JSON

Introducción a JSON

JSON, cuyo nombre corresponde a las siglas JavaScript Object Notation o Notación de Objetos de JavaScript, es un formato ligero de intercambio de datos, que resulta sencillo de leer y escribir para los programadores y simple de interpretar y generar para las máquinas.

JSON es un formato de texto completamente independiente de lenguaje, pero utiliza convenciones que son ampliamente conocidos por los programadores, entre ellos:

· C

· C++

· C#

· Java

· JavaScript

· Perl

· Python

· Entre otros

Dichas propiedades hacen de JSON un formato de intercambio de datos ideal para usar con API REST o AJAX. A menudo se usa en lugar de XML, debido a su estructura ligera y compacta.

Muchos lenguajes de programación proporcionan métodos para analizar una cadena de texto con este formato en un objeto nativo y viceversa.

Según la descripción de Stack Overflow, JSON “define seis tipos de valores: nulo, números, cadenas, booleanos, matrices y objetos”.

Pese a su nombre, no es necesariamente parte de JavaScript, de hecho, es un estándar basado en texto plano para el intercambio de datos, por lo que se usa en muchos sistemas que requieren mostrar o enviar información para ser interpretada por otros sistemas.

Una de las características de JSON, al ser un formato que es independiente de cualquier lenguaje de programación, es que los servicios que comparten información por este método no necesitan hablar el mismo idioma, es decir, el emisor puede ser Java y el receptor Python, pues cada uno tiene su propia librería para codificar y decodificar cadenas en este formato.

Podemos concluir entonces en que JSON es un formato común para ‘serializar’ y ‘deserializar’ objetos en la mayoría de los idiomas.

Por todas las características que te hemos comentado, JSON se ha adoptado ampliamente en el mundo de la programación como una alternativa a XML.

¿Cómo surgió JSON?

A principios de la década de los 90 surgió el problema de que las máquinas pudieran entenderse entre sí. Entonces utilizaban diferentes sistemas operativos y sus programas estaban escritos en diferentes lenguajes de programación. Una de las soluciones fue crear el estándar XML.

Sin embargo, XML presentaba problemas sobre todo cuando se trataba de trabajar con gran volumen de datos, puesto que 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 a principios de los 2000 por Douglas Crockford, un programador conocido como el ‘gurú’ de JavaScript.

Desde entonces JSON se caracteriza por reducir el tamaño de los archivos y el volumen de datos que es necesario transmitir. Por ello fue adquiriendo popularidad hasta convertirse en un estándar. Esto no significa que XML haya dejado de utilizarse, en la actualidad ambos se emplean para el intercambio de datos.

En este punto te preguntarás: ¿Quién está usando JSON actualmente? En primer lugar, su creador es el arquitecto senior de Yahoo JavaScript, así que ya te imaginarás qué formato de intercambio de datos utiliza esta plataforma.

El uso de JSON está creciendo rápidamente en toda la industria, debido a que es una tecnología útil para los desarrollos. Recientemente Twitter migró de XML a JSON. Google Web Toolkit también trabaja con este formato.

Características de JSON

· * JSON es solo un formato de datos.

· * Requiere usar comillas dobles para las cadenas y los nombres de propiedades. Las comillas simples no son válidas.

· * Una coma o dos puntos mal ubicados pueden producir que un archivo JSON no funcione.

· * Puede tomar la forma de cualquier tipo de datos que sea válido para ser incluido en un JSON, no solo arreglos u objetos. Así, por ejemplo, una cadena o un número único podrían ser objetos JSON válidos.

· * A diferencia del código JavaScript, en el que las propiedades del objeto pueden no estar entre comillas, en JSON solo las cadenas entre comillas pueden ser utilizadas como propiedades.

Ventajas y desventajas de JSON y XML

Utilizar JSON o XML depende de las circunstancias y de las preferencias que en cada momento se determinen, pues cada uno tiene sus ventajas y desventajas. Aquí te mencionamos algunas:

JSON

Ventajas

· Es autodescriptivo y fácil de entender.

· Su sencillez le ha permitido posicionarse como alternativa a XML. · Es más rápido en cualquier navegador.

· Es más fácil de leer que XML.

· Es más ligero (bytes) en las transmisiones.

· Se parsea más rápido.

· Velocidad de procesamiento alta.

· Puede ser entendido de forma nativa por los analizadores de JavaScript.

Desventajas

· Algunos desarrolladores encuentran su escueta notación algo confusa. · No cuenta con una característica que posee XML: extensibilidad. · No soporta grandes cargas, solo datos comunes.

· Para la seguridad requiere de mecanismos externos como expresiones regulares. ·

XML

Ventajas

· Tiene un formato estructurado y fácil de comprender.

· Separa radicalmente la información o el contenido de su presentación o formato. · Está diseñado para ser utilizado en cualquier lenguaje o alfabeto. · Su análisis sintáctico es fácil debido a las estrictas reglas que rigen la composición de un documento.

· Tiene soporte a cualquier tipo de datos.

· Se pueden definir estructuras complejas y reutilizables.

Desventajas

· El formato es sumamente estricto.

· Lleva más tiempo procesarlo.

· Complejidad de analizador (parser).

· Un error en cualquier parte del formato puede hacer que todo el documento sea inválido.

Sintaxis JSON

Ya sabes qué es un archivo JSON, ahora debes saber que para crear correctamente un archivo .json, debes seguir la sintaxis correcta.

Hay dos elementos centrales en un objeto JSON: claves (Keys) y valores (Values).

· Las Keys deben ser cadenas de caracteres (strings). Como su nombre en español lo indica, estas contienen una secuencia de caracteres rodeados de comillas.

· Los Values son un tipo de datos JSON válido. Puede tener la forma de un arreglo (array), objeto, cadena (string), booleano, número o nulo.

Un objeto JSON comienza y termina con llaves {}. Puede tener dos o más pares de claves/valor dentro, con una coma para separarlos. Así mismo, cada key es seguida por dos puntos para distinguirla del valor.

Ejemplo:

{“ciudad“:“Buenos Aires”, “país“:“Argentina”}

Aquí tenemos dos pares de clave/valor: ciudad y país son las claves; Buenos Aires y Argentina son los valores.

Tipos de valores

Como se mencionaba antes, los valores contienen un tipo de datos JSON válido, como:

Array

Un array (en español conocido como arreglo o vector) es una colección ordenada de valores. Está rodeado de corchetes [] y cada valor dentro está separado por una coma.

Un valor de un array puede contener objetos JSON, lo que significa que utiliza el mismo concepto de par clave/valor. Por ejemplo:

“estudiantes”: [
{“primerNombre”:”Matias”, “Apellido”:”Fernandez”},
{“primerNombre”:”Juan”, “Apellido”:”Torres”},
{“primerNombre”:”Alexis”, “Apellido”:”García”}
]

En este caso, la información entre corchetes es un array, que tiene tres objetos.

Objeto

Un objeto contiene una clave y un valor. Hay dos puntos después de cada clave y una coma después de cada valor, que también distingue a cada objeto. Ambos están entre comillas.

El objeto, como valor, debe seguir la misma regla que un objeto común.

Ejemplo:

“empleados“: {“nombre“:“Tom”, “apellido“:“Jackson”}

Aquí, empleados es la clave, mientras que todo lo que está dentro de las llaves es el objeto.

Strings

Un string (conocido en español como cadena de caracteres) es una secuencia establecida de cero o más caracteres Unicode. Está encerrado entre dos comillas dobles.

Este ejemplo muestra que Tom es un string ya que es un conjunto de caracteres dentro de una comilla doble.

“Primer Nombre“: “Guillermo”

Número

El número en JSON debe ser un número entero o un punto flotante, como {“Edad“: 30}

Booleano

Puedes usar verdadero o falso como valor, de la siguiente manera: {“Casado“: false}

Nulo

Es para mostrar que no hay información.

{“Tipo de sangre“:“null”}

Datos JSON almacenados

Continuando con esta explicación acerca de qué es JSON, pasemos a hablar de que hay dos formas de almacenar datos JSON: objeto y vector. El primero se ve así:

{

“nombre“: “Alejandro”,

“apellido“: “Martínez”,

“género“: “masculino”

}

Las llaves indican que es un objeto JSON. Implica tres pares clave/valor que están separados por comas.

En cada par, tienes las claves (nombre, apellido y género) seguidas de dos puntos para distinguirlos de los valores (Tom, Jackson, masculino).

Los valores en este ejemplo son strings. Por eso también están entre comillas, similares a las claves.

Usando vectores

Otro método para almacenar datos es un vector (array). Échale un vistazo a éste ejemplo:

{
“nombre“: “Juan”,
“apellido“: “Martin”,
“género“: “Alexis”,
“hobby“:[“fútbol”, “lectura”, “natación”]
}

Lo que diferencia esto del método anterior es el cuarto par clave/valor. Hobby es la clave y hay varios valores (fútbol, lectura, natación) entre corchetes, que representan un vector.

Puede ser útil cuando se combina con JSONP para superar el problema entre dominios. Este proceso funciona utilizando lo que se denomina devoluciones de llamada (callbacks), que solicitarán un elemento específico del vector sin obtener un error “del mismo origen” (same-origin).

Y afortunadamente, un Array también admite bucles, lo que te permite ejecutar comandos repetidos para buscar múltiples datos, haciendo que el proceso sea más rápido y efectivo.

Conclusión

Como puedes ver, se trata de una herramienta útil para intercambiar datos. Tiene muchas ventajas:

· Puedes cargar información de forma asíncrona para que tu sitio web responda mejor y pueda manejar el flujo de datos con mayor facilidad.

· También puedes usarlo para superar problemas de dominio cruzado al intercambiar datos desde otro sitio.

· Un archivo JSON es más simple y más liviano que un archivo XML.

Cómo usar JSON.parse() y JSON.stringify()

El objeto JSON, que está disponible en todos los navegadores modernos, tiene dos útiles métodos para manejar el contenido con formato

JSON: parse y stringify. JSON.parse() toma una cadena JSON y la transforma en un objeto de JavaScript JSON.stringify() toma un objeto de JavaScript y lo transforma en una cadena JSON.

Ejemplo:

```
const myObj = {
 name: 'Sammy',
```

```
 age: 6,
 favoriteFood: 'Tofu'
};

const myObjStr = JSON.stringify(myObj);
console.log(myObjStr);
// "{"name":"Sammy","age":6,"favoriteFood":"Tofu"}"

console.log(JSON.parse(myObjStr));
// Object {name:"Sammy",age:6,favoriteFood:"Tofu"}
```

A pesar de que los métodos se utilizan generalmente en objetos, también se pueden usar en matrices:

```
const myArr = ['bacon', 'lettuce', 'tomatoes'];
const myArrStr = JSON.stringify(myArr);
console.log(myArrStr);
// "["shark","fish","dolphin"]"
console.log(JSON.parse(myArrStr));
// ["shark","fish","dolphin"]
```

JSON.parse()

JSON.parse() puede tomar una función como segundo argumento que puede transformar los valores de objeto antes de que se devuelvan. Aquí los valores del objeto se convierten en mayúsculas en el objeto devuelto del método parse:

```
const user = {
 name: 'Sammy',
```

```
 email: 'Sammy@domain.com',
 plan: 'Pro'
};

const userStr = JSON.stringify(user);
JSON.parse(userStr, (key, value) => {
 if (typeof value === 'string') {
  return value.toUpperCase();
 }
 return value;
});
```

Nota: Las comas al final no son válidas en JSON, por lo que JSON.parse() genera un error si la cadena que se pasa a ella tiene comas al final.

JSON.stringify()

JSON.stringify() puede tomar dos argumentos adicionales: el primero es una función replacer y el segundo es un valor String o Number que se utiliza como un space en la cadena que se devuelve.

La función de reemplazo se puede usar para filtrar los valores, ya que cualquier valor devuelto como undefined estará fuera de la cadena devuelta:

```
const user = {
 id: 229,
 name: 'Sammy',
 email: 'Sammy@domain.com'
};
```

```
function replacer(key, value) {
 console.log(typeof value);
 if (key === 'email') {
 return undefined;
 }
 return value;
}
const userStr = JSON.stringify(user, replacer);
// "{"id":229,"name":"Sammy"}"
```

Y un ejemplo con un argumento space aprobado:

```
const user = {
 name: 'Sammy',
 email: 'Sammy@domain.com',
 plan: 'Pro'
};

const userStr = JSON.stringify(user, null, '...');
// "{
// ..."name": "Sammy",
// ..."email": "Sammy@domain.com",
// ..."plan": "Pro"
// }"
```