Handling Events

Instead of having to register event listeners on elements you'd like to respond to, simply implement the name of the event you want to respond to as a method on your view.

For example, imagine we have a template like this:

```
{{#view "clickable"}}
This is a clickable area!
{{/view}}
```

Let's implement `App.ClickableView` such that when it is clicked, an alert is displayed:

```
App.ClickableView = Ember.View.extend({
  click: function(evt) {
    alert("ClickableView was clicked!");
  }
});
```

Events bubble up from the target view to each parent view in succession, until the root view. These values are read-only. If you want to manually manage views in JavaScript (instead of creating them using the `{{view}}` helper in Handlebars), see the `Ember.ContainerView` documentation below.

Sending Events

To have the click event from `App.ClickableView` affect the state of your application, simply send an event to the view's controller:

```
App.ClickableView = Ember.View.extend({
  click: function(evt) {
    this.get('controller').send('turnItUp', 11);
  }
});
```

Handling Events

Instead of having to register event listeners on elements you'd like to respond to, simply implement the name of the event you want to respond to as a method on your view.

For example, imagine we have a template like this:

```
{{#view "clickable"}}
This is a clickable area!
{{/view}}
```

Let's implement `App.ClickableView` such that when it is clicked, an alert is displayed:

```
App.ClickableView = Ember.View.extend({
  click: function(evt) {
    alert("ClickableView was clicked!");
  }
});
```

Events bubble up from the target view to each parent view in succession, until the root view. These values are read-only. If you want to manually manage views in JavaScript (instead of creating them using the `{{view}}` helper in Handlebars), see the `Ember.ContainerView` documentation below.