

Dariusz Kalbarczyk
Arkadiusz Kalbarczyk



ANGULARJS

PIERWSZE KROKI

Helion 

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Opieka redakcyjna: Ewelina Burska

Projekt okładki: Studio Gravite/Olsztyn

Obarek, Pokoński, Pazdrijowski, Zaprucki

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/angupk>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-0586-1

Copyright © Helion 2015

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Rozdział 1. Wstęp	7
Od czego zacząć	9
Biblioteka i ng-app, czyli bez czego nie może się obejść żadna aplikacja	9
Biblioteka	9
Ng-app	10
Pierwsza aplikacja	11
Framework SPA	13
Podwójne wiązanie	14
Jednostronne wiązanie	14
Dwustronne wiązanie	14
AngularJS i MVC	15
Quiz	16
Rozdział 2. \$scope — niepozorny obiekt	17
Wprowadzenie	17
\$scope i \$rootScope	17
Alternatywa dla \$scope	18
Dziedziczenie	19
Izolowany scope	22
\$digest(), \$apply() i \$watch()	22
Nasłuchiwanie oraz \$watch()	22
\$digest()	24
\$apply()	24
Quiz	26
Rozdział 3. Moduły	27
Wprowadzenie	27
Moduły a kontrolery	28
Moduły a globalna przestrzeń nazw	29
Zmodularyzowana aplikacja	29
Łączenie modułów	30
Quiz	31
Rozdział 4. Dependency Injection — wstrzykiwanie zależności	33
Wprowadzenie	33
Uzyskiwanie zależności	34
Metody wstrzykiwania zależności	35
DI w praktyce	37
Quiz	43

Rozdział 5. Poznaj potęgę dyrektyw	45
Wprowadzenie	45
Nazewnictwo	48
Wbudowane dyrektywy	50
Dyrektywa a	51
Dyrektywa form	51
Dyrektywa input	53
Dyrektywa ngBind	54
Dyrektywa ngBindHtml	54
Dyrektywa ngBindTemplate	55
Dyrektywa ngCloak	56
Dyrektywy ngBlur i ngFocus	57
Dyrektywa ngChange	57
Dyrektywa ngClass	62
Dyrektywa ngRepeat	65
Dyrektywa ngClick	72
Dyrektywa ngController	74
Dyrektywa ngCopy	75
Dyrektywa ngCut	76
Dyrektywa ngDbclick	78
Dyrektywa ngFocus	78
Dyrektywa ngForm	79
Dyrektywa ngHref	79
Dyrektywa ngIf	80
Dyrektywa ngInclude	80
Dyrektywy ngKeydown, ngKeypress i ngKeyup	80
Dyrektywa ngList	81
Dyrektywa ngModel	81
Dyrektywa ngModelOptions	82
Dyrektywy ngMouseDown, ngMouseenter, ngMouseleave, ngMouseMove, ngMouseover i ngMouseup	84
Dyrektywa ngNonBindable	84
Dyrektywa ngPaste	85
Dyrektywa ngPluralize	85
Dyrektywa ngReadonly	88
Dyrektywa ngStyle	88
Dyrektywa ngSubmit	88
Dyrektywa ngSwitch	89
Dyrektywa ngTransclude	89
Dyrektywa ngValue	91
Dyrektywa script	91
Dyrektywa select	93
Dyrektywa textarea	96
Quiz	97
Rozdział 6. Dyrektywy szyte na miarę	99
Wprowadzenie	99
Pierwsza własna dyrektywa	99
Właściwości	101
\$scope vs. scope	105
Quiz	107

Rozdział 7. Filtry	109
Wprowadzenie	109
Filtry wbudowane	110
Operacje na stringach	110
Liczbowe	111
Operacje na datach	112
JSON	113
Filtry dyrektywy ng-repeat	113
Linky	117
Quiz	118
Rozdział 8. Funkcje	119
Wprowadzenie	119
Opis funkcji	119
Funkcja angular.bind	119
Funkcja angular.bootstrap	120
Funkcja angular.copy	120
Funkcja angular.element	122
Funkcja angular.equals	126
Funkcja angular.extend	126
Funkcja angular.forEach	127
Funkcje angular.fromJson i angular.toJson	127
Funkcja angular.identity	127
Funkcja angular.injector	129
Funkcje angular.isArray, angular.isDate, angular.isDefined, angular.isElement, angular.isFunction, angular.isNumber, angular.isObject, angular.isString i angular.isUndefined	131
Funkcje angular.lowercase i angular.uppercase	131
Funkcja angular.module	132
Funkcja angular.reloadWithDebugInfo	132
Quiz	132
Rozdział 9. Routing — lepsza strona nawigacji	133
Wprowadzenie	133
Konfiguracja	134
Widoki	134
Cztery kroki w procesie konfiguracji	151
Quiz	151
Rozdział 10. Animacje	153
Wprowadzenie	153
Jak to działa	154
Obietnice	154
CSS3 Transitions	155
Animacje CSS3 i @keyframes	158
Animacje JavaScript	161
Quiz	167
Rozdział 11. Komunikacja z serwerem	169
Wprowadzenie	169
Klasyczne zapytanie XHR a usługa \$http	169
XHR przy użyciu \$http	170

Odpowiedzi http	172
Promises	172
success() i error()	172
\$.ajax, obietnice i odroczenia	173
\$.ajax.all	176
Przechowywanie odpowiedzi	176
Pozostałe metody \$.ajax	177
Parametry metody \$.ajax	177
Obiekt konfiguracyjny	177
Dane	178
Same origin policy oraz JSONP i CORS na ratunek XHR	179
JSON with padding oraz jego ograniczenia	179
CORS — Cross Origin Resource Sharing	179
Trzecie wyjście: proxy	180
Quiz	180
Rozdział 12. Formularze	181
Wprowadzenie	181
ngFormController	181
Używanie klas CSS	181
Pierwszy formularz	183
Quiz	184
Rozdział 13. Dobre praktyki	185
Wprowadzenie	185
Nazewnictwo i podział plików	185
Organizacja kodu	188
Wydajność	189
Quiz	191
Rozdział 14. Testy	193
Wprowadzenie	193
Jasmine	193
Dopasowania	197
Quiz	204
Rozdział 15. Zakończenie	205
Skorowidz	206

Rozdział 2.

\$scope — niepozorny obiekt

Wprowadzenie

W tym rozdziale zajmiemy się wspomnianym przez nas wcześniej obiektem `$scope`.

Jego podstawowe zadania to:

- ♦ transportowanie modelu pomiędzy widokiem a kontrolerem;
- ♦ nasłuchiwanie zdarzeń bądź zmian zachodzących w modelu;
- ♦ propagacja zmian modelu.

Mimo że odgrywa wyjątkową rolę, `$scope` to wciąż zwykły obiekt POJO. Oznacza to, że możemy dowolnie przypisywać mu oraz modyfikować atrybuty według własnego uznania. Wyróżnia go fakt, iż w większości przypadków jest on za nas automatycznie tworzony i wstrzykiwany.

\$scope i \$rootScope

W fazie ładowania początkowego aplikacji (tzw. *bootstrap*) AngularJS tworzy wiązanie (binduje) pomiędzy znacznikiem zawierającym dyrektywę `ng-app` a wszystkim, co jest zawarte w elementach poniżej.

`$rootScope` jest rodzicem wszystkich obiektów `$scope` i znajduje się najwyżej w hierarchii. Instancja `$rootScope` jest tworzona w momencie bootstrapowania aplikacji. Każdy program posiada dokładnie jeden taki obiekt, po którym dziedziczą wszystkie inne obiekty `scope`. Nie zalecamy przypisywania mu zbyt wielu atrybutów, gdyż jest on czymś na wzór obiektu globalnego, którego nie powinno się zaśmiecać. Przy wykorzystywaniu więcej niż jednej biblioteki lub frameworka istnieje ryzyko wystąpienia

zbieżności nazw atrybutów bądź metod przypisanych do globalnych obiektów. Tego typu problemy są niezwykle uciążliwe w usuwaniu.

Wspominaliśmy już, że każdy element przypisany do `$scope` jest od razu dostępny w widoku. Przypisywanie atrybutów i funkcji do modelu po stronie kontrolera odbywa się w sposób ukazany w listingu 2.1.

Listing 2.1. *Kontroler — przypisanie atrybutów*

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" ng-app="app">
<head>
  <title>Kontroler – przypisanie atrybutów</title>
</head>
<body>
  <div ng-controller="dateCtrl">
    Data: {{original() | date}}
  </div>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/
  ↪1.4.0-beta.5/angular.min.js">
</script>
<script>
  var app = angular.module('app', []);
  app.run(function ($rootScope) {
    $rootScope.dateOriginal = new Date();
  });
  app.controller('dateCtrl', function ($rootScope, $scope) {
    $scope.original = function () {
      return $rootScope.dateOriginal;
    };
  });
</script>
</body>
</html>
```

W powyższym przykładzie zdefiniowaliśmy w `$rootScope` właściwość `dateOriginal`. Następnie w kontrolerze `dateCtrl` stworzyliśmy funkcję `original`, która zwraca nam datę z `$rootScope`.

Alternatywa dla `$scope`

Istnieje również możliwość przypisywania atrybutów do modelu po stronie widoku bez odwoływania się do `scope`. W tym celu korzystamy z dyrektywy `ng-model`. Jest ona dokładnie opisana w rozdziale 5., poświęconym dyrektywom wbudowanym. Na tym etapie warto zapamiętać, że `ng-model` inicjuje nam `$scope`, którego możemy użyć w kontrolerze.

```
<html ng-app>
...
  <div ng-model='wiadomosc'>
    <p {{ wiadomosc }} </p>
  </div>
</html>
```


Stosując się do dobrych praktyk, powinniśmy w miarę możliwości wybierać wariant pierwszy, bliższy ideologii MVC.

Dziedziczenie

W przykładzie z listingu 2.1 wykorzystaliśmy wcześniej wspomniany `$rootScope`. Jak już mówiliśmy, staramy się nie przypisywać atrybutów do obiektu głównego, lecz do nowo utworzonego scope znajdującego się w hierarchii poniżej.

```
app.controller('dateCtrl', function ($scope) {
  $scope.wiadomosc = "Przypisujemy wiadomosc do widoku!";
  $scope.funkcjaA = function() {
    return wiadomosc + "Dodajemy dodatkowe zdanie";
  }
})
```

`$scope` odwzorowuje strukturę DOM. Oznacza to, że możemy swobodnie zagnieżdżać jego obiekty.

Korzystanie z obiektów `$scope` nie wymaga ich wcześniejszej deklaracji. Większość obiektów `$scope` tworzona jest dzięki metodzie `$new()`, wywoływanej za każdym razem, gdy napotykana jest dyrektywa `ng-controller`.

Nowy obiekt zostaje automatycznie zagnieżdżony poniżej obiektu `$rootScope`. Poza jednym wyjątkiem (izolowanym scope) wszystkie obiekty `$scope` mają dostęp do obiektów znajdujących się w hierarchii nad nimi. Jeżeli AngularJS nie znajdzie požądanej informacji w scope na swoim poziomie, to rozpocznie przeszukiwanie obiektu znajdującego się wyżej, aż dojdzie do `$rootScope`.

Zobaczmy na listingu 2.2, jak możemy korzystać z dziedziczenia kontrolerów:

Listing 2.2. Kontrolery — dziedziczenie

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" ng-app="app">
<head>
  <title>Kontrolery – dziedziczenie</title>
</head>
<body>
  <div ng-controller="defaultCtrl">
    <div ng-controller="inheritanceCtrl">
      <input type="text" ng-model="uczen.imie" placeholder="Imie
        ↪Ucznia"></input>
      <button ng-click="poprawaTestu()">Poprawa testu </button>
    </div>
    {{ uczen }}
  </div>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/
    ↪1.4.0-beta.5/angular.min.js">
  </script>
</script>
```

```

var app = angular.module('app', []);
app.controller('defaultCtrl', function ($scope) {
    $scope.uczen = { zdanyTest: false };
});
app.controller('inheritanceCtrl', function ($scope) {
    $scope.poprawaTestu = function () {
        $scope.uczen.zdanyTest = true;
    }
});
</script>
</body>
</html>

```

Ponieważ `inheritanceCtrl` związaliśmy w hierarchii niżej niż `defaultCtrl`, otrzymuje on dostęp do metod kontrolera bazowego. Tutaj warto odnotować, iż dziedziczenie w Angularze odbywa się w jednym kierunku. W tym wypadku kontroler potomny jest silnie powiązany z rodzicem, czyli może odwoływać się do jego metod. Jednakże kontroler bazowy nie może bezpośrednio odwoływać się do potomka. By uzyskać dostęp do owych metod, należy wykorzystać przesyłanie zdarzeń (ang. *event dispatching*). W większości przypadków, kiedy musimy odwoływać się do metod potomnych, oznacza to, że powinniśmy się przyjrzeć naszemu kodowi, gdyż najprawdopodobniej robimy coś źle.

Aby później mieć możliwość odwołania się do naszego scope, musimy umieścić dyrektywę `ng-controller` w dowolnym elemencie DOM znajdującym się na tym samym bądź wyższym poziomie hierarchii co model (a konkretnie nasze odwołanie do niego poprzez interpolację).

```

<html ng-app>
...
<div ng-controller="Kontroler">
    <p> {{ wiadomosc }} </p>
</div>
...
</html>

```

Dyrektywa `ng-controller` należy do grupy tzw. tworzących dyrektyw. Za każdym razem, gdy Angular napotyka jedną z takich dyrektyw, zostaje utworzona nowa instancja scope, dlatego wcześniejsza deklaracja w kontrolerze nie jest wymagana.

Wielu czytelników na pewno zadaje sobie pytanie, jaki jest sens wprowadzenia koncepcji dziedziczenia do scope.

By na nie odpowiedzieć, posłużymy się opisaną w rozdziale 5. dyrektywą `ng-repeat`. Przytoczymy krótki opis tej dyrektywy: *Ng-repeat pozwala nam iterować po dowolnej kolekcji obiektów, dodatkowo tworzy osobne elementy szablonu DOM dla każdego z elementów kolekcji.*

Listing 2.3 najlepiej nam to zobrazuje.

Listing 2.3. Przykład zastosowania dyrektywy *ng-repeat*

```

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" ng-app="app">
<head>
  <title>Przykład zastosowania dyrektywy ng-repeat</title>
</head>
<body>
  <div ng-controller="defaultCtrl">
    <ul>
      <li ng-repeat="oferta in oferty">
        <p> Nazwa: {{ oferta.nazwa }} ||   cena: {{oferta.cena }} </p>
      </li>
    </ul>
  </div>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/
  ↪1.4.0-beta.5/angular.min.js">
</script>
<script>
  var app = angular.module('app', []);
  app.controller('defaultCtrl', function ($scope) {
    $scope.oferty = [
      { nazwa: 'Krzesło', cena: 149.99 },
      { nazwa: 'Stolik', cena: 189.99 },
      { nazwa: 'Szafka', cena: 205.99 },
    ];
  });
</script>
</body>
</html>
```

Zmienne z każdego obiektu w kolekcji zostaną przypisane do scope, by później zostać *zrenderowanymi* przez widok.

Właśnie w tym momencie pojawia się problem. Aby każdą nową zmienną przypisać do \$scope, musielibyśmy nadpisywać poprzednią ze względu na zbieżność nazw atrybutów. Dlatego też każdemu elementowi kolekcji przypisujemy nowy scope. Dana zmienna będzie „żyć” jedynie w obrębie swojego scope. Wszystkie nowo utworzone obiekty układają się w hierarchię przypominającą tę ze struktury DOM. Mamy możliwość wykorzystania tej samej nazwy dla zmiennej w różnych obiektach scope.

Podobnie jak w przypadku programowania zorientowanego obiektowo dziedziczenie pozwala na izolację atrybutów i funkcjonalności poszczególnych elementów modelu.

Dziedziczenie obiektów scope w Angularze odbywa się z użyciem wcześniej wspomnianej metody \$new().

```

var obiektBazowy = $rootScope;
var obiektPochodny = obiektBazowy.$new();
obiektBazowy.imie = 'Marian';
obiektPochodny.nazwisko = 'Kowalski';
```

W celu zniszczenia danego obiektu scope należy zastosować metodę \$destroy(), która usuwa wszystkie obiekty pochodne (i ich pochodne) z obiektu bazowego. Od tej chwili dany scope jest gotowy na „odśmiecanie”, czyli tzw. *garbage collection*.

Izolowany scope

Możliwe jest również utworzenie tzw. *izolowanego scope*, który nie dziedziczy po swoich rodzicach — jest to wcześniej przez nas wspomniany wyjątek. Używamy go podczas tworzenia komponentów, które chcielibyśmy później kilkakrotnie wykorzystać.

Tworząc izolowany scope, tak naprawdę bawimy się z pewnymi własnościami obiektu scope.

Wyobraźmy sobie sytuację, iż stworzyliśmy dyrektywę służącą np. do wyświetlania menu na stronie naszej restauracji. Nasza dyrektywa zawiera szablon dla wyświetlanych informacji. Podpinamy kontroler do modułu, przypisujemy potrawy do `$scope` i przypinamy dyrektywę. Gdybyśmy teraz umieścili kilka tagów z dyrektywą wewnątrz kodu HTML, to wyświetlana byłaby jedna i ta sama informacja. By temu zapobiec, musielibyśmy stworzyć osobny kontroler z nową instancją scope dla każdej potrawy. Pomysł czasochłonny i zmuszający do pisania masy nowego kodu, nie jest to więc najlepsze rozwiązanie. Tutaj właśnie wkraczają izolowane obiekty scope.

Aby odizolować scope, musimy wewnątrz naszej dyrektywy umieścić element scope.

```
...
return {
    scope: {}
}
...
```

Od tej chwili poszczególne instancje dyrektywy będą izolować swój lokalny scope. Możemy wiązać różne elementy przypisane do scope.

\$digest(), \$apply() i \$watch()

Jak wcześniej wspominaliśmy, scope nie służy jedynie jako most dla danych. Do jego obowiązków należy między innymi nasłuchiwanie zmian zachodzących w modelu. W tym celu wykorzystujemy opisany w dalszej części tego rozdziału `$watch`. Scope posiada również umiejętność wprowadzania (propagacji) zmian w modelu, znajdujących się wewnątrz aplikacji bądź pochodzących spoza niej.

Nasłuchiwanie oraz \$watch()

Po przypisaniu `$watch` do wybranego elementu AngularJS zaczyna oczekiwać na ewentualne zmiany. W momencie ich zajścia wywoływana jest tzw. funkcja nasłuchująca (ang. *listener function*), która może reagować na te zmiany. Przyjrzyjmy się bliżej temu, jak wygląda nasłuchiwanie zmian przez kanciastego, ukazane na listingu 2.4.

Listing 2.4. *Nasłuchiwanie*

```

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" ng-app="app">
<head>
  <title>AngularJS - $watch</title>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.1/
    ↪css/bootstrap.min.css">
</head>
<body>
  <div ng-controller="defaultCtrl">
    <div class="well">Liczba: {{number}}</div>
    <div>
      <a class="btn btn-success" href="#" ng-click="add()"> + </a>
      <a class="btn btn-danger" href="#" ng-click="dec()"> - </a>
    </div>
  </div>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/
    ↪1.4.0-beta.5/angular.min.js">
  </script>
  <script>
    var app = angular.module('app', []);
    app.controller('defaultCtrl', function ($scope) {
      $scope.number = 1;
      $scope.$watch('number', function () {
        console.log('Liczba: ' + $scope.number);
      });
      $scope.add = function () {
        $scope.number++;
      };
      $scope.dec = function () {
        $scope.number--;
      };
    });
  </script>
</body>
</html>

```

Wcześniej powiedzieliśmy, że dzięki *live binding* każda zmiana zachodząca w kontekście Angulara jest przez niego wyłapywana. Naturalnie rodzi się więc pytanie, czy każdy element przypisany do \$scope otrzymuje od razu własny obiekt nasłuchujący. Odpowiedź brzmi: nie, gdyż nasłuchiwanie zmian na wszystkich elementach zajęłoby zbyt dużo czasu. Mamy do wyboru dwa sposoby zadeklarowania nasłuchiwanie wybranych elementów:

- ♦ Pierwszy z nich to... interpolacja. Kiedy Angular napotyka interpolację w widoku, to wie, że automatycznie musi stworzyć obiekt nasłuchujący (w tym wypadku *implicit watcher*) na dany element.

```

<div>
  {{ watchedElement }}
</div>

```

- ♦ Istnieje również możliwość tworzenia nasłuchiwanicy własnoręcznie. Struktura typowego obiektu nasłuchującego (ang. *explicit watcher*) prezentuje się mniej więcej tak:

```
$watch('watchedElement', function(newValue, oldValue) {  
    //functions body...  
});
```

Pierwszy parametr to nazwa elementu modelu będącego pod obserwacją. Drugi parametr to funkcja nasłuchująca reagująca na zachodzące zmiany — jej wywołanie następuje za każdym razem, gdy wartość obserwowanego elementu ulega zmianie. Porównanie odbywa się poprzez metodę `angular.equals()`; wykonywana jest również metoda `angular.copy()` w celu zapisania obecnej wartości elementu. Obydwa przypadki zawarte są w naszym poprzednim przykładzie.

Zapewne niejedna osoba zastanawiała się, w jaki sposób Angular dowiaduje się o tych zmianach zachodzących w modelu. Za ich nasłuchiwanie odpowiada cykl `$digest()`.

\$digest()

`$digest` rozpoczyna się jako efekt wywołania `$scope.digest()`. Jest to cykl ewaluacji kolejno wszystkich obiektów nasłuchujących występujących w danym scope oraz jego potomkach. Ponieważ zachodzące zmiany wywołują tzw. funkcje nasłuchujące, które mogą modyfikować dowolne elementy modelu (w tym te sprawdzone już wcześniej), `$digest()` powtarzany jest dopóty, dopóki owe wezwania nie ustaną. Nawet jeżeli podczas wykonywania cyklu nie zostanie wezwana żadna funkcja nasłuchująca, zostanie on powtórzony co najmniej raz w celu upewnienia się, iż nie zaszła żadna zmiana. Jeżeli zdarzy się tak, że cykl wpadnie w pętlę nieskończoną, wówczas po 10 iteracjach zostanie zwrócony błąd.

Wywołanie cyklu następuje automatycznie, np. dzięki dyrektywom `ng-model` czy `ng-click`. Bezpośrednio jednak wywoływany jest najpierw `$apply()`, który to później wywołuje `$digest()`.

Może zaistnieć sytuacja, w której trzeba będzie wywołać `$apply()` manualnie. Angular zbudowany jest tak, by wychwytywać zmiany zachodzące między widokiem a modelem automatycznie, ale dzieje się to wyłącznie w obrębie jego kontekstu. W sytuacji, gdy zmiana modelu odbywa się poza kontekstem Angulara, należy go o niej poinformować, wywołując `$apply()` manualnie — to stąd Angular wie, że musi rozpocząć nasłuchiwanie. Nie powinniśmy nigdy bezpośrednio wywoływać `$digest()`. Prawidłowo powinniśmy wywołać `$apply()`, który później wykona cykl `$digest()`.

\$apply()

Usługa `$apply` zachowuje się jak gońiec wysyłany spoza kontekstu Angulara w celu poinformowania o zaistniałych zmianach. Innymi słowy, `$apply()` służy do integracji Angulara z innymi frameworkami bądź bibliotekami. `$apply()` zawiera funkcję pobieraną jako parametr, za której wykonanie odpowiada `$eval`. Do jego zadań należy również sprawdzenie, czy owa funkcja jest wykonywalna, oraz ewentualne poinformowanie Angulara o wykrytych nieścisłościach poprzez zwrócenie wyjątku. Wykorzystywana jest tu tzw. obsługa wyjątków z poziomu aplikacji (ang. *Application level error handling*). Jej wartość najczęściej doceniana jest wraz ze wzrostem poziomu skomplikowania aplikacji.

Następnie wywoływany jest cykl `$digest()`. Gdy mówimy o integracji z Angularem, mamy na myśli właśnie tę usługę: wystarczy otoczyć kod wewnątrz `$apply()` — prawda, że proste?

Wiesz już, jak działa `$apply()`, przejdźmy teraz do przykładu, który pokaże Ci jego zastosowanie praktyczne. Na pytanie, co stanie się w momencie uruchomienia poniższej strony, najlepiej odpowie listing 2.5:

Listing 2.5. *\$watch*

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" ng-app="app">
<head>
  <title>AngularJS - $watch</title>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.1/
    ↪css/bootstrap.min.css">
</head>
<body>
  <div ng-controller="defaultCtrl">
    <div class="well">Wiadomość: {{msg}}</div>
  </div>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/
    ↪1.4.0-beta.5/angular.min.js">
  </script>
  <script>
    var app = angular.module('app', []);
    app.controller('defaultCtrl', function ($scope) {
      $scope.go = function () {
        setTimeout(function () {
          $scope.msg = 'Wow, jestem opóźnioną informacją!';
          console.log('message:' + $scope.msg);
        }, 2000);
      }
      $scope.go();
    });
  </script>
</body>
</html>
```

Wynik wywołania powyższej strony będzie nie do końca zgodny z naszymi oczekiwaniami. Naszym celem było uaktualnienie w widoku `{{msg}}` po dwóch sekundach. Tak się jednak nie stało, mimo że teoretycznie program zadziałał i po dwóch sekundach w logu otrzymaliśmy oczekiwany tekst. Dlaczego widok nie został uaktualniony? Jak rozwiązać ten problem? Do tego posłuży nam `$apply()`. Przeanalizujmy teraz listing 2.6.

Listing 2.6. *\$apply()*

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" ng-app="app">
<head>
  <title>AngularJS - $apply()</title>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/
    ↪3.3.1/css/bootstrap.min.css">
</head>
<body>
```

```
<div ng-controller="defaultCtrl">
  <div class="well">Wiadomość: {{msg}}</div>
</div>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/
↳1.4.0-beta.5 /angular.min.js">
</script>
<script>
  var app = angular.module('app', []);
  app.controller('defaultCtrl', function ($scope) {
    $scope.go = function () {
      setTimeout(function () {
        $scope.msg = 'Wow, jestem opóźnioną informacją!';
        console.log('message:' + $scope.msg);
        $scope.$apply();
      }, 2000);
    }
    $scope.go();
  });
</script>
</body>
</html>
```

Jak widać, dodaliśmy tylko `$scope.$apply()`, zmieniło to jednak zasadniczo działanie całej aplikacji. Tym razem otrzymaliśmy odpowiedni log oraz zmianę z dwusekundowym opóźnieniem po stronie widoku.

Najważniejsze przesłanie płynące z tej części rozdziału jest takie: wszędzie tam, gdzie AngularJS nie może wykryć zmian samodzielnie, musimy to zrobić ręcznie.

Quiz

1. Co to jest `$scope`?
2. Czym się różni `$scope` od `$rootScope`?
3. Co to jest drzewo DOM?
4. Jak stworzyć izolowany `scope`?
5. Co to są obiekty nasłuchujące?
6. Jak działa cykl `$digest`?
7. W jakich sytuacjach należy korzystać z usługi `$apply`?

Skorowidz

\$apply(), 24, 25
\$digest(), 24
\$inject, 35
\$q, 173
\$q.all, 176
\$routeProvider, 134
\$scope, 105
\$watch(), 22, 25
@keyframes, 158

A

AJAX, 169
akcja animacji, 162
animacja
 dyrektywy ngRepeat, 165
 pomiędzy stronami, 158–162
animacje, 153
 CSS3, 153, 158
 JavaScript, 153, 161
API angular.module(), 28
aplikacje
 SPA, 7, 134
 zmodularyzowane, 29

B

biblioteka
 angular.js, 9
 Bootstrap, 17, 135
 jQuery, 135

C

callback, 169, 172
callback hell, 172
camelCase, 48
CDN, Content Delivery Network, 10
CORS, Cross Origin Resource
 Sharing, 179
CSS, 181
CSS3 Transitions, 153, 155
cykl
 \$digest, 25, 46
 ewaluacji, 24

D

data, 18
definicja szablonu, 109
Dependency Injection Engine, 34
DI, Dependency Injection, 33
dobre praktyki
 nazewnictwo plików, 185
 organizacja kodu, 188
 podział plików, 185
 wydajność, 189
dodawanie
 biblioteki, 9
 filtrów, 110
 konfiguracji, 42
dokumentacja, 7
DOM, Document Object Model, 10
dopasowania, 197
dopasowanie
 toBe, 198
 toBeCloseTo, 199
 toBeDefined, 199
 toBeFalsy, 199
 toBeGreaterThan, 200
 toBeLessThan, 200
 toBeNaN, 200
 toBeNull, 200
 toBeTruthy, 199
 toHaveBeenCalled, 201
 toHaveBeenCalledWith, 201
 toMatch, 202
 toThrow, 202
dyrektywa, 45
 ngBlur, 57
 ngEnd, 70
 a, 51
 form, 51
 input, 53
 ng-app, 10
 ngBind, 54
 ngBindHtml, 54
 ngBindTemplate, 55

ngChange, 57
ngClass, 62
ngClick, 72
ngCloak, 56
ng-controller, 12, 13, 20
ngController, 74
ngCopy, 75
ngCut, 76
ngDbclick, 78
ngFocus, 57, 78
ngForm, 79
ngHref, 79
ngIf, 80
ngInclude, 80
ngKeydown, 80
ngKeypress, 80
ngKeyUp, 80
ngList, 81
ng-model, 13
ngModel, 81
ngModelOptions, 82
ngMouseDown, 84
ngMouseenter, 84
ngMouseleave, 84
ngMousemove, 84
ngMouseover, 84
ngMouseup, 84
ngNonBindable, 84
ngPaste, 85
ngPluralize, 85
ngReadonly, 88
ng-repeat, 21, 113
ngRepeat, 65, 167
ngStart, 70
ngStyle, 88
ngSubmit, 88
ngSwitch, 89
ngTransclude, 89
ngValue, 91
ngView, 146
script, 91
select, 93
textarea, 96

dyrektywy
wbudowane, 50
własne, 99
działanie
Jasmine, 195
serwisu, 39
dziedziczenie, 19

E

Explicit Annotation, 35
explicit watcher, 23

F

fabryka, 38
fabryka mountainsList, 150
filtr, 109
filter, 115
limitTo, 115
linky, 117
orderBy, 113
rangeTime, 137, 141
filtry wbudowane, 110
dyrektywy ng-repeat, 113
JSON, 113
liczbowe, 111
operacje na datach, 112
operacje na stringach, 110
format JSON, 178
formularz, 181
formularz kontaktowy, 183
framework
Jasmine, 193
SPA, 13
funkcja, 119
\$get(), 40
addConfig, 42
all, 176
angular.bind, 119
angular.bootstrap, 120
angular.copy, 120
angular.element, 122
angular.equals, 126
angular.extend, 126
angular.forEach, 127
angular.fromJson, 127
angular.identity, 127
angular.injector, 129
angular.isArray, 131
angular.isDate, 131
angular.isDefined, 131
angular.isElement, 131
angular.isFunction, 131
angular.isNumber, 131
angular.isObject, 131
angular.isString, 131
angular.isUndefined, 131
angular.lowercase, 131

angular.module, 132
angular.reloadWithDebugInfo, 132
angular.toJson, 127
angular.uppercase, 131
compile, 45
FirstCtrl, 13
firstTest, 196
getClass, 145
module.config(), 42
myModule.animation(), 154
nasłuchująca, listener
function, 22
when, 134

G

garbage collection, 21
global namespace, 30
globalna przestrzeń nazw, 29

H

hard coding, 33

I

Implicit Annotation, 35
Implicit DI, 35
implicit watcher, 23
izolowany scope, 22

J

Jasmine, 193
JSON, JavaScript Object
Notation, 113, 170
JSONP, JSON with padding, 169, 179

K

kalendarz, 135, 136
klasy CSS, 181
kompilator HTML, 48
komunikacja z serwerem, 169
komunikat o błędzie, 196
konfiguracja, 134, 151
\$route, 134
modułu, 41
konstruktory dyrektyw, 101
kontroler, 15, 28
bazowy, 20
controller.js, 12
dateCtrl, 18
dziedziczenie, 19
potomny, 20
someCtrl, 28
kontrolki, 181

L, Ł

lista, 149
lista rozwijana, 140
live binding, 23
logika aplikacji, 13, 15
ładowanie początkowe aplikacji, 17
łączenie modułów, 30

M

metoda

\$animate.cancel(), 155
\$destroy(), 21
\$new(), 19, 21
angular.copy(), 24
angular.equals(), 24
angular.module(), 28
error(), 172
DELETE, 177
GET, 176
HEAD, 177
JSONP, 177
PATCH, 177
POST, 177
promise.finally(), 173
PUT, 177
reject(), 173
resolve(), 173
success(), 172

metody

\$http, 177
wstrzykiwania zależności, 35
wywołań dyrektyw, 49

minifikacja, 35

model, 15

model aplikacji, 13

moduł, 8, 27

app, 43

ngAnimate, 153

MVC, Model-View-Controller, 15

N

nasłuchiwanie, 22
nazewnictwo dyrektyw, 48
notacja camelCase, 188

O

obiekt

\$inject, 35, 36
\$rootScope, 17
\$scope, 12
\$\$scope, 17
Factory, 38
konfiguracyjny, 177

obiekt
 promise, 173
 Provider, 40
 Value, 37

obiekty
 deferred, 174, 175
 nasłuchujące, 23

obietnice, promises, 154, 169, 173

obsługa
 animacji, 154
 błędów, 181
 wyjątków, 24

odpowiedzi, 176

odpowiedzi http, 172

odroczenia, deferreds, 173

odśmieszanie, 21

odwołanie do kontrolera, 12

operacje
 na datach, 112
 na stringach, 110

organizacja kodu, 188

P

parametry metody \$http, 177

pierwsza aplikacja, 11

plik, 185

- app.mdl.js, 143
- app.rout.js, 143
- categories-data.js, 138
- controller.js, 13, 28, 29
- data.json, 170, 171
- default.html, 139
- directives/ng-date-picker.js, 136
- edit.tpl.html, 139
- edit-ctrl.js, 139
- filters.js, 137
- filters/filters.js, 137
- firstTest.js, 203
- firstTestSpec.js, 196, 203
- index.html, 30, 145, 203
- index-ctrl.js, 145
- json.tpl.html, 141
- list.tpl.html, 142
- list-ctrl.js, 142
- ng-date-picker.js6, 136
- secondTest.js, 198
- secondTestSpec.js, 198
- style.css, 135
- todos-data.js, 138

pliki JSON, 58

pobieranie
 AngularJS, 8
 Jasmine, 194

podpinanie kontrolera, 11

podwójne wiązanie, 14

prezentacja danych, 16

proces konfiguracji, 151

propagacja, 22

przechowywanie odpowiedzi, 176

przestrzeń nazw, 29

przesyłanie zdarzeń, event
 dispatching, 20

przypisanie atrybutów, 18

R

reakcje łańcuchowe obietnic, 175

reguła @keyframes, 158

reprezentacja problemu, 15

routing, 133

S

same origin policy, 179

scope, 105

serwer proxy, 180

serwis, 39

serwis \$animate, 154

snake-case, 48

SPA, Single Page Applications, 7, 13, 133

specyfikacja, 197

string, 110

struktura
 aplikacji, 187
 katalogów, 135

system ocen, 102

szablon default.tpl.html, 144

szkielet
 animacji, 161
 aplikacji, 12, 29

T

test, 193

test zakończony
 niepowodzeniem, 196
 powodzeniem, 196

testowanie JavaScriptu, 193

tworzenie
 animacji, 153
 CSS, 153
 CSS3 Transitions, 153
 JavaScript, 153
 dyrektyw, 101
 fabryk, 40
 konfiguracji, 134
 stałych, 42

typy obiektów, 37

U

ukrywanie elementów, 166

usługa
 \$apply, 24
 \$http, 169

uzyskiwanie zależności, 34

W

wartość
 NaN, 200
 null, 200

wiązanie
 dwustronne, 14
 jednostronne, 14

widok, 16

właściwości
 dyrektyw, 101
 modułu, 43
 ngFormController, 182

właściwość dateOriginal, 18

wstrzykiwanie
 modułu animacji, 153
 stałej, 42
 Value do Factory, 38
 Value do Providera, 41
 zależności, DI, 33

wydajność, 189

wyjątek, 24

wrażenia regularne, 202

wywołanie \$http, 170

X

XHR, XMLHttpRequest, 169

Z

zaciemnianie kodu, 35

zagnieżdżenie zapytań, 172

zapytanie XHR, 169

zarządzanie
 asynchronicznymi
 operacjami, 172
 zależnościami, 33

zastosowanie
 dyrektywy ng-repeat, 21
 dyrektywy select, 146
 explicit dependency injection, 36
 implicit dependency injection, 35
 obiektu \$inject, 36
 Value, 37

zestaw specyfikacji, 197

zmodularyzowana aplikacja, 29

znak
 |, 109
 dwukropka, 150

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

ANGULARJS jest frameworkiem MVC napisanym w JavaScriptcie. Ta stworzona przez Google'a technologia daje programistom ogromne możliwości, a w ciągu ostatnich dwóch lat dla wielu firm stała się standardem w tworzeniu aplikacji sieciowych. O jej popularności najlepiej świadczą coraz liczniejsze oferty pracy dla programistów AngularJS. Każdy, kto poważnie myśli o tworzeniu aplikacji dla sieci, powinien poznać cechy tego frameworka, jego zalety oraz ograniczenia.

Jeśli i Ty chciałbyś rozpocząć przygodę z AngularJS i w pełni korzystać z opcji oferowanych przez JavaScript, sięgnij po tę książkę! Dowiesz się z niej, jak szybko i sprawnie tworzyć dynamiczne, łatwe w utrzymaniu aplikacje internetowe działające po stronie klienta. Poznasz praktyczne przykłady, które pomogą Ci zrozumieć prezentowany materiał, a także nauczysz się samodzielnie pisać wydajny kod z wykorzystaniem AngularJS, zaś dzięki podsumowującym każdy rozdział pytaniom kontrolnym skutecznie utrwalisz zdobyte wiadomości.

- Podstawy AngularJS i środowisko pracy dewelopera
- Wzorzec MVC w aplikacjach internetowych SPA
- Modularyzacja i wstrzykiwanie zależności
- Dyrektywy, filtry i funkcje
- Sposoby komunikacji z serwerem
- Dobre praktyki w stosowaniu AngularJS
- Testowanie aplikacji internetowych

Nauucz się obsługi nowego frameworka!

Dariusz Kalbarczyk — programista od ponad 15 lat związany z rynkiem IT. Współtwórca grupy AngularJS Warsaw, przez lata współpracujący z największymi firmami na rynku finansowym i telekomunikacyjnym (brał udział m.in. w tworzeniu opartego na najnowszych technologiach bankowego mobilnego systemu transakcyjnego).

Arkadiusz Kalbarczyk — student informatyki w Polsko-Japońskiej Akademii Technik Komputerowych. Swoje umiejętności miał okazję rozwijać w międzynarodowym środowisku: Kuwejcie oraz Irlandii. Współtwórca grupy AngularJS Warsaw.



30213 numer katalogowy

księgarnia internetowa



<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Sprawdź najnowsze promocje:
① <http://helion.pl/promocje>
Książki najchętniej czytane:
② <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
③ <http://helion.pl/nowosci>

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

sięgnij po WIĘCEJ



KOD KORZYŚCI

ISBN 978-83-283-0586-1



9 788328 305861

Informatyka w najlepszym wydaniu

cena: 39,90 zł