# МНОГОПОТОЧНОСТЬ И ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ В РАЗРАБОТКЕ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ ДЛЯ ОПЕРАЦИОННОЙ СИСТЕМЫ ANDROID

**Д. О. Шинкевич**

*fpm.shinkevi@bsu.by;*
*Научный руководитель — М. И. Давидовская, старший преподаватель*

Статья посвящена исследованию многопоточности в операционной системе Android. Рассмотрены различные инструменты для реализации многопоточного выполнения, такие как библиотеки java.util.concurrent, RxJava, а также возможности Android SDK. Описаны особенности проектирования и разработки приложения с применением указанных инструментов на примере приложения для работы с электронной почтой.

***Ключевые слова***: Android; многопоточность; Java; параллелизм; мобильное приложение; RxJava.

В ОС Android доступ к графическому интерфейсу есть только у главного потока приложения. В то же время программы для мобильных устройств обладают нагруженным пользовательским интерфейсом, выполняют сетевые запросы, запросы к базе данных и другие трудоёмкие задачи. Выполнение таких задач в главном потоке может привести к тому, что он окажется заблокированным и не сможет подготовить следующий кадр для отображения или ответить на действия пользователя. Следовательно, чтобы избежать блокировки интерфейса и сохранить отзывчивость приложения, необходимо все потенциально длительные задачи выполнять параллельно в отдельном потоке.

Многопоточность применяется при работе не только с операциями (Activity), но и с другими основными компонентами Android-приложений. Службы (Service) используют для выполнения длительных фоновых задач, широковещательные рассылки (Broadcast Receiver) — для уведомления о результате выполнения фоновой задачи. Доступ к данным с помощью контент-провайдера (Content Provider) должен выполняться асинхронно, так как может занимать длительное время.

Рассмотрим основные подходы к реализации многопоточного выполнения в приложениях, разработанных на языке программирования Java.

1. Первый из них — использование библиотеки java.util.concurrent [1]. Она предоставляет интерфейсы-исполнители *Executor* и *ExecutorService*, которые принимают задачи на исполнение в виде Callable- и Runnable-объектов. Данные интерфейсы автоматически

создают пул потоков для выполнения задач. Многократное использование потоков из пула позволяет уменьшить накладные расходы на создание и завершение потоков. Недостатком способа является сложность обработки результата выполнения параллельного потока. Как правило, для обработки результата применяют функции обратного вызова, которые могут содержать последующие обратные вызовы и логику обработки ошибок. В результате сложность программного кода быстро возрастает.

2. Параллельное выполнение задач можно организовать с помощью возможностей Android SDK. Для выполнения задач в главном потоке Android-приложения используется бесконечный цикл *Looper*, который извлекает задачи из очереди сообщений *MessageQueue* и передает их на выполнение обработчику *Handler* [2]. Объект *Handler* позволяет добавлять задачи в очередь из любого потока, что позволяет передать результат работы фоновой задачи главному потоку для соответствующего обновления интерфейса. Планировщик *WorkManager* и класс *AlarmManager* позволяют запустить задачу при выполнении определённого условия или по расписанию.
3. Третий инструмент — сторонняя библиотека для реактивного программирования RxJava. Она позволяет представить данные как поток наблюдаемых объектов, на который можно подписаться и легко обрабатывать события появления новых данных и событие ошибки. Для того, чтобы генерация данных наблюдаемым объектом выполнялась в фоновом потоке, а обработка события наблюдателем в главном потоке приложения, достаточно вызвать методы *subscribeOn* и *observeOn* с подходящими планировщиками [3].

Для исследования рассмотренных подходов на практике было спроектировано и разработано приложение почтовый клиент. Приложение должно предоставлять базовые функции для работы с электронной почтой, такие как написание и отправка письма, просмотр писем по категориям, работа с черновиками, обновление статуса «прочитано» и «избранное», отложенная отправка писем.

В качестве архитектурного шаблона выбран шаблон «Модель, Представление, Модель представления» (Model-View-ViewModel — MVVM). Согласно данному шаблону источник данных предоставляет их в виде наблюдаемых объектов и не содержит ссылок на получателя данных. В результате модель представления не содержит ссылок на операции или фрагменты с коротким жизненным циклом, значит,

избегаем утечек памяти, связанных с попыткой обращения к представлению, которое завершило работу. Диаграмма пакетов приложения показана на рисунке.

В приложении два источника данных: почтовый сервер, для доступа к которому использован JavaMail API, и локальная база данных закешированых писем для более быстрого доступа, для работы с которой использована библиотека Room. Чтобы предоставить единый интерфейс для работы с данными из разных источников, использован шаблон «Репозиторий». Обращения к базе данных и к серверу являются продолжительными задачами, которые необходимо выполнять асинхронно.

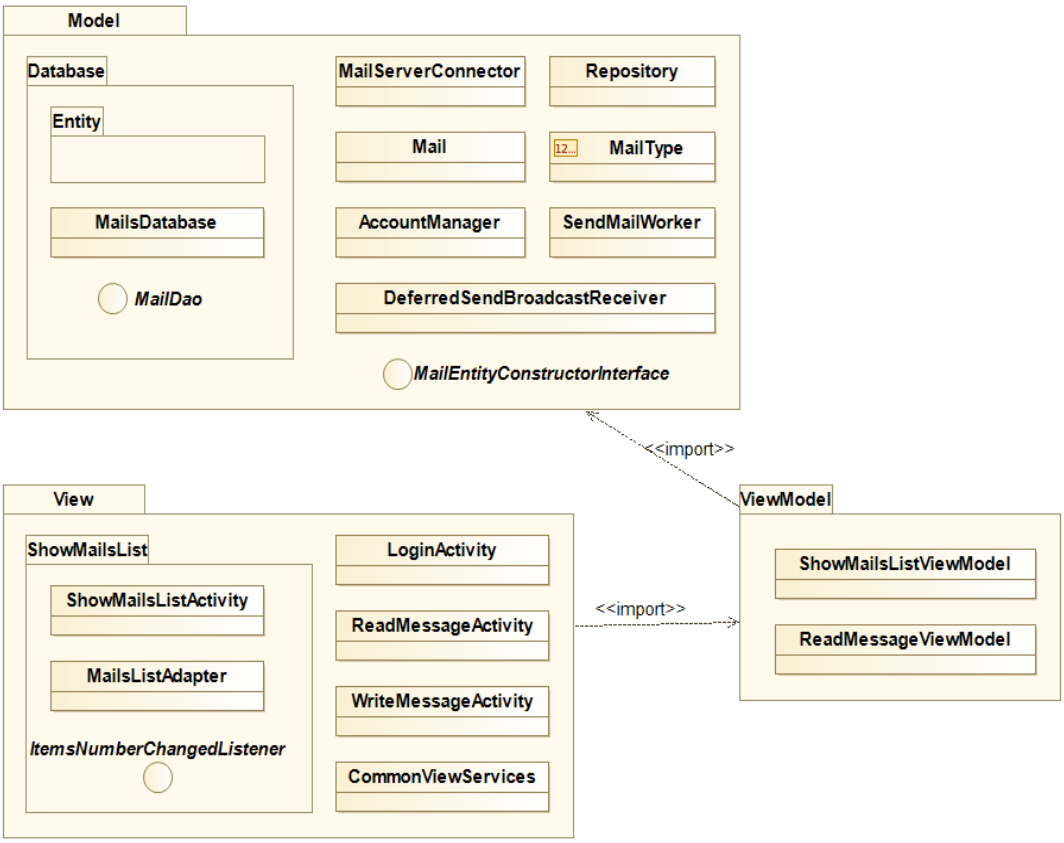


Диаграмма пакетов приложения

Для асинхронного получения данных модель возвращает данные в виде наблюдаемых объектов библиотеки RxJava. Они создаются на основе объектов интерфейса *Callable*, которые реализуют получение данных. Например, для получения писем из базы данных использован наблюдаемый объект *Single*, как показано в листинге. Полный исходный код приложения доступен в репозитории [4]. Модель представления подписывается на наблюдаемый объект и обрабатывает события

появления нового элемента данных — *next*, завершения потока — *complete* или ошибку — *error*. Модель представления преобразует данные в другой наблюдаемый объект *LiveData* и передаёт его в представление. Класс *LiveData* больше подходит для передачи данных представлению, поскольку он предназначен для работы с объектами с жизненным циклом.

```
private <T extends MailEntity> Single<List<Mail>> getLocalMails(
            Supplier<List<T>> mailsFromDBGetter) {
  return Single.fromCallable(() -> {
    List<T> mailEntitiesFromDB = mailsFromDBGetter.get();
    List<Mail> mailsFromDb = new
ArrayList<>(mailEntitiesFromDB.size());
    for (T mailEntity : mailEntitiesFromDB) {
      Mail mail = mailEntityToMail(mailEntity);
      mailsFromDb.add(mail); }
    return mailsFromDb;
  }).subscribeOn(Schedulers.io()); }
```

Получение писем из базы данных с использованием наблюдаемого объекта Single

Планирование задач рассмотрено на примере отложенной отправки письма. Для назначения точного времени выполнения задачи использован класс *AlarmManager*, предназначенный для работы с системной службой оповещений. При наступлении заданного времени выполняется широковещательная рассылка, которая обрабатывается приложением. Для отправки письма необходимо подключение к интернету. Поэтому в приёмнике рассылки запускается новая задача с помощью планировщика *WorkManager*, который позволяет отложить выполнение задачи до тех пор, пока не выполнится определённое условие. В данном случае таким условием является наличие интернет-соединения. Метод *doWork*, содержащий реализацию задачи для *WorkManager*, автоматически вызывается в параллельном потоке.

В работе рассмотрены наиболее популярные инструменты для реализации многопоточного выполнения от средств языка Java и Android SDK до сторонней библиотеки RxJava. Выделены особенности архитектуры приложения, которое взаимодействует с базой данных и сервером асинхронно, рассмотрен процесс доставки данных от источника к представлению, показаны особенности реализации на примере проектирования и разработки почтового клиента. Большое количество приложений для ОС Android предполагают асинхронное выполнение задач. Результаты данной работы имеют практическую значимость и применимы в области разработки мобильных приложений для ОС Android.

## Библиографические ссылки


1. java.util.concurrent. Package summary: Android Developers [Electronic resource]. — Mode of access: https://developer.android.com/reference/java/util/concurrent/package-summary. Date of access: 12.10.2022.
2. Programming Android with Kotlin / Pierre-Olivier Laurence [et al.]. — Sebastopol: O'Reilly Media, Inc., 2021\ 532 p.
3. Multi-Threading Like a Boss in Android With RxJava 2: Gojek Tech [Electronic Resource]. Mode of access: https://www.gojek.io/blog/multi-threading-like-a-boss-in-android-with-rxjava-2Date of access: 15.10.2022.
4. Репозиторий приложения «Почтовый клиент» [Электронный ресурс]. — Режим доступа: https://github.com/DaraShin/MailClient. — Дата доступа: 31.05.2023.
5. *Шинкевич, Д. О.* Многопоточное программирование для ОС Android / Д. О. Шинкевич, М. И. Давидовская // Научные исследования XXI века [Электронный ресурс]. 2023. № 1 (21). С. 17-22. Режим доступа: http://scientific-research.ru/files/JOURNAL. 1-21-.pdf. Дата доступа: 01.05.2023.