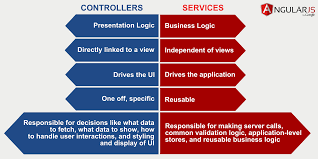
1. **dWhat is a Directive?** - extend HTML with custom, new attributes and elem called **Directives**.

Directives are notes to the AngularJS compiler that allow we to attach behavior to a DOM object.

**In the simplest terms, a directive in AngularJS is a reusable component and independent functionality on a DOM element.** Directive is one of the most powerful tools of AngularJS, and it includes attributes such as ng-show, ng-include, and ng-bind. In addition to this, AngularJS allows developers to define custom directives that express the application specifications and requirements much more clearly and semantically than regular HTML elements.

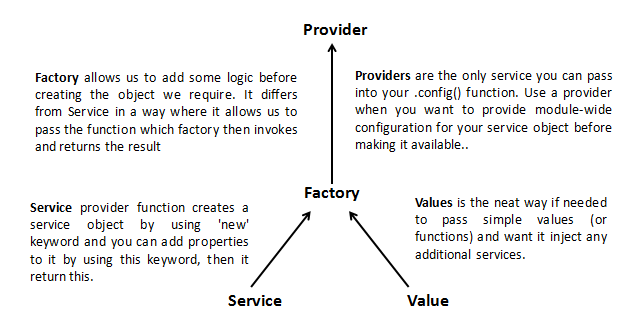


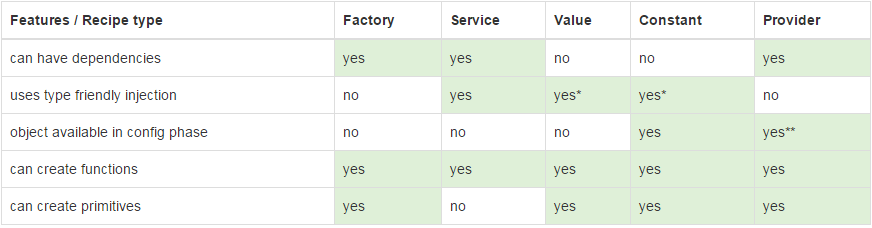
[**Two-Way Data Binding**](http://kodypeterson.com/angularjs-the-3-types-of-data-binding/)

Two way data binding in angularjs framework is approach to **synchronize** the data between **model** and **view**. What it means that if there is any change happens in model ( Back-end ) then view ( front-end ) will be updated and vice versa.

Data binding means that when you change something in the view, the scope model automagically updates

**2.Service** - In AngularJS, services are reusable singleton objects that are used to organize and share code across your app. They can be injected into controllers, filters, directives. AngularJS provides you five ways : service, factory and provider, value, constant to create a service.





**DIFF service and factory**

There's a simple explanation behind why there are two ways: flexibility and support. Many JavaScript developers prefer the method of using a function to return the object (i.e. factory). This is great, except that when you use something like CoffeeScript or TypeScript it for the most part guides you to use a constructor function instead. **The service approach supports this "class-based" approach rather than traditional function-based approach. So, the real easy way to differentiate is that factory supports a function to create a component, while service supports a constructor function. Factory will call the function, Service will new the function**.

**Services-**  are responsible for fetching data from the server and then sharing it across the application $http{method :GET , url: api key}

Thus service is a stateless object that contains some useful functions. These functions can be called from anywhere; Controllers, Directive, Filters etc. Thus we can divide our application in logical units. The business logic or logic to call HTTP url to fetch data from server can be put within a service object.

**3.$q**- A service that helps you run functions asynchronously, and use their return values (or exceptions) when they are done processing.

**Methods-** resolve,reject, notify

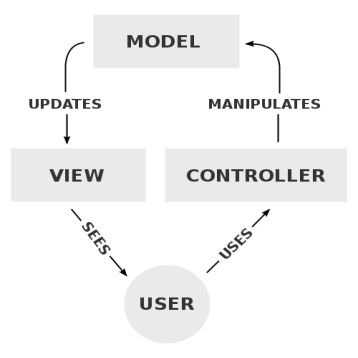
**Properties-** promise then (successCallback, [errorCallback], [notifyCallback])

**4. $watch**- This function is used to **observe changes in a variable on the $scope**. It accepts three parameters: expression, listener and equality object, where listener and equality object are optional parameters. **$watch (watchExpression, listener, [objectEquality])**

**5. $digest()** - This function **iterates through all the watches in the $scope object**, and its **child $scope** objects (if it has any).

**6. $apply()** - Angular do **auto-magically updates only those model changes** which are inside AngularJS context

**7. $digest() is faster than $apply()**, since $apply() triggers watchers on the entire scope chain i.e. on the current scope and its parents or children (if it has) while $digest() triggers watchers on the current scope and its children(if it has).



**1. Model :** The Model is the **logical structure of the data in the application**.It is not to be confused with **database**, and instead it should be seen as the structure of code that represents the data.

**2. View:** A view is the code structure which represents the User Interface, User Interface constitutes everything the user can see, buttons, dialog boxes, modals, menu etc.A given application can have multiple views, and each view represents some section of the model.

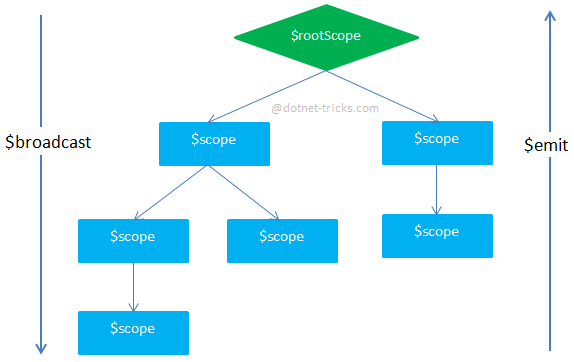
**3. Controller:** A Controller is an *intermediary* between the view and model, it acts the logic or brain of the application.

**4. Scope :** A Scope can be defined as **context in between the model and function**. The task of a controller is to set these models and functions in the scope.

**5. Directives :** It teaches new properties to HTML syntax. It essentially HTML elements with custom elements and attributes.

**6. Expression :** AngularJS expression are denoted using symbol *{{}}* in a HTML document. Expressions are used to access scope models and functions.

AngularJS provides **$on, $emit,- child scope to parent level scope hierachy(**propagates **upward** andtravel up to **$rootScope.scope**) **and $broadcast** – (propagate **downward** towards the children scope). services for event-based communication between controllers.



**Pros**:1.) Easy way to communicate between different Controllers.

**Cons**: 1.) Not easy way to track the flow of an application.2.) Decrease performance.3.) When we use $rootScope then need to manually destroy.

Directive:

1. **Compile** - It traverse the DOM and collect all of the directives. The result is a linking function.(It is executed after the directive is compiled).

2. **Link** - It combines the directives with a scope and produces 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 mode( a function that runs before the directive is compiled).

**ISOLATED SCOPE**

**Scope** in AngularJS inherits from **Parent Scope** default. But, sometimes it is not required specially in the case of development of some common components. In this case directive should not read or write properties value in the parent scope by mistake. So, Isolated Scope came into picture.

Isolated scope does not prototypically inherit from the parent scope. It can access its parent scope through the **$parent** property.  So, Directive has three options for isolating its scope from parent scope. The following are the three options:

1. **scope: false** **-** It is default in Directive. It lets to reuse the scope from the place where the component is being used.
2. **scope: true -** It creates a child scope. This child scope prototypically inherits from the scope where the component is being used.
3. **scope: {...} -** It creates Isolates scope. It does not prototypically inherit from the scope where the component is being used.