Universidad Nacional de La Matanza

Serialización

Agenda

- Introducción
- Formatos: XML, JSON
- Gson
- JSONEncoder/JSONDecoder
- Kotlin Serialization

Serialización

- Serialización es el proceso por el cual un objeto se transforma a un formato que se pueda transferir o almacenar.
- Su objetivo es guardar el estado de un objeto para poder volver a crearlo cuando sea necesario -> **deserialización**.
- Existen múltiples formatos: XML, JSON, Protocol buffers.

Serialización

- Ejemplo XML

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<root>
    <libro>
        <isbn>9781718502680</isbn>
        <titulo>Androids: The Team that Built the Android Operating System</titulo>
        <autor>Chet Haase</autor>
    </libro>
    <libro>
        <isbn>9788427214927</isbn>
        <titulo>Sherlock Holmes: Estudio en escarlata</titulo>
        <autor>Arthur Conan Doyle</autor>
    </libro>
</root>
```

Serialización

- Ejemplo JSON

```
{
   "libros":[
      {
         "isbn":9781718502680,
         "titulo":"Androids: The Team that Built the Android Operating System",
         "autor":"Chet Haase"
      },
      {
         "isbn":9788427214927,
         "titulo":"Sherlock Holmes: Estudio en escarlata",
         "autor":"Arthur Conan Doyle"
      }
   ]
}
```

Serialización

"Nos vamos a quedar con el formato JSON, dado que es el más usado en comunicaciones REST y aplicaciones móviles"

Estado del arte

- En Android tenemos varias opciones, muchas heredadas de Java, pero las más populares son Gson, Moshi, Jackson.
- En iOS, se utilizan JSONDecoder y JSONEncoder

Estado del arte

- Ejemplo de uso de Gson en Android: Serializar

```
class Libro(
    @SerializedName("isbn")
    val id: String,
    @SerializedName("titulo")
    val titulo: String,
    @SerializedName("autor")
    val autor: String
)
```

```
val gson = Gson()
val libro = Libro(
    id = "9781718502680",
    titulo = "Androids: The Team that Built " +
            "the Android Operating System",
    autor = "Chet Haase"
)
val json = gson.toJson(libro)
println(json)
```

```
>> {"isbn":"9781718502680","titulo":"Androids: The Team that Built the Android Operating System","autor":"Chet Haase"}
```

Estado del arte

- Ejemplo de uso de Gson en Android: Deserializar

```
class Libro(
    @SerializedName("isbn")
    val id: String,
    @SerializedName("titulo")
    val titulo: String,
    @SerializedName("autor")
    val autor: String
)
```

```
val gson = Gson()
val json = """
    {
       "isbn" : "9781718502680",
       "titulo" : "Androids: The Team that Built the Android Operating System",
       "autor" : "Chet Haase"
    }"""
val libro = gson.fromJson(json, Libro::class.java)

println("El libro \"${libro.titulo}\" fue escrito por ${libro.autor}.")
```

```
>> El libro "Androids: The Team that Built the Android Operating System" fue escrito por Chet Haase.
```

Estado del arte

- Ejemplo de uso de JSONEncoder en iOS: Serializar

```
struct Libro: Encodable {
    var id: String
    var titulo: String
    var autor: String

    enum CodingKeys: String, CodingKey {
        case id = "isbn"
        case titulo, autor
    }
}
```

```
let libro = Libro(
    id: "9781718502680",
    titulo: "Androids: The Team that Built " +
            "the Android Operating System",
    autor: "Chet Haase"
)

let encoder = JSONEncoder()
encoder.outputFormatting = .prettyPrinted

let json = try encoder.encode(libro)
print(String(data: json, encoding: .utf8)!)
```

```
{
  "autor" : "Chet Haase",
  "titulo" : "Androids: The Team that Built the Android Operating System",
  "isbn" : "9781718502680"
}
Program ended with exit code: 0
```

Estado del arte

- Ejemplo de uso de JSONDecoder en iOS: Deserializar

```
struct Libro: Decodable {
    var id: String
    var titulo: String
    var autor: String

    enum CodingKeys: String, CodingKey {
        case id = "isbn"
        case titulo, autor
    }
}
```

```
let decoder = JSONDecoder()
let json = """
 {
    "isbn" : "9781718502680",
    "titulo" : "Androids: The Team that Built the Android Operating System",
    "autor" : "Chet Haase"
 }
""".data(using: .utf8)!
let libro = try decoder.decode(Libro.self, from: json)

print("El libro \"\(libro.titulo)\" fue escrito por \(libro.autor).")
```

```
>> El libro "Androids: The Team that Built the Android Operating System" fue escrito por Chet Haase.
Program ended with exit code: 0
```

Kotlin Serialization

Kotlin Serialization

- Es una librería de serialización y deserialización propiedad de Jetbrains
- Es multiplataforma: JVM, JavaScript y Native
- Soporta múltiples formatos: JSON, Protobuf, CBOR, Hocon and Properties
- Consiste de un plugin de compilador que genera código para las clases serializables y una librería en runtime que proporciona una API sencilla

Kotlin Serialization

- Integración

En build.gradle del módulo donde se va a usar

```
plugins {
    ...
    kotlin("plugin.serialization") version "1.6.10"
}


...
dependencies {
    implementation(“io.ktor:ktor-serialization-kotlinx-json:2.0.0-beta-1")
}
```

Kotlin Serialization

- Creamos la clase utilizando las annotations

```
@Serializable
class Libro(
    @SerialName("isbn")
    val id: String,
    @SerialName("titulo")
    val titulo: String,
    @SerialName("autor")
    val autor: String
)
```

Kotlin Serialization

- Deserialización

```
val json = """
    {
       "isbn" : "9781718502680",
       "titulo" : "Androids: The Team that Built the Android Operating System",
       "autor" : "Chet Haase"
    }"""
val libro = Json.decodeFromString<Libro>(json)

println("El libro \"${libro.titulo}\" fue escrito por ${libro.autor}.")
```

Kotlin Serialization

- Serialización

```
val libro = Libro(
       id = "9781718502680",
       titulo = "Androids: The Team that Built the Android Operating System",
       autor = "Chet Haase"
)
val json = Json.encodeToString(libro)

println(json)
```

¿Dónde sigo aprendiendo?

- https://kotlinlang.org/docs/serialization.html
- https://github.com/Kotlin/kotlinx.serialization

Integración con APIs

Agenda

- Introducción
- Estado del arte
  - Retrofit
  - Alamofire
- Ktor
- Anexo: Logging

Estado del arte

Android

- La más popular es Retrofit (Square)
- Existen también HttpUrlConnection (java.net) y Volley (Google)

iOS

- La más popular es Alamofire
- Existe también URLSession (iOS, reemplaza a NSURLConnection)

Ktor

Ktor

- Cliente HTTP para crear *requests* y *responses* asíncronos.
- Hecho por JetBrains.
- 100% Kotlin puro.
- Altamente configurable.
- Modelo de *Plugins* instalables en el cliente.
  - Json
  - Cookies
  - Timeout
  - Logging
  - Authentication & Authorization
  - etc...

Ktor

- Dependencia para **commonMain**
  `implementation("io.ktor:ktor-client-core:$ktorVersion")`
- Dependencia para **androidMain**
  `implementation("io.ktor:ktor-client-okhttp:$ktorVersion")`
- Dependencia para **iosMain**
  `implementation("io.ktor:ktor-client-ios:$ktorVersion")`

Ktor

- Como vamos a usar también Kotlin Serialization como serializador/deserializador junto a Ktor, deberemos agregar otras 2 dependencias en **commonMain**:

```
implementation("io.ktor:ktor-client-content-negotiation:$ktorVersion")

implementation("io.ktor:ktor-serialization-kotlinx-json:$ktorVersion")
```

Ktor

- Primero deberemos crear las clases de mapeo de la respuesta del servicio REST a Kotlin

```
@Serializable
class WeatherResponse (
    @SerialName("current")
    val current: CurrentWeatherResponse,
    @SerialName("location")
    val location: LocationResponse
)

@Serializable
class LocationResponse (
    @SerialName("country")
    val country: String,
    @SerialName("region")
    val region: String,
    @SerialName("name")
    val name: String
)

...
```

Ktor

- Se debe crear un objeto del tipo **HttpClient**, el cual puede configurarse a gusto (mediante sus plugins)

```
val httpClient = HttpClient {
    install(ContentNegotiation) {
        json(
            Json {
                ignoreUnknownKeys = true
            }
        )
    }
}
```

Ktor

- Crear un **HttpResponse** a partir de llamar funciones sobre el objeto **HttpClient** creado anteriormente.

```
suspend fun obtenerClima(): WeatherResponse {
    val weatherResponse = httpClient.get("https://api.weatherapi.com/v1/current.json")
{
        parameter("q", "Buenos Aires")
        parameter("key", "api-key")
    }.body<WeatherResponse>()
    return weatherResponse
}
```

- Las funciones que realizan *requests* son **suspend functions… ¡¡¡coroutines!!!**

Ktor en Android

- No olvidar de agregar el permiso para utilizar Internet en el **AndroidManifest** correspondiente.
- También harán falta dependencias para poder utilizar las **kotlin coroutines**. (*androidApp/build.gradle.kts*)

```
val coroutinesVersion = "1.6.0"
implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:$coroutinesVersion")
implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:$coroutinesVersion")
```

Anexo: Logging

Logging

- Por lo general en las apps nos interesa poder ver qué viaja en cada *request*/*response* para poder debuggear el funcionamiento de nuestras apps.
- Ktor permite instalar un *plugin* de *logging* y usarlo junto con alguna otra librería multiplataforma que se encargue del código necesario en iOS y Android, según corresponda. Usaremos **Napier**.

Dependencias

```
// Plugin de ktor
implementation("io.ktor:ktor-client-logging:$ktorVersion")

// Librería de logging multiplataforma
implementation("io.github.aakira:napier:2.6.1")
```

Configurando el cliente

```
val httpClient = HttpClient {
    install(Logging) {
        level = LogLevel.ALL
        logger = object : Logger {
            override fun log(message: String) {
                Napier.v(tag = "HttpClient", message = message)
            }
        }
        logger
    }
}.also {
    initLogger()
}
```

Configurando las apps Android/iOS

- Usaremos el patrón *expect*/*actual*, del que ya hablamos las primeras clases, para usar **DebugAntilog**.
- A pesar de ser el mismo código, esta clase de **Napier** se encarga de detectar la plataforma en la que se ejecuta y hacer lo necesario para loggear en la consola.

¿Dónde sigo aprendiendo?

- https://ktor.io/
- https://github.com/AAkira/Napier
- https://kotlinlang.org/docs/multiplatform-mobile-concurrency-and-coroutines.html

Fin