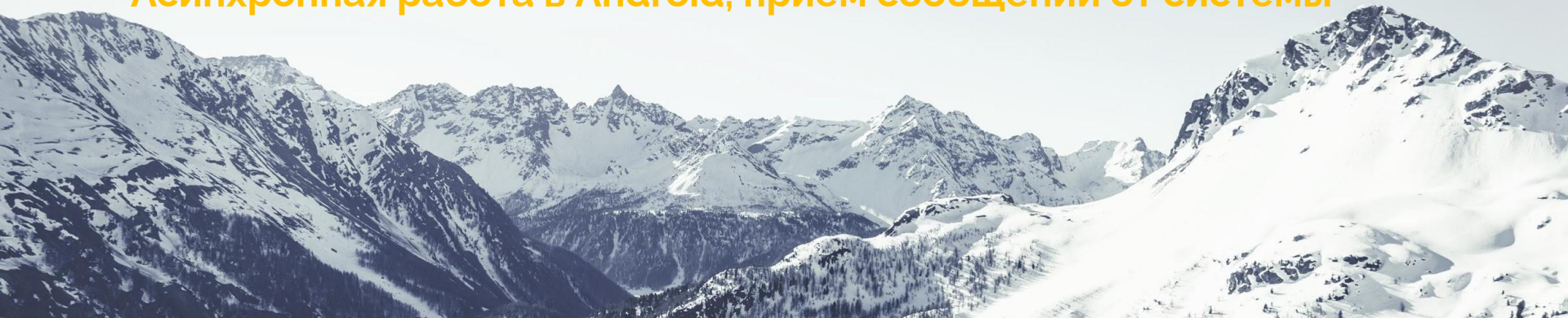
<TeachMeSkills/>

курс

Android разработчик

Асинхронная работа в Android, прием сообщений от системы



Агенда занятия:

Thread, Handler, Main Thread

BroadcastReciever

- Thread, Handler, Main Thread

  BroadcastReciever

Main Thread (UI Thread)

Это основной поток приложения, в котором выполняется отрисовка интерфейса и обработка событий. Все взаимодействия с UI (виджетами, Activity, Fragment) должны происходить в Main Thread.

Почему это важно?

Долгие операции (например, запросы в сеть или работу с БД) нельзя выполнять в Main Thread — это приведёт к «заморозке» интерфейса и ANR (Application Not Responding).

```
if (Looper.getMainLooper().isCurrentThread) {
    // Вы в главном потоке
}
```

Thread — отдельный поток исполнения

Позволяет запускать «тяжёлые» операции вне UI потока. Каждый поток выполняет свою задачу независимо.

Когда использовать: Для простых задач, не требующих взаимодействия между потоками и отмены задач. Когда не нужны дополнительные абстракции (например, Coroutine или Executor).

```
Thread {
    // Код, выполняющийся в отдельном потоке
    val result = doSomeLongTask()
    runOnUiThread {
        // Возвращаемся в главный поток для обновления UI
        textView.text = result
    }
}.start()
```

Handler — коммуникация между потоками

Позволяет отправлять задачи (сообщения или Runnable) из одного потока в другой, чаще всего из фонового в главный.

Как работает: Handler ассоциируется с конкретным Looper'ом (обычно с Looper главного потока).

```
val handler = Handler(Looper.getMainLooper())
Thread {
    // Фоновая работа
    val data = loadData()
    handler.post {
        // Код в главном потоке
        textView.text = data
    }
}.start()
```

Looper

Looper — это объект, который управляет циклом обработки сообщений (event loop) внутри потока. По умолчанию Main Thread (UI-поток) уже содержит свой Looper. Любой поток может получить свой собственный Looper (но только один на поток).

Looper:

- Получает сообщения и Runnable-объекты из очереди (MessageQueue);
- По очереди выполняет их в том же потоке, где был создан Looper.

MessageQueue

MessageQueue — это очередь сообщений, которая хранит задачи (Message и Runnable) для выполнения в определенном потоке.
Каждому Looper соответствует свой MessageQueue.
MessageQueue содержит список всех отложенных и ожидающих исполнения задач для потока.

MessageQueue

Как устроена MessageQueue:

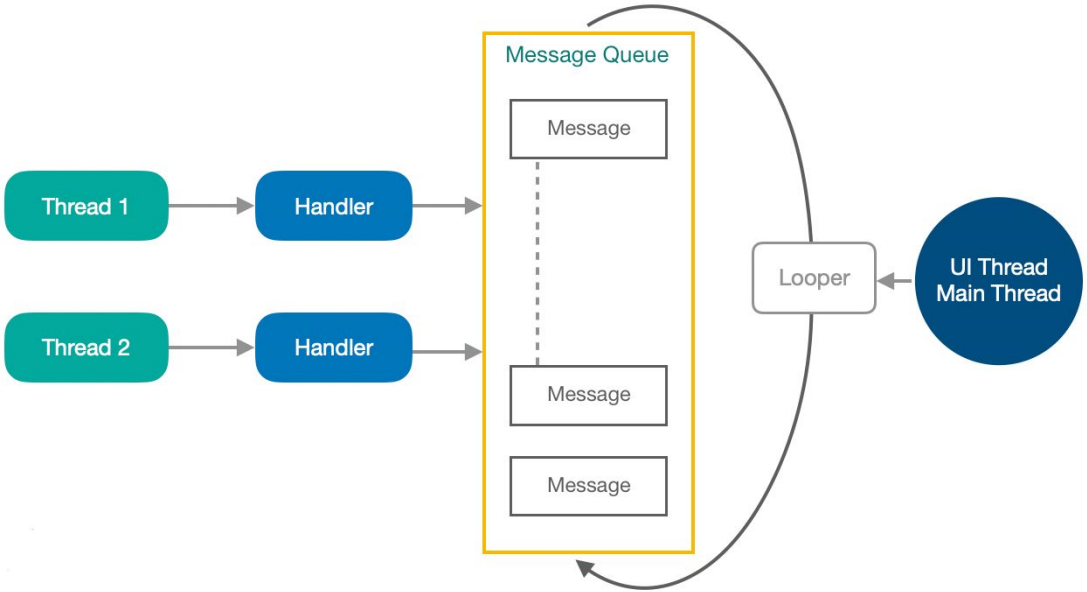
Это обычная очередь (обычно реализована как связный список). В ней лежат объекты типа Message (может содержать данные, ссылку на Handler, время исполнения и пр.). Каждое сообщение имеет метку времени (when), что позволяет реализовать задержки (postDelayed).

Механизм работы

1. Handler создаёт и отправляет Message или Runnable через методы sendMessage(), post() и т.п.
2. Сообщение добавляется в MessageQueue соответствующего Looper’a.
3. Looper работает в цикле: извлекает из MessageQueue очередное сообщение и вызывает для него метод Handler.handleMessage(Message msg), либо выполняет Runnable.
4. После выполнения задачи Looper берёт следующее сообщение из очереди, и так бесконечно, пока очередь не пуста или поток не завершён.

```
// Отправка задачи с задержкой
val handler = Handler(Looper.getMainLooper())
handler.postDelayed({
    // Этот код попадёт в MessageQueue и выполнится через 2 сек в Main Thread
    textView.text = "Привет!"
}, 2000)
```

Механизм работы



Важные детали

MessageQueue — приватная структура, напрямую её почти никогда не используют.
Если очередь пуста, Looper ждёт, не нагружая процессор (оптимизировано под низкое потребление ресурсов).
Если Handler отправил Runnable с задержкой, сообщение просто будет стоять в очереди до нужного времени.
Любые задачи в MessageQueue всегда исполняются в том потоке, которому принадлежит Looper.

Thread, Handler, Main Thread,

- **BroadcastReciever**

BroadcastReceiver

BroadcastReceiver — прием системных и кастомных событий.

BroadcastReceiver — это компонент Android-приложения, который «слушает» широковещательные сообщения (broadcasts) от системы или других приложений. Эти сообщения — это Intent-ы, которые могут передавать информацию о системных событиях (смена сети, загрузка устройства, изменение батареи и т.д.), либо быть отправлены внутри самого приложения.

BroadcastReceiver

Для чего нужен?

- Слежение за изменениями состояния системы (Wi-Fi, батарея, время, Bluetooth, звонки и т.д.);
- Реагирование на собственные (кастомные) события внутри приложения (например, синхронизация, уведомление о завершении задачи);
- Обмен событиями между приложениями (например, sharing).

BroadcastReceiver

Жизненный цикл:

Метод onReceive(Context, Intent) вызывается, когда приходит соответствующее событие. Внутри onReceive нельзя выполнять долгие операции, обработка должна быть быстрой (до 10 секунд). После завершения onReceive объект BroadcastReceiver уничтожается системой.

Регистрация

1. **Статическая** — через манифест (для системных событий: BOOT_COMPLETED, SMS_RECEIVED и пр.).

```
<!-- AndroidManifest.xml -->
<receiver android:name=".MyBroadcastReceiver">
    <intent-filter>
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
    </intent-filter>
</receiver>
```

Регистрация

2. **Динамическая** — в коде (для событий, актуальных только при активном компоненте, например, подключение к интернету).

```
val receiver = object : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        val state = intent.getStringExtra("state")
        Toast.makeText(context, "State: $state", Toast.LENGTH_SHORT).show()
    }
}
registerReceiver(receiver, IntentFilter("com.example.MY_ACTION"))
```

Регистрация

```
class WifiReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        val wifiState = intent.getIntExtra(WifiManager.EXTRA_WIFI_STATE, -1)
        when (wifiState) {
            WifiManager.WIFI_STATE_ENABLED -> Log.d("Receiver", "Wi-Fi включён")
            WifiManager.WIFI_STATE_DISABLED -> Log.d("Receiver", "Wi-Fi выключен")
        }
    }
}

// Регистрация в Activity/Fragment
val receiver = WifiReceiver()
val filter = IntentFilter(WifiManager.WIFI_STATE_CHANGED_ACTION)
registerReceiver(receiver, filter)

// Не забудьте отменить регистрацию!
unregisterReceiver(receiver)
```

Отправка собственных broadcast'ов

```
// Отправка
val intent = Intent("com.example.ACTION_DONE")
intent.putExtra("result", "success")
sendBroadcast(intent)

// Приём
val receiver = object : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        val result = intent.getStringExtra("result")
        Toast.makeText(context, "Результат: $result", Toast.LENGTH_SHORT).show()
    }
}
registerReceiver(receiver, IntentFilter("com.example.ACTION_DONE"))
```

Важные моменты

Не забывайте снимать регистрацию динамических приёмников в onStop()/onPause(), иначе возможна утечка памяти.
Для некоторых событий в Android 8+ (API 26+) статические broadcast'ы работают только для системных событий — кастомные только через динамическую регистрацию!
Внутри onReceive нельзя показывать AlertDialog, запускать тяжёлые задачи, делать сетевые запросы — используйте сервисы (JobIntentService, WorkManager).

Ссылка на Git

Практика

Задача 1.

Реализовать BroadcastReceiver, который отслеживает подключение/отключение зарядки устройства и отображает уведомление/Toast.

Задача 2. Создаём и принимаем собственный broadcast

Реализуйте кнопку "Отправить событие", которая шлёт broadcast с вашим уникальным action.

Сделайте BroadcastReceiver, который ловит это событие и отображает в Toast-е текст из extra.

Задача 3.

Сделайте кнопку "Выполнить расчёт".

При нажатии запускается вычисление в отдельном потоке (Thread).
Результат вычисления отображается на экране (в TextView).

Задача 4.

Напишите код, который определяет, выполняется ли он сейчас в главном потоке или нет, и выводит соответствующее сообщение в лог или Toast.

Задача 5.

Сделайте две Activity:
1. Одна отправляет broadcast с некоторыми данными (например, текстом).
2. Вторая регистрирует приемник и отображает полученные данные в UI.

Домашнее задание

Задача 1.

Сделайте экран, где:

- отображается уровень батареи,
- состояние Wi-Fi,
- авиарежим.

Всё обновляется автоматически через broadcast.

При каждом изменении визуально обновляйте соответствующий элемент UI.

Q&A

Ваши вопросы

Спасибо

<TeachMeSkills/>