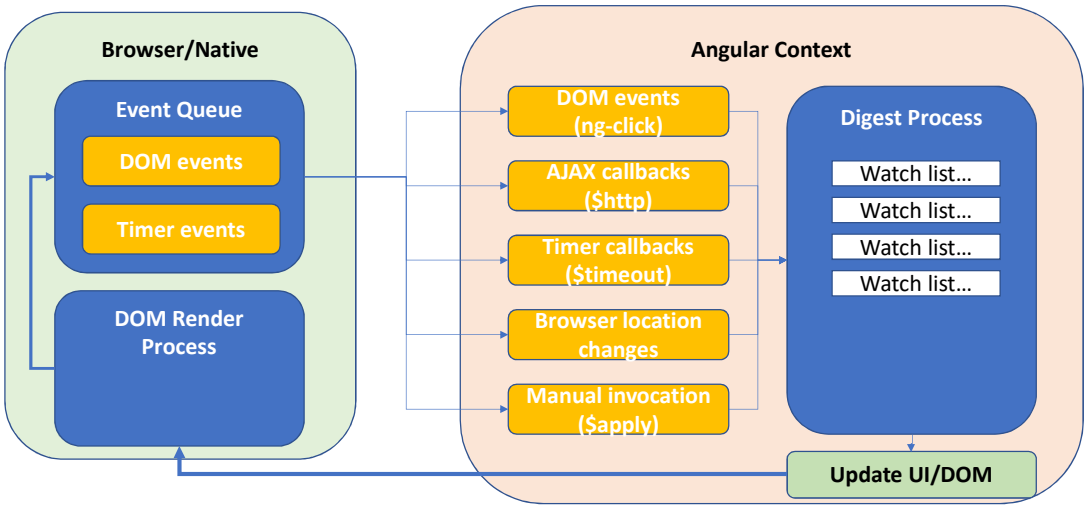
ANGULARJS by Google

Amateur

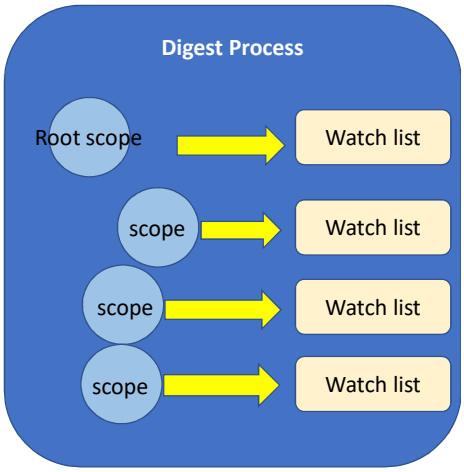
Darryl Ng
issnes@nus.edu.sg
Institute of System Science
National University of Singapore

Digest Process Overview



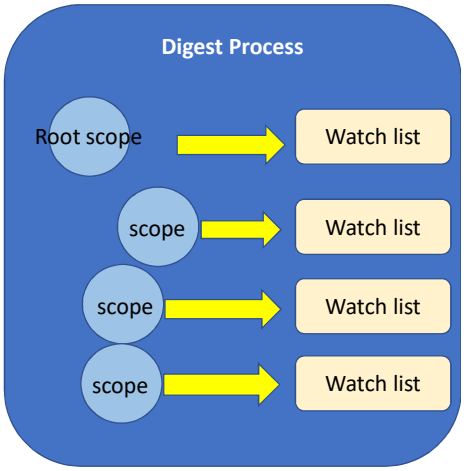
$apply & $digest

- **`$apply, $digest`**
  - Used to kick-in digest process manually
  - Used when scope variables are modified outside 'angular context'
  - UI needs to refresh its data-bindings
  - Usage. **`$scope.apply()`**, **`$scope.digest()`**



$apply & $digest

- **`$apply`**
  - Kick-in digest process on RootScope always
  - Process continues through all child scopes after evaluating RootScope
  - ng-click, $timeout, $http(ajax) operations call $apply
- **`$digest`**
  - Kick-in digest process for current scope only
  - Doesn't start from RootScope



Updating Bindings

- `$apply()` will be trigger whenever a binding is updated in AngularJS context
  - Eg button click inside a controller
- Bindings are not updated when changes are made outside of AngularJS context
  - Eg. in jQuery, callbacks like `setTimeout()`
- `$apply()` need to be called manually so that Angular can update the bindings

Practical Scenario - 1

```
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular.min.js"></script>
    <script type="text/javascript" src="a01.js"></script>
  </head>
  <body ng-app="app">
    <div id="div1" ng-controller="amp">
      <h1>Counter: {{ counter }}</h1>
    </div>
  </body>
</html>
```

```
var app = angular.module('app', []);
app.controller("amp", ["$scope", MyCtrl]);

function MyCtrl($scope) {
        $scope.counter = 0;

        var ticktock = function() {
                $scope.counter++;
                setTimeout(ticktock, 1000);
        };
        setTimeout(ticktock, 1000);
}
```

AngularJS is oblivious to this mutation

Callbacks are executed outside of AngularJS' context. `$scope.counter` is updated but binding in view does not change because `$apply()` is not called

$apply()

- Update the bindings, causes all values to be updated
- Need to call this if bindings are changed outside of AngularJS context
- 3 forms
  - `$apply()` – update all bindings
  - `$apply(function)` – execute the function in the context of AngularJS
  - `$apply(string)` – execute the AngularJS expression in the context of AngularJS
    - Eg `$apply("counter = counter + 1")`
      - Similar to `$scope.counter = $scope.counter + 1`

Practical Scenario - 2

```
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular.min.js"></script>
    <script type="text/javascript" src="a01.js"></script>
  </head>
  <body ng-app="app">
    <div id="div1" ng-controller="amp">
      <h1>Counter: {{ counter }}</h1>
    </div>
  </body>
</html>
```

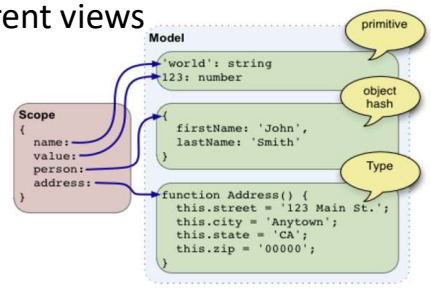
This causes a `$digest()` cycle

Note: Angular has a special service call `$timeout()` which is an Angularized version of `setTime()`

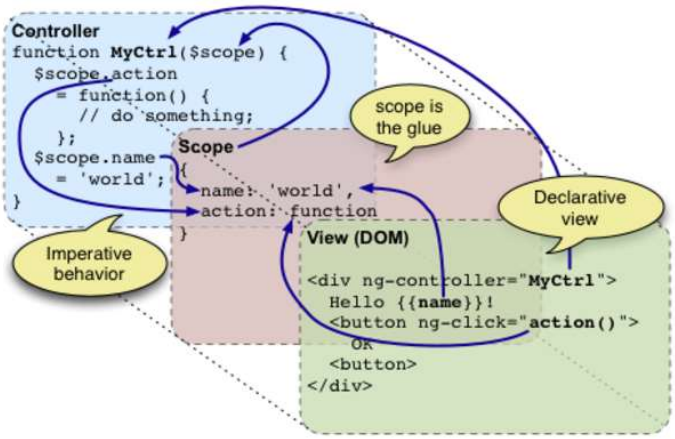
```
var app = angular.module('app', []);
app.controller("amp", ["$scope", MyCtrl]);

function MyCtrl($scope) {
        $scope.counter = 0;
        var ticktock = function() {
                $scope.$apply(function() {
                        $scope.counter++;
                });
                setTimeout(ticktock, 1000);
        };

        setTimeout(ticktock, 1000);
}
```

A Little Recap - 1

- Data binding bin model and view for _____
- _____ helps you organize data in certain way
- _____ breaks functionality for maintainability
- _____ determines the sequence to different views
- _____ decides the events to be handled



A Little Recap - 2



https://www.codeproject.com/Articles/845939/AngularJs-Quick-Start

AngularJS $scope, $emit, and $on

- **Emit** – **Upwards**
  - dispatch an event ***upwards*** through the scope hierarchy
- **Broadcast** – **Downwards**
  - dispatch an event from the parent scope ***downwards*** to all child scopes
- **Broadcast** - **No relationship**
  - ***no parent/child relationships*** between the two controllers
  - use ***$rootScope*** to broadcast the event

```
function parentCtrl($scope){
  $scope.$on('myEvent', function(event, data) {
    console.log(data);
  });
}
```

```
function childCtrl($scope){
  $scope.$emit('myEvent', 'my data');
}
```


Services

- Services and factories are abstractions for encapsulating reusable code
- Service provides cross-cutting concerns
  - "Horizontal" services like authentication, logging, persistence, etc.
- Characteristics of a service
  - Singleton – e.g. typically you only have a SINGLE logging service, therefore should return the same instance
  - Stateless – should not hold data specific to a client/controller

Creating a Service

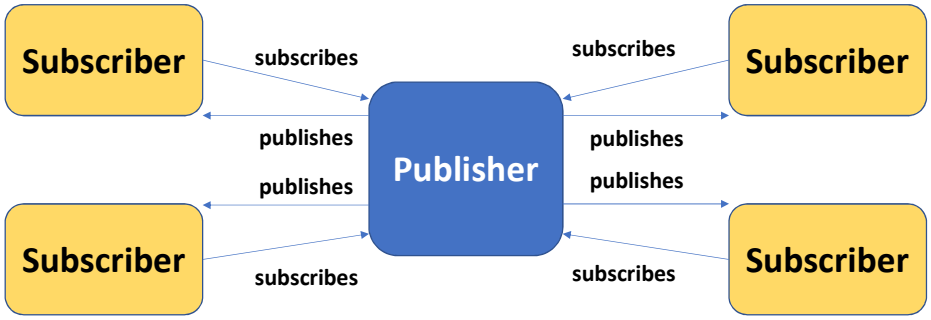
- Services are create with **`myModule.service()`**
- Pass to **`service()`** name and a service creating object
  - The service object configures
  - AngularJS uses your service object to create the service
  - You do not create the service, Angular creates it on your behalf
- Add methods to the service object using this
- Dependencies can be injected into a service

```
MyApp.service("MyService", ["$http", function($http) { ... }];
```


Factories

- Factories are used to create 'things'
  - Things can be anything – a JSON object, a function, a value, etc
  - More flexible than service; services must always return a object
- If you need a 'service' that creates a new object whenever you call it, then that is a good candidate for a factory
  - Eg. Can use to create objects that a stateful.
- Factories are created with **`myModule.factory()`**
  - Similar to a service
  - Angular calls the factory function without the `new` keyword
  - Your factory is responsible for creating the 'thing'
- Need to add `ngResource` to the module's list of dependencies

Communicating Between Controllers Pub/Sub Pattern Model



Communicating Between Controllers - 1 HTML code

```
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular.js"></script>
    <script type="text/javascript" src="b01.js"></script>
  </head>
  <body ng-app="controllerApp">
        <div ng-controller="FirstController">
          <input ng-model="sharedmessage" >
          <button ng-click="publish(sharedmessage);">LOG 1</button>
        </div>
        <div ng-controller="SecondController">
          <input ng-model="sharedmessage" >
          <button ng-click="publish(sharedmessage);">LOG 2</button>
        </div>
        <div ng-controller="ThirdController">
          <input ng-model="sharedmessage" >
          <button ng-click="publish(sharedmessage);">LOG 3</button>
        </div>
  </body>
</html>
```

Communicating Between Controllers - 1
Create Angular Module & Factory Service

```
var app = angular.module('controllerApp', []);

app.factory('mySharedService', function($rootScope) {
    var sharedService = {};

    sharedService.sharedmessage  = '';

    sharedService.prepForPublish = function(msg) {
        this.sharedmessage  = msg;
        this.publishItem();
    };

    sharedService.publishItem = function() {
        $rootScope.$broadcast('handlePublish');
    };

    return sharedService;
});
```


Communicating Between Controllers - 2
Define controller properties and events

```
function FirstController($scope, sharedService) {
    $scope.publish = function(msg) {
        sharedService.prepForPublish(msg);
    };

    $scope.$on('handlePublish', function() {
        $scope.sharedmessage  = sharedService.sharedmessage ;
    });
}

function SecondController($scope, sharedService) {
. . .
}

function ThirdController($scope, sharedService) {
. . .
}
```

Communicating Between Controllers - 3
Define controller properties and events

```
function FirstController($scope, sharedService) {
    $scope.publish = function(msg) {
        sharedService.prepForPublish(msg);
    };

    $scope.$on('handlePublish', function() {
        $scope.sharedmessage  = sharedService.sharedmessage ;
    });
}

function SecondController($scope, sharedService) {
. . .
}

function ThirdController($scope, sharedService) {
. . .
}
```


Communicating Between Controllers - 4
Register controllers

```
app.controller('FirstController', ['$scope', 'mySharedService', FirstController]);

app.controller('SecondController', ['$scope', 'mySharedService', SecondController]);

app.controller('ThirdController', ['$scope', 'mySharedService', ThirdController]);
```

Communicating Between Controllers - 5 Dependency Injection

```
FirstController.$inject = ['$scope', 'mySharedService'];

SecondController.$inject = ['$scope', 'mySharedService'];

ThirdController.$inject = ['$scope', 'mySharedService'];
```


What are Deferred and Promise?

Promise
Customer will get coffee some time in the near future

Deferred
Proprietor prepares the cup of coffee

Resolved
When proprietor signals customer to collect coffee

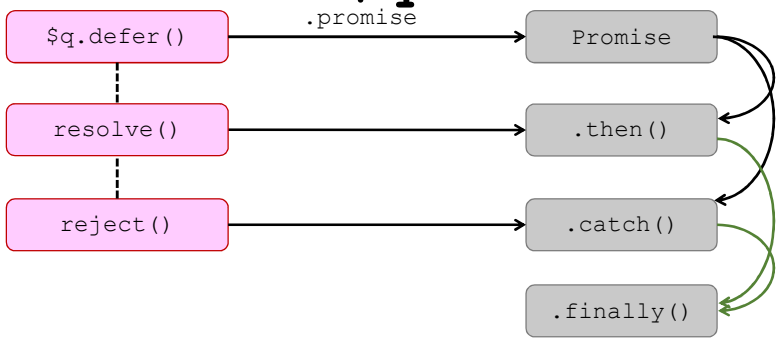
Asynchronous Environment

- JavaScript runs in a single threaded environment
- Prevent blocking of the main thread, all method calls are asynchronous
  - When method completes, invoke the callback to receive result
- Deferred is a way of notifying the invoker that the method has complete its execution
- Another use is to coordinate multiple asynchronous calls
  - Eg. Cordova - when document is loaded and camera is ready, enable 'Take Picture' button

Deferred and Promise

- A promise is always associated with a deferred
- Promise can be in one of 3 states
  - Pending – no resolution yet
  - Resolved – the result of the deferred is success
  - Reject – the result of the deferred is fail
- A promise is said to be resolved if we know the result
  - Either resolved (success) or rejected
- Once a promise has been resolved, in stays resolved
  - Cannot reset its state, use only once
  - Need to create a new promise
- Only a deferred can resolve its corresponding promise

AngularJS Deferred – `$q`



- **`finally`** is a reserved word in Javascript, some ECMAScript environment will flag that as an error
- Workaround is to have **`finally`** in quotes

```
promise["finally"](function() { ... })
```


Sequential Task and Promise Chains

- Scenario where a sequence of task, each dependent on a previous task
- The execution of the next task is dependent on the successful resolution of the previous task
- **`then()`** returns another promise
  - Can associate it with another callback
  - Callback will be invoked if promise is resolved successfully

```
promise.then(function() { ... }) //task 1
    .then(function() { ... }) //task 2
    .then(function() { ... }); //task 3
```

Handling Errors in Chains

- **`then()`** method call can accommodate an error callback
  - The `.catch()`

```
promise.then(successCallback(), errorCallback());
```

```
promise.then(function() { ... }, function() { ... }) //task 1
    .then(function() { ... }, function() { ... }) //task 2
    .then(function() { ... }, function() { ... }); //task 3
```


Working with callbacks – 1
HTML code

```
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular.js"></script>
    <script type="text/javascript" src="b02.js"></script>
  </head>
  <body ng-app="myModule">
        <div ng-controller="HelloCtrl">
          <input ng-model="messages" >
          <button ng-click="publish(sharedmessage);">LOG 1</button>
        </div>

  </body>
</html>
```

Working with callbacks – 2 Angular App

```
var myModule = angular.module('myModule', []);
// From this point on, we'll attach everything to 'myModule'
myModule.factory('HelloWorld', function($timeout) {
    var getMessages = function(callback) {
      $timeout(function() {
        callback(['Hello', 'world!']);
      }, 2000);
    };

    return {
      getMessages: getMessages
    };
});

myModule.controller('HelloCtrl', function($scope, HelloWorld) {
    HelloWorld.getMessages(function(messages) {
      $scope.messages = messages;
    });
});
```


Upgrading to Promise

```
var myModule = angular.module('myModule', []);

// From this point on, we'll attach everything to 'myModule'
myModule.factory('HelloWorld', function($q, $timeout) {
  var getMessages = function() {
    var deferred = $q.defer();

    $timeout(function() {
      deferred.resolve(['Hello', 'world!']);
    }, 2000);
    return deferred.promise;
  };

  return {
    getMessages: getMessages
  };
});

myModule.controller('HelloCtrl', function($scope, HelloWorld) {
    HelloWorld.getMessages().then(function(messages) {
      $scope.messages = messages;
    });
});
```

Why do we need promises?

- make decisions based on the possible results of our call
  - contact external API to get the list of clients
  - do something else while waiting for response
  - once response is received, program displays client info on screen
  - if no response, then display message to the end user
- **`$q`** service
  - deal with promises
  - work with asynchronous functions

More Promises Example - 1

```
var deferred = $q.defer();

// deferred contains the promise to be returned
// to resolve (fulfill) a promise use .resolve
deferred.resolve(data);

// to reject a promise use .reject
deferred.reject(error);
```

More Promises Example – 2 Angular Module

```
var app = angular.module("app", []);
```

http://plnkr.co/edit/kACAcbCUIGSLRHV0qojK?p=preview

More Promises Example – 3 Angular service

```
app.service("githubService", function ($http, $q) {
    var deferred = $q.defer();

    this.getAccount = function () {
        return $http.get('https://api.github.com/users/haroldrv')
            .then(function (response) {
                // promise is fulfilled
                deferred.resolve(response.data);
                // promise is returned
                return deferred.promise;
            }, function (response) {
                // the following line rejects the promise
                deferred.reject(response);
                // promise is returned
                return deferred.promise;
            })
        ;
    };
});
```

More Promises Example – 4 Angular controller

```
app.controller("promiseController", function ($scope, $q, githubService)
{

    githubService.getAccount()
        .then(
            function (result) {
                // promise was fullfilled (regardless of outcome)
                // checks for information will be peformed here
                $scope.account = result;
            },
            function (error) {
                // handle errors here
                console.log(error.statusText);
            }
        );
});
```


More Promises Example – 5 HTML code

```
<!DOCTYPE html>
<html>
  <head>
    <script data-require="angular.js@*" data-semver="1.3.15"
src="https://code.angularjs.org/1.3.15/angular.js">
    </script>
    <link rel="stylesheet" href="style.css" />
    <script src="script.js"></script>
  </head>
  <body ng-app="app">
    <div ng-controller="promiseController">
      <h1>AngularJS $q service and promises</h1>
      Account details: <div>{{account | json }}</div>
    </div>
  </body>
</html>
```

고맙습니다

It’s only here…

- http://www.tothenew.com/blog/angularjs-deferred-promises-basic-understanding/

What and who are you looking at?