使用 Quarto 编写学术和技术文档

Strik0r

2025-10-27

目录

前言

关于本书

本项目主要参考 Quarto Community (2024) 的官方文档，并结合自己的实际需求进行了一些定制化修改。

This is a Quarto book. To learn more about Quarto books visit https://quarto.org/docs/books.

主要任务

Markdown 作为一种轻量级的标记语言，在日常的文档编辑中被广泛使用。然而，Markdown 的功能有些“简陋”，并不能满足学术文档、技术文档的较复杂的排版需求。

LaTeX 作为一种专业的排版系统，在学术文档、技术文档的排版中被广泛使用。然而，LaTeX 的语法较为复杂，学习成本较高，对于初学者来说并不友好，即使熟练使用，也会花费大量的时间，效率不高。

Quarto 是 Markdown 的诸多扩展之一，在保持 Markdown 的简洁、易用的基础上，提供了丰富的功能以满足学术文档、技术文档的排版需求。本项目的工作就是把我们之前基于 LaTeX 的文档内容无缝地迁移到 Quarto 上来，从而实现**一份源代码，各种输出格式**的高效内容编辑工作流。

目标读者 & 前置知识储备

本书不适合在日常写作中不涉及大量公式、或更偏向于 Markdown 特性的读者，也不适合主要使用 LaTeX 且无需 Markdown 特性的用户，或者仅有 PDF 生成需求、没有 HTML 等格式输出和内容分发需求的人群，以及已经使用 Docusaurus 等其他基于 Markdown 的文档工具的读者。

本书适合已经具备一定 Markdown 和 LaTeX 基础，并且希望结合两者优势、提升写作效率的读者。需要使用 LaTeX 的读者，通常会在日常写作中涉及大量公式排版、TikZ 绘图，并希望通过各类 LaTeX 环境来对文档结构进行更灵活的划分。而对 Markdown 有需求的读者，则更希望凭借其简洁的语法，实现高效的图文混排、列表编辑[1]、代码块编辑，并用 `callout` 特性生成醒目的提示信息等内容。

- 需要对 Markdown 有一定的了解，熟悉 Markdown 的语法，能够使用 Markdown 编写文档。
- 需要读者完成 LaTeX 的环境配置，但这并不是必须的。没有完成 LaTeX 配置的读者可以参考 Quarto 的官方文档完成配置。如果你不确定你的 LaTeX 环境是否已经配置完成，可以尝试在系统终端中执行如下命令：

[1] 这里的列表指的是无序列表和有序列表

```
$ pdflatex --version
$ xelatex --version
$ lualatex --version
$ latexmk --version
```

如果上述命令都能正确输出版本信息，则说明你的 LaTeX 环境已经配置完成。否则，请参考 Quarto 的官方文档 (Quarto Community 2024) 或者 这个视频 完成配置。

讲义获取

本系列没有 PDF 版讲义。

第 1 章
环境配置

访问 https://quarto.org/docs/get-started/，点击 `Download Quarto CLI` 按钮下载 `Quarto` 的命令行工具，浏览器会自动下载适合你的操作系统的安装包[1]。接下来，在该网站的 Step 2 中选择你喜欢的工具，我们这里使用 VS Code (Cursor) 作为编辑器。

[1]如果你不放心的话，也可以在下面的表格里手动选择安装包。

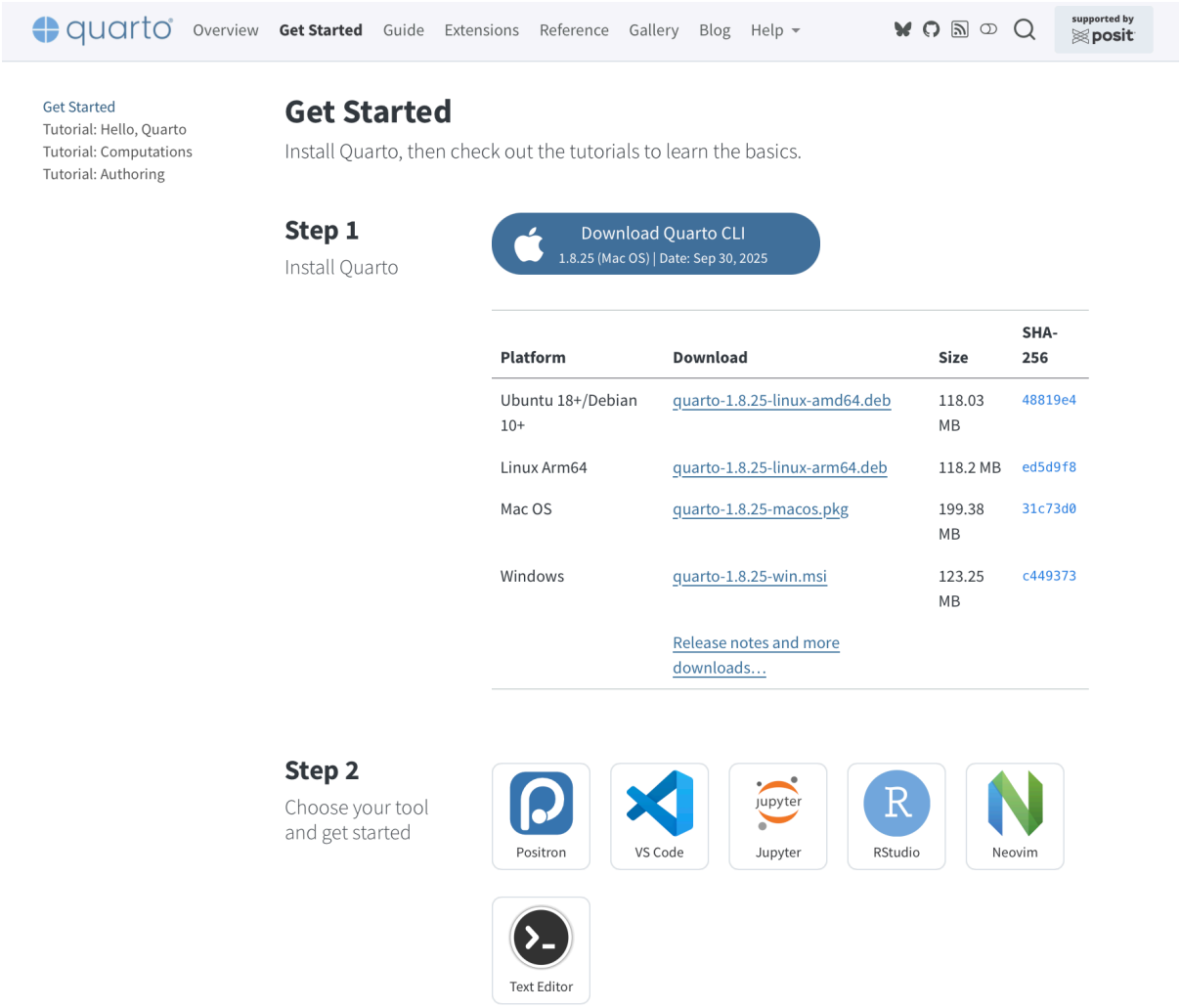


图 1.1: Quarto offcial site > Get Started

下载到 `Quarto CLI` 的安装包后，根据你的操作系统上的安装导引完成安装。安装完成后，打开终端，输入 `quarto --version` 命令，如果显示 `Quarto` 的版本信息，则说明安装成功。

```
$ quarto --version
1.8.25
```

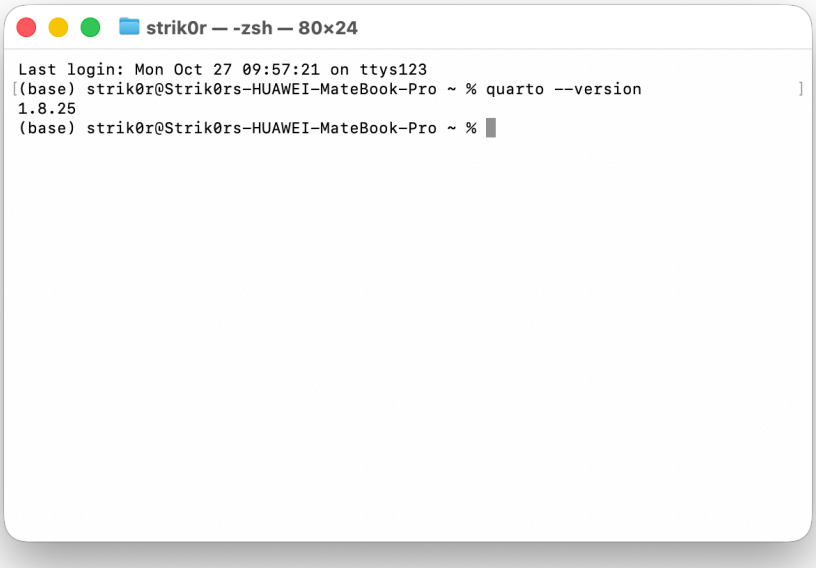


图 1.2: 成功安装 Quarto CLI

接下来，打开 VS Code，在插件市场中搜索 Quarto 插件，安装后重启 VS Code 即可。

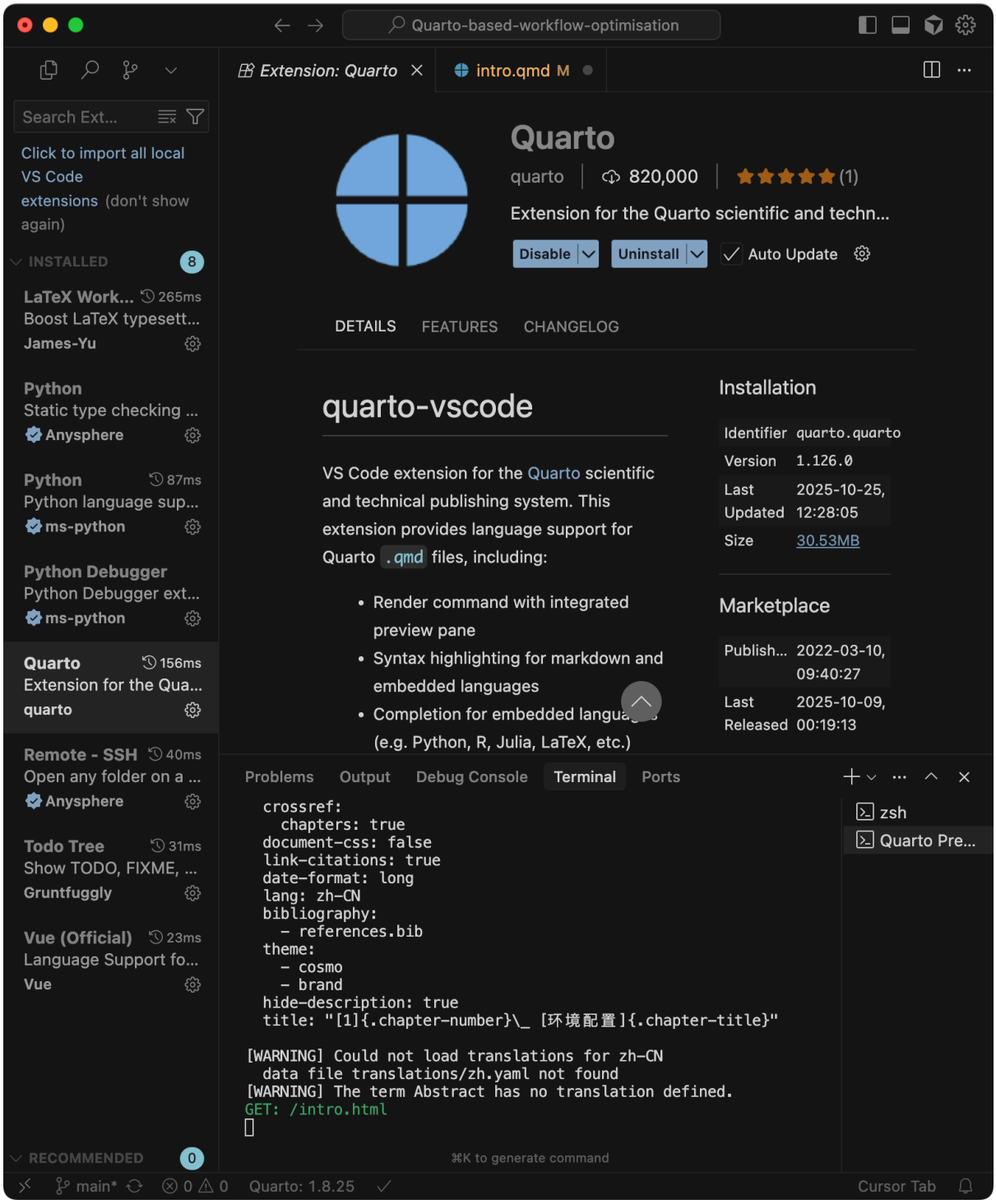


图 1.3: 插件市场 > Quarto

第 2 章
项目初始化与项目结构

在上一节中，我们完成了 `Quarto` 的安装与配置。接下来，我们将开始创建我们的项目。在 VS Code 中，调用命令面板，快捷键是 `Cmd Shift P`[1]。在命令面板中，输入 `Quarto: New Project` 命令，选择项目模板和项目路径，我们这里选择 `Book` 模板。

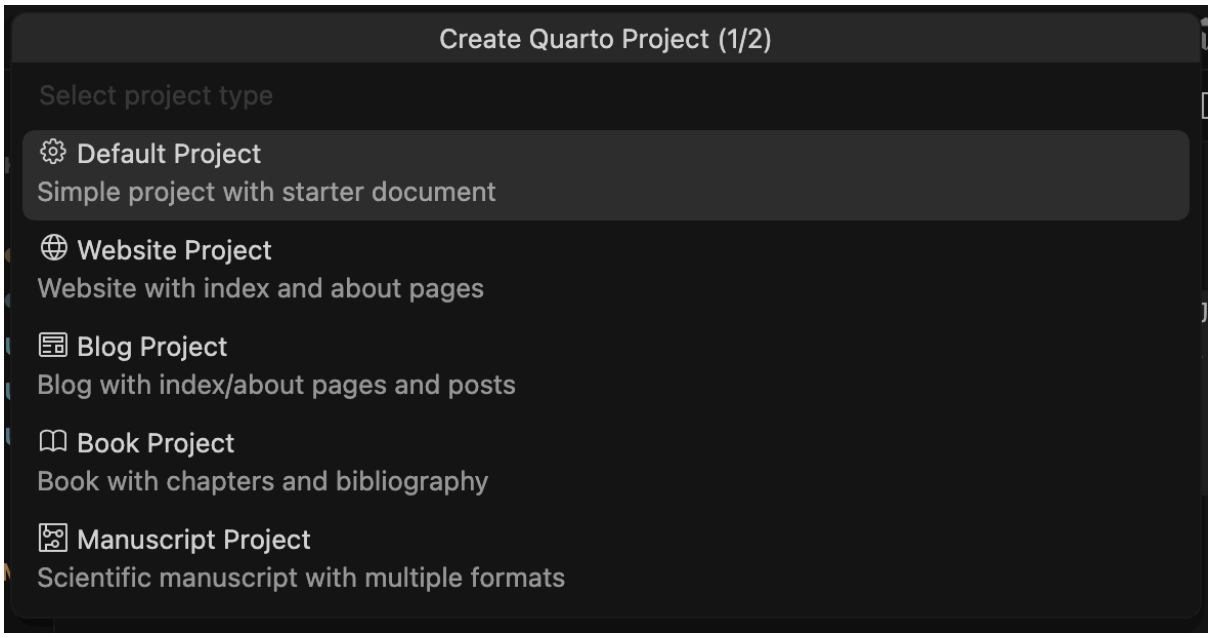


图 2.1: 命令面板 > New Project

新建项目以后，我们可以看到 `Quarto` 为我们生成了一个项目的基本结构：

```
_quarto.yml
cover.png
index.qmd
```

[1]对于 Windows 用户，快捷键是 `Ctrl Shift P`。

```
references.bib
references.qmd
intro.qmd
summary.qmd
```

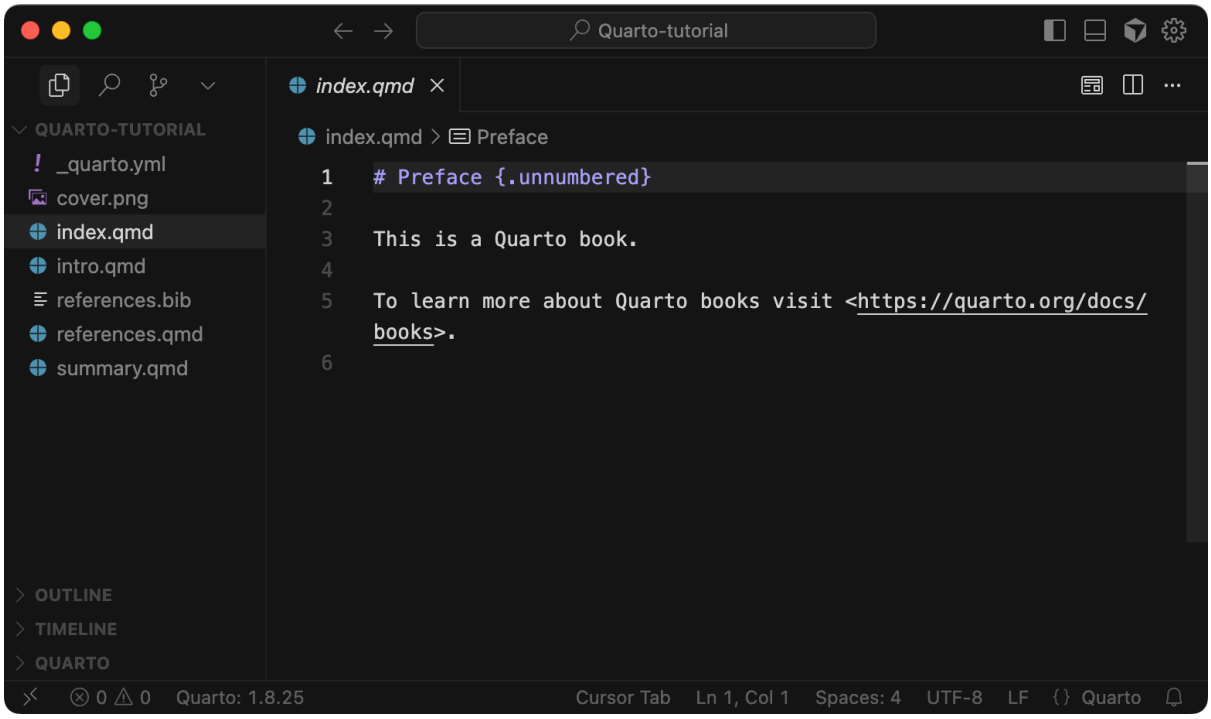


图 2.2: 项目结构

- `_quarto.yml` 是 `Quarto` 的配置文件，我们的大多数项目配置都在这个文件中进行。
- `cover.png` 是封面图片，它实际上只是一个占位符，我想不会有人恶趣味到选择用这个东西作为自己的书籍的封面。
- `index.qmd` 是书籍的首页，首页中应该包含前言、关于本书、致谢等内容。
- 文件名为 `references` 的两个文件是与参考文献有关的。`references.bib` 是参考文献的 BibTeX 文件，用于生成参考文献列表；`references.qmd` 是参考文献的列表页，用于生成参考文献列表。
- 剩余的 `intro.qmd` 和 `summary.qmd` 是项目初始化中生成的两个示例章节。

在 VS Code 的顶部菜单栏选择 `Terminal > New Terminal` 打开终端（快捷键是 `Control Shift` `），执行以下命令以预览项目：

```
$ quarto preview
```

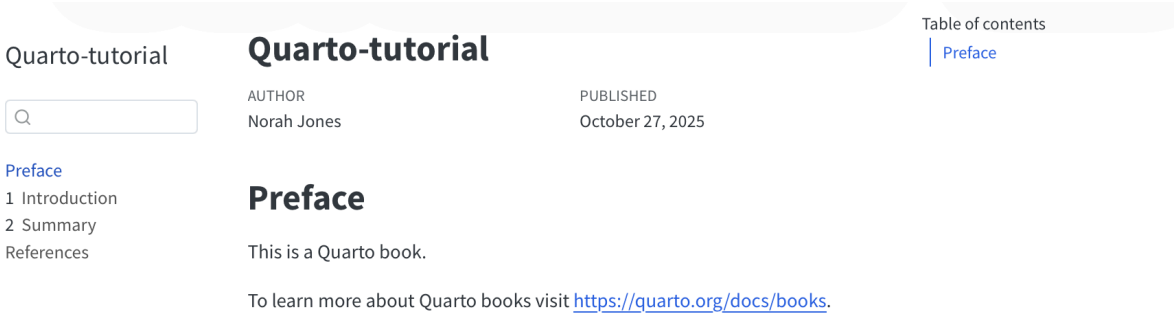


图 2.3: 预览项目

你可以在 `_quarto.yml` 文件中修改项目的语言：

```
lang: zh-CN
```

第 3 章

Markdown 基础

Markdown 是一种轻量级的标记语言，用于将纯文本转换为格式化的文档。它最初由 John Gruber 和 Aaron Swartz 于 2004 年创建，旨在简化文本格式化过程。Markdown 的语法非常简单，易于学习和使用，因此被广泛应用于各种文档、博客、论坛和代码编辑器中。本章节主要介绍了 Markdown 的基础知识。

3.1 标题

Markdown 的标题使用 # 符号来表示，# 的数量表示标题的级别，# 越多，标题的级别越低。例如：

```
# 一级标题
## 二级标题
### 三级标题
#### 四级标题
##### 五级标题
```

> ⚠ 标题级别不建议超过 4 级别，否则可能对 PDF 版本的 LaTeX 渲染产生影响。通常来说，一级标题表示 `chapter`，二级标题表示 `section`，三级标题表示 `subsection`，四级标题表示 `subsubsection`，照这样下去，五级标题就是 `paragraph`。
> 但是经过测试，渲染含有五级标题的 PDF 文档时会报错。

3.2 段落

Markdown 的段落使用空行来表示，段落之间使用空行来分隔。

```
这是一个段落，
代码中的换行会被解释为一个空格。

这是另一个段落，
同样，代码中的换行只能被认为是空格。
```

这是一个段落，代码中的换行会被解释为一个空格。

这是另一个段落，同样，代码中的换行只能被认为是空格。

3.3 列表

Quarto Markdown 支持多种列表类型，也支持混用不同样式的列表。

```
- 无序列表项 1
- 无序列表项 2
- 无序列表项 3
```

- 无序列表项 1
- 无序列表项 2
- 无序列表项 3

```
1. 有序列表项 1
2. 有序列表项 2
3. 有序列表项 3
```

1. 有序列表项 1
2. 有序列表项 2
3. 有序列表项 3

Quarto Markdown 也支持大写字母列表，这一列表使用大写拉丁字母作为标签，非常适合用于选择题的排版：

```
A.  大写字母列表项 1
A.  大写字母列表项 2
A.  大写字母列表项 3
```

A. 大写字母列表项 1
B. 大写字母列表项 2
C. 大写字母列表项 3

大写字母列表项的标签有两个空格。

```
A. 只有一个空格
A.  有两个空格 <!-- 正确 -->
```

只打一个空格是不能触发 Quarto 的大写字母列表的：

```
A. 只有一个空格
A. 只有一个空格
```

A. 只有一个空格 A. 只有一个空格
注意到上述代码的渲染结果是一个段落，而不是一个以大写字母为标签的列表。

类似地，还可以将小写字母和复选框作为列表项的标签：

```
a.  小写字母列表项  <!-- 两个空格 -->
- [ ] 复选框列表项
- [x] 复选框列表项（已勾选）
```

a. 小写字母列表项

☐ 复选框列表项
☒ 复选框列表项（已勾选）

第 4 章

配置 LaTeX

本章节详细介绍了本项目中的 LaTeX 相关配置，包括 PDF 输出格式设置、文档类配置、页面布局、字体设置以及各种宏包的使用。这些设置确保了生成的 PDF 文档具有专业的外观和良好的排版效果。

4.1 PDF 输出格式配置

在 Quarto 项目的 `_quarto.yml` 配置文件中，通过 `pdf` 相关设置可以灵活地指定 PDF 输出的排版样式、文档类、页面尺寸和各种格式参数，从而实现专业而定制化的排版效果。

本项目使用 `ctexbook` 作为 PDF 输出的文档类，这是专门为中文文档设计的 LaTeX 文档类：

```yaml
pdf:
  documentclass: ctexbook
  fontsize: 10pt
  pdf-engine: xelatex
```

`ctexbook` 是专门为中文文档设计的 LaTeX 文档类，支持中文排版，自动处理中文字体、标点符号等。

`fontsize: 10pt` 设置基础字体大小为 10 点，适合学术文档阅读。

`xelatex` 使用 `xelatex` 作为 PDF 生成引擎，`xelatex` 对 Unicode 和现代字体支持更好，特别适合处理中文字符和自定义字体。

4.2 页面布局设置

项目配置了标准的 A4 页面尺寸和合适的边距：

```yaml
geometry:
  - paperwidth=210mm
  - paperheight=297mm
  - top=36.8mm
  - bottom=31.8mm
```

```
  - left=25.4mm
  - right=25.4mm
```

- **页面尺寸**: 210mm × 297mm (A4 标准)
- **上边距**: 36.8mm，为页眉和章节标题留出空间
- **下边距**: 31.8mm，为页脚留出空间
- **左右边距**: 25.4mm，确保文本不会过于靠近页面边缘

4.3 字体配置

在 `_quarto.yml` 文件的 `pdf` 配置中，`include-in-header` 用于向生成的 `LaTeX` 文档头部插入自定义的 `LaTeX` 代码片段。这些代码会在每次编译 PDF 时自动插入到 `LaTeX` 源文件的 `\begin{document}` 之前，常用于字体设置、引入额外的宏包、定义自定义命令或调整全局排版风格等。例如可以用它为整个文档设定等宽字体、插入自定义标题样式、统一公式渲染模式等配置。从而更灵活地微调最终输出文档的外观与功能。

```
include-in-header:
  text: |
    \setmonofont{JetBrains Mono}
```

使用 JetBrains Mono 作为等宽字体，这是一个专为开发者设计的现代等宽字体，具有良好的可读性。你可以访问 https://www.jetbrains.com/lp/mono/ 了解更多关于 JetBrains Mono 的信息和下载该字体。

4.4 数学公式设置

```
include-in-header:
  text: |
    \everymath{\displaystyle}
```

将所有行内数学公式显示为显示模式，使公式更加清晰。

4.5 自定义符号

在使用 LaTeX 编写文档时，时常会需要自定义一些符号，并在多个文件中使用它们，例如，把 `\mathrm{i}` 简化为 `\im`，可以极大提高代码的紧凑程度。在 LaTeX 中，这涉及到两个命令：

```
\newcommand{\im}{\mathrm{i}}
\renewcommand{\im}{\mathrm{i}}
```

其中，第一个命令 `\newcommand` 用于定义新的命令，第二个命令 `\renewcommand` 用于重新定义已有的命令。需要注意的是，对已经定义过的命令进行重新定义时，需要使用 `\renewcommand` 命令，而不是 `\newcommand` 命令，否则将导致报错。也就是说，对于 LaTeX 代码，我们需要保证**每一个自定义命令只调用一次 `\newcommand` 命令，所有后续更改都通过 `\renewcommand` 命令完成**。

对于纯 LaTeX 项目来说，这两个命令已经足够使用，我们可以在文档的导言区中插入相关自定义命令的代码。后续所有正文章节中需要使用的 LaTeX 命令，都已经在导言区中被一次性导入了。

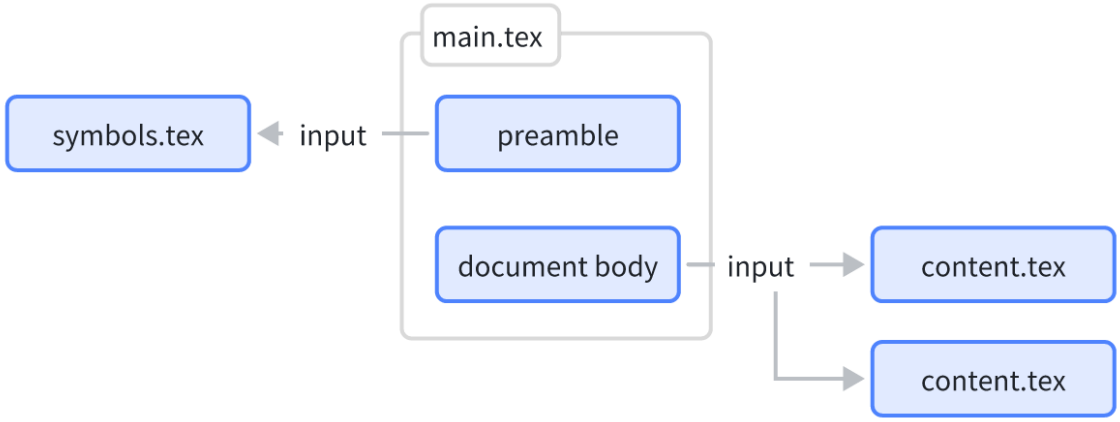


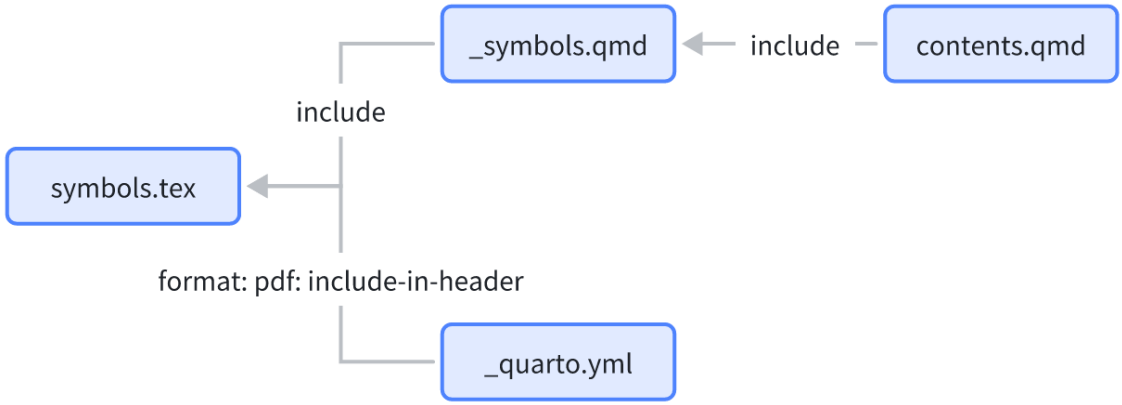
图 4.1: 传统 LaTeX 项目中复用自定义命令的方法

然而，在 `Quarto` 项目中，由于 MathJax 在 HTML 中是基于页面局部渲染的，不像 LaTeX 编译时拥有全局状态，所以每个页面都必须“看到”宏定义才能正确解析。也就是说，**不同.qmd 文件之间是相互独立的**，对于每一个独立的.qmd 文件，都需要“导入”一次已有的自定义命令代码，否则这些自定义命令不会被 MathJax 识别。因此，如果将自定义命令的配置代码放在 `_symbols.qmd` 文件中，则每一个需要用到这些命令的文件都需要使用 `{{< include _symbols.qmd >}}` 命令导入该文件，否则这些自定义命令不会被 MathJax 识别。

图 4.2: Quarto 项目中复用自定义命令的方法

这就引出了矛盾：对于基于 LaTeX 的 PDF 输出来说，一次导入已经足够；但是对于 HTML 输出来说，第二次及其以后的导入都将对“已经定义过的命令”重新调用 `\newcommand` 命令，从而导致 PDF 输出端报错。也就是说，PDF 输出只需要导入一次，而 HTML 输出需要导入多次。

解决方案是，我们把针对两种输出的自定义命令代码分开配置，这需要使用 _quarto.yml 文件中的 `include-in-header` 配置项，以及 Quarto 的 `.content-hidden` 属性。



首先我们创建一个 symbols.tex 文件，在该文件中输入自定义命令有关的代码，例如：

```
\newcommand{\im}{\mathrm{i}}
\newcommand{\jm}{\im}
\newcommand{\rank}{\operatorname{rank}}
\DeclareMathOperator{\Res}{Res}
\DeclareMathOperator{\argmax}{argmax}
\DeclareMathOperator{\argmin}{argmin}
\DeclareMathOperator{\tr}{tr}
\renewcommand{\Re}{\operatorname{Re}}
\renewcommand{\Im}{\operatorname{Im}}
```

然后在 _quarto.yml 文件中添加以下配置：

```
include-in-header:
  text: |
    \input{tex_utils/symbols.tex}
```

这样，在 PDF 输出时，这些自定义命令就会被正确导入。

接下来我们对 HTML 输出进行配置，新建一个 _symbols.qmd 文件，在该文件中导入 symbols.tex 文件，并使用 `.content-hidden` 属性隐藏该文件的内容，设置这部分导入对 PDF 输出不可见：

```
! _quarto.yml    _symbols.qmd
tex_utils > _symbols.qmd
1  ::: {.latex .content-hidden when-format="pdf"}
2  {{< include tex_utils/symbols.tex >}}
3  :::
4
```

上述代码中，`.latex` 表示这个块内的代码为 LaTeX 代码，`.content-hidden when-format="pdf"` 表示这部分代码不作用于 PDF 输出。

在后续使用中，在**每一个**需要使用这些自定义命令的.qmd 章节文件中，导入 _symbols.qmd 文件：

```
! _quarto.yml   _symbols.qmd   sample-chapter.qmd
sample-chapter.qmd > Sample Chapter
1  # Sample Chapter
2
3  {{< include tex_utils/_symbols.qmd >}}
4
5  以下数学命令应该可以正常使用：
6  $\operatorname{rank}\mathbf{A}$, $\rank\mathbf{A}$,
   $\displaystyle\Res_{z = z_0} f(z)$, $\argmax_{x \in X} f(x)$,
   $\argmin_{x \in X} f(x)$, $\tr\mathbf{A}$.
```

预览效果与 PDF 输出效果分别如下，可以看到，自定义 LaTeX 命令在 PDF 输出和 HTML 输出中都能够正确识别。

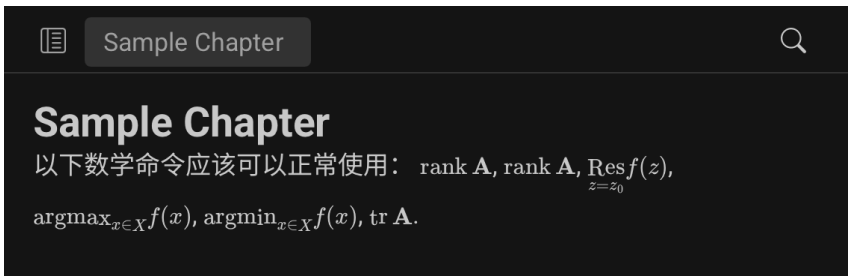


图 4.3: HTML 输出

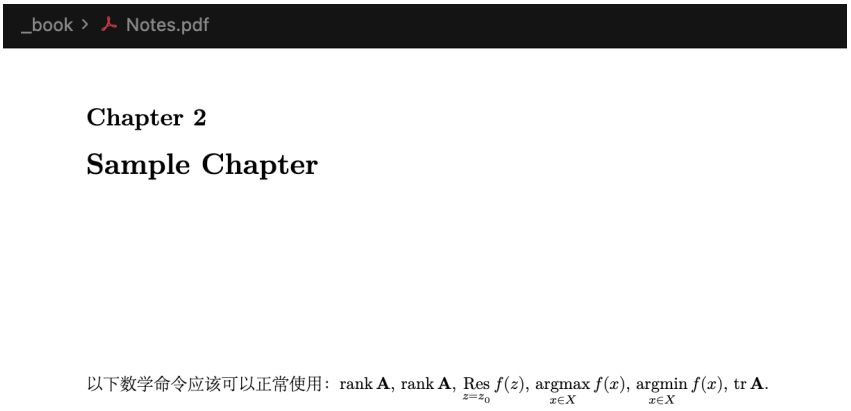

_book › Notes.pdf
Chapter 2
Sample Chapter
以下数学命令应该可以正常使用：$\operatorname{rank}\mathbf{A}$, $\operatorname{rank}\mathbf{A}$, $\operatorname*{Res}_{z=z_0} f(z)$, $\operatorname*{argmax}_{x\in X} f(x)$, $\operatorname*{argmin}_{x\in X} f(x)$, $\operatorname{tr}\mathbf{A}$.

图 4.4: PDF 输出

第 5 章
TikZ 相关设置

TikZ 是一种强大的绘图工具，可以用于绘制各种复杂的图形，在我的 LaTeX 文档中有大量的图表都是由 TikZ 绘制的。但是对 TikZ 的支持是我在工作流迁移中的最大难点。

本方案的缺点与不足

我尝试了所有能匹配 `TikZ` 关键词的 `Quarto` 扩展，包括 `diagram`、`imagify` 和 `tikz`，也查阅了 Stack Overflow 上的有关讨论，但是这些方案都不能成功就地编译 `TikZ` 代码并将绘图结果作为图像输出。并且这些项目的 GitHub 仓库都已经有十几个月没有更新了，也许是烂尾了？再经过长达 12 个小时的爆肝之后，我于 2025 年 10 月 26 日凌晨 3 点放弃了这看起来有些不切实际的尝试。

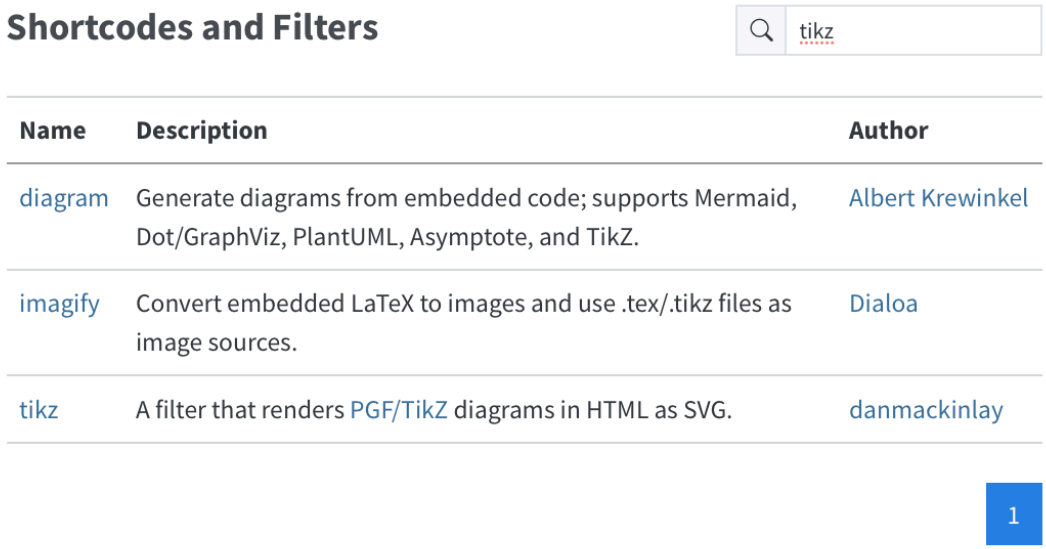


图 5.1: Quarto 扩展搜索结果

我们的方案采用的是一种比较“笨”的方法，即：将 `TikZ` 代码编译为 `PDF` 文件，然后将其转换为 `PNG` 图像，最后将 `PNG` 图像插入到 `Quarto` 文档中。这也算是一种无奈之举了。

悬而未决的问题

如果你有办法解决这个问题，请通过 strik0rium@gmail.com 联系我！

5.1 准备工作

在项目的根目录新建一个名为 `tikzplots` 的目录和三个子目录，其中 `tex/` 目录用于存放 `TikZ` 代码，`pdf/` 目录用于存放 `PDF` 文件，`png/` 目录用于存放 `PNG` 文件。

```
tikzplots/
  tex/
  pdf/
  png/
```

5.2 工作流

在 tikzplots/tex/ 目录下新建一个名为 your_filename.tex 的文件，并输入你的 TikZ 代码。建议选择 standalone 文档类，并进行绘图代码的编写。编写完成后，在终端中执行以下命令以编译 TikZ 代码，并将其转换为 PNG 图像，最后将 PDF 文件和 PNG 图像移动到 tikzplots/pdf/ 和 tikzplots/png/ 目录中。

```
cd tikzplots/tex
pdflatex -interaction=nonstopmode your_filename.tex
sips -s format png -Z 1200 your_filename.pdf --out ../png/your_filename.png
mv your_filename.pdf ../pdf/
rm -f your_filename.aux your_filename.log your_filename.out
```

> **格式转换工具的选择**
>
> 在上述命令中，我们使用了 sips 命令来转换 PDF 文件为 PNG 图像。sips 是 macOS 自带的格式转换工具，也可以使用其他工具。

> **最佳实践**
>
> 你可以根据你选用的工具和具体的执行方案，编写一个 shell 脚本来自动化你的工作流，你的 AI 助手会很乐意帮你完成这项工作。

5.3 完整示例演示

首先创建几个相关的目录：

```
$ mkdir tikzplots
$ cd tikzplots
$ mkdir -p tex pdf png
```

接下来，在 tikzplots/tex/rlc-circuit.tex 文件中输入以下代码：

```
\documentclass{standalone}
\usepackage{tikz}
\usetikzlibrary{arrows, shapes, calc, positioning}
\usepackage{pgfplots}
\usepackage{circuitikz}
\usepackage{amsmath, amssymb, amsfonts, amsthm, mathtools, mathrsfs}
\pgfplotsset{compat=1.18}

\begin{document}
\begin{circuitikz}[scale=0.7]
    \draw (0, 0) to[vsource=$v_\text{s}(t)$] (0, 3)
        to[R=1<ohm>] (4, 3)
        to[L=1<henry>] (4, 0)
```

```
        to[C=1<farad>] (0, 0);
\end{circuitikz}
\end{document}
```

然后执行这些命令编译 TikZ 代码，并将其转换为 PNG 图像，最后将 PDF 文件和 PNG 图像移动到 tikzplots/pdf/ 和 tikzplots/png/ 目录中：

```
$ cd tex
$ pdflatex -interaction=nonstopmode rlc-circuit.tex
This is pdfTeX, Version 3.141592653-2.6-1.40.26 (TeX Live 2024) (preloaded format=pdflatex)
...
Output written on rlc-circuit.pdf (1 page, 31309 bytes).
Transcript written on rlc-circuit.log.
$ sips -s format png -Z 1200 rlc-circuit.pdf --out ../png/rlc-circuit.png
$ mv rlc-circuit.pdf ../pdf/
$ rm -f rlc-circuit.aux rlc-circuit.log rlc-circuit.out
```

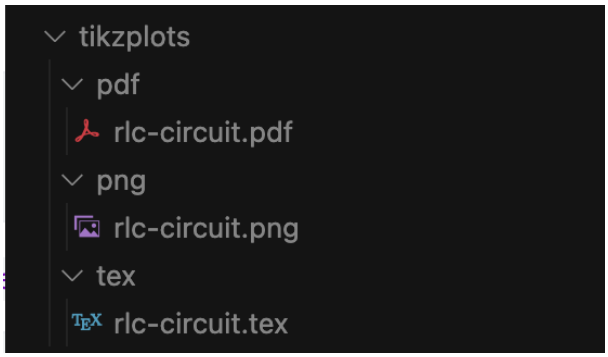


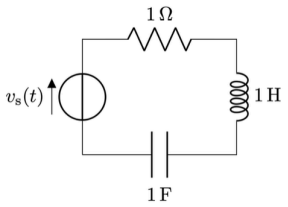
图 5.2: 上述命令行命令的执行效果

接下来在项目正文中插入新生成的 PNG 图像。

```
6
7   RLC circuits are a type of electrical circuit that consists of a
    resistor, an inductor, and a capacitor connected in series or
    parallel. They are used to filter signals and to store energy. Here
    is an example of an RLC circuit:
8
9   ![An RLC circuit](tikzplots/png/rlc-circuit.png){width="50%"}
```

(a) 插入 PNG 图像

RLC circuits are a type of electrical circuit that consists of a resistor, an inductor, and a capacitor connected in series or parallel. They are used to filter signals and to store energy. Here is an example of an RLC circuit:



An RLC circuit

(b) 预览效果

图 5.3: 在项目正文中插入新生成的 PNG 图像

第 6 章
HTML 输出主题

6.1 设置 HTML 输出字体

本节我们以 Google 的开源字体 Roboto 和 Noto Sans SC，以及 JetBrains 的开源等宽字体 JetBrains Mono 为例，介绍文档 HTML 输出字体的设置。文档字体需要通过 css 文件来设置，首先，新建一个 custom.css 文件，在该文件中设置文档字体：

```
/* custom.css */
@import url('https://fonts.googleapis.com/css2?family=Noto+Sans:wght@400;700&display=swap');
@import url('https://fonts.googleapis.com/css2?family=Noto+Sans+SC:wght@400;700&display=swap');
@import url('https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap');
@import url('https://fonts.googleapis.com/css2?family=JetBrains+Mono:wght@400;700&display=swap');

body {
  font-family: "Roboto", "Noto Sans", "Noto Sans SC", sans-serif;
}

code, pre {
  font-family: "JetBrains Mono", monospace !important;
}
```

上述代码设置了 `font-family` 列表，在进行 HTML 渲染时，待渲染的字符会逐级向下地从 `font-family` 列表中查找字体，如果找到则使用该字体，否则继续向下查找。上述代码将文档的英文字体设置为 Roboto，由于中文字符并不包含在 Roboto 中，因此汉字会向下查找到 Noto Sans SC。此外，我们还设置了作用于代码块中的等宽字体为 JetBrains Mono。

在定义 `font-family` 列表时，建议**将英文字体放在中文字体之前**，这是因为中文字体的字符集更大，将中文字体放在前面会导致英文字符直接匹配中文字体，从而按照中文字体的样式渲染。然而，由于中文字体主要是为了适应方块字的特点而设计的，因此就算是同一字体集或者本就设计为共同使用的字体集（如 Roboto 和 Noto Sans 系列），中文字体中的英文字符也会和英文字体集中的字符在紧凑程度方面有较为细微的差别。

为了使该配置对项目生效，在 _quarto.yml 文件中添加以下配置：

```yaml
format:
  html:
    css: custom.css
```

6.2 深色模式

6.2.1 将 cosmo 主题应用于深色模式

6.2.2 代码块

第 7 章

Mermaid 基础

第 8 章
内容渲染与分发

8.1 渲染

使用如下命令渲染内容：

```
$ quarto render
```

8.2 把内容发布到 GitHub Pages

新建一个 GitHub 仓库，然后执行

```
$ quarto publish gh-pages
```

参考文献

Quarto Community. 2024. 《Quarto Guide》. https://quarto.org/docs/guide/.