Processing Unordered Array Items In Order, Using Brute Force

April 30, 2015 By derickbailey

I recently had the opportunity to interview Aria Stewart – a developer at PayPal. The interview was for my RabbitMQ For Developers package (coming soon!) and centered around designing for failure. At one point in the conversation, we were talking about the problem of ordered messages and vector clocks and I mentioned a problem I was having in my current client system. The discussion continued for a moment after the recording ended In that time, she talked about solving the "ordering" problem of messages by rejecting a message that was out of order and sending it to the back of the queue. The idea had me intrigued, and I wanted to see if what she was saying would pan out for me. Having done a few quick tests, I think it will. So I wanted to share the most

basic of the demos I put together to see the idea in action.

Re-Ordering The Out Of Order

To start with, you need to understand that some messages sent across a message queue or otherwise processed asynchronously will arrive out of order. It happens. Sometimes it doesn't matter, but other times this can be catastrophic – as it is in my case.

One of the techniques that can be used to combat this is a simple sequence number on the messages. The sequence number can be used to determine if the message is out of order or not. As an example, some items may end up in your "queue" (array in this case) in an order like this (assume these are sequence ids on an object):

```
1  var items = [1, 2, 4, 3, 6, 5, 9, 7, 8];
```

1.js hosted with ♥ by GitHub view raw

You only need to track the most recently processed item and then compare that to the current message you're looking at. If the current message is 1 number higher than the previous one, process it. If it is not one number higher than the previous one, throw it to the back of the queue and move on to the next message. Repeat until everything has been processed in order

Implementing The Brute Force Ordering

It is fairly trivial to implement this idea with an array as the example. You only need one function and a "previous" value to use, and you can use recursion to do the whole thing.

```
var previousItem = 0;

function processItems (items){
  if (items.length === 0) { return; }

  var item = items.shift();

  if (item === (previousItem + 1)){
    console.log("Processing the item: ", item);
    previousItem = item;
  } else {
    console.log("Out of order! Reject the item and r
    items.push(item);
  }

  processItems (items);
}

// start the whole thing
processItems (items);
```

2.js hosted with ♥ by GitHub view raw

In this example, the first line of the processItems function is the exit condition for the recursion. If there are no items left, exit. The next line gets the first value out of the list. The code then checks to see if this is the next item it needs to process. If it does, it handles it and set the “previous” item to the current one. If the items are out of order, the item is pushed to the end of the array so the next item can be checked. Recursion is used to re-enter the loop and continue processing the entire array

(Please note that this operation is destructive to the array If you need to keep your original array in-tact, make a copy of it before processing it like this.)

The result of this code running, looks like this:

```
$ node index.js

Processing the item: 1
Processing the item: 2
Out of order! Reject the item and re-queue it: 4
Processing the item: 3
Out of order! Reject the item and re-queue it: 6
```

```
Out of order Reject the item and re-queue it: 5
Out of order Reject the item and re-queue it: 9
Out of order Reject the item and re-queue it: 7
Out of order Reject the item and re-queue it: 8
Processing the item: 4
Out of order Reject the item and re-queue it: 6
Processing the item: 5
Out of order Reject the item and re-queue it: 9
Out of order Reject the item and re-queue it: 7
Out of order Reject the item and re-queue it: 8
Processing the item: 6
Out of order Reject the item and re-queue it: 9
Processing the item: 7
Processing the item: 8
Processing the item: 9
```


You can see each item being checked and whether or not it was processed. If you only look at the “Processing the item” messages, you will see that they occur in the correct order. The other items – the ones that are out of order – are shoved to the end of the queue so that they can be dealt with later

Brute Force Is Not Performant

This solution is perfectly acceptable in a situation like mine. But you should know that I call this a “brute force” solution for a reason. There is no real intelligence, in here. It is just “throw it back”, forcefully – no regard for any other context. This is not optimal, nor is it particularly “performant”. In fact, you can see in the output that the out-of-order messages often get shoved to the back of the line multiple times.

It would be a better solution, to prevent the items from getting out of order in the first place. It still be better to cache the messages somewhere while they wait to be ordered. But these options are not always feasible.

If you find yourself working with a queue, though, and having to deal with ordered messages, this is at least one option for dealing with things being out of order

Want To Know More?

The interview I mentioned above is a part of the RabbitMQ For Developers package that I produced.

Pick up your copy of The RabbitMQ For Developers bundle and get all of these interviews, 12 screencasts, an eBook and so much more!

Tweet

RELATED POST

The Docker Management Cheatsheet

What Is RabbitMQ? What Does It Do For Me?

Callbacks First, Then Promises

Ending the Nested Tree of Doom with Chained Promis...

Does ES6 Mean The End Of Underscore / Lodash?

Filed Under: Arrays, Bundles, Design, Error Handling, JavaScript, RabbitMQ, Recursion

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