JSON

Tiempo estimado: 15min

El tipo JSON, utilizado ampliamente hoy en día, no es suministrado nativamente por Redis, pero se puede usar mediante el módulo ReJSON. El objeto de la presente lección.

La lección comienza introduciendo el formato JSON. A continuación, se presenta el módulo ReJSON y cómo usarlo. Y finalmente, se lista los comandos proporcionados por este módulo.

Al finalizar la lección, el estudiante sabrá:

- Cómo compilar el módulo ReJSON.
- Cómo utilizar pares clave-valor de tipo JSON.
- Cómo acceder y modificar campos individuales de una clave JSON.

Introducción

Un formato de datos (*data format*) no es más que una manera de representar un objeto o registro de datos como, por ejemplo, una instancia de una clase C++, C#, Java, JavaScript o Python.

JSON (*JavaScript Object Notation*, Notación de Objetos de JavaScript), pronunciado *yeison*, es un formato para representar estructuras de datos mediante texto. Otro formato muy conocido es XML, pero éste lo hace mediante el uso de etiquetas. Ambos son formatos de representación de datos, pero cada uno de ellos lo hace de una manera distinta. Vamos a ilustrarlo mediante un ejemplo:

```
//JSON
{ "nombre": "The National",
  "añoFormación": 1999,
  "origen": "Cincinnati, OH; Brooklyn, NY",
  "géneros": ["Indie", Post-punk"],
  "sitioWeb": "americanmary.com",
  "miembros": [
    { "nombre": "Matt Berninger", "rol": "vocalista" },
    { "nombre": "Aaron Dessner", "rol": "guitarista, teclista" },
    { "nombre": "Bryce Dessner", "rol": "guitarrista, teclista" },
    { "nombre": "Bryan Devendorf", "rol": "batería" },
    { "nombre": "Scott Devendorf", "rol": "bajista" }
  ]
}

<!-- XML -->
<artista>
  <nombre>The National</nombre>
  <añoFormación>1999</añoFormación>
  <origen>Cincinnati, OH; Brooklyn, NY</origen>
  <géneros>
    <género>Indie</género>
    <género>Post-punk</género>
  </géneros>
  <sitioWeb>americanmary.com</sitioWeb>
  <miembros>
    <miembro>
      <nombre>Matt Berninger</nombre>
      <rol>vocalista</rol>
    </miembro>
    <miembro>
      <nombre>Aaron Dessner</nombre>
      <rol>guitarrista, teclista</rol>
    </miembro>
    <miembro>
      <nombre>Bryce Dessner</nombre>
```



```
      <rol>guitarrista, teclista</rol>
    </miembro>
    <miembro>
      <nombre>Bryan Devendorf</nombre>
      <rol>batería</rol>
    </miembro>
    <miembro>
      <nombre>Scott Devendorf</nombre>
      <rol>bajista</rol>
    </miembro>
  </miembros>
</artista>
```

Como puede observar, la misma información se puede representar en formatos distintos, según las preferencias de cada uno.

Un objeto JSON (*JSON object*) es un objeto formado por un conjunto ordenado de campos, en forma de pares clave-valor. Cada campo (*field*), como acabamos de ver, es un par clave-valor, donde la clave (*key*) identifica el campo y es siempre una cadena de texto y su valor (*value*) puede ser otro objeto JSON, una cadena de texto, un número, un *array*, un valor booleano o un valor nulo.

Su sintaxis es muy parecida a la de los objetos de JavaScript, concretamente es como sigue:

```
ObjetoJSON := { Miembro, Miembro... }
Miembro := Cadena : Valor
Valor := Array | Booleano | Cadena | Nulo | Número | ObjetoJSON
```

Tal como puede observar del ejemplo anterior y de la sintaxis ahora definida, JSON define un formato de representación claro y de fácil comprensión debido a su sintaxis textual. Requiere poca codificación y, por ende, poco procesamiento por parte del software. Se puede usar, pues, tanto para almacenar objetos de datos en disco como para seriarlos y transmitirlos a través de la red.

JSON se encuentra normalizado y su especificación se encuentra disponible en json.org. Es independiente de JavaScript. Y se puede utilizar con otros lenguajes de programación como, por ejemplo, Java, C#, C++, etc.

Módulo ReJSON

El tipo JSON es similar a los *arrays* asociativos, pero con valores de campo no necesariamente de tipo cadena. Es muy útil y utilizado hoy en día. Incluso existe motores de bases de datos cuyo formato de almacenamiento es JSON.

Para poder trabajar con este tipo de datos en Redis, podemos utilizar el módulo ReJSON que se encuentra disponible en redismodules.com. Y como no puede ser de otra manera en Redis, para usarlo hay que descargar su código fuente, compilarlo y cargarlo.

Descarga del código fuente

La manera más fácil de descargarlo es mediante git:

```
$ git clone --depth 1 https://github.com/RedisLabsModules/rejson.git
```

Compilación

Antes de compilarlo, hay que instalar sus dependencias:

```
$ sudo apt install -y build-essential cmake
```

A continuación, hay que ejecutar el *script* bootstrap.sh:

```
$ cd rejson
$ ./bootstrap.sh
```

Y finalmente, compilar:

```
$ cmake --build build --target rejson
```

Si todo ha ido bien, encontrará el módulo rejson.so en la carpeta lib.

Copia del módulo

Lo siguiente es copiar el módulo en el directorio de módulos de la instancia:


```
$ cp lib/rejson.so /opt/redis/modules/
```

Carga del módulo

Finalmente, hay que cargarlo mediante el parámetro de configuración loadmodule o el comando MODULE LOAD:

```
127.0.0.1:6379> MODULE LOAD /opt/redis/modules/rejson.so
OK
127.0.0.1:6379> MODULE LIST
1) 1) "name"
   2) "ReJSON"
   3) "ver"
   4) (integer) 1
127.0.0.1:6379>
```

Creación de par clave-valor

Para crear un par clave-valor de tipo JSON, hay que utilizar el comando JSON.SET como sigue:

```
JSON.SET clave . "valor-JSON"
```

He aquí un ejemplo ilustrativo:

```
127.0.0.1:6379> JSON.SET bands:morrissey . '{"name": "Morrissey", "origin": "UK"}'
OK
127.0.0.1:6379> JSON.GET bands:morrissey
"{\"name\":\"Morrissey\",\"origin\":\"UK\"}"
127.0.0.1:6379>
```

Consulta del valor de un par clave-valor

Para consultar el valor de un par clave-valor, hay que utilizar el comando JSON.GET:

```
JSON.GET clave
```

Ejemplo:

```
127.0.0.1:6379> JSON.GET bands:morrissey
"{\"name\":\"Morrissey\",\"origin\":\"UK\"}"
127.0.0.1:6379>
```

Para acceder a varias claves al mismo tiempo, se puede usar JSON.MGET:

```
JSON.MGET clave clave clave…
```

Consulta de campos

Si necesitamos consultar el valor de un determinado campo, se puede utilizar los comandos JSON.GET y JSON.MGET, usando las siguientes sintaxis:

```
JSON.GET clave campo
JSON.MGET clave clave… campo
```

Veamos un ejemplo ilustrativo:

```
127.0.0.1:6379> JSON.GET bands:morrissey origin
"\"UK\""
127.0.0.1:6379>
```

Para acceder a un subcampo, hay que usar cualquiera de las siguientes sintaxis:

```
campo.subcampo
campo["subcampo"]
```

Existe otros comandos para consultar aspectos relacionados con campos:

```
JSON.TYPE clave campo
JSON.STRLEN clave campo
JSON.ARRINDEX clave campo valor [índiceDeInicio [índiceDeFin]]
JSON.ARRLEN clave campo
JSON.OBJKEYS clave
JSON.OBJKEYS clave campo
JSON.OBJLEN clave
JSON.OBJLEN clave campo
```

El comando JSON.TYPE devuelve el tipo de datos del valor de un campo. JSON.STRLEN, la longitud de un

campo de tipo cadena, mientras que JSON.ARRLEN de un *array* y JSON.OBJLEN el número de campos. JSON.ARRINDEX el índice de un elemento de un campo de tipo *array*. Y JSON.OBJKEYS los nombres de los campos.

Modificación de campos

Para modificar el valor de un campo, hay que utilizar el comando JSON.SET como sigue:

```
JSON.SET clave campo nuevoValor
JSON.SET clave campo nuevoValor NX|XX
```

La cláusula XX se utiliza para restringir la operación a aquellos casos donde existe ya la clave; si no existe, no hará nada. Mientras que NX, a aquellos donde la clave no debe existir.

Ejemplo:

```
127.0.0.1:6379> JSON.GET bands:morrissey origin
"\"UK\""
127.0.0.1:6379> JSON.SET bands:morrissey origin '"England"'
OK
127.0.0.1:6379> JSON.GET bands:morrissey
"{\"name\":\"Morrissey\",\"origin\":\"England\"}"
127.0.0.1:6379>
```

No olvide que el nuevo valor debe estar en formato JSON. Si no lo está, la operación fallará como, por ejemplo, tal como se muestra a continuación:

```
127.0.0.1:6379> JSON.SET bands:morrissey origin England
(error) ERR JSON lexer error SPECIAL_EXPECTED at position 1
127.0.0.1:6379>
```

Incremento de campo numérico

Para incrementar un campo numérico, se puede usar:

```
JSON.NUMINCRBY clave campo incremento
```

Multiplicación de un campo numérico

Para actualizar un campo al resultado de su valor por otro, tenemos JSON.NUMMULTBY:

```
JSON.NUMMULTBY clave campo valor
```

Concatenación

El comando JSON.STRAPPEND permite añadir un texto al final de otro:

```
JSON.STRAPPEND clave campo texto
```

Actualización de campos de tipo array

Si necesitamos añadir un valor al final de un *array*, podemos usar JSON.ARRAPPEND:

```
JSON.ARRAPPEND clave campo elemento
```

Para insertar uno o más elementos en una determinada posición, desplazando los que le siguen una posición, tenemos JSON.ARRINSERT:

```
JSON.ARRINSERT clave campo índice elemento elemento…
```

En cambio, para suprimir un determinado número de elementos al final de un *array*:

```
JSON.ARRPOP clave campo
JSON.ARRPOP clave campo númeroElementos
```

Finalmente, para quedarnos con un fragmento del *array*, disponemos de JSON.ARRTRIM:

```
JSON.ARRTRIM clave campo índiceInicial índiceFinal
```

Supresión de campos

Para suprimir un campo, se puede utilizar el comando JSON.DEL:

```
JSON.DEL clave campo
```

