A JSON Parser

# Un analizador JSON

Esta es una implementación de un analizador JSON en JavaScript:

var json\_parse = function () {  
  
// This is a function that can parse a JSON text, producing a JavaScript  
// data structure. It is a simple, recursive descent parser.  
  
// We are defining the function inside of another function to avoid creating  
// global variables.  
  
 var at, // The index of the current character  
 ch, // The current character  
 escapee = {  
 '"': '"',  
 '\\': '\\',  
 '/': '/',  
 b: 'b',  
 f: '\f',  
 n: '\n',  
 r: '\r',  
 t: '\t'  
 },  
 text,  
  
 error = function (m) {  
  
// Call error when something is wrong.  
  
 throw {  
 name: 'SyntaxError',  
 message: m,  
 at: at,  
 text: text  
 };  
 },  
  
 next = function (c) {  
  
// If a c parameter is provided, verify that it matches the current character.  
  
 if (c && c !== ch) {  
 error("Expected '" + c + "' instead of '" + ch + "'");  
 }  
  
// Get the next character. When there are no more characters,  
// return the empty string.  
  
 ch = text.charAt(at);  
 at += 1;  
 return ch;  
 },  
  
 number = function () {  
  
// Parse a number value.  
  
 var number,  
 string = '';  
  
 if (ch === '-') {  
 string = '-';  
 next('-');  
 }  
 while (ch >= '0' && ch <= '9') {  
 string += ch;  
 next();  
 }  
 if (ch === '.') {  
 string += '.';  
 while (next() && ch >= '0' && ch <= '9') {  
 string += ch;  
 }  
 }  
 if (ch === 'e' || ch === 'E') {  
 string += ch;  
 next();  
 if (ch === '-' || ch === '+') {  
 string += ch;  
 next();  
 }  
 while (ch >= '0' && ch <= '9') {  
 string += ch;  
 next();  
 }  
 }  
 number = +string;  
 if (isNaN(number)) {  
 error("Bad number");  
 } else {  
 return number;  
 }  
 },  
  
 string = function () {  
  
// Parse a string value.  
  
 var hex,  
 i,  
 string = '',  
 uffff;  
  
// When parsing for string values, we must look for " and \ characters.  
  
 if (ch === '"') {  
 while (next()) {  
 if (ch === '"') {  
 next();  
 return string;  
 } else if (ch === '\\') {  
 next();  
 if (ch === 'u') {  
 uffff = 0;  
 for (i = 0; i < 4; i += 1) {  
 hex = parseInt(next(), 16);  
 if (!isFinite(hex)) {  
 break;  
 }  
 uffff = uffff \* 16 + hex;  
 }  
 string += String.fromCharCode(uffff);  
 } else if (typeof escapee[ch] === 'string') {  
 string += escapee[ch];  
 } else {  
 break;  
 }  
 } else {  
 string += ch;  
 }  
 }  
 }  
 error("Bad string");  
 },  
  
 white = function () {  
  
// Skip whitespace.  
  
 while (ch && ch <= ' ') {  
 next();  
 }  
 },  
  
 word = function () {  
  
// true, false, or null.  
  
 switch (ch) {  
 case 't':  
 next('t');  
 next('r');  
 next('u');  
 next('e');  
 return true;  
 case 'f':  
 next('f');  
 next('a');  
 next('l');  
 next('s');  
 next('e');  
 return false;  
 case 'n':  
 next('n');  
 next('u');  
 next('l');  
 next('l');  
 return null;  
 }  
 error("Unexpected '" + ch + "'");  
 },  
  
 value, // Place holder for the value function.  
  
 array = function () {  
  
// Parse an array value.  
  
 var array = [];  
  
 if (ch === '[') {  
 next('[');  
 white();  
 if (ch === ']') {  
 next(']');  
 return array; // empty array  
 }  
 while (ch) {  
 array.push(value());  
 white();  
 if (ch === ']') {  
 next(']');  
 return array;  
 }  
 next(',');  
 white();  
 }  
 }  
 error("Bad array");  
 },  
  
 object = function () {  
  
// Parse an object value.  
  
 var key,  
 object = {};  
  
 if (ch === '{') {  
 next('{');  
 white();  
 if (ch === '}') {  
 next('}');  
 return object; // empty object  
 }  
 while (ch) {  
 key = string();  
 white();  
 next(':');  
 object[key] = value();  
 white();  
 if (ch === '}') {  
 next('}');  
 return object;  
 }  
 next(',');  
 white();  
 }  
 }  
 error("Bad object");  
 };  
  
 value = function () {  
  
// Parse a JSON value. It could be an object, an array, a string, a number,  
// or a word.  
  
 white();  
 switch (ch) {  
 case '{':  
 return object();  
 case '[':  
 return array();  
 case '"':  
 return string();  
 case '-':  
 return number();  
 default:  
 return ch >= '0' && ch <= '9' ? number() : word();  
 }  
 };  
  
// Return the json\_parse function. It will have access to all of the above  
// functions and variables.  
  
 return function (source, reviver) {  
 var result;  
  
 text = source;  
 at = 0;  
 ch = ' ';  
 result = value();  
 white();  
 if (ch) {  
 error("Syntax error");  
 }  
  
// If there is a reviver function, we recursively walk the new structure,  
// passing each name/value pair to the reviver function for possible  
// transformation, starting with a temporary boot object that holds the result  
// in an empty key. If there is not a reviver function, we simply return the  
// result.  
  
 return typeof reviver === 'function' ?  
 function walk(holder, key) {  
 var k, v, value = holder[key];  
 if (value && typeof value === 'object') {  
 for (k in value) {  
 if (Object.hasOwnProperty.call(value, k)) {  
 v = walk(value, k);  
 if (v !== undefined) {  
 value[k] = v;  
 } else {  
 delete value[k];  
 }  
 }  
 }  
 }  
 return reviver.call(holder, key, value);  
 }({'': result}, '') : result;  
  
 };  
}();