WebAPI 連携

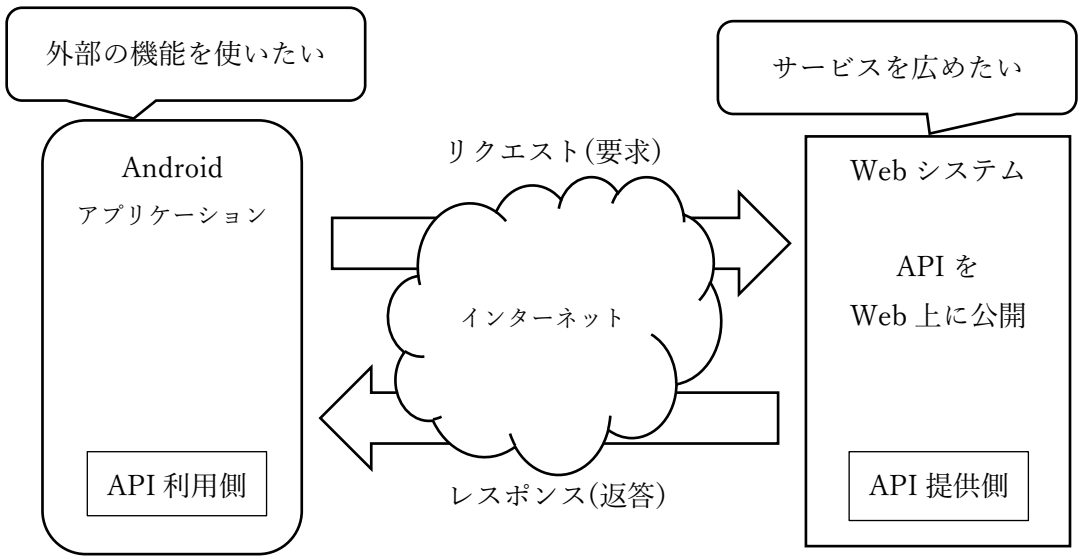
API (アプリケーション・プログラミング・インタフェース)とは

API を一言で表すと、ソフトウェアやプログラム、Web サービス間をつなぐインタフェースのことを指します。インタフェースとは、何かしらの「境界面」「接点」のことを指し、異なる 2 つの事物の間をつなぐという意味を持ちます。
たとえばキーボードやマウス、ディスプレイは人間とコンピューターの境界上で使われる「ユーザーインターフェース」の 1 つとして知られます。また、USB や HDMI といった、機械と機械をつなぐコネクターは「ハードウェアインターフェース」と呼ばれます。
そして API は、主にソフトウェアやプログラム同士をつないでいるインタフェースです。

API の仕組み

API はユーザーが必要とするたびに利用（リクエスト）され、その応答（レスポンス）を得ることで利用されます。リクエストは利用しているアプリケーションで API の利用が必要とされるたびに行われ、期待している結果が得られなかった場合にも都度レスポンスが返却されます。

・WebAPI の仕組み



※ルール（仕様）に則って、両者をつなぐ

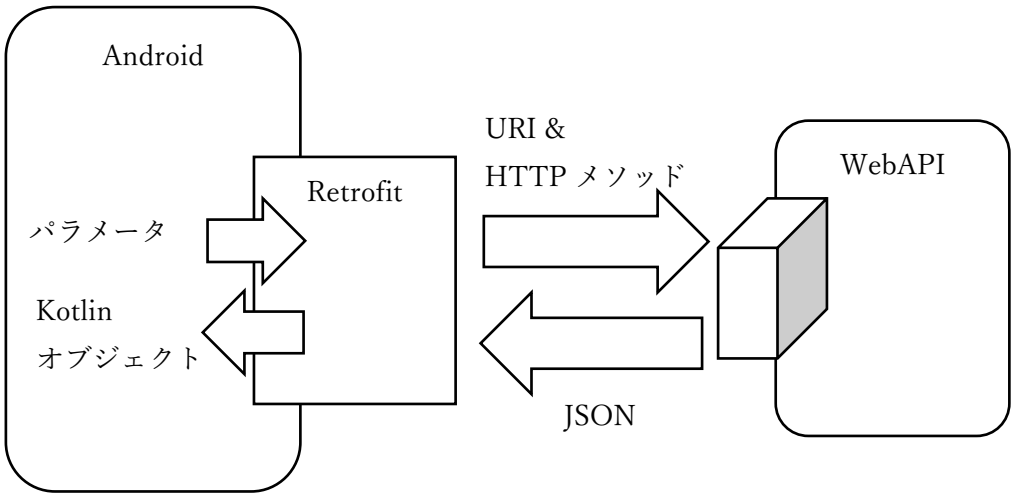
Retrofit の導入

WebAPI を Andoroid アプリケーションで使用するためにはアプリ内に HTTP 通信の処理を組み込む必要があります。もちろんフルスクラッチで作成することは可能ですが、学習コストが高いので、特別な事情が無ければライブラリを使うことをお勧めします。

今回は、Retrofit という HTTP クライアント通信ができるライブラリを導入します。
GET、POST だけでなく、PUT、PATCH、DELETE、OPTIONS、HEAD の HTTP メソッドに対応しているため、REST API にも対応が可能なライブラリです。

・Retrofit 公式サイト
https://square.github.io/retrofit/

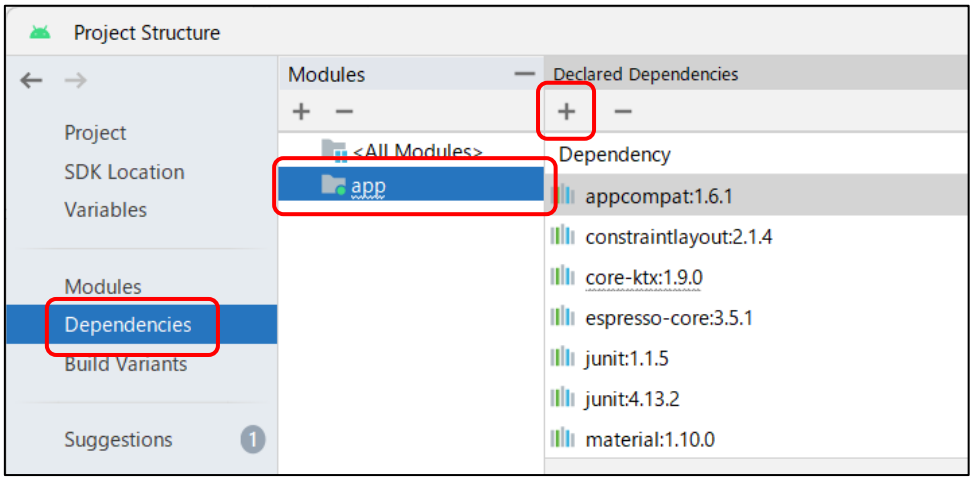
Retrofit では、使用する API をインタフェースとアノテーションにて定義します。
なお、Retrofit の実態は OkHttp のラッパーであり、OkHttpClient を使用して機能を拡張することができます。



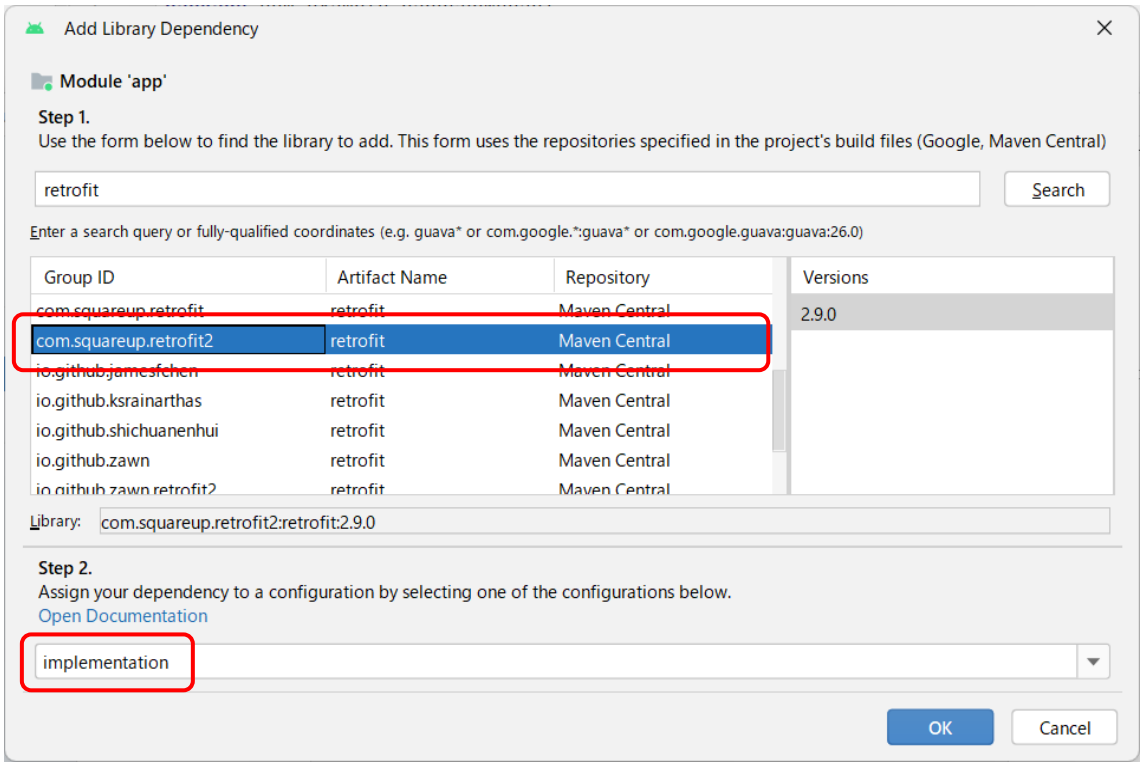
ハンズオン　プロジェクトに Retrofit を導入する

---

1. HandsOnWebAPI のプロジェクトを開く
2. 「Project Structure」をクリックして、プロジェクト構造設定ダイアログを表示する
3. Dependencis を選択して、さらに app を選択、その中にある「＋」ボタンを選択



4. 追加したいライブラリを検索して、対象のライブラリとバージョンを指定する



※上記ライブラリがヒットしない場合は、App レベルの Gradle ファイルに手動追加する

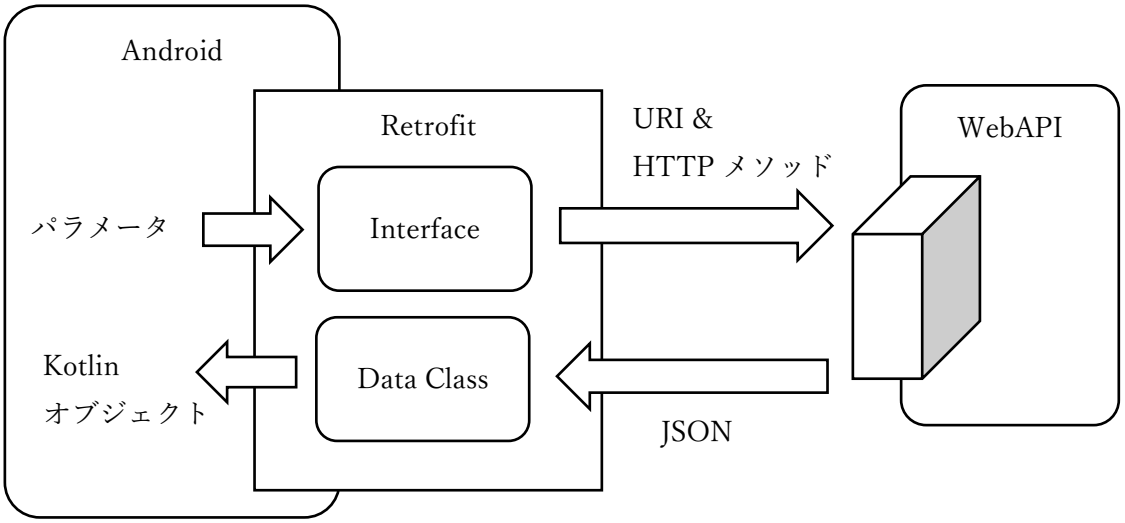
```
dependencies {
    // 検索にヒットしない場合は手動で追記する
    implementation("com.squareup.retrofit2:retrofit:2.9.0")
    implementation("com.squareup.retrofit2:converter-gson:2.9.0")
}
```

5. AndroidManifest.xml にインターネット許可のパーミッション設定を行う

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Retrofit のインタフェースとデータクラスについて

Retrofit では API リクエストを投げるメソッドをインタフェースとして用意する必要があります。HTTP メソッドはアノテーションを入れるだけであとは Retrofit 側が判断して処理を行ってくれます。また、レスポンスは JSON と構造を合わせた Data Class を準備してすることで JSON ではなく、Data Class の Kotlin オブジェクトとして値を受け取ることが可能となる。




```
interface WeatherService {
    @GET("data/2.5/weather")
    fun fetchData(
        @Query("q") city: String,
        @Query("lang") lang: String,
        @Query("appid") apiKey: String
    ): Call<Weather>
}
```

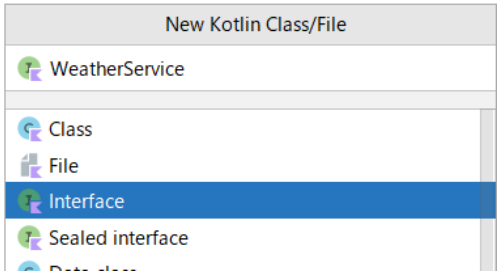
URI＆HTTP メソッド

パラメータ

JSON と同階層を持つ DataClass

ハンズオン　API 接続用のインタフェースを作成する

1．インタフェースを新規作成する



2．作成したインタフェースのトップレベルに WebAPI のベース URL を宣言する

```
const val BASE_URL = "https://api.openweathermap.org/"
interface WeatherService {

}
```

3．メソッドをインタフェースに追加して GET アノテーションを付与する

```
@GET("data/2.5/weather")
fun fetchData(): Call<Weather>
```

アノテーション	概要
@GET	引数にはパスの続きやクエリパラメータ、プレースホルダ{}を設置して変数渡しが可能
@POST	
@PUT	
@DELETE	

4．fetchData メソッドにリクエストパラメータの受取用の引数を追加する

```
fun fetchData(
    @Query("q") city: String,
    @Query("lang") lang: String,
    @Query("appid") apiKey: String
): Call<Weather>
```

※実装後は、以下 URL を指定しているのと同じ動きになります
https://api.openweathermap.org/data/2.5/weather?q={city}&lang={lang}&appid={apiKey}

非同期処理について

Android では、応答の悪さでアプリが反応しなくなるのを防ぐために HTTP 通信などをメインメソッドで実行しようとするとエラーになります。
※HTTP 通信のレスポンスが遅くて画面描画が出来ずに動かなくなるのを防ぐため
https://developer.android.com/reference/android/os/NetworkOnMainThreadException.html

上記の仕様から、Android で HTTP 通信を行うためには、非同期通信でプログラムを行う必要があります。Retrofit では enqueue メソッドを使用することで非同期通信を行うことができます。

ハンズオン　リサイクラービューに WebAPI の結果を表示する

---

1. API の結果を受け取るクラス変数 MutableList を宣言する

```
private var cityList = mutableListOf<String>()
```

2. onCreate メソッドで Retrofit ビルダーを生成する

```
val retrofit = Retrofit.Builder()
    .baseUrl(BASE_URL)
    .addConverterFactory(GsonConverterFactory.create())
    .build()
```

3. Retrofit ビルダーに WeatherService インタフェースをセットする

```
val api = retrofit.create(WeatherService::class.java)
```

4. 都市リストの繰り返し処理を行う

```
cities.forEach { city ->

}
```

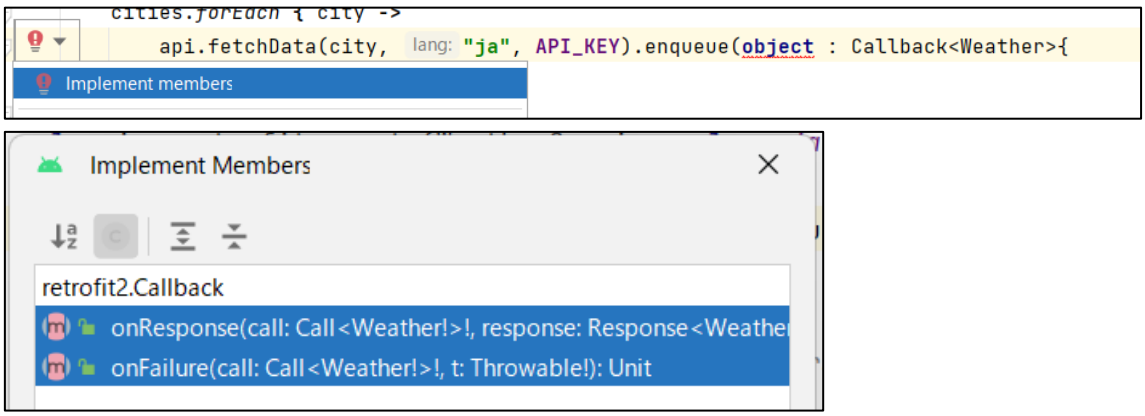
5. GET メソッドにパラメータをセットして非同期通信で実行する処理を記述する

```
api.fetchData(city, "ja", API_KEY).enqueue(object : Callback<Weather>{

})
```

※API キーは各自で取得してください

6. HTTP 通信の正常終了、異常終了のメソッドをオーバライドする



7. 正常終了のコールバックメソッドに WebAPI の結果をリストに格納する

```
response.body()?.let{ weather ->
    cityList.add(weather.name)
}
```

8. リサイクラービューを lateinit var でクラス変数に変更する
9. リスト内容の更新通知を行い、リサイクラービューを最新にする

```
recyclerView.adapter?.let{
    it.notifyDataSetChanged()
}
```

10. リサイクラービューに渡すリストを API の結果が格納されたリストに変更する

```
recyclerView.adapter = WeatherAdapter(cityList, onClick = { cityName ->
```