Métodos JSON, toJSON

Digamos que tenemos un objeto complejo y nos gustaría convertirlo en un string (cadena de caracteres), para enviarlos por la red, o simplemente mostrarlo para fines de registro.

Naturalmente, tal string debe incluir todas las propiedades importantes.

Podríamos implementar la conversión de esta manera:

```
let user = {
  name: "John",
  age: 30,

  toString() {
    return `{name: "${this.name}", age: ${this.age}}`;
  }
};

alert(user); // {name: "John", age: 30}
```

...Pero en el proceso de desarrollo se agregan nuevas propiedades, y otras son renombradas y eliminadas. Actualizar el **toString** cada vez se vuelve penoso. Podemos intentar recorrer las propiedades, pero ¿qué pasa si el objeto es complejo y tiene objetos anidados en las propiedades? Vamos a necesitar implementar su conversión también.

Por suerte no hay necesidad de escribir el código para manejar todo esto. La tarea ya ha sido resuelta.

JSON.stringify

JSON (Notación de objeto JavaScript) es un formato general para representar valores y objetos. Se lo describe como el estándar **RFC 4627**. En un principio fue creado para Javascript, pero varios lenguajes tienen librerías para manejarlo también. Por lo tanto es fácil utilizar JSON para intercambio de información cuando el cliente utiliza JavaScript y el servidor está escrito en Ruby, PHP, Java...

JavaScript proporciona métodos:

- **JSON.stringify** para convertir objetos a **JSON**.
- **JSON.parse** para convertir **JSON** de vuelta a un objeto.

Por ejemplo, aquí hacemos **JSON.stringify** a student:

```
let student = {
  name: 'John',
  age: 30,
  isAdmin: false,
```

```
  courses: ['html', 'css', 'js'],
  spouse: null
};

let json = JSON.stringify(student);

alert(typeof json); // ¡obtenemos un string!

alert(json);
/* Objeto JSON-codificado:
{
  "name": "John",
  "age": 30,
  "isAdmin": false,
  "courses": ["html", "css", "js"],
  "spouse": null
}
*/
```

El método **JSON.stringify(student)** toma al objeto y lo convierte a un string.

La cadena de caracteres json resultante se llama objeto **JSON-codificado** o **serializado** o convertido a String o reunido. Estamos listos para enviarlo por la red o colocarlo en el almacenamiento de información simple.

Hay que resaltar que el objeto **JSON-codificado** tiene varias diferencias importantes con el objeto literal:

- Los strings utilizan comillas dobles. No hay comillas simples o acentos abiertos en JSON. Por lo tanto 'John' pasa a ser "John".
- Los nombres de propiedades de objeto también llevan comillas dobles. Eso es obligatorio. Por lo tanto age:30 pasa a ser "age":30.
- **JSON.stringify** puede ser aplicado a los tipos de datos primitivos también.
- **JSON** admite los siguientes tipos de datos:
  - Objects { ... }
  - Arrays [...]
  - Primitives:
    - strings,
    - numbers,
    - boolean values true/false,
    - null.

Por ejemplo:

```
// un número en JSON es sólo un número
alert( JSON.stringify(1) ) // 1

// un string en JSON sigue siendo una cadena de caracteres, pero con comillas dobles
alert( JSON.stringify('test') ) // "test"

alert( JSON.stringify(true) ); // true

alert( JSON.stringify([1, 2, 3]) ); // [1,2,3]
```

JSON es una especificación de sólo datos independiente del lenguaje, por lo tanto algunas propiedades de objeto específicas de Javascript son **omitidas** por JSON.stringify.

A saber:

- Propiedades de funciones (métodos).
- Propiedades simbólicas.
- Propiedades que almacenan undefined.

```
let user = {
  sayHi() { // ignorado
    alert("Hello");
  },
  [Symbol("id")]: 123, // ignorado
  something: undefined // ignorado
};

alert( JSON.stringify(user) ); // {} (objeto vacío)
```

Normalmente esto está bien. Si esto no es lo que queremos, pronto veremos cómo personalizar el proceso.

Lo mejor es que **se permiten objetos anidados** y se convierten automáticamente.

Por ejemplo:

```
let meetup = {
  title: "Conference",
  room: {
    number: 23,
    participants: ["john", "ann"]
  }
};
```

```
alert( JSON.stringify(meetup) );
/* La estructura completa es convertida a String:
{
  "title":"Conference",
  "room":{"number":23,"participants":["john","ann"]},
}
*/
```

La limitación importante: **no deben existir referencias circulares.**

Por ejemplo:

```
let room = {
  number: 23
};

let meetup = {
  title: "Conference",
  participants: ["john", "ann"]
};

meetup.place = room;       // meetup tiene referencia a room
room.occupiedBy = meetup; // room hace referencia a meetup

JSON.stringify(meetup); // Error: Convirtiendo estructura circular a JSON
```

Aquí, la conversión falla debido a una **referencia circular**: room.occupiedBy hace referencia a meetup, y meetup.place hace referencia a room:

Excluyendo y transformando: sustituto

La sintaxis completa de JSON.stringify es:

```
let json = JSON.stringify(value[, replacer, space])
```

value

Un valor para codificar.

replacer

Array de propiedades para codificar o una función de mapeo function(propiedad, valor).

space

Cantidad de espacio para usar para el formateo

La mayor parte del tiempo, JSON.stringify es utilizado con el primer argumento unicamente. Pero si necesitamos ajustar el proceso de sustitución, como para filtrar las referencias circulares, podemos utilizar el segundo argumento de JSON.stringify.

Si pasamos un array de propiedades, solamente éstas propiedades serán codificadas.

Por ejemplo:

```
let room = {
  number: 23
};

let meetup = {
  title: "Conference",
  participants: [{name: "John"}, {name: "Alice"}],
  place: room // meetup hace referencia a room
};

room.occupiedBy = meetup; // room hace referencia a meetup

alert( JSON.stringify(meetup, ['title', 'participants']) );
// {"title":"Conference","participants":[{},{}]}
```

Aquí probablemente seamos demasiado estrictos. La lista de propiedades se aplica a toda la estructura de objeto. Por lo tanto los objetos en participants están vacíos, porque name no está en la lista.

Incluyamos en la lista todas las propiedades excepto room.occupiedBy esto causaría la referencia circular:

```
let room = {
  number: 23
};

let meetup = {
  title: "Conference",
  participants: [{name: "John"}, {name: "Alice"}],
  place: room // meetup hace referencia a room
};

room.occupiedBy = meetup; // room hace referencia a meetup

alert( JSON.stringify(meetup, ['title', 'participants', 'place', 'name', 'number']) );
/*
{
  "title":"Conference",
  "participants":[{"name":"John"},{"name":"Alice"}],
  "place":{"number":23}
}
*/
```

Formato: espacio

El tercer argumento de JSON.stringify(value, replacer, space) es el número de espacios a utilizar para un formato agradable.

Anteriormente todos los objetos convertidos a String no tenían sangría ni espacios adicionales. Eso está bien si queremos enviar un objeto por la red. El argumento space es utilizado exclusivamente para una salida agradable.

Aquí space = 2 le dice a JavaScript que muestre objetos anidados en varias líneas, con sangría de 2 espacios dentro de un objeto:

```
let user = {
  name: "John",
  age: 25,
  roles: {
    isAdmin: false,
    isEditor: true
  }
};

alert(JSON.stringify(user, null, 2));
/* sangría de dos espacios:
{
  "name": "John",
  "age": 25,
  "roles": {
    "isAdmin": false,
    "isEditor": true
  }
}
*/

/* para JSON.stringify(user, null, 4) el resultado sería más indentado:
{
    "name": "John",
    "age": 25,
    "roles": {
        "isAdmin": false,
        "isEditor": true
    }
}
*/
```

El tercer argumento puede ser también string. En ese caso el string será usado como indentación en lugar de un número de espacios.

El argumento space es utilizado únicamente para propósitos de registro y agradable impresión.

"toJSON" Personalizado

Tal como toString para conversión de String, un objeto puede proporcionar el método toJSON para conversión a JSON. JSON.stringify automáticamente la llama si está disponible.

Por ejemplo:

```
let room = {
  number: 23
};

let meetup = {
  title: "Conference",
  date: new Date(Date.UTC(2017, 0, 1)),
  room
};

alert( JSON.stringify(meetup) );
/*
  {
    "title":"Conference",
    "date":"2017-01-01T00:00:00.000Z",  // (1)
    "room": {"number":23}               // (2)
  }
*/
```

Aquí podemos ver que date (1) se convirtió en un string. Esto es debido a que todas las fechas tienen un método toJSON incorporado que devuelve este tipo de string.

JSON.parse

Para decodificar un string JSON, necesitamos otro método llamado JSON.parse.

La sintaxis:

```
let value = JSON.parse(str, [reviver]);
```

str

string JSON para analizar.

reviver

function(key,value) opcional que será llamado para cada par (propiedad, valor) y puede transformar el valor.

Por ejemplo:

```
// array convertido en String
let numbers = "[0, 1, 2, 3]";

numbers = JSON.parse(numbers);

alert( numbers[1] ); // 1
```

O para objetos anidados:

```
let userData = '{ "name": "John", "age": 35, "isAdmin": false, "friends": [0,1,2,3] }';

let user = JSON.parse(userData);

alert( user.friends[1] ); // 1
```

El JSON puede ser tan complejo como sea necesario, los objetos y arrays pueden incluir otros objetos y arrays. Pero deben cumplir el mismo formato JSON.

Errores comunes

Aquí algunos de los errores más comunes al escribir JSON a mano:

```
let json = `{
  name: "John",                     // error: nombre de propiedad sin comillas
  "surname": 'Smith',               // error: comillas simples en valor (debe ser doble)
  'isAdmin': false                  // error: comillas simples en propiedad (debe ser doble)
  "birthday": new Date(2000, 2, 3), // error: no se permite "new", únicamente valores simples
  "friends": [0,1,2,3]              // aquí todo bien
}`;
```

Además, JSON no admite comentarios. Agregar un comentario a JSON lo hace inválido.

Existe otro formato llamado JSON5, que permite claves sin comillas, comentarios, etcétera. Pero es una librería independiente, no una especificación del lenguaje.

El JSON normal es tan estricto no porque sus desarrolladores sean flojos, sino para permitir la implementación fácil, confiable y muy rápida del algoritmo analizador.

Utilizando reactivador

Imagina esto, obtenemos un objeto meetup convertido en String desde el servidor.

Se ve así:

```
// title: (meetup title), date: (meetup date)
let str = '{"title":"Conference","date":"2017-11-30T12:00:00.000Z"}';
```

…Y ahora necesitamos deserializarlo, para convertirlo de vuelta a un objeto JavaScript.

Hagámoslo llamando a JSON.parse:

```
let str = '{"title":"Conference","date":"2017-11-30T12:00:00.000Z"}';

let meetup = JSON.parse(str);

alert( meetup.date.getDate() ); // Error!
```

¡Upss! ¡Un error!

El valor de meetup.date es un string, no un objeto Date. Cómo puede saber JSON.parse que debe transformar ese string a una Date?

Le pasemos a JSON.parse la función reactivadora como el segundo argumento, esto devuelve todos los valores "tal cual", pero date se convertirá en una Date:

```
let str = '{"title":"Conference","date":"2017-11-30T12:00:00.000Z"}';

let meetup = JSON.parse(str, function(key, value) {
  if (key == 'date') return new Date(value);
  return value;
});

alert( meetup.date.getDate() ); // ¡Ahora funciona!
```

Por cierto, esto funciona también para objetos anidados:

```
let schedule = `{
  "meetups": [
    {"title":"Conference","date":"2017-11-30T12:00:00.000Z"},
    {"title":"Birthday","date":"2017-04-18T12:00:00.000Z"}
  ]
}`;

schedule = JSON.parse(schedule, function(key, value) {
  if (key == 'date') return new Date(value);
  return value;
});

alert( schedule.meetups[1].date.getDate() ); // ¡Funciona!
```

Resumen

- JSON es un formato de datos que tiene su propio estándar independiente y librerías para la mayoría de los lenguajes de programación.

- JSON admite objetos simples, arrays, strings, números, booleanos y null.

- JavaScript proporciona los métodos JSON.stringify para serializar en JSON y JSON.parse para leer desde JSON.

- Ambos métodos admiten funciones transformadoras para lectura/escritura inteligente.

- Si un objeto tiene toJSON, entonces es llamado por JSON.stringify.