As it’s easy to misuse the JavaScript *for(in)* syntax, it’s helpful to take a close look at array enumeration. For starters, let’s ask what’s better: *forEach*, or *for*?

Using *forEach* is generally acceptable for non-performance critical loops. This syntax is semantically obvious, performs well in most scenarios, and avoids many of the classic pitfalls of JavaScript development making it easy to read and author:

Assuming we have this array (used in all the examples here):

var myArray = ["a", "b", "c", "d"];

we can iterate for *forEach* as follows:

myArray.forEach(function (element) {

console.log(element);

});

// prints a b c d

For performance-critical scenarios (“hot paths,” tight loops, etc.), a classic *for* loop is better because the Javascript interpreter can better optimize it. This is also necessary if you need to break from the loop, which you cannot do with forEach.

for (var i = 0, l = myArray.length; i < l; i++) {

console.log(myArray[i]);

};

// prints a b c d

It’s important to cache the array length outside of the loop comparison function (using variable *l* as above), as JavaScript will not do this for you. Be aware of your closures too, especially nested closures. For example:

for (var i = 0, l = myArray.length; i < l; i++) {

msQueueCallback(function () {

console.log(myArray[i]);

});

};

// prints "undefined" 4 times

If it wasn’t obvious why you get *undefined* in the output, don’t use this syntax. Instead, use a simple closure:

for (var i = 0, l = myArray.length; i < l; i++) {

(function (i) {

msQueueCallback(function () { console.log(myArray[i]); });

})(i);

};

// prints a b c d

Alternatively, you can do so with *forEach*:

myArray.forEach(function (element, index) {

console.log(index + ":" + element);

});

// prints 0:a 1:b 2:c 3:d

So what about *for(in)*?

JavaScript’s for(in) loop allows for enumeration of the keys of an item, thus:

for (var i in myArray) {

console.log(i);

};

// prints 0 1 2 3

Which is generally unexpected. The primary use case for *for(in)* is to enumerate the *properties* of an object:

var o = { x: 10, y: 20 };

for (var i in o) {

console.log(i);

};

// prints x y

In general, we recommend you use *Object.keys* to get the set of keys for an object, and then use the normal array enumeration:

var o = { x: 10, y: 20 };

Object.keys(o).forEach(function (element) {

console.log(element);

});

// prints x y

What about reverse enumeration (or your favorite optimization)?

It is true that in many cases enumerating through a collection in reverse is faster:

for (var i = myArray.length; i--;) {

console.log(myArray[i]);

};

// prints d c b a

However, maintaining this kind of code probably costs more in the long run to negate any gains you make in micro-performance. Unless you have real benchmark data to justify the complexity, it’s unlikely that playing tricks like this will actually improve most code in an app.