(http://joshowens.me)

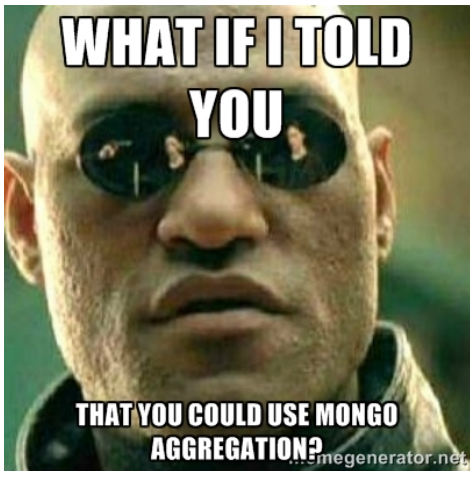
USING MONGODB AGGREGATIONS TO POWER A METEOR.JS PUBLICATION

December 11 2014

mongo (/tag/mongo/), meteor.js (/tag/meteor-js/), aggregation (/tag/aggregation/)

I often hear Meteor.js take some flak for only working with MongoDB. I thought it would be fun to dive into a native Mongo feature and how to implement it in your app. This article is about a way to publish aggregate data using the MongoDB aggregation framework with Meteor.js. What is the MongoDB aggregation framework, you may be asking? The manual says "Aggregations operations process data records and return computed results." which is a fancy way of saying you can find and manipulate data using queries.

This code has been testing and was extracted from a recent commit I did for a client.



The setup

Suppose we have an app we've built, nothing fancy, just an e-vite potluck party planner. You input a date, time, place, and some email addresses of people you want to invite. The goal of the app is to get the invited people to input that they are bringing something to the party. Mr CEO comes over to our desk one day and tells us he wants to make it easier for people to invite their friends, so let's add a modal with select2 autocompletion and we can grab their contacts from their Google account.

We setup everyting and added a Contacts collection to the app to power our fancy new autocompletion widget. Once we get the new feature up on staging, Mr CEO makes a comment that not all of his friends are showing up in the autocomplete list. After a quick discussion, we realize we need to grab previously invited people and add them to the contact list as well.

The first step to grabbing those emails is to get the aggregation framework package installed - it doesn't do anything to fancy, just wraps up some Mongo methods for you. Just `meteor add meteorhacks:aggregate` and you should be in business. This will add an aggregate method to your collections.

Building and publishing the data

We have Events with an array of invite objects that contain the data we want **to access**. Now we can dive into the aggregation framework and build up our pipeline queries.

```
contacts = Events.aggregate([{$match: {creatorId: this.userId}}, {$project
: {invites: 1}}, { $unwind : "$invites" }, {$group: {_id: {email: "$invite
s.email"}}}, {$project: {email: "$_id.email"}}])
```

Aggregate takes an array of queries, each one passing in the results of the previous query. The first thing we do is `$match` any events that the user created. Then we use the `$project` option to only passing along the invites array. Next, we use the `$unwind` option to split each invite array element into it's own record, so if we invited 4 people, we will get 4 results. After that, we `$group` the invites by email address so we only get uniques emails in case we invited someone more than once. Last, we `$project` the email address out of the id field and stick it in as an email attribute on our results.

I've often called publications in Meteor.js the heart and soul of your application. We can take all these results and add some 'soul' to our app by

publishing them to the Contacts collection on the client side:

```
Meteor.publish('previousInviteContacts', function() {
  self = this;
  contacts = Events.aggregate([{$match: {creatorId: this.userId}}, {$proje
ct: {invites: 1}}, { $unwind : "$invites" }, {$group: {_id: {email: "$invi
tes.email"}}}, {$project: {email: "$_id.email"}}])
  _(contacts).each(function(contact) {
    if (contact.email) {
      if (!Contacts.findOne({userId: self.userId, email: contact.email}))
{
        self.added('contacts', Random.id(), {email: contact.email, userId:
 self.userId, name: ''});
      }
    }
  });
});
```

We use underscore to loop over each contact and run some logic checks. The first check is to make sure the contact object contains an email. The next check is to make sure we don't duplicate any emails that already exist in the Contacts collection for this user. If both those logic checks pass, we then hand publish a fake contact record using self.added.

By pushing a fake contact record to the client side using DDP, our select2/autocomplete widget will automatically pick up these new results once we subscribe to this publication on the client side.

Pretty neat, huh?

What else is aggregation good for?

The other interesting thing we could do with the aggregation framework is to use setInterval on some server-side code and run this query every few minutes. Then we aggregate dump out to a collection itself. You can use the $out pipeline operator and it will create or replace the collection it spits out, after the results are ready. We could then reactively subscribe to the aggregation collection and it would work like any normal Meteor.js Collection.

Aggregations for everyone!

I think Mongo Aggregations are a great tool and you should consider using them in your app. The one thing to keep in mind is that none of the publish code I wrote is reactive. That means when a new Event is added for a user, that code won't rerun and we won't get new emails filtering into the Contacts collection on the client side. I decided that wasn't a big deal, so why bother with the extra code to make it reactive?

Use the Pipeline, Luke!

Join the Meteor Club

and get my best Meteor.js tips and tricks in your inbox!

Email Address

Full name

Are you an advanced Meteor.js developer?

- **Yes**
- **No**

Join now!

Josh Owens

It all started with an Atari 800XL, but now Josh is a ruby and javascript developer with 10 years of professional experience. His current love is Meteor.js, which he works with daily.

Cincinnati, Ohio

()

Using MongoDB aggregations to power a Meteor.js publication

56%

Share this post

≡

(http://twitter.com
/share?text=Using%20MongoDB%20aggregations%20to%20power%20a%20Meteor.js%
url=http://joshowens.me/using-mongodb-aggregations-to-power-a-meteor-js-public

(https://www.facebook.com
/sharer/sharer.php?u=http:
//joshowens.me/using-
mongodb-aggregations-
to-power-a-meteor-
js-publication/)

(https://plus.google.com
/share?url=http:
//joshowens.me/using-
mongodb-
aggregations-to-power-
a-meteor-
js-publication/)

**Join the
Meteor Club!**

15 Comments Josh Owens' thoughts Login

Recommend 1 Share Sort by Best

Join the discussion…

yauh • 7 months ago

Neat! That shows again how valuable Meteor Club is!

As it happens your article found me when I actually finished the second draft of our publications chapter where we wanted to deal with aggregations without a package. We did so using MongoInternals.defaultRemoteCollectionDriver().mongo.db (plus we made it reactive as a bonus ;) - https://github.com/meteorinact...

I am still not sure whether I prefer the way through the MongoDB core driver or your approach.

2 • Reply • Share ›

joshjowens Mod ↗ yauh • 7 months ago

Nice! Yeah, I just used Arunoda's package, he really just exposes the MongoInternals stuff in a handy call. Maybe we could figure out a meta way to make a reactive aggregate package?

1 • Reply • Share ›

Michael Bauer • 4 months ago

I know you published this a while back, but I was wondering if you had any examples of using aggregate queries in a publish where you needed the data to be reactive? One thought I had was to have the aggregate dump the data to another collection, but in reading it seems like the $out isn't supported until Mongo 2.6, and Meteor doesn't seem to support MongoDB 2.6 yet.

1 • Reply • Share ›

joshjowens Mod ↗ Michael Bauer • 4 months ago

I haven't tried $out, but Meteor runs fine against 2.6 and has for a while. 1.0.4 or 1.1 will have a big update to the mongo driver under the hood and should let you actually get up to 3.0.

Yes, my thoughts are the same as yours, just use $out to start a new collection and subscribe to that to get the data you need.

1 • Reply • Share ›

Jay ↗ joshjowens • a month ago

@Michael Bauer @joshjowens great post. I currently implement something similarly. I did not choose the $out route (i considered it, however). Instead, I decided to dump into a client-side collection.

It highly depends on the data

I **had** to do this because the aggregation would return results hyper-specific to the connection that subscribed to it.

I like aggregating in a publish function over a method for the following reason: There is no additional API that you need to expose to the client - data stays in collections, period. The client does not know whether the collection it is querying from is from a live-query or an aggregate (unless they carefully watch for pseudo-reactivity).

• Reply • Share ›

Michael Bauer ↗ joshjowens • 4 months ago

Thanks for the input, and the quick reply! Keep up the great work on teaching us noobs how to do proper Meteor.js!

• Reply • Share ›

Adam Wong • 4 months ago

It would be helpful if you'd demonstrate how to subscribe to this data on the client.

• Reply • Share ›

joshjowens Mod ↗ Adam Wong • 4 months ago

Just `Meteor.subscribe('previousInviteContacts')` in your waitOn block. Are you having isues?

• Reply • Share ›

Adam Wong ↗ joshjowens • 4 months ago

Not any more. You can learn a lot in 2 weeks!

• Reply • Share ›

hiveser ↗ Adam Wong • 3 months ago

how to access them in helpers?

(http://joshowens.me/rss/)

(https://plus.google.com

/joshuajowens/posts)

/in/joshuaowens/)