Bootstrap

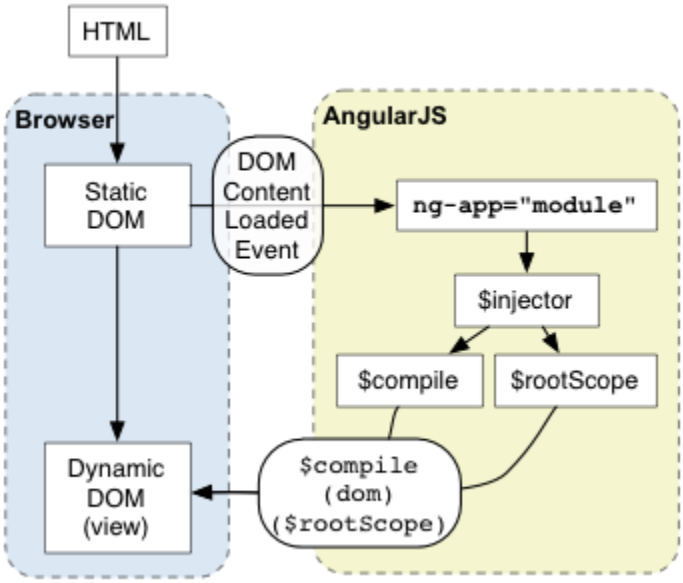
Angular `<script>` Tag

This example shows the recommended path for integrating Angular with what we call automatic initialization.

```
<!doctype html>
<html xmlns:ng="http://angularjs.org" ng-app>
  <body>
    ...
    <script src="angular.js">
  </body>
</html>
```

1. Place the `script` tag at the bottom of the page. Placing script tags at the end of the page improves app load time because the HTML loading is not blocked by loading of the `angular.js` script.
2.
    - Choose: `angular-[version].js` for a human-readable file, suitable for development and debugging.
    - Choose: `angular-[version].min.js` for a compressed and obfuscated file, suitable for use in production.
3. Place `ng-app` to the root of your application, typically on the `<html>` tag if you want angular to auto-bootstrap your application.
4. If you require IE7 support add `id="ng-app"`
5. If you choose to use the old style directive syntax `ng:` then include xml-namespace in `html` to make IE happy. (This is here for historical reasons, and we no longer recommend use of `ng:`.)

Automatic Initialization



Angular initializes automatically upon `DOMContentLoaded` event or when the`angular.js` script is evaluated if at that time `document.readyState` is set to `'complete'`. At this point Angular looks for the `ng-app` directive which designates your application root. If the `ng-app` directive is found then Angular will:

- load the module associated with the directive.
- create the application injector
- compile the DOM treating the `ng-app` directive as the root of the compilation. This allows you to tell it to treat only a portion of the DOM as an Angular application.

```
<!doctype html>
<html ng-app="optionalModuleName">
  <body>
    I can add: {{ 1+2 }}.
    <script src="angular.js"></script>
  </body>
</html>
```

Manual Initialization

If you need to have more control over the initialization process, you can use a manual bootstrapping method instead. Examples of when you'd need to do this include using script loaders or the need to perform an operation before Angular compiles a page.

Here is an example of manually initializing Angular:

```
<!doctype html>
<html>
<body>
  <div ng-controller="MyController">
    Hello {{greetMe}}!
  </div>
  <script
src="http://code.angularjs.org/snapshot/angular.js"></script
>


  <script>
    angular.module('myApp', [])
      .controller('MyController', ['$scope', function
($scope) {
        $scope.greetMe = 'World';
      }]);


    angular.element(document).ready(function() {
      angular.bootstrap(document, ['myApp']);
    });
  </script>
```

```
</body>
</html>
```

Note that we provided the name of our application module to be loaded into the injector as the second parameter of the`angular.bootstrap` function. Notice that `angular.bootstrap` will not create modules on the fly. You must create any custom modulesbefore you pass them as a parameter.

You should call `angular.bootstrap()` *after* you've loaded or defined your modules. You cannot add controllers, services, directives, etc after an application bootstraps.

Note: You should not use the ng-app directive when manually bootstrapping your app.

This is the sequence that your code should follow:

1. After the page and all of the code is loaded, find the root element of your AngularJS application, which is typically the root of the document.
2. Call `angular.bootstrap` to compile the element into an executable, bi-directionally bound application.

Manual Bootstrap Process

You can manually initialized your angular app by using `angular.bootstrap()` function. This function takes the modules as parameters and should be called within `angular.element(document).ready()` function. The angular.element(document).ready() function is fired when the DOM is ready for manipulation.

```
1. <html>
2. <body>
3.  <div ng-controller="Ctrl">
4.  Hello {{msg}}!
5.  </div>
6.  <script src="lib/angular.js"></script>
7.
8.  <script>
9.  var app = angular.module('myApp', []);
10. app.controller('Ctrl', function ($scope) {
11. $scope.msg = 'World';
12. });
13.
14. //manual bootstrap process
15. angular.element(document).ready(function () {
```

```
16.  angular.bootstrap(document, ['myApp']);
17.  });
18.  </script>
19. </body>
20. </html>
```

Note

1. You should not use the ng-app directive when manually bootstrapping your app.
2. You should not mix up the automatic and manual way of bootstrapping your app.
3. Define modules, controller, services etc. before manually bootstrapping your app as defined in above example.

Compiler

Angular's HTML compiler allows the developer to teach the browser new HTML syntax. The compiler allows you to attach behavior to any HTML element or attribute and even create new HTML elements or attributes with custom behavior. Angular calls these behavior extensions directives.

Angular comes pre-bundled with common directives which are useful for building any app. We also expect that you will create directives that are specific to your app. These extensions become a Domain Specific Language for building your application.

All of this compilation takes place in the web browser; no server side or pre-compilation step is involved.

Compiler

Compiler is an Angular service which traverses the DOM looking for attributes. The compilation process happens in two phases.

1. **Compile:** traverse the DOM and collect all of the directives. The result is a linking function.
2. **Link:** combine the directives with a scope and produce a live view. Any changes in the scope model are reflected in the view, and any user interactions with the view are reflected in the scope model. This makes the scope model the single source of truth.

Some directives such as `ng-repeat` clone DOM elements once for each item in a collection. Having a compile and link phase improves performance since the cloned template only needs to be compiled once, and then linked once for each clone instance.