesDao(): ValuesDao

    companion object {
        var INSTANCE: ValuesDatabase? = null
        fun getAppDataBase(context: Context): ValuesDatabase? {
            if (INSTANCE == null){
                synchronized(ValuesDatabase::class){
                    INSTANCE = Room.databaseBuilder(context.applicationContext,
                                                    ValuesDatabase::class.java,
                                                    "values").build()
                }
            }
            return INSTANCE
        }

        fun closeDatabase(){
            INSTANCE?.close()
            INSTANCE = null
        }
    }
}
```

Database

Demo!

Контент провайдер

- Что такое и для чего
- Какие есть в системе
- Как использовать
- Как создавать самому
- Какие есть подводные камни
- Как избежать кражи данных

Content Providers

- Механизм обмена данными между процессами
- Позволяет использовать данные системы
- Процесс может пользоваться контентом другого процесса
- Процесс может предоставлять свои данные другим процессам
- Позволяет гибко настраивать доступ

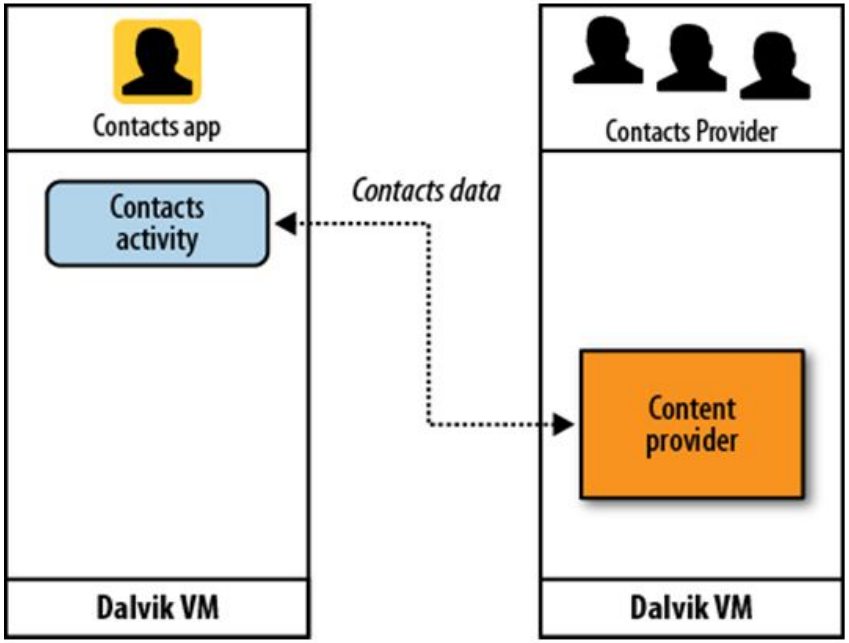
Как он работает

- Обращение происходит по специальному REST-подобному URI вида content://providerName/table
- Все данные организованны как таблицы
- Запросы похожи на SQL запросы
- Вы делаете запрос, получаете данные и работаете с ним

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	5

Какие есть в системе провайдеры

- Browser
- CallLog
- Contacts (People, Phones, Photos, Groups)
- MediaStore (Audio,Images, Video)
- Settings



Как использовать

- Запрашиваем разрешение в манифесте
- Формируем нужный нам URI может быть пути или конкретных данных
- На чтение
  - Делаем запрос в ответ получаем курсор
  - Читаем последовательно
- Вставка данных
  - Делаем insert
- Обновление данных
  - Делаем update
- Удаление
  - Делаем delete

Разрешения

Разрешения нужны для системных провайдеров

Пишутся в манифесте в разделе <user-permission…>

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

Нужно для того чтобы информировать пользователя , устанавливающего
приложение, о том что его данные могут быть прочитаны или изменены

Формируем правильный URI

- URI контент провайдера представляет собой уникальные в системе идентификатор
- Включает имя провайдера
- Имя таблицы в провайдере

Пример content://user_dictionary/words

content:// - всегда присутствует, говорит о том что это URI контента

user_dictionary - идентификатор провайдера

words - имя таблицы

- Может быть ссылкой на данные

Пример content://user_dictionary/words/23

23 - элемент с _ID = 23

Запрашиваем данные

```
fun query (uri : Uri, projection : Array<String>, selection : String,
           selectionArgs : Array<String>,
           sortOrder : String) : Cursor
```

```
// Queries the user dictionary and returns results
val cursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,   // The content URI of the words table
    projection,                         // The columns to return for each row
    selectionClause                     // Selection criteria
    selectionArgs,                      // Selection criteria
    sortedOrder);                       // The sort order for the returned rows
```

Query()	SELECT
Uri	FROM table_name
projection	Col, col, col, ….
selection	WHERE col= value
selectionArgs	Альтернатива ? при использовании placeholder
sortOrder	ORDER BY col, col, …

Код запроса к провайдеру

```
var selectionArgs = arrayOf("")
val searchString = searchWord.text.toString()

if (TextUtils.isEmpty(searchString)) {
    selectionClause = null
} else {
    selectionClause = UserDictionary.Words.WORD + " = ?"
    selectionArgs = arrayOf(mSearchString)
}

cursor = getContentResolver().query(…);
cursor?.run {
    if (count < 1) {
        // Пустой курсор, ничего не нашли по запросу
    }
    else {
        // Обрабатываем полученный данные
    }
}
```

Безопасность запросов

Никогда не используйте напрямую пользовательские данные в запросе

```
val selectionClause =  "var = " + userInput
```

Вместо этого используйте поле для аргументов

```
val selectionClause =  "var = ?”;
val selectionArgs = arrayOf(userInput)
```

Отображение данных

Через курсор адаптер с помощью контейнера (ListView например)

```
val wordListColumns = arrayOf(
    UserDictionary.Words.WORD,
    UserDictionary.Words.LOCALE
)

val wordListItems = arrayOf(R.id.dictWord, R.id.locale)
val cursorAdapter = SimpleCursorAdapter(getApplicationContext(),
    R.layout.wordlistrow, cursor, wordListColumns, wordListItems, 0)
```

Отображение данных

Самостоятельно как пожелаете

```
cursor?.run {
    while (moveToNext()) {
        val word = getString(index)
    }
}
```

Вставка данных

Используем ContentResolver.insert()

```
val uri: Uri

val values = ContentValues()

values.put(UserDictionary.Words.APP_ID, "example.user")
values.put(UserDictionary.Words.LOCALE, "en_US")
values.put(UserDictionary.Words.WORD, "insert")
values.put(UserDictionary.Words.FREQUENCY, "100")

uri = getContentResolver().insert(
        UserDictionary.Word.CONTENT_URI,
        values
)
```

Обновление данных

Используем ContentResolver.update()

```
val updateValues = ContentValues()

val selectionClause = UserDictionary.Words.LOCALE +  "LIKE ?"
val selectionArgs = arrayOf("en_%")
val rowsUpdate = 0
updateValues.putNull(UserDictionary.Words.LOCALE)

rowsUpdate = getContentResolver().update(
        UserDictionary.Words.CONTENT_URI,
        updateValues,
        selectionClause,
        selectionArgs,
)
```

Удаление данных

Используем ContentResolver.delete()

```
val selectionClause = UserDictionary.Words.APP_ID + " LIKE ?"
val selectionArgs = arrayOf("user")

val rowsDeleted = 0

rowsDeleted = getContentResolver().delete(
        UserDictionary.Words.CONTENT_URI,
        selectionClause,
        selectionArgs
)
```

Создание собственного провайдера

- Как понять, что нужен именно он
- Понять как данные будут хранится
- Выбрать имя для провайдера и таблиц
- Реализовать собственно провайдер
- Реализовать дополнительные классы

А нужен ли контент провайдер

- Вы собираетесь отдавать данные в другое приложение
- Вы собираетесь отдавать сложные данные
- Вы собираетесь отдавать потенциально много данных
- Вы собираетесь отдавать данные или файлы по запросу от другого приложения

Вам не нужен контент провайдер если вы собираетесь работать с SQLite только в вашем приложении в достаточно простой конфигурации

Данные

Файлы:

Таблицы:

CategoryID	ParentID	Title	SortOrder
1	0	Electronics	9835
2	1	Mobile Phones	10000
3	1	DVD Systems	10100
4	2	Sony Ericsson	10000
5	2	Nokia	10100
6	2	Motorola	10200
7	2	Samsung	10300
8	0	Apparel	100
9	8	John Players	10000
10	8	Women Sarees	10100
11	9	Shirts	10000
12	9	Pants	10100
13	10	Banarasi Sarees	10000
14	10	Kurta Salwar	10100
16	0	Beverages	9975
17	0	Computer and Accessories	55555
18	0	Baby Items	10400.2
19	0	Health and Beauty	10400.4
20	0	Jewelry	10193.85

URI

- Должно быть уникальным в системе
- Рекомендуется использовать applicationID, например com.example.<appname>.provider
- Имена таблиц лучше выбирать исходя из данных за который они отвечают
- Путь URI может содержать много сегментов
- Каждый уровень URI не означает таблицу

content://com.example.app.provider/table1
content://com.example.app.provider/table2/dataset1
content://com.example.app.provider/table2/dataset2

Манифест

android:authorities - идентификатор провайдера

android:name - класс обработчик

android:process – принцип работы похож на service, начало “:” заставит создать новый именованный процесс

android:readPermission

android:writePermission – строка , разрешения нужные для процесса который будет читать или писать в этот провайдер

```
<provider android:authorities="list"
        android:enabled=["true" | "false"]
        android:exported=["true" | "false"]
        android:grantUriPermissions=["true" | "false"]
        android:icon="drawable resource"
        android:initOrder="integer"
        android:label="string resource"
        android:multiprocess=["true" | "false"]
        android:name="string"
        android:permission="string"
        android:process="string"
        android:readPermission="string"
        android:syncable=["true" | "false"]
        android:writePermission="string" >
    . . .
</provider>
```

Класс реализующий функционал

```
class MyContactsProvider : ContentProvider() {
    override fun onCreate(): Boolean { }

    override fun query(
            uri: Uri,
            projection: Array<String>,
            selection: String,
            selectionArgs: Array<String>,
            sortOrder: String): Cursor { }

    override fun insert(uri: Uri, values: ContentValues): Uri { }
    override fun delete(uri: Uri, selection: String, selectionArgs: Array<String>): Int { }
    override fun getType(uri: Uri): String { }
    override fun update(uri: Uri, values: ContentValues, selection: String,
              selectionArgs: Array<String>): Int { }
}
```

MIME type

Для данных таблиц и прочего - getType(URI)

- Начинается с vnd
- Для одной строки android.cursor.item/
- Для более чем одной строки android.cursor.dir/
- Путь специфичный для вашего приложения

vnd.android.cursor.**dir**/vnd.com.example.provider.table1
vnd.android.cursor.**item**/vnd.com.example.provider.table1

Для файлов - используется getStreamType(URI)

- Обычные MIME типы файлов

```
{ "image/jpeg", "image/png", "image/gif"}
```

Проблемы большинства реализаций

- Несанкционированный доступ к персональным данным пользователя и чувствительной информации.
  - Пермишены
- Уязвимости типа SQL injection в провайдерах, работающих с базами данных.
  - Конструкции типа “val = “ + userData

ТЕХНОТРЕК

Спасибо за внимание!

Кирилл Филимонов Kirill.Filimonov@gmail.com

Юрий Береза – ybereza@gmail.com