JSON

Seminario de lenguajes Android

Introducción

JSON (acrónimo de JavaScript Object Notation) es un formato de texto para la definición de datos. El objetivo principal de JSON es el de definir una sintaxis mediante la cual sea posible representar diferentes estructuras de datos.

Una característica importante de JSON es que, al ser un formato de texto, es totalmente manipulable por un ser humano y no es un formato que solo puede ser interpretado por una máquina.

Un ejemplo sencillo de una estructura expresada en JSON se puede ver a continuación:

```
{
	“nombre”: “Mick”,
	“apellido”: “Jagger”,
	“rol”: “Cantante”
}
```

Tipos de datos

JSON provee un conjunto reducido de tipos de datos que pueden ser utilizados para generar estructuras más complejas: Números, Cadenas, Booleanos, null, Array y Objetos

A continuación se observa un ejemplo de un Objeto JSON, donde cada campo pertenece a un tipo de datos distinto:

```
{
	“numero”: 10,
	“cadena”: “texto”,
	“booleano”: true,
	“arreglo”: [1,2,3],
	“nulo”: null,
	“objeto_vacio”: {},
	“obeto_con_contenido”: {
		“contenido”: true
		“numero”: 5
	}
}
```

¿Cuántos Objetos JSON pueden observarse en el JSON del ejemplo?

¿Por qué JSON?

En general, cuando se trabaja con Android, los objetos viven en memoria, más específicamente en la Heap. Sin embargo, en ciertas situaciones necesitamos guardar la información que contienen estos objetos en un medio persistente, o incluso enviarlas a través de la red a otro dispositivo.

Para llevar a cabo esta tarea, es necesario codificar los objetos que están en memoria en algún formato que pueda ser utilizado en mecanismos de persistencia o de intercambio de información en redes.

JSON es una alternativa estandarizada, de gran difusión y con amplio soporte en diferentes lenguajes y plataformas. Java, Kotlin y Android en particular proveen soporte para la generación e interpretación de estructuras de datos descriptas en JSON.

Android y JSON

Existen numerosas situaciones en donde necesitamos persistir un objeto en una aplicación Android.

Un caso muy común es el que se produce al modificar la orientación del dispositivo, en donde la Activity se destruye y, por lo tanto, Android nos provee un mecanismo para almacenar la información que no queremos que se pierda en el proceso de destrucción y construcción de la nueva Activity.

Sin embargo, este mecanismo de almacenamiento, solo nos permite almacenar tipos de datos simples como cadenas de texto o números.

Pero, ¿qué sucede si queremos almacenar un objeto completo?

En este caso, una buena alternativa es representar nuestro objeto en forma de texto para poder almacenarlo como una cadena de texto. Para ello, codificar nuestro objeto como JSON se convierte en una buena alternativa.

Otro caso similar, es el del almacenamiento de preferencias (SharedPreferences). Este mecanismo de persistencia nuevamente solo nos permite almacenar tipos de datos simples.

Generación de un objeto JSON

Android provee la clase org.json.JSONObject, que nos permite construir un objeto JSON en memoria para luego convertirlo en un String con formato JSON.

El uso de esta clase puede entenderse fácilmente a través de un ejemplo.

```
var jsonObject =  JSONObject()
jsonObject.put("numero", 10)
jsonObject.put("cadena", "mi cadena")
jsonObject.put("valor_null", null)
```

JSONObject es un objeto Java, el cual se encuentra almacenado en memoria. Pero podemos obtener una versión textual en notación JSON de este objeto simplemente invocando el método toString():

```
val jsonString = jsonObject.toString()
```

La variable jsonString contendrá una cadena de la siguiente forma:

```
{
	“numero”: 10,
	“cadena”: “mi_cadena”,
	“valor_null”: null
}
```

Esa cadena podrá ser almacenada o enviada a través de la red como cualquier String. Lo que resta saber ahora es cómo hacer el proceso inverso.

Generación de un objeto JSON a partir de un String

Supongamos que dentro de un archivo de texto tenemos un objeto en formato JSON como el siguiente:

```
{
	“nombre”: “Mick”,
	“apellido”: “Jagger”,
	“rol”: “Cantante”,
	“edad”: 75
}
```

Si obtenemos el contenido del archivo, podríamos entonces haber levantado esa información en una variable de tipo String:

```
val contenido: String = this.leerArchivo()
```

Ahora, supongamos que queremos obtener el valor del atributo “rol”. Esto implicaría procesar de alguna manera el String para poder extraer el valor de dicho atributo.

La clase JSONObject hace esto por nosotros:

```
var jsonObject =  JSONObject(contenido)
var rol = jsonObject.getString("rol")
```

La primer línea se encarga de procesar el String, mientras que la segunda nos devuelve el contenido del atributo “rol” como un String.

Si quisiéramos obtener el valor de un atributo con otro tipo de datos, podemos utilizar los métodos correspondientes. Ejemplo:

```
var edad = jsonObject.getInt("numero")
```

Un ejemplo más completo

Veamos un ejemplo más completo en donde nos encargaremos de generar una cadena JSON a partir de un objeto Java y luego realizaremos el proceso inverso.

Sean las clases java:

```
class Banda {
   var nombre: String? = null
   lateinit var integrantes: Array<Integrante>
}

class Integrante {
   var nombre: String? = null
   var apellido: String? = null
   var edad = 0
}
```

Construimos un conjunto de datos:

```
var i1 =  Integrante();
i1.nombre = "Mick";
i1.apellido = "Jagger";
i1.edad = 75;

var i2 =  Integrante();
i2.nombre = "Keith";
i2.apellido = "Richards";
i2.edad = 75;

var i3 =  Integrante();
i3.nombre = "Ronnie";
i3.apellido = "Wood";
i3.edad = 72;

var i4 =  Integrante();
i4.nombre = "Charlie";
i4.apellido = "Watts";
i4.edad = 78;

var b =  Banda();
b.nombre = "The Rolling Stones";
b.integrantes = arrayOf<Integrante>(i1, i2, i3, i4)
```

Ahora generaremos los objetos JSON para poder convertir fácilmente toda esta información en un String JSON válido.

```
var bandaJson =  JSONObject();
bandaJson.put("nombre", b.nombre);

var integrantesJson =  JSONArray();
for ( i in 0 .. b.integrantes.size) {
   var integranteJson =  JSONObject();
   integranteJson.put("nombre", b.integrantes[i].nombre);
   integranteJson.put("apellido", b.integrantes[i].apellido);
   integranteJson.put("edad", b.integrantes[i].edad);

   integrantesJson.put(i, integranteJson);
}

bandaJson.put("integrantes", integrantesJson);
```

(Notar el uso de JSONArray para definir un arreglo)

Ahora que tenemos nuestro objeto JSON, podemos generar el String con formato JSON utilizando el método toString:

```
val cadenaJson = bandaJson.toString()
```

El contenido de la variable será el siguiente:

```
{
      “nombre”: “The Rolling Stones”,
      “integrantes”: [
            {
                  “nombre”: “Mick”,
                  “apellido”: “Jagger”,
                  “edad”: 75
            },
            {
                  “nombre”: “Keith”,
                  “apellido”: “Richards”,
                  “edad”: 75
            },
            {
                  “nombre”: “Ronnie”,
                  “apellido”: “Wood”,
                  “edad”: 72
            },
            {
                  “nombre”: “Charlie”,
                  “apellido”: “Watts”,
                  “edad”: 78
            }
      ]
}
```

El proceso inverso es análogo y se muestra a continuación:

```
val bandaJson = JSONObject(cadenaJson)
val integrantesJson = bandaJson.getJSONArray("integrantes")

val banda = Banda()
banda.nombre = bandaJson.getString("nombre")
banda.integrantes = arrayOfNulls<Integrante>(integrantesJson.length())

for (i in 0 .. integrantesJson.length()) {
   val integranteJson = integrantesJson.getJSONObject(i)
   val integrante = Integrante()
   integrante.nombre = integranteJson.getString("nombre")
   integrante.apellido = integranteJson.getString("apellido")
   integrante.edad = integranteJson.getInt("edad")
   banda.integrantes[i] = integrante
}
```