

本日の内容

- ・ WebAPI
- ・ Http 接続(Retrofit)
- ・ JSON
- ・ 非同期処理

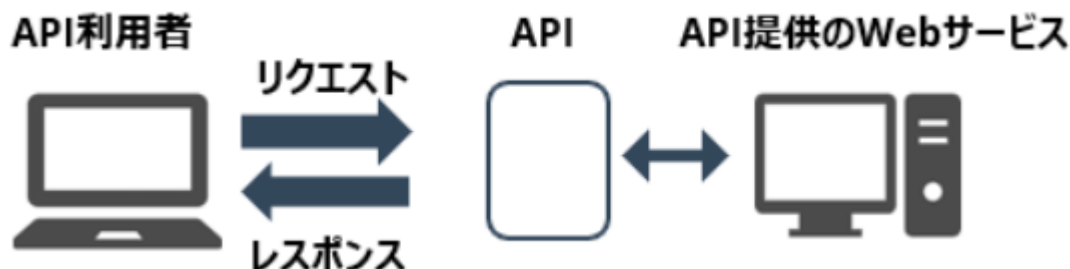
今回の目的：

Retrofit を使用して天気情報 API からデータを取得し、
内容を表示する Android アプリの作成方法を学びます

■1. WebAPI

■1.1 そもそも WebAPI とは？

「Application Programming Interface」の略、Web 上で API のやり取りを可能にしたものです
簡単に言えば、異なるソフトウェア同士がお互いにコミュニケーションをとるための「通訳」のようなものと考えることができます。



例を挙げるなら「Google Map API」、「Amazon API」、「Youtube API」などなど

- ・ 新たな機能やサービス開発を助ける
- ・ より良い顧客体験を提供する
- ・ 自社サービスのユーザを拡大する

上記など目的に企業などが公開している仕組みです。

■1.2 API の役割

考えてみてください。あなたのアプリが天気情報をユーザに提供したいとします。しかし、自分のアプリで気象データを生成することは難しいですよね？ ここで API が役立ちます。気象データを提供する外部のサービスから、そのデータを取得するために API を使用します。

■1.3 RESTful API

今回のお天気アプリで使用するのは RESTful API というタイプの API です。

RESTful API は、Web 上で情報をやり取りするためのシンプルで標準的な方法を提供します。

URL を使ってリソースにアクセスし、HTTP メソッド（GET、POST、PUT、DELETE など）を使って操作を行います。

■1.4 なぜ API が重要か？

データのアクセス
API を使用することで、アプリは必要なデータやサービスに簡単にアクセスできます
効率
すでに提供されているデータやサービスを再利用できるので、開発の時間とコストを節約できます。
拡張性
新しい機能やデータを追加する際に、API を使って既存のシステムと簡単に統合できます。

■1.5 API の具体的な例：天気情報の取得までの流れ

① 必要な情報の特定

まず、アプリが何の情報を必要としているのかを特定します。

例：「特定の都市の現在の天気情報を取得したい」

② 適切な API の選定

必要な情報を提供する API を探します。この例では、OpenWeatherMap というサービスが提供する天気情報 API を使用します。

③ API のドキュメンテーションの確認

選んだ API の公式ドキュメンテーションを読み、どのようにリクエストを送ればよいのか、どんな情報が返ってくるのかを理解します。

④ API エンドポイントの特定

API にアクセスするための URL（エンドポイント）を特定します。

例: <https://api.openweathermap.org/data/2.5/weather>

⑤ 必要なパラメータの特定

API リクエストに必要なパラメータをドキュメンテーションから特定します。

例: 都市名 (q)、言語 (lang)、API キー (appid) など

⑥ API リクエストの実行

エンドポイントとパラメータを組み合わせて、実際に API にリクエストを送ります。

例: https://api.openweathermap.org/data/2.5/weather?q=Tokyo&lang=ja&appid=YOUR_API_KEY

⑦ レスポンスの解析

API から返ってきたレスポンス（通常は JSON 形式）を解析し、必要なデータを取り出します。

例: 東京の現在の天気や気温など

```
{
  "name": "Tokyo",
  "weather": [
    {
      "description": "clear sky"
    }
  ],
  "main": {
    "temp": 298.15
  }
}
```

■2. Retrofit とは

公式ドキュメント: [Retrofit \(square.github.io\)](https://square.github.io/retrofit/)

使い方参考ページ: [最新の Retrofit の使い方についてまとめた \(2022 年 10 月時点\) #Android - Qiita](#)
[Retrofit の基本的な使い方 - くま's Tech 系 Blog \(hatenablog.com\)](#)

まず、Retrofit は一言で言うと、アプリと Web サービス（例えば天気の情報を提供するサービス）との間で情報をやり取りする時に役立つツールです。

インターネット上のサービスと「話す」時には、特定のルールや方法で会話をしなければなりません。Retrofit は、その会話の方法を簡単にしてくれる役割を持っています。

◆Retrofit の主な特徴

1. タイプセーフ

Retrofit はタイプセーフです。これは、コンパイル時に型の検証を行うという意味で、API から返されるデータ型と、アプリケーション内で期待するデータ型が一致することを確認できます。これにより、ランタイムエラーを大幅に削減できます。

2. カスタマイズ可能

Retrofit はカスタマイズが容易で、Gson, Jackson, Moshi などの人気のある JSON パーサーと簡単に統合できます。これにより、API から返される JSON レスポンスをポジョエンティティに簡単に変換できます。

3. 非同期と同期の呼び出し

Retrofit は非同期と同期の両方の API 呼び出しをサポートしています。非同期呼び出しは、API リクエストをバックグラウンドで実行し、メインスレッドで結果を受け取るためのコールバックを提供します。

■2.1 Retrofit の良さは？

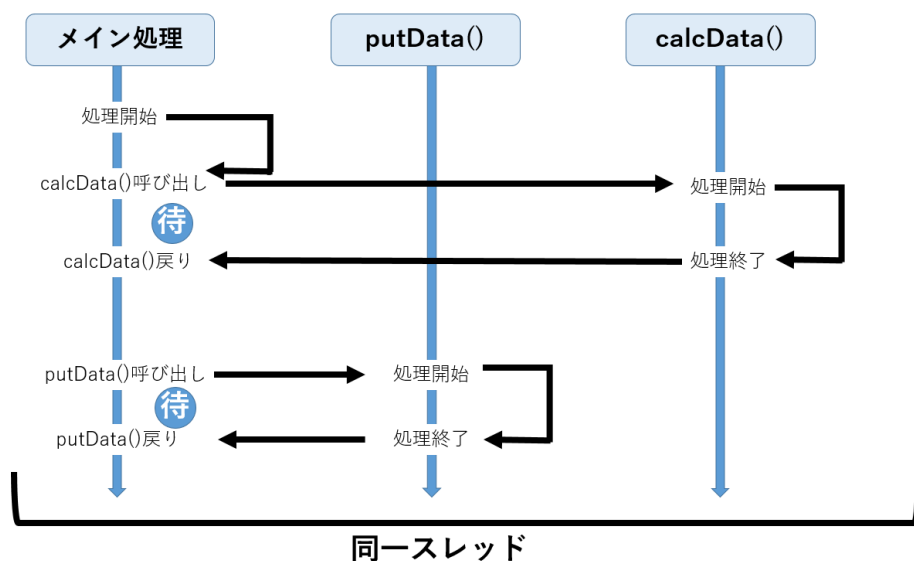
タイプセーフ: これは、情報を安全にやり取りすることができるという意味です。間違った形式のデータを送らないようにしてくれます。

アノテーションベース: これは、特別なマーク (@のような記号) を使って、簡単に設定ができるということです。

非同期処理サポート: アプリが情報を取得する時、少し待たされることがありますよね？これをスムーズに行うためのサポートがあります。

API を使用する上で大切な同期処理、非同期処理についても学びましょう。

■2.2 同期処理

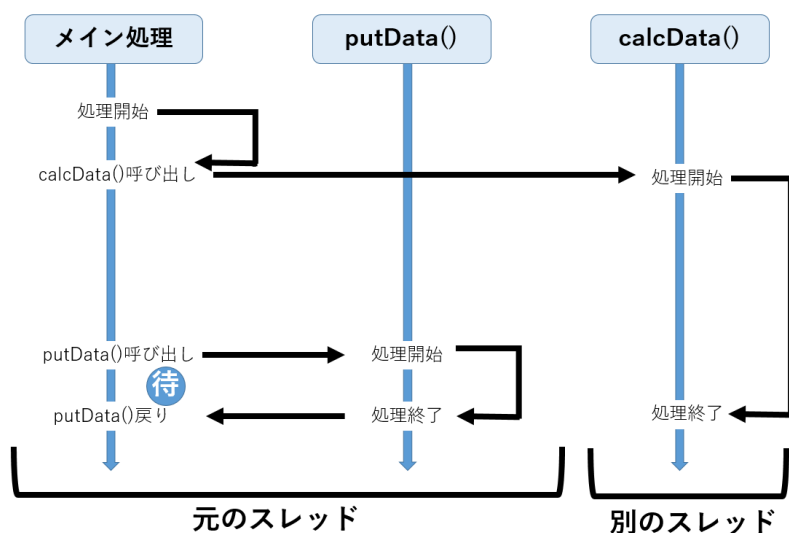


java 言語でも処理の一連の流れをスレッドという単位で扱います。

同ースレッド内で処理を行う場合、それぞれの処理が終わってから次の処理へ移ります。

上記のように、別メソッド処理が終わってからメイン処理を再開する流れを**同期処理**といいます。

■2.3 非同期処理の必要性



もし、`calcData()`の処理がとても時間のかかる処理の場合、ずっと待つことになります。ユーザ視点ではフリーズしているように思えます。これは当然、好ましくありません。これを避けるためには `calcData()`の終了を待たずに、メイン処理と同期せずに処理を進める必要があります。これを**非同期処理**といいます

【※補足】

Android の一番中心となるスレッドは Activity の画面スレッドです。これを **UI スレッド**といいます。非同期で行うスレッドのことを**ワーカースレッド**と言います

■3 コーディング前の事前準備

■3.1 OpenWether の利用準備：アカウント作成

今回使用する API(OpenWeather)は、使うためには API キーが必要となります。

API キーはアカウントを登録することで取得出来ます。

アカウント作成から始めていきましょう。

([Members \(openweathermap.org\)](https://openweathermap.org/members))

Create New Account

We will use information you provided for management and administration purposes, and for keeping you informed by mail, telephone, email and SMS of other products and services from us and our partners. You can proactively manage your preferences or opt-out of communications with us at any time using Privacy Centre. You have the right to access your data held by us or to request your data to be deleted. For full details please see the OpenWeather [Privacy Policy](#).

☒ I am 16 years old and over


☒ I agree with [Privacy Policy](#), [Terms and conditions of sale](#) and [Websites terms and conditions of use](#)


I consent to receive communications from OpenWeather Group of Companies and their partners:

☐ System news (API usage alert, system update, temporary system shutdown, etc)

☐ Product news (change to price, new product features, etc)

☐ Corporate news (our life, the launch of a new service, etc)

 私はロボットではありません

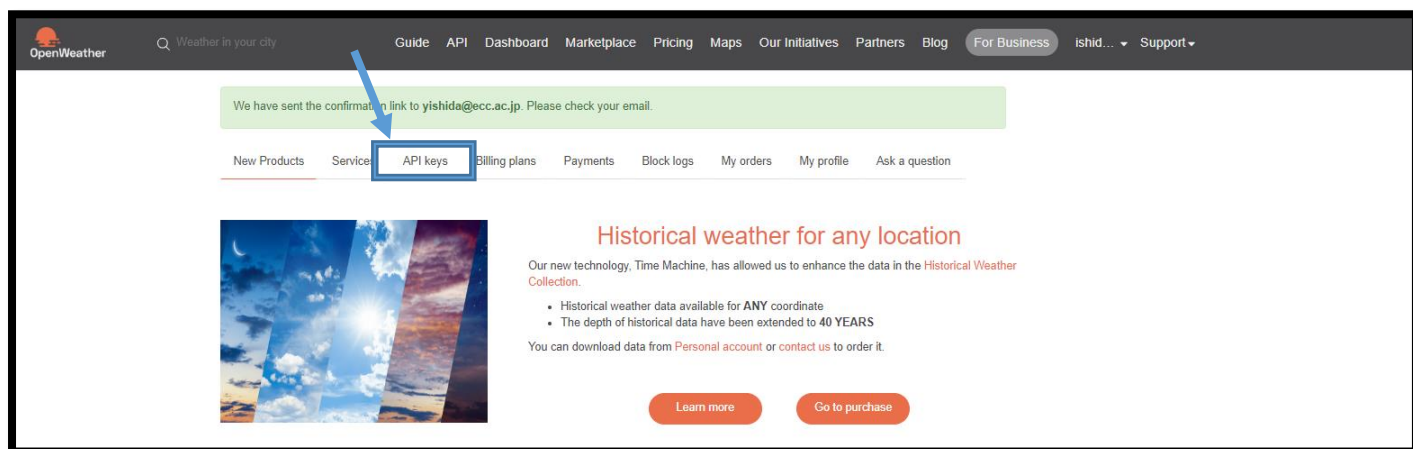

reCAPTCHA
プライバシー・利用規約

Create Account

■3.2 APIKey の取得

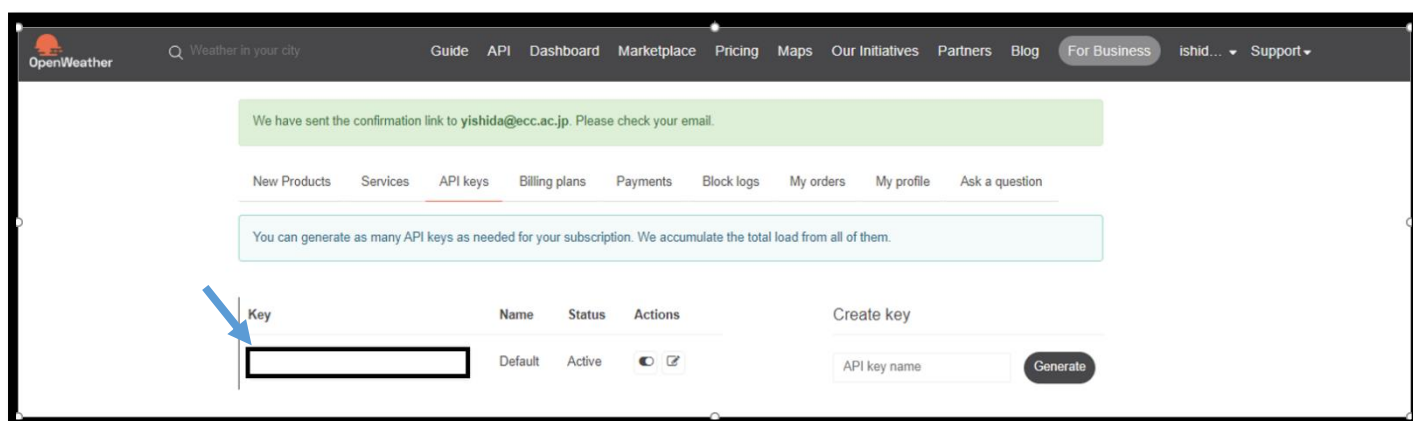
次に、アカウントに紐づいた API キーを取得します。

API Keys タブを選択しましょう



あなた専用の API Key が発行されます。

※API が生成されるのに 5 分程かかる場合があります



この Key 値を後で使用します。

■3.3 OpenWeather の WebAPI 仕様

API キーを取得したら、OpenWeather の webAPI が利用出来ます。

OpenWeather では様々な天気情報を取得出来ます(公式ドキュメント：<https://openweathermap.org/current>)

今回は無料で使用出来る範囲で作成していきます。

`https://api.openweathermap.org/data/2.5/weather?q=city name&lang=ja&appid=API key`

city name に都市名

API key に先程取得した内容

上記でアクセスすることで JSON データを返してくれます。

試しにアクセスしてみましょう

```
https://api.openweathermap.org/data/2.5/weather?q=Himeji&lang=ja&appid=?????
```

■3.4 JSON とは何か？

「JavaScript Object Notification」の略で、JavaScript で値を取り扱うためのドキュメント規格
特に Web 開発において、JSON がどこでも使われるようになったのは、大きなメリットがあったからです！

メリット

- ①テキストベースで軽い
- ②解析しやすい

今回使用する API の応答の内容を簡単に見ておきます。

API 応答のフィールド

- `coord`
 - `coord.lon` 都市の地理的位置、経度
 - `coord.lat` 都市の地理的位置、緯度
- `weather` (詳細 気象条件コード)
 - `weather.id` 気象条件 ID
 - `weather.main` 気象パラメータのグループ(雨、雪、極端など)
 - `weather.description` グループ内の気象条件。あなたはあなたの言語で出力を得ることができます。[詳細情報](#)
 - `weather.icon` 天気アイコン ID
- `base` 内部パラメータ
- `main`
 - `main.temp` 温度。単位デフォルト:ケルビン、メートル法:摂氏、インペリアル:華氏。
 - `main.feels_like` 温度。この温度パラメータは、人間の天候の知覚を説明しています。単位デフォルト:ケルビン、メートル法:摂氏、インペリアル:華氏。
 - `main.pressure` 大気圧(海面、`sea_level` または `grnd_level` データがない場合)、hPa
 - `main.humidity` 湿度、%
 - `main.temp_min` 現時点での最低気温。これは、現在観測されている最小の温度です(大規模な大都市および都市部内)。単位デフォルト:ケルビン、メートル法:摂氏、インペリアル:華氏。
 - `main.temp_max` 現時点での最高気温。これは、現在観測されている最高気温です(大都市や都市部内)。単位デフォルト:ケルビン、メートル法:摂氏、インペリアル:華氏。
 - `main.sea_level` 海面大気圧、hPa

- `main.grnd_level` 地上大気圧、ヘクトパスカル
- `visibility` 視認性、メートル。視程の最大値は 10km です
- `wind`
 - `wind.speed` 風速。単位デフォルト:メートル/秒、メートル:メートル/秒、インペリアル:マイル/時。
 - `wind.deg` 風向、度(気象)
 - `wind.gust` 突風。単位デフォルト:メートル/秒、メートル:メートル/秒、インペリアル:マイル/時
- `clouds`
 - `clouds.all` 曇り、%
- `rain`
 - `rain.1h` 過去 1 時間の雨量、mm
 - `rain.3h` 過去 3 時間の雨量、mm
- `snow`
 - `snow.1h` 過去 1 時間の積雪量、mm
 - `snow.3h` 過去 3 時間の積雪量、mm
- `dt` データ計算の時刻、ユニックス、UTC
- `sys`
 - `sys.type` 内部パラメータ
 - `sys.id` 内部パラメータ
 - `sys.message` 内部パラメータ
 - `sys.country` 国番号 (GB、JP など)
 - `sys.sunrise` 日の出時刻、ユニックス、UTC
 - `sys.sunset` 日没時刻、ユニックス、UTC
- `timezone` UTC からの秒単位のシフト
- `id` 市区町村 ID。組み込みのジオコーダー機能は非推奨になりました。詳しくは[こちらをご覧ください](#)。
- `name` 都市名。組み込みのジオコーダー機能は非推奨になりました。詳しくは[こちらをご覧ください](#)。
- `cod` 内部パラメータ

更に、http 通信を前回は `okHttp` ではなく
`Retrofit` というライブラリを使用して行います。

今回はお天気 API を活用して非同期処理の記述方法や WebAPI のアクセス方法を学んでいきます。

■3.5 プロジェクト読み込み

以下の GitHub のページからクローン or zip ダウンロードでファイルを配置し開いてください
[iyStudy/Kotlin_retfit_sample_weather \(github.com\)](https://github.com/iyStudy/Kotlin_retfit_sample_weather)

■3.6 gradle に依存関係を追加する

「build.gradle(Module: app)」ファイルを開いて、dependencies { }内に依存関係を追加する処理を追加

```
// Retrofit
implementation("com.squareup.retrofit2:retrofit:2.9.0")
implementation("com.squareup.retrofit2:converter-gson:2.9.0")
```

※Sync now で反映を忘れずに！

■3.7 パーMISSIONの設定

今回のように web アクセスを行う場合、Android は web アクセスの許可を AndroidManifest.xml で設定しなければ、権限が無い為 web アクセスを行うことが出来ません。

ネットアクセスを許可する為に以下を追加しましょう

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

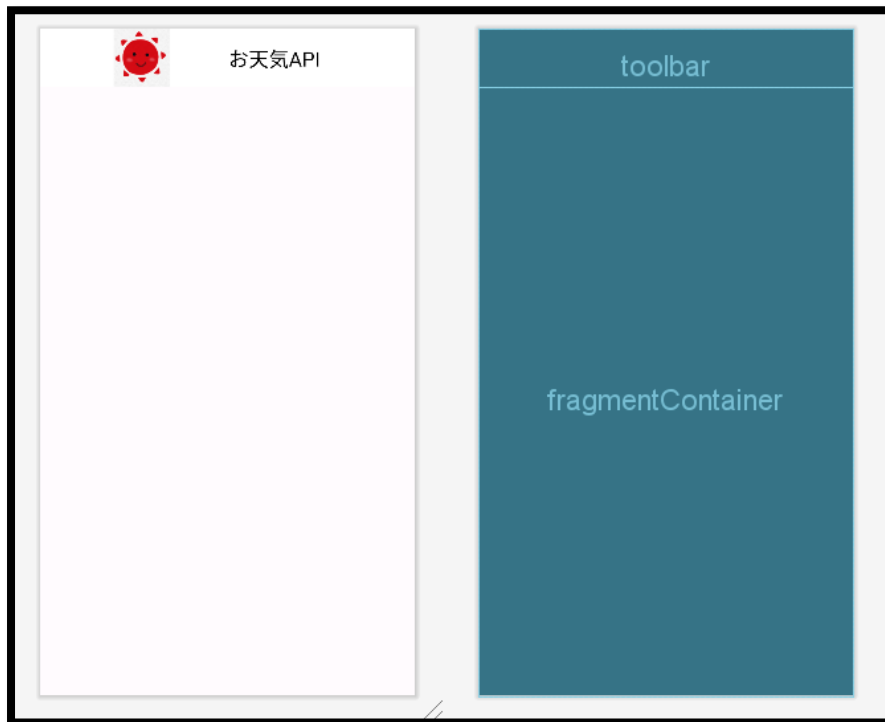
AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Kotlin_retfit_sample_weather"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

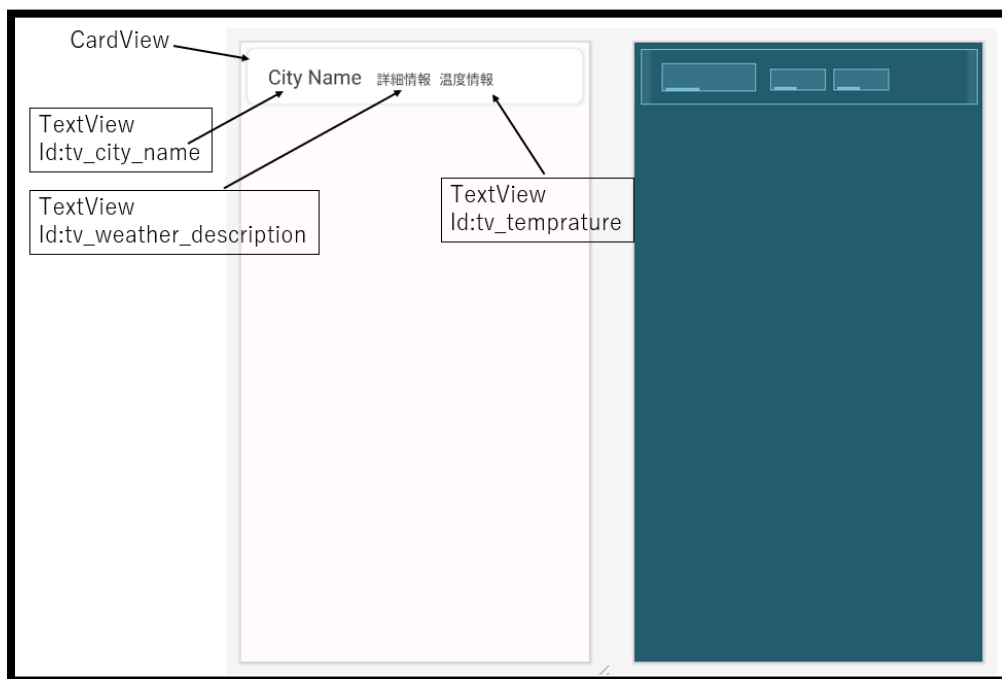
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

■3.8 各レイアウトファイルの準備 及び 確認

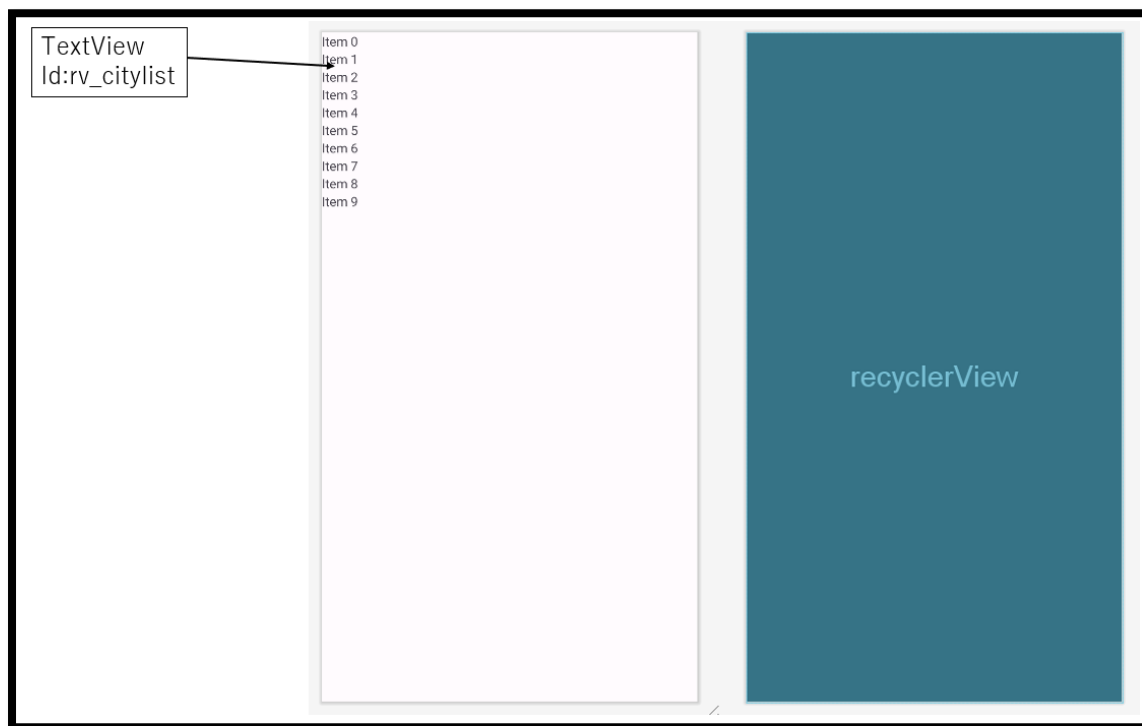
- ・ activity_main.xml (メイン画面)



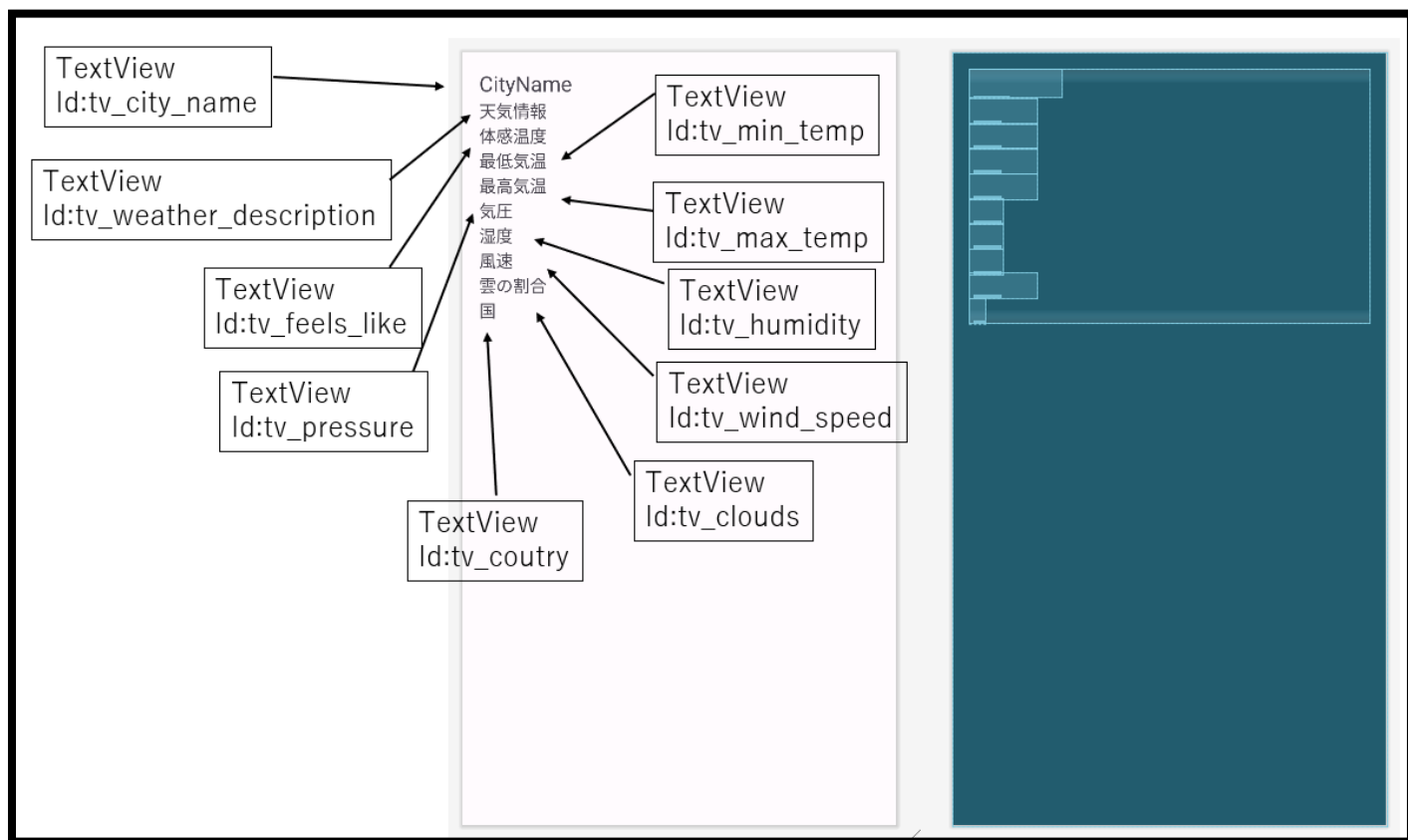
- ・ weather_item.xml (1行分の都市情報画面)



・ fragment_weather_list.xml (都市リスト画面)



・ fragment_weather_details.xml (都市の詳細情報画面)



■4. コーディング

■4.1 データモデルの作成：API からのデータを理解し、アプリで扱える形にする

API からのデータをアプリで使いやすくするためには、そのデータの構造を Java のクラスとして表現する必要があります。このクラスを「データモデル」と呼びます。

◆4.1.2. データモデルって何？

データモデルは、API から取得するデータの「形」や「構造」を Java のクラスで表したものです。これにより、API からのデータをアプリ内で簡単に扱うことができます。

◆4.1.3 データクラス Weather.kt データクラスの例

```
import android.os.Parcelable
import kotlinx.parcelize.Parcelize

@iyStudy *
@Parcelize
data class Weather(
    val weather: List<WeatherDetail>,
    val main: Main,
    val sys: Sys,
    val name: String,
) : Parcelable

@iyStudy
@Parcelize
data class WeatherDetail(
    val id: Int,
    val main: String,
    val description: String,
    val icon: String
): Parcelable
```

上のクラスは、API からのレスポンスデータの一部を表現しています。例えば、name は都市名、main には温度などの情報、weather には天気の詳細などが含まれます。

◆4.1.4 どうやってデータモデルを作るのか？

まず、API のドキュメントやサンプルのレスポンスデータを確認します。そのデータの構造を元に、対応する kt のクラスとして表現します。例えば、API のデータに配列があれば、kt のリストとして表現します。

◆4.1.5 なぜこのステップが大切か？

データモデルを正しく作成しておくことで、後のステップでのデータの取得や表示が簡単になります。
また、データの構造や内容を理解することで、アプリの機能や UI の設計もスムーズに進めることができます。
実際に json を取得し、構造を確認してみましょう。

The diagram illustrates the structure of a JSON response and how it maps to Kotlin data classes. The JSON data is as follows:

```

{
  "coord": {
    "lon": 134.7,
    "lat": 34.8167
  },
  "weather": [
    {
      "id": 803,
      "main": "Clouds",
      "description": "曇りがち",
      "icon": "04d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 302.19,
    "feels_like": 306.54,
    "temp_min": 301.14,
    "temp_max": 302.19,
    "pressure": 1012,
    "humidity": 73,
    "sea_level": 1012,
    "grnd_level": 1011
  },
  "visibility": 10000,
  "wind": {
    "speed": 3.31,
    "deg": 151,
    "gust": 4.91
  },
  "clouds": {
    "all": 70
  },
  "dt": 1694595503,
  "sys": {
    "type": 2,
    "id": 86031,
    "country": "JP",
    "sunrise": 1694551332,
    "sunset": 1694596376
  },
  "timezone": 32400,
  "id": 1862627,
  "name": "姫路市",
  "cod": 200
}

```

The corresponding Kotlin data classes and their fields are:

- coord**
 - lon: 134.7
 - lat: 34.8167
- weather**
 - id: 803
 - main: "Clouds"
 - description: "曇りがち"
 - icon: "04d"
- base**
 - "stations"
- main**
 - temp: 302.19
 - feel_like: 306.54
 - Temp_min: 301.14
 - Temp_max: 302.19
 - pressure: 1012
 - humidity: 73
 - Sea_level: 1012
 - Grnd_level: 1011
- visibility**
 - 10000
- wind**
 - speed: 3.31
 - deg: 151
 - gust: 4.91
- clouds**
 - all: 70
- sys**
 - type: 2
 - id: 86031
 - country: "JP"
 - sunrise: 1694551332
 - sunset: 1694596376
- timesone**
 - 32400
- id**
 - 1862627
- name**
 - "姫路市"
- cod**
 - 200

■4.1.6 ハンズオン：Weather.kt データクラスの作成

手順

- ① build.gradle(app)の plugins 内に id("kotlin-parcelize")を追加
- ② Weather.kt のデータクラス作成
- ③ json データをもとに変数を定義
- ④ Coord クラスを定義し、json データをもとに lon,lat 変数を定義
- ⑤ WeatherDetail というクラスで定義し、weather の変数(id,main,descriptuion,icon)を定義
- ⑥ Main クラスを定義し、json データをもとに main の変数(temp,feel_like..)を定義
- ⑦ Wind クラスを定義し、json データをもとに wind の変数(speed,deg,guest)を定義
- ⑧ Clouds クラスを定義し、json データをもとに wind の変数(clouds)を定義
- ⑨ Sys クラスを定義し、json データをもとに wind の変数(all)を定義

■4.2 API インターフェースの定義：Retrofit でのやり取りの基盤

API とアプリとの間での情報のやり取りをスムーズに行うため、Retrofit では「インターフェース」というものを使います。このインターフェースは、API とのやり取りの「ルール」や「方法」を定義する場所です。

◆4.2.1. なぜインターフェースが必要か？

インターフェースは、どの URL にアクセスするのか、どんなデータを送ったり受け取ったりするのかといった情報を定義する場所です。

Retrofit はこのインターフェースを元に、実際の API へのリクエストを行います。

◆4.2.2. インターフェースの基本的な構造

```
import retrofit2.Call
import retrofit2.http.GET
import retrofit2.http.Query

// 天気情報を取得するための API のインターフェース
interface WeatherApi {

    // @GET アノテーションを使用して、特定のエンドポイントに HTTP GET リクエストを行うメソッドを定義
    // この場合、エンドポイントは"data/2.5/weather"です
    @GET("data/2.5/weather")

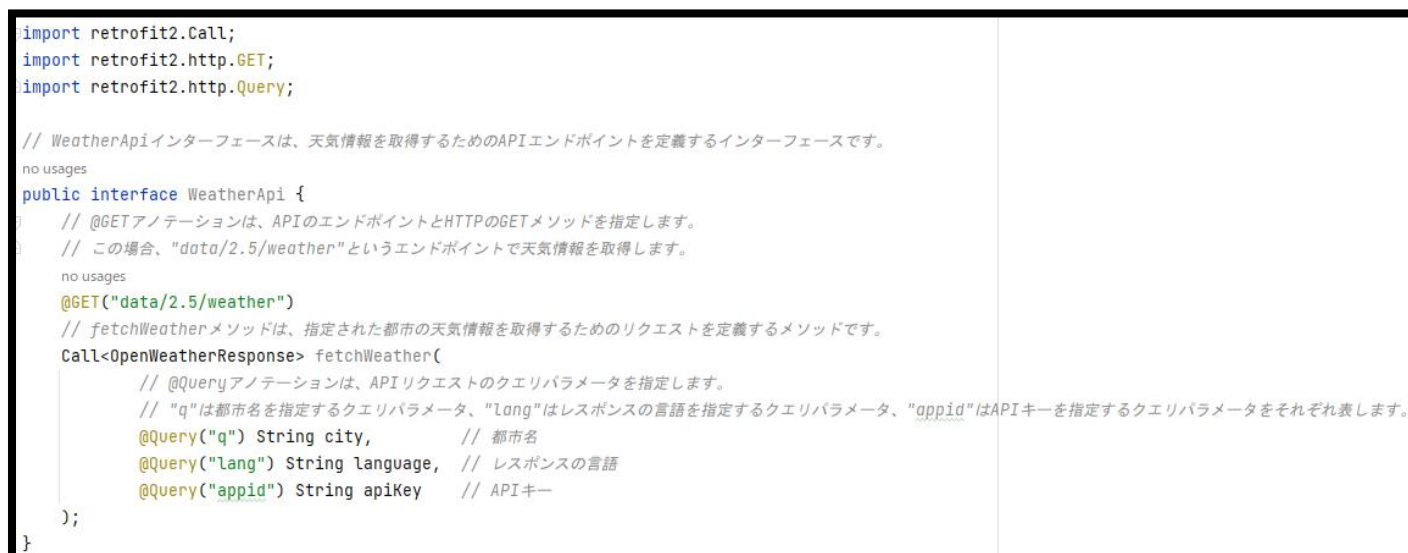
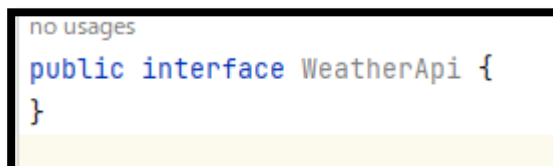
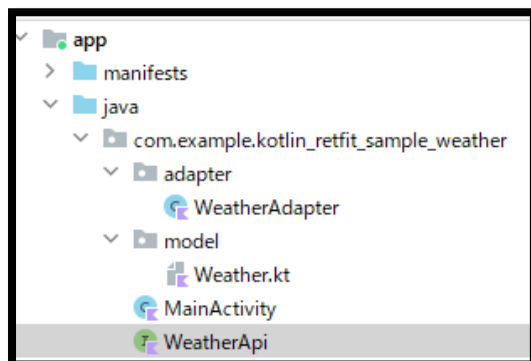
    fun fetchWeather(
        // @Query アノテーションを使用して、リクエストのクエリパラメータを定義
        // "q"は都市の名前を表すクエリパラメータ（例：Tokyo）
        @Query("q") city: String,

        // "lang"はレスポンスの言語を指定するクエリパラメータ（例：ja）
        @Query("lang") lang: String,

        // "appid"は API キーを指定するクエリパラメータ
        // API キーは OpenWeatherMap から取得したものを使用
        @Query("appid") apiKey: String
    ): Call<Weather> // レスポンスとして`Weather`オブジェクトを期待する Call 型を返す
}
```

@GET	この部分は、HTTP の GET メソッドを使って API にアクセスすることを示しています。
"data/2.5/weather"	API のエンドポイント（アクセスする URL の一部）を示しています。
@Query	URL の後ろに付けるパラメータを定義するアノテーションです。 この例では、都市名や言語、API キーをパラメータとして送ります。

■4.2.3 ハンズオン：新規に WeatherApi という名前のインターフェイスファイルを作成してください



■4.3 ハンズオン：Retrofit インスタンスの初期化

Retrofit を使用して API リクエストを行うためには、まず Retrofit のインスタンスを作成する必要があります。

このインスタンスは、API のエンドポイント、データの変換方法などの設定情報を持っています。

1 度作成すればよいだけなので WeatherListFragment クラス内のフィールドで変数を定義し
セットアップ用に定義してある fetchData() メソッド内に記述していきましょう。

手順:

① Retrofit の Builder を使用:

// Retrofit インスタンスを作成するためには、まず設定となる Retrofit.Builder クラスを使用します。

```
val retrofit = Retrofit.Builder()
```

② ベース URL の設定:

// retrofit の baseUrl() メソッドを使用して、API のベース URL を設定します。

// この URL は、後で定義するエンドポイントの URL の基盤となります。

```
.baseUrl("https://api.openweathermap.org/")
```


③ データコンバータの設定:

//APIからのレスポンスデータは通常JSON形式です。

//このJSONデータをJavaのオブジェクトに変換するために、GsonConverterFactoryを追加します。

```
.addConverterFactory(GsonConverterFactory.create())
```

⑤ Retrofit インスタンスの作成:

上記の設定を終えたら、.build()メソッドを呼び出して Retrofit のインスタンスを生成します。

⑥ 初期化完了

下記のようなプログラムになります

```
// Retrofitを使用してAPI通信を行う準備
val retrofit = Retrofit.Builder()
    .baseUrl("https://api.openweathermap.org/") // APIのエンドポイント
    .addConverterFactory(GsonConverterFactory.create()) // 受け取ったJSONをWeatherクラスに変換するためのコンバータ
    .build()
```

注意点:

ベース URL は、末尾に/を含めることが推奨されています。

複数の ConverterFactory を追加することも可能ですが、最初にマッチしたものが使用されます。

なぜこれが必要なのか

Retrofit のインスタンスを作成することで、API エンドポイントへのリクエストを簡単に行うことができます。
また、データの変換やエラーハンドリングなどの機能もこのインスタンスを通じて利用します。

■4.4 ハンズオン；API リクエストの実行

API からデータを取得するためには、リクエストを実行する必要があります。

このセクションでは、Retrofit を使用して API リクエストを送信し、レスポンスを受け取る方法を説明します。

API リクエストを行うメソッドを定義しその中で処理を実装していきましょう。

ファイル名：*WeatherListFragment.kt*

手順

① API_KEY の定義

取得した API_KEY を定数としてフィールドに定義

// OpenWeatherMap の API アクセスに必要なキー

`private val API_KEY = "事前に取得した key を記入"`

② API インターフェースを使ったインスタンスの作成:

天気データを取得する関数(fetchData)内にて、Retrofit インスタンスを使用し、API インターフェースの実装を生成します。

// 定義した API インターフェースからインスタンスを作成

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

③ API へのリクエストの作成:

API インターフェースのメソッドを呼び出して、Call オブジェクトを取得します。

このオブジェクトを使用して、API へのリクエストを制御します。

今回は取得する都市名を"Kyoto"にしておきましょう

`api.fetchWeather("取得する都市名", "ja", API_KEY)`

④ 非同期リクエストの実行:

api インスタンスの enqueue メソッドを使用して非同期リクエストを実行します。

このメソッドにはコールバックを渡すことで、リクエストの結果を受け取ります。

```
.enqueue(object : Callback<Weather> {
    // API リクエストが成功したときの処理
    override fun onResponse(call: Call<Weather>, response: Response<Weather>) {

    }

    // API リクエストが失敗したときの処理
    override fun onFailure(call: Call<Weather>, t: Throwable) {
        Log.e("API_ERROR", "Failed to fetch weather data", t) // エラーログを出力
    }
})
```

⑤ レスポンスの処理:

onResponse メソッドは、リクエストが成功したときに呼び出されます。

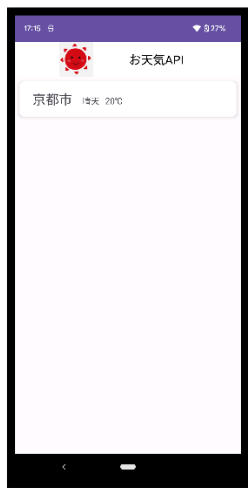
onResponse メソッド内でレスポンスデータ进行处理し、取得した天気データをリストに追加しましょう。

```
response.body()?.let { weather ->
    weatherList.add(weather) // 取得した天気情報をリストに追加
}
```

取得が完了すれば RecyclerView にアダプターをセットするプログラムを追記しましょう。

```
binding.rvCitylist.adapter = WeatherAdapter(weatherList){ }
```

現時点で実行し、動作を確認します。



現時点では Kyoto を指定しているので、京都の天気情報を取得しリストに表示できています。

更に、クリックするとその地域の天気の詳細情報の Fragment へ遷移するように追記しましょう。

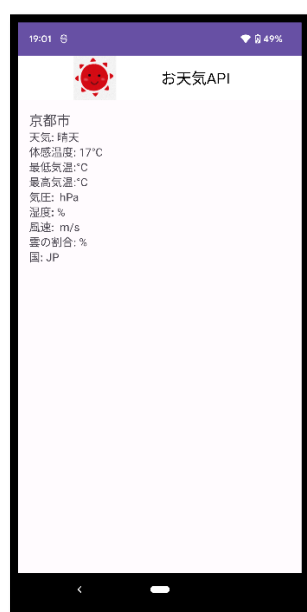
```
selectedWeather ->

    // 各都市のアイテムをクリックした時の動作を定義
    // 次の Fragment へ渡すデータを格納
    parentFragmentManager.setFragmentResult(
        REQUEST_WEATHER_DETAIL,
        bundleOf(SELECTED_WEATHER to selectWeather)
    )

    // 画面遷移
    parentFragmentManager.beginTransaction()

        .replace(fragmentContainer の ID, 遷移する fragment のインスタンス) // 新しいフラグメントに切り替える
        .addToBackStack(null) // バックスタックに追加して、戻るボタンで前のフラグメントに戻れるようにする
        .commit() // 変更を確定する
```

再度実行し、クリックすると詳細画面へ遷移し、情報が一部表示されていることを確認



注意点

ネットワークリクエストは時間がかかる場合があります。そのため、UI スレッドで直接リクエストを実行することは避け、非同期でのリクエストが推奨されます。

レスポンスデータは、定義したデータモデルクラスに自動的にマッピングされます。

なぜこれが必要か

API からデータを取得するためには、適切なリクエストを送信し、レスポンスを受け取る必要があります。Retrofit はこのプロセスを簡単にするツールを提供してくれます。