Chapter 10: Modules and the injector

modules: collections of application configuration information，本身没有instantiate

injector: 创建一个injector，给它一些modules，来初始化这些modules

angular Global object：存储所有已经注册的Angular modules，定义window.angular对象

load once：如果window.angular已经存在，就不再次初始化，而是返回这个对象。

```
function setupModuleLoader(window) {
var ensure = function(obj, name, factory) {
return obj[name] || (obj[name] = factory()); };
var angular = ensure(window,  angular , Object); }
```

angular.module方法：参数是module name和一个该module的dependencies的数组，在module方法内将创建module的工作放在createModule函数内，返回一个module object，包括name属性和requires属性（包括需要的modules）。angular.module还需要维护所有已经注册的modules，并且对于不存在的module要抛出error。

injector不是module loader的一部分，而是一个independent service。

```
it( has a constant that has been registered to a module , function() {
var module = window.angular.module( myModule , []); module.constant( aConstant , 42);
var injector = createInjector([ myModule ]); expect(injector.has( aConstant )).toBe(true);
});
```

关于module和injector的一个规则：modules病不包含实际的application components，只包含recipes，需要靠injector进行实例化才能编程concrete application。module包含一个collection of tasks，所谓“invoke queue”，当injector装在一个module时，injector从module的invoke queue中运行这些tasks。

当我们创建injector时，要遍历所有给了的module name，查看对应的module objects，然后消除invoke queue里面的任务。怎么消除？查找每个invocation array的第一个项目对应的$provide方法。

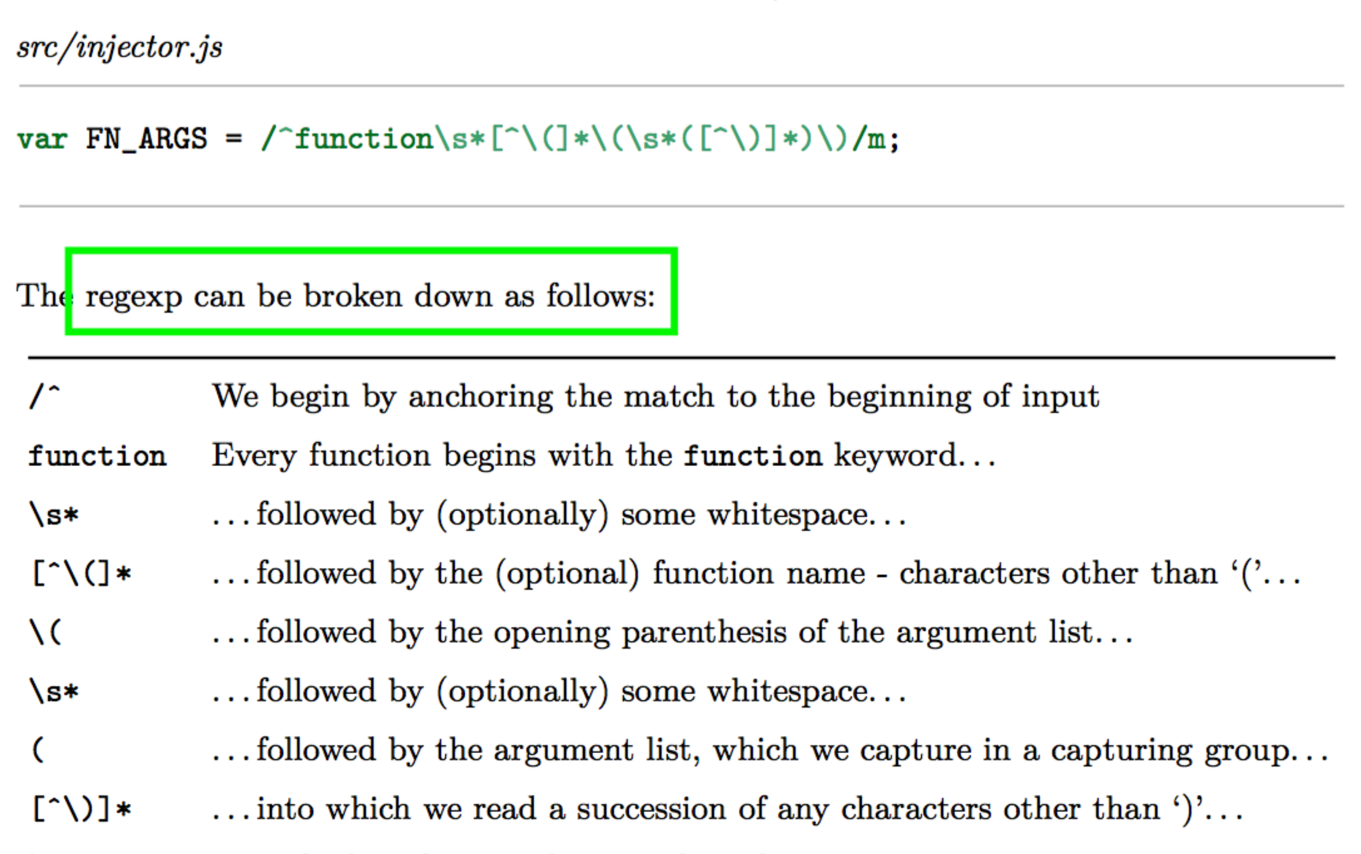
(method).apply:

call 和 apply 都是为了改变某个函数运行时的 context 即上下文而存在的，换句话说，就是为了改变函数体内部 this 的指向。因为 JavaScript 的函数存在「定义时上下文」和「运行时上下文」以及「上下文是可以改变的」这样的概念。二者的作用完全一样，只是接受参数的方式不太一样。例如，有一个函数 func1 定义如下：var func1 = function(arg1, arg2) {};就可以通过 func1.call(this, arg1, arg2); 或者 func1.apply(this, [arg1, arg2]); 来调用。其中 this 是你想指定的上下文，他可以任何一个 JavaScript 对象(JavaScript 中一切皆对象)，call 需要把参数按顺序传递进去，而 apply 则是把参数放在数组里。JavaScript 中，某个函数的参数数量是不固定的，因此要说适用条件的话，当你的参数是明确知道数量时，用 call，而不确定的时候，用 apply，然后把参数 push 进数组传递进去。当参数数量不确定时，函数内部也可以通过 arguments 这个数组来便利所有的参数。

Dependency injection: to invoke functions and construct objects and automatically look up the dependencies they need，给injector一个函数并要求injector来调用这个函数，同时找出这个函数需要哪些参数，并提供这些参数。

怎么找出一个函数需要哪些参数呢？1，显式指定，制定一个$inject属性，2，调用injector.invoke，参数用一个数组包括了函数和所需要的参数，3，不指定，从而让injector试图从函数本身找到所需参数，需要正则表达式来提取出所需要的参数。

http://www.w3school.com.cn/jsref/jsref_obj_regexp.asp



src/injector.js

```
var FN_ARGS = /^function\s*[^\(]*\(\s*([^\)]*)\)/m;
```

The regexp can be broken down as follows:

`/^`	We begin by anchoring the match to the beginning of input
`function`	Every function begins with the **function** keyword. . .
`\s*`	. . . followed by (optionally) some whitespace. . .
`[^\(]*`	. . . followed by the (optional) function name - characters other than ‘(’. . .
`\(`	. . . followed by the opening parenthesis of the argument list. . .
`\s*`	. . . followed by (optionally) some whitespace. . .
`(`	. . . followed by the argument list, which we capture in a capturing group. . .
`[^\)]*`	. . . into which we read a succession of any characters other than ‘)’. . .
`)`	. . . and when done reading we close the capturing group. . .
`\)`	. . . and still match the closing parenthesis of the argument list. . .
`/m`	. . . and define the whole regular expression to match over multiple lines.

Strict Mode: 区别是如果试图inject一个没有显式annotated的function会报错

Instantiating Objects with DI: 不仅inject plain functions，也inject构造函数。难点在于constructor的prototype chain也应该被instantiate，比如如果constructor的prototype包括了其他的方法，在object上也应该体现出来。方法：使用*Object.create*函数

```
function instantiate(Type, locals) {
var UnwrappedType = _.isArray(Type) ? _.last(Type) : Type;
var instance = Object.create(UnwrappedType.prototype);
invoke(Type, instance, locals);
return instance;
}
```

总结：

- How the angular global variable and its module method come to be.
- How modules can be registered.
- That the angular global will only ever be registered once per window, but any given module can be overridden by a later registration with the same name.
- How previously registered modules can be looked up.
- How an injector comes to be.
- How the injector is given names of modules to instantiate, which it will look up from the angular global.
- How application component registrations in modules are queued up and only instan- tiated when the injector loads the module.
- How modules can require other modules and that the required modules are loaded first by the injector.
- That the injector loads each module only once to prevent unnecessary work and problems with circular requires.
- How the injector can be used to invoke a function and how it can look up its arguments from its $inject annotation.
- How the injected function’s this keyword can be bound by supplying it to injector.invoke.
- How a function’s dependencies can be overridden or augmented by supplying a locals object to injector.invoke.
- How array-wrapper style function annotation works.
- How function dependencies can be looked up from the function’s source code.
- How strict DI mode helps make sure you’re not accidentally using non-annotated dependency injection when you don’t mean to.
- injector.annotate如何分析any given function的dependencies
- injector.instantiate如何通过dependency injection进行初始化

Chapter 11: Providers

建立好injector之后，接下来开始构造用来创建进行inject的application components的API。本章讨论providers，Providers是提供如何make dependencies的object，是Angular除了constants之外的所有application components的基础，包括Services, factories, values。

最简单的Provider：一个包括了$get方法的object，injector会调用这个$get方法，并将其返回值作为实际的dependency value。意义：可以对$get方法inject dependencies，也就是说dependencies have dependencies，实现是调用invoke。

lazy instantiations of Dependencies：按需调用，只有在需要返回值的时候才调用这些$get方法，这样就可以不需要按照调用顺序来注册。实现：存下来对应的provider object，注意区分providerCache和instanceCache。这样一个provider的dependency当且仅当在被injected或者被显式地通过injector.get被调用，才会被初始化。

"Everything in Angular is a singleton"

Circular Dependencies chain：循环依赖可能导致circle，无限循环从而内存溢出。解决办法是通过一个变量来标记被lookup的dependency，表示“constructing this dependency”，之后再碰到这个标记就表示有了circle。可以通过一个path数组来记录所有getService，从而可以给出具体报错信息，circle在哪里。

Provider Constructors: 之前的provider是一个包括$get方法的对象，也可以使用构造函数来初始化一个provider。实现时如果发现_.isFunction(provider)，则调用instantiate函数，provider = instantiate(provider)。

两种Injectors：Provider injector和Instance injector。第一个区别：provider constructor可以注入其他的providers，当可以向一个provider注入一个其他的provider的构造函数时，就不应该注入一个实例。第二个区别：injection between provider contractors只关心其他providers，而injection between $get methods and external injector API只关心instances。

Dependency injection的两个phases：1，provider injection，从一个模块的invoke queue注册providers，之后不会再动providerCache。2，instance injection，在runtime发生，当有人调用这个injector的外部API时。

总结：Providers是dependency injection的一个重要环节，它使得dependencies可以有其他的dependencies，也允许dependencies被lazy instantiation。provider injection和instance injection发生在不同阶段，内部实现是使用了两个不同的injector objects。

1. provider object, $get方法,injector.invode
2. dependencies are singletons
3. circular dependencies处理
4. dependency injection的two phaces

Chapter 12: High level Dependency Injection Features

这一章添加了一些高级features，也就是开发者平时使用的功能，实际上这些功能是在之前工作的基础上很简单的工作。

Injecting the $injectors：关于injector最关心的方法可能是get，通过它可以动态地获得dependency。实现方法是在instanceCache上添加$injector属性，获得instanceInjector。同样，provider injector是在providerCache上添加$injector。然而这个API是read-only的，如果想要添加一些dependency（不是添加到module的时候），可以使用$provide对象，通过injecting $provide可以获得我们之前从module invoke queue调用的所有方法。$injector和$provider是用来做一些额外的修改的。

Config Blocks：用来在module loading time执行任意的”configuration functions”，并且向这些函数inject providers。当injector被创建的时候，模块的config函数会被执行。步骤：1，在模块上创建一个用来注册config block的API，扩展invoke queue items到[service, method, arguments]，然后更新已有的queueing methods，将第一个对象替换成$provide object。2，关于invokeLater的顺序，修改queue使得所有的registration invocations在config blocks之前运行。3，另外一种注册config block的方法是在创建module instance的时候设置为第三个参数。

Run blocks：和config blocks很相似，区别是run blocks是从instance cache进行inject的，用处是用来在Angular startup process中运行一段代码。To sum up, config blocks are executed during module loading and run blocks are executed immediately after it.

function module：另外一种定义module的方法，A module can be just a function, which will be injected from the provider injector when loaded.

Hash keys and hash maps：手动实现Javascript的hashmap结构。先实现hashKey，注意haskKey关注的是object的identity，而不是语义值。也就是说，

```
it( does not change when object value changes , function() { var obj = {a: 42};
var hash1 = hashKey(obj);
obj.a = 43;
var hash2 = hashKey(obj);
  expect(hash1).toEqual(hash2);
});
```

然后实现HashMap函数，实现put, get，remove方法。

Factories,values, services：A factory is a function that produces a dependency, can be injected with instance dependencies. 注册一个factory的时候，实现是注册了一个provider object，这个provider的$get方法是注册的工厂函数。

values和常量相像，区别是value只对instance有效，对providers和config blocks无效。

Whereas a factory is a plain old function, a service is a constructor function. When you register a service, the function you give is treated as a constructor and an instance of it will be created.

Decorators：decorator用来修改一些已有的dependency。

```
it( allows changing an instance using a decorator , function() { var module =
window.angular.module( myModule , []); module.factory( aValue , function() {
return {aKey: 42}; });
module.decorator( aValue , function($delegate) { $delegate.decoratedKey = 43;
});
var injector = createInjector([ myModule ]);
  expect(injector.get( aValue ).aKey).toBe(42);
  expect(injector.get( aValue ).decoratedKey).toBe(43);
});
```

实现方法是拿到provider的$get方法，然后调用instanceInjector.invoke(get, provider)。

Revisit：在顶层再家一层，设置module loader，并且注册一些core Angular components。

```
function publishExternalAPI() {  use strict ;
setupModuleLoader(window);
var ngModule = window.angular.module( ng , []);
}
```

ng是Angular自身的所有services, directives, filters, and other components所在的地方。 然后再将 filters, scopes, expression parser和dependency injection features 结合在一起。

In this chapter you have learned:

- How you can inject an `$injector` and how that `$injector` may be either the provider injector or the instance injector depending on where you’re injecting it.
- How you can inject `$provide` and register additional components in config blocks, function modules.
- How config blocks work, and how they are implemented by simply invoking them with the provider injector.
- How a config block can be specified as the third argument to `angular.module`.
- How run blocks work, and how they are implemented by simply invoking them with the instance injector.
- That run blocks are deferred to a moment when all modules are loaded.
- How you can define a function module, which is essentially the same as a config block.
- How Angular’s internal hash key and hash map implementations work, and how they deal with compound data structures by adding a `$$hashKey` attribute.
- How factories are implemented on top of providers.
- How values are implemented on top of factories.
- How services are implemented on top of factories.
- How decorators work, by overriding the `$get` method of the provider they decorate.
- How decorators use the `locals` argument of `injector.invoke` to make the decorated `$delegate` available for injection.
- That there’s an ng module that holds Angular’s core components in every Angular application.
- How filters, scopes, and the expression parser integrate with the dependency injection features.
- How you can adjust the digest TTL by calling a method on `$rootScopeProvider`.