Retrofit et RxJava

Dans une application graphique, on a souvent besoin d'exécuter des opérations asynchrones, afin de pouvoir exécuter des opérations prennant du temps (réseau, fichiers, etc...) sans bloquer le thread qui gère l'interface graphique. Pour ce faire, on va faire appelle à RxJava.

RxJava

RxJava est une biblothèque qui apporte des outils pour faciliter la programmation avec une approche réactive et asynchrone au langage Java. Le concept principal de RxJava est l'utilisation du pattern Observer. Ce dernier consiste en la manipulation d'`Observable`, qui correspond à un flux (de données, d'événements, etc ...) auquel au Observer peut s'abonner pour en observer les changements, on peut égalmement appliquer des opérateurs sur ces flux pour les manipuler.

Comme nous allons appliquer ce modèle à des appels réseau à des API ReST, nous allons utiliser principalement des `Single` qui sont un cas particulier d'Observable qui au lieu d'émettre une série de valeur, émet seulement une valeur ou alors une erreur.

Retrofit

Pour ce tuto nous allons imaginer que nous disposons d'une simple API ReST avec les endpoints suivants :

- GET `/api/todos` : récupérer tous les Todos
- GET `/api/todo/{id}` : récupérer un Todo par son Id
- POST `/api/todos` : créer un Todo
- DELETE `/api/todos/{id}` : supprimer un Todo
- PUT `/api/todos/{id}` : modifier un Todo

Retrofit est une bibliothèque Java qui permet de créer facilement des clients pour des API ReST. Avec Retrofit, on a juste à définir une interface Java qui correspond à notre API, et l'implémentation sera générée.

Voici l'interface Retrofit correspondante à notre API :

```
public class TodoApi {


  @GET("/api/todos")
  Single<List<Todo>> getTodos();
```

```
  @GET("/api/todos/{id}")
  Single<List<Todo>> getTodo(@Path("id") String todoId);

  @POST("/api/todos")
  Single<Todo> createTodo(@Body Todo todo);

  @DELETE("/api/todos/{id}")
  Single<ResponseBody> deleteTodo(@Path("id") String todoId);

  @PUT("/api/todos/{id}")
  Single<Todo> update(@Path("id") String todoId, @Body Todo todo);
}
```

Les annotations suivantes permettent de définir notre interface :

- Annotation de méthode HTTP (`@GET`, `@POST`, `@DELETE`, `@PUT`) : permet de mapper une méthode Java à un endpoint HTTP. Le nom de l'annotation permet de définir la méthode HTTP et sont paramètre permet de définir la route du endpoint
- `@Path` sur un paramètre de méthode permet de templater la route du endpoint
- `@Body` sur un paramètre de méthode permet de passer un objet de modèle en tant que paramètre de la méthode

Pour récupérer l'implémentation d'une interface Retrofit :

```
TodoApi api = new Retrofit.Builder()
                .baseUrl(BASE_URL)
                .addConverterFactory(GsonConverterFactory.create())
                .addCallAdapterFactory(RxJava2CallAdapterFactory.create())
                .build()
                .create(TodoApi.class);
```

Intégration

On va créer une classe de Service qui sera consommée par notre ViewModel.

Revision #2
Created 2021-01-31 14:43:58 UTC by Arsène Lapostolet
Updated 2021-04-02 14:24:32 UTC by Arsène Lapostolet