Лабораторная работа № 7

Обмен сообщениями из потоков в Android.

Общие сведения.

При запуске приложения Android главным потоком исполнения команд является поток пользовательского интерфейса **UI thread**.

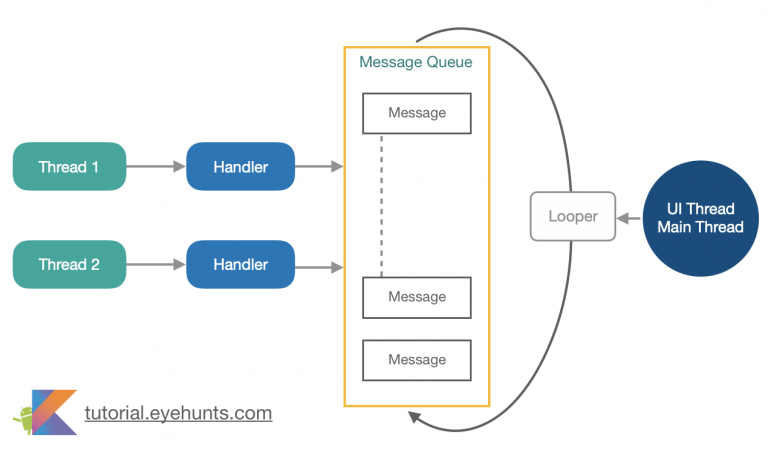
Для избежания блокировки UI thread нагруженные операции выполняются в отдельных потоках.

Такие потоки могут создаваться как в самом UI потоке, так и в других классах, методы которых вызываются в UI thread. Из дочерних потоков нельзя напрямую обращаться компонентам View. Поэтому существуют специальные механизмы организации такого взаимодействия:

– через специальный метод  **post()** элементов пользовательского интерфейса;

– через обмен сообщениями.

Для обмена сообщениями между потоками существует ряд взаимосвязанных компонентов: **Handler, MessageQueue, Looper, Message**.

****

Класс  **MessageQueue** - очередь сообщений, который оперирует объектами **Message**. Каждое сообщение – специального вида объект, который может содержать «полезную нагрузку». В Сообщениях можно передавать примитивы или другие объекты.

Класс **Handler** реализует методы отправки и обработки сообщений.

Класс **Looper** представляет собой бесконечный цикл выборки сообщений, выполняющийся в потоке, к которому он добавлен. Каждый экземпляр **Looper**  привязывается к конкретному потоку и экземпляру **Handler**. Таким образом, образуется цепочка передачи сообщения. Сообщение созданное в одном потоке (№1), используя экземпляр **Handler** другого потока (№2) отправляет через него сообщение, которое попадает в очередь.

В потоке №2 экземпляр **Looper** выбирает сообщения из очереди и они попадают в обработчик сообщений **Handler**-а, который связан с этим потоком.

Так как **Handler** создан в самом потоке №2, то они имеет доступ ко всем его полям и объектам. И, если поток №2 является UI потоком, то из методов **Handler**-а можно обращаться к UI элементам.

Для использования подобного механизма в принимающем потоке (например, UI) нужно получить экземпляр класса **Looper**:

**final** Looper **looper** = Looper.*getMainLooper*();

getMainLooper() возвращает экземпляр **Looper**, связанный с UI-потоком.

Далее необходимо создать экземпляр **Handler** в котором переопределить метод **handleMessage**, отвечающий за обработку принимаемых сообщений. Код метода **handleMessage** может содержать обращение к элементам пользовательского интерфейса:

**final** Handler **handler** = **new** Handler(**looper**) {  
 @RequiresApi(api = Build.VERSION\_CODES.***LOLLIPOP***)  
 @Override  
 **public void** handleMessage(Message msg) {  
 **if** (msg.**sendingUid** == 1) {  
 **for** (String i : (ArrayList<String>) msg.**obj**) {  
 **StringArray**.add(i);  
 }  
 **textList**.setAdapter(**TextAdapter**);  
 **btn\_StartExternal**.setEnabled(**true**);  
 }  
  
 **if** (msg.**sendingUid** == 2) {  
 **btn\_StartInternal**.setEnabled((Boolean) msg.**obj**);  
 }  
 }  
};

В данном примере реализовано два разных варианта обработки сообщений, в зависимости от того, каким потоком сообщение было отправлено. Идентификатор отправителя можно установить в параметре **sendingUid** сообщения.

В первом случае **sendingUid = 1 –** сообщение отправлено потоком внешнего класса и содержит список строк, который необходимо отобразить на **textList**. Такой вариант может возникнуть, например, если внешний класс запрашивает из БД информацию и возвращает ее для отображения пользователю. В этом случае может быть возвращен список объектов.

Во втором случае **sendingUid = 2** – сообщение отправлено дочерним потоком, созданным в UI thread. И задачей обработчика является сделать активной кнопку, которая была деактивирована на время работы дочернего потока.

Обработчики событий нажатия кнопок:

**public void** onClick(View v) {  
 **switch** (v.getId()) {  
 **case** R.id.***btn\_Start1***:  
 **btn\_StartExternal**.setEnabled(**false**);  
 **new** ThreadTask(**handler**).doSomething();  
 **break**;  
  
 **case** R.id.***btn\_Start2***:  
 **btn\_StartInternal**.setEnabled(**false**);  
 **new** Thread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **try** {  
 Thread.*sleep*(3000);  
 *// Имитируем высокую нагрузку* } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 **if** (Build.VERSION.***SDK\_INT*** >= Build.VERSION\_CODES.***LOLLIPOP***) {  
 **message**.**sendingUid** = 2;  
 }  
 **message**.**obj** = **true**;  
 **handler**.sendMessage(**message**);  
 }  
 }).start();  
 }  
 }  
}

При нажатии на кнопку **btn\_Start1** эта кнопка делается неактивной для исключения запуска многих потоков при многократном нажатии командой

**btn\_StartExternal**.setEnabled(**false**);

После чего вызывается метод внешнего класса (в конструктор передается экземпляр **Handler**), который, например, может делать и обрабатывать запрос к БД.

Кодд класса содержит метод, в котором запускается поток, выполняющий сложный запрос, вычисления и т.п.

**public class** ThreadTask {  
  
 Handler **thr\_handler**;  
 **final** Message **message** = Message.*obtain*();  
  
 ThreadTask(Handler main\_handler){  
 **this**.**thr\_handler** = main\_handler;  
 }  
  
 **public void** doSomething() {  
 **new** Thread(**new** Runnable() {  
 @RequiresApi(api = Build.VERSION\_CODES.***LOLLIPOP***)  
 @Override  
 **public void** run() {  
 **try** {  
 Thread.*sleep*(3000);  
 *// Имитируем высокую нагрузку* } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 ArrayList<String> StringArray = **new** ArrayList<String>();  
 **for**(**int** i=0; i<5; i++){  
 StringArray.add(i + **" string from ThreadTask"**);  
 }  
  
 **message**.**sendingUid** = 1;  
 **message**.**obj** = StringArray;  
 **thr\_handler**.sendMessage(**message**);  
 }  
 }).start();  
 }  
}

Для отправки из этого класса сообщения в UI thread его необходимо создать

**final** Message **message** = Message.*obtain*();

затем указать ID отправителя и тело сообщения

**message**.**sendingUid** = 1;

**message**.**obj** = StringArray;

после чего сообщение может быть отправлено

**thr\_handler**.sendMessage(**message**);

Для отправки вызывается метод sendMessage **Handler**-а, который был передан в конструктор класса UI потоком.

Отправленное таким образом сообщение будет обработано методом handleMessage в UI потоке.

При нажатии на кнопку **btn\_Start2** эта кнопка также делается неактивной и создается дочерний поток, в котором происходит выполнение какой-либо работы. Для разблокирования кнопки из потока отправляется аналогичным образом созданное сообщение

**message**.**sendingUid** = 2;  
**message**.**obj** = **true**;  
**handler**.sendMessage(**message**);

которое попадая в обработчик сообщений делает кнопку активной

**if** (msg.**sendingUid** == 2) {  
 **btn\_StartInternal**.setEnabled((Boolean) msg.**obj**);  
}

Пример проекта:

<https://github.com/kafvtpnz/Android2022/tree/L7Thread_Looper_simple>

Ход выполнения проекта.

1 Откройте проект, созданный в лабораторной работе № 6.

2 Измените способ работы с БД таким образом, чтобы запросы к БД выполнялись в отдельных потоках, создаваемых во внешнем, по отношению к DataBaseHandler и Activity классе.