for..in versus for..of Loops

Nov 15, 2016 • *javascript*

The most basic type of iteration method in JavaScript is the **`for`** loop. It takes three expressions; a variable declaration, an expression to be evaluated before each iteration, and an expression to be evaluated at the end of each iteration. For example, this for loop will **`console.log`** each item in an array.

```
const array = ['a', 'b', 'c', 'd'];

for (let i = 0; i < array.length; i++) {
  console.log(array[i]);
}

// Result: a, b, c, d
```

In addition to this **`for`** loop, there are two other types of **`for`** iteration methods we can use: **`for..in`** and **`for..of`**.

The `for..in` Statement

`for..in` is a method for iterating over "enumerable" properties of an object. It therefore applies to all objects (not only **`Object()`**s) that have these properties.

An enumerable property is defined as a property of an object that has an **`Enumerable`** value of true. Essentially, a property is "enumerable", if it is **`enumerable`**. We can check if a property is enumerable by calling **`property.enumerable`**, which will return true or false.

We use the **`for..in`** loop with the following syntax -

```
for (variable in enumerable) {
    // do stuff
}
```

For example, to loop through and **`console.log`** all the values in this Object, we can do the following -

```
const obj = {
    a: 1,
    b: 2,
    c: 3,
    d: 4
}

for (const key in obj) {
    console.log( obj[key] )
}

// Result: 1, 2, 3, 4
```

The **`for ... in`** loop will iterate over inherited properties as well, as long as they are enumerable properties. The **`for ... in`** iteration happens in an arbitrary order. Therefore, it should not be used if things need to happen in their defined sequence.

`for..in` and Objects

The **`for..in`** method provides us the most straightforward way to loop over Object keys and values, since Objects do not have access to the **`forEach`** method that Arrays do.

`for..in` and Arrays

The "key" for values in an Array are the numerical indexes. Therefore, these indexes are essentially just enumerable properties, like Object keys, except they are integers instead of strings.

This means that we can loop over all the values in an Array by retrieving their index using the **`for..in`** Array.

```
const array = ['a', 'b', 'c', 'd'];

for (const index in array) {
    console.log(array[index])
}

// Result: a, b, c, d
```

However, it is generally advised that **`for..in`** not be used with Arrays, particularly because it cannot be guaranteed that the iteration happens in sequence, which is usually important for Arrays.

`for..in` and Strings

Each character in a string has an index. Therefore, similar to Arrays, the indexes are enumerable properties that just happen to be integers.

```
const string = 'Ire Aderinokun';

for (const index in string) {
    console.log(string[index])
}

// Result: I, r, e, , A, d, e, r, i, n, o, k, u, n
```

The `for..of` Statement

`for..of` is a method, introduced in ES2015, for iterating over "iterable collections". These are objects that have a **`[Symbol.iterator]`** property.

The **`[Symbol.iterator]`** property allows us to manually iterate over the collection by calling the **`[Symbol.iterator]().next()`** method to retrieve the next item in the collection.

```
const array = ['a','b','c', 'd'];
const iterator = array[Symbol.iterator]();
console.log( iterator.next().value )
console.log( iterator.next().value )
console.log( iterator.next().value )
console.log( iterator.next().value )

// Result: a, b, c, d
```

The **`for..of`** syntax is essentially a wrapper around the **`[Symbol.iterator]`** to create loops. It uses the following syntax -

```
for (variable of iterable) {
    // do stuff
}
```

`for..of` and Objects

The **`for..of`** loop doesn't work with Objects because they are not "iterable", and therefore don't have a **`[Symbol.iterator]`** property.

`for..of` and Arrays/Strings

The **`for..of`** loop works well with Arrays and Strings, as they are iterable. This method is a more reliable way of looping through an Array in sequence.

```
const array = ['a', 'b', 'c', 'd'];
for (const item of array) {
    console.log(item)
}
// Result: a, b, c, d

const string = 'Ire Aderinokun';
for (const character of string) {
    console.log(character)
}
// Result: I, r, e, , A, d, e, r, i, n, o, k, u, n
```

`for..of` and NodeLists

Finally, another really useful case for **`for..of`** is in iterating of NodeLists. When we query the document for a group of elements, what we get returned is a NodeList, not an Array. This means that we can't iterate over the list using Array methods like **`forEach`**.

To solve this, we can either convert it to an Array using **`Array.from()`**, or use the **`for..of`** loop, which is applicable to more than just Arrays.

```
const elements = document.querySelectorAll('.foo');

for (const element of elements) {
    element.addEventListener('click', doSomething);
}
```

A Comparison

	`for..in`	`for..of`
Applies to	Enumerable Properties	Iterable Collections
Use with Objects?	Yes	No
Use with Arrays?	Yes, but not advised	Yes
Use with Strings?	Yes, but not advised	Yes