

# Collections

Liam McLennan  
@liammclennan



# Outline

- Collections group related models
- Sorting
- Adding and removing elements
- Getting elements
- Collection iterators
- Events

# Collections

- Container for multiple models of the same type
- Retrieve models from the server
- Create models and save them to the server
- Group models by some attribute
- Collection is an array-like object

```
// length property  
collection.length;
```

```
// indexing  
collection.at(0);
```



# Defining New Collection Types

- Define a new type of collection by extending `Backbone.Collection`
- Specify the type of model that the collection holds

```
var Vehicles = Backbone.Collection.extend({  
  model: Vehicle  
});
```

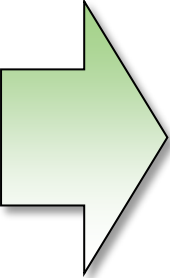
# Defining New Collection Types (cont.)

- Collections can have 'class properties' too

```
var Vehicles = Backbone.Collection.extend({  
  model: Vehicle  
});
```

# Defining New Collection Types (cont.)

- Collections can have 'class properties' too



```
var Vehicles = Backbone.Collection.extend({  
  model: Vehicle  
},  
{  
  myClassProperty: function () {}  
});
```

# Sorted Collections

- Collections are sorted – either by insertion order or by a comparator

```
var Vehicles = Backbone.Collection.extend({  
  model: Vehicle,  
  comparator: function (vehicle) {  
    return vehicle.get('sequence');  
  }  
});
```

# Instantiating a Collection

- To create a new collection object call its constructor function with the 'new' operator
- The simplest case is to create an instance of `Backbone.Collection`

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

- Or use custom types

```
var Vehicles = Backbone.Collection.extend({});  
var fords = new Vehicles();
```



# Instantiating a Collection (cont.)

- You can pass the collections data to the constructor

```
var collection = new Backbone.Collection([  
  model1,  
  model2,  
  model3  
]);
```

- If your collection has an 'initialize' function it will be invoked after the constructor is called.

# add() & remove()

- **add() and remove() work exactly as you would expect**

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

- **Use the 'at' option to insert a model at a specific index and the 'silent' option to suppress the 'add' event.**

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

# Add() & Remove() (cont.)

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

```
collection.remove(model);
```

```
collection.remove([model2, model3]);
```

# at()

- **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
```

# **get() & getByCid()**

- **get()** retrieves a model from a collection by its id

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

- **If your model has not been saved it will not have an id, so use getByCid()**

```
collection.getByCid('c0');
```

# Working with Collections

- Backbone proxies a set of underscore.js collection functions
- **forEach**

```
collection.forEach(function (item) {  
    print(item);  
});
```

```
collection.forEach(print);
```

# Working with Collections

- Backbone proxies a set of underscore.js collection functions
- **forEach**

```
collection.forEach(print);
```

- **map**

```
collection.map(function (item) {  
    return transform(item);  
});
```

# Collection Events

- **Collections raise events when models are added or removed**
  - 'add' event when a model is added
  - 'remove' event when a model is removed

```
collection.on('add', function(model, collection) {  
    console.log(JSON.stringify(model) + ' added');  
});
```

```
collection.on('remove', function(model, collection) {  
    console.log(JSON.stringify(model) + ' removed');  
});
```



# Collection Events (cont.)

- **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) {  
    console.log('name property changed');  
});
```

# Summary

- Use a collection to group multiple model objects of the same type
- Extend Backbone.Collection to define a new collection type
- Provide a comparator function to keep a collection sorted
- Use add, remove, at, get, getbycid, the iterator functions and more to work with collections
- Collections publish events