Документация по API Angular

Используйте документацию по API, когда вам нужно больше информации о специфической возможности.

Модуль ng

Директивы

a

Описание

Модифицированное поведение по умолчанию для тега html A, так как по умолчанию действие требует обязательного наличия заполненного атрибута href.

Причиной этого изменения стало легкое создание ссылок для действий с директивой `ngClick` без изменения страницы и без ее перезагрузки, например: `<a href="" ng-click="model.$save()">Save</a>`

Использование

Эта директива может быть использована как пользовательский элемент, но с предварительным созданием, если вы используете IE:

```
1. <a>
2. </a>
```

form

Описание

Директива создает `FormController`.

Если атрибут `name` указан, тогда контроллер формы опубликует одноименное свойство внутри текущей области видимости, через которое можно получить ссылку на форму.

Псевдоним: `ngForm`

В angular формы могут быть вложенными. Это означает, что внешняя форма действительна, только тогда, когда действительны все ее вложенные формы. Однако браузеры не позволяют вкладывать элементы `<form>` друг в друга, по этой причине angular обеспечивает псевдоним `ngForm`, который ведет себя, так же как и `<form>`, но позволяет использовать вложения.

CSS классы

- `ng-valid` устанавливается если форма не имеет ошибок.
- `ng-invalid` Устанавливается если форма имеет ошибки ввода.
- `ng-pristine` Устанавливается если пользователь еще не взаимодействовал с формой.
- `ng-dirty` Устанавливается когда пользователь уже взаимодействовал с формой.

Отправка формы и действие по умолчанию

Поскольку роль форм в приложениях Angular отличается от классических приложений, желательно для браузера, не перезагружая страницы отправить данные на сервер. Для этого запускается некоторая логика JavaScript, чтобы обработать определенным образом отправку представление формы.

По этой причине Angular предотвращает действие по умолчанию (отправка формы на сервер) если элемент `<form>` имеет установленный атрибут `action`.

Вы можете использовать одну из двух возможностей, для указания метода JavaScript, который должен вызываться при отправке формы:

- `ngSubmit` директива для элемента формы
- `ngClick` директива для первой кнопки или поля ввода с типом submit (input[type=submit])

Для предотвращения двойного срабатывания обработчика, используйте только одну из директив ngSubmit или ngClick. Это из-за следующих правил отправки формы из спецификации html:

- Если форма имеет только один элемент ввода данных, при нажатии на enter в этой форме, будет выполнена ее отправка (ngSubmit).
- Если форма имеет 2 или больше элемента управления и не имеет кнопки или элемента ввода с типом submit, тогда нажатие на enter не приводит к отправке формы
- Если форма имеет одно или более поле ввода и одну или более кнопку или элемент input с типом submit, тогда нажатие enter в любом поле ввода будет вызывать обработчик для первой кнопки или input[type=submit] (`ngClick`) *и* отправлять форму (`ngSubmit`)

Использование

Эта директива может использоваться как дополнительный элемент, но с учетом ограничений IE:

```
<form
       [name="{string}"]>
</form>
```

Параметры

- `name`*`(optional)`* `– {string=} –`

  Имя для формы. Если указано, тогда контроллер формы опубликует ссылку на форму в области видимости, используя это имя.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <form name="myForm" ng-controller="Ctrl">
```

```
      userType: <input name="input" ng-model="userType" required>
      <span class="error" ng-show="myForm.input.$error.required">Required!</span><
  br>
      <tt>userType = {{userType}}</tt><br>
      <tt>myForm.input.$valid = {{myForm.input.$valid}}</tt><br>
      <tt>myForm.input.$error = {{myForm.input.$error}}</tt><br>
      <tt>myForm.$valid = {{myForm.$valid}}</tt><br>
      <tt>myForm.$error.required = {{!!myForm.$error.required}}</tt><br>
    </form>
  </body>
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.userType = 'guest';
}
```

End to end test

```
it('should initialize to model', function() {
 expect(binding('userType')).toEqual('guest');
 expect(binding('myForm.input.$valid')).toEqual('true');
});

it('should be invalid if empty', function() {
 input('userType').enter('');
 expect(binding('userType')).toEqual('');
 expect(binding('myForm.input.$valid')).toEqual('false');
});
```

input

Описание

Элемент ввода HTML с angular привязкой данных. Элемент ввода реализует все типы HTML5 элементов ввода и такие поли филы как проверка ввода для старых браузеров.

Использование

Эта директива может использоваться как дополнительный элемент, но с учетом ограничений IE:

```
<input
       ngModel="{string}"
       [name="{string}"]
       [required]
       [ngRequired="{boolean}"]
       [ngMinlength="{number}"]
       [ngMaxlength="{number}"]
       [ngPattern="{string}"]
```

```
        [ngChange="{string}"]>
</input>
```

Параметры

- `ngModel – {string} –`

  Ассоциированное angular выражение для привязки данных.

- `name(optional) – {string=} –`

  Имя свойства, под которым элемент управления будет доступен в области видимости.

- `required(optional) – {string=} –`

  Устанавливает ключ `required` в ошибках проверки данных, когда у поля отсутствует значение.

- `ngRequired(optional) – {boolean=} –`

  Устанавливает атрибут `required`, если выражение возвращает true

- `ngMinlength(optional) – {number=} –`

  Устанавливает ключ `minlength` в ошибках проверки данных, если значение короче, чем установленное количество знаков.

- `ngMaxlength(optional) – {number=} –`

  Устанавливает ключ `maxlength` в ошибках проверки данных, если значение длиннее, чем установленное количество знаков.

- `ngPattern(optional) – {string=} –`

  Устанавливает ключ `pattern` в ошибках проверки данных, если ввод не соответствует шаблону RegExp. Значение сравнивается с переданным паттерном, если он передается в формате `/regexp/`, в других случаях считается что передано свойство текущей области видимости, которое содержит нужный паттерн.

- `ngChange(optional) – {string=} –`

  Angular выражение, которое будет выполнено, когда измениться содержимое элемента управления в результате взаимодействия с пользователем.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
```

```
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-controller="Ctrl">
      <form name="myForm">
        User name: <input type="text" name="userName" ng-model="user.name" require
d>
        <span class="error" ng-show="myForm.userName.$error.required">
          Required!</span><br>
        Last name: <input type="text" name="lastName" ng-model="user.last"
          ng-minlength="3" ng-maxlength="10">
        <span class="error" ng-show="myForm.lastName.$error.minlength">
          Too short!</span>
        <span class="error" ng-show="myForm.lastName.$error.maxlength">
          Too long!</span><br>
      </form>
      <hr>
      <tt>user = {{user}}</tt><br/>
      <tt>myForm.userName.$valid = {{myForm.userName.$valid}}</tt><br>
      <tt>myForm.userName.$error = {{myForm.userName.$error}}</tt><br>
      <tt>myForm.lastName.$valid = {{myForm.lastName.$valid}}</tt><br>
      <tt>myForm.userName.$error = {{myForm.lastName.$error}}</tt><br>
      <tt>myForm.$valid = {{myForm.$valid}}</tt><br>
      <tt>myForm.$error.required = {{!!myForm.$error.required}}</tt><br>
      <tt>myForm.$error.minlength = {{!!myForm.$error.minlength}}</tt><br>
      <tt>myForm.$error.maxlength = {{!!myForm.$error.maxlength}}</tt><br>
    </div>
  </body>
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.user = {name: 'guest', last: 'visitor'};
}
```

End to end test

```
it('should initialize to model', function() {
  expect(binding('user')).toEqual('{"name":"guest","last":"visitor"}');
  expect(binding('myForm.userName.$valid')).toEqual('true');
  expect(binding('myForm.$valid')).toEqual('true');
});

it('should be invalid if empty when required', function() {
  input('user.name').enter('');
  expect(binding('user')).toEqual('{"last":"visitor"}');
  expect(binding('myForm.userName.$valid')).toEqual('false');
  expect(binding('myForm.$valid')).toEqual('false');
```

```
});

it('should be valid if empty when min length is set', function() {
  input('user.last').enter('');
  expect(binding('user')).toEqual('{"name":"guest","last":""}');
  expect(binding('myForm.lastName.$valid')).toEqual('true');
  expect(binding('myForm.$valid')).toEqual('true');
});

it('should be invalid if less than required min length', function() {
  input('user.last').enter('xx');
  expect(binding('user')).toEqual('{"name":"guest"}');
  expect(binding('myForm.lastName.$valid')).toEqual('false');
  expect(binding('myForm.lastName.$error')).toMatch(/minlength/);
  expect(binding('myForm.$valid')).toEqual('false');
});

it('should be invalid if longer than max length', function() {
  input('user.last').enter('some ridiculously long name');
  expect(binding('user'))
    .toEqual('{"name":"guest"}');
  expect(binding('myForm.lastName.$valid')).toEqual('false');
  expect(binding('myForm.lastName.$error')).toMatch(/maxlength/);
  expect(binding('myForm.$valid')).toEqual('false');
});
```

input [checkbox]

Описание

HTML флаг выбора.

Использование

```
<input type="checkbox"
       ng-model="{string}"
       [name="{string}"]
       [ng-true-value="{string}"]
       [ng-false-value="{string}"]
       [ng-change="{string}"]>
```

Параметры

- `ngModel – {string} –`

  Назначаемое angular выражение для привязки данных.

- `name(optional) – {string=} –`

  Имя свойства, под которым элемент управления будет доступен в области видимости.

- `ngTrueValue(optional) – {string=} –`

  Значение, в которое установится привязанное свойство, если флаг установлен.

- `ngFalseValue(optional) – {string=} –`

  Значение, в которое установится привязанное свойство, если флаг сброшен.

- `ngChange(optional) – {string=} –`

  Angular выражение, которое будет выполнено, когда измениться содержимое элемента управления в результате взаимодействия с пользователем.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <form name="myForm" ng-controller="Ctrl">
      Value1: <input type="checkbox" ng-model="value1"> <br/>
      Value2: <input type="checkbox" ng-model="value2"
                     ng-true-value="YES" ng-false-value="NO"> <br/>
      <tt>value1 = {{value1}}</tt><br/>
      <tt>value2 = {{value2}}</tt><br/>
     </form>
  </body>
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.value1 = true;
  $scope.value2 = 'YES'
}
```

End to end test

```
it('should change state', function() {
  expect(binding('value1')).toEqual('true');
  expect(binding('value2')).toEqual('YES');

  input('value1').check();
  input('value2').check();
```

```
7.    expect(binding('value1')).toEqual('false');
8.    expect(binding('value2')).toEqual('NO');
9.  });
```

input [email]

Описание

Текстовый элемент с проверкой правильности ввода email адреса. Устанавливает ключ `email` в коллекции ошибок, если значение ввода не является правильным адресом email.

Использование

```
1.  <input type="email"
2.         ng-model="{string}"
3.         [name="{string}"]
4.         [required]
5.         [ng-required="{string}"]
6.         [ng-minlength="{number}"]
7.         [ng-maxlength="{number}"]
8.         [ng-pattern="{string}"]>
```

Параметры

- `ngModel – {string} –`

  Ассоциированное angular выражение для привязки данных.

- `name(optional) – {string=} –`

  Имя свойства, под которым элемент управления будет доступен в области видимости.

- `required(optional) – {string=} –`

  Устанавливает ключ `required` в ошибках проверки данных, когда у поля отсутствует значение.

- `ngRequired(optional) – {boolean=} –`

  Устанавливает атрибут `required`, если выражение возвращает true

- `ngMinlength(optional) – {number=} –`

  Устанавливает ключ `minlength` в ошибках проверки данных, если значение короче, чем установленное количество знаков.

- `ngMaxlength(optional) – {number=} –`

  Устанавливает ключ `maxlength` в ошибках проверки данных, если значение длиннее, чем установленное количество знаков.

- `ngPattern(optional) – {string=} –`

  Устанавливает ключ `pattern` в ошибках проверки данных, если ввод не соответствует шаблону RegExp. Значение сравнивается с переданным паттерном, если он передается в формате `/regexp/`, в других случаях считается что передано свойство текущей области видимости, которое содержит нужный паттерн.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <form name="myForm" ng-controller="Ctrl">
      Email: <input type="email" name="input" ng-model="text" required>
      <span class="error" ng-show="myForm.input.$error.required">
        Required!</span>
      <span class="error" ng-show="myForm.input.$error.email">
        Not valid email!</span>
      <tt>text = {{text}}</tt><br/>
      <tt>myForm.input.$valid = {{myForm.input.$valid}}</tt><br/>
      <tt>myForm.input.$error = {{myForm.input.$error}}</tt><br/>
      <tt>myForm.$valid = {{myForm.$valid}}</tt><br/>
      <tt>myForm.$error.required = {{!!myForm.$error.required}}</tt><br/>
      <tt>myForm.$error.email = {{!!myForm.$error.email}}</tt><br/>
    </form>
  </body>
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.text = 'me@example.com';
}
```

End to end test

```
it('should initialize to model', function() {
  expect(binding('text')).toEqual('me@example.com');
  expect(binding('myForm.input.$valid')).toEqual('true');
});

it('should be invalid if empty', function() {
  input('text').enter('');
  expect(binding('text')).toEqual('');
```

```
9.    expect(binding('myForm.input.$valid')).toEqual('false');
10. });
11.
12. it('should be invalid if not email', function() {
13.   input('text').enter('xxx');
14.   expect(binding('myForm.input.$valid')).toEqual('false');
15. });
```

input [number]

Описание

Текстовый элемент для ввода чисел с проверкой и трансформациями. Устанавливает ключ `number` в ошибках проверки данных, если введенное значение не является числом.

Использование

```
1.  <input type="number"
2.         ng-model="{string}"
3.         [name="{string}"]
4.         [min="{string}"]
5.         [max="{string}"]
6.         [required]
7.         [ng-required="{string}"]
8.         [ng-minlength="{number}"]
9.         [ng-maxlength="{number}"]
10.        [ng-pattern="{string}"]
11.        [ng-change="{string}"]>
```

Параметры

- `ngModel – {string} –`

  Ассоциированное angular выражение для привязки данных.

- `name`*`(optional)`* `– {string=} –`

  Имя свойства, под которым элемент управления будет доступен в области видимости.

- `min`*`(optional)`* `– {string=} –`

  Устанавливает ключ `min` в ошибках проверки данных, если введенное значение меньше `min`.

- `max`*`(optional)`* `– {string=} –`

  Устанавливает ключ `max` в ошибках проверки данных, если введенное значение больше `max`.

- `required`*`(optional)`* `– {string=} –`

  Устанавливает ключ `required` в ошибках проверки данных, когда у поля отсутствует значение.

- `ngRequired(optional) – {boolean=} –`

  Устанавливает атрибут `required`, если выражение возвращает true

- `ngMinlength(optional) – {number=} –`

  Устанавливает ключ `minlength` в ошибках проверки данных, если значение короче, чем установленное количество знаков.

- `ngMaxlength(optional) – {number=} –`

  Устанавливает ключ `maxlength` в ошибках проверки данных, если значение длиннее, чем установленное количество знаков.

- `ngPattern(optional) – {string=} –`

  Устанавливает ключ `pattern` в ошибках проверки данных, если ввод не соответствует шаблону RegExp. Значение сравнивается с переданным паттерном, если он передается в формате `/regexp/`, в других случаях считается что передано свойство текущей области видимости, которое содержит нужный паттерн.

- `ngChange(optional) – {string=} –`

  Angular выражение, которое будет выполнено, когда измениться содержимое элемента управления в результате взаимодействия с пользователем.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <form name="myForm" ng-controller="Ctrl">
      Number: <input type="number" name="input" ng-model="value"
                     min="0" max="99" required>
      <span class="error" ng-show="myForm.list.$error.required">
        Required!</span>
      <span class="error" ng-show="myForm.list.$error.number">
        Not valid number!</span>
      <tt>value = {{value}}</tt><br/>
      <tt>myForm.input.$valid = {{myForm.input.$valid}}</tt><br/>
      <tt>myForm.input.$error = {{myForm.input.$error}}</tt><br/>
      <tt>myForm.$valid = {{myForm.$valid}}</tt><br/>
      <tt>myForm.$error.required = {{!!myForm.$error.required}}</tt><br/>
    </form>
  </body>
```

```
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.value = 12;
}
```

End to end test

```
it('should initialize to model', function() {
 expect(binding('value')).toEqual('12');
 expect(binding('myForm.input.$valid')).toEqual('true');
});

it('should be invalid if empty', function() {
 input('value').enter('');
 expect(binding('value')).toEqual('');
 expect(binding('myForm.input.$valid')).toEqual('false');
});

it('should be invalid if over max', function() {
 input('value').enter('123');
 expect(binding('value')).toEqual('');
 expect(binding('myForm.input.$valid')).toEqual('false');
});
```

input [radio]

Описание

HTML радио кнопки.

Использование

```
<input type="radio"
       ng-model="{string}"
       value="{string}"
       [name="{string}"]
       [ng-change="{string}"]>
```

Параметры

- `ngModel – {string} –`

  Ассоциированное angular выражение для привязки данных.

- `name(optional) – {string=} –`

Имя свойства, под которым элемент управления будет доступен в области видимости.

- `value – {string} –`

  Значение, которое получит привязанная модель, после выбора текущей радио кнопки.

- `ngChange(optional) – {string=} –`

  Angular выражение, которое будет выполнено, когда измениться содержимое элемента управления в результате взаимодействия с пользователем.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <form name="myForm" ng-controller="Ctrl">
      <input type="radio" ng-model="color" value="red">  Red <br/>
      <input type="radio" ng-model="color" value="green"> Green <br/>
      <input type="radio" ng-model="color" value="blue"> Blue <br/>
      <tt>color = {{color}}</tt><br/>
     </form>
  </body>
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.color = 'blue';
}
```

End to end test

```
it('should change state', function() {
  expect(binding('color')).toEqual('blue');

  input('color').select('red');
  expect(binding('color')).toEqual('red');
});
```

input [text]

Описание

Стандартный элемент ввода текста HTML с поддержкой привязки данных angular.

Использование

```
<input type="text"
       ng-model="{string}"
       [name="{string}"]
       [required]
       [ng-required="{string}"]
       [ng-minlength="{number}"]
       [ng-maxlength="{number}"]
       [ng-pattern="{string}"]
       [ng-change="{string}"]>
```

Параметры

- `ngModel – {string} –`

  Ассоциированное angular выражение для привязки данных.

- `name(optional) – {string=} –`

  Имя свойства, под которым элемент управления будет доступен в области видимости.

- `required(optional) – {string=} –`

  Устанавливает ключ `required` в ошибках проверки данных, когда у поля отсутствует значение.

- `ngRequired(optional) – {boolean=} –`

  Устанавливает атрибут `required`, если выражение возвращает true

- `ngMinlength(optional) – {number=} –`

  Устанавливает ключ `minlength` в ошибках проверки данных, если значение короче, чем установленное количество знаков.

- `ngMaxlength(optional) – {number=} –`

  Устанавливает ключ `maxlength` в ошибках проверки данных, если значение длиннее, чем установленное количество знаков.

- `ngPattern(optional) – {string=} –`

  Устанавливает ключ `pattern` в ошибках проверки данных, если ввод не соответствует шаблону RegExp. Значение сравнивается с переданным паттерном, если он передается в

формате `/regexp/`, в других случаях считается что передано свойство текущей области видимости, которое содержит нужный паттерн.

- `ngChange(optional) – {string=} –`

  Angular выражение, которое будет выполнено, когда измениться содержимое элемента управления в результате взаимодействия с пользователем.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <form name="myForm" ng-controller="Ctrl">
      Single word: <input type="text" name="input" ng-model="text"
                          ng-pattern="word" required>
      <span class="error" ng-show="myForm.input.$error.required">
        Required!</span>
      <span class="error" ng-show="myForm.input.$error.pattern">
        Single word only!</span>

      <tt>text = {{text}}</tt><br/>
      <tt>myForm.input.$valid = {{myForm.input.$valid}}</tt><br/>
      <tt>myForm.input.$error = {{myForm.input.$error}}</tt><br/>
      <tt>myForm.$valid = {{myForm.$valid}}</tt><br/>
      <tt>myForm.$error.required = {{!!myForm.$error.required}}</tt><br/>
    </form>
  </body>
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.text = 'guest';
  $scope.word = /^\w*$/;
}
```

End to end test

```
it('should initialize to model', function() {
  expect(binding('text')).toEqual('guest');
  expect(binding('myForm.input.$valid')).toEqual('true');
});

```

```
6.  it('should be invalid if empty', function() {
7.    input('text').enter('');
8.    expect(binding('text')).toEqual('');
9.    expect(binding('myForm.input.$valid')).toEqual('false');
10. });
11.
12. it('should be invalid if multi word', function() {
13.   input('text').enter('hello world');
14.   expect(binding('myForm.input.$valid')).toEqual('false');
15. });
```

input [url]

Описание

Элемент для ввода текста с проверкой правильности URL. Устанавливает ключ `url` в ошибках проверки данных, если введенные данные не являются правильным URL.

Использование

```
1.  <input type="url"
2.         ng-model="{string}"
3.         [name="{string}"]
4.         [required]
5.         [ng-required="{string}"]
6.         [ng-minlength="{number}"]
7.         [ng-maxlength="{number}"]
8.         [ng-pattern="{string}"]
9.         [ng-change="{string}"]>
```

Параметры

- `ngModel – {string} –`

  Ассоциированное angular выражение для привязки данных.

- `name`*`(optional)`* `– {string=} –`

  Имя свойства, под которым элемент управления будет доступен в области видимости.

- `required`*`(optional)`* `– {string=} –`

  Устанавливает ключ `required` в ошибках проверки данных, когда у поля отсутствует значение.

- `ngRequired`*`(optional)`* `– {boolean=} –`

  Устанавливает атрибут `required`, если выражение возвращает true

- `ngMinlength`*`(optional)`* `– {number=} –`

Устанавливает ключ `minlength` в ошибках проверки данных, если значение короче, чем установленное количество знаков.

- `ngMaxlength(optional) – {number=} –`

  Устанавливает ключ `maxlength` в ошибках проверки данных, если значение длиннее, чем установленное количество знаков.

- `ngPattern(optional) – {string=} –`

  Устанавливает ключ `pattern` в ошибках проверки данных, если ввод не соответствует шаблону RegExp. Значение сравнивается с переданным паттерном, если он передается в формате `/regexp/`, в других случаях считается что передано свойство текущей области видимости, которое содержит нужный паттерн.

- `ngChange(optional) – {string=} –`

  Angular выражение, которое будет выполнено, когда измениться содержимое элемента управления в результате взаимодействия с пользователем.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <form name="myForm" ng-controller="Ctrl">
      URL: <input type="url" name="input" ng-model="text" required>
      <span class="error" ng-show="myForm.input.$error.required">
        Required!</span>
      <span class="error" ng-show="myForm.input.$error.url">
        Not valid url!</span>
      <tt>text = {{text}}</tt><br/>
      <tt>myForm.input.$valid = {{myForm.input.$valid}}</tt><br/>
      <tt>myForm.input.$error = {{myForm.input.$error}}</tt><br/>
      <tt>myForm.$valid = {{myForm.$valid}}</tt><br/>
      <tt>myForm.$error.required = {{!!myForm.$error.required}}</tt><br/>
      <tt>myForm.$error.url = {{!!myForm.$error.url}}</tt><br/>
    </form>
  </body>
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.text = 'http://google.com';
```

```
}
```

End to end test

```
it('should initialize to model', function() {
  expect(binding('text')).toEqual('http://google.com');
  expect(binding('myForm.input.$valid')).toEqual('true');
});

it('should be invalid if empty', function() {
  input('text').enter('');
  expect(binding('text')).toEqual('');
  expect(binding('myForm.input.$valid')).toEqual('false');
});

it('should be invalid if not url', function() {
  input('text').enter('xxx');
  expect(binding('myForm.input.$valid')).toEqual('false');
});
```

ngApp

Описание

Используйте эту директиву для автоматической инициализации приложения. Только один раз ее можно использовать в документе HTML. Эта директива является корнем вашего приложения, и обычно размещается в корневом элементе страницы.

В примере ниже, если директива `ngApp` не будет размещена в элементе `html`, тогда не будет проведена компиляция документа и выражение `{{ 1+2 }}` не будет разрешено в 3.

`ngApp` это легкий способ инициализации вашего приложения.

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
  <body>
    I can add: 1 + 2 =  {{ 1+2 }}
  </body>
</html>
```

Использование

как атрибут

```
<ANY ng-app="{angular.Module}">
   ...
```

```
3. </ANY>
```

как класс

```
1. <ANY class="ng-app: {angular.Module};">
2.    ...
3. </ANY>
```

Параметры

- `ngApp – {angular.Module} –` не обязательный параметр, который задает имя модуля для загрузки.

ngBind

Описание

Атрибут `ngBind` говорит Angular, что нужно поместить в текстовый элемент значение заданного выражения, и обновлять его содержимое, если измениться значение выражения.

Обычно, вы не будете использовать `ngBind` явно, а вместо этого вы будете использовать выражения в двойных фигурных скобках, которые имитируют предыдущую возможность.

Однако первый сценарий предпочтительнее, т.к. использование `ngBind`, в отличие от привязки с помощью `{{ expression }}` не выводится в браузер перед компиляцией. Так как `ngBind` является атрибутом элемента, то он не отражается в браузере, и делает привязку не видимой для пользователя во время загрузки.

Альтернативным решением проблемы является использование директивы `ngCloak` .

Использование

как атрибут

```
1. <ANY ng-bind="{expression}">
2.    ...
3. </ANY>
```

как класс

```
1. <ANY class="ng-bind: {expression};">
2.    ...
3. </ANY>
```

Параметры

- `ngBind – {expression} –` выражение для вычисления.

Пример

Введите имя текстовое поле; и увидите, как изменится приветственная надпись.

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-controller="Ctrl">
      Enter name: <input type="text" ng-model="name"><br>
      Hello <span ng-bind="name"></span>!
    </div>
  </body>
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.name = 'Whirled';
}
```

End to end test

```
it('should check ng-bind', function() {
  expect(using('.doc-example-live').binding('name')).toBe('Whirled');
  using('.doc-example-live').input('name').enter('world');
  expect(using('.doc-example-live').binding('name')).toBe('world');
});
```

ngBindHtmlUnsafe

Описание

Создает привязку через свойство innerHTML к результату выполнения выражения `expression` в текущем элементе. *Содержимое innerHTML может быть содержать недопустимые знаки!* Эту директиву следует использовать только тогда, когда вас не устраивает директива ngBindHtml, и вы абсолютно доверяете источнику привязки.

См. Документацию по $sanitize для примера.

Использование

как атрибута

```
<ANY ng-bind-html-unsafe="{expression}">
    ...
</ANY>
```

как класс

```
<ANY class="ng-bind-html-unsafe: {expression};">
    ...
</ANY>
```

Параметры

- `ngBindHtmlUnsafe - {expression} -` Выражение для вычисления.

ngBindTemplate

Описание

Директива `ngBindTemplate` указывает, что текст в элементе должен быть заменен на результат привязки ngBindTemplate. В отличие от директивы ngBind, директива ngBindTemplate может содержать внутри множество выражений привязки данных `{{ }}` . (Это требуется, так как некоторые элементы HTML не могут содержать внутри элементы SPAN, такие как TITLE, или OPTION).

Использование

как атрибут

```
<ANY ng-bind-template="{string}">
    ...
</ANY>
```

как класс

```
<ANY class="ng-bind-template: {string};">
    ...
</ANY>
```

Параметры

- `ngBindTemplate - {string} -` шаблон в форме `{{ expression }}` для вычисления.

Пример

Введите текст в текстовые элементы и посмотрите что поменяется.

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-controller="Ctrl">
     Salutation: <input type="text" ng-model="salutation"><br>
     Name: <input type="text" ng-model="name"><br>
```

```
    <pre ng-bind-template="{{salutation}} {{name}}!"></pre>
   </div>
  </body>
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.salutation = 'Hello';
  $scope.name = 'World';
}
```

End to end test

```
it('should check ng-bind', function() {
  expect(using('.doc-example-live').binding('salutation')).
    toBe('Hello');
  expect(using('.doc-example-live').binding('name')).
    toBe('World');
  using('.doc-example-live').input('salutation').enter('Greetings');
  using('.doc-example-live').input('name').enter('user');
  expect(using('.doc-example-live').binding('salutation')).
    toBe('Greetings');
  expect(using('.doc-example-live').binding('name')).
    toBe('user');
});
```

ngChange

Описание

Вызывает определенную функцию, если пользователь изменил значение элемента ввода. Функция не вызывается, если изменение исходит от модели.

Заметьте, эта директива требует наличия директивы `ngModel` в том же элементе.

Использование

Эта директива может использоваться как дополнительный элемент, но с учетом ограничений IE:

```
<ng-change>
</ng-change>
```

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
```

```
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-controller="Controller">
      <input type="checkbox" ng-model="confirmed" ng-change="change()" id="ng-chan
ge-example1" />
      <input type="checkbox" ng-model="confirmed" id="ng-change-example2" />
      <label for="ng-change-example2">Confirmed</label><br />
      debug = {{confirmed}}<br />
      counter = {{counter}}
    </div>
  </body>
</html>
```

Script.js

```
function Controller($scope) {
  $scope.counter = 0;
  $scope.change = function() {
    $scope.counter++;
  };
}
```

End to end test

```
it('should evaluate the expression if changing from view', function() {
  expect(binding('counter')).toEqual('0');
  element('#ng-change-example1').click();
  expect(binding('counter')).toEqual('1');
  expect(binding('confirmed')).toEqual('true');
});

it('should not evaluate the expression if changing from model', function() {
  element('#ng-change-example2').click();
  expect(binding('counter')).toEqual('0');
  expect(binding('confirmed')).toEqual('true');
});
```

ngChecked

Описание

HTML спецификация не трубует от браузеров сохранять особые атрибуты, такие как checked. (Наличие их означает true, а отсутствие false). Это мешает компилятору angular правильно вычислять выражения привязки. Чтобы решить эту проблему и ввели директиву `ngChecked`.

Использование

как атрибут

```
<INPUT ng-checked="{expression}">
   ...
</INPUT>
```

Параметры

- `ngChecked – {expression}` – Angular выражение для вычисления.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
  <body>
    Check me to check both: <input type="checkbox" ng-model="master"><br/>
    <input id="checkSlave" type="checkbox" ng-checked="master">
  </body>
</html>
```

End to end test

```
it('should check both checkBoxes', function() {
  expect(element('.doc-example-live #checkSlave').prop('checked')).toBeFalsy();
  input('master').check();
  expect(element('.doc-example-live #checkSlave').prop('checked')).toBeTruthy();
});
```

ngClass

Описание

Директива `ngClass` позволяет вам установить класс CSS для HTML элементов динамически, путем привязки данных и переопределять их при изменении.

Эта директива не будет добавлять класс, если он уже установлен.

Когда значение выражения изменится, ранее добавленные классы будут удалены, а затем будут установлены новые.

Использование

как атрибут

```
<ANY ng-class="{expression}">
   ...
</ANY>
```

как класс

```
<ANY class="ng-class: {expression};">
   ...
</ANY>
```

Параметры

- `ngClass - {expression} -` Выражение для вычисления. Результатом может быть строка с именами классов, разделенными пробелами, или массив с именами классов, или объект, со свойствами именами классов и логическими значениями.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
  <body>
    <input type="button" value="set" ng-click="myVar='my-class'">
    <input type="button" value="clear" ng-click="myVar=''">
    <br>
    <span ng-class="myVar">Sample Text</span>
  </body>
</html>
```

Style.css

```
.my-class {
  color: red;
}
```

End to end test

```
it('should check ng-class', function() {
  expect(element('.doc-example-live span').prop('className')).not().
    toMatch(/my-class/);

  using('.doc-example-live').element(':button:first').click();

  expect(element('.doc-example-live span').prop('className')).
    toMatch(/my-class/);

  using('.doc-example-live').element(':button:last').click();

  expect(element('.doc-example-live span').prop('className')).not().
    toMatch(/my-class/);
});
```

ngClassEven

Описание

Директивы `ngClassOdd` и `ngClassEven` работают так же, как `ngClass`, исключая то, что их можно использовать только с директивой `ngRepeat`, и они добавляют классы только соответственно для не четных (четных) строк.

Эта директива может применяться только с областью данных `ngRepeat`.

Использование

как атрибут

```
<ANY ng-class-even="{expression}">
   ...
</ANY>
```

как класс

```
<ANY class="ng-class-even: {expression};">
   ...
</ANY>
```

Параметры

- `ngClassEven – {expression} –` Выражение для вычисления. Результатом может быть строка с именами классов, разделенными пробелами, или массив имен классов.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
  <body>
    <ol ng-init="names=['John', 'Mary', 'Cate', 'Suz']">
      <li ng-repeat="name in names">
       <span ng-class-odd="'odd'" ng-class-even="'even'">
         {{name}}      
       </span>
      </li>
    </ol>
  </body>
</html>
```

Style.css

```
.odd {
```

```
  color: red;
}
.even {
  color: blue;
}
```

End to end test

```
it('should check ng-class-odd and ng-class-even', function() {
  expect(element('.doc-example-live li:first span').prop('className')).
    toMatch(/odd/);
  expect(element('.doc-example-live li:last span').prop('className')).
    toMatch(/even/);
});
```

ngClassOdd

Описание

Директивы `ngClassOdd` и `ngClassEven` работают так же, как `ngClass`, исключая то, что их можно использовать только с директивой `ngRepeat`, и они добавляют классы только соответственно для не четных (четных) строк.

Эта директива может применяться только с областью данных `ngRepeat`.

Использование

как атрибут

```
<ANY ng-class-odd="{expression}">
  ...
</ANY>
```

как класс

```
<ANY class="ng-class-odd: {expression};">
  ...
</ANY>
```

Параметры

- `ngClassOdd – {expression} –` Выражение для вычисления. Результатом может быть строка с именами классов, разделенными пробелами, или массив имен классов.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
```

```
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
  <body>
    <ol ng-init="names=['John', 'Mary', 'Cate', 'Suz']">
      <li ng-repeat="name in names">
       <span ng-class-odd="'odd'" ng-class-even="'even'">
         {{name}}      
       </span>
      </li>
    </ol>
  </body>
</html>
```

Style.css

```
.odd {
  color: red;
}
.even {
  color: blue;
}
```

End to end test

```
it('should check ng-class-odd and ng-class-even', function() {
  expect(element('.doc-example-live li:first span').prop('className')).
    toMatch(/odd/);
  expect(element('.doc-example-live li:last span').prop('className')).
    toMatch(/even/);
});
```

ngClick

Описание

Директива ngClick позволяет вам указать свое поведение, которое будет выполнено, когда произойдет клик по элементу.

Использование

как атрибут

```
<ANY ng-click="{expression}">
    ...
</ANY>
```

как класс

```
<ANY class="ng-click: {expression};">
```

```
    ...
</ANY>
```

Параметры

- `ngClick – {expression} –` Выражение для вычисления после щелчка мышью. (Объект события доступен через `$event`)

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
  <body>
    <button ng-click="count = count + 1" ng-init="count=0">
      Increment
    </button>
    count: {{count}}
  </body>
</html>
```

End to end test

```
it('should check ng-click', function() {
  expect(binding('count')).toBe('0');
  element('.doc-example-live :button').click();
  expect(binding('count')).toBe('1');
});
```

ngCloak

Описание

Директива `ngCloak` используется для предотвращения показа в браузере шаблона Angular при загрузке приложения. Используйте эту директиву, чтобы избежать нежелательного эффекта мерцания, вызванного кратковременным отображением html шаблона.

Она может быть применена к элементу <body>, но обычно детализированное управление позволяет извлечь большую выгоду, при отображении представлений в окне браузера.

`ngCloak` работает совместно с стилями css которые определены в файлах `angular.js` и `angular.min.js`. Вот этот css стиль:

```
[ng\:cloak], [ng-cloak], [data-ng-cloak], [x-ng-cloak], .ng-cloak, .x-ng-cloak {
  display: none;
}
```

Когда стиль css будет загружен браузером, все html элементы (включая дочерние) внутри элемента с директивой `ng-cloak` будут скрыты. Когда Angular найдет все директивы, и выполнит компиляцию шаблона, он удалит из элемента атрибут `ngCloak`, что сделает скомпилированные элементы видимыми.

Для лучшего результата скрипт `angular.js` должен быть загружен в секции head вашего html файла; альтернативный подход, поместить определение стиля css во включаемую внешнюю таблицу стилей для приложения.

Старые браузеры, как IE7, не поддерживают селекторы атрибутов (добавлены в CSS 2.1) поэтому они не поймут селектор`[ng\:cloak]`. Чтобы обойти это ограничение, необходимо добавить css класс `ngCloak` в дополнение к директиве `ngCloak`, как это показано в примере ниже.

Использование

как атрибут

```
<ANY ng-cloak>
   ...
</ANY>
```

как класс

```
<ANY class="ng-cloak">
   ...
</ANY>
```

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
  <body>
    <div id="template1" ng-cloak>{{ 'hello' }}</div>
    <div id="template2" ng-cloak class="ng-cloak">{{ 'hello IE7' }}</div>
  </body>
</html>
```

End to end test

```
it('should remove the template directive and css class', function() {
  expect(element('.doc-example-live #template1').attr('ng-cloak')).
    not().toBeDefined();
  expect(element('.doc-example-live #template2').attr('ng-cloak')).
    not().toBeDefined();
});
```

ngController

Описание

Директива `ngController` присваивает поведение области видимости. Это ключевой аспект поддержки в angular дизайнерского паттерна Model-View-Controller.

MVC компоненты в angular:

- Model — модель, это данные из свойств области видимости; области видимости присоединены к DOM.
- View — шаблон (HTML с привязками данных) из которого генерируется представление.
- Controller — директива `ngController` специализирующая класс контроллера; этот класс имеет методы и типичные выражения для отражения бизнес логики вашего приложения.

Обратите внимание, на альтернативную возможность определения контроллера, используя сервис `$route`.

Использование

как атрибут

```
<ANY ng-controller="{expression}">
   ...
</ANY>
```

как класс

```
<ANY class="ng-controller: {expression};">
   ...
</ANY>
```

Информация о директиве

- Эта директива создает новую область видимости.

Параметры

- `ngController - {expression} -` Имя глобально доступной функции конструктора или выражение в текущей области видимости, которое возвращает функцию конструктор.

Пример

Здесь показана простая форма для редактирования контактной информации пользователя. Добавление, удаление, очистка и вывод в окне реализованы с помощью методов в контроллере. Их можно легко вызвать из Angular окружения. Обратите внимание, что внутри контроллера указатель `this` ссылается на текущую область видимости. Это позволяет организовать легкий доступ к данным контроллера из представления. Так же уведомлять об изменении данных и автоматически отображать их на представление.

Index.html

```
<!doctype html>
<html ng-app>
```

```
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-controller="SettingsController">
      Name: <input type="text" ng-model="name"/>
      [ <a href="" ng-click="greet()">greet</a> ]<br/>
      Contact:
      <ul>
        <li ng-repeat="contact in contacts">
          <select ng-model="contact.type">
             <option>phone</option>
             <option>email</option>
          </select>
          <input type="text" ng-model="contact.value"/>
          [ <a href="" ng-click="clearContact(contact)">clear</a>
          | <a href="" ng-click="removeContact(contact)">X</a> ]
        </li>
        <li>[ <a href="" ng-click="addContact()">add</a> ]</li>
     </ul>
    </div>
  </body>
</html>
```

Script.js

```
function SettingsController($scope) {
  $scope.name = "John Smith";
  $scope.contacts = [
    {type:'phone', value:'408 555 1212'},
    {type:'email', value:'john.smith@example.org'} ];

  $scope.greet = function() {
   alert(this.name);
  };

  $scope.addContact = function() {
   this.contacts.push({type:'email', value:'yourname@example.org'});
  };

  $scope.removeContact = function(contactToRemove) {
   var index = this.contacts.indexOf(contactToRemove);
   this.contacts.splice(index, 1);
  };

  $scope.clearContact = function(contact) {
   contact.type = 'phone';
   contact.value = '';
  };
}
```

End to end test

```
it('should check controller', function() {
  expect(element('.doc-example-live div>:input').val()).toBe('John Smith');
  expect(element('.doc-example-live li:nth-child(1) input').val())
    .toBe('408 555 1212');
  expect(element('.doc-example-live li:nth-child(2) input').val())
    .toBe('john.smith@example.org');

  element('.doc-example-live li:first a:contains("clear")').click();
  expect(element('.doc-example-live li:first input').val()).toBe('');

  element('.doc-example-live li:last a:contains("add")').click();
  expect(element('.doc-example-live li:nth-child(3) input').val())
    .toBe('yourname@example.org');
});
```

ngCsp

Описание

Включает поддержку CSP (Content Security Policy). Эта директива должна быть использована в корневом элементе вашего приложения (типично это `<html>` элемент или другой элемент, который содержит директиву ngApp).

Если его включить, производительность шаблонов незначительно пострадает, поэтому не включайте его без крайней необходимости.

Использование

как атрибут

```
<html ng-csp>
   ...
</html>
```

как класс

```
<html class="ng-csp">
   ...
</html>
```

Информация о директиве

- Эта директива выполняется с уровнем приоритета 1000.

ngDblclick

Описание

Директива `ngDblclick` позволяет вам указать пользовательскую функцию для выполнения при возникновении события dblclick.

Использование

как атрибут

```
<ANY ng-dblclick="{expression}">
    ...
</ANY>
```

как класс

```
<ANY class="ng-dblclick: {expression};">
    ...
</ANY>
```

Параметры

- `ngDblclick – {expression} –` Выражение для вычисления после dblclick. (Объект события доступен через параметр `$event`)

Пример

См. `ngClick`

ngDisabled

Описание

Следующая разметка включает кнопку в Chrome/Firefox, но не в IE8 и более старых IE:

```
<div ng-init="scope = { isDisabled: false }">
 <button disabled="{{scope.isDisabled}}">Disabled</button>
</div>
```

Спецификация HTML не требует от браузеров обязательного наличия специальных атрибутов, таких как disabled. (Их наличие означает true, а отсутствие false) Это мешает компилятору angular правильно обрабатывать выражения привязки. Для решения этой проблемы, мы и разработали директиву `ngDisabled`.

Использование

как атрибут

```
<INPUT ng-disabled="{expression}">
    ...
</INPUT>
```

Параметры

- `ngDisabled – {expression} –` Angular выражение для вычисления.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
  <body>
    Click me to toggle: <input type="checkbox" ng-model="checked"><br/>
    <button ng-model="button" ng-disabled="checked">Button</button>
  </body>
</html>
```

End to end test

```
it('should toggle button', function() {
  expect(element('.doc-example-live :button').prop('disabled')).toBeFalsy();
  input('checked').check();
  expect(element('.doc-example-live :button').prop('disabled')).toBeTruthy();
});
```

ngForm

Описание

Псевдоним для директивы `form` . HTML не позволяет вкладывать элементы <form> друг в друга. Эта директива используется для создания вложенных форм, например, если вы хотите проверять отдельно корректность данных.

Использование

Эта директива может использоваться как дополнительный элемент, но с учетом ограничений IE:

```
<ng-form
       [ngForm="{string}"]>
</ng-form>
```

как атрибут

```
<ANY ng-form
     [name="{string}"]>
   ...
</ANY>
```

как класс

```
<ANY class="ng-form [name: {string};]">
```

```
    ...
</ANY>
```

Параметры

- `name|ngForm(optional) – {string=} –` Имя формы. Если оно указанно, контроллер формы опубликует одноименное свойство внутри области видимости, которое будет содержать ссылку на форму.

ngHide

Описание

Директивы `ngHide` и `ngShow` соответственно скрывают или показывают часть дерева DOM.

Использование

как атрибут

```
<ANY ng-hide="{expression}">
    ...
</ANY>
```

как класс

```
<ANY class="ng-hide: {expression};">
    ...
</ANY>
```

Параметры

- `ngHide – {expression} –` Если результат вычисления выражения true, тогда элемент скрывается, иначе отображается.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
  <body>
    Click me: <input type="checkbox" ng-model="checked"><br/>
    Show: <span ng-show="checked">I show up when you checkbox is checked?</span> <
br/>
    Hide: <span ng-hide="checked">I hide when you checkbox is checked?</span>
  </body>
</html>
```

End to end test

```
it('should check ng-show / ng-hide', function() {
  expect(element('.doc-example-live span:first:hidden').count()).toEqual(1);
  expect(element('.doc-example-live span:last:visible').count()).toEqual(1);

  input('checked').check();

  expect(element('.doc-example-live span:first:visible').count()).toEqual(1);
  expect(element('.doc-example-live span:last:hidden').count()).toEqual(1);
});
```

ngHref

Описание

Использование в внутри атрибута href выражений Angular приведет к открытию неправильно адреса URL, если пользователь кликнет на ссылке, перед тем как angular заменит ее на правильную (это произойдет в фазе связывания), что скорее всего приведет к 404 ошибке. Директива `ngHref` решает эту проблему.

Так писать не нужно:

```
<a href="http://www.gravatar.com/avatar/{{hash}}"/>
```

Пишите вот так:

```
<a ng-href="http://www.gravatar.com/avatar/{{hash}}"/>
```

Использование

как атрибут

```
<A ng-href="{template}">
   ...
</A>
```

Параметры

- `ngHref – {template} –` любая строка, которая может содержать выражения в `{{}}`.

Пример

В этом примере используется переменная `link` внутри атрибута `href`:

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
```

```
  <body>
    <input ng-model="value" /><br />
    <a id="link-1" href ng-click="value = 1">link 1</a> (link, don't reload)<br />
    <a id="link-2" href="" ng-click="value = 2">link 2</a> (link, don't reload)<br />
    <a id="link-3" ng-href="/{{'123'}}">link 3</a> (link, reload!)<br />
    <a id="link-4" href="" name="xx" ng-click="value = 4">anchor</a> (link, don't reload)<br />
    <a id="link-5" name="xxx" ng-click="value = 5">anchor</a> (no link)<br />
    <a id="link-6" ng-href="{{value}}">link</a> (link, change location)
  </body>
</html>
```

End to end test

```
it('should execute ng-click but not reload when href without value', function() {
  element('#link-1').click();
  expect(input('value').val()).toEqual('1');
  expect(element('#link-1').attr('href')).toBe("");
});

it('should execute ng-click but not reload when href empty string', function() {
  element('#link-2').click();
  expect(input('value').val()).toEqual('2');
  expect(element('#link-2').attr('href')).toBe("");
});

it('should execute ng-click and change url when ng-href specified', function() {
  expect(element('#link-3').attr('href')).toBe("/123");

  element('#link-3').click();
  expect(browser().window().path()).toEqual('/123');
});

it('should execute ng-click but not reload when href empty string and name specified', function() {
  element('#link-4').click();
  expect(input('value').val()).toEqual('4');
  expect(element('#link-4').attr('href')).toBe('');
});

it('should execute ng-click but not reload when no href but name specified', function() {
  element('#link-5').click();
  expect(input('value').val()).toEqual('5');
  expect(element('#link-5').attr('href')).toBe(undefined);
});

it('should only change url when only ng-href', function() {
  input('value').enter('6');
  expect(element('#link-6').attr('href')).toBe('6');

```

```
36.   element('#link-6').click();
37.   expect(browser().location().url()).toEqual('/6');
38. });
```

ngInclude

Описание

Извлекает, компилирует и включает внешний HTML фрагмент.

Имейте в виду ограничения политики происхождения из одного источника (Same Origin Policy) при включении ресурсов (к примеру, ngInclude не работает с кросс-доменными запросами во всех браузерах и также не может использоваться для доступа к файлам на компьютере пользователя через file://).

Использование

Эта директива может использоваться как дополнительный элемент, но с учетом ограничений IE:

```
1. <ng-include
2.         src="{string}"
3.         [onload="{string}"]
4.         [autoscroll="{string}"]>
5. </ng-include>
```

как атрибут

```
1. <ANY ng-include="{string}"
2.      [onload="{string}"]
3.      [autoscroll="{string}"]>
4.    ...
5. </ANY>
```

как класс

```
1. <ANY class="ng-include: {string}; [onload: {string};] [autoscroll: {string};]">
2.    ...
3. </ANY>
```

Информация о директиве

- Эта директива создает новую область видимости.

Параметры

- `ngInclude|src – {string} –` angular выражение, возвращающее URL. Если хотите передать строку, тогда возьмите ее в кавычки, например `src="'myPartialTemplate.html'"`.
- `onload(optional) – {string=} –` Выражение, которое выполнится, когда новая часть будет загружена.

- `autoscroll(optional)` – {string=} – Вслед за `ngInclude` следует вызвать `$anchorScroll` для прокрутки отображения к загруженному контенту.
  - Если атрибут не установлен, скроллинг отключается.
  - Если атрибут содержит значение, скроллинг включается.
  - В противном случае скроллинг включается, только если выражение возвращает значение true.

События

$includeContentLoaded

Возбуждается постоянно после загрузки контента ngInclude.

Тип: всплывающее

Цель: текущая область видимости

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-controller="Ctrl">
      <select ng-model="template" ng-options="t.name for t in templates">
       <option value="">(blank)</option>
      </select>
      url of the template: <tt>{{template.url}}</tt>
      <hr/>
      <div ng-include src="template.url"></div>
    </div>
  </body>
</html>
```

Template1.html

```
Content of template1.html
```

Template2.html

```
Content of template2.html
```

Script.js

```
function Ctrl($scope) {
```

```
  $scope.templates =
    [ { name: 'template1.html', url: 'template1.html'}
    , { name: 'template2.html', url: 'template2.html'} ];
  $scope.template = $scope.templates[0];
}
```

End to end test

```
it('should load template1.html', function() {
 expect(element('.doc-example-live [ng-include]').text()).
   toMatch(/Content of template1.html/);
});
it('should load template2.html', function() {
 select('template').option('1');
 expect(element('.doc-example-live [ng-include]').text()).
   toMatch(/Content of template2.html/);
});
it('should change to blank', function() {
 select('template').option('');
 expect(element('.doc-example-live [ng-include]').text()).toEqual('');
});
```

ngInit

Описание

Директива `ngInit` выполняет код инициализации перед выполнением построения шаблона во время начальной инициализации приложения.

Использование

как атрибут

```
<ANY ng-init="{expression}">
    ...
</ANY>
```

как класс

```
<ANY class="ng-init: {expression};">
    ...
</ANY>
```

Параметры

- `ngInit – {expression} –` Выражение для вычисления.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
  <body>
    <div ng-init="greeting='Hello'; person='World'">
      {{greeting}} {{person}}!
    </div>
  </body>
</html>
```

End to end test

```
it('should check greeting', function() {
  expect(binding('greeting')).toBe('Hello');
  expect(binding('person')).toBe('World');
});
```

ngList

Описание

Конвертирует входной текст, разделенный заданным знаком разделителем, в массив строк.

Использование

как атрибут

```
<input ng-list="{string}">
   ...
</input>
```

как класс

```
<input class="ng-list: {string};">
   ...
</input>
```

Параметры

- `ngList(optional) – {string=} –` не обязательный знак разделитель, который будет использоваться при разделении строки. Если задается в форме `/something/`, значение будет сконвертировано в регулярное выражение.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <form name="myForm" ng-controller="Ctrl">
      List: <input name="namesInput" ng-model="names" ng-list required>
      <span class="error" ng-show="myForm.list.$error.required">
        Required!</span>
      <tt>names = {{names}}</tt><br/>
      <tt>myForm.namesInput.$valid = {{myForm.namesInput.$valid}}</tt><br/>
      <tt>myForm.namesInput.$error = {{myForm.namesInput.$error}}</tt><br/>
      <tt>myForm.$valid = {{myForm.$valid}}</tt><br/>
      <tt>myForm.$error.required = {{!!myForm.$error.required}}</tt><br/>
     </form>
  </body>
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.names = ['igor', 'misko', 'vojta'];
}
```

End to end test

```
it('should initialize to model', function() {
  expect(binding('names')).toEqual('["igor","misko","vojta"]');
  expect(binding('myForm.namesInput.$valid')).toEqual('true');
});

it('should be invalid if empty', function() {
  input('names').enter('');
  expect(binding('names')).toEqual('[]');
  expect(binding('myForm.namesInput.$valid')).toEqual('false');
});
```

ngModel

Описание

Эта директива указывает Angular, что нужна двухсторонняя привязка данных. Это всегда применяется в директивах `input`, `select`, `textarea`. Вы можете легко писать ваши собственные директивы, используя в них `ngModel`.

`ngModel` отвечает за:

- Привязку представления к модели при использовании с другими директивами, такими как `input`, `textarea` или `select`,
- Реализует поведение для проверки ввода (например, required, number, email, url),
- Устанавливает состояние элементов управления (valid/invalid, dirty/pristine, validation errors),
- Устанавливает требуемый css класс для элемента (`ng-valid`, `ng-invalid`, `ng-dirty`, `ng-pristine`),
- Регистрирует элемент управления в родительской директиве `form`.

Для примеров использования `ngModel`, смотрите:

- `input`
  - `text`
  - `checkbox`
  - `radio`
  - `number`
  - `email`
  - `url`
- `select`
- `textarea`

Использование

как атрибут

```
<input ng-model>
    ...
</input>
```

как класс

```
<input class="ng-model">
    ...
</input>
```

ngMousedown

Описание

Директива ngMousedown позволяет указать функцию, которая будет вызываться для обработки события mousedown (клавиша мыши вниз).

Использование

как атрибут

```
<ANY ng-mousedown="{expression}">
    ...
</ANY>
```

как класс

```
<ANY class="ng-mousedown: {expression};">
    ...
</ANY>
```

Параметры

- `ngMousedown - {expression}` - Выражение которое будет выполняться при возникновении события mousedown. (Объект события доступен через параметр `$event`)

Пример

Смотрите `ngClick`

ngMouseenter

Описание

Специфицирует обработчик для события mouseenter (вход мышью).

Использование

как атрибут

```
<ANY ng-mouseenter="{expression}">
    ...
</ANY>
```

как класс

```
<ANY class="ng-mouseenter: {expression};">
    ...
</ANY>
```

Параметры

- `ngMouseenter - {expression}` - Выражение обработчик события mouseenter. (Объект события доступен через параметр `$event`)

Пример

Смотрите `ngClick`

ngMouseleave

Описание

Указывает обработчик для события mouseleave (выход мышью).

Использование

как атрибут

```
<ANY ng-mouseleave="{expression}">
    ...
</ANY>
```

как класс

```
<ANY class="ng-mouseleave: {expression};">
    ...
</ANY>
```

Параметры

- `ngMouseleave - {expression} -` Выражение обработчик события mouseleave. (Объект события доступен через параметр `$event`)

Пример

Смотрите `ngClick`

ngMousemove

Описание

Указывает обработчик для события mousemove (перемещение мыши).

Использование

как атрибут

```
<ANY ng-mousemove="{expression}">
    ...
</ANY>
```

как класс

```
<ANY class="ng-mousemove: {expression};">
    ...
</ANY>
```

Параметры

- `ngMousemove - {expression} -` Выражение обработчик события mousemove. (Объект события доступен через параметр `$event`)

Пример

Смотрите `ngClick`

ngMouseover

Описание

Указывает обработчик для события mouseover (мышь сверху).

Использование

как атрибут

```
<ANY ng-mouseover="{expression}">
    ...
```

```
3. </ANY>
```

как класс

```
1. <ANY class="ng-mouseover: {expression};">
2.    ...
3. </ANY>
```

Параметры

- `ngMouseover - {expression} -` Выражение обработчик события mouseover. (Объект события доступен через параметр `$event`)

Пример

Смотрите `ngClick`

ngMouseup

Описание

Указывает обработчик для события mouseup.

Использование

как атрибут

```
1. <ANY ng-mouseup="{expression}">
2.    ...
3. </ANY>
```

как класс

```
1. <ANY class="ng-mouseup: {expression};">
2.    ...
3. </ANY>
```

Параметры

- `ngMouseup - {expression} -` Выражение обработчик события mouseup. (Объект события доступен через параметр `$event`)

Пример

Смотрите `ngClick`

ngMultiple

Описание

Спецификация HTML не требует от браузера обязательного наличия специальных атрибутов, таких как multiple. (Его наличие означает true, а отсутствие false) Это не позволяет компилятору angular корректно обрабатывать выражения привязки. Для решения этой проблемы и предназначена директива `ngMultiple`.

Использование

как атрибут

```
<SELECT ng-multiple="{expression}">
    ...
</SELECT>
```

Параметры

- `ngMultiple – {expression} –` Angular выражение для вычисления.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
  <body>
    Check me check multiple: <input type="checkbox" ng-model="checked"><br/>
    <select id="select" ng-multiple="checked">
      <option>Misko</option>
      <option>Igor</option>
      <option>Vojta</option>
      <option>Di</option>
    </select>
  </body>
</html>
```

End to end test

```
it('should toggle multiple', function() {
  expect(element('.doc-example-live #select').prop('multiple')).toBeFalsy();
  input('checked').check();
  expect(element('.doc-example-live #select').prop('multiple')).toBeTruthy();
});
```

ngNonBindable

Описание

Иногда требуется написать код, который является корректным выражением angular, но которое должно выводится как есть, без вычисления. Используйте `ngNonBindable` чтобы заставить angular игнорировать содержимое HTML.

Использование

как атрибут

```
<ANY ng-non-bindable>
   ...
</ANY>
```

как класс

```
<ANY class="ng-non-bindable">
   ...
</ANY>
```

Информация о директиве

- Директива выполняется с приоритетом 1000.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
  <body>
    <div>Normal: {{1 + 2}}</div>
    <div ng-non-bindable>Ignored: {{1 + 2}}</div>
  </body>
</html>
```

End to end test

```
it('should check ng-non-bindable', function() {
  expect(using('.doc-example-live').binding('1 + 2')).toBe('3');
  expect(using('.doc-example-live').element('div:last').text()).
    toMatch(/1 \+ 2/);
});
```

ngPluralize

Описание

Обзор

Директива `ngPluralize` отображает сообщение в соответствии с установленной локализацией в en-US. Эта локализация встроена в файл angular.js и она может быть изменена на другую (смотрите Angular i18n руководство разработчика). Вы настраиваете директиву ngPluralize для указания специфической локализации из многих категорий, и строки будут отображаться в соответствии с этими настройками.

Различные категории и явное указание правил для чисел

В Angular имеет две категории, по умолчанию применяемые для локализации en-US: "one " (первый элемент) и "other " (другой элемент).

Каждой категории могут соответствовать множество чисел (для примера в локализации en-US, категории "other" соответствуют все числа больше 1), явно указанная роль для чисел может соответствовать только одному числу. К примеру, явно заданная роль для "3" соответствует числу 3. Вы увидите использование множественных категорий и явного задания ролей для чисел в следующих частях этой документации.

Настройка ngPluralize

Для настройки ngPluralize предоставлено 2 атрибута: `count` и `when`. Вы можете также использовать необязательный атрибут `offset`.

Значение атрибута `count` может быть строкой или выражением Angular, которое вычисляется в текущей области видимости для получения значения.

Атрибут `when` специфицирует карту значений из категорий и строк для отображения. Его значением должен быть объект JSON, так как Angular сможет его корректно интерпретировать.

В следующем примере показано как настроить ngPluralize:

```
<ng-pluralize count="personCount"
                 when="{'0': 'Nobody is viewing.',
                     'one': '1 person is viewing.',
                     'other': '{} people are viewing.'}">
</ng-pluralize>
```

В примере, `"0: Nobody is viewing."` Это явное задание роли для числа. Если вы не укажете этой роли, число 0 будет соответствовать роли для категории "other" и будет выводится "0 people are viewing" вместо "Nobody is viewing". Вы можете указать и роли для других чисел, к примеру, 12, так будет выводить "12 people are viewing", вы можете указать "a dozen people are viewing".

Вы можете использовать закрытые фигурные скобки (`{}`), как якорь для обрабатываемого числа, которое будет вставлено в результирующую строку. В предыдущем примере, Angular заменит `{}` на `{{personCount}}`. Вместо `{},` может также находится тело выражения {{numberExpression}}.

Настройка ngPluralize с использование offset

Атрибут `offset` кроме того позволяет еще больше управлять процессом плюрализмами текста. Например, вместо сообщения "4 people are viewing this document", вы можете отобразить значение "John, Kate and 2 others are viewing this document". Атрибут offset позволяет вам задать сдвиг для номера в любое желаемое значение. Посмотрите на этот пример:

```
<ng-pluralize count="personCount" offset=2
               when="{'0': 'Nobody is viewing.',
                      '1': '{{person1}} is viewing.',
                      '2': '{{person1}} and {{person2}} are viewing.',
                      'one': '{{person1}}, {{person2}} and one other person are vie
wing.',
                      'other': '{{person1}}, {{person2}} and {} other people are vi
ewing.'}">
</ng-pluralize>
```

Обратите внимание на используемый стиль в двух категориях (one, other), и мы добавили еще три роли для чисел 0, 1 и 2. Когда один человек, возможно John, посмотрит документ, будет показано "John is viewing". Когда три человека посмотрят документ, явной роли не будет найдено, так как сдвиг равен 2 и текущее количество равно 3, Angular использует 1 для решения о том, какую категорию использовать. В нашем случае будет применена категория 'one' и будет выведено "John, Marry and one other person are viewing".

Заметьте, что когда вы определили сдвиг, вы должны задать явно роли для чисел от 0 и до значения сдвига. Если вы используете сдвиг равный, например, 3, вы должны явно указать роли для чисел 0, 1, 2 и 3. Вы должны также указать строки для категорий "one" и "other".

Использование

Эта директива может использоваться как дополнительный элемент, но с учетом ограничений IE:

```
<ng-pluralize
       count="{string|expression}"
       when="{string}"
       [offset="{number}"]>
</ng-pluralize>
```

как атрибут

```
<ANY ng-pluralize
     count="{string|expression}"
     when="{string}"
     [offset="{number}"]>
   ...
</ANY>
```

Параметры

- `count – {string|expression} –` привязанное значение для обработки.
- `when – {string} –` Карта где свойства являются категориями, а их значения являются форматом выводимой строки.
- `offset(optional) – {number=} –` Сдвиг для вычитания от переданного номера.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-controller="Ctrl">
      Person 1:<input type="text" ng-model="person1" value="Igor" /><br/>
      Person 2:<input type="text" ng-model="person2" value="Misko" /><br/>
      Number of People:<input type="text" ng-model="personCount" value="1" /><br/>
```

```
12.
13.      <!--- Example with simple pluralization rules for en locale --->
14.      Without Offset:
15.      <ng-pluralize count="personCount"
16.                    when="{'0': 'Nobody is viewing.',
17.                           'one': '1 person is viewing.',
18.                           'other': '{} people are viewing.'}">
19.      </ng-pluralize><br>
20.
21.      <!--- Example with offset --->
22.      With Offset(2):
23.      <ng-pluralize count="personCount" offset=2
24.                    when="{'0': 'Nobody is viewing.',
25.                           '1': '{{person1}} is viewing.',
26.                           '2': '{{person1}} and {{person2}} are viewing.',
27.                           'one': '{{person1}}, {{person2}} and one other person a
   re viewing.',
28.                           'other': '{{person1}}, {{person2}} and {} other people
   are viewing.'}">
29.      </ng-pluralize>
30.    </div>
31.  </body>
32.</html>
```

Script.js

```
1.  function Ctrl($scope) {
2.    $scope.person1 = 'Igor';
3.    $scope.person2 = 'Misko';
4.    $scope.personCount = 1;
5.  }
```

End to end test

```
1.  it('should show correct pluralized string', function() {
2.    expect(element('.doc-example-live ng-pluralize:first').text()).
3.                                      toBe('1 person is viewing.');
4.    expect(element('.doc-example-live ng-pluralize:last').text()).
5.                                         toBe('Igor is viewing.');
6.
7.    using('.doc-example-live').input('personCount').enter('0');
8.    expect(element('.doc-example-live ng-pluralize:first').text()).
9.                                        toBe('Nobody is viewing.');
10.   expect(element('.doc-example-live ng-pluralize:last').text()).
11.                                       toBe('Nobody is viewing.');
12.
13.   using('.doc-example-live').input('personCount').enter('2');
14.   expect(element('.doc-example-live ng-pluralize:first').text()).
15.                                     toBe('2 people are viewing.');
16.   expect(element('.doc-example-live ng-pluralize:last').text()).
```

```
                    toBe('Igor and Misko are viewing.');

  using('.doc-example-live').input('personCount').enter('3');
  expect(element('.doc-example-live ng-pluralize:first').text()).
                                    toBe('3 people are viewing.');
  expect(element('.doc-example-live ng-pluralize:last').text()).
                    toBe('Igor, Misko and one other person are viewing.');

  using('.doc-example-live').input('personCount').enter('4');
  expect(element('.doc-example-live ng-pluralize:first').text()).
                                    toBe('4 people are viewing.');
  expect(element('.doc-example-live ng-pluralize:last').text()).
                    toBe('Igor, Misko and 2 other people are viewing.');
});

it('should show data-binded names', function() {
  using('.doc-example-live').input('personCount').enter('4');
  expect(element('.doc-example-live ng-pluralize:last').text()).
      toBe('Igor, Misko and 2 other people are viewing.');

  using('.doc-example-live').input('person1').enter('Di');
  using('.doc-example-live').input('person2').enter('Vojta');
  expect(element('.doc-example-live ng-pluralize:last').text()).
      toBe('Di, Vojta and 2 other people are viewing.');
});
```

ngReadonly

Описание

Спецификация HTML не требует, чтобы браузеры проверяли наличие специальных атрибутов, таких как readonly. (Если атрибут установлен, он имеет значение true, если нет, считается, что он имеет значение false) Это не позволяет компилятору angular корректно обрабатывать выражения привязки. Для решения этой проблемы мы добавили директиву `ngReadonly`.

Использование

как атрибут

```
<INPUT ng-readonly
     expression="{string}">
   ...
</INPUT>
```

Параметры

- `expression – {string} –` Angular выражение для вычисления.

Пример

Index.html

```
<!doctype html>
```

```
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
  <body>
    Check me to make text readonly: <input type="checkbox" ng-model="checked"><br/
>
    <input type="text" ng-readonly="checked" value="I'm Angular"/>
  </body>
</html>
```

End to end test

```
it('should toggle readonly attr', function() {
  expect(element('.doc-example-live :text').prop('readonly')).toBeFalsy();
  input('checked').check();
  expect(element('.doc-example-live :text').prop('readonly')).toBeTruthy();
});
```

ngRepeat

Описание

Директива `ngRepeat` создает экземпляры по шаблону для каждого элемента коллекции. Каждый экземпляр шаблона получает собственную область видимости, в котором создаются переменные, имеющиеся в текущем элементе коллекции, и `$index` устанавливается в индекс элемента в массиве или его ключ в карте значений.

Специальные свойства, представленные каждым шаблонным элементом в его локальной области видимости, включают:

- `$index` – `{number}` – сдвиг итератора при повторении элементов (0..length-1)
- `$first` – `{boolean}` – true, если повторяемый элемент является первым элементом в коллекции.
- `$middle` – `{boolean}` – true, если повторяемый элемент не первый и не последний элемент в коллекции.
- `$last` – `{boolean}` – true, если повторяемый элемент является последним элементом коллекции.

Использование

как атрибут

```
<ANY ng-repeat="{repeat_expression}">
    ...
</ANY>
```

как класс

```
<ANY class="ng-repeat: {repeat_expression};">
    ...
```

```
3. </ANY>
```

Информация о директиве

- Эта директива создает новую область видимости.
- Эта директива выполняется с приоритетом 1000.

Параметры

- `ngRepeat – {repeat_expression} –` Выражение, которое указывает как именно нужно обходить коллекцию. В текущей версии поддерживается два формата:
  - `variable in expression` – где variable это определенная пользователем переменная, которая будет содержать текущее значение при перечислении и `expression` – это выражение в области видимости, которое предоставляет коллекцию для итераций.

    Например: `track in cd.tracks`.
  - `(key, value) in expression` – где `key` и `value` могут быть определенными пользователем идентификаторами, а `expression` это выражение в области видимости, которое возвращает коллекцию для итераций.

    Например: `(name, age) in {'adam':10, 'amalie':12}`.

Пример

В этом примере инициализируется область видимости списком людей, и затем они отображаются с использованием директивы `ngRepeat`:

Index.html

```
1. <!doctype html>
2. <html ng-app>
3.   <head>
4.     <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
5.   </head>
6.   <body>
7.     <div ng-init="friends = [{name:'John', age:25}, {name:'Mary', age:28}]">
8.       I have {{friends.length}} friends. They are:
9.       <ul>
10.        <li ng-repeat="friend in friends">
11.          [{{$index + 1}}] {{friend.name}} who is {{friend.age}} years old.
12.        </li>
13.      </ul>
14.    </div>
15.  </body>
16. </html>
```

End to end test

```
1. it('should check ng-repeat', function() {
2.   var r = using('.doc-example-live').repeater('ul li');
3.   expect(r.count()).toBe(2);
4.   expect(r.row(0)).toEqual(["1","John","25"]);
```

```
5.   expect(r.row(1)).toEqual(["2","Mary","28"]);
6. });
```

ngSelected

Описание

Спецификация HTML не требует от браузеров обязательного наличия специальных атрибутов, таких как selected. (Если он указан, считается, что он имеет значение true, иначе - значение false) Это не позволяет компилятору angular корректно обрабатывать выражения привязки. Для решения этой проблемы мы добавили директиву `ngSelected`.

Использование

как атрибут

```
1. <OPTION ng-selected
2.      expression="{string}">
3.    ...
4. </OPTION>
```

Параметры

- `expression – {string} –` Angular выражение для вычисления.

Пример

Index.html

```
1. <!doctype html>
2. <html ng-app>
3.   <head>
4.     <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
5.   </head>
6.   <body>
7.     Check me to select: <input type="checkbox" ng-model="selected"><br/>
8.     <select>
9.       <option>Hello!</option>
10.      <option id="greet" ng-selected="selected">Greetings!</option>
11.    </select>
12.  </body>
13. </html>
```

End to end test

```
1. it('should select Greetings!', function() {
2.   expect(element('.doc-example-live #greet').prop('selected')).toBeFalsy();
3.   input('selected').check();
4.   expect(element('.doc-example-live #greet').prop('selected')).toBeTruthy();
5. });
```

ngShow

Описание

Директивы ngShow и ngHide соответственно отображают и скрывают часть дерева DOM.

Использование

как атрибут

```
<ANY ng-show="{expression}">
    ...
</ANY>
```

как класс

```
<ANY class="ng-show: {expression};">
    ...
</ANY>
```

Параметры

- ngShow – {expression} – Если выражение возвращает true, элемент будет показан, иначе будет скрыт.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
  <body>
    Click me: <input type="checkbox" ng-model="checked"><br/>
    Show: <span ng-show="checked">I show up when your checkbox is checked.</span>
<br/>
    Hide: <span ng-hide="checked">I hide when your checkbox is checked.</span>
  </body>
</html>
```

End to end test

```
it('should check ng-show / ng-hide', function() {
  expect(element('.doc-example-live span:first:hidden').count()).toEqual(1);
  expect(element('.doc-example-live span:last:visible').count()).toEqual(1);

  input('checked').check();

  expect(element('.doc-example-live span:first:visible').count()).toEqual(1);
```

```
8.   expect(element('.doc-example-live span:last:hidden').count()).toEqual(1);
9. });
```

ngSrc

Описание

Использование разметки Angular `{{hash}}` в атрибуте `src` правильно не работает: Браузер воспримет ваше выражение как литерал, до того, как ваше URL будет обработано и выражение `{{hash}}` будет заменено на его значение. Директива `ngSrc` решает эту проблему.

Так делать не нужно:

```
1. <img src="http://www.gravatar.com/avatar/{{hash}}"/>
```

Нужно делать так:

```
1. <img ng-src="http://www.gravatar.com/avatar/{{hash}}"/>
```

Использование

как атрибут

```
1. <IMG ng-src="{template}">
2.    ...
3. </IMG>
```

Параметры

- `ngSrc – {template} –` любая строка, которая может содержать разметку `{{}}`.

ngStyle

Описание

Директива `ngStyle` позволяет устанавливать CSS стили для элементов HTML в зависимости от условий.

Использование

как атрибут

```
1. <ANY ng-style="{expression}">
2.    ...
3. </ANY>
```

как класс

```
1. <ANY class="ng-style: {expression};">
2.    ...
3. </ANY>
```

Параметры

- `ngStyle – {expression} –` Выражение которое возвращает объект, ключами которого выступают имена задаваемых стилей CSS, а значения – это значения задаваемые свойству CSS.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
  <body>
    <input type="button" value="set" ng-click="myStyle={color:'red'}">
    <input type="button" value="clear" ng-click="myStyle={}">
    <br/>
    <span ng-style="myStyle">Sample Text</span>
    <pre>myStyle={{myStyle}}</pre>
  </body>
</html>
```

Style.css

```
span {
  color: black;
}
```

End to end test

```
it('should check ng-style', function() {
  expect(element('.doc-example-live span').css('color')).toBe('rgb(0, 0, 0)');
  element('.doc-example-live :button[value=set]').click();
  expect(element('.doc-example-live span').css('color')).toBe('rgb(255, 0, 0)');
  element('.doc-example-live :button[value=clear]').click();
  expect(element('.doc-example-live span').css('color')).toBe('rgb(0, 0, 0)');
});
```

ngSubmit

Описание

Включает привязку выражения angular к событию onsubmit (отправка формы).

Дополнительно она отключает поведение по умолчанию (отправка формы на сервер и перезагрузка текущей страницы).

Использование

как атрибут

```
<form ng-submit="{expression}">
    ...
</form>
```

как класс

```
<form class="ng-submit: {expression};">
    ...
</form>
```

Параметры

- `ngSubmit – {expression} –` Выражение для вычисления.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <form ng-submit="submit()" ng-controller="Ctrl">
      Enter text and hit enter:
      <input type="text" ng-model="text" name="text" />
      <input type="submit" id="submit" value="Submit" />
      <pre>list={{list}}</pre>
    </form>
  </body>
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.list = [];
  $scope.text = 'hello';
  $scope.submit = function() {
    if (this.text) {
      this.list.push(this.text);
      this.text = '';
    }
  };
}
```

End to end test

```
it('should check ng-submit', function() {
  expect(binding('list')).toBe('[]');
  element('.doc-example-live #submit').click();
  expect(binding('list')).toBe('["hello"]');
  expect(input('text').val()).toBe('');
});
it('should ignore empty strings', function() {
  expect(binding('list')).toBe('[]');
  element('.doc-example-live #submit').click();
  element('.doc-example-live #submit').click();
  expect(binding('list')).toBe('["hello"]');
});
```

ngSwitch

Описание

Условное изменение структуры DOM.

Использование

Эта директива может использоваться как дополнительный элемент, но с учетом ограничений IE:

```
<ANY ng-switch="expression">
<ANY ng-switch-when="matchValue1">...</ANY>
<ANY ng-switch-when="matchValue2">...</ANY>
...
<ANY ng-switch-default>...</ANY>
</ANY>
```

Информация о директиве

- Эта директива создает новую область видимости.

Параметры

- `ngSwitch|on – {*} –` выражение, результат которого сравнивается с содержимым `ng-switch-when`.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
```

```
  <body>
    <div ng-controller="Ctrl">
      <select ng-model="selection" ng-options="item for item in items">
      </select>
      <tt>selection={{selection}}</tt>
      <hr/>
      <div ng-switch on="selection" >
        <div ng-switch-when="settings">Settings Div</div>
        <span ng-switch-when="home">Home Span</span>
        <span ng-switch-default>default</span>
      </div>
    </div>
  </body>
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.items = ['settings', 'home', 'other'];
  $scope.selection = $scope.items[0];
}
```

End to end test

```
it('should start in settings', function() {
 expect(element('.doc-example-live [ng-switch]').text()).toMatch(/Settings Div/);
});
it('should change to home', function() {
 select('selection').option('home');
 expect(element('.doc-example-live [ng-switch]').text()).toMatch(/Home Span/);
});
it('should select deafault', function() {
 select('selection').option('other');
 expect(element('.doc-example-live [ng-switch]').text()).toMatch(/default/);
});
```

ngTransclude

Описание

Вставляет содержимое элемента DOM в месте применения директивы.

Использование

Как атрибут

```
<ANY ng-transclude>
   ...
</ANY>
```

как класс

```
<ANY class="ng-transclude">
   ...
</ANY>
```

Пример

Index.html

```
<!doctype html>
<html ng-app="transclude">
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-controller="Ctrl">
      <input ng-model="title"><br>
      <textarea ng-model="text"></textarea> <br/>
      <pane title="{{title}}">{{text}}</pane>
    </div>
  </body>
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.title = 'Lorem Ipsum';
  $scope.text = 'Neque porro quisquam est qui dolorem ipsum quia dolor...';
}

angular.module('transclude', [])
 .directive('pane', function(){
    return {
      restrict: 'E',
      transclude: true,
      scope: 'isolate',
      locals: { title:'bind' },
      template: '<div style="border: 1px solid black;">' +
                  '<div style="background-color: gray">{{title}}</div>' +
                  '<div ng-transclude></div>' +
                '</div>'
    };
});
```

End to end test

```
it('should have transcluded', function() {
```

```
  input('title').enter('TITLE');
  input('text').enter('TEXT');
  expect(binding('title')).toEqual('TITLE');
  expect(binding('text')).toEqual('TEXT');
});
```

ngView

Описание

Обзор

Директива `ngView` дополняет сервис `$route`, включая отображение шаблона для текущего пути в главном файле (`index.html`). Каждый раз, когда изменяется текущий путь, включается соответствующее изменениям представление, настроенное в сервисе `$route`.

Использование

Эта директива может использоваться как дополнительный элемент, но с учетом ограничений IE:

```
<ng-view>
</ng-view>
```

как атрибут

```
<ANY ng-view>
   ...
</ANY>
```

как класс

```
<ANY class="ng-view">
   ...
</ANY>
```

Информация о директиве

- Эта директива создает новую область видимости.

События

$viewContentLoaded

Возбуждается каждый раз, когда содержимое ngView будет загружено.

Тип: всплывающее

Цель: текущая область видимости ngView

Пример

Index.html

```
<!doctype html>
<html ng-app="ngView">
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-controller="MainCntl">
      Choose:
      <a href="Book/Moby">Moby</a> |
      <a href="Book/Moby/ch/1">Moby: Ch1</a> |
      <a href="Book/Gatsby">Gatsby</a> |
      <a href="Book/Gatsby/ch/4?key=value">Gatsby: Ch4</a> |
      <a href="Book/Scarlet">Scarlet Letter</a><br/>

      <div ng-view></div>
      <hr />

      <pre>$location.path() = {{$location.path()}}</pre>
      <pre>$route.current.templateUrl = {{$route.current.templateUrl}}</pre>
      <pre>$route.current.params = {{$route.current.params}}</pre>
      <pre>$route.current.scope.name = {{$route.current.scope.name}}</pre>
      <pre>$routeParams = {{$routeParams}}</pre>
    </div>
  </body>
</html>
```

Book.html

```
controller: {{name}}<br />
Book Id: {{params.bookId}}<br />
```

Chapter.html

```
controller: {{name}}<br />
Book Id: {{params.bookId}}<br />
Chapter Id: {{params.chapterId}}
```

Script.js

```
angular.module('ngView', [], function($routeProvider, $locationProvider) {
  $routeProvider.when('/Book/:bookId', {
    templateUrl: 'book.html',
    controller: BookCntl
  });
```

```
  $routeProvider.when('/Book/:bookId/ch/:chapterId', {
    templateUrl: 'chapter.html',
    controller: ChapterCntl
  });

  // configure html5 to get links working on jsfiddle
  $locationProvider.html5Mode(true);
});

function MainCntl($scope, $route, $routeParams, $location) {
  $scope.$route = $route;
  $scope.$location = $location;
  $scope.$routeParams = $routeParams;
}

function BookCntl($scope, $routeParams) {
  $scope.name = "BookCntl";
  $scope.params = $routeParams;
}

function ChapterCntl($scope, $routeParams) {
  $scope.name = "ChapterCntl";
  $scope.params = $routeParams;
}
```

End to end test

```
it('should load and compile correct template', function() {
  element('a:contains("Moby: Ch1")').click();
  var content = element('.doc-example-live [ng-view]').text();
  expect(content).toMatch(/controller\: ChapterCntl/);
  expect(content).toMatch(/Book Id\: Moby/);
  expect(content).toMatch(/Chapter Id\: 1/);

  element('a:contains("Scarlet")').click();
  content = element('.doc-example-live [ng-view]').text();
  expect(content).toMatch(/controller\: BookCntl/);
  expect(content).toMatch(/Book Id\: Scarlet/);
});
```

script

Описание

Загружает содержимое тега script, тип которого установлен в `text/ng-template`, в `$templateCache`, так что шаблон может быть использован в `ngInclude`, `ngView` или шаблонах директив.

Использование

Эта директива может использоваться как дополнительный элемент, но с учетом ограничений IE:

```
<script
       type="text/ng-template">
</script>
```

Параметры

- `type – {'text/ng-template'} – должен быть установлен в 'text/ng-template'`

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
  <body>
    <script type="text/ng-template" id="/tpl.html">
      Content of the template.
  </body>
</html>
```

End to end test

```
it('should load template defined inside script tag', function() {
  element('#tpl-link').click();
  expect(element('#tpl-content').text()).toMatch(/Content of the template/);
});
```

select

Описание

HTML элемент SELECT с angular привязкой данных.

ngOptions

Необязательный атрибут `ngOptions` может использоваться для динамической генерации списка элементов `<option>` для элемента `<select>` используя массив или объект, возвращенный заданным в `ngOptions` выражением. Когда элемент выбирается в select меню, значение из массива элементов или свойства объекта привязывается к свойству, заданному с помощью директивы `ngModel` и устанавливается атрибут selected.

Необязательно, но можно жестко задать один элемент `<option>`, значение которого установить в пустую строку, и вложить его в элемент `<select>`. Этот элемент будет соответствовать значению `null` или опции "not selected". Смотрите пример ниже для демонстрации.

Заметьте: `ngOptions` предоставляет итератор для элемента `<option>` который должен использоваться вместо `ngRepeat` когда вам нужно модель `select` привязать не к строковому значению. Это потому, что элемент option в настоящее время может быть привязан только к строковому значению.

Использование

Эта директива может использоваться как дополнительный элемент, но с учетом ограничений IE:

```
<select
        ngModel="{string}"
        [name="{string}"]
        [required]
        [ngRequired="{string}"]
        [ngOptions="{comprehension_expression}"]>
</select>
```

Параметры

- `ngModel – {string} –` angular выражение для привязки данных.
- `name(optional) – {string=} –` Имя свойства в родительской форме, под которым элемент управления в ней публикуется.
- `required(optional) – {string=} –` Это элемент управления будет считаться правильным, только если значение введено.
- `ngRequired(optional) – {string=} –` Добавляет `required` атрибут и `required` проверку содержимого элемента, когда выражение ngRequired возвращает true. Используйте `ngRequired` вместо `required` когда нужно привязать данные к атрибуту `required`.
- `ngOptions(optional) – {comprehension_expression=} –` в одной из следующих форм:
  - когда источник данных - массив:
    - `label for value in array`
    - `select as label for value in array`
    - `label group by group for value in array`
    - `select as label group by group for value in array`
  - когда источник данных - объект:
    - `label for (key , value) in object`
    - `select as label for (key , value) in object`
    - `label group by group for (key, value) in object`
    - `select as label group by group for (key, value) in object`

  Где:

  - `array` / `object`: выражение, которое возвращает массив / объект для итерации.
  - `value`: локальная переменная, которая будет ссылаться на перечисляемый элемент в массиве `array` или перечисляемое значение свойства в объекте `object` при итерации.
  - `key`: локальная переменная, которая будет ссылаться на имя свойства в объекте `object` который используется для итерации.
  - `label`: выражение для вычисления результата, который будет размещен на метке элемента `<option>`. Это выражение должно ссылаться на переменную `value` (например, `value.propertyName`).
  - `select`: Выражения для вычисления результата, который будет привязан к модели родительского элемента `<select>` при выборе текущего элемента. Если не указано, выражение `select` будет по умолчанию равно переменной `value`.
  - `group`: Выражение для вычисления результата, который будет использоваться для группирования элементов option используя DOM элемент `<optgroup>`.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-controller="MyCntrl">
      <ul>
        <li ng-repeat="color in colors">
          Name: <input ng-model="color.name">
          [<a href ng-click="colors.splice($index, 1)">X</a>]
        </li>
        <li>
          [<a href ng-click="colors.push({})">add</a>]
        </li>
      </ul>
      <hr/>
      Color (null not allowed):
      <select ng-model="color" ng-options="c.name for c in colors"></select><br>

      Color (null allowed):
      <span  class="nullable">
        <select ng-model="color" ng-options="c.name for c in colors">
          <option value="">-- chose color --</option>
        </select>
      </span><br/>

      Color grouped by shade:
      <select ng-model="color" ng-options="c.name group by c.shade for c in colors
">
      </select><br/>


      Select <a href ng-click="color={name:'not in list'}">bogus</a>.<br>
      <hr/>
      Currently selected: {{ {selected_color:color}  }}
      <div style="border:solid 1px black; height:20px"
           ng-style="{'background-color':color.name}">
      </div>
    </div>
  </body>
</html>
```

Script.js

```
function MyCntrl($scope) {
```

```
  $scope.colors = [
    {name:'black', shade:'dark'},
    {name:'white', shade:'light'},
    {name:'red', shade:'dark'},
    {name:'blue', shade:'dark'},
    {name:'yellow', shade:'light'}
  ];
  $scope.color = $scope.colors[2]; // red
}
```

End to end test

```
it('should check ng-options', function() {
  expect(binding('{selected_color:color}')).toMatch('red');
  select('color').option('0');
  expect(binding('{selected_color:color}')).toMatch('black');
  using('.nullable').select('color').option('');
  expect(binding('{selected_color:color}')).toMatch('null');
});
```

textarea

Описание

HTML элемент управления textarea с angular привязкой данных. Привязка данных и проверка правильности ввода для этого элемента такие же, как и для других `input элементов`.

Использование

Эта директива может использоваться как дополнительный элемент, но с учетом ограничений IE:

```
<textarea
       ngModel="{string}"
       [name="{string}"]
       [required]
       [ngRequired="{string}"]
       [ngMinlength="{number}"]
       [ngMaxlength="{number}"]
       [ngPattern="{string}"]
       [ngChange="{string}"]>
</textarea>
```

Parameters

- `ngModel – {string}` – angular выражение для привязки данных.
- `name(optional) – {string=}` – Имя свойства формы, которое содержит ссылку на элемент управления.
- `required(optional) – {string=}` – Устанавливает ключ `required` в ошибках проверки данных, если значение в поле не введено.

- `ngRequired(optional)` – `{string=}` – Добавляет атрибут `required` и `required` проверку данных для элемента, когда выражение в ngRequired возвращает true. Используйте `ngRequired` вместо `required` когда вам нужно привязать данные к атрибуту `required`.
- `ngMinlength(optional)` – `{number=}` – Устанавливает ключ `minlength` в ошибках проверки данных, если длина введенного значения меньше указанной.
- `ngMaxlength(optional)` – `{number=}` – Устанавливает ключ `maxlength` в ошибках проверки данных, если длина введенного значения больше указанной.
- `ngPattern(optional)` – `{string=}` – Устанавливает ключ `pattern` в ошибках проверки данных если введенное значение не соответствует заданному регулярному выражению Можно задать значение в формате `/regexp/` или `regexp` которое ссылается на выражение в области видимости.
- `ngChange(optional)` – `{string=}` – Angular выражение, которое будет выполнено когда в результате ввода пользователя элемент поменяет свое содержимое.

Фильтры

currency

Описание

Форматирует число в денежном формате ($1,234.56). Когда явно не указан символ валюты, применяется символ, используемый в текущей локализации.

Использование

В привязках в HTML шаблоне

```
{{ currency_expression | currency[:symbol] }}
```

В JavaScript

```
$filter('currency')(amount[, symbol])
```

Параметры

- `amount` – `{number}` – Входное число.
- `symbol(optional)` – `{string=}` – Символ или идентификатор валюты для отображения.

Возврат

`{string}` – Отформатированная строка в денежном представлении.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
```

```
    <div ng-controller="Ctrl">
      <input type="number" ng-model="amount"> <br>
      default currency symbol ($): {{amount | currency}}<br>
      custom currency identifier (USD$): {{amount | currency:"USD$"}}
    </div>
  </body>
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.amount = 1234.56;
}
```

End to end test

```
it('should init with 1234.56', function() {
  expect(binding('amount | currency')).toBe('$1,234.56');
  expect(binding('amount | currency:"USD$"')).toBe('USD$1,234.56');
});
it('should update', function() {
  input('amount').enter('-1234');
  expect(binding('amount | currency')).toBe('($1,234.00)');
  expect(binding('amount | currency:"USD$"')).toBe('(USD$1,234.00)');
});
```

date

Описание

Форматирует дату в строку в соответствии с переданной форматной строкой.

Форматная строка может состоять из следующих элементов:

- `'yyyy'`: 4 цифры для вывода года (например, 1 => 0001, 2010 => 2010)
- `'yy'`: 2 последние цифры года (00-99). (пример, 2001 => 01, 2010 => 10)
- `'y'`: 1 число для вывода года (например, 1 => 1, 199 => 199)
- `'MMMM'`: месяц в длинном формате (January-December)
- `'MMM'`: месяц в трехбуквенном формате (Jan-Dec)
- `'MM'`: месяц цифрами, с ведущим нулем (01-12)
- `'M'`: месяц цифрами, без ведущего нуля (1-12)
- `'dd'`: день в месяце, с ведущим нулем (01-31)
- `'d'`: день в месяце, без ведущего нуля (1-31)
- `'EEEE'`: День недели в длинном формате,(Sunday-Saturday)
- `'EEE'`: День недели в коротком формате, (Sun-Sat)
- `'HH'`: Час, с ведущим нулем (00-23)
- `'H'`: Час без ведущего нуля (0-23)
- `'hh'`: Час, с ведущим нулем в 12-ти часовом формате (01-12)
- `'h'`: Час без ведущего нуля в 12-ти часовом формате (1-12)
- `'mm'`: Минуты с ведущим нулем (00-59)
- `'m'`: Минуты без ведущего нуля(0-59)

- `'ss'`: Секунды с ведущим нулем (00-59)
- `'s'`: Секунды без ведущего нуля (0-59)
- `'a'`: Указатель am/pm для 12-ти часового формата.
- `'Z'`: 4 цифры (плюс знак), представляющие сдвиг временной зоны (часовой пояс) (-1200-+1200)

Форматная строка может также иметь одно из следующих предопределенных в форматах локализации значений:

- `'medium'`: эквивалент `'MMM d, y h:mm:ss a'` для локализации en_US (пример, Sep 3, 2010 12:05:08 pm)
- `'short'`: эквивалент `'M/d/yy h:mm a'` для локализации en_US (пример, 9/3/10 12:05 pm)
- `'fullDate'`: эквивалент `'EEEE, MMMM d,y'` для локализации en_US (пример, Friday, September 3, 2010)
- `'longDate'`: эквивалент `'MMMM d, y'` для локализации en_US (пример, September 3, 2010)
- `'mediumDate'`: эквивалент `'MMM d, y'` для локализации en_US (пример, Sep 3, 2010)
- `'shortDate'`: эквивалент `'M/d/yy'` для локализации en_US (пример, 9/3/10)
- `'mediumTime'`: эквивалент `'h:mm:ss a'` для локализации en_US (пример, 12:05:08 pm)
- `'shortTime'`: эквивалент `'h:mm a'` для локализации en_US (пример 12:05 pm)

Форматная строка может содержать литеральные значения. Их нужно помещать в одинарные кавычки (пример, `"h 'in the morning'"`). В случае если одинарные кавычки должны присутствовать в выходной строке, используйте двойные одинарные кавычки (пример, `"h o''clock"`).

Использование

В привязке в HTML шаблоне

```
{{ date_expression | date[:format] }}
```

В JavaScript

```
$filter('date')(date[, format])
```

Параметры

- `date – {(Date|number|string)} –` Дата в одном из следующих форматов: объект Date, миллисекунды (строка или число) или строковой формат даты и времени по спецификации ISO 8601 (пример, yyyy-MM-ddTHH:mm:ss.SSSZ и его короткая версия yyyy-MM-ddTHH:mmZ, yyyy-MM-dd или yyyyMMddTHHmmssZ). Если указана временная зона (часовой пояс) во входной строке, то время будет преобразовано во время в локальной временной зоне.
- `format(optional) – {string=} –` Форматирующая строка (смотрите описание). Если не указан, будет использоваться `mediumDate`.

Возврат

`{string}` – Отформатированная строка или входная объект, если он не является допустимым объектом для форматирования.

Пример

Index.html

```
1. <!doctype html>
2. <html ng-app>
```

```
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
  <body>
    <span ng-non-bindable>{{1288323623006 | date:'medium'}}</span>:
        {{1288323623006 | date:'medium'}}<br>
    <span ng-non-bindable>{{1288323623006 | date:'yyyy-MM-dd HH:mm:ss Z'}}</span>:
       {{1288323623006 | date:'yyyy-MM-dd HH:mm:ss Z'}}<br>
    <span ng-non-bindable>{{1288323623006 | date:'MM/dd/yyyy @ h:mma'}}</span>:
       {{'1288323623006' | date:'MM/dd/yyyy @ h:mma'}}<br>
  </body>
</html>
```

End to end test

```
it('should format date', function() {
  expect(binding("1288323623006 | date:'medium'")).
     toMatch(/Oct 2\d, 2010 \d{1,2}:\d{2}:\d{2} (AM|PM)/);
  expect(binding("1288323623006 | date:'yyyy-MM-dd HH:mm:ss Z'")).
     toMatch(/2010\-10\-2\d \d{2}:\d{2}:\d{2} (\-|\+)?\d{4}/);
  expect(binding("'1288323623006' | date:'MM/dd/yyyy @ h:mma'")).
     toMatch(/10\/2\d\/2010 @ \d{1,2}:\d{2}(AM|PM)/);
});
```

filter

Описание

Выбор из массива некоторых элементов и возврат их в новом массиве.

Обратите внимание: Эта функция используется только для типа `Array` в выражениях Angular. Смотрите `ng.$filter` для большей информацией об Angular массивах.

Использование

В привязке HTML шаблона

```
{{ filter_expression | filter:expression }}
```

В JavaScript

```
$filter('filter')(array, expression)
```

Параметры

- `array – {Array} –` входной массив.
- `expression – {string|Object|function()} –` Предикат для использование при выборе элементов из входного массива.

  Может быть одним из:

  - `строка`: Если предикат является подстрокой элемента входного массива, то этот элемент включается в результатирующий массив. Все строки или объекты со строковыми свойствами во входном массиве, которые содержат подстроку предикат будут возвращены. Предикат может быть с отрицанием, для этого нужно применить префикс `!`.

- `объект`: Объект-паттерн, который используется для фильтрации специфических свойств в объектах содержащихся в массиве. Например, предикат `{name:"M", phone:"1"}` вернет массив с элементами, у которых имеется свойство `name` содержащее "M" и имеется свойство `phone` содержащее "1". Если в качестве имени использовать `$` (как в `{$:"text"}`) то в результате будут искаться все объекты у которых любое свойство содержит требуемое значение. Это эквивалент простому текстовому предикату, расмотренному чуть раньше.
- `функция`: функция предикат может быть использована для реализации собственной логики фильтрации. Эта функция вызывается для каждого элемента входного массива. В результате в выходном массиве будут только элементы, для которых эта функция вернет true.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
  <body>
    <div ng-init="friends = [{name:'John', phone:'555-1276'},
                             {name:'Mary', phone:'800-BIG-MARY'},
                             {name:'Mike', phone:'555-4321'},
                             {name:'Adam', phone:'555-5678'},
                             {name:'Julie', phone:'555-8765'}]"></div>

    Search: <input ng-model="searchText">
    <table id="searchTextResults">
      <tr><th>Name</th><th>Phone</th></tr>
      <tr ng-repeat="friend in friends | filter:searchText">
        <td>{{friend.name}}</td>
        <td>{{friend.phone}}</td>
      </tr>
    </table>
    <hr>
    Any: <input ng-model="search.$"> <br>
    Name only <input ng-model="search.name"><br>
    Phone only <input ng-model="search.phone"å><br>
    <table id="searchObjResults">
      <tr><th>Name</th><th>Phone</th></tr>
      <tr ng-repeat="friend in friends | filter:search">
        <td>{{friend.name}}</td>
        <td>{{friend.phone}}</td>
      </tr>
    </table>
  </body>
</html>
```

End to end test

```
it('should search across all fields when filtering with a string', function() {
  input('searchText').enter('m');
  expect(repeater('#searchTextResults tr', 'friend in friends').column('friend.name')).
    toEqual(['Mary', 'Mike', 'Adam']);

  input('searchText').enter('76');
  expect(repeater('#searchTextResults tr', 'friend in friends').column('friend.name')).
    toEqual(['John', 'Julie']);
});

it('should search in specific fields when filtering with a predicate object', function() {
  input('search.$').enter('i');
  expect(repeater('#searchObjResults tr', 'friend in friends').column('friend.name')).
    toEqual(['Mary', 'Mike', 'Julie']);
});
```

json

Описание

Позволяет конвертировать JavaScript объекты в строки JSON.

Этот фильтр обычно используется для отладки. Когда используется нотация с двумя фигурными скобками для привязки происходит автоматическая конвертация в JSON.

Использование

В привязке HTML шаблонов

```
{{ json_expression | json }}
```

В JavaScript

```
$filter('json')(object)
```

Параметры

- `object – {*}` – Любой объект JavaScript (включая массивы и примитивные типы) для фильтрации.

Возврат

`{string}` – строка JSON.

Пример

Index.html

```
<!doctype html>
<html ng-app>
```

```
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
  </head>
  <body>
    <pre>{{ {'name':'value'} | json }}</pre>
  </body>
</html>
```

End to end test

```
it('should jsonify filtered objects', function() {
  expect(binding("{'name':'value'}")).toMatch(/\{\n  "name": ?"value"\n}/);
});
```

limitTo

Описание

Создает новый массив содержащий только указанное количество элементов. Эти элементы берутся с начала или конца входного массива, в зависимости от знака переданного значения (плюс или минус) `limit`.

Примечание: Эта функция используется только с типом `Array` в выражениях Angular. Смотрите `ng.$filter` для получения большего количества информации об массивах Angular.

Использование

```
filter:limitTo(array, limit);
```

Параметры

- `array – {Array} –` Входной массив для усечения.
- `limit – {string|Number} –` Длина возвращаемого массива. Если длина `limit` положительная, элементы будут копироваться из начала входного массива. Если она отрицательна – из конца входного массива. Значение `limit` может быть усечено, если оно больше чем количество элементов во входном массиве.

Возврат

`{Array}` – Новый подмассив с указанным количеством элементов, или входной массив, если указанное количество элементов больше чем длина входного массива.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
```

```
  </head>
  <body>
    <div ng-controller="Ctrl">
      Limit {{numbers}} to: <input type="integer" ng-model="limit">
      <p>Output: {{ numbers | limitTo:limit }}</p>
    </div>
  </body>
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.numbers = [1,2,3,4,5,6,7,8,9];
  $scope.limit = 3;
}
```

End to end test

```
it('should limit the numer array to first three items', function() {
  expect(element('.doc-example-live input[ng-model=limit]').val()).toBe('3');
  expect(binding('numbers | limitTo:limit')).toEqual('[1,2,3]');
});

it('should update the output when -3 is entered', function() {
  input('limit').enter(-3);
  expect(binding('numbers | limitTo:limit')).toEqual('[7,8,9]');
});

it('should not exceed the maximum size of input array', function() {
  input('limit').enter(100);
  expect(binding('numbers | limitTo:limit')).toEqual('[1,2,3,4,5,6,7,8,9]');
});
```

lowercase

Описание

Конвертирует строку в нижний регистр.

Использование

В привязках в HTML шаблонах

```
{{ lowercase_expression | lowercase }}
```

В JavaScript

```
$filter('lowercase')()
```

number

Описание

Форматирует число в текст.

Если на вход передается не число, будут возвращена пустая строка.

Использование

В привязках HTML шаблонах

```
{{ number_expression | number[:fractionSize] }}
```

В JavaScript

```
$filter('number')(number[, fractionSize])
```

Параметры

- `number – {number|string} –` Число для форматирования.
- `fractionSize(optional=2) – {(number|string)=} –` Количество знаков для вывода дробной части числа.

Возврат

`{string}` – отформатированное число с запятой в качестве разделителя тысяч, и с соответствующим количеством знаков в дробной.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-controller="Ctrl">
      Enter number: <input ng-model='val'><br>
      Default formatting: {{val | number}}<br>
      No fractions: {{val | number:0}}<br>
      Negative number: {{-val | number:4}}
    </div>
  </body>
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.val = 1234.56789;
}
```

End to end test

```
it('should format numbers', function() {
  expect(binding('val | number')).toBe('1,234.568');
  expect(binding('val | number:0')).toBe('1,235');
  expect(binding('-val | number:4')).toBe('-1,234.5679');
});

it('should update', function() {
  input('val').enter('3374.333');
  expect(binding('val | number')).toBe('3,374.333');
  expect(binding('val | number:0')).toBe('3,374');
  expect(binding('-val | number:4')).toBe('-3,374.3330');
});
```

orderBy

Описание

Сортировка требуемого массива с использование указанного выражения предиката.

Примечание: эта функция используется только для типов `Array` в Angular выражениях. Смотрите `ng.$filter` для получения более подробной информации об массивах Angular.

Использование

```
filter:orderBy(array, expression[, reverse]);
```

Параметры

- `array – {Array} –` Массив для сортировки
- `expression – {function(*)|string|Array.<(function(*)|string)>} –` Предикат, который используется для сравнения между собой элементов входного массива.

  Может быть одним из следующих:

  - `функция`: Возвращает результат сравнения двух элементов. Результат этой функции будет отсортирован с использованием операторов <, =, >.
  - `строка`: содержит имя свойства объекта, по которому будет проводится сортировка, так 'name' будет сортировать объекты по их свойству 'name'. Необязательный префикс + или – предназначен для указания направления сортировки (например, +name или -name).
  - `массив`: Массив функций или строк предикатов. Для сортировки используется первый предикат в массиве, но если элементы окажутся эквивалентными, используется следующий предикат, и т.д.
- `reverse(optional) – {boolean=} –` Обратный порядок элементов в выходном массиве.

Возврат

`{Array}` – Отсортированная копия входного массива.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-controller="Ctrl">
      <pre>Sorting predicate = {{predicate}}; reverse = {{reverse}}</pre>
      <hr/>
      [ <a href="" ng-click="predicate=''">unsorted</a> ]
      <table class="friend">
        <tr>
          <th><a href="" ng-click="predicate = 'name'; reverse=false">Name</a>
              (<a href ng-click="predicate = '-name'; reverse=false">^</a>)</th>
          <th><a href="" ng-click="predicate = 'phone'; reverse=!reverse">Phone Number</a></th>
          <th><a href="" ng-click="predicate = 'age'; reverse=!reverse">Age</a></th>
        </tr>
        <tr ng-repeat="friend in friends | orderBy:predicate:reverse">
          <td>{{friend.name}}</td>
          <td>{{friend.phone}}</td>
          <td>{{friend.age}}</td>
        </tr>
      </table>
    </div>
  </body>
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.friends =
      [{name:'John', phone:'555-1212', age:10},
       {name:'Mary', phone:'555-9876', age:19},
       {name:'Mike', phone:'555-4321', age:21},
       {name:'Adam', phone:'555-5678', age:35},
       {name:'Julie', phone:'555-8765', age:29}]
  $scope.predicate = '-age';
}
```

End to end test

```
it('should be reverse ordered by aged', function() {
  expect(binding('predicate')).toBe('-age');
```

```
3.    expect(repeater('table.friend', 'friend in friends').column('friend.age')).
4.      toEqual(['35', '29', '21', '19', '10']);
5.    expect(repeater('table.friend', 'friend in friends').column('friend.name')).
6.      toEqual(['Adam', 'Julie', 'Mike', 'Mary', 'John']);
7. });
8.
9. it('should reorder the table when user selects different predicate', function() {
10.   element('.doc-example-live a:contains("Name")').click();
11.   expect(repeater('table.friend', 'friend in friends').column('friend.name')).
12.     toEqual(['Adam', 'John', 'Julie', 'Mary', 'Mike']);
13.   expect(repeater('table.friend', 'friend in friends').column('friend.age')).
14.     toEqual(['35', '10', '29', '19', '21']);
15.
16.   element('.doc-example-live a:contains("Phone")').click();
17.   expect(repeater('table.friend', 'friend in friends').column('friend.phone')).
18.     toEqual(['555-9876', '555-8765', '555-5678', '555-4321', '555-1212']);
19.   expect(repeater('table.friend', 'friend in friends').column('friend.name')).
20.     toEqual(['Mary', 'Julie', 'Adam', 'Mike', 'John']);
21. });
```

uppercase

Описание

Переводит строку в верхний регистр.

Использование

В привязках HTML шаблонов

```
{{ uppercase_expression | uppercase }}
```

В JavaScript

```
$filter('uppercase')()
```

Сервисы

$anchorScroll

Описание

После вычислений, проверяет текущее значение `$location.hash()` и прокручивает экран к элементу, соответствующему роли описанной в спецификации Html5.

Он также следить за `$location.hash()` и осуществляет прокрутку снова при обнаружении изменений. Это поведение можно отключить вызвав `$anchorScrollProvider.disableAutoScrolling()`.

Зависимости

- $window
- $location
- $rootScope

Использование

```
$anchorScroll();
```

$cacheFactory

Описание

Фабрика которая создает объекты кэша.

Использование

```
$cacheFactory(cacheId[, options]);
```

Параметры

- `cacheId – {string} –` Имя или идентификатор для вновь созданного кэша.
- `options`*`(optional)`* `– {object=} –` Объект с настройками, которые специфицируют поведение кэша. Свойства:
  - `{number=} capacity` — превращает кэш в кэш LRU.

Возврат

`{object}` – Созданный объект кэша со следующими доступными методами:

- `{object} info()` — Возвращает идентификатор, размер и настройки для кэша.
- `{void} put({string} key, {*} value)` — Добавляет новую пару ключ-значение в кэш.
- `{{*}} get({string} key)` — Возвращает кэшированное значение для ключа или undefined если значения нет.
- `{void} remove({string} key)` — Удаляет пару ключ-значение из кэша.
- `{void} removeAll()` — Полностью очищает кэш, удаляя все пары ключ-значение.
- `{void} destroy()` — Удаляет ссылку на кэш из $cacheFactory.

$compile

Описание

Компилирует строку HTML или DOM в шаблон и предоставляет шаблону методы, которые могут быть использованы для связи `Области видимости` и шаблона в дальнейшем.

Процесс компиляции работает с деревом DOM и пытается найти в DOM элементы с `директивами`. Для каждой директивы выполняется соответствующая шаблонная функция и собираются функции экземпляра в одну шаблонную функцию, которая затем и возвращается.

Шаблонная функция может использоваться только для создания представления, или, как это происходит в случае с долговременными `повторителями`, для каждого вычисления результирующего представления клонирует оригинальной шаблон DOM.

Index.html

```
<!doctype html>
<html ng-app="compile">
```

```
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-controller="Ctrl">
      <input ng-model="name"> <br>
      <textarea ng-model="html"></textarea> <br>
      <div compile="html"></div>
    </div>
  </body>
</html>
```

Script.js

```
// декларация нового модуля и инъекция $compileProvider
angular.module('compile', [], function($compileProvider) {
  // настройка новой директивы 'compile', которая принимает в параметре
  // фабричную функцию. В нее делается инъекция сервиса '$compile'
  $compileProvider.directive('compile', function($compile) {
    // фабрика для создания директивы со связующей функцией
    return function(scope, element, attrs) {
      scope.$watch(
        function(scope) {
           // следим за изменениями выражения 'compile'
          return scope.$eval(attrs.compile);
        },
        function(value) {
          // когда выражение 'compile' изменено
          // применяем его к текущему DOM
          element.html(value);

          // компиляция текущего DOM и связывание его с текущей
          // областью видимости.
          // Примечание: мы компилируем только .childNodes чтобы не
          // не попасть в бесконечный цикл компилируя себя
          $compile(element.contents())(scope);
        }
      );
    };
  })
});

function Ctrl($scope) {
  $scope.name = 'Angular';
  $scope.html = 'Hello {{name}}';
}
```

End to end test

```
it('should auto compile', function() {
  expect(element('div[compile]').text()).toBe('Hello Angular');
  input('html').enter('{{name}}!');
  expect(element('div[compile]').text()).toBe('Angular!');
});
```

Использование

```
$compile(element, transclude, maxPriority);
```

Параметры

- `element – {string|DOMElement}` – Элемент или строка HTML для компиляции в шаблонную функцию.
- `transclude – {function(angular.Scope[, cloneAttachFn]}` – функция доступа к директивам.
- `maxPriority – {number}` – применяются только директивы с низким приоритетом (Только с действием на родительские элементы, но не на дочерние)

Возврат

`{function(scope[, cloneAttachFn])}` – связующая функция, которая будет использоваться для построения шаблона. (дерева DOM элементов) для области видимости. Где:

- `scope` - `Область видимости` для привязок.
- `cloneAttachFn` - Если `cloneAttachFn` предоставлена, тогда связующая функция будет клонировать шаблон и вызывать функцию `cloneAttachFn` чтобы привязать клонированные элементы к документу DOM в соответствующем расположении.
  Сигнатура `cloneAttachFn` такая:
  `cloneAttachFn(clonedElement, scope)` где:
  - `clonedElement` – клон оригинального элемента переданного в компилятор.
  - `scope` – текущая область видимости с которой работает связующая функция.

Вычисление связующей функции возвращает представление. В нее передается каждый оригинальный элемент, или клон элемента, если предоставлена функция `cloneAttachFn`.

После связывания представление не обновляется, пока не будет вызвана функция $digest, которая обычно запускается Angular автоматически.

Если нужно получить доступ к связанному представлению, можно использовать два способа:

- Если нам не нужно чтобы функция связывания клонировала шаблон, создайте элемент(ы) DOM перед отправкой их компилятору и сохраните на него ссылку.

  ```
  var element = $compile('<p>{{total}}</p>')(scope);
  ```

- Если вам нужно чтобы входной элемент клонировался, ссылка на представление из прошлого примера может быть клонируемым элементом. В этом случае в можете получить доступ к клону через cloneAttachFn:

  ```
  var templateHTML = angular.element('<p>{{total}}</p>'),
  ```

```
    scope = ....;

var clonedElement = $compile(templateHTML)(scope, function(clonedElement, scope) {
  //присоединение клона к DOM документу, как показано раннее
});

//сейчас мы имеем ссылку на клонированный DOM в `clone`
```

Для информации о работе компилятор, смотрите раздел HTML компилятор в Angular в руководстве разработчика.

$controller

Описание

Сервис `$controller` отвечает за создание контроллеров.

Он просто вызывает `$injector`, но извлекает его как службу, так что можно переопределить этот сервис на ВС верию.

Зависимости

- $injector

Использование

```
$controller(constructor, locals);
```

Parameters

- `constructor - {Function|string} -` Если это функция, то считается что это функция конструктор контроллера. В противном случае, считается что это строка, которая будет использоваться для извлечения конструктора контроллера, используя следующие шаги:
  - проверяет, зарегистрирован ли контроллер с нужным именем в сервисе `$controllerProvider`
  - проверяет, есть ли в текущей области видимости свойство с требуемым именем, которое возвращает конструктор
  - проверяет `window[constructor]` в глобальном объекте `window`
- `locals - {Object} -` Зависимости внедряемые в контроллер.

Возврат

`{Object}` – экземпляр требуемого контроллера.

$document

Описание

Это `jQuery (lite)`-обертка, ссылающаяся на элемент браузера `window.document`.

Зависимости

- $window

$exceptionHandler

Описание

Любое не перехваченное исключение в angular выражениях обрабатывается этим сервисом. По умолчанию реализована обработка с помощью `$log.error`, которая просто выводит сообщение об исключении в консоль браузера.

В юнит тестах, если загружен `angular-mocks.js`, этот сервис будет переопределен на mock $exceptionHandler с вспомогательными средствами для тестирования.

Зависимости

- $log

Использование

```
1. $exceptionHandler(exception[, cause]);
```

Параметры

- `exception – {Error} –` Исключение, ассоциированное с ошибкой.
- `cause(optional) – {string=} –` необязательная информация о контексте в котором возникла ошибка.

$filter

Описание

Фильтры используются для форматирования данных показываемых пользователю.

Главный синтаксис в шаблонах следующий:

```
    {{ expression [| filter_name[:parameter_value] ... ] }}
```

Использование

```
1. $filter(name);
```

Параметры

- `name – {String} –` Имя фильтра

Возврат

`{Function}` – функция фильтрации

$http

Описание

Сервис `$http` является базовым сервисом Angular, который используется для коммуникаций с удаленным HTTP сервером с помощью браузерного объекта XMLHttpRequest или JSONP.

Для юнит тестирования приложения использующего сервис `$http`, смотрите $httpBackend mock.

Для изучения высокоуровневых абстракций, посмотрите сервис $resource.

$http API базируется на `deferred/promise APIs` предоставленном сервисом $q. Для простых приложений изучение этих API не представляется важным, тогда как для продвинутого использования, очень важно с ними ознакомится и знать возможности, которые они предоставляют.

Главный способ использования

Сервис `$http` это функция, которая принимает один аргумент — объект с настройками — который используется для генерации HTTP запроса, и возвращает `promise` с двумя определенными в $http методами: `success` и `error`.

```
1. $http({method: 'GET', url: '/someUrl'}).
2.   success(function(data, status, headers, config) {
3.     // эта функция обратного вызова выполнится асинхронно
4.     // когда придет ответ с сервера
5.   }).
6.   error(function(data, status, headers, config) {
7.     // эта функция выполняется асинхронно, если произошла ошибка
8.     // или пришел ответ с сервера с установленным статусом ошибки
9.   });
```

Так как возвращаемым значением функции $http является `promise`, вы можете использовать метод `then` чтобы регистрировать функции обратного вызова, и они будут получать только один аргумент – объект представляющий ответ сервера. Смотрите определение API и типов для более детальной информации.

Ответ сервера со статусом в диапазоне от 200 до 299 считается успешным статусом, и в результате будет вызвана функция обратного вызова success. Заметьте, если ответом сервиса является перенаправление на другую страницу, XMLHttpRequest будет ожидаемо следовать туда, а это означает что функция обратного вызова error для таких ответов вызываться на будет.

Сокращенные методы

Каждый вызов сервиса $http обязательно принимает в параметрах HTTP метод и URL, и POST/PUT запросы к тому же обязательно передают данные, для сокращения их вызова были созданы сокращенные метода:

```
1. $http.get('/someUrl').success(successCallback);
2. $http.post('/someUrl', data).success(successCallback);
```

Полный список сокращенных методов:

- `$http.get`
- `$http.head`

- `$http.post`
- `$http.put`
- `$http.delete`
- `$http.jsonp`

Установка HTTP заголовков

Сервис $http будет автоматически добавлять некоторые HTTP заголовки для всех запросов. Значения по умолчанию могут быть настроены через доступ к конфигурационному объекту `$httpProvider.defaults.headers`, который сейчас содержит конфигурацию по умолчанию:

- `$httpProvider.defaults.headers.common` (заголовки общие для всех запросов):
  - `Accept: application/json, text/plain, * / *`
  - `X-Requested-With: XMLHttpRequest`
- `$httpProvider.defaults.headers.post`: (заголовок по умолчанию для POST запросов)
  - `Content-Type: application/json`
- `$httpProvider.defaults.headers.put` (заголовок по умолчанию для PUT запросов)
  - `Content-Type: application/json`

Чтобы добавить или переопределить значения по умолчанию, просто добавьте или удалите свойства из конфигурационных объектов. Чтобы добавить заголовки для HTTP метода, отличного от POST или PUT, просто добавьте новый объект с именем HTTP метода в нижнем регистре в качестве ключа, например так, `$httpProvider.defaults.headers.get['My-Header']='value'`.

Дополнительно, значения по умолчанию могут быть установлены во время выполнения с помощью объекта `$http.defaults` в таком же стиле, как и разъясненном ранее.

Преобразование запросов и ответов

И запросы, и ответы могут быть преобразованы с использованием функций трансформации. По умолчанию, Angular применяет эти преобразования:

Преобразования запросов:

- Если свойство `data` запроса содержит объект, то он сериализуется в формат JSON.

Преобразования ответов:

- Если обнаружен префикс XSRF, удаляет его (смотрите раздел вопросы безопасности ниже).
- Если обнаружен что ответ в формате JSON, десериализует его используя анализатор JSON.

Для добавления или переопределения преобразований по умолчанию, модифицируйте свойства `$httpProvider.defaults.transformRequest` и `$httpProvider.defaults.transformResponse`. Эти свойства содержат массивы функций преобразований, которые позволяют вам добавлять с помощью методов массива `push` или `unshift` новые функции трансформации в цепочку преобразований. Вы также можете полностью переопределить любые преобразования по умолчанию, присвоив свои функции преобразования этим свойствам напрямую, без массива обертки.

Кроме того, для локального переопределения преобразований запроса/ответа, расширьте объект передаваемый `$http` свойством `transformRequest` и/или `transformResponse` и передайте им объект с конфигурацией.

Кэширование

Чтобы включить кэширование, установите конфигурационное свойство `cache` в `true`. Когда кэширование включено, `$http` сохраняет ответы сервера в локальном кэше. В последующем ответ сервера извлекается из кэша без отправки запроса.

Следует отметить, что даже если данные выдаются из кэша, это делается асинхронным способом, так же как и при отправке запроса на сервер.

Если есть несколько GET запросов на один и тот же URL они кэшируются в один кэш, но кэш пуст, пока не пройдет первый запрос, а все остальные уже будут использовать ответ сервера из кэша.

Перехватчики ответов

Перед началом создания перехватчиков, обязательно изучите `$q and deferred/promise APIs`.

Для целей глобальной обработки ошибок, аутентификации, или любой дочерней синхронной или асинхронной предобработки полученных ответов, желательно иметь возможность перехватывать ответы на http запросы перед тем, как они будут переданы на обработку коду приложения, инициировавшему запросы. Перехватчики ответов используют `promise apis` когда им нужно предобработка в обоих, синхронной и асинхронной манере.

Перехватчики – это фабрики сервисов, которые регистрируются в $httpProvider путем добавления их в массив `$httpProvider.responseInterceptors`. Фабрика создает и внедряет зависимости (если нужно) и возвращает перехватчика – это функция которая работает с `promise` и возвращает оригинальный или новый promise.

```
1. // регистрация сервиса перехватчика
2. $provide.factory('myHttpInterceptor', function($q, dependency1, dependency2) {
3.   return function(promise) {
4.     return promise.then(function(response) {
5.       // что-то делает при успешном статусе ответа сервера
6.     }, function(response) {
7.       // что-то делает при ошибке
8.       if (canRecover(response)) {
9.         return responseOrNewPromise
10.      }
11.      return $q.reject(response);
12.    });
13.  }
14. });
15.
16. $httpProvider.responseInterceptors.push('myHttpInterceptor');
17.
18.
19. // регистрация перехватчика как анонимной функции
20. $httpProvider.responseInterceptors.push(function($q, dependency1, dependency2) {
21.   return function(promise) {
22.     // что-то делаем
23.   }
24. });
```

Соображения безопасности

Когда создаются веб-приложения, угрозы безопасности исходят из:

- JSON уязвимостей

- XSRF

Оба, сервер и клиент, должны работать совместно для ликвидации этих угроз. Angular поставляется с предварительно настроенными стратегиями, учитывающими эти вопросы, но также требуется сотрудничество сервера.

Защита от JSON уязвимостей

JSON уязвимости позволяют веб-сайту третьих лиц подменить ваш URL для ресурса JSON, на запрос JSONP при определенных условиях. Чтобы этому противостоять, ваш сервер может добавлять префикс `")]}',\n"` для всех ответов на запросы в формате JSON. Angular будет автоматически вырезать префикс перед обработкой ответа как JSON.

Для примера, если ваш сервер должен вернуть:

```
1. ['one','two']
```

что уязвимо для атак, ваш сервер может вернуть:

```
1. )]}',
2. ['one','two']
```

Angular будет удалять префикс, перед обработкой JSON.

Защита от Cross Site Request Forgery (XSRF)

XSRF эта техника, с помощью которой сайт злоумышленника может получить секретные данные вашего пользователя. Angular предоставляет механизм для противодействия XSRF. Когда выполняются XHR запросы, сервис $http читает маркер из куки `XSRF-TOKEN` и устанавливает его как HTTP заголовок `X-XSRF-TOKEN`. Поскольку только JavaScript, который работает в вашем домене может читать куки, ваш сервер может быть уверен, что XHR пришло из кода JavaScript выполняемого на вашем домене.

Чтобы это заработало, нужно чтобы ваш сервер установил метку в доступной для чтения из JavaScript сессионной куки с именем `XSRF-TOKEN` в ответ на первый HTTP GET запрос. Следующие XHR запросы сервер может проверить, сравнивая значение куки и присланного HTTP заголовка `X-XSRF-TOKEN`, чтобы убедиться что запрос не поддельный. Метка должна быть уникальной для каждого пользователя и проверяемой для сервера (препятствуйте тому, чтобы JavaScript сам генерировал матки). Мы рекомендуем чтобы метка служила основой для процесса аутентификации на вашем сайте с добавление соли для большей безопасности.

Зависимости

- $httpBackend
- $browser
- $cacheFactory
- $rootScope
- $q
- $injector

Использование

```
1. $http(config);
```

Параметры

- `config – {object} –` Объект описывающий создаваемый запрос. Этот объект имеет следующие свойства:
  - **method** – `{string}` – HTTP метод (пример, 'GET', 'POST', и т.д.)
  - **url** – `{string}` – Абсолютный или относительный URL для ресурса к которому будет запрос.
  - **params** – `{Object.<string|Object>}` – Карта из строк или объектов, которые будут переданы как `?key1=value1&key2=value2` после url. Если значения не строка, тогда оно будет преобразовано в строку в формате JSON.
  - **data** – `{string|Object}` – Данные которые будут отправлены вместе с запросом.
  - **headers** – `{Object}` – Карта из строк, представляющих HTTP заголовки, которые должны быть отправлены на сервер.
  - **transformRequest** – `{function(data, headersGetter)|Array.<function(data, headersGetter)>}` – функция преобразования или массив таких функций. Функция преобразования берет тело и заголовки http запроса и возвращает преобразованную (по умолчанию сериализованую) версию.
  - **transformResponse** – `{function(data, headersGetter)|Array.<function(data, headersGetter)>}` – функция преобразования или массив таких функций. Функция преобразования берет тело и заголовки http ответа и возвращает преобразованную (по умолчанию десериализованую) версию.
  - **cache** – `{boolean|Cache}` – Если true, по умолчанию кэш $http будет использоваться для кэширования GET запросов, однако если экземпляр кэша построен с помощью `$cacheFactory`, он может быть использован для кэширования.
  - **timeout** – `{number}` – задержка по времени в миллисекундах.
  - **withCredentials** - `{boolean}` – следует ли устанавливать флаг `withCredentials` для XHR объекта. Смотрите доверенные запросы для более детальной информациию

Возврат

`{HttpPromise}` – Возвращает объект `promise` со стандартным методом `then` и двумя специфичными методами:`success` и `error`. Метод `then` имеет два аргумента, первый – функция обратного вызова, которая будет выполнена в случае успешного ответа, второй – функция обратного вызова, которая будет выполнена в случае ошибки. Методы `success` и `error` имеют по одному аргументу – функция, которая будет выполнена в случае успешного завершения запроса или ошибки соответственно. Аргументы переданы в эти функции это деструктурированный объект ответа сервера переданный в метод `then`. Объект ответа от сервера имеет свойства:

- **data** – `{string|Object}` – Тело ответа преобразованное с помощью функций преобразования.
- **status** – `{number}` – код HTTP статуса ответа.
- **headers** – `{function([headerName])}` – Функция возвращающая заголовок по названию.
- **config** – `{Object}` – Конфигурационный объект, который был использован для создания запроса.

Методы

delete(url, config)

Сокращение для отправки запроса `DELETE`.

Параметры:

- `url – {string} –` Относительный или абсолютный URL по которому отправляется запрос
- `config(optional) – {Object=} –` Необязательный конфигурационный объект

Возврат:

`{HttpPromise}` – Объект будущего

get(url, config)

Сокращение метода для запроса `GET`.

Параметры:

- `url – {string} –` Относительный или абсолютный URL по которому отправляется запрос
- `config`*`(optional)`* `– {Object=} –` Необязательный конфигурационный объект

Возврат:

`{HttpPromise}` – Объект будущего

head(url, config)

Сокращение для запроса `HEAD`.

Параметры:

- `url – {string} –` Относительный или абсолютный URL по которому отправляется запрос
- `config`*`(optional)`* `– {Object=} –` Необязательный конфигурационный объект

Возврат:

`{HttpPromise}` – Объект будущего

jsonp(url, config)

Сокращение для запроса `JSONP`.

Параметры:

- `url – {string} –` Относительный или абсолютный URL по которому отправляется запрос. Должен содержать строку `JSON_CALLBACK`.
- `config`*`(optional)`* `– {Object=} –` Необязательный конфигурационный объект

Возврат:

`{HttpPromise}` – Объект будущего

post(url, data, config)

Сокращение для запроса `POST`.

Параметры:

- `url – {string} –` Относительный или абсолютный URL определенный для обработки запроса
- `data – {*} –` данные запроса
- `config`*`(optional)`* `– {Object=} –` Необязательный конфигурационный объект

Возврат:

`{HttpPromise}` – Future object

put(url, data, config)

Сокращение для запроса `PUT`.

Параметры:

- `url – {string} –` Относительный или абсолютный URL предназначенный для обработки запроса
- `data – {*} –` Данные запроса
- `config(optional) – {Object=} –` Необязательный конфигурационный объект

Параметры

`{HttpPromise}` – объект будущего

Properties

defaults

Эквивалент времени выполнения для свойства `$httpProvider.defaults`. Позволяет настроить заголовки по умолчанию для запросов и преобразователей для ответов сервера.

Смотрите выше главы "Настройка HTTP заголовков" и "Преобразование запросов и ответов".

pendingRequests

Массив конфигурационных объектов для текущих ожидающих запросов. Используется главным образом для целей отладки.

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-controller="FetchCtrl">
      <select ng-model="method">
        <option>GET</option>
        <option>JSONP</option>
      </select>
      <input type="text" ng-model="url" size="80"/>
      <button ng-click="fetch()">fetch</button><br>
      <button ng-click="updateModel('GET', 'http-hello.html')">Sample GET</button>
      <button ng-click="updateModel('JSONP', 'http://angularjs.org/greet.php?callback=JSON_CALLBACK&name=Super%20Hero')">Sample JSONP</button>
      <button ng-click="updateModel('JSONP', 'http://angularjs.org/doesntexist&callback=JSON_CALLBACK')">Invalid JSONP</button>
      <pre>http status code: {{status}}</pre>
      <pre>http response data: {{data}}</pre>
```

```
    </div>
  </body>
</html>
```

http-hello.html

```
Hello, $http!
```

Script.js

```
function FetchCtrl($scope, $http, $templateCache) {
  $scope.method = 'GET';
  $scope.url = 'http-hello.html';

  $scope.fetch = function() {
    $scope.code = null;
    $scope.response = null;

    $http({method: $scope.method, url: $scope.url, cache: $templateCache}).
      success(function(data, status) {
        $scope.status = status;
        $scope.data = data;
      }).
      error(function(data, status) {
        $scope.data = data || "Request failed";
        $scope.status = status;
    });
  };

  $scope.updateModel = function(method, url) {
    $scope.method = method;
    $scope.url = url;
  };
}
```

End to end test

```
it('should make an xhr GET request', function() {
  element(':button:contains("Sample GET")').click();
  element(':button:contains("fetch")').click();
  expect(binding('status')).toBe('200');
  expect(binding('data')).toMatch(/Hello, \$http!/);
});

it('should make a JSONP request to angularjs.org', function() {
  element(':button:contains("Sample JSONP")').click();
  element(':button:contains("fetch")').click();
  expect(binding('status')).toBe('200');
  expect(binding('data')).toMatch(/Super Hero!/);
```

```
});

it('should make JSONP request to invalid URL and invoke the error handler',
    function() {
  element(':button:contains("Invalid JSONP")').click();
  element(':button:contains("fetch")').click();
  expect(binding('status')).toBe('0');
  expect(binding('data')).toBe('Request failed');
});
```

$httpBackend

Описание

HTTP backend используется как `сервис` который исправляет несовместимости объектов XMLHttpRequest или JSONP в различных браузеров.

Вы не должны использовать этот сервис явно, вместо этого используйте его абстракции более высокого уровня: `$http` или $resource.

В ходе тестирования этот объект может быть заменен объектом mock $httpBackend с заранее определенными ответами.

Зависимости

- $browser
- $window
- $document

$interpolate

Описание

Компилирует строку с разметкой в функцию интерполяции. Это сервис используется сервисом компиляции HTML `$compile` для привязки данных. Смотрите `$interpolateProvider` для настройки разметки интерполяции.

```
        var $interpolate = ...; // injected

        var exp = $interpolate('Hello {{name}}!');

        expect(exp({name:'Angular'}).toEqual('Hello Angular!');
```

Зависимости

- $parse

Использование

```
$interpolate(text[, mustHaveExpression]);
```

Параметры

- `text – {string} –` Текст с разметкой для интерполяции.
- `mustHaveExpression(optional) – {boolean=} –` Если установлено в true, тогда строка интерполяции должна иметь встроенные выражения, чтобы вернуть функцию интерполяции. Строка, в которой нет встроенных выражений, вместо функции интерполяции будет возвращать null.

Возврат

`{function(context)}` – функция интерполяции, которая используется для компиляции интерполируемой строки. Она имеет следующие параметры:

- `context`: объект, который принимает участие в вычислении любых встроенных выражений.

Методы

endSymbol()

Символ, указывающий на конец интерполируемой строки. По умолчанию это `}}`.

Используйте `$interpolateProvider#endSymbol` для изменения символа.

Возврат:

`{string}` – конечный символ.

startSymbol()

Символ, указывающий на начало интерполируемой строки. По умолчанию это `{{`.

Используйте `$interpolateProvider#startSymbol` для изменения символа начала.

Возврат:

`{string}` – символ начала.

$locale

Описание

Служба $locale предоставляет локализационные роли для различных компонентов Angular. По нынешнему состоянию имеет публичное api:

- `id – {string} –` локализационный идентификатор форматирования `языкаИдентификатор-страныИдентификатор` (например, `en-us`)

$location

Описание

Сервис $location разбирает URL в адресной строке браузера (базируется на window.location) и делает URL доступной для вашего приложения. Изменение URL в адресной строке отражается на сервис $location и изменения в $location отображаются в адресной строке браузера.

Сервис $location:

- Показывает текущий URL в адресной строке браузера, и вы можете
  - Отслеживать изменение URL.

    - Изменять URL.
- Синхронизирует URL с браузером, когда пользователь
    - Изменит адресную строку
    - Кликнет на кнопке вперед или назад (или кликнет на ссылке в истории).
    - Кликнет по ссылке.
- Изменяет объект URL с помощью методов для установки (protocol, host, port, path, search, hash).

Для получения более подробной информации смотрите Руководство разработчика:Службы Angular: Использование $location

Зависимости

- $browser
- $sniffer
- $rootElement

Методы

absUrl()

Этот метод только получает значение.

Возвращает полное представление url с закодированными сегментами в соответствии со спецификацией RFC 3986.

Возврат:

`{string}` – полный url

hash(hash)

Этот метод применяется для получения и установки значения.

Возвращает фрагмент хэша когда вызывается без параметров.

Изменяет фрагмент хэша, когда вызывается с параметром и возвращает `$location`.

Параметры:

- `hash(optional) – {string=} –` Новый фрагмент хэша

Возврат

`{string}` – фрагмент хэша

host()

Этот метод только возвращает значение.

Возвращает хост для текущего url.

Возврат:

`{string}` – хост для текущего url.

path(path)

Этот метод возвращает и задает значение пути.

Возвращает путь для текущего url когда вызывается без параметров.

Изменяет путь, когда вызывается с параметром, и возвращает `$location`.

Примечание: Путь должен всегда начинаться со слеша (/), этот метод будет добавлять начальный слеш, если он пропущен.

Параметры:

- `path(optional)` – `{string=}` – Новый путь

Возврат:

`{string}` – путь

port()

Этот метод возвращает значение порта, для текущего url.

Возврат:

`{Number}` – порт

protocol()

Этот метод возвращает протокол для текущего url.

Возврат:

`{string}` – протокол для текущего url

replace()

Если вызван, все изменения $location в текущем цикле `$digest` будут перемещены текущая запись истории, и будет создана и вставлена новая.

search(search, paramValue)

Этот метод возвращает и задает строку поиска.

Возвращает строку поиска (как объект) для текущего url когда вызывается без параметров.

Изменяет строку параметров, когда вызывается с параметрами и возвращает `$location`.

Параметры:

- `search(optional)` – `{string|object<string,string>=}` – Новая строка поиска, или хэш объект
- `paramValue(optional)` – `{string=}` – Если `search` строка, тогда `paramValue` будет переопределять только первый параметр поиска. Если значение задано в `null`, тогда параметр будет удален.

Возврат:

`{string}` – строка поиска в виде объекта

url(url)

Этот метод возвращает и устанавливает текущий url.

Возвращает url (например, `/path?a=b#hash`) когда вызывается без параметров.

Изменяет путь, строку поиска и хэш, когда вызывается с параметром и возвращает `$location`.

Параметры:

- `url(optional)` – {string=} – Новый url без базового адреса (например, `/path?a=b#hash`)

Возврат:

{string} – url

$log

Описание

Простой сервис для логирования. По умолчанию реализована запись сообщений в консоль браузера (если она предоставлена).

Главная цель данного сервиса это простая отладка и разрешение проблем.

Зависимости

- $window

Методы

error()

Записывает сообщение об ошибке

info()

Записывает информационное сообщение

log()

Записывает лог сообщение

warn()

Записывает сообщение с предупреждением

Пример

Index.html

```
1. <!doctype html>
2. <html ng-app>
3.   <head>
4.     <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
5.     <script src="script.js"></script>
6.   </head>
7.   <body>
8.     <div ng-controller="LogCtrl">
9.       <p>Reload this page with open console, enter text and hit the log button...<
   /p>
10.      Message:
11.      <input type="text" ng-model="message"/>
12.      <button ng-click="$log.log(message)">log</button>
13.      <button ng-click="$log.warn(message)">warn</button>
```

```
      <button ng-click="$log.info(message)">info</button>
      <button ng-click="$log.error(message)">error</button>
    </div>
  </body>
</html>
```

Script.js

```
function LogCtrl($scope, $log) {
  $scope.$log = $log;
  $scope.message = 'Hello World!';
}
```

$parse

Описание

Конвертирует Angular выражение в функцию.

```
var getter = $parse('user.name');
var setter = getter.assign;
var context = {user:{name:'angular'}};
var locals = {user:{name:'local'}};

expect(getter(context)).toEqual('angular');
setter(context, 'newValue');
expect(context.user.name).toEqual('newValue');
expect(getter(context, locals)).toEqual('local');
```

Использование

```
$parse(expression);
```

Параметры

- `expression – {string} –` Строковое выражение для компиляции.

Возврат

`{function(context, locals)}` – функция, которая представляет скомпилированное выражение:

- `context` – `{object}` – объект, на котором вычисляются любые встроенные выражения (типично это область видимости).
- `locals` – `{object=}` – локальные переменные объекта контекста, используются для переопределения значений в `context`.

  Возвращенная функция также имеет свойство `assign`, если выражения назначаемые, тогда это позволяет установить значения для выражений.

$q

Описание

Это promise/deferred реализация, вдохновленная Kris Kowal's Q.

CommonJS Promise описывает promise как интерфейс для взаимодействия с объектом, который предоставляет результат своих действий в асинхронной манере, и может в момент взаимодействия быть еще неопределенным.

С точки зрения борьбы с ошибками, deferred и promise API ключевые слова `try`, `catch` и `throw` программируются асинхронно даже в синхронном программах.

```
1.  // для целей этого примера, посмотрите применение переменных `$q` и `scope`,
2.  // доступных в текущей области видимости (они должны быть инъекцированы или вставл
    ены в нее).
3.
4.  function asyncGreet(name) {
5.    var deferred = $q.defer();
6.
7.    setTimeout(function() {
8.      // функция выполняется асинхронно в будущем цикле событий, нам нужно обернуть
9.      // ваш код и передать в вызов $apply чтобы уведомить об его изменениях.
10.     scope.$apply(function() {
11.       if (okToGreet(name)) {
12.         deferred.resolve('Hello, ' + name + '!');
13.       } else {
14.         deferred.reject('Greeting ' + name + ' is not allowed.');
15.       }
16.     });
17.   }, 1000);
18.
19.   return deferred.promise;
20. }
21.
22. var promise = asyncGreet('Robin Hood');
23. promise.then(function(greeting) {
24.   alert('Success: ' + greeting);
25. }, function(reason) {
26.   alert('Failed: ' + reason);
27. });
```

Не очевидно, для чего это нужно и зачем так усложнять код. Выигрыш появляется в результате гарантий выполнения promise и deferred API.

Дополнительно promise api позволяет использовать композицию, что очень трудно сделать с традиционным подходом (CPS) к функциям обратного вызова. Для большей информации смотрите Q документацию, особенно раздел о последовательном и параллельном присоединении promise.

Deferred API

Новый экземпляр объекта deferred создается вызовом функции `$q.defer()`.

Цель объекта deferred используя ассоциированный экземплят Promise уведомить об успешном или нет завершении асинхронной задачи.

Методы

- `resolve(value)` – разрешает производный promise значением `value`. Если значение value отвергнуто с помощью конструктора `$q.reject`, promise будет отвергнут.
- `reject(reason)` – отвергает производный promise с указанием причины `reason`. Это эквивалентно разрешению с использованием конструктора `$q.reject`.

Свойства

- promise – `{Promise}` – объект promise ассоциированный с deferred.

Promise API

Новый экземпляр promise создается когда создается экземпляр объекта и может извлекаться с помощью метода `deferred.promise`.

Целью объекта promise является позволить заинтересованным объектам получить доступ к результату отложенной (deferred) задаче, когда она закончится.

Методы

- `then(successCallback, errorCallback)` – независимо от того, когда promise было или будет разрешено или отвергнуто, выполнится одна из функций обратного вызова, как только результат станет доступным. В функцию обратного вызова передается один аргумент – результат или причина неудачи.

  Этот метод возвращает новый *promise*, который разрешается или отвергается с возвращаемым значением для `successCallback` или `errorCallback`.

Цепочки promise

Потому что вызов `then` возвращает новый отложенный promise, это позволяет легко создавать цепочку из promise:

```
1. promiseB = promiseA.then(function(result) {
2.   return result + 1;
3. });
4.
5. // promiseB будет разрешен после разрешения promiseA и его значением будет
6. // результат promiseA увеличенный на 1
```

Это позволяет создавать цепочки из любого количества promise которые могут разрешаться другими promise (каждый отложит свое разрешение), это позволяет установить паузу/отложить определение promise в любой части цепочки. Это делает простым реализацию полного api для перехватчика ответов $http.

Различия между Q Криса Ковала и $q

Они имеют три главных отличия:

- $q интегрирован с `ng.$rootScope.Scope` Модель области видимости имеет механизм наблюдения, что означает быстрое распространение разрешения или отказа в вашу модель,

избегая ненужных перерисовок в браузере, что привело бы к мерцанию пользовательского интерфейса.
- $q promise признаются движком шаблонов в angular, что означает, что в шаблонах вы можете использовать promise в области видимости, так, как будто это результирующие значения.
- Q имеет больше возможностей, чем $q, но это делает его более увесистым. $q маленький, но содержит всю требующуюся функциональность для реализации асинхронных задач.

Тестирование

```
1. it('should simulate promise', inject(function($q, $rootScope) {
2.   var deferred = $q.defer();
3.   var promise = deferred.promise;
4.   var resolvedValue;
5.
6.   promise.then(function(value) { resolvedValue = value; });
7.   expect(resolvedValue).toBeUndefined();
8.
9.   // Симуляция разрешения для promise
10.  deferred.resolve(123);
11.  // Заметьте, что функция 'then' не вызывается синхронно.
12.  // Это потому, что мы хотим чтобы promise API всегда работало асинхронно,
13.  // не зависимо от синхронного или асинхронного вызова.
14.  expect(resolvedValue).toBeUndefined();
15.
16.  // Распространение promise разрешений функцией 'then' используя $apply().
17.  $rootScope.$apply();
18.  expect(resolvedValue).toEqual(123);
19.});
```

Зависимости

- $rootScope

Методы

all(promises)

Комбинирует несколько promise в один promise, и разрешается когда все входные promise будут разрешены.

Параметры:

- `promises – {Array.<Promise>} –` Массив объектов promise.

Возврат

`{Promise}` – Возвращает единственный promise, который будет разрешен с массивом значений, каждое значение которого соответствует promise с тем же индексом, что и в массиве `promises`. Если любой из promise будет отвергнут, результат общего promise также будет отвергнут.

defer()

Создает объект `Deferred` который представляет задачу, которая будет завершена в будущем.

Возврат:

`{Deferred}` –

Возвращает новый экземпляр объекта deferred.

reject(reason)

Создает promise с состоянием отвергнут с указанием причины этого `reason`. Это будет отвергать всю вышестоящую цепочку promise. Если в отвергните последний promise в цепочке, вам не нужно делать это для остальных.

Если сравнивать deferred/promise с поведением обработки ошибок try/catch/throw, тогда `reject` это как ключевое слово `throw` в JavaScript. Это также означает, что если вы «ловите» ошибку через функцию обратного вызова promise и вы хотите переслать ошибку из текущего другому promise, вы используете повторный вызов "throw" , а здесь возврат ошибки построен через вызов `reject`.

```
promiseB = promiseA.then(function(result) {
  // success: сделать что-то и разрешить promiseB
  //          со старым или новым результатом
  return result;
}, function(reason) {
  // error: обработка ошибки, если получилось обработать,
  //        разрешаем promiseB с newPromiseOrValue,
  //        иначе передаем отклонено в promiseB
  if (canHandle(reason)) {
   // обработка ошибок и востановление
   return newPromiseOrValue;
  }
  return $q.reject(reason);
});
```

Параметры:

- `reason – {*} –` Константа, сообщение, исключение или любой другой объект, объясняющий причину отказа.

Возврат:

`{Promise}` – Возвращает promise разрешенный отказом с причиной `reason`.

when(value)

Обертывает объект, который может стать значением или возможно promise в $q promise. Это полезно, когда вы имеете дело с объектом, которое может быть или не быть promise, или если promise исходит от источника, которому нет доверия.

Параметры:

- `value – {*} –` Значение или promise

Возврат

`{Promise}` – возвращает promise для переданного значения или promise

$rootElement

Корневой элемент приложения Angular. Это элемент в котором определена директива `ngApp` или элемент, переданный в метод `angular.bootstrap`. В нем публикуется ссылка на сервис `$injector` для приложения, который можно получить с помощью `$rootElement.injector()`.

$rootScope

Описание

Каждое приложение имеет одну корневую область видимости. Все другие области видимости являются для нее дочерними. Области видимости предоставляют механизм для отслеживания изменений в модели и предоставляют цикл обработки событий. Смотрите руководство разработчика: области видимости.

$route

Описание

Используется для связывания URL с контроллерами и представлениями (HTML частями). Он отслеживает изменения `$location.url()` и пытается по карте путей найти для него существующее определение.

Вы можете определить маршруты через `$routeProvider` API.

Сервис `$route` обычно используется совместно с директивой `ngView` и сервисом `$routeParams`.

Зависимости

- $location
- $routeParams

Методы

reload()

Пытается перезагрузить текущий маршрут, если `$location` не изменен.

Как результат `ngView` создает новую область видимости, и пересоздается контроллер.

Свойства

current

Ссылается на текущее определение маршрута. Которое содержит:

- `controller`: Конструктор контроллера, который определен в определении маршрута.
- `locals`: Локальная карта, которая используется для сервиса `$controller` для создания экземпляра контроллера. `locals` содержит разрешенные значения в карте `resolve`. Дополнительно `locals` также содержит:
  - `$scope` – Текущая область видимости для маршрута.
  - `$template` – Шаблон HTML для текущего маршрута.

routes

Массив всех настроенных маршрутов.

Events

$routeChangeError

Транслируется, если любой из разрешаемых promise был отвергнут.

Тип: нисходящее

Цель: корневая область видимости

Параметры:

- `current – {Route}` – Информация о текущем маршруте.
- `previous – {Route}` – Информация о предыдущем маршруте.
- `rejection – {Route}` – Отвергнут ли promise. Обычно ошибка или плохой promise.

$routeChangeStart

Транслируется перед изменением маршрута. В этой точке сервис route начинает разрешение всех требуемых зависимостей, нужных для изменения маршрута. Обычно это включает извлечение шаблона представления, так и любых других зависимостей, определенных в свойстве маршрута `resolve`. После того как все зависимости будут разрешены, генерируется событие `$routeChangeSuccess`.

Тип: нисходящее

Цель: корневая область видимости

Параметры:

- `next – {Route}` – Информация о маршруте для перехода.
- `current – {Route}` – Информация о текущем маршруте.

$routeChangeSuccess

Транслируется когда все зависимости маршрута будут разрешены. `ngView` слушает его для директив чтобы создать контроллер и отобразить представление.

Тип: нисходящее

Цель: корневая область видимости

Параметры:

- `angularEvent – {Object}` – Синтезированный объект события.
- `current – {Route}` – Информация о текущем маршруте.
- `previous – {Route|Undefined}` – Информация о предыдущем маршруте, или undefined, если текущий маршрут первый.

$routeUpdate

Транслируется когда свойство `reloadOnSearch` установлено в false, и мы повторно используем тот же экземпляр контроллера.

Тип: нисходящее

Цель: корневая область видимости

Example

Этот пример показывает как в случае изменения хэша в URL сервис `$route` находит маршрут следующего URL, и `ngView` получает свое представление.

Заметьте, что этот пример использует строковые шаблоны, это сделано для чтобы получить его в процессе работы.

Index.html

```
<!doctype html>
<html ng-app="ngView">
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-controller="MainCntl">
      Choose:
      <a href="Book/Moby">Moby</a> |
      <a href="Book/Moby/ch/1">Moby: Ch1</a> |
      <a href="Book/Gatsby">Gatsby</a> |
      <a href="Book/Gatsby/ch/4?key=value">Gatsby: Ch4</a> |
      <a href="Book/Scarlet">Scarlet Letter</a><br/>

      <div ng-view></div>
      <hr />

      <pre>$location.path() = {{$location.path()}}</pre>
      <pre>$route.current.templateUrl = {{$route.current.templateUrl}}</pre>
      <pre>$route.current.params = {{$route.current.params}}</pre>
      <pre>$route.current.scope.name = {{$route.current.scope.name}}</pre>
      <pre>$routeParams = {{$routeParams}}</pre>
    </div>
  </body>
</html>
```

Book.html

```
controller: {{name}}<br />
Book Id: {{params.bookId}}<br />
```

Chapter.html

```
controller: {{name}}<br />
Book Id: {{params.bookId}}<br />
Chapter Id: {{params.chapterId}}
```

Script.js

```
angular.module('ngView', [], function($routeProvider, $locationProvider) {
  $routeProvider.when('/Book/:bookId', {
    templateUrl: 'book.html',
    controller: BookCntl,
    resolve: {
      // I will cause a 1 second delay
      delay: function($q, $timeout) {
        var delay = $q.defer();
        $timeout(delay.resolve, 1000);
        return delay.promise;
      }
    }
  });
  $routeProvider.when('/Book/:bookId/ch/:chapterId', {
    templateUrl: 'chapter.html',
    controller: ChapterCntl
  });

  // configure html5 to get links working on jsfiddle
  $locationProvider.html5Mode(true);
});

function MainCntl($scope, $route, $routeParams, $location) {
  $scope.$route = $route;
  $scope.$location = $location;
  $scope.$routeParams = $routeParams;
}

function BookCntl($scope, $routeParams) {
  $scope.name = "BookCntl";
  $scope.params = $routeParams;
}

function ChapterCntl($scope, $routeParams) {
  $scope.name = "ChapterCntl";
  $scope.params = $routeParams;
}
```

End to end test

```
it('should load and compile correct template', function() {
  element('a:contains("Moby: Ch1")').click();
  var content = element('.doc-example-live [ng-view]').text();
  expect(content).toMatch(/controller\: ChapterCntl/);
  expect(content).toMatch(/Book Id\: Moby/);
  expect(content).toMatch(/Chapter Id\: 1/);

  element('a:contains("Scarlet")').click();
  sleep(2); // promises are not part of scenario waiting
```

```
  content = element('.doc-example-live [ng-view]').text();
  expect(content).toMatch(/controller\: BookCntl/);
  expect(content).toMatch(/Book Id\: Scarlet/);
});
```

$routeParams

Описание

Текущие параметры для маршрутизации. Параметры маршрутизации, это комбинация методов сервиса `$location` `search()`, и `path()`. Параметр `path` извлечен, когда найдет путь в `$route`.

В случае конфликта имен параметров, параметры `path` приоритетнее параметров `search`.

Сервис гарантирует что объект `$routeParams` не будет изменяться (но его свойства будут меняться) даже когда маршрут будет изменен.

Зависимости

- $route

$templateCache

Описание

Кэш используемый для хранения html шаблонов.

Смотрите `$cacheFactory`.

$timeout

Описание

Angular обертка над `window.setTimeout`. Передаваемая внутрь функция `fn` обернута в блок try/catch и при любом исключении обработка проводится сервисом `$exceptionHandler`.

Возвращает значение для зарегистрированной функции с типом promise который будет разрешен установленное время истечет, и функция будут выполнена.

Для отмены запроса на установку таймаута, вызовите `$timeout.cancel(promise)`.

В тестах вы можете использовать `$timeout.flush()` для синхронизации очереди отложенных функций.

Зависимости

- $browser

Использование

```
$timeout(fn[, delay][, invokeApply]);
```

Parameters

- `fn – {function()}` – функция которая должна быть выполнена.

- `delay(optional=0)` – {number=} – Задержка в миллисекундах.
- `invokeApply(optional=true)` – {boolean=} – Если установить в false пропускается проверка модели на изменения, иначе функция `fn` будет вызвана в блоке `$apply`.

Возврат

`{Promise}` – Promise который будет разрешен по истечению времени таймаута. Значение promise будет разрешено возвращенным значением из функции `fn`.

Methods

cancel(promise)

Отменяет задачу, ассоциированную с `promise`. Как результат этот promise будет разрешен как отвергнутый.

Параметры:

- `promise(optional)` – {Promise=} – Promise возвращенный функцией `$timeout`.

Возврат:

`{boolean}` – Возвращает `true`, если задача еще не выполнена или была успешно отменена.

$window

Описание

Просто ссылка на объект браузера `window`. Хотя `window` и доступен глобально в JavaScript, при его использовании возникает проблема при тестировании, т.к. он является глобальной переменной. В angular мы всегда ссылаемся на него через сервис `$window`, т.к. в этом случае его можно переопределить, удалить или заменить другим при тестировании.

Любые выражения, вычисляемые в любой области видимости не должны работать с глобальным объектом window.

Пример

Index.html

```
1. <!doctype html>
2. <html ng-app>
3.   <head>
4.     <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
5.   </head>
6.   <body>
7.     <input ng-init="$window = $service('$window'); greeting='Hello World!'" type="
   text" ng-model="greeting" />
8.     <button ng-click="$window.alert(greeting)">ALERT</button>
9.   </body>
10. </html>
```

Типы

Module

Описание

Интерфейс для настройки `modules` Angular.

Методы

config(configFn)

Используйте этот метод для регистрации работы, которую нужно сделать во время загрузки модулю.

Параметры:

- `configFn – {Function} –` Эта функция выполняется во время загрузки модуля. Обычно используется для конфигурации сервиса.

constant(name, object)

Так как константы являются фиксированными значениями, они устанавливаются перед выполнением других методов. Смотрите `$provide.constant()`.

Параметры:

- `name – {string} –` имя константы
- `object – {*} –` значение константы.

controller(name, constructor)

Смотрите `$controllerProvider.register()`.

Параметры:

- `name – {string} –` имя контроллера.
- `constructor – {Function} –` Функция конструктор.

directive(name, directiveFactory)

Смотрите `$compileProvider.directive()`.

Параметры:

- `name – {string} –` имя директивы
- `directiveFactory – {Function} –` Фабричная функция для создания нового экземпляра директивы.

factory(name, providerFunction)

Смотрите `$provide.factory()`.

Параметры:

- `name – {string} –` имя сервиса
- `providerFunction – {Function} –` Функция для создания нового экземпляра сервиса.

filter(name, filterFactory)

Смотрите `$filterProvider.register()`.

Параметры:

- `name – {string} –` Имя фильтра.

- `filterFactory – {Function}` – Фабричная функция для создания нового экземпляра фильтра.

provider(name, providerType)

Смотрите `$provide.provider()`.

Параметры:

- `name – {string}` – имя сервиса
- `providerType – {Function}` – Функция конструктор для создания нового экземпляра сервиса.

run(initializationFn)

Используйте этот метод для регистрации работы, которую нужно сделать когда инжектор загрузит все модули.

Параметры:

- `initializationFn – {Function}` – Эта функция выполнится после создания инжектора. Используется для инициализации приложения.

service(name, constructor)

Смотрите `$provide.service()`.

Параметры:

- `name – {string}` – имя сервиса
- `constructor – {Function}` – Функция конструктор для создания экземпляров сервиса.

value(name, object)

Смотрите `$provide.value()`.

Параметры:

- `name – {string}` – Имя сервиса
- `object – {*}` – экземпляр объекта сервиса.

Свойства

name

Возврат:

`{string}` – Имя модуля.

requires

Список всех модулей, которые инжектор будет загружать перед загрузкой текущего модуля.

Возврат:

`{Array.<string>}` – Список имен модулей, которые должны быть загружены перед текущим модулем.

Attributes

Описание

Разделяемый объект между функциями компиляции / связывания директив, который содержит нормализованные атрибуты элемента DOM. Эти значения отображаются для текущего состояния привязки `{{ }}`. Нормализация необходима, т.к. Angular позволяет записывать один и тот же атрибут разными способами:

```
<span ng:bind="a" ng-bind="a" data-ng-bind="a" x-ng-bind="a">
```

Методы

$set(name, value)

Устанавливает значение атрибута для DOM элемента.

Параметры:

- `name – {string}` – Нормализованное имя атрибута элемента для модификации. Это имя изменяется в оригинальную форму с использование свойства `$attr` .
- `value – {string}` – Значение для установки.

Свойства

$attr

Возврат:

`{object}` – Карта которая содержит для элемента DOM имена атрибутов и соответствующие нормализованные имена. Это нужно для того, чтобы возвращать прежнее имя атрибуту после работы с ним.

Scope

Описание

Корневая область видимости может быть извлечена с использованием ключа `$rootScope` с сервисом `$injector`. Дочерние области видимости создаются с использованием метода `$new()`. (Другие области видимости создаются автоматически, когда компилируется HTML шаблон.)

Вот пример кода простой области видимости, который демонстрирует, как вы можете взаимодействовать с вашей областью видимости.

```
1.  angular.injector(['ng']).invoke(function($rootScope) {
2.     var scope = $rootScope.$new();
3.     scope.salutation = 'Hello';
4.     scope.name = 'World';
5.
6.     expect(scope.greeting).toEqual(undefined);
7.
8.     scope.$watch('name', function() {
9.       scope.greeting = scope.salutation + ' ' + scope.name + '!';
10.    }); // начало отслеживания значений
11.
12.    expect(scope.greeting).toEqual(undefined);
```

```
13.    scope.name = 'Misko';
14.    // по прежнему старое значение, т.к. watch еще не вызван
15.    expect(scope.greeting).toEqual(undefined);
16.
17.    scope.$digest(); // известить (вызвать) все watches
18.    expect(scope.greeting).toEqual('Hello Misko!');
19. });
```

Наследование

Область видимости может наследоваться от другой области видимости, как показано в этом примере:

```
1.  var parent = $rootScope;
2.  var child = parent.$new();
3.
4.  parent.salutation = "Hello";
5.  child.name = "World";
6.  expect(child.salutation).toEqual('Hello');
7.
8.  child.salutation = "Welcome";
9.  expect(child.salutation).toEqual('Welcome');
10. expect(parent.salutation).toEqual('Hello');
```

Использование

```
1.  Scope([providers][, instanceCache]);
```

Параметры

- `providers(optional)` – `{Object.<string, function()>=}` – Карта фабрик для сервисов, которые должны быть предоставлены текущей области видимости. По умолчанию только ng.
- `instanceCache(optional)` – `{Object.<string, *>=}` – Предоставляет предварительно созданные сервисы, которые добавляются / заменяют сервисы, указанные в параметре `providers`. Это облегчает юнит-тестирование, когда нужно переопределить сервисы по умолчанию.

Возврат:

`{Object}` – Вновь созданная область видимости.

Методы

$apply(exp)

`$apply()` используется для выполнения выражений angular из кода вне angular Фреймворка. (К примеру, из обработчика события DOM в браузере, setTimeout, XHR или других библиотек). Так когда код выполняется внутри angular фреймворка, нам нужно выполнить определенные этапы жизненного цикла области видимости, это обработка исключений, отслеживание изменений.

Жизненный цикл

Псевдо код `$apply()`

```
function $apply(expr) {
  try {
    return $eval(expr);
  } catch (e) {
    $exceptionHandler(e);
  } finally {
    $root.$digest();
  }
}
```

Метод области видимости `$apply()` проходит через следующие стадии:

1. Выражение выполняется используя метод `$eval()`.
2. Любые исключения возникшие при выполнении выражения передаются на обработку сервису `$exceptionHandler`.
3. Слушатели событий изменения `watch` будут немедлено оповещены после выполнения выражения через вызов метода `$digest()`.

Параметры:

- `exp(optional) – {(string|function())=} –` Выражение angular для выполнения.
  - `string`: выполняются используя синтаксис определенный для выражения.
  - `function(scope)`: выполняется функция с текущей областью видимости переданной в качестве параметра.

Возврат:

`{*}` – Результат выполнения выражения.

$broadcast(name, args)

Отправляет событие с именем `name` вниз, для всех дочерних областей видимости (и их дочерних тоже), уведомляя слушателей, зарегистрированных с помощью `ng.$rootScope.Scope#$on`.

Жизненный цикл события начинается в области видимости, для которой был вызван метод `$broadcast`. Все слушатели `listeners` события с именем `name` в текущей области видимости будут извещены. После этого событие распространяется на всех прямые и косвенные дочерние области видимости, по пути уведомляя о наступления события всех слушателей в них. Событие не может быть отменено.

Любые исключения, возникшие в обработчиках события для всех слушателей `listeners` будут переданы на обработку сервису `$exceptionHandler`.

Параметры:

- `name – {string} –` Имя события для отправки.
- `args – {...*} –` Необязательные для установки аргументы, которые будут переданы обработчикам события.

Возврат:

`{Object}` – Объект представляющий событие, смотрите `ng.$rootScope.Scope#$on`

$destroy()

Удаляет текущую область видимости (и все дочерние области видимости) из родительской области видимости. Удаление подразумевает, что вызовы метода `$digest()` больше не должны распространяться на текущую область видимости, и на ее дочерние области видимости. Удаление также подразумевает, что удаленные области видимости доступны для уничтожения сборщику мусора.

Метод `$destroy()` обычно используется в директивах, таких как `ngRepeat` для управления развертыванием элементов в цикле.

Перед удалением области видимости будет послано событие `$destroy` удаляемой области видимости и всем ее дочерним областям видимости. Код приложения может регистрировать обработчики для события `$destroy`, чтобы перед удалением можно было выполнить требуемый код очистки.

$digest()

Обрабатывает всех наблюдателей `watchers` для текущей области видимости и для ее дочерних областей видимости. Потому что слушатели `watcher` могут изменить модель, (что вновь вызовет `$digest()`) `$digest()` задерживает вычисление `watchers` до того момента, пока не обработают другие слушатели. Это означает, что можно попасть в бесконечный цикл. Эта функция будет выкидывать исключение `'Исчерпан лимит итераций'`, если номер итерации превышает 10.

Обычно вам не нужно вызывать `$digest()` в `контроллерах` или в `директивах`. Вместо этого вызывайте `$apply()`(обычно из `директив`), которая сама вызовет `$digest()`.

Если вы хотите получать уведомления каждый раз, когда вызывается `$digest()`, вы можете зарегистрировать функцию `watchExpression` с `$watch()` без прослушивания событий.

Вам может понадобится вызов `$digest()` внутри юнит-тестов, чтобы симулировать жизненный цикл области видимости.

Пример:

```
1.  var scope = ...;
2.  scope.name = 'misko';
3.  scope.counter = 0;
4.
5.  expect(scope.counter).toEqual(0);
6.  scope.$watch('name', function(newValue, oldValue) {
7.    scope.counter = scope.counter + 1;
8.  });
9.  expect(scope.counter).toEqual(0);
10.
11. scope.$digest();
12. // no variable change
13. expect(scope.counter).toEqual(0);
14.
15. scope.name = 'adam';
16. scope.$digest();
17. expect(scope.counter).toEqual(1);
```

$emit(name, args)

Отправляет событие с именем `name` вверх по всей иерархии областей видимости, уведомляя всех зарегистрированных с помощью `ng.$rootScope.Scope#$on` слушателей.

Жизненный цикл события начинается с области видимости, в которой вызван `$emit`. Все слушатели `listeners` для события с именем `name` в этой области видимости будут извещены. После чего событие всплывает вплоть до корневой области видимости, по пути уведомляя всех зарегистрированных слушателей. Распространение события можно отменить, если один из слушателей сделает это.

Любое исключение, возникшее в слушателях `listeners` будет передано сервису `$exceptionHandler` для обработки.

Параметры:

- `name – {string} –` Имя события.
- `args – {...*} –` Необязательно установленный аргумент, который будет передан внутрь обработчикам события.

Возврат:

`{Object}` – Объект, представляющий событие, смотрите `ng.$rootScope.Scope#$on`

$eval(expression)

Выполняет выражение `expression` в текущей области видимости и возвращает его результат. Любые исключения, возникшие в выражении, распространяются (необработанные). Обычно используется когда нужно выполнить выражение Angular.

Пример:

```
1. var scope = ng.$rootScope.Scope();
2. scope.a = 1;
3. scope.b = 2;
4.
5. expect(scope.$eval('a+b')).toEqual(3);
6. expect(scope.$eval(function(scope){ return scope.a + scope.b; })).toEqual(3);
```

Параметры:

- `expression(optional) – {(string|function())=} –` Выражение angular, которое должно быть выполнено.
  - `string`: выполняется используя синтаксис, определенный для выражения.
  - `function(scope)`: выполняется функция, параметром которой является текущая область видимости.

Возврат:

`{*}` – Результат вычисления выражения.

$evalAsync(expression)

Выполняет выражение в текущей области видимости в асинхронной манере.

`$evalAsync` гарантирует, что выражение будет выполнено, с:

- будет выполнено в текущем контексте выполнения скрипта (перед формированием любого DOM).
- Вызовется итерация `цикла $digest` после выполнения выражения `expression`.

Любое исключение, возникшее при выполнении выражения, будет передано на обработку сервису `$exceptionHandler`.

Параметры:

- `expression(optional)` – `{(string|function())=}` – Выражение angular которое будет выполнено.
  - `string`: выполняется с использование синтаксиса для выражения.
  - `function(scope)`: выполняется функция с параметром, установленным в текущую область видимости.

$new(isolate)

Создает новую дочернюю `область видимости`.

Родительская область видимости будет распространять `$digest()` и события `$digest()`. Область видимости может быть удалена из иерархии используя метод `$destroy()`.

`$destroy()` должен быть вызван для области видимости, когда требуется для нее и ее дочерних областей видимости, отделиться от родительской области, и перестать следить за изменениями, а также прослушивать события.

Параметры:

- `isolate` – `{boolean}` – Если установить в true, тогда область видимости не наследуется прототипно от родительской области. Область видимости изолируется и не может просматривать свойства родительской области видимости. Когда создаются виджеты, это используется для того, чтобы виджет не мог работать с состоянием родительской области видимости.

Возврат:

`{Object}` – Созданная дочерняя область видимости.

$on(name, listener)

Подписка слушателя для события определенного типа. Смотрите `$emit` для понимания жизненного цикла событий.

Функция слушатель события имеет формат: `function(event, args...)`. Объект события `event` передается слушателю и содержит следующие атрибуты:

- `targetScope` - `{Scope}`: область видимости, в которой был вызван один из методов: `$emit` или `$broadcast`.
- `currentScope` - `{Scope}`: текущая область видимости, в которой обрабатывается событие.
- `name` - `{string}`: имя события.
- `stopPropagation` - `{function=}`: вызов функции `stopPropagation` будет отменять дальнейшее распространение события (доступно только для событий вызванных с помощью `$emit`).
- `preventDefault` - `{function}`: вызов `preventDefault` устанавливает флаг `defaultPrevented` в true.
- `defaultPrevented` - `{boolean}`: true если `preventDefault` был вызван.

Параметры:

- `name` – `{string}` – Имя события для подписки.
- `listener` – `{function(event, args...)}` – Функция обработчик события.

Возврат:

`{function()}` – Возвращает функцию, выполнение которой отменяет прослушивание события для текущего слушателя.

$watch(watchExpression, listener, objectEquality)

Регистрирует функцию обратного вызова `listener`, которая будет выполнена после изменений `watchExpression`.

- Выражение `watchExpression` вычисляется при каждой итерации цикла `$digest()` и должно возвращать значение, которое отслеживается.
  (Поскольку `$digest()` выполняется когда обнаруживаются изменения, выражение `watchExpression` может выполнятся несколько раз и должно быть идемпотентным)
- Функция `listener` вызывается только когда значение текущее значение выражения `watchExpression` и предыдущее значение этого выражения не эквивалентны (исключение является стартовое выполнение, смотрите дальше). Эквивалентность определяется с помощью функции `angular.equals`. Для сохранения значения объектов, которые в дальнейшем будут сравниваться, используется функция `angular.copy`. Это также означает, что отслеживание множества свойств, будет иметь неблагоприятные последствия для памяти и производительности.
- Обработка изменения подписчиком `listener` может изменять модель, в результате вновь будет вызван обработчик `listener`. Процесс перезапуска будет продолжаться до тех пор, пока не будет обнаружено новых изменений. Количество итераций перезапуска не может быть больше 10, для предотвращения бесконечных циклов.

Если вам нужно получать уведомления при каждом запуске `$digest`, вы может зарегистрировать выражение `watchExpression` без подписчиков `listener`. (Т.к. `watchExpression` может выполняться несколько раз в течении цикла `$digest`, будьте готовы к тому, что ваш слушатель будет вызван это же количество раз.)

После регистрации слушателя в области видимости, функция `listener` будет вызываться асинхронно (с `$evalAsync`) при инициализации наблюдения. В редких случаях это не желательно, т.к. слушатель вызывается когда результат `watchExpression` еще не изменился. Для обнаружения этого в сценарии функции `listener`, вы может сравнивать `newVal` и `oldVal`. Если эти значения идентичны (===), значит слушать вызван процессом инициализации.

Пример:

```
1. // Давайте предположим, что зависимость области видимости $rootScope
2. var scope = $rootScope;
3. scope.name = 'misko';
4. scope.counter = 0;
5.
6. expect(scope.counter).toEqual(0);
7. scope.$watch('name', function(newValue, oldValue) { scope.counter = scope.coun
   ter + 1; });
8. expect(scope.counter).toEqual(0);
9.
10. scope.$digest();
11. // переменные не изменились
12. expect(scope.counter).toEqual(0);
13.
14. scope.name = 'adam';
```

```
15. scope.$digest();
16. expect(scope.counter).toEqual(1);
```

Параметры:

- `watchExpression – {(function()|string)} –` Выражение, которое будет вычисляться на каждой итерации цикла `$digest`. Если возвращаемое значение изменилось, будет вызван слушатель `listener`.
  - `string`: вычисляется как выражение
  - `function(scope)`: вычисляется с параметром, установленным в текущую область видимости.
- `listener(optional) – {(function()|string)=} –` Функция обратного вызова, которая вызывается при изменении возвращаемого значения выражения `watchExpression`.
  - `string`: вычисляется как выражение
  - `function(newValue, oldValue, scope)`: вычисляется с параметрами новое значение, старое значение и текущая область видимости.
- `objectEquality(optional) – {boolean=} –` Сравнивать эквивалентность объектов, а не ссылки.

Возврат:

`{function()}` – Возвращает функцию для завершения прослушивания данного события данных слушателем.

Свойства

$id

Возврат:

`{number}` – уникальный идентификатор области видимости (монотонно возрастающая цифровая последовательность) обычно используемый для отладки.

События

$destroy

Распространяется вниз, перед тем, как область видимости и все ее дочерние области видимости будут разрушены.

Тип: нисходящее

Цель: области, которые будут разрушаться

FormController

Описание

`FormController` содержит все элементы управления и вложенные формы, а также состояние для них, такие как valid/invalid или dirty/pristine.

Каждая директива `form` создает экземпляр `FormController`.

Свойства

$pristine

True, если пользователь еще не взаимодействовал с формой.

$dirty

True, если пользователь уже взаимодействовал с формой.

$valid

True, если все вложенные формы и элементы управления содержат «правильные» данные (проходят процесс проверки).

$invalid

True, если хотя бы одна вложенная форма или элемент управления содержат «недопустимые» данные (не проходят проверку).

$error

Это хэш объект, который содержит ссылки на элементы управления или вложенные формы в которых содержаться «недопустимые» данные, где:

- ключи, это проверочные метки (имена ошибок) — такие как `required`, `url` или `email`),
- значения, это массив элементов управления или форм, которые не соответствуют текущему правилу проверки данных.

NgModelController

Описание

`NgModelController` предоставляет API для директивы `ng-model`. Контроллер содержит сервисы для привязки данных, их проверки, обновления CSS, форматирования значений и разбора ввода. Он не должен содержать логику отображения DOM или слушания событий DOM. `NgModelController` предназначен для расширения другими директивами, где директива манипулирует с DOM, а `NgModelController` осуществляет привязку данных.

Этот пример показывает как использовать `NgModelController` с пользовательским элементом управления для добавления привязки данных. Обратите внимание, как различные директивы (`contenteditable`, `ng-model`, и `required`) работают вместе для достижения желаемого результата.

Index.html

```
1.  <!doctype html>
2.  <html ng-app="customControl">
3.    <head>
4.      <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
5.      <script src="script.js"></script>
6.    </head>
7.    <body>
8.      <form name="myForm">
9.       <div contenteditable
10.           name="myWidget" ng-model="userContent"
11.           required>Change me!</div>
12.        <span ng-show="myForm.myWidget.$error.required">Required!</span>
13.       <hr>
```

```
    <textarea ng-model="userContent"></textarea>
   </form>
  </body>
</html>
```

Style.css

```
[contenteditable] {
  border: 1px solid black;
  background-color: white;
  min-height: 20px;
}

.ng-invalid {
  border: 1px solid red;
}
```

Script.js

```
angular.module('customControl', []).
  directive('contenteditable', function() {
    return {
      restrict: 'A', // only activate on element attribute
      require: '?ngModel', // get a hold of NgModelController
      link: function(scope, element, attrs, ngModel) {
        if(!ngModel) return; // do nothing if no ng-model

        // Specify how UI should be updated
        ngModel.$render = function() {
          element.html(ngModel.$viewValue || '');
        };

        // Listen for change events to enable binding
        element.bind('blur keyup change', function() {
          scope.$apply(read);
        });
        read(); // initialize

        // Write data to the model
        function read() {
          ngModel.$setViewValue(element.html());
        }
      }
    };
  });
```

End to end test

```
it('should data-bind and become invalid', function() {
```

```
  var contentEditable = element('[contenteditable]');

  expect(contentEditable.text()).toEqual('Change me!');
  input('userContent').enter('');
  expect(contentEditable.text()).toEqual('');
  expect(contentEditable.prop('className')).toMatch(/ng-invalid-required/);
});
```

Методы

$render()

Вызывается когда нужно обновить представление. Предполагается что пользователь будет вызывать этот метод для директивы ng-model.

$setValidity(validationErrorKey, isValid)

Изменяет статус проверки данных и уведомляет форму, когда элемент управления изменяет свой статус проверки данных. (форма не уведомляется, если для элемента управления уже ранее был установлен статус - invalid).

Этот метод должен вызываться валидаторами – ни или функциями разбора или форматирования.

Параметры:

- `validationErrorKey – {string} –` имя валидатора. `validationErrorKey` будет ассоциирован с `$error[validationErrorKey]=isValid`, и это можно использовать для привязки данных. `validationErrorKey` должен быть в верблюжьей нотации и будет преобразоваться в имя с разделителем тире, для имен классов CSS.
  Например: `myError` добавит классы `ng-valid-my-error` и `ng-invalid-my-error` и к нему можно привязываться, как `{{someForm.someControl.$error.myError}}` .
- `isValid – {boolean} –` true, если текущее состояние valid, или false – если invalid.

$setViewValue(value)

Читает значение из представления.

Этот метод должен вызываться из обработчика событий DOM. К примеру директивы `input` или `select` вызывают его.

Внутри он вызывает всех форматеров `formatters` и если результат является допустимым, обновляет модель и уведомляем всех зарегистрированных слушателей об изменении.

Параметры:

- `value – {string} –` Значение из представления.

Свойства

$viewValue

Актуальное значение в представлении в виде строки.

$modelValue

Значение модели, к которой привязан элемент управления.

$parsers

Когда элемент управления читает значение из DOM, вызываются все функции для очистки / конвертации введенного значения и проверки его допустимости.

$formatters

Когда модель меняет свое значение, выполняются все эти функции для конвертации значения в удобный для пользователя формат.

$error

Хэш объект, содержащий ключи ошибок и массив элементов с этими ошибками.

$pristine

True, если пользователь не взаимодействовал с элементом управления.

$dirty

True, если пользователь уже взаимодействовал с элементом управления.

$valid

True, если нет ошибок.

$invalid

True, если есть хоть одна ошибка в элементе управления.

Глобальное API

angular.bind

Описание

Возвращает функцию, которая вызывает функцию `fn` в контексте `self` (`self` доступна через `this` для `fn`). Вы можете передать аргументы `args` в функцию. Эта возможность также известна как function currying.

Использование

```
1. angular.bind(self, fn, args);
```

Параметры

- `self – {Object}` – Контекст, в котором функция `fn` должна быть вызвана.
- `fn – {function()}` – Функция для связывания.
- `args – {...*}` – Необязательные аргументы, которые будут переданы в функцию `fn`.

Возврат

`{function()}` – Функция, которая оборачивает функцию `fn` со всеми указанными зависимостями.

angular.bootstrap

Описание

Используйте эту функцию для ручного старта приложения angular.

Смотрите: Bootstrap

Использование

```
1. angular.bootstrap(element[, modules]);
```

Параметры

- `element – {Element} –` DOM элемент, который будет корневым для приложения angular.
- `modules(optional) – {Array<String|Function>=} –` массив имен модулей или декларации модулей. Смотрите: `modules`

Возврат

`{AUTO.$injector}` – Возвращает новый инжектор для приложения.

angular.copy

Описание

Создает глубокую копию аргумента `source`, который может быть объектом или массивом.

- Если не предоставлен объект назначения destination, будет создана копия объекта или массива.
- Если объект назначения предоставлен, все его элементы (для массива) или свойства (для объекта) будут удалены, а затем в него будут скопированы все элементы / свойства из источника source.
- Если `source` не является массивом или объектам, он просто будет возвращен.

Примечание: эта функция используется во многих типах объектов в выражениях Angular. Смотрите `ng.$filter` для большей информации о массивах Angular.

Использование

```
1. angular.copy(source[, destination]);
```

Параметры

- `source – {*} –` Источник, который будет использован для создания копии. Может быть любого типа, включая примитивные `null` и `undefined`.
- `destination(optional) – {(Object|Array)=} –` Приемник, в который будет копироваться source. Если указан, должен иметь тот же тип, что и `source`.

Возврат

`{*}` – Копия или обновленный `destination`, если `destination` указан.

angular.element

Описание

Оборачивает родной DOM элемент или HTML строку в элемент jQuery. `angular.element` может быть псевдонимом функции jQuery, если jQuery доступно, или функцией, которая оборачивает элемент или строку в Angular облегченную реализацию jQuery (обычно именуемый как jqLite).

Реальный jQuery всегда имеет приоритет перед jqLite, при условии что он был загружен до возникновения события `DOMContentLoaded`.

jqLite это маленький, API-совместимый с jQuery объект, который позволяет Angular манипулировать DOM. jqLite поддерживает только наиболее часто требуемые функции из-за своего малого размера, так что поддерживается только часть jQuery API - методы, аргументы и работа со стилями.

Примечания: Все элементы, на которые ссылаются в Angular всегда оборачиваются в jQuery или jqLite; они ни когда не ссылаются на родные DOM элементы.

Angular jQuery lite предоставляет следующие методы:

- addClass()
- after()
- append()
- attr()
- bind() - Does not support namespaces
- children() - Does not support selectors
- clone()
- contents()
- css()
- data()
- eq()
- find() - Limited to lookups by tag name
- hasClass()
- html()
- next() - Does not support selectors
- parent() - Does not support selectors
- prepend()
- prop()
- ready()
- remove()
- removeAttr()
- removeClass()
- removeData()
- replaceWith()
- text()
- toggleClass()
- triggerHandler() - Doesn't pass native event objects to handlers.
- unbind() - Does not support namespaces
- val()
- wrap()

В дополнение к вышеуказанным, Angular предоставляет дополнительные методы для обоих jQuery и jQuery lite:

- `controller(name)` – извлекает контроллер для текущего элемента или его родителя. По умолчанию извлекается контроллер, ассоциированный с директивой `ngController`. Если `name` предоставлено в верблюжьей нотации как имя директивы, тогда будет извлечен контроллер для этой директивы (например, `'ngModel'`).
- `injector()` – извлекает инжектор для текущего элемента или его родителя.
- `scope()` – извлекает область видимости для текущего элемента или его родителя.

- `inheritedData()` – то же, что и `data()`, но идет вверх по DOM пока не будет найдено требуемое значение или не будет достигнут корневой элемент.

Использование

```
1. angular.element(element);
```

Параметры

- `element – {string|DOMElement} –` HTML строка или DOM элемент, который должен быть обернут в jQuery.

Возврат

`{Object}` – jQuery объект.

angular.equals

Описание

Определяет, эквивалентны ли два объекта или два значения. Поддерживаются значимые типы, объекты и массивы.

Два объекта или значения считаются эквивалентными, если является истинным каждое из следующих утверждений:

- Оба объекта или значения имеют одинаковый тип и значение (===).
- Оба объекта или значения имеют одинаковые свойства с одинаковым типом и с одинаковыми значениями (=== для свойств).
- Оба значение NaN. (В JavasScript, NaN == NaN => false. Но мы сделали что два NaN эквивалентны)

Во время сравнения свойств, свойства с типом `function` и свойства, имена которых начинаются на `$` - игнорируются.

Области видимости и объект DOM Window будут сравниваться только по идентификаторам (===).

Использование

```
1. angular.equals(o1, o2);
```

Параметры

- `o1 – {*} –` Объект или значение для сравнения.
- `o2 – {*} –` Объект или значение для сравнения.

Возврат

`{boolean}` – True, если аргументы эквивалентны.

angular.extend

Описание

Расширяет целевой объект `dst` путем копирования всех свойств из объекта или объектов `src` в объект `dst`. Вы можете указать несколько объектов `src`.

Использование

```
angular.extend(dst, src);
```

Параметры

- `dst – {Object} –` Объект приемник.
- `src – {...Object} –` Объект или объекты источники.

angular.forEach

Описание

Вызывает функцию `iterator` для каждого элемента коллекции `obj`, которая может быть объектом или массивом. Функция `iterator` вызывается как `iterator(value, key)`, где `value` это значение текущего свойства объекта или элемент массива, и `key` это имя свойства в объекте или индекс элемента в массиве. Контекст `context` для функции передавать не обязательно.

Примечание: Эта функция раньше называлась `angular.foreach`.

```
var values = {name: 'misko', gender: 'male'};
var log = [];
angular.forEach(values, function(value, key){
  this.push(key + ': ' + value);
}, log);
expect(log).toEqual(['name: misko', 'gender:male']);
```

Использование

```
angular.forEach(obj, iterator[, context]);
```

Параметры

- `obj – {Object|Array} –` Объект для итерации.
- `iterator – {Function} –` функция итерации.
- `context(optional) – {Object=} –` Объект, в контексте которого будет выполнена функция итератор (будет доступен через `this`).

Возврат

`{Object|Array}` – Ссылка на `obj`.

angular.fromJson

Описание

Десериализует строку JSON.

Использование

```
1. angular.fromJson(json);
```

Параметры

- `json – {string} –` JSON строка для десериализации.

Возврат

`{Object|Array|Date|string|number}` – Десериализованное значение.

angular.identity

Описание

Эта функция возвращает первый аргумент. Эта функция обычно используется при написании кода в функциональном стиле.

```
1. function transformer(transformationFn, value) {
2.   return (transformationFn || identity)(value);
3. };
```

Использование

```
1. angular.identity();
```

angular.injector

Описание

Создает функцию инжектор, которая будет использоваться для извлечения служб при инъекциях зависимостей (смотрите инъекция зависимостей).

Использование

```
1. angular.injector(modules);
```

Параметры

- `modules – {Array.<string|Function>} –` Список функций модулей или их псевдонимов. Смотрите `angular.module`. Модуль ng будет добавлен по умолчанию.

Возврат

`{function()}` – функция инжектор. Смотрите `$injector`.

Пример

Типичное использование:

```
1. // создание инжектора
2. var $injector = angular.injector(['ng']);
```

```
3.
4. // используйте инжектор для старта вашего приложения
5. // для автоматической инъекции зависимостей или явно
6. $injector.invoke(function($rootScope, $compile, $document){
7.   $compile($document)($rootScope);
8.   $rootScope.$digest();
9. });
```

angular.isArray

Описание

Определяет, является ли переданный объект массивом.

Использование

```
1. angular.isArray(value);
```

Параметры

- `value – {*} –` ссылка для проверки.

Возврат

`{boolean}` – True, если `value` является массивом.

angular.isDate

Описание

Определяет, является ли value датой.

Использование

```
1. angular.isDate(value);
```

Параметры

- `value – {*} –` ссылка для проверки.

Возврат

`{boolean}` – True, если `value` является датой.

angular.isDefined

Описание

Определяет, является ли значение ссылки определенным.

Использование

```
1. angular.isDefined(value);
```

Параметры

- `value` – `{*}` – Ссылка для проверки.

Возврат

`{boolean}` – True, если `value` определено.

angular.isElement

Описание

Определяет, указывает ли ссылка на DOM элемент (или на обернутый jQuery элемент).

Использование

```
1. angular.isElement(value);
```

Параметры

- `value` – `{*}` – Ссылка для проверки.

Возврат

`{boolean}` – True, если `value` является элементом DOM (или оберткой jQuery для элемента).

angular.isFunction

Описание

Определяет, является ли ссылка функцией.

Использование

```
1. angular.isFunction(value);
```

Параметры

- `value` – `{*}` – Ссылка для проверки.

Возврат

`{boolean}` – True, если `value` является функцией.

angular.isNumber

Описание

Определяет, является ли аргумент числом.

Использование

```
1. angular.isNumber(value);
```

Параметры

- `value` – `{*}` – проверяемый аргумент.

Возврат

`{boolean}` – True, если `value` является числом.

angular.isObject

Описание

Определяет, указывает ли ссылка на объект. В отличии от `typeof` в JavaScript, `null` не считается объектом.

Использование

```
1. angular.isObject(value);
```

Параметры

- `value – {*} –` ссылка для проверки.

Возврат

`{boolean}` – True, если `value` является объектом, но не `null`.

angular.isString

Описание

Определяет, является ли ссылка строкой.

Использование

```
1. angular.isString(value);
```

Параметры

- `value – {*} –` ссылка для проверки.

Возврат

`{boolean}` – True, если `value` является строкой.

angular.isUndefined

Описание

Определяет, является ли аргумент не определенным.

Использование

```
1. angular.isUndefined(value);
```

Параметры

- `value – {*} –` ссылка для проверки.

Возврат

`{boolean}` – True, если `value` не определено.

angular.lowercase

Описание

Конвертирует строку в нижний регистр.

Использование

```
angular.lowercase(string);
```

Параметры

- `string – {string} –` Строка для конвертации в нижний регистр.

Возврат

`{string}` – строка в нижнем регистре.

angular.mock

Пространство имен для 'angular-mocks.js' который содержит инструменты для тестирования связанного кода.

angular.module

Описание

`angular.module` является глобальным местом создания и регистрации Angular модулей. Все модули (движка angular и другие) которые должны быть доступны для приложения должны быть зарегистрированы используя этот механизм.

Модуль

Модуль размещает в себе сервисы, директивы, фильтры и конфигурационную информацию. Модули используется для настройки `$injector`.

```
// создание нового модуля
var myModule = angular.module('myModule', []);

// регистрация нового сервиса
myModule.value('appName', 'MyCoolApp');

// конфигурация существующих сервисов внутри блока конфигурации.
myModule.config(function($locationProvider) {
  // Конфигурация сервиса
  $locationProvider.hashPrefix('!');
});
```

Также можно создать инжектор и загрузить ваши модули, как показано здесь::

```
var injector = angular.injector(['ng', 'MyModule'])
```

Однако, по большей части вы будете использовать `ngApp` или `angular.bootstrap` для упрощения этого процесса.

Использование

```
angular.module(name[, requires], configFn);
```

Параметры

- `name – {!string} –` Имя модуля для создания или извлечения.
- `requires(optional) – {Array.<string>=} –` Если указан, тогда новый модуль будет создан. Если не указан, тогда модуль будет извлечен для дальнейшей настройки.
- `configFn – {Function} –` Необязательная конфигурационная функция для модуля. Также доступна через `Module#config()`.

Возврат

`{module}` – новый модуль реализующий `angular.Module` api.

angular.noop

Описание

Эта функция выполняет операцию отрицания (не). Эта функция может быть использована при написании кода в функциональном стиле.

```
function foo(callback) {
  var result = calculateResult();
  (callback || angular.noop)(result);
}
```

Использование

```
angular.noop();
```

angular.toJson

Описание

Сериализация входного аргумента в строковой формат JSON.

Использование

```
angular.toJson(obj[, pretty]);
```

Параметры

- `obj – {Object|Array|Date|string|number} –` Вход для сериализации в JSON.
- `pretty(optional) – {boolean=} –` Если установлено в true, тогда выходной JSON будет содержать символы новой строки и пробелы.

Возврат

`{string}` – Строка JSON, представляющая `obj`.

angular.uppercase

Описание

Конвертирует указанную строку в верхний регистр.

Использование

```
angular.uppercase(string);
```

Параметры

- `string – {string} –` Строка для конвертации в верхний регистр.

Возврат

`{string}` – входная строка в верхнем регистре.

angular.version

Описание

Объект, содержащий информацию о текущей версии AngularJS. Этот объект имеет следующие свойства:

- `full` – `{string}` – Строка полной версии, такая как "0.9.18".
- `major` – `{number}` – Номер старшей версии, такой как "0".
- `minor` – `{number}` – Номер младшей версии, такой как "9".
- `dot` – `{number}` – Номер исправления, такой как "18".
- `codeName` – `{string}` – Кодовое имя релиза, такое как "jiggling-armfat".

Использование

```
angular.version
```

Модуль ng-mock

Сервисы

$exceptionHandler

Описание

Поддельная реализация сервиса `ng.$exceptionHandler`, которая регенерирует или логирует ошибки, передаваемые ему. Смотрите $exceptionHandlerProvider для информации о настройке.

```
describe('$exceptionHandlerProvider', function() {

  it('should capture log messages and exceptions', function() {

```

```
    module(function($exceptionHandlerProvider) {
      $exceptionHandlerProvider.mode('log');
    });

    inject(function($log, $exceptionHandler, $timeout) {
      $timeout(function() { $log.log(1); });
      $timeout(function() { $log.log(2); throw 'banana peel'; });
      $timeout(function() { $log.log(3); });
      expect($exceptionHandler.errors).toEqual([]);
      expect($log.assertEmpty());
      $timeout.flush();
      expect($exceptionHandler.errors).toEqual(['banana peel']);
      expect($log.log.logs).toEqual([[1], [2], [3]]);
    });
  });
});
```

$httpBackend

Описание

Поддельная серверная реализация HTTP, подходящая для модульного тестирования приложения, использующего `сервис $http`.

Замечание: Эта поддельная реализация серверного HTTP подходит для end-to-end тестирования, а для серверных разработчиков можете посмотреть менее функциональную версию e2e $httpBackend mock.

Во время модульного тестирования мы хотим, чтобы наши тесты выполнялись быстро не имели внешних зависимостей, так что мы не хотим отравлять реальные запросы XHR или JSONP на реальный сервер. Все что нам реально нужно, это проверить, был ли определенный запрос отправлен, или нет, и в качестве альтернативы, мы даем приложению возможность делать запросы, на которые будет реагировать поддельный объект заранее определенными ответами, а мы сможем проверить утверждения в тестах.

Этот поддельный объект может отвечать статическими или динамическими ответами, через `expect` и `when` apis и других, удобных версий (`expectGET`, `whenPOST`, и т.д.).

Когда приложение Angular нуждается в каких-либо данных с сервера, вызывается сервис $http, который отправляет запрос на реальный сервер используя сервис $httpBackend. С внедрением зависимостей легко внедрить поддельный $httpBackend (который имеет тот же API что и $httpBackend) и использовать его для проверки запросов и ответов при тестировании данных, без отправки запроса на реальный сервер.

Существует два способа указать мок(поддельному) объекту, какие данные должны быть возвращены как http ответы, когда тестируемый код делает запросы:

- `$httpBackend.expect` – указывает ожидания для запроса
- `$httpBackend.when` – задает определение бэкенд

Запрос ожиданий и бэкенд определений

Запрос ожиданий предоставляет способ сделать утверждения о созданных запросах, и определить ответы на эти запросы. Тест будет ошибочным, если ожидаемые запросы не сделаны, или сделаны в неправильном порядке.

Бэкенд определения позволяют вам определить поддельный бэкенд для вашего приложения, который не проверяет, сделан ли конкретный запрос или нет, а просто возвращает подготовленный ответ, если запрос был сделан. Во время прохождения теста, тест будет считать что данные получены и работать с ними.

	Запрос ожиданий	**Бэкенд определения**
Синтаксис	.expect(...).respond(...)	.when(...).respond(...)
Типичное использование	strict unit tests	loose (black-box) unit testing
Выполняет несколько запросов	нет	да
Проверка порядка запросов	да	нет
Обязательный запрос	да	нет
Обязательный ответ	Не обязательно (см. ниже)	да

В случаях когда оба, бэкенд определения и запроса ожидания указаны при выполнении теста, запрос ожиданий выполняется в первую очередь.

Если для запроса ожиданий не имеет заданного ответа, то алгоритм будет искать бэкенд определение для этого запроса чтобы получить ответ.

Если запрос не находит ожидания ли если ожидание не имеет определенного ответа, бэкенд определения будут оцениваются все по очереди, чтобы увидеть, содержат ли они соответствующий запрос. Ответ из первого совпадающего определения будет возвращен.

Сброс HTTP запросов

Используемый в рабочей версии $httpBackend всегда отвечает на запросы асинхронно. Если мы оставим это поведение в тестировочном варианте, мы будем создавать тесты в асинхронной манере, которые трудно создавать, отслеживать и поддерживать. В то же время тестировочная подделка, не должна отвечать синхронно, так как это приведет к изменению тестируемого кода. По этой причине у поддельного объекта $httpBackend имеется метод `flush()`, который позволяет тесту явно сбрасывать результаты отложенных запросов, таким образом сохраняя асинхронное api бэкенда, позволяет тесту выполняться синхронно.

Модульное тестирование с поддельным $httpBackend

```
1. // контроллер
2. function MyController($scope, $http) {
3.   $http.get('/auth.py').success(function(data) {
4.     $scope.user = data;
5.   });
6.
7.   this.saveMessage = function(message) {
8.     $scope.status = 'Saving...';
```

```
    $http.post('/add-msg.py', message).success(function(response) {
      $scope.status = '';
    }).error(function() {
      $scope.status = 'ERROR!';
    });
  };
}

// тестирование контроллера
var $httpBackend;

beforeEach(inject(function($injector) {
  $httpBackend = $injector.get('$httpBackend');

  // бэкенд определение для всех тестов
  $httpBackend.when('GET', '/auth.py').respond({userId: 'userX'}, {'A-Token': 'xxx
'});
}));


afterEach(function() {
  $httpBackend.verifyNoOutstandingExpectation();
  $httpBackend.verifyNoOutstandingRequest();
});


it('should fetch authentication token', function() {
  $httpBackend.expectGET('/auth.py');
  var controller = scope.$new(MyController);
  $httpBackend.flush();
});


it('should send msg to server', function() {
  // сейчас вы не заботитесь об аутентификации, но
  // контроллер по прежнему будет отправлять запрос и
  // $httpBackend будет отвечать, без необходимости
  // указывать ожидание и ответ для запроса
  $httpBackend.expectPOST('/add-msg.py', 'message content').respond(201, '');

  var controller = scope.$new(MyController);
  $httpBackend.flush();
  controller.saveMessage('message content');
  expect(controller.status).toBe('Saving...');
  $httpBackend.flush();
  expect(controller.status).toBe('');
});


it('should send auth header', function() {
  $httpBackend.expectPOST('/add-msg.py', undefined, function(headers) {
    // проверка, был ли отправлен заголовок, если не ожидается
    // тест закончится неудачно
```

```
61.     return headers['Authorization'] == 'xxx';
62.   }).respond(201, '');
63.
64.   var controller = scope.$new(MyController);
65.   controller.saveMessage('whatever');
66.   $httpBackend.flush();
67. });
```

Methods

expect(method, url, data, headers)

Создает новый ожидание для запроса.

Параметры:

- `method – {string} –` HTTP метод.
- `url – {string|RegExp} –` HTTP url.
- `data`*`(optional)`* `– {(string|RegExp)=} –` HTTP тело запроса.
- `headers`*`(optional)`* `– {(Object|function(Object))=} –` HTTP заголовки или функция которая извлекает http заголовок и возвращает true если заголовок найден в текущем ожидании.

Возврат:

`{requestHandler}` – возвращает объект, который имеет метод `respond`, который контролирует как обрабатывается совпадающий запрос.

- respond – `{function([status,] data[, headers])|function(function(method, url, data, headers)}` – Метод respond принимает набор статических данных которые будут возвращены, или функцию, которая может возвращать массив, содержащий состояние ответа (число), данные ответа (строка) и заголовки ответа (объект).

expectDELETE(url, headers)

Создает новое ожидание для запроса DELETE. Для более детальной информации смотрите `expect()`.

Параметры:

- `url – {string|RegExp} –` HTTP url.
- `headers`*`(optional)`* `– {Object=} –` HTTP заголовки.

Возврат:

`{requestHandler}` – возвращает объект имеющий метод `respond`, который контролирует как обрабатывается совпадающий запрос.

expectGET(url, headers)

Создает новое ожидание для запроса GET. Для большей информации смотрите `expect()`.

Параметры:

- `url – {string|RegExp} –` HTTP url.

- `headers(optional)` – {Object=} – HTTP заголовки.

Возврат:

{requestHandler} – Возвращает объект, содержащий метод `respond`, который контролирует как будет обрабатываться совпадающий запрос.

expectHEAD(url, headers)

Создает новое ожидание для запроса HEAD. Для большей информации смотрите `expect()`.

Параметры:

- `url` – {string|RegExp} – HTTP url.
- `headers(optional)` – {Object=} – HTTP заголовки.

Возврат:

{requestHandler} – Возвращает объект, который содержит метод `respond`, контролирующий как будет обрабатываться совпадающий запрос.

expectJSONP(url)

Создает новое ожидание для запроса JSONP. Для большей информации смотрите `expect()`.

Параметры:

- `url` – {string|RegExp} – HTTP url.

Возврат:

{requestHandler} – Возвращает объект, содержащий метод `respond`, который контролирует как будет обрабатываться совпадающий запрос.

expectPATCH(url, data, headers)

Создает новое ожидание для запроса PATCH. Для большей информации смотрите `expect()`.

Параметры:

- `url` – {string|RegExp} – HTTP url.
- `data(optional)` – {(string|RegExp)=} – HTTP тело запроса.
- `headers(optional)` – {Object=} – HTTP заголовки.

Возврат:

{requestHandler} – Возвращает элемент, содержащий метод `respond`, который контролирует как будет обрабатываться совпадающий запрос.

expectPOST(url, data, headers)

Создает новое ожидание для запроса POST. Для большей информации смотрите `expect()`.

Параметры:

- `url` – {string|RegExp} – HTTP url.
- `data(optional)` – {(string|RegExp)=} – HTTP запроса тела.
- `headers(optional)` – {Object=} – HTTP заголовки.

Возврат:

`{requestHandler}` – Возвращает объект, который содержит метод `respond`, контролирующий как будет обрабатываться совпадающий запрос.

expectPUT(url, data, headers)

Создает новое ожидание для запроса PUT. Для большей информации смотрите `expect()`.

Параметры:

- `url – {string|RegExp} –` HTTP url.
- `data`*`(optional)`* `– {(string|RegExp)=} –` HTTP тело запроса.
- `headers`*`(optional)`* `– {Object=} –` HTTP заголовки.

Возврат:

`{requestHandler}` – Возвращает объект, содержащий метод `respond`, который контролирует как обрабатывается совпадающий запрос.

flush(count)

Разрешает все ожидающие запросы с использование подготовленных ответов.

Параметры:

- `count`*`(optional)`* `– {number=} –` Номер ответов для разрешения (в порядке их объявления). Если не определено, все запросы будут разрешены. Если нет ожидающих запросов, тогда этот метод возбудить исключение (это обычный соглашение при программировании).

resetExpectations()

Сбрасывает все ожидания запросов, но сохраняет все бэкенд определения. Типично, вы можете вызывать метод resetExpectations при выполнении много фазных тестов, когда вам нужно повторно использовать тот же поддельный объект $httpBackend.

verifyNoOutstandingExpectation()

Проверяет, что все запросы, определенные с использование `expect` api, были сделаны. Если любой из запросов не был сделан, verifyNoOutstandingExpectation выбрасывает исключение.

Как правило, вы будете вызывать этот метод после каждого тестового случая, когда утверждение используется внутри блока "afterEach".

```
1. afterEach($httpBackend.verifyExpectations);
```

verifyNoOutstandingRequest()

Проверяется что нет не сброшенных результатов запросов.

Как правило, вы будете вызывать этот метод после каждого тестового случая, когда утверждение используется внутри блока "afterEach".

```
1. afterEach($httpBackend.verifyNoOutstandingRequest);
```

when(method, url, data, headers)

Создает новое бэкенд определение.

Параметры:

- `method – {string} –` HTTP метод.
- `url – {string|RegExp} –` HTTP url.
- `data`*`(optional)`* `– {(string|RegExp)=} –` HTTP тело запроса.
- `headers`*`(optional)`* `– {(Object|function(Object))=} –` HTTP заголовки или функция которая извлекает объект http заголовков и возвращает true, если имеются в текущем определении.

Возврат:

`{requestHandler}` – Возвращает объект, который содержит метод `respond`, контролирующий как будет обрабатываться совпадающий запрос.

- respond – `{function([status,] data[, headers])|function(function(method, url, data, headers)}` – Метод respond устанавливает статические данные которые должны возвращаться, или функцию которая будет возвращать массив содержащий статус ответа (число), данные ответа (строка) и заголовки ответа (объект).

whenDELETE(url, headers)

Создает новое бэкенд определение для запроса DELETE. Для большей информации смотрите `when()`.

Параметры:

- `url – {string|RegExp} –` HTTP url.
- `headers`*`(optional)`* `– {(Object|function(Object))=} –` HTTP заголовки.

Возврат:

`{requestHandler}` – Возвращает объект, который содержит метод `respond`, контролирующий как будет обрабатываться совпадающий запрос.

whenGET(url, headers)

Создает новое бэкенд определение для запроса GET. Для большей информации смотрите `when()`.

Параметры:

- `url – {string|RegExp} –` HTTP url.
- `headers`*`(optional)`* `– {(Object|function(Object))=} –` HTTP заголовки.

Возврат:

`{requestHandler}` – Возвращает объект, который содержит метод `respond`, контролирующий как будет обрабатываться совпадающий запрос.

whenHEAD(url, headers)

Создает новое бэкенд определение для запроса HEAD. Для большей информации смотрите `when()`.

Параметры:

- `url – {string|RegExp} –` HTTP url.

- `headers(optional) – {(Object|function(Object))=} –` HTTP заголовки.

Возврат:

`{requestHandler}` – Возвращает объект, который содержит метод `respond`, контролирующий как будет обрабатываться совпадающий запрос.

whenJSONP(url)

Создает новое бэкенд определение для запроса JSONP. Для большей информации смотрите `when()`.

Параметры:

- `url – {string|RegExp} –` HTTP url.

Возврат:

`{requestHandler}` – Возвращает объект, который содержит метод `respond`, контролирующий как будет обрабатываться совпадающий запрос.

whenPOST(url, data, headers)

Создает новое бэкенд определение для запроса POST. Для большей информации смотрите `when()`.

Параметры:

- `url – {string|RegExp} –` HTTP url.
- `data(optional) – {(string|RegExp)=} –` HTTP тело запроса.
- `headers(optional) – {(Object|function(Object))=} –` HTTP заголовки.

Возврат:

`{requestHandler}` – Возвращает объект, который содержит метод `respond`, контролирующий как будет обрабатываться совпадающий запрос.

whenPUT(url, data, headers)

Создает новое бэкенд определение для запроса PUT. Для большей информации смотрите `when()`.

Параметры:

- `url – {string|RegExp} –` HTTP url.
- `data(optional) – {(string|RegExp)=} –` HTTP тело запроса.
- `headers(optional) – {(Object|function(Object))=} –` HTTP заголовки.

Возврат:

`{requestHandler}` – Возвращает объект, который содержит метод `respond`, контролирующий как будет обрабатываться совпадающий запрос.

$log

Описание

Мок реализация `ng.$log` которая собирает все сообщения системы логирования в массив (один массив на каждый уровень логирования). Эти массивы предоставляются через свойство `logs` для каждого уровня логирования, например для уровня ошибок этот массив доступен через `$log.error.logs`.

Методы

assertEmpty()

Утверждение, что все методы логирования не имеют сообщений. Если сообщение имеется, выбрасывается исключение.

reset()

Сбрасывает все массивы сообщений в пустые массивы.

Свойства

logs

Массив сообщений системы логирования.

$timeout

Описание

Этот сервис простая декорация для сервиса `ng.$timeout` с добавление метода "flush".

Использование

```
1. $timeout();
```

Методы

flush()

Сбрасывает потребителям результаты очереди отложенных задач.

Глобальное API

angular.mock.dump

Описание

Примечание: это не экземпляр для инъекций зависимостей, только глобально доступная функция.

Метод для сериализации управляющих объектов angular (области видимости, элементы, и т.д.) в строку, обычно используемый для отладки.

Этот метод также доступен для window, и может быть использован для отображения объекта в отладочной консоли.

Использование

```
angular.mock.dump(object);
```

Параметры

- `object – {*} –` любой объект для преобразования в строку.

Возврат

`{string}` – сериализованная строка представляющая аргумент.

angular.mock.inject

Описание

Примечание: Эта функция также доступна на объекте window для более легкого использования.
Примечание: Доступна только с jasmine.

Внедряет функцию обертку другой функции в функцию, где должна использоваться основная функция. Метод inject() создает новый экземпляр сервиса `$injector` для каждого теста, который используется для разрешения зависимостей.

Смотрите также `модули`

Пример типично теста jasmine показывающий как используется метод inject.

```
angular.module('myApplicationModule', [])
    .value('mode', 'app')
    .value('version', 'v1.0.1');


describe('MyApp', function() {

  // Вам нужно загрузит модули, которые вы хотите протестировать,
  // по умолчанию загружается только модуль "ng".
  beforeEach(module('myApplicationModule'));


  // inject() используется для внедрения аргументов - всех заданных функций
  it('should provide a version', inject(function(mode, version) {
    expect(version).toEqual('v1.0.1');
    expect(mode).toEqual('app');
  }));


  // Методы inject и module могут также использовать по другому,
  // внутри it или beforeEach
  it('should override a version and test the new version is injected', function()
{
    // module() работает с функциями или строками (псевдонимами модулей)
    module(function($provide) {
      $provide.value('version', 'overridden'); // здесь переопределяется версия
    });
```

```
27.
28.     inject(function(version) {
29.       expect(version).toEqual('overridden');
30.     });
31.   ));
32. });
```

Использование

```
1. angular.mock.inject(fns);
```

Параметры

- `fns – {...Function}` – любое количество функций, которые будут внедрены используя injector.

angular.mock.module

Описание

Примечание: Для упрощения доступа, эта функция опубликована также на объекте window.
Примечание: Доступна только с jasmine.

Эта функция регистрирует код конфигурационного модуля. Она собирает конфигурационную информацию, которая будет использована при создании инжектор через `inject`.

Смотрите `inject` для примера использования.

Использование

```
1. angular.mock.module(fns);
```

Параметры

- `fns – {...(string|Function)}` – любое количество модулей, которые представляются как строки псевдонимов или как функции инициализации анонимных модулей. Эти модули используются для конфигурации инжектора. Модули 'ng' и 'ngMock' загружаются автоматически.

angular.mock.TzDate

Описание

Примечание: это не экземпляр для инъекций, просто глобально видимый мок класс для `Date`.

Мок класс для типа Date, который принимает временную зону в качестве аргумента конструктора.

Главная цель создания этого класса с временной зоной, указание требуемой временной зоны, так что вы можете тестировать код в зависимости от временной зоны установленной на машине, где код выполняется.

Использование

```
TzDate(offset, timestamp);
```

Параметры

- `offset – {number} –` Сдвиг для нужной временной зоны в часах (дробная часть для представления минут)
- `timestamp – {(number|string)} –` Временной отпечаток UNIX для требуемого времени в *UTC*

Пример

!!!! ПРЕДУПРЕЖДЕНИЕ !!!!! Это не полный объект Date так что безопасно можно вызывать только методы, которые были реализованы. Что еще хуже, экземпляры TzDate наследуются от Date с использованием прототипа.

Мы делаем все возможное, для перехвата «нереализованных» методов, но т.к. список методов является не полным, некоторые не стандартные методы могут отсутствовать. Это может в результате приводить к ошибкам типа: "Date.prototype.foo вызвана на не реализующем объекте Object".

```
var newYearInBratislava = new TzDate(-1, '2009-12-31T23:00:00Z');
newYearInBratislava.getTimezoneOffset() => -60;
newYearInBratislava.getFullYear() => 2010;
newYearInBratislava.getMonth() => 0;
newYearInBratislava.getDate() => 1;
newYearInBratislava.getHours() => 0;
newYearInBratislava.getMinutes() => 0;
```

Модуль auto

Сервисы

$injector

Описание

`$injector` используется для извлечения экземпляров объектов как определено в `provider`, создания типов, вызова методов и загрузки модулей.

Следующий код всегда выполнится успешно:

```
var $injector = angular.injector();
expect($injector.get('$injector')).toBe($injector);
expect($injector.invoke(function($injector){
  return $injector;
}).toBe($injector);
```

Нотация функции для инъекции

JavaScript не имеет нотаций, но нотации нужны для инъекции зависимостей. Ниже показаны все допустимые способы нотаций функций для инъекции аргументов. Эти способы эквивалентны.

```
// выведением (работает только если код не минифицирован / обфрусцирован)
$injector.invoke(function(serviceA){});

// нотацией
function explicit(serviceA) {};
explicit.$inject = ['serviceA'];
$injector.invoke(explicit);

// в одной строке
$injector.invoke(['serviceA', function(serviceA){}]);
```

Вывод

В JavaScript вызов `toString()` для функции возвращает ее определение. Это определение можно разобрать и извлечь аргументы. *Примечание:* Это не работает с минификацией кода и обфрускацией, т.к. эти инструменты меняют названия переменных.

Нотация $inject

Параметры могут быть заданы путем добавления `$inject` в функцию инъекции.

В одну строку

В этом случае передается массив и именами для инъекции, в котором последним элементом является выполняемая функция.

Методы

annotate(fn)

Возвращает массив имен сервисов, которые функция запрашивает для инъекции. Это API используется для указания, какие сервисы нужны для инъекции в функцию для ее выполнения. Есть три способа, как функция должна аннотироваться если требуются зависимости.

Имена аргументов

Простая форма при которой зависимости извлекаются из аргументов функции. Это делается с помощью конвертации функции в строку используя метод `toString()` и извлечения имен переменных.

```
// Получение
function MyController($scope, $route) {
  // ...
}

// затем
expect(injector.annotate(MyController)).toEqual(['$scope', '$route']);
```

Этот метод не работает когда применяется минификация / обфрускация. Это поддерживается следующими способами аннотации.

Свойство `$inject`

Если функция имеет свойство `$inject` и его значением является массив строк, тогда строки будут интерпретироваться как имена сервисов для инъекции в функцию.

```
// Получаем
var MyController = function(obfuscatedScope, obfuscatedRoute) {
  // ...
}
// Определяем зависимости функции
MyController.$inject = ['$scope', '$route'];

// проверяем
expect(injector.annotate(MyController)).toEqual(['$scope', '$route']);
```

Нотация в виде массива

Часто желательно в одной строке определять функцию с зависимостями, и тогда использование свойства `$inject` не очень удобно. В этом случае следует использовать нотацию с массивом, что не страдает при минификации и часто является лучшим выбором:

```
// Мы можем написать так (не безопасно для minification / obfuscation)
injector.invoke(function($compile, $rootScope) {
  // ...
});

// нотация с помощью свойства $inject
var tmpFn = function(obfuscatedCompile, obfuscatedRootScope) {
  // ...
};
tmpFn.$inject = ['$compile', '$rootScope'];
injector.invoke(tmpFn);

// Для более удобной работы поддерживается нотации в одну строку
injector.invoke(['$compile', '$rootScope', function(obfCompile, obfRootScope)
  {
  // ...
}]);

// Поэтому
expect(injector.annotate(
   ['$compile', '$rootScope', function(obfus_$compile, obfus_$rootScope) {}])
 ).toEqual(['$compile', '$rootScope']);
```

Параметры:

- `fn - {function|Array.<string|Function>}` - Функция, для которой нужны имена сервисов, от которых она зависит, для их извлечения в дальнейшем.

Возврат:

`{Array.<string>}` – Имена сервисов, которые требуются функции.

get(name)

Возвращает экземпляр сервиса.

Параметры:

- `name – {string} –` Имя для извлечения экземпляра.

Возврат:

`{*}` – требуемый экземпляр.

instantiate(Type, locals)

Создает новый экземпляр типа JS. Этот метод вызывает функцию конструктора оператором new, и снабжает ее всеми аргументами, которые указаны в нотации.

Параметры:

- `Type – {function} –` нотация функции конструктора.
- `locals(optional) – {Object=} –` Необязательный объект. Если предоставлен, тогда любые аргументы, перед тем как вызвать `$injector`, сначала ищутся в одноименном свойстве этого объекта.

Возврат:

`{Object}` – новый экземпляр указанного типа.

invoke(fn, self, locals)

Вызывает метод и предоставляет ему аргументы с помощью сервиса `$injector`.

Параметры:

- `fn – {!function} –` Функция для вызова. Аргументы функции должны иметь допустимую нотацию.
- `self(optional) – {Object=} –` Контекст `this` для вызываемого метода.
- `locals(optional) – {Object=} –` Необязательный объект. Если предоставлен, тогда перед тем как вызвать `$injector`, аргументы для метода будут искаться в одноименных свойствах этого объекта.

Возврат:

`{*}` – значение, которое возвращает вызываемая функция.

$provide

Описание

Сервис `$provide` регистрирует новых провайдеров для `$injector`. Провайдеры, это фабрики для создания экземпляров. Провайдеры разделяют имена с экземплярами и именуются с использование суффикса `Provider`.

Объект провайдера имеет метод `$get()`. Инжектор вызывает метод `$get` для создания нового экземпляра сервиса. Провайдер может иметь дополнительные методы, которые позволяют его настраивать.

```
1. function GreetProvider() {
```

```
2.    var salutation = 'Hello';
3.
4.    this.salutation = function(text) {
5.      salutation = text;
6.    };
7.
8.    this.$get = function() {
9.      return function (name) {
10.       return salutation + ' ' + name + '!';
11.     };
12.   };
13. }
14.
15. describe('Greeter', function(){
16.
17.   beforeEach(module(function($provide) {
18.     $provide.provider('greet', GreetProvider);
19.   }));
20.
21.   it('should greet', inject(function(greet) {
22.     expect(greet('angular')).toEqual('Hello angular!');
23.   }));
24.
25.   it('should allow configuration of salutation', function() {
26.     module(function(greetProvider) {
27.       greetProvider.salutation('Ahoj');
28.     });
29.     inject(function(greet) {
30.       expect(greet('angular')).toEqual('Ahoj angular!');
31.     });
32.   });
```

Методы

constant(name, value)

Значение константы, но в отличии от `value` может быть использовано в конфигурационной функции (других модулей) can be injected into configuration function (other modules) и оно не перехватывается `decorator`.

Параметры:

- `name – {string} –` имя константы.
- `value – {*} –` значение константы.

Возврат:

`{Object}` – зарегистрированный экземпляр

decorator(name, decorator)

Декоратор для сервиса, позволяет перехватывать создание экземпляра сервиса. Возвращаемый экземпляр может быть оригинальным экземпляром, эли новым экземпляром, который делегирует работу оригинальному экземпляру.

Параметры:

- `name – {string} –` Имя для декоратора сервиса.
- `decorator – {function()} –` Эта функция будет вызвана когда необходимо создать требуемый сервис. Эта функция вычисляется с помощью метода `injector.invoke` и соответственно поддерживает все возможности внедрения зависимостей. Локально внедряемые аргументы:
  - `$delegate` – Экземпляр оригинального сервиса, для которого нужно изменить api, настроить, применить паттерн декоратор, или просто вернуть его.

factory(name, $getFn)

Короткая версия для конфигурации сервиса, если требуется только метод `$get`.

Параметры:

- `name – {string} –` Имя для экземпляра.
- `$getFn – {function()} –` Функция $getFn для создания экземпляра. Внутри это преобразуется в `$provide.provider(name, {$get: $getFn})`.

Возврат:

`{Object}` – зарегистрированный экземпляр провайдера

provider(name, provider)

Регистрирует провайдера для сервиса. Провайдеры могут извлекаться и могут иметь дополнительные методы для настройки.

Параметры:

- `name – {string} –` Имя экземпляра. Примечание: провайдер будет доступен через `name + 'Provider'`.
- `provider – {(Object|function())} –` Если это:
  - `Object`: который должен иметь метод `$get`. Метод `$get` будет вызываться используя `$injector.invoke()`, когда нужно создать экземпляр.
  - `Constructor`: новый экземпляр провайдера будет создан используя `$injector.instantiate()`, который рассматривается как `object`.

Возврат:

`{Object}` – зарегистрированный экземпляр провайдера

service(name, constructor)

Сокращение для регистрации сервиса для данного класса.

Параметры:

- `name – {string} –` Имя экземпляра.
- `constructor – {Function} –` Класс (функция конструктор), который будет создавать экземпляры.

Возврат:

`{Object}` – зарегистрированный экземпляр провайдера

value(name, value)

Сокращение для настройки сервисов, содержащих метод `$get`.

Параметры:

- `name – {string} –` имя экземпляра.
- `value – {*} –` значение.

Возврат:

`{Object}` – зарегистрированный экземпляр провайдера

Модуль ngCookie

Сервисы

$cookies

Описание

Провайдер для чтения / записи куков браузера.

Предоставляется только простой объект, и добавляются или удаляются свойства из этого объекта, новые куки создаются / удаляются в конце текущего $eval.

Зависимости

- $browser

Пример

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
  </body>
</html>
```

Script.js

```
function ExampleController($cookies) {
  // Получение куков
  var favoriteCookie = $cookies.myFavorite;
  // установка куков
  $cookies.myFavorite = 'oatmeal';
```

```
6. }
```

$cookieStore

Описание

Предоставляет ключ-значение (строка-объект) хранилище, которое хранит сессионные куки. Объекты вставляются или извлекаются из хранилища автоматически, при этом сериализуются или десериализуются с использование методов angular: toJson и fromJson.

Зависимости

- $cookies

Методы

get(key)

Возвращает значение для ключа куки.

Параметры:

- `key – {string} –` идентификатор для просмотра.

Возврат:

`{Object}` – Десериализованное значение куки.

put(key, value)

Устанавливает значение для ключа куки.

Параметры:

- `key – {string} –` идентификатор для установки значения.
- `value – {Object} –` значение для сохранения.

remove(key)

Удаляет определенную куки.

Параметры:

- `key – {string} –` Идентификатор удаляемой куки.

Модуль ngMockE2E

Сервисы

$httpBackend

Описание

Мок реализация HTTP для end-to-end тестирования или backend-less разработчиков для приложений использующих `сервис $http`.

Примечание: Для целей блочного тестирования используется другая мок реализация, смотрите unit-testing $httpBackend mock.

Эта реализация может использоваться для статических или динамических ответов с использованием when api или его сокращений (whenGET, whenPOST, и т.д.) и необязательно отправляет запросы через реальный $httpBackend для определенных запросов (например, для взаимодействия с некоторыми удаленными сервисами или для получения шаблонов с сервера).

Как и в блочном тестировании, в сценариях end-to-end тестирования, или в сценариях, когда приложение будет разрабатываться, с реальное серверное api заменяется на моки, но часто возникает необходимости обойти мок, и осуществить запрос к реальному серверу (например, для получения шаблонов или статических файлов от сервера). Чтобы настроить такое поведение используйте passThrough обработчик запроса для when вместо respond.

К тому же, мы не хотим вручную сбрасывать результаты выполнения запросов потребителям, как мы это делали при модульном тестировании. По этой причине e2e $httpBackend автоматически сбрасывает значения из моков, закрывая симуляцию поведения объекта XMLHttpRequest.

Чтобы настроить приложение на запуск с http взаимодействием, вам нужно создать модуль зависимый от ngMockE2E и модулей вашего приложения и определить мок:

```
1.  myAppDev = angular.module('myAppDev', ['myApp', 'ngMockE2E']);
2.  myAppDev.run(function($httpBackend) {
3.    phones = [{name: 'phone1'}, {name: 'phone2'}];
4.
5.    // возврат текущего списка телефонов
6.    $httpBackend.whenGET('/phones').respond(phones);
7.
8.    // добавление нового телефона в массив телефонов
9.    $httpBackend.whenPOST('/phones').respond(function(method, url, data) {
10.     phones.push(angular.fromJSON(data));
11.   });
12.   $httpBackend.whenGET(/^\/templates\//).passThrough();
13.   //...
14. });
```

После этого, начните загрузку приложения с этого модуля.

Methods

when(method, url, data, headers)

Создает новое ожидание бэкенд.

Параметры:

- `method – {string} –` HTTP метод.
- `url – {string|RegExp} –` HTTP url.
- `data`*`(optional)`* `– {(string|RegExp)=} –` HTTP тело запроса.
- `headers`*`(optional)`* `– {(Object|function(Object))=} –` HTTP заголовки или функция извлекающая объект с http заголовков и возвращающая true, если заголовки найдены в текущем определении.

Возврат:

`{requestHandler}` – Возвращает объект, содержащий методы `respond` и `passThrough`, которые контролируют как обрабатываются совпадающие запросы.

- respond – `{function([status,] data[, headers])|function(function(method, url, data, headers)}` – Этот метод устанавливает статические данные для возврата или функцию, которая будет возвращать массив, содержащий статус ответа (число), данные ответа (строка) и заголовки ответа (объект).
- passThrough – `{function()}` – Любой запрос, который совпадает с определенным ожиданием в обработчике `passThrough`, будет проходить через реальный бэкенд (и XHR запрос будет отправлен на сервер).

whenDELETE(url, headers)

Создает новое бэкенд определение для запросов DELETE. Для большей информации смотрите `when()`.

Параметры:

- `url – {string|RegExp} –` HTTP url.
- `headers`*`(optional)`* `– {(Object|function(Object))=} –` HTTP заголовки.

Возврат:

`{requestHandler}` – Возвращает объект, содержащий методы `respond` и `passThrough`, которые контролируют как обрабатываются совпадающие запросы.

whenGET(url, headers)

Создает новое бэкенд определение для запросов GET. Для большей информации смотрите `when()`.

Параметры:

- `url – {string|RegExp} –` HTTP url.
- `headers`*`(optional)`* `– {(Object|function(Object))=} –` HTTP заголовки.

Возврат:

`{requestHandler}` – Возвращает объект, содержащий методы `respond` и `passThrough`, которые контролируют как обрабатываются совпадающие запросы.

whenHEAD(url, headers)

Создает новое бэкенд определение для запросов HEAD. Для большей информации смотрите `when()`.

Параметры:

- `url – {string|RegExp} –` HTTP url.
- `headers`*`(optional)`* `– {(Object|function(Object))=} –` HTTP заголовки.

Возврат:

`{requestHandler}` – Возвращает объект, содержащий методы `respond` и `passThrough`, которые контролируют как обрабатываются совпадающие запросы.

whenJSONP(url)

Создает новое бэкенд определение для запросов JSONP. Для большей информации смотрите `when()`.

Параметры:

- `url – {string|RegExp} –` HTTP url.

Возврат:

`{requestHandler}` – Возвращает объект, содержащий методы `respond` и `passThrough`, которые контролируют как обрабатываются совпадающие запросы.

whenPATCH(url, data, headers)

Создает новое бэкенд определение для запросов PATCH. Для большей информации смотрите `when()`.

Параметры:

- `url – {string|RegExp} –` HTTP url.
- `data`*`(optional)`* `– {(string|RegExp)=} –` HTTP тело запроса.
- `headers`*`(optional)`* `– {(Object|function(Object))=} –` HTTP заголовки.

Возврат:

`{requestHandler}` – Возвращает объект, содержащий методы `respond` и `passThrough`, которые контролируют как обрабатываются совпадающие запросы.

whenPOST(url, data, headers)

Создает новое бэкенд определение для запросов POST. Для большей информации смотрите `when()`.

Параметры:

- `url – {string|RegExp} –` HTTP url.
- `data`*`(optional)`* `– {(string|RegExp)=} –` HTTP тело запроса.
- `headers`*`(optional)`* `– {(Object|function(Object))=} –` HTTP заголовки.

Возврат:

`{requestHandler}` – Возвращает объект, содержащий методы `respond` и `passThrough`, которые контролируют как обрабатываются совпадающие запросы.

whenPUT(url, data, headers)

Создает новое бэкенд определение для запросов PUT. Для большей информации смотрите `when()`.

Параметры:

- `url – {string|RegExp} –` HTTP url.
- `data`*`(optional)`* `– {(string|RegExp)=} –` HTTP тело запроса.
- `headers`*`(optional)`* `– {(Object|function(Object))=} –` HTTP заголовки.

Возврат:

`{requestHandler}` – Возвращает объект, содержащий методы `respond` и `passThrough`, которые контролируют как обрабатываются совпадающие запросы.

Модуль ngResource

Сервисы

$resource

Описание

Фабрика, которая создает объект ресурса, позволяющий вам взаимодействовать с сервером в манере RESTful.

Возвращаемый объект ресурса имеет методы, которые представляют высокоуровневое поведение, нужное для взаимодействия с сервисом более низкого уровня `$http`.

Установка

Чтобы использовать $resource включите файл `angular-resource.js` в список загружаемых файлов Angular. Вы можете найти этот файл в Google CDN, и он будет находится в code.angularjs.org.

В конце загрузите модуль в ваше приложение:

```
angular.module('app', ['ngResource']);
```

и можете начинать его использовать!

Зависимости

- $http

Использование

```
1. $resource(url[, paramDefaults][, actions]);
```

Параметры

- `url – {string} –` Параметризированный шаблон URL в котором параметры имеют префикс «:», как в `/user/:username`. Если вы используете URL с номером порта (например, `http://example.com:8080/api`), вам нужно экранировать его двойным обратным слешем, как показано здесь: `$resource('http://example.com\\:8080/api')`.
- `paramDefaults(optional) – {Object=} –` по умолчанию значение параметров `url`. Они могут переопределяться в методах действий. Значение каждого ключа в объекте параметров будет связано с шаблоном url, и все пары ключ-значение будут добавлены в url в строку поиска `?`, если шаблон url не содержит параметра с именем ключа. Для шаблона `/path/:verb` и параметра `{verb:'greet', salutation:'Hello'}` в результате получим URL `/path/greet?salutation=Hello`. Если значение параметра указанно с префиксом `@`, тогда значение этого параметра извлекается из объекта данных (обычно для не GET операций).

- `actions(optional)` – {Object.<Object>=} – Хэш с декларацией пользовательских действий, которые расширяют установленные по умолчанию действия. Декларация должна быть в следующем формате:

```
{action1: {method:?, params:?, isArray:?},
 action2: {method:?, params:?, isArray:?},
 ...}
```

Где:

  - `action` – {string} – Имя действия. Это имя, которое будет установлено для метода в вашем ресурсе.
  - `method` – {string} – метод HTTP запроса. Допустимые значения: `GET`, `POST`, `PUT`, `DELETE`, и `JSONP`
  - `params` – {object=} – Необязательно, пред установки для параметров действия.
  - isArray – {boolean=} – Если true, тогда объект возвращаемый этим действием является массивом, смотрите раздел `возврат`.

Возврат

`{Object}` – Объект определенного для ресурса «класса» с действиями по умолчанию для ресурса и расширенный пользовательскими действиями. По умолчанию содержит следующие действия:

```
{ 'get':    {method:'GET'},
  'save':   {method:'POST'},
  'query':  {method:'GET', isArray:true},
  'remove': {method:'DELETE'},
  'delete': {method:'DELETE'} };
```

Выполнение этих методов вызывает `ng.$http` с указанным http методом, назначением и параметрами. Когда данные возвращаются с сервера, создается экземпляр объекта для класса ресурса. Для действий `save`, `remove` и `delete` доступны как методы, с префиксом `$`. Это позволяет легко поддерживать CRUD операции (create, read, update, delete) на полученных с сервера данных:

```
1. var User = $resource('/user/:userId', {userId:'@id'});
2. var user = User.get({userId:123}, function() {
3.   user.abc = true;
4.   user.$save();
5. });
```

Важно, что сразу после вызова метода объекта $resource, возвращается пустая ссылка (объект или массив в зависимости `isArray`). Когда данные будут возвращены с сервера, существующая ссылка заполняется фактическими данными. Это полезный трюк, т.к. обычно ресурсы назначаются для моделей, которые отображаются в представлении. Когда ресурс пустой, он не отображается в представлении, но как только приходят данные, представление перерисовывается, для показа новых данных. Это означает, что в большинстве случаев не нужно писать функцию обратного вызова для методов действий.

Методы действий на объекте класса или на экземпляре объекта могут быть вызваны со следующими параметрами:

- HTTP GET "класс" действия: `Resource.действие([parameters], [success], [error])`
- не-GET "класс" действия: `Resource.действие([parameters], postData, [success], [error])`
- не-GET экземпляра действия: `экземпляр.$action([parameters], [success], [error])`

Пример

Ресурс представляющий кредитную карту

```
1. // определение класса CreditCard
2. var CreditCard = $resource('/user/:userId/card/:cardId',
3.  {userId:123, cardId:'@id'}, {
4.   charge: {method:'POST', params:{charge:true}}
5.  });
6.
7. // Вы можете извлечь коллекцию из сервера
8. var cards = CreditCard.query(function() {
9.   // GET: /user/123/card
10.  // сервер вернул: [ {id:456, number:'1234', name:'Smith'} ];
11.
12.  var card = cards[0];
13.  // каждый элемент это экземпляр CreditCard
14.  expect(card instanceof CreditCard).toEqual(true);
15.  card.name = "J. Smith";
16.  // не GET методы могут вызываться на экземпляре
17.  card.$save();
18.  // POST: /user/123/card/456 {id:456, number:'1234', name:'J. Smith'}
19.  // сервер вернул: {id:456, number:'1234', name: 'J. Smith'};
20.
21.  // Вызов вашего метода на экземпляре
22.  card.$charge({amount:9.99});
23.  // POST: /user/123/card/456?amount=9.99&charge=true {id:456, number:'1234', name
   :'J. Smith'}
24. });
25.
26. // вы можете создать новый экземпляр как здесь
27. var newCard = new CreditCard({number:'0123'});
28. newCard.name = "Mike Smith";
29. newCard.$save();
30. // POST: /user/123/card {number:'0123', name:'Mike Smith'}
31. // сервер вернул: {id:789, number:'01234', name: 'Mike Smith'};
32. expect(newCard.id).toEqual(789);
```

Возвращенный из функции создания ресурса объект «класса» содержит статические методы для каждого определенного действия.

Выполнение этих методов вызывает `$http` с шаблоном `url` с полученными методом `method` и параметрами `params`. Когда данные возвращены сервером, тогда создается объект с определенным для ресурса типом, в который содержит все не-GET методы, доступные через методы с префиксом `$`. Это позволяет легко поддерживать операции CRUD (create, read, update, delete) на полученных с сервера данных.

```
var User = $resource('/user/:userId', {userId:'@id'});
var user = User.get({userId:123}, function() {
  user.abc = true;
  user.$save();
});
```

Стоить отметить, что в случае успешного завершения запросов `get`, `query` или других действий, будет вызвана соответствующая функция обратного вызова, которой передается ответ сервера, а также функция для получения $http заголовков, поэтому для получения доступа к заголовкам можно переписать код выше в следующей манере:

```
var User = $resource('/user/:userId', {userId:'@id'});
User.get({userId:123}, function(u, getResponseHeaders){
  u.abc = true;
  u.$save(function(u, putResponseHeaders) {
    //u => сохраненный объект
    //putResponseHeaders => получатель заголовков $http
  });
});
```

Бизнес клиент

Давайте посмотрим, как можно создать бизнес клиента, используя сервис `$resource`:

Index.html

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-controller="BuzzController">
      <input ng-model="userId"/>
      <button ng-click="fetch()">fetch</button>
      <hr/>
      <div ng-repeat="item in activities.data.items">
        <h1 style="font-size: 15px;">
          <img src="{{item.actor.thumbnailUrl}}" style="max-height:30px;max-width:
30px;"/>
          <a href="{{item.actor.profileUrl}}">{{item.actor.name}}</a>
          <a href ng-click="expandReplies(item)" style="float: right;">Expand repl
ies: {{item.links.replies[0].count}}</a>
        </h3>
        {{item.object.content | html}}
        <div ng-repeat="reply in item.replies.data.items" style="margin-left: 20px
;">
          <img src="{{reply.actor.thumbnailUrl}}" style="max-height:30px;max-width
:30px;"/>
```

```
          <a href="{{reply.actor.profileUrl}}">{{reply.actor.name}}</a>: {{reply.content | html}}
        </div>
      </div>
    </div>
  </body>
</html>
```

Script.js

```
function BuzzController($resource) {
  this.userId = 'googlebuzz';
  this.Activity = $resource(
    'https://www.googleapis.com/buzz/v1/activities/:userId/:visibility/:activityId/:comments',
    {alt:'json', callback:'JSON_CALLBACK'},
    {get:{method:'JSONP', params:{visibility:'@self'}}, replies: {method:'JSONP', params:{visibility:'@self', comments:'@comments'}}}
  );
}

BuzzController.prototype = {
  fetch: function() {
    this.activities = this.Activity.get({userId:this.userId});
  },
  expandReplies: function(activity) {
    activity.replies = this.Activity.replies({userId:this.userId, activityId:activity.id});
  }
};
BuzzController.$inject = ['$resource'];
```

Модуль ngSanitize

Директивы

ngBindHtml

Описание

Создает привязки, которые очищают результат вычисления выражения `expression` с помощью сервиса $sanitize и результат устанавливается с помощью innerHTML в текущий элемент.

Пример смотрите в документации $sanitize.

Использование

как атрибут

```
<ANY ng-bind-html="{expression}">
```

```
    ...
</ANY>
```

как класс

```
<ANY class="ng-bind-html: {expression};">
    ...
</ANY>
```

Параметры

- `ngBindHtml – {expression} –` Выражение для вычисления.

Фильтры

linky

Описание

Находит ссылки во входном тексте и преобразует из в ссылки html. Поддерживаются http/https/ftp/mailto и email ссылки.

Использование

В привязках в HTML шаблонах

```
<span ng-bind-html="linky_expression | linky"></span>
```

В JavaScript

```
$filter('linky')(text)
```

Параметры

- `text – {string} –` входной текст.

Возврат

`{string}` – Текст с ссылками Html.

Пример

Index.html

```
<!doctype html>
<html ng-app="ngSanitize">
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-controller="Ctrl">
    Snippet: <textarea ng-model="snippet" cols="60" rows="3"></textarea>
    <table>
      <tr>
```

```
        <td>Filter</td>
        <td>Source</td>
        <td>Rendered</td>
      </tr>
      <tr id="linky-filter">
        <td>linky filter</td>
        <td>
          <pre>&lt;div ng-bind-html="snippet | linky"&gt;<br>&lt;/div&gt;</pre>
        </td>
        <td>
          <div ng-bind-html="snippet | linky"></div>
        </td>
      </tr>
      <tr id="escaped-html">
        <td>no filter</td>
        <td><pre>&lt;div ng-bind="snippet"&gt;<br>&lt;/div&gt;</pre></td>
        <td><div ng-bind="snippet"></div></td>
      </tr>
    </table>
  </body>
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.snippet =
    'Pretty text with some links:\n'+
    'http://angularjs.org/,\n'+
    'mailto:us@somewhere.org,\n'+
    'another@somewhere.org,\n'+
    'and one more: ftp://127.0.0.1/.';
}
```

End to end test

```
it('should linkify the snippet with urls', function() {
  expect(using('#linky-filter').binding('snippet | linky')).
    toBe('Pretty text with some links:&#10;' +
         '<a href="http://angularjs.org/">http://angularjs.org/</a>,&#10;' +
         '<a href="mailto:us@somewhere.org">us@somewhere.org</a>,&#10;' +
         '<a href="mailto:another@somewhere.org">another@somewhere.org</a>,&#10;'
+
         'and one more: <a href="ftp://127.0.0.1/">ftp://127.0.0.1/</a>.');
});

it ('should not linkify snippet without the linky filter', function() {
  expect(using('#escaped-html').binding('snippet')).
    toBe("Pretty text with some links:\n" +
         "http://angularjs.org/,\n" +
         "mailto:us@somewhere.org,\n" +
```

```
        "another@somewhere.org,\n" +
        "and one more: ftp://127.0.0.1/.");
});

it('should update', function() {
  input('snippet').enter('new http://link.');
  expect(using('#linky-filter').binding('snippet | linky')).
    toBe('new <a href="http://link">http://link</a>.');
  expect(using('#escaped-html').binding('snippet')).toBe('new http://link.');
});
```

Сервисы

$sanitize

Описание

Вход очищается путем разбора лексем html. Все безопасные лексемы (из белого списка) затем возвращаются обратно в строку с экранированным html. Это означает, что не безопасный ввод может быть возвращен в строку, однако, так как наш анализатор является более строгим, чем типичный парсер браузера, вполне возможно, что некоторые данные, которые будут признаны как верный HTML в браузере, не будут признаны таковыми здесь.

Использование

```
$sanitize(html);
```

Параметры

- `html – {string} –` Html вход.

Возврат

`{string}` – очищенный html.

Пример

Index.html

```
<!doctype html>
<html ng-app="ngSanitize">
  <head>
    <script src="http://code.angularjs.org/1.0.6/angular.min.js"></script>
    <script src="script.js"></script>
  </head>
  <body>
    <div ng-controller="Ctrl">
       Snippet: <textarea ng-model="snippet" cols="60" rows="3"></textarea>
        <table>
          <tr>
            <td>Filter</td>
            <td>Source</td>
```

```
            <td>Rendered</td>
          </tr>
          <tr id="html-filter">
            <td>html filter</td>
            <td>
              <pre>&lt;div ng-bind-html="snippet"&gt;<br/>&lt;/div&gt;</pre>
            </td>
            <td>
              <div ng-bind-html="snippet"></div>
            </td>
          </tr>
          <tr id="escaped-html">
            <td>no filter</td>
            <td><pre>&lt;div ng-bind="snippet"&gt;<br/>&lt;/div&gt;</pre></td>
            <td><div ng-bind="snippet"></div></td>
          </tr>
          <tr id="html-unsafe-filter">
            <td>unsafe html filter</td>
            <td><pre>&lt;div ng-bind-html-unsafe="snippet"&gt;<br/>&lt;/div&gt;</pre></td>
            <td><div ng-bind-html-unsafe="snippet"></div></td>
          </tr>
        </table>
      </div>
  </body>
</html>
```

Script.js

```
function Ctrl($scope) {
  $scope.snippet =
    '<p style="color:blue">an html\n' +
    '<em onmouseover="this.textContent=\'PWN3D!\'">click here</em>\n' +
    'snippet</p>';
}
```

End to end test

```
it('should sanitize the html snippet ', function() {
  expect(using('#html-filter').element('div').html()).
    toBe('<p>an html\n<em>click here</em>\nsnippet</p>');
});

it('should escape snippet without any filter', function() {
  expect(using('#escaped-html').element('div').html()).
    toBe("&lt;p style=\"color:blue\"&gt;an html\n" +
         "&lt;em onmouseover=\"this.textContent='PWN3D!'\"&gt;click here&lt;/em&gt;\n" +
         "snippet&lt;/p&gt;");
});
```

```
12.
13. it('should inline raw snippet if filtered as unsafe', function() {
14.   expect(using('#html-unsafe-filter').element("div").html()).
15.     toBe("<p style=\"color:blue\">an html\n" +
16.          "<em onmouseover=\"this.textContent='PWN3D!'\">click here</em>\n" +
17.          "snippet</p>");
18. });
19.
20. it('should update', function() {
21.   input('snippet').enter('new <b>text</b>');
22.   expect(using('#html-filter').binding('snippet')).toBe('new <b>text</b>');
23.   expect(using('#escaped-html').element('div').html()).toBe("new &lt;b&gt;text&lt;
    /b&gt;");
24.   expect(using('#html-unsafe-filter').binding("snippet")).toBe('new <b>text</b>');
25. });
```