Retrofit

# Hvad er Retrofit

Retrofit er en REST-klient til Android og Java by Square. Det gør det relativt nemt at hente og uploade JSON (eller andre strukturerede data) via en REST-baseret webservice. I Retrofit konfigurerer du, hvilken konverter der bruges til dataserialisering. Typisk for JSON bruger du GSon, men du kan tilføje tilpassede konvertere til at behandle XML eller andre protokoller. Retrofit bruger OkHttp biblioteket til HTTP anmodninger.

# Hvorfor har jeg brugt retrofit?

jeg brugte retrofit til at fech noget data fra en Rest api bare for at prøve og har viden om hvordan man henter data fra en rest, fordi jeg var ny i android og jeg vil gerne vide, hvad der er den bedste og hurtigste måde at hente data fra en Rest Api i en Android app.

# Brug af Retrofit

-For at arbejde med Retrofit behøver du grundlæggende tre klasser.

-Model klasse, som bruges til at kortlægge JSON data til

-Grænseflader, der definerer de mulige HTTP-operationer

Retrofit.Builder class - Instance, der bruger grænsefladen og Builder API, som gør det muligt at definere URL slutpunktet for HTTP operationen.

Hver metode til en grænseflade repræsenterer et muligt API-opkald. Den skal have en HTTP-annotation (GET, POST, osv.) For at angive anmodningstypen og den relative URL. Returværdien ombryder svaret i et Opkaldsobjekt med typen af det forventede resultat.

@GET("users")

Call<List<User>> getUsers()

Du kan bruge udskiftningsblokke og forespørgselsparametre til at justere webadressen. En erstatningsblok tilføjes til den relative URL med {}. Ved hjælp af @Path-annotationen på metodeparameteren er værdien af denne parameter bundet til den specifikke udskiftningsblok.

@GET("users/{name}/commits")

Call<List<Commit>> getCommitsByName(@Path("name") String name)

Query parametre tilføjes med @Query-annotationen på en metodeparameter. De tilføjes automatisk i slutningen af webadressen.

@GET("users")

Call<User> getUserById(@Query("id") Integer id)

@Body-annotationen på en metodeparameter fortæller Retrofit at bruge objektet som anmodningsorgan til opkaldet.

@POST("users")

Call<User> postUser(@Body User user)

# [3. Retrofit converters and adapters](http://www.vogella.com/tutorials/Retrofit/article.html#retrofit-converters-and-adapters)

Retrofit kan konfigureres til at bruge en specifik konverter. Denne konverter håndterer data (de) serialisering. Flere omformere er allerede tilgængelige for forskellige serialiseringsformater.

At konvertere til og fra JSON:

Gson: com.squareup.retrofit: converter-gson

Jackson: com.squareup.retrofit: converter-jackson

Moshi: com.squareup.retrofit: converter-moshi

At konvertere til og fra protokollbuffere:

Protobuf: com.squareup.retrofit: converter-protobuf

Wire: com.squareup.retrofit: converter-wire

At konvertere til og fra XML:

Enkel XML: com.squareup.retrofit: converter-simplexml

Udover de listede konvertere kan du også oprette brugerdefinerede konvertere til at behandle andre protokoller ved at underklassificere Converter.Factory-klassen

[Adapters](http://www.vogella.com/tutorials/Retrofit/article.html#retrofit-adapters):

Retrofit kan også udvides med adaptere til at blive involveret med andre biblioteker som RxJava 2.x, Java 8 og Guava.

En oversigt over tilgængelige adaptere findes på Github square / retrofit / retrofit-adapters /.

For eksempel kan RxJava 2.x adapteren opnås ved at bruge Gradle:

compile 'com.squareup.retrofit2:adapter-rxjava2:latest.version'

Apache Maven:

<dependency>

<groupId>com.squareup.retrofit2</groupId>

<artifactId>adapter-rxjava2</artifactId>

<version>latest.version</version>

</dependency>

For at tilføje en adapter skal retrofit2.Retrofit.Builder.addCallAdapterFactory (Factory) -metoden bruges

Retrofit retrofit = **new** Retrofit.Builder()

.baseUrl("https://api.example.com")

.addCallAdapterFactory(RxJava2CallAdapterFactory.create())

.build();

Når denne adapter anvendes, er Retrofit-grænsefladerne i stand til at returnere RxJava 2.x-typer, fx Observable, Flowable eller Single og så videre.

@GET("users")

Observable<List<User>> getUsers();

# 4. [Authentication with OkHttp interceptors](http://www.vogella.com/tutorials/Retrofit/article.html#authentication-with-okhttp-interceptors)

Du kan bruge en interceptor til dette. En interceptor bruges til at ændre hver anmodning, før den udføres, og ændrer anmodningsoverskriften. Fordelen er, at du ikke behøver at tilføje @Header ("Authorization") til hver API-metoden definition.

Til at tiføje en interceptor, skal du bruge det okhttp3.OkHttpClient.Builder.addInterceptor(Interceptor) method i det OkHttp Builder.

OkHttpClient okHttpClient = **new** OkHttpClient().newBuilder().addInterceptor(**new** Interceptor() {

@Override

**public** okhttp3.Response intercept(Chain chain) **throws** IOException {

Request originalRequest = chain.request();

Request.Builder builder = originalRequest.newBuilder().header("Authorization",

Credentials.basic("aUsername", "aPassword"));

Request newRequest = builder.build();

**return** chain.proceed(newRequest);

}

}).build();

Den oprettede OkHttp-klient skal tilføjes til din Retrofit-klient med metoden retrofit2.Retrofit.Builder.client (OkHttpClient).

Retrofit retrofit = **new** Retrofit.Builder()

.baseUrl("https://api.example.com")

.client(okHttpClient)

.build();

Du kan igen bemærke brugen af Credentials-klassen til Basic-godkendelse. Igen, hvis du vil bruge et API-token, skal du bare bruge token i stedet.

# Konklusion

Min konklusion er retrofit er en type sikker REST-klient til Android og Java., det giver funktionalitet til brugerdefinerede overskrifter, , multipart request body, file uploads and downloads, mocking responses og mere.

Jeg vil meget anbefale at bruge retrofit til at fetch noget fra reatapi, fordi det er nemt at forstår hvordan retrofir virker.