第0章 前言

经过了整整三个月的努力，我终于将这本已在心中酝酿了很久的小书写完了。写书真是一种很奇特的经历，整个过程既让人觉得很纠结，很惶恐，也令人感到很兴奋，很快乐。我个人认为，在如今的互联网上，人们只要能善用搜索引擎，基本上就可以找到自己想要了解的任何信息了。因而在这个时代，写书的目的已不应该只是单纯地普及知识了，它应该更多地表现作者自己的一些观点和经验。因为只有这种个性化的东西才是任何人工智能的产物所无法替代的，这些东西当然未必正确，但它能刺激思考，引发讨论，使人沉淀，而这些恰恰是如今互联网上所缺少的。所以，我希望通过这本小书来介绍一下个人对`Markdown`这种写作方式的看法和使用经验，以此来抛砖引玉，引起大家对`Markdown`更多的关注，进而将软件开源的精神推广至写作领域。毕竟，文字作品才是我们人类开发时间最长，数量最多的一种“软件”。

为什么要写这本书?

写这本书的最初念头源自于一次在Facebook上的抱怨。由于我自己是一个`Markdown`的重度使用者，在日常做笔记、写文章、翻译书籍时，经常需要搜寻各种使用`Markdown`写作的解决方案。而与此同时，市面上的各种博客、论坛、云端笔记服务也都纷纷加入了对`Markdown`的支持，这说明使用这门标志语言的用户并不在少数，但我却惊讶地发现自己市场上竟然找不到一本介绍`Markdown`的专著。于是就在Facebook上分享了下面这个想法：

> 我是觉得markdown写作可以延伸出很多东西啊，写论文涉及LaTeX、Mermaid等，制作电子书涉及gitbook，建构博客涉及Hexo，居然没人写本书！可惜了……

很自然地，这条想法分享的下面就有朋友留言建议我“不如你来写吧”。虽然当时我只是在表达自己需要这样一本书，最好请某位专业人士来写一本，但与朋友的讨论让我重新审视了自己所分享的这个想法。这个想法实际上说明了我为什么喜欢用`Markdown`来写作的原因：

- 第一，`Markdown`是开源软件，符合开放、自由、专注于任务，便于同行协作的工作哲学。
- 第二，`Markdown`符合“数据与呈现样式、用户界面分离”程序设计思维。
- 第三，`Markdown`的纯文本特性使我们可以想管理程序员源代码一样管理自己的文字作品。

总结一下，就是`Markdown`可以让人们“像写程序一样写作”，这让我意识到写这样一本书的意义已经不仅仅是介绍一门轻量级的标记语言，而是在推广一种强调自由、开放、合作的价值观和方法论了。而这种价值观和方法论原本就是我多年以来一直在坚持的，如今既然看到没有人写一本关于`Markdown`的专著，不如就自己来为它的推广做点事吧。

这本书写了些什么?

在这本书中，我以一篇本科毕业论文的写作过程为导引，介绍了`Markdown`在完成论文的规划、撰写、修改、发布这些不同任务阶段中的应用。全书被分成了六个章节和两个附录：

- **第1章 使用Markdown写作**：在这一章中，我们介绍了`Markdown`是什么、它有什么优势和劣势以及它所倡导的写作理念。需要说明的是，这一章的内容是为对`Markdown`一无所知的朋友准备的。如果读者自认为已经对`Markdown`有所了解，或者不想纠缠于技术概念，想快点进入“如何使用`Markdown`”的议题，也可以选择跳过这一章。
- **第2章 写作的前期准备**：在这一章中，我们首先介绍了几款值得一试的`Markdown`编辑器。然后，我们以论文的前期规划为导引，带大家学习了使用`Markdown`的标记来拟定论文大纲、表列论文的参考资料、并通过设定待办事项来安排写作的进度。

- **第3章 撰写一篇论文**：在这一章中，我们继续以论文的正式写作过程为导引，逐步深入地介绍了其余主要的`Markdown`标记，以及它们的具体使用。这其中既会包含用来表示段落、强调、引用、代码这些基本元素的原生`Markdown`标记，也会涉及到与表格、图形相关的扩展标记，以及它们的基本用法。
- **第4章 谈谈数学问题**：在这一章中，我们首先介绍了如何在`Markdown`文档中插入$\LaTeX$标记，以呈现数学公式。然后，我们会具体介绍如何用$\LaTeX$标记来描述基本四则运算、二项式方程、矩阵运算以及集合运算等数学问题。
- **第5章 作品的审阅与维护**：在这一章中，我们围绕着如何"像维护程序项目一样维护`Markdown`项目"的议题展开了一系列的讨论。首先，我们介绍了一款可以让人们更专注于文字内容审阅和修改的`Markdown`编辑器。然后，考虑到`Markdown`的应用目前尚不够普及的现实问题，为了让更多的人参与作品的审阅，我们为大家介绍了一款专用于转换标记语言格式的工具。最后，为了从时间维度上对项目的修改进行管理，我们也对如何用git版本控制系统对`Markdown`项目进行管理和维护，做了一个基本介绍。
- **第6章 Markdown的其他应用**：在这一章中，我们为大家介绍了如何用`Markdown`制作演示文稿、线上电子书以及撰写博客。集中展示了`Markdown`作为一种写作方式的广泛适用性。
- **附录A Makefile简易教程**：在这篇附录中，我们为大家介绍了`Makefile`文件的基本写法，以便搭配第5章中介绍的格式转换工具批量地将`Markdown`文档转换成其他格式的文档。
- **附录B 了解一下Node.js**：考虑到本书介绍的gitbook和Hexo都要基于Node.js运行环境来部署，而这个运行环境如今已经形成了如此庞大的软件生态系统，我认为有必要用一篇附录专门介绍一下Node.js以及它的安装和配置。

开源运动简介

在读者正式开始阅读本书之前，我还希望对开源运动做一个简单的介绍。从本质上来说，软件的开源事实上是针对软件工程问题提出的一个解决方案。而说起软件工程这档事，我相信计算机和软件工程专业的学生应该都不陌生，我们早年见都背下来过一些流水线式的项目开发流程。首先是在项目定义阶段要做可行性分析、需求分析这些事，再来进入到开发阶段要做概要设计、详细设计、设计实现等步骤，最后是维护阶段的运行与维护。仿佛软件开发就像《摩登时代》里的工厂流水线，分工明确、井然有序。目的是让程序员成为流水线上的工人，使他们成为生产机器中的一个螺丝钉，无需创意、无需个性，只要够熟练就行。很多大型企业的开发项目也确实是按照这个路数走的，很多程序员被戏称"码农"也正是这个原因。

但是，等我工作了若干年之后再来看这套工程管理模型，感觉这基本上就是个"计划经济"。首先，绝大部分软件在开发初期根本不会有那么多人参与，通常是两三个人要做所有的事情。分那么多阶段，那么多工序是没有意义的。再来，就算是有了一定规模的公司，他们会让很多人参与一个项目，往往都是为了维护已有的软件，程序员的主要任务是维护该软件的版本，并在此基础上开发新的版本，在这种情况下，他们其实已经有了现成的开发框架，这些人只需要根据特定的需求将该框架填充成具体的专用软件即可。对于原框架来说，这更像是增加了一个特性分支。例如说，JetBrain团队开发了一款名为IntelliJ IDEA的通用IDE，而Android Studio则又是专用于Android开发的IDE，它就是基于IntelliJ IDEA开发出来的。我们可以将它视为IntelliJ IDEA项目的一个分支。这更像是某种意义上的维护工作，它的可行性，需求是一目了然的，也不需要概要设计，只需要按照其原有的插件体系把功能实现即可。然后，bug修复才是这个项目的主要工作。所以，如何让那么多人一块有效地，有序地发现bug、报告bug、解决bug成为了主要问题。

上世纪的七十年代和八十年代爆发了两次所谓的软件危机[1]，那时候的许多软件项目都出现了预算超支、发布时间严重拖延、质量管理缺失等问题。大量的项目因此而失败，问题很严重，以致于北约这样的组织都要专门开会来讨论这个问题。但这些高高在上的人物讨论出来的东西就是我们上面所说的软件工程理论。按照《人月神话》作者佛瑞德·布鲁克斯（Frederick P. Brooks）的说法，这需要大量的银弹、人员来支撑，只有大型企业，科研机构才能做到。当然对于这些机构来说，这套理论确实能解决一些问题。尤其在互联网时代来临之前，这似乎也是我们唯一的选择。

但大型机构都存在官僚主义的问题，组织繁杂、沟通成本高昂、开发效率低下，随着时间的推移它们往往都会离人们的实际需求越来越远，就像是基督教的那些中世纪大教堂，高高在上、脱离现实地定期发布信息，内容庞杂而滞后，对于其周边的、下游的开发者和中小软件开发是毫无帮助。于是Linux之父林纳斯·托瓦兹（Linus Torvalds）在独自开发Linux内核的过程中走出了一条新的道路：开放源码、社区协作。简单来说，就是由软件项目的创始人开发出一个不成熟的初始版本，然后将其丢到一个开发者社区中，让其在开发者自发性的修改和分享中自然生长。最后，项目创始人会根据其生长情况将自己认可的部分纳入到项目的主分支中。这种乱中有序的组织形式让Linux项目获得了巨大的成功，给软件开发的工程管理提供了一种新的**实践经验**。

无独有偶，上世纪九十年代末期，网景公司[2]在与微软公司的浏览器大战中败下阵来，面临着公司的生存危机。他们决定试试开源的方式。埃里克·雷蒙（Eric Raymond）就是网景公司当时的策略顾问，他在帮助网景公司的过程中根据自己的新的写出了他那本闻名天下的代表作：《大教堂与集市》。这本书为开源运动奠定了**理论基础**，它系统阐述了互联网条件下的协作模式，同行审评的优势，回答了《人月神话》中提出的银弹问题，人员管理成本问题。如今，微软、苹果这些曾经的大教堂都纷纷加入了开源领域。开源作为软件工程的另一种组织形式已经毋庸置疑。

最后需要提醒的是，开源运动和理查德·斯托曼（Richard Stallman）领导的自由软件运动[3]不是一回事。开源运动更多的是一种软件工程的管理方式，虽然也强调开放源码、免费分享的自由精神，但并没有太强烈的道德要求。而自由软件运动则更像是一种宗教性的意识形态运动，他们对于“确保用户使用软件的自由”有着一种近乎苛刻的道德要求，譬如，他们会要求所有基于自由软件开发的产品都不仅要开放源码，还必须要允许用户修改该产品软件的源码，或变更其硬件的使用方式，让用户真正地享有“自由”，这难免让人觉得有一些乌托邦式的理想主义。而在我个人看来，如此激烈的主张在客观上反而会给源代码的分享带来了不少的阻力。

---

1. 注释：请参考https://zh.wikipedia.org/wiki/软件危机↩
2. 注释：请参考https://zh.wikipedia.org/wiki/网景↩
3. 注释：请参考https://zh.wikipedia.org/wiki/自由软件↩