URL Routing

kceiw edited this page 25 days ago · 43 revisions

◀ Back (Multiple Named Views) Next (The Components) ▶

Most states in your application will probably have a url associated with them. URL Routing was not an afterthought to the state mechanics, but was figured into the design from the beginning (all while keeping states separate from url routing)

Here's how you set a basic url.

```
$stateProvider
    .state('contacts', {
        url: "/contacts",
        templateUrl: 'contacts.html'
    })
```

Now when the user accesses `index.html/contacts` then the 'contacts' state would become active and the main `ui-view` will be populated with the 'contacts.html' partial. Alternatively, if the user were to transition to the 'contacts' state via `transitionTo('contacts')` then the url would be updated to `index.html/contacts`

URL Parameters

Basic Parameters

Often, URLs have dynamic parts to them which are called parameters. There are several options for specifying parameters. A basic parameter looks like this:

```
$stateProvider
    .state('contacts.detail', {
        url: "/contacts/:contactId",
        templateUrl: 'contacts.detail.html',
        controller: function ($stateParams) {
            // If we got here from a url of /contacts/42
            expect($stateParams).toBe({contactId: "42"});
        }
    })
```

Alternatively you can also use curly brackets:

```
// identical to previous example
url: "/contacts/{contactId}"
```

Examples:

- `'/hello/'` - Matches only if the path is exactly '/hello/'. There is no special treatment for trailing slashes, and patterns have to match the entire path, not just a prefix.
- `'/user/:id'` - Matches '/user/bob' or '/user/1234!!!' or even '/user/' but not '/user' or '/user/bob/details'. The second path segment will be captured as the parameter 'id'.
- `'/user/{id}'` - Same as the previous example, but using curly brace syntax.
- `'/user/{id:int}'` - The param is interpreted as Integer.

Note:

Clone this wiki locally

- Parameter names may contain only word characters (latin letters, digits, and underscore) and must be unique within the pattern (across both path and search parameters).

Using Parameters in Links

To create a link that passes parameters, use the state name like a function and pass it an object with parameter names as keys. The proper `href` will be generated.

For example, using the above state which specified a `contactId` parameter, create a link like so:

```
<a ui-sref="contacts.detail({contactId: id})">View Contact</a>
```

The value for `id` can be anything in scope.

Regex Parameters

A bonus to using curly brackets is the ability to set a Regular Expression rule for the parameter:

```
// will only match a contactId of one to eight number characters
url: "/contacts/{contactId:[0-9]{1,8}}"
```

Examples:

- `'/user/{id:[^/]*}'` - Same as `'/user/{id}'` from the previous example.
- `'/user/{id:[0-9a-fA-F]{1,8}}'` - Similar to the previous example, but only matches if the id parameter consists of 1 to 8 hex digits.
- `'/files/{path:.*}'` - Matches any URL starting with '/files/' and captures the rest of the path into the parameter 'path'.
- `'/files/*path'` - Ditto. Special syntax for catch all.

Warning:

- Don't put capturing parentheses into your regex patterns, the UrlMatcher in ui-router adds those itself around the entire regex. You're effectively introducing a second capture group for the same parameter, which trips up the numbering in the child URL. You can use non-capturing groups though, i.e. (?:...) is fine.
- Regular expression can't include forward slashes as that's route segment delimiter
- Route parameters with regular expressions can't be optional or greedy

Query Parameters

You can also specify parameters as query parameters, following a '?':

```
url: "/contacts?myParam"
// will match to url of "/contacts?myParam=value"
```

If you need to have more than one, separate them with an '&':

```
url: "/contacts?myParam1&myParam2"
// will match to url of "/contacts?myParam1=value1&myParam2=wowcool"
```

Using Parameters without Specifying Them in State URLs

You still can specify what parameters to receive even though the parameters don't appear in the url. You need to add a new field params in the state and create links as specified in Using Parameters in Links

For example, you have the state.

```
.state('contacts', {
        url: "/contacts",
        params: {
            param1: null
        }
        templateUrl: 'contacts.html'
    })
```

The link you create is

```
<a ui-sref="contacts({param1: value1})">View Contacts</a>
```

Or can you pass them to $state.go() too.

```
$state.go('contacts', {param1: value1})
```

URL Routing for Nested States

Appended Routes (default)

When using url routing together with nested states the default behavior is for child states to append their url to the urls of each of its parent states.

```
$stateProvider
  .state('contacts', {
     url: '/contacts',
     ...
  })
  .state('contacts.list', {
     url: '/list',
     ...
  });
```

So the routes would become:

- **'contacts' state** matches `"/contacts"`
- **'contacts.list' state** matches `"/contacts/list"` . The urls were combined.

Absolute Routes (^)

If you want to have absolute url matching, then you need to prefix your url string with a special symbol '^'.

```
$stateProvider
  .state('contacts', {
     url: '/contacts',
     ...
  })
  .state('contacts.list', {
     url: '^/list',
     ...
  });
```

So the routes would become:

- **'contacts' state** matches `"/contacts"`
- **'contacts.list' state** matches `"/list"` . The urls were **not** combined because `^` was used.

$stateParams Service

As you saw previously the $stateParams service is an object that will have one key per url parameter. The $stateParams is a perfect way to provide your controllers or other services with the individual parts of the navigated url.

Note: $stateParams service must be specified as a state controller, and it will be scoped so only the relevant parameters defined in *that* state are available on the service object.

```
// If you had a url on your state of:
url: '/users/:id/details/{type}/{repeat:[0-9]+}?from&to'

// Then you navigated your browser to:
'/users/123/details//0'

// Your $stateParams object would be
{ id:'123', type:'', repeat:'0' }

// Then you navigated your browser to:
'/users/123/details/default/0?from=there&to=here'

// Your $stateParams object would be
{ id:'123', type:'default', repeat:'0', from:'there', to:'here' }
```

Important `$stateParams` Gotcha

In state controllers, the `$stateParams` object will only contain the params that were registered with that state. So you will not see params registered on other states, including ancestors.

```
$stateProvider.state('contacts.detail', {
   url: '/contacts/:contactId',
   controller: function($stateParams){
      $stateParams.contactId  //*** Exists! ***//
   }
}).state('contacts.detail.subitem', {
   url: '/item/:itemId',
   controller: function($stateParams){
      $stateParams.contactId //*** Watch Out! DOESN'T EXIST!! ***//
      $stateParams.itemId //*** Exists! ***//
   }
})
```

Instead, use a resolve statement in the parent route.

```
$stateProvider.state('contacts.detail', {
   url: '/contacts/:contactId',
   controller: function($stateParams){
      $stateParams.contactId  //*** Exists! ***//
   },
   resolve:{
      contactId: ['$stateParams', function($stateParams){
          return $stateParams.contactId;
      }]
   }
}).state('contacts.detail.subitem', {
   url: '/item/:itemId',
   controller: function($stateParams, contactId){
      contactId //*** Exists! ***//
      $stateParams.itemId //*** Exists! ***//
   }
})
```

$urlRouterProvider

$urlRouterProvider has the responsibility of watching $location. When $location changes it runs through a list of rules one by one until a match is found. $urlRouterProvider is used behind the scenes anytime you specify a `url` in a state configuration. All urls are compiled into a UrlMatcher object (see $urlMatcherFactory below).

There are several methods on $urlRouterProvider that make it useful to use directly in your module config.

`when()` for redirection

Parameters:

- `what` **String** | **RegExp** | **UrlMatcher** The incoming path that you want to redirect.
- `handler` **String** | **Function** The path you want to redirect your user to.

`handler` as String

If handler is a string, it is treated as a redirect, and is interpolated according to the syntax of match (i.e. like String.replace() for RegExp, or like a UrlMatcher pattern otherwise).

```
app.config(function($urlRouterProvider){
    // when there is an empty route, redirect to /index
    $urlRouterProvider.when('', '/index');

    // You can also use regex for the match parameter
    $urlRouterProvider.when(/aspx/i, '/index');
})
```

`handler` as Function

If the handler is a function, it is injectable. It gets invoked if $location matches. You have the option of inject the match object as $match

The handler can return:

- **falsy** to indicate that the rule didn't match after all, then $urlRouter will continue trying to find another one that matches.
- a **String**, which is treated as a redirect and passed to $location.url()
- **nothing** or any **truthy** value tells $urlRouter that the url was handled

Here's the actual code that we use to register state's that have urls behind the scenes.

```
$urlRouterProvider.when(state.url, ['$match', '$stateParams', function ($match, $state
    if ($state.$current.navigable != state || !equalForKeys($match, $stateParams)) {
        $state.transitionTo(state, $match, false);
    }
}]);
```

`otherwise()` for invalid routes

Parameters:

- `path` **String** | **Function** The url path you want to redirect to or a function `rule` that returns the url path. The function version is passed two params: `$injector` and `$location` .

```
app.config(function($urlRouterProvider){
    // if the path doesn't match any of the urls you configured
    // otherwise will take care of routing the user to the specified url
    $urlRouterProvider.otherwise('/index');

    // Example of using function rule as param
    $urlRouterProvider.otherwise(function($injector, $location){
        ... some advanced code...
    });
})
```

`rule()` for custom url handling

Parameters:

- `handler` **Function** A function that takes in the $injector and $location services as arguments. You are responsible for returning a valid path as a string.

```
app.config(function ($urlRouterProvider) {
   // Here's an example of how you might allow case insensitive urls
   // Note that this is an example, and you may also use
   // $urlMatcherFactory.caseInsensitive(true); for a similar result.
   $urlRouterProvider.rule(function ($injector, $location) {
       //what this function returns will be set as the $location.url
        var path = $location.path(), normalized = path.toLowerCase();
        if (path != normalized) {
            //instead of returning a new url string, I'll just change the $location.pa
            $location.replace().path(normalized);
        }
        // because we've returned nothing, no state change occurs
    });
})
```

$urlMatcherFactory and UrlMatchers

Defines the syntax for url patterns and parameter placeholders. This factory service is used behind the scenes by $urlRouterProvider to cache compiled UrlMatcher objects, instead of having to re-parse url patterns on every location change. Most users will not need to use $urlMatcherFactory directly, however it could be useful to craft a UrlMatcher object and pass it as the url to the state config.

Please refer to the comment documentation within the $urlMatcherFactory file to learn more.

```
var urlMatcher = $urlMatcherFactory.compile("/home/:id?param1");
$stateProvider.state('myState', {
    url: urlMatcher
});
```

◄ Back (Multiple Named Views)   Next (The Components) ►

Use the table of contents to the top right to navigate to other wiki sections! ui-router rocks!