Android

Zarządzanie cyklem życia i stanem

Piotr Fulmański

Instytut Nauk Ekonomicznych i Informatyki,
Państwowa Wyższa Szkoła Zawodowa w Płocku, Polska

March 7, 2016

Table of contents

Co w tym wykładzie

- Wprowadzenie do procesów aplikacji i identyfikatorów użytkowników
- Cykl życia aktywności
- Obsługa stanu egzemplarzy klasy aktywności
- Wprowadzenie do zadań i pokrewieństwo zadań

Techniczna definicji pojedynczej aplikacji

Aplikacja na Android (w technicznym ujęciu) obejmuje wszystkie komponenty uruchomione z tym samym identyfikatorem użytkownika i w jednym procesie oraz powiązane z nadrzędnym obiektem klasy Application. Obiekt ten pełni funkcję centralnego kontekstu dla wszystkich komponentów (aktywności, usług, odbiorników typu BroadcastReceiver i dostawców treści). Wszystkie te elementy są łączone w plik APK.

Cykl życia aplikacji

Każda aplikacja na Android jest „zapisywana” w obiekcie klasy Application. Cykl życia obiektu klasy Application

1. W momencie uruchamiania aplikacji wywoływana jest metoda onCreate .
2. Kiedy system żąda od aplikacji przywrócenia możliwych zasobów, wywoływana jest metoda onLowMemory .
3. Czasem w momencie zatrzymywania aplikacji wywoływana jest metoda onTerminate .
4. Kiedy w trakcie działania aplikacji zmienia się konfiguracja urządzenia, wywoływana jest metoda onConfigurationChanged

Ważną kwestią związaną z obiektami klasy Application jest to, że są one tworzone w momencie uruchamiania całej aplikacji, ale nie są usuwane do czasu jej zamknięcia. Są trwalsze od aktywności, usług, odbiorników typu Broadcast Receiver i innych komponentów.
Android dla każdej aplikacji stosuje odrębny proces i identyfikator użytkownika.

Identyfikator użytkownika, proces i wątki aplikacji

Kiedy użytkownik po raz pierwszy żąda komponentu (klasy Activity, Service, BroadcastReceiver lub ContentProvider) aplikacji na Android, jest ona uruchamiana z unikatowym identyfikatorem użytkownika (identyfikatory użytkownika to zwykle `app_n`, gdzie wartość `n` jest zwiększana dla każdej aplikacji) i umieszczana w nowym procesie systemowym powiązanym z tym identyfikatorem.
Każdy proces domyślnie działa w jednym wątku głównym (klasa Thread). Wątek główny jest często nazywany wątkiem interfejsu użytkownika, jest to jednak mylące określenie, ponieważ oprócz aktywności z wątku korzystają także odbiorniki typu BroadcastReceiver, dostawcy treści i usługi.
Choć Android udostępnia sensowne i łatwe w użyciu ustawienia domyślne, to jednocześnie zapewnia pełną kontrolę: można uruchomić kilka aplikacji w tym samym procesie lub z wykorzystaniem tego samego identyfikatora użytkownika. Można też uruchomić jedną aplikację w kilku procesach.

Priorytety procesów

Priorytet: 1 *Działający na pierwszym planie* Proces z aktywnością, z której korzysta użytkownik, z usługą powiązaną z używaną aktywnością, z usługą wykonującą jedną z metod obsługi cyklu życia lub z działającym odbiornikiem typu BroadcastReceiver.

Priorytet: 2 *Widoczny* Proces, który nie jest używany na pierwszym planie, ale obejmuje aktywność potencjalnie wpływającą na wygląd ekranu lub usługę powiązaną z tego rodzaju aktywnością.

Priorytet: 3 *Usługa* Proces obejmujący usługę uruchomioną za pomocą metody startService (i który nie spełnia kryteriów dla procesów z pierwszego planu i widocznych).

Priorytet: 4 *Działający w tle* Proces obejmujący zatrzymaną aktywność. Istnieć może wiele procesów tego rodzaju.

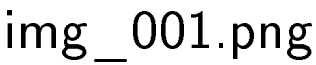
Priorytet: 5 *Pusty* Proces, który nie obejmuje aktualnych komponentów aplikacji.

Komponenty odbiorników typu BroadcastReceiver, usług i dostawców treści (podobnie jak aktywności) są domyślnie wiązane z głównym procesem aplikacji. Choć korzystają z tego samego procesu, mają inny cykl życia (z innymi metodami). Jednym z najważniejszych aspektów pisania aplikacji na Android jest cykl życia aktywności.

Cykl życia aktywności

Aktywności mają trzy hierarchiczne etapy cyklu życia:

- cały cykl życia (od utworzenia do usunięcia),
- cykl życia w stanie widoczności (od ponownego uruchomienia do zatrzymania),
- cykl życia w stanie działania na pierwszym planie (od wznowienia do wstrzymania).



Definition (onCreate)

Wywoływana przy początkowym tworzeniu aktywności. To tu należy przeprowadzić standardową statyczną konfigurację – utworzyć widoki, powiązać dane z listami itd. Udostępnia też obiekt klasy Bundle obejmujący wcześniej zamrożony stan aktywności (jeśli taki obiekt istnieje).
Następną metodą zawsze jest onStart .

Definition (onRestart)

Wywoływana po zatrzymaniu aktywności, ale przed jej ponownym uruchomieniem.
Następną metodą zawsze jest onStart .

Definition (onStart)

Wywoływana, kiedy aktywność staje się widoczna dla użytkownika. Następną metodą jest onResume, jeśli aktywność zaczyna działać na pierwszym planie, lub onStop, jeżeli jest ukrywana.

Definition (onResume)

Wywoływana, kiedy aktywność wchodzi w interakcje z użytkownikiem. Na tym etapie aktywność znajduje się na szczycie stosu aktywności i trafiają do niej dane wejściowe od użytkownika.
Następną metodą zawsze jest onStart .

Definition (onPause)

Wywoływana, kiedy system ma zacząć wznawianie wcześniejszej aktywności. Zwykle zatwierdzane są wtedy niezapisane zmiany w trwałych danych, zatrzymywane są animacje oraz inne operacje obciążające procesor itd. Metoda musi działać szybko, ponieważ następna aktywność nie zostanie wznowiona do czasu zwrócenia sterowania przez tę metodę. Następną metodą jest onResume, jeśli aktywność zaczyna działać na pierwszym planie, lub onStop, jeżeli jest ukrywana.

Definition (onStop)

Wywoływana, kiedy aktywność nie jest widoczna dla użytkownika, ponieważ inna aktywność została wznowiona i zakrywa daną. Może to być wynik uruchomienia nowej aktywności, przeniesienia innej istniejącej aktywności przed daną lub usuwania danej aktywności. Następną metodą jest onRestart, jeśli dana aktywność ponownie wchodzi w interakcje z użytkownikiem, lub onDestroy, jeżeli aktywność jest usuwana.

Definition (onDestroy)

Jest to ostatnie wywołanie zgłaszane przed usunięciem aktywności. Może to mieć miejsce z powodu kończenia pracy aktywności (wywołania dla niej metody finish) lub z uwagi na tymczasowe usunięcie danego egzemplarza aktywności przez system w celu odzyskania zasobów. Do rozróżniania tych sytuacji służy metoda isFinishing.

Metody cyklu życia

Klasa bazowa

```
public abstract class LifecycleActivity extends Activity {
  private static final String LOG_TAG = "LifecycleExplorer";
  private NotificationManager notifyMgr;
  private boolean enableNotifications;
  private final String className;

  public LifecycleActivity() {
    super();
    this.className = this.getClass().getName();
  }

  public LifecycleActivity(final boolean enableNotifications) {
    this();
    this.enableNotifications = enableNotifications;
  }
```

Metody cyklu życia

Klasa bazowa

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  notifyMgr = (NotificationManager)
  getSystemService(Context.NOTIFICATION_SERVICE);
  debugEvent("onCreate");
}

@Override
  protected void onStart() {
  debugEvent("onStart");
  super.onStart();
}

@Override
protected void onResume() {
  debugEvent("onResume");
  super.onResume();
}
```

```
@Override
protected void onPause() {
  debugEvent("onPause");
  super.onPause();
}
```

```
private void debugEvent(final String method) {
  long ts = System.currentTimeMillis();
  Log.d(LOG_TAG, " *** " + method + " " + className + " " + ts
  if (enableNotifications) {
    Notification notification =
      new Notification(android.R.drawable.star_big_on,
        "Zdarzenia cyklu życia: " + method, 0L);
    RemoteViews notificationContentView =
      new RemoteViews(getPackageName(),
        R.layout.custom_notification_layout);
    notification.contentView = notificationContentView;
```

```
        notification.contentIntent =
          PendingIntent.getActivity(this, 0, null, 0);
        notification.flags |= Notification.FLAG_AUTO_CANCEL;
        notificationContentView.setImageViewResource(
          R.id.image, android.R.drawable.btn_star);
        notificationContentView.setTextViewText(
          R.id.lifecycle_class, getClass().getName());
        notificationContentView.setTextViewText(
          R.id.lifecycle_method, method);
        notificationContentView.setTextColor(
          R.id.lifecycle_method, R.color.black);
        notificationContentView.setTextViewText(
          R.id.lifecycle_timestamp, Long.toString(ts));
        notifyMgr.notify((int) System.currentTimeMillis(), notific
      }
    }
}
```

Metody cyklu życia

Klasa główna, ekran główny aplikacji

```
public class Main extends LifecycleActivity {
  private Button finish;
  private Button activity2;
  private Chronometer chrono;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    finish = (Button) findViewById(R.id.finishButton);
    finish.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        finish();
      }
    });
```

```
    activity2 = (Button) findViewById(R.id.activity2Button);
    activity2.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        startActivity(new Intent(Main.this,
          Activity2.class));
      }
    });
    chrono = (Chronometer) findViewById(R.id.chronometer);
}

@Override
protected void onResume() {
    super.onResume();
    chrono.setBase(SystemClock.elapsedRealtime());
    chrono.start();
}
```

```
  @Override
  protected void onPause() {
    chrono.stop();
    super.onPause();
  }
}
```

Zmiany konfiguracji

Pułapką, na którą trzeba uważać w Androidzie, jest to, że przy zmianie konfiguracji system domyślnie usuwa i odtwarza bieżącą aktywność. Ponieważ zmiana orientacji (z pionowej na poziomą lub na odwrót) także jest rodzajem zmiany konfiguracji, prowadzi to do wielu operacji usuwania i odtwarzania aktywności. Ma to miejsce za każdym razem, kiedy użytkownik obraca telefon lub wysuwa klawiaturę.

Zapisywanie i odtwarzanie stanu egzemplarza

Stan egzemplarza (ang. *instance state*) to stan, który aktywności muszą same przywrócić do postaci, w jakiej użytkownik go pozostawił. Najważniejsze jest to, że stan egzemplarza jest zapisywany wtedy, kiedy to system (a nie programista) usuwa aktywność. Pamiętajmy jednak, że

- Stan egzemplarza jest zapisywany, kiedy system usuwa aktywność (na przykład w wyniku zmiany konfiguracji).
- Stan egzemplarza nie jest zapisywany po wywołaniu metody `finish` (jest ona wywoływana domyślnie po wciśnięciu przycisku Back).

Zapisywanie i odtwarzanie stanu egzemplarza

Stan egzemplarza jest zachowywany w obiekcie klasy Bundle. Jest to pakiet danych typu Parcelable (można je przekazywać między procesami). Może obejmować dane typów prostych, łańcuchy znaków i tablice elementów tych typów. Możliwe jest też przekazywanie innych typów Parcelable.
System zapisuje sensowne informacje domyślne na temat stanu egzemplarza, można jednak przesłonić metodę onSaveInstanceState i zastąpić ją lub wzbogacić.
Elementy są odtwarzane albo w metodzie onCreate, która przyjmuje jako dane wejściowe obiekt klasy Bundle, albo w metodzie onRestoreInstanceState. Do odtwarzania wartości najczęściej stosuje się metodę onCreate, jednak można też użyć metody onRestoreInstanceState, co pozwala oddzielić odtwarzanie od inicjowania innych komponentów.

Zapisywanie i odtwarzanie stanu egzemplarza

```
private static final String COUNT_KEY = "cKey";
private TextView numResumes;
private int count;

@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity3);
  numResumes = (TextView) findViewById(R.id.numResumes);
}

@Override
protected void onResume() {
  super.onResume();
  numResumes.setText(String.valueOf(count));
  count++;
}
```

Zapisywanie i odtwarzanie stanu egzemplarza

```
@Override
protected void onRestoreInstanceState(Bundle savedInstanceState)
  if ((savedInstanceState != null) &&
    savedInstanceState.containsKey(COUNT_KEY)) {
    count = savedInstanceState.getInt(COUNT_KEY);
  }
  super.onRestoreInstanceState(savedInstanceState);
}

@Override
protected void onSaveInstanceState(Bundle outState) {
  outState.putInt(COUNT_KEY, count);
  super.onSaveInstanceState(outState);
}
```

Korzystanie ze stanu egzemplarza niezwiązanego z konfiguracją

Stan egzemplarza niezwiązany z konfiguracją to dowolne rozbudowane dane o stanie, które trzeba przekazać z bieżącego egzemplarza aktywności do jego przyszłego odpowiednika, tworzonego w wyniku wprowadzenia zmian w konfiguracji. Technika ta dotyczy tylko bieżącego egzemplarza oraz egzemplarza odtwarzanego natychmiast po usunięciu pierwotnego. Ponadto trzeba uważać, aby nie przekazywać danych w rodzaju łańcuchów znaków, elementów graficznych lub innych zasobów, które mogą ulec zmianie w wyniku modyfikacji konfiguracji. W końcu omawiana technika dotyczy stanu niezwiązanego z konfiguracją.

Korzystanie ze stanu egzemplarza niezwiązanego z konfiguracją

Uwaga: Metody `onRetainNonConfigurationInstance( )` oraz `getLastNonConfigurationInstance( )` zostały wycofane począwszy od Android-a w wersji 3.0 (API level 13). Właściwym zamiennikiem dla nich są metody `setRetainInstance(boolean)` oraz `getRetainInstance()`, które jednakże działają na fragmentach a tymi zajmiemy się dopiero na kolejnych zajęciach. W takim razie dobrze przed wywołaniem metody `onRetainNonConfigurationInstance( )` dodać adnotację `@SuppressWarnings("deprecation")`.

Korzystanie ze stanu egzemplarza niezwiązanego z konfiguracją

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity3);
  numResumes = (TextView) findViewById(R.id.numResumes);
  Date date = (Date) this.getLastNonConfigurationInstance();
  if (date != null) {
    Toast.makeText(this, "Obiekt \"LastNonConfiguration\": "
      + date, Toast.LENGTH_LONG).show();
  }
}

[...]

@SuppressWarnings("deprecation")
@Override
public Object onRetainNonConfigurationInstance() {
  return new Date();
```

Wykonywanie operacji za pomocą zadań

W ten sposób zacierają się granice między aplikacjami. Dla użytkownika aplikacja składa się z aktywności z różnych miejsc (map, przeglądarki, klienta pocztowego, kontaktów i aparatu fotograficznego – to tylko kilka wbudowanych możliwości). Ważny jest cel użytkownika – niezależnie od tego, jak wiele komponentów służy do jego realizacji. W Androidzie takie operacje z udziałem wielu aplikacji to zadania.
Omówiono już techniczną definicję grupy komponentów z jednego podstawowego pakietu spakowanego do pliku APK. Jest to aplikacja na Android. Jednak, jak wcześniej wspomniano, użytkownik nie postrzega aplikacji w ten sposób. Dla użytkownika aplikacja składa się ze wszystkich aktywności potrzebnych do wykonania operacji. W Androidzie takie grupy aktywności to zadania.
Zadanie jest zawsze uruchamiane przez jedną aktywność – tak zwaną aktywność główną. Aktywność główna jest zwykle uruchamiana z poziomu ekranu głównego (aplikacji Launcher). Następnie każda aktywność związana z zadaniem jest dodawana do stosu aktywności zadania, a całe zadanie system traktuje jak odrębną jednostkę.
Aktywności wywoływane przez aplikację łączy pokrewieństwo z zadaniem.

Podsumowanie

W tym rozdziale skoncentrowano się na tym, czym są aplikacje na Android i jak wygląda cykl życia aktywności. Aktywności są podstawowym komponentem każdej aplikacji na Android, a tworzenie i usuwanie elementów w ramach cyklu życia aktywności ma bardzo duże znaczenie. Oprócz znajomości cyklu życia niezwykle istotna jest też wiedza o tym, jak zapisywać i odtwarzać stan egzemplarzy aktywności. Od tego może zależeć komfort pracy użytkowników. Zarządzanie cyklem życia w środowisku, które nie gwarantuje działania aplikacji do czasu jej zamknięcia, a zamiast tego usuwa i tworzy komponenty na żądanie, bywa skomplikowane.
Inną ważną kwestią jest grupowanie przez Android aktywności według celów użytkownika (niezależnie od aplikacji, z których te aktywności pochodzą). System traktuje takie grupy jak odrębne zadania. Są one istotne, ponieważ stanowią logiczne punkty nawigacyjne dla użytkowników i dzięki stosowi aktywności umożliwiają powrót do punktu wyjścia.