DERICKBAILEY.COM

Trade Secrets Of A Developer / Entrepreneur

ABOUT TWITTER G+ RSS

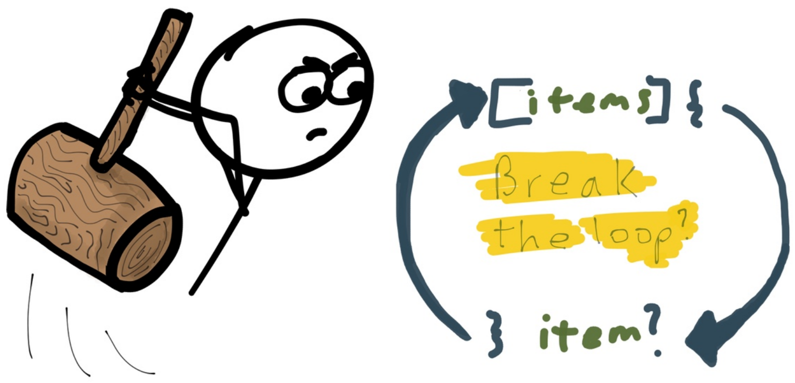
BLOG COURSES PRODUCTS NEWSLETTER PUBLICATIONS

PODCASTS SPEAKING

# Find An Item By Property, In An Array, Without Using ES6 Methods

February 29, 2016 By derickbailey

A reader recently sent me an email, asking a question about how to find an item in an array, based on a property / attribute that would match. Meaning, the object instance will be different, but the attribute (such as “id”) will be the same.



This person wanted to know if there's a good way to do a loop on the array and exit immediately when the item is found – and without ES6 methods like find and findIndex.

## The Sample Code and Question

The code and question they provides are as follows:

> *I'm searching through an array for an element. I'm doing something like this:*

```
var roomId = ... // get the room id you want

// hold the room you found
var foundRoom;

// search for the room
roomsList.forEach(function (room) {
  if (room.id === roomId) {
    foundRoom = room;
  }
});


foundRoom // ... and use foundRoom now that y
```

1.js hosted with ❤ by GitHub view raw

*Note that I can't use array.prototype.find() or findIndex() because they aren't well-supported yet. And I can't use array.prototype.indexOf() because the objects aren't identical; they just have one property in common.*

*The above code works but it must run through the entire array even if the item found is the first item. What a waste! If I return from the function in the if condition, it only returns from the inner function.*

*My question is this: How do you exit the forEach once your condition is satisfied?*

## My Suggestion: For-Loops

The easiest way to handle this, and provide compatibility across any JavaScript runtime environment, would be to use a simple for loop with a break statement.

```
var roomId = ... // get the room id you want to find
var foundRoom;

var length = roomList.length;
var idx = 0;

// regular for loop
for (idx; idx <= length; idx++) {

  // get the room and make the comparison
  var room = roomList[idx];
  if (room.id === roomId) {
    foundRoom = room;

    // we're done! get out now
    break;
  }
}
```

```



foundRoom // ... and use foundRoom now that you have it
```

2.js hosted with ❤ by GitHub view raw

The break statements lets you exit a loop at any point in the process. In this case, the loop can be broken once the desired room is found.

## Why Not Use A For-Loop?

My suggestion above should hopefully be simple enough to get the job done and not be confusing to anyone. I would even venture to guess that most people would react with some of “duh” or other snarky response, as if it were so completely obvious that anyone should know this.

But I don’t think everyone knows this… or at least, I don’t think everyone is willing to write this code.

In recent years, there has been a huge push in the JavaScript community to abandon the “old” and “bad” ways of doing things. And somewhere along the lines, the simple for-loop got lumped in with the bad way of doing things.

There are some legit reasons to not use for-loops, after all:

- You need to properly optimize the loop by only examining the “length” of the array once
- You have to declare a lot of variables, like the iteration index, item variable, etc
- Variables are not scoped to the for-loop, but to the surrounding function / code
- … and probably a few other things I’m forgetting at the moment

While these may seem like trivial issues, they can cause some problems… all of which can be mitigated easily, by wrapping this loop into a function:

```
// create an API for find a room by id
// -----------------------------------

function findRoomById(roomList, roomId){
  var foundRoom;

  var length = roomList.length;
  var idx = 0;

  // regular for loop
  for (idx; idx <= length; idx++) {

    // get the room and make the comparison
    var room = roomList[idx];
    if (room.id === roomId) {
      foundRoom = room;

      // we're done! get out now
      break;
    }
  }

  // send it back
  return foundRoom;
}

// now use this new API
// --------------------

var someListOfRooms = ... // get a list of rooms to searc
var roomId = ... // get the room id you want to find

// get the correct room
var theRoom = findRoom(someListOfRooms, roomId);
```

3.js hosted with ❤ by GitHub view raw

But, because of these issues – and for various other reasons – the JavaScript community has largely said “don’t use for-loops.” Instead, they say we should prefer to use the forEach method found in the original question.

In general, I think this is fair advice. But I also think it’s a terrible idea to throw away your hammer when you buy a new screwdriver.

## Use The *RIGHT* Tool. Not The *NEW* Tool

Using the “new” and “better” syntax in a language is a good thing (most of the time). There’s a reason the syntax was added, after all. But throwing away the old syntax and saying we should never use it again, can be a bad idea.

Having the right tool for the job is important. In this case, the right tool is the old syntax that has been supported since the dawn of JavaScript; a for-loop.

I don’t actually know if the person asking the question was in a situation where they thought they could only use forEach loops.  If they were, I would recommend re-evaluating the reason why.

Yes, there are much smaller ways of finding the room in question using ES6 code. But not every environment supports these methods yet, and compatibility with your target environment is a concern that has to be dealt with.

In the end, I think it’s good to use the new tools that we have, when they make since. Sometimes, though, the old tools we have laying around are still the right tools to use.

Tweet

## RELATED POST

**10 Myths About Docker That Stop Developers Cold**

**Docker Recipes Update: Speed Up npm install In Mou...**

**Update and a Bonus Recipe for the Docker Recipes e...**

**A Sneak Peak at Docker Recipes for Node.js Develop...**

**Docker Recipes for Node.js: Pre-sale Dates & ...**

Filed Under: Anti-Patterns, Arrays, ES5, ES6, Functions, Iterators, JavaScript, Patterns, Principles, Tips And Tricks, Variable Scope

### About derickbailey

Derick Bailey is a developer, entrepreneur, author, speaker and technology leader in central Texas (north of Austin). He's been a professional developer since the late 90’s, and has been writing code since the late 80’s. In his spare time, he gets called a spamming marketer by people on Twitter, and blurts out all of the stupid / funny things he's ever done in his career on his email newsletter.

## DERICK BAILEY AROUND THE WEB

Twitter: @derickbailey

Google+: DerickBailey

Screencasts: WatchMeCode.net

eBook: Building Backbone Plugins