Angular Expressions

Angular expressions are JavaScript-like code snippets that are usually placed in bindings such as `{{ expression }}`.

For example, these are valid expressions in Angular:

- `1+2`
- `a+b`
- `user.name`
- `items[index]`

Angular Expressions vs. JavaScript Expressions

Angular expressions are like JavaScript expressions with the following differences:

- **Context:** JavaScript expressions are evaluated against the global `window`. In Angular, expressions are evaluated against a `scope` (api/ng/type/$rootScope.Scope) object.
- **Forgiving:** In JavaScript, trying to evaluate undefined properties generates `ReferenceError` or `TypeError`. In Angular, expression evaluation is forgiving to `undefined` and `null`.
- **No Control Flow Statements:** you cannot use the following in an Angular expression: conditionals, loops, or exceptions.
- **Filters:** You can use filters (guide/filter) within expressions to format data before displaying it.

If you want to run more complex JavaScript code, you should make it a controller method and call the method from your view. If you want to `eval()` an Angular expression yourself, use the `$eval()` (api/ng/type/$rootScope.Scope#$eval) method.

Example

```
<span>
  1+2={{1+2}}
</span>
```

You can try evaluating different expressions [in this example online (https://docs.angularjs.org/guide/expression)]:

```
<!-- index.html  -->
<div ng-controller="ExampleController" class="expressions">
  Expression:
  <input type='text' ng-model="expr" size="80"/>
  <button ng-click="addExp(expr)">Evaluate</button>
  <ul>
   <li ng-repeat="expr in exprs track by $index">
     [ <a href="" ng-click="removeExp($index)">X</a> ]
     <tt>{{expr}}</tt> => <span ng-bind="$parent.$eval(expr)"></span>
    </li>
  </ul>
</div>
```

```
/* script.js */
angular.module('expressionExample', [])
  .controller('ExampleController', ['$scope', function($scope) {
    var exprs = $scope.exprs = [];
    $scope.expr = '3*10|currency';
    $scope.addExp = function(expr) {
      exprs.push(expr);
    };

    $scope.removeExp = function(index) {
      exprs.splice(index, 1);
    };
  }]);
```

Context

Angular does not use JavaScript's `eval()` to evaluate expressions. Instead Angular's $parse (api/ng/service/$parse) service processes these expressions.

Angular expressions do not have access to global variables like `window`, `document` or `location`. This restriction is intentional. It prevents accidental access to the global state – a common source of subtle bugs.

Instead use services like `$window` and `$location` in functions called from expressions. Such services provide mockable access to globals.

```
<!-- index.html  -->
<div class="example2" ng-controller="ExampleController">
  Name: <input ng-model="name" type="text"/>
  <button ng-click="greet()">Greet</button>
  <button ng-click="window.alert('Should not see me')">Won't greet</button>
</div>
```

```
/* script.js */
angular.module('expressionExample', [])
  .controller('ExampleController', ['$window', '$scope', function($window, $scope) {
    $scope.name = 'World';

    $scope.greet = function() {
      $window.alert('Hello ' + $scope.name);
    };
  }]);
```

Forgiving

Expression evaluation is forgiving to undefined and null. In JavaScript, evaluating `a.b.c` throws an exception if `a` is not an object. While this makes sense for a general purpose language, the expression evaluations are primarily used for data binding, which often look like this:

```
{{a.b.c}}
```

It makes more sense to show nothing than to throw an exception if `a` is undefined (perhaps we are waiting for the server response, and it will become defined soon). If expression evaluation wasn't forgiving we'd have to write bindings that clutter the code, for example: `{{(a||{}).b||{}).c}}`

Similarly, invoking a function `a.b.c()` on `undefined` or `null` simply returns `undefined`.

No Control Flow Statements

Apart from the ternary operator (`a ? b : c`), you cannot write a control flow statement in an expression. The reason behind this is core to the Angular philosophy that application logic should be in controllers, not the views. If you need a real conditional, loop, or to throw from a view expression, delegate to a JavaScript method instead.

$event

Directives like `ngClick` (api/ng/directive/ngClick) and `ngFocus` (api/ng/directive/ngFocus) expose a `$event` object within the scope of that expression. The object is an instance of a jQuery Event Object (http://api.jquery.com/category/events/event-object/) when jQuery is present or a similar jqLite object.

```
<!-- index.html -->
<div ng-controller="EventController">
  <button ng-click="clickMe($event)">Event</button>
  <p><code>$event</code>: <pre> {{$event | json}}</pre></p>
  <p><code>clickEvent</code>: <pre>{{clickEvent | json}}</pre></p>
</div>
```

```
/* script.js */
angular.module('eventExampleApp', []).
  controller('EventController', ['$scope', function($scope) {
    /*
     * expose the event object to the scope
     */
    $scope.clickMe = function(clickEvent) {
      $scope.clickEvent = simpleKeys(clickEvent);
      console.log(clickEvent);
    };

    /*
     * return a copy of an object with only non-object keys
     * we need this to avoid circular references
     */
    function simpleKeys (original) {
      return Object.keys(original).reduce(function (obj, key) {
        obj[key] = typeof original[key] === 'object' ? '{ ... }' : original[key];
        return obj;
      }, {});
    }
  }]);
```

Note in the example above how we can pass in `$event` to `clickMe`, but how it does not show up in `{{$event}}`. This is because `$event` is outside the scope of that binding.

One-time binding

An expression that starts with `::` is considered a one-time expression. One-time expressions will stop recalculating once they are stable, which happens after the first digest if the expression result is a non-undefined value (see value stabilization algorithm below).

```
<!-- index.html  -->
<div ng-controller="EventController">
  <button ng-click="clickMe($event)">Click Me</button>
  <p id="one-time-binding-example">One time binding: {{::name}}</p>
  <p id="normal-binding-example">Normal binding: {{name}}</p>
</div>
```

```
/* script.js */
angular.module('oneTimeBidingExampleApp', []).
  controller('EventController', ['$scope', function($scope) {
    var counter = 0;
    var names = ['Igor', 'Misko', 'Chirayu', 'Lucas'];
    /*
     * expose the event object to the scope
     */
    $scope.clickMe = function(clickEvent) {
      $scope.name = names[counter % names.length];
      counter++;
    };
  }]);
```

Why this feature

The main purpose of one-time binding expression is to provide a way to create a binding that gets deregistered and frees up resources once the binding is stabilized. Reducing the number of expressions being watched makes the digest loop faster and allows more information to be displayed at the same time.

Value stabilization algorithm

One-time binding expressions will retain the value of the expression at the end of the digest cycle as long as that value is not undefined. If the value of the expression is set within the digest loop and later, within the same digest loop, it is set to undefined, then the expression is not fulfilled and will remain watched.

1. Given an expression that starts with `::`, when a digest loop is entered and expression is dirty-checked, store the value as V
2. If V is not undefined, mark the result of the expression as stable and schedule a task to deregister the watch for this expression when we exit the digest loop
3. Process the digest loop as normal
4. When digest loop is done and all the values have settled process the queue of watch deregistration tasks. For each watch to be deregistered check if it still evaluates to value that is not `undefined`. If that's the case, deregister the watch. Otherwise keep dirty-checking the watch in the future digest loops by following the same algorithm starting from step 1

How to benefit from one-time binding

When interpolating text or attributes. If the expression, once set, will not change then it is a candidate for one-time expression.

```
<div name="attr: {{::color}}">text: {{::name}}</div>
```

When using a directive with bidirectional binding and the parameters will not change

```
someModule.directive('someDirective', function() {
return {
  scope: {
    name: '=',
    color: '@'
  },
  template: '{{name}}: {{color}}'
};
});
```

```
<div some-directive name="::myName" color="My color is {{::myColor}}"></div>
```

When using a directive that takes an expression

```
<ul>
<li ng-repeat="item in ::items">{{item.name}};</li>
</ul>
```