

**FACULTAD DE CIENCIAS E INGENERÍA**

**INGENERIA DE SISTEMAS E INFORMÁTICA**

**CURSO DE:**

**ALGORITMICA II**

**TEMA**

**TRABAJO DE INVESTIGACION/ FINAL**

JASON API PARA JAVA

**DOCENTE:** CORONEL CASTILLO ERIC GUSTAVO

**SECCIÓN :** 501 B

**TURNO :** NOCHE

**ESTUDIANTE :** VILCHEZ FLORES, MELINA DEL ROSARIO

RUIZ RAFAEL, HERNANDO B.

CHOQUE SALVATIERRA, RICARDO

**LOS OLIVOS**

**INDICE**

1. **ANTECEDENTES …………………………………………………………… 3**
2. **HISTORIA Y ESPECIFICACIÓN OFICIAL**
3. **ORIGENES**
4. **RESUMEN ……………………………………………………………………. 4**
5. **DESARROLLO ………………………………………………………………. 5**
6. **INTRODUCCION JSON**
7. **TIPOS DE ESTRUCTURADO (**Objetos y Matrices**)**
8. **PROCESAMIENTOS JSON**

* La API de Java para procesamiento JSON
* La API de modelos de objetos
* La API de streaming
* **Tabla 1**. Clases principales de la API
* **Listado 2.** Representación JSON de una búsqueda en los posts
* **Listado 3.** Procesamiento de posts
* **Tabla 2.** Clases principales de la API de streaming
* **Listado 4.** Procesamiento de posts

1. **CASOS ……………………………………………………………………… 13**
2. **CASO 1. (Ejm. frutería-nombre, objeto, array )**
3. **CASO 2. (Clase Hijo)**
4. **RECOMENDACIÓN ……………………………………………………….. 20**
5. **CONCLUSION ……………………………………………………………... 20**
6. **INFORMACIÓN ADICIONAL …………………………………………….. 21**
7. **BIBLIOGRAFIA ……………………………………………………………. 21**

**ANTECEDENTES**

1. **HISTORIA**

A finales de los años 90 XML era el formato para intercambio de datos con mayor implantación. No obstante, presentaba problemas sobre todo cuando se trataba de trabajar con ficheros con gran volumen de datos donde el procesamiento se volvía lento.

Surgieron entonces intentos para definir formatos que fueran más ligeros y rápidos para el intercambio de información. Uno de ellos fue JSON, promovido y popularizado por Douglas Crockford y sus colaboradores a principios de los años 2000.

JSON se caracteriza por reducir el tamaño de los archivos y el volumen de datos que es necesario transmitir frente a otros estándares como XML. Por ello JSON fue adquiriendo popularidad hasta convertirse en un estándar. Esto no significa que XML haya dejado de utilizarse. En la actualidad se utiliza tanto XML como JSON para el intercambio de datos. Utilizar uno u otro depende de las circunstancias y de las preferencias que en cada momento se determinen.

1. **ORIGENES:**

JSON nació como una alternativa a XML, el fácil uso en JavaScript que generó un gran número de seguidores de esta alternativa.

En sus orígenes JSON estuvo ligado a JavaScript, y de hecho su notación puede decirse que está inspirada en la notación de objetos JavaScript, con el tiempo se ha convertido en un estándar independiente de datos, no ligado a ningún lenguaje en concreto, de la misma forma que XML no está ligado a ningún lenguaje en concreto.

Al igual que un fichero XML es un fichero de texto, un fichero JSON también lo es.

1. **DESARROLLO**

**API de Java para procesamiento JSON**

1. **Introducción a JSON**

JSON (JavaScript Object Notation) es un formato para intercambio de datos liviano, basado en texto e independiente del lenguaje de programación, que resulta fácil de escribir y leer tanto para los seres humanos como para las máquinas.

1. **Tipos de estructurado**

JSON puede representar dos tipos estructurados: **objetos y matrices**.

* Un objeto es una colección no ordenada de cero o más pares de nombres/valores.
* Una matriz es una secuencia ordenada de cero o más valores.
* (Los valores pueden ser cadenas, números, booleanos, nulos y estos dos tipos estructurados)

Ejemplo aplicativo:

**Listado 1.** Ejemplo de representación de un objeto en JSON.

El Listado 1 constituye un ejemplo tomado de Wikipedia que muestra la representación JSON de un objeto que describe a una persona. El objeto tiene valores de cadena para nombre y apellido, un valor numérico para la edad, un valor de objeto que representa el domicilio de la persona y un valor de matriz de objetos de números telefónicos.

{

"firstName": "John",

"lastName": "Smith",

"age": 25,

"address": {

"streetAddress": "21 2nd Street",

"city": "New York",

"state": "NY",

"postalCode": 10021

},

"phoneNumbers": [

{

"type": "home",

"number": "212 555-1234"

},

{

"type": "fax",

"number": "646 555-4567"

}

]

}

JSON suele utilizarse en aplicaciones Ajax, configuraciones, bases de datos y servicios web RESTful. Todos los sitios web populares ofrecen JSON como formato para intercambio de datos con sus servicios web RESTful.

1. **Procesamiento JSON**

**La API de Java para procesamiento JSON** (JSR 353) proporciona rutinas API portátiles que permiten analizar, generar, transformar y consultar JSON usando rutinas API de modelos de objetos y de streaming.

**La API de modelos de objetos** crea una estructura de árbol, de acceso aleatorio, que representa los datos JSON almacenados en la memoria. Es posible recorrer el árbol y formular consultas. Este modelo de programación es el más flexible y posibilita el procesamiento en casos en que se requiera acceso aleatorio a la totalidad del contenido de la memoria. Sin embargo, a menudo no es tan eficiente como el modelo de streaming y requiere más memoria.

**La API de streaming** ofrece un modo de analizar y generar JSON en streams. Le otorga al programador el control sobre el análisis y la generación. La API de streaming ofrece un analizador basado en eventos y brinda al desarrollador de aplicaciones la posibilidad de "pedir" el evento siguiente en lugar de tener que ocuparse del evento en una devolución de llamada. De este modo, el desarrollador cuenta con mayor control procedimental del procesamiento JSON. El código de aplicación puede procesar o descartar el evento del analizador y pedir el siguiente evento (extraer el evento). El modelo de streaming es adecuado para el procesamiento local cuando no se requiere acceso aleatorio a otras porciones de la información. De manera similar, la API de streaming permite generar JSON bien formado en stream escribiendo un evento por vez.

**La API de modelos de objetos** es similar a la API de modelos de objetos de documento (DOM) para XML. Es una API de alto nivel que proporciona modelos de objetos inmutables para estructuras de objetos y matrices JSON. Estas estructuras JSON se representan como modelos de objetos usando los tipos de Java JsonObject y JsonArray. En la Tabla 1 se incluyen las clases e interfaces principales de la API de modelos de objetos.

JsonObject suministra una vista Map para obtener acceso a la colección no ordenada de cero o más pares de nombres/valores del modelo. De modo similar, JsonArray ofrece una vista List para obtener acceso a la secuencia ordenada de cero o más valores del modelo.

**Tabla 1**. Clases principales de la API de modelos de objetos

|  |  |
| --- | --- |
| **Clase o interfaz** | **Descripción** |
| Json | Contiene métodos estáticos para crear lectores, escritores, constructores de JSON y sus objetos de fábrica. |
| JsonGenerator | Escribe datos JSON en forma de stream, con un valor por vez. |
| JsonReader | Lee datos JSON de un stream y crea un modelo de objeto en la memoria. |
| JsonObjectBuilder JsonArrayBuilder | Crean un modelo de objeto o un modelo de matriz en la memoria agregando valores del código de aplicación. |
| JsonWriter | Escribe un modelo de objeto de la memoria en un stream. |
| JsonValue JsonObject JsonArray JsonString JsonNumber | Representan tipos de datos para valores en datos JSON. |

JsonObject, JsonArray, JsonString y JsonNumber son subtipos de JsonValue. Hay constantes definidas en la API para valores JSON nulos, verdaderos y falsos.

La API de modelos de objetos usa patrones generadores para crear estos modelos de objetos desde cero. El código de aplicación puede usar la interfaz JsonObjectBuilder para crear modelos que representen objetos JSON. El modelo que se obtiene es del tipo JsonObject. El código de aplicación puede usar la interfaz JsonArrayBuilder para crear modelos que representen matrices JSON. El modelo que se obtiene es del tipo JsonArray.

Estos modelos de objetos también pueden crearse a partir de un origen de entrada (por ejemplo InputStream o Reader) usando la interfaz JsonReader. De modo similar, los modelos de objetos pueden escribirse en un origen de salida (por ejemplo OutputStream o Writer) usando la clase JsonWriter.

Por ejemplo, escribamos código para efectuar una búsqueda en los posts públicos de Facebook usando la API de modelos de objetos. La API de Facebook devuelve los resultados de la búsqueda en el formato JSON que se muestra en el Listado 2.

**Listado 2.** Representación JSON de una búsqueda en los posts públicos de Facebook

{

"data" : [

{ "from" : { "name" : "xxx", ... }, "message" : "yyy", ... },

{ "from" : { "name" : "ppp", ... }, "message" : "qqq", ... },

...

],

...

}

Podemos usar la API de modelos de objetos para obtener nombres y sus posts públicos relacionados con el término java. En el Listado 3, las líneas 1 a 3 crean JsonReader; la línea 5 crea JsonObject para los resultados; la línea 7 itera respecto de cada resultado; y las líneas 8 a 11 obtienen el nombre de la persona que publicó el post y el post en sí, y los imprimen. Nótese que JsonReader y otros objetos de esta API pueden usarse en la instrucción try-with-resources (que también se denomina gestión automática de recursos [ARM]).

**Listado 3.** Procesamiento de posts de Facebook con la API de modelos de objetos

URL url = new URL("https://graph.facebook.com/search?q=java&type=post");

try (InputStream is = url.openStream();

JsonReader rdr = Json.createReader(is)) {

JsonObject obj = rdr.readObject();

JsonArray results = obj.getJsonArray("data");

for (JsonObject result : results.getValuesAs(JsonObject.class)) {

System.out.print(result.getJsonObject("from").getString("name"));

System.out.print(": ");

System.out.println(result.getString("message", ""));

System.out.println("-----------");

}

}

**La API de streaming**

La API de streaming es similar a la API de streaming para XML (StAX) y está constituida por las interfaces JsonParser y JsonGenerator. JsonParser contiene métodos para analizar datos JSON usando el modelo de streaming. JsonGenerator contiene métodos para escribir datos JSON en un origen de salida. En la Tabla 2 se incluyen las clases e interfaces principales de la API de streaming.

**Tabla 2.** Clases principales de la API de streaming

|  |  |
| --- | --- |
| **Clase o interfaz** | **Descripción** |
| Json | Contiene métodos estáticos para crear analizadores y generadores JSON, y sus objetos de fábrica. |
| JsonParser | Representa un analizador basado en eventos que puede leer datos JSON en un stream. |
| JsonGenerator | Escribe datos JSON en forma de stream, con un valor por vez. |

**JsonParser**

proporciona acceso directo de solo lectura a datos JSON usando el modelo de programación con análisis pull. En este modelo, el código de aplicación controla el subproceso y llama métodos en la interfaz del analizador para hacer avanzar el analizador u obtener datos JSON del estado actual del analizador.

**JsonGenerator**

proporciona métodos para escribir datos JSON en un stream. El generador puede usarse para escribir pares de nombres/valores en objetos JSON y valores en matrices JSON.

**La API de streaming**

es una API de bajo nivel diseñada para procesar grandes cantidades de datos JSON de manera eficiente. Con esta API, es posible implementar otros marcos JSON (por ejemplo, enlaces JSON).

A continuación, utilizaremos la API de streaming con el mismo fin que se hizo uso de la API de modelos de objetos, es decir, llevar a cabo una búsqueda de posts públicos de Facebook sobre java. En el Listado 4, las líneas 1 a 3 crean un analizador de streaming; las líneas 4 a 5 obtienen el evento siguiente; la línea 6 busca el evento KEY\_NAME; las líneas 8 a 11 leen los nombres y los imprimen; y las líneas 14 a 16 leen los posts públicos y los imprimen. El uso de la API de streaming proporciona un modo eficiente de obtener acceso a nombres y sus posts públicos si se compara con el uso de la API de modelos de objetos para llevar a cabo la misma tarea.

**Listado 4.** Procesamiento de posts de Facebook con la API de streaming

URL url = new URL("https://graph.facebook.com/search?q=java&type=post");

try (InputStream is = url.openStream();

JsonParser parser = Json.createParser(is)) {

while (parser.hasNext()) {

Event e = parser.next();

if (e == Event.KEY\_NAME) {

switch (parser.getString()) {

case "name":

parser.next();

System.out.print(parser.getString());

System.out.print(": ");

break;

case "message":

parser.next();

System.out.println(parser.getString());

System.out.println("---------");

break;

}

}

}

}

**JSON: ESPECIFICACIÓN FORMAL**

La especificación formal de JSON es relativamente simple.

Gráficamente se define con las siguientes figuras.

Figura 1: un objeto JSON está formado por uno o varios pares string: value (cadena:valor)

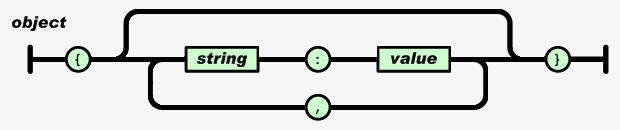


Figura 2: un valor en JSON puede ser un string (cadena), un número, un objeto JSON, un array, el valor true, el valor false o el valor null.

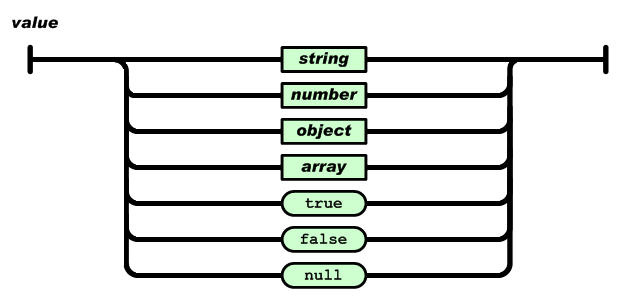


Figura 3: un array o arreglo en JSON está formado por valores delimitados entre corchetes y separados por comas.

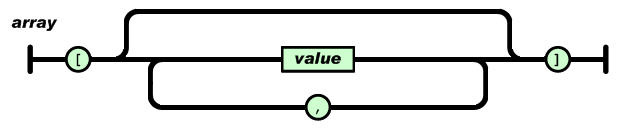


Figura 4: una cadena JSON queda formada de forma análoga a como se forma en muchos lenguajes de programación. Determinados caracteres han de usar una secuencia de escape.

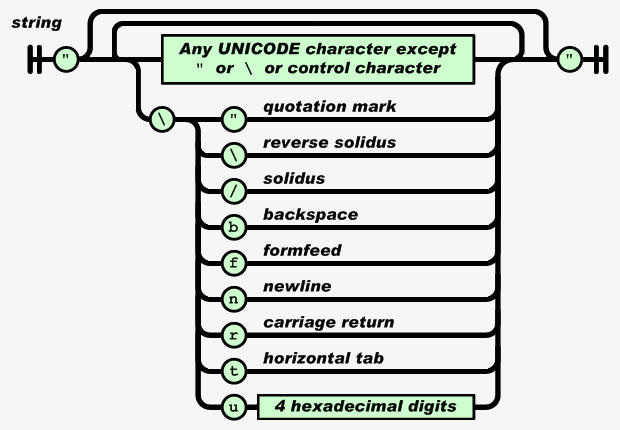
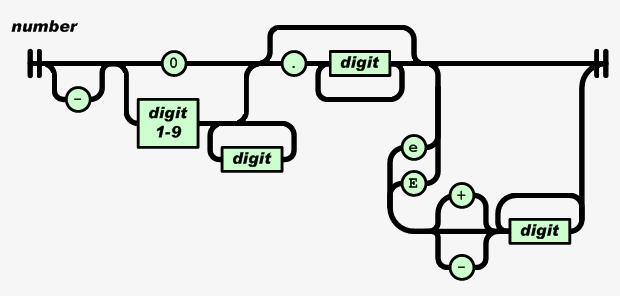


 Figura 5: un número JSON queda formado de forma análoga a como se forma en muchos lenguajes de programación.



1. **CASOS:**

**En los siguientes casos se hara el uzo de API JSON**

**CASO01.**

Veamos un sencillo ejemplo de JSON:

Imaginemos que tenemos una frutería y que queremos obtener el nombre y la cantidad de fruta y verdura que tenemos. En un principio vamos a suponer que tenemos lo siguiente:

 - Fruta:

* 10 manzanas
* 20 Peras
* 30 Naranjas

 - Verduras

* 80 lechugas
* 15 tomates
* 50 pepinos

Para empezar, nos tenemos que familiarizar con la sintaxis de Json:

JSON Nombre/Par de Valores

**Para asignar a un nombre** un valor debemos usar los dos puntos ':' este separador es el equivalente al igual ('=') de cualquier lenguaje.

"Nombre" : "Geeky Theory"

**Valores JSON**

Los tipos de valores que podemos encontrar en Json son los siguientes:

* Un número (entero o float)
* Un string (entre comillas simples)
* Un booleano (true o false)
* Un array (entre corchetes [] )
* Un objeto (entre llaves {})
* Null

**Objetos JSON**

Los objetos JSON se identifican entre llaves, un objeto puede ser en nuestro caso una fruta o una verdura.

**{ "NombreFruta":"Manzana" , "Cantidad":20 }**

**Arrays JSON**

En un Json puedes incluir arrays, para ellos el contenido del array debe ir entre corchetes []:

**{**

**"Frutas": [**

**{ "NombreFruta":"Manzana" , "cantidad":10 },**

**{ "NombreFruta":"Pera" , "cantidad":20 },**

**{ "NombreFruta":"Naranja" , "cantidad":30 }**

**]**

**}**

Una vez explicado el funcionamiento de la sintaxis JSON, vamos a aplicar nuestro ejemplo de la frutería.

**{"Fruteria":**

**[**

**{"Fruta":**

**[**

**{"Nombre":"Manzana","Cantidad":10},**

**{"Nombre":"Pera","Cantidad":20},**

**{"Nombre":"Naranja","Cantidad":30}**

**]**

**},**

**{"Verdura":**

**[**

**{"Nombre":"Lechuga","Cantidad":80},**

**{"Nombre":"Tomate","Cantidad":15},**

**{"Nombre":"Pepino","Cantidad":50}**

**]**

**}**

**]**

**}**

Como podemos observar, hemos creado un objeto llamado *frutería* y, dentro de ese objeto hemos almacenado un array de dos elementos. El primer elemento del array contiene un objeto llamado *fruta* y el segundo elemento del array contiene otro objeto llamado *verdura*. Estos objetos a su vez contienen un array cuyo contenido es el nombre y la cantidad de cada fruta o verdura.

Imaginemos que nos gustaría saber la cantidad de manzanas que tenemos. El path de este array sería el siguiente:

**Path: json['Fruteria'][0]['Fruta'][0]['Cantidad']**

Observamos que la cantidad de manzanas se almacena dentro del primer elemento del array que contiene el objeto *Frutería*, y a su vez dentro del primer elemento del array que contiene el objeto *Fruta*.

**CASO 02.**

En este caso se usará la clase Hijo para definir la estructura que tendrá el json. Hijo es un poco sencillo como se ve a continuación:

|  |
| --- |
| public class Hijo { |
|  |  |
|  | private String nombre; |
|  | private String apellido; |
|  | private int edad; |
|  | private Hijo hijo; |
|  |  |
|  | public Hijo() { |
|  | } |
|  |  |
|  | public Hijo getHijo() { |
|  | return hijo; |
|  | } |
|  | public void setHijo(Hijo hijo) { |
|  | this.hijo = hijo; |
|  | } |
|  | public int getEdad() { |
|  | return edad; |
|  | } |
|  | public void setEdad(int edad) { |
|  | this.edad = edad; |
|  | } |
|  | public Hijo(String nombre, String apellido) { |
|  | this.nombre = nombre; |
|  | this.apellido = apellido; |
|  | } |
|  | public String getNombre() { |
|  | return nombre; |
|  | } |
|  | public void setNombre(String nombre) { |
|  | this.nombre = nombre; |
|  | } |
|  | public String getApellido() { |
|  | return apellido; |
|  | } |
|  | public void setApellido(String apellido) { |
|  | this.apellido = apellido; |
|  | } |
|  | } |

El siguiente código muestra como leer una cadena json haciendo uso de la librería gson y sus métodos, para instanciar un objeto Hijo con sus datos según el json:

|  |
| --- |
| import com.google.gson.Gson; |
|  |  |
|  | public class GsonSimpleRead { |
|  |  |
|  | public GsonSimpleRead() { |
|  | String json = "{\"nombre\":\"Juan\"," |
|  | + "\"apellido\":\"Carabela\"," |
|  | + "\"edad\":45, " |
|  | + "\"hijo\":{" |
|  | + "\"nombre\":\"Pepe\"," |
|  | + "\"apellido\":\"Carabela\"," |
|  | + "\"edad\":21," |
|  | + " \"hijo\":null" |
|  | + "}" |
|  | + "}"; |
|  | Gson gson = new Gson(); |
|  | Hijo hijo = gson.fromJson(json, Hijo.class); |
|  | System.out.println(hijo.getEdad()); |
|  | System.out.println(hijo.getHijo().getNombre()); |
|  | } |
|  |  |
|  | public static void main(String[] args) { |
|  | GsonSimpleRead gson = new GsonSimpleRead(); |
|  | } |
|  |  |
|  | } |

Este código da como resultado lo siguiente:  
  45  
  Pepe

Por otro lado si se desea escribir una cadena json en base a un objeto, en este caso Hijo, simplemente se debe utilizar el siguiente código:

|  |
| --- |
| import com.google.gson.Gson; |
|  |  |
|  | public class GsonSimpleWrite { |
|  |  |
|  | public GsonSimpleWrite() { |
|  |  |
|  | Hijo hijo = new Hijo(); |
|  | hijo.setNombre("Juan"); |
|  | hijo.setApellido("Carabela"); |
|  | hijo.setEdad(46); |
|  |  |
|  | Hijo hijo2 = new Hijo(); |
|  | hijo2.setNombre("Rambo"); |
|  | hijo2.setApellido("Carabela"); |
|  | hijo2.setEdad(21); |
|  | hijo2.setHijo(null); |
|  |  |
|  | hijo.setHijo(hijo2); |
|  |  |
|  | Gson gson = new Gson(); |
|  |  |
|  | String json = gson.toJson(hijo); |
|  | System.out.println(json); |
|  | } |
|  |  |
|  | public static void main(String[] args) { |
|  |  |
|  | GsonSimpleWrite gson = new GsonSimpleWrite(); |
|  | } |
|  |  |
|  | } |

El código anterior da como resultado la siguiente salida en consola:

{  
   "nombre":"Juan",  
   "apellido":"Carabela",  
   "edad":46,  
   "hijo":{  
      "nombre":"Rambo",  
      "apellido":"Carabela",  
      "edad":21  
   }  
}

1. **RECOMENDACIÓN:**

Se recomienda usar JSON antes que XML ya que una de las mayores ventajas que tiene su uso es que puede ser leído por cualquier lenguaje de programación. Por lo tanto, puede ser usado para el intercambio de información entre distintas tecnologías**.**

1. **CONCLUSION:**

En conclusión, la API de Java para procesamiento JSON proporciona las capacidades siguientes:

* Analiza streams de entrada en objetos inmutables o streams de eventos
* Escribe streams de eventos u objetos inmutables como streams de salida
* Navega mediante programación objetos inmutables
* Construye mediante programación objetos inmutables con generadores
* La API sirve como base para desarrollar rutinas API de transformación, consulta, enlace u otras manipulaciones de datos. JAX-RS 2.0 suministra integración nativa para la API de Java para procesamiento JSON.

1. **INFORMACIÓN ADICIONAL**

Jitendra Kotamraju, integrante principal del equipo técnico de Oracle, dirige el proyecto de especificación del procesamiento JSON y es uno de los ingenieros más importantes en relación con GlassFish. Antes de dirigir el proyecto de procesamiento JSON, estuvo a cargo de la especificación e implementación de JAX-WS 2.2.

1. **BIBLIOGRAFIA**

https://www.aprenderaprogramar.com (reseña histórica blog)

http://www.oracle.com/technetwork/es/articles/java/api-java-para-json-2251318-esa.html (Pág. oficial Oracle)

http://www.w3schools.com/json/json\_syntax.asp

https://geekytheory.com

http://jsonapi.org/ (Abaut)

https://elbauldelprogramador.com/buenas-practicas-para-el-diseno-de-una-api-restful-pragmatica/ (Blog el baúl del programador)