柜台

对页面、元素等进行计数。

通过计数器功能，您可以访问和修改页面、标题、数字等的计数器。此外，您可以为您想要计数的其他事物定义自定义计数器。

由于计数器在整个文档过程中都会发生变化，因此它们的当前值是*与上下文相关的*。建议在继续此处之前先阅读有关上下文的章节。

访问计数器

要访问计数器的原始值，我们可以使用该get函数。此函数返回一个数组：计数器可以有多个级别（在节、小节等标题的情况下），数组中的每一项对应一个级别。

```
#set heading(numbering: "1.")

= Introduction
Raw value of heading counter is
#context counter(heading).get()
```

1. Introduction

Raw value of heading counter is (1,)

显示计数器

通常，我们希望以更易于理解的方式显示计数器的值。为此，我们可以调用display计数器上的函数。此函数检索当前计数器值，并使用提供的或自动推断的编号对其进行格式化。

```
#set heading(numbering: "1.")

= Introduction
Some text here.

= Background
The current value is: #context {
  counter(heading).display()
}

Or in roman numerals: #context {
  counter(heading).display("I")
}
```

1. Introduction

Some text here.

2. Background

The current value is: 2.

Or in roman numerals: II

修改计数器

要修改计数器，可以使用step和update方法：

- 该step方法将计数器的值加一。因为计数器可以有多个级别，所以它可以选择接受一个level参数。如果给定，计数器将在给定深度处步进。
- 该update方法允许您任意修改计数器。在其基本形式中，您给它一个整数（或多个级别的数组）。为了获得更大的灵活性，您还可以为其提供一个接收当前值并返回新值的函数。

航向计数器在显示航向之前步进，因此Analysis即使第二次更新后计数器为 6，也会得到数字 7。

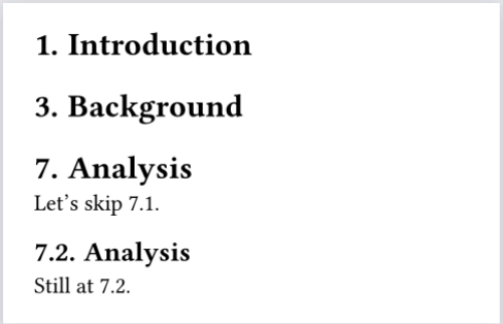
```
#set heading(numbering: "1.")

= Introduction
#counter(heading).step()

= Background
#counter(heading).update(3)
#counter(heading).update(n => n * 2)

= Analysis
Let's skip 7.1.
#counter(heading).step(level: 2)

== Analysis
Still at #context {
  counter(heading).display()
}
```



页计数器

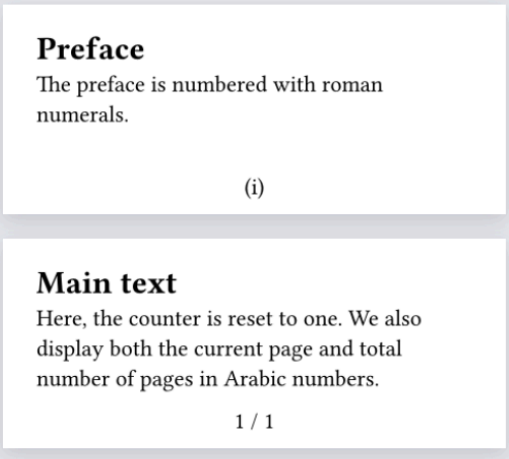
页计数器很特殊。它会在每个分页符处自动步进。但与其他计数器一样，您也可以手动步进。例如，您的前言可以使用罗马页码，然后切换到主要内容的阿拉伯页码，并将页码计数器重置为 1。

```
#set page(numbering: "(i)")

= Preface
The preface is numbered with
roman numerals.

#set page(numbering: "1 / 1")
#counter(page).update(1)

= Main text
Here, the counter is reset to one.
We also display both the current
page and total number of pages in
Arabic numbers.
```



定制柜台

要定义您自己的计数器，请使用counter字符串作为键调用该函数。该键全局标识计数器。

```
#let mine = counter("mycounter")
#context mine.display() \
#mine.step()
#context mine.display() \
#mine.update(c => c * 3)
#context mine.display()
```

0
1
3

如何迈步

当您定义和使用自定义计数器时，一般来说，您应该首先步进计数器，然后显示它。这样，计数器的步进行为可以取决于它所步进的元素。如果您正在为定理编写计数器，则定理的定义将首先包含计数器步骤，然后才显示计数器和定理的内容。

```
#let c = counter("theorem")
#let theorem(it) = block[
  #c.step()
  *Theorem #context c.display():*
  #it
]

#theorem[$1 = 1$]
#theorem[$2 < 3$]
```

Theorem 1: $1 = 1$

Theorem 2: $2 < 3$

标题计数器的示例可以很好地解释其背后的基本原理：标题计数器的更新取决于标题的级别。通过直接在标题之前跳转，当遇到 2 级标题时，我们可以正确地从 跳转1到。1.1如果我们要追随标题，我们将不知道该迈向什么。

由于计数器应始终在其计数的元素之前步进，因此它们始终从零开始。这样，它们在第一次显示时就处于同一状态（发生在第一步之后）。

时间旅行

计数器可以穿越时空！您可以在达到计数器的最终值之前找到它，甚至可以确定文档中任何特定位置的值是什么。

```
#let mine = counter("mycounter")

= Values
#context [
  Value here: #mine.get() \
  At intro: #mine.at(<intro>) \
  Final value: #mine.final()
]

#mine.update(n => n + 3)

= Introduction <intro>
#lorem(10)

#mine.step()
#mine.step()
```

Values

Value here: (0,)
At intro: (3,)
Final value: (5,)

Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

其他种类的状态

该类型与状态类型counter密切相关。阅读其文档，了解有关 Typst 状态管理的更多详细信息，以及为什么它不只使用普通变量作为计数器。

key 斯特 或者 标签 或者 选择器 或者 地点 或者 功能 *必需的* *位置性*

标识该计数器的密钥。

- 如果是字符串，则创建一个仅受手动更新影响的自定义计数器，
- 如果是page函数的话，通过页数进行计数，
- 如果它是一个选择器，则对与选择器匹配的元素进行计数。例如，
  - 提供一个元素函数：计算该类型的元素，
  - 提供一个<label>：对具有该标签的元素进行计数。

定义

get *语境化*

检索当前位置的计数器值。始终返回一个整数数组，即使计数器只有一个数字。

这相当于.counter.at(here())

自己。得到（）->整数 大批

display *语境化*

显示带有编号的计数器的当前值并返回格式化的输出。

兼容性： 为了与 Typst 0.10 及更低版本兼容，此功能在没有既定上下文的情况下也可以工作。然后，它将创建不透明的上下文内容，而不是直接返回编号的输出。此行为将在未来版本中删除。

```
自己。展示（
  汽车 斯特 功能，
  两个都： 布尔值，
）-> 任何
```

numbering 汽车 或者 斯特 或者 功能 *位置性*

编号模式或函数，指定如何显示计数器。如果给定一个函数，该函数将接收计数器的每个数字作为单独的参数。如果数字数量不同，例如对于标题参数，您可以使用参数接收器。

如果省略或设置为`auto`，则使用计数元素的编号样式或模式（`"1.1"`如果不存在此类样式）显示计数器。

默认：`auto`

both 布尔值

如果启用，则一起显示当前和最终的顶级计数。两者都可以通过单一编号模式来设计样式。当`"1 / 1"`给出类似的模式时，页码属性使用它来显示当前页数和总页数。

默认：`false`

at *语境化*

检索给定位置处的计数器值。始终返回一个整数数组，即使计数器只有一个数字。

必须`selector`与文档中的一个元素完全匹配。最有用的选择器类型是标签和位置。

兼容性： 为了与 Typst 0.10 及更低版本兼容，如果是`selector`位置，则此函数也可以在没有已知上下文的情况下工作。此行为将在未来版本中删除。

```
自己。在（ 标签 选择器 地点 功能 ）-> 整数 大批
```

selector 标签 或者 选择器 或者 地点 或者 功能 *必需的* *位置性*

应检索计数器值的位置。

final *语境化*

检索文档末尾的计数器值。始终返回一个整数数组，即使计数器只有一个数字。

```
自己。最终的（ 没有任何 地点 ）-> 整数 大批
```

location 没有任何 或者 地点 *位置性*

兼容性： 此参数仅用于与 Typst 0.10 及更低版本兼容，不应再使用。

默认：`none`

step

将计数器的值加一。

更新将在返回内容插入文档的位置生效。如果您不将输出放入文档中，则不会发生任何事情！例如，如果您编写.计数器更新始终按布局顺序应用，在这种情况下，Typst 不知道何时步进计数器。

```
let _ = counter(page).step()
```

```
自己。步（等级： 整数 ）-> 内容
```

level 整数

踏入计数器的深度。默认为1.

默认：1

update

更新计数器的值。

就像 一样step，仅当您将生成的内容放入文档中时才会发生更新。

```
自己。更新（ 整数 大批 功能 ）-> 内容
```

update 整数 或者 大批 或者 功能 *必需的* *位置性*

如果给定一个整数或整数数组，则将计数器设置为该值。如果给定一个函数，该函数接收先前的计数器值（每个数字作为单独的参数）并且必须返回新值（整数或数组）。