第一篇

概念篇——扎根于基础

第1章 引　　言

本章主要介绍前端技术的发展历史和从 MVC 架构进化到 MVVM 的历程。笔者意图通过对这些内容的描述，使读者对工作和学习中遇到的一些代码的标准和年代特征形成一些基本判断和认识，并充分了解 Vue 这样的 MVVM 框架对高效率和高质量项目开发所起到的作用。

1.1 前端技术的发展

纵观整个前端发展史，我们可以发现，几个关键的时间节点均与重大的技术飞跃息息相关，如 Ajax 的诞生、Node 的问世等。笔者将结合这几个点，和大家一起回顾前端开发的历史发展轨迹，展望其未来发展前景。

1.1.1 从静态走向动态

最初的网页是欧洲粒子物理研究所的科学家为了方便查看共享文档和论文，而基于 XML（Extensible Markup Language，可扩展标记语言）创造的，这也是在前端开发中，最重要的全局对象被称为 document 而不是 context、page、application 等的原因。当时，网页只具备文本图片的显示及页面间相互跳转（Hyper Link）的功能，因此人们称其语言为 HTML（Hyper Text Markup Language，超文本标记语言）。

最初的 Web，功能十分单一，开发也并不复杂。开发者先把写好的网页放在服务器指定位置（网站根目录）下，然后将映射 URL（Uniform Resource Locator，统一资源定位器）分享给使用者，使用者在浏览器地址栏输入 URL 即可访问网页内容，如图 1.1 所示。

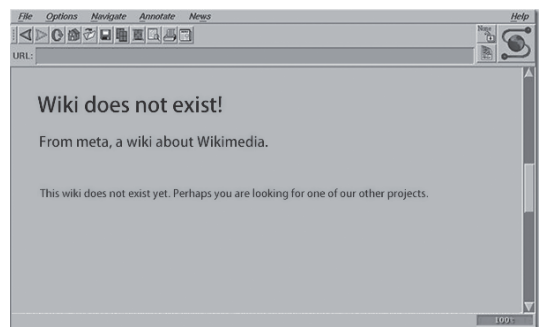

图 1.1　早期的浏览器和网页

早期的 HTML 文件作为静态文件，即使只有部分内容是需要变动的，也是有多少种变动的可能性，开发者就需要准备多少份文档。这对开发者来说是非常不友好的，并且开发者无法与用户进行交互。

CGI（Common Gateway Interface，通用网关接口）的出现改善了这一情况。CGI 作为服务器拓展功能，可以从数据库或者文件系统获取数据，在将数据渲染为 HTML 文档后，返回客户端，从而实现网页的动态生成。在接收到用户请求后，CGI 还可以在服务端进行处理，并返回对应的处理结果，如图 1.2 所示。

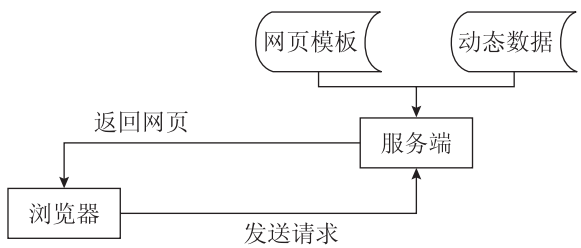

图 1.2　动态网页渲染流程

CGI 被广泛认为是服务端脚本语言的鼻祖。然而，它也有非常致命的缺陷。首先，CGI 每接收到一个请求，都会新开一个进程进行处理，占用服务器的 CPU 和内存，当请求量成千上万时，服务器可能无法支撑以致崩溃。其次，黑客很容易通过不完善的 CGI 程序非法进入开发者的服务器系统，这从安全方面来考虑是绝对不允许的。

以后来人的角度来看，笔者认为 CGI 出现的最大意义就是给当时刚起步的 Web 提供了一个发展方向。在这之后，PHP、JSP、ASP 等各种服务端语言层出不穷，不仅弥补了 CGI 的缺陷，而且在性能上愈加高效，在开发上愈加简捷。这些语言的出现和广泛应用，使得 Web 技术飞速发展，前端网页从此从静态走向动态，这个时代被称为 Web 1.0 时代。

1.1.2　从后端走向前端

在 Web 1.0 时代，前后端是如何协作的呢？因为网页是在服务端使用动态脚本语言和模板引擎渲染出来的，所以前端一般先写模板，写好后交付给后端套用，之后前后端联调，以确认模板套用无误。

在这种开发环境下，前后端耦合密切，项目开发需要很高的沟通成本。在模板引擎的变量、判断和循环、宏区块等语法糖的支持下，前端也可以利用环境变量来实现部分

业务逻辑。如果前端表现得稍微弱势一些，其就很有可能迫于后台压力在视图层实现一些业务代码。同时，整个项目的代码质量也随之降低。

网站的这种组织架构还会带来另外一些问题。例如，页面哪怕仅有一小块内容需要变更，浏览器也需要重新请求和渲染整个页面。一方面，网站资源的传输耗费了更多的时间；另一方面，页面重载的用户体验也十分糟糕。

举一个例子，用户在登录页面输入了错误密码时，服务器要将校验信息渲染到页面并传给浏览器。实际上，页面只是多了一行类似于“密码错误”的提示，然而网站资源需要重新进行传输，同时页面还会丢失用户输入的表单数据（即便到了今天，这种现象依然可以在一些政府和国企的老旧网站中看到）。

当时虽然出现了各种页面和数据的缓存技术，稍有成效地缓解了这一问题，但也无法从根本上解决问题。于是，从事 Web 开发的前辈们开始探寻其他一些解决方案，如 Ajax（Asynchronous JavaScript And XML，异步 JavaScript 和 XML）异步数据加载。

Ajax 通过 XMLHttpRequest 对象，可以在不重载页面的情况下与 Web 服务器交换数据，再加上 JavaScript 的 document 对象，开发者可以很轻松地实现页面局部内容刷新。

从 1999 年开始，ActiveX 和 XMLHttpRequest 陆续问世，Ajax 的星星之火渐渐燃起。时间推移到 2005 年，Google 公司发布了全面使用 Ajax 打造的 Gmai1（如图 1.3 所示）和 Gmap 两款应用。人们惊讶地发现，原来使用异步数据传输获得的应用体验是如此良好。自此，Ajax 获得了井喷式发展。

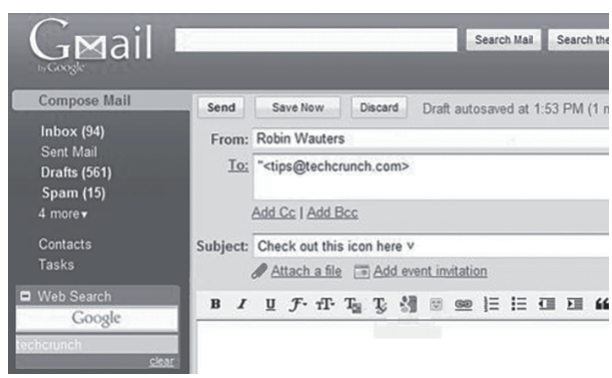

图 1.3　Gmail 使用界面

得益于 Ajax 的发展，前后端分离的趋势日渐明显，前端不再需要依赖后台环境生存，所有服务器数据都可以通过异步交互来获取。在取得一个良好定义的 RESTful

（Representational State Transfer，表述性状态转移）接口后，两端甚至可以在零沟通成本的情况下并行完成项目任务。

随着 Google V8 引擎问世，PC 和移动端设备性能的提高，ES6 和 H5 的日趋成熟，浏览器端的计算能力和功能性似乎愈加过剩，开发者开始将越来越多的业务逻辑代码迁移到前端，前端路由的概念也逐渐清晰。

路由这个概念首先出现在后台。要实现传统 Web 网页间的跳转，开发者需要先在后台设置页面的路由规则，之后服务器根据用户的请求检索路由规则列表，并返回相应的页面。前端路由则是在浏览器端配置路由规则，通过侦听浏览器地址的变化，异步加载和更新页面内容。

可以这么说，Ajax 实现了无刷新的数据交互，而前端路由实现了无刷新的页面跳转；Ajax 将 Web Page 发展成 Web App，而前端路由则给了 Web App 更多的可能，如 SPA（Single Page Application，单页面应用），如图 1.4 所示。

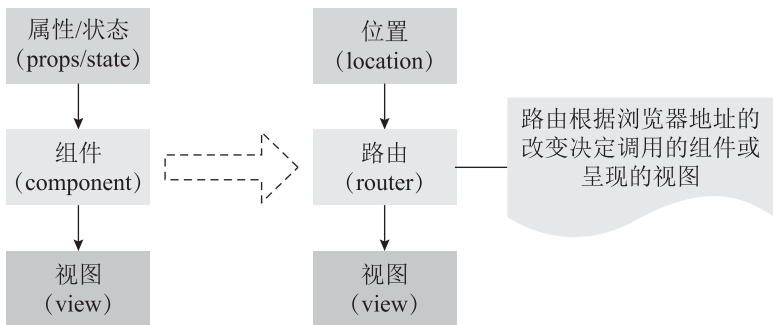

图 1.4　单页面应用 SPA

Angular、React、Vue 等知名的前端框架都有前端路由的概念。在之后的章节中，笔者会专门讲解前端路由的实现原理和 Vue.js 项目的核心内容之一——Vue Router。

现在，很多 Web 项目采用这样的架构，后台只负责数据的存取和组装，而前端则负责业务逻辑层和视图层的全部工作。这一路走来，项目重心已从后端转移到了前端。

1.1.3　从前端走向全端

下面是笔者在 2018 年春节时，在 CSDN（国内的技术交流社区）的官网上截取的一张图，如图 1.5 所示。读之深有体会，有兴趣的同学可以细细品味，这里不再多作赘述。

图 1.5　2018 年 CSDN 春联

若要说 2009 年 Web 界最为爆炸性的新闻，那一定是 Node.js 的问世。

2009 年 2 月，一个名叫 Ryan Dahl 的开发者在博客上宣布准备基于 Chrome V8 引擎创建一个轻量级的 Web 服务器并提供一套组件库。

同年 5 月，Ryan Dahl 在 GitHub 上发布了最初版本的 Node.js，这标志着 Node.js 的诞生。从此，JavaScript 也占据了服务端编程语言的一席之地。前端工程师可以以很低的成本用 Node.js 和 MongoDB 搭建一个后台。乍一看，前端工程师和全栈工程师之间的距离，只在于一个 DataBase（数据库）。

从 Node.js 诞生至今，无论是新手还是专家，大批量地涌入 Node 社区，大家围绕着项目，使用并贡献着自己的力量，努力使之适用于更多的应用场景。这些年来，人们对 Node.js 褒贬不一，但毋庸置疑的是，它的问世必是前端发展史上浓墨重彩的一笔，如图 1.6 所示。

图 1.6　Node.js 主页

近年，随着微信小程序和支付宝小程序的问世，前端技术早已超脱了 Web 和 Hybrid 应用的范围。前端工程师很容易基于固有技术栈快速上手和开发小程序类微应用。以微信小程序为例，框架使用语法通用的 WXML 代替 HTML、WXSS 代替 CSS，开发语言由 HTML+CSS+JS 变为 WXML+WXSS+JS。此外，与 Vue.js 一样，它们也是基于 MVVM（Model-View-ViewModel，模型－视图－视图模型）的，如图 1.7 所示。

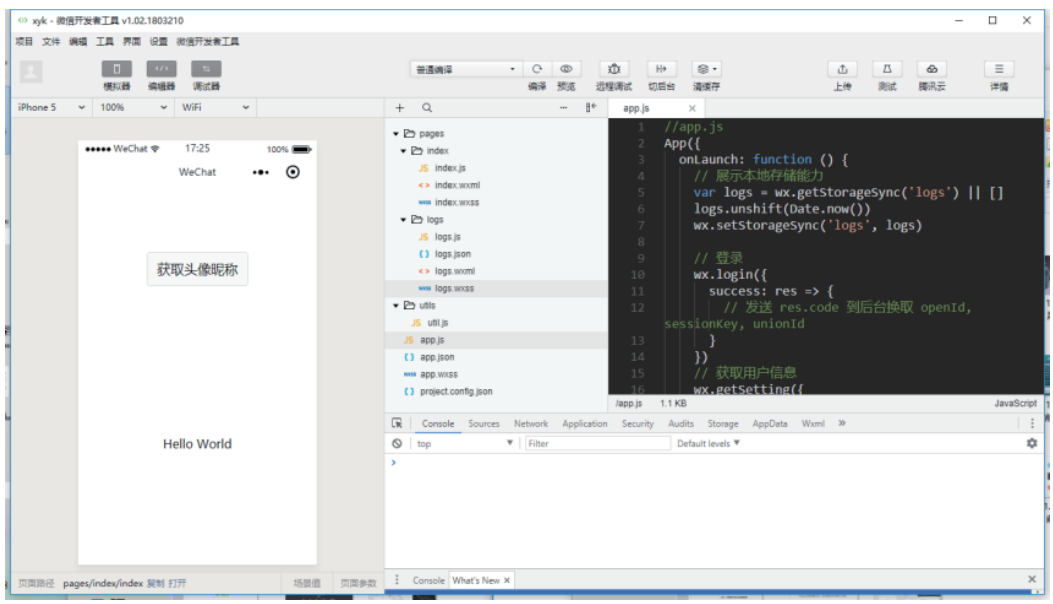

图 1.7　微信小程序开发

2018 年 3 月 20 日，IT 技术界又爆出一热点新闻：多家公司联合共建快应用的标准和平台。

快应用类似于小程序，用户不用下载和安装即可使用和生成桌面快捷方式，但二者的区别是，快应用不必依赖微信或者支付宝这样的第三方平台，它是手机厂商从系统应用层面支持的。对于前端工程师来说，这又是一则喜讯，因为开发快应用使用的也是前端技术栈。

可以预见，未来的前端和前端衍生技术很有可能遍布从 Web 到桌面应用，从 PC、移动端到智能电视、游戏机等各个角落。

未来的工程师也许只分为两种：一种是负责数据方面的云端工程师；另一种则是全端（前端）工程师。

1.2 MVVM 族员——Vue.js

MVVM 本质上是 MVC（Model-View-Controller，模型－视图－控制器）的改进版，其最重要的特性是数据绑定（data binding），此外还包括依赖注入、路由配置、数据模板等特性。

1.2.1 从 MVC 到 MVVM

MVC 模式在 Web 1.0 时代曾被广泛应用于 Web 架构中，然而其诞生的时间比 Web 早几年。最初，MVC 被应用于桌面程序中，在 PHP、JSP 等脚本语言诞生之后，逐渐成为 Web 开发的主流模式。

View（视图层）是用户能够看到并进行交互的客户端界面，如桌面应用的图形界面、浏览器端渲染的网页等；Model（模型）指业务模型，用于计算、校验、处理和提供数据，但不直接与用户产生交互；Controller（控制器）则负责收集用户输入的数据，向相关模型请求数据并返回相应的视图来完成交互请求，如图 1.8 所示。

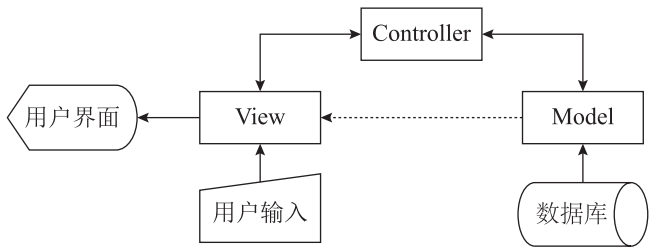

图 1.8 MVC 模式

MVC 模式实现了 Model 和 View 的代码分离，Model 专注于数据，View 专注于表达，Controller 则在 Model 和 View 之间架起了一座桥梁。即使采用同一个 Model 的数据，如果调用不同的 View（如柱状图和表格），也会得到不同的页面呈现。这样的设计，不仅减少了 Model 层的冗余代码，使得 Model 和 View 更加灵活和易于维护，同时简化了项目的架构和管理。

随着技术日新月异的更迭，MVC 渐渐演化出更多的形态。虽然这些模式都有特定的名称，然而实际上它们都是 MVC 的衍生版本。因此，有的开发者也会将其统一称作

MV* 模式，MVVM 即其中的一种。

与 MVC 模式一样，MVVM 的主要目的是分离 View（视图）和 Model（模型），ViewModel 层封装了界面展示和操作的属性和接口。通过数据绑定，我们可以将 View 和 ViewModel 关联在一起。当 ViewModel 中的数据发生变化时，View 也会同步进行更新，如图 1.9 所示。

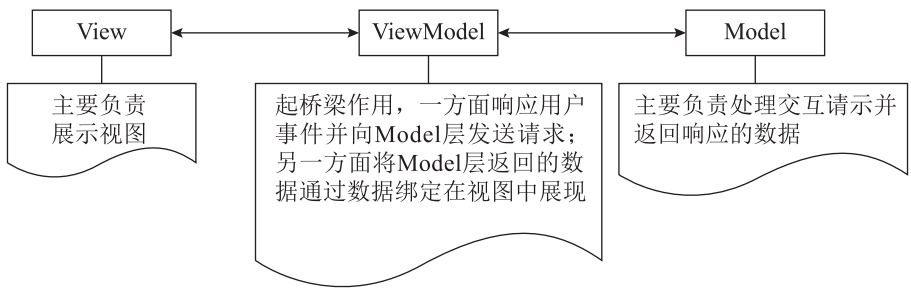

图 1.9　MVVM 模式

MVVM 模式解耦了视图和模型。在模式中，每一个视图都有对应的一个 ViewModel，同时 ViewModel 与模型建立联系。当接收到用户请求后，ViewModel 获取模型响应的数据，并通过数据绑定将相应的视图页面重新渲染。模型层的数据只需要传入 ViewModel 即可实现视图的同步更新，从而实现了视图和模型之间的松散耦合。

与 MVC 不同的是，MVC 是系统架构级别的，而 MVVM 是用于单页面上的。因此，MVVM 的灵活性要远大于 MVC。如果将这里的 Model 抛开，只看 VVM，这就是一个组件（如 treeview）的设计模式。因此，MVVM 模式也是组件化开发的最佳实践。

1.2.2　Vue.js 简介

Vue.js 是一套轻量级 MVVM 框架，由时任 Google 工程师的尤雨溪（现担任阿里巴巴 Weex 团队技术顾问）创作并开源。截至本书编写时，Vue.js 已在 GitHub 获得 star 数 18.7 万个，而同为 MVVM 框架且更早诞生的 React 获得的 star 数不过 17.3 万个，Angular 则是 5.9 万个，如图 1.10 所示。

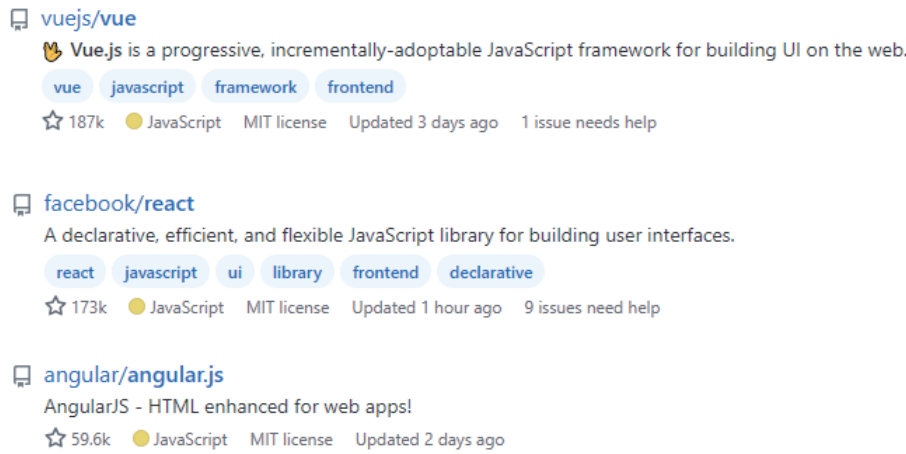

图 1.10　MVVM 框架单指标影响力对比

与其他重量级框架不同的是，Vue 的核心库只关注 View 层，并且提供尽可能简单的 API 以实现数据绑定、组件复用等机制，且非常容易学习并嵌入其他库。同时，Vue 也完全有能力支持采用 SPA 设计和组合其他 Vue 生态库的系统。

1.3　Vue 与 React

在 MVVM 框架一族中，Vue.js 的表现十分优秀。在 1.3 节和 1.4 节中，我们将分别看到 Vue 和 React，以及 Vue 和 Angular 的对比表现。

Vue 和 React 都是轻量级框架，不过总体来看，Vue 的性能是高于 React 的，笔者简单罗列了以下几点。

1.3.1　虚拟 DOM

在处理用户界面时，DOM 操作成本是最高的，Vue 与 React 都在渲染流程中采用虚拟 DOM 以降低页面开销，如图 1.11 所示。不过，Vue 的虚拟 DOM 实现的层级更高一些，这也意味着 Vue 比 React 更轻量、性能更高。

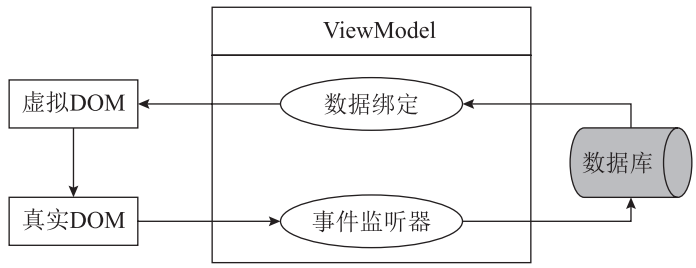

图 1.11　渲染流程

1.3.2　功能性组件

Vue 和 React 都提供一些功能性组件以减少用户开销。笔者运行 GitHub 上的一个测试项目（https：//github.com/chrisvfritz/vue-render-performance-comparisons），该项目将渲染 10 000 个列表条目 100 次，得到的测试结果如表 1.1 所示。

表 1.1　测 试 结 果

次　　数	渲染时间（ms）	
	Vue	React
第一次	22	63
第二次	22	64
第三次	23	62
第四次	22	63
第五次	22	63

React 和 Vue 的速度都很快，不过显然 Vue 的渲染速度更快，这是因为 React 有大量用于提供警告和错误提示信息的检查机制。

1.3.3　轻量级——将与核心库无关的业务封装成独立库

React 和 Vue 都将着重点放在核心库上，也都有专门负责路由和全局状态管理等功能的配套库。例如，与 React 配套的库有 React Router、Redux，与 Vue 配套的库有 Vue Router、Vuex。

1.3.4 视图模板

React 采用 JSX 渲染组件，而 Vue 采用模板，如 .vue 扩展名的文件。

JSX 是使用 XML 语法编写 JavaScript 的一种语法糖。语法如下：

```
class HelloMessage extends React.Component { render() {
        return (
          <div>
               Hello {this.props.name}
          </div>
    );
  }}

ReactDOM.render(
  <HelloMessage name="Taylor" />,
  mountNode
);
```

通过 JSX，我们可以只用 JavaScript 来构建视图组件。不过，对于从传统 HTML+CSS+JS 分离开发走向组件化开发的前端工程师来说，这种语法并不友好。

Vue 提供了更简单的模板。语法如下：

```
<template>
<div class="demo-title">{{title}}</div>
</template>

<script>
  export default {
    data () {
      return {
        title: 'Hello World'
      }
    }
  }
</script>

<style scoped>
.demo-title {
    font-size: 24px;
    font-weight: 600;
}
</style>
```

Vue 模板更贴合 HTML，而不是用更高层的东西去封装它，学习曲线十分平缓。在 Vue 模板的 style 标签上标注 scoped 属性可划分作用域，使 CSS 样式表只作用于当前组

件（具体实现机制将在后续章节中描述）。

由于 Vue 模板更贴近原生，因此，我们很容易混入其他一些东西，如 HTML 的预处理器（Pug/Jade 等）、CSS 的预处理器（LESS、SASS/SCSS 等），以及更高版本（高级）的脚本语言（TypeScript、ES 6 JavaScript 等）。Vue 模板的语法也更符合传统开发习惯，并易于团队分析和代码维护。

1.3.5　其他

除框架本身外，Vue 在其他方面也具有一些优势，如 Vue 的状态管理库 vuex 和路由库 vue-router 都由官方维护更新，从而保证了这些库与 Vue 本身的统一性。React 的相关库则由社区进行维护，不过，这也使得 React 的社区生态更加繁荣一些。此外，Vue 提供了项目快速构建工具——vue-cli 脚手架，提供了包含 npm 依赖管理、webpack 模块打包、vue-router 前端路由、eslint 语法检测、单元测试等集成功能，能够让开发者快速构建一个高质量的项目环境。

1.4　Vue 与 Angular

在代码体积和性能方面，Vue 都比 Angular 1、Angular 2 表现得优异许多，这里不再赘述。笔者选择以下几个方面来对比分析 Vue 和 Angular 的表现。

1.4.1　模板语法

Vue 的许多语法与 Angular 十分相似，可以认为 Angular 是 Vue 的灵感之源。因为尤雨溪当时在 Google 创意实验室使用的就是 Google 主推的 Angular 框架。然而，随着使用程度不断加深，其尤感觉 Angular 十分笨重，因此才创造了 Vue。在 Vue 的诞生过程中，有很多地方都借鉴了 Angular 的语法习惯。

Angular 2 语法：

```
<input type="text" [(ngModel)]="name"/>
<button (click)="onSave($event)">Save</button>
<ul>
   <li *ngFor="letheroofheroes" [title]="hero.name" (click)="delete
(hero)">{{hero.name}}</li>
</ul>
<form #heroForm (ngSubmit)="submit()"></form>
```

Vue 语法：

```
<input type="text" v-model="name"/>
<button v-on:click="onSave($event)">Save</button>
<ul>
  <li v-for="heroinheroes" v-bind:title="hero.name" v-on:click="delete
(hero)">{{hero.name}}</li>
</ul>
<form ref="heroForm" v-on:submit="submit()"></form>
```

1.4.2 脏检测

Vue 与 Angular 1 最大的区别在于 Vue 没有脏检测机制。Angular 1 中存在多个 watcher，当 watcher 越来越多时，检测耗时会越来越长。因为作用域每发生一次变化，所有 watcher 都要重新计算，而一些 watcher 在计算之后可能又会导致新的变化，并引发所有 watcher 重新计算，从而进入一种无限循环的脏检测。

Angular 1 的处理方式是设置循环上限，如 10 次，当循环达到 10 次时，即中止循环。显然，这种脏检测机制性能十分低下、耗时长，并不适合大型 Web 应用。

Vue 的处理方式则是全局只设置一个 watcher，用这一个 watcher 来记录和更新一组关联对象的值，从而回避了脏检测的问题。

有意思的是，Vue 最初是参考 Angular 的，而 Angular 2 则借鉴了 Vue 的机制，采用相似的设计来解决脏检测存在的问题。

1.4.3 双向数据绑定

轻量级框架与重量级框架的划分标准是，框架是否过分参与系统结构级的架构和功能上的伸缩拓展。与 Vue、React 这样的轻量级框架相比，Angular 除参与单向数据流的视图渲染、事件绑定外，还参与了 View 对 Model 层的数据更新，即双向数据绑定。显然，Angular 是一个重量级框架。

在单向数据绑定中，视图模板和动态数据被渲染成网页后，数据流即中止，如图 1.12 所示。之后，由 ViewModel 接手与 View 层的数据绑定。View 层不可以直接修改 Model 层的数据，如果需要修改 Model 层的数据，则由 ViewModel 发起请求，这中间存在 ViewModel 与 Model 之间的数据同步传输。

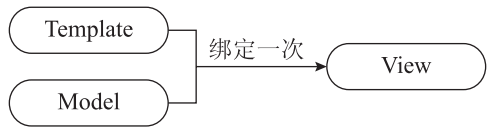

图 1.12　单向数据绑定

然而，在双向数据绑定中，Model 和 View 始终建立着联系，Model 层的数据也一直保持着真实的状态，如图 1.13 所示。

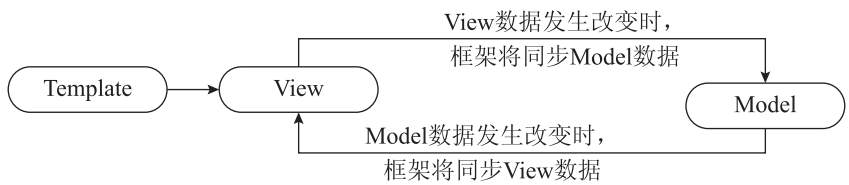

图 1.13　双向数据绑定

1.4.4　学习曲线

最后一点，广为人知且津津乐道的是，Angular 的学习曲线十分陡峭，初学者可能会有一种坐过山车的感觉。不过，笔者在 2016 年，接触过一个使用 Angular 1 进行开发的项目，当时感觉坡度是有的，但没有那么夸张，也可能是因为应用比较浅吧。

Vue 的学习曲线则较为平缓。在 Ember、Knockout、Angular、React 等前辈踏平的道路上，Vue 有更多趋于成熟的最佳实践可以拿来使用，也有更多的经验教训可供参考，从而使开发者设计出更简便的 API 来实现更复杂的功能。同时，这有效降低了团队开发成本，并使得大型 Web 项目的构建变得更加容易。