Plugins

Schemas are pluggable, that is, they allow for applying pre-packaged capabilities to extend their functionality. This is a very powerful feature.

Example

Plugins are a tool for reusing logic in multiple schemas. Suppose you have several models in your database and want to add a `loadedAt` property to each one. Just create a plugin once and apply it to each `Schema`:

```
// loadedAt.js
module.exports = function loadedAtPlugin(schema, options) {
  schema.virtual('loadedAt').
    get(function() { return this._loadedAt; }).
    set(function(v) { this._loadedAt = v; });

  schema.post(['find', 'findOne'], function(docs) {
    if (!Array.isArray(docs)) {
      docs = [docs];
    }
    const now = new Date();
    for (const doc of docs) {
      doc.loadedAt = now;
    }
  });
};

// game-schema.js
const loadedAtPlugin = require('./loadedAt');
const gameSchema = new Schema({ ... });
gameSchema.plugin(loadedAtPlugin);

// player-schema.js
const loadedAtPlugin = require('./loadedAt');
const playerSchema = new Schema({ ... });
playerSchema.plugin(loadedAtPlugin);
```

We just added last-modified behavior to both our `Game` and `Player` schemas and declared an index on the `lastMod` path of our Games to boot. Not bad for a few lines of code.

Global Plugins

Want to register a plugin for all schemas? The mongoose singleton has a `.plugin()` function that registers a plugin for every schema. For example:

```
const mongoose = require('mongoose');
mongoose.plugin(require('./loadedAt'));

const gameSchema = new Schema({ ... });
const playerSchema = new Schema({ ... });
// `loadedAtPlugin` gets attached to both schemas
const Game = mongoose.model('Game', gameSchema);
const Player = mongoose.model('Player', playerSchema);
```

Apply Plugins Before Compiling Models

Because many plugins rely on middleware, you should make sure to apply plugins **before** you call `mongoose.model()` or `conn.model()`. Otherwise, any middleware the plugin registers won't get applied.

```
// loadedAt.js
module.exports = function loadedAtPlugin(schema, options) {
  schema.virtual('loadedAt').
    get(function() { return this._loadedAt; }).
    set(function(v) { this._loadedAt = v; });

  schema.post(['find', 'findOne'], function(docs) {
    if (!Array.isArray(docs)) {
      docs = [docs];
    }
    const now = new Date();
    for (const doc of docs) {
      doc.loadedAt = now;
    }
  });
};

// game-schema.js
const loadedAtPlugin = require('./loadedAt');
const gameSchema = new Schema({ ... });
const Game = mongoose.model('Game', gameSchema);

// `find()` and `findOne()` hooks from `loadedAtPlugin()` won't get applied
// because `mongoose.model()` was already called!
gameSchema.plugin(loadedAtPlugin);
```

Officially Supported Plugins

The Mongoose team maintains several plugins that add cool new features to Mongoose. Here's a couple:

- mongoose-autopopulate: Always `populate()` certain fields in your Mongoose schemas.
- mongoose-lean-virtuals: Attach virtuals to the results of Mongoose queries when using `.lean()`.
- mongoose-cast-aggregation

You can find a full list of officially supported plugins on Mongoose's plugins search site.

Community!

Not only can you re-use schema functionality in your own projects, but you also reap the benefits of the Mongoose community as well. Any plugin published to npm and with 'mongoose' as an npm keyword will show up on our search results page.