Virtualtype

VirtualType()

Parameters

- options «Object»
  - [options.ref] «string|function» if `ref` is not nullish, this becomes a populated virtual
  - [options.localField] «string|function» the local field to populate on if this is a populated virtual.
  - [options.foreignField] «string|function» the foreign field to populate on if this is a populated virtual.
  - [options.justOne=false] «boolean» by default, a populated virtual is an array. If you set `justOne`, the populated virtual will be a single doc or `null`.
  - [options.getters=false] «boolean» if you set this to `true`, Mongoose will call any custom getters you defined on this virtual
  - [options.count=false] «boolean» if you set this to `true`, `populate()` will set this virtual to the number of populated documents, as opposed to the documents themselves, using `Query#countDocuments()`
  - [options.match=null] «Object|Function» add an extra match condition to `populate()`
  - [options.limit=null] «Number» add a default `limit` to the `populate()` query
  - [options.skip=null] «Number» add a default `skip` to the `populate()` query
  - [options.perDocumentLimit=null] «Number» For legacy reasons, `limit` with `populate()` may give incorrect results because it only executes a single query for every document being populated. If you set `perDocumentLimit`, Mongoose will ensure correct `limit` per document by executing a separate query for each document to `populate()`. For example, `.find().populate({ path: 'test', perDocumentLimit: 2 })` will execute 2 additional queries if `.find()` returns 2 documents.

- [options.options=null] «Object» Additional options like `limit` and `lean`.

VirtualType constructor

This is what mongoose uses to define virtual attributes via `Schema.prototype.virtual`.

Example:

```
const fullname = schema.virtual('fullname');
fullname instanceof mongoose.VirtualType // true
```

VirtualType.prototype.applyGetters()

Parameters

- value «Object»
- doc «Document» The document this virtual is attached to

Returns:

- «any» the value after applying all getters

Applies getters to `value`.

VirtualType.prototype.applySetters()

Parameters

- value «Object»
- doc «Document»

Returns:

- «any» the value after applying all setters

Applies setters to `value`.

VirtualType.prototype.get()

Parameters

- VirtualType, «Function(Any|» Document)} fn

Returns:

- «VirtualType» this

Adds a custom getter to this virtual.

Mongoose calls the getter function with the below 3 parameters.

- `value`: the value returned by the previous getter. If there is only one getter, `value` will be `undefined`.
- `virtual`: the virtual object you called `.get()` on
- `doc`: the document this virtual is attached to. Equivalent to `this`.

Example:

```
const virtual = schema.virtual('fullname');
virtual.get(function(value, virtual, doc) {
  return this.name.first + ' ' + this.name.last;
});
```

VirtualType.prototype.set()

Parameters

- VirtualType, «Function(Any|» Document)} fn

Returns:

- «VirtualType» this

Adds a custom setter to this virtual.

Mongoose calls the setter function with the below 3 parameters.

- `value`: the value being set
- `virtual`: the virtual object you're calling `.set()` on
- `doc`: the document this virtual is attached to. Equivalent to `this`.

Example:

```
const virtual = schema.virtual('fullname');
virtual.set(function(value, virtual, doc) {
  const parts = value.split(' ');
  this.name.first = parts[0];
  this.name.last = parts[1];
});

const Model = mongoose.model('Test', schema);
const doc = new Model();
// Calls the setter with `value = 'Jean-Luc Picard'`
doc.fullname = 'Jean-Luc Picard';
doc.name.first; // 'Jean-Luc'
doc.name.last; // 'Picard'
```