Angular Js v1.3.0 Services Cheat Sheet

by Roman K. (Roman) via cheatography.com/19465/cs/2484/

animate

The $animate service provides rudimentary DOM manipulation functions to insert, remove and move elements within the DOM, as well as adding and removing classes. This service is the core service used by the ngAnimate $animator service which provides high-level animation hooks for CSS and JavaScript.
Methods
enter(element, parent, after, [done]);
Inserts the element into the DOM either after the after element or as the first child within the parent element. Once complete, the done() callback will be fired (if provided).
element, *DOMElement*: the element which will be inserted into the DOM
parent, *DOMElement*: the parent element which will append the element as a child (if the after element is not present)
after, *DOMElement*: the sibling element which will append the element after itself
done(optional), *Function=*: callback function that will be called after the element has been inserted into the DOM
move(element, parent, after, [done]);
Moves the position of the provided element within the DOM to be placed either after the after element or inside of the parent element. Once complete, the done() callback will be fired (if provided).
element, *DOMElement*: the element which will be moved around within the DOM
parent. *DOMElement*: the parent element where the element will be inserted into (if the after element is not present)
after, *DOMElement*: the sibling element where the element will be positioned next to
done(optional), *Function=*: the callback function (if provided) that will be fired after the element has been moved to its new position
addClass(element, className, [done]);

animate (cont)

Adds the provided className CSS class value to the provided element. Once complete, the done() callback will be fired (if provided).
element, *DOMElement*: the element which will have the className value added to it
className, *string*: the CSS class which will be added to the element
done(optional), *Function=*: the callback function (if provided) that will be fired after the className value has been added to the element
removeClass(element, className, [done]);
Removes the provided className CSS class value from the provided element. Once complete, the done() callback will be fired (if provided).
element, *DOMElement*: the element which will have the className value removed from it
className, *string*: the CSS class which will be removed from the element
done(optional), *Function=*: the callback function (if provided) that will be fired after the className value has been removed from the element
setClass(element, add, remove, [done]);
Adds and/or removes the given CSS classes to and from the element. Once complete, the done() callback will be fired (if provided).
element, *DOMElement*: the element which will have its CSS classes changed removed from it
add, *string*: the CSS classes which will be added to the element
remove, *string*: the CSS class which will be removed from the element
done(optional), *Function=*: the callback function (if provided) that will be fired after the CSS classes have been set on the element

cacheFactory

Factory that constructs Cache objects and gives access to them.
Usage:
$cacheFactory(cacheId, [options]);
Arguments:
cacheId, *string*: Name or id of the newly created cache.
options(optional), *object*: Options object that specifies the cache behavior. Properties:
{number=} capacity — turns the cache into LRU cache.
Returns:
object: Newly created cache object with the following set of methods:
{object} info() — Returns id, size, and options of cache.
{{}} put({string} key, *{} value)* — Puts a new key-value pair into the cache and returns it.
{{}} get({string} key)* — Returns cached value for key or undefined for cache miss.
{void} remove({string} key) — Removes a key-value pair from the cache.
{void} removeAll() — Removes all cached values.
{void} destroy() — Removes references to this cache from $cacheFactory.
Methods:
info(); — Get information about all the caches that have been created
Returns:
Object — key-value map of cacheId to the result of calling cache#info
get(cacheId); — Get access to a cache object by the cacheId used when it was created.
Parameters:
cacheId,string — Name or id of a cache to access.
Returns:

cacheFactory (cont)

object: Cache object identified by the cacheId or undefined if no such cache.

compile

Compiles an HTML string or DOM into a template and produces a template function, which can then be used to link scope and the template together.
Usage
$compile(element, transclude, maxPriority);
Arguments
element, *(string | DOMElement)*: Element or HTML string to compile into a template function.
transclude, *function(angular.Scope, cloneAttachFn=)*:
function available to directives.
maxPriority, *number*: only apply directives lower than given priority (Only effects the root element(s), not their children)
Returns
function(scope, cloneAttachFn=): a link function which is used to bind template (a DOM element/tree) to a scope. Where:
scope - A Scope to bind to.
cloneAttachFn - If cloneAttachFn is provided, then the link function will clone the template and call the cloneAttachFn function allowing the caller to attach the cloned elements to the DOM document at the appropriate place. The cloneAttachFn is called as: *cloneAttachFn(clonedElement, scope)* where:
clonedElement - is a clone of the original element passed into the compiler.
scope - is the current scope with which the linking function is working with.
Calling the linking function returns the element of the template. It is either the original element passed in, or the clone of the element if the cloneAttachFn is provided.

interpolate

Compiles a string with markup into an interpolation function. This service is used by the HTML $compile service for data binding.
Usage
$interpolate(text, [mustHaveExpression], [trustedContext], [allOrNothing]);
text - {string} - The text with markup to interpolate.
mustHaveExpression (optional) - {boolean} - if set to true then the interpolation string must have embedded expression in order to return an interpolation function. Strings with no embedded expression will return null for the interpolation function.
trustedContext (optional) - {string} - when provided, the returned function passes the interpolated result through $sce.getTrusted(interpolatedResult, trustedContext) before returning it. Refer to the $sce service that provides Strict Contextual Escaping for details.
allOrNothing (optional) - {boolean} - if true, then the returned function returns undefined unless all embedded expressions evaluate to a value other than undefined.
Returns:
function(context) - an interpolation function which is used to compute the interpolated string. The function has these parameters:
- context: evaluation context for all expressions embedded in the interpolated text
Methods:
startSymbol(); - Symbol to denote the start of expression in the interpolated string. Defaults to {{.
Use $interpolateProvider#startSymbol to change the symbol.
Returns:
{string} - start symbol.

interpolate (cont)

endSymbol(); - Symbol to denote the end of expression in the interpolated string. Defaults to }}.
Use $interpolateProvider#endSymbol to change the symbol.

parse

Converts Angular expression into a function.
Usage
$parse(expression);
Arguments:
expression - {string} - String expression to compile.
Returns:
function(context, locals) - a function which represents the compiled expression:
context – {object} – an object against which any expressions embedded in the strings are evaluated against (typically a scope object).
locals – {object=} – local variables context object, useful for overriding values in context.
The returned function also has the following properties:
literal – {boolean} – whether the expression's top-level node is a JavaScript literal.
constant – {boolean} – whether the expression is made entirely of JavaScript constant literals.
assign – {?function(context, value)} – if the expression is assignable, this will be set to a function to change its value on the given context.

controller

$controller service is responsible for instantiating controllers.
Usage
$controller(constructor, locals);
Arguments

controller (cont)

constructor, *(function() | string)*: If called with a function then it's considered to be the controller constructor function. Otherwise it's considered to be a string which is used to retrieve the controller constructor using the following steps:
- check if a controller with given name is registered via $controllerProvider
- check if evaluating the string on the current scope returns a constructor
- if $controllerProvider#allowGlobals, check window[constructor] on the global window object (not recommended)

exceptionHandler

Any uncaught exception in angular expressions is delegated to this service. The default implementation simply delegates to $log.error which logs it into the browser console.
Example:
angular.module('exceptionOverride', []).factory('$exceptionHandler', function () {`
return function (exception, cause) {
exception.message += (caused by '" + cause + "')';`
throw exception;
};
});
This example will override the normal action of $exceptionHandler, to make angular exceptions fail hard when they happen, instead of just logging to the console.
Usage
$exceptionHandler(exception, [cause]);
Arguments
exception, *Error*: Exception associated with the error.
cause (optional), *string*: optional information about the context in which the error was thrown.

sce

$sce is a service that provides Strict Contextual Escaping services to AngularJS.
Usage
$sce();
Methods:
isEnabled(); - Returns a boolean indicating if SCE is enabled.
Returns:
Boolean - true if SCE is enabled, false otherwise. If you want to set the value, you have to do it at module config time on $sceProvider.
parseAs(type, expression); - Converts Angular expression into a function. This is like $parse and is identical when the expression is a literal constant. Otherwise, it wraps the expression in a call to $sce.getTrusted(type, result)
Parameters:
type - {string} - The kind of SCE context in which this result will be used.
expression - {string} - String expression to compile.
Returns:
function(context, locals) - a function which represents the compiled expression:
context – {object} – an object against which any expressions embedded in the strings are evaluated against (typically a scope object).
locals – {object=} – local variables context object, useful for overriding values in context.
trustAs(type, value); - Delegates to $sceDelegate.trustAs. As such, returns an object that is trusted by angular for use in specified strict contextual escaping contexts (such as ng-bind-html, ng-include, any src attribute interpolation, any dom event binding attribute interpolation such as for onclick, etc.) that uses the provided value. See * $sce for enabling strict contextual escaping.
Parameters:

sce (cont)

type - {string} - The kind of context in which this value is safe for use. e.g. url, resource_url, html, js and css.
value - {*} - The value that that should be considered trusted/safe.
Returns
* - A value that can be used to stand in for the provided value in places where Angular expects a $sce.trustAs() return value.
trustAsHtml(value); - Shorthand method. $sce.trustAsHtml(value) → $sceDelegate.trustAs($sce.HTML, value)
Parameters:
value - * - The value to trustAs.
Returns:
* - An object that can be passed to $sce.getTrustedHtml(value) to obtain the original value. (privileged directives only accept expressions that are either literal constants or are the return value of $sce.trustAs.)
trustAsUrl(value); - Shorthand method. $sce.trustAsUrl(value) → $sceDelegate.trustAs($sce.URL, value)
Parameters:
value - * - The value to trustAs.
Returns:
* - An object that can be passed to $sce.getTrustedUrl(value) to obtain the original value. (privileged directives only accept expressions that are either literal constants or are the return value of $sce.trustAs.)
trustAsResourceUrl(value); - Shorthand method. $sce.trustAsResourceUrl(value) → $sceDelegate.trustAs($sce.RESOURCE_URL, value)
Parameters:
value - * - The value to trustAs.
Returns:

sce (cont)

* - An object that can be passed to $sce.getTrustedResourceUrl(value) to obtain the original value. (privileged directives only accept expressions that are either literal constants or are the return value of $sce.trustAs.)

trustAsJs(value); - Shorthand method. $sce.trustAsJs(value) → $sceDelegate.trustAs($sce.JS, value)
Parameters:
value, * - The value to trustAs.
Returns:
* - An object that can be passed to $sce.getTrustedJs(value) to obtain the original value. (privileged directives only accept expressions that are either literal constants or are the return value of $sce.trustAs.)

getTrusted(type, maybeTrusted); - Delegates to $sceDelegate.getTrusted. As such, takes the result of a $sce.trustAs() call and returns the originally supplied value if the queried context type is a supertype of the created type. If this condition isn't satisfied, throws an exception.
Parameters:
type - {string} - The kind of context in which this value is to be used.
maybeTrusted - {*} - The result of a prior $sce.trustAs call.
Returns:
* - The value the was originally provided to $sce.trustAs if valid in this context. Otherwise, throws an exception.

getTrustedHtml(value); - Shorthand method. $sce.getTrustedHtml(value) → $sceDelegate.getTrusted($sce.HTML, value)
Parameters:
value - {*} - The value to pass to $sce.getTrusted.
Returns:

sce (cont)

* - The return value of $sce.getTrusted($sce.HTML, value)

getTrustedCss(value); - Shorthand method. $sce.getTrustedCss(value) → $sceDelegate.getTrusted($sce.CSS, value)
Parameters
value - {*} - The value to pass to $sce.getTrusted.
Returns:
* - The return value of $sce.getTrusted($sce.CSS, value)

getTrustedUrl(value); - Shorthand method. $sce.getTrustedUrl(value) → $sceDelegate.getTrusted($sce.URL, value)
Parameters:
value - {*} - The value to pass to $sce.getTrusted.
Returns:
* - The return value of $sce.getTrusted($sce.URL, value)

getTrustedResourceUrl(value); - Shorthand method. $sce.getTrustedResourceUrl(value) → $sceDelegate.getTrusted($sce.RESOURCE_URL, value)
Parameters:
value - {*} - The value to pass to $sceDelegate.getTrusted.
Returns:
* - The return value of $sce.getTrusted($sce.RESOURCE_URL, value)

getTrustedJs(value); - Shorthand method. $sce.getTrustedJs(value) → $sceDelegate.getTrusted($sce.JS, value)
Parameters:
value - * - The value to pass to $sce.getTrusted.
Returns:
* - The return value of $sce.getTrusted($sce.JS, value)

sce (cont)

parseAsHtml(expression); - Shorthand method. $sce.parseAsHtml(expression string) → $sce.parseAs($sce.HTML, value)
Parameters:
expression - {string} - String expression to compile.
Returns:
function(context, locals) - a function which represents the compiled expression:
context – {object} – an object against which any expressions embedded in the strings are evaluated against (typically a scope object).
locals – {object=} – local variables context object, useful for overriding values in context.

parseAsCss(expression); - Shorthand method. $sce.parseAsCss(value) → $sce.parseAs($sce.CSS, value)
Parameters:
expression - {string} - String expression to compile.
Returns:
function(context, locals) - a function which represents the compiled expression:
context – {object} – an object against which any expressions embedded in the strings are evaluated against (typically a scope object).
locals – {object=} – local variables context object, useful for overriding values in context.

parseAsUrl(expression); - Shorthand method. $sce.parseAsUrl(value) → $sce.parseAs($sce.URL, value)
Parameters:
expression - {string} - String expression to compile.
Returns:
function(context, locals) - a function which represents the compiled expression:
context – {object} – an object against which any expressions embedded in the strings are evaluated against (typically a scope object).

sce (cont)

locals – {object=} – local variables context object, useful for overriding values in context.

parseAsResourceUrl(expression); - Shorthand method. $sce.parseAsResourceUrl(value) → $sce.parseAs($sce.RESOURCE_URL, value)

Parameters:

expression - {string} - String expression to compile.

Returns:

function(context, locals) - a function which represents the compiled expression:

context – {object} – an object against which any expressions embedded in the strings are evaluated against (typically a scope object).

locals – {object=} – local variables context object, useful for overriding values in context.

parseAsJs(expression); - Shorthand method. $sce.parseAsJs(value) → $sce.parseAs($sce.JS, value)

Parameters:

expression - {string} - String expression to compile.

Returns:

function(context, locals) - a function which represents the compiled expression:

context – {object} – an object against which any expressions embedded in the strings are evaluated against (typically a scope object).

locals – {object=} – local variables context object, useful for overriding values in context.

sceDelegate

$sceDelegate is a service that is used by the $sce service to provide Strict Contextual Escaping (SCE) services to AngularJS.

Methods:

sceDelegate (cont)

trustAs(type, value); - Returns an object that is trusted by angular for use in specified strict contextual escaping contexts (such as ng-bind-html, ng-include, any src attribute interpolation, any dom event binding attribute interpolation such as for onclick, etc.) that uses the provided value. See $sce for enabling strict contextual escaping.

Parameters:

type - {string} - The kind of context in which this value is safe for use. e.g. url, resourceUrl, html, js and css.

value - {*} - The value that that should be considered trusted/safe.

Returns:

*- A value that can be used to stand in for the provided value in places where Angular expects a $sce.trustAs() return value.

valueOf(value); - If the passed parameter had been returned by a prior call to $sceDelegate.trustAs, returns the value that had been passed to $sceDelegate.trustAs. If the passed parameter is not a value that had been returned by $sceDelegate.trustAs, returns it as-is.

Parameters:

value - {*} - The result of a prior $sceDelegate.trustAs call or anything else.

Returns:

{*} - The value that was originally provided to $sceDelegate.trustAs if value is the result of such a call. Otherwise, returns value unchanged.

getTrusted(type, maybeTrusted); - Takes the result of a $sceDelegate.trustAs call and returns the originally supplied value if the queried context type is a supertype of the created type. If this condition isn't satisfied, throws an exception.

Parameters:

type - {string} - The kind of context in which this value is to be used.

sceDelegate (cont)

maybeTrusted - {*} - The result of a prior $sceDelegate.trustAs call.

Returns:

*- The value the was originally provided to $sceDelegate.trustAs if valid in this context. Otherwise, throws an exception.

filter (service)

Filters are used for formatting data displayed to the user.

Usage

$filter(name);

name, *String*: Name of the filter function to retrieve

Returns

Function: the filter function

http

The $http service is a core Angular service that facilitates communication with the remote HTTP servers via the browser's XMLHttpRequest object or via JSONP.

Usage

$http(config);

Arguments

config, *object*: Object describing the request to be made and how it should be processed. The object has following properties:

method – {string} – HTTP method (e.g. 'GET', 'POST', etc)

url – {string} – Absolute or relative URL of the resource that is being requested.

params – {Object.<string|Object>} – Map of strings or objects which will be turned to ?key1=value1&key2=value2 after the url. If the value is not a string, it will be JSONified.

data – {string|Object} – Data to be sent as the request message data.

http (cont)

headers – {Object} – Map of strings or functions which return strings representing HTTP headers to send to the server. If the return value of a function is null, the header will not be sent.
xsrfHeaderName – {string} – Name of HTTP header to populate with the XSRF token.
xsrfCookieName – {string} – Name of cookie containing the XSRF token.
transformRequest – {function(data, headersGetter)|Array.<function(data, headersGetter)>} – transform function or an array of such functions. The transform function takes the http request body and headers and returns its transformed (typically serialized) version.
transformResponse – {function(data, headersGetter)|Array.<function(data, headersGetter)>} – transform function or an array of such functions. The transform function takes the http response body and headers and returns its transformed (typically deserialized) version.
cache – {boolean|Cache} – If true, a default $http cache will be used to cache the GET request, otherwise if a cache instance built with $cacheFactory, this cache will be used for caching.
timeout – {number|Promise} – timeout in milliseconds, or promise that should abort the request when resolved.
withCredentials - {boolean} - whether to set the withCredentials flag on the XHR object. See requests with credentials for more information.
responseType - {string} - see requestType.
Returns
HttpPromise: Returns a promise object with the standard then method and two http specific methods: success and error. The then method takes two arguments a success and an error callback which will be called with a response object.

http (cont)

The success and error methods take a single argument - a function that will be called when the request succeeds or fails respectively. The arguments passed into these functions are destructured representation of the response object passed into the then method. The response object has these properties:. *data* – {string|Object} – The response body transformed with the transform functions.
status – {number} – HTTP status code of the response.
headers – {function([headerName])} – Header getter function.
config – {Object} – The configuration object that was used to generate the request.
statusText – {string} – HTTP status text of the response.
Methods
get(url, [config]);: Shortcut method to perform GET request.
Parameters:
url — {string} — Relative or absolute URL specifying the destination of the request
config(optional) — {Object} — Optional configuration object
Returns:
HttpPromise — Future object
delete(url, [config]);: Shortcut method to perform DELETE request.
Parameters
url — {string} — Relative or absolute URL specifying the destination of the request
config (optional) — {Object}: Optional configuration object
Returns
HttpPromise — Future object
head(url, [config]); — Shortcut method to perform HEAD request.
Parameters

http (cont)

url — {string} — Relative or absolute URL specifying the destination of the request
config (optional) — {Object} — Optional configuration object
Returns
HttpPromise — Future object
jsonp(url, [config]); — Shortcut method to perform JSONP request.
Parameters
url — {string} — Relative or absolute URL specifying the destination of the request. The name of the callback should be the string JSON_CALLBACK.
config (optional) — {Object} — Optional configuration object
Returns
HttpPromise — Future object
post(url, data, [config]); — Shortcut method to perform POST request.
Parameters
url — {string} — Relative or absolute URL specifying the destination of the request
data — {*} — Request content
config (optional) — {Object} — Optional configuration object
Returns
HttpPromise — Future object
put(url, data, [config]); — Shortcut method to perform PUT request.
Parameters
url — {string} — Relative or absolute URL specifying the destination of the request
data — {*} — Request content
config (optional) — {Object} — Optional configuration object
Returns
HttpPromise — Future object
patch(url, data, [config]); — Shortcut method to perform PATCH request.

http (cont)

Parameters
url — {string} — Relative or absolute URL specifying the destination of the request
data — {*} — Request content
config (optional) — {Object} — Optional configuration object
Returns
HttpPromise — Future object
Properties
pendingRequests {Array.<Object>} — Array of config objects for currently pending requests. This is primarily meant to be used for debugging purposes.
defaults — Runtime equivalent of the $httpProvider.defaults property. Allows configuration of default headers, withCredentials as well as request and response transformations.
See "Setting HTTP Headers" and "Transforming Requests and Responses" sections above.

httpBackend

HTTP backend used by the service ($http) that delegates to XMLHttpRequest object or JSONP and deals with browser incompatibilities.

q

A promise/deferred implementation inspired by Kris Kowal's Q.
Usage:
$q(resolver);
Arguments:
resolver - {function(function, function)} - Function which is responsible for resolving or rejecting the newly created promise. The first parameter is a function which resolves the promise, the second parameter is a function which rejects the promise.
Returns:
Promise - The newly created promise.
Methods:

q (cont)

defer(); - Creates a Deferred object which represents a task which will finish in the future.
Returns:
Deferred - Returns a new instance of deferred.
reject(reason); - Creates a promise that is resolved as rejected with the specified reason. This api should be used to forward rejection in a chain of promises. If you are dealing with the last promise in a promise chain, you don't need to worry about it. When comparing deferreds/promises to the familiar behavior of try/catch/throw, think of reject as the throw keyword in JavaScript. This also means that if you "catch" an error via a promise error callback and you want to forward the error to the promise derived from the current promise, you have to "rethrow" the error by returning a rejection constructed via reject.. Parameters:
reason - {*} - Constant, message, exception or an object representing the rejection reason.
Returns:
Promise - Returns a promise that was already resolved as rejected with the reason.
when(value); - Wraps an object that might be a value or a (3rd party) then-able promise into a $q promise. This is useful when you are dealing with an object that might or might not be a promise, or if the promise comes from a source that can't be trusted.
Parameters:
value - {*} - Value or a promise
Returns:
Promise - Returns a promise of the passed value or promise
all(promises); - Combines multiple promises into a single promise that is resolved when all of the input promises are resolved.

q (cont)

Parameters:
promises - {Array.<Promise>Object.<Promise>} - An array or hash of promises.
Returns:
Promise - Returns a single promise that will be resolved with an array/hash of values, each value corresponding to the promise at the same index/key in the promises array/hash. If any of the promises is resolved with a rejection, this resulting promise will be rejected with the same rejection value.

interval

Angular's wrapper for window.setInterval. The fn function is executed every delay milliseconds.
Usage
$interval(fn, delay, [count], [invokeApply]);
Arguments:
fn - {function()} - A function that should be called repeatedly.
delay - {number} - Number of milliseconds between each function call.
count (optional) - {number} - Number of times to repeat. If not set, or 0, will repeat indefinitely. (default: 0)
invokeApply (optional) - {boolean} - If set to false skips model dirty checking, otherwise will invoke fn within the $apply block. (default: true)
Returns:
promise - A promise which will be notified on each iteration.
Methods:
cancel(promise); - Cancels a task associated with the promise.
Parameters:
promise - {promise} - returned by the $interval function.
Returns:
boolean - Returns true if the task was successfully canceled.

timeout

Angular's wrapper for window.setTimeout. The fn function is wrapped into a try/catch block and delegates any exceptions to $exceptionHandler service.
Usage:
$timeout(fn, [delay], [invokeApply]);
Arguments:
fn - {function()} - A function, whose execution should be delayed.
delay, (optional) - {number} - Delay in milliseconds.(default: 0)
invokeApply, (optional) - {boolean} - If set to false skips model dirty checking, otherwise will invoke fn within the $apply block. (default: true)
Returns:
Promise - Promise that will be resolved when the timeout is reached. The value this promise will be resolved with is the return value of the fn function.
Methods:
cancel([promise]); - Cancels a task associated with the promise. As a result of this, the promise will be resolved with a rejection.
Parameters:
promise, (optional) - {Promise} - Promise returned by the $timeout function.
Returns:
boolean - Returns true if the task hasn't executed yet and was successfully canceled.

document

A jQuery or jqLite wrapper for the browser's window.document object.

window

A reference to the browser's window object. While window is globally available in JavaScript, it causes testability problems, because it is a global variable. In angular we always refer to it through the $window service, so it may be overridden, removed or mocked for testing.

locale

$locale service provides localization rules for various Angular components. As of right now the only public api is:
id – {string} – locale id formatted as languageId-countryId (e.g. en-us)

location

The $location service parses the URL in the browser address bar (based on the window.location) and makes the URL available to your application. Changes to the URL in the address bar are reflected into $location service and changes to $location are reflected into the browser address bar.
Methods:
absUrl(); - This method is getter only. Return full url representation with all segments encoded according to rules specified in RFC 3986.
Returns:
string - full url
url([url], [replace]); - This method is getter / setter. Return url (e.g. /path?a=b#hash) when called without any parameter. Change path, search and hash, when called with parameter and return $location.
Parameters:
url (optional) - {string} - New url without base prefix (e.g. /path?a=b#hash)
replace (optional) - {string} - The path that will be changed
Returns:
{string} - url
protocol(); - This method is getter only. Return protocol of current url.
Returns:
string - protocol of current url
host(); - This method is getter only. Return host of current url.
Returns:
string - host of current url.

location (cont)

port(); - This method is getter only. Return port of current url.
Returns:
Number - port
path([path]); - This method is getter / setter. Return path of current url when called without any parameter. Change path when called with parameter and return $location. Note: Path should always begin with forward slash (/), this method will add the forward slash if it is missing.
Parameters:
path (optional) - {string} - New path
Returns:
{string} - path
search(search, [paramValue]); - This method is getter / setter. Return search part (as object) of current url when called without any parameter. Change search part when called with parameter and return $location.
Parameters:
search - {stringObject.<string>Object.<Array.<string>> } - New search params - string or hash object.
When called with a single argument the method acts as a setter, setting the search component of $location to the specified value. If the argument is a hash object containing an array of values, these values will be encoded as duplicate search parameters in the url.
paramValue (optional) - {stringArray.<string>boolean} If search is a string, then paramValue will override only a single search property.
If paramValue is an array, it will override the property of the search component of $location specified via the first argument.
If paramValue is null, the property specified via the first argument will be deleted.

location (cont)

If paramValue is true, the property specified via the first argument will be added with no value nor trailing equal sign.
Returns:
Object - If called with no arguments returns the parsed search object. If called with one or more arguments returns $location object itself.
hash([hash]); - This method is getter / setter. Return hash fragment when called without any parameter. Change hash fragment when called with parameter and return $location.
Parameters:
hash (optional) - {string} - New hash fragment
Returns:
string - hash
replace(); - If called, all changes to $location during current $digest will be replacing current history record, instead of adding new one.
Events:
$locationChangeStart - Broadcasted before a URL will change. This change can be prevented by calling preventDefault method of the event. See $rootScope.Scope for more details about event object. Upon successful change $locationChangeSuccess is fired.
Type:
broadcast
Target:
root scope
$locationChangeSuccess
Broadcasted after a URL was changed.
Type:
broadcast
Target:
root scope

anchorScroll

When called, it checks current value of $location.hash() and scrolls to the related element, according to rules specified in Html5 spec.
Usage
`$ancho rSc roll();`

log

Simple service for logging. Default implementation safely writes the message into the browser's console (if present).
Methods:
log(); - Write a log message
info(); - Write an information message
warn(); - Write a warning message
error(); - Write an error message
debug(); - Write a debug message

rootElement

The root element of Angular application. This is either the element where ngApp was declared or the element passed into angular.bootstrap. The element represent the root element of application. It is also the location where the applications $injector service gets published, it can be retrieved using $rootElement.injector().

rootScope

Every application has a single root scope. All other scopes are descendant scopes of the root scope. Scopes provide separation between the model and the view, via a mechanism for watching the model for changes. They also provide an event emission/broadcast and subscription facility.

templateCache

The first time a template is used, it is loaded in the template cache for quick retrieval. You can load templates directly into the cache in a script tag, or by consuming the $templateCache service directly.

templateCache (cont)

Adding via the script tag:
<script type="text/ng-template" id="templateId.html">
<p>This is the content of the template</p>
</script>
Adding via the $templateCache service:
var myApp = angular.module('myApp', []);
myApp.run(function($templateCache) {
$templateCache.put('templateId.html', 'This is the content of the template');
});
To retrieve the template later, simply use it in your HTML:
<div ng-include=" 'templateId.html' "></div>
or get it via Javascript:
$templateCache.get('templateId.html')

templateRequest

The $templateRequest service downloads the provided template using $http and, upon success, stores the contents inside of $templateCache. If the HTTP request fails or the response data of the HTTP request is empty then a $compile error will be thrown (the exception can be thwarted by setting the 2nd parameter of the function to true).
Usage:
$templateRequest(tpl, [ignoreRequestError]);
Arguments:
tpl - {string} - The HTTP request template URL
ignoreRequestError, (optional) - {boolean} - Whether or not to ignore the exception when the request fails or the template is empty
Returns:
Promise - the HTTP Promise for the given.
Properties:
totalPendingRequests - {number} - total amount of pending template requests being downloaded.