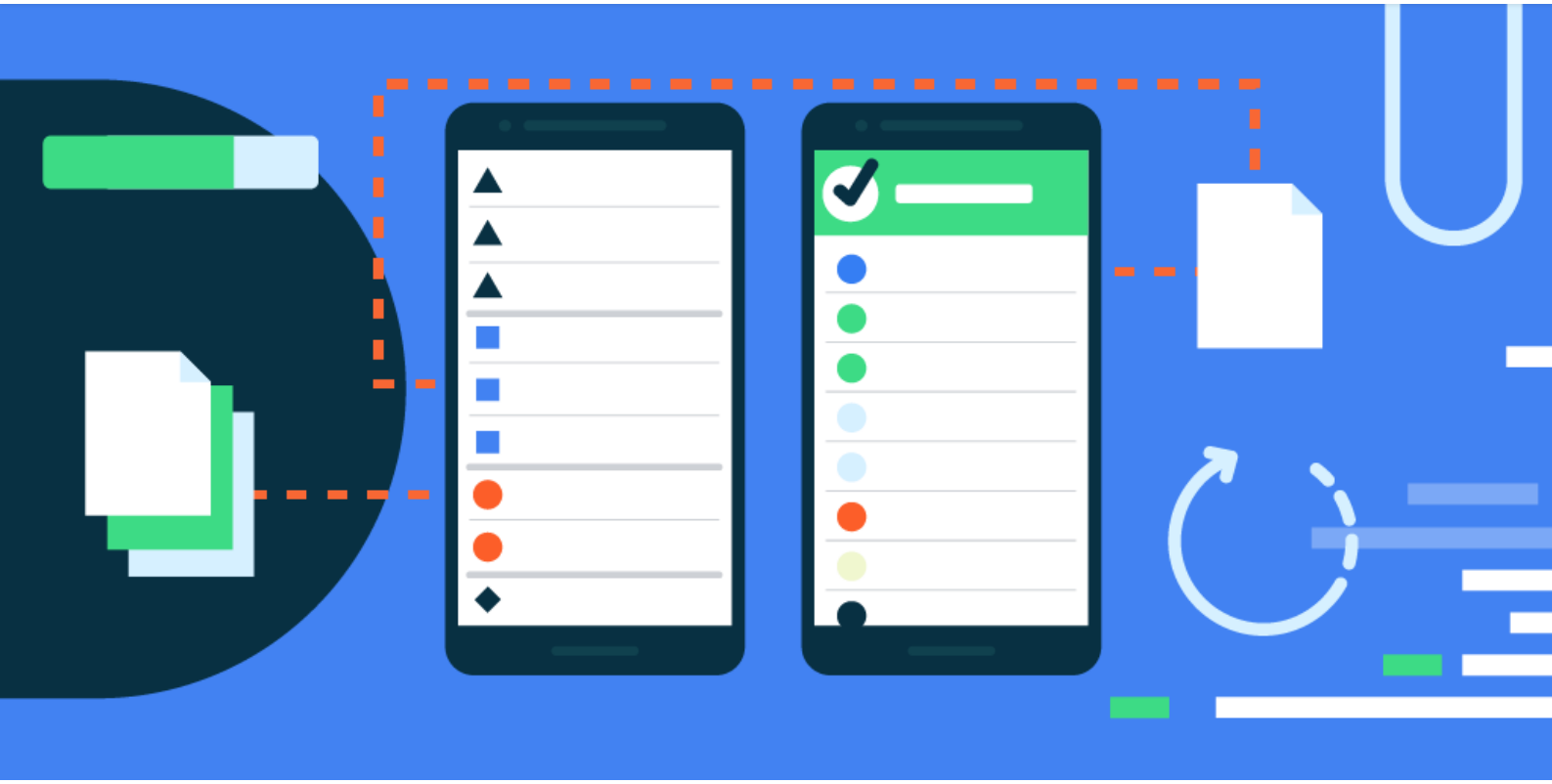


使用 Paging 3 实现分页加载

谷歌开发者
已认证的官方帐号
已关注

作者 / Florina Muntenescu

Paging 库可以帮助您优雅地渐进加载大型数据集合，同时也可以减少网络的使用和系统资源的消耗。基于您的反馈我们得知，Paging 2.0 API 还不能满足开发者们的需求——开发者们希望以更简便的方式处理错误；以更灵活的方式实现列表数据的转换操作，例如 map 和 filter；以及支持分割符、页眉和页脚。基于以上反馈，我们推出了 Paging 3.0。这是一个完全使用 Kotlin 协程重写的库(依然支持 Java 用户)，它将为您提供您所要求的功能。

Paging 3 亮点

Paging 3 的 API 对分页加载时可能需要实现的常见功能提供了支持:

- 跟踪获取前一页或后一页所需要的参数；
- 当用户滚动到现有数据的末尾时，自动请求正确的下一页；
- 确保不会同时触发多个请求；
- 跟踪加载状态，并支持您在 RecyclerView 的列表项或者界面中的其他地方展示它。为失败的加载提供简便的重试功能；
- 无论您是否使用 Flow、LiveData、RxJava Flowable 或 Observable，都可以对需要展示的列表使用 map 或 filter 这类常见的操作；
- 提供实现列表分隔符的简便方法；
- 简化了数据缓存，确保不会让您在每次配置更改时都执行数据转换。

我们还让 Paging 3 的一些组件向后兼容 Paging 2.0。因此，如果您已经在应用中使用了 Paging，则可以逐步 **迁移至 Paging 3**。

在您的应用中使用 Paging 3

假设我们正在实现一个展示所有狗狗的应用。狗狗的数据从 GoodDoggos API 获得，该 API 支持基于索引的分页。让我们研究下需要实现的 Paging 组件，以及如何将 Paging 集成到现有的应用架构。接下来的例子将使用 Kotlin 及其协程功能编写，如果您需要使用 LiveData/RxJava 实现的 Java 编程语言示例，请参阅 **Android 开发者文档 | Paging 3 库概述**。

下图为您应用的各个层级中推荐直接接入 Paging 的 Android 应用架构:

Paging 组件及其在应用架构的集成

定义数据源

数据源的定义取决于您从哪里加载数据。您仅需实现 **PagingSource** 或者 PagingSource 与 **RemoteMediator** 的组合:

- 如果您从 **单个源** 加载数据，例如网络、本地数据、文件等，实现 PagingSource 即可，如果您使用了 **Room**，从 2.3.0-alpha 开始，它将默认为您实现 Paging Source，请参见: **Android 开发文档 | 使用 Room DAO 访问数据**；
- 如果您从一个 **多层级数据源** 加载数据，就像带有本地数据库缓存的网络数据源那样。那么您需要实现 RemoteMediator 来合并两个数据源到一个本地数据库缓存的 PagingSource 中。

PagingSource

PagingSource 可以定义一个分页数据的数据源，以及从该数据源获取数据的方式。
PagingSource 应当为资源库层的一部分。您可以实现 **load()** 函数来从数据源获取分页数据，并返回加载好的数据和加载前后页的参数信息。load() 是一个挂起函数，您可以在这里调用其他的 **挂起函数**，例如网络请求:

```
class DoggosRemotePagingSource(
    val backend: GoodDoggosService
) : PagingSource<Int, Dog>() {
  override suspend fun load(
    params: LoadParams<Int>
  ): LoadResult<Int, Dog> {
    try {
      // 未定义时加载第 1 页
      val nextPageNumber = params.key ?: 1
      val response = backend.getDoggos(nextPageNumber)
      return LoadResult.Page(
        data = response.doggos,
        prevKey = null, // 仅向后翻页
        nextKey = response.nextPageNumber + 1
      )
    } catch (e: Exception) {
        // 在此块中处理错误
        return LoadResult.Error(exception)
    }
  }
}
```

分页数据的容器被称为 **PagingData**，每次刷新数据时，都会创建一个 PagingData 的实例。如果要创建 PagingData 数据流，您需要创建一个 Pager 实例，并提供一个 **PagingConfig** 配置对象和一个可以告诉 Pager 如何获取您实现的 PagerSource 的实例的函数，以供 Pager 使用。

您要在 ViewModel 中构造 Pager 对象并向 UI 暴露一个 Flow。Flow 有一个方便的 **cachedIn()** 方法，该方法使得数据流可以被共享，也让您可以在 CoroutineScope 中缓存 Flow 的内容。这样一来，如果您在数据流中实现了任何转换操作，当 Activity 被重建并使得您从 flow 中获取数据时，不会再次触发这些操作。由于我们希望数据在配置产生变化后仍然存在，缓存应当尽可能靠近 UI 层，但又不能在 UI 层中，那么最好的位置便是在 **ViewModel** 中，并使用 **viewModelScope**:

```
val doggosPagingFlow = Pager(PagingConfig(pageSize = 10)) {
  DogRemotePagingSource(goodDoggosService)
}.flow.cachedIn(viewModelScope)
```

PagingDataAdapter

为了将 RecyclerView 与 PagingData 联系起来，您需要实现一个 **PagingDataAdapter**:

```
class DogAdapter(diffCallback: DiffUtil.ItemCallback<Dog>) :
  PagingDataAdapter<Dog, DogViewHolder>(diffCallback) {
  override fun onCreateViewHolder(
    parent: ViewGroup,
    viewType: Int
  ): DogViewHolder {
    return DogViewHolder(parent)
  }

  override fun onBindViewHolder(holder: DogViewHolder, position: Int) {
    val item = getItem(position)
    if(item == null) {
      holder.bindPlaceholder()
    } else {
      holder.bind(item)
    }
  }
}
```

接下来，在您的 `Activity/Fragment` 中，您需要收集 `Flow<PagingData>` 并将其提交给 `PagingDataAdapter` 。下面是一个在 `Activity` 的 `onCreate()` 函数中实现该操作的示例:

```
val viewModel by viewModels<DoggosViewModel>()

val pagingAdapter = DogAdapter(DogComparator)
val recyclerView = findViewById<RecyclerView>(R.id.recycler_view)
recyclerView.adapter = pagingAdapter

lifecycleScope.launch {
  viewModel.doggosPagingFlow.collectLatest { pagingData ->
    pagingAdapter.submitData(pagingData)
  }
}
```

分页数据转换

展示一个过滤后的列表

转换 PagingData 流与您在其他数据流中所做的同类操作相似。举例来说，如果我们只想要展示 Flow<PagingData> 中那些调皮的狗狗，我们可能需要映射 Flow 对象并过滤 PagingData:

```
doggosPagingFlow.map { pagingData ->
        pagingData.filter { dog -> dog.isPlayful }
    }
```

有分隔符的列表

向列表中添加 **分隔符** 同样是分页数据转换，这里我们通过转换 PagingData 向列表中插入分隔对

型类以支持新的分隔项。当您修改 PagingData 并插入分隔符时，您会用到 insertSeparators 转换：

```
pager.flow.map { pagingData: PagingData<Dog> ->
  pagingData.map { doggo ->
    // 将数据流中的项目转换为 UiModel.DogModel。
    UiModel.DogModel(doggo)
  }
  .insertSeparators<UiModel.DogModel, UiModel> { before: Dog, after: Dog ->
      return if(after == null) {
       // 我们到了列表的末尾
          null
      } else if (before == null || before.breed != after.breed) {
          // 上下品种不同，显示分隔符
          UiModel.SeparatorItem(after.breed)
      } else {
          // 无分隔符
          null
      }
    }
  }
}.cachedIn(viewModelScope)
```

就像前面一样，我们会在数据到达 UI 层之前使用 cachedIn，这样便可以缓存所有已经加载的数据以及数据转换的结果。当配置发生改变时，这些缓存就会被复用。

使用 RemoteMediator 进行高级分页操作

当您从一个 **多层级数据源** 加载数据时，应当实现一个 RemoteMediator。举例来说，在此类的实现中，您应当从网络请求数据并存入数据库。每当数据库中没有数据可以被展示时，就会触发 load() 方法。基于 PagingState 和 LoadType，我们可以构造下一页的数据请求。

由于 Paging 库并不知道您的 API 是怎样的，所以定义如何构造和获取前一页和下一页的远程数据的工作便需要由您自己来完成。举例来说，您可以将您从网络接收到的每个项目与远程关键字关联起来并存入数据库。

```
override suspend fun load(loadType: LoadType, state: PagingState<Int, Dog>): Me

    val page = ...  // 基于 loadType 和 state 进行计算

    try {
        val doggos = backend.getDoggos(page)
        doggosDatabase.doggosDao().insertAll(doggos)

        val endOfPaginationReached = emails.isEmpty()
        return MediatorResult.Success(endOfPaginationReached = endOfPaginationRe
    } catch (exception: Exception) {
        return MediatorResult.Error(exception)
    }
}
```

如果您从网络请求数据并存入数据库，那么数据库才是屏幕上所展示数据的真正数据源——这意味着 UI 会展示从数据库获取的数据，所以您需要为您的数据库实现 PagingSource。如果您正在使用 Room，那么您只需要向您的 DAO 添加一个返回 PagingSource 的查询：

```
@Query("SELECT * FROM doggos")
fun getDoggos(): PagingSource<Int, Dog>
```

```
val pagingSourceFactory = { database.doggosDao().getDoggos() }

return Pager(
     config = PagingConfig(pageSize = NETWORK_PAGE_SIZE),
     remoteMediator = DoggosRemoteMediator(service, database),
     pagingSourceFactory = pagingSourceFactory
).flow
```

您可以参阅文档了解 **使用 RemoteMediator** 的详细信息。如果您需要 RemoteMediator 在应用中的完整实现，可以参阅 **Paging codelab** 和 **Paging 相关代码**。

我们将 Paging 3 设计为一个帮您涵盖简单和复杂情形下的分页加载的库。它可以让您更方便地使用大规模数据集合，无论数据来自网络、数据库、内存缓存还是上述几种情况的组合。Paging 库基于 **协程和 Flow** 实现，使得它可以很简单地调用挂起函数并且处理数据流。

Paging 3 仍然处于 alpha 版本，我们需要您帮助我们进一步优化！请参阅以下资源开始使用 Paging:

- **Android 开发文档 | Paging 3 库概述**
- **Codelab | Android Paging**
- **代码示例 | Paging With Network Sample**
- **IssueTracker**

发布于 09-25

Android　Android 开发

推荐阅读

Google|再见 SharedPreferences 拥抱...

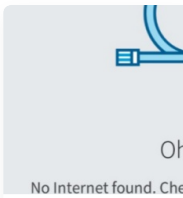
hi-dhl

安卓学习<—>：布局与控件

Arendelle

详谈高大上的图片加载框架 Glide

Code4Android



良好的离线体验 再 Try again, la

Hevin

1 条评论　切换为时间排序

写下你的评论...

jiuzhevange　18 小时前

backend.getDoggos(nextPageNumber)如果是异步的，load返回值该如何处理

赞

▲ 赞同 19　▼　1 条评论　分享　喜欢　收藏　申请转载　…