AngularUI Router and it's trasitionTo() Method

Posted At : November 17, 2015 9:41 AM | Posted By : Cutter
Related Categories: Development, AngularJS, JavaScript

Angular applications are, typically, single page applications. So when you move around in an Angular application and an area (or areas) of the screen changes, you're said to be moving from one "state" to another. To simplify handling these states you can utilize the very powerful **Angular-UI Router**. From the GitHub site:

> AngularUI Router is a routing framework for AngularJS, which allows you to organize the parts of your interface into a state machine. Unlike the $route service in the Angular ngRoute module, which is organized around URL routes, UI-Router is organized around states, which may optionally have routes, as well as other behavior, attached.

Yes, Angular's built in ngRoute module can give you the basics. AngularUI Router, on the other hand, provides the developer with a simple way of defining very complex state management. Or it seems simple, until...[WHAM!] you hit the brick wall.

The AngularUI Router is so easy, the documentation is... well, it's "OK". Sometimes you just run into a situation where the documentation is not clear enough. For instance, what if you're trying to call the same state, but with new parameters?

"What do you mean, Cutter?" Take the following example. A user is browsing a category in an online store. On the screen is a layout loaded with inventory of that category. Clicking on an item would bring up the detail of that item. After the detail of that item there are links to additional items, either related to the original, or of a similar nature. Seems pretty straightforward, right?

Wrong! (Well, not wrong, but....) It's not always so straightforward.Categories fall under the app, and inventory falls under a category. This says that we need a nested state approach. To illustrate, let me give you some basic routing here to kick it off.

```
$stateProvider
  .state('store', {
   url: '/',
   templateUrl: '/partials/store/index.html'
  })
  .state('store.category', {
   url: '/{cat:\\d+}',
   templateUrl: '/partials/categories/index.html'
  })
  .state('store.category.item', {
   params: {itemid: null},
   onEnter: function () {
    // Do something here
   }
  });
```

The initial state ("store") is the site wrapper (navigation and footer and stuff). The category state would likely include some lookup of inventory for the category, and lay out that inventory on the page. The item state might open a modal for that item, with a title and image. It gets an itemid as a param, but for security reasons it is not in the url. There's also no 'view' associated with the item state, with the onEnter handling any UI changes.

So, what's the problem? Well, the issue is the additional inventory items that display below the item details. If a user is looking at an item, then chances are they're in the "store.category.item" state. The additional inventory links on this view would also link to the "store.category.item" state, with the difference being the itemid being passed. If the param were in the URL it might be different, but since it's intentionally being hidden the URL does not change. So a user clicking on a link with a *ui-sref* (for the same state they are currently in) just chokes. The same happens if your controller or service tries to call $state.go('same.state', params). You can add a reload *option* to your $state.go() call, but it will reload all the way up the parent state as well.

The requirement ran as followed:

- Want to reload a child state, without reloading the parent
- Want to change the parameters of that state
- The (to be loaded) child state has no UI of it's own, no template, and no URL params

Initially I just tried something like this:

```
<a ui-sref="store.category.item({itemid: item.id})">{{item.name}}</a>
```

It was really frustrating to find my click wasn't working, and took even longer to figure out it was because I was already in the "store.category.item" state. So then I tried something like this:

```
<a ui-sref="store.category.item({itemid: item.id})" ui-sref-opts="{reload: true}">{{item.name}}</a>
```

But, of course, that would reload the entire state tree, parents on down. This was unacceptable. Finally, I found that I couldn't use the standard *ui-sref* bits to do what I needed to do. So, I ripped that out, and replaced it with an ngClick method in my controller:

```
'use strict';
```

```
class MyController {
 // ... constructor and stuff, then...

 goToItem (itemid) {
  this.$state.go('store.category.item', {itemid: itemid}, {inherit: true, notify: true});
 }
}

MyController.$inject = ['$state']'

export {MyController};
```

```
<a ng-click="ctrl.goToItem(item.id)">{{item.name}}</a>
```

Closer. I could set a break point and see that it was trying, at least. But still not right yet. I scoured the **Guide** for information, and then went looking through the **API Reference**, before finally stumbling upon *$state.transitionTo()*.

The *$state.transitionTo()* method is used by *$state.go()*, under the hood. But, if you don't read through the documentation, it is easy to miss a little nugget in the description of the *reload* option:

> **reload** (v0.2.5) - {boolean=false|string=|object=}, If true will force transition even if the state or params have not changed, aka a reload of the same state. It differs from reloadOnSearch because you'd use this when you want to force a reload when everything is the same, including search params. **if String, then will reload the state with the name given in reload, and any children**. if Object, then a stateObj is expected, will reload the state found in stateObj, and any children.

What??? That's not what the *go()* method's reload option does? In the *go()* method, the reload option is strictly a boolean. This says that, if I supply it with a string of the state to reload, that it would reload from that state down through it's children (if it were a parent of the one I was calling). By setting it to the same state I was trying to call, I ensured that it **only** reloads my item state, inheriting data from parent states. So, I tried it out...

```
goToItem (itemid) {
  let statename = 'store.category.item';
  this.$state.transitionTo(statename, {itemid: itemid}, {reload: statename, inherit: true, notify: true});
 }
```

Success! I could merrily bounce around among my inventory, reloading only the

named state as I went. Glorious jubilation all around as the band played on (OK, going a little overboard here). Point is, it worked.

The AngularUI Router is immensely powerful, and can truly provide some complex (yet elegant) state management within your application. I highly recommend reading over the guide to see what might be available to you.