Understanding AngularJS $watch(), $digest() and $apply()

gularJS has a powerful data binding mechanism. When you do changes in a variable on the $scope object within your view, the $scope object auto-magically updates itself. Similarly, whenever the $scope object changes, the view updates itself with the new value. AngularJS handle data-binding mechanism with the help of three powerful functions: $watch(), $digest() and $apply(). Most of the time AngularJS will call the $scope.$watch() and $scope.$digest() functions for you, but in some cases you may have to call these functions yourself to update new values. Therefore it is really good to know how they work.

## **$watch()**

The $scope.watch() 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.

1. **<!DOCTYPE html>**
2. **<html>**
3. **<head>**
4. **<title>AngularJS Watch</title>**
5. **<script src="lib/angular.js"></script>**
6. **<script>**
7. **var myapp = angular.module("myapp", []);**
8. **var myController = myapp.controller("myController", function ($scope) {**
9. **$scope.name = 'seed infotech.com';**
10. **$scope.counter = 0;**
11. **$scope.$watch('name', function (newValue, oldValue) {**
12. **$scope.counter = $scope.counter + 1;**
13. **});**
14. **});**
15. **</script>**
17. **</head>**
18. **<body ng-app="myapp" ng-controller="myController">**
19. **<input type="text" ng-model="name" />**
20. **<br />**
21. **Counter: {{counter}}**
22. **</body>**
23. **</html>**

## **$digest()**

The $scope.$digest() function iterates through all the watches in the $scope object, and its child $scope objects (if it has any). When $digest() iterates over the watches, it checks if the value of the expression has changed. If the value has changed, AngularJS calls the change callback(listener) with the new value and the old value.

The $digest() function is called whenever AngularJS thinks it is necessary. For example, after a button click, or after an AJAX call. You may have some corner cases where AngularJS does not call the $digest() function for you. In that case you may have to call this function yourself.

1. **<!DOCTYPE html>**
2. **<html>**
3. **<head>**
4. **<title>AngularJS Digest</title>**
5. **<script src="lib/angular.js"></script>**
6. **<script>**
7. **var myapp = angular.module("myapp", []);**
8. **var myController = myapp.controller("myController", function ($scope) {**
10. **$scope.datetime = new Date();**
12. **$scope.updateTime = function () {**
13. **$scope.datetime = new Date();**
14. **}**
16. **document.getElementById("updateTimeButton").addEventListener('click', function () {**
17. **console.log("update time clicked");**
18. **$scope.datetime = new Date();**
20. **console.log($scope.datetime);**
21. **});**
22. **});**
23. **</script>**
25. **</head>**
26. **<body ng-app="myapp" ng-controller="myController">**
27. **<button ng-click="updateTime()">Update time - ng-click</button>**
28. **<button id="updateTimeButton">Update time</button>**
29. **<br />**
30. **{{datetime | date:'yyyy-MM-dd HH:mm:ss'}}**
31. **</body>**
32. **</html>**

When you will click on second button, the data binding is not updated. Since $scope.$digest() is not called after the second button's event listener is executed. In this way on clicking the second button the time will be updated in the $scope.data.time variable, but the new time will never displayed.

To fix this issue you need to add a $scope.$digest() call to the second button event listener, like this:

1. **<script type="text/javascript">**
2. **document.getElementById("updateTimeButton").addEventListener('click', function () {**
3. **console.log("update time clicked");**
4. **$scope.datetime = new Date();**
6. ***//to update $scope***
7. **$scope.$digest();**
8. **console.log($scope.datetime);**
9. **});</script>**

## **$apply()**

Angular do auto-magically updates only those model changes which are inside AngularJS context. When you do change in any model outside of the Angular context (like browser DOM events, setTimeout, XHR or third party libraries), then you need to inform Angular of the changes by calling $apply() manually. When the $apply() function call finishes AngularJS calls $digest() internally, so all data bindings are updated.

In above example, instead of calling $digest() function inside the button listener function you can used the $apply() function like this:

1. **<script>**
2. **document.getElementById("updateTimeButton").addEventListener('click', function () {**
3. **$scope.$apply(function () {**
4. **console.log("update time clicked");**
5. **$scope.datetime = new Date();**
7. **console.log($scope.datetime);**
8. **});**
9. **});**
10. **</script>**

#### **Note**

1. $digest() is faster than $apply(), since $apply() triggers watchers on the entire scope chain while $digest() triggers watchers on the current scope and its children(if it has).
2. When error occurs in one of the watchers, $digest() can not handled errors via $exceptionHandler service, In this case you have to handle exception yourself. While $apply() uses try catch block internally to handle errors and if error occurs in one of the watchers then it passes errors to $exceptionHandler service.