Scripst 的使用方法

article 样式

AnZrew AnZreww AnZrewww

2025-03-11

摘要: Scripst 是一款简约易用的 Typst 语言模板，适用于日常文档、作业、笔记、论文等多种场景

关键词: Scripst； Typst； 模板

目录

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 是一款简约易用的 Typst 语言模板，适用于日常文档、作业、笔记、论文等多种场景。

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，就可以用类似 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 非常适合轻量级日常文档的编写。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 提供了一些简约的，可便利日常文档生成的模板样式。

1.2.1 在线使用

Scripst 包已经提交至社区，在联网的状态下，可直接使用

```
#import "@preview/scripst:1.1.1": *
```

来引入 Scripst 的模板。你也可以通过 `typst init` 来一键使用模板创建新的项目：

```
typst init @preview/scripst:1.1.1 project_name
```

这种方法无需手动下载模板文件，只需要在文档中引入即可。

1.2.2 离线使用

解压使用

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。Script 模板采用模块化设计，你可以轻松找到并修改模板中你需要修改的部分。

本地包管理

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，无需担心模板文件难以修改，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/preview/scripst/1.1.1              # in Linux
%APPDATA%\typst\packages\preview\scripst\1.1.1                   # in Windows
~/Library/Application Support/typst/packages/local/scripst/1.1.1 # macOS
```

你可以执行指令

```
cd ~/.local/share/typst/packages/preview/scripst/1.1.1
git clone https://github.com/An-314/scripst.git 1.1.1
```

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 "@preview/scripst:1.1.1": *
```

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 @preview/scripst:1.1.1 project_name
```

在引入模板后通过这样的方式创建一个 `article` 文件：

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

Scripst 的模板提供了一些参数，用来定制文档的样式。

```
#let scripst(
  template: "article",  // str: ("article", "book", "report")
  title: "",            // str, content, none
  info: "",             // str, content, none
  author: (),           // str, content, array, none
  time: "",             // str, content, none
  abstract: none,       // str, content, none
  keywords: (),         // array
  font-size: 11pt,      // length
  contents: false,      // bool
  content-depth: 2,     // int
  matheq-depth: 2,      // int: (1, 2, 3)
  counter-depth: 2,     // int: (1, 2, 3)
  cb-counter-depth: 2,  // int: (1, 2, 3)
  header: true          // bool
```

```
  lang: "zh",           // str: ("zh", "en", "fr", ...)
  par-indent: 2em,      // length
  par-leading: none,    // length
  par-spacing: none,    // length
  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 Scripst 提供了三种模板，分别是 article 、book 和 report 。

本模板采用 article 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
title	`content`, `str`, `none`	`""`	文档标题

文档的标题。（不为空时）会出现在文档的开头和页眉中。

2.3 info

参数	类型	默认值	说明
info	`content`, `str`, `none`	`""`	文档信息

文档的信息。（不为空时）会出现在文档的开头和页眉中。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
author	str, content, array, none	()	文档作者

文档的作者。要传入 str 或者 content 的列表，或者直接的 str 或者 content 对象。

Note

注意，如果是一个作者的情况，可以只传入 str 或者 content，在多个作者的时候传入一个 str 或者 content 的列表，例如：author: ("作者 1", "作者 2")

会在文章的开头以 $\min(\#\text{authors}, 3)$ 个作为一行显示。

2.5 time

参数	类型	默认值	说明
time	content, str, none	""	文档时间

文档的时间。会出现在文档的开头和页眉中。

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）会出现在文档的开头。

建议在使用摘要前，首先定义一个 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开头。

2.8 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。

2.9 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`。

2.10 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`。

2.11 matheq-depth

参数	类型	可选值	默认值	说明
matheq-depth	`int`	`1`, `2`, `3`	`2`	数学公式的深度

数学公式编号的深度。默认为 `2`。

Note

计数器的详细表现见 小节 2.12 。

2.12 counter-depth

参数	类型	可选值	默认值	说明
counter-depth	int	1, 2, 3	2	计数器的深度

文中 figure 环境中的图片 image，表格 table，以及代码 raw 的计数器深度。默认为 2。

Note 计数器的详细表现

一个计数器的深度为 1 时，计数器的编号会是全局的，不会受到章节的影响，即 1, 2, 3, ...。

一个计数器的深度为 2 时，计数器的编号会受到一级标题的影响，即 1.1, 1.2, 2.1, 2.2, ...。但如果此时整个文档没有一级标题，Scripst 会自动将其转化为深度为 1 的情况。

一个计数器的深度为 3 时，计数器的编号会受到一级标题和二级标题的影响，即 1.1.1, 1.1.2, 1.2.1, 1.2.2, 2.1.1, ...。但如果此时整个文档没有二级标题但有一级标题，Scripst 会自动将其转化为深度为 2 的情况；如果没有一级标题，Scripst 会自动将其转化为深度为 1 的情况。

2.13 cb-counter-depth

参数	类型	可选值	默认值	说明
cb-counter-depth	int	1, 2, 3	2	countblock 的计数器深度

countblock 环境中的计数器深度。默认为 2。

如果改变了 countblock 计数器的默认深度，你在使用时候还需要指定改变了的深度，或者重新封装函数。详情见 小节 3.6.4 。

2.14 header

参数	类型	默认值	说明
header	bool	true	页眉

是否生成页眉。默认为 true。

Note

页眉包括文档的题目、信息和当前所在的章节标题。

- 如果三者都存在，则将会三等分地显示在页眉中。
- 如果文档没有信息，页眉仅会在最左最右显示文档的题目和当前所在的章节标题。
  - 进而如果文档没有题目，页眉仅会在最右侧显示当前所在的章节标题。
- 如果文档没有任何一级标题，页眉将仅在最左最右显示文档的题目和信息。
  - 进而如果文档没有信息，页眉仅会在最左侧显示文档的题目。
- 如果什么都没有，页眉将不会显示。

2.15 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"`。

接受 ISO_639-1 编码格式传入，如`"zh"`、`"en"`、`"fr"`等。

2.16 par-indent

参数	类型	默认值	说明
par-indent	`length`	`2em`	段落首行缩进

段落首行缩进。默认为 `2em`。如果调节成 `0em`，则为不缩进。

2.17 par-leading

参数	类型	默认值	说明
par-leading	`length`	跟随 `lang` 设置	段落内的行间距

段落内间距。在中文文档中默认是 `1em`。

Note

默认值会随着语言的选择而变化，具体情况见下表

语言类型	默认值
东亚文字（汉语、韩语、日语等）	1em
阿拉伯文字（阿拉伯语,波斯语等）	0.75em

语言类型	默认值
斯拉夫文字（俄语，保加利亚语等）	0.7em
南亚、东南亚、阿姆哈拉文字（泰语、越南语、缅甸语、印地语、阿姆哈拉语等）	0.85em
其他文字	0.6em

,

2.18 par-spacing

参数	类型	默认值	说明
par-spacing	`length`	跟随 `lang` 设置	段落间距

段落间距。在中文文档中默认是 `1.2em`。

Note

默认值会随着语言的选择而变化，具体情况见下表

语言类型	默认值
东亚文字（汉语、韩语、日语等）	1.2em
阿拉伯文字（阿拉伯语,波斯语等）	1.25em
斯拉夫文字（俄语，保加利亚语等）	1.2em
南亚、东南亚、阿姆哈拉文字（泰语、越南语、缅甸语、印地语、阿姆哈拉语等）	1.3em
其他文字	1em

,

2.19 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 扉页

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

3.2 目录

如果 `contents` 参数为 `true`，则会生成目录，效果见本文档目录。

3.3 文字样式与环境

Scripst 提供了一些常用的文字样式和环境，如粗体、斜体、标题、图片、表格、列表、引用、链接、数学公式等。

3.3.1 字体

这是正常的文本。 This is a normal text.

这是粗体的文本。 This is a bold text.

这是斜体的文本。 This is an italic text.

安装 CMU Serif 字体以获得更好（类似 LaTeX）的显示效果。

3.3.2 环境

3.3.2.1 标题

一级标题编号随文档语言而异，包括中文/罗马数字/希腊字母/假名/阿拉伯文数字/印地文数字等，其余级别标题采用阿拉伯数字编号。

3.3.2.2 图片

图片环境会自动编号，如下所示：

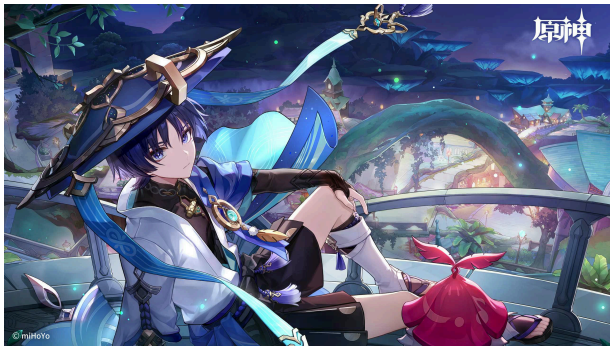

图 3.3.1 散宝

3.3.2.3 表格

得益于 tablem 包，使用本模板时可以用 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 3.3.1 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 3.3.2 tablem 表格示例

可以选择 numbering: none,使得表格不编号，如上所示，前面章节的表格并没有进入全文的表格计数器。

3.3.2.4 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：

$$a^2 + b^2 = c^2$$
$$\frac{1}{2} + \frac{1}{3} = \frac{5}{6} \tag{3.3.1}$$

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：

$$
\begin{aligned}
\nabla \cdot \boldsymbol{E} &= \frac{\rho}{\varepsilon_0} \\
\nabla \cdot \boldsymbol{B} &= 0 \\
\nabla \times \boldsymbol{E} &= -\frac{\partial \boldsymbol{B}}{\partial t} \\
\nabla \times \boldsymbol{B} &= \mu_0 \left(\boldsymbol{J} + \varepsilon_0 \frac{\partial \boldsymbol{E}}{\partial t} \right)
\end{aligned}
\tag{3.3.2}
$$

3.3.3 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: 3
```

第一项 1

第二项 2

第三项 3

3.3.4 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

3.3.5 链接

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

Google

3.3.6 超链接与文献引用

利用`<lable>`和`@lable` 可以实现超链接和文献引用。

3.4 #newpara()函数

默认某些模块不自动换行。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。

3.5 labelset

得益于 typst 中的 label 函数，除了给这种类型添加标签外，还可以通过 label 方便地为所引用的对象设置样式。

因此，Scripst 内置了一些常用的设置，你可以通过直接添加 label 来设置样式。

```
== Schrödinger equation <hd.x>

下面是 Schrödinger 方程：
$
  i hbar dv(,t) ket(Psi(t)) = hat(H) ket(Psi(t))
$ <text.blue>
其中
$
  ket(Psi(t)) = sum_n c_n ket(phi_n)
$ <eq.c>
是波函数。由此可以得到定态的 Schrödinger 方程：
$
  hat(H) ket(Psi(t)) = E ket(Psi(t))
$
<text.teal>
其中 $E$<text.red> 是#[能量]<text.lime>。
```

Schrödinger equation

下面是 Schrödinger 方程：

$$i\hbar\frac{\mathrm{d}}{\mathrm{d}t}|\Psi(t)\rangle = \hat{H}|\Psi(t)\rangle \tag{3.5.1}$$

其中

$$|\Psi(t)\rangle = \sum_n c_n|\varphi_n\rangle$$

是波函数。由此可以得到定态的 Schrödinger 方程：

$$\hat{H}|\Psi(t)\rangle = E|\Psi(t)\rangle \tag{3.5.2}$$

其中 E 是能量。

目前 Scripst 提供了以下的设置：

标签	功能
`eq.c`	给数学环境的公式取消编号
`hd.c`	给标题取消编号，但还在目录中显示
`hd.x`	给标题取消编号，且不在目录中显示
`text.{color}`	给文本设置颜色 `color in (black, gray, silver, white, navy, blue, aqua, teal, eastern, purple, fuchsia, maroon, red, orange, yellow, olive, green, lime,)`

表 3.5.1 Label Set

⚠

注意上面的字符串已经被用于样式设置，你可以对其的样式设置进行重载，但不要在使用 abel 和 refrence 时使用这些字符串。

3.6 countblock

Definition 3.1 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。

现在你看到的就是一个 `definition` 块，它是一个计数器模块的例子。

3.6.1 默认提供的 countblock

Scripst 默认提供了一些计数器，你可以直接使用。分别是：

- 定义：`#definition`
- 定理：`#theorem`
- 命题：`#proposition`
- 引理：`#lamma`
- 推论：`#corollary`
- 评论：`#remark`
- 断言：`#claim`
- 练习：`#exercise`

- 问题：`#problem`
- 例子：`#example`
- 注记：`#note`
- 提醒：`#caution`

这些函数的参数和效果是一样的，只是计数器的名称不同。

```
#definition(
  subname: [],
  count: true,
  lab: none,
  cb-counter-depth: 2,
)[
  ...
]
```

参数说明如下

参数	类型	默认值	说明
subname	array	[]	该条目的名称
count	bool	true	是否计数
lab	str	none	该条目的标签
cb-counter-depth	int	2	计数器的深度

下面是一个示例：

```
#theorem(subname: [_Fermat's Last Theorem_], lab: "fermat")[

  No three $a, b, c in NN^+$ can satisfy the equation
  $
    a^n + b^n = c^n
  $
  for any integer value of $n$ greater than 2.
]
#proof[Cuius rei demonstrationem mirabilem sane detexi. Hanc marginis
exiguitas non caperet.]
```

就会创建一个定理块，并且计数：

Theorem 3.1 ***Fermat's Last Theorem***

No three $a, b, c \in \mathbb{N}^+$ can satisfy the equation

$$a^n + b^n = c^n \tag{3.6.1}$$

for any integer value of n greater than 2.

Proof. Cuius rei demonstrationem mirabilem sane detexi. Hanc marginis exiguitas non caperet.

■

3.6.1.1 subname 参数

subname 是会显示在计数器后的信息，例如定理名称等。在上述例子中是“Fermat's Last Theorem”。

3.6.1.2 lab 参数

此外，你可以使用 lab 参数来为这个块添加一个标签，以便在文中引用。例如刚才的 fermat 定理块，你可以使用@fermat 来引用它。

```
Fermat 并没有对 @fermat 给出公开的证明。
```

Fermat 并没有对 Theorem 3.1 给出公开的证明。

另外，我们需要说明，在提供的这些默认封装好的函数中，proposition, lemma, corollary, remark, claim, 是共用同一个计数器的，效果如下：

Lemma 3.1
这是一个引理，请你证明它。

Proposition 3.2
这是一个命题，请你证明它。

Corollary 3.3
这是一个推论，请你证明它。

Remark 3.4
这是一个评论，请你注意它。

Claim 3.5
这是一个断言，请你证明它。

而其余的计数器是互相独立的。

3.6.1.3 count 参数

此外，对于 count 参数，如果你不想计数，可以将其设置为 false。note 和 caution 默认不计数。如果你想要计数，可以将其设置为 true。

```
#note(count: true)[

  这是一个注记，请你注意它。
]

#note[

  这是一个注记，请你注意它。
]
```

Note 3.1

这是一个注记，请你注意它。

Note

这是一个注记，请你注意它。

3.6.1.4 cb-counter-depth 参数

对于该参数的详细解释见 小节 3.6.4。

3.6.2 cb 全局变量

Scripst 通过全局变量 cb 记录着所有可以使用的计数器，以及全局的计数器深度 cb-counter-depth。

Scripst 中默认的 cb 是这样的：

```
#let cb = (
  "def": ("Definition", mycolor.green, "def"),
  "thm": ("Theorem", mycolor.blue, "thm"),
  "prop": ("Proposition", mycolor.violet, "prop"),
  "lem": ("Lemma", mycolor.violet-light, "prop"),
  "cor": ("Corollary", mycolor.violet-dark, "prop"),
  "rmk": ("Remark", mycolor.violet-darker, "prop"),
  "clm": ("Claim", mycolor.violet-deep, "prop"),
  "ex": ("Exercise", mycolor.purple, "ex"),
  "prob": ("Problem", mycolor.orange, "prob"),
  "eg": ("Example", mycolor.cyan, "eg"),
  "note": ("Note", mycolor.grey, "note"),
  "cau": ("⚠", mycolor.red, "cau"),
  "cb-counter-depth": 2,
)
```

3.6.3 countblock 的新建与注册

Scripst 提供了 `add-countblock` 函数来添加（或重载）一个计数器，以及 `reg-countblock` 函数来注册这个计数器。你可以通过在文档开头

```
#let cb = add-countblock(cb, "test", "This is a test", teal)
#show: reg-countblock.with("test")
```

来创建一个 countblock。

> **Note**
>
> 上面的代码意味着我们先更新了 `cb`，再将其的计数器加入整个文档中。

3.6.3.1 函数 add-countblock

函数 `add-countblock` 的参数如下

```
#add-countblock(cb, name, info, color, counter-name: none) {return cb}
```

参数说明如下

参数	类型	默认值	说明
cb	dict		计数器字典
name	str		计数器的名称
info	str		计数器的信息
color	color		计数器的颜色
counter-name	str	none	计数器的编号

- `cb` 是一个字典，其格式如小节 3.6.2 所示。该函数的作用就是将 `cb` 更新，在使用时需要按照显示赋值。

  > **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。

- `name: (info, color, counter-name)`是一个计数器的基本信息。在渲染时，计数器的左上角会显示 `info counter(counter-name)`例如 `Theorem 1.1` 作为该计数器的编号；颜色会是 `color` 颜色的。
- `counter-name` 是计数器的编号，如果没有指定，那么会使用 `name` 作为编号。

3.6.3.2 函数 reg-countblock

函数 reg-countblock 的参数如下

```
#show reg-countblock.with(name, cb-counter-depth: 2)
```

参数说明如下

参数	类型	默认值	说明
counter-name	str		计数器的编号
cb-counter-depth	int	2	计数器的深度

- counter-name 是计数器的编号，也就是在 add-countblock 中（未指定是 name）显示指定的参数。例如默认提供的 clm 的计数器是 prop。
- cb-counter-depth 是你该计数器的深度，你可以指定为 1, 2, 3。

此后你就可以使用 countblock 函数来使用这个计数器。

3.6.4 countblock 的计数器

前面并没有提到 cb-counter-depth 参数，在这一章我们详细讲解这个参数，以及其实现方式。

全局变量 cb 中的 cb-counter-depth 默认值是 2。所以默认提供的 countblock 函数的计数器深度是 2。

> **Note**
>
> 如果你直接更改全局变量里的 cb-counter-depth，默认提供的计数器是不会改变的。这是因为在创建计数器时，会将原先的 cb.at("cb-counter-depth") 作为默认值传入。当更新 cb 时，原先的 cb-counter-depth 不会改变。所以你需要重新注册这个计数器。

计数器的逻辑与 小节 2.12 的相同。

如果你需要注册一个深度为 3 的计数器，你可以这样做：

```
#let cb = add-countblock(cb, "test1", "This is a test1", green)
#show: reg-countblock.with("test1", cb-counter-depth: 3)
```

此外你可以通过 reg-default-countblock 函数来注册默认的计数器。例如你**希望所有的默认的计数器都是深度为 3 的**，你可以这样做：

```
#show: reg-default-countblock.with(cb-counter-depth: 3)
```

当然，如果你仅仅这么做还不够，因为封装好的计数器还是以 2 为默认值。如果你直接调用

```
```

```
#definition[
  这是一个定义，请你理解它。
]
```

那么这个计数器的深度还是 2。

Definition 3.2

这是一个定义，请你理解它。

所以你需要指定深度为 3：

```
#definition(cb-counter-depth: 3)[
  这是一个定义，请你理解它。
]
```

Definition 3.6.3

这是一个定义，请你理解它。

当然，你可以直接**进一步对其进行封装**：

```
#let definition = definition.with(cb-counter-depth: 3)
```

之后再使用 `definition` 函数就会默认使用深度为 3 的计数器

```
#definition[
  这是一个定义，请你理解它。
]
```

Definition 3.6.4

这是一个定义，请你理解它。

Note

事实上，前文提到的 `cb-counter-depth` 参数就是在文档初始化的时候调用 `reg-default-countblock` 函数来设置的。

3.6.5 countblock 的使用

在定义并且注册一个块之后，就可以使用 `countblock` 函数来创建一个块：

```
#countblock(
  name,
  cb,
```

```
  cb-counter-depth: cb.at("cb-counter-depth"), // default: 2
  subname: "",
  count: true,
  lab: none
)[
  ...
]
```

参数说明如下

参数	类型	默认值	说明
name	str		计数器的名称
cb	dict		计数器字典
cb-counter-depth	int	cb.at("cb-counter-depth")	计数器的深度
subname	str		该条目的名称
count	bool	true	是否计数
lab	str	none	该条目的标签

- name 是计数器的名称，也就是在 add-countblock 中显示指定的参数。
- cb 是一个字典，其格式如小节 3.6.2 所示。注意，你需要传含有该计数器的（最新的）cb，所以一定需要先更新 cb，再传入。
- cb-counter-depth 是你该计数器的深度，你可以指定为 1, 2, 3。
- subname 是会显示在计数器后的信息，例如定理名称等。
- count 是一个布尔值，如果你不想计数，可以将其设置为 false。
- lab 是一个字符串，如果你想要为这个块添加一个标签，以便在文中引用，可以使用这个参数。

例如，我想使用我在 小节 3.6.3 中创建的 test 计数器：

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

This is a test 3.1

$$1 + 1 = 2$$

当然也可以将其封装成另一个函数：

```
#let test = countblock.with("test", cb)
```

然后使用 test 函数：

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

This is a test 3.2

$$1 + 1 = 2$$

当然，对于在 小节 3.6.4 中注册的深度为 3 的 `test1` 计数器，我们需要在使用时指定深度：

```
#countblock("test1", cb, cb-counter-depth: 3)[
  1 + 1 = 2
]
#let test1 = countblock.with("test1", cb, cb-counter-depth: 3)
#test1[
  1 + 1 = 2
]
```

This is a test1 3.6.1

$$1 + 1 = 2$$

This is a test1 3.6.2

$$1 + 1 = 2$$

3.6.6 总结

Scripst 提供了一种简单的计数器模块，你可以通过 `add-countblock` 函数来添加一个计数器，通过 `reg-countblock` 函数来注册这个计数器，然后通过 `countblock` 函数来使用这个计数器。

对于默认的计数器，其深度为 2，你可以通过 `reg-default-countblock` 函数来注册默认的计数器。

如果你希望所有 countblock 的深度为 2，那么在你注册和使用的时候不必在意深度。

如果你希望所有 countblock 的深度为 3，那么你需要在注册和使用的时候指定深度。

Example

下面给出一个例子：使用者希望包括默认的所有 countblock 的计数器深度都是 3，但希望 `remark` 与先前默认绑定的 `proposition`, `lemma`, `corollary`, `claim` 的计数器独立出来。再创建一个深度为 3 的 `algorithm` 计数器。

```
#show: scripst.with(
  // ...
  cb-counter-depth: 3,
)
#let cb = add-countblock(cb, "rmk", "Remark", mycolor.violet-darker)
#let cb = add-countblock(cb, "algorithm", "Algorithm", mycolor.yellow)
#show: reg-countblock.with("rmk", cb-counter-depth: 3)
#show: reg-countblock.with("algorithm", cb-counter-depth: 3)
#let definition = definition.with(cb-counter-depth: 3)
#let theorem = theorem.with(cb-counter-depth: 3)
// ...
#let remark = countblock.with("rmk", cb, cb-counter-depth: 3) // 这里需要重新封装是因为其计数器改变了
#let algorithm = countblock.with("algorithm", cb, cb-counter-depth: 3)
```

放在文档的开头，`#script` 之后即可。

3.7 一些其他的块

3.7.1 空白块

此外，Scripst 还提供了这样的无标题的块，你可以自定义颜色来使用。

例如

```
#blankblock(color: color.red)[
  这是一个红色的块。
]
```

这是一个红色的块。

3.7.2 证明与■（证明结束）

```
#proof[
  这是一个证明。
]
```

Proof.

这是一个证明。

■

这提供一个简单的证明环境，以及证毕符号。

3.7.3 解答

```
#solution[
  这是一个解答。
]
```

Solution.

这是一个解答。

这提供一个简单的解答环境。

3.7.4 分隔符

```
#separator
```

可以使用 `#separator` 函数来插入一个分隔符。

四 结语

上文展示了 Scripst 的使用方法，以及模板的参数说明和效果展示。

希望这篇文档能够帮助你更好地使用 typst 和 Scripst。

也欢迎你为 Scripst 提出建议、改善方法及贡献代码。

感谢您对 typst 和 Scripst 的支持！