

ANDROID V.

6. týden, KIV/MKZ 2017

L. Pešička

OBSAH

- ◉ Služby
- ◉ Poskytovatelé obsahu

LOADERS

◉ Život před loadery

- **public void startManagingCursor(Cursor)**
 - Životní cyklus Cursoru dle životního cyklu Activity
 - Cursor deaktivován, když aktivita zastavena
 - Cursor closed, když aktivita zrušena (destroy)
 - Aktivita zastavena a později restartována, cursor requery
- **public Cursor managedQuery()**
 - kromě dotazu i management kurzoru
 - viz předchozí funkce
- Dotazy na UI vlákně...
- Konfigurační změny aktivity...

LOADERS

Od Android 3.0

- ◉ **Loader** - získání dat
 - CursorLoader, AsyncTaskLoader
- ◉ **LoaderManager** - spravuje Loadery
 - callbacky onCreateLoader, onLoadFinish, ...
- ◉ Cursorové operace jsou prováděné asynchronně (samostatné vlákno)
- ◉ Zachování dat při změně konfigurace aktivity (nejsou zbytečné dotazy navíc)
- ◉ Update datových zdrojů - aktualizace cursoru

CURSOR LOADER

- ⦿ asynchronní nahrávání dat u aktivit a fragmentů
- ⦿ od Android 3.0 (API 11)
- ⦿ CursorLoader
- ⦿ LoaderCallbacks<Cursor> rozhraní

CURSOR LOADER

- ◉ inicializace dotazu
 - `LoaderManager.initLoader()`
- ◉ spuštění dotazu
 - `onCreateLoader()`
- ◉ dokončení dotazu
 - `onLoadFinished()`
- ◉ dojde ke změně dat
 - `onLoaderReset()`

použití např.

<http://developer.android.com/training/load-data-background/handle-results.html>

LOADERS - LITERATURA

- ⦿ <http://www.androiddesignpatterns.com/2012/07/loaders-and-loadermanager-background.html>
- ⦿ <http://www.androiddesignpatterns.com/2012/07/understanding-loadermanager.html>
(doporučuji podívat se)

CO JE TO LOOPER?

- ◉ Looper transformuje běžné vlákno - **běží dokud `run()` neskončí** do něčeho běžícího kontinuálně dokud aplikace běží
- ◉ Looper poskytuje frontu pro joby, které je třeba udělat
- ◉ Vlákno defaultně nemá message loop, která je s ním spojená. Looper je třída, která to zařídí.

ANDROID LOOPER

- ◉ Třída obsahuje frontu zpráv (MessageQueue)
- ◉ Je asociovaná s vláknem, z kterého byla vytvořena
- ◉ Pevně spjaté (vlákno, fronta zpráv)
- ◉ prepare()
 - Zkontroluje, zda již looper není, vytvoří
- ◉ loop()
 - Kontrola zpráv ve frontě
- ◉ UI vlákno má **implicitní Looper**, který nám umožní obsloužit zprávy na UI vlákně

VYTVŮŘENÍ LOOPERU

- 1) Extend Thread
- 2) Call `Looper.prepare()` to initialize Thread as a looper
- 3) Create one or more Handlers to process the incoming messages
- 4) Call `Looper.loop()` to process messages until the loop is told to quit.

<http://stackoverflow.com/questions/7597742/what-is-the-purpose-of-looper-and-how-to-use-it>

PŘÍKLAD LOOPERU

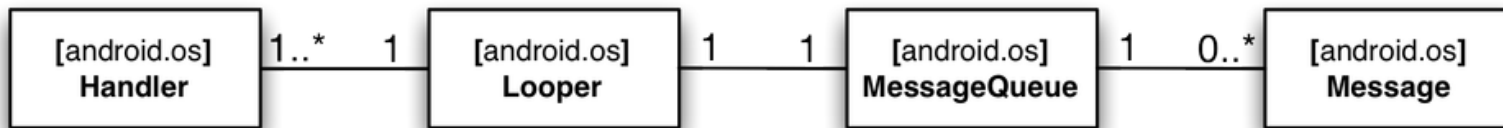
```
class SampleLooper {  
    @Override  
    public void run() {  
        try {  
            // preparing a looper on current thread  
            // the current thread is being detected implicitly  
            Looper.prepare();  
  
            // now, the handler will automatically bind to the  
            // Looper that is attached to the current thread  
            // You don't need to specify the Looper explicitly  
            handler = new Handler();  
  
            // After the following line the thread will start  
            // running the message loop and will not normally  
            // exit the loop unless a problem happens or you  
            // quit() the looper (see below)  
            Looper.loop();  
        } catch (Throwable t) {  
            Log.e(TAG, "halted due to an error", t);  
        }  
    }  
}
```

PŘÍKLAD LOOPERU

```
handler.post(new Runnable()  
{  
    public void run() {  
        //This will be executed on thread using Looper.  
    }  
});
```

A **Handler** allows you to **send** and **process** **Message** and **Runnable** objects associated with a thread's **MessageQueue**. Each Handler instance is associated with a single thread and that thread's message queue. When you create a new Handler, it is bound to the thread / message queue of the thread that is creating it – from that point on, it will deliver messages and runnables to that message queue and execute them as they come out of the message queue.

PŘÍKLAD LOOPERU

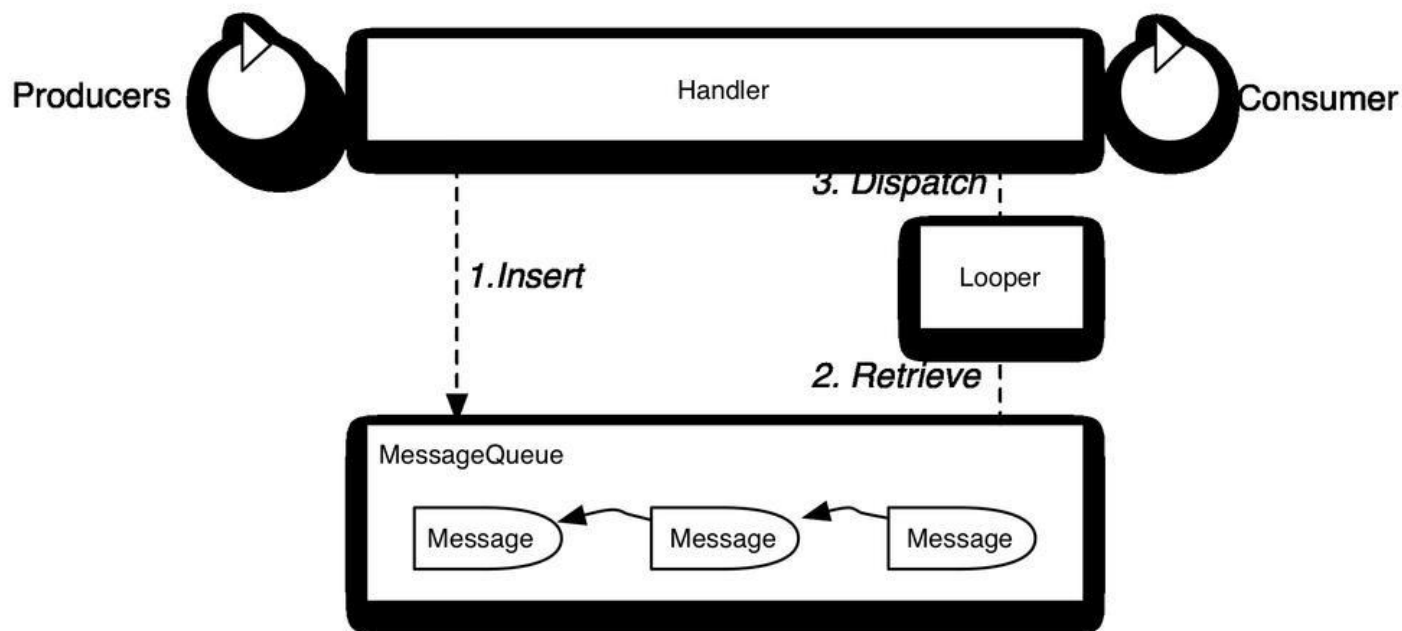


Looper	Message dispatcher spojený s jedním vláknem
Handler	Zpracování zpráv. Vkládání zpráv do fronty
MessageQueue	Seznam zpráv, které budou zpracovány na vlákně
Message	Zpráva, která bude vykonána na vlákně

LOOPER, HANDLER

Podrobný popis:

<https://www.safaribooksonline.com/library/view/efficient-android-threading/9781449364120/ch04.html>



VÍCE HANDLERŮ

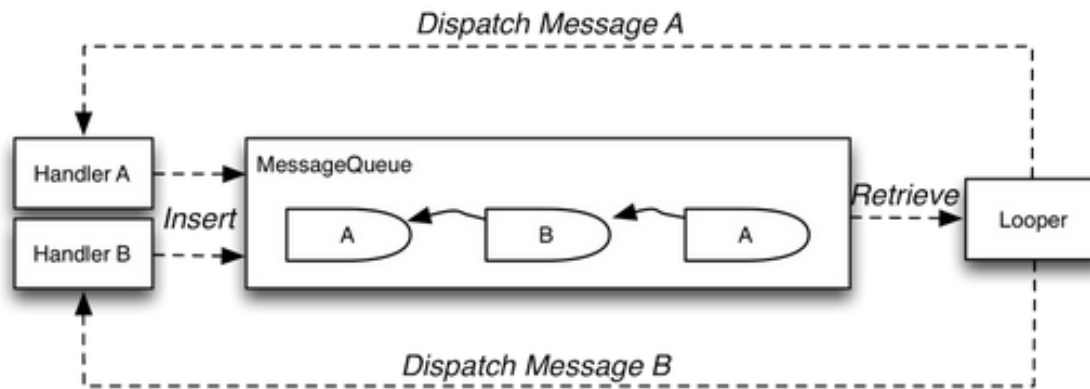


Figure 4-9. Multiple handlers using one Looper. The handler inserting a message is the same handler processing the message.

ANDROID HANDLER

- ◉ Přidat zprávu do fronty zpráv
- ◉ Další operace (odstranit, updatovat)
- ◉ Handler je připojený k looperu a tedy k vlákn
- ◉ Více handlerů může být k jednomu vlákn

POZNÁMKY

Odkaz:

<http://stackoverflow.com/questions/7597742/what-is-the-purpose-of-looper-and-how-to-use-it>

Looper umožní postupně vykonat tasky na jednom vlákně

Handler definuje tasky, které je třeba vykonat

PROČ SLUŽBY?

- ◉ práce v pozadí by měla být dělána ve vlákně
- ◉ když ale jen vytvoříme nové vlákno, je zde riziko, že pokud aktivita nebude viditelná, OS může zastavit vlákno bez varování
- ◉ např. MP3 přehrávač - nechceme zastavit přehrávání, pokud uživatel stiskne HOME
- ◉ řešením je služba

SLUŽBA - FUNKCIONALITA DALŠÍM APLIKACÍM

- ◉ aplikační proces se pustí, když jiná aplikace chce využít naší služby

SLUŽBY

- ◉ dlouhotrvající operace na pozadí
- ◉ nemá uživatelské rozhraní
 - lze notifikovat na status bar, že je něco hotové
- ◉ příklady využití:
 - přehrávání hudby
 - síťové transakce
 - souborové I/O
 - interakce s content providerem

DVĚ VARIANTY SLUŽBY

◉ spuštěná

- aktivita ji spustí přes `startService()`
- služba může běžet, i když už aktivita skončila
- např. stáhne soubor a pak sama sebe ukončí

◉ bound

- aplikace zavolá `bindService()`
- komunikace klient - server (žádost-odpověď)
- služba běží tak dlouho, dokud je k ní aplikace „připojena“
- více komponent se může připojit ke službě najednou

daná služba může fungovat oběma způsoby

POZNÁMKY

- ◉ spuštění služby:
`startService(new Intent(this, MyService.class));`
- ◉ službu lze deklarovat jako privátní v manifestu (ostatní se k ní nedostanou)
- ◉ služba běží v hlavním vlákně hostujícího procesu (!)
 - nevytváří vlastní vlákno
 - neběží v odděleném procesu (není-li dáno jinak)
 - uvnitř služby vytvořit vlákno pro náročné činnosti

METODY SLUŽBY

- ◉ podtřídu třídy Service, přepsat callbacky
- ◉ **onStartCommand()**
 - chce-li někdo spustit službu přes `startService()`
 - může běžet nedefinovanou dobu
 - vlastní ukončení: `stopSelf()`, `stopService()`
 - chceme-li jen bound službu, nemusíme implementovat
- ◉ **onBind()**
 - když se chce jiná komponenta připojit k naší službě (zavolá `bindService()`)
 - poskytnout rozhraní pro komunikaci klienta se službou - vrátit `Ibinder`
 - vždy je potřeba implementovat, i když třeba jen vrátit `null`

METODY SLUŽBY

◉ onCreate()

- když je služba prvně vytvořena
- před zavoláním onStartCommand(), onBind()
- pokud již služba běží, nevolá se

◉ onDestroy()

- při zrušení služby, úklid

DOBA BĚHU SLUŽBY

- ⊙ při vytvoření `startService()`
 - dokud se nezastaví sama
 - nezávislý životní cyklus na původní komponentě
- ⊙ při spuštění přes `bindService()`
 - při „odbindování“ ze všech klientů
- ⊙ Android systém ukončí službu jen při nedostatku paměti

DEKLARACE SLUŽBY V MANIFESTU

```
<manifest ... >
```

```
...
```

```
<application ... >
```

```
  <service android:name=".ExampleService" />
```

```
...
```

```
</application>
```

```
</manifest>
```

dále:

- lze nastavit práva pro spuštění služby
- proces, ve kterém služba běží
- lze definovat intenty, které dovolí dalším komponentám službu zavolat
- `android:exported= " false "` .. privátní služba

SLUŽBA DOSTUPNÁ JINÝM APLIKACÍM

```
<service android:name="DownloadService"  
        android:exported="true">  
</service>
```

STARTOVANÁ SLUŽBA

dva způsoby vytvoření:

- ◉ Service

- je třeba vytvořit nové vlákno, v kterém služba pracuje

- ◉ IntentService

- používá worker vlákno pro obsluhu požadavků
- nemusíme tedy vytvářet vlákno sami
- implementovat `onHandleIntent()`

STARTOVANÁ SLUŽBA

- INTENT SERVICE

- ◉ automaticky vytvoří worker vlákno pro obsluhu požadavků
- ◉ vytvoří frontu požadavků postupně po jednom předávaných funkci `onHandleIntent()`
- ◉ zastaví službu po obsloužení všech požadavků
- ◉ poskytuje implementaci `onBind()` vrací null
- ◉ poskytuje implementaci `onStartCommand()`
 - předá záměr frontě požadavků => `on HandleIntent()`

zbývá na nás především implementace `onHandleIntent()`

PŘÍKLAD SLUŽBY

```
public class HelloIntentService extends IntentService {  
    /* Konstruktor volá nadřazený konstruktor IntentService(String)  
    * se jménem pracovního vlákna */  
    public HelloIntentService() {  
        super("HelloIntentService");  
    }  
  
    protected void onHandleIntent(Intent intent) {  
        // zde bychom vykonávali nějakou činnost  
        long endTime = System.currentTimeMillis() + 5*1000;  
        while (System.currentTimeMillis() < endTime) {  
            synchronized (this) {  
                try {  
                    wait(endTime - System.currentTimeMillis());  
                } catch (Exception e) { }  
            } } }  
    }  
}
```

PŘÍKLAD SLUŽBY

- lze přeprogramovat i další metody, ale vždy zavolat nadřazenou implementaci:

```
public int onStartCommand(Intent intent, int flags, int startId) {  
    Toast.makeText(this, "service starting", Toast.LENGTH_SHORT)  
        .show();  
    return super.onStartCommand(intent, flags, startId);  
}
```

ROZŠÍŘENÍ TŘÍDY SERVICE

- ◉ Například pokud potřebujeme multithreading (nestačí nám jedna fronta na jedno pracovní vlákno) jako u IntentService
- ◉ Složitější kód než u IntentService

PŘÍKLAD

```
public class HelloService extends Service {  
    private Looper mServiceLooper;  
    private ServiceHandler mServiceHandler;
```

```
// Handler přijímá zprávy z vlákna
```

```
private final class ServiceHandler extends Handler {  
    public ServiceHandler(Looper looper) { super(looper); }
```

```
@Override
```

```
public void handleMessage(Message msg) {
```

```
    // zde by se dělal něco užitečného
```

```
    // stejné jako předchozí příklad
```

```
    // zastavení služby
```

```
    stopSelf(msg.arg1);
```

```
}
```

```
}
```

PŘÍKLAD - POKRAČOVÁNÍ

```
public void onCreate() {  
    // spustíme nové vlákno - služba jinak běží v hlavním vlákně  
    // dáme nižší prioritu, CPU-intenzivní nebude rušit UI.  
    HandlerThread thread = new  
    HandlerThread("ServiceStartArguments",  
        Process.THREAD_PRIORITY_BACKGROUND);  
    thread.start();  
  
    mServiceLooper = thread.getLooper();  
    mServiceHandler = new ServiceHandler(mServiceLooper);  
}
```

PŘÍKLAD - POKRAČOVÁNÍ

```
public int onStartCommand(Intent intent, int flags, int startId) {  
    Toast.makeText(this, "sluzba bezi", Toast.LENGTH_SHORT).show();  
  
    // Pro každý požadavek pošle zprávu spustit job se startID  
  
    Message msg = mServiceHandler.obtainMessage();  
    msg.arg1 = startId;  
    mServiceHandler.sendMessage(msg);  
  
    return START_STICKY;  
}
```

PŘÍKLAD - DOKONČENÍ

```
public IBinder onBind(Intent intent) {  
    // neposkytujeme vazbu, vrátíme null  
    return null;  
}  
  
public void onDestroy() {  
    Toast.makeText(this, "service done", Toast.LENGTH_SHORT)  
        .show();  
}  
}
```

JAK POKRAČOVAT, ZABIJE-LI SYSTÉM SLUŽBU

návratová hodnota onStartCommand():

Návratová hodnota	popis
START_NOT_STICKY	Znovu nevytvářet službu
START_STICKY	Znovu vytvořit službu a zavolat onStartCommand(), Např. pro přehrávač hudby
START_REDELIVER_INTENT	Jako předchozí + poslední intent, který byl doručen službě Např. stažení souboru

SPUŠTĚNÍ SLUŽBY

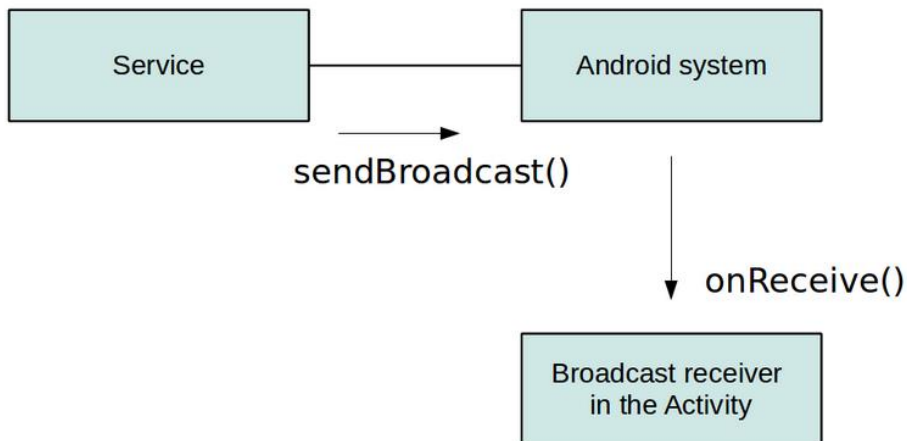
- ◉ předáním intentu v `startService()`
 - Android zavolá `onStartCommand()`
 - předá jí intent

```
Intent intent = new Intent(this, HelloService.class);  
startService(intent);
```

- ◉ pokud služba neběží, zavolá napřed `onCreate()` a potom `onStartCommand()`
- ◉ vícekrát voláno `startService()`
 - vícekrát volána `onStartCommand()`,
 - služba zůstává jen jedna, zastavena když jednou zavoláno `stopSelf()`

JAK VÝSLEDEK ZPÁTKY?

- ◉ klient vytvoří PendingIntent
- ◉ broadcast - `getBroadcast()`
- ◉ doručí jej v Intentu, který startoval službu
- ◉ služba využije broadcast pro doručení výsledku



Zdroj:

<http://www.vogella.com/tutorials/AndroidServices/article.html>

PŘÍKLAD

- ◉ Hlavní aktivita pustí službu
- ◉ Intentová služba (jednodušší kód)
- ◉ Služba stáhne soubor
- ◉ Služba informuje prostřednictvím broadcastu, zda se jí to povedlo

PŘÍKLAD - HLAVNÍ AKTIVITA

Registrace a odregistrace broadcast receiveru:

```
@Override
protected void onResume() {
    super.onResume();
    registerReceiver(receiver, new IntentFilter(DownloadService.NOTIFICATION));
}
@Override
protected void onPause() {
    super.onPause();
    unregisterReceiver(receiver);
}
```

PŘÍKLAD - HLAVNÍ AKTIVITA

Broadcast receiver v hlavní aktivitě:

```
private BroadcastReceiver receiver = new BroadcastReceiver() {  
  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        Bundle bundle = intent.getExtras();  
        if (bundle != null) {  
            String string = bundle.getString(DownloadService.FILEPATH);  
            int resultCode = bundle.getInt(DownloadService.RESULT);  
            if (resultCode == RESULT_OK) {  
                Toast.makeText(MainActivity.this,  
                    "Download complete. Download URI: " + string,  
                    Toast.LENGTH_LONG).show();  
                textView.setText("Download done");  
            } else {  
                Toast.makeText(MainActivity.this, "Download failed",  
                    Toast.LENGTH_LONG).show();  
                textView.setText("Download failed");  
            }  
        }  
    }  
};
```

PŘÍKLAD - HLAVNÍ AKTIVITA

Spuštění služby:

```
public void onClick(View view) {  
  
    Intent intent = new Intent(this, DownloadService.class);  
    // add infos for the service which file to download and where to store  
    intent.putExtra(DownloadService.FILENAME, "index.html");  
    intent.putExtra(DownloadService.URL,  
        "http://www.vogella.com/index.html");  
    startService(intent);  
    textView.setText("Service started");  
}
```

PŘÍKLAD - SLUŽBA

Informuje o svém výsledku broadcastem:

```
private void publishResults(String outputPath, int result) {  
    Intent intent = new Intent(NOTIFICATION);  
    intent.putExtra(FILEPATH, outputPath);  
    intent.putExtra(RESULT, result);  
    sendBroadcast(intent);  
}
```

PŘÍKLAD - SLUŽBA

```
public class DownloadService extends IntentService {

    private int result = Activity.RESULT_CANCELED;
    public static final String URL = "urlpath";
    public static final String FILENAME = "filename";
    public static final String FILEPATH = "filepath";
    public static final String RESULT = "result";
    public static final String NOTIFICATION = "com.vogella.android.service.receiver";

    public DownloadService() { super("DownloadService"); }

    // will be called asynchronously by Android
    @Override
    protected void onHandleIntent(Intent intent) {
        String urlPath = intent.getStringExtra(URL);
        String fileName = intent.getStringExtra(FILENAME);
        File output = new File(Environment.getExternalStorageDirectory(),
            fileName);
        if (output.exists()) {
            output.delete();
        }

        InputStream stream = null;
        FileOutputStream fos = null;
    }
}
```

PŘÍKLAD - SLUŽBA

```
try {
    URL url = new URL(urlPath);
    stream = url.openConnection().getInputStream();
    InputStreamReader reader = new InputStreamReader(stream);
    fos = new FileOutputStream(output.getPath());
    int next = -1;
    while ((next = reader.read()) != -1) {
        fos.write(next);
    }
    // successfully finished
    result = Activity.RESULT_OK;
} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (stream != null) {
        try {
            stream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    if (fos != null) {
        try {
            fos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
publishResults(output.getAbsolutePath(), result);
```

ZASTAVENÍ SLUŽBY

- ◉ služba sama: `stopSelf()`
- ◉ z jiné komponenty: `stopService()`
- ◉ po požadavku systém zruší službu
 - nemá bezprostřední efekt
 - přidá zprávu do fronty zpráv obsluhovaných hlavním vláknem
- ◉ `stopSelf(int)`
 - předáme ID
 - pokud mezitím přijme další požadavek, ID nebude odpovídat a nezastaví ji

VÁZANÁ SLUŽBA

- ◉ implementovat **onBind()**
 - vrací **IBinder** - rozhraní pro komunikaci se službou
- ◉ aplikace volá **bindService()**
 - pro získání rozhraní
- ◉ specifikovat interface, jak bude klient komunikovat se službou
- ◉ více klientů se může připojit ke službě
- ◉ když ji nepotřebujeme: **unbindService()**
- ◉ když už žádný klient, systém službu zruší

VYTVOŘENÍ BOUND SERVICE

jak definovat interface:

- ◉ rozšíření Binder třídy

- vytvořit vlastní interface rozšířením Binderu
- klient může přímo přistupovat k veřejným metodám

- ◉ použití Messengeru

- funkcionalita napříč několika procesy
- definujeme Handler
- posílání příkazů službě přes Message objekty
- všechny požadavky do jednoho vlákna

- ◉ použití AIDL

- Android Interface Definition Language (.aidl)

PŘÍKLAD

```
public class LocalService extends Service {  
    // Binder pro klienty  
    private final IBinder mBinder = new LocalBinder();  
    // generátor čísel (příklad)  
    private final Random mGenerator = new Random();  
  
    public class LocalBinder extends Binder {  
        LocalService getService() {  
            // vrátíme instanci of LocalService  
            // klient může volat public metody  
            return LocalService.this;  
        }  
    }  
}
```

PŘÍKLAD - POKRAČOVÁNÍ

```
public IBinder onBind(Intent intent) {  
    return mBinder;  
}
```

```
/* metody pro klienta */
```

```
public int getRandomNumber() {  
    return mGenerator.nextInt(100);  
}  
}
```

VYUŽITÍ SLUŽBY Z PŘÍKLADU

```
public class BindingActivity extends Activity {  
    LocalService mService;  
    boolean mBound = false;  
  
    protected void onCreate(Bundle savedInstanceState) {..  
  
    protected void onStart() {  
        super.onStart();  
        // Bind to LocalService  
        Intent intent = new Intent(this, LocalService.class);  
        bindService(intent, mConnection,  
Context.BIND_AUTO_CREATE);  
    }  
}
```

VYUŽITÍ SLUŽBY

```
protected void onStop() {  
    super.onStop();  
    // Unbind from the service  
    if (mBound) {  
        unbindService(mConnection);  
        mBound = false;  
    }  
}  
  
public void onClick(View v) {  
    if (mBound) {  
        // Call a method from the LocalService.  
        int num = mService.getRandomNumber();  
        ...  
    }  
}
```

VYUŽITÍ SLUŽBY

```
/** definuje callbacky předané bindService() */
```

```
private ServiceConnection mConnection = new  
ServiceConnection() {
```

```
    public void onServiceConnected(ComponentName  
className, IBinder service) {
```

```
        LocalBinder binder = (LocalBinder) service;
```

```
        mService = binder.getService();
```

```
        mBound = true;
```

```
    }
```

```
{  
    public void onServiceDisconnected(ComponentName arg0)
```

```
    {  
        mBound = false;
```

```
    } };
```

NOTIFIKACE UŽIVATELI

- ◉ služba může informovat uživatele
 - Toast
 - Status Bar Notification
 - může spustit aktivitu, např. prohlížení souboru

SLUŽBA NA POPŘEDÍ

- ◉ musí poskytovat notifikaci pro status bar
- ◉ např. přehrávač hudby
 - označit jako službu na popředí
 - ve status baru zobrazí aktuální písničku
 - aktivní interakce s přehrávačem
- ◉ Na popředí: `startForeground(int, Notification)`
- ◉ Nebude na popředí: `stopForeground()`

SLUŽBA NA POPŘEDÍ

From your **main activity**, start the service with the following code:

```
Intent i = new Intent(context, MyService.class);
context.startService(i);
```

Then in your **service** for `onCreate()` you would build your notification and set it as foreground like so:

```
Intent notificationIntent = new Intent(this, MainActivity.class);

PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,
    notificationIntent, 0);

Notification notification = new NotificationCompat.Builder(this)
    .setSmallIcon(R.mipmap.app_icon)
    .setContentTitle("My Awesome App")
    .setContentText("Doing some work...")
    .setContentIntent(pendingIntent).build();

startForeground(1337, notification);
```

Zdroj: <http://stackoverflow.com/questions/6397754/android-implementing-startforeground-for-a-service>

POSKYTOVATELÉ OBSAHU

- ◉ zpřístupnění uložených dat aplikacím
- ◉ způsob jak sdílet data napříč aplikacemi
- ◉ ve formě tabulky, vrací cursor (př. soubor)

- ◉ Android má poskytovatele obsahu běžných typů
 - audio, video, obrázky, kontakty, kalendář
 - viz [android.provider](#)
- ◉ poskytnout vlastní data
 - vlastní content provider
 - přidat data ke stávajícímu poskytovateli

POSKYTOVANÉ ANDROIDEM

Odkaz:

<http://developer.android.com/reference/android/provider/package-summary.html>

- ◉ Kalendář
- ◉ Kontakty
- ◉ MediaStore
 - Album, umělec, žánr, playlist
- ◉ Telephony
 - SMS, MMS
- ◉ Callog
 - Informace o hovorech (odchozí, příchozí)

PŘÍSTUP KLIENTA

```
ContentResolver cr = getContentResolver();
```

IPC

klient - ContentResolver

server - ContentProvider

- ◉ Android sám identifikuje content provider
- ◉ pomocí URI - jakou tabulku či záznam chceme
- ◉ pojmenované tabulky, pojmenované sloupce
- ◉ každý záznam má jednoznačné **_ID**

URI

- ⊙ jednoznačně identifikuje datovou množinu
- ⊙ všechna začínají **content://**
- ⊙ příklady:

android.provider.**Contacts**.**Phones**.CONTENT_URI

android.provider.**Contacts**.**Photos**.CONTENT_URI

každá položka má unikátní číselný identifikátor

⇒ přidání ID na konec cesty

⇒ `ContentUris.withAppendedId(Uri uri, long id)`

DOTAZ NA POSKYTOVATELE

OBSAHU

- ◉ URI, které identifikuje poskytovatele
- ◉ jména datových položek
- ◉ typy dat těchto položek
- ◉ (ID záznamu) - pro konkrétní záznam

content://.../25 (pro ID 25)

ID lze snadno přidat:

- `ContentUris.withAppendedId()`
- `Uri.withAppendedPath()`

MIME TYPE

- ⦿ metoda `getType (Uri uri)`
- ⦿ MIME type dat, která jsou na daném URI
 - soubor
 - tabulka - type: `vnd.android.cursor.dir`
 - subtype: `vnd.namespace.typ_dat`
 - 1 řádek - type: `vnd.android.cursor.item`

PŘÍKLAD - UDÁLOST V KALENDÁŘI

```
Intent intent = new Intent(Intent.ACTION_EDIT);  
  
intent.setType("vnd.android.cursor.item/event");  
  
intent.putExtra("title", myTitle);  
intent.putExtra("description", myDescription);  
intent.putExtra("eventLocation", myAddress);  
  
startActivity(intent);
```


PŘÍKLAD - KALENDÁŘ

```
if (Build.VERSION.SDK_INT >= 14) {
    Intent intent = new Intent(Intent.ACTION_INSERT)
        .setData(Events.CONTENT_URI)
        .putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME, beginTime.getTimeInMillis())
        .putExtra(CalendarContract.EXTRA_EVENT_END_TIME, endTime.getTimeInMillis())
        .putExtra(Events.TITLE, "Yoga")
        .putExtra(Events.DESRIPTION, "Group class")
        .putExtra(Events.EVENT_LOCATION, "The gym")
        .putExtra(Events.AVAILABILITY, Events.AVAILABILITY_BUSY)
        .putExtra(Intent.EXTRA_EMAIL, "rowan@example.com,trevor@example.com");
    startActivity(intent);
}

else {
    Calendar cal = Calendar.getInstance();
    Intent intent = new Intent(Intent.ACTION_EDIT);
    intent.setType("vnd.android.cursor.item/event");
    intent.putExtra("beginTime", cal.getTimeInMillis());
    intent.putExtra("allDay", true);
    intent.putExtra("rrule", "FREQ=YEARLY");
    intent.putExtra("endTime", cal.getTimeInMillis()+60*60*1000);
    intent.putExtra("title", "A Test Event from android app");
    startActivity(intent);
}
```

PŘÍKLAD - KONTAKTY

```
public void readContacts() {
```

```
    ContentResolver cr = getContentResolver();
```

```
    Cursor cur =
```

```
    cr.query(ContactsContract.Contacts.CONTENT_URI, null, null,  
    null, null);
```

```
    if (cur.getCount() > 0) {
```

```
        while (cur.moveToNext()) { ... }
```

```
    }
```

```
}
```

DOTAZOVÁNÍ

- ◉ ContentResolver.**query()**
- ◉ Activity.**managedQuery()**
- ◉ vrací objekt Cursor
- ◉ stejné argumenty

```
Uri myPerson =  
ContentUris.withAppendedId(People.CONTENT_URI, 23);  
Cursor cur = managedQuery(myPerson, null, null, null, null);
```

Dnes se místo `managedQuery()` používá `LoadCursor`

NAHRAZENÍ MANAGED QUERY

```
Cursor c =  
managedQuery(Contacts.CONTENT_URI, null,  
null, null, Contacts.DISPLAY_NAME);
```

```
Cursor cursor =  
getContentResolver().query(contentUri, null,  
null, null, Contacts.DISPLAY_NAME);
```

K ČEMU SLOUŽILO MANAGED QUERY?

- ◉ managedQuery volalo startManagingCursor (cursor)
- ◉ Životní cyklus cursoru v závislosti na aktivitě
- ◉ Deaktivován, když aktivita stopped
- ◉ Uzavřen, když aktivita destroyed
- ◉ Requery, když je aktivita stopped -> restarted

Problémy:

- ◉ Nezachovávalo data např. při otočení obrazovky - znovu requery - zpomalení
- ◉ Dotazy dělá na UI threadu

NÁHRADA - LOADER

- ◉ Operace s kurzorem - asynchronně, neblokuje UI vlákno
- ◉ Při restartu aktivity kvůli konfiguračním změnám zachovává data
- ◉ Monitoruje data - requery při změně dat

<http://www.androiddesignpatterns.com/2012/07/loaders-and-loadermanager-background.html>

LOADER MANAGER

```
public class SampleActivity extends Activity implements LoaderManager.LoaderCallbacks<D> {  
    public Loader<D> onCreateLoader(int id, Bundle args) { ... }  
    public void onLoadFinished(Loader<D> loader, D data) { ... }  
    public void onLoaderReset(Loader<D> loader) { ... }  
  
    /* ... */  
}
```

onCreateLoader - vrací nový Loader

onLoadFinish - zde klient updatuje UI novými daty

onLoaderReset - reset, zbavit se starých dat

Příklad:

<http://www.androiddesignpatterns.com/2012/07/understanding-loadermanager.html>

PARAMETRY QUERY()

- ◉ názvy datových sloupců, které vrátíme
 - null = všechny sloupce
- ◉ jaké řádky vrátet
 - null = všechny řádky
 - klauzule WHERE SQL dotazu (bez WHERE)
- ◉ argumenty výběru
- ◉ setřídění
 - null = defaultní, nesetříděné
 - klauzule ORDER BY

CO DOTAZ VRACÍ

- ◉ množina 0 nebo více záznamů
- ◉ **_ID** jednoznačné číslo záznamu
- ◉ **_COUNT** počet záznamů (stejně pro vš. řádky)
- ◉ pomocí kurzoru můžeme iterovat přes množinu výsledků - pouze pro čtení
- ◉ úprava, přidání dat - přes ContentResolver objekt

ZPRACOVÁNÍ DAT

```
if (cur.moveToFirst()) {  
    String name;  
    String phoneNum;  
    int nameColumn = cur.getColumnIndex(People.NAME);  
    int phoneColumn = cur.getColumnIndex(People.NUMBER);  
  
    do {  
        name = cur.getString(nameColumn);  
        phoneNum = cur.getString(phoneColumn);  
        ..  
    } while (cur.moveToNext());  
}
```

BINÁRNÍ DATA

- ◉ obrázky, zvuky, ...
- ◉ přímo v tabulce (BLOB)
 - menší data cca do 50KB
 - **cursor.getBlob()**
 - vrací byte array
- ◉ řetězec **content**:
 - **ContentResolver.openInputStream()**
 - získat InputStream pro čtení dat

MODIFIKACE DAT

- ◉ přidání záznamů
 - ◉ změna záznamů
 - ◉ mazání záznamů
-
- ◉ metody ContentResolveru
 - ◉ příslušná práva

PŘIDÁNÍ ZÁZNAMŮ

- ◉ nastavit pár klíč - hodnota
 - objekt ContentValues
 - klíč - jméno sloupce, hodnota - co chceme vložit
- ◉ `contentResolver.insert()`
 - předat URI poskytovatele
 - předat ContentValues mapu
 - vrátí URI nového záznamu (včetně ID)

PŘÍKLAD

```
import android.provider.Contacts.People;  
import android.content.ContentResolver;  
import android.content.ContentValues;
```

```
ContentValues values = new ContentValues();
```

```
// přidání kontaktu, nastavení jako oblíbený
```

```
// 1 = jako favorites
```

```
values.put(People.NAME, "Bart Simpson");
```

```
values.put(People.STARRED, 1);
```

```
Uri uri =
```

```
    getContentResolver().insert(People.CONTENT_URI, values);
```

PŘIDÁNÍ HODNOT K ZÁZNAMU

- ◉ můžeme měnit hodnoty záznamu, přidávat informace...

z předchozího slidu

```
phoneUri = Uri.withAppendedPath(uri,  
People.Phones.CONTENT_DIRECTORY);
```

```
values.clear();
```

```
values.put(People.Phones.TYPE, People.Phones.TYPE_MOBILE);
```

```
values.put(People.Phones.NUMBER, "1233214567");
```

```
getContentResolver().insert(phoneUri, values);
```

POKRAČOVÁNÍ

// přidáme e-mail

```
emailUri = Uri.withAppendedPath(uri,  
People.ContactMethods.CONTENT_DIRECTORY);  
values.clear();
```

// ContactMethods.KIND pro rozlišení email, IM, ...

```
values.put(People.ContactMethods.KIND, Contacts.KIND_EMAIL);  
values.put(People.ContactMethods.DATA, "test@example.com");  
values.put(People.ContactMethods.TYPE,  
People.ContactMethods.TYPE_HOME);
```

```
getContentResolver().insert(emailUri, values);
```


BINÁRNÍ DATA - VLOŽENÍ

- ◉ `ContentValues.put()`
 - malá binární data
- ◉ `ContentResolver.openOutputStream()`
 - content: URI

SMAZÁNÍ ZÁZNAMU

- ◉ **ContentResolver.delete()**
 - URI určité řádky
 - URI typ záznamu a jaké (clause WHERE)

VYTVOŘENÍ CONTENT PROVIDERA

- ◉ úložiště dat
 - file storage, SQL databáze, ...
- ◉ rozšíření ContentProvider pro přístup k datům
- ◉ deklarovat poskytovatele obsahu v manifestu aplikace

ROZŠÍŘENÍ TŘÍDY CONTENT PROVIDER

- ◉ query()
 - vrací Cursor
 - ◉ SQLiteCursor
 - ◉ MatrixCursor
- ◉ insert()
- ◉ update()
- ◉ delete()
- ◉ getType()
- ◉ onCreate()

DALŠÍ

◉ public static final URI **CONTENT_URI**

- definovat
- obsahuje content: , který náš poskytovatel obsluhuje

```
public static final Uri CONTENT_URI =  
Uri.parse("content://com.example.transportationprovider");
```

◉ definovat jména sloupců, které se vrací klientovi

- odpovídají sloupcům v SQL databázi
- zahrnout integer _id
 - v SQLite `_ID INTEGER PRIMARY KEY AUTOINCREMENT`

DALŠÍ

- ◉ datové typy každého sloupce
 - klient je potřebuje pro čtení dat
- ◉ nový datový typ
 - nový MIME vrácený `ContentProvider.getType()`

pro 1 záznam:

`vnd.android.cursor.item/vnd.spolecnost.contenttyp`

žádost o záznam vlaku 122:

`content://com.example.transportationprovider/trains/122`

vrací MIME typ:

`vnd.android.cursor.item/vnd.example.rail`

MIME

pro více záznamů:

`vnd.android.cursor.dir/vnd.spolecnost.contenttype`

žádost o všechny vlakové záznamy:

`content://com.example.transportationprovider/trains`

vrací MIME:

`vnd.android.cursor.dir/vnd.example.rail`

DALŠÍ

◉ pro velká data

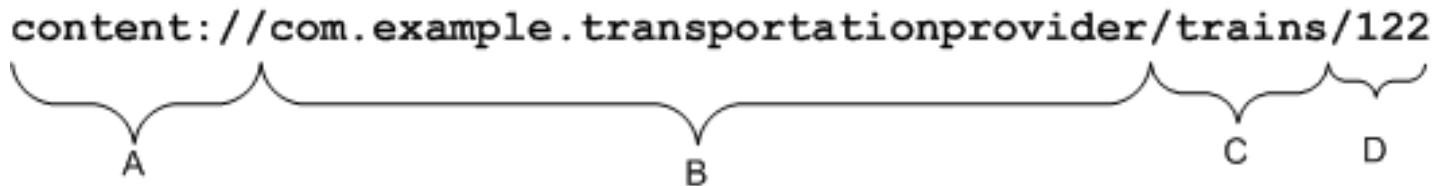
- **content:** URI string
- další pole **_data** - přesná cesta k souboru
 - není přímo pro klienta, ale pro ContentResolvera
- klient volá `ContentResolver.openInputStream()`
- CR si vyžádá `_data` položku záznamu
- CR má vyšší práva než klient, dostane se k souboru

DEKLARACE CONTENT PROVIDERA

- ◉ element **<provider>** v manifestu
 - není-li deklarovaný, Android jej nevidí
- ◉ atribut **name**
 - plně kvalifikované jméno CP
- ◉ atribut **authorities**
 - authority část content: URI
- ◉ další atributy
 - práva číst-zapisovat data
 - enable / disable poskytovatele
 - multiprocess true - dovolí instanci v každém klientském procesu

CONTENT URI - SOUHRNNĚ

`content://com.example.transportationprovider/trains/122`



A - říká, že jde o content provider

B - authority část URI, identifikuje content providera
plně kvalifikované jméno třídy (na malá písmena)
deklarovaná v elementu <provider>

C - cesta, určuje požadovaná data
0 (jen 1 typ dat) či více segmentů
(land/bus, land/train, sea/ship, sea/submarin,...)

D - id konkrétního záznamu (_ID)

UKÁZKA

- ◉ požádáme o výběr kontaktu

```
Intent intent = new Intent(Intent.ACTION_PICK,  
ContactsContract.Contacts.CONTENT_URI);  
startActivityForResult(intent, cisloZadosti);
```

- ◉ uživateli se spustí aktivita, vybere kontakt
- ◉ vrátí se nám vybraný kontakt

```
protected void onActivityResult  
(int requestCode, int resultCode, Intent data) {
```

```
    Uri vybranyKontakt = data.getData();  
    ..zobraz.. vybranyKontakt.toString();  
}
```

zdroj: <http://www.root.cz/clanky/programovani-pro-android-v-prikladech/>

UKÁZKA

URI kontaktu

content://
com.android.contacts/
contacts/
lookup/0r2-39434B2F3141435
1434D4157/2

OK

URI kontaktu

content://
com.android.contacts/
contacts/
lookup/0r1-4D433F274B3F274
94157/1

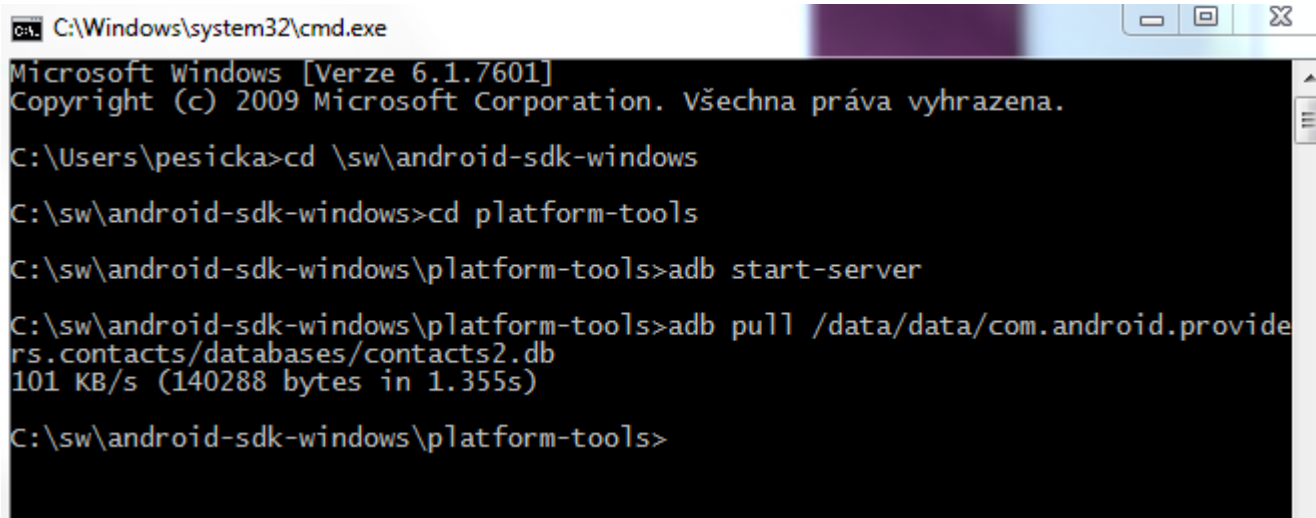
OK

URI kontaktu

content://
com.android.contacts/
contacts/
lookup/0r3-452F4D494D49452
74B3D373B/3

OK

DATABÁZE KONTAKTŮ



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Verze 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Všechna práva vyhrazena.

C:\Users\pesicka>cd \sw\android-sdk-windows
C:\sw\android-sdk-windows>cd platform-tools
C:\sw\android-sdk-windows\platform-tools>adb start-server
C:\sw\android-sdk-windows\platform-tools>adb pull /data/data/com.android.providers.contacts/databases/contacts2.db
101 KB/s (140288 bytes in 1.355s)
C:\sw\android-sdk-windows\platform-tools>
```

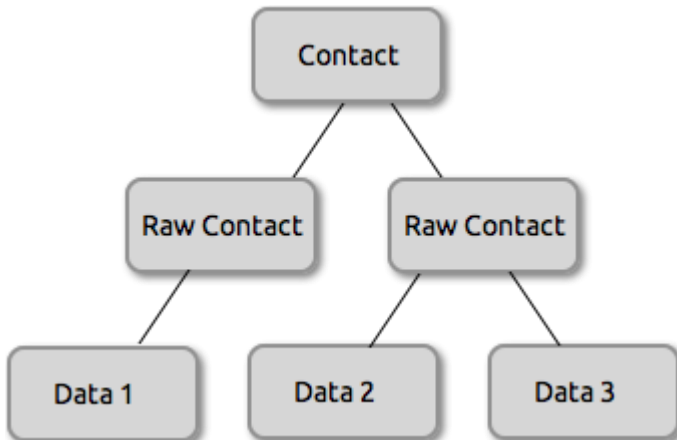
stáhnutí databáze kontaktů:

`adb pull`

`/data/data/com.android.providers.contacts/databases/contacts2.db`

stáhnout dále např. SQLite Database Browser

DATABÁZE KONTAKTŮ



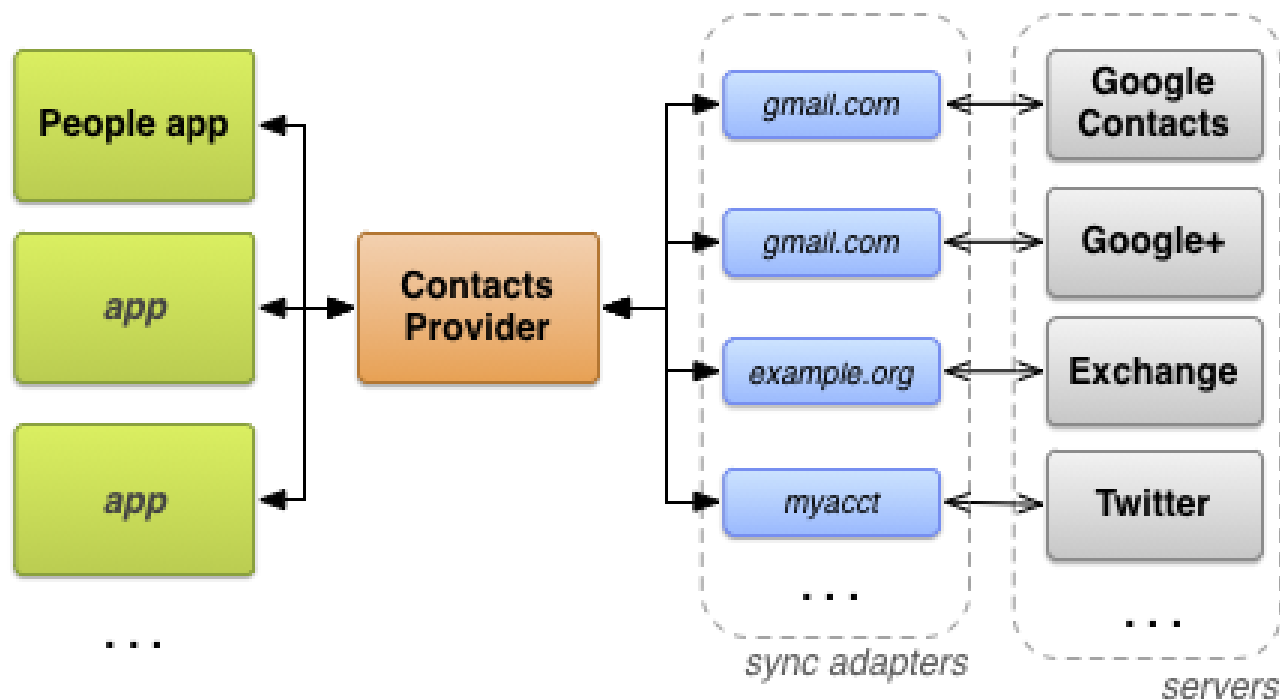
jeden kontakt

= jeden záznam v tabulce Contact

= jeden nebo více záznamů v Raw Contact (Google, další adresáře)

= rozšiřující údaje v tabulce data

DATABÁZE KONTAKTŮ



viz

<http://developer.android.com/guide/topics/providers/contacts-provider.html>

DATABÁZE KONTAKTŮ

SQLite Database Browser - C:/sw/android-sdk-windows/platform-tools/contacts2.db

File Edit View Help

Database Structure Browse Data Execute SQL

SQL string:

```
select * from contacts;
```

Execute query

Error message from database engine:

No error

Data returned:

id	name raw	photo id	custom rir	send to v	times con	last time	starred	in visible	has phone	lookup	status up	single
1	1			0	0	0	0	1	1	0r1-4D433F274B3F27494157	0	
2	2			0	0	0	0	1	1	0r2-39434B2F31414351434...	0	
3	3			0	0	0	0	1	1	0r3-452F4D494D4945274B3...	0	

DATABÁZE KONTAKTŮ

SQLite Database Browser - C:/sw/android-sdk-windows/platform-tools/contacts2.db

File Edit View Help

Database Structure Browse Data Execute SQL

SQL string:

```
select * from raw_contacts;
```

Execute query

Error message from database engine:

No error

Data returned:

id	is r	aco	aco	sou	versio	dirt	dele	con	aqq	aqq	cus	sen	time	last	star	display name	disc	disc	pho	pho	sort key	sort key alt	nar
1	0				2	1	0	1	0	0		0	0	0	0	Tomas Marny	M...	40	3	Tomas Marny	Marny, Tomas	0	
2	0				2	1	0	2	0	0		0	0	0	0	Josef Novotny	N...	40	3	Josef Novotny	Novotny, Josef	0	
3	0				2	1	0	3	0	0		0	0	0	0	Petr Trpaslik	Tr...	40	3	Petr Trpaslik	Trpaslik, Petr	0	

DATABÁZE KONTAKTŮ

SQLite Database Browser - C:/sw/android-sdk-windows/platform-tools/contacts2.db

File Edit View Help

Database Structure Browse Data Execute SQL

SQL string:

```
select * from data;
```

Execute query

Error message from database engine:

No error

Data returned:

id	pack	mime	raw	is pr	is su	data	data1	data2	data3	data	data	data	data	data	data	data	data	data	data	data	data	data
1	5	1	0	0	0		1 234-567-890	2														
2	6	1	0	0	0		Tomas Marny	Tomas	Marny					1	3							
3	5	2	0	0	0		5555555555	3														
4	6	2	0	0	0		Josef Novotny	Josef	Novotny					1	3							
5	5	3	0	0	0		(999) 999-9999	2														
6	6	3	0	0	0		Petr Trpaslik	Petr	Trpaslik					1	3							

KONTAKTY

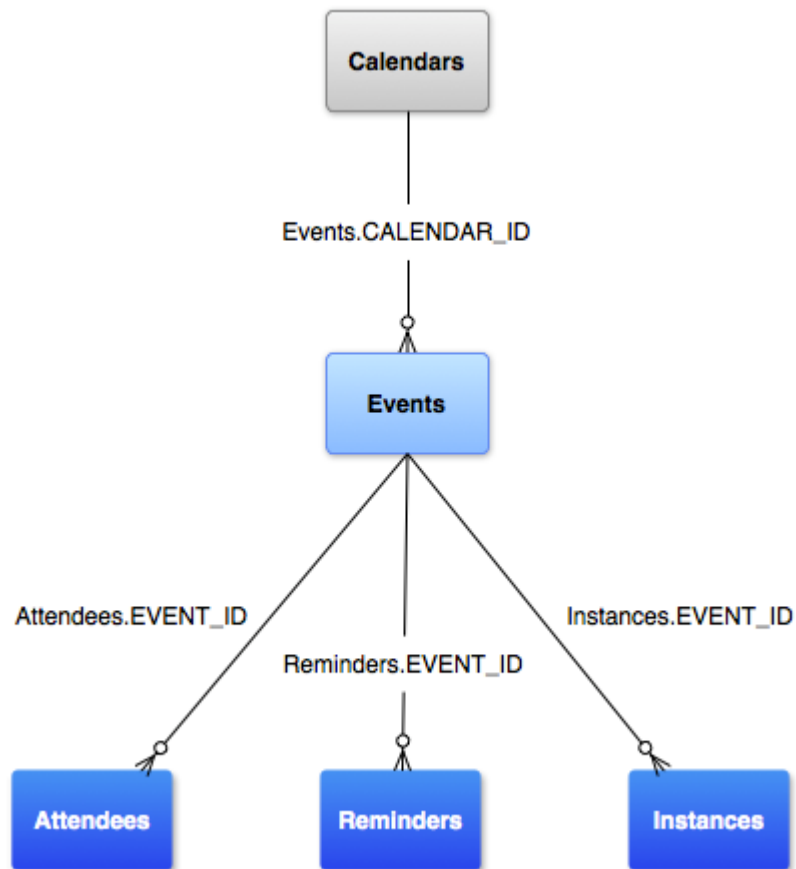
tutorial

<http://app-solut.com/blog/2011/03/working-with-the-contactscontract-to-query-contacts-in-android/>

odkaz na vzorový příklad

ContactsContract API L5 (Android 2.0)

CALENDAR PROVIDER



viz

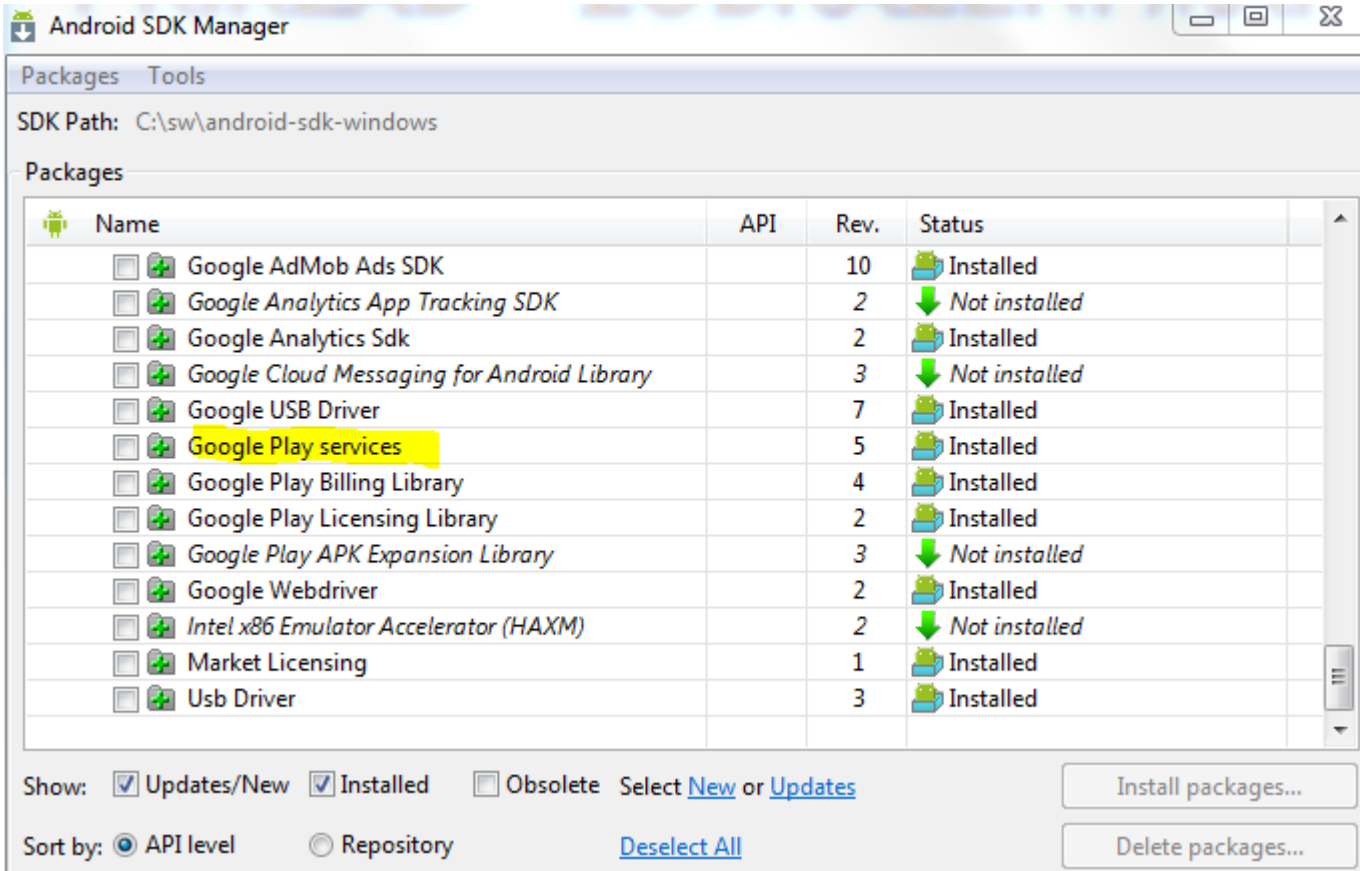
<http://developer.android.com/guide/topics/providers/calendar-provider.html>

<class>.CONTENT_URI
např.: Events.CONTENT_URI

více kalendářů
(jméno, barva, synchronizace)

event
instances
výskyt dané události

GOOGLE PLAY SERVICES



The screenshot shows the Android SDK Manager window. The 'Packages' tab is selected, and the SDK Path is set to 'C:\sw\android-sdk-windows'. The list of packages is as follows:

Name	API	Rev.	Status
<input type="checkbox"/> Google AdMob Ads SDK		10	Installed
<input type="checkbox"/> Google Analytics App Tracking SDK		2	Not installed
<input type="checkbox"/> Google Analytics Sdk		2	Installed
<input type="checkbox"/> Google Cloud Messaging for Android Library		3	Not installed
<input type="checkbox"/> Google USB Driver		7	Installed
<input type="checkbox"/> Google Play services		5	Installed
<input type="checkbox"/> Google Play Billing Library		4	Installed
<input type="checkbox"/> Google Play Licensing Library		2	Installed
<input type="checkbox"/> Google Play APK Expansion Library		3	Not installed
<input type="checkbox"/> Google Webdriver		2	Installed
<input type="checkbox"/> Intel x86 Emulator Accelerator (HAXM)		2	Not installed
<input type="checkbox"/> Market Licensing		1	Installed
<input type="checkbox"/> Usb Driver		3	Installed

At the bottom, the 'Show' section has checkboxes for 'Updates/New' (checked), 'Installed' (checked), and 'Obsolete' (unchecked). The 'Sort by' section has radio buttons for 'API level' (selected) and 'Repository' (unchecked). There are buttons for 'Install packages...', 'Delete packages...', and a link for 'Deselect All'.

GOOGLE PLAY SERVICES



Android 2.2 (Froyo) a výše

LITERATURA

- ◉ Android SDK
- ◉ příklady z netu
 - <http://www.ibm.com/developerworks/opensource/library/os-android-networking/>
 - <http://mobiforge.com/developing/story/using-google-maps-android>
 - a další