NTI/PAA - PROGRAMOVÁNÍ MOBILNÍCH APLIKACÍ

5. Persistentní ukládání dat, preference, soubory, SQLite

Ing. Igor Kopetschke – TUL, NTI

http://www.nti.tul.cz

Android – Persistentní ukládání dat

- Android poskytuje několik způsobů, jak ukládat data dle účalu a jejich typu
  - **SharedPreferences**
    - Ukládání **primitivních** datových typů v podobě **key / value**
    - Původně určeno pro ukládání nastavení, ale obecně možno použít pro jakýkoli účel
  - **Internal Storage**
    - Ukládání souborů do **interní** paměti zařízení
    - Tyto soubory jsou chápány jako "**privátní**" pro danou aplikaci
    - Při **deinstalaci** aplikace jsou tyto soubory **smazány**
  - **External Storage**
    - Ukládání souborů na SD kartu nebo interní veřejnou paměť
    - Tyto soubory jsou "**veřejně**" přístupné aplikacemi i po připojení přes USB
  - **SQLite databáze**
    - Ukládání strukturovaných dat do privátní databáze. Pouze primitivní datové typy, String, Bundle, Blob, jiné DT nutno serializovat.
  - **Síťová úložiště**
    - Cloudy, webové služby aj.

Android – SharedPreferences

- Třída pro ukládání a načítání párů klíč / hodnota
- Určeno pouze pro datové typy:
  `boolean, float, int, long, String`
- Data jsou přístupná pro všechny komponenty aplikace
- Uchovávány v souborech typu XML v umístění:
  `/data/data/<jméno aplikace>/shared_prefs/`

- Příklad souboru s preferencemi

```
<?xml version='1.0' encoding='utf-8' standalone='yes'?>
      <map>
            <long name="longkey" value="1"/>
            <boolean name="booleankey" value="false"/>
            <string name="stringkey">text</string>
      </map>
```

Android – SharedPreferences

Instanci SharedPreferences lze získat následovně:

- Pomocí **`PreferenceManager`**

  `PreferenceManager.getDefaultSharedPreference( Context )`

  - Vrátí výchozí SharedPreference pro všechny komponenty aplikace

- Pomocí **`Context`**

  `Context.getSharedPreference( String name, int mode )`

  - Vrátí SharedPreference identifikované názvem souboru (**`name`**)
  - **`mode`** určuje přístupnost
    - **MODE_PRIVATE** – pouze v rámci **aplikace** nebo z aplikací se stejným **user ID**
    - **MODE_WORLD_READABLE**, **MODE_WORLD_WRITABLE** – možnost sdílet preference **napříč** dalšími aplikacemi – zavrženo od API 17, nahrazeno použitím Service, BroadcastReceiver a ContentProvider

- V rámci **`Activity`**

  `getPreference( int mode )`

  - Vrátí SharedPreference pouze pro konkrétní **aktivitu** v daném **`mode`**

SharedPreferences – ukládání dat

- K editaci a uložení se používá **SharedPreferences.Editor**
- Instance editoru se získá metodou **edit()**
- Uložení změn se provádí voláním metody **commit()**

```
public static final String PREFS_NAME = "MyPrefsFile";

SharedPreferences settings =
    getSharedPreferences(PREFS_NAME, MODE_PRIVATE);

SharedPreferences.Editor editor = settings.edit();
editor.putBoolean("silentMode", true);
editor.putLong("result", 35698);
editor.putString("nadpis", "Nejaky nadpis");
editor.commit();
```

SharedPreferences – načítání dat

- Pro každý podporovaný datový typ existuje příslušný getter
- **getInt, getString, getFloat, getBoolean, getLong**
- Každý getter má 2 parametry – klíč a defaultní hodnotu
- Defaultní hodnota je použita v případě, že daný klíč neexistuje

```
public static final String PREFS_NAME = "MyPrefsFile";

SharedPreferences pref =
    getSharedPreferences(PREFS_NAME, MODE_PRIVATE);

SharedPreferences.Editor editor = settings.edit();
boolean silent = pref.getBoolean("silentMode", false);
long res = pref.getLong("result", 0);
Strign txt = pref.getString("nadpis", "");
```

Android – Internal Storage

- Ukládání soukromých souborů pro aplikaci
- Využívá interní paměť zařízení, vhodná pro malé soubory
- Uchovávány v umístění:
  `/data/data/<jméno aplikace>/files/`

- Získání instancí pro FileInputStream, FileOutputStream
  - **`openFileInput(String filename)`** – stream pro čtení
  - **`openFileOutput(String filename, int mode)`** – stream pro zápis
- Čtení a zápis
  - **`read(), write()`** – analogicky jako v Javě
- Uzavření streamu
  - **`flush(), close()`**

Android – Internal Storage + cache

- Použití soukromé souborové cache
- Po ukončení aplikace může být smazána, pokud dochází místo v interní paměti
- Uchovávány v umístění:
  `/data/data/<jméno aplikace>/cache/`
- Získání reference na cache
  - ```
    File cachedir = getCacheDir()
    File f = new File( cachedir, "filename")
    ```
- Získání instancí pro FileInputStream, FileOutputStream
  - ```
    new FileInputStream(f)
    new FileOutputStream(f)
    ```
- Čtení, zápis, uzavření
  - **`read(), write(), flush(), close()`**

Android – External Storage

- Ukládání na veřejné úložiště v zařízení nebo na SD kartu
- Vhodná pro velké a ne-soukromé soubory
- V manifestu nutné oprávnění
  ```
  android.permission.WRITE_EXTERNAL_STORAGE
  android.permission.READ_EXTERNAL_STORAGE
  ```
- Uchovávány v umístění:
  `/sdcard/Android/data/<jméno aplikace>/files/`
- Získání reference na adresář
  - ```
    File extdir = getExternalFilesDir( path )
    File f = new File( extdir, "filename")
    ```
- Získání instancí pro FileInputStream, FileOutputStream
  - ```
    new FileInputStream(f)
    new FileOutputStream(f)
    ```
- Čtení, zápis, uzavření
  - ```
    read(), write(), flush(), close()
    ```
    ```

Android – External Storage + cache

- Použití externí souborové cache
- Po ukončení aplikace může být smazána, pokud dochází místo v interní paměti
- Uchovávány v umístění:
  `/sdcard/Android/data/<jméno aplikace>/cache/`
- Získání reference na cache
  - ```
    File cachedir = getExternalCacheDir()
    File f = new File( cachedir, "filename")
    ```
- Získání instancí pro FileInputStream, FileOutputStream
  - ```
    new FileInputStream(f)
    new FileOutputStream(f)
    ```
- Čtení, zápis, uzavření
  - `read(), write(), flush(), close()`

Android – databáze SQLite

- Slouží k ukládání strukturovaných dat v soukromé databázi
- Nejrozšířenější DB na embedded zařízeních a třeba i v browserech a mnoha aplikacích
- Její instalace je malá a přitom poměrně rychlá
- Kompletní databáze je v 1 souboru (*.db)
- Transakce jsou atomické, trvanlivé a konzistentní i po pádu systému
- Podporuje většinu standardů SQL92
- V Androidu se k SQLite DB nepřistupuje přímo, ale pomocí specializovaných tříd
- Datové typy v třídě Cursor
  - FIELD_TYPE_INTEGER, FIELD_TYPE_NULL, FIELD_TYPE_STRING, FIELD_TYPE_FLOAT, FIELD_TYPE_BLOB

SQLite – vytvoření, otevření, upgrade

- Třída **SQLiteOpenHelper** jako rodič pro konkrétní DB
- Potomek obsahuje
  - jméno DB souboru
  - verzi DB
  - definici struktury DB
  - překrytí metody **onCreate()** – volána, pokud DB ještě neexistuje
  - překrytí metody **onUpgrade()** – volána, když se zvýší verze DB
- Díky rodiči poskytuje metody pro získání instance třídy **SQLiteDatabase**
  - getReadableDatabase()
  - getWritableDatabase()

SQLiteOpenHelper - příklad

```
1.  public class DictionaryOpenHelper extends SQLiteOpenHelper {
2.      private static final int DATABASE_VERSION = 2;
3.      private static final String DATABASE_NAME = "databaze";
4.
5.      DictionaryOpenHelper(Context context) {
6.          super(context, DATABASE_NAME, null, DATABASE_VERSION);
7.      }
8.
9.      @Override
10.     public void onCreate(SQLiteDatabase db) {
11.         String create =  "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
12.                 KEY_WORD + " TEXT, " +
13.                 KEY_DEFINITION + " TEXT);";
14.         db.execSQL(create);
15.     }
16.
17.     @Override
18.     public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
19.         //Upgrade databaze - většinou vytvoření dočasných tabulek a překopírování dat
20.         //nebo smazání DB a vytvoření nové
21.     }
22. }
```

Zavolá se, pokud databáze není vytvořená

Zavolá se, pokud se zvýší *DATABASE_VERSION*

SQLite – Přístup k DB

- Pro přístup vytvořit instanci potomka **SQLiteOpenHelper**
- Databázi získat pomocí **dbHelper.getWritableDatabase()**

```
1. SQLiteOpenHelper dbHelper = new DictionaryOpenHelper(mCtx);
2. SQLiteDatabase mDb = dbHelper.getWritableDatabase();
```

- Na konci je třeba DB uzavřít

```
1. mDb.close();
```

SQLite – Vkládání dat

- Po získání odkazu na databázi, provedeme vložení dat příkazem `insert(table,nullColumnHack,values);`
- Metoda `insert` vrací ID právě vloženého řádku nebo -1, pokud nastala chyba

- `ContentValues`
  - Dvojice klíč/hodnota
  - Používá se při vkládání nebo updatování tabulek
  - Klíč odpovídá jménu sloupce tabulky

```
1. ContentValues initialValues = new ContentValues();
2. initialValues.put(KEY_WORD, "word");
3. initialValues.put(KEY_DEFINITION, "definice slova word -...");
4. mDb.insert(DICTIONARY_TABLE_NAME, null, initialValues);
```

SQLite – Vyhledávání

- K prohledávání databáze slouží metoda `query`
- Vrací `Cursor`
  - Prostředek pro procházení výsledky dotazu, čtení řádků a sloupců
  - Po provedení potřebných úkonů je třeba `Cursor` zavřít

```
1. Cursor c = mDb.query(DICTIONARY_TABLE_NAME,
2.                 null, //sloupce
3.                 null, //klauzule WHERE
4.                 null, //parametry,pokud se vyskytují ? ve WHERE
5.                 null, //groupBy
6.                 null, //having
7.                 null);//orderBy
```

query(TABLE_DOCUMENTS, new String[] {KEY_DOCUMENTS_ID, KEY_DOCUMENTS_CREATEDBY_ID }, KEY_ID + "=?“, new String[] {id});

SQLite – Vyhledávání

- Čtení obsahu DB probíhá až v momentě, kdy zavoláme příslušnou metodu Cursoru (`getString`, `getInt`...)
- Cursor pouze ukazuje na výsledek vyhledávání (šetření paměti)

```
1. Cursor c = mDb.query(DICTIONARY_TABLE_NAME, null, null, null, null, null,
   null);
2. if (c.moveToFirst()) {//posune na začátek, vrátí false, pokud je cursor
   prázdný
3.    int colWord = c.getColumnIndex(KEY_WORD); //číslo sloupce "word"
4.    int colDef = c.getColumnIndex(KEY_DEFINITION); //číslo sloupce "definition"
5.    do {
6.          String word = c.getString(colWord);
7.          String def = c.getString(colDef);
8.          ...
9.    } while (c.moveToNext()); //posune cursor na další řádek
10.   c.close();
11. }
```

SQLite – další metody Cursoru

- `moveToPrevious()`
- `getCount()`
- `getColumnIndexOrThrow(String columnName)`
- `getColumnName(int columnIndex)`
- `getColumnNames()`
- `getColumnCount()`
- `moveToPosition(int position)`
- `getPosition()`
- `isClosed()`
- ...

SQLite – update dat

- Aktualizace dat se provádí pomocí metody `update(table, values, whereClause, whereArgs);`

- Metoda vrací počet aktualizovaných řádků

```
1. ContentValues updatedValues = new ContentValues();
2. updatedValues.put(KEY_DEFINITION, "nová definice");
3.
4. mDb.update(DICTIONARY_TABLE_NAME,
5.         updatedValues,
6.         KEY_WORD+"='word'", //klauzule WHERE
7.         null); //parametry, pokud se vyskytují ? ve WHERE
```

SQLite – mazání dat

- Mazání dat se provádí pomocí metody `delete(table, whereClause, whereArgs);`
- Metoda vrací počet smazaných řádků, pokud `whereClause` **není** `null`, jinak `0`
- Pro smazání všech řádků a získání počtu předáme do `whereClause` hodnotu `"1"`

```
1. mDb.delete(DICTIONARY_TABLE_NAME,
2.         KEY_WORD+"='word'", //klauzule WHERE
3.         null); //parametry, pokud se vyskytují ? ve WHERE
```

SQLite – rawQuery, SQLiteQueryBuilder

- **rawQuery(String sql, String[] args)**
  - alternativní způsob zasílání SELECT dotazů do DB
  - Vrací instanci Cursor
  - **sql** – SQL dotaz s možností parametrizace argumentů (?)
  - **args** - hodnoty parametrizovaných argumentů

```
db.rawQuery("select * from tbl where _id = ?", new String[] {id})
```

- **SQLiteQueryBuilder**
  - další z pomocných tříd pro práci s SQLite
  - Slouží pro usnadnění sestavování SQL dotazů
  - pro podrobnosti viz dokumentace API

SQLite – zámek DB

- Defaultně se SQLite stará sám o zamykání DB v kritických sekcích a zabraňuje tak poškození dat při zápisu z více vláken

- Pozor na (ne)vrácené výjimky – použít např. frontu nebo jinou synchronizaci přístupu

- Časově náročné
  - Pokud se k DB přistupuje pouze z jednoho vlákna, je dobré zamykání vypnout

```
1. mDb = dbHelper.getWritableDatabase();
2. mDb.setLockingEnabled(false);
```

SQLite – DB a životní cyklus aktivity

- DB je nutné po jejím otevření zavřít
  - Open/close v rámci jednoho požadavku
    - Pomalejší
  - V rámci životního cyklu activity
    - Pozor na správné pořadí open/close
- Nezavírat DB, pokud nejsou uzavřeny i cursory
- Obdobně je nutné zavřít i cursory
- Z důvodů paměťové náročnosti se nevyplácí používat různé formy ORM nebo vytvářet pomocné třídy pro obalení dat z databáze

Použité a doporučené zdroje

- http://developer.android.com/
- http://www.zdrojak.cz/serialy/vyvijime-pro-android/
- http://www.itnetwork.cz/java/android
- https://users.fit.cvut.cz/cermaond/dokuwiki
- Google…

.. A to je pro dnešek vše

DĚKUJI ZA POZORNOST