How to bind events

Views respond to user's actions. How could we know if a user did something on the page? Using **events**! Events are the very important part of Backbone and in the next lessons we will see that not only views may have them.

Let's add a Remove button for each item in the unordered list and bind a simple `click` event handler to it. Event's handler is not going to actually remove anything for now but we will have some foundation for upcoming lessons. So first of all modify the `render` function to include Remove button:

```
render: function() {
  this.$el.html(this.model.title + "<a href='#' class='btn btn-danger'>remove</a>");
  return this;
}
```

Now bind an event. Each view may have an `events` property that contains an object with a list of events. This answers the question why we had to create a separate view for a list and for an item – this way we can bind different events to them:

```
events: {
  'click a': function() {console.log('removed');}
},
```

`click` is the event's name; then after a whitespace you can provide tag name, class name or an id of an element to bind event to. If you don't provide anything, this event will be bound to the top-level element which is `li` in our case.

Next you provide event's handler. By the way we can just specify handler's here and declare the actual function later like this:

```
events: {
  'click a': 'removeEvent'
},
removeEvent: function() {console.log('removed');}
```

Inside the handler we can also reference the actual view by employing `this`:

```
removeEvent: function() {console.log(this);}
```

Try that out!

You can find the full list of built-in BackboneJS events at http://backbonejs.org/#Events-catalog. jQuery also provides us with a handful of other events like `click`, so browse its documentation to learn more.

Backbone's events mixed into any object

Backbone's events may be mixed in into any object. Backbone's views (as well as collections and models) already have events mixed in.

Let's see a really quick example:

```
var object = {};

_.extend(object, Backbone.Events);

object.on("alert", function(msg) {
  alert("Triggered " + msg);
});

object.trigger("alert", "an event");
```

We have a basic JS object and then use `extend` to equip it with events. Next we can call the on method to bind an event. This method is similar to jQuery's on: it binds an event with a specific name and the specified a callback that should be fired when this event happens. The `trigger` method fires that specific event. Once again, this method is similar to the one provided by jQuery.

Let me pinpoint one more thing. Inside the `EventsListView` we have the following line:

```
$('#app').html(this.el);
```

Can't we use

```
$('#app').html(this.$el.html());
```

instead? If you paste this code and reload the page, everything will seem to be the same, but clicking on Remove buttons will make no effect. This is because events are contained inside the element and we just take its HTML and basically lose all the events. Moreover, if you inspect the list with Firebug, the top-level `ul` tag is lost. So please keep this in mind and never use jQuery's `html` method in such cases.