

Лекция #B-12

*Многопоточность и GUI. Метод runOnUiThread.*

Содержание:

1. [Доступ к GUI из контекста *Activity*](#_bookmark0).
2. [Доступ к GUI из других классов, не обладающих контекстом](#_bookmark1)

[активности](#_bookmark1).



1. Доступ к GUI из контекста *Activity*.

Изменять какие-либо значения у графических элементов можно только лишь в т.н. "Главном Графическом потоке". Доступ к этому потоку может осуществляться только из методов текущей

активности. Это значит, что устанавливать и изменять графику можно только в методе **onCreate()** и всех параллельных ему методах. Правильно это звучит так - "доступ к графике возможен только лишь из

контекста текущей активности".

Рассмотрим примеры. Пускай во всей активности будет только одна метка *TextView* и одна кнопка *Button* (которые предварительно разместились в компоновке *R.layout.run\_on\_ui\_thread\_review\_layout.xml*). Задачей будет программирование обыкновенного счетчика

- метка отображает количество нажатий на кнопку. Здесь, используя привычную стратегию в *Swing*, можно было бы обойтись слушателем в виде анонимного класса:

public class MainActivity extends Activity{ int counter;

TextView tv; Button bCounter;

@Override

protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);

setContentView(R.layout.*run\_on\_ui\_thread\_review\_layout*); tv = (TextView) findViewById(R.id.*tv*);

bCounter = (Button) findViewById(R.id.*bCounter*);

bCounter.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View v) {

**tv.setText("" + (counter++));**

}

});

}

}

Здесь процесс изменения графического элемента - текстовой метки полностью лежит в контексте активности - внутри метода *onCreate().* Однако, представим себе ситуацию, что необходимо по

нажатию на кнопку сделать автоматический счетчик от 1-го до 10-ти, который изменял бы значение числа с определенным интервалом, например в полсекунды.

Для такой ситуации необходимо выделить новый поток и код программы определенным образом изменится:

public class MainActivity extends Activity{ int counter;

TextView tv; Button bCounter;

@Override

protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);

setContentView(R.layout.*run\_on\_ui\_thread\_review\_layout*);

tv = (TextView) findViewById(R.id.*tv*);

bCounter = (Button) findViewById(R.id.*bCounter*); bCounter.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View v) {

**Runnable r = new Runnable() {**

**@Override**

**public void run() {**

**while(counter <= 10){**

**tv.setText("" + (counter++));**

**Thread.*sleep*(500);**

**}**

**}**

**};**

**Thread t = new Thread(r); t.start();**

}

});

}

}

Таким образом, поток управляет распределением времени. Но, запустив данный код на выполнение, в момент нажатия на кнопку, а точнее - *при самом первом обращении к переменной tv*, произойдет

выброс из приложения и Logcat любезно предоставит пояснения случившегося:

android.view.ViewRootImpl$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.

Этот комментарий как раз и говорит о том, что доступ к графическим элементам может

производиться только из контекста активности - из главного графического потока. В случае примера, рассмотренного выше, доступ к текстовой метке производится из нового потока! Это и спровоцировало аварийный выброс из приложения.

Следует запомнить это сообщение на будущее – оно всегда будет свидетельствовать о желании изменить графические компоненты не из основного GUI-потока.

На помощь придет метод класса *Activity runOnUiThread***()**.

Этот метод является аналогом метода конструктора *Thread()* для потока.

Он также, как и конструктор принимает в себя объект типа *Runnable*. Напоминаю, что объекты такого типа являются своего рода динамическими процессами, которыми управляет (запускает,

устанавливает) объект типа *Thread*. Посмотрите на пример выше - переменная *r* является процессом, который состоит из набора команд (изменение текста и приостановка потока). Переменная *t* только лишь управляет этим процессом - создает новый поток и запускает процесс на выполнение в этом потоке.



Принцип работы метода *runOnUiThread()* схожий. *Этот метод только лишь передает процесс*

*на выполнение в главный графический поток*. И больше ничего. Поэтому, как будет видно дальше, для метода *runOnUiThread()* не требуется вызывать что-либо похожее на *.start(),* как у потока *Thread*. Простым языком этот метод только вклинивает принимаемый *Runnable*-процесс в GUI.

Основываясь на изложенном, необходимо изменить код так, чтобы передать процесс доступа и

изменения текстовой метки из нового потока в GUI. Для этого сначала необходимо создать этот процесс и затем передать его в метод *runOnUiThread()* :

public class MainActivity extends Activity{ int counter;

TextView tv; Button bConter;

@Override

protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);

setContentView(R.layout.run\_on\_ui\_thread\_review\_layout); tv = (TextView) findViewById(R.id.tv);

bConter = (Button) findViewById(R.id.bCounter); bConter.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View v) { Runnable r = new Runnable() {

@Override

public void run() {

while(counter <= 10){

**Runnable process = new Runnable() {**

**@Override**

**public void run() {**

**tv.setText("" + (counter++));**

**}**

**};**

**runOnUiThread(process);**

Thread.sleep(500);

}

}

};

Thread t = new Thread(r); t.start();

}

});

}

}

Именно этот код и будет результатом поставленной задачи создания автоматического счетчика.



1. Доступ к GUI из других классов, не обладающих контекстом активности.

Однако, существует и другая проблема. В рассмотренном выше примере использование метода *runOnUiThread()* происходило непосредственно внутри самой активности, поэтому доступ к этому методу был очевиден. Существует очень много ситуаций, да и на практике происходит зачастую именно так, что приходится изменять графические объекты не из класса *Activity*, а из других классов, которые не относятся к активности *Activity*.

Рассмотрим эту ситуацию подробнее.

Добавим, не удаляя существующий код, новый функционал - три текстовых поля, в которые будут вводиться имя, фамилия и возраст человека, кнопка "Create person", которая будет создавать

инкапсулирующий объект на основании введенных данных и вторая кнопка "Show Person Info", выводящая информацию об этом человеке в уже существующую текстовую метку:

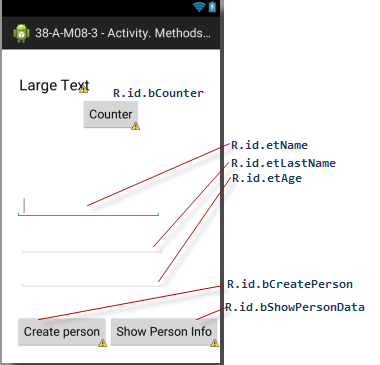


Рис. 1 компоновка рассматриваемой задачи

# res/layout/run\_on\_ui\_thread\_review\_layout.xml

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<RelativeLayout xmlns:android=*"*[*http://schemas.android.com/apk/res/android*](http://schemas.android.com/apk/res/android)*"* android:layout\_width=*"match\_parent"* android:layout\_height=*"match\_parent"* >

<TextView

android:id=*"@+id/tv"* android:layout\_width=*"wrap\_content"* android:layout\_height=*"wrap\_content"* android:layout\_alignParentLeft=*"true"* android:layout\_alignParentTop=*"true"* android:layout\_marginLeft=*"25dp"* android:layout\_marginTop=*"39dp"* android:text=*"Large Text"*

android:textAppearance=*"?android:attr/textAppearanceLarge"* />

<Button

android:id=*"@+id/bCounter"* android:layout\_width=*"wrap\_content"* android:layout\_height=*"wrap\_content"* android:layout\_below=*"@+id/textView1"* android:layout\_centerHorizontal=*"true"* android:layout\_marginTop=*"71dp"* android:text=*"Counter"* />

<Button

android:id=*"@+id/bShowPersonInfo"* android:layout\_width=*"wrap\_content"* android:layout\_height=*"wrap\_content"* android:layout\_alignBottom=*"@+id/bCreatePerson"* android:layout\_alignParentRight=*"true"* android:text=*"Show Person Info"* />

<EditText

android:id=*"@+id/etName"* android:layout\_width=*"wrap\_content"* android:layout\_height=*"wrap\_content"* android:layout\_alignParentLeft=*"true"* android:layout\_centerVertical=*"true"* android:layout\_marginLeft=*"20dp"* android:ems=*"10"* android:inputType=*"textPersonName"* >

<requestFocus />

</EditText>

<Button

android:id=*"@+id/bCreatePerson"* android:layout\_width=*"wrap\_content"* android:layout\_height=*"wrap\_content"* android:layout\_alignLeft=*"@+id/editText1"* android:layout\_alignParentBottom=*"true"* android:layout\_marginBottom=*"22dp"* android:text=*"Create person"* />

<EditText

android:id=*"@+id/etLastName"* android:layout\_width=*"wrap\_content"* android:layout\_height=*"wrap\_content"* android:layout\_alignLeft=*"@+id/tv"* android:layout\_below=*"@+id/etName"* android:layout\_marginTop=*"16dp"* android:ems=*"10"* android:inputType=*"textPersonName"* />

<EditText

android:id=*"@+id/etAge"* android:layout\_width=*"wrap\_content"* android:layout\_height=*"wrap\_content"* android:layout\_alignLeft=*"@+id/etLastName"* android:layout\_below=*"@+id/etLastName"* android:layout\_marginTop=*"20dp"* android:ems=*"10"*

android:inputType=*"phone"* />

</RelativeLayout>

Для динамического кода нам понадобится инкапсулирующий класс *Person* и поле этого типа в классе *MainActivity*:

# 

# MainActivity.java

public class MainActivity extends Activity implements OnClickListener{ int counter;

TextView tv; Button bConter; EditText etName;

EditText etLastName; EditText etAge; Button bCreatePerson;

Button bShowPersonData;

**Person person;**

@Override

protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);

setContentView(R.layout.run\_on\_ui\_thread\_review\_layout); tv = (TextView) findViewById(R.id.tv);

bConter = (Button) findViewById(R.id.bCounter); bConter.setOnClickListener(new OnClickListener() {

. . . . .

});

etName = (EditText) findViewById(R.id.etName); etLastName = (EditText) findViewById(R.id.etLastName); etAge = (EditText) findViewById(R.id.etAge);

bCreatePerson = (Button) findViewById(R.id.bCreatePerson); bCreatePerson.setOnClickListener(this);

bShowPersonData = (Button) findViewById(R.id.bShowPersonInfo); bShowPersonData.setOnClickListener(this);

}

@Override

public void onClick(View v) {

switch (v.getId()) { case R.id.*bCreatePerson*:

person = new Person(etName.getText().toString(),

etLastName.getText().toString(), Integer.*parseInt*(etAge.getText().toString()));

break;

case R.id.*bShowPersonInfo*: person.printInfo();

break;

}

}

}

**class Person{**

**String name; String lastName; int age;**

**public Person(String name, String lastName, int age) { this.name = name;**

**this.lastName = lastName; this.age = age;**

**}**

**public void printInfo(){**

**}**

**}**



Логика приложения будет следующей - при нажатии на кнопку "Create person" произойдет инициализация перемен- ной *person* и создание объекта. А при нажатии на кнопку "Show Person Info" произойдет отображение информации о че- ловеке с анимацией - каждая следующая буква выводимого текста будет отображаться с небольшой задержкой (20 мсек).

Эта задача очень похожа по механизму на счетчик за исключением одной особенности - в счетчике поток управления *Thread* создавался внутри класса *Activity*, где был непосредственный доступ к методу *runOnUiThread()*. В этом случае управляющий поток *Thread* будет размещаться в другом - неActivity классе - значит, доступа к методу *runOnUiThread()* не будет. Для решения этой проблемы можно пойти несколькими путями:

1. *создание статических констант типа Activity в классе MainActivity и инициализация их в методе onCreate()*. Также необходимо сделать статическими полями те, которые будут использоваться другим классом - поле метки *tv* :

public class MainActivity extends Activity implements OnClickListener{

**public static Activity *activity*;**

int counter;

**public static TextView *tv*;**

Button bConter; EditText etName; EditText etLastName; EditText etAge; Button bCreatePerson;

Button bShowPersonData;

Person person;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

**activity = this;**

setContentView(R.layout.run\_on\_ui\_thread\_review\_layout); tv = (TextView) findViewById(R.id.tv);

bConter = (Button) findViewById(R.id.bCounter); bConter.setOnClickListener(new OnClickListener() {

. . . . .

});

etName = (EditText) findViewById(R.id.etName); etLastName = (EditText) findViewById(R.id.etLastName); etAge = (EditText) findViewById(R.id.etAge);

bCreatePerson = (Button) findViewById(R.id.bCreatePerson); bCreatePerson.setOnClickListener(this);

bShowPersonData = (Button) findViewById(R.id.bShowPersonInfo); bShowPersonData.setOnClickListener(this);

}

@Override

public void onClick(View v) {

switch (v.getId()) { case R.id.*bCreatePerson*:

person = new Person(etName.getText().toString(),

etLastName.getText().toString(), Integer.*parseInt*(etAge.getText().toString()));

break;

case R.id.*bShowPersonInfo*: person.printInfo();

break;

}

}

}



class Person{

String name; String lastName; int age;

public Person(String name, String lastName, int age) { this.name = name;

this.lastName = lastName; this.age = age;

}

public void printInfo(){

StringBuilder sb = new StringBuilder(); sb.append("Name : " + name); sb.append("\n");

sb.append("Last name : " + lastName); sb.append("\n");

sb.append("Age : " + age);

}

}

Таким об

|  |  |  |
| --- | --- | --- |
| MainActivity.*tv*.setText(sb); |  | |
|  | |
| разом текст отобразится мгновенно. Не состоит труда заменить метод *printInfo()* так, чтобы через сек к текстовой метке добавлялся новый символ (необходимо ввести дополнительно статическое  *unter*):  son{ name; lastName; e;  **int *printCounter*;**  Person(String name, String lastName, int age) { this.name = name;  this.lastName = lastName; this.age = age;  void printInfo(){  final StringBuilder sb = new StringBuilder(); sb.append("Name : " + name);  sb.append("\n");  sb.append("Last name : " + lastName); sb.append("\n");  sb.append("Age : " + age); | | |
| MainActivity.*tv*.setText(""); Runnable r = new Runnable() {  @Override  public void run() {  while(MainActivity.*tv*.length() < sb.length()){  **Runnable rInfo = new Runnable() {**  **@Override**  **public void run() {**  **MainActivity.*tv*.append("" + sb.charAt(*printCounter*++));**  **}**  **};**  **MainActivity.*activity*.runOnUiThread(rInfo);**  Thread.*sleep*(50);  }  }  }; | |  |
|  |

задержку в 20 м поле *printCo*

class Per

String String int ag

**static**

public

}

public



Thread t = new Thread(r); t.start();

}

}

Здесь опять же, как и в случае с обычным счетчиком - создается процесс *Runnable r*, который будет выполнять главные действия:

* циклическое создание процесса изменения текста *Runnable rInfo*;
* отправку этого процесса в GUI с помощью метода *runOnUiThread()*;
* приостановление потока *Thread.sleep()*.

Процесс же *Runnable rInfo* состоит только из добавления к текущему тексту нового символа.

1. *Передать ссылки на активность и используемые графические объкты в класс Person*

Т.е. вместе с именем, фамилией и возрастом передать ссылку *this* и *tv* и таким образом наступит возможность обращения не к полям класса *MainActvity*, а к полям своего класса *Person*:

public class MainActivity extends Activity implements OnClickListener{ public static Activity *activity*;

int counter;

public static TextView *tv*; Button bConter;

EditText etName; EditText etLastName; EditText etAge; Button bCreatePerson;

Button bShowPersonData; Person person;

@Override

protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);

. . . . .

}

@Override

public void onClick(View v) {

switch (v.getId()) { case R.id.*bCreatePerson*:

**person = new Person(etName.getText().toString(),**

**etLastName.getText().toString(), Integer.*parseInt*(etAge.getText().toString()), this,**

***tv*);**

break;

case R.id.*bShowPersonInfo*: person.printInfo();

break;

}

}

}



class Person{ String name; String lastName; int age;

**Activity activity; TextView tv;**

**public Person(String name, String lastName, int age, Activity activity, TextView tv) { this.name = name;**

**this.lastName = lastName; this.age = age; this.activity = activity; this.tv = tv;**

**}**

public void printInfo(){

StringBuilder sb = new StringBuilder(); sb.append("Name : " + name); sb.append("\n");

sb.append("Last name : " + lastName); sb.append("\n");

sb.append("Age : " + age);

**tv.setText(sb);**

}

}

Оба этих подхода имеют и полюсы и минусы. Второй способ более удачен с точки зрения принципа инкапсуляции - все действия должны по возможности происходить внутри класса. Однако, в данной

задаче происходил доступ только к одному графическому элементу. А для работы со большим их количеством придется увеличивать количество передаваемых параметров конструктора *Person*, что приведет к усложнению читабельности кода, в свою очередь, первый способ отличается простотой.