Backbone Router

What sets modern web apps apart from the traditional single-page application is the provision of linkable, book-markable URL's. Previously, hash fragments (#page) were used to provide these perma-links, but now the History API allows the use of standard URL's (/page).

Backbone.Router provides methods for routing client-side pages, and connecting them to actions and events. For browsers which don't support the History API (basically just pre IE10 now), the Router handles graceful fallback by translating to the fragment version of the URL.

Customizing **Backbone.Router** works like every other Backbone component - using **.extend** to create a new constructor you'll use to instantiate instances.

```
var Workspace = Backbone.Router.extend({
  routes: {
    'about': 'about',                    // #about
    'books/:query': 'search',          // #books/fiction
    'books/:query/p:page': 'search'    // #search/fiction/p7
  },
  about: function () {
    // do stuff
  },
  search: function (query, page) {
    // do stuff
  }
});
```

The **routes** hash maps urls to functions on your router, or just function definitions if you prefer. The structure is similar to the **events** hash in Backbone views.

```
routes: {
  'help': 'help',              // show the help "page"
  'search': 'search',          // show the search "page"
  'about': function () {
    // show the about "page"
  }
}
```

Routes can contain parameter parts - **:param**, which match a single URL component between slashes; and splat parts - ***splat**, which can match any number of URL components.

```
routes: {
  "search/:query/p:page": 'search',  // matches a fragment of #search/obama/p2
  "file/*path": 'file'               // matches file/folder/file.txt
}
```

Part of a route can be made optional by surrounding it in parentheses **(/:optional)**.

```
routes: {
  "docs/:section(/:subsection)": "docs"  // matches docs/faq and docs/faq/installing
}
```

Trailing slashes should be treated as unique routes, and should be specified as options if you want to handle a trailing slash as the same route

```
routes: {
  "help(/)": "help"  // matches help and help/
}
```

When a route is matched through either the user typing the route into the url or hitting the back button, an event is emitted: "**route:action**" where "action" is the action defined for the route. This gives the ability to attach an additional handler.

```
routes: {
  "search/:term": "search"  // route:search event will be emitted
}

router.on("route:search", function (term) {
  // ...
});
```

When creating a new router, you can pass a **routes** hash directly as an option

```
var AppRouter = Backbone.Router.extend({
    help: function () {
        ...
    },
    search: function (term) {
        ...
    }
});

var appRouter = new AppRouter({
    routes: {
        'help': 'help',
        'search/:term': 'search'
    }
});
```

The Router object includes two API functions: **route** and **navigate**. The **route** API creates a custom route that can take up to 3 parameters: the route string, the name of the action to trigger, and an optional callback.

```
var AppRouter = Backbone.Router.extend({
  initialize: function () {
    this.route('truck/:id', 'truck');
  },
  showTruck: function (id) {
    console.log('goto truck ', id);
  }
});
```

Whenever you reach a point in your application that you'd like to save as a URL, call **navigate** in order to update the URL.

```
navigate('about');
```

If you also want to call the route function, set the **trigger** option to true. To update the URL without creating an entry in the browser's history, set the **replace** option to true.

```
app.navigate("help/troubleshooting", { trigger: true });
app.navigate("help/troubleshooting", { trigger: true, replace: true });
```

Backbone.history

History is a global router to handle **hashchange** events or **pushState** calls, match appropriate routes, and trigger callbacks. You don't need to create these yourself since Backbone.history already contains one.

start

When all of your Routers have been instantiated, call **Backbone.history.start()** to begin monitoring **hashchange** and **pushState** events, and dispatching routes.

To indicate that you'd like to use HTML5 **pushState** support in your application, use **Backbone.history.start({ pushState: true })**. If you're supporting browsers that don't support it natively and you want to use full page refreshes instead, you can add **{ hashChange: false }** to the options you pass to **start**.

If your application is not being served from the root url, i.e. "/", of your domain, you can specify where the root really is as an option:

Backbone.history.start({ pushState: true, root: "/public/search/" })

If the server has already rendered the entire page, and you don't want the initial route to trigger when starting History, pass **silent: true** as an option.

If you're using **pushState,** the use of real URLs requires your web server to be able to correctly render those pages, so back-end changes are required as well.