Documents

Mongoose documents represent a one-to-one mapping to documents as stored in MongoDB. Each document is an instance of its Model.

Documents vs Models

Document and Model are distinct classes in Mongoose. The Model class is a subclass of the Document class. When you use the Model constructor, you create a new document.

```
const MyModel = mongoose.model('Test', new Schema({ name: String }));
const doc = new MyModel();

doc instanceof MyModel; // true
doc instanceof mongoose.Model; // true
doc instanceof mongoose.Document; // true
```

In Mongoose, a "document" generally means an instance of a model. You should not have to create an instance of the Document class without going through a model.

Retrieving

When you load documents from MongoDB using model functions like `findOne()`, you get a Mongoose document back.

```
const doc = await MyModel.findOne();

doc instanceof MyModel; // true
doc instanceof mongoose.Model; // true
doc instanceof mongoose.Document; // true
```

Updating Using `save()`

Mongoose documents track changes. You can modify a document using vanilla JavaScript assignments and Mongoose will convert it into MongoDB update operators.

```
doc.name = 'foo';

// Mongoose sends an `updateOne({ _id: doc._id }, { $set: { name: 'foo' } })`
// to MongoDB.
await doc.save();
```

The `save()` method returns a promise. If `save()` succeeds, the promise resolves to the document that was saved.

```
doc.save().then(savedDoc => {
  savedDoc === doc; // true
});
```

If the document with the corresponding `_id` is not found, Mongoose will report a `DocumentNotFoundError`:

```
const doc = await MyModel.findOne();

// Delete the document so Mongoose won't be able to save changes
await MyModel.deleteOne({ _id: doc._id });

doc.name = 'foo';
await doc.save(); // Throws DocumentNotFoundError
```

Updating Using Queries

The `save()` function is generally the right way to update a document with Mongoose. With `save()`, you get full validation and middleware.

For cases when `save()` isn't flexible enough, Mongoose lets you create your own MongoDB updates with casting, middleware, and limited validation.

```
// Update all documents in the `mymodels` collection
await MyModel.updateMany({}, { $set: { name: 'foo' } });
```

Note that `update()`, `updateMany()`, `findOneAndUpdate()`, etc. do not execute `save()` middleware. If you need save middleware and full validation, first query for the document and then `save()` it.

Validating

Documents are casted and validated before they are saved. Mongoose first casts values to the specified type and then validates them. Internally, Mongoose calls the document's `validate()` method before saving.

```
const schema = new Schema({ name: String, age: { type: Number, min: 0 } });
const Person = mongoose.model('Person', schema);

let p = new Person({ name: 'foo', age: 'bar' });
// Cast to Number failed for value "bar" at path "age"
await p.validate();

let p2 = new Person({ name: 'foo', age: -1 });
```

```
// Path `age` (-1) is less than minimum allowed value (0).
await p2.validate();
```

Mongoose also supports limited validation on updates using the `runValidators` option. Mongoose casts parameters to query functions like `findOne()`, `updateOne()` by default. However, Mongoose does *not* run validation on query function parameters by default. You need to set `runValidators: true` for Mongoose to validate.

```
// Cast to number failed for value "bar" at path "age"
await Person.updateOne({}, { age: 'bar' });

// Path `age` (-1) is less than minimum allowed value (0).
await Person.updateOne({}, { age: -1 }, { runValidators: true });
```

Read the validation guide for more details.

Overwriting

There are 2 different ways to overwrite a document (replacing all keys in the document). One way is to use the `Document#overwrite()` function followed by `save()`.

```
const doc = await Person.findOne({ _id });

// Sets `name` and unsets all other properties
doc.overwrite({ name: 'Jean-Luc Picard' });
await doc.save();
```

The other way is to use `Model.replaceOne()`.

```
// Sets `name` and unsets all other properties
await Person.replaceOne({ _id }, { name: 'Jean-Luc Picard' });
```

Next Up

Now that we've covered Documents, let's take a look at Subdocuments.