1 Знакомство с Angular

В этой главе:

- краткий обзор фреймворков и библиотек JavaScript;
- общий обзор Angular 1 и 2;
- набор инструментов для Angular-разработчика;
- пример приложения.

Angular 2 — фреймворк с открытым исходным кодом, написанный на JavaScript и поддерживаемый компанией Google. Он представляет собой полностью переработанную версию своего популярного предшественника, AngularJS. С помощью Angular вы можете разрабатывать приложения на JavaScript (применяя синтаксис ECMAScript 5 или 6), Dart или TypeScript. В книге мы будем использовать TypeScript. Причины, по которым был выбран именно этот синтаксис, описаны в приложении Б.

ПРИМЕЧАНИЕ

Мы не ждем от вас опыта работы с AngularJS, но рассчитываем, что вы знакомы с синтаксисом JavaScript и HTML, а также понимаете, из чего состоит веб-приложение. Кроме того, предполагается, что вы имеете представление о CSS и знакомы с ролью объекта DOM в браузере.

Мы начнем эту главу с очень краткого обзора нескольких популярных фреймворков JavaScript. Далее рассмотрим архитектуру AngularJS и Angular 2, выделяя улучшения, привнесенные в новую версию. Кроме того, кратко рассмотрим инструменты, используемые Angular-разработчиками. Наконец, мы взглянем на приложение-пример, которое будем создавать на протяжении книги.

ПРИМЕЧАНИЕ

Наша книга посвящена фреймворку Angular 2, и для краткости мы будем называть его Angular. Если мы упомянем AngularJS, то это значит, что речь идет о версиях 1.x данного фреймворка.

1.1. Примеры фреймворков и библиотек JavaScript

Обязательно ли использовать фреймворки? Нет, можно написать клиентскую часть веб-приложений на чистом JavaScript. В этом случае не нужно изучать что-то новое, достаточно знания языка JavaScript. Отказ от фреймворков приведет к возникновению трудностей при поддержке совместимости между браузерами, а также к увеличению циклов разработки. Фреймворки же позволяют полностью управлять архитектурой, шаблонами проектирования и стилями кода вашего приложения. Большая часть современных веб-приложений написаны при сочетании нескольких фреймворков и библиотек.

В этом разделе мы кратко рассмотрим популярные фреймворки и библиотеки для работы с JavaScript. В чем заключается разница между ними? *Фреймворки* позволяют структурировать ваш код и заставляют писать его определенным способом. *Библиотеки* обычно предлагают несколько компонентов и API, которые могут быть использованы по желанию в любом коде. Другими словами, фреймворки предоставляют большую гибкость при разработке приложения.

Angular — один из многих фреймворков, применяемых для разработки веб-приложений.

1.1.1. Полноценные фреймворки

Содержат все, что понадобится для разработки веб-приложения. Они предоставляют возможность легко структурировать ваш код и поставляются вместе с библиотекой, содержащей компоненты и инструменты для сборки и развертывания приложения.

Например, *Ext JS* — полноценный фреймворк, созданный и поддерживаемый компанией Sencha. Он поставляется вместе с полным набором UI-компонентов, включающим продвинутые сетки данных и таблицы (их наличие критически важно для разработки промышленных приложений для офисов). Ext JS значительно увеличивает объем кода программы — вам не удастся найти созданное с его помощью приложение, которое весит меньше 1 Мбайт. Кроме того, данный фреймворк довольно глубоко внедряется — будет трудно при необходимости переключиться на другой инструмент.

Sencha также предлагает фреймворк Sencha Touch, используемый при создании веб-приложений для мобильных устройств.

1.1.2. Легковесные фреймворки

Позволяют структурировать веб-приложение, предлагают способ настройки навигации между представлениями, а также разбивают приложения на слои, реализуя шаблон проектирования «Модель — Представление — Контроллер» (Model — View — Controller, MVC). Кроме того, существует группа легковесных фреймворков, которые специализируются на тестировании приложений, написанных на JavaScript.

Angular — фреймворк с открытым исходным кодом, предназначенный для разработки веб-приложений. Упрощает создание пользовательских компонентов, которые могут быть добавлены в документы HTML, а также реализацию логики приложения. Активно использует привязку данных, содержит модуль внедрения зависимостей, поддерживает модульность и предоставляет механизм для настройки маршрутизации. AngularJS был основан на шаблоне MVC, в отличие от Angular. Последний не содержит элементов для создания пользовательского интерфейса.

Ember.js — это фреймворк с открытым исходным кодом, основанный на MVC; служит для разработки веб-приложений. Содержит механизм маршрутизации и поддерживает двухстороннюю привязку данных. В коде этого фреймворка используется множество соглашений, что повышает продуктивность разработчиков ПО.

Jasmine — фреймворк с открытым исходным кодом, предназначенный для тестирования кода JavaScript. Не требует наличия объекта DOM. Содержит набор функций, проверяющих, ведут ли себя части приложения запланированным образом. Нередко используется вместе с Karma — программой для запуска тестов, которая позволяет проводить проверки в разных браузерах.

1.1.3. Библиотеки

Библиотеки, рассмотренные в этом подразделе, служат для разных целей и могут быть задействованы в веб-приложениях вместе с другими фреймворками или самостоятельно.

jQuery — популярная библиотека для JavaScript. Довольно проста в использовании и не требует значительного изменения стиля написания кода для веб-программ. Позволяет находить объекты DOM и манипулировать ими, а также обрабатывать события браузера и справляться с несовместимостью браузеров. Это расширяемая библиотека, разработчики со всего мира создали для нее тысячи плагинов. Если вы не можете найти плагин, который отвечает вашим нуждам, то всегда можете создать его самостоятельно.

Bootstrap — библиотека компонентов для создания пользовательского интерфейса с открытым исходным кодом, разработанная компанией Twitter. Они строятся согласно принципам адаптивного веб-дизайна, что значительно повышает ценность библиотеки, если ваше веб-приложение должно автоматически подстраивать свой макет в зависимости от размера экрана устройства пользователя. В этой книге мы будем использовать Bootstrap при разработке приложения-примера.

ПРИМЕЧАНИЕ

> В компании Google была разработана библиотека компонентов под названием Material Design, которая может стать альтернативой Bootstrap. Она оптимизирована для использования на разных устройствах и поставляется вместе с набором интересных элементов пользовательского интерфейса.

React — созданная компанией Facebook библиотека с открытым исходным кодом, предназначенная для сборки пользовательских интерфейсов. Представляет

собой слой V в аббревиатуре MVC. Не внедряется глубоко, и ее можно применять вместе с любой другой библиотекой или фреймворком. Создает собственный виртуальный объект DOM, минимизируя доступ к объекту DOM браузера, в результате чего повышается производительность. Что касается отрисовки содержимого, React вводит формат JSX — расширение синтаксиса JavaScript, которое выглядит как XML. Использование JSX рекомендуется, но не обязательно.

Polymer — библиотека, созданная компанией Google для сборки пользовательских компонентов на основе стандарта Web Components. Поставляется вместе с набором интересных настраиваемых элементов пользовательского интерфейса, которые можно включить в разметку HTML в виде тегов. Кроме того, содержит компоненты приложений, предназначенных для работы в режиме офлайн, а также элементы, использующие разнообразные API от Google (например, календарь, карты и др.).

RxJS — набор библиотек, необходимых для создания асинхронных программ и программ, основанных на событиях, с использованием наблюдаемых коллекций. Позволяет приложениям работать с асинхронными потоками данных наподобие серверного потока котировок акций или событий, связанных с движением мыши. С помощью RxJS потоки данных представляются в виде наблюдаемых последовательностей. Эту библиотеку можно применять как с другими фреймворками JavaScript, так и без них. В главах 5 и 8 вы увидите примеры использования наблюдаемых последовательностей в Angular.

Чтобы увидеть статистику, которая показывает, на каких сайтах задействованы те или иные фреймворки, вы можете посетить страницу BuiltWith: http://trends.builtwith.com/javascript.

Переход от Flex к Angular

Мы работаем на компанию Farata Systems, которая за много лет разработала много довольно сложного ПО, используя фреймворк Flex от Adobe. Это очень продуктивный фреймворк, созданный на основе строго типизированного скомпилированного языка ActionScript. Приложения, написанные с его помощью, развертываются в плагине для браузера Flash Player (применяется как виртуальная машина). Когда веб-сообщество начало отходить от плагинов, мы потратили два года, пытаясь найти замену Flex. Мы экспериментировали с разными фреймворками, основанными на JavaScript, но эффективность труда наших разработчиков серьезно падала. Наконец, мы увидели свет в конце туннеля, когда объединили язык TypeScript, фреймворк Angular и библиотеку для работы с пользовательским интерфейсом, такую как Angular Material.

1.1.4. Что такое Node.js

Node.js (или просто *Node*) — не просто фреймворк или библиотека. Это еще и среда времени выполнения. На протяжении большей части книги мы будем использовать время выполнения Node для запуска различных утилит наподобие Node Package Manager (npm). Например, для установки TypeScript можно запустить npm из командной строки:

```
npm install typescript
```

Фреймворк Node.js используется для разработки программ на языке JavaScript, которые функционируют вне браузера. Вы можете создать серверный слой веб-приложения на JavaScript или Typescript; в главе 8 вы напишете веб-сервер, применяя Node. Компания Google разработала для браузера Chrome высокопроизводительный движок JavaScript V8. Его можно задействовать для запуска кода, написанного с помощью API Node.js. Фреймворк Node.js включает в себя API для работы с файловой системой, доступа к базе данных, прослушивания запросов HTTP и др.

Члены сообщества JavaScript создали множество утилит, которые будут полезны при разработке веб-приложений. Используя движок JavaScript для Node, вы можете запускать их из командной строки.

1.2. Общий обзор AngularJS

Теперь вернемся к основной теме нашей книги: фреймворку Angular. Этот раздел — единственный, посвященный AngularJS.

Перечислим причины, по которым AngularJS стал таким популярным.

- Фреймворк содержит механизм создания пользовательских тегов и атрибутов HTML с помощью концепции директив, которая позволяет расширять набор тегов HTML в соответствии с потребностями приложения.
- AngularJS не слишком глубоко внедряется. Вы можете добавить атрибут `ng-app` к любому тегу `<div>`, и AngularJS будет управлять содержимым лишь этого тега. Остальная часть веб-страницы может быть написана на чистом HTML и JavaScript.
- Фреймворк позволяет легко связывать данные с представлениями. Изменение данных приводит к автоматическому обновлению соответствующего элемента представления, и наоборот.
- AngularJS поставляется с настраиваемым маршрутизатором; он разрешает соотносить шаблоны URL с соответствующими компонентами приложения, которые изменяют представление на веб-странице в зависимости от установленных соотношений.
- Поток данных приложения определяется в *контроллерах*, они являются объектами JavaScript, содержащими свойства и функции.
- Приложения, созданные с помощью AngularJS, используют иерархию *областей видимости* — объектов, необходимых для сохранения данных, общих для контроллеров и представлений.
- Фреймворк содержит модуль внедрения зависимостей, который позволяет разрабатывать слабо связанные приложения.

В то время как в jQuery были упрощены манипуляции с объектом DOM, AngularJS позволяет разработчикам отвязывать логику приложения от интерфейса путем структурирования приложения по шаблону проектирования MVC. На рис. 1.1 показан пример рабочего потока приложения, созданного с помощью этого фреймворка.

Рис. 1.1. Пример архитектуры приложения, написанного с использованием AngularJS

AngularJS может управлять всем веб-приложением. Для этого включите директиву `ng-app` в HTML-тег `<body>`:

```
<body ng-app="ProductApp">
```

На рис. 1.1, чтобы получить данные о продукте, пользователь загружает приложение ❶ и вводит идентификатор продукта ❷. Представление оповещает контроллер ❸, который обновляет модель ❹ и выполняет HTTP-запрос ❺ на удаленный сервер, используя сервис `$http`. AngularJS заполняет свойства модели на основе полученных данных ❺, и изменения в модели автоматически отразятся в пользовательском интерфейсе с помощью *связывающего выражения* ❻. После этого пользователь увидит данные о запрошенном продукте ❼.

AngularJS автоматически обновляет представление при модификации данных модели. Изменения в пользовательском интерфейсе вносятся в модель, если пользователь меняет данные в элементах управления представления, связанных с вводом. Такой двунаправленный механизм обновлений называется *двухсторонней привязкой*, он показан на рис. 1.2.

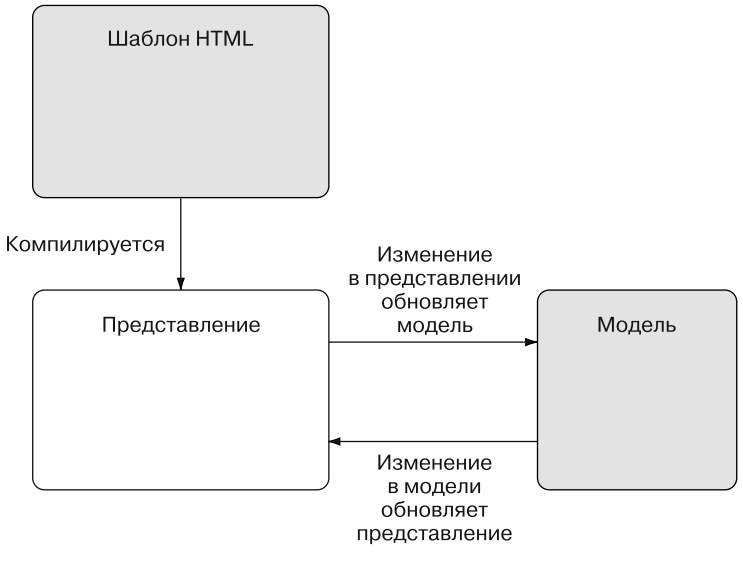

Рис. 1.2. Двухсторонняя привязка

В AngularJS модель и представление тесно связаны, поскольку двухсторонняя привязка означает, что один из элементов пары автоматически обновляет другой. Подобная возможность автоматического обновления очень удобна, но имеет свою цену.

При каждом обновлении модели фреймворк запускает специальный цикл `$digest`, который проходит по всему приложению, применяет привязку данных

и обновляет DOM, когда это необходимо. Каскадные обновления приводят к нескольким запускам цикла `$digest`, что может повлиять на производительность крупных приложений, использующих двухстороннюю привязку.

Манипуляции с объектом DOM браузера — самая медленная операция. Чем меньше приложение обновляет DOM, тем лучше оно работает.

Данные модели существуют в контексте определенного объекта `$scope`, области видимости AngularJS формируют иерархию объектов. Элемент `$rootScope` создается для целого приложения. Контроллеры и директивы (пользовательские компоненты) имеют собственные объекты `$scope`, и принципы работы областей видимости в фреймворке могут быть сложны для понимания.

Вы можете реализовать модульность, создавая и загружая объекты `module`. Когда конкретный модуль зависит от других объектов (таких как контроллеры, модули или сервисы), экземпляры этих объектов создаются и *внедряются* путем использования механизма внедрения зависимостей, представленного в AngularJS. В следующем фрагменте кода показывается один из способов, которым фреймворк внедряет один объект в другой:

```
var SearchController = function ($scope) {
        //..
};

SearchController['$inject'] = ['$scope'];

angular.module('auction').controller('SearchController', SearchController);
```

Определяет SearchController как функцию-конструктор, имеющую параметр $scope

Добавляет свойство $inject в контроллер, давая команду внедрить объект $scope в функцию-конструктор

Указывает, что объект SearchController должен стать контроллером модуля аукциона

В этом фрагменте кода квадратными скобками обозначен массив, и AngularJS может внедрять сразу несколько объектов следующим образом: `['$scope', 'myCustomService']`.

Данный фреймворк зачастую используется для создания одностраничных приложений, в которых лишь отдельные фрагменты страницы (подпредставления) обновляются в результате действий пользователя или из-за отправки данных на сервер. Хорошим примером таких подпредставлений является веб-приложение, показывающее котировки: при продаже акции обновляется лишь элемент, содержащий значение цены.

Навигация между представлениями в AngularJS выполняется с помощью конфигурирования компонента маршрутизатора `ng-route`. Можно указать количество параметров `.when`, чтобы направить приложение к соответствующему представлению в зависимости от шаблона URL. В следующем фрагменте кода

маршрутизатору предписывается использовать разметку из файла `home.html` и контроллер `HomeController`, если только URL не содержит элемент `/search`. В этом случае представление отрисует страницу `search.html`, и в качестве контроллера будет применен объект `SearchController`:

```
angular.module('auction', ['ngRoute'])
    .config(['$routeProvider', function ($routeProvider) {
        $routeProvider
          .when('/', {
             templateUrl: 'views/home.html',
             controller: 'HomeController' })
          .when('/search', {
              templateUrl: 'views/search.html',
              controller: 'SearchController' })
          .otherwise({
              redirectTo: '/'
    });
}]);
```

Маршрутизатор фреймворка поддерживает глубокое связывание, которое представляет собой способность поместить в закладки не только целую веб-страницу, но и определенное состояние внутри нее.

Теперь, после общего обзора AngularJS, взглянем, что нам может предложить Angular 2.

1.3. Общий обзор Angular

Фреймворк Angular гораздо производительнее, чем AngularJS. Он более понятен для освоения; архитектура приложений была упрощена, и код с его помощью легче читать и писать.

В этом разделе приводится краткий обзор Angular, в котором выделяются его более сильные стороны относительно AngularJS. Подробный архитектурный обзор Angular доступен в документации продукта на https://angular.io/guide/architecture.

1.3.1. Упрощение кода

Во-первых, Angular-приложение содержит стандартные модули в форматах ECMAScript 6 (ES6), Asynchronous Module Definition (AMD) и CommonJS. Обычно один модуль находится в одном файле. Нет необходимости прибегать к синтаксису, характерному для фреймворков, для загрузки и использования модулей. Запустите универсальный загрузчик модулей SystemJS (он рассматривается в главе 2) и добавьте операторы импорта, чтобы применить функциональные возможности, реализованные в загруженных модулях. Вам не нужно волноваться

насчет правильного использования тегов `<script>` в ваших файлах HTML. Если для модуля A требуются функции, содержащиеся в модуле B, просто импортируйте модуль B в модуль A.

Файл HTML для посадочной страницы вашего приложения содержит модули Angular и их зависимости. Код приложения загружается путем загрузки корневого модуля приложения. Все необходимые компоненты и сервисы будут загружены на основании объявлений в операторах `module` и `import`.

В следующем фрагменте кода показано типичное содержимое файла `index.html` приложения, созданного с помощью Angular, куда вы можете включить требуемые модули фреймворка. В сценарий `systemjs.config.js` входит конфигурация загрузчика SystemJS. Вызов `System.import('app')` загружает высокоуровневый элемент приложения, сконфигурированный в файле `systemjs.config.js` (показан в главе 2). Пользовательский тег `<app>` представляет собой значение, определенное в свойстве `selector` корневого компонента:

```
<!DOCTYPE html>
<html>
<head>
  <title>Angular seed project</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">

  <script src="node_modules/core-js/client/shim.min.js"></script>
  <script src="node_modules/zone.js/dist/zone.js"></script>
  <script src="node_modules/typescript/lib/typescript.js"></script>
  <script src="node_modules/systemjs/dist/system.src.js"></script>
  <script src="node_modules/rxjs/bundles/Rx.js"></script>
  <script src="systemjs.config.js"></script>
  <script>
    System.import('app').catch(function(err){ console.error(err); });
  </script>
</head>

<body>
<app>Loading...</app>
</body>
</html>
```

HTML-фрагмент в каждом приложении содержится либо внутри компонента (свойство `template`), либо в файле, на который ссылается этот компонент с помощью свойства `templateURL`. Второй вариант позволяет дизайнерам работать над интерфейсом вашего приложения, не изучая Angular.

Компонент Angular является центральным элементом новой архитектуры. На рис. 1.3 показана общая схема примера Angular-приложения, состоящего из четырех компонентов и двух сервисов; все они находятся внутри модуля внедрения зависимостей (Dependency Injection, DI). Этот модуль внедряет сервис `Http` в сервис `Service1`, а тот, в свою очередь, внедряется в компонент `GrandChild2`. Эта

схема отличается от приведенной рис. 1.1, где был показан принцип работы AngularJS.

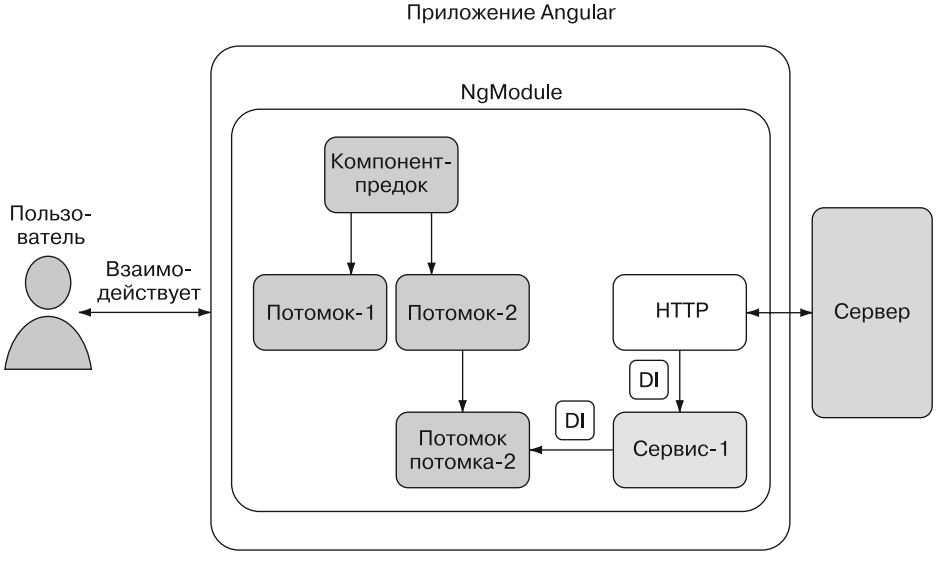

Рис. 1.3. Пример архитектуры Angular-приложения

Самый простой способ объявления компонента заключается в написании класса с помощью TypeScript (можно использовать ES5, ES6 или Dart). В приложении Б мы кратко расскажем о том, как писать компоненты Angular на TypeScript, а также приведем пример кода. Попробуйте понять этот код, прочитав минимальное количество пояснений.

Класс TypeScript содержит аннотацию метаданных `@NgModule` и представляет собой модуль. Класс TypeScript включает аннотацию метаданных `@Component` и представляет собой компонент. Аннотация `@Component` (также известная как декоратор) имеет свойство `template`, указывающее на фрагмент HTML, который должен быть отрисован в браузере.

Аннотации метаданных дают возможность изменять свойства компонентов во время разработки. Шаблон HTML может включать в себя выражения для привязки данных, окруженных двойными фигурными скобками. Ссылки на обработчики событий помещены в свойство `template` аннотации `@Component` и реализованы как методы класса. Еще одним примером аннотации метаданных выступает аннотация `@Injectable`, позволяющая отметить элемент, с которым должен работать модуль DI.

Аннотация `@Component` также содержит селектор, объявляющий имя пользовательского тега, который будет использован в документе HTML. Когда Angular видит элемент HTML, чье имя соответствует селектору, он знает, какой элемент

его реализует. В следующем фрагменте HTML показан родительский компонент `<auction-application>` с одним потомком, `<search-product>`:

```
<body>
  <auction-application>
    <search-product [productID]= "123"></search-product>
  </auction-application>
</body>
```

Предок отправляет данные потомкам путем привязки к входным свойствам потомка (обратите внимание на квадратные скобки в предыдущем фрагменте кода), а потомок общается с предками, отправляя события с помощью своих выходных свойств.

В конце главы вы найдете рис. 1.7, на котором показана главная страница (компонент-предок), чьи элементы-потомки окружены толстыми границами.

В следующем фрагменте кода показывается класс `SearchComponent`. Вы можете включить его в документ HTML как `<search-product>`, поскольку его объявление содержит свойство `selector` с таким же именем:

```
@Component({
  selector: 'search-product',
  template:
    `<form>
      <div>
        <input id="prodToFind" #prod>
        <button (click)="findProduct(prod.value)">Find Product</button>
        Product name: {{product.name}}
      </div>
    `</form>

})
  class SearchComponent {
    @Input() productID: number;

    product: Product; // Опустим код класса Product

    findProduct(prodName: string){
    // Здесь будет расположен реализация обработчика щелчков кнопкой мыши
  }
  // Здесь будет расположен другой код
}
```

Если вы знакомы с любым объектно-ориентированным языком, имеющим классы, то должны понять большую часть предыдущего фрагмента кода. Аннотированный класс `SearchComponent` объявляет переменную `product`, которая может представлять объект с несколькими свойствами, и одно из них (`name`) привязано к представлению (`{{product.name}}`). Локальная переменная шаблона `#prod` будет иметь ссылку на элемент `<input>`, поэтому вам не нужно запрашивать DOM для того, чтобы получить введенное значение.