Retrofit Method Types and Annotations

Talha Çalışır
3 min read · Just now

What is Retrofit ?

Retrofit is a popular HTTP client library for Android that makes it easy to connect to RESTful web services. It typically uses JSON format for data exchange. When making HTTP requests with Retrofit, we use various HTTP methods: GET, POST, PUT, and DELETE. This article will explain what these methods are and their typical use cases.

Get Method:

What is it? The GET method is used to retrieve data from the server. This type of request is generally used for read-only operations and does not change any data on the server.

Use Cases:

- Fetching user information

- Retrieving a list of products
- Listing news articles

Retrofit Example:

```
interface ApiService {
    @GET("users/{userId}")
    fun getUser(@Path("userId") userId: Int): Call<User>
}
```

In this example, the `getUser` method is used to fetch information about a specific user.

POST Method:

What is it? The POST method is used to send new data to the server and create a new resource. This is commonly used for creating new records.

Use Cases:

- Registering a new user
- Adding a new product
- Submitting form data

Retrofit Example:

```
interface ApiService {
    @POST("users")
    fun createUser(@Body newUser: User): Call<User>
}
```

In this example, the `createUser` method is used to create a new user and send the `newUser` object to the server.

PUT Method:

What is it? The PUT method is used to update existing data on the server. This method generally replaces the entire resource.

Use Cases:

- Updating user information
- Modifying product details
- Replacing a record

Retrofit Example:

```
interface ApiService {
    @PUT("users/{userId}")
    fun updateUser(@Path("userId") userId: Int, @Body updatedUser: User): Call<User
}
```

In this example, the `updateUser` method is used to update a specific user and send the `updatedUser` object to the server.

DELETE Method:

What is it? The DELETE method is used to remove existing data from the server. This method deletes the specified resource.

Use Cases:

- Deleting a user account
- Removing products
- Clearing records

Retrofit Example:

```
interface ApiService {
    @DELETE("users/{userId}")
```

```
    fun deleteUser(@Path("userId") userId: Int): Call<Void>
}
```

In this example, the `deleteUser` method is used to delete a specific user.

Summary:

Using HTTP methods (GET, POST, PUT, DELETE) with Retrofit facilitates effective communication with RESTful web services in Android applications. Each method has specific functions and use cases. GET is for reading data, POST is for creating data, PUT is for updating data, and DELETE is for removing data. By correctly using these methods, you can manage server interactions in your applications efficiently and smoothly.

We continute with Commonly Used Annotations in Retrofit: Usage and Explanations

Body:

An object can be specified for use as an HTTP request body with the `@Body` annotation.

```
@POST("users/new")
Call<User> createUser(@Body User user);
```

FORM ENCODED AND MULTIPART:

Methods can also be declared to send form-encoded and multipart data.

Form-encoded data is sent when `@FormUrlEncoded` is present on the method. Each key-value pair is annotated with `@Field` containing the name and the object providing the value.

```
@FormUrlEncoded
@POST("user/edit")
Call<User> updateUser(@Field("first_name") String first, @Field("last_name") String
```

Multipart:

Multipart requests are used when `@Multipart` is present on the method. Parts are declared using the `@Part` annotation.

```
@Multipart
@PUT("user/photo")
Call<User> updateUser(@Part("photo") RequestBody photo, @Part("description") Reques
```

Multipart parts use one of `Retrofit` 's converters or they can implement `RequestBody` to handle their own serialization.

HEADER:

```
@Headers("Cache-Control: max-age=640000")
@GET("widget/list")
Call<List<Widget>> widgetList();
```

```
@Headers({
    "Accept: application/vnd.github.v3.full+json",
    "User-Agent: Retrofit-Sample-App"
})
@GET("users/{username}")
Call<User> getUser(@Path("username") String username);
```

Note that headers do not overwrite each other. All headers with the same name will be included in the request.

A request Header can be updated dynamically using the `@Header` annotation. A corresponding parameter must be provided to the `@Header` . If the value is null, the header will be omitted. Otherwise, `toString` will be called on the value, and the result used.

```
@GET("user")
Call<User> getUser(@Header("Authorization") String authorization)
```

Similar to query parameters, for complex header combinations, a `Map` can be used.

```
@GET("user")
Call<User> getUser(@HeaderMap Map<String, String> headers)
```

Thank you. I referenced from original Retrofit page.

Retrotif Android Mobile Library Goggle

Edit profile

Written by Talha Çalışır

3 Followers

Software Engineer, Violinist. Androiddo Enthusiast, Action Figure Collector

More from Talha Çalışır