Seneca
SCHOOL OF INFORMATION AND COMMUNICATIONS TECHNOLOGY

BTI425

Web Programming for Apps and Services

Notes | Weekly
Resources | Graded work
Policies | Standards
Professors & addendum | Code examples

Data associations or relations introduction

This document discusses some introductory data assocations or relations topics.

Associated or related data

Let's talk more about associated or related data.

Almost always, an entity collection will be somehow associated or related to another entity collection. The kinds of associations or relationships can include the following:

- One to many (probably the best-known and best-understood)
- One to one
- Many to many
- Self-referenceing one-to-many
- Self-referencing one-to-one

Over time, we'll discuss these, but to get started, we'll probably end up using *one to many* associations or relationships.

Embedded documents

In a recent class/session, you learned about *subdocuments* (aka embedded documents) as one way to compose a MongoDB-stored document that has an embedded "subdocument". For example, a "term" (word) document has a collection of "definition" subdocuments.

This *embedded documents* organization scheme is discussed in the MongoDB documentation article:
Model One-to-Many Relationships with Embedded Documents

As you would expect, the *document* will be described by a Mongoose schema in your web API.

In addition, the *subdocument* will *also* be desribed by a Mongoose schema in your web API. (Although it is possible to declare the subdocument schema inline in the hosting document schema, we enjoy some future benefits by creating a separate schema.)

Use this design for the following scenarios or preferences:

- The query strategy will often or always want to fetch a document with all of its associated or related data
- The data in the sub-documents is typically stable and unchanging; almost archival in nature
- The amount of data in the sub-documents is relatively or contextually not too large

Can I see an example?

A how-to document has been prepared to support this topic.

Document references

If your scenario is different, an alternative (and perhaps more familiar to those with relational database management system experience) organization scheme is *document references*:
Model One-to-Many Relationships with Document References

Use this design for the following scenarios or preferences:

- Data repetition would be a bad idea (too much for example)
- Data used in a frequently-updated transactional manner
- There is a flexible and unpredictable query strategy, where it is likely that each entity in the association or relation may be separately queried for whatever purpose

Which entity gets the document reference?

For the following discussion, assume that we're designing the structure of two associated or related entities. Into which entity do we add the reference? Here's some getting-started guidance (but your scenario may result in adjusted guidance):

- The growth of the data in the association or relation determines where to store the reference
    - We're referring to general growth, or rate of growth
- One of the entity collections in the association or relationship will typically naturally have many more items (by a large factor) than the other entity collection
- Therefore, store the reference in the entity that's in the collection with the most items

For example, assume that each smartphone "maker" (e.g. Apple) will have a large number of smartphone "models" available for sale (e.g. iPhone Xs, iPhone Xr, iPhone 8, iPhone 7). In that situation, in the "model" entity, add a reference to the "maker".

What does the reference look like?

What does the reference look like? What is its format and data type? Here's our guidance:

- Its name should be based on the name of the associated or related collection, and end with "`Id`" (yes, there are other naming schemes possible, but start with this, and adjust when you have more experience)
- Its value must be the MongoDB unique identifier (the 24-character ObjectId) of the associated or related object

For example, assume that we're looking at a smartphone "models" object (from above). In the MongoDB database itself, when using the Compass tool, the reference looks like this:
`makerId: ObjectId("507f1f77bcf86cd799439011")`

When delivered to a requestor, as JSON (via a web API), the reference looks like this:
`"makerId": "507f1f77bcf86cd799439011"`

Can I see an example?

A how-to document has been prepared to support this topic.

Query strategy

For documents with embedded (sub-)documents, simply query (get) the items that match whatever criteria you have (all, some, specific by identifier). You'll get back all the associated or related data.

For the other scenario, there are ways of looking at querying. Let's continue to use the "maker" and "model" example above.

Assume that you want to query all, some, or one "maker". Simply get what you want from the "maker" collection, without considering any other collections. Easy.

Next, assume that you want to query for all, some, or one "model". Similar to above, get what you want from the "models" collection.

Next, assume that you want to query for all, some, or one "model" from a specific "maker". In your query statement, you will have two search criteria:

1. The "maker" identifier (and you must have that beforehand)
2. If desired, the "models" matching criteria that meets your need (e.g. screen size larger than 4.0 inches)

In a variation on the previous query, assume that you want the "maker" data *included* in the result, when querying the "models" collection. Well, the query must include a command (`populate()`) that will *de-reference* the associated or related "maker" for each "models" item returned. We'll see that in a future code example.

Finally, assume that you want to query for one "maker" including all (or some of) its "models". This is a two-step query, and each result is returned separately. (You can then do whatever you want in your code - including combining them into a new data structure.) Here's the two-step query strategy:

1. Fetch the specifically-desired "maker".
2. Using the "maker" identifier, fetch all (or some) matching "models".

Happy coding!

© 2020 - Seneca School of ICT