1 Введение в Vue.js

В этой главе

- Шаблоны проектирования MVC и MVVM.
- Определение реактивного приложения.
- Описание жизненного цикла Vue.
- Оценка архитектуры Vue.js.

Мы уже давно привыкли к интерактивным сайтам. В середине 2000-х годов, на заре Web 2.0, интерактивность и вовлечение пользователей в процесс были в центре внимания. Именно в этот период появились компании Twitter, Facebook и YouTube. Благодаря бурному росту социальных сетей и пользовательского контента Интернет менялся в лучшую сторону.

Чтобы поспевать за этими изменениями и предоставлять бóльшую интерактивность, разработчики начали создавать библиотеки и фреймворки, которые упрощали построение интерактивных сайтов. В 2006 году Джон Резиг (John Resig) выпустил jQuery, тем самым значительно облегчив написание клиентских скриптов внутри HTML. Со временем появились и другие подобные проекты. Сначала они были большими, монолитными и навязывали разработчикам свое видение. Теперь же произошел сдвиг в сторону компактных облегченных библиотек, которые очень просто добавить в любое приложение. Вот мы и подошли к Vue.js.

Vue.js — это библиотека, позволяющая внедрять интерактивное поведение и дополнительные возможности в любой контекст, в котором выполняется JavaScript. Vue можно использовать как на отдельных страницах, решая простые задачи, так и в качестве фундамента для полноценных промышленных приложений.

ПРИМЕЧАНИЕ

В Интернете названия Vue и Vue.js практически взаимозаменяемы. В этой книге я чаще использую более простой вариант, Vue, оставляя Vue.js для тех случаев, когда речь идет непосредственно о коде или библиотеке.

Вы увидите, как Vue и вспомогательные библиотеки позволяют создавать полноценные сложные веб-приложения. Мы исследуем все, от интерфейса, с которым взаимодействуют посетители, до базы данных, снабжающей наш код информацией.

Увидим также, каким образом каждый из приводимых здесь примеров вписывается в общую картину, рассмотрим лучшие методики, которые предлагает индустрия веб-разработки, и научимся интегрировать представленный далее код в собственные проекты, как существующие, так и новые.

Эта книга рассчитана в основном на веб-разработчиков, у которых уже есть некоторый опыт работы с JavaScript и общее представление об HTML и CSS. Тем не менее благодаря универсальности своего *программного интерфейса* (application programming interface, API) Vue подходит для разработчиков всех мастей и проектов любой степени сложности. Она станет надежным подспорьем, даже если вам нужно создать лишь небольшой прототип или простую любительскую программу для личного пользования.

1.1. На плечах у гигантов

Прежде чем начинать писать код для нашего первого приложения или углубляться в недра Vue, следует обратиться к истории программного обеспечения. Лишь зная о проблемах и вызовах, с которыми сталкивался мир веб-разработки в прошлом, можно по достоинству оценить преимущества, которыми обладает эта библиотека.

1.1.1. Шаблон проектирования MVC

Клиентский шаблон проектирования *«модель — представление — контроллер»* (Model — View — Controller, MVC) лежит в основе множества современных фреймворков для разработки веб-приложений (если вы уже знакомы с MVC, можете пролистывать дальше).

Здесь следует упомянуть, что со временем концепция MVC претерпела некоторые изменения. В так называемой классической версии MVC был предусмотрен отдельный набор правил для того, каким образом должны взаимодействовать между собой представление, контроллер и модель. Мы же остановимся на упрощенной интерпретации этого клиентского шаблона проектирования, которая лучше подходит для современных реалий.

MVC применяется для разделения ответственности в приложении (рис. 1.1). Представление отвечает за вывод информации пользователю — это *графический пользовательский интерфейс* (graphical user interface, GUI). В центре находится контроллер. Он помогает передавать события из представления в модель, а данные — из

модели в представление. Внизу мы видим модель, которая содержит бизнес-логику и может предоставлять некое хранилище данных.

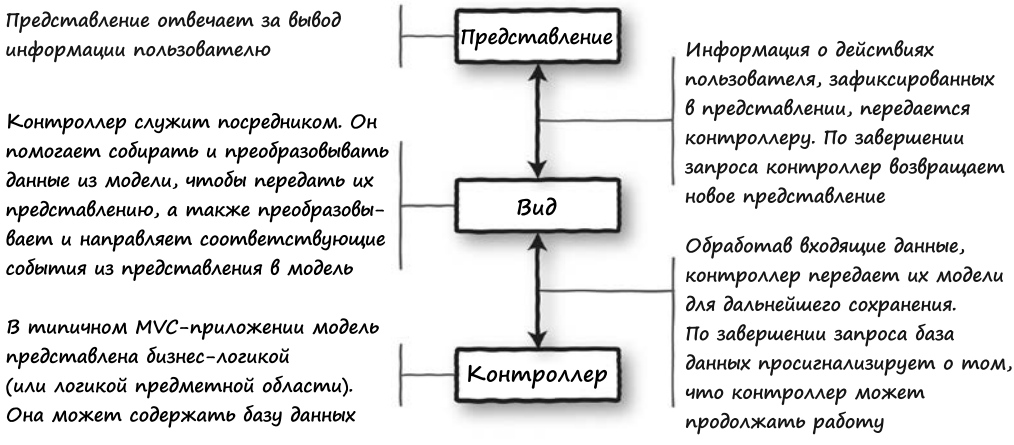

Рис. 1.1. Взаимоотношения модели, представления и контроллера согласно MVC

ДОПОЛНИТЕЛЬНО

Если хотите узнать больше о шаблоне проектирования MVC, можете начать со статьи Мартина Фаулера (Martin Fowler), посвященной эволюции MVC: martinfowler.com/eaaDev/uiArchs.html.

Авторы веб-фреймворков любят этот шаблон проектирования за его надежную, проверенную временем архитектуру. Если вас интересует устройство современных веб-фреймворков, советую почитать книгу Эммитта А. Скотта-младшего (Emmitt A. Scott Jr.) *SPA Design and Architecture* (Manning, 2015).

В современной разработке MVC часто задействуется в рамках одного приложения, предоставляя отличный механизм для разделения программного кода на разные роли. В сайтах, основанных на MVC, каждый запрос инициирует передачу потока информации от клиента к серверу, от сервера к базе данных, а затем обратно. Этот процесс отнимает много времени и ресурсов, что сказывается на отзывчивости пользовательского интерфейса.

Со временем приложения, основанные на веб-технологиях, стали более интерактивными. Теперь они применяют асинхронные запросы и клиентскую архитектуру MVC, чтобы передача данных на сервер не блокировала выполнение и не требовала ожидания ответа. Веб-страницы все больше напоминают настольные программы, поэтому задержки, связанные с клиент-серверным взаимодействием, могут замедлить или даже нарушить работу приложения. И вот тут на выручку приходит следующий шаблон.

Несколько слов о бизнес-логике

Клиентский шаблон проектирования MVC отличается большой гибкостью при выборе места реализации бизнес-логики. На рис. 1.1 логика сконцентрирована в модели. Но это было сделано лишь для простоты, можно было выбрать для этого другие уровни приложения, включая контроллер. С тех пор как в 1979 году шаблон MVC был представлен Трюгве Ринскаугом (Trygve Reenskaug) для языка Smalltalk-76, он претерпел некоторые изменения.

Представьте процесс проверки почтового индекса, введенного пользователем.

- Представление может содержать код на JavaScript, который проверяет почтовый индекс по мере ввода или перед отправкой.
- Модель способна проверить почтовый индекс при создании объекта для хранения входящих данных.
- База данных может накладывать ограничения на поле почтового индекса, то есть модель тоже применяет бизнес-логику. Некоторые разработчики считают это плохим тоном.

Иногда бывает сложно определить, что именно представляет собой бизнес-логика. Мы отдельно остановимся на том, как и где она должна реализовываться. Вы также узнаете, каким образом Vue и сопутствующие библиотеки помогают изолировать функциональность в строго заданных рамках.

1.1.2. Шаблон проектирования MVVM

Когда JavaScript-фреймворки начали поддерживать асинхронные методы программирования, исчезла необходимость в обновлении всей веб-страницы при каждом запросе. Частичное обновление представления позволяет сайтам и приложениям быстрее реагировать, но в определенной степени приводит к дублированию кода. Логика представления часто повторяет бизнес-логику.

Шаблон MVVM (Model — View — ViewModel — «модель — представление — модель представления») — это следующий этап в развитии MVC. Его отличительными чертами являются *модель представления* (ViewModel, VM) и связывание данных. MVVM обеспечивает основу для построения клиентских приложений, которые делают взаимодействие с пользователем и обратную связь более отзывчивыми, избегая при этом затратного дублирования кода в рамках всей архитектуры. Этот шаблон упрощает также модульное тестирование. Но имейте в виду, что для простых пользовательских интерфейсов он может оказаться избыточным.

MVVM позволяет создавать веб-приложения, которые мгновенно реагируют на действия пользователя и позволяют свободно переходить от одной задачи к другой. Модель представления тоже способна играть несколько ролей (рис. 1.2). Такая консолидация ответственности имеет одно фундаментальное последствие для представлений приложения: при изменении данных внутри VM автоматически обновляются все представления, связанные с этой моделью. Механизм связывания

делает данные доступными и гарантирует, что любые их изменения будут отражены в представлении.

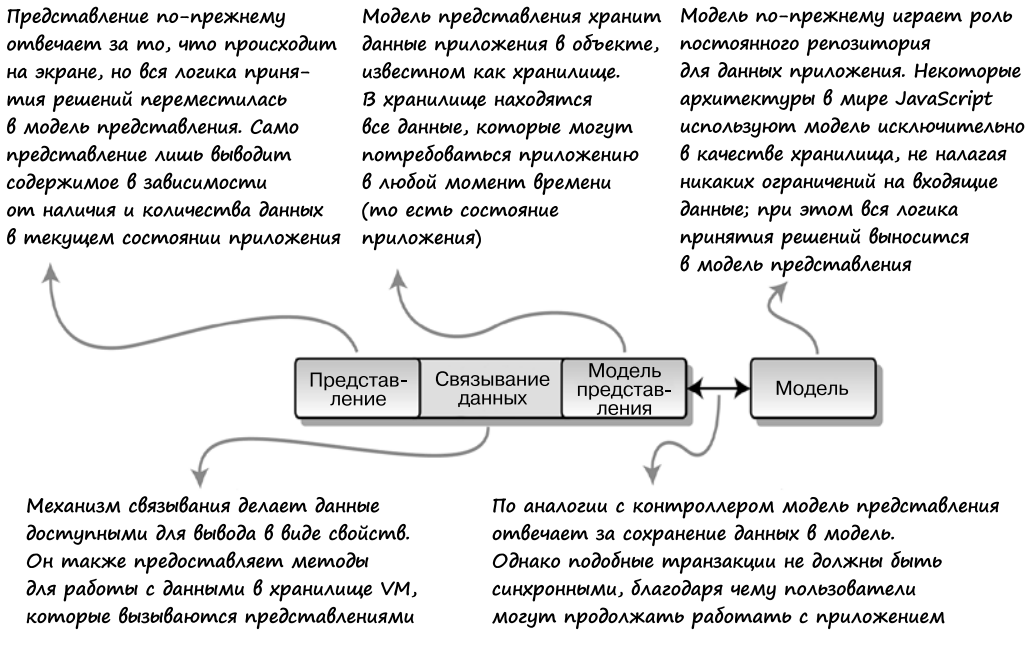

Рис. 1.2. Элементы шаблона проектирования MVVM

ДОПОЛНИТЕЛЬНО

Больше информации об MVVM можно найти в статье Мартина Фаулера (Martin Fowler), посвященной шаблону проектирования Presentation Model: martinfowler.com/eaaDev/PresentationModel.html.

1.1.3. Что такое реактивное приложение

Сама по себе парадигма реактивного программирования не нова. Однако в веб-приложениях ее начали использовать относительно недавно, в основном благодаря наличию фреймворков Vue, React и Angular.

В Интернете много информации о теории реактивности, но нас интересует более узкая тема. Чтобы веб-приложение можно было назвать реактивным, оно должно выполнять следующие функции:

- ❑ отслеживать изменения в своем состоянии;
- ❑ распространять уведомления об изменениях для всех своих компонентов;
- ❑ автоматически отрисовывать представления в ответ на изменения состояния;
- ❑ своевременно реагировать на действия пользователя.

Для достижения этих целей реактивные веб-приложения по возможности применяют принципы проектирования MVVM, асинхронные методики, позволяющие избежать блокирования взаимодействия с пользователем, и элементы функционального программирования.

Шаблон проектирования MVVM и реактивное программирование не имеют прямого отношения друг к другу, но у них общая направленность: обе концепции стремятся сделать приложение более отзывчивым и надежным с точки зрения пользователя.

ДОПОЛНИТЕЛЬНО

Если хотите узнать больше о реактивной парадигме программирования Vue, почитайте руководство «Подробно о реактивности» по адресу ru.vuejs.org/v2/guide/reactivity.html.

1.1.4. Калькулятор на JavaScript

Чтобы разобраться в понятиях реактивности и связывания данных, напишем калькулятор на чистом JavaScript (листинг 1.1).

Листинг 1.1. Калькулятор на JavaScript: chapter-01/calculator.html

```
<!DOCTYPE>
<html>
  <head>
    <title>A JavaScript Calculator</title>
    <style>
     p, input { font-family: monospace; }
     p, { white-space: pre; }
    </style>
  </head>
  <body>
    <div id="myCalc">
      <p>x <input class="calc-x-input" value="0"></p>
      <p>y <input class="calc-y-input" value="0"></p>
      <p>--------------------</p>
      <p>= <span class="calc-result"></span></p>
    </div>
    <script type="text/javascript">
     (function(){

       function Calc(xInput, yInput, output) {
         this.xInput = xInput;
         this.yInput = yInput;
         this.output = output;
       }

       Calc.xName = 'xInput';
```

Поля для ввода значений x и y, привязанных к функции runCalc

Вывод введенных значений x и y

Конструктор для создания экземпляра Calc

```
Calc.yName = 'yInput';

Calc.prototype = {
  render: function (result) {
    this.output.innerText = String(result);
  }
};

function CalcValue(calc, x, y) {
  this.calc = calc;
  this.x = x;
  this.y = y;
  this.result = x + y;
}

CalcValue.prototype = {
  copyWith: function(name, value) {
    var number = parseFloat(value);

    if (isNaN(number) || !isFinite(number))
      return this;

    if (name === Calc.xName)
      return new CalcValue(this.calc, number, this.y);

    if (name === Calc.yName)
      return new CalcValue(this.calc, this.x, number);

    return this;
  },
  render: function() {
    this.calc.render(this.result);
  }
};

function initCalc(elem) {

  var calc =
    new Calc(
      elem.querySelector('input.calc-x-input'),
      elem.querySelector('input.calc-y-input'),
      elem.querySelector('span.calc-result')
    );
  var lastValues =
    new CalcValue(
      calc,
      parseFloat(calc.xInput.value),
      parseFloat(calc.yInput.value)
    );

  var handleCalcEvent =
```

Конструктор значений для экземпляра Calc

Инициализация компонента Calc

Обработчик событий

```
          function handleCalcEvent(e) {
            var newValues = lastValues,
                elem = e.target;

            switch(elem) {
              case calc.xInput:
                newValues =
                  lastValues.copyWith(
                    Calc.xName,
                    elem.value
                  );
                break;
              case calc.yInput:
                newValues =
                  lastValues.copyWith(
                    Calc.yName,
                    elem.value
                  );
                break;
            }

            if(newValues !== lastValues){
              lastValues = newValues;
              lastValues.render();
            }
          };

        elem.addEventListener('keyup', handleCalcEvent, false);

        return lastValues;
      }

      window.addEventListener(
        'load',
        function() {
          var cv = initCalc(document.getElementById('myCalc'));
          cv.render();
        },
        false
      );

    }());
    </script>
  </body>
</html>
```

Регистрация обработчика для события keyup

Это калькулятор на языке ES5 JavaScript (более современной версией JavaScript, ES6/2015, воспользуемся позже). Мы задействуем выражение IIFE (немедленно вызываемая функция), которое запускает наш скрипт. Конструктор отвечает за хранение значений, а обработчик `handleCalcEvent` реагирует на событие `keyup`.

1.1.5. Калькулятор на Vue

Пока что не стоит обращать внимания на синтаксис Vue в этом примере. Мы не стремимся понять каждый фрагмент кода, а лишь пытаемся сравнить две реализации. Хотя, если вы хорошо понимаете пример с JavaScript, вам не составит большого труда разобраться со следующим кодом (листинг 1.2), по крайней мере на теоретическом уровне.

Листинг 1.2. Калькулятор на Vue: chapter-01/calculatorvue.html

```
<!DOCTYPE html>
<html>
<head>
  <title>A Vue.js Calculator</title>
  <style>
    p, input { font-family: monospace; }
    p { white-space: pre; }
  </style>
</head>
<body>
  <div id="app">
    <p>x <input v-model="x"></p>
    <p>y <input v-model="y"></p>
    <p>---------------------</p>
    <p>= <span v-text="result"></span></p>
  </div>

  <script src="https://unpkg.com/vue/dist/vue.js"></script>
  <script type="text/javascript">
  function isNotNumericValue(value) {
    return isNaN(value) || !isFinite(value);
  }
    var calc = new Vue({
      el: '#app',
      data: { x: 0, y: 0, lastResult: 0 },
      computed: {
        result: function() {
          let x = parseFloat(this.x);
          if(isNotNumericValue(x))
          return this.lastResult;

          let y = parseFloat(this.y);
          if(isNotNumericValue(y))
          return this.lastResult;

          this.lastResult = x + y;

          return this.lastResult;
        }
      }
    });
  </script>
</body>
</html>
```

Привязка к DOM для нашей программы

Поля ввода для нашей программы

Результаты будут выводиться в этом элементе span

Элемент script, который подключает библиотеку Vue.js

Инициализация приложения

Подключение к DOM

Переменные, объявленные в приложении

Вычисления выполняются с помощью вычисляемого свойства