
alight Documentation

Release 0.13.0

Oleg Nechaev

December 03, 2016

1	API	3
2	alight.bootstrap	5
3	Events	7
4	Controllers	11
5	Scope	13
6	Change Detector	17
7	How does it work?	21
8	Create Directives	23
9	Inheritance of directives	25
10	Directives	27
11	Base filters	29
12	Create filters	31
13	Async filters	33
14	Text bindings	37
15	Text directives	39
16	One-time binding	41
17	Isolated Angular Light	43
18	Namespaces	45
19	A few ways to bind a model to the DOM	47
20	Directive preprocessor	49
21	Component	51

22	Migration from 0.10 to 0.12	53
23	FAQ	55

Contents:

- Alight in 7 steps
- How to bind HTML
- How to bind events
- How to bind attributes
- How to attach templates
- [Collection of examples](#)

Contents 2:

1.1 alight.debug

Turn on a logging debug information

- `alight.debug.scan` - logging scan operations
- `alight.debug.directive` - logging binding directives
- `alight.debug.parser` - logging parsing expressions
- `alight.debug.watch` - logging function `$watch`
- `alight.debug.watchText` - logging function `$watchText`

1.2 alight.bootstrap(option)

It lets you a few ways to bind DOM, read more

1.3 alight.bind(scope, element, option)

Bind a change detector to the DOM element, alias **`alight.applyBindings`**

- `scope` - an instance of `Scope` (or `ChangeDetector`)
- `element` - DOM element
- `option.skip_top = false` - Skip binding the top DOM element
- `option.skip_attr = ['al-repeat', 'al-app']` - Skip attributes for a binding, for the top element
- `option.attachDirective = {}` - Attach a custom directive

```
var scope = alight.Scope();  
var element = document.body;  
alight.bind(scope, element);
```

1.4 alight.Scope()

Create a new scope

1.5 **alight.ChangeDetector([scope])**

Create a new change detector, read more

1.6 **alight.ctrl**

Dictionary of controllers, alias for **alight.controllers**

1.7 **alight.filters**

Dictionary of filters

1.8 **alight.directives**

Dictionary of directives, short alias **alight.d**

1.9 **alight.text**

Collection of text directives

1.10 **alight.hook**

Different hooks

1.11 **alight.nextTick(callback)**

Execute the function on next tick

alight.bootstrap

A few ways to bind DOM.

2.1 Binding to all elements with attribute [al-app]

- `alight.bootstrap()`

2.2 Binding by selector

- `alight.bootstrap('#someId')`
- `alight.bootstrap('#someId', scope)`

2.3 Binding to element

- `alight.bootstrap(element)`
- `alight.bootstrap(element, scope)`

2.4 Binding to array of elements

- `alight.bootstrap([element1, element2, element3])`
- `alight.bootstrap([element1, element2, element3], scope)`

Note: If you provide **scope**, then all elements will be binded to this scope, otherwise every dom will have own scope.

2.5 Example

Listing 2.1: Example of bootstrap

```
alight.bootstrap('#app', {  
  name: 'world',  
  click: function() {  
    this.name = 'user'  
  }  
});
```

Events

A way to bind events.

Listing 3.1: Syntax

```
<div al-on.eventname.modifier="expression"> </div>  
<div @eventname.modifier="expression"> </div>
```

3.1 Modifiers for keydown, keypress, keyup

- Filter by extra key: alt, ctrl (control), meta, shift
- Filter by keycode: enter, tab, delete, backspace, esc, space, up, down, left, right, <any number of key code>

3.2 Special modifiers for all events

- stop - calls stopPropagation
- prevent - calls preventDefault
- nostop - voids stopPropagation for click, dblclick, submit
- noprevent - voids preventDefault for click, dblclick, submit
- noscan - voids \$scan
- throttle-<number>
- debounce-<number>

Modifiers “prevent” and “stop” are on by default for click, dblclick and submit. Available arguments in expressions: **\$event**, **\$element**, **\$value**

3.3 Examples

- al-on.keyup=”onKeyUp(\$event)”
- al-on.keydown.tab=”onTab()”
- @keyup=”onKeyUp(\$event)”

- `@keyup="key=$event.keyCode"`
- `@keydown.tab="onTab()"`
- `@keydown.enter="onEnter($event)"`
- `@keydown.13="onEnter($event)"`
- `@keydown.13.prevent="onEnter($event)"`
- `@keydown.control.enter="onEnter($event)"`
- `@keydown.control.shift.enter="onEnter($event)"`
- `@input="value=$event.target.value"`
- `@submit="onSubmit()"`
- `@submit.noprevent="onSubmit()"`
- `@click="onClick($event)"`
- `@click.noprevent="onClick()"`
- `@click.noprevent.stop="onClick()"`
- `@mousemove.noscan="onMousemove($event)"`
- `@mousemove.throttle-300="onMousemove($event)"`
- `@mousemove.debounce-300="onMousemove($event)"`

3.4 You can make aliases and filters for events

It lets you:

- bind a few events to one expression
- makes custom modifiers (filters)
- make aliases for events

Listing 3.2: A few ways to make aliases

```
align.hooks.eventModifier['enter'] = 'keydown blur';
align.hooks.eventModifier['enter'] = ['keydown', 'blur'];
align.hooks.eventModifier['enter'] = (event, env) => {}
align.hooks.eventModifier['enter'] = {
  event: 'keydown blur',           // can be omitted
  fn: (event, env) => {}           // can be omitted
  init: (scope, element) => {}    // can be omitted
}
align.hooks.eventModifier.enter = {
  event: ['keydown', 'blur'],
  fn: (event, env) => {
    env.stop = true; // stop the event
  }
}
```

3.5 Examples

- Throttle and Debounce for event-input and mousemove
- Custom modifier/alias

Controllers

4.1 Overview

- You can invoke a controller using directive `al-ctrl="controllerName"`, also you can declare a top controller in directive `al-app="controllerName"`.
- Your controller should be placed in `alight.ctrl.controllerName` or in `scope.$ns.ctrl` for a private controller or in global window.
- Your controller can be a simple function or a class.
- If your controller is a class (has a prototype), then Scope's methods (`$scope`, `$watch`, `$getValue` etc) are available in "this".

4.2 Controller as a function

Listing 4.1: Example controller as function

```
<div al-app="main">
  {{myVar}}
</div>
```

Listing 4.2: Example controller as function

```
alight.ctrl.main = function(scope) {
  scope.myVar = 123;
}
```

Listing 4.3: Example controller as function

```
function main(scope) {
  scope.myVar = 123;
}
```

[Example on jsfiddle](#)

4.3 Controller as a class

Listing 4.4: Example controller as a class

```
<div al-app="main">
  <button al-click="click()">Click</button>
  {{name}}
</div>
```

Listing 4.5: Example controller as a class

```
function main() {
  this.name = 'linux';
}
main.prototype.click = function() {
  this.name += '!'
}
```

[Example on jsfiddle](#)

Listing 4.6: Controller in TypeScript

```
class main {
  constructor() {
    this.name = 'linux';
  }

  click(value) {
    this.name += '!'
  }
}
```

Listing 4.7: Controller in CoffeeScript

```
class main
  constructor: () ->
    @.name = 'linux'

  click: () ->
    @.name += '!'
```

Scope

It lets you observe changes in your scope, it uses change detector.

5.1 Scope.\$watch(name, callback, option)

Set the tracking variable. Also you can track system events, it returns object with method stop()

Name:

- **<expression>** - Expression/model
- **<a function>** - Track the result of the function, the function are called every iteration of \$scan.
- **“\$destroy”** - Track a destroying scope
- **“\$any”** - Track a modifying any object
- **“\$finishScan”** - a callback is executed as soon as \$scan finish work
- **“\$finishBinding”** - the callback is called when a binding process finishes, [sample](#)
- **“\$finishScanOnce”**
- **“\$onScanOnce”** - the callback is called in scan loop

Callback:

The callback is called with two parameters. The first parameter contains the object after it has been changed, the second parameter contains the object before it has been changed. If the watched variable is not initialized, the second parameter is undefined.

Option:

- **option** = true or **option.isArray** = true - watch an array
- **option.readOnly** = true - You can use it if the *callback* doesn't modify the scope. (an optimization option).
- **option.deep** = true | integer - a deep comparison for the object, watches 10 hierarchy depth levels by default, as alternative an integer which contains the depth of levels to watch.
- **option.isArray**
- **option.OneTime**
- **option.onStop**

Optimization tip: If *callback* returns ‘\$scanNoChanges’ then \$scan will not run extra-loop (like readonly watch)

5.2 Scope.\$watchGroup(expression, callback)

Watches a group of expressions. There are two approaches which can be used:

1. It returns a group object. You need to manually add more watcher objects to the group ([Example](#)).
2. Use an array as expression and watch it's items ([Example](#)).

expression:

- array of <expressions> - items which will be watched

callback

- will be executed when any item of the watched group has been changed

5.3 Scope.\$compile(expression, option)

Compile an expression

option:

- **option.input** - list of input arguments
- **option.no_return** - a function will not return any result (compile a statment)
- **option.string** - result of method will convert to string

Listing 5.1: Example of \$compile

```
var scope = alight.Scope();
var fn = scope.$compile('"hello " + title')
scope.title = 'linux'
fn(scope) // return "hello linux"
scope.title = 'macos'
fn(scope) // return "hello macos"
```

Listing 5.2: Example with input

```
var fn = scope.$compile('title + v', { input:['v'] })
fn(scope, ' X') // return "macos X"
```

Listing 5.3: Example with no_return

```
var fn = scope.$compile('title = v', { input:['v'], no_return:true })
fn(scope, 'linux') // scope.title = "linux"
```

5.4 Scope.\$eval(expression)

Execute an expression

5.5 Scope.\$watchText(tpl, callback)

Track the template

5.6 Scope.\$destroy()

Destroy the Scope.

5.7 Scope.\$scan(callback or option)

Starts the search for changes, returns a watch statistic

- **callback** - Method will be called when \$scan finishes a work, even if \$scan has already started from other a place.
- **option.callback** - see above
- **option.skipWatch** - skip specific watch
- **option.late** = (*true/false*) - If there is a few \$scan commands, Angular Light will call only last one.

Listing 5.4: Example with \$scan

```
var scope = align.Scope();
scope.$watch('title', function(value) {
  console.log('title =', value)
}); // make observing
scope.title = 'new'
scope.$scan()
// print title = new
scope.title = 'linux'
scope.$scan()
// print title = linux
scope.$scan()
// do nothing
```

5.8 Scope.\$getValue(name)

Take the value of the variable, also you can use Scope.\$eval

5.9 Scope.\$setValue(name, value)

Set the value of the variable

Listing 5.5: Example with \$setValue

```
var scope = {}
scope.var = 1;
scope.path.var = 2;
```

```
scope.path[scope.key] = 3;

// equal
var scope = alight.Scope();

scope.setValue('var', 1);
scope.setValue('path.var', 2);
scope.setValue('path[key]', 3);
```

Change Detector

It lets you observe changes in your scope

6.1 ChangeDetector.watch(name, callback, option)

Set the tracking variable. Also you can track system events, it returns object with method stop()

Name:

- **<expression>** - Expression/model
- **<a function>** - Track the result of the function, the function are called every iteration of \$scan.
- **“\$destroy”** - Track a destroying scope
- **“\$any”** - Track a modifying any object
- **“\$finishScan”** - a callback is executed as soon as \$scan finish work
- **“\$finishBinding”** - the callback is called when a binding process finishes, [sample](#)
- **“\$finishScanOnce”**
- **“\$onScanOnce”** - the callback is called in scan loop

Callback:

The callback is called with two parameters. The first parameter contains the object after it has been changed, the second parameter contains the object before it has been changed. If the watched variable is not initialized, the second parameter is undefined.

Option:

- **option** = true or **option.isArray** = true - watch an array
- **option.readOnly** = true - You can use it if the *callback* doesn't modify the scope. (an optimization option).
- **option.deep** = true | integer - a deep comparison for the object, watches 10 hierarchy depth levels by default, as alternative an integer which contains the depth of levels to watch.
- **option.isArray**
- **option.OneTime**
- **option.onStop**

Optimization tip: If *callback* returns ‘\$scanNoChanges’ then \$scan will not run extra-loop (like readonly watch)

6.2 ChangeDetector.watchGroup(expression, callback)

Watches a group of expressions. There are two approaches which can be used:

1. It returns a group object. You need to manually add more watcher objects to the group ([Example](#)).
2. Use an array as expression and watch it's items ([Example](#)).

expression:

- array of <expressions> - items which will be watched

callback

- will be executed when any item of the watched group has been changed

6.3 ChangeDetector.compile(expression, option)

Compile an expression

option:

- **option.input** - list of input arguments
- **option.no_return** - a function will not return any result (compile a statment)
- **option.string** - result of method will convert to string

Listing 6.1: Example of \$compile

```
var scope = {};  
var cd = alight.ChangeDetector(scope)  
var fn = cd.compile('"hello " + title')  
scope.title = 'linux'  
fn(scope) // return "hello linux"  
scope.title = 'macos'  
fn(scope) // return "hello macos"
```

Listing 6.2: Example with input

```
var fn = cd.compile('title + v', { input:['v'] })  
fn(scope, ' X') // return "macos X"
```

Listing 6.3: Example with no_return

```
var fn = cd.compile('title = v', { input:['v'], no_return:true })  
fn(scope, 'linux') // scope.title = "linux"
```

6.4 ChangeDetector.eval(expression)

Execute an expression

6.5 ChangeDetector.watchText(tpl, callback)

Track the template

6.6 ChangeDetector.new([scope])

Create a child ChangeDetector, if scope is omitted, then it will used parent scope

6.7 ChangeDetector.destroy()

Destroy the Scope.

6.8 ChangeDetector.scan(callback or option)

Starts the search for changes, returns a watch statistic

- **callback** - Method will be called when \$scan finishes a work, even if \$scan has already started from other a place.
- **option.callback** - see above
- **option.skipWatch** - skip specific watch
- **option.late** = (*true/false*) - If there is a few \$scan commands, Angular Light will call only last one.

Listing 6.4: Example with \$scan

```
var scope = {};
var cd = alight.ChangeDetector(scope);
cd.watch('title', function(value) {
    console.log('title =', value)
}); // make observing
scope.title = 'new'
cd.scan()
// print title = new
scope.title = 'linux'
cd.scan()
// print title = linux
cd.scan()
// do nothing
```

6.9 ChangeDetector.getValue(name)

Take the value of the variable, also you can use ChangeDetector.eval

6.10 ChangeDetector.setValue(name, value)

Set the value of the variable

Listing 6.5: Example with \$setValue

```
var scope = {}
scope.var = 1;
scope.path.var = 2;
scope.path[scope.key] = 3;

// equal
var scope = {}
var cd = alight.ChangeDetector(scope);

cd.setValue('var', 1);
cd.setValue('path.var', 2);
cd.setValue('path[key]', 3);
```

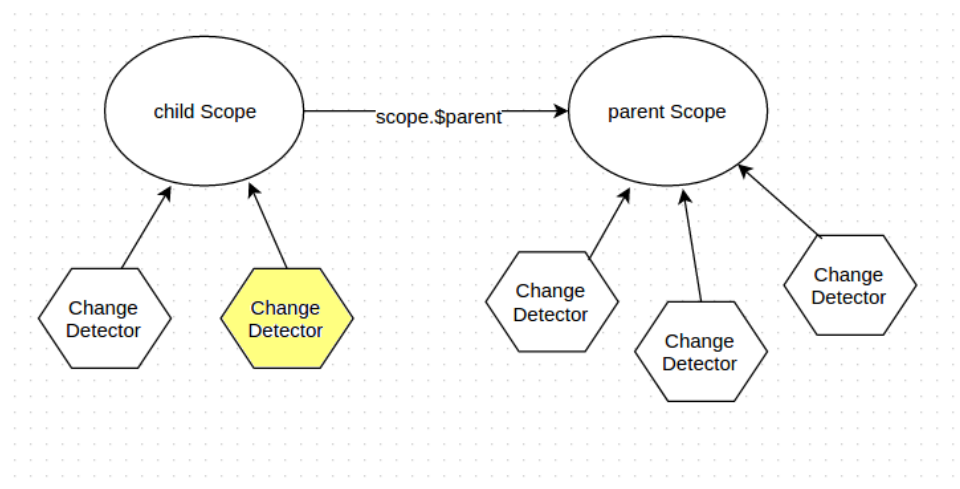

How does it work?

- Scope is a object with user's data which can have a lot of information, it doesn't have own functional, `$scope.$watch` is a just a link to `activeChangeDetector.watch`
- `ChangeDetector` - is a dirty-checking tool which implement "watch", "scan" etc.
- one `ChangeDetector` instance observes only one object (e.g. scope)
- Different directives makes own `ChangeDetectors` and observe your data (your scope), so a few (10, 20) change detectors can observe the same scope. e.g. `ng-repeat` observes every item of array, `ng-if` and `ng-include` make own CD for child template with the same scope.

You can't call `$watch` or `$new` anytime. When you call `scope.$parent.$watch`, your parent can have a few CD, and your parent doesn't know which CD should serves your watch-expression. It's why you should take needed CD and call `CD.watch`.

On the other side you can call `scope.$watch` (inside link function) because in this moment one of the CD is active, so `scope.$watch` calls `activeCD.watch`

Any other scope's methods doesn't have this problem and they can be called any time, (`$eval`, `$getValue`, `$setValue` etc)



Create Directives

Directives should be placed in **alight.directives.prefix**, you can choose any prefix, for example al-text (*alight.directives.al.text*), bo-text (*alight.directives.bo.text*), also you can place your directives in *scope.\$ns.directives* they will be private.

A hyphen should be changed to underscores, example: `<input al-my-some-directive />` it will be in `alight.directives.al.mySomeDirective`

The prefixes are needed in order to be able to catch the lack of used directives. For example, if you use the directive “al-texta”, but it does not exists (a mistake in the name or js-file isn’t loaded), then aLight will throw an exception.

An example of directive **al-text**, the directive is called when the binding process comes to an DOM-element

Listing 8.1: Example of directive al-text

```
alight.directives.al.text = function(scope, element, expression, env) {
  // Track the given expression
  scope.$watch(expression, function(value) {
    // set a text to the DOM-element
    $(element).text(value)
  });
};
```

Input arguments:

- **scope** - current Scope
- **cd** - change detector
- **element** - element of DOM
- **name** - value of attribute
- **env** - access to different options
- env. **attrName** - name of attribute (directive)
- env. **attributes** - list of attributes
- env. **takeAttr(name, skip=true)** - take a value of the attribute, if skip=true then the attribute will skip a binding process, sample
- env. **skippedAttr()** - list of not active attributes
- env. **stopBinding** = false - stop binding for child elements
- env. **changeDetector** - access to ChangeDetector
- env. **parentChangeDetector** - access to parent ChangeDetector (if scope was changed)

8.1 Attributes of directive:

- **priority** - you can set priority for a directive
- **template** - custom template
- **templateUrl** - link to template
- **scope** (false/true) - if it is true, there will be a new, empty scope
- **ChangeDetector** (false/true/'root') - create a new change detector
- **restrict** = 'A', can be 'AEM', 'A' matches attribute name, 'E' matches element name, 'M' matches class name
- **link** - the method is called after template, scope
- **init** - the method is called when the directive is made, before template, scope. You usually need **link** instead of it.
- **stopBinding** (false/true) - stop binding for child elements
- **anyAttr** - you can make a custom attribute, look. directive preprocessor

Listing 8.2: Directives with attributes

```
alight.directives.al.stop = {  
  priority: -10,  
  template: '<b></b>',  
  stopBinding: true,  
  link: function(scope, element, name, env) {  
  }  
};
```

If “stopBinding” is true, then the process of binding will skip child DOM-elements, it is necessary for the directives which are themselves controlled subsidiary of DOM, such as al-repeat, al-controller, al-include, al-stop, etc.

When you try to bind the same element, you have to skip used directives, otherwise you will have a recursion problem. You can use option **skip_attr**: **env.skippedAttr()** for this, **env.skippedAttr()** gives you a list of used directives.

```
alight.directives.al.stop = {  
  stopBinding: true,  
  link: function(scope, element, name, env) {  
    alight.bind(scope, element, {  
      skip_attr: env.skippedAttr()  
    })  
  }  
};
```

Inheritance of directives

If you want make inheritance of a directive, you need call a parent directive after that you can replace methods of the directive. For example, **al-value** has a few methods:

- onDom - binding to DOM
- updateModel - updateing the model
- watchModel - \$watch model
- updateDom - updateting DOM
- initDom - set first value to DOM, updateDom(init_value)
- start - It's called when the directive is ready

Make a directive al-value with deferred updating of model:

Listing 9.1: Inherit al-value

```
alight.directives.al.valueDelay = function(scope, cd, element, value, env) {  
  // create a source directive  
  var dir = alight.directives.al.value(scope, cd, element, value, env);  
  
  // save the old method for update the model  
  var oldUpdate = dir.updateModel;  
  var timer = null;  
  
  // change the method  
  dir.updateModel = function() {  
    if(timer) clearTimeout(timer);  
    timer = setTimeout(function() {  
      timer = null;  
  
      // call the default method for update the model  
      oldUpdate();  
    }, 500);  
  }  
  return dir;  
}
```

9.1 Examples of inheritance

Examples below for v0.10, but in v0.11 it works similar * al-value -> al-value-delay * al-show -> al-show-slow * al-repeat, add “\$even”, “\$odd” into the directive * al-repeat, change input expression my-repeat=”list.foreach item” * al-repeat, change rendering

Directives

10.1 Events

Binding events to an element can be done using one of the following directives:

There is also a short variant for events:

- `al-click`, [todo sample](#)
- `al-dblclick`
- `al-submit`, [todo sample](#)
- `al-blur`, [Sample](#)
- `al-change`, [Sample](#)
- `al-focus`, [Sample](#)
- `al-keydown`, [Sample](#)
- `al-keypress`, [Sample](#)
- `al-keyup`, [Sample](#)
- `al-mousedown`, [Sample](#)
- `al-mouseenter`, [Sample](#)
- `al-mouseleave`, [Sample](#)
- `al-mousemove`, [Sample](#)
- `al-mouseover`, [Sample](#)
- `al-mouseup`, [Sample](#)

10.2 Controls

- `al-model` - two-way binding to user input elements ([Example](#)). This directive is a proxy for `al-checked`, `al-radio`, `al-value` and `al-select`.
- `al-checked`, [todo sample](#)
- `al-radio` [sample1](#) [sample2](#)
- `al-value`, [todo sample](#)

- al-select + al-option, [example](#)

10.3 Attributes

- al-disable
- al-enable
- al-focused, two-way bind for focus events. [Sample](#)
- al-readonly

10.4 Special directives

- al-app, init application with current element, examples
- al-ctrl
- al-cloak, hide current element until activate the application, examples
- al-class/al-css, [todo sample](#), [animated sample](#)
- **al-style** examples
- **al-show**, [sample with animation](#)
- al-hide, [sample with animation](#)
- al-html
- **al-if**, [sample with animation](#)
- al-ifnot, [sample with animation](#)
- **al-include**, loads a html block from the server, [sample with animation](#)
- al-init
- al-repeat
- al-src
- al-stop, stops a bind process for the element and his children.
- al-text, [example](#)
- al-attr

10.5 Bind-once

- bo-if
- bo-ifnot
- bo-src
- bo-switch
- bo-switchDefault
- bo-switchWhen

Base filters

11.1 date

- To convert the date to string
- Input argument: date

Listing 11.1: example

```
<div>{{when | date:yyyy-mm-dd }}</div>
```

11.2 filter

- To filter the list
- Input argument: variable

Example

11.3 slice

- To slice the list
- input arguments: numbers

Listing 11.2: example

```
<div al-repeat="it in list | slice:a"></div>  
<div al-repeat="it in list | slice:a,b"></div>
```

11.4 toArray

- converts an object to array (for al-repeat)
- input arguments: key, value

Listing 11.3: example

```
<div al-repeat="item in object | toArray:key,value track by key">
```

Example

11.5 orderBy

- sorts an array by key (for al-repeat)
- input arguments: key, reverse

Listing 11.4: example

```
<div al-repeat="item in array | orderBy:key,reverse">
```

Example

11.6 throttle

- makes delay for output, pattern *debounce*
- input arguments: delay

Listing 11.5: example

```
<input al-value="link" type="text" />  
<p>{{link | throttle:300 | loadFromServer}}</p>
```

Example

11.7 json

- display data in json style

Listing 11.6: example

```
{{data.value | json}}  
{{this | json}}
```

Create filters

12.1 Overview

A filter should be placed in **alight.filters**, or in `scope.$ns` for private filters.

Listing 12.1: Example filter

```
alight.filters.double = function(value, expression, scope) {  
  return value + expression + value  
}
```

[Example on jsfiddle](#)

12.2 Input arguments

- expression - an input expression
- scope - scope

The filter should return a handler/function, one instance of filter.

Async filters

13.1 Overview

Async filters let you transform data in async mode, a filter should be placed in **alight.filters** or in `scope.$ns` for private filters.

13.2 Input arguments

- expression - an input expression
- scope - scope
- env - extra functional
- `env.setValue(value)` - set value of filter

Listing 13.1: Example of async filter

```
alight.filters.asyncFilter = function(expression, scope, env) {};  
alight.filters.asyncFilter.prototype = {  
  watchMode: 'deep',  
  onChange: function(value) {},  
  onStop: function() {}  
}
```

- watchMode, you can set 'simple'/'array'/'deep', if you need to change a watch mode for the input
- onChange - it's executed on every change of input
- onStop - it's executed when a watch object was removed

Listing 13.2: Example of async filter

```
alight.filters.trottle = function(delay, scope, env) {  
  delay = Number(delay);  
  var to;  
  this.onChange = function(value) {  
    if(to) clearTimeout(to);  
    to = setTimeout(function() {  
      to = null;  
      env.setValue(value);  
    }, delay);  
  };  
}
```

```

        scope.$scan();
    }, delay);
}
}
alight.filters.trottle.prototype.watchMode = 'simple';

```

Listing 13.3: Example 2 of async filter

```

alight.filters.trottle = function(delay, scope, env) {
    this.delay = Number(delay);
    this.scope = scope;
};

alight.filters.trottle.prototype.onChange = function(value) {
    var that = this;
    if(that.to) clearTimeout(that.to);

    that.to = setTimeout(function() {
        that.setValue(value);
        that.scope.$scan();
    }, that.delay);
}

```

Listing 13.4: Example in TypeScript

```

class Trottle {
    constructor(delay, scope, env) {
        this.delay = Number(delay);
        this.scope = scope;
    }

    onChange(value) {
        if(this.to) clearTimeout(this.to);

        this.to = setTimeout( () => {
            this.setValue(value);
            this.scope.$scan();
        }, this.delay);
    }
}

alight.filters.trottle = Trottle;

```

Listing 13.5: Example in CoffeeScript

```

class Trottle
  constructor: (@delay, @scope) ->

  onChange: (value) ->
    if this.to
      clearTimeout this.to

    this.to = setTimeout () =>
      this.setValue value
      this.scope.$scan()
    , this.delay

```

```
align.filters.trottle = Trottle
```

13.3 Examples

- Example throttle
- Example throttle, TypeScript
- Example throttle, CoffeeScript
- Example with loading from server

Text bindings

It's a way to draw information to DOM:

```
<div>Hello {{name}}!</div>
<a href="http://example.com/{{link}}">link</a>
```

Also you can use method “bind once” for optimization, for that you should append “=” to begin of expression.

```
<div>Hello {{=name}}!</div>
<a href="http://example.com/{{=link}}">link</a>
```

Also you can use one time binding

If you can't use tags {{ }}, you can change this to {# #}, {< >}, ## ## or something like this, length of tags should be equal 2 symbols:

```
alight.utils.pars_start_tag = '{#';
alight.utils.pars_finish_tag = '#'};
```

For complex text bindings you can use text directives

Text directives

15.1 Overview

An able to control a declarative data binding in the HTML

Listing 15.1: example how to use text directives

```
<div al-app>
  counter {{#counter}}
</div>
```

Listing 15.2: text directive counter

```
alight.text.counter = function(callback, expression, scope) {
  var n = 0;
  setInterval(function() {
    n++;
    callback(n)      // set result
    scope.$scan()    // $digest
  }, 1000);
}
```

15.2 Input arguments

- **callback** - a function to set a value
- **expression** - expression of directive
- **scope** - scope
- **env** - extra functional
- env. **finally** - a function to set the final value, after that \$watch will be removed.
- env. **setter** = callback
- env. **setterRaw** = send value directly

15.3 Examples

- Information output delay
- An counter
- An counter with setterRaw
- Sample with bindonce

One-time binding

16.1 Overview

Waits when an expression has a value (a non-undefined value), then stops watching. You need append "::" in front of expression for using One-Time binding. It works with \$watch, \$watchText, directives and declarative bindings.

Listing 16.1: example

```
<div class="red {{::class}}"> {{::text}} </div>
<div al-show="::visible"></div>
<li al-repeat="it in ::list">...</li>
and so on
```

16.2 Examples

- Sample with declarative bindings
- Sample with al-repeat

Isolated Angular Light

Angular Light can be invisible/inaccessible for alien code, you should make a function **alightInitCallback(alightBuilder)**, it has to be callable for Angular Light. Angular Light starts the function and gives “alightBuilder()” as an argument on load.

It’s useful feature, when your webapp will be used on alien page. It lets use different versions of Angular Light on the same page.

- [Example 1](#)
- [Example 2](#)

Namespaces

Every scope and child scopes of it can have own sets of directives, controllers and filters. You should make an object **\$ns** in scope. This can resolve conflict of names. This lets create private directives, filters and controllers.

If you want to inherit global directives, you may set **scope.\$ns.inheritGlobal** = true.

Example

A few ways to bind a model to the DOM

19.1 1. Auto binding, `al-app`

`alight()` is called on start system, it takes each element with `al-app` and bind it.

Listing 19.1: html

```
<div al-app al-init="title='Hello!'">
  <input al-value="title" type="text" class="form-control" />
  <p>{{title}}</p>
</div>
```

[Example on jsfiddle](#)

19.2 2. Manual binding with `alight()`

You can bind custom elements with `alight()`

Listing 19.2: html

```
<div id="app" al-init="title='Hello!'">
  <input al-value="title" type="text" class="form-control" />
  <p>{{title}}</p>
</div>
```

Listing 19.3: javascript

```
alight('#app'); // bind to DOM
```

[Example on jsfiddle](#)

19.3 3. To bind to element with no DOM

Listing 19.4: html

```
<div id="app"></div>
```

Listing 19.5: javascript

```
var tag = document.createElement('div'); // element
tag.innerHTML = '<input al-value="title" type="text" class="form-control" /><p>{{title}}</p>'; // t

align(tag, {title: 'Hello!'}); // binding

document.querySelector('#app').appendChild(tag); // append to DOM
```

[Example on jsfiddle](#)

19.4 4. Manual binding #2

Listing 19.6: html

```
<div id="app">
  <input al-value="name" type="text" />
  {{name}} <br/>
  <button @click="click()">Set Hello</button>
</div>
```

Listing 19.7: javascript

```
var data = {
  name: 'Some text',
  click: function() {
    data.name = 'Hello';
  }
};

align('#app', data);
```

[Example on jsfiddle](#)

Directive preprocessor

Directive preprocessor lets you control process of creating directives. You can make custom attributes and make different transformations.

Objects:

- **alight.directivePreprocessor** - a default preprocessor, you can change it
- **alight.hooks.directive** - a list of handlers, you can append/remove them

Listing 20.1: Example how to create attribute ‘bold’

```
alight.hooks.directive.ext.splice(1, 0, {  
  code: 'bold', // not necessary  
  fn: function() {  
    if(this.directive.bold) this.element.innerHTML = '<b>' + this.element.innerHTML + '</b>'  
  }  
})
```

Listing 20.2: How to use it

```
alight.directives.al.example = {  
  bold: true  
}
```

- [Example on plunkr](#)
- [Article \[ru\]: «»](#)

Component

- Component is an isolated part of application, it has own change detector.
- Component has an isolated scope.
- Defined properties (:rating, :max in example below) are transferred to scope of component.
- You can choose a template returning template/templateId/templateUrl.
- Events onStart and onDestroy.
- If you use a prefix for name, then it throw error if a component doesn't exist (e.g. al-component).
- You need to attach component.es.js to use it.

Listing 21.1: Syntax

```
alight.createComponent('rating', (scope, element, env) => {
  return {
    template,
    templateId,
    templateUrl,
    props,
    onStart,
    onDestroy
  };
})
```

Listing 21.2: Example of componenet

```
<rating :rating="rating" :max="max" @change="rating=$event.value"></rating>
```

Listing 21.3: Example of componenet

```
alight.createComponent('rating', (scope, element, env) => {
  scope.change = (value) => scope.$dispatch('change', value)
  return {template: `<ul class="rating"><li al-repeat="_ in max" al-class="filled: $index<rating" (
  })
```

If you want to define callback on property change or change type of observing of a property, you can do this in “props”

Listing 21.4: Configuring properties

```
alight.createComponent('somecomponent', (scope, element, env) => {  
  return {  
    props: {  
      prop0: function(value) {},  
      prop1: {  
        watchMode: 'array'  
      },  
      prop2: {  
        watchMode: 'deep',  
        onChange: function(value) {}  
      }  
    }  
  };  
})
```

[Example on jsfiddle](#)

Migration from 0.10 to 0.12

22.1 Controllers

Listing 22.1: html 0.10

```
<div al-controller="main"></div>
```

Listing 22.2: html 0.12

```
<div al-ctrl="main"></div>
```

22.2 Directives

Listing 22.3: directive in 0.10

```
alight.directives.al.test = function(element, value, scope, env) {};
```

Listing 22.4: directive in 0.10

```
alight.directives.al.test = {  
  scope: true, // true / 'isolate' / 'root'  
  link: function(element, value, scope, env) {}  
}
```

Listing 22.5: directive in 0.12

```
alight.directives.al.test = function(scope, element, value, env) {};
```

Listing 22.6: directive 0.12

```
alight.directives.al.test = {  
  scope: true, // it make a new clean scope (object)  
  link: function(scope, element, value, env) {}  
}
```


23.1 How can I take a scope?

for Angular Light v0.10 and older

- Have a scope by tags
- Publish a scope from controller
- Publish a scope using a directive
- Take a scope by a name of controller
- A way to take Scope by element - `alight.getScopeByElement(element)`
- A way to take Scope by element with a directive
- How to call Alert from `al-click`
- **Where can I take a debug version** [Here](#) you can get release and debug of any version, also you can use bower:
`bower install alight`
- **Where is \$http service** Angular Light doesn't have it, you can use `jQuery.ajax` or anyone that you usually use.
But there is `alight.f$.ajax` for inner purpose.
- **Angular Light updates a whole DOM or it tracks changes and update only changed elements?** Only changed elements.
- **I need Angular Light runs my function when a bindings process will finish.** You can observe it
`scope.$scan("$finishBinding", callback)`
- **Is it possible to pass jquery \$element to function from al-click?** You can use `$event` that contains element,
`al-click="someFunc($event)"`
- **How can I use al-repeat not as attribute, but as a comment?** `al-repeat` supports comment binding, [example](#)
- **Why al-show with an empty array doesn't hide my element: `al-show="model.array"`?**
Because there is javascript and `!!model.array` always give you true, you can use
`al-show="model.array.length"`
- **Where is “\$odd, \$even, \$first, \$last, \$rest” for al-repeat?** You can append any attributes, [example how to append \\$odd \\$even](#)
- **How to sort an array?** [Manual sort](#) [Filter](#) [orderBy](#) [Sort](#) props of an object [Call a method](#)
- **Can I rename directives?** Yes, `alight.directives.al.myKeypress = alight.directives.al.keypress`

- **How to call my function when Scope.\$scan finish digets process?** You can pass your function as callback
`Scope.$scan(callback)`
- **How to redraw bind-once?** [Example](#)
- **Where is DI?** Angular Light doesn't have it, but you can make it for yourself, [example](#)
- **Why al-repeat displays not all items?** Probably a input list contains doubles, in this case you should declare ID - using **track by**

```
<div al-repeat="item in list track by $index">
```

```
<div al-repeat="item in list track by items.id">
```

Look at al-repeat