mongoose

Document

Document.prototype.$errors

Type:

- «property»

Hash containing current validation $errors.

Document.prototype.$getAllSubdocs()

Get all subdocs (by bfs)

Document.prototype.$ignore()

Parameters

- path «String» the path to ignore

Don't run validation on this path or persist changes to this path.

Example:

```
doc.foo = null;
doc.$ignore('foo');
doc.save(); // changes to foo will not be persisted and validators won't be run
```

Document.prototype.$isDefault()

Parameters

- [path] «String»

Returns:

- «Boolean»

Checks if a path is set to its default.

Example

```
MyModel = mongoose.model('test', { name: { type: String, default: 'Val '} });
const m = new MyModel();
m.$isDefault('name'); // true
```

Document.prototype.$isDeleted()

Parameters

- [val] «Boolean» optional, overrides whether mongoose thinks the doc is deleted

Returns:

- «Boolean» whether mongoose thinks this doc is deleted.

Getter/setter, determines whether the document was removed or not.

Example:

```
const product = await product.remove();
product.$isDeleted(); // true
product.remove(); // no-op, doesn't send anything to the db

product.$isDeleted(false);
product.$isDeleted(); // false
product.remove(); // will execute a remove against the db
```

Document.prototype.$isEmpty()

Returns:

- «Boolean»

Returns true if the given path is nullish or only contains empty objects. Useful for determining whether this subdoc will get stripped out by the minimize option.

Example:

```
const schema = new Schema({ nested: { foo: String } });
const Model = mongoose.model('Test', schema);
const doc = new Model({});
doc.$isEmpty('nested'); // true
doc.nested.$isEmpty(); // true

doc.nested.foo = 'bar';
doc.$isEmpty('nested'); // false
doc.nested.$isEmpty(); // false
```

Document.prototype.$isNew

Type:

- «property»

Boolean flag specifying if the document is new.

Document.prototype.$locals

Type:

- «property»

Empty object that you can use for storing properties on the document. This is handy for passing data to middleware without conflicting with Mongoose internals.

Example:

```
schema.pre('save', function() {
  // Mongoose will set `isNew` to `false` if `save()` succeeds
  this.$locals.wasNew = this.isNew;
});

schema.post('save', function() {
  // Prints true if `isNew` was set before `save()`
  console.log(this.$locals.wasNew);
});
```

Document.prototype.$markValid()

Parameters

- path «String» the field to mark as valid

Marks a path as valid, removing existing validation errors.

Document.prototype.$op

Type:

- «property»

A string containing the current operation that Mongoose is executing on this document. May be `null`, `'save'`, `'validate'`, or `'remove'`.

Example:

```
const doc = new Model({ name: 'test' });
doc.$op; // null

const promise = doc.save();
doc.$op; // 'save'

await promise;
doc.$op; // null
```

Document.prototype.$parent()

Alias for `parent()`. If this document is a subdocument or populated document, returns the document's parent. Returns `undefined` otherwise.

Document.prototype.$session()

Parameters

- [session] «ClientSession» overwrite the current session

Returns:

- «ClientSession»

Getter/setter around the session associated with this document. Used to automatically set `session` if you `save()` a doc that you got from a query with an associated session.

Example:

```
const session = MyModel.startSession();
const doc = await MyModel.findOne().session(session);
doc.$session() === session; // true
doc.$session(null);
doc.$session() === null; // true
```

If this is a top-level document, setting the session propagates to all child docs.

Document.prototype.$set()

Parameters

- path «String|Object» path or object of key/vals to set
- val «Any» the value to set
- [type] «Schema|String|Number|Buffer|*» optionally specify a type for "on-the-fly" attributes
- [options] «Object» optionally specify options that modify the behavior of the set

Alias for `set()`, used internally to avoid conflicts

Document.prototype.$where

Type:

- «property»

Set this property to add additional query filters when Mongoose saves this document and `isNew` is false.

Example:

```
// Make sure `save()` never updates a soft deleted document.
schema.pre('save', function() {
  this.$where = { isDeleted: false };
});
```

Document.prototype.depopulate()

Parameters

- path «String»

Returns:

- «Document» this

Takes a populated field and returns it to its unpopulated state.

Example:

```
Model.findOne().populate('author').exec(function (err, doc) {
  console.log(doc.author.name); // Dr.Seuss
  console.log(doc.depopulate('author'));
  console.log(doc.author); // '5144cf8050f071d979c118a7'
})
```

If the path was not provided, then all populated fields are returned to their unpopulated state.

Document.prototype.directModifiedPaths()

Returns:

- «Array»

Returns the list of paths that have been directly modified. A direct modified path is a path that you explicitly set, whether via `doc.foo = 'bar'`, `Object.assign(doc, { foo: 'bar' })`, or `doc.set('foo', 'bar')`.

A path `a` may be in `modifiedPaths()` but not in `directModifiedPaths()` because a child of `a` was directly modified.

Example

```
const schema = new Schema({ foo: String, nested: { bar: String } });
const Model = mongoose.model('Test', schema);
await Model.create({ foo: 'original', nested: { bar: 'original' } });

const doc = await Model.findOne();
doc.nested.bar = 'modified';
doc.directModifiedPaths(); // ['nested.bar']
doc.modifiedPaths(); // ['nested', 'nested.bar']
```

Document.prototype.equals()

Parameters

- doc «Document» a document to compare

Returns:

- «Boolean»

Returns true if this document is equal to another document.

Documents are considered equal when they have matching `_id`s, unless neither document has an `_id`, in which case this function falls back to using `deepEqual()`.

Document.prototype.errors

Type:

- «property»

Hash containing current validation errors.

Document.prototype.get()

Parameters

- path «String»
- [type] «Schema|String|Number|Buffer|*» optionally specify a type for on-the-fly attributes
- [options] «Object»
  - [options.virtuals=false] «Boolean» Apply virtuals before getting this path
  - [options.getters=true] «Boolean» If false, skip applying getters and just get the raw value

Returns the value of a path.

Example

```
// path
doc.get('age') // 47

// dynamic casting to a string
doc.get('age', String) // "47"
```

Document.prototype.getChanges()

Returns:

- «Object»

Returns the changes that happened to the document in the format that will be sent to MongoDB.

Example:

```
const userSchema = new Schema({
  name: String,
  age: Number,
  country: String
});
const User = mongoose.model('User', userSchema);
const user = await User.create({
  name: 'Hafez',
  age: 25,
  country: 'Egypt'
});

// returns an empty object, no changes happened yet
user.getChanges(); // { }

user.country = undefined;
user.age = 26;

user.getChanges(); // { $set: { age: 26 }, { $unset: { country: 1 } } }

await user.save();

user.getChanges(); // { }
```

Modifying the object that `getChanges()` returns does not affect the document's change tracking state. Even if you `delete user.getChanges().$set`, Mongoose will still send a `$set` to the server.

Document.prototype.id

Type:

- «property»

The string version of this documents _id.

Note:

This getter exists on all documents by default. The getter can be disabled by setting the `id` option of its `Schema` to false at construction time.

```
new Schema({ name: String }, { id: false });
```

Document.prototype.init()

Parameters

- doc «Object» document returned by mongo

Initializes the document without setters or marking anything modified.

Called internally after a document is returned from mongodb. Normally, you do **not** need to call this function on your own.

This function triggers `init` middleware. Note that `init` hooks are synchronous.

Document.prototype.inspect()

Helper for console.log

Document.prototype.invalidate()

Parameters

- path «String» the field to invalidate. For array elements, use the `array.i.field` syntax, where `i` is the 0-based index in the array.
- errorMsg «String|Error» the error which states the reason `path` was invalid
- value «Object|String|Number|any» optional invalid value
- [kind] «String» optional `kind` property for the error

Returns:

- «ValidationError» the current ValidationError, with all currently invalidated paths

Marks a path as invalid, causing validation to fail.

The `errorMsg` argument will become the message of the `ValidationError`.

The `value` argument (if passed) will be available through the `ValidationError.value` property.

```
doc.invalidate('size', 'must be less than 20', 14);

doc.validate(function (err) {
  console.log(err)
  // prints
  { message: 'Validation failed',
    name: 'ValidationError',
    errors:
     { size:
        { message: 'must be less than 20',
          name: 'ValidatorError',
          path: 'size',
          type: 'user defined',
          value: 14 } } }
})
```

Document.prototype.isDirectModified()

Parameters

- path «String|Array<String>»

Returns:

- «Boolean»

Returns true if `path` was directly set and modified, else false.

Example

```
doc.set('documents.0.title', 'changed');
doc.isDirectModified('documents.0.title') // true
doc.isDirectModified('documents') // false
```

Document.prototype.isDirectSelected()

Parameters

- path «String»

Returns:

- «Boolean»

Checks if `path` was explicitly selected. If no projection, always returns true.

Example

```
Thing.findOne().select('nested.name').exec(function (err, doc) {
   doc.isDirectSelected('nested.name') // true
   doc.isDirectSelected('nested.otherName') // false
   doc.isDirectSelected('nested')  // false
})
```

Document.prototype.isInit()

Parameters

- path «String»

Returns:

- «Boolean»

Checks if `path` is in the `init` state, that is, it was set by `Document#init()` and not modified since.

Document.prototype.isModified()

Parameters

- [path] «String» optional

Returns:

- «Boolean»

Returns true if any of the given paths is modified, else false. If no arguments, returns `true` if any path in this document is modified.

If `path` is given, checks if a path or any full path containing `path` as part of its path chain has been modified.

Example

```
doc.set('documents.0.title', 'changed');
doc.isModified()                      // true
doc.isModified('documents')           // true
doc.isModified('documents.0.title')   // true
doc.isModified('documents otherProp') // true
doc.isDirectModified('documents')     // false
```

Document.prototype.isNew

Type:

- «property»

Boolean flag specifying if the document is new.

Document.prototype.isSelected()

Parameters

- path «String|Array<String>»

Returns:

- «Boolean»

Checks if `path` was selected in the source query which initialized this document.

Example

```
const doc = await Thing.findOne().select('name');
doc.isSelected('name') // true
doc.isSelected('age')  // false
```

Document.prototype.markModified()

Parameters

- path «String» the path to mark modified
- [scope] «Document» the scope to run validators with

Marks the path as having pending changes to write to the db.

Very helpful when using Mixed types.

Example:

```
doc.mixed.type = 'changed';
doc.markModified('mixed.type');
doc.save() // changes to mixed.type are now persisted
```

Document.prototype.modifiedPaths()

Parameters

- [options] «Object»
  - [options.includeChildren=false] «Boolean» if true, returns children of modified paths as well. For example, if false, the list of modified paths for `doc.colors = { primary: 'blue' };` will **not** contain `colors.primary`. If true, `modifiedPaths()` will return an array that contains `colors.primary`.

Returns:

- «Array»

Returns the list of paths that have been modified.

Document.prototype.overwrite()

Parameters

- obj «Object» the object to overwrite this document with

Overwrite all values in this document with the values of `obj`, except for immutable properties. Behaves similarly to `set()`, except for it unsets all properties that aren't in `obj`.

Document.prototype.parent()

If this document is a subdocument or populated document, returns the document's parent. Returns `undefined` otherwise.

Document.prototype.populate()

Parameters

- path «String|Object|Array» either the path to populate or an object specifying all parameters, or either an array of those
- [select] «Object|String» Field selection for the population query
- [model] «Model» The model you wish to use for population. If not specified, populate will look up the model by the name in the Schema's `ref` field.
- [match] «Object» Conditions for the population query
- [options] «Object» Options for the population query (sort, etc)
  - [options.path=null] «String» The path to populate.
  - [options.retainNullValues=false] «boolean» by default, Mongoose removes null and undefined values from populated arrays. Use this option to make `populate()` retain `null` and `undefined` array entries.
  - [options.getters=false] «boolean» if true, Mongoose will call any getters defined on the `localField`. By default, Mongoose gets the raw value of `localField`. For example, you would need to set this option to `true` if you wanted to add a `lowercase` getter to your `localField`.
  - [options.clone=false] «boolean» When you do `BlogPost.find().populate('author')`, blog posts with the same author will share 1 copy of an `author` doc. Enable this option to make Mongoose clone populated docs before assigning them.

    - [options.match=null] «Object|Function» Add an additional filter to the populate query. Can be a filter object containing MongoDB query syntax, or a function that returns a filter object.

    - [options.transform=null] «Function» Function that Mongoose will call on every populated document that allows you to transform the populated document.

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

- [callback] «Function» Callback

Returns:

- «Promise,null»

Populates paths on an existing document.

Example:

```
await doc.populate([
  'stories',
  { path: 'fans', sort: { name: -1 } }
]);
doc.populated('stories'); // Array of ObjectIds
doc.stories[0].title; // 'Casino Royale'
doc.populated('fans'); // Array of ObjectIds

await doc.populate('fans', '-email');
doc.fans[0].email // not populated

await doc.populate('author fans', '-email');
doc.author.email // not populated
doc.fans[0].email // not populated
```

Document.prototype.populated()

Parameters

- path «String»

Returns:

- «Array,ObjectId,Number,Buffer,String,undefined»

Gets _id(s) used during population of the given `path`.

Example:

```
Model.findOne().populate('author').exec(function (err, doc) {
  console.log(doc.author.name)         // Dr.Seuss
  console.log(doc.populated('author')) // '5144cf8050f071d979c118a7'
})
```

If the path was not populated, returns `undefined`.

Document.prototype.replaceOne()

Parameters

- doc «Object»
- options «Object»
- callback «Function»

Returns:

- «Query»

Sends a replaceOne command with this document `_id` as the query selector.

Valid options:

- same as in Model.replaceOne

Document.prototype.save()

Parameters

- [options] «Object» options optional options
  - [options.session=null] «Session» the session associated with this save operation. If not specified, defaults to the document's associated session.
  - [options.safe] «Object» (DEPRECATED) overrides schema's safe option. Use the `w` option instead.
  - [options.validateBeforeSave] «Boolean» set to false to save without validating.
  - [options.validateModifiedOnly=false] «Boolean» If `true`, Mongoose will only validate modified paths, as opposed to modified paths and `required` paths.
  - [options.w] «Number|String» set the write concern. Overrides the schema-level `writeConcern` option
  - [options.j] «Boolean» set to true for MongoDB to wait until this `save()` has been journaled before resolving the returned promise. Overrides the schema-level `writeConcern` option
  - [options.wtimeout] «Number» sets a timeout for the write concern. Overrides the schema-level `writeConcern` option.
  - [options.checkKeys=true] «Boolean» the MongoDB driver prevents you from saving keys that start with '$' or contain '.' by default. Set this option to `false` to skip that check. See restrictions on field names

    - [options.timestamps=true] «Boolean» if `false` and timestamps are enabled, skip timestamps for this `save()`.

- [fn] «Function» optional callback

Returns:

- «Promise,undefined» Returns undefined if used with callback or a Promise otherwise.

Saves this document by inserting a new document into the database if document.isNew is `true`, or sends an updateOne operation **only** with the modifications to the database, it does not replace the whole document in the latter case.

Example:

```
product.sold = Date.now();
product = await product.save();
```

If save is successful, the returned promise will fulfill with the document saved.

Example:

```
const newProduct = await product.save();
newProduct === product; // true
```

Document.prototype.schema

Type:

- «property»

The document's schema.

Document.prototype.set()

Parameters

- path «String|Object» path or object of key/vals to set
- val «Any» the value to set
- [type] «Schema|String|Number|Buffer|*» optionally specify a type for "on-the-fly" attributes
- [options] «Object» optionally specify options that modify the behavior of the set

Sets the value of a path, or many paths.

Example:

```
// path, value
doc.set(path, value)

// object
doc.set({
    path  : value
  , path2 : {
       path  : value
    }
})

// on-the-fly cast to number
doc.set(path, value, Number)

// on-the-fly cast to string
doc.set(path, value, String)

// changing strict mode behavior
doc.set(path, value, { strict: false });
```

Document.prototype.toJSON()

Parameters

- options «Object»

Returns:

- «Object»

The return value of this method is used in calls to JSON.stringify(doc).

This method accepts the same options as Document#toObject. To apply the options to every document of your schema by default, set your schemas `toJSON` option to the same argument.

```
schema.set('toJSON', { virtuals: true })
```

See schema options for details.

Document.prototype.toObject()

Parameters

- [options] «Object»
  - [options.getters=false] «Boolean» if true, apply all getters, including virtuals
  - [options.virtuals=false] «Boolean» if true, apply virtuals, including aliases. Use `{ getters: true, virtuals: false }` to just apply getters, not virtuals

- [options.aliases=true] «Boolean» if `options.virtuals = true`, you can set `options.aliases = false` to skip applying aliases. This option is a no-op if `options.virtuals = false`.
- [options.minimize=true] «Boolean» if true, omit any empty objects from the output
- [options.transform=null] «Function|null» if set, mongoose will call this function to allow you to transform the returned object
- [options.depopulate=false] «Boolean» if true, replace any conventionally populated paths with the original id in the output. Has no affect on virtual populated paths.
- [options.versionKey=true] «Boolean» if false, exclude the version key (`__v` by default) from the output
- [options.flattenMaps=false] «Boolean» if true, convert Maps to POJOs. Useful if you want to `JSON.stringify()` the result of `toObject()`.
- [options.useProjection=false] «Boolean»
  - If true, omits fields that are excluded in this document's projection. Unless you specified a projection, this will omit any field that has `select: false` in the schema.

Returns:

- «Object» js object

Converts this document into a plain-old JavaScript object (POJO).

Buffers are converted to instances of mongodb.Binary for proper storage.

Options:

- `getters` apply all getters (path and virtual getters), defaults to false
- `aliases` apply all aliases if `virtuals=true`, defaults to true
- `virtuals` apply virtual getters (can override `getters` option), defaults to false
- `minimize` remove empty objects, defaults to true
- `transform` a transform function to apply to the resulting document before returning
- `depopulate` depopulate any populated paths, replacing them with their original refs, defaults to false
- `versionKey` whether to include the version key, defaults to true
- `flattenMaps` convert Maps to POJOs. Useful if you want to JSON.stringify() the result of toObject(), defaults to false
- `useProjection` set to `true` to omit fields that are excluded in this document's projection. Unless you specified a projection, this will omit any field that has `select: false` in the schema.

Getters/Virtuals

Example of only applying path getters

```
doc.toObject({ getters: true, virtuals: false })
```

Example of only applying virtual getters

```
doc.toObject({ virtuals: true })
```

Example of applying both path and virtual getters

```
doc.toObject({ getters: true })
```

To apply these options to every document of your schema by default, set your schemas `toObject` option to the same argument.

```
schema.set('toObject', { virtuals: true })
```

Transform

We may need to perform a transformation of the resulting object based on some criteria, say to remove some sensitive information or return a custom object. In this case we set the optional `transform` function.

Transform functions receive three arguments

```
function (doc, ret, options) {}
```

- `doc` The mongoose document which is being converted
- `ret` The plain object representation which has been converted
- `options` The options in use (either schema options or the options passed inline)

Example

```
// specify the transform schema option
if (!schema.options.toObject) schema.options.toObject = {};
schema.options.toObject.transform = function (doc, ret, options) {
  // remove the _id of every document before returning the result
  delete ret._id;
  return ret;
}

// without the transformation in the schema
doc.toObject(); // { _id: 'anId', name: 'Wreck-it Ralph' }

// with the transformation
doc.toObject(); // { name: 'Wreck-it Ralph' }
```

With transformations we can do a lot more than remove properties. We can even return completely new customized objects:

```
if (!schema.options.toObject) schema.options.toObject = {};
schema.options.toObject.transform = function (doc, ret, options) {
  return { movie: ret.name }
}
```

```
// without the transformation in the schema
doc.toObject(); // { _id: 'anId', name: 'Wreck-it Ralph' }

// with the transformation
doc.toObject(); // { movie: 'Wreck-it Ralph' }
```

Note: if a transform function returns `undefined`, the return value will be ignored.

Transformations may also be applied inline, overridding any transform set in the options:

```
function xform (doc, ret, options) {
  return { inline: ret.name, custom: true }
}

// pass the transform as an inline option
doc.toObject({ transform: xform }); // { inline: 'Wreck-it Ralph', custom: true }
```

If you want to skip transformations, use `transform: false`:

```
schema.options.toObject.hide = '_id';
schema.options.toObject.transform = function (doc, ret, options) {
  if (options.hide) {
    options.hide.split(' ').forEach(function (prop) {
      delete ret[prop];
    });
  }
  return ret;
}

const doc = new Doc({ _id: 'anId', secret: 47, name: 'Wreck-it Ralph' });
doc.toObject();                                        // { secret: 47, name: 'Wreck-it Ralph' }
doc.toObject({ hide: 'secret _id', transform: false });// { _id: 'anId', secret: 47, name: 'Wreck-
doc.toObject({ hide: 'secret _id', transform: true }); // { name: 'Wreck-it Ralph' }
```

If you pass a transform in `toObject()` options, Mongoose will apply the transform to subdocuments in addition to the top-level document. Similarly, `transform: false` skips transforms for all subdocuments.

Note that this behavior is different for transforms defined in the schema

if you define a transform in `schema.options.toObject.transform`, that transform will **not** apply to subdocuments.

```
const memberSchema = new Schema({ name: String, email: String });
const groupSchema = new Schema({ members: [memberSchema], name: String, email });
const Group = mongoose.model('Group', groupSchema);

const doc = new Group({
  name: 'Engineering',
  email: 'dev@mongoosejs.io',
  members: [{ name: 'Val', email: 'val@mongoosejs.io' }]
});
```

```
// Removes `email` from both top-level document **and** array elements
// { name: 'Engineering', members: [{ name: 'Val' }] }
doc.toObject({ transform: (doc, ret) => { delete ret.email; return ret; } });
```

Transforms, like all of these options, are also available for `toJSON`. See this guide to `JSON.stringify()` to learn why `toJSON()` and `toObject()` are separate functions.

See schema options for some more details.

During save, no custom options are applied to the document before being sent to the database.

Document.prototype.toString()

Helper for console.log

Document.prototype.undefined

Returns:

- «Array<Document>» array of populated documents. Empty array if there are no populated documents associated with this document.

Gets all populated documents associated with this document.

Document.prototype.unmarkModified()

Parameters

- path «String» the path to unmark modified

Clears the modified state on the specified path.

Example:

```
doc.foo = 'bar';
doc.unmarkModified('foo');
doc.save(); // changes to foo will not be persisted
```

Document.prototype.update()

Parameters

- doc «Object»
- options «Object»
- callback «Function»

Returns:

- «Query»

Sends an update command with this document `_id` as the query selector.

Example:

```
weirdCar.update({$inc: {wheels:1}}, { w: 1 }, callback);
```

Valid options:

- same as in Model.update

Document.prototype.updateOne()

Parameters

- doc «Object»
- [options] «Object» optional see `Query.prototype.setOptions()`
  - [options.lean] «Object» if truthy, mongoose will return the document as a plain JavaScript object rather than a mongoose document. See `Query.lean()` and the Mongoose lean tutorial.
  - [options.strict] «Boolean|String» overwrites the schema's strict mode option
  - [options.timestamps=null] «Boolean» If set to `false` and schema-level timestamps are enabled, skip timestamps for this update. Note that this allows you to overwrite timestamps. Does nothing if schema-level timestamps are not set.
- callback «Function»

Returns:

- «Query»

Sends an updateOne command with this document `_id` as the query selector.

Example:

```
weirdCar.updateOne({$inc: {wheels:1}}, { w: 1 }, callback);
```

Valid options:

- same as in Model.updateOne

Document.prototype.validate()

Parameters

- [pathsToValidate] «Array|String» list of paths to validate. If set, Mongoose will validate only the modified paths that are in the given list.
- [options] «Object» internal options
  - [options.validateModifiedOnly=false] «Boolean» if `true` mongoose validates only modified paths.
  - [options.pathsToSkip] «Array|string» list of paths to skip. If set, Mongoose will validate every modified path that is not in this list.
- [callback] «Function» optional callback called after validation completes, passing an error if one occurred

Returns:

- «Promise» Promise

Executes registered validation rules for this document.

Note:

This method is called `pre` save and if a validation rule is violated, save is aborted and the error is returned to your `callback`.

Example:

```
doc.validate(function (err) {
  if (err) handleError(err);
  else // validation passed
});
```

Document.prototype.validateSync()

Parameters

- pathsToValidate «Array|string» only validate the given paths
- [options] «Object» options for validation
  - [options.validateModifiedOnly=false] «Boolean» If `true`, Mongoose will only validate modified paths, as opposed to modified paths and `required` paths.
  - [options.pathsToSkip] «Array|string» list of paths to skip. If set, Mongoose will validate every modified path that is not in this list.

Returns:

- «ValidationError,undefined» ValidationError if there are errors during validation, or undefined if there is no error.

Executes registered validation rules (skipping asynchronous validators) for this document.

Note:

This method is useful if you need synchronous validation.

Example:

```
const err = doc.validateSync();
if (err) {
  handleError(err);
} else {
  // validation passed
}
```