# Backbone Collections

Backbone Collections group related models together, add a convenient way to persist them, and provide a set of helpful functions for dealing with models. Like a model, a collection can be the basis of a view, and binding a collection's change event to a view's render function will automatically keep them in sync.

## Features of Backbone Collections:

- Collections have sorting capability that not only allows you to sort the collection, but also to keep it sorted as models are added and removed.
- Collections support convenient ways of adding and removing models from the collection. There are ~ 46 iterator methods for doing things like finding a model, reducing a list, mapping a list, sorting, grouping, and shuffling.
- Collections publish a number of events: model added, model removed, model changed, model destroyed, collection synchronized, collection reset, and validation error.

Collections, like models, have a url property, and can be used to retrieve a set of models from the server, or to create a new model on the server.

A collection is an array-like object. Like an array, it has a length property indicating how many elements it contains. Like an array, it can be indexed into, although the syntax is different from an array. Instead of using [] notation, you use the at() method.

## Indexing into a collection:

```
// you can populate a collection by passing an array into the constructor
var collection = new Backbone.Collection([
    { name: 'thing' },
    { name: 'other' }
]);

// checking the length of the collection
console.log(collection.length);

// indexing into the collection (starts at index 0)
console.log(collection.at(0));
```

## Defining new collection types:

```
// define a custom model using extend
var Book = Backbone.Model({});

// define a new type of collection by extending Backbone.Collection
var Books = Backbone.Collection.extend({
    model: Book
});

var books = new Books([
    { title: 'Where the Red Fern Grows', author: 'Wilson Rawls' },
    { title: 'Gravity\'s Rainbow', author: 'Thomas Pynchon' }
]);

console.log(books.length);
console.log(books.at(1));
console.log(JSON.stringify(books.at(1)));
console.log(JSON.stringify(books));
```

## Defining new collection types:

```
// collections can have 'class' properties like models do
var Books = Backbone.Collection.extend({
    model: Book
}, {
    oneBook: function () {
        return new Book({ title: 'unknown', author: 'unknown' });
    }
});
var book = Books.oneBook();
console.log(JSON.stringify(book));
```

Collections are sorted by insertion order or by comparator

```
var Books = Backbone.Collection.extend({
    model: Book,
    comparator: function (book) {
        return book.get('sequence');
    }
});

var books = new Books([
    {
        title: 'Where the Red Fern Grows',
        author: 'Wilson Rawls',
        sequence: 2
    },
    {
        title: 'Gravity\'s Rainbow',
        author: 'Thomas Pynchon',
        sequence: 1
    },
    {
        title: 'Wool',
        author: 'Hugh Howey',
        sequence: 3
    }
]);

console.log(JSON.stringify(books));
```

## Second type of comparator function takes two arguments: should return -1 if args are in correct order, 0 if they compare the same, and 1 if their order should be reversed

```javascript
// comparator definition for a collection
comparator: function (book1, book2) {
    return book1.get('sequence') < book2.get('sequence') ? -1 : 1;
});
```

## Instantiating a Collection - call its constructor function with the 'new' operator

```
var collection = new Backbone.Collection();

// usually you'll be using custom types
var Books = Backbone.Collection.extend({});
var books = new Books();
```

## You can pass the collection's data to the constructor

```
// passing data to collection at instantiation
var collection = new Backbone.Collection([
    model1,
    model2,
    model3
]);
```

## If your collection has an initialize method, it will be invoked after the constructor is called

```
// initialize is no-op by default
var Books = Backbone.Collection.extend({
    initialize: function () {
        console.log('collection created');
    }
});
```

## add() & remove() - work exactly as you'd expect

```
var model = new Backbone.Model();
var collection = new Backbone.Collection();
collection.add(model);
collection.remove(model);
```

## options when adding - use { at: index } to insert at a specific index, and 'silent' to suppress the add event

```
var model = new Backbone.Model();
var collection = new Backbone.Collection();
collection.add(model, { at: 2 });
collection.at(2);

// suppress the 'add' event
var anotherModel = new Backbone.Model();
collection.add(anotherModel, { silent: true });
```

## add() and remove() both work on a single model, or an array of models

```
// remove a single model
collection.remove(model);

// remove a list of models
collection.remove([model1, model2]);

// add a list of models
collection.add([
    new Backbone.Model({ name: 'James', age: 45 }),
    new Backbone.Model({ name: 'Susanne', age: 46 })
]);
```

## You don't have to add Backbone models. Add plain objects and Backbone will transform them into Backbone models for you.

```
collection.add([
    { name: 'James', age: 45 },
    { name: 'Susanne', age: 46 }
]);

console.log(collection);
```

## When a model is added to a collection, it triggers the 'add' event

```
// setting an event handler for the 'add' event
collection.on('add', function (model, collection) {
    console.log('added ' + model.get('name') + ' at index ' +
        this.indexOf(model));
});

collection.add({ name: 'Leo', age: 10 });
collection.add({ name: 'Sadie', age: 11 }, { silent: true });
console.log(JSON.stringify(collection));
```

at() - retrieves a model from a collection by the index of the model in the collection

```
collection.at(0); // first model
collection.at(collection.length - 1); // last model

var collection = new Backbone.Collection([
    { name: 'James', age: 45 },
    { name: 'Susanne', age: 46 },
    { name: 'Leo', age: 9 }
]);
console.log(JSON.stringify(collection.at(0)));
console.log(JSON.stringify(collection.at(collection.length - 1)));
```

## get()  - retrieve a model from a collection by its id, or by it's client id

```
collection.get(1);
collection.get('c0');
```

## if the model has not been saved it will not have an id, so use get(clientId) - the client identifier

```
// adding a model that won't have been saved to the server yet
var collection = new Backbone.Collection([
    { name: 'James', age: 46 }
]);
console.log(collection.at(0).cid);
```

## Working with collections - Backbone proxies a set of underscore.js collection functions, e.g. forEach

```
function safeLog(content) {
    if (console && console.log) {
        console.log(content);
    }
}

collection.forEach(function (model) {
    safeLog(model);
});

// alternatively
collection.forEach(safeLog);
```

map() - produces a new array of values by passing each item in the collection through a transformation function

```
var collection = new Backbone.Collection([
    { name: 'James', age: 46 },
    { name: 'Leo', age: 10 },
    { name: 'Sadie', age: 11 }
]);

// create a new array of models with names capitalized
var mapped = collection.map(function (model) {
    return model.get('name').toUpperCase();
});
console.log(JSON.stringify(mapped));
```

## reduce() - iterates over the collection, producing an aggregate result

```
var start = 0;
var collectiveAge = collection.reduce(function (memo, model) {
    return memo + model.get('age');
}, start);
console.log(collectiveAge);
```

## Collection events - we can detect when a model has been added or removed by listening for add or remove events on the collection

```
var collection = new Backbone.Collection();
collection.on('add', function (model, collection) {
    console.log(JSON.stringify(model) + ' added');
});
collection.on('remove', function (model, collection) {
    console.log(JSON.stringify(model) + ' removed');
});
var model = new Backbone.Model({ name: 'Sadie', age: 11 });
collection.add(model);
collection.remove(model);
```

Collections forward model change events - bind to 'change' or 'change:attribute' events

```
collection.on('change', function (model, options) {
    console.log(JSON.stringify(model) + ' changed');
});
collection.on('change:name', function (model, options) {
    console.log('name property changed');
});
var model = new Backbone.Model();
collection.add(model);
model.set('age', 57);
model.set('name', 'Jimmy');
```