Consuming RESTful WebService

Jigisha Patel

Agenda

- Understanding RESTful Web Service
- Overview of RetroFit
- Overview of Gson library
- Consuming RESTful Web Service

REpresentational State Transfer (REST)ful Web Service

- REST is an architecture style for distributed hypermedia systems.
- The REST architecture is client/server, cacheable, stateless and layered.

- The Web services that are built on REST architecture are considered as lightweight, scalable and maintainable.
- The RESTful web service provides a secure mechanism to the application for consuming API in stateless manner.

RESTful Methods

- GET – retrieve all objects provided by API
- POST – create new object
- PUT – update all objects
- DELETE – delete all objects

Retrofit

- Retrofit is a type-safe HTTP client for Android and Java.
- Retrofit is a REST client library that will establish a network connection and communicate with the server, and then receive and parse the response data into a format the app can use.

- Retrofit includes built-in support for popular web data formats such as XML and JSON.

The network layer

- Retrofit creates a network API for the app based on the content from the web service.
- It fetches data from the web service and routes it through a separate converter library that knows how to decode the data and return it in the form of useful objects.

- Retrofit creates most of the network layer including critical details such as running the requests on background threads.

The converter library - Gson

- The converter library helps Retrofit to convert the data it gets from the web service into user-defined object.

- Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object.

Add dependencies for Retrofit and Gson

- In your app level gradle, build.gradle (Module: app), add the following dependency for Retrofit and Gson.

```
implementation "com.squareup.retrofit2:retrofit:$retrofitVersion"
implementation "com.squareup.retrofit2:converter-gson:$gsonVersion "
```

where $retrofitVersion is the version of the retrofit you want to use in the application such as 3.0.0.

Gson built-in type adapters

- Gson has built-in support for reading and writing Java's core data types:
  - Primitives (int, float, char...)
  - Wrappers (Integer, Float, Character...)
  - Arrays, Collections, Lists, Sets, and Maps
  - Strings

The model class

- To parse the JSON data and converts it into Java objects, a model class needs to be created to store the parsed results.
- When Gson parses the JSON, it matches the keys by name and fills the data objects with appropriate values.

- If the keys in the JSON objects are different than the data class properties, you can use @SerializedName annotation to specify the JSON object keys.
- This is useful when the JSON object keys contains spaces or other characters that aren't permitted in Java field names.
- You can create individual data class object for nested objects.

Web API service

- As a part of your network layer, create an interface that specifies how Retrofit should connect to the web server using HTTP requests.
- You can configure the dynamic URLs by using @Query annotations or by sending URL as a parameter along with the method call.

- You can use @Get(“.”) annotation with method call to indicate usage of only base url (without endpoints added) to fetch the data.

Retrofit & Gson object

- Create instance of Retrofit and Gson library to start using the network services and converting JSON string into Java objects.

```
Gson gson = new GsonBuilder().create();

Retrofit retrofit = new Retrofit.Builder()
        .baseUrl(api.BASE_URL)
        .addConverterFactory(GsonConverterFactory.create(gson))
        .build();

api = retrofit.create(Api.class);
```

Retrofit Service

- Use the Retrofit create() method to initialize the Retrofit service.
- As the service calls are expensive, you should use singleton instance of the service such that the Retrofit service instance is created only once and exposed to the app using public object.

```
public static synchronized RetrofitClient getInstance() {
    if (instance == null) {
        instance = new RetrofitClient();
    }
    return instance;
}
```

Initiating retrieval

- Once the Retrofit service is ready, you can start reading responses.

```
Call<Person> call = RetrofitClient.getInstance().getMyApi().retrieveResponse();

try {
   call.enqueue(new Callback<Person>() {
      @Override
      public void onResponse(Call<Person> call, Response<Person> response) {
         //TODO on successful response
      }
      @Override
      public void onFailure(Call<Person> call, Throwable t) {
         //TODO on unsuccessful response
         Log.e(TAG, "An error has occured" + t.getLocalizedMessage());
      }
   });
}catch (Exception ex) {
   Log.e(TAG, ex.getLocalizedMessage());
}
```

Call interface

- The call interface initiates an invocation of a Retrofit method that sends a request to a webserver and returns a response.
- Each call yields its own HTTP request and response pair.
- Calls may be executed synchronously with execute(), or asynchronously with enqueue(retrofit.Callback<T>).
- In either case the call can be canceled at any time with cancel().
- A call that is busy writing its request or reading its response may receive a IOException.

References

- https://restfulapi.net/
- https://square.github.io/retrofit/
- https://github.com/google/gson