

**Scripst 的使用方法**

**report 模板**

AnZrew

2025-02-27

**摘要：** Scripst 是 Typst 语言的模板，用来生成简约的日常使用的文档，以满足文档、作业、笔记、论文等需求

**关键词：** Scripst; Typst; 模板

## 目录

<b>1 使用 Scripst 排版 Typst 文档</b>	<b>1</b>
1.1 使用 Typst	1
1.2 使用 Scripst	1
<b>2 模板参数说明</b>	<b>3</b>
2.1 template	3
2.2 title	4
2.3 info	4
2.4 author	4
2.5 time	4
2.6 abstract	5
2.7 keywords	5
2.8 preface	5
2.9 font_size	5
2.10 contents	6
2.11 content_depth	6
2.12 matheq_depth	6
2.13 lang	6
2.14 body	6
<b>3 模板效果展示</b>	<b>7</b>
3.1 文档开头	7
3.2 前言	7
3.3 目录	7
3.4 字体与环境	7
3.4.1 字体	7
3.4.2 环境	7
3.4.3 列举	9
3.4.4 引用	9

3.4.5 链接 . . . . .	9
3.4.6 超链接与文献引用 . . . . .	10
3.5 #newpara()函数 . . . . .	10
3.6 countblock . . . . .	10
3.6.1 countblock 的新建与注册 . . . . .	10
3.6.2 countblock 的使用 . . . . .	11
<b>4 结语 . . . . .</b>	<b>13</b>

## 前言

Typst 是一种简单的文档生成语言，它的语法类似于 Markdown 的轻量级标记，利用合适的 `set` 和 `show` 指令，可以高自由度地定制文档的样式。

Scripst 是本人编写的 Typst 模板，用来生成简约的日常使用的文档，以满足文档、作业、笔记、论文等需求。

# 一 使用 Scripst 排版 Typst 文档

## 1.1 使用 Typst

Typst 是使用起来比 LaTeX 更轻量的语言，一旦模板编写完成，就可以以类似 Markdown 的轻量标记完成文档的编写。

相比 LaTeX，Typst 的优势在于：

- 极快的编译速度
- 语法简单、轻量
- 代码可扩展性强
- 更简单的数学公式输入
- ...

所以，Typst 对于轻量级日常文档的编写是非常合适的。只需要花费撰写 Markdown 的时间成本，就能得到甚至好于 LaTeX 的排版效果。

可以通过下面的方式安装 Typst：

```
sudo apt install typst # Debian/Ubuntu
sudo pacman -S typst # Arch Linux
winget install --id Typst.Typst # Windows
brew install typst # macOS
```

你也可以在 [Typst 的 GitHub 仓库](#) 中找到更多的信息。

## 1.2 使用 Scripst

在 Typst 的基础上，Scripst 提供了一些模板，用来生成简约的日常使用的文档。

可以在 [Scripst 的 GitHub 仓库](#) 找到并下载 Scripst 的模板。

可以选择 `<> code` → `Download ZIP` 来下载 Scripst 的模板。在使用时，只需要将模板文件放在你的文档目录下，然后在文档的开头引入模板文件即可。

⚠ 要考虑清楚项目的目录结构，以便正确引入模板文件。

```
project/
├── src/
│   ├── main.typ
│   ├── ...
│   └── components.typ
├── pic/
│   └── ...
├── main.typ
├── chap1.typ
├── chap2.typ
└── ...
```

如果项目的目录结构如上所示，那么在 `main.typ` 中引入模板文件的方式应该是：

```
#import "src/main.typ": *
```

这种方法的好处是，你可以随时调整模板中的部分参数。由于编者在编写模板时采用模块化的设计，你可以轻松找到并修改模板中你需要修改的部分。

一个更好的方法是，参考官方给出的[本地的包管理文档](#)，将模板文件放在本地包管理的目录`{data-dir}/typst/packages/{namespace}/{name}/{version}`下，这样就可以在任何地方使用 Scripst 的模板了。

当然，无需担心不能修改模板文件，你可以直接在文档中使用 `#set`，`#show` 等指令来覆盖模板中的部分参数。

例如该模板的应该放在

```
~/ .local/share/typst/packages/local/scripst/1.1.0 # in Linux
%APPDATA%\typst\packages\local\scripst\1.1.0      # in Windows
```

如果是这样的目录结构，那么在文档中引入模板文件的方式应该是：

```
#import "@local/scripst:1.1.0": *
```

这样的好处是你可以直接通过 `typst init` 来一键使用模板创建新的项目：

```
typst init @local/scripst:1.1.0 project_name
```

在引入模板后通过这样的方式创建一个 `report` 文件：

```
#show: scripst.with(
  template: "report",
  title: [Scripst 的使用方法],
```

```
info: [这是文章的模板],
author: ("作者1", "作者2", "作者3"),
time: datetime.today().display(),
abstract: [摘要],
keywords: ("关键词1", "关键词2", "关键词3"),
preface: [前言]
contents: true,
content_depth: 2,
matheq_depth: 2,
lang: "zh",
)
```

这些参数以及其含义见 [小节 2](#)。

这样你就可以开始撰写你的文档了。

## 二 模板参数说明

Scriptst 的模板提供了一些参数，用来定制文档的样式。

```
#let scriptst(
  template: "book", // str: ("article", "book", "report")
  title: "",        // str, content, none
  info: "",         // str, content, none
  author: (),       // array
  time: "",         // str, content, none
  abstract: none,   // str, content, none
  keywords: (),     // array
  preface: none,    // str, content, none
  font_size: 11pt,  // length
  contents: false,  // bool
  content_depth: 2, // int
  matheq_depth: 2, // int: (1, 2)
  lang: "zh",       // str: ("zh", "en", "fr", ...)
  body,
) = {
  ...
}
```

### 2.1 template

参数	类型	可选值	默认值	说明
template	str	("article", "book", "report")	"article"	模板类型

目前 Scriptst 提供了三种模板，分别是 article、book 和 report。

本模板采用 report 模板。

- **article**: 适用于日常文档、作业、小型笔记、小型论文等
- **book**: 适用于书籍、课程笔记等
- **report**: 适用于实验报告、论文等

此外的字符串传入会导致 **panic**: "Unknown template!"。

## 2.2 title

参数	类型	默认值	说明
<b>title</b>	<b>content, str, none</b>	<b>""</b>	文档标题

文档的标题。（不为空时）会出现在文档的封面和页眉中。

## 2.3 info


参数	类型	默认值	说明
<b>info</b>	<b>content, str, none</b>	<b>""</b>	文档信息

文档的信息。（不为空时）会出现在文档的封面和页眉中。可以作为文章的副标题或者补充信息。

## 2.4 author

参数	类型	默认值	说明
<b>author</b>	<b>array</b>	<b>()</b>	文档作者

文档的作者。要传入 **str** 或者 **content** 的列表。

 注意，如果是一个作者的情况，请不要传入 **str** 或者 **content**，而是传入一个 **str** 或者 **content** 的列表，例如：**author**: ("作者",)

会在文章的开头以 **min(#authors, 3)** 个作为一行显示。

## 2.5 time

参数	类型	默认值	说明
<b>time</b>	<b>content, str, none</b>	<b>""</b>	文档时间

文档的时间。会出现在文档的封面和页眉中。

你可以选择用 **typst** 提供的 **datetime** 来获取或者格式化时间，例如今天的时间：

```
datetime.today().display()
```

## 2.6 abstract

参数	类型	默认值	说明
abstract	content, str, none	none	文档摘要

文档的摘要。（不为空时）会出现在文档的摘要和目录页。

建议在使用摘要前，实现定义一个 **content**，例如：

```
#let abstract = [
    这是一个简单的文档模板，用来生成简约的日常使用的文档，以满足文档、作业、笔记、论文等
    需求。
]

#show: scripst.with(
    ...
    abstract: abstract,
    ...
)
```

然后将其传入 **abstract** 参数。

## 2.7 keywords

参数	类型	默认值	说明
keywords	array	()	文档关键词

文档的关键词。要传入 **str** 或者 **content** 的列表。

和 **author** 一样，参数是一个列表，而不能是一个字符串。

只有在 **abstract** 不为空时，关键词才会出现在文档的摘要页。

## 2.8 preface

参数	类型	默认值	说明
preface	content, str, none	none	文档前言

文档的前言。（不为空时）会出现在文档的正文之前。

## 2.9 font\_size



参数	类型	默认值	说明
font_size	length	11pt	文档字体大小

文档的字体大小。默认为 11pt。

参考 length 类型的值，可以传入 pt、mm、cm、in、em 等单位。

## 2.10 contents

参数	类型	默认值	说明
contents	bool	false	是否生成目录

是否生成目录。默认为 false。

## 2.11 content\_depth

参数	类型	默认值	说明
content_depth	int	2	目录的深度

目录的深度。默认为 2。

## 2.12 matheq\_depth

参数	类型	可选值	默认值	说明
matheq_depth	int	1, 2	2	数学公式的深度

数学公式编号的深度。默认为 2。

一般会在不分章节的情况下使用 1，分章节的情况下使用 2。

## 2.13 lang

参数	类型	默认值	说明
lang	str	"zh"	文档语言

文档的语言。默认为"zh"。

可以传入"zh"、"en"、"fr"等语言。

## 2.14 body

在使用 `#show: scripst.with(...)` 时，body 参数是不用手动传入的，typst 会自动将剩余的文档内容传入 body 参数。

## 三 模板效果展示

### 3.1 文档开头

文档的开头会显示标题、信息、作者、时间、摘要、关键词等信息。如该文档的开头所示。

标题、信息、作者、时间内容全部在封面页，封面页不显示页码。

摘要、关键词在摘要页，摘要和目录页用罗马数字编号。

### 3.2 前言

前言会显示在正文之前，如该文档的前言所示。

### 3.3 目录

如果 `contents` 参数为 `true`，则会生成目录亦如该文档的目录所示。

### 3.4 字体与环境

从正文开始，页码计数器会重置，并且以阿拉伯数字编号。

Scripst 提供了一些常用的字体和环境，如粗体、斜体、标题、图片、表格、列表、引用、链接、数学公式等。

#### 3.4.1 字体

这是正常的文本。 This is normal text.

这是粗体的文本。 **This is bold text.**

这是斜体的文本。 *This is italic text.*

安装 CMU Serif 字体以获得更好（类似 LaTeX）的显示效果。

#### 3.4.2 环境

##### 3.4.2.1 标题

一级标题采用中文/罗马数字编号（取决于文档语言），其余级别标题采用阿拉伯数字编号。

##### 3.4.2.2 图片

图片环境会自动编号，如下所示：



图 1 散宝

3.4.2.3 表格

得益于 `tablem` 包，可以用 Markdown 的方式编写表格，如下所示：

```
#figure(  
  three-line-table[  
    | 姓名 | 年龄 | 性别 |  
    | --- | --- | --- |  
    | 张三 | 18 | 男 |  
    | 李四 | 19 | 女 |  
  ],  
  caption: [ `three-line-table` 表格示例 ],  
)
```

姓名	年龄	性别
张三	18	男
李四	19	女

表 1 `three-line-table` 表格示例

```
#figure(  
  tablem[  
    | 姓名 | 年龄 | 性别 |  
    | --- | --- | --- |  
    | 张三 | 18 | 男 |  
    | 李四 | 19 | 女 |  
  ],  
  caption: [ `tablem` 表格示例 ],  
)
```

姓名	年龄	性别
张三	18	男
李四	19	女

表 2 `tablem` 表格示例

可以选择 `numbering: none`, 使得表格不编号，就像前面章节的表格并没有进入全文的表格计数器一样。

3.4.2.4 数学公式

数学公式有行内和行间两种模式。

行内公式： $a^2 + b^2 = c^2$ 。

行间公式：

$$a^2 + b^2 = c^2$$

$$\frac{1}{2} + \frac{1}{3} = \frac{5}{6} \quad (3.1)$$

是拥有编号的。

得益于 `physica` 包，`typst` 本身简单的数学输入方式得到了极大的扩展，并且仍然保留简介的特性：

$$\begin{aligned} \nabla \cdot \mathbf{E} &= \frac{\rho}{\varepsilon_0} \\ \nabla \cdot \mathbf{B} &= 0 \\ \nabla \times \mathbf{E} &= -\frac{\partial \mathbf{B}}{\partial t} \\ \nabla \times \mathbf{B} &= \mu_0 \left( \mathbf{J} + \varepsilon_0 \frac{\partial \mathbf{E}}{\partial t} \right) \end{aligned} \quad (3.2)$$

### 3.4.3 列举

`typst` 为列举提供了简单的环境，如所示：

- 第一项  
- 第二项  
- 第三项

- 第一项
- 第二项
- 第三项

+ 第一项  
3. 第二项  
+ 第三项

1. 第一项
3. 第二项
4. 第三项

/ 第一项：1  
/ 第二项：2  
/ 第三项：23

第一项 1  
第二项 2  
第三项 3

### 3.4.4 引用

```
#quote(attribution: "爱因斯坦", block:
true)[
  God does not play dice with the
  universe.
]
```

*God does not play dice with the universe.*  
— 爱因斯坦

### 3.4.5 链接

```
#link("https://www.google.com/")
[Google]
```

### 3.4.6 超链接与文献引用

利用`<label>`和`@label` 可以实现超链接和文献引用。

## 3.5 #newpara()函数

由于设计的时候，有些模块不会自动换行。这是有必要的，例如数学公式后面如果不换行就表示对上面的数学公式的解释。

但有时候我们需要换行，这时候就可以使用 `#newpara()` 函数。

区别于官方提供的 `#parbreak()` 函数，`#newpara()` 函数会在段落之间插入一个空行，这样无论在什么场景下，都会开启新的自然段。

只要你觉得需要换行，就可以使用 `#newpara()` 函数。

## 3.6 countblock

`countblock` 是 `Scripst` 提供的一个计数器模块，用来对文档中的某些可以计数的内容进行计数。

全局变量 `cb` 记录着所有可以使用的计数器，你可以通过 `add_countblock` 函数来添加一个计数器。

默认的 `countblock` 有

```
#let cb = (
  "thm": ("Theorem", rgb("#817ffaa5")),
  "def": ("Definition", rgb("#72ab68ab")),
  "prob": ("Problem", rgb("#ac2df653")),
  "prop": ("Proposition", rgb("#6f68abab")),
  "note": ("Note", rgb("#464040ad")),
  "cau": ("⚠", rgb("#f62d2d53")),
)
```

这些计数器已经初始化，你可以直接使用。

**Note** 由于 `typst` 语言的函数不存在指针或引用，传入的变量不能修改，我们只能通过显示的返回值来修改变量。并且将其传入下一个函数。目前作者没有找到更好的方法。

### 3.6.1 countblock 的新建与注册

同时，你可以通过 `add_countblock` 函数来添加（或重载）一个计数器，再通过 `register_countblock` 函数来注册这个计数器。

```
#let cb = add_countblock("test", "This is a test", teal)
#show: register_countblock.with("test")
```

此后你就可以使用 `countblock` 函数来对这个计数器进行计数。

### 3.6.2 countblock 的使用

采用 `countblock` 函数来创建一个块：

```
#countblock(
  name,
  subname,
  count: true,
  cb: cb,
)[
  ...
]
```

其中 `name` 是计数器的名称，`subname` 是创建该条目的名称，`count` 是是否计数，`cb` 是计数器的列表。例如

```
#countblock("thm", subname: "Fermat's Little Theorem", cb)[

  If  $p$  is a prime number, then for any integer  $a$ , the number  $a^p - a$  is
  an integer multiple of  $p$ .
  $
     $a^p \equiv a \pmod p$ 
  $
]
#proof[略。]
```

就会创建一个定理块，并且计数：

#### Theorem 3.1 *Fermat's Little Theorem*

If  $p$  is a prime number, then for any integer  $a$ , the number  $a^p - a$  is an integer multiple of  $p$ .

$$a^p \equiv a \pmod p \quad (3.3)$$

*Proof.* 略。

■

其中 `subname` 如传入，是需要指定的。

你也可以将其封装成另一个函数：

```
#let test = countblock.with()
```

对于刚才创建的 `test` 计数器，你可以使用 `countblock` 函数来计数：

```
#countblock("test", cb)[
  1 + 1 = 2
]

#test[
  1 + 2 = 3
]
```

**This is a test 3.1**     $1 + 1 = 2$

**This is a test 3.2**     $1 + 2 = 3$

其余默认给定的计数器也可以使用，直接封装好的函数：

```
#definition(subname:[...])[]
#theorem(subname:[...])[]
#proposition(subname:[...])[]
#problem(subname:[...])[]
#note[]
#caution[]
```

### Definition 3.1

这是一个定义，请你理解它。

### Theorem 3.2


这是一个定理，请你证明它。

### Proposition 3.1

这是一个命题，请你证明它。

### Note

这是一个注记，请你注意它。

 这是一个提醒，请你谨慎它。

这些计数器编号的逻辑是：

- 如果没有章节，那么只有一个计数器编号

- 如果有章节，那么计数器编号是章节号.本章节内截至此块出现过的该种块的数量  
如此，你可以注册和使用任意数量的计数器。

## 四 结语

上面展示了 Scripst 的使用方法，以及模板的参数说明和效果展示。

希望这篇文章能够帮助你更好地使用 typst 和 Scripst。

也欢迎你为 Scripst 提出建议和改进建议，甚至贡献代码。

感谢您对 typst 和 Scripst 的支持！