An 80/20 Guide to Mongoose Discriminators

by Valeri Karpov
@code_barbarian (http://www.twitter.com/code_barbarian)
July 24, 2015

Discriminators are a powerful yet unfortunately poorly documented (https://github.com/Automattic/mongoose/issues/2743) feature of mongoose (http://npmjs.org/package/mongoose). Discriminators enable you to store documents with slightly different schemas in the same collection and query them back in a consistent way. In this article, you'll learn about how to use discriminators to store different types of events. You'll also see how to use the aggregation framework to run rudimentary analyses.

Why Discriminators?

Suppose you're using mongoose to track 2 different types of events; a user clicking a link, and a user buying a product. Storing both types of event in the same collection would be handy so you could use the MongoDB aggregation framework (http://thecodebarbarian.com/2015/06/26/crunching-nba-finals-history-with-mongodb) for tasks like calculating how many users that clicked on a certain link bought a certain product. However, these two event types have slightly different schema requirements. A `ClickedLinkEvent` should track the URL the user clicked on and the page they were on when they clicked it, but these fields would be irrelevant for the `PurchasedEvent` schema. Instead, the `PurchasedEvent` schema should track the product id and the final purchase price.

If you didn't know about discriminators, you might implement this as a single schema using mongoose's `Mixed` type (http://mongoosejs.com/docs/schematypes.html#mixed). The `Mixed` type is mongoose's wildcard type - mongoose doesn't run casting or validation on `Mixed` fields.

```
var eventSchema = new mongoose.Schema({
  // The type of event
  kind: {
    type: String,
    required: true,
    enum: ['ClickedLink', 'Purchased']
  },
  // The time the event took place
  time: {
    type: Date,
    default: Date.now
  },
  /* Arbitrary data associated with the event.
   * `{}` corresponds to `Mixed` type in mongoose,
   * so no validation is run on this field */
  data: {}
});

var Event = mongoose.model('Event', eventSchema);
```

Unfortunately, using `Mixed` defeats the purpose of using mongoose in the first place. If you're not going to validate the data at all, you should consider just using the MongoDB driver (https://github.com/mongodb/node-mongodb-native) directly.

```
var e = new Event({
  kind: 'ClickedLink',
  data: { badField: 'abc' }
});

// No error, 'badField' is perfectly valid
assert.ifError(e.validateSync());
```

The `discriminator()` Function

Suppose you created a general event model that looked like what you see below.

```
var options = { discriminatorKey: 'kind' };

var eventSchema = new mongoose.Schema(
  {
    // The time the event took place
    time: {
      type: Date,
      default: Date.now
    }
  },
  options);
var Event = mongoose.model('Event', eventSchema);
```

The `discriminatorKey` option tells mongoose to add a path to the schema called 'kind' and use it to track which type of document this is. For instance, suppose you declared two discriminators, `ClickedLinkEvent` and `PurchasedEvent`, as shown below.

```
// ClickedLinkEvent
var clickedEventSchema = new mongoose.Schema(
  {
    from: { type: String, required: true },
    to: { type: String, required: true }
  },
  options);
var ClickedLinkEvent = Event.discriminator('ClickedLink',
  clickedEventSchema);

// PurchasedEvent
var purchasedSchema = new mongoose.Schema(
  {
    product: { type: mongoose.Schema.Types.ObjectId }
  },
  options);
var PurchasedEvent = Event.discriminator('Purchased',
  purchasedSchema);
```

The `ClickedLinkEvent` and `PurchasedEvent` discriminators work almost exactly like regular mongoose models. For instance, you can create a new `ClickedLinkEvent` and mongoose validation will ensure that the `to` field is specified.

```
var e = new ClickedLinkEvent({
  from: 'http://www.google.com'
});

console.log(e.kind); // Prints 'ClickedLink'
console.log(e.time); // Prints current time

var error = e.validateSync();
assert.ok(error);
// Prints ['to'] because no 'to' link specified
console.log(Object.keys(error.errors));
```

Note that the schema for the `ClickedLinkEvent` discriminator is the **union** of `eventSchema` and `clickedEventSchema`. That is, the schema for `ClickedLinkEvent` has:

- The discriminator field `kind`
- The `time` field from `eventSchema`
- The `from` and `to` from `clickedEventSchema`

However, `ClickedLinkEvent` is different from a conventional model. In particular, documents that are instances of `ClickedLinkEvent` and `PurchasedEvent` get stored in the 'events' collection. Querying with the `Event` model can then find **all** documents that are of either type.

```
ClickedLinkEvent.create({ from: 'abc', to: '123' }, function(err) {
  assert.ifError(err);
  var product = { product: '000000000000000000000000c' };
  PurchasedEvent.create(product, function(err) {
    assert.ifError(err);

    Event.find({}, function(error, events) {
      assert.ifError(error);

      // Got back both events!
      assert.equal(events.length, 2);
      assert.equal(events[0].kind, 'ClickedLink');
      assert.equal(events[1].kind, 'Purchased');

      // `from` field gets pulled in too
      assert.equal(events[0].from, 'abc');

      example2();
    });
  });
});
```

For instance, if you were to look at the 'events' collection in MongoDB after running the above code, you'd see:

```
> db.events.find().pretty()
{
    "_id" : ObjectId("55afdeeb3b91d05562821ab4"),
    "from" : "abc",
    "to" : "123",
    "kind" : "ClickedLink",
    "time" : ISODate("2015-07-22T18:20:27.248Z"),
    "__v" : 0
}
{
    "_id" : ObjectId("55afdeeb3b91d05562821ab5"),
    "product" : ObjectId("00000000000000000000000c"),
    "kind" : "Purchased",
    "time" : ISODate("2015-07-22T18:20:27.378Z"),
    "__v" : 0
}
```

However, if you use the `ClickedLinkEvent` discriminator to query, you'll get back just the documents that have `kind === 'ClickedLink'`.

```
ClickedLinkEvent.find({}, function(error, events) {
  assert.ifError(error);
  assert.equal(events.length, 1);
  assert.equal(events[0].kind, 'ClickedLink');

  assert.equal(events[0].from, 'abc');
});
```

Using the Aggregation Framework

In mongoose, aggregations are discriminator-aware, so you can do tasks like 'find the most commonly purchased products' without even being aware of the discriminator's existence.

```
// Get back top 5 most purchased products
PurchasedEvent.aggregate([
  { $group: { _id: '$product', count: { $inc: 1 } } },
  { $sort: { count: -1 } },
  { $limit: 5 }
], callback);
```

However, you have the additional ability to switch back and forth between aggregating across all events versus aggregating across just `PurchasedEvent` documents without any joins. For instance, suppose you wanted to compare the number of users that purchased product `00000000000000000000000c` and the number of users that visited the page `/product/00000000000000000000000c`. You can achieve this with a single aggregation:

```
Event.aggregate([
  {
    $match: {
      $or: [
        { kind: 'ClickedLink', to: '/product/000000000000000000000000c' },
        { kind: 'Purchased', product: mongoose.Types.ObjectId('000000000000000000000000c') }
      ]
    }
  },
  {
    $group: {
      _id: '$kind',
      count: { $sum: 1 }
    }
  }
]);
```

Conclusion

Discriminators are a powerful mongoose feature that enable you to store similar documents in the same collection with different schema constraints. They are often handy in situations when you're tempted to just use a `Mixed` type and bypass validation entirely. In particular, for applications like events tracking and user permissions, discriminators can be indispensable.

Found a typo or error? Open up a pull request! This post is available as markdown on Github (https://github.com/vkarpov15/thecodebarbarian.com/blob/master/lib/posts/20150724_mongoose_discriminators.md)

"ColoumnName" : "alpha"

see more

• Reply • Share ›

vkarpov15 Mod ➚ Darshan A.N. • 3 months ago

I think you mean to do `Profile.findByIdAndUpdate(profileId, { $push: {EventMetrics:{Metrics}})`

• Reply • Share ›

Darshan A.N. ➚ vkarpov15 • 3 months ago

Yes, Thats right, I wanted to do like that, But i have so many eventmetrics inside a profile collection, in which i want to append Metrics array into particular EventMetrics array.

For ex. My data looks like this(one of Profileid doc):

{
"Name": "Proove",
"EventMetrics": [
{
"Metrics": [
{
"_id": "5afbdd34fd2ddc2c6063c4d5",
"Name": "First",
"ColoumnName": "alpha",
"GroupBy": "kuch bhi",
"Function": "Min",
"Window": 900,
"Delay": 40

see more

• Reply • Share ›

vkarpov15 Mod ➚ Darshan A.N. • 3 months ago

In that case you'll want to use `findOneAndUpdate()` with array filters: http://thecodebarbarian.com...

• Reply • Share ›

Darshan A.N. • 4 months ago

Hi,

I m trying to use populate( ) in node js.

Well I m trying to access, objectId of one collection into another collection.

for eg., i have collections called Project and events,

where i have schema like this.

Project schema:
const projectSchema = mongoose.Schema({
_id: mongoose.Schema.Types.ObjectId,
projectName: { type: String, required: true, unique: true },
projectDescription: { type: String, required: false },
dimensions: { type: [], required: false },
events: {type: mongoose.Schema.Types.ObjectId, ref: 'EnrichedEvent'},
customers: { type: [], required: false }
});

see more

• Reply • Share ›

Darshan A.N. • 5 months ago

This article's corresponding screen casting is not working.

• Reply • Share ›

vkarpov15 Mod ➚ Darshan A.N. • 5 months ago

Thanks for the heads up, I removed the dead link in https://github.com/vkarpov1... . No idea why youtube removed that video, and I no longer have the screencast video.

• Reply • Share ›

Darshan A.N. ➚ vkarpov15 • 5 months ago

Hey,

I have some schemas, which i can't process properly.

```
const mongoose = require('mongoose');

const delSchema = mongoose.Schema({

name: { type: String, required: true },

type: {type:String, enum:[ "dimension", "metric","ID"], required:true },

position : {type: Number, required: true},

dataType : {type:String, enum:["int", "float","string"], required:true }

});

const jsSchema = mongoose.Schema({

name: { type: String, required: true }
```

see more

• Reply • Share ›

vkarpov15 Mod ➚ Darshan A.N. • 5 months ago

That's strange, the schemas look right so this might be a bug. What does the raw data look like? Like how are you constructing this document?

• Reply • Share ›

Darshan A.N. ➚ vkarpov15 • 5 months ago

Hi,

my output should look like something like this:

```
{
"project": "5ab8ccbff9445245d10cad85",
"description":"source",
"source" : "gottilla",
"name": "Enriched EVENT",
"type": "Enriched",
"parentId": "uiop",
"delimiter": ",",
"Eschema": [
"delSchema" : [
"name": "darshan",
"type": "dimension",
"position": 7
"datatype": "int"
```

see more

• Reply • Share ›

vkarpov15 Mod ➚ Darshan A.N. • 5 months ago

I get what the output should look like. What does the input look like?

Reply • Share ›

Darshan A.N. ➚ vkarpov15 • 4 months ago

Hi,

Input should look like this:
NOTE: i m using Postman as user agent

{
"project": "5ab8ccbff9445245d10cad85",
"description":"source",
"source" : "gottilla",
"name": "Enriched EVENT",
"type": "Enriched",
"parentId": "uiop",
"delimiter": ",",
"Eschema": ["delSchema" : [// if i enter fwschema, it should follow the fwschema only
"name": "darshan", // example: "startIndex" : 5
"type": "dimension", // "endIndex" : 9
"position": 7 // Please look at my schema in my first comment
"datatype": "int"
]

]
}
2 Reply • Share ›

vkarpov15 Mod ➚ Darshan A.N. • 4 months ago

"Eschema": ["delSchema" : [ ]]

That should be

"Eschema": {
"name": "darshan", // example: "startIndex" : 5
"type": "dimension", // "endIndex" : 9
"position": 7 // Please look at my schema in my first comment
"datatype": "int"
}
1 Reply • Share ›

Darshan A.N. ➚ vkarpov15 • 4 months ago

Its working, but only for one of the schema, cant post other schema information.:(anyways Thanks buddy.:) i guess i need to validate it in UI Part.

Reply • Share ›

Ping Zhang • 2 years ago

Thanks for sharing!

Reply • Share ›

ga19892 • 2 years ago

Thanks, very good post

Reply • Share ›

Gitesh Gupta • 3 years ago

This is an Amazing post, exactly what is needed to solve mongo DB's Schema inheritance!

Reply • Share ›

mlgbx • 3 years ago

Thanks for this great post!

Reply • Share ›