$scope.$apply() vs $scope.$digest()

`$apply()` and `$digest()` are two core, and sometimes confusing, aspects of AngularJS. To understand how AngularJS works one needs to fully understand how `$apply()` and `$digest()`work. This article aims to explain what `$apply()` and `$digest()` really are, and how they can be useful in your day-to-day AngularJS programming.

`$apply` and `$digest` Explored

AngularJS offers an incredibly awesome feature known as two way data binding which greatly simplifies our lives. Data binding means that when you change something in the view, the `scope`model *automagically* updates. Similarly, whenever the `scope` model changes, the view updates itself with the new value. How does does AngularJS do that? When you write an expression (`{{aModel}}`), behind the scenes Angular sets up a watcher on the `scope` model, which in turn updates the view whenever the model changes. This `watcher` is just like any watcher you set up in AngularJS:

```
1  $scope.$watch('aModel', function(newValue, oldValue) {
2    //update the DOM with newValue
3  });
```

The second argument passed to `$watch()` is known as a listener function, and is called whenever the value of `aModel` changes. It is easy for us to grasp that when the value of `aModel` changes this listener is called, updating the expression in HTML. But, there is still one big question! How does Angular figure out when to call this listener function? In other words, how does AngularJS know when`aModel` changes so it can call the corresponding listener? Does it run a function periodically to check whether the value of the `scope` model has changed? Well, this is where the `$digest` cycle steps in.

It's the `$digest` cycle where the watchers are fired. When a watcher is fired, AngularJS evaluates the `scope` model, and if it has changed then the corresponding listener function is called. So, our next question is when and how this `$digest` cycle starts.

The `$digest` cycle starts as a result of a call to `$scope.$digest()`. Assume that you change a`scope` model in a handler function through the `ng-click` directive. In that case AngularJS automatically triggers a `$digest` cycle by calling `$digest()`. When the `$digest` cycle starts, it fires each of the watchers. These watchers check if the current value of the `scope` model is different from last calculated value. If yes, then the corresponding listener function executes. As a result if you have any expressions in the view they will be updated. In addition to `ng-click`, there are several other built-in directives/services that let you change models (e.g. `ng-model`, `$timeout`, etc) and automatically trigger a `$digest` cycle.

So far, so good! But, there is a small gotcha. In the above cases, Angular doesn't directly call`$digest()`. Instead, it calls `$scope.$apply()`, which in turn calls `$rootScope.$digest()`. As a result of this, a digest cycle starts at the `$rootScope`, and subsequently visits all the child scopes calling the watchers along the way.

Now, let's assume you attach an `ng-click` directive to a button and pass a function name to it. When the button is clicked, AngularJS wraps the function call within `$scope.$apply()`. So, your function executes as usual, change models (if any), and a `$digest` cycle starts to ensure your changes are reflected in the view.

Note: `$scope.$apply()` automatically calls `$rootScope.$digest()`. The `$apply()` function comes in two flavors. The first one takes a function as an argument, evaluates it, and triggers a`$digest` cycle. The second version does not take any arguments and just starts a `$digest` cycle when called. We will see why the former one is the preferred approach shortly.

When Do You Call `$apply()` Manually?

If AngularJS usually wraps our code in `$apply()` and starts a `$digest` cycle, then when do you need to do call `$apply()` manually? Actually, AngularJS makes one thing pretty clear. It will account for only those model changes which are done inside AngularJS' context (i.e. the code that changes models is wrapped inside `$apply()`). Angular's built-in directives already do this so that any model changes you make are reflected in the view. However, if you change any model outside of the Angular context, then you need to inform Angular of the changes by calling `$apply()` manually. It's like telling Angular that you are changing some models and it should fire the `watchers` so that your changes propagate properly.

For example, if you use JavaScript's `setTimeout()` function to update a `scope` model, Angular has no way of knowing what you might change. In this case it's your responsibility to call `$apply()`manually, which triggers a `$digest` cycle. Similarly, if you have a directive that sets up a DOM event listener and changes some models inside the handler function, you need to call `$apply()` to ensure the changes take effect.

Let's look at an example. Suppose you have a page, and once the page loads you want to display a message after a two second delay. Your implementation might look something like the JavaScript and HTML shown in the following listing.

By running the example, you will see that the delayed function runs after a two second interval, and updates the `scope` model `message`. Still, the view doesn't update. The reason, as you may have guessed, is that we forgot to call `$apply()` manually. Therefore, we need to update our`getMessage()` function as shown below.

If you run this updated example, you can see the view update after two seconds. The only change is that we wrapped our code inside `$scope.$apply()` which automatically triggers`$rootScope.$digest()`. As a result the watchers are fired as usual and the view updates.

Note: By the way, you should use `$timeout` service whenever possible which is `setTimeout()` with automatic `$apply()` so that you don't have to call `$apply()` manually.

Also, note that in the above code you could have done the model changes as usual and placed a call to `$apply()` (the no-arg version) in the end. Have a look at the following snippet:

```
$scope.getMessage = function() {
  setTimeout(function() {
    $scope.message = 'Fetched after two seconds';
    console.log('message:'+ $scope.message);
    $scope.$apply(); //this triggers a $digest
  }, 2000);
};
```

The above code uses the no-arg version of `$apply()` and works. Keep in mind that you should always use the version of `$apply()` that accepts a function argument. This is because when you pass a function to `$apply()`, the function call is wrapped inside a `try...catch` block, and any exceptions that occur will be passed to the `$exceptionHandler` service.

How Many Times Does the `$digest` Loop Run?

When a `$digest` cycle runs, the watchers are executed to see if the `scope` models have changed. If they have, then the corresponding listener functions are called. This leads to an important question. What if a listener function itself changed a `scope` model? How would AngularJS account for that change?

The answer is that the `$digest` loop doesn't run just once. At the end of the current loop, it starts all over again to check if any of the models have changed. This is basically dirty checking, and is done to account for any model changes that might have been done by listener functions. So, the `$digest`cycle keeps looping until there are no more model changes, or it hits the max loop count of 10. It's always good to stay idempotent and try to minimize model changes inside the listener

functions.

Note: At a minimum, `$digest` will run twice even if your listener functions don't change any models. As discussed above, it runs once more to make sure the models are stable and there are no changes.

Conclusion

I hope this article has clarified what `$apply` and `$digest` are all about. The most important thing to keep in mind is whether or not Angular **can** detect your changes. If it cannot, then you must call`$apply()` manually.