4-ЛЕКЦИЯ. СОБЫТИЯ И ПРОЦЕССЫ В ПРИЛОЖЕНИЯХ

4.1. Процессы при создании приложении и управление составляющими приложения

Android пытается поддерживать процесс приложения как можно дольше, но когда потребуются ресурсы старые процессы будут вытеснены по иерархии важности.

Существует 5 уровней иерархии важности (процессы первого уровня из списка будут удалены последними) :

1.Процесс с которым взаимодействует пользователь(Foreground process)
К таким процессам относится например: активити с которым взаимодействует пользовать; сервис(экземпляр Service), с которым взаимодействует пользователь; сервис запущенный методом startForeground(); сервис, который выполняет один из методов своего жизненного цикла; BroadcastReceiver который выполняет метод onReceive().

2.Видимый процесс
Процесс, в котором не выполнены условия из пункта №1, но который влияет на то, что пользователь видит на экране. К примеру, вызван метод onPause() активити.

3.Сервисный процесс
Служба запущенная методом startService().

4.Фоновый процесс
Процесс выполняемый в фоновом режиме, который невиден пользователю.

5.Пустой процесс

Отмечу, что в компонентах приложения существует метод onLowMemory(), но полагаться на то, что данный метод будет вызван нельзя, также как нельзя на 100% полагаться на метод onDestroy(), поэтому логику сохранения данных или каких-либо настроек можно осуществить в методе onStop(), который(как уверяют) точно вызывается.

Когда запускается приложение, система создает «главный» поток выполнения для данного приложения, который также называется UI-потоком. Этот поток очень важен, так как именно в нем происходит отрисовка виджетов(кнопочек, списков), обработка событий вашего приложения. Система не создает отдельный поток для каждого экземпляра компонента. Все компоненты, которые запущенны в одном процессе будут созданы в потоке UI. Библиотека пользовательского интерфейса Android не является потоково-безопасной, поэтому необходимо соблюдать два важных правила:

1) Не блокировать поток UI.
2) Не обращаться к компонентам пользовательского интерфейса не из UI-потока.

Теперь, предположим, что у нас возникла задача — загрузить картину в ImageView из сети и тут же ее отобразить. Как мы поступим? По логике: мы создадим отдельный поток, который и сделает всю нашу работу, примерно так:

```
public void onClick(View v) {
   new Thread(new Runnable() {
      public void run() {
         Bitmap b = loadImageFromNetwork("http://example.com/image.png");
         mImageView.setImageBitmap(b);       }
   }).start(); }
```

Выглядит правдоподобно, так как мы вынесли операцию загрузки картинки в отдельный поток. Проблема в том, что мы нарушили правило №2. Исправить эту проблему можно с помощью следующих методов:

Activity.runOnUiThread(Runnable)

```
View.post(Runnable)
View.postDelayed(Runnable, long)
```

К примеру, воспользуемся первым из них:

```
public void onClick(View v) {
   new Thread(new Runnable() {
      public void run() {
         final Bitmap bitmap =
loadImageFromNetwork("http://example.com/image.png");
         mImageView.post(new Runnable() {
            public void run() {
               mImageView.setImageBitmap(bitmap);    }    });
      }
   }).start();  }
```

Теперь реализация потоково-безопасная: сетевая операция выполняется в отдельном потоке, а к ImageView обращаемся из потока UI.
К счастью, данные операции можно объединить с помощью наследования класса Handler и реализации нужной логики, но лучшее решение — наследовать класс AsyncTask.

AsyncTask позволяет выполнить асинхронную работу и делать обновления пользовательского интерфейса.

Для обновления реализуйте метод onPostExecute(), а всю фоновую работу заключите в метод doInBackground(). После того, как вы реализуете свою собственную задачу, необходимо ее запустить методом execute().

Приводим пример AsyncTask, в котором реализована задача загрузки и отображения картинки(вариант с аннотациями и отклонением от применения стандартного протокола диалогов):

```
@EActivity(R.layout.main)
public class DownloadImageActivity extends Activity {
        ProgressDialog progress;

        @ViewById
        ImageView image;

        @ViewById
        Button runButt;

        String pictUrl = "http://www.bigfoto.com/sites/main/churfirsten_switzerland-xxx.JPG";

        @BeforeCreate
        void getProgressDialog(){
           progress = new ProgressDialog(this);
           progress.setMessage("Loading...");          }

        @Click(R.id.runButt)
        void runClick(){
                new DownloadImageTask().execute(pictUrl);          }

        class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
                @Override
                protected Bitmap doInBackground(String... params) {
                   publishProgress(new Void[]{}); //or null
```

```
            String url = "";
            if( params.length > 0 ){
                url = params[0];            }

            InputStream input = null;
           try {
           URL urlConn = new URL(url);
                input = urlConn.openStream();            }
            catch (MalformedURLException e) {
            e.printStackTrace();                }
          catch (IOException e) {
            e.printStackTrace();                }

            return BitmapFactory.decodeStream(input);          }

            @Override
            protected void onProgressUpdate(Void... values) {
                super.onProgressUpdate(values);
                progress.show();            }

            @Override
            protected void onPostExecute(Bitmap result) {
                super.onPostExecute(result);
                progress.dismiss();
                image.setImageBitmap(result);            }        } }
```

А теперь рассмотрим самый правильный вариант с точки зрения работы с диалогами:

```
public class DownloadImageActivity extends Activity {
        ImageView image;
      Button runButt;

        final                static                String                PICT_URL                =
"http://www.bigfoto.com/sites/main/churfirsten_switzerland-xxx.JPG";
        final int PROGRESS_DLG_ID = 666;
        final static String DEBUG_TAG = "+++ImageDownloader+++";

        @Override
        public void onCreate(Bundle savedInstanceState){
           super.onCreate(savedInstanceState);
           setContentView(R.layout.main);
                loadViews();        }

        private void loadViews(){
                runButt = (Button)findViewById(R.id.runButt);
           image =(ImageView)findViewById(R.id.image);        }

        @Override
        protected Dialog onCreateDialog(int dialogId){
                ProgressDialog progress = null;
```

```
        switch (dialogId) {
        case PROGRESS_DLG_ID:
                progress = new ProgressDialog(this);
              progress.setMessage("Loading...");

                break;
        }
        return progress;
}

public void runButtonHandler(View button){
        if(button.getId() == R.id.runButt)
           new DownloadImageTask().execute(PICT_URL);
}

class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
        @Override
        protected Bitmap doInBackground(String... params) {
                publishProgress(new Void[]{});

           String url = "";
           if( params.length > 0 ){
                url = params[0];
           }

           InputStream input = null;

           try {
                URL urlConn = new URL(url);
              input = urlConn.openStream();
           }
      catch (MalformedURLException e) {
           Log.d(DEBUG_TAG,"Oops, Something wrong with URL...");
           e.printStackTrace();
                }
      catch (IOException e) {
           Log.d(DEBUG_TAG,"Oops, Something wrong with inpur stream...");
                        e.printStackTrace();
                }

      return BitmapFactory.decodeStream(input);
           }

           @Override
           protected void onProgressUpdate(Void... values) {
                super.onProgressUpdate(values);
                showDialog(PROGRESS_DLG_ID);
           }

           @Override
           protected void onPostExecute(Bitmap result) {
                super.onPostExecute(result);
```

```
                dismissDialog(PROGRESS_DLG_ID);
                image.setImageBitmap(result);
            }
        }
}
```

Когда стало побольше, но лучше использовать стандартный протокол работы с диалогами. Также я убрал все аннотации, чтобы было проще попробовать данный код. Не забудьте добавить в свою разметку кнопочки атрибут с указанным значением: android:onClick=«runButtonHandler» в оффициальном документе также, как и в моем случае, не используется preExecute(), но если вам понадобится выполнить какие-то действия с вашим пользовательским интерфейсом до начала выполнения задачи, то смело используйте данный метод. Параметры передаваемые в AsyncTask:

1. Параметры(в нашем случае адрес URL).
2. Прогресс (единицы задающие ход изменения задачи). В нашем случае не используется.
3. Результат выполнения задачи(в нашем случае объект Bitmap)

Код довольно прост: всю фоновую работу мы выполняем в методе doInBackGround(), вызывая метод publishProgress(), чтобы во время загрузки картинки крутился наш ProgressDialog при вызове метода onProgressUpdate(). После выполнения фоновой работы вызывается метод onPostExecute() в который передается результат работы метода doInBackGround(), в нашем случае это объект Bitmap и тут же мы его загружаем в ImageView.

Отмечу пару важных моментов, которые нужно учитывать:

1) Метод doInBackGround() выполняется в фоновом потоке, потому доступа к потоку UI внутри данного метода нет.
2) Методы onPostExecute() и onProgressUpdate() выполняются в потоке UI, потому мы можем смело обращаться к нашим компонентам UI.

4.2. События и процессы в приложениях Android (Activities, Fragments, Intents)

Событие – это взаимодействие пользователя с контентом, которое можно отслеживать независимо от просмотров страниц или экранов. Загрузки, клики по мобильным объявлениям, использование гаджетов, элементов Flash и AJAX, воспроизведение видео – все эти действия можно отслеживать в качестве *событий*.

Чтобы настроить отслеживание событий, потребуется время, однако мы настоятельно рекомендуем вам это сделать. Это даст вам такую информацию о взаимодействиях с сайтом или приложением, которую трудно (или вовсе невозможно) получить другим способом.

Отслеживание событий доступно не только для веб-сайтов, но и для приложений, однако в последнем случае требуется дополнительная настройка, которую может выполнить только профессиональный веб-разработчик.

Во время настройки отслеживания событий можно определить и связать с отдельными событиями следующие компоненты.

Значения этих компонентов отображаются в отчетах по событиям. Чем точнее вы зададите настройки, тем проще будет интерпретировать отчеты. В отчетах названия этих компонентов используются для названий отдельных событий. Например, событие из категории *Видео* с действием *Воспроизведение* и ярлыком *Смешное* будет отображаться в отчетах как *Видео* + *Воспроизведение* + *Смешное*.

Схожие типы событий на сайте объединяются в категории. Категории лежат в основе отслеживания событий. Именно по ним в первую очередь сортируются события в отчетах. Примерами категорий могут служить *Видео* и *Загрузки*. В зависимости от содержания ресурса они могут быть более узкими или широкими.

Описательное обозначение конкретной категории событий. Для определения действия можно использовать любую текстовую строку, даже с самым подробным описанием. Например, *Воспроизведение* или *Приостановку* можно определить как действия в категории *Видео*. Допускается также использовать более точное определение и создать действие *Видео подходит к концу*. Оно фиксирует момент, когда ползунок воспроизведения достигает отметки в 90% длительности видеоролика.

Дополнительное описательное обозначение, которое можно использовать для дальнейшего структурирования данных. В качестве ярлыков можно использовать любые текстовые строки. Числовая переменная. Можно использовать явное значение, такое как 30, либо переменную, например *downloadTime*.

Отражает количество взаимодействий с категорией событий. Хотя скрытый счет не фигурирует в стандартных отчетах Google Analytics, его можно получить при помощи API.

Когда запускается компонент приложения и приложение не имеет других запущенных компонентов, Android создает новый процесс для приложения с одним потоком исполнения. По умолчанию все компоненты одного приложения запускаются в одном процессе, в потоке называемом «главный». Если компонент приложения запускается и уже существует процесс для данного приложения(какой-то компонент из приложения существует), тогда компонент запущен в этом процессе и использует его поток выполнения. Вы можете изменить данное поведение, задав разные процессы для разных компонентов вашего приложения. Кроме того вы можете добавить потоки в любой процесс.

Задать отдельный процесс для компонента можно с помощью файла манифеста. Каждый тег компонента(activity, service, receiver и provider) поддерживает атрибут android:process. Данный атрибут позволяет задать процесс, в котором будет выполняться компонент. Также вы можете задать процесс в котором будут выполняться компоненты разных приложений. Также данный атрибут поддерживается тегом application, что позволяет задать определенный процесс для всех компонентов приложения.

Активность(Activities) - окно, несущее графический интерфейс пользователя. Окно активности обычно занимает весь экран устройства, однако вполне возможно создавать полупрозрачные или плавающие диалоговые окна. Мобильные приложения обычно являются многооконными, т. е. содержат несколько активностей, по одной на каждое окно. Одна из активностей определяется как "главная", и именно ее пользователь видит при первом запуске приложения.

Каждый экран приложения является наследником класса Activity. Для создания активности необходимо создать класс-наследник класса Activity напрямую или через любого его потомка. В этом классе необходимо реализовать все методы, вызываемые системой для управления жизненным циклом активности. Таких методов семь:

onCreate()	- метод, вызываемый системой при создании активности. В реализации метода необходимо инициализировать основные компоненты активности и в большинстве случаев вызвать метод setContentView() для подключения соответствующего XML-файла компоновки (layoutfile). После метода onCreate() всегда вызывается метод onStart().
onRestart()	- метод, вызываемый системой при необходимости запустить приостановленную активность. После этого метода всегда вызывается метод onStart().
onStart()	- метод, вызываемый системой непосредственно перед тем, как активность станет видимой для пользователя. После этого метода вызывается onResume().
onResume()	- метод, вызываемый системой непосредственно перед тем, как

	активность начнет взаимодействовать с пользователем. После этого метода всегда вызывается onPause().
onPause()	- метод, вызываемый системой при потере активностью фокуса. В этом методе необходимо фиксировать все изменения, которые должны быть сохранены за пределами текущей сессии. После этого метода вызывается onResume(), если активность вернется на передний план, или onStop(), если активность будет скрыта от пользователя.
onStop()	- метод, вызываемый системой, когда активность становится невидимой для пользователя. После этого метода вызывается либо onRestart(), если активность возвращается к взаимодействию с пользователем, либо onDestroy(), если активность уничтожается.
onDestroy()	- метод, вызываемый системой перед уничтожением активности. Этот метод вызывается либо когда активность завершается, либо когда система уничтожает активность, чтобы освободить ресурсы. Можно различать эти два сценария с помощью метода isFinishing(). Это последний вызов, который может принять активность.

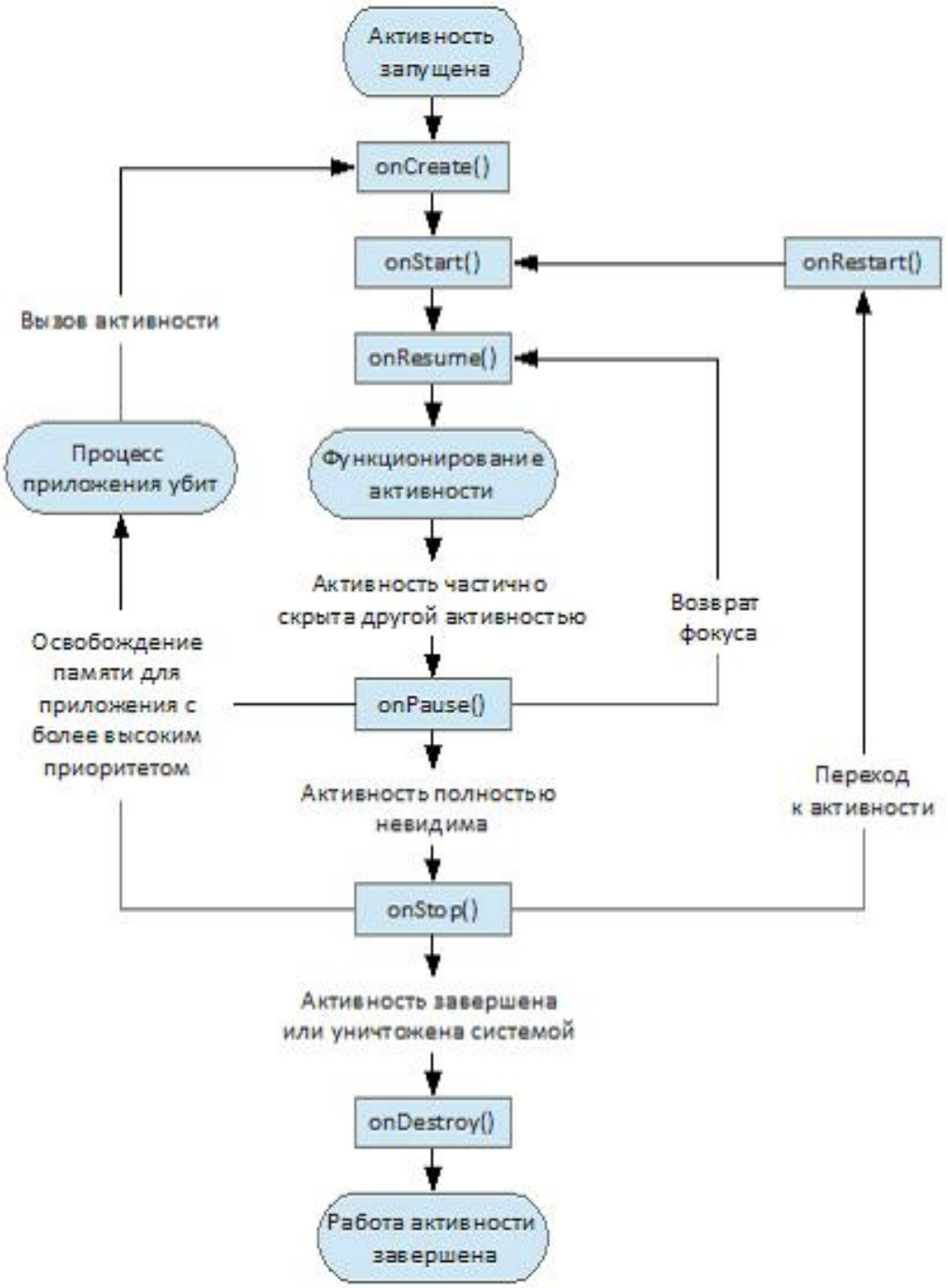

Рис. 1. Жизненный цикл активности

Контрольные вопросы:

1. Какие процессы используются при создании приложении ?
2. Какие процессы используются при управление составляющими приложения ?
3. Какие события используются в приложениях Android ?
4. Какие методы наследует класс Activities ?
5. В каком порядке существует жизненный цикл активности ?