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




Jakob Jenkov

Last update: 2014-08-28

The AngularJS `$scope` functions `$watch()`, `$digest()` and `$apply()` are some of the central functions in AngularJS. Understanding `$watch()`, `$digest()` and `$apply()` is essential in order to understand AngularJS.

When you create a data binding from somewhere in your view to a variable on the `$scope` object, AngularJS creates a "watch" internally. A watch means that AngularJS watches changes in the variable on the `$scope` object. The framework is "watching" the variable. Watches are created using the`$scope.$watch()` function which I will cover later in this text.

At key points in your application AngularJS calls the `$scope.$digest()` function. This function iterates through all watches and checks if any of the watched variables have changed. If a watched variable has changed, a corresponding listener function is called. The listener function does whatever work it needs to do, for instance changing an HTML text to reflect the new value of the watched variable. Thus, the`$digest()` function is what triggers the data binding to update.

Most of the time AngularJS will call the `$scope.$watch()` and `$scope.$digest()` functions for you, but in some situations you may have to call them yourself. Therefore it is really good to know how they work.

The `$scope.$apply()` function is used to execute some code, and then call `$scope.$digest()` after that, so all watches are checked and the corresponding watch listener functions are called. The `$apply()` function is useful when integrating AngularJS with other code.

I will get into more detail about the `$watch()`, `$digest()` and `$apply()` functions in the remainder of this text.

## $watch()

The `$scope.watch()` function creates a watch of some variable. When you register a watch you pass two functions as parameters to the `$watch()` function:

- A value function

- A listener function

Here is an example:

```
$scope.$watch(function() {},
              function() {}
             );
```

The first function is the value function and the second function is the listener function.

The value function should return the value which is being watched. AngularJS can then check the value returned against the value the watch function returned the last time. That way AngularJS can determine if the value has changed. Here is an example:

```
$scope.$watch(function(scope) { return scope.data.myVar },
              function() {}
             );
```

This example valule function returns the `$scope` variable `scope.data.myVar`. If the value of this variable changes, a different value will be returned, and AngularJS will call the listener function.

Notice how the value function takes the `scope` as parameter (without the `$` in the name). Via this parameter the value function can access the `$scope` and its variables. The value function can also watch global variables instead if you need that, but most often you will watch a `$scope` variable.

The listener function should do whatever it needs to do if the value has changed. Perhaps you need to change the content of another variable, or set the content of an HTML element or something. Here is an example:

```
$scope.$watch(function(scope) { return scope.data.myVar },
              function(newValue, oldValue) {
                  document.getElementById("").innerHTML =
                      "" + newValue + "";
              }
             );
```

This example sets the inner HTML of an HTML element to the new value of the variable, embedded in the`b` element which makes the value bold. Of course you could have done this using the code **`{{ data.myVar }}`**, but this is just an example of what you can do inside the listener function.

# $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 calls the value function for each watch. If the value returned by the value function is

different than the value it returned the last time it was called, the listener function for that watch is called.

The `$digest()` function is called whenever AngularJS thinks it is necessary. For instance, after a button click handler has been executed, or after an AJAX call returns (after the `done()` / `fail()` callback function has been executed).

You may encounter some corner cases where AngularJS does not call the `$digest()` function for you. You will usually detect that by noticing that the data bindings do not upate the displayed values. In that case, call `$scope.$digest()` and it should work. Or, you can perhaps use `$scope.$apply()` instead which I will explain in the next section.

## $apply()

The `$scope.$apply()` function takes a function as parameter which is executed, and after that`$scope.$digest()` is called internally. That makes it easier for you to make sure that all watches are checked, and thus all data bindings refreshed. Here is an `$apply()` example:

```
$scope.$apply(function() {
    $scope.data.myVar = "Another value";
});
```

The function passed to the `$apply()` function as parameter will change the value of `$scope.data.myVar`. When the function exits AngularJS will call the `$scope.$digest()` function so all watches are checked for changes in the watched values.

## Example

To illustrate how `$watch()`, `$digest()` and `$apply()` works, look at this example:

```
<div ng-controller="myController">
    {{data.time}}

    <br/>
    <button ng-click="updateTime()">update time - ng-click</button>
    <button id="updateTimeButton"  >update time</button>
</div>


<script>
    var module       = angular.module("myapp", []);
    var myController1 = module.controller("myController", function($scope) {

        $scope.data = { time : new Date() };

        $scope.updateTime = function() {
            $scope.data.time = new Date();
        }
```

```
        document.getElementById("updateTimeButton")
                .addEventListener('click', function() {
            console.log("update time clicked");
            $scope.data.time = new Date();
        });
    });
</script>
```

This example binds the `$scope.data.time` variable to an interpolation directive which merges the variable value into the HTML page. This binding creates a watch internally on the `$scope.data.time` variable.

The example also contains two buttons. The first button has an `ng-click` listener attached to it. When that button is clicked the `$scope.updateTime()` function is called, and after that AngularJS calls `$scope.$digest()` so that data bindings are updated.

The second button gets a standard JavaScript event listener attached to it from inside the controller function. When the second button is clicked that listener function is executed. As you can see, the listener functions for both buttons do almost the same, but when the second button's listener function is called, the data binding is not updated. That is because the `$scope.$digest()` is not called after the second button's event listener is executed. Thus, if you click the second button the time is updated in the `$scope.data.time`variable, but the new time is never displayed.

To fix that we can add a `$scope.$digest()` call to the last line of the button event listener, like this:

```
document.getElementById("updateTimeButton")
        .addEventListener('click', function() {
    console.log("update time clicked");
    $scope.data.time = new Date();
    $scope.$digest();
});
```

Instead of calling `$digest()` inside the button listener function you could also have used the `$apply()`function like this:

```
document.getElementById("updateTimeButton")
        .addEventListener('click', function() {
    $scope.$apply(function() {
        console.log("update time clicked");
        $scope.data.time = new Date();
    });
});
```

Notice how the `$scope.$apply()` function is called from inside the button event listener, and how the update of the `$scope.data.time` variable is performed inside the function passed as parameter to the `$apply()` function. When the `$apply()` function call finishes AngularJS calls `$digest()` internally, so all data bindings are updated.