Data Store چیه؟

Jetpack DataStore یه راه حل برای ذخیره سازی داده ست که این اجازه رو به ما میده که جفت های از

Key-value و یا یه شی از یه نوعی رو به وسیله protocol buffers ذخیره کنیم.

نکته:اگه به یه Dataset بزرگ یا برای کارهای پیچیده احتیاج داشتیم از Room استفاده می کنیم.

DataStore دو نوع مختلف از پیاده سازی رو مطرح می کنه:

1-Preferences DataStore:

دیتا رو ذخیره و به وسیله key به دیتا دسترس داریم.این نوع از پیاده سازی به schema از پیش تعریف شده

نیازی نداره،و ایمنی رو هم فراهم نمی کنه.

2-Proto DataStore:

دیتا رو به عنوان یه نمونه از نوع داده سفارشی ذخیره می کنه.این پیاده سازی نیازمنده اینکه ما براش

Schema رو براش تعریف و از protocol buffers استفاده کنیم،اما ایمنی نوع (Type) رو فراهم می کنه.

Using DataStore correctly یا استفاده صحیح از DataStore:

برای استفاده صحیح از DataStore این قوانین رو همیشه به یاد داشته باشید:

1-**هرگز بیشتر از یه نمونه از DataStore برای ضمیمه کردن فایل تو همون پردازش نسازید**.با انجام این کار

تمام عملکرد DataStore از بین میره.اگه چندین DataStore فعال برای ضمیمه کردن فایل تو همون پردازش

وجود داشته باشه،DataStore “IllegalStateException” رو پرتاب خواهد کرد وقتی که میخونه یا دیتا رو آپدیت

می کنه.

2-**نوع کلی (عمومی) DataStore باید تغییر ناپذیر باشه**.تغییر یه نوع به کاررفته در DataStore هرگونه تضمینی

رو باطل میکنه که DataStore به صورت ذاتی فراهم می کنه و میسازه،(سخته گرفتن باگ ها!).این به شدت

پیشنهاد میشه که ما از protocol budders که تغییرناپذیری یه Api ساده و سریال سازی (serialization) کارآمد

رو تضمین می کنه.

3-هرگز از SingleProcessDataStore و MultiPtocessDataStore رو برای یک فایل استفاده نکنید.اگه میخواید که

به DataStore دسترسی پیدا کنید از بیشتر از یک Process همیشه از MultiProcessDataStore استفاده کنید.

Setup:

دپندنسی ها لازم رو به gradle اضافه می کنیم.

**Preferences DataStore**

// Preferences DataStore (SharedPreferences like APIs)  
dependencies {  
 implementation("androidx.datastore:datastore-preferences:1.0.0")  
  
 // optional - RxJava2 support  
 implementation("androidx.datastore:datastore-preferences-rxjava2:1.0.0")  
  
 // optional - RxJava3 support  
 implementation("androidx.datastore:datastore-preferences-rxjava3:1.0.0")  
}  
  
// Alternatively - use the following artifact without an Android dependency.  
dependencies {  
 implementation("androidx.datastore:datastore-preferences-core:1.0.0")  
}

### Proto DataStore

// Typed DataStore (Typed API surface, such as Proto)  
 dependencies {  
 implementation("androidx.datastore:datastore:1.0.0")  
  
 // optional - RxJava2 support  
 implementation("androidx.datastore:datastore-rxjava2:1.0.0")  
  
 // optional - RxJava3 support  
 implementation("androidx.datastore:datastore-rxjava3:1.0.0")  
 }  
  
 // Alternatively - use the following artifact without an Android dependency.  
 dependencies {  
 implementation("androidx.datastore:datastore-core:1.0.0")  
 }

ساخت یه Preferences DataStore:

از ویژگی by برای ساخته شدن preferencesDataStore استفاده می کنیم تا یه نمونه از Datastore<Preferences>

بسازیم.در بالاترین سطح از فایل کاتلین یکبار صداش می زنیم به این ترتیب در بقیه برنامه مون بهش

دسترسی داریم.این باعث میشه که اون آسونتر نگه داری کنه از DataStore به عنوان یه Singleton.از طرف

دیگه از RxPreferenceDataStoreBuilder استفاده کنید اگه شما از RxJava استفاده می کنید.پارامتر اجباری

Name همون اسم Preferences DataStore هست.

// At the top level of your kotlin file:  
val Context.dataStore: DataStore<Preferences> by preferencesDataStore(name = "settings")

خوندن از Preferences DataStore:

چون Preferences DataStore از schema از پیش تعریف شده استفاده نمیکنه،برای هر value یه نوع متد رو

تعریف کنیم ما به یه نمونه از DataStore<Preferences> نیاز داریم برای ذخیره کردن.برای مثال،یه key تعریف

می کنیم برای یه مقدار int، برای همین از intPreferencesKey() استفاده می کنیم.بعدش،از ویژگی

DataStore.data برای نمایش value ذخیره شده و از یه Flow استفاده می کنیم

val EXAMPLE\_COUNTER = intPreferencesKey("example\_counter")  
val exampleCounterFlow: Flow<Int> = context.dataStore.data  
 .map { preferences **->** // No type safety.  
 preferences[EXAMPLE\_COUNTER] ?: 0  
 }

نوشتن روی Preferences DataStore:

Preferences DataStore متد edit() رو فراهم میکنه که به صورت تعاملی دیتا رو در DataStore آپدیت میکنه.

متدهای transform یه بلوک کد هستند که تو اون میتونیم مقادیر رو در صورت نیاز آپدیت کنیم.

suspend fun incrementCounter() {  
 context.dataStore.edit { settings **->** val currentCounterValue = settings[EXAMPLE\_COUNTER] ?: 0  
 settings[EXAMPLE\_COUNTER] = currentCounterValue + 1  
 }  
}

**Define a schema**:

Proto DataStore نیازمند یه schema از پیش تعریف شده در یه proto فایل در میسر app/src/main/proto/.

این schema تعریف میکنه نوع رو برای شی که ما اصرار دارید باشه در Proto DataStore تون.

syntax = "proto3";  
  
option java\_package = "com.example.application";  
option java\_multiple\_files = true;  
  
message Settings {  
 int32 example\_counter = 1;  
}

Create a Proto DataStore:

دو مرحله برای ساخت یه Proto DataStore برای ذخیره کردن نوع object ها وجود داره:

1-تعریف یه کلاسی که Serializer<T> رو پیاده سازی کرده باشه به جای T نوع تعریف شده در فایل proto

قرار میگیره.این کلاس serializer به DataStore میگه که نوع دیتا رو چطور بخونه و بنویسه.مطمئن شید که

Serializer شامل یه مقدار پیش فرض برای استفاده باشه اگه هنوز file ساخته نشده بود.

2-با استفاده از by یه نمونه از DataStore<T> بسازید،به جای T نوع file proto رو تعریف کنید.اون رو در

بالاترین جای فایل کاتلین تعریف کنید که در بقیه برنامه در دسترس باشه.پارامتر filename به DataStore

میگه از کدوم file برای ذخیره سازی دیتا استفاده کنه،و پارامتر serializer میگه اسم DataStore کلاس

Serializer تعریف شده در قدم اول رو بهش پاس میدیم.

object SettingsSerializer : Serializer<Settings> {  
 override val defaultValue: Settings = Settings.getDefaultInstance()  
  
 override suspend fun readFrom(input: InputStream): Settings {  
 try {  
 return Settings.parseFrom(input)  
 } catch (exception: InvalidProtocolBufferException) {  
 throw CorruptionException("Cannot read proto.", exception)  
 }  
 }  
  
 override suspend fun writeTo(  
 t: Settings,  
 output: OutputStream) = t.writeTo(output)  
}  
  
val Context.settingsDataStore: DataStore<Settings> by dataStore(  
 fileName = "settings.pb",  
 serializer = SettingsSerializer  
)

خوندن از یه Proto DataStore:

از DataStore.data برای نمایش Flow مناسب از شی ذخیره شده استفاده می کنند.

val exampleCounterFlow: Flow<Int> = context.settingsDataStore.data  
 .map { settings **->** // The exampleCounter property is generated from the proto schema.  
 settings.exampleCounter  
 }

نوشتن روی یه Proto DataStore:

Proto DataStore متد updateDate() رو فراهم می کنه که به صورت تعاملی شی ذخیره شده رو آپدیت می کنه.

updateData() به شما وضعیت اخیر دیتا رو به عنوان یه نمونه از نوع دیتا (data type) میدهند و دیتا رو به

صورت تعاملی در یه اپراتور تغییر-خوندن-نوشتن آپدیت می کنند.

suspend fun incrementCounter() {  
 context.settingsDataStore.updateData { currentSettings **->** currentSettings.toBuilder()  
 .setExampleCounter(currentSettings.exampleCounter + 1)  
 .build()  
 }  
}

**Use DataStore in synchronous code**:

یکی از مزایای اصلی DataStore همزمانی API ست،اما این امکان همیشه نیست که ما بتونیم از یه همزمانی

به نا همزمانی برسیم. این امکان وجود داره اگه ما روی کد خارجی دیگه داریم کار می کنیم که اون از

همزمانی I/O استفاده کرده یا اگه ما یه وابستگی داشته باشیم که یه ناهمزمانی رو برای ما فراهم بکنه.

کاتلین کروتین با فراهم کردن runBlocking() که یه کروتین بیلدر هست، کمک میکنه که شکاف بین

همزمانی و ناهمزمانی پر بشه.ما میتونیم با استفاده از runBlocking() دیتا رو از DataStore همزمانی بخونیم.

RxJvava پیشنهاد میده که روی مند Flowable اجرا بشه.کد پایین ترد رو صدا میزنه تا وقتی که DataStore یه

دیتا رو برگردونه.

val exampleData = runBlocking { context.dataStore.data.first() }

انجام دادن اپراتورهای همزمان I/O روی ترد UI میتونه باعث ANRs یا UI jank بشه.ما میتونیم این مشکلات رو

کاهش بدیم با ناهمزمانی پیش بارگذاری دیتا از DataStore.

override fun onCreate(savedInstanceState: Bundle?) {  
 lifecycleScope.launch {  
 context.dataStore.data.first()  
 // You should also handle IOExceptions here.  
 }  
}

این روش،DataStore به طورناهمزمان دیتا رو میخونه و اونا رو تو مموری کش میکنه(caches)

هم زمانی با استفاده از runBlocking() ممکن سریعتر باشه یا ممکن که از اپراتور های I/O جلوگیری کنه

درمجموع اگه مقدار اولیه خوندنش تموم شده باشه

**Use DataStore in multi-process code**: