

# Хранение данных в Android

Говоровский Андрей



Не забываем  
отмечаться на занятии :)

А что было в  
прошлый раз?



# Agenda

Какие данные храним

SharedPreferences

Internal/External storage

AccountManager

SQLite & ContentProvider

ORM & Room

# Зачем хранить данные





# Какие данные обычно храним

1

Кеширование  
контента вашего API

2

Настройки  
приложения  
(например тема,  
шрифт)

3

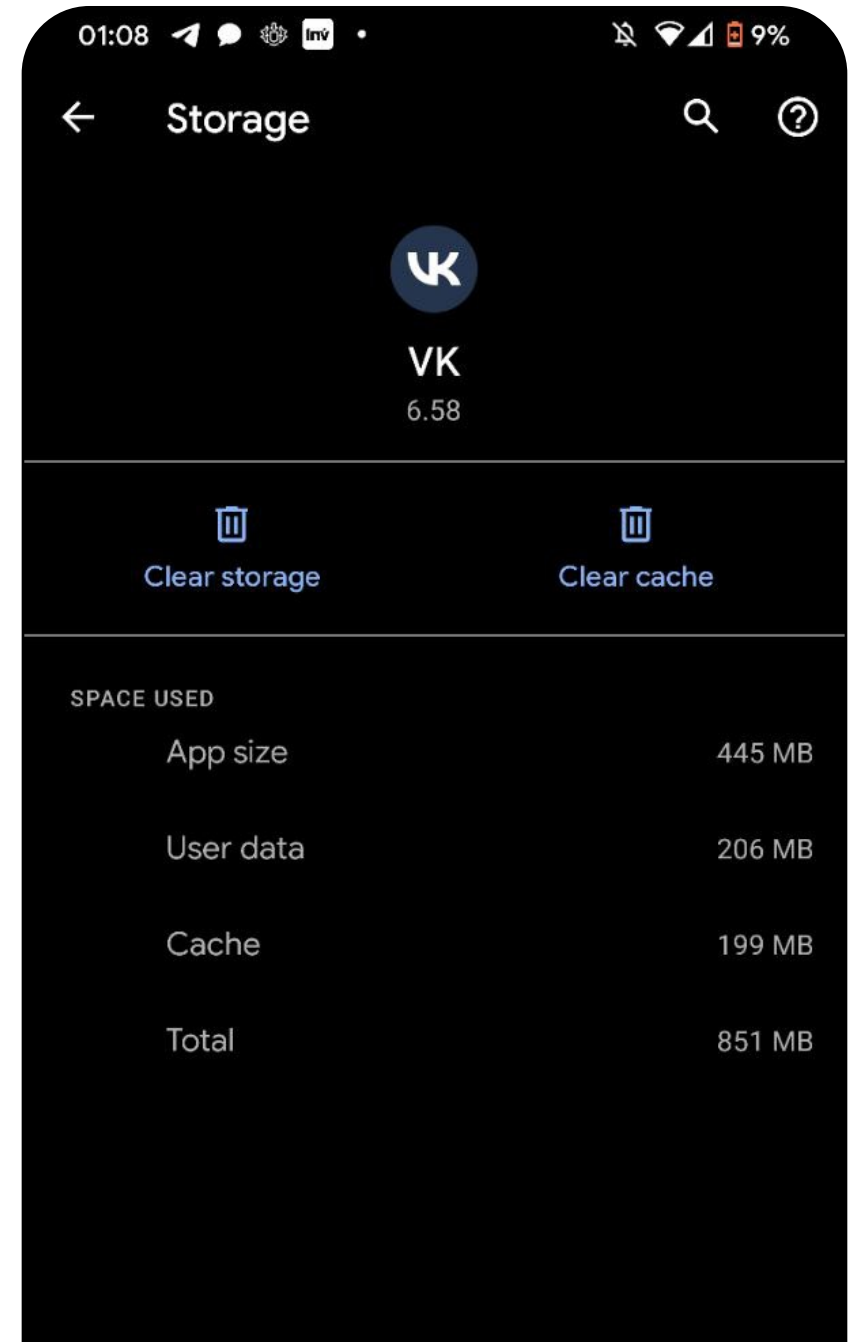
Авторизация/токены/  
device\_id/etc

4

Контент, который  
создает сам юзер  
(фоторедактор etc)

# Какие есть проблемы с хранением данных

- Пользователь может очистить данные и кеш в приложении
- Бесконечный рост данных
- Безопасность хранения и ограничение доступа
- Доступность данных



# SharedPreferences





# Shared Preferences

В `SharedPreferences` можно хранить следующие типы данных:

- Примитивные типы (`boolean`, `float`, `int`, `long`)
- Строки (`String`)
- Множества строк (`Set<String>`)

# Shared Preferences

Для работы есть три метода:

1. `getSharedPreferences(String name, int mode)`

2. `getPreferences(int mode)`

3. `getDefaultSharedPreferences(Context context)`

Режимы работы:

- `MODE_PRIVATE`
- `MODE_WORLD_READABLE`
- `MODE_WORLD_WRITEABLE`

# Shared Preferences

Пример работы чтение -

```
const val PREFS_NAME = "MyPrefsFile"  
val settings = getSharedPreferences(PREFS_NAME, 0)  
val silent = settings.getBoolean("silentMode", false)
```

Запись -

```
val settings = getSharedPreferences(PREFS_NAME, 0)  
val editor = settings.edit()  
editor.putBoolean("silentMode", silentMode)  
editor.apply()
```

# Internal Storage



# Internal Storage

Доступ к файлам в Internal Storage имеет только ваше приложение. Пользователь (в общем случае) доступа не имеет.

Чтение из хранилища:

```
FileInputStream openFileInput(String name)
```

Запись в хранилище:

```
FileOutputStream openFileOutput(String name,  
int mode)
```

# Internal Storage

- `File getFilesDir()` – путь до приватной директории приложения
- `File getDir(String name, int mode)` - открывает/создает директорию в приватном хранилище
- `File getCacheDir()` – путь до директории для хранения кэшей
- `boolean deleteFile(String name)` – удаляет приватный файл
- `String[] fileList()` – список приватных файлов

# External Storage



# External Storage

Добавить пермишены в `AndroidManifest.xml`:

- `WRITE_EXTERNAL_STORAGE` (с 19 API не всегда нужно!)
- `READ_EXTERNAL_STORAGE`

Проверка состояния внешнего хранилища:

```
Environment.getExternalStorageState()
```

Возможные состояния хранилища:

- `Environment.MEDIA_MOUNTED`
- `Environment.MEDIA_MOUNTED_READ_ONLY`



# External Storage

```
fun isExternalStorageWritable: Boolean () {  
    val state = Environment.getExternalStorageState()  
    return Environment.MEDIA_MOUNTED == state  
}  
  
fun isExternalStorageReadable: Boolean () {  
    val state = Environment.getExternalStorageState()  
    return Environment.MEDIA_MOUNTED.equals(state) ||  
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state));  
}
```

# Общедоступные директории


Android сканирует некоторые директории с целью предоставить пользователю удобный доступ к ним. Получить путь до них можно с помощью:

```
File getExternalStoragePublicDirectory (String type)
```

Типы директорий:

- DIRECTORY\_MUSIC
- DIRECTORY\_PICTURES
- DIRECTORY\_DOWNLOADS
- DIRECTORY\_RINGTONES
- ...

# Общедоступные директории



```
fun getAlbumStorageDir(albumName: String): File
{
    // Get the directory for the user's public pictures directory.
    val file = File(
        Environment.getExternalStoragePublicDirectory(
            Environment.DIRECTORY_PICTURES
        ), albumName
    );
    if (!file.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}
```

# “Приватные” директории

Как Internal Storage для вашего приложения, но на внешнем носителе:

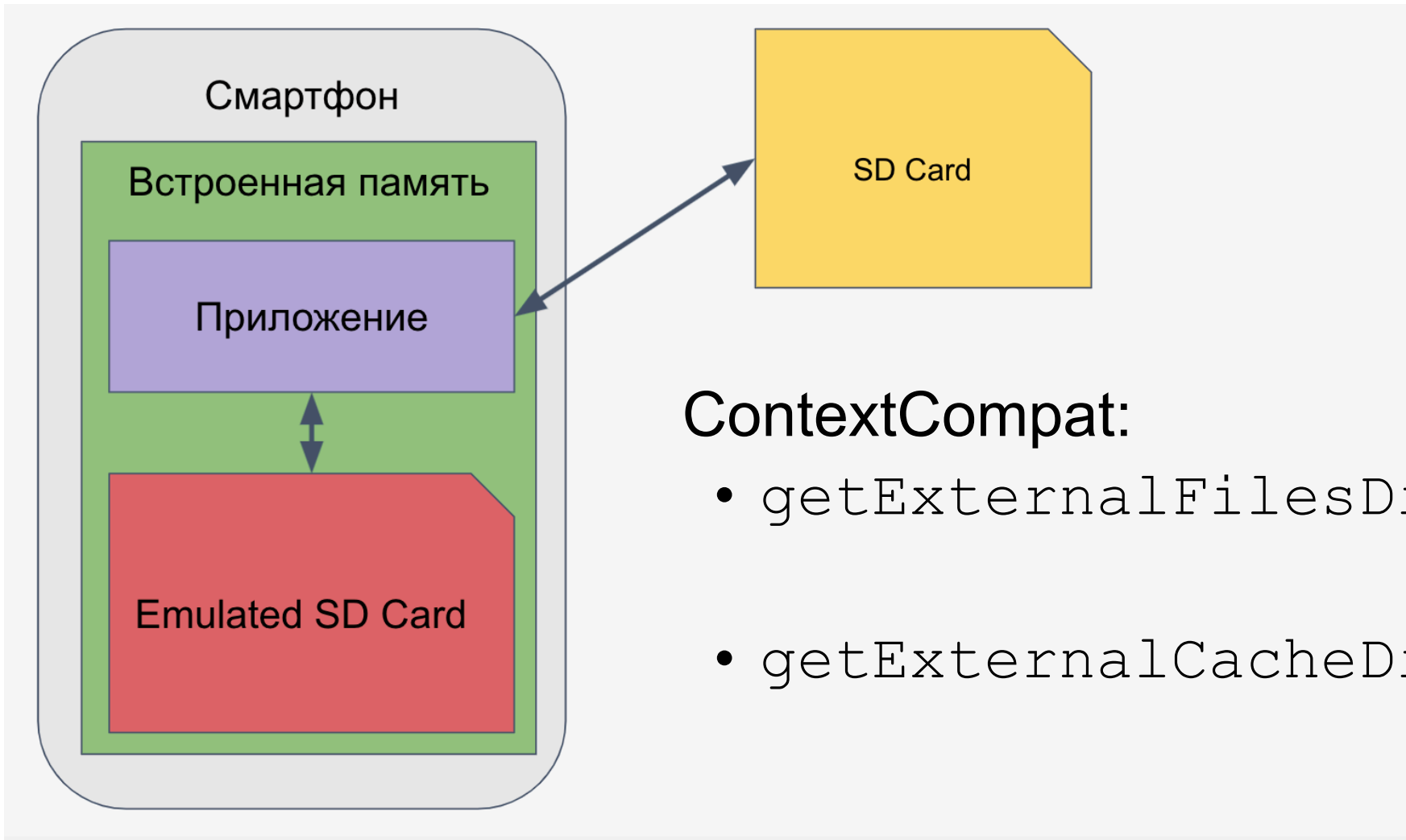
- `File getExternalFilesDir(String type)` – путь до приватной директории приложения
- `File getExternalCacheDir()` - путь до директории для хранения кэшей

`<uses-permission`

`android:name="android.permission.WRITE_EXTERNAL_STORAGE"`

`android:maxSdkVersion="18"`

# Внешнее хранилище, но не внешнее



**ContextCompat:**

- `getExternalFilesDirs`
- `getExternalCacheDirs`

# Куда устанавливается приложение?

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

Возможные варианты для `installLocation`:

- `internalOnly`
- `auto`
- `preferExternal`

# AccountManager

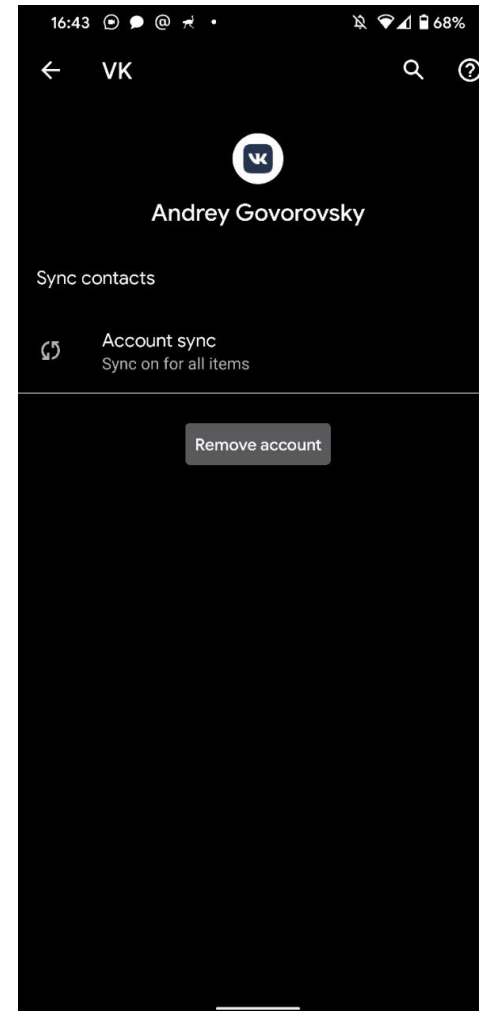
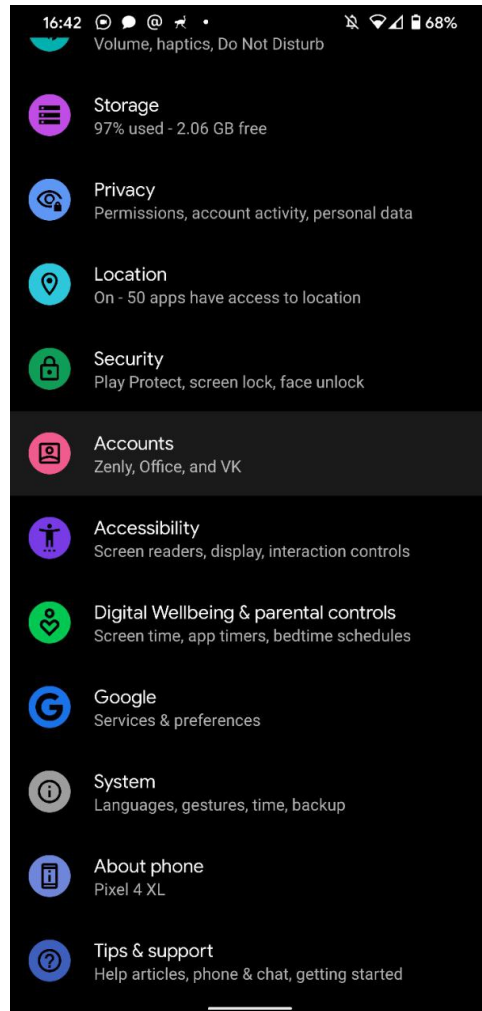


# AccountManager

- Тесная интеграция с системой
- Данные не теряются
- Сложность и оверхед при реализации
- Есть баги при получении данных



# AccountManager



# SQL



# SQLiteDatabase

ОСНОВНЫЕ МЕТОДЫ -

1. insert
2. delete
3. update
4. query
5. execSQL
6. rawQuery
7. beginTransaction/endTimeTransaction

# SQLiteOpenHelper

```
class DbHelper(  
    context: Context?,  
    ) : SQLiteOpenHelper(context, DB_NAME, null, DB_VERSION) {  
    companion object {  
        const val DB_VERSION = 1  
        const val DB_NAME = "MY_DB"  
    }  
  
    override fun onCreate(db: SQLiteDatabase?) {  
        TODO("Not yet implemented")  
    }  
  
    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {  
        TODO("Not yet implemented")  
    }  
  
    override fun onDowngrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {  
        TODO("Not yet implemented")  
    }  
}  
  
val db = DbHelper(this).readableDatabase.query(...)
```

# SQLiteDatabase. Insert



```
val db = DBHelper(this).readableDatabase

val contentValues = ContentValues().apply {
    put("title", title)
    put("subtitle", subtitle)
}


val newId = db.insert(TABLE_NAME, null, values)
```

# SQLiteDatabase. Delete



```
val selection: String = COLUMN_NAME_TITLE.toString() + " LIKE ?"  
val selectionArgs = arrayOf("MyTitle")  
  
val affectedRows = db.delete(TABLE_NAME, selection, selectionArgs)
```

# SQLiteDatabase. Update



```
val values = ContentValues()  
    values.put(COLUMN_NAME_TITLE, title)  
  
val selection: String = COLUMN_NAME_TITLE.toString() + " LIKE ?"  
val selectionArgs = arrayOf("MyTitle")  
  
val count = db.update(  
    TABLE_NAME,  
    values,  
    selection,  
    selectionArgs  
)
```

# SQLiteDatabase. Query

```
val projection = arrayOf<String>(
    COLUMN_NAME_TITLE,
    COLUMN_NAME_SUBTITLE
)

val selection: String = COLUMN_NAME_TITLE.toString() + " = ?"
val selectionArgs = arrayOf("My Title")


val sortOrder: String = COLUMN_NAME_SUBTITLE.toString() + " DESC"

val c: Cursor = db.query(
    TABLE_NAME,
    projection,
    selection,
    selectionArgs,
    null /*groupBy*/,
    null /*having*/,
    sortOrder
)
```



# SQLiteDatabase. execSQL


Используем, если НЕ нужно вернуть данные. Например CREATE, DROP, PRAGMA и тд



```
val sql = "INSERT INTO " + TABLE_NAME.toString() +  
    " (" + COLUMN_NAME_TITLE.toString() + "," + COLUMN_NAME_SUBTITLE.toString() + ")" +  
    " VALUES (?, ?)"  
val params = arrayOf<Any>(title, subtitle)  
  
db.execSQL(sql, params)
```

# SQLiteDatabase.rawQuery


rawQuery может вернуть данные. Используется для сложных запросов



```
val query = "SELECT " +  
    COLUMN_NAME_TITLE.toString() + "," + COLUMN_NAME_SUBTITLE.toString() +  
    " FROM " + TABLE_NAME.toString() + " WHERE " + COLUMN_NAME_TITLE.toString() + "=?"  
  
val params = arrayOf(title)  
  
val cursor = db.rawQuery(query, params)
```

# SQLiteDatabase. Transactions

Must-have фича. Позволяет кардинально ускорять батч-запросы и производить откаты в случае неуспешных операций



```
db.beginTransaction()  
try {  
    //select, insert, update, delete...  
    db.setTransactionSuccessful()  
} finally {  
    db.endTransaction()  
}
```

**А можно проще?**

# ORM



# Room

**Room** — это высокоуровневый интерфейс для низкоуровневых привязок SQLite, встроенных в Android, о которых вы можете узнать больше в документации. Он выполняет большую часть своей работы во время компиляции, создавая API-интерфейс поверх встроенного SQLite API, поэтому вам не нужно работать с Cursor или ContentResolver.



```
@Entity
class Person {
    @PrimaryKey val name: String
    val age: Int
    favoriteColor: String
}
```

# Room. DAO

```
@Dao
interface PersonDao {

    // Добавление Person в бд
    @Insert
    fun insertAll(vararg people: Person)

    // Удаление Person из бд
    @Delete
    fun delete(person: Person)

    // Получение всех Person из бд
    @Query("SELECT * FROM person")
    fun getAllPeople(): List<Person>

    // Получение всех Person из бд с условием
    @Query("SELECT * FROM person WHERE favoriteColor LIKE :color")
    fun getAllPeopleWithFavoriteColor(String color): List<Person>

}
```

# Room. Create DB



```
@Database(entities =
{Person::class.java /*, AnotherEntityType::class.java */}, version = 1)
abstract class AppDatabase : RoomDatabase {
    abstract fun getPersonDao(): PersonDao
}

val db = Room.databaseBuilder(getApplicationContext(),
    AppDatabase::class.java, "populus-database").build()

val everyone = db.getPersonDao().getAllPeople()
```



# References

<https://developer.android.com/guide/topics/data/data-storage.html>

<https://developer.android.com/guide/topics/data/install-location.html>

<https://habr.com/ru/post/336196/>

<https://developer.android.com/about/versions/11/privacy/storage>

Спасибо за  
внимание

