



Factories and Services



© Wahlin Consulting – All Rights Reserved

Factory and Service Overview

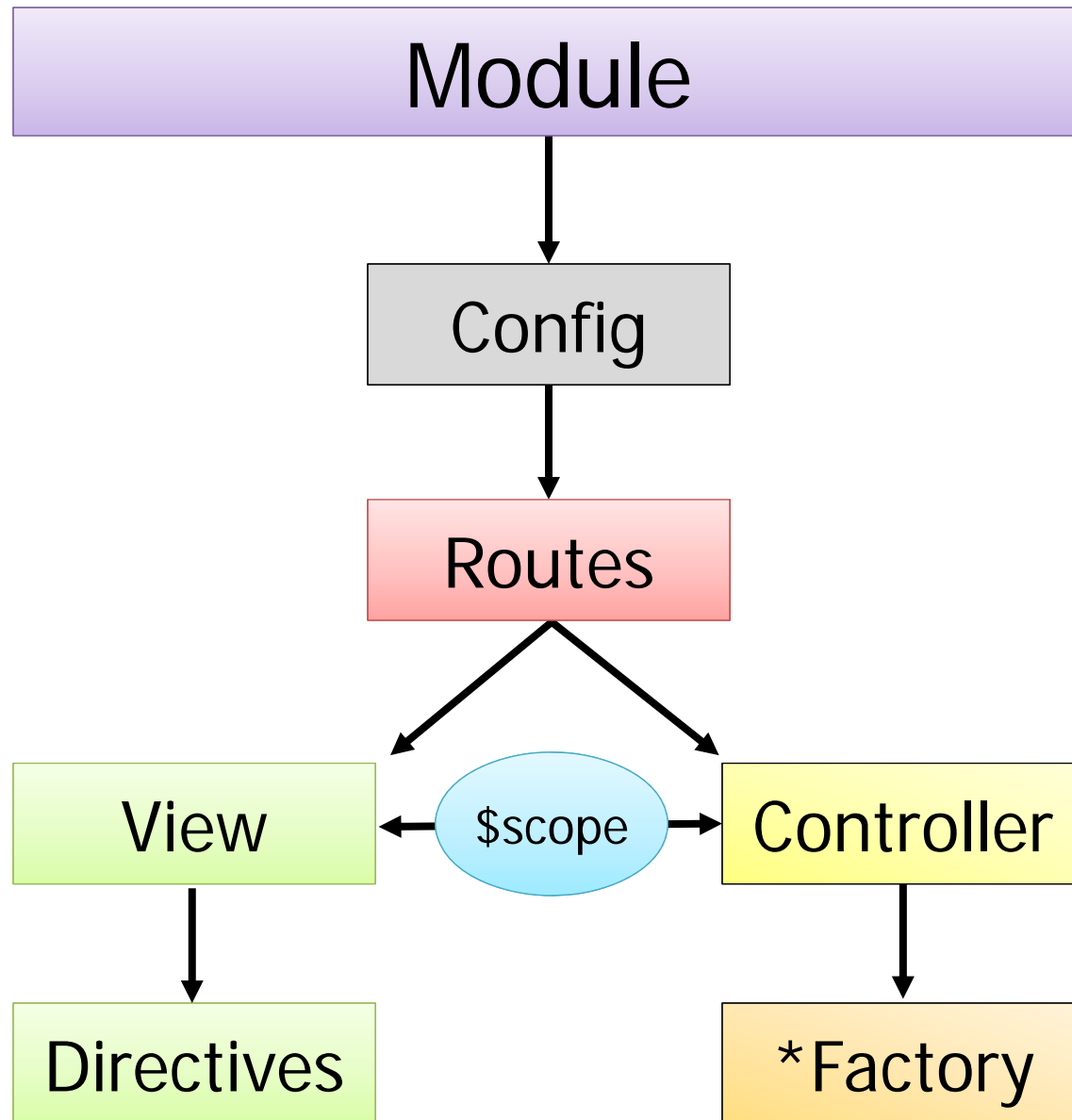
Creating a Factory

Creating a Service

Defining Application Values

Making Ajax Calls from a Factory/Service

The Big Picture

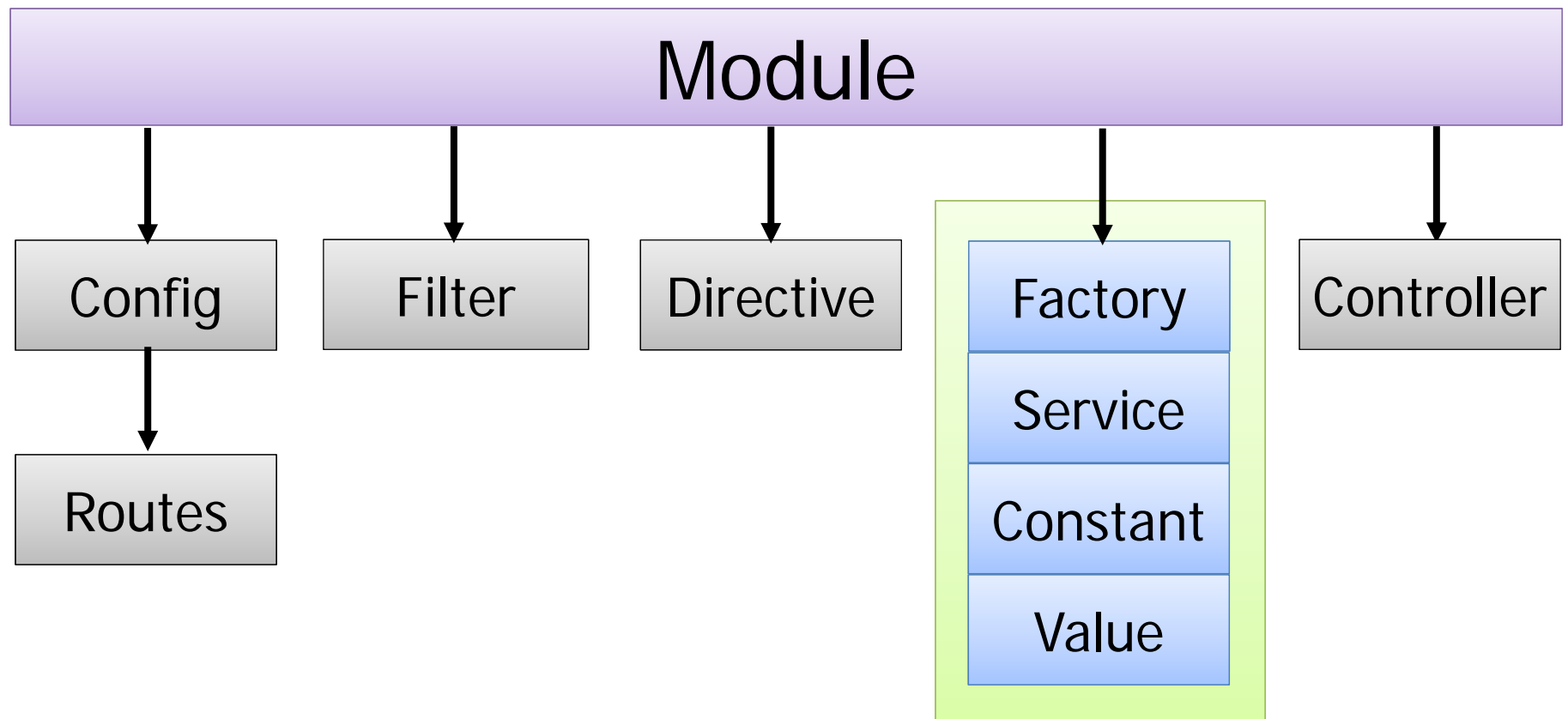


Factory and Service Overview

Factories and Services

- **AngularJS supports the concept of factories and services**
- **Includes several built-in factories/services**
- **Singletons that perform re-useable tasks:**
 - Ajax calls
 - Business rules
 - Calculations
 - Share data between controllers

Creating Factories, Services and More



Built-in AngularJS Services

- AngularJS provides many built-in services:

\$http

\$timeout

\$window

\$location

\$q

\$rootScope

\$interval

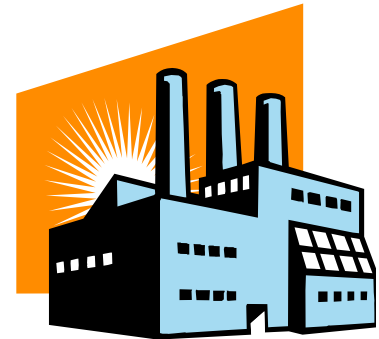
\$filter

\$log



Creating a Factory

What is a Factory?

- **Create a custom factory when you need to:**
 - Define re-useable tasks
 - Share code or state between controllers
- **Factories:**
 - Create and return a custom object
 - Created using the `module.factory()` function
 - Can be injected into other components
 - Can have dependencies



Creating a Factory

```
(function () {  
  
    var customersFactory = function () {  
        var customers = [...];  
        var factory = {};  
        factory.getCustomers = function () {  
            return customers;  
        };  
        return factory;   
    };  
  
    angular.module('customersApp')  
        .factory('customersFactory', customersFactory);  
  
}()); 
```

Alternate Syntax

```
(function () {  
    var customersFactory = function () {  
        var customers = [...];  
  
        return {  
            getCustomers: function () {  
                return customers;  
            }  
        };  
    };  
});  
  
angular.module('customersApp')  
    .factory('customersFactory', customersFactory);  
})();
```

Return object literal

Injecting a Factory



Inject factory

```
var CustomersController = function ($scope, customersFactory)
{
    function init() {
        $scope.customers = customersFactory.getCustomers();
    }
};

CustomersController.$inject = ['$scope', 'customersFactory'];

angular.module('customersApp')
    .controller('CustomersController', CustomersController);
```

Creating a Service

What is a Service?

- **Similar to a factory as far as functionality**
- **Services:**
 - Service function represents the returned object as opposed to a custom object like in a factory
 - Created using the `module.service()` function
 - Can be injected into other components
 - Can have dependencies



Creating a Service

```
(function () {
```

Function represents the
returned object

```
    var customersService = function () {  
        var customers = [ ... ];
```

Relies on
"this"

```
        this.getCustomers = function () {  
            return customers;
```

```
        };
```

```
    };
```

```
    angular.module('customersApp')
```

```
        .service('customersService', customersService);
```

```
})();
```

Add service to module

Injecting a Service

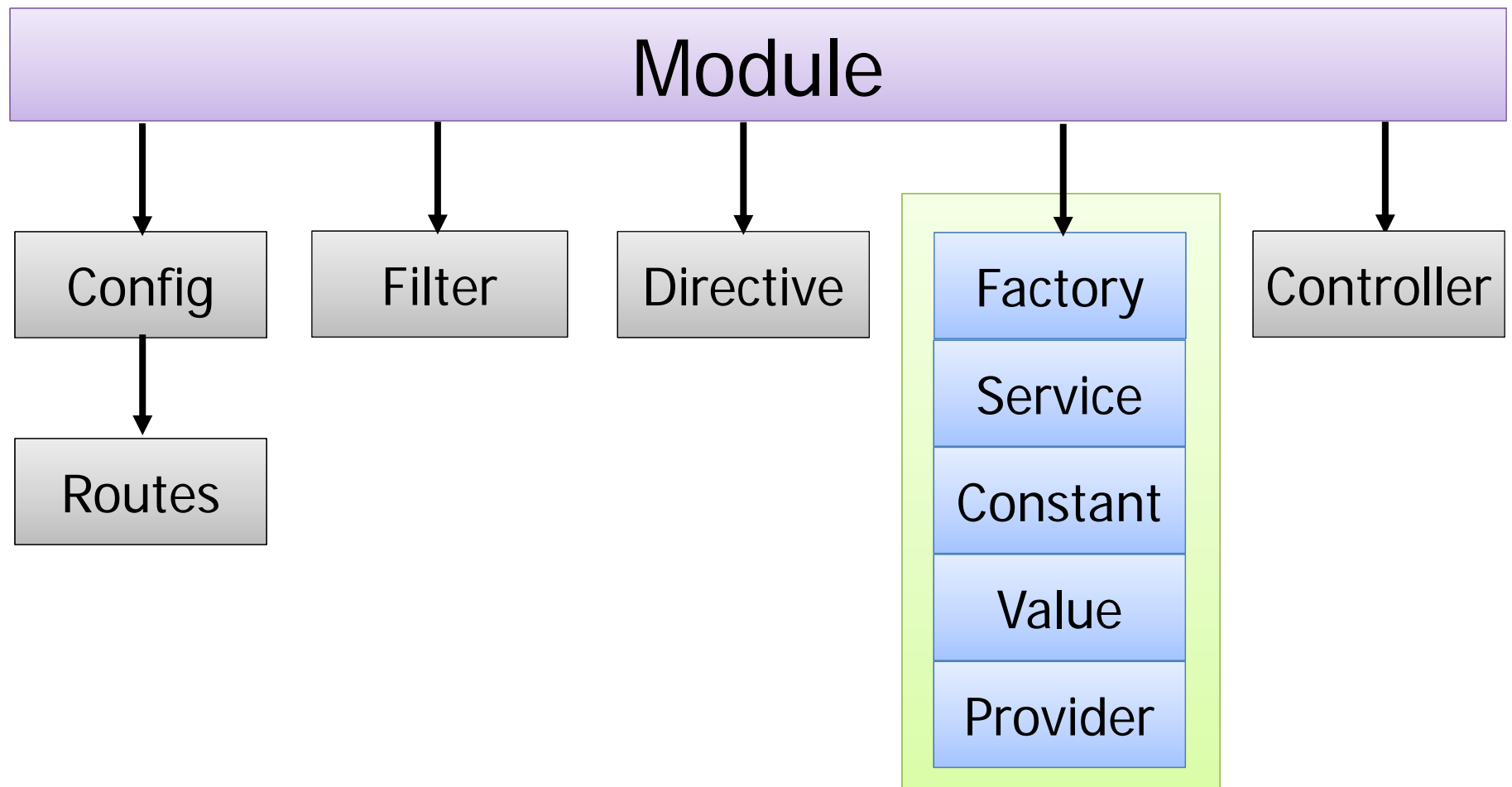


Inject service

```
var CustomersController = function ($scope, customersService) {  
    function init() {  
        $scope.customers = customersService.getCustomers();  
    }  
};  
  
CustomersController.$inject = ['$scope', 'customersService'];  
  
angular.module('customersApp')  
    .controller('CustomersController', CustomersController);
```


Defining Application Values

Beyond Factories and Services



Value and Constant

- An *value* can be created using `module.value(key,value)`:

```
angular.module('customersApp').value('version', '1.0');
```

Can't be injected
into config()

- A *constant* can be created using `module.constant(key,value)`

```
angular.module('customersApp').constant('version', '1.0');
```

Can be injected into
config()

Making Ajax Calls from a Factory/Service

AngularJS and Ajax

- AngularJS services can be used to make Ajax calls

\$http

\$resource

The \$http Service

- The \$http service provides Ajax functionality
- Uses the browser's XMLHttpRequest object
- Makes asynchronous requests
- Supports multiple HTTP verbs:
 - \$http.head
 - \$http.put
 - \$http.get
 - \$http.delete
 - \$http.post
 - \$http.jsonp



Using the \$http Service

```
(function () {
```

Service injected

```
    var customersFactory = function ($http) {  
        var factory = {};  
        factory.getCustomers = function () {  
            return $http.get('api/customers');  
        };  
        return factory;  
    };  
};
```

Make async call

```
angular.module('customersApp')  
    .factory('customersFactory', customersFactory);
```

```
})();
```

Accessing \$http Response Data

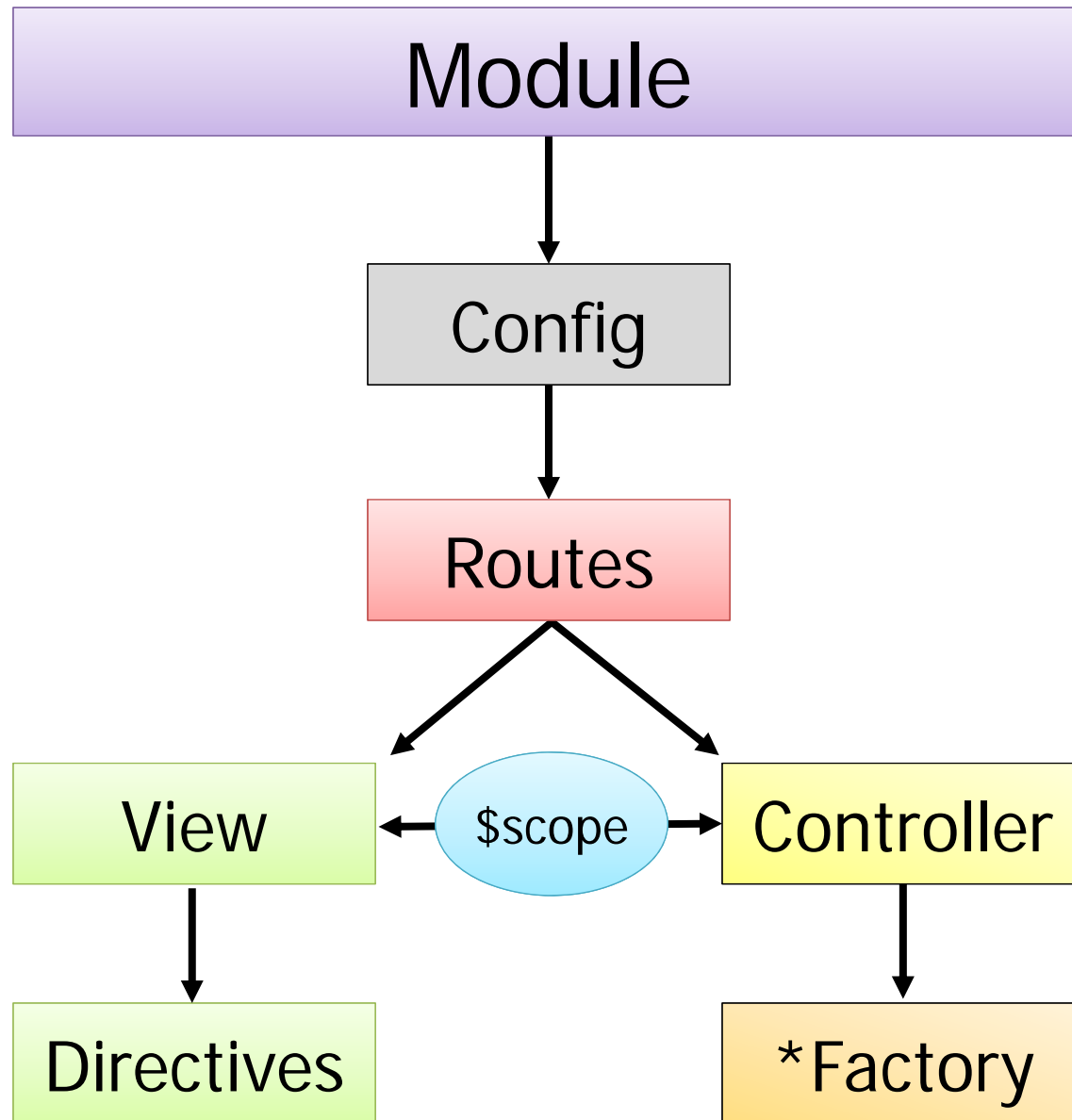
- **\$http makes asynchronous calls:**
 - Relies on \$q service's deferred/promise APIs
 - Access data by calling **then()** or **success()/error()**

```
customersFactory.getCustomers()  
    .success(function (customers) {  
        $scope.customers = customers;  
    })  
    .error(function (data, status, headers, config) {  
        //error!  
    });
```

Called within
controller

Summary

The Big Picture



Summary

- **Factories and services are singletons**
- **Provide a place to store re-useable tasks:**
 - Ajax calls
 - Business rules
 - More...
- **Create using a module's factory() or service() functions**
- **Access by injecting the factory/service**