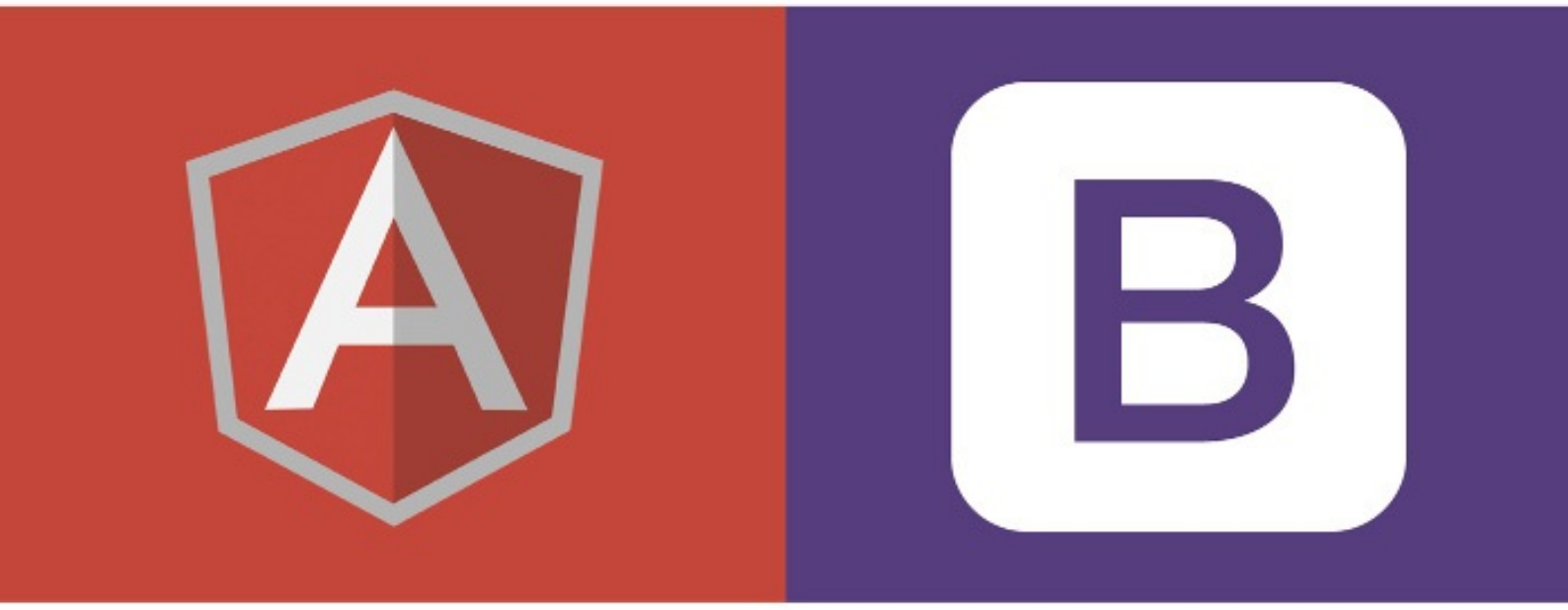
Stephen Radford

Sviluppare applicazioni web con AngularJS e Bootstrap

UTILIZZARE
pattern MVC
con AngularJS

CREARE
interfacce responsive
con Bootstrap



"Il genio è la capacità di ridurre la complessità in semplicità."
– C. W. Ceram

APOGEO

SVILUPPARE APPLICAZIONI WEB CON ANGULARJS E BOOTSTRAP

Stephen Radford

ISBN edizione cartacea: 9788850333424

Durante la mia carriera ho sviluppato progetti di diverse dimensioni, da piccoli siti commerciali a interi social network. Erano tutti accomunati dalla necessità di disporre di JavaScript e CSS ben strutturati.

Questo libro affronta due fantastici progetti open source che derivano da questa esigenza: Bootstrap e AngularJS.

Gli argomenti del libro

Il Capitolo 1, “Hello, {{name}}”, esamina i fondamenti di AngularJS e Bootstrap attraverso la realizzazione di una semplice app Hello, World.

Il Capitolo 2, “Sviluppare con AngularJS e Bootstrap”, presenta l’app principale che svilupperemo nel corso del libro, il sistema delle griglie di Bootstrap e alcuni componenti di AngularJS.

Il Capitolo 3, “I filtri”, illustra alcuni dei filtri integrati in AngularJS e la creazione di un filtro.

Il Capitolo 4, “Routing”, si basa sul router integrato di AngularJS, e illustra l’utilizzo dei partial per creare un’app multiview.

Il Capitolo 5, “Le viste”, esamina il sistema delle griglie di Bootstrap e approfondisce i partial.

Il Capitolo 6, “CRUD”, spiega come, dopo aver impostato le viste, possiamo implementare, creare, leggere, aggiornare e cancellare le funzioni.

Il Capitolo 7, “AngularStrap”, descrive il modulo di terze parti che consente di utilizzare i plug-in di Bootstrap tramite AngularJS.

Il Capitolo 8, “Connessione al server”, esamina due sistemi ufficiali per collegarsi a un server.

Il Capitolo 9, “I task runner”, spiega come minificare tutti i file JS e Less utilizzando Grunt e gulp.

Il Capitolo 10, “Personalizzare Bootstrap”, mostra come personalizzare facilmente Bootstrap dopo aver impostato Grunt.js.

Il Capitolo 11, “Validazione”, affronta gli strumenti di validazione subito pronti all’uso; li implementeremo e gestiremo gli errori del server.

Il Capitolo 12, “Strumenti della community”, presenta alcuni strumenti creati dalla community di AngularJS.

L’Appendice A, “Persone e progetti”, presenta alcune persone importanti nell’ambiente di AngularJS e Bootstrap, oltre a progetti rilevanti.

L’Appendice B, “In caso di dubbio”, offre risposte agli eventuali dubbi dei lettori.

L’Appendice C, “Risposte ai quiz”, comprende tutte le risposte alle domande dei quiz presenti nel libro.

Che cosa occorre per il libro

AngularJS e Bootstrap non hanno dipendenze e per questo libro dovrete solo disporre di un browser e di un editor di testo. Vi consiglio Chrome e Atom.

A chi si rivolge il libro

Se vi interessa lo sviluppo web moderno, sicuramente conoscerete Bootstrap e AngularJS. Questo libro si rivolge a lettori con un minimo di esperienza in JavaScript che desiderano impegnarsi nello sviluppo di web app.

Tuttavia è assolutamente necessaria la conoscenza di JavaScript. Se non conoscete la differenza tra una stringa e un oggetto, correte ai ripari. Ovviamente, se avete già utilizzato AngularJS o Bootstrap e volete saperne di più, vi sentirete a vostro agio.

Convenzioni

In questo libro troverete alcuni stili di carattere che differenziano diversi tipi di informazioni. Di seguito alcuni esempi e una spiegazione del loro significato.

Il codice che troverete nel testo, i nomi delle tabelle del database, i nomi e le estensioni dei file, i percorsi, gli URL, l'input degli utenti e gli handle di Twitter vengono presentati in `monospaziato`.

Un frammento di codice è formattato nel seguente modo:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title></title>
</head>
<body>

</body>
</html>
```

Gli input e output da riga di comando si presentano nel modo seguente:

```
open -a 'Google Chrome' --args -allow-file-access-from-files
```

Termini nuovi, parole importanti, cartelle o directory ed elementi dell'interfaccia sono riportati in *corsivo*.

> **NOTA**
>
> I suggerimenti, gli avvertimenti e le note importanti appaiono in questo modo.

Scarica i file degli esempi

Sul sito dell'editore originale inglese, Packt Publishing, potete scaricare i file degli esempi presentati del testo. Per farlo è necessario registrarsi gratuitamente all'indirizzo https://www.packtpub.com/register. Quindi andate sulla scheda del libro all'indirizzo https://www.packtpub.com/web-development/learning-web-development-bootstrap-and-angularjs (per comodità in forma abbraviata http://bit.ly/packt-ab) e fate clic su *Code Files*. Per problemi di download potete contattare la nostra redazione all'indirizzo libri@apogeonline.com.

Stephen Radford è uno sviluppatore web a tutto tondo nativo di Bristol, che vive nel centro di Leicester, in Gran Bretagna. Dopo aver frequentato all'università il corso di Grafica e Comunicazione Visiva, si è trasferito in questa città, dove è stato assunto da una delle più importanti società di marketing online del paese.

Mentre lavorava per alcune agenzie, Stephen ha sviluppato diversi progetti collaterali, tra cui FTPloy, un SaaS concepito per rendere accessibile a tutti il deployment continuo. Questo progetto si è classificato tra i finalisti nella categoria "Side Project of the Year" dei .Net Awards.

Attualmente, insieme con il suo socio, gestisce Cocoon, una società di sviluppo web che realizza e gestisce app come FTPloy e Former. Cocoon lavora anche a stretto contatto con un gruppo di startup e aziende che trasformano idee in siti e app.

Vorrei ringraziare quanti mi hanno sostenuto durante la stesura di questo libro. Prima di tutto, il mio partner, Declan. Mi è stato di grande supporto e non potrei desiderare di avere accanto nessuna persona migliore di lui. Paul Mckay è stato il primo al quale ho mostrato il libro e mi ha anche aiutato con il mio profilo biografico perché, stranamente, ho incontrato grandi difficoltà a scrivere sulla mia vita. Ovviamente, voglio ringraziare i miei genitori. Mio padre ha atteso pazientemente la copia cartacea del libro; spero che ora sia in bella mostra in soggiorno.

Tasos Bekos, ingegnere informatico, utilizza le tecnologie web da più di dieci anni. Ha lavorato come sviluppatore freelance e consulente per alcune delle più importanti società internazionali finanziarie e di telecomunicazioni. Attualmente lavora come sviluppatore per ZuluTrade, dove si avvale delle più moderne tecnologie front-end. Conosce a fondo AngularJS ed è un membro attivo della community open source, nella quale opera come principale collaboratore al progetto AngularUI Bootstrap. Quando non scrive codice, trascorre il tempo a giocare con i suoi due figli.

Jack Hsu è uno sviluppatore web specializzato negli strumenti e nelle tecnologie front-end. È il principale sviluppatore front-end presso Nulogy, dove mette la sua conoscenza di JavaScript e AngularJS al servizio dei colleghi. Prima di Nulogy, ha lavorato per numerose aziende, tra cui The Globe & Mail, Ontario Institute of Cancer Research e Wave Accounting. Nel tempo libero gioca ai videogiochi, esplora i diversi quartieri di Toronto o viaggia per il mondo. Sul suo blog personale è possibile leggere numerosi post riguardanti la programmazione.

Ole B. Michelsen lavora da più di 12 anni nello sviluppo web e si è laureato in informatica presso la DIKU, all'università di Copenhagen. Di recente si è specializzato nello sviluppo JavaScript front-end, concentrandosi in particolare su WebRTC e su framework per app single page.

Jurgen Van de Moere è nato nel 1978 ed è cresciuto a Evergem, in Belgio, con i genitori, la sorella e i suoi animali domestici. A 6 anni ha iniziato ad aiutare il padre, proprietario di un negozio di informatica, ad assemblare i computer per i clienti. Mentre gli amici giocavano ai videogiochi, Jurgen preferiva scrivere script e programmi per risolvere i problemi che erano costretti ad affrontare i clienti del padre. Dopo essersi laureato in latino e matematica presso il college Sint-Lievens a Gand, Jurgen ha proseguito la sua formazione presso l'università della stessa città, dove ha frequentato informatica. All'università il suo nome utente Unix era "jvandemo," il nickname che usa ancora oggi su Internet. Nel 1999 ha iniziato la carriera professionale presso Infoworld. Dopo anni di duro lavoro come sviluppatore e network engineer, nel 2005 e nel 2006 ha ricoperto posizioni manageriali. Sentendosi uno sviluppatore, e avvertendo la mancanza di scrivere codice, nel 2007 ha deciso di porre fine alla carriera di manager per dedicarsi di nuovo alla sua vera passione: lo sviluppo. Da allora ha studiato e lavorato da casa, in Belgio, dove vive

attualmente con la sua fidanzata, suo figlio e i suoi cani. In un mondo in rapida evoluzione di applicazioni data-intensive e real-time, si concentra sulle tecnologie legate a JavaScript e basate su AngularJS e Node.js. I suoi numerosi contributi pubblici e privati hanno permesso di avviare molteplici progetti di successo in tutto il mondo. Se vi serve una consulenza per il vostro progetto, potete scrivergli all'indirizzo `hire@jvandemo.com`. Potete anche seguirlo su Twitter (`@jvandemo`) o leggere il suo blog (`http://www.jvandemo.com`).

Hello, {{name}}

Il modo migliore per imparare a programmare è scrivere codice, ed è proprio questo ciò che faremo. Per comprendere quanto è facile essere subito operativi con Bootstrap e AngularJS, realizzeremo un’applicazione molto semplice che ci consentirà di digitare un nome e visualizzarlo sulla pagina in tempo reale. Scoprirete così l’efficacia del binding dei dati bidirezionale e il linguaggio per template integrato di Angular. Utilizzeremo Bootstrap per attribuire uno stile e una struttura all’app.

Prima di installare i framework, creeremo la struttura delle cartelle e il file `index.html` che sarà la base dell’app.

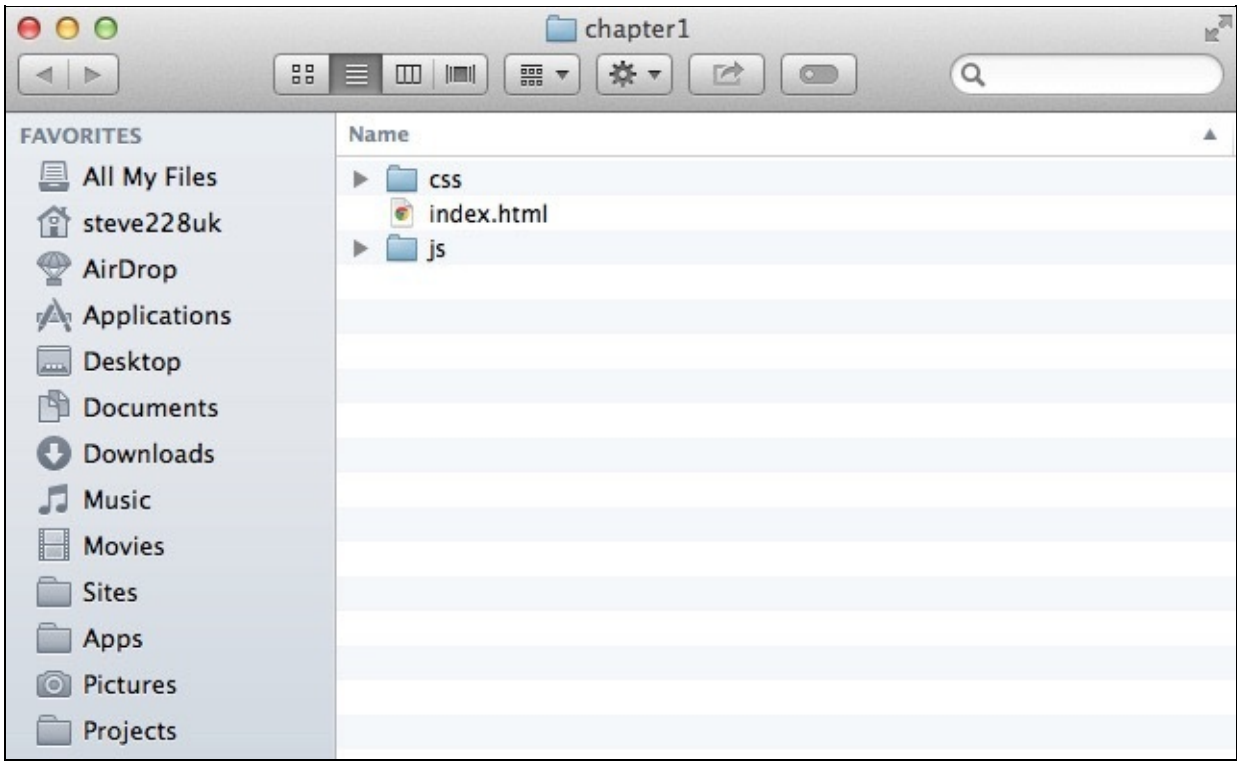
Impostazione

Per realizzare l'app con Angular e Bootstrap, procediamo all'impostazione, che consiste nel creare una pagina HTML e includere alcuni file. Innanzitutto create una nuova directory, *chapter1*, e apritela nel vostro editor. Create al suo interno un nuovo file, `index.html`, e immettete questo codice boilerplate:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title></title>
</head>
<body>

</body>
</html>
```

Si tratta di una pagina HTML standard con la quale opereremo dopo aver integrato Angular e Bootstrap.

Create due cartelle all'interno della cartella *chapter1*: *css* e *js*. La struttura completa delle cartelle dovrebbe essere simile alla seguente.



Installazione di AngularJS e Bootstrap

Installare questi framework è semplice come includere file CSS o JavaScript nella pagina. Lo potremmo fare con un *content delivery network* (CDN) come Google Code o MaxCDN, ma per ora recupereremo i file manualmente. Esaminiamo i passi che dovreste conoscere quando integrerete AngularJS e Bootstrap nel progetto.

Installazione di Bootstrap

Andate all'indirizzo `http://getbootstrap.com` e fate clic sul pulsante *Download Bootstrap*. Otterrete un file ZIP con l'ultima versione di Bootstrap che comprende CSS, font e file JavaScript. Le versioni precedenti includevano una directory delle immagini, che nella Versione 3 diventa *icon fonts*.

Per quest'app, ci interessa per ora soltanto un file: `bootstrap.min.css` presente nella directory *css*. Il foglio di stile fornisce tutta la struttura e quegli elementi graziosi, tra cui pulsanti e messaggi di avviso, per cui Bootstrap è conosciuto. Copiatelo nella directory *css* del progetto e aprite il file `index.html` nell'editor di testo.

Integrare Bootstrap è facile come collegare il file CSS che abbiamo appena copiato. Aggiungete quanto segue all'interno del tag `<head>`. Inserite questo tag nell'elemento `<head>` della pagina:

```
<link rel="stylesheet" href="css/bootstrap.min.css">
```

Installazione di AngularJS

Dopo aver integrato Bootstrap nella web app, procedete a installare Angular. Visitate il sito `https://angularjs.org/` e fate clic sul pulsante *Download*. Vedrete alcune opzioni; a voi serve la versione stabile minificata.

Copiate il file che avete scaricato nella directory *js* del progetto e aprite il file `index.html`. Angular può essere integrato nell'app come qualsiasi altro file JavaScript.

È preferibile includerlo nel tag `<head>` della pagina, altrimenti alcune funzioni a cui ricorrerete nel libro non saranno attive. Anche se non è necessario, dovrete compiere altri passi per integrare ancora di più Angular nel file HTML.

Inserite questo tag `<script>` nell' `<head>` della pagina.

```
<script src="js/angular.min.js"></script>
```

Tutto fatto? Quasi. Dobbiamo dire ad Angular che intendiamo utilizzarlo nell'app. Angular richiede questo bootstrap, e il framework semplifica moltissimo questa procedura. Dovete semplicemente includere un altro attributo nel tag `<html>` di apertura:

```
<html lang="en" ng-app>
```

Ecco fatto! Ora Angular sa che vogliamo utilizzarlo.

NOTA

Angular ci consente anche di far precedere questi attributi da `data-` (per esempio `data-ng-app`) nel caso volessimo scrivere HTML5 valido.

Utilizzare AngularJS

Abbiamo visto molta teoria alla base di Angular; è giunto il momento di metterla in pratica. Dopo aver creato un'app funzionante, vedremo come abbellirla con Bootstrap.

Riaprite il file `index.html`, ma questa volta apritelo anche nel browser così da vedere ciò su cui state lavorando. Ecco a che punto siamo arrivati:

```
<html lang="en" ng-app>
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="css/bootstrap.min.css">
  <title></title>
  <script type="text/javascript" src="js/angular.min.js"></script>
</head>
<body>

</body>
</html>
```

Abbiamo impostato Bootstrap e Angular e abbiamo inizializzato l'app con l'attributo `ng-app` nel tag di apertura `<html>`; ora passiamo rapidamente all'azione.

Realizzeremo un'app Hello, World un po' diversa. Invece di far comparire questa scritta, avremo un campo di testo che effettuerà il binding dei dati e li ripeterà automaticamente nella vista; tutto questo senza scrivere una sola riga di JavaScript.

Iniziamo ad aggiungere il tag `<h1>` nel tag `<body>`:

```
<h1>Hello, World</h1>
```

Nel browser vedrete che Bootstrap ha aggiornato la visualizzazione predefinita. Al posto del font Times New Roman compare l'Helvetica e i margini di troppo attorno al bordo sono stati eliminati.

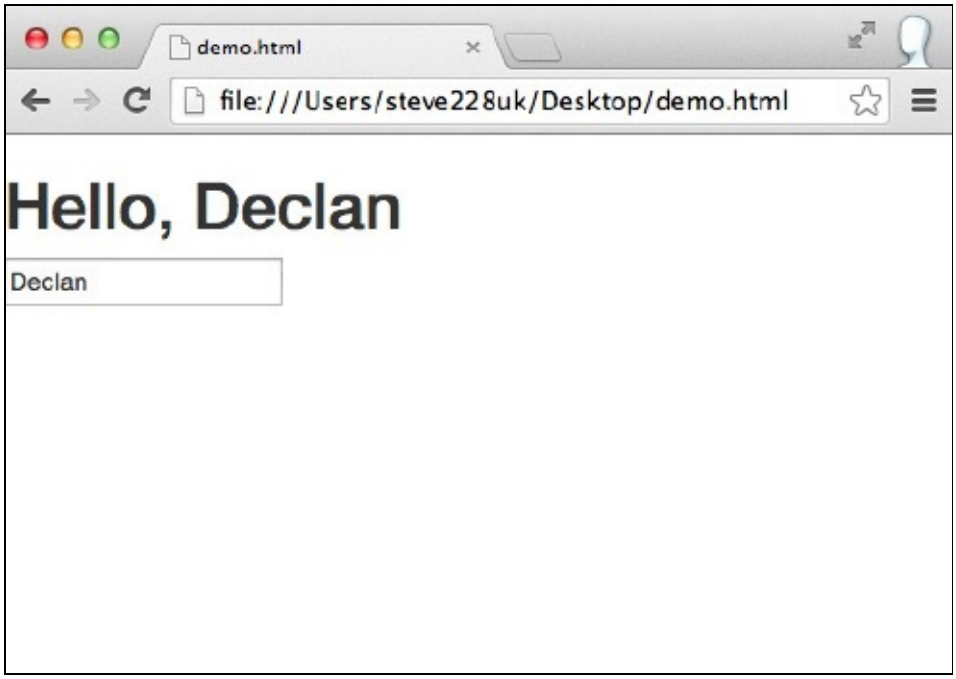
Ora dobbiamo includere l'input di testo e specificare anche il modello che intendiamo utilizzare. Il modello può essere di qualsiasi tipo, ma in questo caso sarà una stringa che l'input restituirà:

```
<input type="text" ng-model="name">
```

L'attributo `ng-model` dichiara il binding del modello su questo elemento, e tutto ciò che digiteremo nel campo di testo sarà associato a esso da Angular. Ovviamente non verrà visualizzato come per magia sulla pagina; dovremo dire al framework dove ripeterlo. Per visualizzare il modello sulla pagina, è sufficiente racchiudere il nome tra doppie parentesi graffe:

```
{{name}}
```

Inseritelo al posto di `World` nel tag `<h1>` e aggiornate la pagina nel browser. Se digitate il vostro nome nel campo di testo, vedrete che viene visualizzato automaticamente nell'header in tempo reale. Angular lo fa al posto vostro senza dover scrivere una sola riga di JavaScript.



Un risultato ottimo, ma sarebbe opportuno avere un'impostazione predefinita che eviti di far sembrare che non funziona ancora prima che un utente digiti il suo nome. Fortunatamente tutto ciò che è compreso tra le parentesi graffe viene analizzato come un'espressione AngularJS, e così è possibile verificare se il modello ha un valore; in caso contrario, può ripetere `World`. In Angular questa è un'espressione, ed è sufficiente aggiungere il doppio pipe come in JS:

```
{{name || 'World'}}
```

NOTA

Angular descrive in questo modo un'espressione: "Frammenti di codice simili a JavaScript che di solito sono posti in binding come `{{ espressione }}`."

È opportuno ricordare che si tratta di JavaScript, ed ecco perché dobbiamo inserire le virgolette per segnalare che si tratta di una stringa e non del nome di un modello. Se provaste a cancellarle, vedreste che Angular non visualizza di nuovo nulla. Ciò accade perché i modelli `name` e `World` non sono definiti.

Questi modelli possono essere definiti direttamente all'interno dell'HTML utilizzando, come abbiamo visto, un attributo, ma è anche possibile assegnare a essi un valore da un controller. A questo scopo create un nuovo file JS, `controller.js`, e includetelo nell'app:

```
<script type="text/javascript" src="js/controller.js"></script>
```

Inseritelo dopo aver incluso Angular nella pagina per evitare errori.

I controller sono semplicemente funzioni che Angular può utilizzare; esaminiamone uno:

```
function AppCtrl($scope){
}
```

Qui abbiamo dichiarato il controller (in sostanza una semplice funzione del costruttore JavaScript) e in esso abbiamo inserito lo scope. Lo *scope* è ciò a cui possiamo accedere all'interno della vista. Su un'unica pagina possono esistere molteplici controller e molteplici scope. Si tratta in sostanza di un oggetto JavaScript dei nostri modelli e delle nostre funzioni, sui quali Angular opera la sua magia; per esempio, lo scope della nostra applicazione per il momento appare così:

```
{
 name: "Stephen"
}
```

Lo scope cambia a seconda di ciò che si digita nel campo di testo. A esso si può accedere sia dalla vista sia dal controller.

Dopo aver creato il controller, dobbiamo dire ad Angular che intendiamo utilizzarlo. Per la nostra app ci serve un solo controller; aggiungiamo un secondo attributo al tag `<html>`:

```
ng-controller="AppCtrl"
```

Questo attributo dice ad Angular che vogliamo utilizzare la funzione `AppCtrl` che abbiamo appena creato come controller per la pagina. Potremmo ovviamente aggiungerlo a qualsiasi elemento sulla pagina compreso, se volessimo, il body.

Per verificare che tutto funzioni, specificheremo un valore iniziale per il modello. È facile come impostare una proprietà su qualsiasi oggetto:

```
function AppCtrl($scope) {
  $scope.name = "World";
}
```

Se aggiornate l'app nel browser, vedrete che `World` è precompilato come valore del modello. Questo è un ottimo esempio dell'efficace *binding dei dati bidirezionale* di Angular. Ci consente di utilizzare dati predefiniti di una API o di un database e poi modificarli direttamente nella vista prima di riottenerli nel controller.

NOTA

Angular descrive il binding dei dati come "la sincronizzazione dei dati tra i componenti del modello e della vista". Così, se modifichiamo il valore di un modello nella vista o nel controller JavaScript, tutto si aggiorna di conseguenza.

Bootstrap

Dopo aver creato l'applicazione Hello World e aver verificato che tutto funzioni come previsto, è il momento di utilizzare Bootstrap per aggiungere stile e struttura

alla nostra app. Per ora l'app è mal allineata sulla sinistra e tutto sembra troppo fitto; rimediamo con un po' di *scaffolding*. Bootstrap offre un ottimo sistema di griglie responsive *mobile first* che possiamo utilizzare aggiungendo alcuni div e alcune classi.

Prima, però, racchiudiamo il contenuto in un contenitore per far subito un po' di ordine.

NOTA

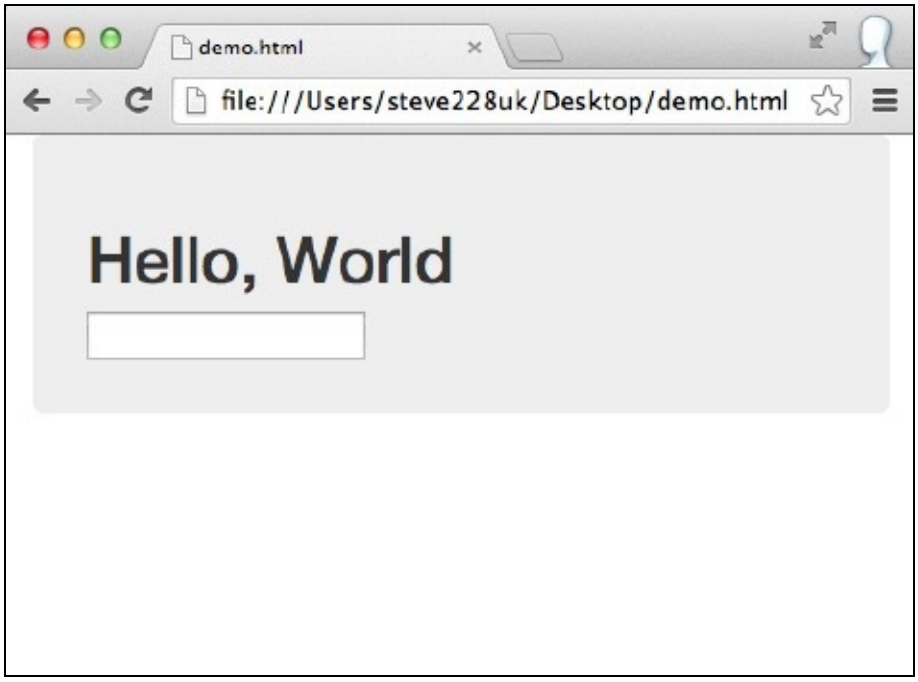
Il concetto di *mobile first* consiste nel progettare/sviluppare innanzitutto per gli schermi più piccoli e aggiungere elementi al design invece di eliminarli.

```
<div class="container">
  <h1>Hello, {{name || 'World'}}</h1>
  <input type="text" ng-model="name">
</div>
```

Se ridimensionate la finestra del browser, dovreste iniziare a osservare la capacità di adattamento del framework e vedere la finestra comprimersi.

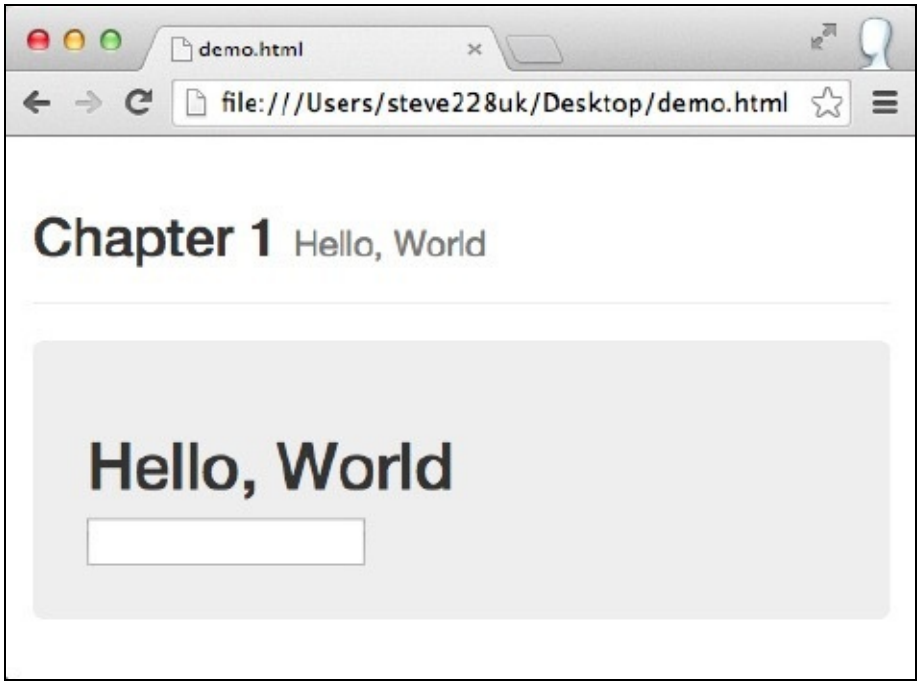
Può essere una buona idea racchiuderlo in quello che nella terminologia di Bootstrap è un *Jumbotron* (nelle versioni precedenti si chiamava *Hero Unit*). Farà risaltare molto di più il titolo. Possiamo ottenere questo risultato racchiudendo i tag `<h1>` e `<input>` in un nuovo div associato alla classe `jumbotron`:

```
<div class="container">
  <div class="jumbotron">
    <h1>Hello, {{name || 'World'}}</h1>
    <input type="text" ng-model="name">
</div>
</div>
```



Ha un aspetto decisamente migliore, ma il contenuto è ancora troppo vicino alla parte superiore della finestra del browser. Possiamo introdurre un ulteriore miglioramento con un header di pagina, anche se il campo di testo mi sembra ancora fuori posto. Sistemiamo innanzitutto l'header di pagina:

```
<div class="container">
  <div class="page-header">
    <h2>Chapter 1 <small>Hello, World</small></h2>
  </div>
  <div class="jumbotron">
    <h1>Hello, {{name || 'World'}}</h1>
    <input type="text" ng-model="name">
</div>
</div>
```

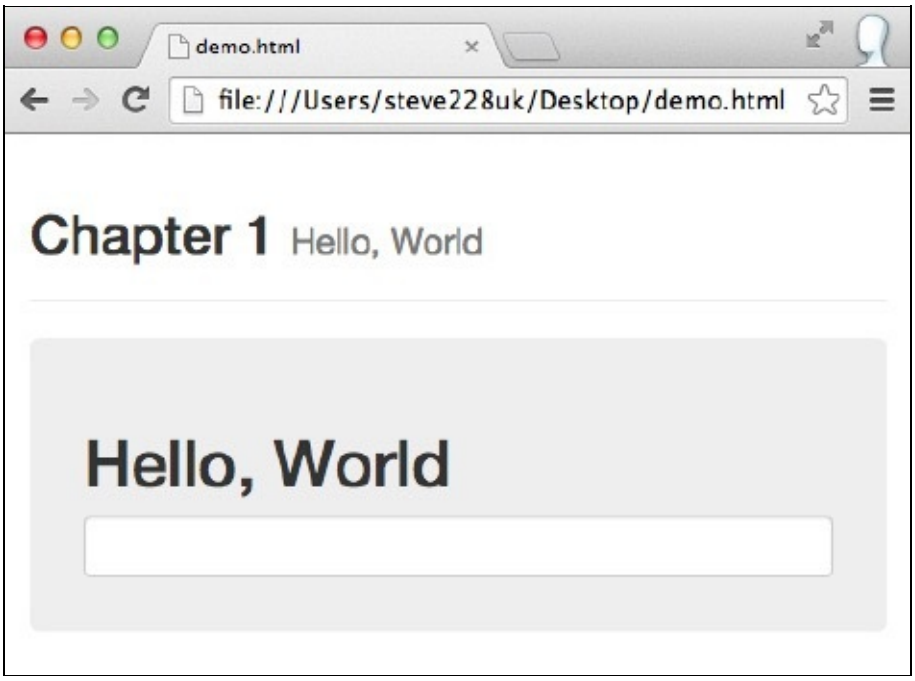


Ho inserito il numero e il titolo del capitolo. Il tag `<small>` all'interno del tag `<h2>` permette di distinguere efficacemente il numero dal titolo del capitolo.

La classe `page-header` aggiunge altro margine e padding, oltre a un sottile bordo inferiore. L'ultimo elemento che potremmo migliorare è la casella di testo. Bootstrap offre alcuni ottimi stili di testo e vale la pena utilizzarli. Per prima cosa dobbiamo aggiungere la classe `form-control` all'input di testo.

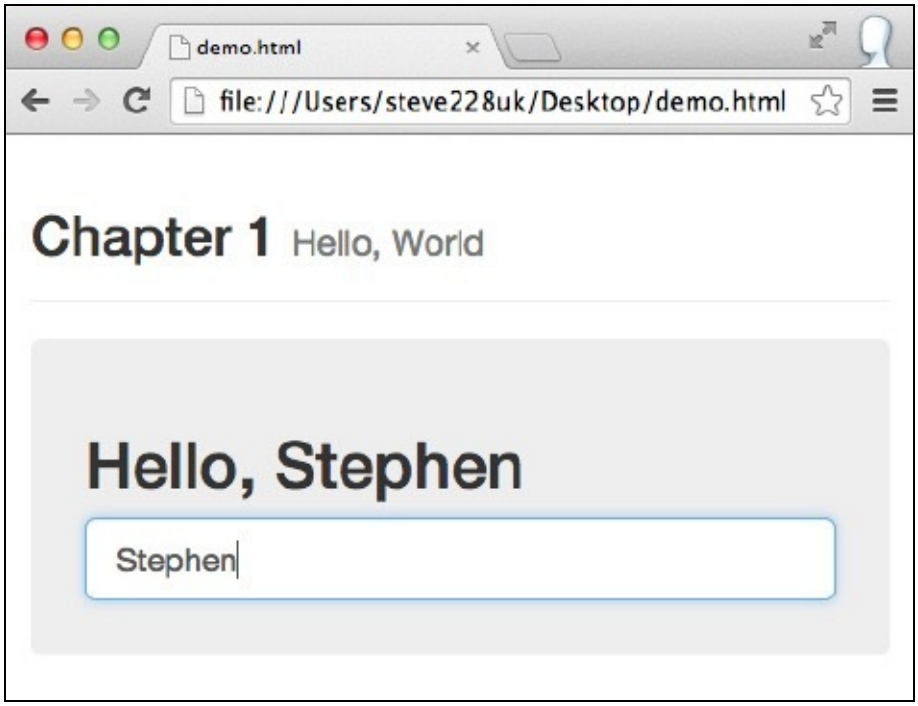
In questo modo la larghezza verrà impostata al 100% e alcuni aspetti stilistici miglioreranno, come i bordi arrotondati e un bagliore nel momento in cui l'elemento ottiene il focus:

```
<input type="text" ng-model="name" class="form-control">
```



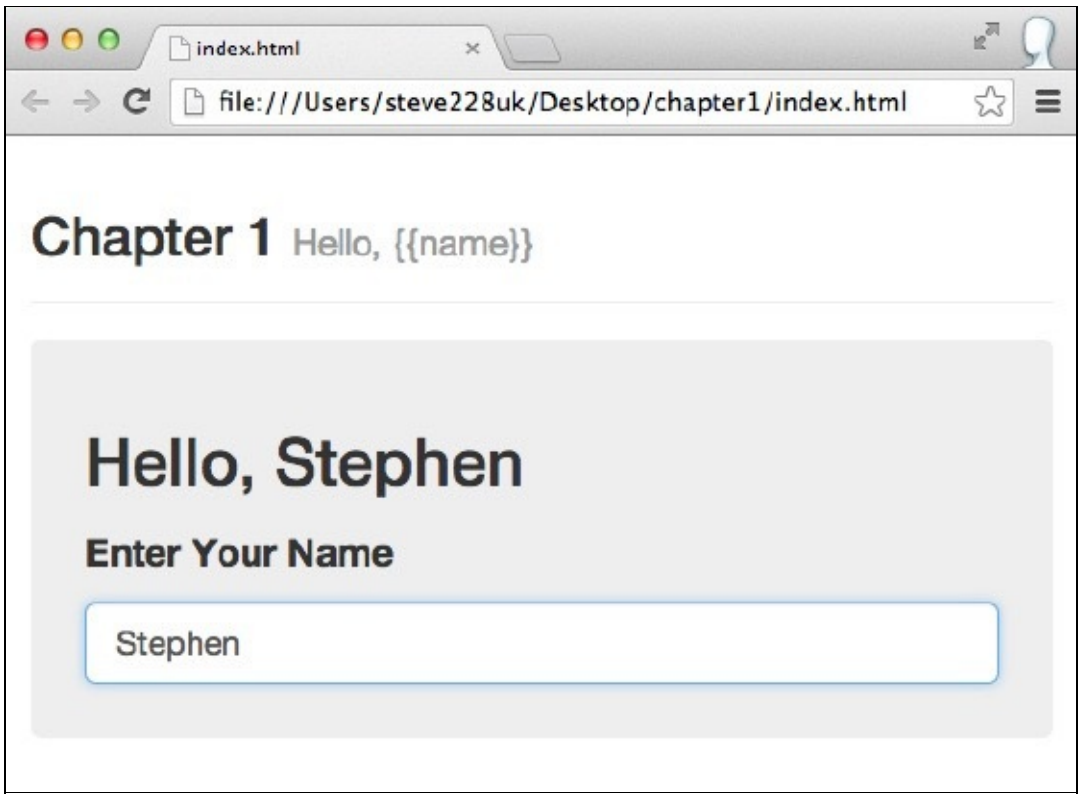
Va molto meglio, ma sembra ancora un po' piccolo rispetto all'header. Bootstrap offre altre due classi che rimpiccioliscono o ingrandiscono l'elemento, rispettivamente `input-lg` e `input-sm`. In questo caso, scegliamo la classe `input-lg` e la aggiungiamo all'input:

```
<input type="text" ng-model="name" class="form-control input-lg">
```



Dobbiamo ancora risolvere la questione della spaziatura perché è troppo a ridosso del tag `<h1>`. Forse è una buona idea aggiungere un'etichetta, così l'utente capirà che deve digitare nella casella. Bootstrap ci consente di prendere due piccioni con una fava perché include un margine nell'etichetta:

```
<label for="name">Enter Your Name</label>
<input type="text" ng-model="name" class="form-control input-lg" id="name">
```



Quiz

1. Come viene inizializzato Angular sulla pagina?
2. Che cosa si usa per visualizzare sulla pagina un valore del modello?
3. A che cosa corrisponde l’acronimo MVC?
4. Come creiamo un controller e come diciamo ad Angular che intendiamo utilizzarlo?
5. In Bootstrap 3 qual è il nuovo nome di Hero Unit?

Riepilogo

La nostra app è bella e funziona proprio come dovrebbe; riepiloghiamo ciò che abbiamo imparato in questo primo capitolo.

Innanzitutto abbiamo visto come è facile installare AngularJS e Bootstrap includendo un solo file JavaScript e un unico foglio di stile. Abbiamo anche osservato come viene inizializzata un'applicazione Angular e abbiamo iniziato a realizzare la nostra prima app.

L'app Hello, World che abbiamo creato, seppur molto semplice, illustra alcune caratteristiche fondamentali di Angular:

- espressioni;
- scope;
- modelli;
- binding dei dati bidirezionale.

Tutto questo è stato possibile senza scrivere una sola riga di JavaScript; infatti il controller che abbiamo creato serviva solo a illustrare il binding bidirezionale e non era un componente obbligatorio dell'app.

Con Bootstrap, abbiamo utilizzato alcuni dei numerosi componenti disponibili, tra cui le classi `jumbotron` e `page-header` per attribuire all'app un po' di stile e sostanza. Inoltre abbiamo visto in azione il nuovo design responsive *mobile first* senza dover affollare il markup con classi o elementi non necessari.

Nel Capitolo 2 esamineremo più nel dettaglio alcune caratteristiche fondamentali di AngularJS e Bootstrap e presenteremo il progetto che implementeremo nel corso del libro.

Sviluppare con AngularJS e Bootstrap

Dopo aver creato ufficialmente la prima app utilizzando AngularJS e Bootstrap, è giunto il momento di fare un salto di qualità.

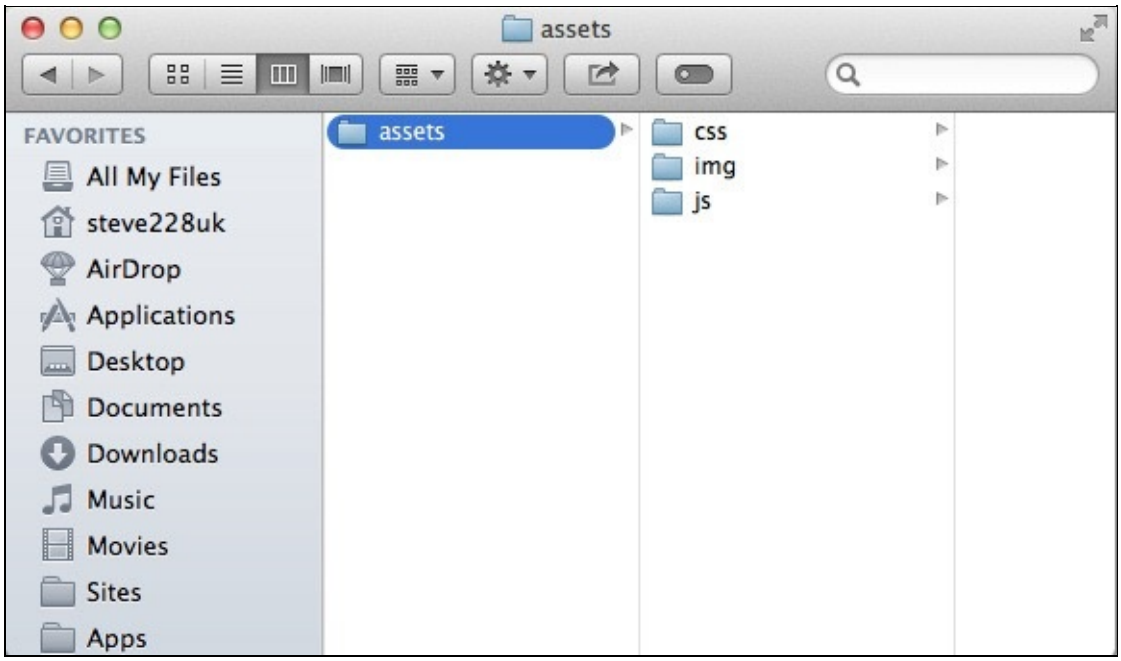
In questo libro vi servirete di entrambi i framework per sviluppare un’app di gestione dei contatti con tanto di ricerca testuale, creazione, modifica e cancellazione.

Esamineremo una base di codice che può essere mantenuta e sfrutteremo tutto il potenziale di entrambi i framework. Iniziamo a sviluppare!

Impostazione

Creiamo rapidamente una nuova directory per l'app e impostiamo una struttura simile all'app Hello, World che abbiamo sviluppato nel Capitolo 1.

Per esempio, nella struttura di cartelle mostrata nella prossima figura, abbiamo inserito le directory nella directory *assets* per tenere tutto in ordine. Copiate Angular e Bootstrap, come descritto nel Capitolo 1 all'interno delle directory pertinenti e create il file `index.html` nella radice, che diventerà la base dell'app di gestione dei contatti. Il seguente snippet è una semplice pagina HTML con integrati Bootstrap e Angular. Abbiamo anche inizializzato Angular sulla pagina con l'attributo `ng-app` nel tag `<html>`.



Ecco come dovrebbe apparire in questa fase:

```
<!DOCTYPE html>
<html lang=”en” ng-app>
<head>
  <meta charset=”utf-8”>
  <title>Contacts Manager</title>
  <link rel=”stylesheet” href=”assets/css/bootstrap.min.css”>
  <script type=”text/javascript”
    src=”assets/js/angular.min.js”></script>
</head>
<body>

</body>
</html>
```

Scaffolding

Dopo aver impostato il file e la struttura delle cartelle di base, possiamo iniziare a effettuare lo scaffolding dell'app grazie a Bootstrap. Oltre a offrire una serie di componenti, come la navigazione e i pulsanti, che possiamo utilizzare nell'app di gestione dei contatti, Bootstrap propone anche un sistema di griglie responsive molto potente di cui sfrutteremo ogni potenzialità.

Barra di navigazione

Avremo bisogno di una *barra di navigazione* (`navbar`) per passare da una vista all'altra. Ovviamente sarà collocata in cima allo schermo.

Osserviamo la barra di navigazione completa prima di esaminarla più a fondo:

```
<nav class="navbar navbar-default"role="navigation">
    <div class="navbar-header">
        <button type="button" class="navbar-toggle"
          data-toggle="collapse" data-target="#nav-toggle">
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
        </button>
        <a class="navbar-brand" href="/">Contacts Manager</a>
    </div>

    <div class="collapse navbar-collapse" id="nav-toggle">
        <ul class="nav navbar-nav">
            <li class="active"><a href="/">Browse</a></li>
            <li><a href="/add">Add Contact</a></li>
        </ul>
        <form class="navbar-form navbar-right" role="search">
            <input type="text" class="form-control"
            placeholder="Search">
        </form>
    </div>
</nav>
```

Questo codice potrebbe intimorirvi, considerato che è quello di un componente della pagina molto semplice, ma se l'analizzate nel dettaglio, noterete che ogni elemento è necessario.

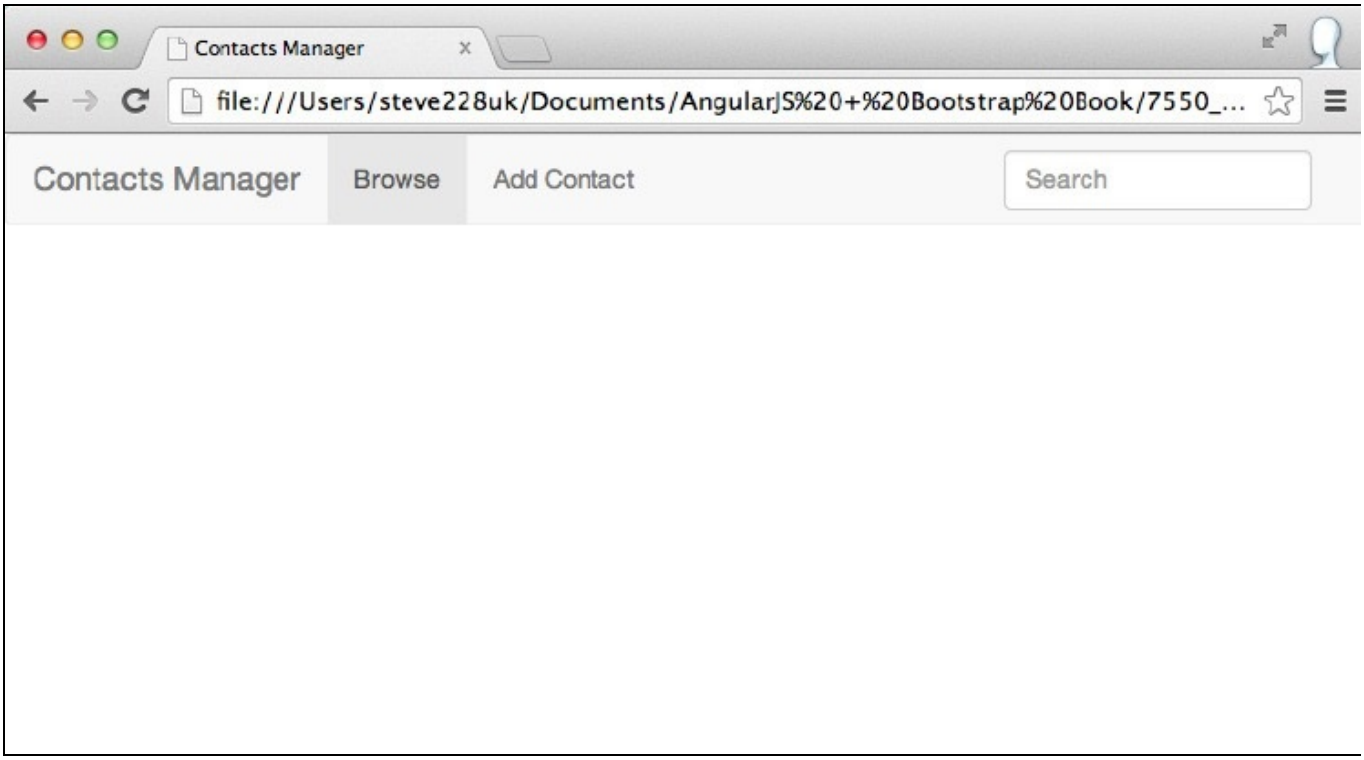
Il tag `<nav>` contiene tutto ciò che è presente nella barra di navigazione. Al suo interno si trovano due sezioni: `navbar-header` e `navbar-collapse`. Questi elementi sono riservati alla navigazione mobile e controllano ciò che viene mostrato o nascosto dal pulsante interruttore.

L'attributo `data-target` per il pulsante corrisponde direttamente all'attributo `id` dell'elemento `navbar-collapse`; in questo modo Bootstrap sa che cosa deve attivare. La prossima schermata riproduce la barra di navigazione su dispositivi più grandi di un tablet.

Includeremo la barra di navigazione direttamente all'interno del tag `<body>`. In questo modo occuperemo tutta la larghezza della finestra del browser.

Se la ridimensionate, vedrete che Bootstrap visualizza l'header mobile con il pulsante interruttore per schermi di dimensioni inferiori ai 768 px, ossia le dimensioni dello schermo di un iPad con orientamento verticale. Tuttavia se fate clic sul pulsante per muovervi nella navigazione, vedrete che non succede nulla. Ciò accade perché non abbiamo incluso in Bootstrap il file JavaScript, presente nel file ZIP che abbiamo scaricato in precedenza.

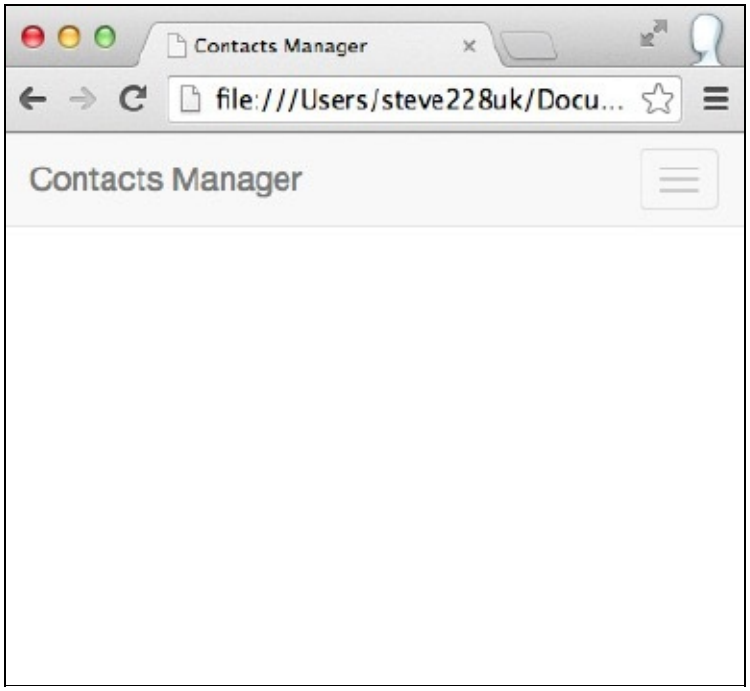
Copiatelo nella directory *js* dell'app e fatevi riferimento nel file `index.html`. Dovrete includere anche jQuery nell'applicazione, poiché il JS di Bootstrap dipende da questo.



Potete trovare l'ultima versione all'indirizzo `http://jquery.com/`; ancora una volta aggiungetela alla directory e includetela nella pagina prima di `bootstrap.js`. Verificate che i file JavaScript compaiano nel seguente ordine:

```
<script src="assets/js/jquery.min.js"></script>
<script src="assets/js/bootstrap.min.js"></script>
<script src="assets/js/angular.min.js"></script>
```

Se riaggiornate la pagina dovreste riuscire a far clic sul pulsante interruttore per visualizzare la navigazione per dispositivi mobili.



Le griglie di Bootstrap

Il sistema a griglia con 12 colonne di Bootstrap è molto potente e ci permette di effettuare lo scaffolding della nostra app responsive con pochissimi elementi, approfittando del CSS modulare. La griglia è composta da righe e colonne che è possibile adattare utilizzando una serie di classi. Prima di iniziare, dobbiamo inserire un contenitore per le righe, altrimenti il framework non si comporterà come previsto. È sufficiente un semplice tag `<div>` da collocare sotto la barra di navigazione:

```
<div class="container"></div>
```

In questo modo la griglia sarà centrata e aggiungeremo la proprietà `max-width` per disporre tutto per bene.

Esistono quattro prefissi per le classi, che definiscono il comportamento delle colonne. Utilizzeremo perlopiù il prefisso `col-sm-`, che comprime le colonne in modo che appaiano una sopra l'altra quando il contenitore ha una larghezza inferiore a 750 px.

Le altre classi si riferiscono tutte a diverse dimensioni dello schermo dei dispositivi e si comportano in modo simile. La seguente tabella, tratta da `http://getbootstrap.com/`, mostra le differenze tra le quattro classi.

	Cellulari (<768 px)	**Tablet (≥768 px)**	**Desktop (≥992 px)**	**Desktop (≥1200 px)**
Comportamento della griglia	Sempre orizzontale	All'inizio compressa, orizzontale sopra i breakpoint		
Larghezza massima	Nessuna			

del contenitore	(automatico)	750 px	970 px	1170 px
Prefisso delle classi	`.col-xs-`	`.col-sm-`	`.col-md-`	`.col-lg-`
Larghezza massima della colonna	Automatica	60 px	78 px	95 px
Offset	N/D	Sì		
Ordinamento delle colonne	N/D	Sì		

Creiamo rapidamente un layout a due colonne con un'area del contenuto principale e una barra laterale. Siccome la griglia è costituita da 12 colonne, dovremo far sì che l'area del contenuto rientri in esse, altrimenti avremo dello spazio vuoto.

Otto colonne per l'area del contenuto e quattro per la barra laterale dovrebbero essere una soluzione ottimale, ma come implementarle?

All'interno del contenitore creiamo un nuovo tag `<div>` con la classe `row`. Possiamo impostare quante righe desideriamo; ciascuna può contenere al massimo dodici colonne:

```
<div class=”container”>
    <div class=”row”>

    </div>
</div>
```

Siccome vogliamo che le colonne vengano visualizzate su dispositivi mobili, utilizzeremo il prefisso `col-sm-`. La creazione di una colonna è semplice; è sufficiente scegliere il prefisso opportuno e aggiungere il numero delle colonne che si desidera impostare. Osservate il layout di base a due colonne:

```
<div class=”container”>
    <div class=”row”>
        <div class=”col-sm-8”>
            This is our content area
        </div>
        <div class=”col-sm-4”>
            Here is our sidebar
        </div>
    </div>
</div>
```

Se lo visualizzate su uno schermo di dimensioni superiori a quello di un dispositivo mobile, Bootstrap aggiungerà automaticamente 30 px di spazio tra le colonne (15 px su ciascun lato). Tuttavia, talvolta vorrete inserire altro spazio tra le colonne e distanziarle un po'. Bootstrap permette di ottenere questo risultato aggiungendo un'altra classe alla colonna.

Scegliete di nuovo il prefisso opportuno, ma questa volta aggiungete la parola chiave `offset`:

```
<div class=”col-sm-4 col-sm-offset-1”></div>
```

In questo caso il numero presente alla fine controlla il numero delle colonne su cui si imposta l'offset. La nuova classe aggiunge un ulteriore margine a sinistra.

NOTA

Ricordate: la somma delle colonne in una riga, compreso l'offset, non deve essere superiore a 12.

All'interno delle colonne è possibile nidificare altre righe e colonne per creare un layout più complesso:

```
<div class="container">
    <div class="row">
        <div class="col-sm-8">
            <div class="row">
                <div class="col-sm-6">
                    <p>Lorem ipsum dolor…<p>
                </div>
                <div class="col-sm-6">
                    <p>Class aptent taciti…</p>
                </div>
            </div>
        </div>
    </div>
</div>
```

Così si creeranno due colonne all'interno del contenitore principale che abbiamo impostato prima. A titolo di esempio, abbiamo inserito all'interno del testo fittizio.

Nel browser vedrete che ora sono presenti tre colonne. Tuttavia, siccome la griglia è nidificata, possiamo creare una nuova riga e una sola colonna, tre colonne o quello che richiede il layout.

Classi helper

Bootstrap include alcune classi helper che possiamo utilizzare per adattare il layout. In genere sono funzionali e rispondono a un unico scopo. Osserviamo alcuni esempi.

Elementi flottanti

Gli elementi flottanti sono spesso essenziali per creare un layout efficace sul Web, e Bootstrap offre due classi per rendere mobili gli elementi a sinistra o a destra:

```
<div class="pull-left">...</div>
<div class="pull-right">...</div>
```

Per utilizzare efficacemente gli elementi flottanti, dobbiamo racchiuderli in una classe `clearfix`. In questo modo li isoleremo, e il flusso del documento verrà visualizzato come previsto:

```
<div class="clearfix">
  <div class="pull-left">...</div>
  <div class="pull-right">...</div>
</div>
```

Se le classi `float` si trovano direttamente all'interno di un elemento con la classe `row`, gli elementi "mobili" vengono isolati automaticamente da Bootstrap e non è necessario applicare manualmente la classe `clearfix`.

Centrare gli elementi

Insieme con gli elementi flottanti, spesso è necessario centrare gli elementi a livello di blocco. Bootstrap permette di farlo con la classe `center-block`:

```
<div class="center-block">...</div>
```

In questo modo si impostano le proprietà del margine sinistro e destro su automatico, e l'elemento sarà centrato.

Mostrare o nascondere

Potreste voler mostrare o nascondere gli elementi con il CSS; Bootstrap offre due classi per ottenere questo risultato:

```
<div class="show">...</div>
<div class="hidden">...</div>
```

È importante osservare che la classe imposta la proprietà `display` su `block`; quindi applicatela soltanto agli elementi a livello di blocco e non a quelli `inline` o `inline-block`.

Bootstrap include inoltre numerose classi per permettere agli elementi di venire mostrati o nascosti su schermi di dimensioni specifiche. Le classi utilizzano le stesse dimensioni predefinite della griglia di Bootstrap.

Per esempio, il codice seguente nasconderà un elemento su uno schermo con dimensioni specifiche:

```
<div class="hidden-md"></div>
```

Nasconderà l'elemento su dispositivi con schermi di dimensioni medie, ma lo lascerà ancora visibile su dispositivi mobili, tablet e grandi desktop. Per nascondere un elemento su più dispositivi, dovete utilizzare più classi:

```
<div class="hidden-md hidden-lg"></div>
```

Le classi `visible` hanno un comportamento opposto e mostrano gli elementi su schermi di dimensioni specifiche. Tuttavia, a differenza delle classi `hidden`, bisogna impostare il valore `display`, che può essere `block`, `inline` o `inline-block`:

```
<div class="visible-md-block"></div>
<div class="visible-md-inline"></div>
<div class="visible-md-inline-block"></div>
```

Ovviamente è possibile utilizzare le diverse classi in associazione. Se per esempio volete che compaia un elemento a livello di blocco su uno schermo piccolo, che in seguito deve diventare `inline-block`, dovrete utilizzare il codice seguente:

```
<div class="visible-sm-block visible-md-inline-block"></div>
```

Se non ricordate le diverse dimensioni delle classi, consultate il paragrafo "Le griglie di Bootstrap".

Utilizzare le direttive

Senza saperlo abbiamo già utilizzato ciò che Angular definisce *direttive*. Si tratta in sostanza di potenti funzioni che possono essere chiamate da un attributo o dal suo stesso elemento. Angular ne include molte. Sia che intendiamo eseguire il ciclo dei dati, gestire i clic o inviare i form, Angular velocizzerà tutte queste operazioni.

Abbiamo utilizzato una direttiva per inizializzare Angular sulla pagina tramite `ng-app`, e tutte le direttive che esamineremo in questo capitolo vengono usate allo stesso modo: aggiungendo un attributo a un elemento.

Prima di osservare le altre direttive integrate, è necessario creare rapidamente un controller. Create un nuovo file e chiamatelo `controller.js`. Salvatelo nella directory *js* all'interno del progetto e apritelo nell'editor.

Come abbiamo visto nel Capitolo 1, i controller sono semplici funzioni standard del costruttore JS nelle quali è possibile inserire i servizi di Angular come `$scope`. Di queste funzioni viene creata un'istanza quando Angular rileva l'attributo `ng-controller`. In questo modo possiamo avere molteplici istanze dello stesso controller all'interno dell'applicazione e riutilizzare buona parte del codice. Questa dichiarazione di funzione è tutto ciò di cui abbiamo bisogno per il controller:

```
function AppCtrl(){
}
```

Per comunicare al framework che questo è il controller che intendiamo utilizzare, dobbiamo includerlo nella pagina dopo che Angular viene caricato, e aggiungere la direttiva `ng-controller` al tag di apertura `<html>`:

```
<html ng-controller="AppCtl">
…
<script type="text/javascript"
  src="assets/js/controller.js"></script>
```

ng-click e ng-mouseover

Una delle operazioni fondamentali che è possibile eseguire con JavaScript consiste nel gestire un evento clic. A questo scopo si utilizza l'attributo `onclick` su un elemento ricorrendo a jQuery o a un listener di eventi. In Angular ricorreremo a una direttiva.

Come esempio, creeremo un pulsante che aprirà una finestra di avviso: un'operazione semplice. Iniziamo ad aggiungere il pulsante all'area del contenuto creata in precedenza:

```
<div class="col-sm-8">
    <button>Click Me</button>
</div>
```

Se lo visualizzate nel browser, vedrete un pulsante HTML standard: fin qui nessuna sorpresa. Prima di associare la direttiva a questo elemento, dobbiamo creare un handler nel controller. Si tratta di una semplice funzione all'interno del controller associato allo scope. È molto importante associare la funzione allo scope, altrimenti sarà impossibile accedervi dalla vista:

```
function AppCtl($scope){
    $scope.clickHandler = function(){
        window.alert('Clicked!');
    };
}
```

Come sappiamo, possiamo avere molteplici scope su una pagina; questi sono oggetti ai quali in Angular possono accedere la vista e il controller. Per fare in modo che il controller vi abbia accesso, abbiamo inserito il servizio `$scope` nel controller. Questo servizio mette a disposizione lo scope che Angular crea sull'elemento al quale abbiamo aggiunto l'attributo `ng-controller`.

Angular si basa molto sull'inserimento delle dipendenze, che forse potreste non conoscere. Come abbiamo visto, Angular è suddiviso in moduli e servizi. Ciascuno di questi moduli e servizi dipende l'uno dall'altro e l'inserimento delle dipendenze consente la trasparenza referenziale. Durante il test dell'unità, possiamo anche simulare degli oggetti che verranno inseriti per confermare i risultati dei test. L'inserimento delle dipendenze ci permette di dire ad Angular da quali servizi dipende il controller, e il framework li convertirà per noi.

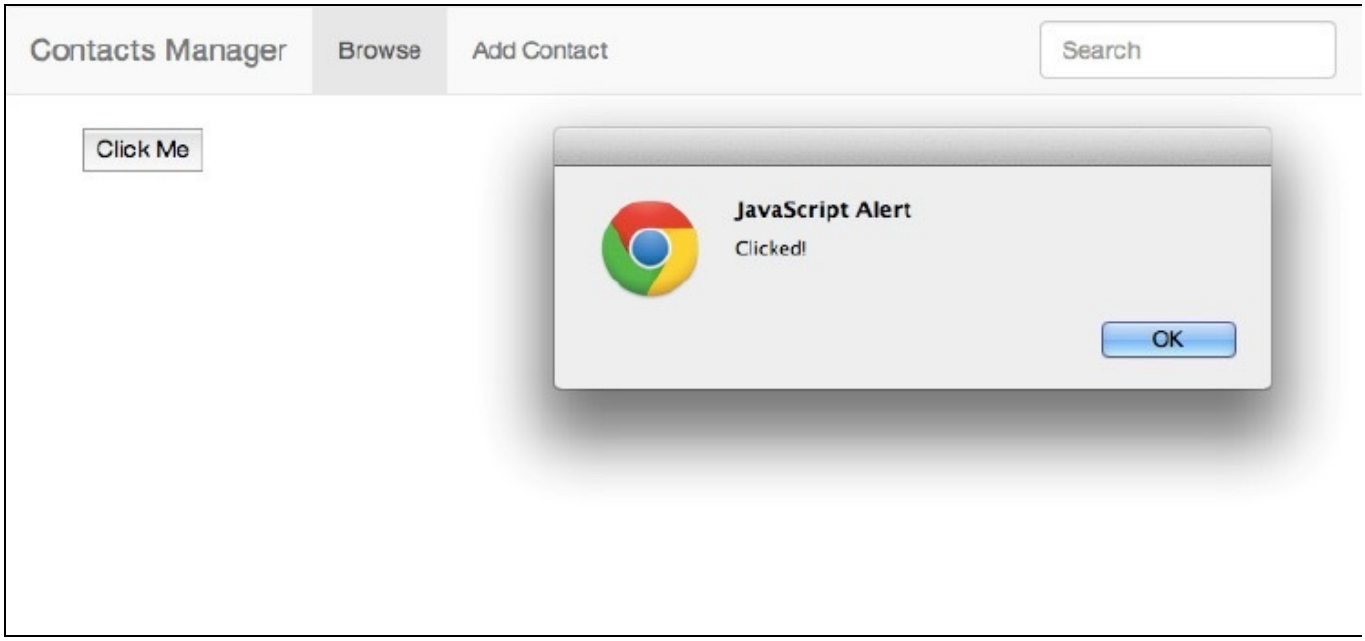
Potete trovare una spiegazione dettagliata sull'inserimento delle dipendenze di AngularJS nella documentazione ufficiale all'indirizzo `https://docs.angularjs.org/guide/di`.

Ora l'handler è impostato; come abbiamo fatto in precedenza, dobbiamo aggiungere la direttiva al pulsante, come attributo aggiuntivo. Questa volta passeremo il nome della funzione che intendiamo eseguire, che in questo caso è `clickHandler`. Angular valuterà tutto ciò che inseriremo nella direttiva come espressione

di AngularJS; pertanto dobbiamo fare attenzione ad aggiungere due parentesi che indicano che stiamo chiamando una funzione:

```
<button ng-click="clickHandler()">Click Me</button>
```

Se lo visualizzate nel browser, vedrete una finestra di avviso quando farete clic sul pulsante. Non è necessario includere la variabile `$scope` quando si chiama la funzione nella vista. Le funzioni e le variabili alle quali è possibile accedere dalla vista rimangono all'interno dello scope corrente o di qualsiasi scope precedente.



Se desiderate visualizzare la finestra di avviso all'*hover* (cioè al passaggio) invece che al clic, è sufficiente modificare il nome della direttiva in `ng-mouseover`, poiché entrambe funzionano nello stesso modo.

ng-init

La direttiva `ng-init` valuta un'espressione sullo scope corrente e può essere utilizzata da sola o in associazione con altre direttive. La priorità di esecuzione è massima rispetto ad altre direttive, per fare in modo che l'espressione venga valutata in tempo.

Ecco un semplice esempio della direttiva `ng-init` in azione:

```
<div ng-init="test = 'Hello, World'"></div>
{{test}}
```

In questo modo comparirà sullo schermo *Hello, World* quando l'applicazione sarà caricata nel browser. Sopra abbiamo impostato il valore del modello di test e utilizzato la sintassi con le doppie parentesi graffe per visualizzarlo.

ng-show e ng-hide

Talvolta dovremo controllare la visualizzazione programmata di un elemento. Sia `ng-show` sia `ng-hide` possono essere controllate dal valore restituito da una funzione o da un modello.

Possiamo avvalerci della funzione `clickHandler` che abbiamo creato per mostrare in azione la direttiva `ng-click` al fine di rendere visibile o nascondere l'elemento. A questo scopo creeremo un nuovo modello e alterneremo il valore tra vero e falso.

Creiamo l'elemento che mostreremo e nasconderemo. Immettete il seguente codice sotto il pulsante:

```
<div ng-hide="isHidden">
    Click the button above to toggle.
</div>
```

Il valore nell'attributo `ng-hide` è il nostro modello. Siccome è compreso nello scope, possiamo facilmente modificarlo nel controller:

```
$scope.clickHandler = function(){
    $scope.isHidden = !$scope.isHidden;
};
```

Qui stiamo invertendo il valore del modello, che a sua volta rende visibile o meno il `<div>`.

Se lo aprite nel browser, vedrete che l'elemento è nascosto per impostazione predefinita. Esistono alcuni sistemi per gestire questo aspetto. Potremmo impostare il valore di `$scope.hidden` su `true` nel controller, oppure impostare il valore di `hidden` su `true` utilizzando la direttiva `ng-init`. Altrimenti è possibile ricorrere alla direttiva `ng-show` che ha un comportamento contrario rispetto a `ng-hide` e renderà visibile un elemento se il valore di un modello è impostato su `true`.

NOTA

Verificate che Angular sia caricato nell'header, altrimenti `ng-hide` e `ng-show` non funzioneranno correttamente. Questo perché Angular utilizza le sue classi per nascondere gli elementi e queste devono essere caricate al rendering della pagina.

ng-if

Angular include anche una direttiva `ng-if` che si comporta in modo simile a `ng-show` e `ng-hide`. Tuttavia `ng-if` elimina l'elemento dal DOM, mentre `ng-show` e `ng-hide` rendono visibili o meno gli elementi.

Osserviamo rapidamente come è possibile utilizzare `ng-if` nel codice precedente:

```
<div ng-if=”isHidden”>
    Click the button above to toggle.
</div>
```

Se volessimo invertire il significato dell’istruzione, sarebbe sufficiente aggiungere un punto esclamativo prima dell’espressione:

```
<div ng-if=”!isHidden”>
    Click the button above to toggle.
</div>
```

ng-repeat

Ben presto, durante lo sviluppo di una web app, avrete bisogno di rappresentare un array di item. Per esempio, nell’app di gestione dei contatti, potrebbe essere un elenco di contatti, o qualsiasi altra cosa. Angular consente di raggiungere questo scopo con la direttiva `ng-repeat`.

Vediamo un esempio di alcuni dati che potreste incontrare. Si tratta di un array di oggetti con molteplici proprietà al suo interno. Per visualizzare i dati, dovremo riuscire ad accedere a ogni proprietà. `ng-repeat` serve a questo scopo.

Ecco il controller con un array di oggetti del contatto assegnati al modello dei contatti:

```
function AppCtrl($scope){

    $scope.contacts = [
        {
            name: ‘John Doe’,
            phone: ‘01234567890’,
            email: ‘john@example.com’
        },
        {
            name: ‘Karan Bromwich’,
            phone: ‘09876543210’,
            email: ‘karan@email.com’
        }
    ];

}
```

In questo caso abbiamo solo due contatti, ma come potete immaginare, un’API potrebbe servirne centinaia che non sarebbe possibile gestire senza `ng-repeat`.

Aggiungete un array di contatti al controller e assegnatelo a `$scope.contacts`. Aprite il file `index.html` per creare un tag `<ul>`. Ripeteremo una voce presente in questo elenco non ordinato, quindi questo è l’elemento al quale dobbiamo aggiungere la direttiva:

```
<ul>
    <li ng-repeat=”contact in contacts”></li>
</ul>
```

Se sapete come funzionano i cicli in PHP o Ruby, vi sentirete a vostro agio. Create una variabile alla quale poter accedere all'interno dell'elemento corrente che entra nel ciclo. La variabile dopo la parola chiave `in` fa riferimento al modello che abbiamo creato su `$scope` nel controller. Così è possibile accedere a qualsiasi proprietà impostata sull'oggetto, e ogni iterazione o item ripetuti acquisiscono un nuovo scope. Possiamo visualizzarli sulla pagina utilizzando la sintassi delle doppie parentesi graffe di Angular, come abbiamo visto nel Capitolo 1:

```
<ul>
    <li ng-repeat="contact in contacts">
        {{contact.name}}
    </li>
</ul>
```

Così verrà visualizzato, come previsto, il nome all'interno della voce dell'elenco e sarà possibile accedere facilmente a qualsiasi proprietà dell'oggetto del contatto facendo riferimento a esso tramite la sintassi a punto standard.

ng-class

Spesso capita di voler modificare o aggiungere una classe a un elemento. A questo scopo possiamo ricorrere alla direttiva `ng-class`, per definire una classe da aggiungere o da eliminare in base al valore di un modello.

Esistono due sistemi per utilizzare `ng-class`. Nella sua forma più semplice, Angular applicherà il valore del modello come classe CSS all'elemento:

```
<div ng-class="exampleClass"></div>
```

Se il modello al quale facciamo riferimento è indefinito o falso, Angular non applicherà la classe. Questo sistema è ottimo per le singole classi, ma se volessimo avere maggiore controllo o applicare più classi a un solo elemento? Provate questo codice:

```
<div ng-class="{className: model, class2: model2}"></div>
```

In questo caso l'espressione è un po' diversa. Disponiamo di un insieme di nomi di classi e il modello con il quale confrontarlo. Se il modello restituisce `true`, la classe verrà aggiunta all'elemento.

Osserviamolo in azione. Utilizzeremo le caselle di controllo con l'attributo `ng-model` che abbiamo già esaminato nel Capitolo 1 per applicare alcune classi a un paragrafo:

```
<p ng-class="{'text-center': center, 'text-danger': error}">
    Lorem ipsum dolor sit amet
</p>
```

Ho aggiunto due classi Bootstrap `text-center` e `text-danger`. Queste esaminano un paio di modelli, che possiamo modificare rapidamente con alcune caselle di controllo:

```
<label><input type="checkbox" ng-model="center"> text-
  center</label>
<label><input type="checkbox" ng-model="error"> text-
  danger</label>
```

NOTA

Le virgolette semplici che racchiudono i nomi delle classi nell'espressione sono necessarie solo quando si utilizzano i trattini (`-`), altrimenti Angular genera un errore.

Quando queste caselle di controllo sono spuntate, le classi opportune vengono applicate all'elemento.

ng-style

In modo simile a `ng-class`, questa direttiva ci consente di assegnare dinamicamente uno stile a un elemento con Angular. A titolo di esempio creeremo una terza casella di controllo che applicherà altri stili all'elemento paragrafo.

La direttiva `ng-style` utilizza un oggetto JavaScript standard, e le parole chiave saranno la proprietà che intendiamo modificare (per esempio colore e sfondo). Questo si può applicare da un modello o un valore restituito da una funzione.

Esaminiamo come associarlo a una funzione che verificherà un modello. Poi possiamo aggiungerlo alla casella di controllo per cambiare gli stili.

Aprite il file `controller.js` e create una nuova funzione associata allo scope. Qui la chiameremo `styleDemo`:

```
$scope.styleDemo = function(){
    if(!$scope.styler){
        return;
    }

    return {
        background: 'red',
        fontWeight: 'bold'
    };
};
```

All'interno della funzione dobbiamo verificare il valore di un modello; in questo esempio si chiama `styler`. Se è falso, non restituirà nulla, altrimenti restituirà un oggetto con le proprietà CSS. Abbiamo utilizzato `fontWeight` invece di `font-weight` nell'oggetto restituito. Entrambi vanno bene e Angular applicherà automaticamente la sintassi camel case nella corretta proprietà CSS. Ricordate che quando si utilizzano

i trattini nelle parole chiave degli oggetti JavaScript, queste vanno racchiuse tra virgolette.

Questo modello sarà associato a una casella di controllo, come abbiamo fatto con `ng-class`:

```
<label><input type=”checkbox” ng-model=”styler”> ng-style</label>
```

L’ultima cosa che dobbiamo fare è aggiungere la direttiva `ng-style` all’elemento del paragrafo:

```
<p .. ng-style=”styleDemo()”>
    Lorem ipsum dolor sit amet
</p>
```

Angular è in grado di richiamare questa funzione ogni volta che cambia lo scope. Ciò significa che non appena il valore del modello cambia da `false` a `true`, verranno applicati gli stili e viceversa.

ng-cloak

L’ultima direttiva che esamineremo è `ng-cloak`. Quando utilizzate i template di Angular in una pagina HTML, vengono visualizzate temporaneamente le doppie parentesi graffe prima che AngularJS abbia finito di caricare e compilare tutto sulla pagina. Per ovviare a questo comportamento, dobbiamo nascondere temporaneamente il template prima che termini il rendering.

Angular consente di farlo con la direttiva `ng-cloak`, che imposta un ulteriore stile sull’elemento mentre viene caricato, ovvero `display: none !important;`.

NOTA

Per evitare il flashing durante il caricamento del contenuto, è importante che Angular venga caricato nella sezione `head` della pagina HTML.

Quiz

1. Che cosa aggiungiamo in cima alla pagina per poter passare da una vista all’altra?
2. Quante colonne comprende il sistema a griglia di Bootstrap?
3. Che cos’è una direttiva e come viene utilizzata la maggior parte di esse?
4. Quale direttiva dobbiamo utilizzare per eseguire il ciclo dei dati?

Riepilogo

In questo capitolo abbiamo affrontato molti argomenti; prima di proseguire con il prossimo, riepiloghiamoli.

Bootstrap ci consente di creare rapidamente una navigazione responsive. Dobbiamo includere il file JavaScript presente nella versione di Bootstrap che abbiamo scaricato per rendere disponibile il pulsante interruttore per la navigazione mobile.

Abbiamo anche esaminato il potente sistema a griglia responsive incluso in Bootstrap e creato un semplice layout a due colonne. Durante questa operazione abbiamo analizzato i quattro diversi prefissi delle classi per le colonne e nidificato la griglia. Per adattare il layout abbiamo scoperto alcune classi helper incluse nel framework che ci consentono di rendere mobili, centrare e nascondere gli elementi.

Abbiamo esaminato in dettaglio le direttive integrate in Angular, le funzioni che è possibile utilizzare dalla vista. Prima di osservarle in azione, abbiamo creato un controller, ossia una funzione in cui possiamo passare i servizi di Angular sfruttando l'inserimento delle dipendenze.

Le direttive che abbiamo descritto saranno fondamentali man mano che svilupperemo l'app di gestione dei contatti nel corso del libro. Direttive come `ng-click` e `ng-mouseover` sono in sostanza nuovi sistemi per gestire gli eventi, operazione che avete sicuramente svolto con jQuery o vanilla JavaScript, ma direttive come `ng-repeat` rappresenteranno forse un modo tutto nuovo di lavorare, che introdurrà una logica nella vista per eseguire il ciclo dei dati e visualizzarli sulla pagina.

Abbiamo anche analizzato le direttive che valutano i modelli nello scope e compiono diverse azioni sulla base dei loro valori. `ng-show` e `ng-hide` mostrano o nascondono un elemento sulla base del valore di un modello. Lo abbiamo visto anche con `ng-class`, che ci ha permesso di aggiungere delle classi agli elementi sulla base dei valori dei modelli.

I filtri

Nel capitolo precedente abbiamo esaminato uno dei componenti fondamentali di AngularJS: le direttive. Come molti framework, Angular offre altri paradigmi che ci aiutano a sviluppare le app. I filtri permettono di manipolare e ordinare facilmente i dati dalla vista o dal controller, e come per le direttive, ne esistono alcuni efficaci già integrati.

Si possono applicare in molti casi e in questo capitolo ne esamineremo alcuni. Per esempio, è possibile manipolare una stringa, convertendola, localizzandola o troncandola. I filtri consentono anche di operare con altri tipi JavaScript, come array e oggetti. Forse vi capiterà di impostare una ricerca per filtrare un set di dati di cui avete eseguito il ciclo usando `ng-repeat`. Tutto questo è possibile con i filtri.

Prima di osservare alcuni filtri inclusi, esamineremo come è possibile applicare un filtro dalla vista.

Applicare un filtro dalla vista

È possibile applicare i filtri direttamente alle espressioni all'interno dei template. Ricordate che un'espressione è tutto ciò che è compreso all'interno della sintassi con doppie parentesi graffe o di una direttiva:

```
{{expression | filter}}
```

È facile servirsi di un filtro; è sufficiente aggiungere il simbolo pipe (`|`) seguito dal nome del filtro che intendiamo applicare all'espressione. Possiamo procedere allo stesso modo per applicare più filtri a una sola espressione. È possibile concatenarne più di uno e applicarli in sequenza. Nell'esempio seguente, `filter2` verrà applicato all'output del `filter1` e così via:

```
{{expression | filter1 | filter2 | filter3}}
```

Alcuni filtri possono avere degli argomenti, che si possono applicare ricorrendo a una sintassi simile:

```
{{expression | filter:argument1:argument2}}
```

In questo capitolo illustreremo alcuni filtri inclusi in Angular direttamente dalla vista utilizzando la sintassi che abbiamo esaminato. Vedremo come applicare gli stessi filtri dal controller e come crearne uno.

Currency e number

Il primo filtro che analizzeremo formatta i numeri in valuta. Nella versione localizzata britannica-americana, aggiunge al posto giusto una virgola per separare le migliaia dai decimali. Li fa anche precedere dal simbolo opportuno:

```
{{12345 | currency}}
```

Il simbolo della valuta dipenderà dalla versione localizzata. Se utilizziamo quella britannica-americana, per impostazione predefinita, Angular antepone il simbolo del dollaro (`$`), ma possiamo passare il simbolo da noi scelto come argomento:

```
{{12345 | currency:'£'}}
```

È importante ricordarsi di racchiuderlo tra virgolette, come se fosse una stringa.

Angular include anche un secondo filtro per formattare i numeri, che ci offre maggiore controllo; ci consente di specificare il numero di cifre decimali a cui vogliamo arrotondare il numero:

```
{{12345.225 | number:2}}
```

L’output di questo filtro sarà `12,345.23`. Come potete osservare, il numero è stato arrotondato a due decimali ed è stata aggiunta una virgola per separare le migliaia.

Lowercase e uppercase

Questi due filtri sono forse i più semplici inclusi in Angular. Convertono la stringa in minuscolo o maiuscolo:

```
{{‘Stephen’ | lowercase}}

{{‘Declan’ | uppercase}}
```

L’output di questi filtri è il seguente:

```
stephen
DECLAN
```

limitTo

In alcuni casi potreste voler limitare il numero di caratteri di una stringa o di un array; questo risultato si può ottenere facilmente in AngularJS utilizzando il filtro `limitTo`:

```
{{‘Lorem ipsum dolor sit amet’ | limitTo:15}}
```

Questo filtro accetta un solo argomento, che è il numero di caratteri al quale dovrebbe limitarsi l’input. In questo caso abbiamo limitato i caratteri di una stringa, ma potrebbe trattarsi di un array in una direttiva `ng-repeat`, per esempio:

```
<div ng-repeat=”array | limitTo:2”></div>
```

Date

Quando si opera con i dati da una API, spesso la data viene fornita come ora UNIX o come timestamp completo. Non è molto comodo, ma fortunatamente Angular include un sistema semplice per formattare le date con un filtro:

```
{{expression | date:format}}
```

Questo filtro accetta un argomento: `format`. Per esempio, se abbiamo un timestamp e intendiamo ottenere l’anno, potremmo raggiungere facilmente questo risultato con l’espressione seguente:

```
{{725508723000 | date:’yyyy’}}
```

Possiamo combinarlo con l'input giorno mese e avremo facilmente una stringa di data standard:

```
{{725508723000 | date:'dd/MM/yyyy'}}
```

Ecco un elenco di alcuni degli elementi più utili da cui può essere costituita la stringa `format`. Un elenco completo si può trovare sul sito di AngularJS.

Elemento	Output	Esempio
yyyy	Anno in quattro cifre	2013
yy	Anno in due cifre	13
MMMM	Mese per esteso	Dicembre
MMM	Mese abbreviato	Dic
MM	Mese preceduto da zero	01
M	Mese in numero	1
dd	Giorno preceduto da zero	01
d	Giorno in numero	1
EEEE	Giorno della settimana	Lunedì
EEE	Giorno abbreviato della settimana	Lun
HH	Ora nel formato 24 ore preceduta da zero	01
H	Ora nel formato 24 ore	1
hh	Ora nel formato 12 ore preceduta da zero	01
h	Ora nel formato 12 ore	1
mm	Minuto preceduto da zero	05
m	Minuto	5
ss	Secondo preceduto da zero	09
s	Secondo	9
a	AM/PM	AM o PM
Z	Fuso orario	+0100
ww (solo 1.3+)	Settimana dell'anno preceduta da zero	03
w (solo1.3+)	Settimana dell'anno	3

Esistono inoltre alcuni formati predefiniti che possiamo utilizzare; esaminiamone uno:

```
{{725508723000 | date:'medium'}}
```

La parola chiave `medium` è solo uno dei formati di default che questo filtro riconosce, e genera Dec 28, 1992 2:12:03 AM.

Ecco un elenco completo dei formati predefiniti accettati dal filtro `date`.

Parola chiave	Equivalente	Esempio
`medium`	MMM d, y h:mm:ss a	Sep 3, 2010 12:05:08 pm

short	M/d/yy h:mm a	9/3/10 12:05 pm
fullDate	EEEE, MMMM d,y	Friday, September 3, 2010
longDate	MMMM d, y	September 3, 2010
mediumDate	MMM d, y	Sep 3, 2010
shortDate	M/d/yy	9/3/10
mediumTime	h:mm:ss a	12:05:08 pm
shortTime	h:mm a	12:05 pm

Possiamo anche includere valori letterali nella stringa del formato; per esempio:

```
{{725508723000 | date:”h ‘in the morning’”}}
```

I valori letterali devono essere racchiusi tra virgolette semplici. Dobbiamo quindi cambiare le virgolette semplici che racchiudono l’argomento con le virgolette doppie. Se voleste includere nella stringa una sola virgoletta, dovreste utilizzare due virgolette semplici:

```
{{725508723000 | date:”h ‘o’’clock’”}}
```

Filter

Questo filtro, con un nome che può generare confusione, consente di selezionare facilmente un set secondario di item in un array. All’interno della vista è possibile utilizzarlo combinato con la direttiva `ng-repeat` che abbiamo illustrato nel capitolo precedente.

Con esso possiamo sviluppare uno strumento di ricerca potente che filtrerà l’array. Esaminiamo l’esempio `ng-repeat` del Capitolo 2:

```
<ul>
    <li ng-repeat=”contact in contacts”>
        {{contact.name}} - {{contact.phone}}
    </li>
</ul>
```

Prima di aggiungere il filtro, dobbiamo aggiungere l’oggetto pattern che sarà utilizzato per la selezione dall’array. Può essere un modello, una stringa, un oggetto pattern o una funzione. Mentre creiamo una ricerca, aggiungiamo un modello a un input di testo:

```
<input type=”text” ng-model=”search”>
```

Non resta che associare il modello alla direttiva `ng-repeat`. Lo potete fare come con qualsiasi altro filtro: inserite un pipe seguito dal nome del filtro. In questo caso dobbiamo anche aggiungere un argomento che indica al filtro quale modello, stringa, oggetto o funzione intendiamo utilizzare:

```
<li ng-repeat=”contact in contacts | filter:search”>
```

Così possiamo servirci del campo di testo che creiamo per effettuare qualsiasi ricerca nell’array, che comprende nomi, numeri di telefono e indirizzi e-mail. Tuttavia, che cosa succede se intendiamo restringere la ricerca soltanto alla proprietà `name` dei nostri oggetti? È sufficiente cambiare il modello:

```
<input type=”text” ng-model=”search.name”>
```

È importante che lasciamo il nome del modello di `ng-repeat` su `search`, altrimenti il filtro non si limiterà alla proprietà desiderata.

In alternativa potremmo utilizzare la sintassi seguente per la direttiva `ng-repeat` per limitare il filtro a proprietà specifiche. In questo caso possiamo lasciare il nome del modello su `search`:

```
<li ng-repeat=”contact in contacts | filter:{‘name’: search}”>
```

orderBy

Oltre a filtrare l’oggetto all’interno della direttiva `ng-repeat`, possiamo anche ordinarlo. È un’ottima soluzione se i dati forniti da un array non sono già ordinati o se non esiste un’opzione per farlo.

Attualmente l’oggetto è tutto confuso e non presenta alcun ordine apparente. Osserviamo come possiamo ordinarlo per nome:

```
<li ng-repeat=”contact in contacts | filter:search |
  orderBy:’name’”>
```

Il primo argomento che possiamo passare è una stringa con il nome della proprietà in base alla quale intendiamo ordinare l’array. Se volessimo filtrare secondo il numero di telefono o l’indirizzo e-mail, potremmo passare questi valori.

Possiamo anche passare un secondo argomento attraverso un booleano che controlla se il filtro deve invertire l’ordine o meno:

```
<li ng-repeat=” .. | orderBy:’name’:true”>
```

JSON

L’ultimo filtro incluso serve soprattutto per il debugging. Trasformerà qualsiasi oggetto JavaScript in una stringa JSON per la visualizzazione sulla pagina.

Prendiamo l’array di contatti che abbiamo utilizzato nel capitolo precedente per mostrare in azione `ng-repeat` e applichiamo il filtro `json`:

```
{{contacts | json}}
```

Ciò che segue è l’output della vista:

```
 [
   {
      “name”:”John Doe”,
      “phone”:”01234567890”,
      “email”:”john@example.com”
   },
   {
      “name”:”Karan Bromwich”,
      “phone”:”09876543210”,
      “email”:”karan@email.com”
   },
   {
      “name”:”Declan Proud”,
      “phone”:”2341234231”,
      “email”:”declan@email.com”
   },
   {
      “name”:”Paul McKay”,
      “phone”:”912345678”,
      “email”:”p.mckay@domain.com”
   }
]
```

Come potete vedere, è una semplice rappresentazione JSON dell’array di oggetti creato in precedenza.

Applicare i filtri da JavaScript

Talvolta vorrete applicare un filtro utilizzando JavaScript, di solito dal controller; è importante capire come possiamo ottenere questo risultato, tramite due opzioni.

Possiamo inserire il servizio `$filter` nel controller e utilizzare qualsiasi filtro incluso nell'applicazione. In alternativa possiamo inserire il filtro come suo servizio dedicato e utilizzarlo da solo. Entrambi i metodi sono perfettamente validi e spetta a voi scegliere quello che preferite.

Esaminiamoli ricorrendo innanzitutto al servizio `$filter`. Consideriamo il filtro `json` che abbiamo appena visto e utilizziamo `console.log` sullo stesso array. Inseriamo il servizio nel controller:

```
function AppCtl($scope, $filter){
…
}
```

Ottimo! Ora possiamo utilizzarlo così come facciamo con `$scope`. A questo fine è sufficiente chiamarlo come una funzione e passare il nome del filtro che desideriamo usare, che nel nostro caso è `json`:

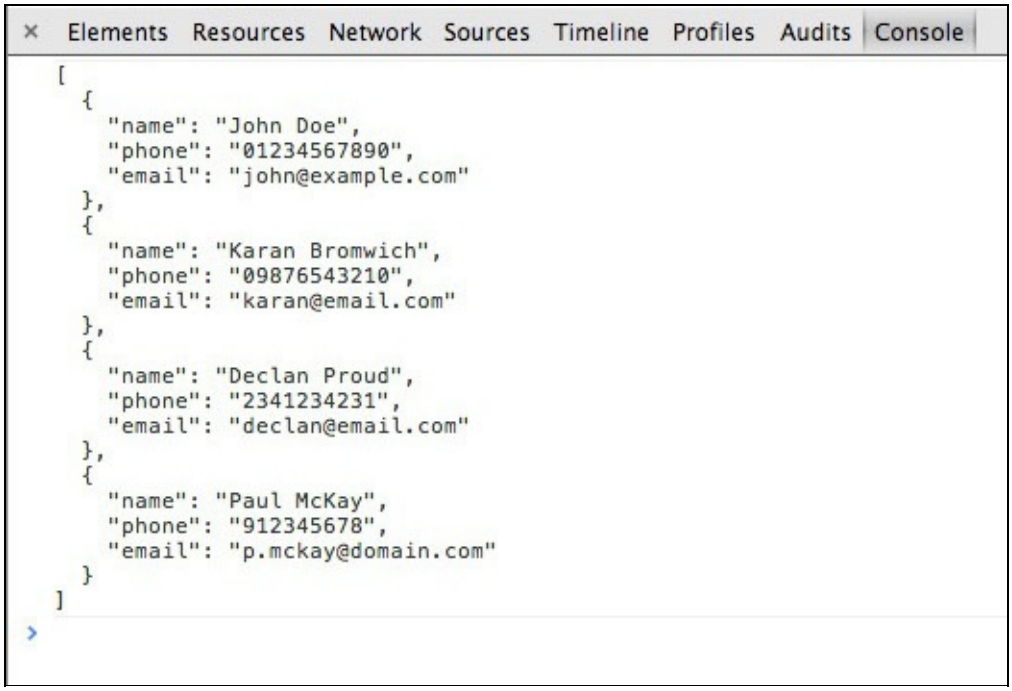
```
$filter('json');
```

La stringa restituisce il filtro stesso; possiamo vederlo nell'output se applichiamo direttamente la sintassi `console.log`. Ciò significa che possiamo chiamare immediatamente la funzione aggiungendo una seconda serie di parentesi subito dopo:

```
$filter('json')($scope.contacts);
```

Come sappiamo, il filtro `json` non accetta argomenti. Tuttavia il primo argomento di tutte le funzioni dei filtri è in realtà l'input. Non lo vediamo quando le chiamiamo dalla vista poiché Angular opera la sua magia dietro le quinte per semplificare le cose.

Se racchiudete l'espressione precedente in una `console.log`, vedrete che l'output è identico a quello nella vista utilizzando lo stesso filtro:



In alternativa, se non volete ricorrere al servizio `$filter`, potete inserire ogni filtro separatamente come un servizio. Nel pattern vengono chiamati `filternameFilter`. Per il nostro esempio, dobbiamo inserire `jsonFilter`:

```
function AppCtl($scope, jsonFilter){
…
}
```

Può essere utilizzato allo stesso modo della funzione restituita dal servizio `$filter`, così da permetterci di passare l’oggetto da filtrare:

```
jsonFilter($scope.contacts);
```

Ora che sappiamo come utilizzare i filtri dal controller, vediamo come crearne uno.

Sviluppare un filtro

Come abbiamo visto, il filtro `limitTo` è utilissimo per troncare stringhe di testo. Ho sempre sentito la necessità di un filtro che aggiungesse dei puntini di sospensione nel caso in cui una stringa superasse il limite.

Fortunatamente Angular ci consente di estendere i filtri inclusi e svilupparne uno nuovo.

Moduli

A questo scopo dobbiamo creare ciò che viene chiamato *modulo*. Esso ci consente di associare un filtro e utilizzarlo nelle viste o nei controller. La documentazione di AngularJS spiega perfettamente cos'è un modulo:

> Si può pensare a un modulo come a un contenitore per le diverse parti di un'app: controller, servizi, filtri, direttive e così via.

Ma perché dovremmo utilizzarne uno? Ci sono un paio di motivi. La ragione più importante che giustifica la loro esistenza è la facilità con cui è possibile creare codice riutilizzabile.

Immaginate di star lavorando su una piattaforma di blogging. Potreste sviluppare un modulo destinato a un media browser o a un media uploader. Conterrebbe una serie di controller, servizi e filtri, tutti insieme. Se voleste utilizzare questo media browser in un altro progetto, potreste semplicemente copiare il modulo.

Esistono altri motivi per utilizzare i moduli. Se si collauda l'unità, è sufficiente che i test carichino solo i moduli pertinenti per assicurare la rapidità dell'operazione, e il codice diventa più semplice da comprendere e seguire perché ogni componente è ben inserito tra gli altri elementi.

Creare un modulo

È molto facile creare un modulo; ci consente di ampliare il core perché Angular non ammette i filtri personalizzati senza la creazione di un modulo. In questo caso realizziamo un nuovo modulo chiamato `contactsMgr`. Il secondo argomento è un semplice array vuoto. Possono esserci quanti moduli desideriamo e li possiamo includere come dipendenze, ma per il momento questo lo lasceremo vuoto:

```
angular.module('contactsMgr', []);
```

Tuttavia è necessario inserire una piccola modifica nel modo in cui viene aggiunto il controller. Attualmente è solo una funzione, ma è necessario aggiungerlo al modulo perché Angular lo possa recuperare:

```
angular.module(‘contactsMgr’, [])
.controller(‘AppCtl’, function($scope, jsonFilter){
…
});
```

Possiamo concatenare i controller nel modulo. Avrete notato che è necessario utilizzare il metodo `controller`. Il primo argomento è il nome del controller, mentre il secondo è la funzione di callback con i servizi inseriti.

Se ora caricate l’app, vedrete che nulla funziona come previsto e non viene trovata la funzione del controller. Questo perché non abbiamo indicato ad Angular quale modulo desideriamo utilizzare. È necessario aggiungere il nome del modulo alla direttiva `ng-app`:

```
<html lang=”en” ng-app=”contactsMgr” ng-controller=”AppCtl”>
```

Una volta inserito, tutto dovrebbe iniziare a funzionare per il verso giusto. Ora stiamo utilizzando il modulo appena creato.

Creare un filtro

Dopo aver creato il modulo e averlo visto in azione, possiamo lavorare sul filtro `limitTo` migliorandolo. Conviene pensare innanzitutto alle operazioni che vorremo svolgesse. Possiamo suddividere le funzioni in pochi brevi passi.

- Considera l’input con un solo argomento per il limite.
- Verifica la lunghezza dell’input rispetto al limite.
- Se l’input è maggiore del limite, tronca la stringa e aggiunge dei puntini di sospensione.
- Altrimenti restituisce l’input.

Operando con i modelli, la creazione di un filtro risulta molto simile alla creazione di un controller:

```
.filter(‘truncate’, function(){
});
```

Proprio come abbiamo fatto quando abbiamo spostato il controller su un nuovo modulo, utilizziamo un nuovo metodo che accetta due argomenti: il nome del filtro e

una funzione di callback. Come visto quando abbiamo applicato i filtri dal controller, quando si chiama un filtro, esso restituisce una seconda funzione, che dobbiamo aggiungere qui:

```
.filter('truncate', function(){
    return function(){
    };
})
```

Abbiamo anche scoperto che il primo argomento di un filtro è sempre l'input o i dati che filtreremo. All'interno di questa funzione possiamo anche includere argomenti aggiuntivi. Nel caso di `truncate`, occorre un argomento per indicare al filtro a quanti caratteri dovrebbe limitare la stringa:

```
.filter('truncate', function(){
    return function(input, limit){
    };
})
```

La costruzione del filtro è completa e ora possiamo utilizzarlo nello stesso modo degli altri filtri esaminati in precedenza. Ovviamente non abbiamo impostato alcuna logica e non viene restituito nulla dalla funzione filtro; pertanto non verrà visualizzato nulla sulla pagina.

Ora dobbiamo verificare la lunghezza, troncare la stringa e aggiungere dei puntini di sospensione. Si possono ottenere questi risultati in una stringa grazie a un'istruzione ternaria:

```
return (input.length > limit) ? input.substr(0, limit)+'…' :
  input;
```

Verifichiamo la lunghezza della stringa, e se è superiore al limite, la tagliamo e aggiungiamo dei puntini di sospensione. Se non soddisfa la nostra condizione, verrà restituito l'input originario. Questo è importante perché Angular non visualizzerà nulla se non viene restituito nulla dal filtro.

Assembliamo il tutto ed esaminiamo la funzione completa:

```
.filter('truncate', function(){
  return function(input, limit){
    return (input.length > limit) ? input.substr(0, limit)+'…'
      : input;
  };
})
```

Possiamo utilizzare il nuovo filtro nello stesso modo del filtro `limitTo` integrato; sostituiamolo e osserviamo il risultato:

```
{{'Lorem ipsum dolor sit amet' | truncate:15}}
```

Come previsto, ora l'output include dei puntini di sospensione, mentre in precedenza la stringa era troncata dopo il limite.

Quiz

1. Come si può applicare un filtro dalla vista?
2. In che modo passiamo gli argomenti al filtro dalla vista?
3. Quale filtro dovremmo utilizzare per creare una *live search*?
4. Come possiamo utilizzare un filtro dal controller?
5. Che cosa dobbiamo creare prima di sviluppare un filtro?

Riepilogo

Ora dovreste conoscere la funzione e la grande utilità dei filtri, ma riepiloghiamo tutto ciò che abbiamo affrontato in questo capitolo.

Abbiamo iniziato a osservare come viene applicato un filtro direttamente dalla vista utilizzando la sintassi che prevede l'uso del pipe e la separazione degli argomenti con due punti. Dopo aver analizzato le basi, abbiamo esaminato i numerosi filtri inclusi.

Alcuni sono fondamentali e non richiedono alcun argomento, ma ce ne sono anche di più avanzati che consentono di ordinare o filtrare un array di oggetti.

Oltre ad applicare i filtri dalla vista, abbiamo anche considerato i due metodi per filtrare dal controller. Possiamo utilizzare il servizio incluso `$filter` o decidere di inserire i filtri separatamente.

Infine abbiamo visto come estendere Angular per creare un filtro al fine di tagliare una stringa di testo. Prima però abbiamo visto come creare un modulo che contenesse i filtri e i controller. Dopo averlo realizzato, siamo riusciti a creare un filtro e a utilizzarlo nello stesso modo di quelli integrati.

Abbiamo affrontato molti paradigmi e concetti fondamentali di Angular. Nel prossimo capitolo vedremo come impostare il sistema di routing per gestire molteplici viste e controller per l'app di gestione dei contatti.

Routing

Tutte le web app sono costituite da più pagine o viste, e Angular ha tutte le carte in regola per gestire questo aspetto con il router (inteso come sistema di routing, e non come dispositivo fisico). Forse conoscete il sistema di routing nei framework lato server, come Ruby on Rails o Laravel. Angular è interamente lato client, e tutta la magia accade all'interno del file HTML al quale punta il browser. In questo capitolo esamineremo come creare route statiche contenenti dei parametri. Individueremo inoltre alcune insidie nelle quali potreste imbattervi.

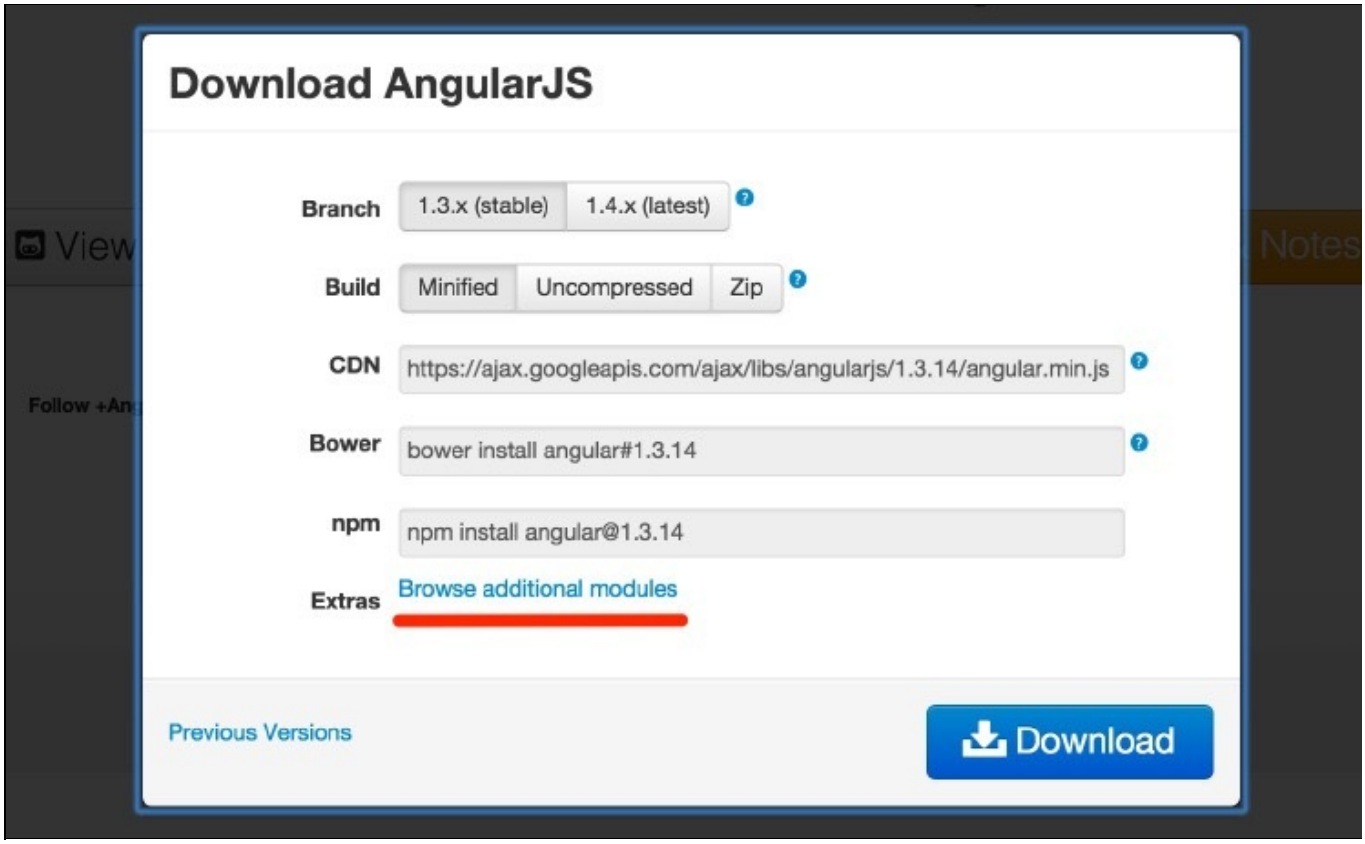
Prima di iniziare elenchiamo le route che occorreranno per l'app di gestione dei contatti.

- *Index*: sarà la pagina principale, che elencherà tutti i contatti in una tabella.
- *View Contact*: qui potremo visualizzare il contatto nel dettaglio e modificare qualunque informazione correlata.
- *Add Contact*: includerà un form che ci consentirà di aggiungere un contatto al gestore.

Queste sono le route fondamentali; vediamo come crearle.

Installare ngRoute

Fin da AngularJS 1.2, il router si è presentato come un modulo separato ed esterno rispetto ai componenti fondamentali di Angular. Il file che ci occorre, `angular-route.min.js`, può essere scaricato dal sito web di Angular nella sezione *Extras* all'interno della finestra di download.



Dopo averlo scaricato, trascinatelo nella directory *js* del progetto e includetelo nella pagina dopo AngularJS:

```
<script src="assets/js/angular-route.min.js"></script>
```

Inoltre è necessario comunicare al modulo che intendiamo utilizzare il router; dobbiamo quindi aggiungerlo alla lista delle dipendenze del modulo. Possono essere presenti quante dipendenze desideriamo; per ora è sufficiente includere `ngRoute`:

```
angular.module('contactsMgr', ['ngRoute'])
```

Creare route di base

Come sappiamo, per configurare il router all'interno di AngularJS, è necessario un modulo. Nel Capitolo 3 ne abbiamo creato uno al fine di sviluppare un filtro personalizzato. Possiamo utilizzare questo stesso modulo per le route.

Le route vengono create nel metodo `config` del modulo dell'applicazione:

```
angular.module('contactsMgr', ['ngRoute'])
.config(function($routeProvider){

})
```

Il metodo accetta una funzione anonima in cui inserire il servizio `$routeProvider` necessario, che presenta soltanto due metodi: `when` e `otherwise`. Per aggiungere una route, utilizziamo il metodo `when`, che accetta due parametri: il percorso della stringa e le opzioni della route in quanto oggetto:

```
angular.module('contactsMgr', ['ngRoute'])
.config(function($routeProvider){
    $routeProvider.when('/', {});
})
```

Nell'oggetto delle opzioni della route sono presenti due proprietà che ci interessano: `controller` e `templateUrl`. La proprietà `controller` chiama un costruttore `controller` esistente o ne definisce uno nuovo tramite una funzione anonima. La proprietà `templateUrl` consente di definire il percorso a un file HTML che ospiterà l'intero markup della vista. In alternativa potremmo definire il template direttamente all'interno dell'oggetto route. Tuttavia, così facendo, la situazione si potrebbe ben presto complicare; questa soluzione è consigliabile solo per template contenenti una o due righe.

Esaminiamo la route che definiremo per la pagina `index`:

```
$routeProvider.when('/', {
    controller: 'indexCtl',
    templateUrl: 'assets/partials/index.html'
});
```

Il percorso del template è relativo al file HTML di base, quindi include la directory *assets*. Possiamo procedere e creare il template HTML. In Angular questi elementi vengono chiamati *partial* e li utilizzeremo per tutte le viste.

L'argomento `controller` all'interno della route è facoltativo, ma l'abbiamo incluso perché ci servirà per l'applicazione. Creiamo il `controller` che ci consente di sviluppare modelli e funzioni soltanto per la vista `index`.

Nel file `controller.js` possiamo concatenarlo alla fine:

```
.controller('indexCtrl', function($scope){

});
```

Aggiungiamo la seconda route con il metodo `config`. Ospiterà il form per aggiungere i contatti:

```
$routeProvider.when('/', {
    controller: 'indexCtl',
    templateUrl: 'assets/partials/index.html'
})

.when('/add-contact', {
    controller: 'addCtl',
    templateUrl: 'assets/partials/add.html'
});
```

Come nel caso dei controller, possiamo concatenare le route. Non ci resta che creare i `controller` e i `partial` opportuni:

```
.controller('addCtl', function($scope){

});
```

L'ultima operazione che è necessario svolgere prima che Angular renda operativo il router consiste nell'includere nella pagina la direttiva `ng-view`, che inserisce il partial definito nella route:

```
<div class="container">
    <ng-view></ng-view>
</div>
```

NOTA

È possibile includere `ng-view` soltanto una volta per pagina.

Si può includere questa direttiva come suo elemento. Qui abbiamo preferito includerla come un elemento nel file `index.html` all'interno della radice. Se è presente qualcosa nel contenitore, cancellatelo e sostituitelo con `ng-view`.

Se aprite il progetto nel browser, vedrete che la route è stata aggiunta all'URL ed è preceduta dal simbolo `#`. Purtroppo se userete Chrome, probabilmente i partial non riusciranno a caricarsi. Se aprite la console, vedrete un errore simile al seguente:

```
Cross origin requests are only supported for HTTP.
```

Esistono due soluzioni. Possiamo caricare il codice su un server web oppure, con Chrome, possiamo avviare il browser utilizzando un flag per consentire richieste cross-origin per il protocollo `file://` in ambiente OS X o per `c:/` in ambiente Windows.

In OS X, eseguite questo comando nel terminale:

```
open -a 'Google Chrome' --args -allow-file-access-from-files
```

In altri sistemi basati su Unix eseguite il seguente:

```
google-chrome  --allow-file-access-from-files
```

In Windows, è necessario modificare il collegamento sul desktop per aggiungere un flag alla fine della destinazione:

```
C:\ ... \Application\chrome.exe --allow-file-access-from-files
```

Se non volete eseguire Chrome con un flag, potete far girare l’app di gestione dei contatti su un server web. Potreste servirvi di quello integrato in Python o PHP, o di un’app vera e propria come MAMP o WAMP.

Modificate la directory del progetto ed eseguite il seguente comando per testare l’applicazione su un server web Python:

```
python -m SimpleHTTPServer 8000
```

Potete accedere a `localhost:8000` nel browser per visualizzare l’app. In alternativa, se preferite eseguire un server web integrato in PHP, potete farlo nel seguente modo:

```
php -S localhost:8000
```

Route con parametri

Dopo aver impostato più route, dobbiamo capire come includere i parametri al loro interno. Questo è un aspetto importante per consentire un certo dinamismo all'interno dell'app di gestione dei contatti; per esempio, li utilizzeremo per visualizzare un contatto specifico facendo riferimento a un ID numerico o a un indice.

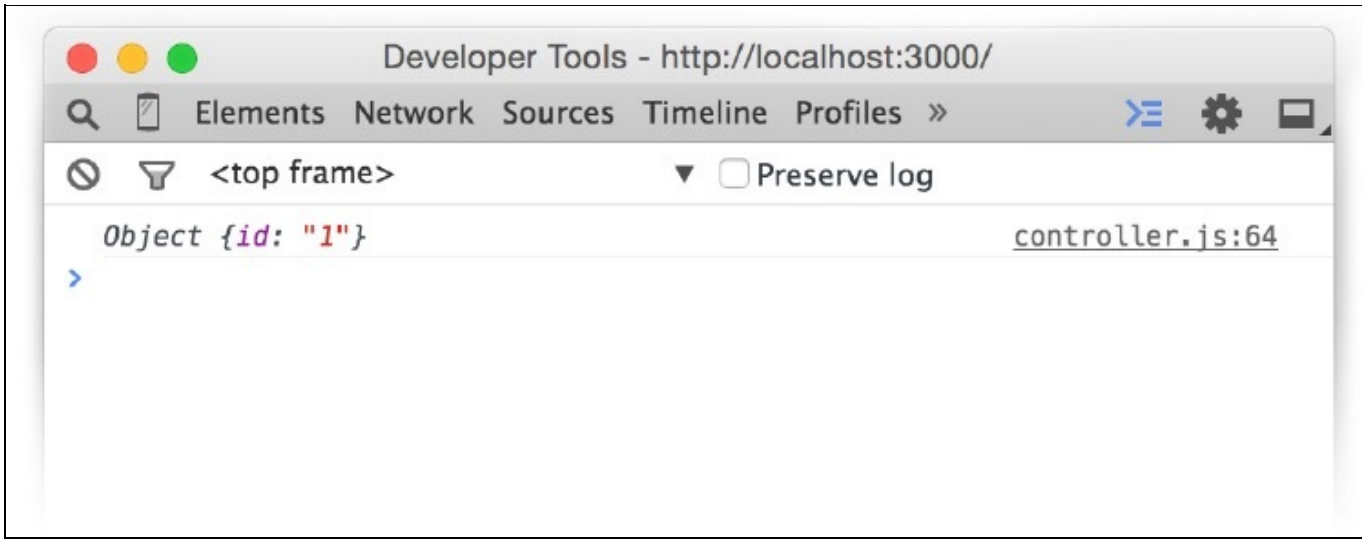
È facile inserirli; è sufficiente aggiungere un segnaposto con due punti seguiti dal nome del parametro che intendiamo creare. Osserviamo la route che realizzeremo per visualizzare il contatto. Possiamo di nuovo concatenarla alle route già esistenti:

```
.when('contact/:id', {
    controller: 'contactCtl',
      templateUrl: 'assets/partials/contact.html'
});
```

Possiamo aggiungere facilmente nel controller quanti parametri desideriamo. Sarà sufficiente inserire un servizio e potremo accedere a tutti i parametri della route come oggetti:

```
.controller('contactCtl', function($scope, $routeParams){
    console.log($routeParams);
});
```

Se accedete a `localhost:8000/#/contact/1` e aprite la console, vedrete i parametri della route registrati come oggetto JS.



Ciò significa che possiamo accedere a qualsiasi proprietà dell'oggetto tramite la sintassi standard:

```
$routeParams.id;
```

Route di fallback

L’ultima route da configurare è quella che verrà visualizzata quando non si trova nessuna route corrispondente. Potreste creare una pagina 404, ma vediamo come è possibile reindirizzare una route invece di visualizzare un template.

Per creare una route di fallback, utilizziamo il secondo metodo che ci offre il servizio `$routeProvider`: `otherwise`:

```
.otherwise({
    redirectTo: ‘/’
});
```

In questo modo, se la route richiesta non corrisponde a nessuna di quelle definite nel router, Angular ci riporta alla pagina dell’index.

Routing in HTML5 o eliminazione del simbolo

Ora abbiamo configurato tutte le route fondamentali e possiamo accedere a partial distinti per ognuna. È un ottimo risultato, anche se le route che seguono il simbolo `#` nell'URL non ci piacciono. Fortunatamente esiste un sistema facile per rimediare, attivando ciò che Angular definisce `html5Mode`.

Questa modalità consente ad Angular di sfruttare il `pushState` nei browser moderni fornendo al contenuto un fallback per i browser legacy, come IE 8.

Abilitare HTML5Mode

Per consentire questa nuova modalità, riesaminiamo il metodo `config`. Come abbiamo fatto in precedenza, dobbiamo inserirvi un servizio:

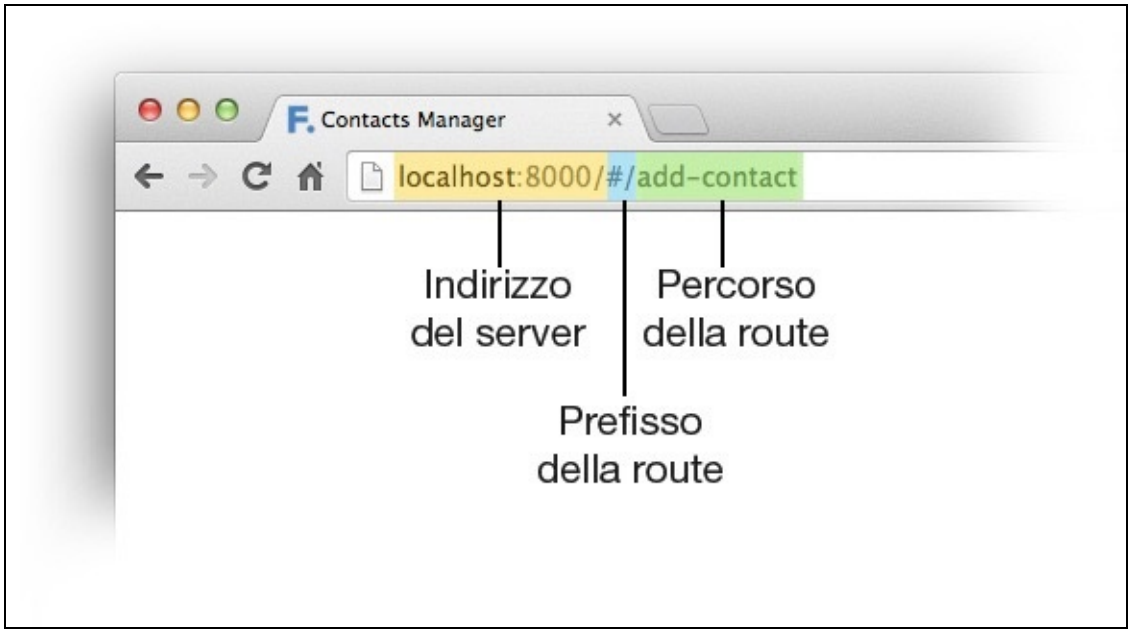
```
.config(function($routeProvider, $locationProvider){
    ...
    $locationProvider.html5Mode(true);
})
```

Ora abbiamo inserito un secondo servizio: `$locationProvider`, che ci consente di sfruttare il metodo `html5Mode`, che accetta un valore booleano per attivarlo o disattivarlo.

Il servizio offre un secondo metodo, e anche se non ne approfitteremo nel corso dello sviluppo dell'app, è opportuno conoscerlo. Il metodo `hashPrefix` permette di aggiungere nell'URL un prefisso dopo il simbolo `#`. Per esempio, potremmo aggiungere un punto esclamativo e trasformare il prefisso in un hashbang (`#!`):

```
$locationProvider.hashPrefix('!');
```

La prossima figura mostra l'URL dell'applicazione e suddivide l'indirizzo nelle diverse sezioni della route.


Contacts Manager
localhost:8000/#/add-contact
Indirizzo
del server
Prefisso
della route
Percorso
della route

Collegare le route

Collegare le route non è diverso da collegare le pagine su un sito web. Utilizziamo un tag àncora e, invece di collegare la pagina, colleghiamo la route.

Per esempio, se volessimo collegare il pulsante *Add Contact* nella barra di navigazione, dovremmo immettere questo codice:

```
<a href=”/add-contact”>Add Contact</a>
```

Al clic sul link, Angular visualizzerebbe automaticamente il partial corretto e cambierebbe anche l’URL. Se avete deciso di non utilizzare `html5Mode`, potete comunque effettuare il collegamento mediante un’àncora, ma l’attributo `href` sarà leggermente diverso, perché è necessario aggiungere il simbolo del cancelletto:

```
<a href=”#/add-contact”>Add Contact</a>
```

Quiz

1. Quale file o modulo dobbiamo includere per consentire il routing?
2. Quale metodo utilizziamo per creare le route?
3. Che cosa dobbiamo inserire nel metodo per riuscire a creare una route?
4. In che modo creiamo una route?
5. Che cosa possiamo utilizzare quando nessuna route corrisponde al percorso corrente?
6. In che modo possiamo eliminare il simbolo # nell'URL?

Riepilogo

In questo capitolo abbiamo trasformato l'applicazione, e da un'app con un'unica pagina siamo passati a una multipagina e multiroute nella quale svilupperemo il nostro gestore di contatti. Abbiamo iniziato a progettare le route fondamentali dell'app prima di installare il modulo necessario.

Abbiamo visto come è possibile utilizzare nel modulo il metodo `config` per impostare le route, e per farlo abbiamo inserito il servizio `$routeProvider` e utilizzato i metodi `when` e `other`. Così abbiamo impostato route statiche e dinamiche contenenti parametri.

Infine abbiamo analizzato come eliminare il simbolo `#` dall'URL mediante il `pushState` di HTML5 e collegare entrambi i tipi di route. Nel prossimo capitolo popoleremo i partial con i layout che svilupperemo grazie a Bootstrap.

Le viste

Nel Capitolo 4 abbiamo visto come è possibile trasformare l’applicazione in una web app con più route e più viste. Abbiamo sfruttato il router di Angular e impostato i partial per tutte le viste fondamentali. Ora è il momento di creare le viste mediante Bootstrap così da poter popolare l’app con i dati. Analizziamo uno per uno i partial.

Popolare la vista Index

La vista Index è ciò che viene visualizzato quando apriamo l'app. Forse conviene elencare qui tutti i contatti; infatti avremo bisogno di un rapido accesso alle informazioni memorizzate.

Una tabella può essere un'opzione efficace, ma è necessario riflettere su ciò che verrà memorizzato nel gestore dei contatti. Ecco una lista dei possibili item:

- Nome
- Indirizzo e-mail
- Numero di telefono
- Indirizzo
- Sito web
- Note
- Gravatar (un servizio di avatar riconosciuto a livello mondiale sviluppato dai creatori di WordPress)

Non occorre visualizzare tutte queste informazioni nella vista Index. Non dimenticate che sarà anche presente l'opzione per scorrere i contatti e quindi avremo la possibilità di visualizzare più informazioni.

Una scelta ragionevole sembra quella di visualizzare nome, indirizzo e-mail e numero di telefono in una tabella con un link su cui fare clic.

Aprite il partial della vista Index; si trova in `assets/partials/index.html`. Per ora questo file è vuoto; aggiungete all'inizio l'header della pagina:

```
<div class=”page-header”>
  <h1>All Contacts</h1>
</div>
```

Non dobbiamo racchiuderlo in un contenitore poiché il partial è nidificato nel file principale dell'app `index.html` sulla route e vi abbiamo già incluso il contenitore:

```
<table>
  <thead>
    <tr>
      <th>Name</th>
      <th>Email Address</th>
      <th>Phone Number</th>
      <th>Actions</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Karan Bromwich</td>
      <td>karan@example.com</td>
      <td>01234 56734</td>
      <td><a href=”#”>View</a></td>
    </tr>
```

```
    <tr>
      <td>Declan Proud</td>
      <td>declan@example.com</td>
      <td>01234 567890</td>
      <td><a href=”#”>View</a></td>
    </tr>
  </tbody>
</table>
```

È una struttura funzionale, anche se la pagina non ha un bell’aspetto. Come la maggior parte dei componenti, Bootstrap permette di applicare gli stili alle tabelle, anche se è necessario includere una classe aggiuntiva per renderli disponibili. Aggiungete la classe `table` al tag di apertura della tabella e Bootstrap metterà subito un po’ d’ordine aggiungendo i bordi necessari e facendo in modo che occupi l’intera larghezza.

Sono presenti inoltre alcune classi secondarie che è possibile includere per vivacizzare un po’ la tabella.

- `table-bordered`: aggiunge un bordo attorno a tutti i lati della tabella e delle celle.
- `table-striped`: aggiunge uno sfondo grigio a righe alternate per facilitare la lettura.
- `table-hover`: modifica lo sfondo della riga all’hover.
- `table-condensed`: elimina una parte del padding superiore e inferiore in modo da ridurre l’altezza della tabella.

Oltre a queste classi, ne esistono altre che è possibile applicare alle righe o alle celle e che colorano lo sfondo delle righe per contestualizzarle.

- `active`: aggiunge alla riga lo stato hover.
- `success`: colora lo sfondo di verde, per indicare che un’azione ha avuto successo.
- `info`: attira l’attenzione su una riga o su una cella segnalando un’informazione.
- `warning`: indica che può essere necessaria un’azione e colora la cella di giallo.
- `danger`: richiama l’attenzione su un errore o su un problema.

Per il momento aggiungeremo la classe `table-striped`, ma sta a voi sperimentare le altre classi integrate.

La tabella inizia ad avere un bell’aspetto. Vedrete però che su uno schermo di dimensioni più piccole risulta tagliata orizzontalmente. Per rimediare è necessario racchiuderla in un altro elemento che ci permetta di effettuare lo scorrimento su schermi più piccoli:

```
<div class=”table-responsive”>
…
</div>
```

Ora ha un aspetto nettamente migliore poiché il contenuto non viene più tagliato e il layout responsive non si interrompe. L'ultima operazione che faremo sulla tabella è trasformare il link della vista in un pulsante. Bootstrap offre numerosi stili per i pulsanti di cui possiamo avvalerci.

Tutti i pulsanti sono combinazioni delle classi relative:

- al pulsante predefinito;
- al contesto;
- alle dimensioni.

Tutte insieme offrono il controllo che ci occorre al fine di scegliere il pulsante giusto per l'occasione giusta. Combiniamole per creare un pulsante che ben si integri nella tabella:

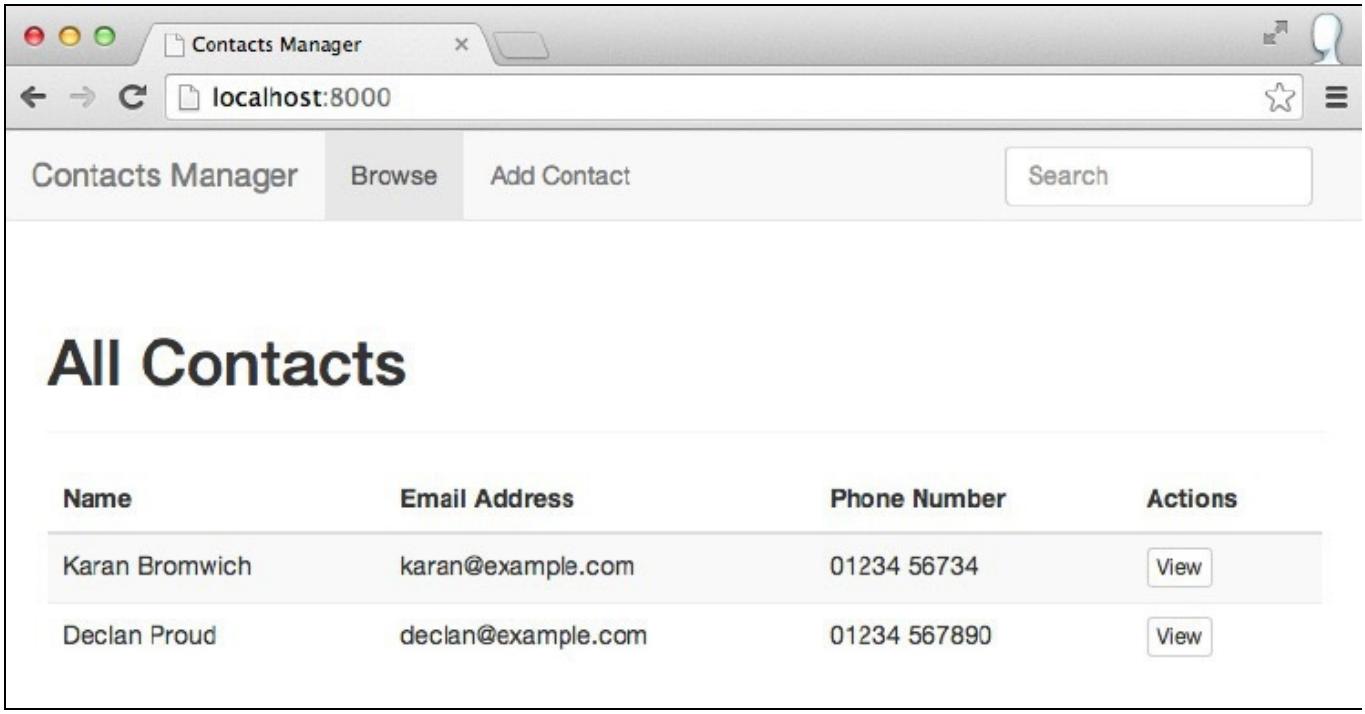
```
<a href="#" class="btn btn-default btn-xs">View</a>
```

La prima classe applica alcuni stili predefiniti ai pulsanti; la seconda assegna il colore (in questo caso, quello predefinito è il bianco) e l'ultima definisce le dimensioni. In alternativa potremmo utilizzare una tra le classi descritte nella prossima tabella per modificare il colore del pulsante.

Nome della classe	Descrizione
`btn-default`	Pulsante bianco con un bordo grigio
`btn-primary`	Pulsante blu
`btn-success`	Pulsante verde
`btn-info`	Pulsante azzurro
`btn-warning`	Pulsante arancione
`btn-danger`	Pulsante rosso
`btn-link`	Pulsante con lo stile di un link

Oltre a impostare le dimensioni predefinite, altre tre classi le modificano. Nel codice precedente abbiamo già utilizzato `btn-xs` per rimpicciolire molto il pulsante, ma potremmo utilizzare anche `btn-sm` per fare in modo che appaia un po' più piccolo di quello predefinito o `btn-lg` per ingrandirlo.

La vista Index è completa ed è pronta per essere popolata. Nell'immagine seguente osserviamo il risultato.


Contacts Manager
localhost:8000
Contacts Manager
Browse
Add Contact
Search
All Contacts
Name
Email Address
Phone Number
Actions
Karan Bromwich
karan@example.com
01234 56734
View
Declan Proud
declan@example.com
01234 567890
View

Popolare la vista Add Contact

È chiaro ciò di cui avremo bisogno nella vista *Add Contact*: un form che ci consenta di digitare le informazioni richieste. Fortunatamente Bootstrap ci offre molto controllo nell'organizzazione dei campi. Abbiamo già elaborato i dati che memorizzeremo; ora si tratta solo di individuare qual è il tipo di campo migliore:

- *Name*: campo di testo
- *Email address*: campo e-mail
- *Phone number*: campo per il numero di telefono
- *Address*: area di testo
- *Website*: campo di testo
- *Notes*: area di testo
- *Gravatar*: N/D

Siccome *Gravatar* usa l'indirizzo e-mail per fornire l'immagine, qui non dobbiamo richiedere alcuna informazione aggiuntiva. Abbiamo in tutto sei campi; quindi due colonne sono sufficienti.

Innanzitutto apriamo il form e aggiungiamo al suo interno le colonne. Abbiamo già la classe del contenitore; ci occorrono una nuova riga e le due colonne. Come abbiamo visto nel Capitolo 2, il sistema a griglia di Bootstrap contiene 12 colonne; ricordiamocelo quando creeremo il layout:

```
<form>
  <div class="row">
    <div class="col-sm-6">

    </div>
    <div class="col-sm-6">

    </div>
  </div>
</form>
```

Come abbiamo fatto in precedenza, utilizziamo il prefisso `col-sm` per fare in modo che le colonne si comprimano sui tablet o sui dispositivi mobili più piccoli.

Potremmo inserire le etichette e gli input direttamente nelle colonne, ma per una spaziatura ottimale, è necessario racchiudere gli elementi in un tag `div` con la classe `form-group`:

```
<div class="form-group">
  <label for="name">Name</label>
  <input type="text" id="name">
</div>
```

Per sfruttare gli stili di Bootstrap negli input, aggiungiamo la classe `form-control`. Se aggiungessimo la classe `control-label` all'etichetta, applicherebbe un po' di padding aggiuntivo:

```
<div class="form-group">
  <label for="name" class="control-label">Name</label>
  <input type="text" id="name" class="form-control">
</div>
```

Aggiungiamo all'interno gli elementi restanti: il nome, il numero di telefono e l'indirizzo nella colonna di sinistra e l'indirizzo e-mail, il sito web e le note in quella di destra.

Form orizzontali

Il form presenta molto bene. Se non vi piacciono le etichette in alto, potete collocarle a sinistra con una leggera modifica. Aggiungendo la classe `form-horizontal` al tag di apertura del form, le classi `form-group` si comportano come le righe di una griglia; pertanto possiamo applicare le classi delle colonne agli elementi al loro interno. Ecco il codice d'esempio:

```
<div class="form-group">
  <label for="name" class="col-sm-4 control-label">Name</label>
  <div class="col-sm-8">
    <input type="text" id="name" class="form-control">
  </div>
</div>
```

Dopo aver incluso la classe `form-horizontal`, vedrete che è possibile aggiungere all'etichetta le classi applicabili alle colonne di Bootstrap. Siccome `form-control` imposta la larghezza su 100%, si adatta all'elemento genitore che dobbiamo racchiudere in un elemento aggiuntivo. Abbiamo incluso anche la classe `control-label` e quindi l'etichetta è centrata verticalmente.

Il form appare molto meno disordinato grazie alla classe `form-horizontal`; proseguiamo e racchiudiamo tutti gli input nell'elemento `form-control`.

Talvolta vorrete fornire all'utente maggiori informazioni di quelle strettamente necessarie. Le potete includere sotto l'input corrispondente utilizzando un tag `span` associato alla classe `help-block`:

```
<div class="form-group">
  <label for="notes" class="col-sm-4 control-label">Notes</label>
  <div class="col-sm-8">
    <textarea id="notes" class="form-control"></textarea>
    <span class="help-block">Any additional information about the
  contact.</span>
  </div>
</div>
```

Non ci resta che aggiungere un pulsante di invio. Nelle versioni precedenti di Bootstrap, era racchiuso in un elemento con la classe `form-actions`. In Bootstrap 3, invece, è sufficiente utilizzare lo stesso `form-group` che abbiamo usato finora. Se applicherete lo stile `form-horizontal`, dovrete distanziare le colonne:

```
<div class=”form-group”>
  <div class=”col-sm-offset-4 col-sm-8”>
    <button class=”btn btn-primary”>Add Contact</button>
  </div>
</div>
```

Poiché l’etichetta occupa quattro colonne, è necessario staccare il pulsante dello stesso spazio in modo che sia ben allineato.

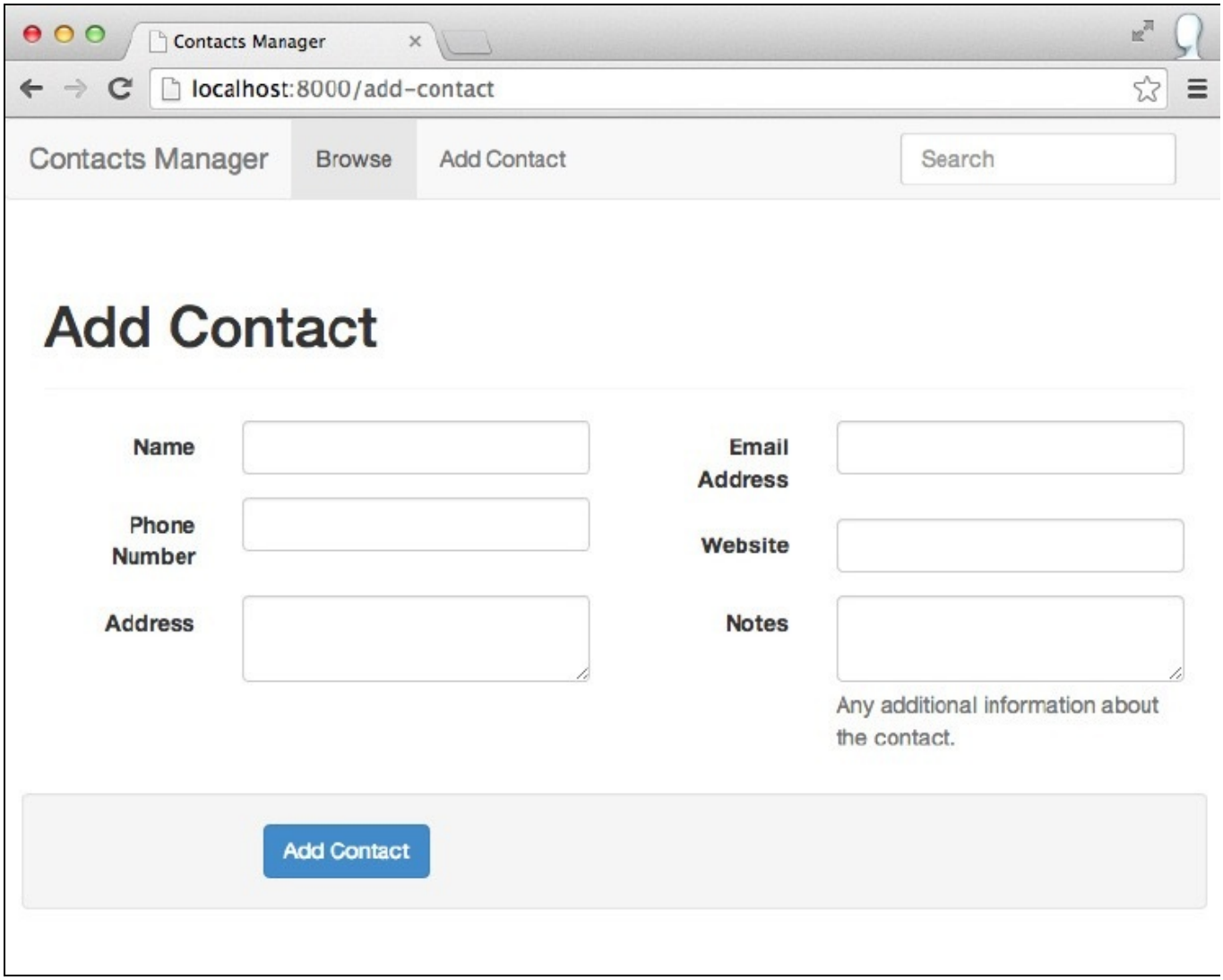
A me piaceva il contrasto che offriva la vecchia classe `form-actions`. Fortunatamente possiamo ottenere un risultato simile ricorrendo al componente `well` di Bootstrap. In questo caso abbiamo spostato la classe `form-group` che contiene il pulsante *submit* subito sotto la riga già esistente (ricordatevi che `form-horizontal` fa sì che tutti i gruppi si comportino come righe) e abbiamo aggiunto la classe `well`:

```
<div class=”form-group well”>
  <div class=”col-sm-offset-2 col-sm-10”>
    <input type=”submit” class=”btn btn-primary” value=”Add
      Contact”>
  </div>
</div>
```

Infine, per completare la pagina, assegneremo lo stesso elemento `page-header` che abbiamo incluso nella vista Index e lo collocheremo subito all’inizio del markup:

```
<div class=”page-header”>
  <h1>Add Contact</h1>
</div>
```

Il risultato finale sarà il seguente.


Contacts Manager
localhost:8000/add-contact
Contacts Manager
Browse
Add Contact
Search
Add Contact
Name
Email Address
Phone Number
Website
Address
Notes
Any additional information about the contact.
Add Contact

Popolare la vista View Contact

L’ultimo partial che dobbiamo popolare è la schermata in cui verrà visualizzato il contatto. Siamo stati tentati di inserirlo come un form, ma ci piace l’idea di avere del testo statico, che possiamo decidere di modificare in alcune parti.

Dovremo visualizzare le stesse informazioni che abbiamo immesso nella vista *Add Contact*, oltre al gravatar.

Titolo e gravatar

Innanzitutto includeremo una classe `page-header`. Ospiterà un tag `h1` con all’interno il nome del contatto:

```
<div class=”page-header”>
  <h1>Declan Proud</h1>
</div>
```

Qui includeremo anche il gravatar: vediamo come. Per ora utilizzeremo alcune immagini segnaposto provenienti da `http://placehold.it`. Forse non conoscete questo sito: qui trovate segnaposto di tutte le dimensioni. Abbiamo bisogno di un’immagine 50 px × 50 px, che inseriremo mediante il codice seguente:

```
<img src=”http://placehold.it/50x50”>
```

Modificatene pure le dimensioni a vostro piacimento. Potete inserirla subito prima del nome del contatto nel tag `h1`. Vogliamo anche aggiungere la classe `img-circle`:

```
<div class=”page-header row”>
  <h1><img src=”http://placehold.it/50x50” class=”img-circle”>
  Declan Proud</h1>
</div>
```

Questa classe è una delle tre disponibili per applicare un po’ di stile alle immagini; aggiunge un arrotondamento degli angoli pari al 50% per creare un cerchio. È anche disponibile `img-rounded`, che arrotonda gli angoli, oltre a `img-thumbnail`, che aggiunge un bel bordo doppio.

La classe form-horizontal

Siamo fortunati perché possiamo riutilizzare buona parte di quello che abbiamo fatto nella vista *Add Contact*. La classe `form-horizontal` si comporterà altrettanto bene se è presente del contenuto statico al posto dei campi. Questa pagina diventerà in

seguito la schermata di editing, oltre a essere la scheda del contatto; pertanto è utile poter utilizzare la classe per entrambe le viste.

Questa volta però ricorreremo a un tag `div` e non a un elemento del form per racchiudere il layout con due colonne:

```
<div class=”form-horizontal”>
  <div class=”row”>
    <div class=”col-sm-6”>
      ...
    </div>
    <div class=”col-sm-6”>
    ...
    </div>
  </div>
</div>
```

A parte questa piccola modifica, il layout è identico. Abbiamo la stessa riga e le sei colonne della vista create in precedenza.

Possiamo anche utilizzare le classi `form-group` oltre alle classi `control-label`, che ci consentono di strutturare efficacemente l’etichetta senza usare una lista o una tabella, come possiamo vedere di seguito:

```
<div class=”form-group”>
  <label for=”name” class=”col-sm-4 control-label”>Name</label>
  <div class=”col-sm-8”>
    <p>Declan Proud</p>
  </div>
</div>
```

Tuttavia se lo carichiamo nel browser, vedrete che l’etichetta e il contatto non sono ben allineati. Per rimediare possiamo includere un’altra classe nel tag `paragraph`:

```
<p class=”form-control-static”>Declan Proud</p>
```

Aggiungetela in tutti i campi. Abbiamo ripreso lo stesso layout precedente con nome, numero di telefono e indirizzo nella colonna di sinistra e indirizzo e-mail e sito web e note in quella di destra.

Siamo soddisfatti di questo layout. Tuttavia, quando lo ridimensioniamo, sembra che rimanga molto spazio bianco a destra. Sarebbe preferibile modificare le colonne principali in modo che siano visibili in questo layout anche su dispositivi mobili.

Nella riga sostituiamo la classe `col-sm-6` con la seguente:

```
<div class=”col-xs-6”>
…
</div>
```

In questo modo otterremo due colonne su dispositivi più piccoli ed eviteremo il problema dello spazio bianco in eccesso.

Di seguito potete osservare il risultato:


Contacts Manag
localhost:8000/c...
Contacts Manager
50 x 50
Declan Proud
Name
Declan Proud
Email Address
declan@example.com
Phone Number
0123 4567890
Website
http://declan.com
Address
123, Some Street
Leicester
LE1 2AB
United Kingdom
Notes
Here are some notes about this contact.

Quiz

1. Perché non è necessario includere un contenitore nei partial?
2. Oltre a `table`, quali classi è possibile aggiungere a una tabella per attribuire un po’ di stile?
3. Come possiamo creare un grande pulsante azzurro?
4. In che cosa devono essere racchiusi le etichette e gli input?
5. In che modo la classe `form-horizontal` modifica il form?
6. Che cosa utilizziamo per visualizzare un messaggio di aiuto aggiuntivo per un campo del form?
7. Quali tre classi possiamo applicare alle immagini e quali risultati producono?

Riepilogo

Ora abbiamo tre layout principali completi. Abbiamo ottenuto questo risultato senza scrivere una sola riga di CSS sfruttando gli stili e i componenti fondamentali di Bootstrap.

Nella vista Index abbiamo esaminato come Bootstrap include stili predefiniti e applica le classi secondarie per aggiungere un po' di stile. È un pattern che Bootstrap utilizza sempre, e l'abbiamo visto in azione con i pulsanti e gli input.

Con la vista *Add Contact* abbiamo imparato quali sono i sistemi migliori per organizzare un form e abbiamo stabilito di applicare `form-horizontal`. La potremo riutilizzare quando creeremo la scheda del contatto.

Nel prossimo capitolo impareremo a collegare le viste a dati dinamici. Utilizzeremo AngularJS per creare contatti, passare da uno all'altro e visualizzarli nella tabella, modificarli o cancellarli.

CRUD

Finora abbiamo esplorato i paradigmi di Angular e sviluppato la struttura dell'app grazie a Bootstrap. In questo capitolo ci baseremo sulle idee e i concetti che abbiamo esaminato nel corso del libro per rendere operativa l'app.

CRUD è l'acronimo di *Create, Read, Update* e *Delete* (creare, leggere, aggiornare e cancellare), tutto ciò che serve per sviluppare un'app di gestione dei contatti efficiente.

Analizzeremo ogni lettera di questo acronimo e vedremo come è possibile sfruttare pienamente Angular.

Read

Per il momento ignoreremo la parte creativa e passeremo subito alla lettura dei dati. Ne utilizzeremo di fittizi sotto forma di un array di oggetti nel controller.

Nel prossimo listato vediamo come vengono formattati e ciò che dobbiamo includere.

```
.controller('indexCtl', function($scope){

    $scope.contacts = [
        {
            name: 'Stephen Radford',
            phone: '0123456789',
            address: '123, Some Street\nLeicester\nLE1 2AB',
            email: 'stephen@email.com',
            website: 'stephenradford.me',
            notes: ''
        },
        {
            name: 'Declan Proud',
            phone: '91234859',
            address: '234, Some Street\nLeicester\nLE1 2AB',
            email: 'declan@declan.com',
            website: 'declanproud.me',
            notes: 'Some notes about the contact.'
        }
    ];

})
```

Siccome abbiamo associato l'array allo scope, possiamo accedervi direttamente all'interno della vista. Mediante la direttiva `ng-repeat` esaminata nel Capitolo 2, è possibile avviare il ciclo dell'array per visualizzare i contatti nella tabella:

```
<tr ng-repeat="contact in contacts">
  <td>{{contact.name}}</td>
  <td>{{contact.email}}</td>
  <td>{{contact.phone}}</td>
  <td><a href="/contact/{{$index}}" class="btn btn-default btn-
    xs">View</a></td>
</tr>
```

Questo codice ha un aspetto molto familiare. Abbiamo associato la direttiva all'elemento che intendiamo ripetere (in questo caso la riga della tabella) e utilizzato le doppie parentesi graffe per visualizzare i dati dei contatti.

Nel link osserverete qualcosa di leggermente diverso. La direttiva `ng-repeat` ci consente di ottenere l'indice dell'oggetto corrente utilizzando `$index`. Ricorderete che la route del singolo contatto accetta un numero ID. Utilizzeremo l'indice dell'array come ID in modo da poter accedere facilmente al contatto quando ne avremo bisogno.

Condividere i dati tra le viste

Ciò che abbiamo realizzato finora presenta un problema. Anche se funziona perfettamente per la vista Index, è del tutto inutile quando intendiamo visualizzare un unico contatto. Infatti il nostro array di contatti è contenuto nel controller dell'indice e pertanto non può essere condiviso.

Condividere i dati mediante $rootScope

Sono disponibili due sistemi per condividere i dati tra le viste: il primo è `$rootScope`. Così come esiste uno scope per ogni vista, l'applicazione stessa ne ha uno, lo scope radice (*root scope*), che funziona allo stesso modo.

Per utilizzarlo è necessario inserire un nuovo servizio che si chiama `$rootScope`:

```
.controller('indexCtl', function($scope, $rootScope){

    $rootScope.contacts = [
        {
            name: 'Stephen Radford',
            phone: '0123456789',
            address: '123, Some Street\nLeicester\nLE1 2AB',
            email: 'stephen@email.com',
            website: 'stephenradford.me',
            notes: ''
        },
        {
            name: 'Declan Proud',
            phone: '91234859',
            address: '234, Some Street\nLeicester\nLE1 2AB',
            email: 'declan@declan.com',
            website: 'declanproud.me',
            notes: 'Some notes about the contact.'
        }
    ];

    $scope.contacts = $rootScope.contacts;

})
```

Possiamo associare nello stesso modo degli elementi all'oggetto scope radice. In questo caso l'abbiamo aggiunto anche allo scope della vista, ma avremmo potuto modificare facilmente la vista per accedere direttamente allo scope radice:

```
<tr ng-repeat="contact in $root.contacts">
```

Per accedere dalla vista a qualsiasi elemento nello scope radice, è sufficiente anteporre al nome del modello il prefisso `$root` seguito da un punto.

Questo metodo di condivisione dei dati non sfrutta tutti gli strumenti che ci offre Angular e genera molta confusione nell'applicazione.

NOTA

Il ricorso a `$rootScope`, e soprattutto il suo accesso dalla vista, è una pratica sconsigliata e può ostacolare la gestione del progetto; per condividere i dati è preferibile utilizzare un controller a livello dell'applicazione o un servizio personalizzato.

Creare un servizio personalizzato

Una soluzione migliore per condividere i dati tra le viste è sviluppare un sistema personalizzato. Un servizio è in sostanza una classe alla quale è possibile accedere dopo averla inserita nei controller, così come abbiamo visto con `$scope`.

In AngularJS ne esistono tre tipi: `.service()`, `.factory()` e `.value()`. Tutti sono *singleton*, dei design pattern che fanno in modo che di essi esista una sola istanza su un oggetto. Li esamineremo tutti prima di svilupparne uno.

Value

Il più essenziale dei tre è il metodo `value`. Aggiungiamolo al modulo:

```
.value('demoService', 'abc123');
```

Come possiamo vedere, si tratta di un servizio semplicissimo che accetta due parametri: il nome del servizio che intendiamo creare e il valore che dovrebbe avere. Questi dati possono essere condivisi nell'applicazione inserendoli nei controller:

```
.controller('indexCtl', function($scope, demoService){
    $scope.demo = demoService;
});
```

È molto semplice, ma offre un modo rapido e facile per condividere i dati. Potremmo voler condividere, per esempio, una chiave API tra molteplici controller. Il metodo `value` sarebbe la soluzione ideale.

Factory

`Value` è semplice ed efficace, ma non offre molte funzioni. Invece il servizio `factory` di Angular ci permette di chiamare altri servizi tramite la DI (*dependency injection*), e consente anche l'inizializzazione del servizio e l'inizializzazione lazy. Riscriviamo l'esempio contenente `value` sostituendolo con `factory`:

```
.factory('demoService', function demoServiceFactory(){
  return 'abc123';
});
```

NOTA

In questo caso abbiamo chiamato la funzione `[serviceName]Factory`. Anche se non è necessario, è opportuno farlo perché consente un debugging molto più semplice nell'analisi dello stack.

Funzionerà, ma per un'app così essenziale, è un sistema fin troppo elaborato ed è preferibile utilizzare `value`. Come abbiamo scoperto, `factory` può chiamare altri servizi. Creiamone un altro e inseriamo il `demoService` iniziale:

```
.factory('anotherService', function ['demoService',
  anotherServiceFactory(demoService){
```

```
  return demoService;
});
```

Il servizio `factory` può anche modificare il valore fornitoci da `demoService`, ma viene utilizzato, più frequentemente, per connettersi a un'API. Come nel caso di `value`, `factory` restituisce tipi JavaScript. Ecco come restituisce un oggetto:

```
.factory('anotherService', function ['demoService',
  anotherServiceFactory(demoService){
    return {
      connect: function(){
      }
    };
  });
```

Il metodo `connect` definito nell'oggetto restituito sarà direttamente accessibile quando inseriamo il servizio nel controller:

```
.controller('indexCtl', function($scope, anotherService){
    anotherService.connect();
});
```

Service

L'ultimo tipo di servizio presente in AngularJS viene chiamato `service` (un nome che genera confusione, come il filtro `Filter`). Produce un singleton al pari di `value` e `factory`, invocando però un costruttore mediante l'operatore `new`. Anche questo può generare confusione; cerchiamo di fare chiarezza.

Nel seguente caso è presente la funzione del costruttore `Notifier` che è un semplice alias per il metodo `window.alert` del browser:

```
function Notifier(){
  this.send = function(msg){
    window.alert(msg);
  }
}
```

Potremmo aggiungerla direttamente nel controller e chiamarla in questo modo:

```
var notifier = new Notifier();
notifier.send('Hello, World');
```

Anche se funzionerebbe, non è la soluzione ottimale. E se volessimo utilizzare nuovamente la funzione `Notifier` in un altro controller? Come sappiamo, possiamo condividere questo tipo di elementi ricorrendo a un servizio di AngularJS. Con `factory`, otterremmo qualcosa del genere:

```
.factory('notifierService', function notifierFactoryService(){
  return new Notifier();
});
```

Niente male, ma questo è proprio il tipo di risultato per il quale è stato concepito `service`. Crea un'istanza e restituisce un oggetto:

```
.service('notifierService', Notifier);
```

Ecco fatto! Angular prenderà il costruttore, ne creerà un'istanza e restituirà l'oggetto. Possiamo inserirlo nel controller e utilizzarlo come previsto:

```
.controller('indexCtl', function($scope, notifierService){
    notifierService.send('Hello, World');
});
```

Sviluppare un servizio

Potremmo utilizzare `service` o `factory` per condividere i contatti tra i controller, ma siccome non creiamo alcuna istanza, serviamoci di `factory`:

```
.factory('contacts', function contactsFactory(){

})
```

Come abbiamo visto, è solo l'oggetto restituito a contenere i metodi e le proprietà pubbliche. È possibile sfruttare questo aspetto includendo l'array di contatti privatamente, consentendo che sia restituito e accessibile soltanto da metodi pubblici:

```
var contacts = [
    …
];
```

Il servizio includerà due metodi per leggere i contatti. Il primo restituirà l'intero array, mentre il secondo un solo oggetto contatto che dipende dall'indice assegnato a esso. Restituiamo l'oggetto contenente questi due metodi e vediamo in che cosa consistono:

```
return {
    get: function(){
        return contacts;
    },
    find: function(index){
        return contacts[index];
    }
};
```

Entrambi sono funzioni molto "di base". Il metodo `get` restituisce l'intero array, mentre il metodo `find` accetta un indice che restituisce il contatto richiesto. Assembliamo il tutto e osserviamo il risultato:

```
.factory('contacts', function(){
    var contacts = [
        {
            name: 'Stephen Radford',
            phone: '0123456789',
            address: '123, Some Street\nLeicester\nLE1 2AB',
            email: 'stephen@email.com',
            website: 'stephenradford.me',
```

```
            notes: ''
        },
        {
            name: 'Declan Proud',
            phone: '91234859',
            address: '234, Some Street\nLeicester\nLE1 2AB',
            email: 'declan@declan.com',
            website: 'declanproud.me',
            notes: 'Some notes about the contact.'
        }
    ];
    return {
        get: function(){
            return contacts;
        },
        find: function(index){
            return contacts[index];
        }
    };
})
```

Ora possiamo inserire questo servizio nel controller, così come abbiamo fatto con `$scope` e `$rootScope`:

```
.controller('indexCtl', function($scope, contacts){
    $scope.contacts = contacts.get();
})
```

Abbiamo anche passato al nuovo metodo i contatti nello scope, e questo ha messo ordine nel controller. Se aggiungiamo `console.log` al servizio di contatti, vedremo che non è possibile vedere i dati grezzi, ma solo i due metodi ai quali abbiamo avuto accesso.

Utilizzare i parametri della route

Ora che il servizio funziona a dovere, possiamo iniziare a popolare la vista contenente un unico contatto. A questo scopo dovremo estrarre l'ID dalla route.

Nel Capitolo 4 abbiamo esaminato rapidamente come è possibile accedere ai parametri della route, ma ricapitoliamo l'argomento. Innanzitutto è necessario inserire un altro servizio nel controller dell'unico contatto: `$routeParams`. Questo servizio restituisce un oggetto con tutti i parametri nella route come proprietà separate:

```
.controller('contactCtl', function($scope, $routeParams,
  contacts){
```

```
    $scope.contact = contacts.find($routeParams.id);
});
```

In questo caso abbiamo accesso al parametro `id` che utilizziamo per trovare il contatto corretto mediante il servizio che abbiamo creato in precedenza. Lo passiamo alla vista creando un nuovo modello sullo scope chiamato `contact`.

Estraiamo tutte le informazioni pertinenti nel partial `contact.html`. Ricordate che tutti i dati sono proprietà del contatto del modello e possiamo accedervi in questo modo:

```
{{contact.name}}
```

Funziona tutto tranne per alcuni dettagli. I dati riguardanti l'indirizzo dovrebbero rispettare i fine riga, e ci piacerebbe ricavare dinamicamente l'immagine del gravatar. Per ottenere questi risultati è necessario creare un filtro e una direttiva.

Creare una direttiva personalizzata

Abbiamo sperimentato l'efficacia delle direttive e finora non avevamo alcun motivo per svilupparne una nuova. Ma per includere il gravatar, è necessario crearne una personalizzata.

Così come per i controller, i filtri e i servizi, la nuova direttiva deve essere associata al modulo utilizzando il metodo pertinente:

```
.directive('gravatar', function(){
})
```

Come per i controller, i filtri e i servizi, il metodo `directive` richiede due parametri. Il primo è il nome della direttiva, il secondo è una funzione. Una direttiva deve restituire un oggetto, e le proprietà dell'oggetto restituito definiscono il comportamento della direttiva.

La prima proprietà che imposteremo è `restrict`; definisce il modo in cui è possibile utilizzare la direttiva. Abbiamo già visto come è possibile avvalersi della maggior parte delle direttive come attributi o elementi personalizzati, ma Angular ci consente anche di usarle in altri due modi. È possibile impostare questi valori per la proprietà `restrict`.

- `A`: la direttiva può essere associata solo utilizzando un attributo, `<div gravatar> </div>`.
- `E`: la direttiva può essere utilizzata come elemento personalizzato, `<gravatar> </gravatar>`.

- `C`: la direttiva può essere utilizzata aggiungendola come classe all'elemento, `<div class="gravatar"></div>`.
- `M`: consente l'esecuzione della direttiva attraverso un commento HTML, `<!-- directive: gravatar -->`.

NOTA

Per le direttive è preferibile utilizzare attributi ed elementi rispetto a classi e commenti.

L'impostazione predefinita di Angular è la prima, come attributo. Possiamo utilizzare però la combinazione dei valori menzionati per mettere a punto il modo in cui intendiamo utilizzare la direttiva. Se la impostiamo su `AE` potremo chiamarla tramite un attributo o un elemento personalizzato:

```
.directive('gravatar', function(){
    return {
        restrict: 'AE'
    }
})
```

Se è necessario, potremmo anche creare un template per la direttiva. Può essere incluso direttamente nell'oggetto utilizzando la proprietà `template`, oppure, ricorrendo alla proprietà `templateUrl`, possiamo caricare dall'URI specificato un file di template esterno. Siccome qui creiamo soltanto un tag `img`, possiamo aggiungerlo direttamente nell'oggetto:

```
.directive('gravatar', function(){
    return {
        restrict: 'AE',
        template: '<img ng-src="{{img}}" class="{{class}}">'
    }
})
```

Questo template si comporta come le viste. Abbiamo aggiunto due segnaposto per l'URI dell'immagine oltre alle classi che intendiamo includere.

Per far funzionare il tutto, occorre associare una funzione alla proprietà `link` nell'oggetto. Da qui è possibile accedere allo scope, all'elemento al quale è associata la direttiva e a qualsiasi attributo di quell'elemento:

```
.directive('gravatar', function(){
    return {
        restrict: 'AE',
        template: '<img src="{{img}}" class="{{class}}">',
        link: function(scope, elem, attrs){

        }
    }
})
```

Per recuperare l'immagine del gravatar, si deve applicare la funzione hash all'indirizzo e-mail del contatto tramite `md5`. Purtroppo non è un metodo nativo di

JavaScript ed è necessario includere una libreria separata. Ne abbiamo inclusa una negli asset riguardanti questo capitolo, che potete scaricare; può essere inclusa come variabile single-line:

```
.directive(‘gravatar’, function(){
    return {
        restrict: ‘AE’,
        template: ‘<img src=”{{img}}” class=”{{class}}”>’,
        replace: true,
        link: function(scope, elem, attrs){
            var md5=function(s){function
L(k,d){return(k<<d)|(k>>>(32-d))}function K(G,k){var
I,d,F,H,x;F=(G&2147483648);H=(k&2147483648);I=(G&1073741824);d=(k&
1073741824);x=(G&1073741823)+(k&1073741823);if(I&d){return(x^21474
83648^F^H)}if(I|d){if(x&1073741824){return(x^3221225472^F^H)}else{
return(x^1073741824^F^H)}}else{return(x^F^H)}}function
r(d,F,k){return(d&F)|((~d)&k)}function
q(d,F,k){return(d&k)|(F&(~k))}function
p(d,F,k){return(d^F^k)}function
n(d,F,k){return(F^(d|(~k)))}function
u(G,F,aa,Z,k,H,I){G=K(G,K(K(r(F,aa,Z),k),I));return
K(L(G,H),F)}function
f(G,F,aa,Z,k,H,I){G=K(G,K(K(q(F,aa,Z),k),I));return
K(L(G,H),F)}function
D(G,F,aa,Z,k,H,I){G=K(G,K(K(p(F,aa,Z),k),I));return
K(L(G,H),F)}function
t(G,F,aa,Z,k,H,I){G=K(G,K(K(n(F,aa,Z),k),I));return
K(L(G,H),F)}function e(G){var Z;var F=G.length;var x=F+8;var k=(x-
(x%64))/64;var I=(k+1)*16;var aa=Array(I-1);var d=0;var
H=0;while(H<F){Z=(H-
(H%4))/4;d=(H%4)*8;aa[Z]=(aa[Z]|(G.charCodeAt(H)<<d));H++}Z=(H-
(H%4))/4;d=(H%4)*8;aa[Z]=aa[Z]|(128<<d);aa[I-2]=F<<3;aa[I-
1]=F>>>29;return aa}function B(x){var
k=””,F=””,G,d;for(d=0;d<=3;d++){G=(x>>>(d*8))&255;F=”0”+G.toString
(16);k=k+F.substr(F.length-2,2)}return k}function
J(k){k=k.replace(/rn/g,”n”);var d=””;for(var
F=0;F<k.length;F++){var
x=k.charCodeAt(F);if(x<128){d+=String.fromCharCode(x)}else{if((x>1
27)&&(x<2048)){d+=String.fromCharCode((x>>6)|192);d+=String.fromCh
arCode((x&63)|128)}else{d+=String.fromCharCode((x>>12)|224);d+=Str
ing.fromCharCode(((x>>6)&63)|128);d+=String.fromCharCode((x&63)|12
8)}}}return d}var C=Array();var P,h,E,v,g,Y,X,W,V;var
S=7,Q=12,N=17,M=22;var A=5,z=9,y=14,w=20;var
o=4,m=11,l=16,j=23;var
U=6,T=10,R=15,O=21;s=J(s);C=e(s);Y=1732584193;X=4023233417;W=25623
83102;V=271733878;for(P=0;P<C.length;P+=16){h=Y;E=X;v=W;g=V;Y=u(Y,
X,W,V,C[P+0],S,3614090360);V=u(V,Y,X,W,C[P+1],Q,3905402710);W=u(W,
V,Y,X,C[P+2],N,606105819);X=u(X,W,V,Y,C[P+3],M,3250441966);Y=u(Y,X
,W,V,C[P+4],S,4118548399);V=u(V,Y,X,W,C[P+5],Q,1200080426);W=u(W,V
,Y,X,C[P+6],N,2821735955);X=u(X,W,V,Y,C[P+7],M,4249261313);Y=u(Y,X
,W,V,C[P+8],S,1770035416);V=u(V,Y,X,W,C[P+9],Q,2336552879);W=u(W,V
,Y,X,C[P+10],N,4294925233);X=u(X,W,V,Y,C[P+11],M,2304563134);Y=u(Y
,X,W,V,C[P+12],S,1804603682);V=u(V,Y,X,W,C[P+13],Q,4254626195);W=u
(W,V,Y,X,C[P+14],N,2792965006);X=u(X,W,V,Y,C[P+15],M,1236535329);Y
=f(Y,X,W,V,C[P+1],A,4129170786);V=f(V,Y,X,W,C[P+6],z,3225465664);W
=f(W,V,Y,X,C[P+11],y,643717713);X=f(X,W,V,Y,C[P+0],w,3921069994);Y
=f(Y,X,W,V,C[P+5],A,3593408605);V=f(V,Y,X,W,C[P+10],z,38016083);W=
f(W,V,Y,X,C[P+15],y,3634488961);X=f(X,W,V,Y,C[P+4],w,3889429448);Y
=f(Y,X,W,V,C[P+9],A,568446438);V=f(V,Y,X,W,C[P+14],z,3275163606);W
=f(W,V,Y,X,C[P+3],y,4107603335);X=f(X,W,V,Y,C[P+8],w,1163531501);Y
=f(Y,X,W,V,C[P+13],A,2850285829);V=f(V,Y,X,W,C[P+2],z,4243563512);
W=f(W,V,Y,X,C[P+7],y,1735328473);X=f(X,W,V,Y,C[P+12],w,2368359562)
;Y=D(Y,X,W,V,C[P+5],o,4294588738);V=D(V,Y,X,W,C[P+8],m,2272392833)
;W=D(W,V,Y,X,C[P+11],l,1839030562);X=D(X,W,V,Y,C[P+14],j,425965774
0);Y=D(Y,X,W,V,C[P+1],o,2763975236);V=D(V,Y,X,W,C[P+4],m,127289335
3);W=D(W,V,Y,X,C[P+7],l,4139469664);X=D(X,W,V,Y,C[P+10],j,32002366
56);Y=D(Y,X,W,V,C[P+13],o,681279174);V=D(V,Y,X,W,C[P+0],m,39364300
74);W=D(W,V,Y,X,C[P+3],l,3572445317);X=D(X,W,V,Y,C[P+6],j,76029189
);Y=D(Y,X,W,V,C[P+9],o,3654602809);V=D(V,Y,X,W,C[P+12],m,387315146
1);W=D(W,V,Y,X,C[P+15],l,530742520);X=D(X,W,V,Y,C[P+2],j,329962864
5);Y=t(Y,X,W,V,C[P+0],U,4096336452);V=t(V,Y,X,W,C[P+7],T,112689141
5);W=t(W,V,Y,X,C[P+14],R,2878612391);X=t(X,W,V,Y,C[P+5],O,42375332
41);Y=t(Y,X,W,V,C[P+12],U,1700485571);V=t(V,Y,X,W,C[P+3],T,2399980
```

```
690);W=t(W,V,Y,X,C[P+10],R,4293915773);X=t(X,W,V,Y,C[P+1],O,224004
4497);Y=t(Y,X,W,V,C[P+8],U,1873313359);V=t(V,Y,X,W,C[P+15],T,42643
55552);W=t(W,V,Y,X,C[P+6],R,2734768916);X=t(X,W,V,Y,C[P+13],O,1309
151649);Y=t(Y,X,W,V,C[P+4],U,4149444226);V=t(V,Y,X,W,C[P+11],T,317
4756917);W=t(W,V,Y,X,C[P+2],R,718787259);X=t(X,W,V,Y,C[P+9],O,3951
481745);Y=K(Y,h);X=K(X,E);W=K(W,v);V=K(V,g)}var
i=B(Y)+B(X)+B(W)+B(V);return i.toLowerCase()};
        }
    }
})
```

Dopo aver incluso la funzione `md5`, potremo accedere all'immagine del contatto dal gravatar. Passeremo due elementi alla funzione del link. Il primo sarà l'indirizzo e-mail, mentre il secondo parametro facoltativo sarà costituito dalle dimensioni dell'immagine che intendiamo recuperare.

Gli attributi passati alla funzione sono semplici oggetti ai quali possiamo accedere. Per esempio, se intendiamo recuperare il valore dell'attributo dell'indirizzo e-mail, possiamo accedervi mediante `attrs.email`.

Abbiamo anche definito la proprietà `replace`, che sostituirà l'elemento al quale abbiamo associato la direttiva con il template specificato. Per impostazione predefinita, Angular aggiungerà il template come elemento figlio.

Finiamo rapidamente e sperimentiamo la nuova direttiva:

```
.directive('gravatar', function(){
    return {
        restrict: 'AE',
        template: '<img src="{{img}}" class="{{class}}">',
        link: function(scope, elem, attrs){
            var md5 = function(s){ ... };
            var size = (attrs.size) ? attrs.size : 64;
            scope.img = 'http://gravatar.com/avatar/'+md5(attrs.email)+'?s='+size;
            scope.class = attrs.class;
        }
    }
})
```

Abbiamo utilizzato un operatore ternario per avere la possibilità di impostare le dimensioni dell'immagine. Abbiamo anche associato le classi assegnate all'elemento, oltre a unire l'URL del gravatar allo scope.

La direttiva è pronta. Proviamo a utilizzarla con un attributo:

```
<div gravatar email="{{contact.email}}" size="50" class="img-
  circle"></div>
```

Ottimo, sembra funzionare tutto. Viene visualizzato il gravatar e viene inclusa la classe che produce una bella immagine circolare. Purtroppo essa è racchiusa nel tag `div` al quale abbiamo associato la direttiva e quindi si sposta su una nuova riga. Ciò accade perché non abbiamo comunicato ad Angular che intendiamo sostituire l'elemento esistente con il template compilato. A questo scopo impostiamo su `true` la proprietà `replace` dell'oggetto della direttiva:

```
.directive('gravatar', function(){
    return {
        restrict: 'AE',
        template: '<img src="{{img}}" class="{{class}}">',
        replace: true,
        link: function(scope, elem, attrs){
            var md5=function(s){ … };
            var size = (attrs.size) ? attrs.size : 64;
            scope.img = 'http://gravatar.com/avatar/'+md5(attrs.email)+'?s='+size;
            scope.class = attrs.class;
        }
    }
})
```

Se aggiorniamo il browser, vedremo che al posto dell'immagine racchiusa nell'elemento originario è presente la nostra immagine. In alternativa avremmo potuto chiamare la direttiva tramite un elemento personalizzato:

```
<gravatar email="{{contact.email}}" size="50" class="img-
  circle"></gravatar>
```

Se non serve specificatamente il supporto a IE8 (non più supportato in AngularJS 1.3+), le direttive che inseriscono nuovi elementi tramite l'utilizzo di un template dovrebbero venire chiamate mediante un elemento personalizzato, come nell'esempio precedente.

Le direttive che manipolano elementi esistenti, per esempio chiamando un plug-in jQuery, dovrebbero venire chiamate soltanto tramite un attributo.

La prossima figura mostra la nuova direttiva quando viene visualizzato un contatto.

Rispettare i fine riga

Attualmente i campi `address` e `notes` non rispettano i fine riga perché le nuove righe devono essere convertite in interruzioni di riga HTML. Fortunatamente Angular semplifica al massimo questa operazione mediante un filtro personalizzato.

Come abbiamo visto realizzando un filtro nel Capitolo 3, descriveremo rapidamente il filtro `paragraph` che è necessario creare per convertire il fine riga `\n` in `<br />`:

```
.filter('paragraph', function(){
    return function(input){
        return (input) ? input.replace(/\n/g, '<br />') : input;
    };
})
```

Se aggiungiamo questo filtro all'indirizzo usando la sintassi che prevede l'operatore pipe (|), noteremo qualcosa di strano. Le interruzioni di pagina vengono convertite in entità HTML e visualizzate sulla pagina. Per motivi di sicurezza, Angular esegue automaticamente l'escape delle entità HTML per impedire lo scripting cross-site.

Siccome non intendiamo ovviamente visualizzarle, dobbiamo utilizzare una direttiva inclusa per legare il modello alla pagina. La direttiva `ng-bind-html` produrrà esattamente questo risultato. Eccola in azione nel tag `paragraph`:

```
<p class="form-control-static" ng-bind-html="contact.address |
  paragraph"></p>
```

Ma non funziona ancora. Se controlliamo la console, Angular genera il seguente errore:

```
Error: [$sce:unsafe] Attempting to use an unsafe value in a safe
  context.
```

Infatti Angular richiede il modulo `ngSanitize` che è possibile scaricare dalla sezione *Extras* all'indirizzo https://angularjs.org/. Il modulo filtra i frammenti di codice pericolosi, come gli script, producendo un output ripulito e sicuro.

Procuratevelo dal sito di Angular, aggiungetelo alla directory *js* e includetelo nel file principale `index.html`:

```
<script type="text/javascript" src="/assets/js/angular-
  sanitize.min.js"></script>
```

Dopo aver incluso il modulo nella pagina, è necessario inserirlo come dipendenza del nostro modulo, così come abbiamo fatto con `ngRoute`:

```
angular.module('contactsMgr', ['ngRoute', 'ngSanitize'])
```

Se aggiornate la pagina, vedrete che ora l'indirizzo occupa più righe, come previsto.

Impostare la ricerca e aggiungere la pageClass active

L'ultimo argomento da affrontare nell'ambito del *Read* è la ricerca. Poiché la ricerca filtrerà la tabella nella vista Index, dobbiamo reindirizzarla quando iniziamo a digitare. Inoltre il modello che utilizzeremo dovrà essere accessibile ovunque.

Dobbiamo anche capire come impostare la `pageClass active`. Al momento è fissa su *Browse* ma sarebbe bello renderla dinamica.

Entrambi questi obiettivi non si possono raggiungere con `ng-view`; è necessario creare un controller per l'intera applicazione. Questo significa che saremo in grado di accedere ovunque al modello per la ricerca:

```
.controller('appCtl', function($scope, $location){

});
```

Per reindirizzare la pagina abbiamo inserito il servizio `$location`. Esso offre l'accesso al metodo `path`, che possiamo sfruttare per realizzare il nostro intento. A differenza dei controller della route, è necessario chiamarlo sulla pagina aggiungendolo al tag di apertura `HTML`:

```
<html lang="en" ng-app="contactsMgr" ng-controller="appCtl">
```

La ricerca

Ora che il controller è inizializzato, possiamo procedere. La direttiva `ng-keyup` si attiverà non appena inizieremo a digitare nella casella di ricerca, reindirizzandoci alla vista Index.

Aggiungiamo l'handler al `controller`:

```
.controller('appCtl', function($scope, $location){

    $scope.startSearch = function(){
        $location.path('/');
    };

});
```

Si tratta di una funzione abbastanza semplice e di facile comprensione. Quando aggiungeremo la direttiva alla casella di ricerca, essa modificherà il percorso corrente nella vista Index. Procediamo e impostiamola. Se non l'avete già fatto, è il momento di assegnare anche un modello alla casella di ricerca:

```
<form class="navbar-form navbar-right" role="search">
    <input type="text" class="form-control" placeholder="Search" ng-
      model="search" ng-keyup="startSearch()">
</form>
```

Ora che abbiamo visto come reindirizzare, è sufficiente filtrare `ng-repeat` come abbiamo fatto in precedenza:

```
<tr ng-repeat="contact in contacts | filter:search">
```

Ricordate che se volete limitarlo al nome, è necessario modificare il `model` nella casella di ricerca nel seguente modo:

```
<input type="text" class="form-control" placeholder="Search" ng-
  model="search.name" ng-keyup="startSearch()">
```

La pageClass active

Infine è necessario impostare la `pageClass active`. Dobbiamo verificare il percorso corrente e aggiungere, se necessario, una classe. A questo scopo utilizziamo `ng-class` e

una funzione nel controller dell’app.

La funzione verificherà se il percorso corrente è lo stesso di quello che è stato passato:

```
$scope.pageClass = function(path){
    return (path == $location.path()) ? ‘active’ : ‘’;
};
```

Se c’è corrispondenza, restituirà la classe `active`, altrimenti nulla. Aggiungiamola a entrambi gli elementi della navigazione:

```
<li ng-class=”pageClass(‘/’)”><a href=”/”>Browse</a></li>
<li ng-class=”pageClass(‘/add-contact’)”><a
  href=”/add-contact”>Add Contact</a></li>
```

Dopo aver aggiunto la direttiva `ng-class` all’elemento della lista, la pagina corrente mostrata dalla classe `active` diventa interamente dinamica e corretta.

Create

Finora abbiamo trascurato la prima lettera dell'acronimo CRUD, ma ora è venuto il momento di fare un passo indietro e impostare il form di aggiunta dei contatti. Dobbiamo innanzitutto verificare che tutti gli input del form abbiano associato il modello opportuno:

```
<input type="text" id="name" class="form-control" ng-
  model="contact.name">
```

Poiché abbiamo realizzato un servizio per gestire i contatti, è ragionevole estenderlo per riuscire a creare con esso i contatti. Definiamo un metodo `create` che porti il contatto nell'array e lo aggiunga all'oggetto del servizio:

```
create: function(contact){
    contacts.push(contact);
}
```

Ora riflettiamo su ciò che vogliamo accada dopo l'invio del form. Intendiamo inserire un contatto nell'array utilizzando il metodo appena creato nel servizio, fornire un feedback positivo su questa operazione e infine cancellare tutto dal form:

```
$scope.submit = function(){
    contacts.create($scope.contact);
    $scope.contact = null;
    $scope.added = true;
};
```

Con il precedente codice siamo riusciti nel nostro intento: abbiamo passato il contatto al servizio, reimpostato il modello del contatto e definito un modello che possiamo verificare per fornire il feedback.

Esistono due sistemi per chiamare la funzione appena creata. Potremmo aggiungerla come direttiva `ng-click` al pulsante *submit*, ma forse è più saggio e accessibile aggiungerla a una direttiva `ng-submit` nel form:

```
<form class="form-horizontal" ng-submit="submit()">
```

Si tratta di includere la finestra di avviso per comunicare all'utente che il contatto è stato aggiunto con successo. Vogliamo che per impostazione predefinita sia nascosta, così utilizzando `ng-show` e osservando il `model` aggiunto, possiamo decidere quando visualizzarla:

```
<div class="alert alert-success" ng-show="added">
  The contact was added successfully.
</div>
```

Update

Come ricorderete, non abbiamo creato una vista destinata unicamente alla modifica dei contatti. Possiamo sfruttare lo stesso partial che abbiamo utilizzato per aggiungere i contatti oppure fare qualcosa di più stimolante con la vista contenente un solo contatto e creare una direttiva per modificare i dati.

È improbabile che sia necessario modificare in una volta sola tutti i dati di un contatto; spesso si tratterà di cambiare soltanto il numero di telefono o l'indirizzo e-mail. La nostra idea è quella di visualizzare il testo a fianco di un pulsante *Edit*. Al clic su di esso, potremo modificare quel dato del contatto.

Chiamiamo la direttiva `editable` ed eseguiamola come abbiamo fatto in precedenza:

```
.directive('editable', function(){
    return {

    };
})
```

È opportuno lasciare la scelta di includerla come elemento personalizzato oppure utilizzarla mediante un attributo. Per il momento la utilizzeremo soltanto come attributo, ma forse in futuro potrebbero richiederla altri progetti:

```
.directive('editable', function(){
    return {
        restrict: 'AE',
        templateUrl: '/assets/partials/editable.html'
    };
})
```

Questa volta non sarà sufficiente una riga di markup, così abbiamo deciso di utilizzare la proprietà `templateUrl`. Si tratta soltanto di creare il reference partial, come abbiamo fatto con le route.

Scope

Oltre a decidere di utilizzare un URL per il template, studieremo una nuova proprietà: `scope`. Essa ci offre maggiore controllo sullo scope che intendiamo utilizzare con la direttiva.

In questo caso se la impostiamo come hash si creerà un nuovo scope isolato. Non eredita dal genitore, e quindi non dovremo preoccuparci della lettura o della modifica accidentale dei dati nello scope della vista. In questo caso abbiamo aggiunto due valori alla funzione hash dello `scope`:

```
.directive('editable', function(){
    return {
```

```
        restrict: 'AE',
        templateUrl: '/assets/partials/editable.html',
        scope: {
            value: '=editable',
            field: '@fieldType'
        }
    };
})
```

La chiave è il nome che assegniamo al nuovo scope, mentre il valore è un attributo dell'elemento. Notate che abbiamo anteposto due diversi prefissi ai valori, che pertanto si comporteranno in due modi molto diversi.

NOTA

Ricordate che gli attributi separati da un trattino sono convertiti in CamelCase da Angular.

Quando il prefisso corrisponde al segno `=`, possiamo legare un modello dello scope genitore allo scope della direttiva. Non dobbiamo quindi utilizzare la sintassi `{{}}` e possiamo approfittare del binding dei dati bidirezionale.

Ne avremo bisogno perché modificheremo il valore del modello collegato all'interno della direttiva.

Se anteponiamo il simbolo `@`, la direttiva utilizzerà il valore letterale dell'attributo. Possiamo utilizzare la sintassi `{{}}` per passare il valore di un modello o immettere una stringa. Quando utilizziamo il simbolo `@` nessun modello è collegato.

Controller

Abbiamo visto in precedenza come possiamo utilizzare il metodo `link` nella direttiva, ma ne abbiamo anche un altro a disposizione. Il metodo `controller` funziona e si comporta proprio come un controller direttamente associato al modulo. Possiamo inserire qualsiasi servizio necessario e tutto ci sembrerà molto familiare.

La differenza rispetto a `link` consiste nell'ordine nel quale viene elaborato. Il metodo `controller` viene eseguito prima che l'applicazione abbia finito di essere compilata, mentre il metodo `link` dopo. Dovremmo utilizzare sempre controller, a meno che non creiamo una direttiva wrapper per un plug-in jQuery o qualcosa che deve essere eseguito dopo che tutto ha finito di caricarsi.

In precedenza abbiamo utilizzato `link` per la direttiva del gravatar poiché volevamo descrivere le differenze tra i due approcci all'interno della direttiva. Aggiungiamo il metodo `controller` alla direttiva. In questa fase dovrebbe presentarsi così:

```
.directive('editable', function(){
    return {
        restrict: 'AE',
```

```
        templateUrl: '/assets/partials/editable.html',
        scope: {
            value: '=editable',
            field: '@fieldType'
        },
        controller: function($scope){

        }
    };
})
```

NOTA

La proprietà `controller` nella direttiva si comporta come una sorta di API e consente alle altre direttive di comunicare le une con le altre, a differenza di `link`.

Assemblare il tutto

Abbiamo impostato lo scope e il controller. Ora è il momento di popolare il partial e definire la funzionalità. Visualizziamo il valore del modello e includiamo il pulsante *Edit* che attiverà l'editor:

```
<span ng-bind-html="value | paragraph"></span> <button class="btn
  btn-default btn-xs">Edit</button>
```

Come ricorderete, abbiamo impostato l'alias del nome dell'attributo modificabile su `value`, ed è proprio questo che utilizzeremo. Dobbiamo inoltre prevedere tipi di campi di una riga o di più righe; pertanto ricorreremo a `ng-bind-html` e al filtro `paragraph`.

Sfrutteremo `ng-show` e `ng-hide` per analizzare un modello nello scope della direttiva. È opportuno consentire all'utente di cancellare le modifiche e quindi non modificheremo il valore che abbiamo passato direttamente. Aggiungere il seguente codice al controller crea un nuovo modello che possiamo modificare, oltre a generare qualcosa che `ng-show` e `ng-hide` possono tenere d'occhio:

```
$scope.editor = {
    showing: false,
    value: $scope.value
};
```

È possibile utilizzare il modello `editor.showing` per creare due sezioni nel template. Una sezione verrà visualizzata prima che facciamo clic su *Edit*, l'altra dopo:

```
<div ng-hide="editor.showing">
  <span ng-bind-html="value | paragraph"></span> <button class="btn
    btn-default btn-xs">Edit</button>
</div>
<div ng-show="editor.showing">

</div>
```

Creiamo la funzione che utilizzeremo per mostrare o nascondere l'editor, che chiameremo da `ng-click`:

```
$scope.toggleEditor = function(){
    $scope.editor.showing = !$scope.editor.showing;
};
```

Ora agganciamo tutto al pulsante *Edit* nel partial:

```
<span ng-bind-html=”value | paragraph”></span> <button class=”btn
  btn-default btn-xs” ng-click=”toggleEditor()”>Edit</button>
```

La direttiva ci permetterà di scegliere il tipo di input necessario (testo, e-mail, area di testo e così via) anche se preferiamo l’impostazione predefinita, una casella di testo di una riga, perché è ciò che utilizzeremo con maggiore frequenza. In questo caso abbiamo optato per un operatore ternario per verificare se è stato impostato un valore o se si deve utilizzare l’impostazione predefinita:

```
$scope.field = ($scope.field) ? $scope.field : ‘text’;
```

La direttiva `ng-if` è stata aggiunta di recente in AngularJS 1.2. A differenza di `ng-show` e `ng-hide`, associa o dissocia elementi dal *Document Object Model* (DOM) se non soddisfano la condizione; in questo caso è perfetta per verificare il tipo di campo:

```
<div ng-show=”editor.showing”>
  <div ng-if=”field == ‘textarea’”>
    <textarea ng-model=”editor.value” class=”form-
      control”></textarea>
  </div>
  <div ng-if=”field != ‘textarea’”>
    <input type=”{{field}}” ng-model=”editor.value” class=”form-
      control”>
  </div>
</div>
```

Come potete vedere, è semplice da utilizzare e offre un controllo completo sia che intendiamo utilizzare un’area di testo o un elemento di input. Abbiamo anche popolato l’attributo `type` del tag `input` con il valore che è stato passato e incluso in entrambi gli elementi nel nuovo modello.

Non ci resta che inserire nel template due pulsanti per salvare o cancellare. Sarebbe opportuno separarli con un filetto orizzontale:

```
<div ng-hide=”editor.showing”>
  <span ng-bind-html=”value | paragraph”></span> <button class=”btn
    btn-default btn-xs” ng-click=”toggleEditor()”>Edit</button>
</div>
<div ng-show=”editor.showing”>
  <div ng-if=”field == ‘textarea’”>
    <textarea ng-model=”editor.value” class=”form-
      control”></textarea>
  </div>
  <div ng-if=”field != ‘textarea’”>
    <input type=”{{field}}” ng-model=”editor.value” class=”form-
      control”>
  </div>
  <hr>
  <button class=”btn btn-success btn-xs” ng-
    click=”save()”>Save</button>
  <button class=”btn btn-default btn-xs” ng-
    click=”toggleEditor()”>Cancel</button>
</div>
```

Abbiamo associato il pulsante *Save* a una nuova funzione `save` che dobbiamo ancora creare; il pulsante *Cancel* utilizza la stessa funzione `toggleEditor` precedente.

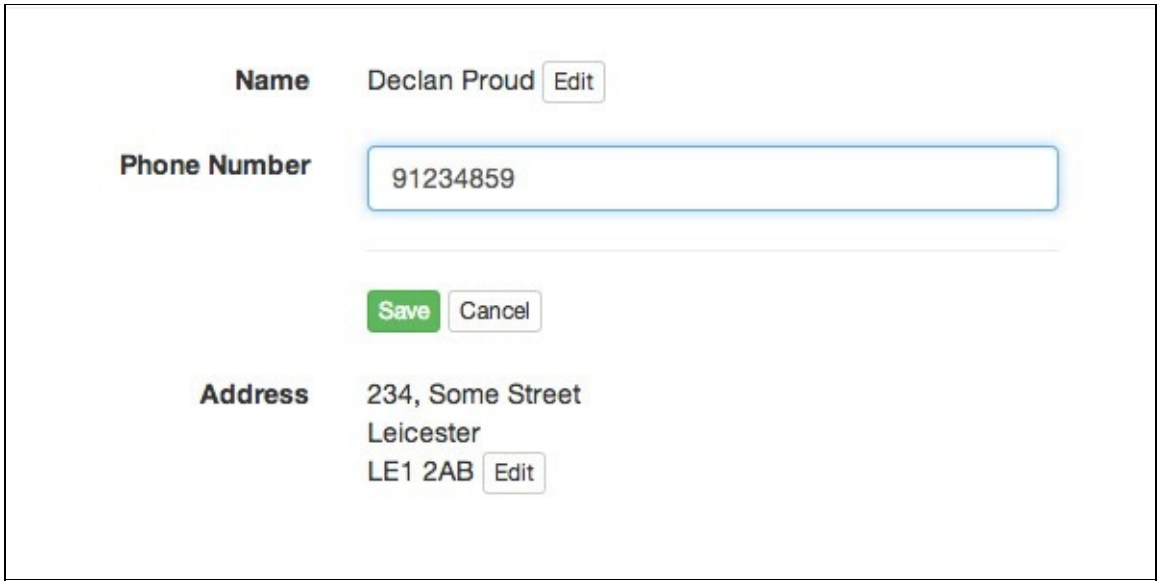
La funzione `save` è semplice; assegna il nuovo modello che abbiamo creato a quello che abbiamo legato in precedenza alla direttiva e chiama la funzione `toggleEditor` per nascondere tutto:

```
$scope.save = function(){
    $scope.value = $scope.editor.value;
    $scope.toggleEditor();
};
```

Abbiamo terminato la direttiva modificabile, ma come possiamo utilizzarla? Nel partial `contact.html` abbiamo visualizzato in precedenza tutti i modelli all'interno dei tag `paragraph` ricorrendo alla sintassi con le doppie parentesi graffe. Ora che la direttiva fa tutto questo al posto nostro, possiamo sostituire il contenuto del tag `<p>` e aggiungere due attributi:

```
<p class="form-control-static" editable="contact.email" field-
  type="email"></p>
```

Dopo aver sostituito tutti i modelli, dovreste ottenere un editor visivo di facile utilizzo per ogni sezione del contatto.



Delete

È possibile cancellare i contatti grazie, nuovamente, al servizio. Creeremo un metodo finale che accetterà un indice dell'array e lo cancellerà. Siccome `delete` è una parola chiave di JavaScript, utilizzeremo `destroy` come nome del metodo:

```
destroy: function(index){
    contacts.splice(index, 1);
}
```

È un metodo molto semplice. Prendiamo l'indice e utilizziamo il metodo nativo `splice` per cancellarlo dall'array. Ora dobbiamo creare una funzione sullo scope della vista `index` in grado di chiamare questo metodo dal servizio:

```
$scope.delete = function(index){
    contacts.destroy(index);
};
```

Infine aggiungiamo un pulsante alla colonna delle azioni della tabella per cancellare al clic il contatto voluto:

```
<button class=”btn btn-danger btn-xs” ng-
  click=”delete($index)”>Delete</button>
```

Quiz

1. Indicate due modi con i quali è possibile condividere i dati tra le viste.
2. Quali sono i tre tipi di servizi disponibili?
3. Che cosa è necessario includere per utilizzare `ng-bind-html`?
4. Qual è la differenza tra i metodi `link` e `controller` di una direttiva?
5. Quando si utilizza lo scope isolato, che cosa significano i prefissi `=` e `@`?
6. In che modo possiamo limitare una direttiva a un elemento e a un commento?
7. Perché è necessario creare un `controller` per tutta l'applicazione?
8. In che modo otteniamo l'indice di un oggetto da `ng-repeat`?

Riepilogo

In questo capitolo abbiamo affrontato molti argomenti ed è una buona idea riepilogarli. A grandi linee abbiamo trasformato ciò che erano template statici in un'app pienamente funzionante che permette il CRUD (*Create*, *Read*, *Update*, *Delete*).

Nel frattempo abbiamo scoperto molto altro. Abbiamo esaminato il sistema migliore per condividere i dati tra le viste creando un servizio personalizzato per gestire i contatti. Il servizio completo ci consente, ovunque nell'applicazione, di recuperare tutti i contatti, trovarne uno solo, aggiungerne uno nuovo o cancellarlo.

Oltre a definire un servizio personalizzato, abbiamo anche creato nuove direttive. La prima ci ha permesso di visualizzare l'immagine di un gravatar basata su un indirizzo e-mail. Abbiamo scoperto i molteplici e diversi modi di utilizzo di una direttiva, tramite un attributo o perfino un commento HTML.

La seconda direttiva che abbiamo creato è un po' più complessa. Abbiamo considerato scope isolati e la differenza tra i metodi `link` e `controller` di una direttiva; inoltre abbiamo sviluppato un ottimo sistema per modificare i dati del contatto.

Nel capitolo seguente vedremo in azione la libreria di terze parti AngularStrap che ci consente di avvalerci dei plug-in di Bootstrap all'interno di Angular.

AngularStrap

Abbiamo descritto l’elevato numero di componenti presenti in Bootstrap; ora esamineremo l’utilizzo dei plug-in JavaScript disponibili. È possibile creare delle direttive per ciascuno di essi, ma la community di Angular fornisce già un ricco modulo chiamato AngularStrap.

In questo capitolo considereremo AngularStrap, i plug-in che Bootstrap ci permette di utilizzare e il modo in cui è possibile inserirli nell’applicazione.

Installare AngularStrap

Innanzitutto dobbiamo scaricare AngularStrap. Potete procurarvelo dal sito `http://mgcrea.github.io/angular-strap/`. Fate clic sul pulsante di download in alto a destra e scaricate l'ultima versione come file ZIP.

Quest'ultimo contiene, tra l'altro, tutti i moduli singoli sotto forma di un comodo file minificato. Troverete i due file che ci servono nella directory *dist*. Copiate `angular-strap.min.js` e `angular-strap.tpl.min.js` nella directory *js* del progetto. Includeteli nel file `index.html` della radice dopo Angular e prima del modulo del progetto:

```
<script type="text/javascript" src="/assets/js/angular-
strap.min.js"></script>
<script type="text/javascript" src="/assets/js/angular-
strap.tpl.min.js"></script>
```

Come per ogni altro modulo, è necessario inserirlo nell'applicazione. La dichiarazione è situata nella prima riga del `controller.js`; il nome del modulo AngularStrap è `mgcrea.ngStrap`. Di seguito abbiamo aggiunto AngularStrap come una dipendenza del modulo `contactsMgr`:

```
angular.module('contactsMgr', ['ngRoute', 'ngSanitize',
  'mgcrea.ngStrap'])
```

Ma non è finita qui. AngularStrap dipende dal modulo `ngAnimate` che dobbiamo ancora includere nel progetto. Lo possiamo scaricare facendo clic sul link *Extras* nella finestra modale di download all'indirizzo `https://angularjs.org/`.

Aggiungete la versione minificata alla directory *js* del progetto e includetela prima di AngularStrap:

```
<script type="text/javascript" src="/assets/js/angular-
  animate.min.js"></script>
```

Il modulo `ngAnimate` non deve essere inserito nel progetto a meno che non vogliate usarlo al di fuori delle direttive di Bootstrap. Consente di sfruttare le animazioni CSS in moduli come `ngShow` e `ngHide` per ottenere un effetto di transizione invece di visualizzare semplicemente qualcosa.

Potremmo anche sviluppare delle animazioni da utilizzare insieme con AngularStrap, anche se AngularMotion è il suo compagno perfetto. Si tratta di un semplice foglio di stile che contiene animazioni predefinite pronte da utilizzare con il modulo `ngAnimate`.

Possiamo scaricare l’ultima versione dal sito `http://mgcrea.github.io/angular-motion/` facendo clic sul pulsante di download in alto a destra. Anche questo file ZIP contiene i file sorgente oltre alla versione minificata pronta per la produzione, presente nella directory *dist*. Copiatela nel progetto e includetela nella pagina come foglio di stile secondario:

```
<link rel=”stylesheet” href=”/assets/css/angular-motion.min.css”>
```

Ora potremo creare effetti di transizione, far scorrere, ridimensionare e ruotare le animazioni fornite da AngularMotion insieme con le direttive AngularStrap.

Forse vi sembrerà strano che non sia necessario includere lo script dei plug-in di Bootstrap. C’è un motivo: le direttive che abbiamo incluso con AngularStrap non sono semplici funzioni wrapper che eseguono jQuery, ma riscritture complete che sfruttano appieno Angular.

Utilizzare AngularStrap

Dopo aver installato AngularStrap, esaminiamo alcuni suoi plug-in e come utilizzarli.

Prima di iniziare, impostiamo rapidamente un ambiente demo. Duplichiamo il file `index.html` e rinominiamolo `demo.html`. Modificheremo anche il controller in `demoCtrl` e lo aggiungeremo al file `controller.js`. In questo modo avremo uno spazio in cui operare.

La finestra modale

Una finestra modale è un paradigma UI molto comune nelle app. È un ottimo sistema per visualizzare poche informazioni senza indirizzare l'utente a una nuova pagina.

Può essere chiamata al clic su un pulsante a cui è applicata la direttiva `bs-modal`:

```
<button class=”btn btn-primary” bs-modal=”modal”>Show
  Modal</button>
```

Il valore che viene passato è un modello dello scope; è un hash contenente due valori: `title` e `content`. Eccolo all'interno del controller:

```
$scope.modal = {
    title: ‘Modal Title’,
    content: ‘Modal content’
};
```

Esistono alcune opzioni da sfruttare con la direttiva `modal`; sono applicate all'elemento come attributi e fatte precedere da `data-`. Per esempio, se intendiamo modificare l'animazione, possiamo scrivere questo codice:

```
<button class=”btn btn-primary” bs-modal=”modal” data-
  animation=”am-fade-and-scale”>Show Modal</button>
```

che utilizzerà l'animazione `fade-and-scale` di AngularMotion. Ricordatevi di visitare il sito di AngularMotion per una lista completa delle animazioni disponibili.

La tabella seguente, tratta dal sito di AngularStrap, illustra la lista di tutte le opzioni disponibili per la direttiva `modal`.

Nome	Tipo	Impostazione predefinita	Descrizione
`animation`	stringa	`am-fade`	Applica un'animazione CSS.
`backdropAnimation`	stringa	`am-fade`	Applica un'animazione CSS allo sfondo.
`placement`	stringa	`'top'`	Posiziona la direttiva `modal`: in alto (`top`)/in basso (`bottom`)/al centro (`center`).

`title`	stringa	`''`	Valore predefinito del titolo.
`content`	stringa	`''`	Valore predefinito del contenuto.
`html`	booleano	`false`	Sostituisce `ng-bind` con `ng-bind-html`.
`backdrop`	booleano o `'static'`	`true`	Include un elemento modale dello sfondo. Utilizza `static` per lo sfondo, che non chiude la finestra modale al clic.
`keyboard`	booleano	`true`	Chiude la finestra modale quando si preme il tasto Esc.
`container`	stringa/`false`	`false`	Aggiunge la finestra modale a un elemento specifico. Esempio: `container: 'body'`.
`template`	percorso	`false`	Se fornito, scavalca il template predefinito.
`contentTemplate`	percorso	`false`	Se fornito, recupera il partial e lo include come contenuto interno.

Tooltip

I tooltip sono un ottimo sistema per offrire suggerimenti e consigli senza essere invadenti. AngularStrap ne semplifica l'inclusione; li attiviamo con il clic, l'hover o il focus:

```
<button class="btn btn-link" bs-tooltip="tooltip">what's
  this?</button>
```

In questo caso abbiamo un pulsante (al quale è stato applicato uno stile in modo da farlo sembrare un link grazie alle classi di Bootstrap) e abbiamo incluso la direttiva `bsTooltip`.

Così come abbiamo fatto con la direttiva `modal`, possiamo passare un modello alla direttiva. Questa volta dobbiamo includere soltanto la proprietà `title` nell'oggetto:

```
$scope.tooltip = {
    title: 'Tooltip Title'
};
```

Per impostazione predefinita il tooltip comparirà all'hover sul pulsante, ma è possibile modificarla facilmente utilizzando gli attributi `data` che abbiamo visto in precedenza:

```
<button class="btn btn-link" bs-tooltip="tooltip" data-
  trigger="click">what's this?</button>
```

La direttiva ci consente anche di legarla a un input e mostrare il tooltip al focus. Anche il posizionamento può essere definito dall'attributo `data`:

```
<input type="text" bs-tooltip="tooltip" data-trigger="focus" data-
  placement="right">
```

Questo codice mostrerà il tooltip a destra quando l'input ottiene il focus. Di seguito potete vedere la lista di tutte le opzioni presente nella documentazione di AngularStrap.

Nome	Tipo	Impostazione predefinita	Descrizione
`animation`	stringa	`am-fade`	Applica un'animazione CSS.
`placement`	stringa	`'top'`	Posiziona il tooltip: `top`/`bottom`/`left`/`right` o qualsiasi altra combinazione come `bottom-left`.
`trigger`	stringa	`'hover'`	Definisce come si attiva il tooltip: `click`/`hover`/`focus`.
`title`	stringa	`''`	Valore predefinito del titolo.
`html`	booleano	`false`	Sostituisce `ng-bind` con `ng-bind-html`.
`delay`	numero/oggetto	`0`	Ritarda la comparsa o la scomparsa del tooltip (`ms`); non si applica a un'attivazione manuale. Se viene indicato un numero, il ritardo si applica sia a `hide` sia a `show`. La struttura dell'oggetto è la seguente: `delay: { show: 500, hide: 100 }`.
`container`	stringa/`false`	`false`	Aggiunge la finestra modale a un elemento specifico. Esempio: `container: 'body'`
`template`	percorso	`false`	Se fornito, scavalca il template predefinito.
`contentTemplate`	percorso	`false`	Se fornito, recupera il partial e lo include come contenuto all'interno.

Popover

I popover sono una sorta di tooltip esteso e forniscono un'area `title` e `content`. Simili ai tooltip, si possono attivare al clic, all'hover o al focus:

```
<button class=”btn btn-primary” bs-popover=”popover”>Show
  Popover</button>
```

Il modello associato è identico per formato a quello utilizzato per la finestra modale poiché contiene le proprietà `title` e `content`:

```
$scope.popover = {
    title: ‘Title’,
    content: ‘Popover content’
};
```

Ovviamente tutto è modificabile mediante gli attributi `data`. La prossima tabella riporta una lista completa delle opzioni.

Nome	Tipo	Impostazione predefinita	Descrizione
`animation`	stringa	`am-fade`	Applica un'animazione CSS.
`placement`	stringa	`'top'`	Posiziona il tooltip: `top`/`bottom`/`left`/`right`, o qualsiasi altra combinazione come `bottom-left`.

`trigger`	stringa	`'hover'`	Definisce come si attiva il tooltip: `click`/`hover`/`focus`.
`title`	stringa	`''`	Valore predefinito del titolo.
`content`	stringa	`''`	Valore predefinito del contenuto.
`html`	booleano	`false`	Sostituisce `ng-bind` con `ng-bind-html`.
`delay`	numero/oggetto	`0`	Ritarda la comparsa o la scomparsa del tooltip (`ms`); non si applica a un'attivazione manuale. Se viene indicato un numero, il ritardo si applica sia a `hide` sia a `show`. La struttura dell'oggetto è la seguente: `delay: { show: 500, hide: 100 }`
`container`	stringa/`false`	`false`	Aggiunge la finestra modale a un elemento specifico. Esempio: `container: 'body'`.
`template`	percorso	`false`	Se fornito, scavalca il template predefinito.
`contentTemplate`	percorso	`false`	Se fornito, recupera il partial e lo include come contenuto all'interno.

Alert

Abbiamo già visto come è possibile utilizzare le finestre di Bootstrap per offrire un feedback agli utenti. AngularStrap ci consente di evidenziarle, applicare un effetto di transizione o permettere agli utenti di rimuoverle. Utilizziamo la direttiva `alert` aggiungendo l'attributo `bs-alert` a un elemento:

```
<button class="btn btn-primary" bs-alert="alert">Show
  Alert</button>
```

L'oggetto del modello definisce non solo il titolo e il contenuto ma anche la classe `context` che utilizzeremo. Può essere `success`, `info`, `warning` o `danger` e modificherà opportunamente il colore dello sfondo e del testo:

```
$scope.alert = {
    title: 'Title',
    content: 'Alert content',
    type: 'success'
};
```

Definiremo con precisione dove verrà aggiunta la finestra di avviso; possiamo utilizzare l'attributo `data-container` per definire un elemento specifico in cui desideriamo visualizzarla. Creiamo un nuovo elemento in cima alla pagina per il contenitore:

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

Aggiungiamolo al pulsante mediante l'attributo `data-container`:

```
<button class="btn btn-primary" bs-alert="alert"
  data-container="#alertContainer">Show Alert</button>
```

Ora quando faremo clic sul pulsante, in cima allo schermo comparirà la finestra di avviso. Sul sito di AngularStrap è disponibile la lista seguente di tutte le opzioni disponibili.

Nome	Tipo	Impostazione predefinita	Descrizione
`animation`	stringa	`am-fade`	Applica un'animazione CSS.
`placement`	stringa	`'top'`	Posiziona il tooltip: `top/bottom/left/right` o qualsiasi altra combinazione come `bottom-left`.
`title`	stringa	`''`	Valore predefinito del titolo.
`content`	stringa	`''`	Valore predefinito del contenuto.
`type`	stringa	`'info'`	Valore predefinito del tipo.
`keyboard`	booleano	`true`	Chiude la finestra di avviso quando si preme il tasto Esc.
`container`	stringa/`false`	`false`	Aggiunge la finestra modale a un elemento specifico. Esempio: `container: 'body'`.
`template`	percorso	`false`	Se fornito, scavalca il template predefinito.

Utilizzare i servizi di AngularStrap

La maggioranza dei moduli inclusi in AngularStrap espone anche i servizi per l'applicazione. Possiamo utilizzarli per mostrare elementi, come finestre modali, finestre di avviso e popover senza essere costretti a ricorrere alle direttive.

Vediamo come utilizzare il servizio `$alert` per mostrare una finestra di avviso dal controller. Ci serviremo della direttiva `ng-click` su un pulsante per avviarla. Creiamo innanzitutto un pulsante e associamo la direttiva `ng-click`:

```
<button class="btn btn-success" ng-click="showAlert()">Alert via
  Service</button>
```

Imposteremo rapidamente la funzione `showAlert()` nel controller. Come prima cosa occorre creare una finestra di avviso utilizzando questo servizio. Inseriamo `$alert` nel controller e creiamo una nuova istanza di una finestra di avviso con il codice seguente:

```
controller('demoCtl', function($scope, $alert){
    var alert = $alert({
        title: 'Alert Title!',
        content: 'Here\'s some content.',
        type: 'danger',
        container: '#alertContainer',
        show: false
    });
});
```

Il costruttore del servizio accetta una funzione hash seguendo lo stesso pattern accettato dalla direttiva. Qui possiamo includere qualsiasi opzione, come il contenitore al quale desideriamo aggiungere la finestra di avviso. Per impostazione predefinita la finestra di avviso che viene creata comparirà automaticamente. Per nasconderla è necessario includere la proprietà `show` e impostarla su `false`.

Infine non rimane che definire l'handler `showAlert()`. L'istanza della finestra di avviso creata dal servizio ci offre tre metodi di cui possiamo servirci: `show()`, `hide()` e `toggle()`. Utilizziamo `show()`:

```
$scope.showAlert = alert.show;
```

Se facciamo clic sul nuovo pulsante, la finestra di avviso comparirà in cima alla pagina (o ovunque abbiamo posizionato il contenitore) e si comporterà come previsto.

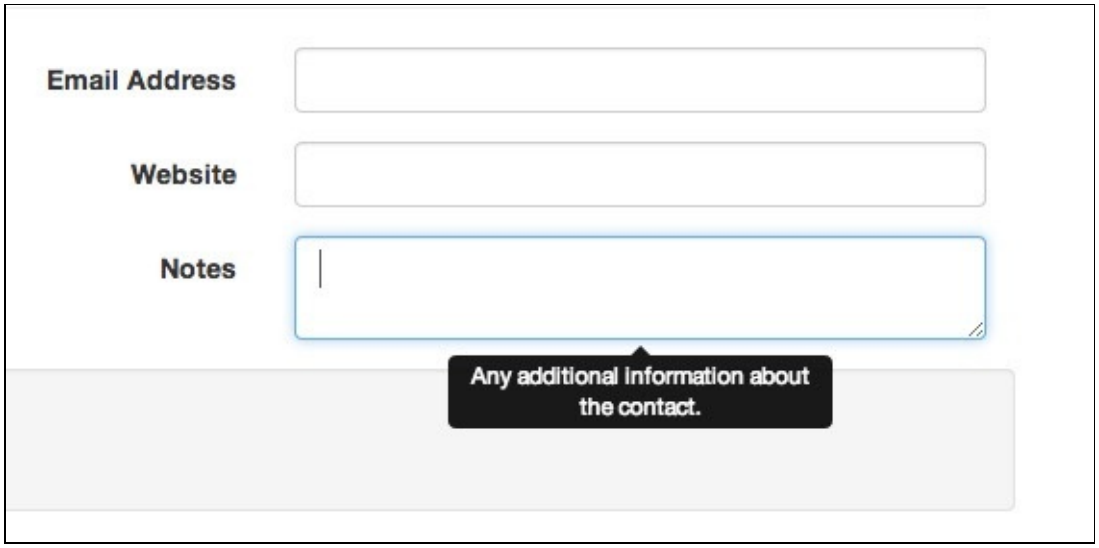
Integrare AngularStrap

Dopo aver visto come è possibile utilizzare molti plug-in, è necessario renderli operativi e vivacizzare l'app di gestione dei contatti. Utilizzeremo i plug-in `tooltip` e `alert` per fornire agli utenti suggerimenti e feedback.

Sostituiamo innanzitutto il testo del suggerimento sotto la casella *Notes* nella vista *Add Contact* con un tooltip:

```
<textarea id="notes" class="form-control" ng-model="contact.notes"
  bs-tooltip data-title="Any additional information about the
    contact." data-trigger="focus"
      data-placement="bottom"></textarea>
```

Invece di creare un modello e legarlo alla direttiva, ha più senso sfruttare l'attributo `data-title` disponibile. In questo caso abbiamo preferito collocarlo sotto e avviarlo al focus.



Potrebbero essere opportune due finestre di avviso. Una è quella preesistente dopo l'aggiunta di un nuovo contatto; l'altra compare dopo aver cancellato un contatto nella vista Index.

Consideriamo prima la finestra di avviso preesistente. Si deve sostituire l'elemento `alert` con il contenitore creato in precedenza:

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

Possiamo inserire il servizio `$alert` e preparare l'istanza della finestra di avviso prima di visualizzarla all'interno della funzione `submit`. Essa avrà la seguente configurazione:

```
var alert = $alert({
    title: 'Success!',
    content: 'The contact was added successfully.',
```

```
    type: ‘success’,
    container: ‘#alertContainer’,
    show: false
});
```

L’aggiungeremo all’`alertContainer` creato in precedenza. In questo caso il context richiede un messaggio di successo; pertanto abbiamo impostato `type` su `success`.

Non ci resta che mostrare la finestra di avviso dopo che è stato creato con successo un contatto:

```
$scope.submit = function(){
    contacts.add($scope.contact);
    $scope.contact = null;
    alert.show();
};
```

Possiamo procedere allo stesso modo quando cancelliamo un contatto per offrire maggiore feedback all’utente. Come prima, collochiamo il contenitore nella posizione in cui vorremmo che comparissero le finestre di avviso nella vista Index:

```
<div id=”alertContainer”></div>
```

Dobbiamo inserire il servizio `$alert` nel controller:

```
.controller(‘indexCtrl’, function($scope, contacts, $alert){
```

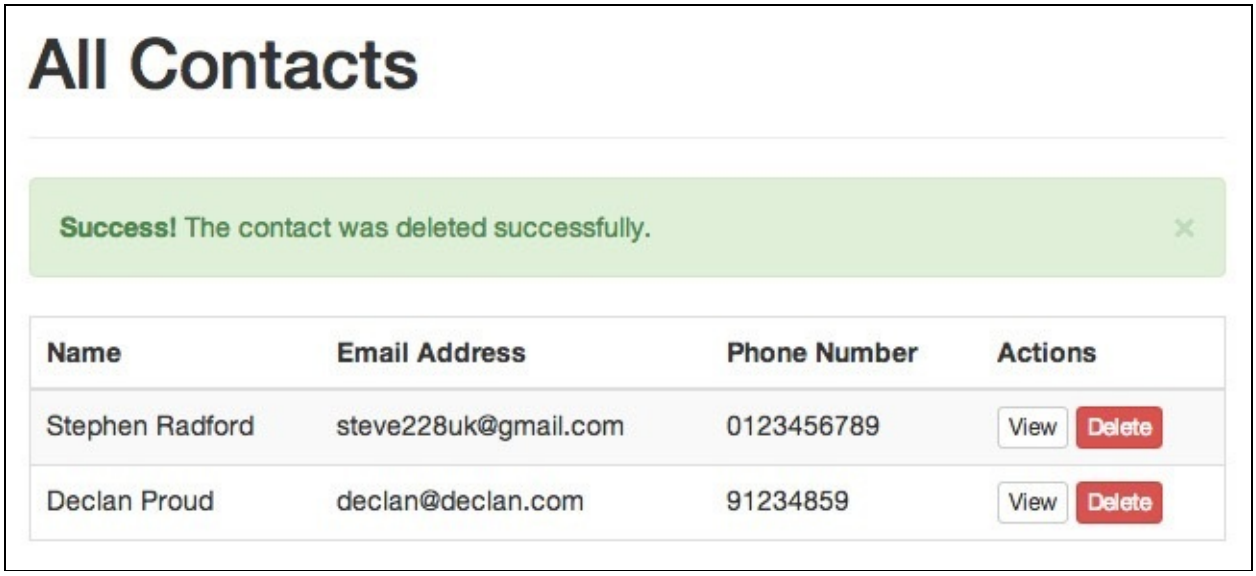
Ora possiamo utilizzare il servizio appena inserito per creare `deletionAlert`:

```
var deletionAlert = $alert({
    title: ‘Success!’,
    content: ‘The contact was deleted successfully.’,
    type: ‘success’,
    container: ‘#alertContainer’,
    show: false
});
```

Per concludere, non ci resta che far comparire la finestra di avviso quando facciamo clic sul pulsante *delete*:

```
$scope.delete = function(index){
    contacts.destroy(index);
    deletionAlert.show();
};
```

Ecco come dovrebbe essere l’output:


All Contacts
Success! The contact was deleted successfully.
Name
Email Address
Phone Number
Actions
Stephen Radford
steve228uk@gmail.com
0123456789
View
Delete
Declan Proud
declan@declan.com
91234859
View
Delete

Quiz

1. Da quale modulo dipende AngularStrap?
2. Qual è il nome del progetto che possiamo utilizzare per le animazioni CSS predefinite?
3. Che cosa è necessario anteporre agli attributi per utilizzarli come opzioni all’interno delle direttive di AngularStrap?
4. Quali sono i quattro sistemi con i quali è possibile attivare un popover o un tooltip?
5. Quali sono i tre metodi disponibili per creare un’istanza mediante il servizio di alert?

Riepilogo

In questo capitolo abbiamo visto come è facile avvalersi dei numerosi moduli integrati in AngularStrap. Anche se non possono utilizzare direttamente JavaScript di Bootstrap, sono tutti componenti legati a questo framework e operano senza soluzione di continuità all'interno dell'applicazione. Abbiamo esaminato solo alcuni dei plug-in disponibili e il modo in cui è possibile utilizzarli mediante le direttive. Talvolta una direttiva non è la soluzione migliore; in alternativa è possibile ricorrere ai servizi compresi in AngularStrap. Nel capitolo seguente vedremo come è possibile connettere l'applicazione al server per recuperare e memorizzare i contatti.

Connessione al server

Finora l’applicazione è ancora interamente front-end ed è, pertanto, piuttosto inutile. È necessario memorizzare i contatti per poterli recuperare in seguito. A questo scopo ci connetteremo a un server che ospiterà un’API RESTful che genera JSON.

Angular offre diverse possibilità per connettersi al server. In questo capitolo ne esamineremo alcune, oltre a introdurre delle alternative che potrete approfondire in seguito.

Non vedremo come sviluppare l’aspetto relativo al lato server, che non rientra negli ambiti della nostra trattazione. Tuttavia è presente nelle risorse scaricabili del libro.

Ecco gli argomenti che affronteremo:

- come estrarre i dati dal server mediante `$http`;
- come utilizzare e dove trovare `ngResource`;
- le alternative della community tra cui `RestAngular`;
- integrazione nell’applicazione della nuova connessione con il server.

Mettiamoci all’opera.

Connettersi con $http

Angular include già alcuni metodi di basso livello per recuperare e inviare i dati. Se avete utilizzato `$.ajax`, `$.post` o `$.get` in jQuery, sarete a vostro agio.

Come sapete, questi metodi sono disponibili sotto forma di un servizio che è possibile inserire nei controller o nei servizi. Di seguito potete osservare il servizio `$http` inserito nel controller:

```
.controller('indexCtrl', function($scope, contacts, $alert,
  $http){

})
```

Il servizio include alcuni metodi che funzionano con tutti i verbi del protocollo `REST`. I metodi seguenti sono disponibili all'interno di `$http`.

- `$http.get()`: accetta un URL e un oggetto `config` facoltativo. Esegue una richiesta HTTP `GET`.
- `$http.head()`: accetta un URL e un oggetto `config` facoltativo. Esegue una richiesta HTTP `HEAD`.
- `$http.post()`: accetta un URL, un oggetto data e un oggetto `config` facoltativo. Esegue una richiesta HTTP `POST`.
- `$http.put()`: accetta un URL, un oggetto data e un oggetto `config` facoltativo. Esegue una richiesta HTTP `PUT`.
- `$http.delete()`: accetta un URL e un oggetto `config` facoltativo. Esegue una richiesta HTTP `DELETE`.
- `$http.jsonp()`: accetta un URL e un oggetto `config` facoltativo. Il nome della funzione di callback dovrebbe essere la stringa `JSON_CALLBACK`.
- `$http.patch()`: accetta un URL, un oggetto data e un oggetto `config` facoltativo. Esegue una richiesta HTTP `PUT`.

Tutti questi metodi sono scorciatoie per la funzione principale `$http()`, che accetta un argomento: un oggetto. Le funzioni prima menzionate impostano il verbo e/o il tipo di contenuto che intendiamo recuperare.

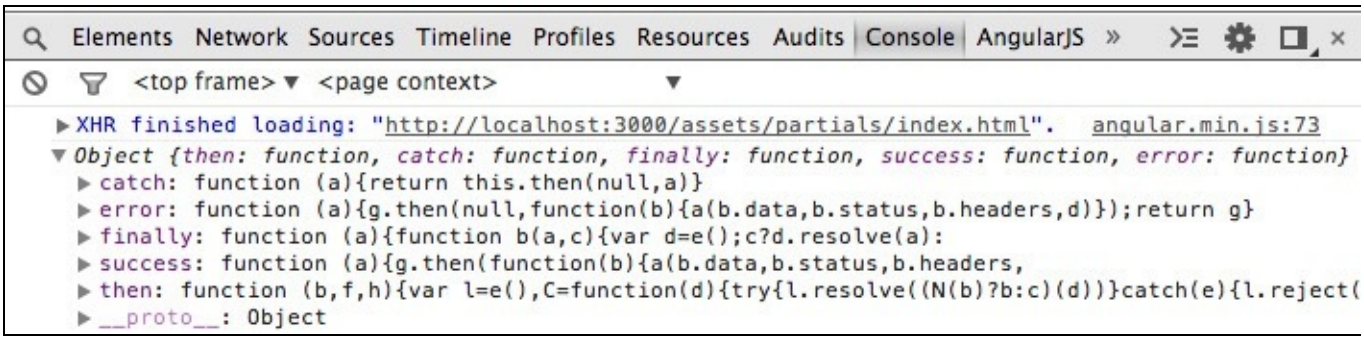
Per esempio, i due frammenti di codice seguenti sono identici, ma il secondo è molto più leggibile:

```
$http({
    method: 'GET',
    url: 'http://localhost:8000'
});
```

```
$http.get('http://localhost:8000');
```

Recuperare i dati è facile e Angular beneficia dei pattern Promises sviluppati da Promises/A+ e resi popolari da jQuery. Il pattern ci consente di determinare facilmente se l'URL al quale abbiamo avuto accesso ha restituito una risposta positiva o ha generato un errore.

Può sembrare complesso, ma in sostanza si tratta di una serie di metodi che possiamo concatenare per creare facilmente un approccio `try/catch` per le chiamate asincrone. Se racchiudiamo `console.log` in `$http.get()`, vedremo tutti i metodi disponibili visualizzati nella console.



Tutti i metodi accettano un'unica funzione di callback, con l'eccezione di `then`, che accetta due metodi, uno per il successo dell'operazione e un altro per l'errore. Vediamo come possiamo utilizzarli. Nel controller dell'index, sostituiamo alla riga `contacts.get()` il seguente codice:

```
$http.get('http://localhost:8000')
.success(function(data){
    $scope.contacts = data;
})
.error(function(){
    window.alert('There was an error!');
});
```

Angular si occuperà del resto. La funzione di callback per il metodo `success` viene eseguita quando viene restituito un codice di status `2xx`; altrimenti viene eseguito un errore.

Avremmo potuto abbreviare il codice precedente utilizzando il metodo `then` e due funzioni di callback, nel seguente modo:

```
$http.get('http://localhost:8000')
.then(function(result){
    $scope.contacts = result.data;
}, function(){
    window.alert('There was an error!');
});
```

Questo fa però risparmiare poco codice ed è meno leggibile per altri sviluppatori che potrebbero non conoscere AngularJS. Notate che `data` non è l'argomento passato alle funzioni di callback all'interno del metodo `then`; otteniamo invece un oggetto contenente dati, stato e header.

Inviare i dati

Come il recupero dei dati, il loro invio mediante `$http` è molto facile e simile all'implementazione di jQuery. La funzione `$http.post()` si comporta nello stesso modo di `$http.get()` ma accetta un secondo parametro: un hash contenente tutti i dati che intendiamo inviare al server:

```
$http.post('http://localhost:8000', {
    name: 'Declan Proud',
    email: 'declan@example.com',
    ...
});
```

Analogamente, il metodo `post` restituisce anche una promise con gli stessi metodi esaminati prima:

```
$http.post('http://localhost:8000', {
    name: 'Declan Proud',
    email: 'declan@example.com',
    ...
})
.success(function(){
    ...
})
.error(function(){
    ...
});
```

Connettersi con ngResource

Gli helper della connessione di basso livello come `$http` sono ottimi per connessioni singole, ma diventano ben presto scomodi per la gestione di un intero progetto. Fortunatamente Angular offre un altro sistema con il quale accedere ai dati lato server grazie a un modulo facoltativo chiamato `ngResource`.

Includere ngResource

Come `ngRoute`, il modulo `ngResource` si trova al link *Extras* nella finestra modale di download all'indirizzo https://angularjs.org/. Scaricatelo e trascinatelo nella directory *js* del progetto. Includetelo dopo Angular nel file radice HTML:

```
<script type=””text/javascript”” src=”/assets/js/angular-
  resource.min.js”></script>
```

Verificate che il modulo `contactsMgr` sappia che `ngResource` è una dipendenza:

```
angular.module('contactsMgr', ['ngRoute', 'ngSanitize',
  'mgcrea.ngStrap', 'ngResource'])
```

Configurare ngResource

Il modulo espone il servizio `$resource` che è possibile inserire nei controller o nei servizi. I metodi inclusi sono di un livello più alto, e infatti utilizzano `$http` per interagire con il server. Inseriamo il servizio `$resource` nel servizio di gestione dei contatti creato nel Capitolo 7 e vediamo come connettersi al server:

```
.factory('Contact', function ContactFactory($resource){
...
})
```

Il servizio include un metodo che utilizzeremo per impostare la connessione. Restituisce alcune funzioni sotto forma di un oggetto `resource`. Saranno queste funzioni a recuperare e a inviare i dati al e dal server.

Vediamo come possiamo servirci di questo metodo singolo e ciò che restituisce:

```
var Resource = $resource('http://localhost:8000/contacts/:id',
  {id: '@id'});
```

Il codice restituirà l'oggetto seguente, consistente in azioni che è possibile utilizzare per recuperare, salvare o cancellare i dati:

```
{
    'get': {method:'GET'},
    'save': {method:'POST'},
```

```
    'query': {method:'GET', isArray:true},
    'remove': {method:'DELETE'},
    'delete': {method:'DELETE'}
};
```

Il primo parametro è la radice della risorsa sul server. Per esempio, se stessimo sviluppando un sistema di blogging, avremmo alcune risorse come post, tag e autori. È anche possibile aggiungere dei segnaposto, come potremmo fare durante la creazione di una route.

Il secondo parametro è un hash che include i valori predefiniti dei segnaposto. Se il valore predefinito di un segnaposto presentasse il prefisso contenente il simbolo `@`, quel valore sarebbe recuperato dall'oggetto `data` che viene passato quando accediamo al server.

Possiamo anche passare un terzo parametro per estendere le azioni predefinite che vengono restituite. Aggiungiamo un metodo `update` che utilizzerà il verbo `PUT` per aggiornare un contatto esistente sul server:

```
var Resource = $resource('http://localhost:8000/contacts/:id',
  {id: '@id'}, {
    update: {method: 'PUT'}
});
```

Come potete vedere, si tratta di un oggetto JS standard in cui è possibile definire molteplici azioni personalizzate. Esistono alcuni elementi che è possibile includere all'interno dell'oggetto `configuration` associato all'azione, ma probabilmente sarà necessario impostare soltanto `method` e `isArray`. La proprietà `method` seleziona quale verbo HTTP è necessario utilizzare (in questo caso `PUT`), e `isArray` è un booleano che serve a indicare a `ngResource` se il server restituirà un unico item o un array di item.

Richiedere al server

Siamo riusciti a configurare `ngResource`; ora dobbiamo solo attivarlo, operazione molto facile. Si tratta di utilizzare una delle azioni restituite dall'oggetto `$resource`.

Intendiamo recuperare tutti i contatti: il metodo `query` sembra la soluzione ideale. Utilizza il metodo `GET` e la proprietà `isArray` viene verificata:

```
.factory('Contact', function ContactFactory($resource){
    var Resource = $resource('http://localhost:8000/:id', {id:
  '@id'}, {
        update: {method: 'PUT'}
    });
    return {
        get: function(){
            return Resource.query();
        },
        ...
```

```
    };
})
```

Ecco fatto! Non dobbiamo preoccuparci dell'unwrapping delle promise perché questo aspetto viene gestito automaticamente da `ngResource`. Non ci resta che rieseguire la chiamata `$http` in `indexCtl` al metodo `Contact.get()`:

```
$scope.contacts = Contact.get();
```

Siccome non utilizzeremo più un array di tipo hard-code, non possiamo accedere a singoli contatti servendoci di un indice. La maggior parte delle API restituisce un ID degli item e questa non fa eccezione. Modifichiamo `{{$index}}` da `ng-repeat` utilizzato nel link, per avvalerci dell'ID invece che dei singoli contatti; dovrebbe trovarsi all'incirca a riga 23 nel file `partials/index.html`:

```
<a href="/contact/{{contact.id}}" class="btn btn-default btn-
  xs">View</a>
```

È necessario modificare il metodo `find` nel servizio dei contatti per recuperare un unico contatto in base all'ID a esso assegnato. Abbiamo già impostato la risorsa per ammettere un parametro `id`; non ci resta che popolarlo quando utilizzeremo il metodo `get` della risorsa:

```
find: function(id){
    return Resource.get({id: id});
},
```

Abbiamo anche modificato il nome del parametro da `index` a `id` per renderlo più leggibile se qualcun altro dovesse lavorare in seguito al progetto.

Inviare al server

Questo è ciò che viene recuperato dal server, ma in che modo possiamo creare un nuovo contatto o aggiornarne uno esistente con `ngResource`? Esaminiamo prima come crearne uno e osserviamo come `ngResource` gestisce questo aspetto. Modificheremo il metodo `create` nel servizio di gestione dei contatti per restituire una nuova istanza della risorsa:

```
create: function(){
    return new Resource();
},
```

Questo diventerà il modello nella vista *Add Contact*. Si comporta come previsto, ma dà accesso a un metodo `$save` per passarlo al server. Chiamiamo il nuovo metodo `create` e assegniamolo al modello `contact` all'interno di `addCtl`:

```
$scope.contact = Contact.create();
```

Tutto dovrebbe comportarsi come previsto quando carichiamo la vista; ora è necessario sostituire al metodo `contacts.set`, ormai non più attivo, l'handler `submit` per la nuova funzione `$save` offerta dalla risorsa:

```
$scope.submit = function(){
    $scope.contact.$save();
    $scope.contact = Contact.create();
    alert.show();
};
```

Abbiamo anche modificato `$scope.contact`, che non viene più eliminato ma recupera una nuova istanza della risorsa dal servizio.

Analogamente possiamo utilizzare la stessa azione `update` creata in precedenza per salvare le modifiche di un contatto esistente. A questo scopo è necessario servirsi di un evento personalizzato in modo che il controller sappia quando salviamo le modifiche all'interno della direttiva modificabile. Gli eventi personalizzati si comportano proprio come gli omologhi nativi JavaScript, al pari di `click` e `mouseover`. È possibile mettersi in ascolto ed eseguire delle azioni quando vengono attivati.

Per creare un evento personalizzato, utilizziamo il metodo `$scope.$emit`. Accetta due parametri: il nome dell'evento e l'array di parametri che intendiamo passare al listener. In questo caso non è necessario passare alcun parametro; chiamiamo semplicemente l'evento salvato e inseriamolo nella funzione `$scope.save` all'interno della direttiva modificabile:

```
$scope.$emit('saved');
```

Ascoltare un evento è altrettanto facile: è sufficiente utilizzare il metodo `$scope.$on`. Esso accetta due parametri: il primo è il nome dell'evento da ascoltare, il secondo è la funzione dell'handler. Aggiungiamo il seguente codice al controller `contactCtl`:

```
$scope.$on('saved', function(){
    ...
});
```

Se avessimo passato dei parametri all'evento, sarebbero stati accessibili come parametri nel listener degli eventi e il primo sarebbe sempre stato l'evento JS.

Il listener eseguirà il metodo `$update` sul modello `contact`. Tuttavia, siccome l'evento viene emesso prima che il modello abbia concluso l'aggiornamento, dobbiamo spingerlo alla fine dello stack o della coda corrente. Se conoscete JavaScript, saprete che è possibile utilizzare `setTimeout`. È proprio ciò che faremo, ma invece di servirci di `setTimeout`, opteremo per il servizio wrapper di Angular: `$timeout`.

Dobbiamo inserirlo nel controller:

```
.controller('contactCtrl', function($scope, $routeParams, Contact,
  $timeout){
...
})
```

Poi si tratta semplicemente di utilizzarlo come `setTimeout`:

```
$scope.$on('saved', function(){
    $timeout(function(){
        $scope.contact.$update();
    }, 0);
});
```

Ora se modificherete un contatto e farete clic sul pulsante *save*, i dati verranno salvati sul server.

Cancellare i contatti

Infine, non ci resta che impostare il pulsante *delete*. Modifichiamo il metodo all'interno del servizio dei contatti:

```
destroy: function(id){
    resource.delete({id: id});
}
```

In questo caso chiamiamo il metodo `delete` sulla risorsa; volendo avremmo potuto utilizzare `remove` che svolge la stessa operazione. Abbiamo nuovamente modificato il nome del parametro `index` per renderlo più leggibile.

Dobbiamo ancora compiere un'operazione in questa istanza: aggiornare la funzione `delete` nel controller `indexCtl`. Analizziamo allora il metodo finito:

```
$scope.delete = function(index){
    Contact.destroy($scope.contacts[index].id);
    $scope.contacts.splice(index, 1);
    alert.show();
};
```

Siccome dobbiamo sia eseguire il ping del server sia rimuoverlo dall'array locale, è necessario continuare a utilizzare `index`. Nella chiamata `Contact.destroy()` accediamo al contatto opportuno e recuperiamo il suo ID. Lo cancelliamo direttamente dall'array locale utilizzando il metodo JS nativo `splice` per fare in modo che tutto sia sincronizzato.

Gestione degli errori

È possibile gestire gli errori come faremmo con `$http`. Tutte le azioni che abbiamo esaminato accettano due funzioni di callback: una per il successo e l'altra per l'errore. Ecco il metodo `get` con incluse entrambe le funzioni di callback:

```
return Resource.get({id: id}, function(){
    window.alert('Success!');
}, function(){
    window.alert('Error!');
});
```

Così possiamo informare l'utente che esiste un problema o eseguire altre azioni se necessario. Siccome lo facciamo da un servizio, è opportuno includere due parametri per consentire che queste funzioni di callback vengano impostate dal controller quando viene chiamato il metodo in questione:

```
find: function(id, success, error){
    return Rgesource.get({id: id}, success, error);
},
```

Sistemi alternativi di connessione

Abbiamo già esaminato alcuni sistemi con i quali è possibile connettersi al server e configurare l'app per sfruttare `ngResource`. Esistono altri moduli che è possibile utilizzare per connettersi a un server; ne considereremo rapidamente due.

RestAngular

RestAngular è un progetto della community che offre un servizio per connettersi ad API RESTful, come `ngResource`. Presenta alcune differenze significative che è opportuno conoscere.

L'aspetto più importante da tenere presente è che RestAngular utilizza promise proprio come `$http`. Pertanto si accede a un pattern per determinare se una chiamata ha avuto successo o meno, ma questo significa che esistono passi aggiuntivi da compiere e che non è possibile assegnarlo semplicemente a un modello.

Anche se è necessario scrivere un po' più di codice a causa dell'utilizzo delle promise, non dovrete trascrivere i segnaposto seguendo il pattern REST; RestAngular lo farà al posto vostro.

Preferiamo `ngResource`. RestAngular richiede alcuni passaggi in più, mentre `ngResource` funziona perfettamente con i nostri servizi. Tuttavia vale sempre la pena sperimentare ciò che potrebbe rivelarsi efficace per le proprie esigenze e quindi consigliamo di provare anche RestAngular.

Utilizzare RestAngular

È possibile scaricare RestAngular dal sito `https://github.com/mgonto/restangular` e includerlo come qualunque altro modulo. Vedremo rapidamente come impostarlo e in che modo è possibile ottenere una lista dei contatti.

Apprezziamo la capacità di RestAngular di impostare un URL di base. È possibile definirlo nel metodo globale `config` del modulo tramite il servizio `RestangularProvider`:

```
.config(function($routeProvider, $locationProvider,
  RestangularProvider){
    RestangularProvider.setBaseUrl('http://localhost:8000/');
})
```

Dopo aver impostato l'URL di base, possiamo utilizzare RestAngular nominando la risorsa alla quale intendiamo accedere e un metodo di RestAngular:

```
Restangular.all('contacts').then(function(contacts){
    $scope.contacts = contacts;
});
```

Come potete notare, RestAngular si basa sul pattern delle promise utilizzato da `$http` ed è necessario eseguire l'unwrapping per assegnare al modello i dati restituiti.

Leggete la documentazione di RestAngular su GitHub per una lista completa dei metodi che è possibile utilizzare.

Firebase

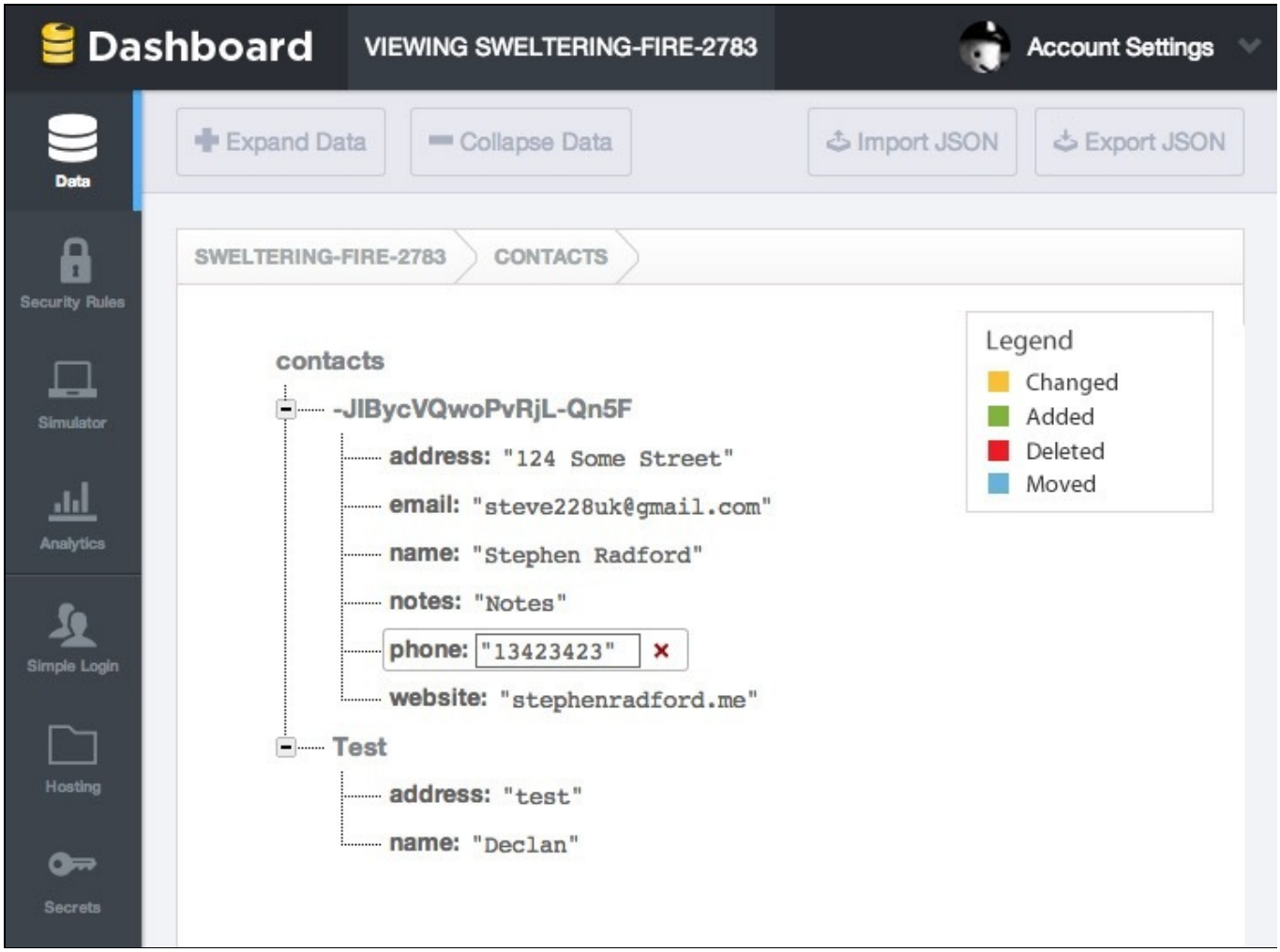
Firebase è un servizio relativamente nuovo che consente di creare facilmente un'applicazione in tempo reale senza scrivere una sola riga di codice di back-end. Quando si opera con Angular, l'azienda offre un'utile libreria di helper per sincronizzare facilmente i dati con il suo servizio.

La dashboard di Firebase consente di visualizzare i dati in una struttura ad albero comprimibile simile a quella mostrata successivamente.

Dopo aver creato un account all'indirizzo `http://firebase.com` e configurato l'app, è giunto il momento di impostare AngularFire. Dobbiamo includere il client Firebase e AngularFire, che si possono trovare all'indirizzo `http://angularfire.com`.

È molto facile recuperare i dati da Firebase, considerando che tutto avviene in tempo reale e qualsiasi modifica compiuta altrove si riflette automaticamente nell'applicazione. Come la maggior parte dei moduli, AngularFire espone un servizio, in questo caso `$firebase`. Possiamo inserirlo nei controller, nelle direttive o nei servizi, nel seguente modo:

```
.factory('Contact', function ContactFactory($resource, $firebase){
...
})
```



È possibile avviare la connessione con Firebase utilizzando il suo client JS:

```
var contacts = new
Firebase(“https://<yourbase>.firebaseio.com/contacts”);
```

Il servizio AngularFire facilita il recupero dei dati da Firebase. Questo è ciò che potremmo fare con il metodo `get` del nostro servizio:

```
get: function(){
    return $firebase(contacts);
},
```

Anche aggiungere i contatti è molto facile. Dopo aver recuperato i dati, otteniamo l’accesso ai metodi `$add`, `$remove` e `$update`:

```
$firebase(contacts).$add({
    name: ‘Declan Proud’,
    ...
});
```

Se aprite il pannello di controllo di Firebase e aggiungete manualmente un contatto, vedrete che questo comparirà automaticamente nella lista dei contatti. Ovviamente è esagerato per un’app come questa che gestisce i contatti, ma apre infinite possibilità per client di chat, notifiche e altro ancora senza essere costretti a scrivere una sola riga di codice di back-end.

Quiz

1. Che tipo di oggetto restituisce il metodo `$http`?
2. In che modo è possibile ottenere un array di contatti e assegnarli a un modello con `$http`?
3. Che cosa significa il simbolo `@` in una configurazione di parametri predefinita?
4. Indicate le due differenze principali tra ngResource e RestAngular.
5. In che modo Firebase crea l’applicazione?

Riepilogo

In questo capitolo abbiamo trasformato l'applicazione da un'app front-end che utilizzava dati di tipo hard-code in una che si interfaccia con un'API per memorizzare e recuperare le informazioni. Abbiamo scoperto come è flessibile Angular esaminando quattro diversi metodi per connettersi a un server.

I servizi di basso livello come `$http` sono ottimi in alcuni casi, ma per lo sviluppo di un'applicazione completa, abbiamo visto che è opportuno utilizzare qualcosa di più raffinato. Moduli come `ngResource` mantengono la base di codice gestibile e rispondente al principio DRY (*Don't Repeat Yourself*).

Nel prossimo capitolo approfondiremo questo concetto analizzando due runner di codice: Grunt e gulp.

I task runner

Il progetto ha un bell’aspetto, ma non è efficiente. Abbiamo incluso dieci file JavaScript, che comportano dieci richieste di rete, senza considerare il foglio di stile. Ciò significa che la pagina impiegherà più tempo a caricarsi. Al termine del caricamento, il browser dovrà recuperare ogni file JavaScript e compilarlo.

Potremmo prendere manualmente questi file e concatenarli in uno solo. Tuttavia lavorando al progetto è probabile che continueremo a effettuare altre modifiche, e sarebbe irritante ripetere continuamente questo processo.

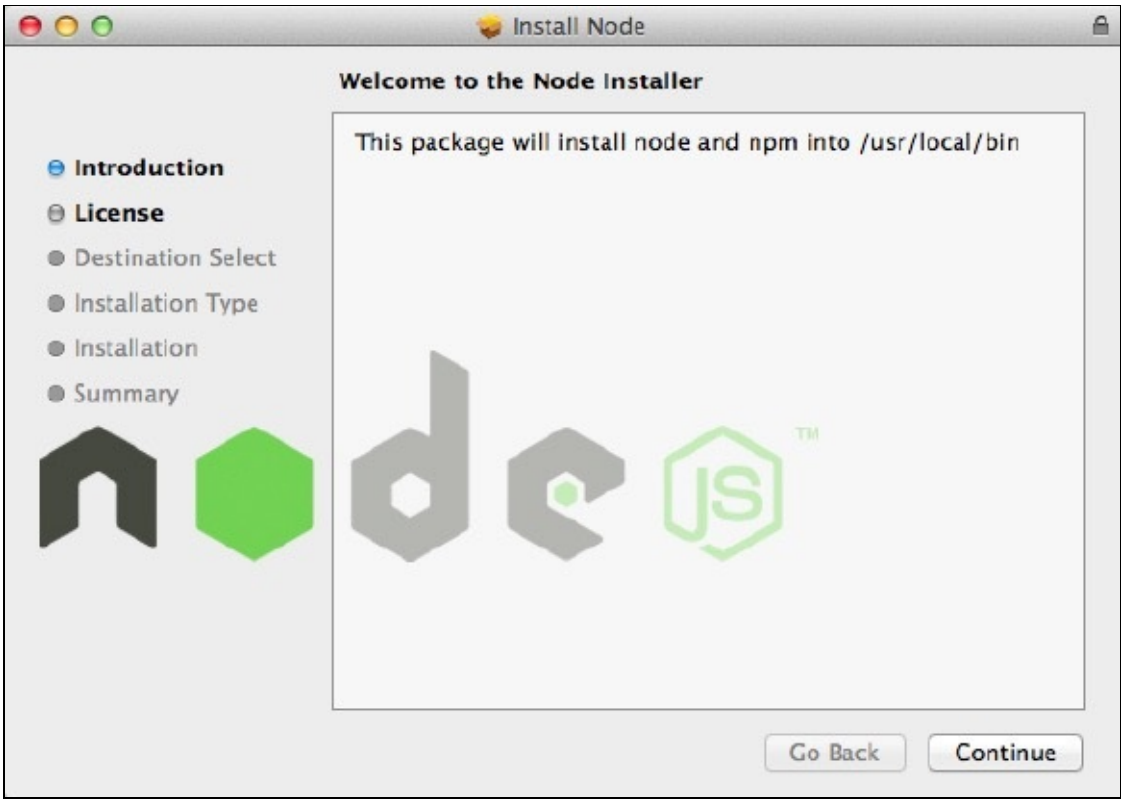
I *task runner* sono un ottimo sistema per automatizzare attività noiose. Non dovremo più concatenare e minificare manualmente, ma avremo un’installazione che controllerà le modifiche nei file e li creerà automaticamente.

Anche se non li avete mai utilizzati, probabilmente avrete sentito parlare di task runner come Grunt e gulp. In questo capitolo li metteremo entrambi alla prova per concatenare e minificare i file JavaScript in un unico file.

Installare Node e NPM

Sia Grunt sia gulp si basano su Node e il suo *Node Package Manager* (NPM). Se avete già installato e configurato Node, tralasciate questo paragrafo. Esamineremo l'installazione su Mac, ma il processo è simile per Windows. Gli utenti Linux dovranno compilare dal file sorgente o visitare `https://github.com/joyent/node/wiki/Installing-Node.js-via-package-manager` per installarlo tramite un gestore di pacchetti.

Andate all'indirizzo `http://nodejs.org/download/` e scaricate l'installer apposito per la piattaforma. Apritelo, accettate il contratto di licenza e completate la procedura.



Se tutto si svolge come previsto, dovreste veder comparire un messaggio simile:

Node was installed at

/usr/local/bin/node

npm was installed at

/usr/local/bin/npm

Verificate che `/usr/local/bin` si trovi in `$PATH`. Se non siete sicuri che si trovi nel vostro percorso, eseguite questo comando da terminale:

```
echo $PATH
```

Cercate `/usr/local/bin`. Se non lo trovate, aggiungete quanto segue a `~/.bash_profile` oppure `~/.zshrc`:

```
export PATH=/usr/local/bin:$PATH
```

Alla fine della procedura Node e NPM saranno installati. Potrete accedere ai comandi `node` e `npm`, e vi sarà possibile installare Grunt o gulp nel progetto.

> **NOTA**
>
> Potrebbe essere necessario riavviare la sessione del terminale per far funzionare tutto come previsto.

Utilizzare Grunt

Dopo aver installato Node, sfrutterete Grunt. La configurazione avviene in tre fasi: lo strumento da riga di comando, l'installazione locale di Grunt nel progetto e la configurazione del Gruntfile.

Installare l'interfaccia a riga di comando

Installare la *command-line interface* (CLI) è molto facile grazie a NPM. È sufficiente eseguire quanto segue nel terminale. Il flag `-g` verificherà l'installazione globale di Grunt:

```
npm install -g grunt-cli
```

A seconda dei permessi, potreste eseguirlo come root. Su sistemi OS X o basati su *nix, si tratta di eseguirlo con il prefisso `sudo`. In Windows, è necessario aprire la shell di comando come amministratori.

Al termine dell'installazione, il comando `grunt` sarà disponibile e verrà aggiunto al percorso del sistema, e potrà venire eseguito da qualsiasi directory.

Installare Grunt

Per utilizzare Grunt nel progetto è necessario aggiungere due file: `package.json` e `Gruntfile.js`.

Il file `package.json` non viene utilizzato da Grunt, ma da NPM. Indica al gestore quali pacchetti servono al progetto quando eseguiamo l'installer. `Gruntfile.js` è ciò che configura Grunt. Fornisce diverse indicazioni al task runner, da quali file considerare a quali attività eseguire.

Creare un file package.json

Creiamo il file `package.json` affinché NPM sappia quali file recuperare. Di seguito il file JSON completato. Il Node Package Manager ci consentirà di crearlo facilmente eseguendo il comando `npm init`:

```
{
  "name": "ContactsMgr",
  "version": "1.0.0",
  "description": " A simple contacts manager in AngularJS +
    Bootstrap",
  "dependencies": {
    "grunt": "~0.4.1",
```

```
    "grunt-contrib-uglify": "~0.2.0",
    "grunt-contrib-watch": "~0.5.3"
  }
}
```

Come potete notare, si tratta di un oggetto JSON standard con alcune proprietà chiave impostate. Il nome, la versione e la descrizione sono richiesti, ma utilizzati soltanto quando distribuiamo il progetto sotto forma di un pacchetto su NPM. La proprietà `dependencies` è il punto in cui avvengono processi interessanti.

Grunt si trova in cima alla lista delle dipendenze. Gli altri pacchetti che abbiamo incluso svolgeranno la parte più rilevante del lavoro. Il pacchetto `grunt-contrib-uglify` concatenerà e minificherà i file JS e l'ultimo pacchetto nella lista controllerà i file alla ricerca di modifiche ed eseguirà attività specifiche.

NOTA

È importante ricordare che il nome del pacchetto non deve contenere spazi o caratteri speciali.

Creare il file Gruntfile.js

Gruntfile è molto importante. Consideratelo come il manuale di istruzioni di Grunt, che senza di esso non saprebbe che cosa fare. Tutta la configurazione avviene all'interno della seguente funzione wrapper di Grunt:

```
module.exports = function(grunt){

};
```

NOTA

È importante che `Gruntfile.js` venga salvato nella radice dove si vuole eseguire Grunt e che il nome del file inizi con la G maiuscola.

La maggior parte dei plug-in richiederà l'utilizzo del metodo `initConfig` di Grunt, ed è ciò che utilizzeremo con il plug-in `uglify`:

```
module.exports = function(grunt){

    grunt.initConfig({
       ...
    });

};
```

All'interno possiamo configurare i plug-in utilizzando il nome come chiave. Possiamo anche ottenere le informazioni direttamente dal file `package.json`, come il nome da utilizzare nelle attività:

```
grunt.initConfig({
    pkg: grunt.file.readJSON('package.json')
});
```

Questo codice caricherà il file `JSON` e lo assegnerà alla chiave `pkg`, consentendoci di accedere a qualsiasi informazione impostata in precedenza avvalendoci della sintassi standard per i template (`<%= %>`).

Quando configureremo l'attività `uglify`, imposteremo due proprietà: `options` e `build`. La proprietà `options` consente di definire elementi come i banner che intendiamo includere nel file compilato, creare una mappa dei file sorgente o se intendiamo concatenare per il debugging.

La proprietà `build` è il target e possiamo attribuirle qualsiasi nome desideriamo. Per esempio, una potrebbe chiamarsi `dev` e un'altra `production` con diverse opzioni. Può accettare le proprietà `src` e `dest`, che ci consentono di impostare quali file vengono inclusi e quali generati. È anche possibile definire un altro oggetto `options`, utile quando intendiamo utilizzare molteplici target. Ecco l'attività `uglify` con l'hash `options`:

```
grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    uglify: {
        options: {
            banner: '/*! <%= pkg.name %> <%=
              grunt.template.today("yyyy-mm-dd") %> */\n'
        }
    }
});
```

In questo caso abbiamo incluso il banner nell'oggetto `options`. Siccome il file `package.json` è stato convertito in un oggetto JS, è sufficiente utilizzare la sintassi standard per accedere al nome. Grunt comprende anche due helper di cui possiamo avvalerci. Vedrete che in questo caso ricaviamo la data odierna, ma è anche possibile utilizzare il metodo `grunt.template.date` per formattare un timestamp JS. Può essere utile quando intendiamo includere la data in un banner o in un nome file.

Ora impostiamo il target. La proprietà `src` può essere una stringa o un array. In questo caso, siccome utilizziamo alcuni file JS, dovremo servirci di un array. La proprietà `dest` è il percorso relativo al file che vogliamo che Grunt crei per impostazione predefinita, ma può anche essere modificato mediante il metodo `grunt.file.setBase`. Abbiamo aggiunto l'oggetto `build` e impostato le proprietà `src` e `dest`:

```
grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    uglify: {
        options: {
            banner: '/*! <%= pkg.name %> <%=
              grunt.template.today("yyyy-mm-dd") %> */\n'
        },
        build: {
            src: [
                'assets/js/vendor/jquery.js',
```

```
                'assets/js/vendor/bootstrap.js',
                'assets/js/vendor/angular.js',
                'assets/js/vendor/angular-animate.js',
                'assets/js/vendor/angular-resource.js',
                'assets/js/vendor/angular-route.js',
                'assets/js/vendor/angular-sanitize.js',
                'assets/js/vendor/angular-strap.js',
                'assets/js/vendor/angular-strap.tpl.js',
                'assets/js/controller.js'
            ],
            dest: 'assets/js/build/<%= pkg.name %>.js'
        }
    }
});
```

Abbiamo utilizzato il nome del pacchetto come nome file e tutti i file minificati sono stati sostituiti con le versioni non minificate. Poiché minificheremo tutto, è necessario utilizzare le versioni di sviluppo dei file per evitare problemi durante la compilazione.

L'ordine dei file nell'array `src` è lo stesso con cui saranno inclusi nel file di destinazione. Siccome abbiamo bisogno che jQuery sia incluso prima di Angular e Angular prima dei moduli, è importante impostarli bene.

Il plug-in è compilato, ma Grunt non sa che intendiamo utilizzare l'attività `uglify` che abbiamo scaricato in precedenza da NPM. A questo scopo è necessario utilizzare il metodo `loadNpmTasks`:

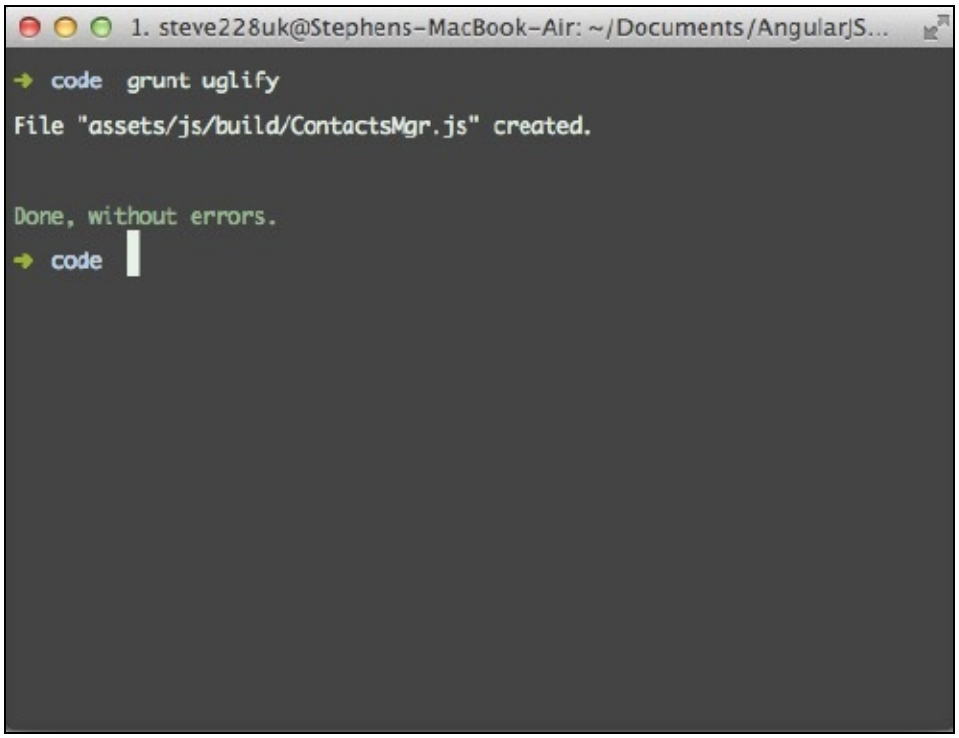
```
grunt.loadNpmTasks('grunt-contrib-uglify');
```

Si inserisce nella funzione `module.exports` e fa in modo che il Gruntfile completo sia come il seguente:

```
module.exports = function(grunt){

    grunt.initConfig({
        pkg: grunt.file.readJSON('package.json'),
        uglify: {
            options: {
                banner: '/*! <%= pkg.name %> <%=
                  grunt.template.today("yyyy-mm-dd") %> */\n'
            },
            build: {
                src: [
                    'assets/js/vendor/jquery.js',
                    'assets/js/vendor/bootstrap.js',
                    'assets/js/vendor/angular.js',
                    'assets/js/vendor/angular-animate.js',
                    'assets/js/vendor/angular-resource.js',
                    'assets/js/vendor/angular-route.js',
                    'assets/js/vendor/angular-sanitize.js',
                    'assets/js/vendor/angular-strap.js',
                    'assets/js/vendor/angular-strap.tpl.min.js',
                    'assets/js/controller.js'
                ],
                dest: 'assets/js/build/<%= pkg.name %>.js'
            }
        }
    });

    grunt.loadNpmTasks('grunt-contrib-uglify');

};
```

Eseguire Grunt

Ora possiamo eseguire l'attività `grunt uglify` che genererà il file `ContactsMgr.js` completo.



Se sostituite ai dieci script il nuovo file nella radice `index.html` e caricate l'applicazione, noterete che non funziona nulla e vedrete il seguente errore della console:

```
Error: [$injector:unpr] Unknown provider: a
```

Come parte del processo di minificazione, i nomi delle variabili vengono sostituiti con le loro versioni abbreviate. Sappiamo che Angular si basa molto sulla *dependency injection*, che osserva il nome della variabile per inserire il servizio corretto nei controller e nelle direttive.

Fortunatamente Angular offre una soluzione rapida e facile. Si tratta di cambiare le funzioni nelle quali inseriamo i servizi, con gli array contenenti i nomi dei servizi che intendiamo inserire e la funzione come suoi valori. Ecco come appare `config` del modulo:

```
.config([‘$routeProvider’, ‘$locationProvider’,
  function($routeProvider, $locationProvider){

...

}])
```

Finché la funzione è l'ultima nell'array, Angular esaminerà le variabili e utilizzerà il servizio corrispondente dell'array. Grunt non modificherà il valore degli item

nell’array poiché si tratta di stringhe e non di nomi di variabili; è quindi importante che l’ordine nell’array corrisponda a ciò che viene inserito nella funzione.

È sufficiente aggiungere questi wrapper per l’array nel file `controller.js` poiché a tutte le librerie e ai moduli che abbiamo utilizzato è già stato applicato questo processo.

Ricordate che anche i controller nelle direttive devono utilizzare la notazione degli array.

Impostare watch

Siamo riusciti a configurare Grunt per compilare i file e funziona benissimo. Tuttavia non sarebbe più un processo automatico se dovessimo eseguire `grunt uglify` ogni volta che introduciamo una modifica nei file JavaScript. Grunt può tenere d’occhio per noi questi aspetti ed eseguire automaticamente alcune attività quando apportiamo delle modifiche ai file.

A questo scopo utilizziamo il pacchetto `grunt-contrib-watch` che abbiamo recuperato prima da NPM. La configurazione è molto semplice e richiede solo due proprietà: `files` e `tasks`:

```
watch: {
    files: [
        ‘assets/js/*.js’
    ],
    tasks: [‘uglify’]
},
```

Utilizziamo l’asterisco come carattere jolly affinché Grunt individui i file `.js` nella directory *assets/js*. Possiamo collocare quante attività desideriamo nell’array task, e verranno eseguite in ordine.

L’esecuzione di `grunt watch` nel Terminale farà in modo che Grunt continui a venire eseguito in background. Non appena un file viene modificato, entra in azione ed esegue l’attività `uglify`, concatenando e minificando i file JavaScript.

Creare l’attività predefinita

Spesso avrete bisogno di eseguire molteplici attività in una volta sola. Grunt vi consente di farlo registrando l’attività:

```
grunt.registerTask(‘default’, [‘uglify’]);
```

Il primo parametro è il nome dell’attività, mentre il secondo è l’array delle attività che desideriamo eseguire. L’utilizzo della parola chiave `default` comunica a Grunt che

questa è l’attività da eseguire quando non ne specifichiamo una. Per esempio, potremmo eseguire l’attività `default` in uno dei due modi seguenti:

```
grunt default
grunt
```

Utilizzare gulp

Gulp è abbastanza nuovo e prende le mosse da Grunt. A causa della sua vita relativamente breve, non esistono molti plug-in disponibili. Tuttavia c'è `uglify` e il loro numero è in continua crescita. Il vantaggio è che gulp mira a semplificare la configurazione e a eseguire le attività più velocemente di Grunt: spetta a voi decidere quale dei due è più adatto al vostro progetto.

Come Grunt, gulp è costituito da due parti: lo strumento da riga di comando globale e l'installazione locale che includeremo nel progetto.

Installare gulp globalmente

È molto facile installare gulp globalmente mediante un unico comando NPM:

```
npm install -g gulp
```

Potreste doverlo eseguire come radice utilizzando `sudo` o attraverso il prompt dei comandi di Windows come amministratori.

Al termine dell'installazione, il comando `gulp` sarà disponibile dal terminale.

Installare le dipendenze di gulp

Proprio come con Grunt, è necessario creare un file `package.json`, che conterrà tutte le dipendenze del progetto. Iniziamo a installare gulp:

```
{
  "name": "ContactsMgr",
  "version": "1.0.0",
  "description": "A simple contacts manager in AngularJS +
    Bootstrap",
  "dependencies": {
    "gulp": "~3.6.0"
  }
}
```

Possiamo aggiungere manualmente `uglify` al file `package.json`, oppure ricorrere a NPM per questa operazione:

```
npm install --save-dev  gulp-uglify
```

Il flag `save-dev` indica a NPM che vogliamo che lo aggiunga al file `package.json`. In alternativa potremmo utilizzare il flag `--save`, ma siccome gulp viene usato soltanto in fase di sviluppo, non ci serve durante la produzione.

A differenza del plug-in `uglify` di Grunt, non concatenerà i file e dovremo ricorrere a un altro plug-in per questo scopo:

```
npm install --save-dev gulp-concat
```

Impostare gulpfile

A differenza di Grunt, il file gulpfile non deve iniziare con la lettera G maiuscola, anche se ospita la configurazione e si colloca nella radice del progetto. Se il concetto alla base del file è simile, la configurazione è molto diversa.

Ricorderete che Gruntfile richiedeva una funzione `wrapper` per poter accedere a tutti i metodi Grunt. Con gulp è un po' differente e ogni pacchetto che otterremo grazie a NPM potrebbe essere necessario nel file. Includete quanto segue all'inizio del gulpfile.

```
var gulp = require('gulp');
var uglify = require('gulp-uglify');
var concat = require('gulp-concat');
var pkg = require('./package.json');
```

Possiamo anche includere le informazioni del file `package.json` richiedendolo nello stesso modo di un pacchetto NPM. I caratteri `./` all'inizio, indicano a Node di guardare nella stessa directory del `gulpfile` quando eseguiamo gulp.

In questo caso non esiste alcun oggetto di configurazione, e questo semplifica le cose. Tutto avviene all'interno delle attività. Ecco l'attività `uglify` completa:

```
gulp.task('uglify', function(){
    gulp.src(paths.js)
    .pipe(concat('ContactsMgr.min.js'))
    .pipe(uglify())
    .pipe(gulp.dest('assets/js/build'));
});
```

Il metodo `gulp.task` accetta due parametri: il nome dell'attività e una funzione anonima che contiene tutto ciò che farà l'attività.

Abbiamo incluso una variabile in `gulp.src`. Così la potremo utilizzare in seguito e offrirà maggiore flessibilità senza essere costretti a scriverla ogni volta:

```
var paths = {
    js: [
        'assets/js/vendor/jquery.js',
        'assets/js/vendor/bootstrap.js',
        'assets/js/vendor/angular.js',
        'assets/js/vendor/angular-animate.js',
        'assets/js/vendor/angular-resource.js',
        'assets/js/vendor/angular-route.js',
        'assets/js/vendor/angular-sanitize.js',
        'assets/js/vendor/angular-strap.js',
        'assets/js/vendor/angular-strap.tpl.min.js',
        'assets/js/controller.js'
```

```
    ]
};
```

Ecco l'oggetto `paths` al quale facciamo riferimento. Si tratta dello stesso array di file che abbiamo incluso prima in `Gruntfile`.

Gulp utilizza gli operatori pipe per elaborare i dati. Tutti i pacchetti ai quali abbiamo fatto riferimento all'inizio del file `gulpfile` sono funzioni. Il plug-in `concat` accetta il nome del file che intendiamo generare come output. Abbiamo recuperato il nome dal file `package.json` e aggiunto l'estensione `.js`. Il plug-in `uglify` offre un numero di opzioni che possiamo passare come un hash JS per facilitare il debugging, e il metodo `gulp.dest` che utilizziamo ci consente di immettere il nome della directory dell'output.

Dopo aver impostato l'attività, potremmo eseguire `gulp uglify`, ma prima è necessario impostare anche `watch`. A differenza di Grunt, non è necessario includere un ulteriore plug-in per questo scopo, poiché si presenta come un altro metodo di gulp:

```
gulp.task('watch', function(){
    gulp.watch(paths.js, ['uglify']);
});
```

L'impostazione è molto semplice. Creiamo una nuova attività e utilizziamo il metodo `gulp.watch`, attraversando l'array di file che intendiamo osservare e poi l'array di attività che vogliamo eseguire quando questi file cambiano.

Impostiamo rapidamente un'attività predefinita per completare il file `gulpfile`:

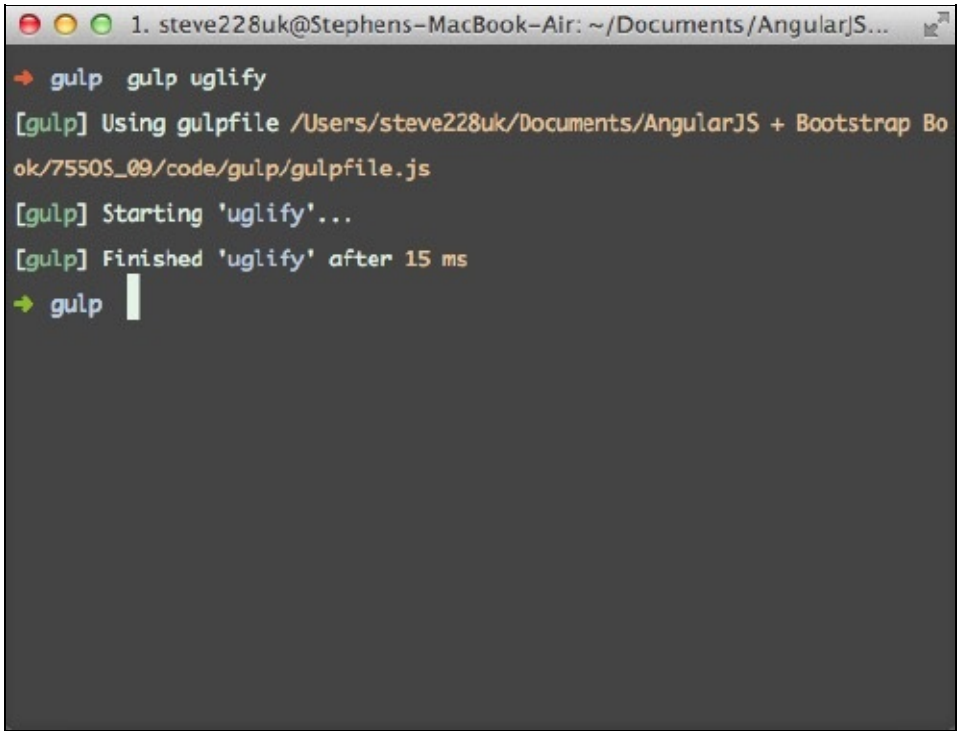
```
gulp.task('default', ['uglify']);
```

Ecco il file `gulpfile.js` completo che configura efficacemente le attività:

```
var gulp = require('gulp');
var uglify = require('gulp-uglify');
var concat = require('gulp-concat');
var pkg = require('./package.json');

var paths = {
    js: [
        'assets/js/vendor/jquery.js',
        'assets/js/vendor/bootstrap.js',
        'assets/js/vendor/angular.js',
        'assets/js/vendor/angular-animate.js',
        'assets/js/vendor/angular-resource.js',
        'assets/js/vendor/angular-route.js',
        'assets/js/vendor/angular-sanitize.js',
        'assets/js/vendor/angular-strap.js',
        'assets/js/vendor/angular-strap.tpl.min.js',
        'assets/js/controller.js'
    ]
};

gulp.task('uglify', function(){
    gulp.src(paths.js)
    .pipe(concat(pkg.name+'.js'))
```

```
    .pipe(uglify())
    .pipe(gulp.dest('assets/js/build'));
});

gulp.task('watch', function(){
    gulp.watch(paths.js, ['uglify']);
});

gulp.task('default', ['uglify']);
```

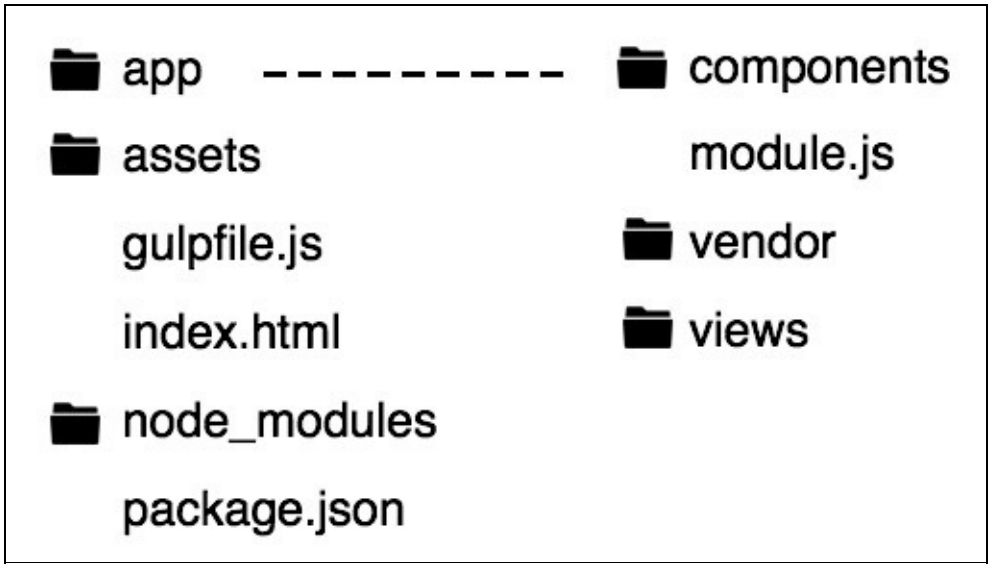
Ora possiamo eseguire `gulp uglify` o `gulp` nel terminale per concatenare e minificare i file JavaScript. A parte un singolo utilizzo, possiamo anche eseguire `gulp watch` per verificare l'esistenza di eventuali modifiche ed eseguire automaticamente l'attività `uglify`.



Riorganizzare il progetto

Dopo aver impostato il task runner, è il momento di riorganizzare il progetto per ottenere una base di codice più gestibile. Separereremo i controller, le direttive, i filtri e i servizi in file distinti per tenere tutto in ordine.

Iniziamo a elaborare una nuova struttura di directory come illustra la seguente figura.



Abbiamo spostato tutto fuori della directory *assets/js* nella directory *app* della radice. Quando passeremo alla produzione, non intendiamo distribuire i file sorgente, quindi è una buona idea toglierli dalla directory *assets* dove rimarranno i file compressi.

La nuova directory *app* è stata strutturata in modo leggermente diverso. Ci sono tre cartelle *components*, *vendor* e *views* e un file `module.js`. I componenti sono item condivisi, quindi nel caso dell'app, collochiamo qui le direttive e i servizi. La cartella *vendor* contiene tutti i file JS di terze parti mentre la cartella *views* tutti i principali controller per le viste.

Dopo aver creato le nuove directory, possiamo iniziare a separare i controller in diversi file per includerli nella directory *views*. Devono ancora essere associati al modulo e a questo scopo possiamo utilizzare la dichiarazione `angular.module('contactsMgr')` all'inizio del file.

Ecco, per esempio, il controller `contactCtl`; nominatelo `contact.js` e inseritelo nella cartella *views*:

```
angular.module('contactsMgr').controller('contactCtl', ['$scope',
  '$routeParams', 'contacts', '$timeout', function($scope,
    $routeParams, contacts, $timeout){
    $scope.contact = contacts.find($routeParams.id);
```

```
    $scope.$on('saved', function(){
        $timeout(function(){
            $scope.contact.$update();
        }, 0);
    });
}]);
```

Ora che tutti i controller sono stati separati, passiamo alle direttive. Le sposteremo nella cartella *components* con la nuova directory *app* della radice. Così come con i controller, è necessario verificare che siano ancora associate al modulo.

Copiate ogni direttiva in file separati e assegnate a esse l'estensione `.directive.js`. Per esempio, il gravatar si troverà nella cartella *components* e sarà `gravatar.directive.js`, mentre la direttiva modificabile sarà `editable.directive.js`.

Siccome i filtri sono anch'essi componenti condivisi, possiamo collocarli insieme con le direttive e nominarli allo stesso modo. Posizionate i due filtri in file separati: `truncate.filter.js` e `newLine.filter.js`. L'ultimo componente è il servizio `contacts` che utilizziamo per connetterci al server. Create un nuovo file `contacts.service.js` e copiatelo.

Dopo aver sistemato viste e componenti, dobbiamo verificare che il nuovo file `module.js` contenga ciò che dovrebbe, ossia il modulo. Copiate il contenuto del file `contactsMgr.js` dalla directory *assets/js* nel file `module.js`. Dopo aver spostato tutti i file, cancellate il contenuto della directory *js*. Qui aggiungeremo in seguito un file interamente minificato.

Dobbiamo configurare Grunt/gulp affinché osservi le nuove directory e generi l'ouput nella directory *js* all'interno della cartella *assets*. I percorsi aggiornati nel Gruntfile o i file gulpfile dovrebbero essere i seguenti:

```
'app/vendor/jquery.js',
'app/vendor/bootstrap.js',
'app/vendor/angular.js',
'app/vendor/angular-animate.js',
'app/vendor/angular-resource.js',
'app/vendor/angular-route.js',
'app/vendor/angular-sanitize.js',
'app/vendor/angular-strap.js',
'app/vendor/angular-strap.tpl.js',
'app/module.js',
'app/components/**/*.js',
'app/views/**/*.js'
```

Angular si occupa della gestione delle dipendenze, ma ciò che conta è l'ordine. Carichiamo `jquery.js` prima di `angular.js` affinché Angular sappia che vogliamo utilizzare questo e non il `jqLite` incluso.

Tutti i moduli `vendor` necessitano di Angular che deve essere incluso prima di essi. Inoltre i componenti e le viste richiedono il caricamento del modulo, altrimenti non

saprà a che cosa associarsi.

Prima di eseguire il task runner desiderato, modificate la destinazione da *assets/js/build* ad *assets/js*.

Eseguite il task runner per compilare l’applicazione che avete riorganizzato. Infine modificate il file al quale fate riferimento in `index.html` ora che la destinazione è cambiata:

```
<script type=”text/javascript”
  src=”assets/js/ContactsMgr.js”></script>
```

Aprite il browser per controllare che tutto appaia come previsto. Se tutto si comporta secondo i piani, il gestore dei contatti dovrebbe funzionare alla perfezione.

NOTA

Se non funziona come dovrebbe, controllate la console. Probabilmente avete tralasciato qualcosa o l’avete incluso nell’ordine sbagliato.

Quiz

1. Su quale ambiente si basano sia Grunt sia gulp?
2. Perché serve un file `package.json`?
3. Quale plug-in viene utilizzato per minificare i file?
4. Che cosa è necessario fare ai file Angular prima di minificarli?

Riepilogo

In questo capitolo abbiamo esaminato due strumenti molto efficaci e simili. Ci hanno consentito non solo di ridurre notevolmente il numero delle richieste HTTP, ma anche di riorganizzare completamente l’app.

Sia Grunt sia gulp ottengono lo stesso risultato, ed è una questione strettamente personale scegliere il task runner da utilizzare. Riteniamo gulp più rapido e facile da configurare, ma indubbiamente Grunt vanta un numero maggiore di plug-in ed è uno strumento disponibile da più tempo e più collaudato.

Nel capitolo seguente vedremo come utilizzare questi due task runner per prendere i file Less che utilizza Bootstrap e compilarli in una versione personalizzata.

Personalizzare Bootstrap

Finora l’applicazione è piuttosto convenzionale. Siamo riusciti a sfruttare appieno Bootstrap, ma l’aspetto predefinito non ha nulla di nuovo. Bootstrap è progettato per essere personalizzato e utilizza Less o il preprocessore CSS SASS per velocizzare e semplificare questa operazione.

Nel corso di questo capitolo studieremo come è possibile compilare il sorgente Less di Bootstrap prima di personalizzare l’aspetto affinché l’applicazione sia veramente una nostra creazione. Affronteremo i seguenti argomenti.

- I fondamenti di Less.
- Personalizzare Bootstrap.
- Compilare Less con Grunt o gulp.
- Impostare LiveReload.
- I temi di Bootstrap.

Compilare Less con Grunt o gulp

Prima di iniziare a personalizzare l'app, è una buona idea scoprire come trasformare numerosi file Less in un unico foglio di stile. Abbiamo già visto come impostare Grunt e gulp per concatenare e minificare i file JavaScript; ora utilizzeremo questi stessi task runner per compilare Less.

Scaricare il sorgente

Innanzitutto procuriamoci l'ultima versione di Bootstrap e inseriamo i file Less nel progetto. Visitiamo `http://getbootstrap.com/` e facciamo clic su *Download Bootstrap*. Compariranno tre opzioni: *Bootstrap*, *Source code* e *Sass*. Scegliamo l'opzione *Source code* perché include i file Less che è possibile personalizzare e compilare.

Come potete notare, il sorgente di Bootstrap è dieci volte più grande della versione minificata. Per tenere in ordine il tutto, copiate la directory *less* dal materiale che avete scaricato nella cartella *assets* all'interno del progetto. La directory contiene i 40 file `less` che costituiscono gli stili di Bootstrap.

Compilare con Grunt

Come abbiamo visto, Grunt è un efficiente task runner. Possiamo avvalerci della minificazione dei JavaScript per automatizzare la compilazione di Less con CSS. A questo scopo Grunt ricorre a un plug-in che è possibile recuperare da NPM. Includiamolo nel `package.json` del progetto ed eseguiamo `npm install` dal terminale. Tuttavia è molto più semplice eseguire un unico comando e lasciare che NPM aggiunga la dipendenza nel file `package.json` del progetto, nel modo seguente:

```
npm install grunt-contrib-less --save-dev
```

Dopo aver installato il plug-in, possiamo configurarlo nel file Grunt. Tutto avviene nuovamente nell'oggetto `config`. Imposteremo due target: uno per lo sviluppo e un altro per la produzione. Così potremo definire le opzioni per ogni scenario. Per esempio, per la produzione potremmo voler minificare il CSS, ma questa operazione non è sempre auspicabile in fase di sviluppo.

Segue la configurazione completa per l'attività `less`:

```
less: {
    dev: {
        files: {
            'assets/css/bootstrap.css':
  'assets/less/bootstrap.less'
```

```
        }
    },
    production: {
        options: {
            cleancss: true
        },
        files: {
            'assets/css/bootstrap.css':
  'assets/less/bootstrap.less'
        }
    }
}
```

Nel target `dev` non abbiamo impostato opzioni, ma `production` presenta il flag `cleancss` impostato su `true` per ridurre le dimensioni dei file minificando l'output. L'oggetto `files` usa la sintassi abbreviata per le proprietà `src` e `dest` che abbiamo esaminato configurando Grunt e utilizzando `uglify` per i file JavaScript.

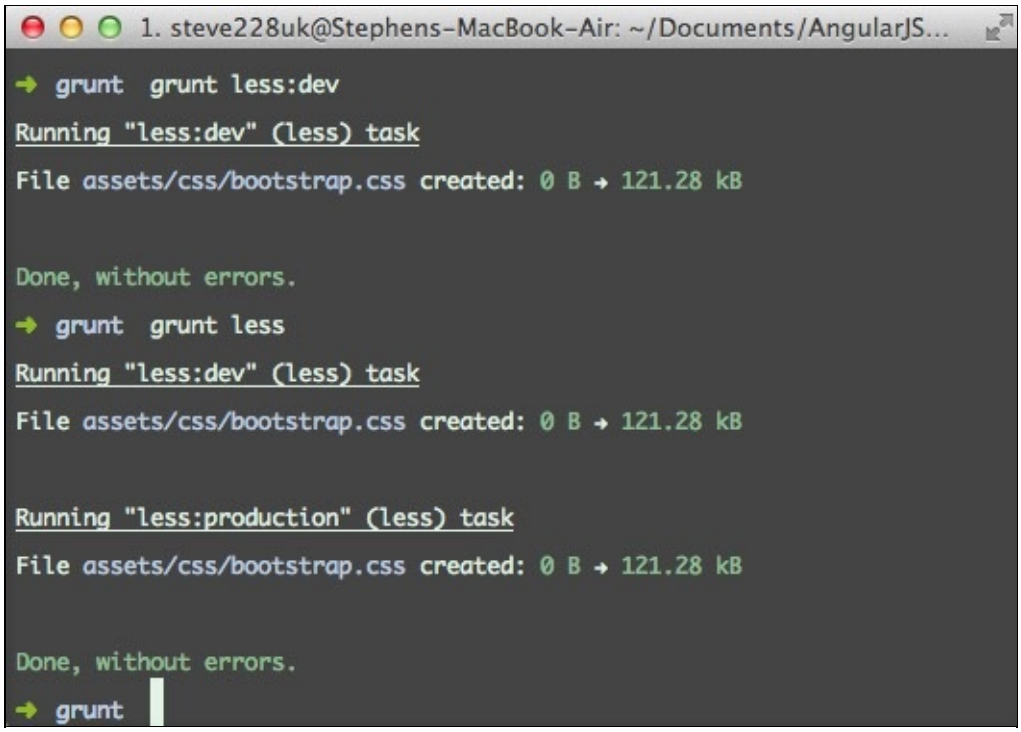
Non dimenticate di caricare anche questo modulo da NPM, altrimenti Grunt non riuscirà ad accedere all'attività `less`. Inseritelo nel wrapper di Grunt:

```
grunt.loadNpmTasks('grunt-contrib-less');
```

Ora sarà possibile compilare gli stili. Possiamo eseguire quanto segue nel terminale per svolgere l'attività:

```
grunt less
```

Questo però produce un effetto non voluto, poiché l'attività viene eseguita su entrambi i target uno dopo l'altro.



Possiamo limitarla a un solo target specificando il nome del target dopo i due punti:

```
grunt less:dev
```

Impostare Watch e LiveReload

Ovviamente il principio alla base dell'utilizzo di un task runner come Grunt o gulp è automatizzare tutti questi processi. È possibile utilizzare il plug-in `watch` come abbiamo fatto nel Capitolo 9 per eseguire un'attività quando un file cambia. Questo ci permette anche di utilizzare `live reload` sulla pagina con l'ausilio di un plug-in del browser.

La configurazione è facile poiché abbiamo già installato il plug-in. Ecco l'impostazione corrente per l'attività `watch`:

```
watch: {
    files: [
        'assets/js/*.js'
    ],
    tasks: ['uglify']
}
```

Potremmo aggiungere la directory *less* all'array `files` e l'attività `less` all'array `tasks`. In questo modo entrambe le attività verranno eseguite quando cambierà un file `.js` o `.less`; tuttavia non è questo il nostro obiettivo. Separandole in due target, avremo maggiore controllo sulle attività che saranno eseguite:

```
watch: {
    js: {
        files: [
            'assets/js/*.js'
        ],
        tasks: ['uglify']
    },
    less: {
        files: [
            'assets/less/*.less'
        ],
        tasks: ['less:dev']
    }
}
```

Per quanto riguarda la configurazione, non cambia nulla. Abbiamo solo separato i due tipi di file per eseguire rispettivamente `uglify` e le attività `less:dev`.

Il plug-in ha anche un altro asso nella manica. Funge da server per il plug-in `LiveReload` per molteplici browser. Per utilizzarlo con l'app, è necessario includere un ulteriore tag `script` nella pagina:

```
<script src="http://localhost:35729/livereload.js"></script>
```

In alternativa esiste un'estensione per Chrome, Firefox e Safari, che può essere scaricata da http://livereload.com/. Dopo averla installata, nel browser sarà possibile effettuare il ping del server LiveReload ogni volta che si verificano delle modifiche.

Impostare Grunt per eseguire il ping del nuovo plug-in del browser è molto facile; è sufficiente impostare la proprietà `livereload` su `true` in un oggetto `options`. Aggiungiamola rapidamente al target `less`:

```
less: {
    files: [
        ‘assets/less/*.less’
    ],
    tasks: [‘less:dev’],
    options: {
        livereload: true
    }
}
```

Se aprite il browser e attivate LiveReload, vedrete che la pagina si ricarica ogni qualvolta si effettuano delle modifiche ai file `less`. Ottimo, ma non sarebbe preferibile se si aggiornasse solo il CSS e non l’intera pagina? Se cambia un file, Grunt ricaricherà l’intera pagina. Per ottenere questo risultato, è possibile aggiungere un secondo target alla configurazione che verifica la presenza di modifiche nel file `bootstrap.css`:

```
css: {
    files: [
        ‘assets/css/bootstrap.css’
    ],
    options: {
        livereload: true
    }
}
```

Se disattiviamo LiveReload nel target `less`, il browser riceverà un nuovo file CSS e non ricaricherà più la pagina.

Compilare con gulp

Ora esaminiamo l’altro task runner: gulp. Proprio come Grunt, gulp utilizza un altro plug-in per consentire la compilazione di Less. Sarà necessario includere un secondo plug-in per LiveReload, poiché non è già incluso in Grunt.

Installiamo e configuriamo il plug-in `less`. Possiamo farlo da riga di comando eseguendo:

```
npm install gulp-less --save-dev
```

Così lo installeremo nel progetto e lo includeremo anche nel file `package.json` per un uso futuro. Per utilizzarlo nel `gulpfile`, ricorreremo al metodo `require` di Node per includere il pacchetto. Inseritelo all’inizio del `gulpfile`:

```
var less = require(‘gulp-less’);
```

Creiamo una nuova attività chiamata `less` per gestire tutta la compilazione, servendoci del plug-in che abbiamo appena incluso:

```
gulp.task('less', function(){

});
```

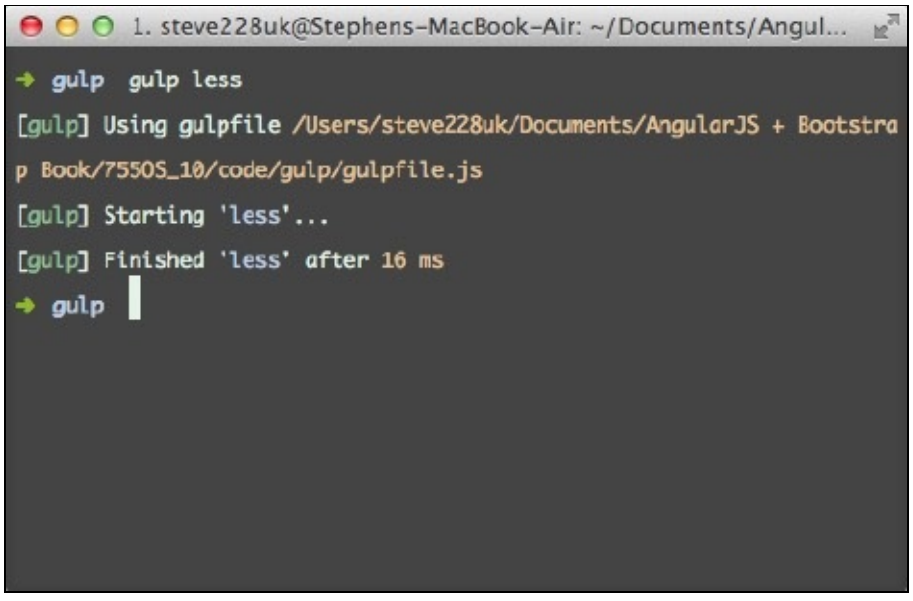
Come abbiamo fatto con JavaScript, utilizzeremo il metodo `src` di gulp con due operatori pipe per ottenere il risultato desiderato. Consideriamo l'attività completa ed esaminiamola nel dettaglio:

```
gulp.task('less', function(){
    gulp.src('assets/less/bootstrap.less')
    .pipe(less({
        filename: 'bootstrap.css'
    }))
    .pipe(gulp.dest('assets/css'));
});
```

In questo caso abbiamo incluso un solo file; gli altri sono inclusi tramite `@import`. Il plug-in Less accetta qualsiasi parametro accettato dal compilatore Less ufficiale. In questa configurazione impostiamo un nome file, ma per impostazione predefinita utilizzerà il nome file del sorgente.

Infine ci serviremo del metodo `gulp.dest` per esportare il file compilato nella directory *CSS*. Questi pipe rappresentano i passi che l'attività esegue in ordine ed è possibile aggiungerne facilmente altri o riordinarli in futuro se fosse necessario.

Ora Gulp è configurato per utilizzare il plug-in Less ed è pronto per compilare gli stili mediante il comando `less` di gulp.



Impostare Watch e LiveReload

Ovviamente mancheremmo l'obiettivo dell'automazione se dovessimo eseguire questo comando manualmente. Abbiamo già scoperto che `watch` è integrato in gulp; includere `less` nell'attuale configurazione consiste nell'aggiungere solo un'altra riga di codice:

```
gulp.task('watch', function(){
    gulp.watch(paths.js, ['uglify']);
    gulp.watch(paths.less, ['less'])
});
```

Facciamo riferimento a una nuova proprietà all'interno dell'oggetto `paths`. Aggiungiamola affinché gulp sappia dove si trovano i file che stiamo considerando:

```
less: 'assets/less/*.less'
```

Utilizziamo il carattere asterisco per fare riferimento a ogni singolo file Less; così gulp può vedere esattamente quando qualcosa è cambiato.

Impostare LiveReload richiede più lavoro perché comporta l'installazione di un altro plug-in. Recuperiamolo da NPM con il seguente comando:

```
npm install --save-dev gulp-livereload
```

Dopo averlo installato, fate riferimento al plug-in all'inizio del `gulpfile`:

```
var livereload = require('gulp-livereload');
```

La funzione restituita è il servizio LiveReload e lo possiamo utilizzare per indicare all'estensione del browser quali file sono cambiati. Per configurarlo correttamente è necessario svolgere due operazioni nell'attività `watch`. Innanzitutto facciamo riferimento al server LiveReload a cui possiamo passare i file `change` da un evento attivato dal metodo `gulp.watch`:

```
gulp.task('watch', function(){

    var server = livereload();

    gulp.watch(paths.js, ['uglify']);
    gulp.watch(paths.less, ['less']).on('change', function(file){
        server.changed(file.path);
    });

});
```

Abbiamo assegnato la funzione `livereload()` alla variabile server e aggiunto un listener per l'evento `change` al watcher. Dopo che l'evento verifica un oggetto file, è possibile passare il percorso del file al server.

Come con Grunt, è necessario affrontare la questione del ricaricamento del browser quando cambia un file non CSS. Possiamo risolverla aggiungendo un terzo watcher. Di seguito l'attività completata con questa integrazione:

```
gulp.task(‘watch’, function(){

    var server = livereload();

    gulp.watch(paths.js, [‘uglify’]);
    gulp.watch(paths.less, [‘less’]);
    gulp.watch(‘assets/css/bootstrap.css’).on(‘change’,
  function(file){
        server.changed(file.path);
    });

});
```

NOTA

Non dimenticate di sostituire nel file `index.html`, il file CSS minificato con quello appena compilato.

Less

Per comprendere meglio ciò che offre Less, introduciamone i fondamenti per capirne la natura e la sintassi alla base del preprocessore.

Osserveremo quattro tra le sue principali caratteristiche: importazione, mixin, regole annidate e variabili. Una lista completa è disponibile sul sito `http://lesscss.org/features/`.

Il vantaggio di Less è che se non volete utilizzare nessuna sintassi o funzione nuova, non sarete costretti a farlo. Qualsiasi CSS valido è anche Less valido.

Importazione di file

Come per CSS, in Less possiamo includere un file all'interno di un altro. Inoltre segue la stessa sintassi di CSS:

```
@import "file.less"
```

Tuttavia, a differenza di CSS, che effettua un'ulteriore richiesta HTTP per il file al quale si fa riferimento, Less unirà il file quando è compilato. Se aprite `bootstrap.less`, vedrete tutti i file Less necessari ai quali viene fatto riferimento.

NOTA

Con le versioni più recenti di Less potete tralasciare `.less` quando includete i file e compilate.

Variabili

Bootstrap utilizza le variabili Less per consentirci di modificare i colori e i font degli elementi `variable`. Possiamo cambiarli rapidamente aprendo il file `variables.less`. Una variabile viene definita dal simbolo `@` seguito dal suo nome, per esempio:

```
@brand-primary:         #428bca;
```

Si tratta di semplici riferimenti da utilizzare negli stili. Possiamo chiamare la variabile facendovi riferimento all'interno delle proprietà:

```
color: @brand-primary;
```

Durante la compilazione, Less sostituirà a questi riferimenti il colore definito in precedenza. Bootstrap utilizza queste variabili in tutti i suoi elementi; pertanto è possibile modificare rapidamente colori e font in questo file.

Regole annidate

Forse uno dei pattern più ostici di CSS è l’assegnazione degli stili agli elementi figlio. Invece di annidare l’elemento figlio all’interno dell’elemento genitore, è necessario scrivere una regola separata. Ecco un esempio:

```
div {
  background: #ccc;
}

div a {
  color: #000;
}

div a:hover {
  color: #fff;
}
```

Con Less, possiamo annidare queste regole per tenere tutto più in ordine:

```
div {
  background: #ccc;
  a {
    color: #000;
    &:hover {
      color: #fff;
    }
  }
}
```

Come potete notare nell’esempio precedente, è possibile annidare pseudoclassi utilizzando la sintassi `&`. Si tratta di un riferimento alla regola riguardante il genitore. Nello stesso modo è possibile anche definire una classe secondaria. Per esempio, ecco un pulsante con due stili per i colori arancione e blu:

```
button {
  color: #fff;
  &.orange {
    background: orange;
  }
  &.blue {
    background: blue;
  }
}
```

Mixin

Un mixin consente di includere gli stili da un’altra regola. Accetta anche argomenti che è possibile passare per ottenere maggiore flessibilità. Vediamo un esempio:

```
.border-radius(@radius: 5px) {
  border-radius: @radius;
}
```

Possiamo utilizzare questo mixin negli stili:

```
button {
  .border-radius;
```

```
}
```

Il valore predefinito che abbiamo impostato verrà utilizzato automaticamente. Tuttavia possiamo scavalcarlo facilmente racchiudendolo tra parentesi:

```
button {
  .border-radius(15px);
}
```

Personalizzare gli stili di Bootstrap

Poiché utilizziamo il sorgente di Bootstrap, possiamo analizzare e personalizzare qualsiasi file. Less estende CSS e ci offre nuove funzioni sfruttate da Bootstrap.

Caratteri tipografici

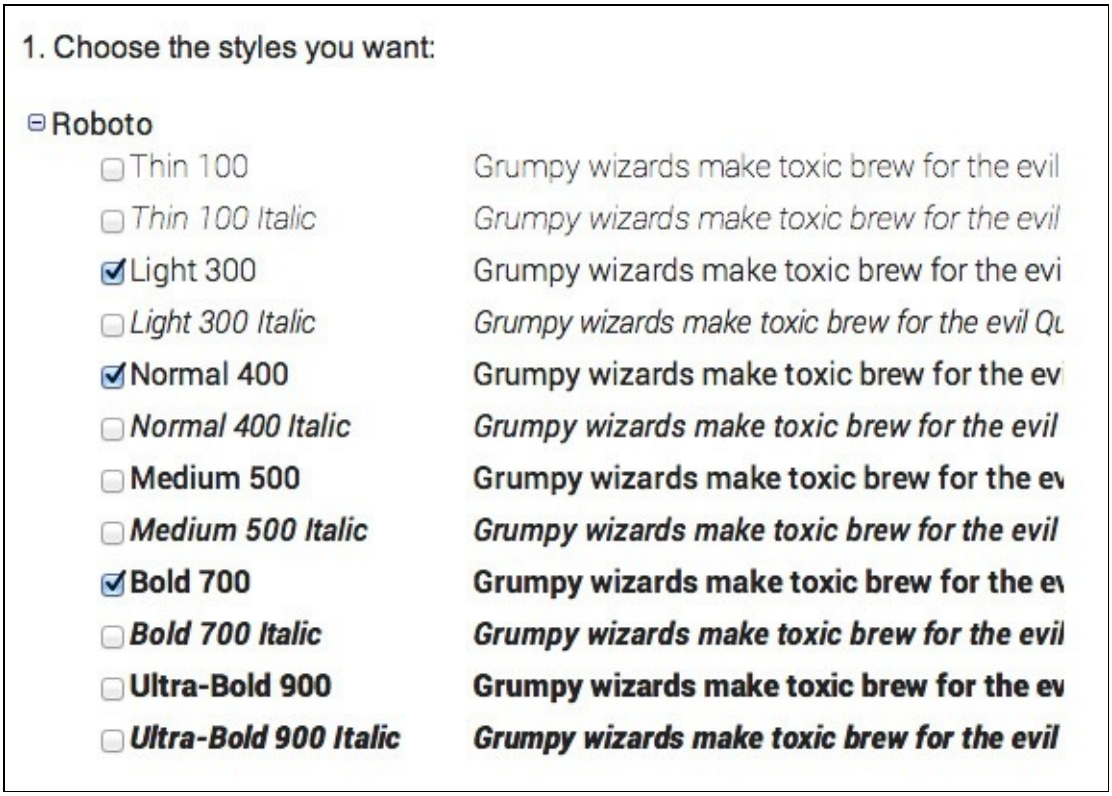
Bootstrap utilizza Helvetica, forse il font più popolare al mondo. Per dare all'applicazione un po' di carattere, vediamo come è possibile sostituire a questo un altro font proveniente dalla libreria Google Fonts, visitabile all'indirizzo `https://www.google.com/fonts`. Date un'occhiata e trovate quello che più vi piace. Per ora utilizzeremo Roboto, un carattere bastone.

Aggiungete il font alla raccolta e selezionate gli stili *Light*, *Normal* e *Bold*, come nella figura riportata alla pagina successiva.

Copiate la riga `@import` e includetela all'inizio del file `bootstrap.less`, così avrete accesso alla famiglia di font Roboto nei vostri stili. Cercate `Typography` nel file `variables.less`. La sezione inizia all'incirca a riga 38. Modificheremo la variabile `@font-family-sans-serif` che Bootstrap utilizza per impostazione predefinita come base:

```
@font-family-sans-serif:  Roboto, “Helvetica Neue”, Helvetica,
  Arial, sans-serif;
```

Potremmo anche cambiare la dimensione del carattere, ma per ora non effettuiamo alcuna modifica; ci sembra tutto grazioso ed equilibrato.



navbar

Cercate `navbar` in `variables.less`; la sezione dovrebbe iniziare all'incirca a riga 324. Effettueremo alcune modifiche per renderlo meno anonimo. Iniziamo a cambiare il grigio scialbo in un colore più stimolante, come un bel blu metallizzato:

```
@navbar-default-bg:                #667591;
```

Cambieremo anche il colore del testo e dei link affinché si intonino con il colore più scuro dello sfondo:

```
@navbar-default-color:                  #fff;
@navbar-default-link-color:             #fff;
@navbar-default-link-hover-color:       #ccc;
@navbar-default-link-active-color:      #fff;
```

Per quanto riguarda la barra di navigazione mobile, è necessario modificare anche il colore del pulsante interruttore:

```
@navbar-default-toggle-hover-bg:           darken(@navbar-default-
  bg, 15%);
@navbar-default-toggle-icon-bar-bg:        #fff;
@navbar-default-toggle-border-color:       #fff;
```

Less include diverse funzioni helper che è possibile utilizzare per modificare i colori, tra cui `saturate`, `desaturate` e `fade`. Le due funzioni di Bootstrap più utilizzate sono `darken` e `lighten`. Schiariscono o scuriscono percentualmente un colore; sono

ideali per gli stati hover. In questo caso abbiamo optato per la variabile `darken` e l'abbiamo passata per lo sfondo della barra di navigazione.

Infine modifichiamo l'altezza ed eliminiamo l'arrotondamento degli angoli per ottenere un aspetto più ordinato:

```
@navbar-height:                   60px;
@navbar-border-radius:            0;
```

Form

I form di Bootstrap sono forse gli elementi più riconoscibili. Effettuiamo alcune modifiche per renderli leggermente diversi. Modificheremo `variables.less` e un paio di altri file. Innanzitutto in `variables.less` eliminiamo l'arrotondamento degli angoli dagli input:

```
@input-border-radius:             0;
```

Possiamo anche modificare il colore del bordo e l'ombra del focus:

```
@input-border-focus:              #667591;
```

Forse l'ombra è un po' eccessiva, ma possiamo eliminarla facilmente modificando un mixin di Bootstrap. In passato tutti i mixin del framework erano contenuti nell'unico file: `mixins.less`, ma questa impostazione è recentemente cambiata e i mixin sono stati suddivisi in file separati.

Nella directory *mixins*, aprite `forms.less` e cercate `.form-control-focus`. È il mixin che assegna gli stili al focus per gli elementi del form; potete osservarlo nelle righe di codice seguenti:

```
.form-control-focus(@color: @input-border-focus) {
  @color-rgba: rgba(red(@color), green(@color), blue(@color), .6);
  &:focus {
    border-color: @color;
    outline: 0;
    .box-shadow(~"inset 0 1px 1px rgba(0,0,0,.075), 0 0 8px
      @{color-rgba}");
  }
}
```

Il mixin modifica il colore del bordo, elimina il contorno predefinito del browser e aggiunge il box con ombra. Per il momento trasformiamo quest'ultimo in un commento:

```
//.box-shadow(~"inset 0 1px 1px rgba(0,0,0,.075), 0 0 8px @{color-
  rgba}");
```

Less ci consente di utilizzare i commenti `//` nei fogli di stile, che non saranno riprodotti nell'output. Se invece ricorriamo ai commenti (`/* */`) CSS standard, si rifletteranno nell'output.

Nella vista *Add Contact* utilizzeremo anche un `well`, che avrà un aspetto strano visto che i campi di input dell'utente ora sono privi di angoli arrotondati. Potremmo rimuovere l'arrotondamento da tutti gli elementi impostando su 0 la variabile `@border-radius-base`, ma per ora apriamo il file `wells.less`. Nella classe `well` base, impostiamo su 0 `border-radius` per eliminare gli angoli arrotondati.

Pulsanti

Bootstrap include molti colori e dimensioni per i pulsanti. Possiamo effettuare le modifiche nel file `variables`. Troverete la sezione cercando `Buttons`; dovrebbe essere all'incirca alla riga 140. Bootstrap utilizza gli stessi colori per molti suoi componenti; la sezione riguardante i colori inizia intorno alla riga 6.

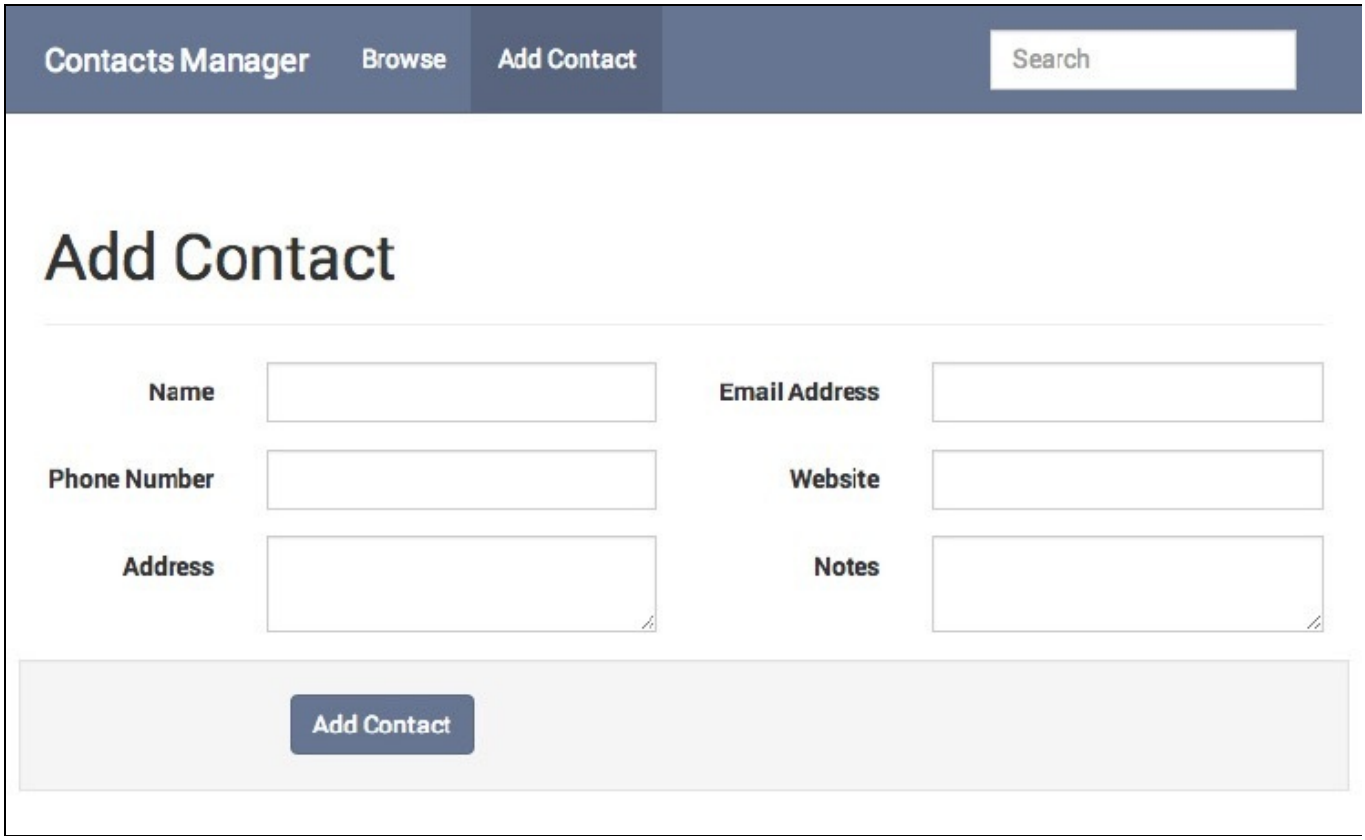
```
@brand-primary:         #428bca;
@brand-success:         #5cb85c;
@brand-info:            #5bc0de;
@brand-warning:         #f0ad4e;
@brand-danger:          #d9534f;
```

Modifichiamo `brand-primary` affinché si intoni allo sfondo della barra di navigazione:

```
@brand-primary:         #667591;
```

Bootstrap utilizza la funzione `darken` per applicare il colore del bordo del pulsante e i colori dello stato hover.

È possibile personalizzare ogni aspetto degli stili, ma per il momento non cambieremo nulla. Il *Contacts Manager* appare meno anonimo e sciatto grazie ai nuovi colori e caratteri più vivaci. Al termine della personalizzazione, l'applicazione dovrebbe avere il seguente aspetto.


Contacts Manager
Browse
Add Contact
Search
Add Contact
Name
Email Address
Phone Number
Website
Address
Notes
Add Contact

I temi di Bootstrap

Un cambiamento rilevante di Bootstrap 3 è stata l’eliminazione di tutti gli stili visivi. Infatti Bootstrap intende essere un framework su cui sviluppare e non semplicemente da utilizzare con l’aspetto predefinito.

Tuttavia gli stili esistono ancora: sono stati spostati in un file separato, incluso nel sorgente. Il file `theme.less` reintroduce le sfumature presenti in Bootstrap 2; è sufficiente importarlo nel file `Less` principale.

Aprite `bootstrap.less` e aggiungete quanto segue alla fine del file per includere il tema:

```
@import “theme.less”;
```

Per vedere un esempio relativo all’utilizzo del tema di Bootstrap, date un’occhiata a `getbootstrap.com/examples/theme`.

Dove trovare altri temi di Bootstrap

Esistono alcuni siti web che propongono temi di Bootstrap. Offrono un sistema rapido e veloce per aggiungere un po’ di carattere all’aspetto standard del framework. Consultate i siti seguenti se desiderate sperimentare uno di questi temi:

- http://www.blacktie.co/
- https://wrapbootstrap.com/
- http://startbootstrap.com/
- http://bootswatch.com/

Quiz

1. Quali sono alcune delle caratteristiche principali che aggiunge Less?
2. In che modo è possibile fare riferimento a una pseudoclasse in una regola annidata?
3. Come è possibile modificare il font?
4. A cosa serve il file `theme.less`?

Riepilogo

Nel Capitolo 9 abbiamo esaminato i task runner per concatenare e minificare i file JavaScript in un unico file. In questo capitolo abbiamo osservato come è possibile estenderli al sorgente Less di Bootstrap e compilarlo nel CSS.

Ciò consente di personalizzare l'aspetto di Bootstrap utilizzando l'estensione di Less nel CSS: regole annidate, variabili e mixin. L'applicazione ha acquisito un certo carattere e abbiamo anche visto come è possibile reintrodurre un po' dello stile visivo proveniente da Bootstrap 2.

Nel prossimo capitolo descriveremo la validazione in AngularJS e la sua integrazione nell'app.

Validazione

Tutto funziona bene e ha un bell’aspetto, ma al momento non esiste alcun tipo di validazione per i form o gli errori che potrebbero essere rinviati dal server. In questo capitolo esamineremo come opera la validazione in AngularJS e come è possibile combinarla con gli stili di Bootstrap per fornire un feedback all’utente.

Osserveremo inoltre come è possibile espandere le regole integrate creando un validator personalizzato sulla base dei concetti che abbiamo appreso nei capitoli precedenti.

Validazione dei form

Una delle caratteristiche nascoste di AngularJS è la validazione nativa. Sono disponibili una validazione di base dei tipi di input HTML5 più comuni, oltre a direttive personalizzate come `required`, `pattern` e `minlength`, solo per citarne alcune. Vedremo come aggiungerle all'applicazione ed estendere la validazione integrata con un validator personalizzato.

Per utilizzare la validazione di AngularJS, è necessario aggiungere qualcosa al tag di apertura `form`:

```
<form name=”addForm” novalidate class=”form-horizontal” ng-
  submit=”submit()”>
```

La riga di codice precedente contiene il tag `form` presente nel partial `add.html`. Abbiamo aggiunto `name` oltre a un attributo `novalidate`. L'attributo `name` assegna un oggetto allo scope corrente; pertanto possiamo accedervi dalla vista e dal controller. L'attributo `novalidate` disattiva la validazione nativa del browser. L'abbiamo fatto perché la gestiremo noi e intendiamo evitare che l'impostazione predefinita rappresenti un ostacolo o generi risultati non voluti.

Angular validerà automaticamente l'input dell'indirizzo e-mail, e farà lo stesso se cambiamo il campo del sito web da testo a URL. Possiamo aggiungere rapidamente l'attributo necessario a qualsiasi input che consideriamo obbligatorio:

```
<input type=”text” id=”name” class=”form-control” ng-
  model=”contact.name” required>
```

In alternativa possiamo utilizzare `ng-required`. Imposterà su `true` l'attributo richiesto del browser se anche l'espressione di AngularJS è `true`. Per esempio, al momento del pagamento, potreste voler aggiungere una casella di controllo per permettere all'utente di digitare un indirizzo diverso per la spedizione e la fatturazione. Nei casi in cui la casella è selezionata, è possibile impostare i campi su `required` nel modo seguente:

```
<label><input type=”checkbox” ng-model=”shippingAddress”> Send this to
  another address</label>
<div ng-show=”shippingAddress”>
    <input type=”text” ng-required=”shippingAddress”>
</div>
```

Siccome non abbiamo bisogno di nessuna condizione nell'applicazione, manteniamo l'attributo `required` predefinito. Aggiungiamolo rapidamente al nome, al numero di telefono e all'indirizzo e-mail poiché sono questi i campi del contatto richiesti più frequentemente.

Ora è necessario impedire al form di inviare i dati quando non è completamente validato. Possiamo farlo all'interno del controller oppure disattivando il pulsante *submit*.

Aggiungendo un nome al form, AngularJS ha creato un nuovo modello, che ci offre l'accesso diretto a esso all'interno della vista. È leggermente diverso da quello che creeremmo di solito perché racchiude molte proprietà per verificare la validità non solo del form, ma anche di eventuali elementi specifici da noi indicati. Vediamo rapidamente come disattivare il pulsante *submit*:

```
<input type="submit" class="btn btn-primary" value="Add Contact"
  ng-disabled="addForm.$invalid">
```

Possiamo utilizzare il nuovo modello in una direttiva. In questo caso si tratta di `ng-disabled` e comunichiamo ad AngularJS di disattivare il pulsante se il form non è valido. In alternativa avremmo potuto verificare la presenza di errori:

```
<input type="submit" class="btn btn-primary" value="Add Contact"
  ng-disabled="addForm.$error.required || addForm.$error.email">
```

La proprietà `$error` del modello è un hash con i diversi tipi di errori generati dal form. Possono essere errori di validazione degli indirizzi e-mail, di corrispondenza dei pattern o di campi mancanti. Ovviamente la verifica di ogni tipo comporta maggiori possibilità di sbagliare e richiede molto più codice. Tuttavia è più prolisso e questo talvolta favorisce la chiarezza.

Inoltre abbiamo accesso a `$dirty` e a `$pristine`. Questi due flag riconoscono se l'utente ha digitato qualcosa nel form e possono essere utilizzati per varie operazioni, tra cui aggiungere delle classi.

Poiché questo è semplicemente un modello, potremmo anche verificare se il form è valido dall'interno della funzione `submit` in `addCtl`:

```
$scope.submit = function(){

    if(!$scope.addForm.$valid){
        return window.alert('Error!');
    }

    $scope.contact.$save();
    $scope.contact = contacts.create();
    alert.show();
};
```

Se cancelliamo l'attributo `ng-disabled` e facciamo clic sul pulsante, vedremo comparire una finestra di avviso del browser. È un ottimo risultato, ma potremmo impostarlo ancora meglio facendolo sembrare parte dell'applicazione. Abbiamo già definito un messaggio di avviso di Bootstrap; aggiungiamolo rapidamente per i casi in cui si verifichi un errore di validazione.

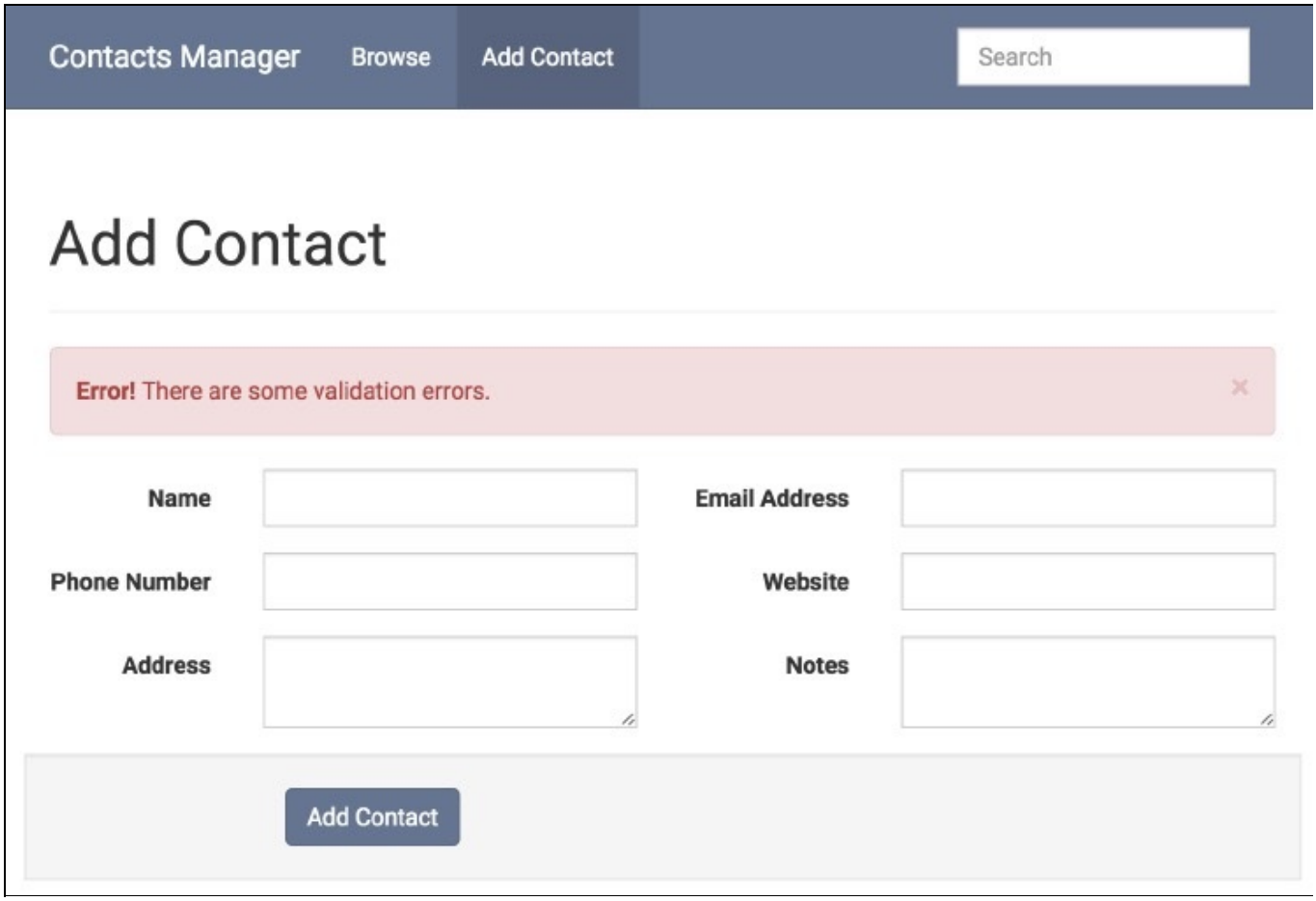
Sostituiamo la variabile `alert` all'inizio del file `add.js` con un nuovo oggetto contenente le finestre di avviso con messaggi di successo e di errore:

```
var alerts = {
    success: $alert({
        title: 'Success!',
        content: 'The contact was added successfully.',
        type: 'success',
        container: '#alertContainer',
        show: false
    }),
    error: $alert({
        title: 'Error!',
        content: 'There are some validation errors.',
        type: 'danger',
        container: '#alertContainer',
        show: false
    })
}
```

Ora possiamo sostituire la vecchia chiamata `alert` e la finestra di avviso del browser nel metodo `submit` del controller:

```
$scope.submit = function(){

    if(!$scope.addForm.$valid){
        return alerts.error.show();
    }

    $scope.contact.$save();
    $scope.contact = contacts.create();
    alerts.success.show();
};
```

Se ricaricate l'applicazione e fate clic sul pulsante *submit*, vedrete comparire un messaggio di avviso di Bootstrap che vi informa della presenza di alcuni errori di validazione.



Sarebbe molto utile sapere di quali errori si tratta. Fortunatamente AngularJS consente di scoprire quali modelli generano errori e definire di conseguenza errori o stili.

Possiamo accedere a ogni input del form, anche se è necessario specificare un nome per ciascuno al fine di riuscire a validarlo. Per esempio, se possiamo aggiungere un nome per il telefono nel campo riservato al numero telefonico, possiamo validare il campo accedendo a esso tramite il form:

```
addForm.phone.$valid
```

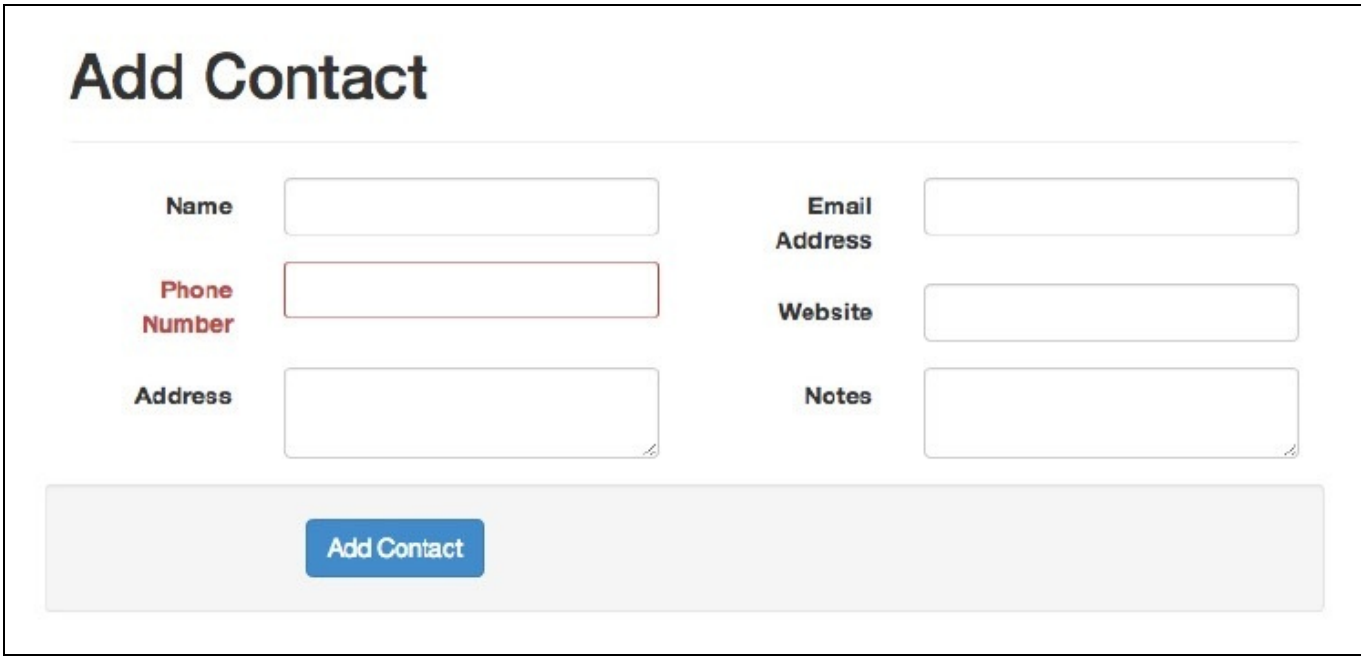
È possibile utilizzare la direttiva `ng-class` nei gruppi di form per verificare la validità dell'input e aggiungere la classe `has-error` nel caso in cui non fosse valido:

```
<div class="form-group" ng-class="{'has-error':
  !addForm.phone.$valid}">
```

NOTA

AngularJS aggiunge proprie classi agli elementi del form basate sulla validità, ma poiché intendiamo utilizzare la classe `has-error` di Bootstrap, abbiamo optato per `ng-class`.

Purtroppo questo codice aggiungerà, per impostazione predefinita, la classe `error`, e probabilmente non è questo il nostro obiettivo.



La soluzione consiste nell'impostare un secondo modello su `true` nel caso in cui il form non sia valido quando lo inviamo.

```
$scope.submit = function(){

    $scope.formErrors = false;

    if(!$scope.addForm.$valid){
        $scope.formErrors = true;
        return alerts.error.show();
    }

    $scope.contact.$save();
    $scope.contact = contacts.create();
    alerts.success.show();
};
```

Abbiamo impostato, all'inizio, `formErrors` su `false` per eliminare le classi `error` nel caso in cui il form sia validato correttamente. Possiamo modificare la direttiva `ng-class` per vedere sia le validazioni sia il nuovo modello:

```
<div class="form-group" ng-class="{'has-error': formErrors &&
  !addForm.phone.$valid}">
```

Se entrambi gli input non sono validi e il modello `formErrors` è impostato su `true`, verrà aggiunta la classe. Aggiungiamo la direttiva `ng-class` a tutti i gruppi di form in attesa di validazione. Non dimenticate di cambiare il modello al quale fate riferimento in quello che avete inserito come attributo `name`.

Validazione dei pattern

Abbiamo impostato una validazione di base ma, come sappiamo, Angular offre altre direttive per renderla più rigorosa. Per esempio, ora possiamo digitare tutto ciò

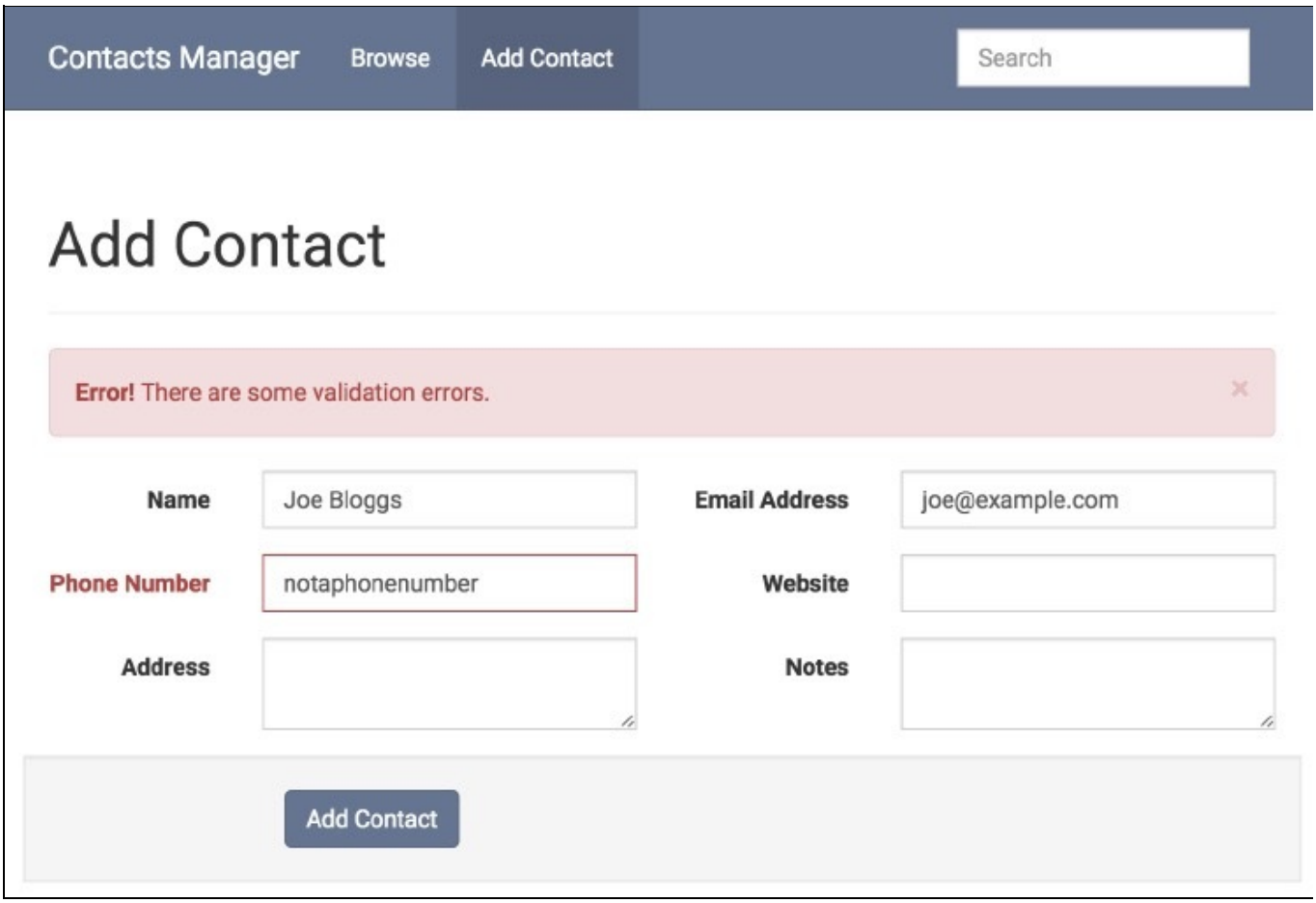
che desideriamo nel campo, e sarà aggiunto tutto. Ma questa non è una soluzione ideale poiché il nostro scopo è ottenere un numero di telefono per ogni contatto.

La direttiva `ng-pattern` consente di definire un pattern REGEX (espressione regolare) con cui confrontare l'input. Per il numero di telefono accetteremo le cifre, ma anche il segno più per i numeri internazionali e le parentesi per i numeri opzionali e quelli statunitensi. Consentiremo anche gli spazi e le lineette per suddividere i numeri.

Aggiungiamo la direttiva `ng-pattern` all'input riguardante il numero telefonico e limitiamolo per ora ai numeri interi:

```
<input type="tel" name="phone" id="phone" class="form-control" ng-
  model="contact.phone" required="true" ng-pattern="/^[0-9]/">
```

Quando digiteremo del testo nel campo e invieremo il form, vedremo che AngularJS genererà un errore di validazione.



Solo quando digiteremo un numero, il form potrà essere inviato. Dobbiamo ancora consentire i caratteri talvolta presenti in un numero telefonico. Si tratta di aggiungere i caratteri ammessi tra parentesi quadre. Ecco l'espressione regolare completa:

```
/^[0-9+()-]/
```

Potremmo anche estenderla e forzare una determinata struttura o lunghezza massima, ma per ora questo è il risultato che stavamo cercando.

Utilizzare minlength, maxlength, min e max

Le altre quattro direttive offerte da AngularJS per la validazione non sono così interessanti, ma si possono rivelare preziose. Per esempio, la direttiva `minlength` è ideale per garantire la sicurezza della password e `min`/`max` si dimostrano utili nei casi in cui è presente un carrello, consentendo soltanto un minimo di uno e un massimo di una determinata quantità di articoli.

Le quattro direttive vengono utilizzate nello stesso modo e accettano un numero come valore. Sia `ng-minlength` sia `ng-maxlength` considerano la lunghezza dell'input, mentre `ng-min` e `ng-max` considerano l'effettivo valore numerico.

Eccole in azione. Come per tutte le direttive, è possibile utilizzare una combinazione per ottenere le regole desiderate.

```
<input type="number" ng-min="1" ng-max="5">
<input type="password" ng-minlength="8" ng-maxlength="255">
```

Creare un validator personalizzato

AngularJS gestisce in modo efficace la maggioranza dei tipi di input e dei casi di utilizzo. Tuttavia, talvolta avrete necessità di avere un controllo maggiore. Fortunatamente è possibile creare proprie validazioni personalizzate. I validator personalizzati sono direttive con un requisito speciale. Per renderle operative è necessario impostare `ng-model` sull'elemento.

Siccome la nostra applicazione non ha un effettivo bisogno di un validator personalizzato, vediamo come è possibile crearne uno per verificare che l'input non sia presente in una lista predefinita. Può essere utile per far sì che un nome utente sia univoco.

Chiamiamo questa direttiva `uniqueList` e inseriamola nel file `contactsMgr.directives.js`. La limiteremo solo ad `attribute` e utilizzeremo il metodo `link`. Per un riepilogo veloce, rileggete il Capitolo 6 in cui abbiamo descritto la creazione di direttive personalizzate.

Possiamo inserire un controller come il quarto parametro della funzione `link`. Angular sa quale controller intendiamo utilizzare osservando la proprietà `require` della

direttiva. L'abbiamo impostata su `ngModel`, per avere un accesso diretto a un'API per la direttiva `ng-model`:

```
.directive('uniqueList', function(){
    return {
        restrict: 'A',
        require: 'ngModel',
        link: function(scope, elem, attrs, ctrl){

        }
    };
});
```

La documentazione di AngularJS illustra tutti i controller di base. Le informazioni su `ngModelController` sono reperibili all'indirizzo

https://docs.angularjs.org/api/ng/type/ngModel.NgModelController.

All'interno si trova un metodo chiave che possiamo sfruttare: `$setValidity`. Ci consente di definire se un modello è valido o meno. Lo utilizzeremo insieme con `scope.$watch` per verificare la validità qualora il modello cambi.

Impostiamo innanzitutto `scope.$watch` nel metodo link della direttiva. Possiamo recuperare il nome del modello dall'oggetto `attrs`:

```
.directive('uniqueList', function(){
    return {
        restrict: 'A',
        require: 'ngModel',
        link: function(scope, elem, attrs, ctrl){
            scope.$watch(attrs.ngModel, function(value){

            });
        }
    };
});
```

Come sappiamo, è possibile accedere al nuovo e unico valore da un watcher. Per questo esempio ci serve soltanto il nuovo valore. Lo confronteremo con una lista di nomi utente. Potrebbe essere una richiesta HTTP al server, ma per il momento sarà di tipo hard-code su un array o dei nomi:

```
.directive('uniqueList', function(){
    var usernames = [
        'bob',
        'john',
        'paul'
    ];

    return {
        restrict: 'A',
        require: 'ngModel',
        link: function(scope, elem, attrs, ctrl){
            scope.$watch(atts.ngModel, function(value){

            });
        }
    };
});
```

La validità del modello consiste in una semplice verifica per controllare se il valore si trova nell'array:

```
var valid = (usernames.indexOf(value) > -1) ? false : true;
```

Il metodo `$setValidity` sul controller `ngModel` è molto semplice e contiene soltanto due parametri. Il primo è la chiave `error` mentre il secondo è un booleano nel caso in cui sia valido o meno. Impostiamolo nel modo seguente:

```
var valid = (usernames.indexOf(value) > -1) ? false : true;
ctrl.$setValidity('uniqueList', valid);
```

Ora che tutto è definito, esaminiamo la direttiva completa e il suo utilizzo:

```
.directive('uniqueList', function(){
    var usernames = [
        'bob',
        'john',
        'paul'
    ];

    return {
        restrict: 'A',
        require: 'ngModel',
        link: function(scope, elem, attrs, ctrl){
            scope.$watch(atts.ngModel, function(value){
                var valid = (usernames.indexOf(value) > -1) ? false :
                  true;
                ctrl.$setValidity('uniqueList', valid);
            });
        }
    };
});
```

Per usarla, è sufficiente associarla all'input tramite un attributo:

```
<input type="text" ng-model="contact.name" unique-list>
```

Quiz

1. Indicate tre direttive che è possibile utilizzare per la validazione.
2. In che modo verifichiamo la validità del form?
3. Come possiamo accedere al controller `ngModel` durante la creazione di un validator personalizzato?
4. In che modo verifichiamo l’input sulla base di un’espressione regolare?

Riepilogo

Abbiamo visto come funziona la validazione in AngularJS e come è possibile combinarla con gli stili di Bootstrap per rendere l'app più *user friendly*. La validazione dei form è stata semplificata con le direttive integrate in AngularJS e la validazione automatica degli input HTML5, come `email` e `tel`.

Siamo riusciti a verificare la validità del form nella fase di invio e a visualizzare messaggi di errore e avvertimenti appropriati nel caso in cui questa operazione non abbia successo. Anche se i validator integrati sono ottimi per la maggior parte dei casi, abbiamo provato a svilupparne uno tramite una direttiva.

Ora che l'applicazione è completa, nel prossimo capitolo vedremo alcuni strumenti sviluppati dalla community che ci semplificano la vita quando operiamo con AngularJS.

Strumenti della community

La nostra app di gestione dei contatti è terminata. Siamo passati da una pagina vuota a un'applicazione CRUD *single-page* completa che si connette al server ed è interamente valida. Questo capitolo illustrerà due strumenti della community molto efficaci e utili che vi semplificheranno la vita quando utilizzerete AngularJS.

Imposteremo Batarang e ng-annotate per il nostro progetto. Il primo migliorerà, tra l'altro, l'aspetto dello scope, mentre il secondo ci consentirà di minificare molto più facilmente i file JavaScript. In questo capitolo affronteremo i seguenti argomenti.

- Batarang e ng-annotate.
- Installare Batarang e ng-annotate.
- Verificare lo scope.
- Monitorare la performance dell'app.
- Utilizzare ng-annotate con Grunt e gulp.

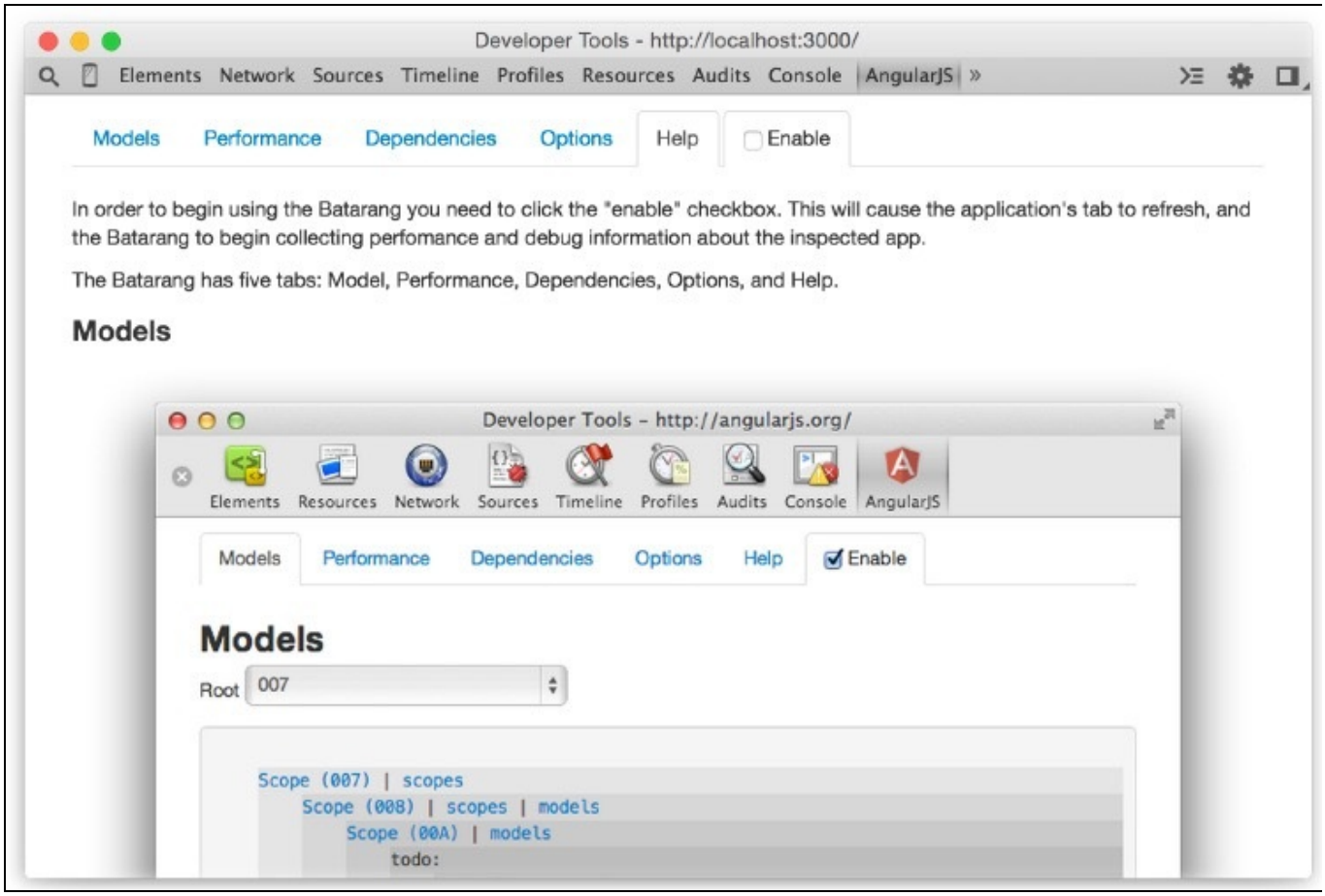
Batarang

Batarang è un'estensione di Chrome (ci spiace per gli utenti di Firefox e Safari, ma in fin dei conti AngularJS è un progetto di Google) che offre una scheda aggiuntiva negli strumenti degli sviluppatori per consentirci di creare un profilo ed effettuare il debugging delle app di AngularJS.

Installare Batarang

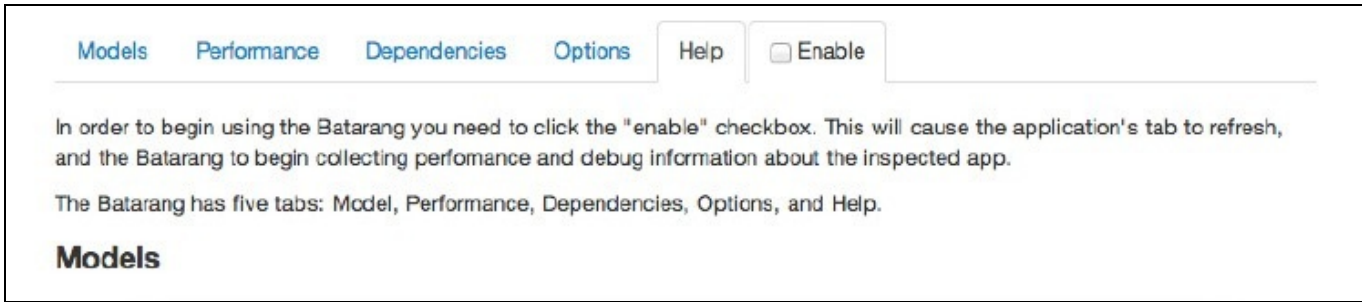
È facile installare Batarang poiché è un'estensione di Chrome. Visitiamo il sito `https://chrome.google.com/webstore/` e cerchiamo Batarang nella casella in alto a sinistra della pagina. Dovrebbe essere l'unico risultato nella sezione delle estensioni, come mostra la seguente schermata.

Fate clic sul pulsante per installarlo e vedrete comparire una nuova scheda nel web inspector, come mostra la seguente schermata.


Developer Tools - http://localhost:3000/
Elements Network Sources Timeline Profiles Resources Audits Console AngularJS
Models Performance Dependencies Options Help Enable
In order to begin using the Batarang you need to click the "enable" checkbox. This will cause the application's tab to refresh, and the Batarang to begin collecting perfomance and debug information about the inspected app.
The Batarang has five tabs: Model, Performance, Dependencies, Options, and Help.
Models
Developer Tools - http://angularjs.org/
Elements Resources Network Sources Timeline Profiles Audits Console AngularJS
Models Performance Dependencies Options Help Enable
Models
Root 007
Scope (007) | scopes
Scope (008) | scopes | models
Scope (00A) | models
todo:

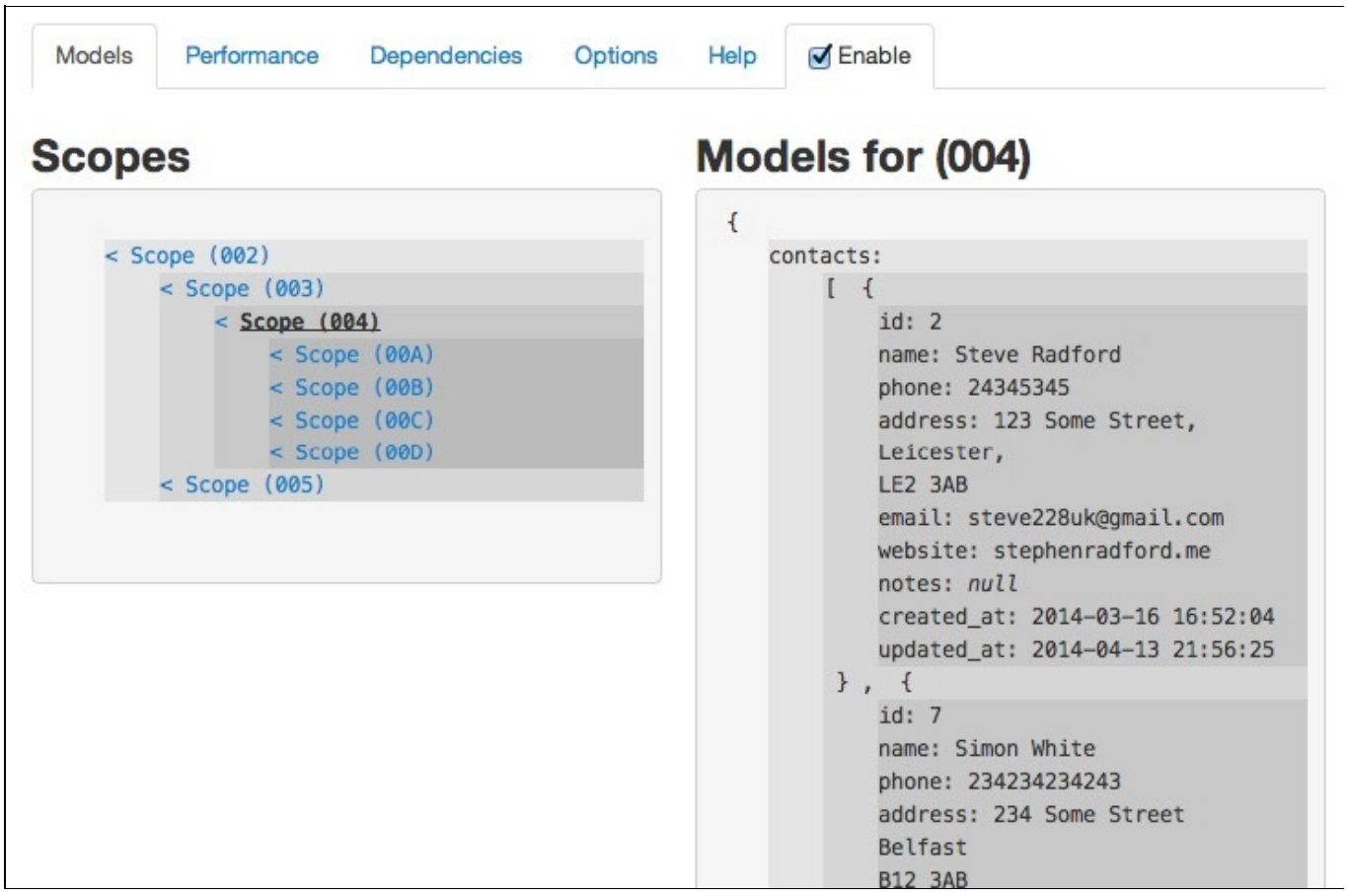
Verificare lo scope e le proprietà

Forse la funzione più utile di Batarang è la capacità di verificare i diversi scope presenti nell'applicazione. L'estensione aggiunge una nuova scheda al web inspector; esaminiamola.



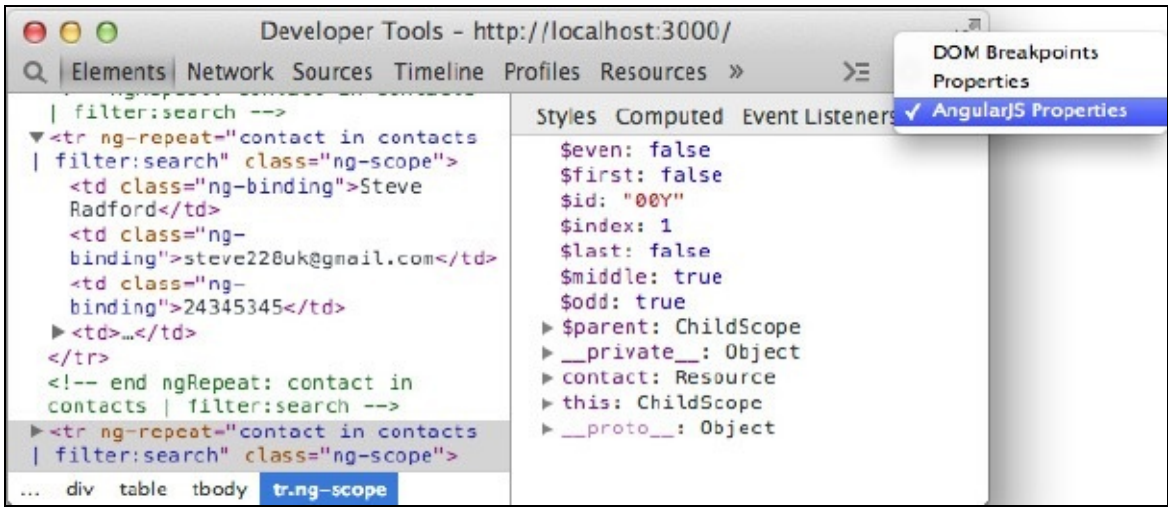
Batarang è costituito da cinque schede: *Models*, *Performance*, *Dependencies*, *Options* e *Help*. Per utilizzare Batarang nell'applicazione, è necessario selezionare la casella *Enable*. In questo modo la pagina si aggiornerà e l'estensione inizierà a raccogliere le informazioni necessarie.

Esistono tre sistemi per verificare lo scope e le proprietà mediante Batarang. Quello più ovvio è fare clic sulla scheda *Models*, dove vedrete una lista di tutti gli scope annidati.



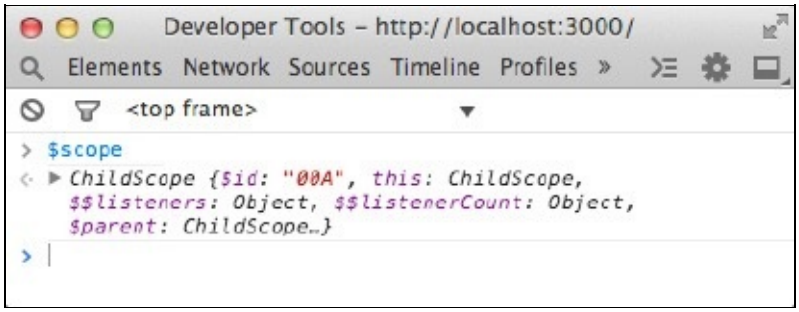
Alla sinistra della scheda *Models* è visibile una lista di tutti gli scope della pagina. Selezionandoli, compariranno a destra i modelli contenuti nello scope. Si aggiornano automaticamente quando cambia un valore sulla pagina, consentendo di vedere con estrema facilità ciò che accade al loro interno.

Batarang aggiunge anche un'altra scheda quando ispeziona gli elementi. Si chiama *AngularJS Properties* e mostra tutto ciò che Angular associa a un elemento: rileva se è pari, l'indice dell'item in un ng-repeat o se un elemento del form è valido o meno, come mostra la seguente schermata.



L'ultimo e utile strumento che offre Batarang per la verifica è quello presente nella console.

Dopo aver attivato Batarang, potrete digitare `$scope` nella console per vedere lo scope dell'ultimo elemento selezionato, come mostra la figura.

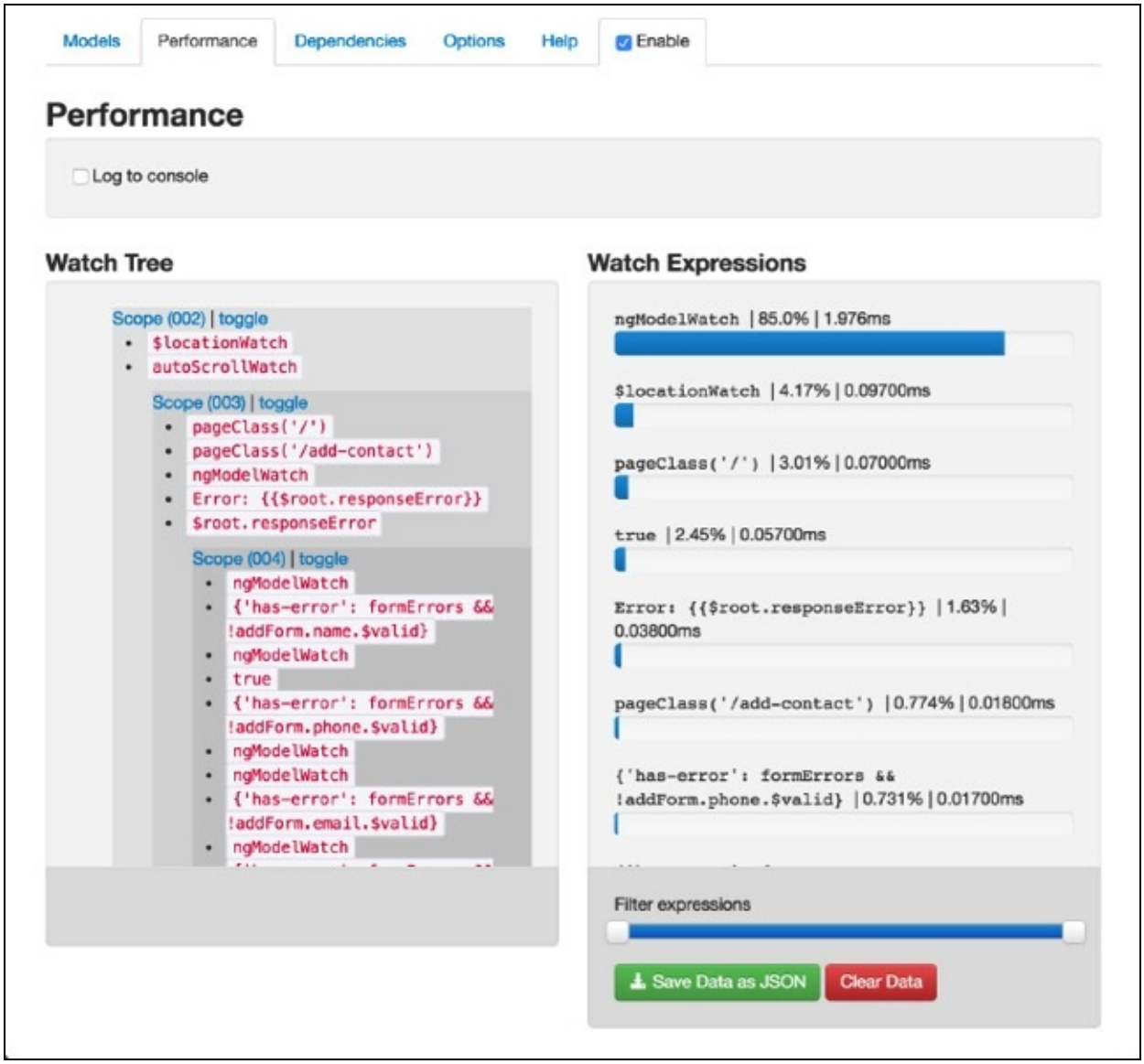


Monitorare la performance

La scheda *Performance* riporta un'utile lista di tutto ciò che viene verificato alla ricerca di eventuali cambiamenti da parte di Angular, oltre a un elenco di espressioni con la durata di esecuzione.

Vale la pena tenere d’occhio la performance mentre l’app si sviluppa. Forse scoprirete di aver esagerato con le funzioni `$scope.$watch` o che l’esecuzione di un filtro impiega molto tempo.

Quella che segue è la schermata che raffigura la scheda *Performance* della pagina *Add Contact* dell’applicazione. A sinistra compare la sezione *Watch Tree*. È una lista di tutti i watcher sulla pagina e mostra in quale scope si trovano. A destra è visibile un’analisi delle espressioni che impiegano più tempo.



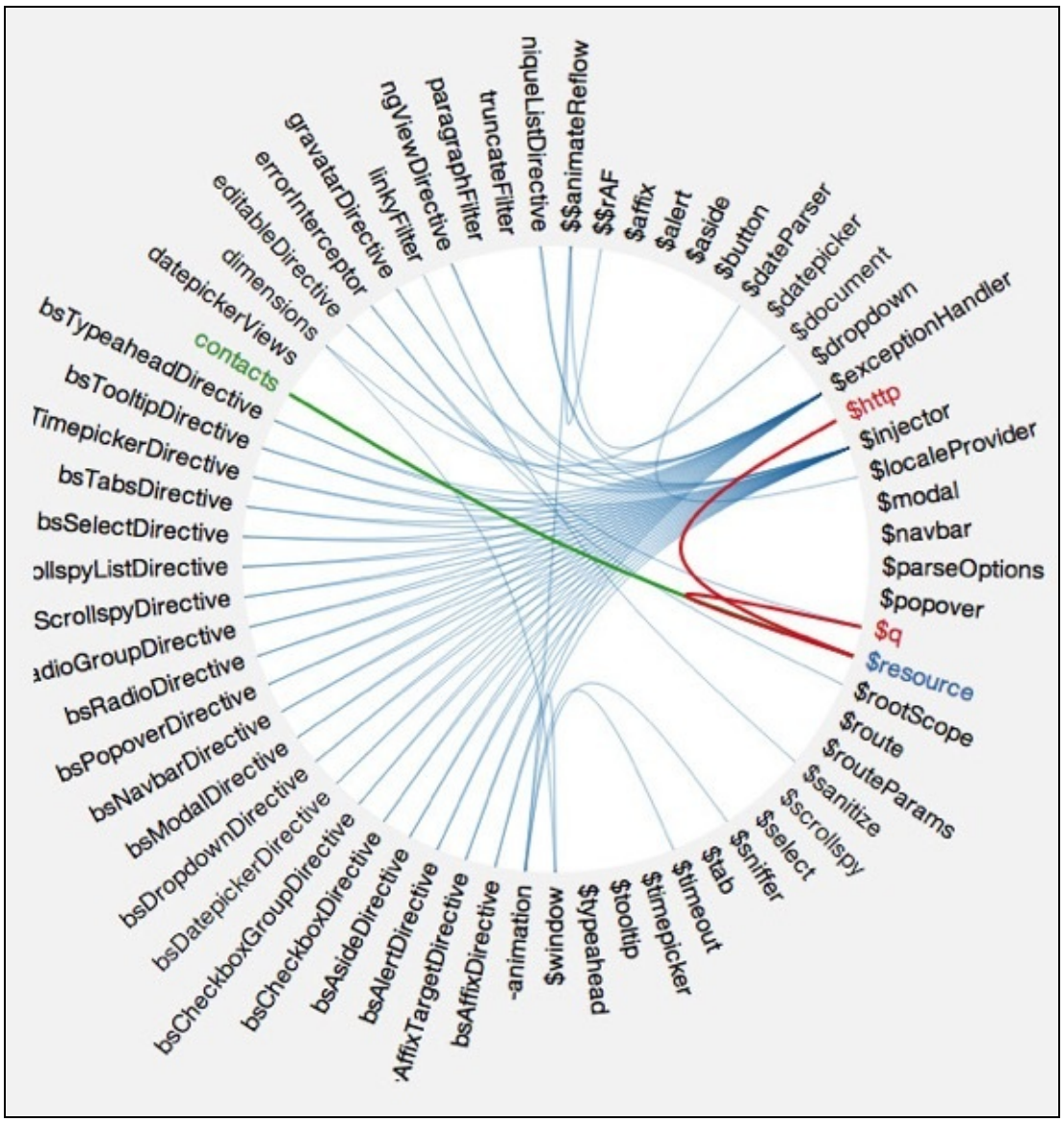
Visualizzare le dipendenze

La scheda più interessante di Batarang è sicuramente *Dependencies*. Contiene un grafico interattivo che mostra tutto ciò su cui si basano i servizi dell’applicazione.

Quando passate con il mouse sopra il nome di un servizio, i nomi dei servizi che si basano su di esso vengono evidenziati in verde, mentre i servizi che dipendono da

esso vengono evidenziati in rosso.

Nella schermata seguente, siamo passati con il mouse sopra *$resource*, che ha evidenziato *$http* e *$q* in rosso e *contacts* in verde:



Probabilmente avrete notato che nella schermata precedente ci sono alcuni servizi che non dipendono da nulla e da essi non dipende nulla. Accade abbastanza spesso, ma può anche significare che non vengono utilizzati dall’applicazione e quindi è opportuno tenerli d’occhio.

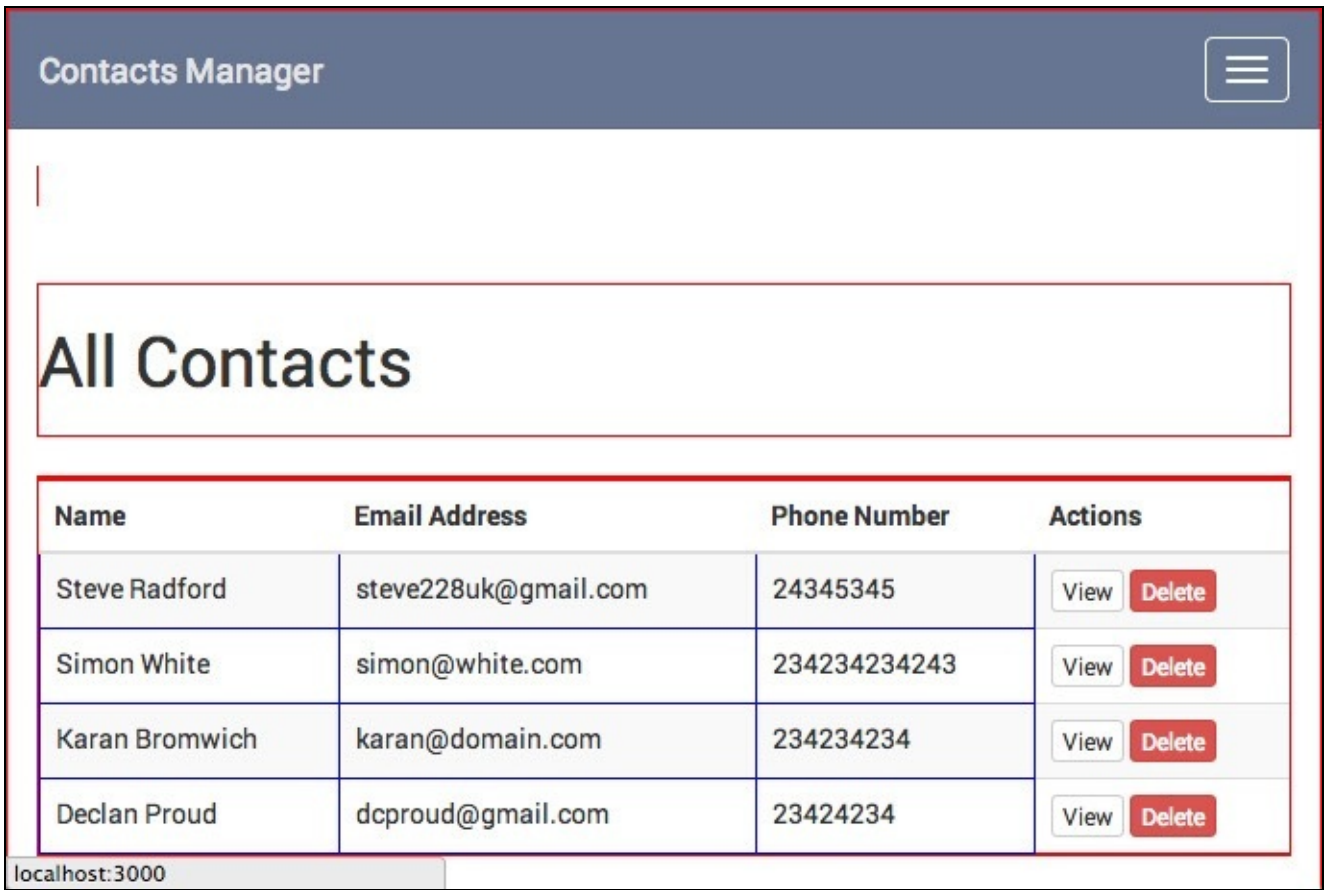
Opzioni di Batarang

Nella scheda *Options* sono presenti tre utili caselle di controllo, che evidenziano con un bordo colorato le applicazioni, i binding e gli scope sulla pagina.

Le applicazioni presentano un bordo verde, i binding un bordo blu mentre gli scope uno rosso. Si tratta di una raffigurazione efficace per capire se una parte

dell’app si trova in un determinato scope o se si sta veramente verificando un binding.

Ecco come appare l’app dopo aver attivato tutto:



ng-annotate

ng-annotate è un progetto open source di Olov Lassus teso a eliminare le conseguenze più frustranti derivanti dall'utilizzo di AngularJS. Ricorderete che quando abbiamo minificato il codice, abbiamo racchiuso le dipendenze in un array per tenere in ordine i nomi. Il progetto rende superfluo questo requisito osservando il codice e racchiudendolo al posto nostro. In sintesi lo strumento è un pre-minificatore e dovremmo usarlo per preparare il codice per l'attività `uglify`.

In passato abbiamo utilizzato ngMin di Brian Ford per ottenere lo stesso risultato. Questo progetto è stato deprecato in favore di ng-annotate.

Installare ng-annotate

Lo strumento ng-annotate è un pacchetto `npm` e può essere installato tramite riga di comando. Esistono altri pacchetti che funzionano con le installazioni di Grunt e gulp, ma per il momento esaminiamo come è possibile eseguire manualmente il progetto.

1. Aprite il terminale ed eseguite il prossimo comando per installare globalmente il pacchetto sul computer:

```
npm install –g ng-annotate
```

> **NOTA**
>
> Ricordate che se restituisce un errore legato ai permessi, dovrete eseguirlo come amministratore. Potete farlo con il comando `sudo` in un sistema basato su *nix oppure eseguendo il prompt dei comandi come amministratore su Windows.

Vi fornirà un nuovo comando da utilizzare nel terminale. Se eseguite `ng-annotate`, dovrebbe comparire il file help per lo strumento. È facile utilizzare ng-annotate in un file: è sufficiente avvalersi del comando `ng-annotate` seguito dalle opzioni e dal nome del file che intendiamo elaborare.

2. Nella directory *js* del progetto eseguite

```
ng-annotate -r  controllers/app.js
```

Dovrebbe restituire il contenuto del controller dell'app ma con una piccola variazione: vedrete che tutte le annotazioni esistenti attorno alle dipendenze sono state cancellate. Infatti abbiamo comunicato a ng-annotate di eliminarle con l'opzione `-r`.

3. Possiamo anche far sì che ng-annotate le aggiunga nuovamente mediante `-a`. Queste due opzioni possono essere utilizzate insieme o da sole:

```
ng-annotate -ra controllers/app.js
```

4. Questa volta viene restituito il controller con le annotazioni. Possiamo notare che ng-annotate ha operato la sua magia perché le virgolette singole che racchiudono le dipendenze sono state sostituite da virgolette doppie.

Ora `ng-annotate` funziona, ma è inutile se è necessario eseguirlo manualmente. In che modo è possibile inserirlo nei task runner? Scopriamolo.

Utilizzare ng-annotate con Grunt

Per utilizzare ng-annotate con Grunt, dobbiamo installare un altro pacchetto `npm`. Questa volta non è globale, ma è necessario installarlo nel progetto. Seguite questi passi.

1. Passate alla cartella del progetto nel terminale:

```
cd ~/path/to/contacts-manager
```

2. Ora installate il modulo Grunt salvandolo nelle dipendenze `dev` del progetto in modo da poterlo sempre installare nuovamente in seguito; inoltre un altro sviluppatore che lavorerà al progetto potrà installare così tutto il necessario:

```
npm install grunt-ng-annotate --save-dev
```

3. Prima di effettuare qualsiasi modifica nel Gruntfile, non dimenticate di caricare le attività dal pacchetto appena installato:

```
grunt.loadNpmTasks('grunt-ng-annotate');
```

Configurare l'attività

Non ci resta che configurare l'attività `ngAnnotate` nel Gruntfile ed effettuare due piccole modifiche nell'attività `watch`. Ecco come procederemo.

1. Aggiungiamola nell'attività `less` all'interno dell'oggetto `grunt.initConfig`.

```
ngAnnotate: {

}
```

2. Come la maggior parte delle attività di Grunt, `ngAnnotate` accetta un hash `options` oltre a più target. Impostiamo prima `options`:

```
ngAnnotate: {
    options: {
        remove: true,
        add: true,
        singleQuotes: true
```

```
    }
}
```

Abbiamo definito tre opzioni. Abbiamo scelto di eliminare qualsiasi annotazione preesistente per aggiungerne di nuove con ng-annotate e utilizzare virgolette singole e non doppie. È anche possibile definire un'espressione regolare se intendiamo operare in una sezione specifica.

3. Il target guarderà tutti i file nella directory *js* e li salverà con l'estensione `.annotated.js`. In questo modo è possibile impostare l'attività `watch` per eseguire ng-annotate e poi l'attività `uglify` affinché cerchi i file che terminano con `.annotated.js`.

4. Gli unici file ai quali dobbiamo prestare attenzione sono i nostri. Tutto nella directory *vendor* sarà preannotato, o non sarà legato ad Angular. Possiamo configurare Grunt affinché consideri tutto tranne le cartelle *vendor* e *build*:

```
ngAnnotate: {
    options: {
        remove: true,
        add: true,
        singleQuotes: true
    },
    app: {
        src: [
            'assets/js/**/*.js',
            '!assets/js/vendor/*.js',
            '!assets/js/build/*.js'
        ]
    }
}
```

5. Il primo percorso nell'array `src` include tutti i file JS nella directory *js* del progetto. Come abbiamo previsto, è necessario ignorare le cartelle *vendor* e *build*. I due item che seguono nell'array fanno proprio questo. Il punto esclamativo prima del percorso indica a Grunt che intendiamo escludere tutti i file JS in queste cartelle.

6. Dopo aver specificato il sorgente, è necessario comunicare a ng-annotate ciò che vogliamo fare con i file dopo che sono stati elaborati. Intendiamo rinominare il file per includere l'estensione `.annotate.js` e salvarli nello stesso posto:

```
ngAnnotate: {
    options: {
        remove: true,
        add: true,
        singleQuotes: true
    },
    app: {
        src: [
            'assets/js/**/*.js',
            '!assets/js/vendor/*.js',
            '!assets/js/build/*.js'
        ],
        expand: true,
        ext: '.annotated.js',
        extDot: 'last'
    }
}
```

7. Nel codice precedente abbiamo aggiunto tre nuove proprietà per il target: `expand`, `ext` ed `extDot`. La proprietà `expand` divide il percorso e ci consente di cambiare l'estensione. La proprietà `ext` modifica l'estensione mentre la proprietà `extDot` comunica a Grunt quale punto nel nome file guardare. Nel nostro caso è l'ultimo e questo ci tutela nel caso dovessimo utilizzare più punti nei nomi file.

8. Ora siamo pronti a eseguire l'attività nel terminale. Eseguiamola con il flag `--verbose` e vediamo ciò che accade:

```
grunt ngAnnotate --verbose
```

9. Tutto sembra funzionare alla perfezione ma creando i nuovi file `.annotated.js`, avremo dei problemi quando eseguiremo l'attività una seconda volta. Otterremo questo output quando eseguiremo nuovamente il comando precedente:

```
Writing assets/js/modules/contactsMgr.services.annotated.
  annotated.js
```

10. L'attività ha preso in considerazione non solo i file `.js` nelle directory ma anche i file `.annotated.js`. Ciò accade perché Grunt non conosce la differenza. Siccome prende l'ultimo punto per determinare l'estensione, è sufficiente aggiungere un'altra esclusione all'array `src` per completare l'attività:

```
ngAnnotate: {
    options: {
        remove: true,
        add: true,
        singleQuotes: true
    },
    app: {
        src: [
            'assets/js/**/*.js',
            '!assets/js/**/*.annotated.js',
            '!assets/js/vendor/*.js',
            '!assets/js/build/*.js'
        ],
        expand: true,
        ext: '.annotated.js',
        extDot: 'last'
    }
}
```

1. Cancellate gli altri file `.annotated.annotated.js` nella directory *js* ed eseguite l'attività un'ultima volta.

Impostare l'attività watch

Ora che l'attività `ngAnnotate` funziona come previsto, è possibile effettuare alcune modifiche nelle attività `watch` e `uglify` per eseguirla automaticamente e minificare i file JavaScript, nel modo seguente.

1. Innanzitutto è necessario modificare il target `js` nell'attività `watch` per eseguire `ngAnnotate` invece di `uglify`:

```
js: {
    files: [
        'assets/js/**/*.js'
    ],
    tasks: ['ngAnnotate']
},
```

2. Aggiungiamo anche due esclusioni nell'array `files`. Non dobbiamo osservare la directory *build*, ed è necessario comunicare a Grunt di ignorare tutti i file che terminano con `.annotated.js`:

```
js: {
    files: [
        'assets/js/**/*.js',
        '!assets/js/build/*.js',
        '!assets/js/modules/**/*.annotated.js',
        '!assets/js/controllers/**/*.annotated.js'
    ],
    tasks: ['ngAnnotate']
},
```

3. Verifichiamo rapidamente se funziona tutto. Nel terminale avviamo l'attività `watch`:

```
grunt watch
```

4. Ora se salvate un file JS, come il controller dell'app, Grunt dovrebbe rilevare una modifica e attivare `ngAnnotate`:

```
Running "watch" task
Waiting…OK
>> File "assets/js/controllers/app.js" changed.

Running "ngAnnotate:app" (ngAnnotate) task
>> 8 files successfully generated.

Done, without errors.
```

5. Sembra funzionare bene. Possiamo creare un nuovo target che controlli i file annotati alla ricerca delle modifiche ed esegua l'attività `uglify`:

```
annotated: {
    files: [
        'assets/js/**/*.annotated.js',
    ],
    tasks: ['uglify']
},
```

6. È facile, e cerca le modifiche nei file con estensione `.annotated.js`. Infine dobbiamo effettuare due modifiche nell'array `src` all'interno del target di build di `uglify`:

```
src: [
    'assets/js/vendor/jquery.js',
    'assets/js/vendor/bootstrap.js',
```

```
        'assets/js/vendor/angular.js',
        'assets/js/vendor/angular-animate.js',
        'assets/js/vendor/angular-resource.js',
        'assets/js/vendor/angular-route.js',
        'assets/js/vendor/angular-sanitize.js',
        'assets/js/vendor/angular-strap.js',
        'assets/js/vendor/angular-strap.tpl.js',
        'assets/js/modules/*.annotated.js',
        'assets/js/controllers/*.annotated.js'

    ],
```

7. Abbiamo effettuato due modifiche agli ultimi due item nell'array per guardare i file generati invece dei loro omologhi non annotati. Eseguiamo nuovamente l'attività `watch` di Grunt e risalviamo un file JS per vedere cosa succede:

```
Running "watch" task
Waiting…OK
>> File "assets/js/controllers/app.js" changed.

Running "ngAnnotate:app" (ngAnnotate) task
>> 8 files successfully generated.

Done, without errors.
Running "uglify:build" (uglify) task
File "assets/js/build/ContactsMgr.js" created.

Done, without errors.
```

8. Quando abbiamo salvato il file `app.js`, Grunt ha rilevato una modifica e ha eseguito l'attività `ngAnnotate` generando otto file. L'attività `watch` ha rilevato la presenza di modifiche nei file annotati appena generati e ha creato il file `ContactsMgr.js` eseguendo l'attività `uglify`. Dopo tutti questi cambiamenti, ecco come appare `Gruntfile.js`:

```
module.exports = function(grunt){

  grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    watch: {
      js: {
        files: [
        'assets/js/**/*.js',
        '!assets/js/build/*.js',
        '!assets/js/**/*.annotated.js'
        ],
        tasks: ['ngAnnotate']
      },
      annotated: {
        files: [
        'assets/js/**/*.annotated.js',
        ],
        tasks: ['uglify']
      },
        less: {
        files: [
        'assets/less/*.less'
        ],
        tasks: ['less:dev']
      },
        css: {
          files: [
          'assets/css/bootstrap.css'
          ],
        options: {
          livereload: true
```

```
        }
      }
    },
      uglify: {
        options: {
        banner: '/*! <%= pkg.name %> <%=
          grunt.template.today("yyyy-mm-dd") %> */\n'
            },
              build: {
              src: [
              'assets/js/vendor/jquery.js',
      'assets/js/vendor/bootstrap.js',
          'assets/js/vendor/angular.js',
          'assets/js/vendor/angular-animate.js',
          'assets/js/vendor/angular-resource.js',
          'assets/js/vendor/angular-route.js',
          'assets/js/vendor/angular-sanitize.js',
          'assets/js/vendor/angular-strap.js',
          'assets/js/vendor/angular-strap.tpl.js',
          'assets/js/modules/**/*.annotated.js',
          'assets/js/controllers/**/*.annotated.js'
          ],
          dest: 'assets/js/build/<%= pkg.name %>.js'
        }
      },
        less: {
          dev: {
          files: {
          'assets/css/bootstrap.css':
                'assets/less/bootstrap.less'
        }
      },
        production: {
        options: {
        cleancss: true
        },
        files: {
          'assets/css/bootstrap.css':
                'assets/less/bootstrap.less'
        }
      }
    },
      ngAnnotate: {
        options: {
          remove: true,
          add: true,
          singleQuotes: true
          },
          app: {
            src: [
              'assets/js/**/*.js',
              '!assets/js/**/*.annotated.js',
              '!assets/js/vendor/*.js',
              '!assets/js/build/*.js'
              ],
                expand: true,
                ext: '.annotated.js',
                extDot: 'last'
            }
        }
    });

    grunt.loadNpmTasks('grunt-contrib-uglify')
    grunt.loadNpmTasks('grunt-contrib-watch');
    grunt.loadNpmTasks('grunt-contrib-less');
    grunt.loadNpmTasks('grunt-ng-annotate');

    grunt.registerTask('default', ['ngAnnotate', 'uglify']);

};
```

Utilizzare ng-annotate con gulp

Proprio come per qualsiasi cosa abbiamo visto finora, esiste anche una versione gulp di ng-annotate che ci consentirà di utilizzare questo task runner alternativo, se è quello che abbiamo scelto. Studiamo come è possibile impostarlo affinché operi con gulpfile preesistente.

1. Recuperiamo il pacchetto da `npm`:

```
npm install gulp-ng-annotate --save-dev
```

2. Apriamo `gulpfile.js` e inseriamo il pacchetto appena installato:

```
var ngAnnotate = require('gulp-ng-annotate');
```

3. In questo modo avremo una nuova funzione da utilizzare con gli operatori pipe di gulp. È incredibile la maggiore rapidità di installazione di gulp per utilizzare ng-annotate rispetto a Grunt. È sufficiente aggiungere un nuovo pipe all'attività `uglify`:

```
gulp.task('uglify', function(){
    gulp.src(paths.js)
    .pipe(concat(pkg.name+'.js'))
    .pipe(ngAnnotate())
    .pipe(uglify())
    .pipe(gulp.dest('assets/js/build'));
});
```

4. Il pipe con la funzione `ngAnnotate` può essere inserito prima del pipe `uglify`. L'attività `watch` è già impostata per eseguire l'attività `uglify` ogni volta che cambia un file JS; pertanto questo è tutto ciò che dobbiamo fare in gulpfile.

5. Ora possiamo ricorrere a `uglify` di gulp per eseguire manualmente ng-annotate e minificare il JavaScript, oppure utilizzare `watch` di gulp per rilevare automaticamente i cambiamenti. Dopo queste due piccole modifiche, ecco come appare il file `gulpfile.js` :

```
var gulp = require('gulp');
var uglify = require('gulp-uglify');
var concat = require('gulp-concat');
var pkg = require('./package.json');
var less = require('gulp-less');
var livereload = require('gulp-livereload');
var ngAnnotate = require('gulp-ng-annotate');

var paths = {
    js: [
        'assets/js/vendor/jquery.js',
        'assets/js/vendor/bootstrap.js',
        'assets/js/vendor/angular.js',
        'assets/js/vendor/angular-animate.js',
        'assets/js/vendor/angular-resource.js',
        'assets/js/vendor/angular-route.js',
        'assets/js/vendor/angular-sanitize.js',
        'assets/js/vendor/angular-strap.js',
        'assets/js/vendor/angular-strap.tpl.js',
        'assets/js/modules/**/*.js',
        'assets/js/controllers/**/*.js'

    ],
    less: 'assets/less/**/*.less'
```

```
};

gulp.task('uglify', function(){
    gulp.src(paths.js)
    .pipe(concat(pkg.name+'.js'))
    .pipe(ngAnnotate())
    .pipe(uglify())
    .pipe(gulp.dest('assets/js/build'));
});

gulp.task('watch', function(){

    var server = livereload();
    gulp.watch(paths.js, ['uglify']);
    gulp.watch(paths.less, ['less']);
    gulp.watch('assets/css/bootstrap.css').on('change',
  function(file){
        server.changed(file.path);
    });

});

gulp.task('less', function(){
    gulp.src('assets/less/bootstrap.less')
    .pipe(less({
        filename: 'bootstrap.css'
    }))
    .pipe(gulp.dest('assets/css'));
});

gulp.task('default', ['uglify']);
```

Quiz

1. Che cos’è Batarang?
2. Indicate tre strumenti che offre Batarang.
3. Quali sono i due sistemi per verificare lo scope?
4. Che cosa abbiamo utilizzato prima di ng-annotate?
5. Quali opzioni offre ng-annotate?
6. Quale problema risolve ng-annotate?

Riepilogo

In questo capitolo abbiamo trattato due strumenti della community efficaci e molto affidabili e utilizzati. Batarang rappresenta il coltellino svizzero degli strumenti di debugging che aiuta a sviluppare fantastiche web app con Angular; ng-annotate ci solleva dall'irritante incombenza delle annotazioni durante la minificazione dei file.

Questi due strumenti sono facoltativi e non sono necessari quando si utilizza Angular, ma entrambi vi aiuteranno lungo il cammino e li sfrutterete regolarmente. Sono soltanto due degli strumenti creati dalla sorprendente community di AngularJS. Esplorate e scoprite che cos'altro offre per facilitarvi la vita. E se non riuscite proprio a trovare ciò che state cercando, sviluppatelo voi e proponetelo alla community!

Persone e progetti

Sia AngularJS sia Bootstrap vantano un nutrito seguito e due enormi community, a dimostrazione che i progetti basati su entrambi i framework sono numerosi. Dietro ogni progetto operano sviluppatori scrupolosi. Scopriamo allora alcune persone e progetti degni di nota.

Progetti e persone dietro Bootstrap

Attualmente Bootstrap è giunto alla terza versione; ciò significa che esistono numerose estensioni e strumenti utili che vi possono aiutare durante lo sviluppo.

Il team principale

Bootstrap è nato all'interno di Twitter ed è la creatura di due ingegneri dell'azienda: Mark Otto (`@mdo`) e Jacob Thornton (`@fat`). Tutti e due hanno in seguito lasciato questa compagnia, ma continuano a contribuire al progetto.

Attualmente si contano quasi 600 collaboratori a Bootstrap, il che sottolinea la forza di un software open source.

Entrambi i creatori originari hanno abbandonato Twitter, ma Mark continua a essere il collaboratore e gestore più attivo di Bootstrap.

- *URL*: `http://www.getbootstrap.com`
- *Twitter*: `@twbootstrap`
- *Persone*: Mark Otto (`@mdo`), Jacob Thornton (`@fat`) e altri.

Bootstrap Expo

Quando parliamo con chi conosce un po' Bootstrap ma non l'ha ancora sperimentato, scopriamo che è convinto che il framework sia uno stile fisso. Buona parte della sua cattiva fama deriva dai primi progetti basati su di esso. In passato ogni pagina dei plug-in jQuery veniva sviluppata con Bootstrap privo di stili.

Come sappiamo, Bootstrap è molto più di questo; è un framework front-end a tutto tondo. Per sfatare questa convinzione errata, un creatore di Bootstrap ha aperto il blog Bootstrap Expo in cui illustra gli utilizzi più stimolanti di Bootstrap nel Web.

- *URL*: `http://expo.getbootstrap.com`
- *Persone*: Mark Otto (`@mdo`)

BootSnipp

BootSnipp è una risorsa incredibile per chiunque operi con Bootstrap. Nella sua versione più semplice, è una raccolta di componenti precodificati che è possibile

copiare e incollare nel progetto. È possibile trovare strumenti per la navigazione, immagini scorrevoli e finestre modali accompagnate con uno stile.

Ma BootSnipp non è solo questo. Potete effettuare una ricerca in base alle versioni di Bootstrap, esaminare le altre preziose risorse e utilizzare gli utili costruttori di form e pulsanti nei quali è sufficiente trascinare gli elementi desiderati e copiare l'HTML.

- *URL*: `http://www.bootsnipp.com`
- *Twitter*: `@BootSnipp`
- *Persone*: Maksim Surguy (`@msurguy`)

Linee guida sul codice di @mdo

Quando il progetto si amplierà, dovrete seguire alcuni standard per quanto riguarda HTML e CSS. Ovviamente quando lavorerete a un progetto con altre persone, la situazione si può complicare e la necessità di una guida sugli stili risulterà più evidente.

Mark Otto, uno dei co-creatori di Bootstrap, ha ideato una guida esauriente sugli standard che utilizza nei suoi progetti. Vale la pena dare un'occhiata, anche se una buona parte dipende dalle preferenze personali e dalle esigenze vostre e del vostro team. Servitevene come punto di partenza per definire i set di standard e di regole.

- *URL*: `http://codeguide.co/`
- *Persone*: Mark Otto (`@mdo`)

Roots

Roots è uno starter theme open source di WordPress che accompagna Bootstrap, Grunt e Boilerplate HTML5 per sviluppare fantastici temi WordPress. Anche se non abbiamo affrontato questo argomento nel corso del libro, vorremmo segnalarvi la possibilità che Bootstrap offre per realizzare qualsiasi cosa. È una piattaforma molto versatile e solida su cui creare i progetti.

- *URL*: `http://roots.io/`
- *Persone*: Ben Word (`@retlehs`)

Shoelace

Se avete delle difficoltà a sviluppare o visualizzare le griglie con Bootstrap, vale la pena dare un'occhiata a Shoelace. È un pratico strumento che consente di sviluppare in modo interattivo la griglia delle applicazioni e generare tutto il markup di cui abbiamo bisogno in HTML, Jade o EDN.

È possibile salvare e condividere le griglie e verificare il loro aspetto su dispositivi di diverse dimensioni. Si possono aggiungere altre classi alle righe e alle colonne, anche se per impostazione predefinita è possibile utilizzare soltanto le colonne nei dispositivi di piccole dimensioni. Ovviamente potrete sostituirle in seguito, se fosse necessario.

- *URL*: `http://www.shoelace.io`
- *Persone*: Erik Flowers (`@Erik_UX`) e Shaun Gilchrist

Snippet di Bootstrap 3 per Sublime Text

Sublime Text 2 e 3 sono editor di testo molto popolari tra gli sviluppatori e non sorprende che esista un plug-in che include innumerevoli snippet per il vostro framework front-end preferito.

Il plug-in vi consente di inserire rapidamente qualsiasi componente di Bootstrap semplificandone molto l'utilizzo. Può essere installato tramite Package Control cercando `Bootstrap 3 Snippets`, oppure potete procurarvelo da GitHub.

- *URL*: `https://github.com/JasonMortonNZ/bs3-sublime-plugin`
- *Persone*: Jason Morton (`@JasonMortonNZ`)

Font Awesome

Il progetto ha preso il via per sostituire le icone di immagine di Bootstrap 2 con alcune equivalenti. Si è sviluppato fino a diventare una delle principali raccolte di font di icone che può essere utilizzata con o senza Bootstrap.

Se desiderate estendere la gamma di icone già ricca di Bootstrap, provate Font Awesome.

- *URL*: `http://fortawesome.github.io/Font-Awesome/`

- *Persone*: Dave Gandy (`@davegandy`)

Bootstrap Icons

Bootstrap comprende già una fantastica collezione di icone e si può facilmente estendere con Font Awesome. Bootstrap Icons è stato creato per cercare rapidamente tra le icone e recuperare la classe necessaria.

È molto più facile trovare qui ciò che vi serve rispetto alla documentazione ufficiale, perché ogni icona è stata taggata con più parole chiave. Digitate ciò che cercate nella casella di ricerca e il sito lo troverà.

- *URL*: `http://bootstrapicons.com/`
- *Persone*: Brent Swisher (`@BrentSwisher`)

Progetti e persone dietro AngularJS

Alcuni progetti della community di AngularJS possono diventare parte del flusso di lavoro quotidiano. Ne abbiamo già esaminati alcuni nel libro, ma ora ne descriveremo altri di una certa importanza.

Il team principale

Come sapete, AngularJS è un progetto Google e, come tale, il suo team principale è costituito da dipendenti di questa azienda. Forse non sapete però che il framework fu creato in Brat Tech LLC da Miško Hevery e Adam Abrons per supportare un servizio di archiviazione JSON chiamato `http://getangular.com/`. In seguito abbandonarono il servizio e resero open source il framework AngularJs che abbiamo imparato a conoscere e apprezzare.

Miško continua a gestire il progetto come dipendente Google collaborando con diversi altri ingegneri. Insieme l'hanno ampliato, rilasciato un'infinità di moduli aggiuntivi e creato la popolare estensione per Chrome Batarang.

- *URL*: `https://angularjs.org/`
- *Persone*: Miško Hevery (`@mhevery`), Adam Abrons (`@abrons`), Brian Ford (`@briantford`), Brad Green (`@bradlygreen`), Igor Minar (`@IgorMinar`), Votja Jína (`@vojtajina`) e altri.

RestAngular

Abbiamo già illustrato i due sistemi con cui Angular consente di connettersi a un'API: `$http` e `ngResource`. Esiste anche un progetto della community molto popolare chiamato RestAngular che segue un approccio diverso.

La differenza principale rispetto a `ngResource` consiste nel fatto che utilizza le promise, mentre `$resource` esegue automaticamente il loro unwrapping. A seconda di come funziona il vostro progetto, può essere un sistema efficace poiché potrete risolvere i dati della route mediante `$routeProvider.resolve`.

Se operate con una RESTful API nel progetto e `ngResource` non vi convince del tutto, date un'occhiata a RestAngular.

- *URL*: `https://github.com/mgonto/restangular`

- *Persone*: Marin Gonto (`@mgonto`)

AngularStrap e AngularMotion

Abbiamo già studiato e utilizzato AngularStrap nel nostro progetto. Offre una raccolta fantastica di tutti i principali plug-in di Bootstrap per le direttive native di AngularJS. Se utilizzate Bootstrap insieme con AngularJS (come abbiamo fatto nel libro), è un modulo imperdibile.

AngularMotion è progettato per venire utilizzato con AngularStrap, anche se non è strettamente necessario. Si tratta di animazioni già pronte che funzionano nativamente con `ngAnimate` e aggiungono un tocco in più al progetto. Si possono utilizzare con `ng-show` e `ng-hide` e con direttive come `ng-repeat` per animare l'aggiunta o la cancellazione degli item.

- *URL*: `http://mgcrea.github.io/angular-strap/` e `http://mgcrea.github.io/angular-motion`
- *Persone*: Olivier Louvignes (`@olouv`)

AngularUI

Il progetto AngularUI è probabilmente il più grande che è sorto dalla community di AngularJS. Si divide in diversi moduli tra cui UI-Utils, UI-Modules e UI-Router.

Il modulo UI-Utils viene descritto come il "coltellino svizzero" degli strumenti, ed effettivamente lo è. Consente operazioni come evidenziare del testo, verificare le pressioni di tasti o rendere un elemento fisso allo scroll.

Gli UI-Modules sono moduli AngularJS con dipendenze esterne per Google Maps o plug-in jQuery. Qui troverete alcuni strumenti efficaci, e personalmente ho utilizzato il modulo Select2 in diverse occasioni.

Forse il progetto più popolare è il modulo UI-Router. Offre un vero routing annidato per Angular. Consente di suddividere la pagina in stati; per esempio potreste avere uno stato per la barra laterale e un altro per il contenuto principale. Possono avere entrambi un partial; inoltre permette di sviluppare in modo più semplice grandi pagine o web app.

Esiste perfino un modulo per Bootstrap simile ad AngularStrap, che vale la pena considerare.

- *URL*: `http://angular-ui.github.io/`
- *Twitter*: `@angularui`
- *Persone*: Nate Abele (`@nateabele`), Tasos Bekos (`@tbekos`), Andrew Joslin (`@andrewtjoslin`), Pawel Kozlowski (`@pkozlowski_os`), Dean Sofer (`@Unfolio`), Douglas Duteil (`@douglasduteil`) e altri.

Mobile AngularUI

A differenza del progetto AngularUI, questo ha lo scopo di sviluppare un'interfaccia utente. Si tratta di un semplice framework mobile che utilizza sia AngularJS sia Bootstrap 3. Gli elementi appaiono abbastanza nativi, anche se alcune parti come la barra di navigazione laterale potrebbero essere migliorate. È ancora ai suoi albori, ma ha un serio potenziale e vale la pena seguire la sua evoluzione.

- *URL*: `http://mobileangularui.com/`
- *Twitter*: `@mobileangularui`
- *Persone*: mcasimir

Ionic

Ionic è incredibile. Lo è davvero. Riunisce tutto ciò che è efficace in AngularJS e Cordova/Phonegap per consentirvi di sviluppare straordinarie app ibride con i linguaggi web che già conoscete.

Tutto sembra nativo ed è facilissimo iniziare anche se non avete mai sviluppato un'app. Utilizza AngularJS e l'estensione UI-Router insieme con il loro codice. L'aspetto migliore è che è interamente open source e chiunque può contribuirvi su GitHub.

È stato anche realizzato il modulo `ngCordova`, ricco di direttive che è possibile sfruttare per interfacciarsi facilmente con numerosi plug-in Cordova.

- *URL*: `http://ionicframework.com/`
- *Twitter*: `@ionicframework`
- *Persone*: The Drifty Team (`http://drifty.com/`): Andrew Joslin (`@ajoslin`) e altri.

AngularGM

Anche se AngularUI include una direttiva per utilizzare le Google Maps all’interno di Angular, preferiamo l’approccio più semplice di AngularGM. Questo modulo consente la creazione semplice delle Google Maps nel progetto, insieme con marker, InfoWindows e polyline.

Potete personalizzare quasi tutto ciò che desiderate, dalla modifica dei colori e dalle impostazioni della mappa all’utilizzo di un elemento personalizzato per l’InfoWindow o l’icona non standard per il marker.

- *URL*: `https://github.com/dylanfprice/angular-gm`
- *Persone*: Dylan Price

Ora tocca a voi…

Il numero di progetti open source riguardanti Bootstrap e AngularJS è incredibile. Tutti possono contribuirvi, perfino voi! Se trovate un bug, segnalatelo, o se sapete come risolverlo, inviate una *pull request* e diventate un *contributor*.

Naturalmente non tutti i problemi sono stati risolti. Ora che sapete come utilizzare entrambi i framework, tocca a voi divertirvi a sviluppare qualcosa di straordinario.

In caso di dubbio

Anche gli sviluppatori più esperti non sanno talvolta che pesci pigliare e non c’è nulla di male nel chiedere una mano.

Esistono alcuni strumenti specifici riguardanti Bootstrap e AngularJS che possono aiutarvi, nel caso vi servisse.

La documentazione ufficiale

Se vi imbattete in un problema o avete bisogno di rinfrescare la memoria consultate innanzitutto la documentazione ufficiale.

Sia Bootstrap sia AngularJS vantano un’ottima documentazione. In passato erano sorte delle lamentele su quella di AngularJS, ritenuta confusa e con pochi esempi, ma negli ultimi anni è migliorata notevolmente. Per ulteriori dettagli, visitate i siti `http://www.angularjs.org` e `http://www.getbootstrap.com`.

L’issue tracker di GitHub

Sia Angular sia Bootstrap sono ospitati su GitHub ed entrambi si avvalgono dell’*issue tracker* del servizio. Se scoprite un bug in uno dei due framework, segnalatelo qui.

Ovviamente se conoscete la natura del problema e sapete come risolverlo, potete inviare una *pull request* e contribuire al progetto. Per ulteriori dettagli, visitate `http://www.github.com/angular/angular.js/issues` e `http://www.github.com/twbs/bootstrap/issues`.

Stack Overflow

Forse pensavate che ne avremmo parlato. Stack Overflow è una risorsa eccezionale e un ottimo strumento da utilizzare se avete una domanda specifica. Il più delle volte troverete qualcun altro che ha posto lo stesso quesito e potrete leggere le risposte. Altrimenti ponete una nuova domanda e taggatela come “AngularJS” o “Twitter Bootstrap”. Per ulteriori dettagli visitate `http://www.stackoverflow.com`.

Il gruppo Google AngularJS

Man mano che utilizzate AngularJS, vi possiamo garantire che dopo alcune ricerche su Google, vi ritroverete in questo gruppo. Si tratta del gruppo/forum ufficiale di AngularJS ed è molto attivo.

Contiene più di 11.000 argomenti e vale la pena effettuare qui una ricerca prima di porre una nuova domanda. Per ulteriori dettagli visitate

`https://groups.google.com/forum/#!forum/angular`.

Egghead.io

Se cercate dei tutorial video su AngularJS, Egghead.io è probabilmente la risorsa migliore. Offre un servizio di abbonamento a pagamento, ma si può consultare gratuitamente buona parte della sua libreria. Se volete disporre di altro materiale video informativo, visitate `https://egghead.io/tags/free`.

Twitter

Può sembrare una segnalazione bizzarra come risorsa di supporto, ma su Twitter si trovano persone molto utili. Forse non è il luogo migliore per porre domande complesse, ma per piccole dritte può essere prezioso.

Qui si incontrano appassionati di entrambi i framework ed è possibile partecipare alle rispettive community. I due framework hanno account Twitter ufficiali: `@angularjs` e `@twbootstrap`.

A proposito, se volete mandarmi un tweet, io sono `@steve228uk`.

Sicuramente, conoscendo più a fondo AngularJS e Bootstrap, farete riferimento alla documentazione e chiederete sempre meno aiuto. Abbiamo imparato che è molto importante trasmettere ciò che si conosce.

Forse chiederete una mano per sviluppare una direttiva molto specifica, ma ciò non significa che non saprete aiutare anche voi gli altri. Quando avete un po' di tempo, accedete a Stack Overflow e provate a rispondere ad alcune domande; scommetto che riuscirete a rispondere a molte più domande di quelle che avreste mai immaginato!

Risposte ai quiz

Capitolo 1

1. Utilizzando l’attributo `ng-app`.
2. La sintassi che comprende doppie parentesi graffe: `{{model}}`.
3. *Model View Controller*.
4. Creiamo un controller utilizzando un costruttore JS standard e l’attributo `ng-controller`.
5. Jumbotron.

Capitolo 2

1. Una barra di navigazione (`navbar`) di Bootstrap.
2. 12 colonne.
3. È una funzione chiamata da un attributo o un elemento personalizzato.
4. `ng-repeat`.

Capitolo 3

1. Con il simbolo pipe in un modello: `{{ modelName | filter }}`.
2. Con i due punti: `{{ modelName | filter:arg1:arg2 }}`.
3. Il filtro chiamato `filter`.
4. Utilizziamo il servizio `$filter` inserendo il filtro come servizio seguendo il pattern `filternameFilter`.
5. Un modulo AngularJS.

Capitolo 4

1. `ngRoute`.
2. Il metodo `config` nel modulo.
3. Il servizio `$routeProvider`.
4. Con il metodo `$routeProvider.when`.
5. Il metodo `$routeProvider.otherwise`.
6. Utilizzando html5Mode.

Capitolo 5

1. Perché è incluso nella vista della radice.
2. Diverse classi: `table-bordered`, `table-striped`, `table-hover` e `table-condensed`.
3. Con `<button class=”btn btn-primary btn-lg”><button>`.
4. Nella classe `form-group`.
5. Le etichette vengono allineate alla sinistra degli elementi.
6. La classe `help-block`.
7. `img-circle` per creare un’immagine dalla forma circolare, `img-rounded` per crearne una dai bordi arrotondati e `img-thumbnail` per aggiungere un bordo doppio.

Capitolo 6

1. Servizio personalizzato, `$rootScope`, controller a livello dell’applicazione.
2. `value`, `service` e `factory`.
3. Il modulo `ngSanitize`.
4. Il metodo `controller` consente alla direttiva di comunicare con le altre direttive, mentre il metodo `link` no.
5. Il segno `=` significa che è possibile legare direttamente un modello; `@` indica che la direttiva utilizzerà il valore letterale dell’attributo.
6. Impostando la proprieta `restrict` su EM.
7. Per aggiungere alcune funzioni helper alla barra di navigazione.
8. Utilizzando `$index`.

Capitolo 7

1. Da `ngAnimate`.
2. AngularMotion.
3. `bs-`.
4. `click`, `hover`, `focus` e manualmente.
5. `show`, `hide` e `toggle`.

Capitolo 8

1. Una promise.
2. Con `$http.get('http://localhost:8000').success(function(data) { $scope.contacts = data });`.
3. Si comporta come un segnaposto.
4. RESTAngular utilizza le promise e non è necessario trascrivere i segnaposto seguendo il pattern REST.
5. In tempo reale.

Capitolo 9

1. Su Node.
2. Per indicare a NPM di quali pacchetti abbiamo bisogno.
3. `Uglify`.
4. Utilizzare la notazione degli array.

Capitolo 10

1. Variabili, mixin e regole annidate.
2. Utilizzando il pattern `&:before` oppure `&:after`.
3. Modificando la variabile in `variables.less`.
4. Usa gli stili di Bootstrap per renderlo simile a Bootstrap 2.

Capitolo 11

1. Una qualsiasi delle seguenti: `required`, `ng-required`, `ng-pattern`, `ng-minlength`, `ng-maxlength`, `ng-min` e `ng-max`.
2. Verificando le proprietà `$valid` o `$invalid`.
3. Chiamandolo nella proprietà `require`.
4. Utilizzando `ng-pattern`.

Capitolo 12

1. Un’estensione di Chrome che ci permette di verificare le app di AngularJS.
2. Uno dei seguenti: *Models*, *Performance*, *Dependencies*, *Inspector*, evidenziazione delle applicazioni, binding e scope.
3. Nella scheda *Models* del web inspector selezionando *AngularJS Properties*.
4. ngMin.
5. `remove`, `add` e `singleQuotes`.
6. La necessità di annotare manualmente le dipendenze.