Web 开发经典丛书

TypeScript 入门与区块链项目实战

[美] 雅科夫·法因(Yakov Fain)
[俄] 安东·莫伊谢耶夫(Anton Moiseev) 著
王红滨 王 勇 何 鸣 译

清华大学出版社
北 京

北京市版权局著作权合同登记号　图字：01-2020-6252

Yakov Fain　Anton Moiseev
TypeScript Quickly
EISBN: 978-1-61729-594-2

图书在版编目(CIP)数据

TypeScript 入门与区块链项目实战 / (美) 雅科夫・法因(Yakov Fain)，(俄罗斯)安东・莫伊谢耶夫著；王红滨，王勇，何鸣译. —北京：清华大学出版社，2021.4
(Web 开发经典丛书)
书名原文：TypeScript Quickly
ISBN 978-7-302-57830-7

I. ①T…　II. ①雅…　②安…　③王…　④王…　⑤何…　III. ①JAVA 语言－程序设计　IV. ①TP312.8

中国版本图书馆 CIP 数据核字(2021)第 057266 号

责任编辑：王　军
封面设计：孔祥峰
版式设计：思创景点
责任校对：
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：
经　　销：全国新华书店
开　　本：170mm×240mm　　印　　张：　　字　　数：千字
版　　次：2020 年　月第 1 版　　印　　次：2020 年　月第 1 次印刷
定　　价：元

产品编号：

致　　谢

作者 Yakov：感谢我最好的朋友 Sammy，在我撰写该书时所营造的温馨舒适的环境。很遗憾无法与 Sammy 交谈。与其他狗一样，Sammy 对家庭成员的爱超越了对它自己的爱。

作者 Anton：要感谢 Yakov 和本书所使用的开源项目的贡献者。没有他们定期为项目投入大量的时间，以及他们通过持续不断的工作对团队的支持，本书无法付梓。还要感谢家人在我写作期间给予的忍耐和理解。

特别要感谢提供有价值反馈的多位书评人，他们是：Ahmad F Subahi、Alexandros Dallas、Brian Daley、Cameron Presley、Cameron Singe、Deniz Vehbi、Floris Bouchot、George Onofrei、George Thomas、Gerald James Stralko、Guy Langston、Jeff Smith、Justin Kahn、Kent R. Spillner、Kevin Orr、Lucas Pardue、Marko Letic、Matteo Battista、Paul Brown、Polina Keselman、Richard Tuttle、Srihari Sridharan、Tamara Forza 以及 Thomas Overby Hansen。

前　言

本书是一本有关编程语言 TypeScript 的书籍，据开发者在 Stack Overflow 上的调查，TypeScript是最受欢迎的编程语言之一。值得信赖的 *ThoughtWork's Technology Radar* 的最新一期文章指出：“TypeScript 是一种经过深思熟虑的语言，且其不断改进的工具和 IDE 给我们留下极其深刻的印象。利用一个良好的 TypeScript 类型定义库，我们在得益于丰富的 JavaScript 库的同时还能获得类型安全的保障。”

若每天使用 TypeScript，就会更加喜欢它。对 TypeScript 的喜爱源于它允许我们将关注点放在要解决的主要问题上，而不必将精力放在诸如对象属性名的输入错误等问题上。与用 JavaScript 编写的代码相比，在 TypeScript 程序中，在运行时出错的可能性更低。同时，许多 IDE 都能提供很棒的 TypeScript 支持，并且可以从我们的项目使用的第三方库中引导我们穿过 API 的“迷宫”。

虽然 TypeScript 非常出色，但它是一种最终需要编译到 JavaScript 的语言，因此我们也要讨论一些关于 JavaScript 的问题。1995 年 5 月，在经过 10 天的艰苦工作后，Brendan Eich 开发了 JavaScript 编程语言。该脚本语言不需要编译器，主要是想将其用在 Netscape Navigator Web 浏览器上。

在浏览器上部署 JavaScript 程序不需要编译器。在 JavaScript 源文件中添加<script>标记(或对源文件的引用)，就能指引浏览器加载并分析代码，然后在浏览器的 JavaScript 引擎上执行代码。开发者喜欢这种语言的简洁性——无须说明变量的类型且无须使用任何工具。开发者仅仅需要使用文本编辑器编写代码并将其应用到 Web 页面上。

在你第一次开始学习 JavaScript 时，可在两分钟内编写自己的程序并看到它运行。无须安装或配置什么，程序不需要编译，因为 JavaScript 是一种解释性语言。

JavaScript 还是一种动态类型语言，能带给软件开发者额外的自由。不需要预先声明对象的属性，如果在运行时对象的某个属性还未定义，JavaScript 引擎将在运行阶段创建属性。

实际上，在 JavaScript 中是无法声明变量类型的。JavaScript 引擎将基于赋值猜测类型(例如，var x=123 意味着 x 是 number 类型)；如果之后的脚本有一个赋值为 x="678"，x 的类型将自动从 number 变为 string。是你真的想改变 x 的类型，还是说这是个错误？你只能在运行时才能知道，因为没有编译器警告你。

JavaScript 是一种非常“宽容”的语言，如果代码库非常小的话，这没什么问题，因为此时项目中唯有你参与。大多数情况下，你会记得 x 是一个 number 类型的变量，你不需要任何帮助。当然，若你始终为当前的雇主工作，变量 x 的类型你始终不会忘记。

JavaScript 流行开来并成为 Web 开发中的标准编程语言。但是 20 年前，开发者使用 JavaScript 开发的是一些仅包含交互内容的 Web 页面，今天我们开发的复杂文本应用程序是由开发小组共同开发的包含成千上万行代码的程序。并非每个小组成员都记得 x 被定义为 number。为最小化运行错误的数量，JavaScript 开发者将编写单元测试并执行代码审查。

得益于 IDE 的自动完成、易于重构等特点，软件开发者可以进一步提高生产力。但如果编程语言允许将属性自由地添加到对象中，随意改变其类型，一个 IDE 将会如何帮助你进行优化呢？

Web 开发者需要更好的语言，但试图用一种语言替换 JavaScript，用于支持所有不同类型的浏览器是不现实的。因此，新的最终编译到 JavaScript 的语言应运而生。它们对工具更友好，当然程序在部署前仍然需要转换到 JavaScript，以便所有浏览器都能支持它。TypeScript 是这些语言中的一种，读完本书后，你会发现该语言能够脱颖而出的原因。

关 于 本 书

哪些人适合阅读本书

本书是为那些想在开发 Web 或独立应用程序时变得更有效率的软件工程师而编写的。两位作者都是从业者，他们为从业者编写了这本书。该书不仅使用基本代码示例解释语言的语法，而且还开发了多个应用程序，展示如何在较流行的库和框架中使用 TypeScript。

在编写这本书时，作者针对书中的代码示例举办了研讨会，为他们提供了关于本书内容的早期反馈。真心希望你能享受通过这本书学习 TypeScript 的过程。

希望你对 HTML、CSS 以及使用最新 ECMAScript 规范的 JavaScript 有所了解。如果你只熟悉 ECMAScript5 语法，先去浏览一下附录将会帮助你更容易地解书中的代码示例——附录提供了最新的 JavaScript 相关知识的介绍。

本书的组织方式：学习路线

本书分为两部分。在第 I 部分，利用一些较小的代码块讲解 TypeScript 的各种语法元素。在第 II 部分，将 TypeScript 实际应用于多个版本的区块链 App 开发中。如果你的目标是快速学习 TypeScript 的语法和工具，那么第 I 部分就能足以满足你的需求。

第 1 章从 TypeScript 开发开始。你将编译并运行非常基本的程序，从用 TypeScript 编写程序到将其编译为可运行的 JavaScript 代码，以便了解 TypeScript 的工作流程。还将介绍 TypeScript 与 JavaScript 编程的优点，并介绍 Visual Studio Code 编辑器。

第 2 章讲解如何使用类型声明变量和函数。你将学习如何使用 type 关键字声明类型别名，以及如何使用类和接口声明自定义类型。这将有助于你理解标称类型系统和结构类型系统之间的区别。

第 3 章讲解类继承工作原理以及何时使用抽象类。你将学习 TypeScript 中如何使用接口强制类具有已知签名的方法，而不必关注方法的具体实现细节。还介绍“接口编程”的含义。

第 4 章专门讨论枚举和泛型。本章介绍使用枚举的好处、数字型和字符型枚举的语法、泛型类型的用途以及如何编写支持泛型的类、接口和函数。

第 5 章介绍装饰器、映射类型和条件类型。这是高级 TypeScript 类型的相关知识，你应该

熟悉泛型的语法以便理解本章的内容。

第 6 章涉及各种工具。将介绍源映射和 TSLinter 的使用(尽管 TSLinter 已被淘汰，但许多开发人员仍在使用它)。然后将展示如何使用 Webpack 编译和绑定 TypeScript 应用程序。还将学习如何使用以及为什么使用 Babel 编译 TypeScript。

第 7 章介绍如何在 TypeScript 应用程序中使用 JavaScript 库。首先解释类型定义文件的作用，然后介绍一个在 TypeScript 应用程序中使用 JavaScript 库的小应用程序。最后，介绍将现有 JavaScript 项目逐步升级到 TypeScript 的过程。

在第Ⅱ部分，将 TypeScript 应用于区块链应用程序的开发。你可能会想：“我工作过的公司都没有使用区块链技术，当我的目标是精通 TypeScript 时，为什么还要学习区块链知识呢？”我们不想让示例应用成为另一个 ToDo 示例，因此寻找了一种热门技术，针对其实践应用第Ⅰ部分中介绍的不同的 TypeScript 元素和技术。了解 TypeScript 如何在一个不同寻常的应用程序中使用，可以让这些内容更加实用，即使你在不久的将来不会使用区块链技术。

在本书第Ⅱ部分，将开发多个版本的区块链应用程序：独立版本、Web 版本、Angular 版本、React.js 版本以及 Vue.js 版本。请自由阅读那些你感兴趣的章节，但一定要阅读第 8 章和第 10 章，其中介绍了基本概念。

第 8 章介绍区块链应用的原理。你将了解哈希函数的用途、区块挖掘的含义以及向区块链添加新的区块时需要工作证明的原因。

第 9 章介绍如何为区块链创建 Web 客户端。这个应用程序不会使用任何 Web 框架，仅使用 HTML、CSS 和 TypeScript。还将创建一个用于生成哈希的小库，它可以在 Web 客户端和独立客户端中使用。你还将了解如何在浏览器中调试 TypeScript 代码。

第 10 章展示区块链应用程序的代码，该应用程序使用消息服务器在区块链成员之间进行通信。我们使用 TypeScript 构建了 Node.js 和 WebSocket 服务器，并且展示了如何使用最长链规则来达成选择共识。你将看到使用 TypeScript 接口、抽象类、访问限定符、枚举和泛型的实际例子。

第 11 章简要介绍如何使用 TypeScript 开发 Angular Web 应用程序，第 12 章展示使用 Angular 框架开发区块链 Web 客户端的代码。

第 13 章简要介绍如何使用 TypeScript 开发 React.js Web 应用程序，第 14 章回顾使用 React 开发区块链 Web 客户端的代码。

类似地，第 15 章介绍使用 TypeScript 开发 Vue.js Web 应用程序，第 16 章则回顾使用 Vue 开发区块链 Web 客户端的代码。

关于代码

本书中有许多示例的源代码，既有有编号的代码清单，也有普通文本。在这两种情况下，源代码都是用固定宽度字体格式化的，以将其与普通文本分开。有时，代码也会以粗体显示与

本章前面描述步骤不同的代码，例如，当新功能添加到现有代码行时。

在大多数情况下，原始的源代码已经被重新格式化；我们添加了换行符和缩进，以适应本书的页面空间。在少数情况下，即使这样做还不够，需要在代码清单中使用行延续标记(➥)。此外，代码作为普通文本时，源代码中的注释通常会被删除。代码列表中的代码有大量注释，它们标记了重要的概念。

第Ⅰ部分是关于该语言的语法，大多数代码示例都是在 TypeScript Playground 上在线发布的——这个交互式工具可以快速检查 TypeScript 代码的语法并将其编译为 JavaScript。书中根据需要提供了指向这些代码的链接。

本书的第Ⅱ部分包括多个项目，这些项目使用 TypeScript 以及一些流行的库和框架(例如 Angular、React.js 和 Vue.js)开发应用程序。这些应用程序的源代码可以在 GitHub 上找到，网址是 https://github.com/yfain/getts。

我们测试了本书涉及的所有应用程序，但是 TypeScript 和其他库可能会发布新版本，并伴随着重大改变。如果你在尝试运行其中一个项目时遇到了错误，请在本书的 GitHub 库中打开一个问题。

关于作者

Yakov Fain 是两家 IT 公司的联合创始人：Farata Systems 和 SuranceBay。他有许多个人著书或合著的书，例如 Java 编程书籍 *24-Hour Trainer, Angular Development with TypeScript*, *Java Programming for Kids* 等。作为一名 Java 专家，他讲授多个关于 Web 和 Java 相关技术的课程和研讨会，并在国际会议上发表演讲。

Anton Moiseev 是 SuranceBay 的首席软件工程师。他使用 Java 和.NET 技术进行企业级应用开发已经有十多年。他具有坚实的后台开发基础和对 Web 相关技术的高度专注，能够使前端与后端无缝协作。他讲授了很多关于 AngularJS 和 Angular 框架的培训课程。

关于封面插图

本书封面插图的标题是 Bourgeoise Florentine。插图取自一个包含不同国家连衣裙的名为 *Costumes civils actuels de tous les peuples connus* 的图集，该图集于 1788 年在法国出版，作者是 Jacques Grasset de Saint-Sauveur (1757—1810)。每幅插图都是手工绘制和着色的。Grasset de Saint-Sauveur 的种类多样的图集清晰地提醒我们 200 年前世界上的城镇和地区在文化上的差异性。人们彼此隔绝，讲不同的语言和方言。在街上或乡下，只要看他们的衣着，就很容易辨认出他们住在哪里，他们的职业或地位是什么。

从那时起，我们的着装方式发生了变化，当时如此丰富的地区差异也逐渐消失。现在很难区分不同大陆的居民，更不用说不同的城镇、地区或国家了。也许我们用文化差异性换来了更为多样化的个人生活——更加多样化和快节奏的科技生活。

在一个很难从一堆书中分辨出一本计算机书的时代，Manning 利用书的封面来展示计算机行业的创造性和主动性，书的封面基于两个世纪前丰富多样的地区生活，并通过 Grasset de Saint-Sauveur 的画作重现生机。

目　录

第 I 部分　精通 TypeScript 语法

第 1 章　熟悉 TypeScript …… 3
1.1　使用 TypeScript 编程的理由 …… 3
1.2　典型的 TypeScript 工作流 …… 7
1.3　使用 TypeScript 编译器 …… 8
1.4　了解 Visual Studio Code …… 12
1.5　本章小结 …… 14

第 2 章　基本类型与自定义类型 …… 15
2.1　声明变量类型 …… 15
2.1.1　基本类型标注 …… 16
2.1.2　函数声明中的类型 …… 20
2.1.3　union 类型 …… 21
2.2　定义自定义类型 …… 23
2.2.1　使用 type 关键字 …… 24
2.2.2　将类用作自定义类型 …… 25
2.2.3　将接口用作自定义类型 … 27
2.2.4　结构化还是名义类型系统 …… 29
2.2.5　自定义类型的 unions …… 31
2.3　any 和 unknown 类型，以及用户定义的类型保护 …… 33
2.4　微型项目 …… 35
2.5　本章小结 …… 36

第 3 章　面向对象编程的类和接口 …… 37
3.1　类 …… 37
3.1.1　开始了解类继承 …… 38
3.1.2　访问修饰符 public、private、protected …… 39
3.1.3　静态变量及 singleton(单例)设计模式示例 …… 41
3.1.4　super()方法与 super 关键字 …… 43
3.1.5　抽象类 …… 45
3.1.6　方法重载 …… 48
3.2　使用 interface …… 53
3.2.1　执行合同 …… 53
3.2.2　扩展接口 …… 55
3.2.3　接口编程 …… 57
3.3　本章小结 …… 60

第 4 章　使用枚举和泛型 …… 61
4.1　使用枚举 …… 61
4.1.1　数字型枚举 …… 61
4.1.2　字符串枚举 …… 64
4.1.3　使用常量枚举 …… 66
4.2　使用泛型 …… 67
4.2.1　理解泛型 …… 67
4.2.2　创建自己的泛型类型 …… 72
4.2.3　创建泛型函数 …… 76
4.2.4　强制执行高阶函数的返回类型 …… 80
4.3　本章小结 …… 81

第 5 章　装饰器与高级类型 …… 83
5.1　装饰器 …… 84
5.1.1　创建类装饰器 …… 85
5.1.2　创建函数装饰器 …… 90
5.2　映射类型 …… 92

5.2.1 只读映射类型 …… 92
5.2.2 声明自己的映射类型 …… 96
5.2.3 其他内置的映射类型 …… 97
5.3 条件类型 …… 99
5.4 本章小结 …… 104

第 6 章 开发工具集 …… 105
6.1 源映射 …… 106
6.2 TSLint linter …… 108
6.3 使用 Webpack 绑定代码 …… 111
6.3.1 使用 Webpack 绑定 JavaScript …… 112
6.3.2 使用 Webpack 绑定 TypeScript …… 116
6.4 使用 Babel 编译器 …… 119
6.4.1 在 JavaScript 中使用 Babel …… 122
6.4.2 在 TypeScript 中使用 Babel …… 124
6.4.3 在 TypeScript 与 Webpack 中使用 Babel …… 126
6.5 工具介绍 …… 128
6.5.1 Deno 介绍 …… 128
6.5.2 ncc 介绍 …… 129
6.6 本章小结 …… 132

第 7 章 在项目中同时使用 TypeScript 和 JavaScript …… 133
7.1 类型定义文件 …… 133
7.1.1 了解类型定义文件 …… 134
7.1.2 类型定义文件与 IDE …… 135
7.1.3 shim 与类型定义 …… 138
7.1.4 创建自己的类型定义文件 …… 139
7.2 使用 JavaScript 库的 TypeScript 应用程序示例 …… 140
7.3 在 JavaScript 项目中使用 TypeScript …… 148
7.4 本章小结 …… 151

第 II 部分 基于 TypeScript 的区块链应用

第 8 章 开发区块链应用 …… 155
8.1 区块链简介 …… 156
8.1.1 加密哈希函数 …… 157
8.1.2 区块由什么组成 …… 159
8.1.3 什么是区块挖掘 …… 160
8.1.4 哈希和随机数的迷你项目 …… 162
8.2 开发第一个区块链 …… 164
8.2.1 项目结构 …… 164
8.2.2 创建一个原始区块链 …… 167
8.2.3 使用工作证明创建区块链 …… 170
8.3 本章小结 …… 173

第 9 章 开发基于浏览器的区块链节点 …… 175
9.1 运行区块链 Web 应用 …… 176
9.1.1 项目结构 …… 176
9.1.2 使用 npm 脚本部署应用 …… 178
9.1.3 使用区块链 Web 应用 …… 179
9.2 Web 客户端 …… 182
9.3 挖掘区块 …… 187
9.4 使用 crypto API 生成哈希 …… 191
9.5 独立的区块链客户端 …… 194
9.6 在浏览器中调试 TypeScript …… 196
9.7 本章小结 …… 198

第 10 章 使用 Node.js、TypeScript 和 WebSocket 进行客户端-服务器通信 …… 199
10.1 使用最长链规则解决冲突 …… 200
10.2 为区块链添加服务器 …… 202
10.3 项目结构 …… 203

10.4 项目的配置文件…… 204
10.4.1 配置 TypeScript 编译环境…… 204
10.4.2 package.json 包含什么 …… 206
10.4.3 配置 nodemon …… 207
10.4.4 运行区块链 App …… 208
10.5 WebSocket 简介 …… 213
10.5.1 HTTP 和 WebSocket 协议的对比…… 214
10.5.2 将数据从节点服务器推送到普通客户端…… 215
10.6 回顾通知工作流…… 219
10.6.1 回顾服务器代码…… 221
10.6.2 回顾客户端代码…… 231
10.7 本章小结…… 240

第 11 章 使用 TypeScript 开发 Angular 应用程序 ……241
11.1 使用 Angular CLI 生成并运行一个新的应用程序 …… 242
11.2 查看生成的 App…… 244
11.3 Angular 服务和依赖注入 …… 250
11.4 使用 ProductService 注入的应用 …… 253
11.5 使用 TypeScript 进行抽象编程 …… 255
11.6 开始处理 HTTP 请求…… 257
11.7 表单入门…… 261
11.8 Router 基础…… 265
11.9 本章小结…… 269

第 12 章 使用 Angular 开发区块链客户端 ……271
12.1 启动 Angular 区块链应用程序 …… 271
12.2 回顾 AppComponent…… 273
12.3 回顾 TransactionForm Component …… 277
12.4 回顾 BlockComponent…… 278
12.5 回顾服务…… 281
12.6 本章小结…… 283

第 13 章 使用 TypeScript 开发 React.js 应用程序 ……285
13.1 使用 React 开发最简单的网页 …… 286
13.2 使用 Create React App 生成并运行一个新应用 …… 288
13.3 管理组件的状态…… 293
13.3.1 向基于类的组件添加状态…… 293
13.3.2 使用钩子管理函数组件的状态…… 294
13.4 开发一个天气应用程序…… 297
13.4.1 向 App 组件添加状态钩子…… 298
13.4.2 在 App 组件中使用 useEffect 钩子获取数据…… 300
13.4.3 使用 props…… 306
13.4.4 子组件如何将数据传递给其父组件…… 311
13.5 Virtual DOM …… 313
13.6 本章小结…… 314

第 14 章 使用 React.js 开发区块链客户端 ……315
14.1 启动客户端和消息服务器…… 315
14.2 lib 目录中发生的变化…… 318
14.3 smart App 组件 …… 320
14.3.1 添加事务 …… 322
14.3.2 生成一个新区块 …… 324
14.3.3 解释 useEffect()钩子函数…… 325

14.3.4 使用 useCallback()钩子的记忆化缓存(Memoization)…… 327
14.4 presentation 组件——TransactionForm…… 330
14.5 presentation 组件——PendingTransactionsPanel…… 333
14.6 presentation 组件——BlocksPanel 和 BlockComponent …… 335
14.7 本章小结…… 337

第 15 章 使用 TypeScript 开发 Vue.js 应用程序 ……339
15.1 使用 Vue 开发最简单的 Web 页面 …… 340
15.2 使用 Vue CLI 生成和运行新应用程序 …… 343
15.3 开发有路由支持的单页应用程序 …… 349
15.3.1 使用 Vue Router 生成一个新应用程序…… 350
15.3.2 在主视图中显示 products 列表…… 353
15.3.3 使用 Vue Router 传递数据 …… 358
15.4 本章小结…… 362

第 16 章 用 Vue.js 开发区块链客户端 ……365
16.1 启动客户端和消息服务器…… 366
16.2 App 组件…… 369
16.3 presentation 组件——TransactionForm…… 372
16.4 presentation 组件——PendingTransactionsPanel…… 376
16.5 presentation 组件——BlocksPanel 和 Block…… 378
16.6 本章小结…… 382
后记…… 382

附录 A JavaScript 基础知识 ……383
A.1 如何运行代码示例 …… 383
A.2 关键字 let 和 const…… 383
A.2.1 var 关键字和 hoisting(提升)…… 384
A.2.2 使用 let 和 const 的块级作用域…… 385
A.3 字面量模板 …… 386
A.4 可选参数和默认值 …… 388
A.5 箭头函数表达式 …… 389
A.6 rest 运算符 …… 391
A.7 spread 运算符…… 393
A.8 解构 …… 394
A.8.1 解构对象 …… 394
A.8.2 解构数组 …… 397
A.9 类和继承 …… 398
A.9.1 构造函数 …… 400
A.9.2 super 关键字和 super 函数…… 401
A.9.3 静态类成员 …… 402
A.10 异步处理…… 403
A.10.1 回调地狱 …… 404
A.10.2 promise…… 404
A.10.3 同时执行多个 promise …… 407
A.10.4 async-await …… 408
A.11 模块…… 410
A.12 转换器…… 413

第Ⅰ部分

精通TypeScript语法

在本书的第Ⅰ部分，首先通过对 TypeScript 和 JavaScript 的比较来帮助你了解使用 TypeScript 的好处。然后，将陆续讲解 TypeScript 的各类语法元素并采用小代码片段进行演示。学习如何使用内置类型和声明自定义类型，如何使用类及接口，如何使用泛型、枚举、装饰器、映射类型和条件类型。你将由浅入深地逐步学习开发者常用的 TypeScript 工具，如编译器、linter、调试器和 bundler 等。最后，将介绍一种在 App 中同时使用 TypeScript 和 JavaScript 的方法。

为满足那些喜欢通过观看视频进行学习的读者的需求，Yakov Fain 发布了一系列视频(参见 http://mng.bz/m4M8)，这些视频描述了本书第Ⅰ部分的相关材料。如果你的目标是快速学习 TypeScript 的语法和工具集，本书第Ⅰ部分正好能够满足你的需要。

第1章

熟悉TypeScript

本章要点：

- 与 JavaScript 比较，使用 TypeScript 编程的优势
- 如何将 TypeScript 代码编译成 JavaScript
- 如何使用 Visual Studio 代码编辑器

本章目标是帮助你开启 TypeScript 开发的旅程。首先，对 JavaScript 表示我们的敬意，然后解释为什么要用 TypeScript 编程。最后将编译并运行一个非常简单的程序，帮助你了解从使用 TypeScript 编写代码到将代码编译成一段可执行的 JavaScript 的完整工作流程。

如果你是一位经验丰富的 JavaScript 开发人员，那么的确需要一个能够说服你使用 TypeScript 的恰当理由，因为 TypeScript 必须被编译为 JavaScript 后才能部署。如果你是一位后端开发人员，希望学习前端生态系统，也需要知道为什么要学习 TypeScript，而不是学习 JavaScript。首先，让我们告诉你这些理由。

1.1 使用 TypeScript 编程的理由

TypeScript 是微软在 2012 年发布的开源项目，是一种最终要编译为 JavaScript 的编程语言。由 TypeScript 编写的程序首先需要被转编译为 JavaScript，然后才可以在浏览器中或者在独立的 JavaScript 引擎中执行。

转编译和编译的差别在于，编译直接将程序的源代码编译为字节码或机器码，而转编译首先要将一种语言转换为另一种语言，例如从 TypeScript 转换为 JavaScript。但是在 TypeScript 社区中，更流行用编译来描述这一过程，因此在本书中，我们将采用编译这个词来描述将

TypeScript 转换为 JavaScript 的过程。

你也许想知道，为什么要不辞劳苦地先用 TypeScript 编写程序，然后再将其编译为 JavaScript，而不是直接就用 JavaScript 编写程序呢？要回答这个问题，让我们先从更高级的层面上看看 TypeScript。

TypeScript 是 JavaScript 的超集，因此你可以任取一个 JavaScript 文件(例如 myProgram.js)，将扩展名从.js 改为.ts，这样 myProgram.ts 就可能成为一个合法的 TypeScript 文件。之所以说“可能”，是因为源 JavaScript 代码可能隐藏着与类型有关的错误(它可能动态地改变对象属性的类型，或者在声明对象后，增加了新的类型)以及其他问题，但这些问题只有在 JavaScript 代码被编译后才能被发现。

提示 在第 7.3 节中，我们将提供一些将 JavaScript 代码迁移到 TypeScript 代码的技巧。

通常，“超集”这个词，意味着它包含集合拥有的一切，还包含集合没有的一些东西。如图 1.1 所示，TypeScript 作为 ECMAScript 的超集，它是所有版本 JavaScript 的规范定义。ES.Next 代表 ECMAScript 的最新修订，但目前尚未完成。

图 1.1 作为超集的 TypeScript

除了支持 JavaScript 集外，TypeScript 也支持静态类型，而 JavaScript 仅支持动态类型。此处的“类型”意指给程序变量分配的类型。

对支持静态类型的编程语言来说，在使用变量前，必须为变量声明一种类型。对 TypeScript 来说，可以将变量声明为某种类型，此后，所有试图将与定义类型不同类型的值赋给该变量的尝试都会导致编译错误。

对 JavaScript 来说，情况却并非如此，因为 JavaScript 直到运行时才知道程序中变量的类型。即使在运行时，仍然可以通过给变量分配不同类型的值的方式来改变变量的类型。对 TypeScript 来说，如果你声明某个变量为字符串类型，在程序中为其分配数字值将会在编译时出现错误。

```
let customerId: string;
customerId = 123; // 编译错误
```

JavaScript 在运行时才确定变量类型，而且变量的类型可以动态变换，如下实例所示。

```
let customerId = "A15BN"; // OK，customerId 被定义为字符型
customerId = 123; // OK，从现在开始，customerId 变为数字型
```

接下来考虑一个 JavaScript 函数，该函数提供价格打折计算。函数包含两个参数，均为数字型。

```
function getFinalPrice(price, discount) {
return price - price / discount;
}
```

你如何知道参数一定是数字类型的呢？首先，该程序是你在不久前编写好的，你具有非凡的记忆力，刚好能够记住所有参数类型；其次，给参数使用描述性名称，这些名称恰好暗示出它们的类型；最后，通过阅读函数的代码猜测出参数的类型。

上述函数虽然非常简单，假设有人调用了该函数，折扣被该调用者以字符类型提供给函数，则函数在运行时将给出“NaN”错误。

```
console.log(getFinalPrice( 100, "10%")); // 控制台屏幕显示 NaN
```

该实例给出了错误使用函数造成运行错误的情况之一。在 TypeScript 中，你可以给函数的参数提供类型，因此此类运行错误是不可能发生的。如果有人在调用函数时，采用错误的参数类型调用函数，这类错误在你定义类型时就会被发现，让我们看看其实际情况。

TypeScript 官方网页(www.typescriptlang.org)提供了语言文档和背景，可以输入你的 TypeScript 代码片段，代码片段将立即被编译为 JavaScript 代码。

通过网页 http://mng.bz/Q0Mm，将看到 TypeScript 背景的小代码片段，在"10%"下面有一条红色波浪线。如果你将鼠标放在错误代码上，将会看到一个解释错误的提示，如图 1.2 所示。

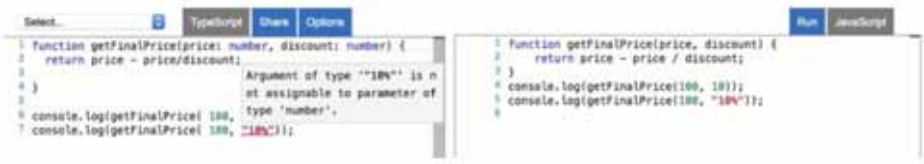

图 1.2　使用 TypeScript 运行环境

定义类型时，在用 TypeScript 编译器编译该代码前，由 TypeScript 静态代码分析器就能发现该错误。此外，定义变量类型时，编辑器或者 IDE(集成开发环境)将根据特征自动为函数 getFinalPrice()提供建议的参数名称和类型。

在运行前发现错误不好吗？我们认为当然好。大多数具有这类语言(如 Java、C++以及 C#)背景的开发者都会想当然地认为，此类错误应该在编译时被发现，这也是他们喜欢 TypeScript 的主要原因之一。

注意　有两类编程错误——一类错误是当你输入时立即就会被工具报告。另外一类错误是由使用程序的用户报告。采用 TypeScript 编程将会大大减少后者出现的次数。

提示　TypeScript 网站(www.typescriptlang.org)有一个被称为“文档和教程”的部分你可以在此找到用在不同环境中，例如 ASP.NET、React 以及其他环境中配置 TypeScript 的提示。

某些“坚定”的 JavaScript 开发者认为，由于在使用 TypeScript 时需要定义类型，因此会降低他们的开发速度，而使用 JavaScript 的效率会更高。请记住，在 TypeScript 中，类型定义是可选的——你可以继续用 JavaScript 编写程序并且仍然可以在工作流中使用 tsc。为什么呢？因为你将能够使用最新的 ECMAScript 语法(例如 async 和 await)并将你的 JavaScript 代码编译为 ES5。由此，你的代码就可以在更早期的浏览器上运行。

但是大多数 Web 开发者不是 JavaScript 的“死忠粉”，他们能欣赏使用 TypeScript 所带来的好处。事实上，所有强类型语言都提供更好的工具支持，并由此提高了生产效率(即便是对于“死忠粉”来说也是如此)。话虽如此，我们希望强调的是，TypeScript 为使用者在他们想要的时间和地方提供静态类型语言所带来的好处，而且并未阻止使用者在想用时方便地使用过去的动态 JavaScript 对象。

超过 100 种编程语言被编译为 JavaScript(参考如下网址：http://mng.bz/MO42)。但使 TypeScript 能够鹤立鸡群的原因是其设计者遵循 ECMAScript 规范，在实现 JavaScript 未来特性方面比 Web 浏览器开发商更快一些。

可以在 GitHub(https://github.com/tc39/proposals)上找到当前对新的 ECMAScript 特性的提案。每个最终被包含在下一版 ECMAScript 规范中的提案都会经历几个阶段。如果某个提案进入第 3 阶段，则该提案极有可能会包含在最新版本的 TypeScript 中。

2017 年夏天，async 和 await 关键字(参考本书附录 A.10.4 相关内容)被加入 ECMAScript 规范 ES2017 中(又称 ES8)。大约一年后，主要的浏览器开发商才开始支持这些关键字，而 TypeScript 早在 2015 年 11 月就已经开始支持这些关键字了。TypeScript 开发者在浏览器支持这些关键字的三年前就已经能够开始使用这些关键字了。最棒的是，你可以在今天的 TypeScript 代码中使用未来的 JavaScript 才能用到的语法，并且可以将其编译成早期的被所有浏览器支持的 JavaScript 语法(例如 ES5)！

话虽如此，仍然需要分清最新的 ECMAScript 规范描述的语法与 TypeScript 特有的语法之间的差异。我们建议读者首先阅读附录，这样就能知道 ECMAScript 和 TypeScript 的前世今生。

尽管 JavaScript 引擎在根据变量值猜测变量类型方面提供了一些相当不错的工作，但开发工具在不知道变量类型的情况下，能为开发者提供的帮助还是比较有限的。在中大型应用中，JavaScript 这一缺陷降低了软件开发者的编程效率。

TypeScript 遵循最新的 ECMAScript 规范，添加了类型、接口、装饰器、类成员变量(字段)、范型、枚举，关键字 public、protected 以及 private 等。查验 TypeScript 的路线图(https://github.com/Microsoft/TypeScript/wiki/Roadmap)可以发现未来 TypeScript 版本中可能具有和将会具有的内容。还有一件事：由 TypeScript 创建的 JavaScript 代码非常容易阅读，看起来就像手写代码一样。

关于 TypeScript 的 5 个事实如下：

- TypeScript 的核心开发者是 Anders Hejlsberg，他也是 Turbo Pascal 和 Delphi 的设计者，以及微软 C#的首席架构师。
- 2014 年年底，Google 与微软接洽，询问微软能否将装饰器引入 TypeScript 中，使该语言能够用于开发 Angular2 框架。微软同意了这一请求，这一决定对 TypeScript 的流行起到了巨大的作用，因为成千上万开发者使用 Angular。
- 截至 2019 年 12 月，npmjs.org 网站的 tsc 每周大约有几百万次下载，注意这并不是唯一的 TypeScript 下载网站。关注最新的统计情况，请参考 www.npmjs.com/package/typescript。

- 根据软件分析业界著名公司 Redmonk 提供的数据，TypeScript 在 2019 年 1 月的编程语言排名中位列第 12 位(排名情况参考 http://mng.bz/4eow)。
- 根据 Stack Overflow 公司 2019 年开发人员调查结果，TypeScript 在最受欢迎的语言排名中名列第 3 位(排名情况参考 https://insights.stackoverflow. com/survey/2019)。

下面开始介绍配置处理过程以及如何在你的计算机上使用 tsc。

1.2　典型的 TypeScript 工作流

从编写代码开始直到部署应用，我们将以此开始熟悉 TypeScript 的工作流程。图 1.3 给出了这样一个工作流，假设 App 的全部源代码都由 TypeScript 编写。

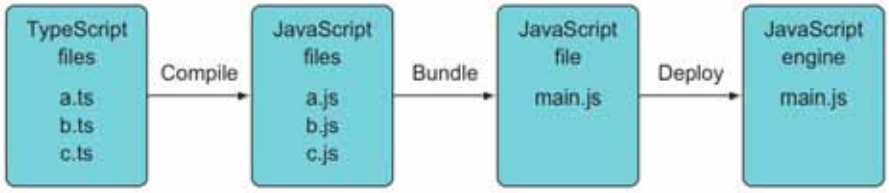

图 1.3　部署用 TypeScript 编写的 App

如图 1.3 所示，项目包含 3 个 TypeScript 文件：a.ts、b.ts、c.ts。这些文件将会由 TypeScript 编译器(tsc)编译为 JavaScript，产生 3 个新文件类型：a.js、b.js、c.js。在本节后面，我们将介绍如何让编译器创建特定版本的 JavaScript。

针对此情况，某些 JavaScript 开发者可能会说，“TypeScript 强制我在编写代码与观察代码运行之间，使用额外的编译步骤。”但是你是否真正希望坚持采用 JavaScript 的 ES5 版本，忽略所有由 ES6、ES7、ES8 以至最新的 ES.next 版本所带来的最新语法呢？如果希望采用新语法，则工作流中都需要有一个编译步骤——需要将你用最新的 JavaScript 版本编写的源代码编译为得到广泛支持的 ES5 语法。

图 1.3 仅包含 3 个文件，而现实中的项目可能会包含成百上千的文件。开发者并不希望将如此大量的文件都部署到 Web 服务器或者独立的 JavaScript 应用程序上，因此我们通常将这些文件打包在一起(也可认为是“连接”)。

JavaScript 开发者通常使用不同的绑定器，类似 Webpack 或 Rollup，这些工具不仅将 JavaScript 文件连接在一起，而且能够优化代码并删除无用的代码(执行“摇树优化”操作，将无用代码删除掉。该方法最先在 Rollup 中被采用)。若你的 App 包含几个模块，每个模块可以作为单独的包部署。

图 1.3 仅仅展示一个部署包——main.js。如果它是一个 Web App，其 HTML 文件中将包含<script src='main.js'>标识。如果该 App 运行在一个独立的 JavaScript 引擎上，例如 Node.js，则可以下列语句启动它(当然前提是 Node.js 已经被安装)。

```
node  main.js
```

JavaScript 生态环境包括几千个库，这些库没有用 TypeScript 重写。好消息是，你的 App 除了可使用 TypeScript，还可使用任何已有的 JavaScript 库。

如果你仅仅在你的 App 上增加 JavaScript 库，TypeScript 编译器不会在你使用这些库的 API 时，自动完成或提供错误信息。但是有一些以.d.ts(详见第 6 章)结尾的特殊类型定义的文件，如果这些文件存在，TypeScript 编译器将告诉你错误，并提供针对该库的上下文相关的帮助。TypeScript 编译器用了比较流行的 JavaScript 库 lodash。

该图包含类型定义文件 lodash.d.ts，该文件在开发期间被 tsc 使用。图 1.4 还包含实际存在的 JavaScript 库 lodash.js，在部署期间将与 App 的其余部分打包。术语“打包”的含义为将几个 Script 文件合并为一个文件的过程。

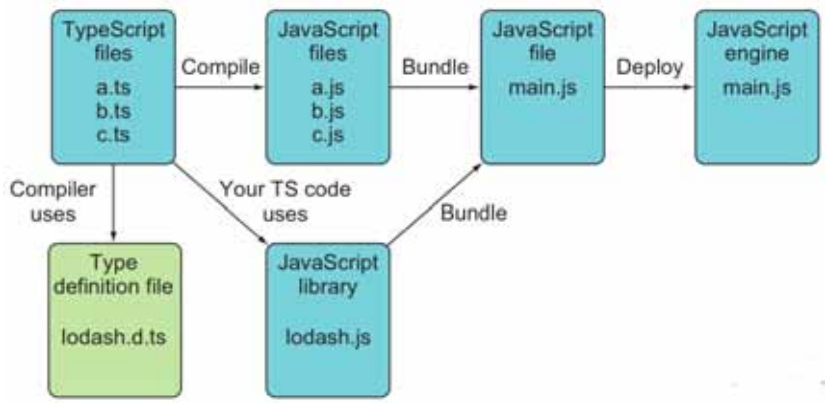

图 1.4 部署用 TypeScript 和 JavaScript 编写的 App

1.3 使用 TypeScript 编译器

通过以上内容，我们学习了如何将基本的 TypeScript 文件编译为 JavaScript 版本。编译器 tsc 可以与你所选择的 IDE(集成开发环境)打包，也可以作为插件安装到 IDE 上，但是我们推荐的方法是，最好是使用 Node.js 附带的 npm 包，在你的 IDE 上独立安装 tsc。

Node.js(或简称 Node)不仅是一个框架或库——它也是 JavaScript 运行环境。采用 Node 运行各类实用程序，类似 nmp 或在没有浏览器的场合下启动 JavaScript 代码。

就此开始吧，首先需要从 https://nodejs.org 下载并安装当前最新版本的 Node.js。它将安装 node 和 npm。

用 npm，可以把软件安装到你自己本地的项目目录中，也可以全局方式安装，以便能够在多个项目中使用。我们将使用 npm 安装来自 npm 库(位于 www. npmjs.com)的 tsc 和其他包，该库包含 50 多万个不同的包。

可以在终端窗口中，用以下命令以全局方式安装 tsc(采用-g 选项)。

```
npm install -g typescript
```

注意　为简单起见，我们将在本书的第I部分使用全局方式安装的 tsc。但在实际项目中，通常愿意以本地方式将 tsc 安装到项目目录下，方法是在项目的 package.json 中的 devDependencies 段中添加 tsc。第 8 章列举区块链项目的示例时，将讨论具体实现细节。

在本书的代码示例中，我们使用的是 TypeScript 版本 3 或更新的版本。若想知道你所用的 tsc 的版本，在终端窗口中运行如下代码即可。

```
tsc -v
```

现在让我们看看如何将一个简单程序从 TypeScript 编译到 JavaScript。创建一个目录，编写包含以下内容的 main.ts 文件(见代码清单 1.1)。

代码清单 1.1　main.ts 文件

```
 function getFinalPrice(price: number, discount: number) {
  return price - price/discount;
}

console.log(getFinalPrice(100, 10));
console.log(getFinalPrice(100, "10%"));
```

包含类型的函数参数

正确的函数调用

错误的函数调用

使用以下命令将 main.ts 编译成 main.js。

```
tsc main
```

运行该命令将返回错误信息“参数‘10%’的类型不能被分配为“数字”类型”，但是仍将生成包含以下内容的 main.js 文件(见代码清单 1.2)。

代码清单 1.2　生成的 main.js 文件

```
function getFinalPrice(price, discount) {
  return price - price/discount;
}

console.log(getFinalPrice(100, 10));
console.log(getFinalPrice(100, "10%"));
```

没有定义类型的参数

正确的函数调用

错误的函数调用，但是仅在运行时显示错误。

你可能会问，“如果存在编译错误，生成 JavaScript 文件的意义是什么呢？”当然，从 JavaScript 角度来看，main.js 文件的内容是合法的。但是在实际的 TypeScript 项目中，我们不希望为错误文件生成代码。

tsc 提供许多编译选项，在文档(http://mng.bz/rfl4)中有这些选项的描述，其中一个选项为 noEmitOnError。删除 main.js 文件并试着编译 main.ts。

```
tsc  main --noEmitOnError  true
```

执行该命令，在 main.ts 文件中的错误被更正前，不会创建 main.js 文件。

提示 开启 noEmitOnError 选项意味着，在 TypeScript 文件中所有的错误被更正前，先前创建的 JavaScript 文件不会被替换。

编译器的--t 选项允许定义目标 JavaScript 语法。例如，可以使用同样的源文件并生成它的符合 ES5、ES6 或最新语法的 JavaScript 对等版本。此处给出将代码编译为 ES5-兼容的语法：

```
tsc --t ES5 main
```

tsc 允许预先配置编译过程(定义源和目的目录，目标等)，若你的项目目录中包含 tsconfig.json 文件，则只需要在命令行输入 tsc，编译器将从 tsconfig.json 文件中阅读所有选项。示例 tsconfig.json 文件如代码清单 1.3 所示。

代码清单 1.3 tsconfig.json 文件

```
{
  "compilerOptions": {
    "baseUrl": "src",

    "outDir": "./dist",
    "noEmitOnError": true,
    "target": "es5"
  }
}
```

需要转编译的.ts 文件位于 src 目录中

在 dist 目录下保存生成的.js 文件

如果文件存在编译错误，不要生成 JavaScript 文件

将 TypeScript 文件转编译为 ES5 语法

提示 编译器的目标选项也用于语法检查。例如，若定义 es3 作为编译目标，TypeScript 将对代码中的 getter 方法提出异议。它不知道如何将 getter 编译成语言的 ECMAScript 3 版本。

让我们看看是否能独自完成以下指令。

(1) 在 main.ts 文件所在的目录中创建名为 tsconfig.json 的文件，为 tsconfig.json 文件增加如下内容。

```
{
  "compilerOptions": {
      "noEmitOnError": true,
      "target": "es5",
      "watch": true
  }
}
```

注意最后一个选项，watch。编译器会观察你的 TypeScript 文件，当文件发生改变时，tsc 将重新编译这些文件。

(2) 在终端窗口中，回到 tsconfig.json 文件所在的目录，运行以下命令。

```
tsc
```

你将看到本节前面描述过的错误信息，但是编译器不会退出，因为它的运行方式为 watch

模式。文件 main.js 不会被创建。

(3) 更改错误，代码将会自动重新编译。可以发现，这次创建了 main.js 文件。

如果打算关闭 watch 模式，仅需要在终端窗口中，按下 Ctrl+C 快捷键即可。

提示 开始一个新的 TypeScript 项目时，可以在任意目录下运行命令 tsc --init。输入后将会创建一个包含所有编译器选项的 tsconfig.json 文件，当然，大部分选项将会被注释掉。若需要的话，去掉注释即可。

注意 使用 extends 属性，tsconfig.json 文件可以继承其他文件的配置。第 10 章中将提供包含三个配置文件的简单项目：第一个含有整个项目公共的 tsc 编译器选项；第二个面向客户；第三个面向项目的服务器部分。详情请读者参阅本书的第 10.4.1 节。

TypeScript 的 REPL 环境

REPL 表示读取-评估-打印-循环(Read-Evaluate-Print-Loop)，它涉及简单的交互语言外壳，该外壳允许快速执行代码片段。www.typescriptlang.org/play 的 TypeScript Playground 是一个针对 REPL 的示例，允许在浏览器中写、编译和执行小代码片段。

以下示例告诉你如何才能使用 TypeScript Playground 将 TypeScript 类编译为 ES5 版本的 JavaScript。

Transpiling TypeScript to ES5

下图展示了如何将相同的代码转编译到 ES6 版本的 JavaScript。

Transpiling TypeScript to ES6

Playground 包含一个选项菜单，可在菜单上选择编译器的选项。而且还可以选择编译目标，例如 ES2018 或 ES5。

如果你喜欢从命令行而不是从浏览器运行小代码片段，请安装 TypeScript Node REPL，可在 https://github.com/TypeStrong/ts-node 找到相关文档。

1.4 了解 Visual Studio Code

集成开发环境(IDE)和代码编辑器提高了开发者的生产率，Visual Studio Code、WebStorm、Eclipse、Sublime Text、Atom、Emacs、Vim 等工具为 TypeScript 提供了极大的支持。在本书中，我们决定使用微软开发的开源和免费的 Visual Studio Code(VS Code)代码编辑器，读者当然可以使用其他任何能够使用 TypeScript 的代码编辑器或集成开发环境。

注意 按照 Stack Overflow 2019 开发者报告(https://insights.stackoverflow.com/survey/2019)，VS Code 是最流行的开发环境，超过 50%的受访者使用 VS Code。顺便说一下，VS Code 是用 TypeScript 编写的。

实际项目中，能够提供好的上下文相关的帮助和支持，对重构来说非常重要。对静态类型语言中出现的所有 TypeScript 变量或者函数名重新命名的工作，IDE 瞬间就能完成，但由于 JavaScript 不支持类型，因此情况并非如此。在 TypeScript 编码时，如果在函数、类或者变量的命名时出现了错误，错误处将会被标红。

可以从 https://code.visualstudio.com 处下载 VS Code。安装过程和你的计算机使用的操作系统有关，具体情况参考 VS Code 文档(https://code.visualstudio. com/docs)的设置部分。

安装完成后，启动 VS Code。然后，使用 File | Open 菜单选项，打开 chapter1/ vscode 目录，该目录包含本书的代码示例。示例包括先前部分的 main.ts 文件以及一个作为示例的 tsconfig.json 文件。图 1.5 中"10%"包含一个红色波浪下画线，表明存在一个错误。如果你将鼠标指针悬停在带下画线的代码上，它将给出与图 1.2 相同的错误信息。

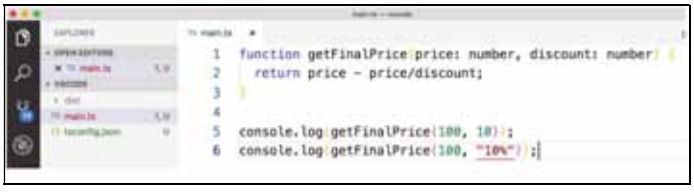

图 1.5 VS Code 中突出显示的错误

TypeScript 的 VS Code 模式

VS Code 支持两种类型的 TypeScript 模式：文件范围与显式项目。文件范围是非常有限的，因为它不允许一个文件中的脚本使用另外一个文件中声明的变量。显式的项目模式需要在项目目录中包含一个 tsconfig.json 文件。

本节附带的 tsconfig.json 文件如代码清单 1.4 所示。

代码清单 1.4 vscode/tsconfig.json

```
{
  "compilerOptions": {
    "outDir": "./dist",
```

将生成的 JavaScript 文件保存至 dist 目录中

```
    "noEmitOnError": true,
    "lib": ["dom", "es2015"]
  }
}
```

所有错误修正后，才能生成 JavaScript

要添加的库，以防 tsc 会因为未知的 APIs 而报错，例如 console()

如果希望从命令提示符就能够打开 VS Code，需要将 VS Code 的可执行性添加到你所用计算机的 PATH 环境变量中。在 Windows 环境下，设置过程可以自动完成。

在 macOS 下，启动 VS Code，选择 View | Command Palette 菜单选项，输入 shell command，点选该操作：Shell Command：Install ‘code’ Command in PATH。然后重新启动终端窗口并在任意目录输入 code。VS Code 启动后，可以处理你所在目录下的文件。

在前面，我们在不同的终端窗口中编译代码，但是 VS Code 带有集成终端。这种方式是我们不用离开编辑窗口，就能使用命令行提示符窗口。为打开 VS Code 终端窗口，从菜单项选择 View | Terminal 或 Terminal | New Terminal 命令。

图 1.6 给出了执行 tsc 后的集成终端的视图。右边箭头指向的“+”图标表明允许打开任意数量的终端视图。图中最后一行包含错误的代码行被注释掉了，最终 tsc 将在 dist 目录中创建 main.js 文件。

图 1.6　在 VS Code 上运行 tsc 命令

提示　VS Code 选择计算机上 Node.JS 中的 tsc 编译器。打开任意一个 TypeScript 文件，你将看到右侧底部工具栏上显示的 tsc 版本。如果希望使用已经以全局模式安装在计算机上的 tsc，单击底部右角处的版本号，选择需要的 tsc 编译器。

在图 1.6 中，在左侧黑色面板的底部，可以看到方形光标——用于发现并安装来自 VS Code 市场的扩展。这些扩展将使在 VS Code 中的 TypeScript 编程更加灵活。

- ESLint——集成 JavaScript linter 并检查代码的可读性和可维护能力。
- Prettie——通过分析代码并以自身规则重新格式化代码，强制实现一致性类型。
- Path Intellisense——自动完成文件路径。

关于使用 VS Code 实现 TypeScript 编程的更多细节，请查看 https://code.visualstudio.com/docs/languages/typescript 上的相关产品文档。

提示 StackBlitz(https://stackblitz.com)是一个非常不错的在线 IDE。它由 VS Code 驱动，但你并不需要将其安装到计算机上。

注意 本书第 II 部分包含不同版本区块链应用示例。尽管读者可以选择性阅读本书第 II 部分，但我们还是推荐至少阅读第 8 和第 9 章内容。

1.5 本章小结

- TypeScript 是 JavaScript 的超集。用 TypeScript 编写的程序必须首先转编译为 JavaScript，然后才能在浏览器或者独立的 JavaScript 引擎上执行。
- 即使尚未用 TypeScript 编译器(tsc)编译你的代码，TypeScript 静态代码分析器就可以在你定义类型时发现错误。
- 无论何时何地，TypeScript 都会带给你静态类型语言具有的好处。在需要时，不必停止使用旧的动态 JavaScript 对象。
- TypeScript 遵循最新 ECMAScript 规范，且为它们添加了类型、接口、装饰器、类成员变量(字段)、泛型、枚举，以及 public、protected、private 等关键字。检查位于 https://github.com/Microsoft/TypeScript/wiki/Roadmap 的 TypeScript 路线图可以发现已经存在的和未来 TypeScript 版本中将会具有的内容。
- 开始新的 TypeScript 项目时，只需要在任意目录下运行命令 tsc --init。它将创建 tsconfig.json 文件，包含大部分被注释掉了的编译器选项，需要时去掉相关的注释即可。