在一个分层良好的 Angular 应用中，Controller 这一层应该很薄。也就是说，应用里大部分的业务逻辑和持久化数据都应该放在 Service 里。为此，理解 AngularJS 中的几个 Provider 之间的区别很有必要。

Provider 创建的新服务都可以用来注入。包括：

- provider
- factory
- service
- constant
- value

另外，内建的服务 `$controller` 和 `$filter` 也可以被注入，同时也可以使用这些服务来获得新的过滤器和控制器。

下面介绍一下各自的用法和 provider、factory、value 三者之间的区别。

provider

用于产生一个可配置的 Service，由两部分组成。第一部分的变量和函数是可以在 `app.config` 函数中访问的，可以在它们被其他地方访问到之前来修改它们。定义方式如下：

```
app.provider('myProvider', function(){
    var a = '';
    var func = function(){};
})
```

在 `app.config` 函数对 `a` 进行修改：

```
app.config(function(myProviderProvider){
    myProvider.a = 'hello world';
})
```

这也是在有如此简单的 factory 的情况下还使用 provider 的原因。

第二部分的变量和函数是通过 `$get()` 函数返回的，可以在任何传入了该 provider 的控制器中进行访问的。

```
app.provider('myProvider', function(){
    this.$get = function(){
        return {
            foo: function(){},
            a: a
        }
    }
})
```

factory

factory 返回一个对象。只需要创建一个对象，为它添加属性，然后返回这个对象。在控制器中注入该 factory，即可使用它的所有属性。

```
app.factory('myFactory', function(){
    var fac = {};
    fac.a = 'hello world';
    fac.foo = function(){};
    return fac;
})
```

看得出来，factory 的第二个参数就是 provider 中 `$get` 要对应的函数实现。

service

service 类似于一个构造器， 通过 `new` 关键字实例化对象，将一些属性和方法直接添加到 `this` 上，在创建 service 对象时， `this` 会被作为返回值返回。

```
app.service('myService', function(){
    var a = '';
    this.setA = function(){};
    this.getA = function(){};
    this.foo = function(){};
})
```

注入 `myService` 的控制器可以访问到绑定在 `myService` 中 `this` 上的 `setA()` , `getA()` 和 `foo()` 三个方法。

constant

constant 用于定义常量，一旦定义就不能被改变。可以被注入到任何地方，但是不能被装饰器(decorator)装饰。

```
app.constant('APP_KEY', 'a1s2d3f4')
```

value

与 constant 一样，可以用来定义值。但与 constant 的区别是：可以被修改，可以被 decorator 装饰，不能被注入到 config 中。

```
app.value('version', '1.0')
```

value 通常用来为应用设置初始值。

decorator

比较特殊，它不是 provider 。它是用来装饰其他 provider 的，不过 constant 除外，因为从源码可以看出，constant 不是通过 `provider()` 方法创建的。

下面是一个用 decorator 装饰 value 的栗子。

```
app.value('version', '1.0');
app.decorator('version', function ($delegate) {
    return $delegate + '.1';
})
```

那如果要使用前面的 `myService` service，但是其中缺少一个你想要的 greet 函数。可以修改 service 吗？答案是不行！但是可以装饰它：

```
app.decorator('myService', function($delegate){
    $delegate.greet = function(){
        return "Hello, I am a new function of 'myService'";
    }
})
```

`$delegate` 代表实际上的 service 实例。

装饰一个 service 的能力是非常实用的，尤其是当我们想要使用第三方的 service 时，此时不需要将代码复制到我们的项目中，而只需要进行一些修改即可。

源码

下面是 provider、factory、service、value、constant 和 decorator 的底层实现：

```
function provider(name, provider_) {
    assertNotHasOwnProperty(name, 'service');
    if (isFunction(provider_) || isArray(provider_)) {
        provider_ = providerInjector.instantiate(provider_);
    }
    if (!provider_.$get) {
        throw $injectorMinErr('pget', "Provider '{0}' must define $get factory method.", name);
    }
    return providerCache[name + providerSuffix] = provider_;
}

function factory(name, factoryFn, enforce) {
    return provider(name, {
        $get: enforce !== false ? enforceReturnValue(name, factoryFn) : factoryFn
    });
}

function service(name, constructor) {
    return factory(name, ['$injector', function($injector) {
        return $injector.instantiate(constructor);
    }]);
}

function value(name, val) { return factory(name, valueFn(val), false); }
```

```
function constant(name, value) {
    assertNotHasOwnProperty(name, 'constant');
    providerCache[name] = value;
    instanceCache[name] = value;
}

function decorator(serviceName, decorFn) {
    var origProvider = providerInjector.get(serviceName + providerSuffix),
        orig$get = origProvider.$get;

    origProvider.$get = function() {
        var origInstance = instanceInjector.invoke(orig$get, origProvider);
        return instanceInjector.invoke(decorFn, null, {$delegate: origInstance});
    };
}
```

可以看出，除了 constant，另外几个最终调用的都是 provider（decorator比较特殊，不算）。

总结

可以看出，从 service 到 factory 再到 provider，复杂程度依次递增，那么问题来了。

什么时候使用 provider 而不用 factory ？

provider 是 factory 的加强版。当需要一个可配置的 factory 的时候，使用 provider。

简单介绍一下 AngularJS 运行应用的过程，分两个阶段，config 阶段和 run 阶段。config 阶段是设置任何的 provider 的阶段。也是设置任何的指令，控制器，过滤器以及其它东西的阶段。在 run 阶段，AngularJS 会编译你的 DOM 并启动应用。

factory 和 service 的区别是什么？

factory 是普通 function，而 service 是一个构造器(constructor)，这样 Angular 在调用 service 时会用 new 关键字，而调用 factory 时只是调用普通的 function，所以 factory 可以返回任何东西，而 service 可以不返回。