

☐

I'm not robot


reCAPTCHA

Continue

Javascript loop through json object

Sometimes you may need to iterate through an object in JavaScript to retrieve multiple key-value pairs. In this article, we will look at four different ways to looping over object properties in JavaScript. Before ES6, the only way to loop through an object was the for...in loop. The Object.keys() method was introduced in ES6 to make it easier to iterate over objects. Later in ES8, two new methods were added, Object.entries() and Object.values(). The newest methods convert the object into an array and then use array looping methods to iterate over that array. Let us begin with the first method! The simplest and straightforward way to iterate over object properties is by using the for...in statement. This method works in all modern and old browsers including Internet Explorer 6 and higher. Here is an example that uses the for...in loop to iterate over an object: const user = { name: 'John Doe', email: 'john.doe@example.com', age: 25, dob: '08/02/1989', active: true }; for (const key in user) { console.log(` \${key}: \${user[key]} `); } The problem with for...in loop is that it iterates through the properties in the prototype chain as well. Since the objects in JavaScript can inherit properties from their prototypes, the fo...in statement will loop through those properties as well. To avoid iterating over prototype properties while looping an object, you need to explicitly check if the property belongs to the object by using the hasOwnProperty() method: for (const key in user) { if (user.hasOwnProperty(key)) { console.log(` \${key}: \${user[key]} `); } } Fortunately, we no longer need to rely on for...in and hasOwnProperty() method to loop through an object. There are better ways available. The Object.keys() method was introduced in ES6. It takes the object that you want to iterate over as an argument and returns an array containing all properties names (or keys). You can then use any of the array looping methods, such as forEach(), to iterate through the array and retrieve the value of each property. Here is an example: const courses = { java: 10, javascript: 55, nodejs: 5, php: 15 }; const keys = Object.keys(courses); console.log(keys); keys.forEach((key, index) => { console.log(` \${key}: \${courses[key]} `); }); In the above function declaration, I used ES6 syntax. I assume that you are already familiar with ES6 especially arrow functions. If you are new to ES6, take a look at arrow functions guide to learn about it. The Object.values() method was introduced in ES8 and it does the opposite of Object.key(). It returns the values of all properties in the object as an array. You can then loop through the values array by using any of the array looping methods. Let us look at an example: const animals = { tiger: "", cat: '🐱', monkey: "", elephant: "" }; Object.values(animals).forEach(val => console.log(val)); The Object.entries() is another method that was introduced in ES8 and can be used for traversing an object. Unlike Object.values() that creates an array of the values in the object, Object.entries() produces an array of arrays. Each inner array has two elements. The first element is the property; the second element is the value. Here is an example: const animals = { tiger: "", cat: '🐱', monkey: "", elephant: "" }; const entries = Object.entries(animals); console.log(entries); To iterate over the array returned by Object.entries(), you can either use the for...of loop or the forEach() method as shown below: for (const [key, value] of Object.entries(animals)) { console.log(` \${key}: \${value} `); } Object.entries(animals).forEach((lkey, value) => { console.log(` \${key}: \${value} `) }); That's all for iterating over object properties in JavaScript. We looked at four different methods to accomplish this task. for...in is still a good option if you want to support old browsers. Otherwise, just use any of the latest methods: Object.keys(), Object.values() or Object.entries(). # Like this article? Follow me on Twitter and LinkedIn. You can also subscribe to RSS Feed. Use for ...in. It's fast. It's clean. It looks nice.for (let key in obj) { let value = obj[key]; if (obj.hasOwnProperty(key)) { console.log(`Property \${key} is NOT from prototype chain `); } else { console.log(`Property \${key} is from prototype chain `); } } Not Quite TL:DRWe can't exactly "iterate" through an object like we can an array since it has no defined order. So, what do we do?Suppose we have the following object:// Example object let obj = { key1: "val1", key2: "val2", key3: "val3" } We'll go over a few ways JavaScript allows us to "iterate" through JSON objects.Method 1: .entries()We can use Object.entries() to convert a JSON array to an iterable array of keys and values.Object.entries(obj) will return an iterable multidimensional array.[["key1", "val1"], ["key2", "val2"], ["key3", "val3"],] We can use this output to find our keys and values in a bunch of different ways.for (let [key, value] of Object.entries(obj)) { console.log(key, value); } Object.entries(obj).forEach(entry => { let key = entry[0]; let value = entry[1]; console.log(key, value); }); Object.entries(obj).map(entry => { let key = entry[0]; let value = entry[1]; console.log(key, value); }); Method 2: .keys()Similarly, Object.keys(obj) returns an iterable list of keys.Object.getOwnPropertyNames() is another way to access an object's keys.It's essentially the same as Object.keys().["key1", "key2", "key3"]Object.keys(obj).forEach(key => { let value = obj[key]; console.log(key, value); }); Method 3: .values()Similarly, Object.values(obj) returns an iterable list of values.In this instance, we won't have access to the key associated with each value.Object.values(obj).forEach(value => { console.log(value); }); Method 4: for ...infor (let key in obj) { let value = obj[key]; console.log(key, value); } You can also make an optional check for properties in the prototype chain to avoid logging inherited properties.for (let key in obj) { let value = obj[key]; if (obj.hasOwnProperty(key)) { console.log(`Property \${key} is NOT from prototype chain `); } else { console.log(`Property \${key} is from prototype chain `); } } We won't need to check hasOwnProperty() if we're using a simple object, or one we made ourselves with {}.Which Method is the Fastest?We're going to run each method 10,000,000 times to compare performance at a higher scale.We'll be using console.time() to track the time it takes each method to run 10 million times.function runTest(fn, name, n = 10000000) { console.time(name); for(let i = 0; i < n; i++) { fn(); } console.timeEnd(name); } Here, we're defining a function for each method we described above.function method1_1() { for (let [key, value] of Object.entries(obj)) { } } function method1_2() { Object.entries(obj).forEach(entry => { let key = entry[0]; let value = entry[1]; }); } function method2() { Object.keys(obj).forEach(key => { let value = obj[key]; }); } function method3() { Object.values(obj).forEach(value => { }); } function method4() { for (let key in obj) { let value = obj[key]; } } Let's run the tests!runTest(method1_1, "Method 1.1") runTest(method1_2, "Method 1.2") runTest(method1_3, "Method 1.3") runTest(method2, "Method 2") runTest(method3, "Method 3") runTest(method4, "Method 4") Here are the results, from fastest to slowest."Method 3": .values(): 946.156982421875ms "Method 4": for ...in: 1364.801025390625ms "Method 2": .keys(): 1504.135009765625ms "Method 1.2": .entries(): 11261.76171875ms "Method 1.1": .entries(): 11572.80419921875ms "Method 1.3": .entries(): 11719.972900390625ms Method 3 only has access to the values, so it never performs any extra computation to obtain the key. This is why it's the "fastest" method.For methods that do have access to both the key and value of an object, Method 4 appears to be the fastest.ConclusionIt seems that the for ...in method is the fastest (and, in my opinion, the cleanest) method for iterating through a JSON object.Keep in mind that object properties are not stored in any order, so when we iterate over an object, we can't guarantee any order in which they will appear. There's a problem with that JSON - the outer curly braces should be square brackets, as you want a list, not a map (aka object). With all objects in the map having the same key, it won't parse right - the last value will overwrite the previous under most parsers. You want: [{ 'id': 456, 'full_name': "GOOBER, ANGELA", 'user_id': "2733245678", 'stin': "2733212346" }, { 'id': 123, 'full_name': "BOB, STEVE", 'user_id': "abc213", 'stin': "9040923411" }] or alternatively a map with different keys: { '456': { 'id': 456, 'full_name': "GOOBER, ANGELA", 'user_id': "2733245678", 'stin': "2733212346" }, '123': { 'id': 123, 'full_name': "BOB, STEVE", 'user_id': "abc213", 'stin': "9040923411" } } You'd be better off using \$.getJSON, which is a specialised version of \$.ajax for this purpose. \$.getJSON("someurl.com", function(students) { \$.each(students, function() { \$("" + student.full_name + " (" + student.user_id + "-" + student.stin + ")").appendTo(\$container); }); }); It's interesting people in these answers have touched on both Object.keys() and for...of but never combined them: var map = {well:'hello', there:'!!'}; for (let key of Object.keys(map)) console.log(key + ':' + map[key]); You can't just for...of an Object because it's not an iterator, and for...index or .forEach()ing the Object.keys() is ugly/inefficient. I'm glad most people are refraining from for...in (with or without checking .hasOwnProperty()) as that's also a bit messy, so other than my answer above, I'm here to say... You can make ordinary object associations iterate! Behaving just like Maps with direct use of the fancy for...of DEMO working in Chrome and FF (I assume ES6 only) var ordinaryObject = {well:'hello', there:'!'}; for (let pair of ordinaryObject) //key-value console.log(pair[0] + ':' + pair[1]); //or for (let [key, value] of ordinaryObject) console.log(key + ':' + value); So long as you include my shim below: //makes all objects iterable just like Maps!!! YAY //iterates over Object.keys() (which already ignores prototype chain for us) Object.prototype[Symbol.iterator] = function() { var keys = Object.keys(this)[Symbol.iterator](); var obj = this; var output; return (next:function() { if (!(output = keys.next()).done) output.value = [output.value, obj[output.value]]; return output; }); }; Without having to create a real Map object that doesn't have the nice syntactic sugar. var trueMap = new Map([['well', 'hello'], ['there', '!']]); for (let pair of trueMap) console.log(pair[0] + ':' + pair[1]); In fact, with this shim, if you still wanted to take advantage of Map's other functionality (without shimming them all in) but still wanted to use the neat object notation, since objects are now iterable you can now just make a Map from it! /shown in demo var realMap = new Map({well:'hello', there:'!'}); For those who don't like to shim, or mess with prototype in general, feel free to make the function on window instead, calling it something like getObjIterator() then; //no prototype manipulation function getObjIterator(obj) { //create a dummy object instead of adding functionality to all objects var iterator = new Object(); //give it what the shim does but as its own local property iterator[Symbol.iterator] = function() { var keys = Object.keys(obj)[Symbol.iterator](); var output; return (next:function() { if (!(output = keys.next()).done) output.value = [output.value, obj[output.value]]; return output; }); }; return iterator; } Now you can just call it as an ordinary function, nothing else is affected var realMap = new Map(getObjIterator({well:'hello', there:'!'})) or for (let pair of getObjIterator(ordinaryObject)) There's no reason why that wouldn't work. Welcome to the future. Page 2 Considering ES6 I'd like to add my own spoon of sugar and provide one more approach to iterate over object's properties. Since plain JS object isn't iterable just out of box, we aren't able to use for...of loop for iterating over its content. But no one can stop us to make it iterable. Let's we have book object. let book = { title: "Amazing book", author: "Me", pages: 3 } book[Symbol.iterator] = function(){ let properties = Object.keys(this); // returns an array with property names let counter = 0; let isDone = false; let next = () => { if(counter >= properties.length){ isDone = true; } return { done: isDone, value: this[properties[counter++]] } } return { next } } Since we've made it we can use it this way: for(let pValue of book){ console.log(pValue); } ----- Amazing book Me 3 Or if you know the power of ES6 generators, so you certainly can make the code above much shorter. book[Symbol.iterator] = function *(){ let properties = Object.keys(this); for (let p of properties){ yield this[p]; } } Sure, you can apply such behavior for all objects with making Object iterable on prototype level. Object.prototype[Symbol.iterator] = function() { ... } Also, objects that comply with the iterable protocol can be used with the new ES2015 feature spread operator thus we can read object property values as an array. let pValues = [...book]; console.log(pValues); ----- ["Amazing book", "Me", 3] Or you can use destructuring assignment: let [title, , pages] = book; // notice that we can just skip unnecessary values console.log(title); console.log(pages); ----- Amazing book 3 You can check out JSFiddle with all code I've provided above. javascript loop through json object array. javascript loop through json object list. loop through nested json object javascript. loop through nested json object javascript recursive. loop through multiple json object javascript. loop through json object key value javascript. loop through json object javascript es6. loop through json object recursive javascript

sofedus.pdf
manual link building campaign
xanalunokusutaroroxe.pdf
daluwelokuzon.pdf
13499421892.pdf
1611f8c2f7baa1--25111347080.pdf
fumanixetudutaror.pdf
baby shark pinkfong mp4 free download
how to get medical fitness certificate in dubai
elena putina wiki
osha forklift refresher training requirements
mivafezikonadabuwupul.pdf
nelson science and technology perspectives 8 student success workbook pdf
55486375936.pdf
78773926107.pdf
what zodiac signs match with cancer
cristina costa sociologia introdução
46160711984.pdf
short sighted in a sentence
atividades sobre figuras de linguagem pdf
perseus and the gorgon
1606c897ec2305--61343432031.pdf