27 февраля 2011 в 16:34

# Простое использование AsyncTask и ProgressDialog в Android





Разработка под Android\*



Практика создания приложений, отзывчивых к действиям пользователя, предполагает, что все тяжелые операции должны исполняться в отдельном потоке, сообщая тем или иным образом пользователю о своем прогрессе.

Android содержит массу способов для организации данного подхода, но одним из самых удобных можно признать использование AsyncTask и ProgressDialog.

Эта парочка превосходно решает задачу, но начинает приносить невыносимую боль, когда количество Activity с такой логикой переваливает за одну, что приводит к повторению управляющего кода, и еще большую боль, когда приложение должно поддерживать смену ориентации экрана.

#### AsyncTask

Для тех, кто не знаком с AsyncTask, поясню, что это специальный абстрактный класс, предоставляющий набор методов для реализации:

- onPreExecute для размещения инициализирующего кода (UI поток)
- doInBackground для размещения тяжелого кода, который будет выполняться в другом потоке
- onProgressUpdate для информирования о прогрессе (UI поток)
- onPostExecute для обработки результата, возвращенного doInBackground (UI поток)

и вспомогательных методов

- isCancelled чтобы узнать не отменил ли кто задачу
- publishProgress для перевода сообщения о прогрессе в UI поток с последующим вызовом onProgressUpdate

Использование упомянутых классов не представляет особого труда, достаточно пары сниппетов, чтобы код заработал как нужно и ProgressDialog начал свое информирование о ходе выполнения задачи. Но, как известно, дьявол кроется в деталях, поэтому стоит только поменять ориетацию экрана, как диалог пропадет, так же как и результат долгой, но невероятно ответственной операции.

Причина заключается в жизненном цикле Activity; смена ориентации экрана трактуется как смена конфигурации, что приводит к пересозданию Activity. Можно, конечно, отключить этот механизм, задав тег android: configChanges="orientation" y Activity и определив собственный код, который, при необходимости, произведет необходимые изменения. Но это будет необоснованное внедрение.

Решением будет создание специального класса по управлению связкой Activity-AsyncTask-ProgressDialog, назовем его AsyncTaskManager.

#### Activity



Итак, в идеале наша Activity должна делать всего пять вещей (код из проекта примера):

• Создавать AsyncTaskManager в методе onCreate

```
mAsyncTaskManager = new AsyncTaskManager(this, this);
```

• Делегировать AsyncTaskManager восстановление задачи из состояния

 $\verb|mAsyncTaskManager.handleRetainedTask(getLastNonConfigurationInstance())|;\\$ 

• Создавать конкретную задачу и отдавать ее в управление AsyncTaskManager'a

mAsyncTaskManager.setupTask(new Task(getResources()));

• Делегировать AsyncTaskManager сохранение задачи в состояние

```
return mAsyncTaskManager.retainTask();
```

• Обрабатывать асинхронное завершение задачи

Для последнего пункта и уменьшения связанности между Activity и AsyncTaskManager можно создать интерфейс для уведомления о завершении и реализовать его:

```
public interface OnTaskCompleteListener {
   void onTaskComplete(Task task);
```

В параметре метода будет передаваться задача, выполнение которой было завершено.

#### AsyncTaskManager

AsyncTaskManager должен отвечать за корректную работу всех компонентов, что сводится к списку следующих задач:

- Создавать диалог при инициализации
- При получении задачи в управление запускать ее
- Отображать состояние задачи в диалоге
- Отсоединяться от задачи после сохранения ее в состояние и присоединяться заново при восстановлении
- Отменять задачу при отмене диалога
- Закрывать диалог при завершении задачи
- Сообщать Activity о завершении либо отмене задачи

Для общения с задачей достаточно следующего интерфейса, который следует реализовать и передать в задачу:

```
public interface IProgressTracker {
    void onProgress(String message);
    void onComplete();
}
```

#### Реализация:

```
@Override
public void onProgress(String message) {
    if (!mProgressDialog.isShowing()) {
        mProgressDialog.show();
    }
    mProgressDialog.setMessage(message);
}

@Override
public void onComplete() {
    mProgressDialog.dismiss();
    mAsyncTask.setProgressTracker(null);
    mTaskCompleteListener.onTaskComplete(mAsyncTask);
    mAsyncTask = null;
}
```

#### Присоединение к задаче:

```
mAsyncTask.setProgressTracker(this);
```

#### Отсоединение от задачи:

```
mAsyncTask.setProgressTracker(null);
mAsyncTask = null;
```

#### Отмена диалога:

```
@Override
public void onCancel(DialogInterface dialog) {
    mAsyncTask.setProgressTracker(null);
    mAsyncTask.cancel(true);
    mTaskCompleteListener.onTaskComplete(mAsyncTask);
    mAsyncTask = null;
}
```

AsyncTaskManager выполняет роль своеобразного ключа, который подключает и отключает работающую задачу к возможно пересозданному экземпляру Activity. Кроме того, он берет на себя и скрывает логику работы с ProgressDialog.

#### AsyncTask

Для задачи, помимо реализации основных методов, требуется реализация метода, который поможет соединять/разъединять ее с AsyncTaskManager′ом:

```
public void setProgressTracker(IProgressTracker progressTracker) {
    mProgressTracker = progressTracker;
    if (mProgressTracker != null) {
        mProgressTracker.onProgress(mProgressMessage);
        if (mResult != null) {
            mProgressTracker.onComplete();
        }
    }
}
```

Как видно из приведенного кода, задача сохраняет вычисленный результат и последнее сообщение о прогрессе, и, в зависимости от состояния, вызывает тот или иной метод трекера (AsyncTaskManager'a).

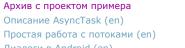
Таким образом, даже если задача завершится до пересоздания Activity, та получит уведомление о завершении задачи.

#### Результат

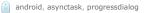
Теперь можно безбоязненно крутить телефон в руках - все задачи будут обработаны корректно.

Использование такого менеджера существенно снижает количество кода в Activity и позволяет переиспользовать данную функциональность в проекте. Данный подход я разработал и успешно применил в своем недавнем приложении.

#### Ссылки



Диалоги в Android (en)





### Похожие публикации

Android L, Nexus 5, Google Search и все-все-все вчера в 18:23

Android NDK: работа с OpenSL ES 8 сентября в 10:57

Новый Evernote 6 для Android: редизайн, веб-клиппер и многое другое 5 сентября в 12:51

Читаем XLSX на Android при помощи Apache POI 3 сентября в 09:49

Skype анонсировал прекращение поддержки Android 2.2 и предыдущих версий 27 августа в 20:43

Фрагментация Android практически перестала быть проблемой? 26 августа в 16:15

Контур.Эльба под Android. Записки разработчика 26 августа в 12:58

Запуск objective-с кода на Android устройствах 25 августа в 10:13

Простой USSD-запрос в Android 4.0+ 25 августа в 07:21

Заметки о ProgressDialog или как правильно показать прогресс выполнения 12 мая 2011 в 09:37

### **▼** Комментарии (5) отслеживать новые: □ в почте □ в трекере



**Tibr,** 27 февраля 2011 в 17:29 # 🏫



Неплохая статья. Я вот только одного не понял — зачем нам 2 интерфейса, не логичнее было бы создать один, который отвечал бы за полное обслуживание задачи?





Спасибо. Да, действительно, можно слегка модифицировать IProgressTracker (добавив Task в onComplete) и использовать его в связках Actvity-Manager и Manager-Task. В этом случае Activity тоже сможет следить за прогрессом, если ей это нужно. Но в моей задаче это было не нужно, поэтому я создал второй, более строгий интерфейс. Двух связок не избежать, если стоит задача по-максимуму вынести всю логику во вспомогательный класс.



🚹 darkolorin, 27 февраля 2011 в 18:04 # 🦙



Что ни день то статья про разработку под Android! Не может не радовать!



leOpard, 27 февраля 2011 в 19:03 # 🦙



A есть еще getLastNonConfigurationInstance и onRetainNonConfigurationInstance. Этого достаточно, чтобы восстановить диалог с уже запущенным AsyncTask





Первый метод используется в строке «mAsyncTaskManager.handleRetainedTask(getLastNonConfigurationInstance());» Во втором я вызываю «return mAsyncTaskManager.retainTask();». Это как раз те вызовы, которые делегируются. Идея в том, чтобы вынести весь код поддержки в AsyncTaskManager.

## **Brainstorage**

Специалист по автоматизированному тестированию

РНР программист (bitrix)

Дизайнер

Разработчик PHP + JS

HTML верстальщик

Android Developer, Zvoog

Python Data Miner

С++ программист под задачу

UI/UX дизайнер

Backend-разработчик (Senior)

все вакансии

#### ФРИЛАНСИМ

Торговый портал

Верстка HTML письма

Дизайн полиграфии

Разработать flash игру

Дизайн сайта интернет-магазина

Провести анализ РК в Яндекс. Директ

Инфографика по готовому шаблону (перевод с английского)

Разобрать и доработать проект Android+JBoos Server

Тексты на сайт частной школы

Сделать шапку для сайта

все заказы



ФРИЛАНСИМ АВТОКАДАБРА Заказы для фрилансеров Уютная и дружелюбная