mongoose

Working With Dates

SPONSOR Dell Small Business — Shop Dell Small Business and get doorbuster savings up to 50% off.

Here's how you declare a path of type `Date` with a Mongoose schema:

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

const userSchema = new mongoose.Schema({
  name: String,
  // `lastActiveAt` is a date
  lastActiveAt: Date
});
const User = mongoose.model('User', userSchema);
```

When you create a user document, Mongoose will cast the value to a native JavaScript date using the `Date()` constructor.

```
const user = new User({
  name: 'Jean-Luc Picard',
  lastActiveAt: '2002-12-09'
});
user.lastActiveAt instanceof Date; // true
```

An invalid date will lead to a `CastError` when you validate the document.

```
const user = new User({
  name: 'Jean-Luc Picard',
  lastActiveAt: 'not a date'
});
user.lastActiveAt instanceof Date; // false
user.validateSync().errors['lastActiveAt']; // CastError
```

Validators

Dates have two built-in validators: `min` and `max`. These validators will report a `ValidatorError` if the given date is strictly less than `min` or strictly greater than `max`.

```
const episodeSchema = new mongoose.Schema({
  title: String,
  airedAt: {
    type: Date,
    // The dates of the first and last episodes of
    // Star Trek: The Next Generation
    min: '1987-09-28',
    max: '1994-05-23'
  }
});
const Episode = mongoose.model('Episode', episodeSchema);

const ok = new Episode({
  title: 'Encounter at Farpoint',
  airedAt: '1987-09-28'
});
ok.validateSync(); // No error

const bad = new Episode({
  title: 'What You Leave Behind',
  airedAt: '1999-06-02'
});
bad.airedAt; // "1999-06-02T00:00:00.000Z"

// Path `airedAt` (Tue Jun 01 1999 20:00:00 GMT-0400 (EDT)) is after
// maximum allowed value (Sun May 22 1994 20:00:00 GMT-0400 (EDT)).
bad.validateSync();
```

Querying

MongoDB supports querying by date ranges and sorting by dates. Here's some examples of querying by dates, date ranges, and sorting by date:

```
// Find episodes that aired on this exact date
return Episode.find({ airedAt: new Date('1987-10-26') }).
  then(episodes => {
    episodes[0].title; // "Where No One Has Gone Before"
    // Find episodes within a range of dates, sorted by date ascending
    return Episode.
      find({ airedAt: { $gte: '1987-10-19', $lte: '1987-10-26' } }).
      sort({ airedAt: 1 });
  }).
  then(episodes => {
    episodes[0].title; // "The Last Outpost"
    episodes[1].title; // "Where No One Has Gone Before"
  });
```

Casting Edge Cases

Date casting has a couple small cases where it differs from JavaScript's native date parsing. First, Mongoose looks for a `valueOf()` function on the given object, and calls `valueOf()` before casting the date. This means Mongoose can cast moment objects to dates automatically.

```
const moment = require('moment');
const user = new User({
  name: 'Jean-Luc Picard',
  lastActiveAt: moment.utc('2002-12-09')
});
user.lastActiveAt; // "2002-12-09T00:00:00.000Z"
```

By default, if you pass a numeric string to the Date constructor, JavaScript will attempt to convert it to a year.

```
new Date(1552261496289); // "2019-03-10T23:44:56.289Z"
new Date('1552261496289'); // "Invalid Date"
new Date('2010'); // 2010-01-01T00:00:00.000Z
```

Mongoose converts numeric strings that contain numbers outside the range of representable dates in JavaScript and converts them to numbers before passing them to the date constructor.

```
require: Date Tutorial.*Example 1.4.3]
```

Timezones

MongoDB stores dates as 64-bit integers, which means that Mongoose does **not** store timezone information by default. When you call `Date#toString()` , the JavaScript runtime will use your OS' timezone.