AsciiDoc 离线手册

Table of Contents

前言 1
- 项目产生的原因 2

介绍 3
- 关于 AsciiDoc 4
- 文档结构 5
  - 文档 5
  - 文档行 5
  - 块 6
  - 文本和内联元素 6
  - 编码和 AsciiDoc 文件 6
- 重要概念 7
  - 文档 7
  - 元素 7
  - 属性 7
  - 宏 8
  - 预处理器指令 8
- 文档处理 9
- 规范化 10

块 11
- 块的简介 12
  - 什么是块 12
  - 块的形式 12
  - 内容模型 12
  - 上下文 13
  - 块的样式 14
  - 块的共性 14

前言

《**AsciiDoc** 离线手册》是一本用于查阅 AsciiDoc 文档的帮助手册。

项目产生的原因

1. 虽然 AsciiDoc 的在线文档已经很详尽了。但开发者们并没有提供切实可用的，离线的文档或用户手册。
2. AsciiDoc 比 markdown 更为复杂，在不熟悉 AsciiDoc 语法的情况下，时常需要去查阅一下在线文档。
3. 将常用的文档和用户手册页翻译，整理成中文，方便我离线使用。

更多详情详见：README

□□

关于 AsciiDoc

AsciiDoc 是一种轻量级语义标记语言，主要用于编写技术文档。该语言可用于生成各种演示丰富的输出格式，所有这些格式都来自以简洁、人类可读的纯文本格式编码的内容。

AsciiDoc 语法很直观，因为它建立在用于标记和结构化文本的完善的纯文本约定之上。不熟悉 AsciiDoc 的人可能仅通过查看就可以猜出许多语法元素的用途。这是因为语法元素经过精心选择，看起来像它们的意思（科技行业长期采用的一种做法）。

AsciiDoc 语言与其生成的输出格式无关。AsciiDoc 处理器可以解析和理解 AsciiDoc 源文档，并将解析后的文档结构转换为一种或多种输出格式，例如 HTML、PDF、EPUB3、手册页或 DocBook。生成多种输出格式的能力是 AsciiDoc 的主要优势之一。此功能能使其能够用于静态站点生成器、IDE、git 工具和服务、CI/CD 系统和其他软件。

AsciiDoc 弥合了易于编写与技术创作和发布的严格要求之间的差距。AsciiDoc 只需要文本编辑器即可阅读或编写，因此入门门槛较低。

文档结构

在此页面上，你将了解 AsciiDoc 文档的整体结构。此时不要担心语法的细节。稍后将在文档中详细介绍该主题。现在，我们的目标只是了解 AsciiDoc 文档的组成部分。

文档

AsciiDoc 是一种纯文本书写格式，没有样板附件或序言。一个 AsciiDoc 文档可能只包含一个句子（或者甚至是一个字符）。

以下示例是一个有效的 AsciiDoc 文档，其中一个段落包含一个句子：

```
这是一个基础的 AsciiDoc 文档。
```

当然，你可以拥有比单个句子更多的内容。我们在这里要强调的是，上手很简单。

AsciiDoc 文档是一系列（按行）堆叠在一起的块（block）。这些块通常由空行彼此偏移（尽管在某些情况下这些可能是可选的）。

要将上一个文档从一个段落扩展为两个段落，你可以用一个空行分隔两个段落：

```
这是一个基础的 AsciiDoc 文档。

这个文档包含两个段落。
```

AsciiDoc 文档可能以文档标题（document header）开头。尽管文档标题是可选的，但它经常被使用，因为它允许你指定文档标题并以文档属性的形式设置文档范围的配置和可重用文本。

```
= 文档标题
:reproducible:

这是一个由 {author} 创建的 AsciiDoc 文档。

这个文档有两个段落。
它还有一个标题用来指定文档的标题。
```

几乎任何块的组合都构成了有效的 AsciiDoc 文档（文档类型规定了一些结构要求）。文档的范围可以从单个句子到多部分的书。

文档行

文档行（Line）是 AsciiDoc 中的一个重要结构。一行定义为两边由换行符或文档边界分隔的文本。语法的许多方面必须占据一整行。这就是为什么我们说 AsciiDoc 是一种面向行的语言。

例如，章节标题必须单独占一行。对于属性条目、块标题、块属性列表、块宏、列表项、块定界符等也是如此。

部分标题示例，必须占一行

```
=== 章节标题
```

属性条目示例，它也必须至少占据一行

```
:name: value
```

扩展到两行的属性条目示例

```
:name: value \
more value
```

空行也可能很重要。标题与正文由一个空行分开。许多块也由空行分隔，正如你在前面的两段示例中看到的那样。

相反，段落内容中的行是微不足道的。在学习 AsciiDoc 语法时请牢记这些要点。

块

AsciiDoc 文档中的块（Blocks）奠定了文档结构。一些块可能包含其他块，因此文档结构本质上是分层的（即树结构）。你可以预览此部分结构，例如，通过启用自动目录。块的示例包括段落、章节、列表、分隔块、表格和块宏。

块很容易识别，因为它们通常被空行从其他块分隔（尽管并不总是必需的）。块总是从新的一行开始，在一行的末尾终止，并与左边距对齐。

每个块都可以有一行或多行块元数据。此元数据可以采用块属性、块锚点或块标题的形式。这些元数据行必须位于块本身的上方并与其直接相邻。

章节、非逐字分隔的块和 AsciiDoc 表格单元格可能包含其他块。尽管块形成了层次结构，但即使是嵌套块也从左边距开始。通过要求块从左边距开始，避免了必须跟踪和维护缩进级别的单调乏味，并使内容更可重用。

文本和内联元素

被标记、定界符和元数据线包围着的是文本。文本是文档的主要焦点，也是 AsciiDoc 语法给它这么多空间的原因。文本最常出现在块的行中（例如，段落）、块标题（例如，章节标题）和列表项中，dan当然还可存在于其他地方。

文本可能会被替换。替换（Substitutions）将标记解释为文本格式，用文本或非文本元素替换宏，扩展属性引用，并执行其他类型的文本替换。

除非另有说明，否则普通文本受所有替换的约束。逐字文本（Verbatim text）受最小替换集的约束，以允许它以源码中的样式显示在输出中。也可以禁用所有替换，以便将文本传递到未修改的输出（即原始）。文本的解析最终成为内联元素和其他形式的转换的混合体。

编码和 AsciiDoc 文件

AsciiDoc 文件是具有 `.adoc` 文件扩展名的文本文件（例如，*document.adoc*）。大多数 AsciiDoc 处理器假定文件中的文本使用 UTF-8 编码。仅当文件以 BOM 开头时才支持 UTF-16 编码。

AsciiDoc 处理器可以从字符串（即字符序列）中处理 AsciiDoc。然而，大多数时候你会将 AsciiDoc 文档保存到一个文件中。

重要概念

本页向你介绍在学习 AsciiDoc 时会遇到的一些概念和术语。稍后将在文档中更深入地介绍每个概念。使用此页面作为开始熟悉行话的一种方式。

文档

文档（Document）表示为 AsciiDoc 中的顶级块元素。它由

1. 一个可选的文档标题（document header）

和

以下其中之一

- 前面有一个可选的前言（preamble）的一个或多个章节（section）
- 或仅一系列顶级块

组成。

可以使用文档标题配置文档。文档标题本身不是块，而是为文档提供元数据，例如文档标题（document title）和文档属性（document attributes）。

元素

元素（Element）是文档中可识别、可寻址和可组合的内容块。AsciiDoc 文档仅仅是它包含的所有元素的组合。

元素是类型的层次结构，其中一个元素可能是一组元素的特化。例如，侧边栏块（sidebar block）是块元素，因此它共享所有块元素的特征，并且还添加了一些自己的特征。

元素包括文档本身、章节、块、块宏、分隔符、内联短语和宏。

块元素（block element）垂直（按行）堆叠在其他块元素的上方或下方。块元素通常简称为块（blocks）。块构成文档的主要树结构。

内联元素（inline element）是块元素或其属性之一（例如，块标题）内的一段内容。内联元素包括格式化文本（斜体、粗体等）、内联宏和元素引用。填补这些元素之间空白的是未替换的文本。内联元素的结构不如块状元素，因为它们更倾向于替换而不是树状结构。

属性

属性（attribute）是一个名称/值对，用于存储和声明 AsciiDoc 语言中的元数据。属性可用于影响语法、控制行为、自定义样式、激活或配置集成，或存储内联替换内容。属性真正将 AsciiDoc 与其他轻量级标记语言区分开来。

属性实际上是一个抽象术语。有两种具体的属性分类：文档属性（document attribute）和元素属性（element attribute）。

文档属性

顾名思义，文档属性直接与文档相关联。它们用于在运行时导出有关文档的信息、控制处理器的行为以及存储可重用的值或短语。因此，它们是一种与 AsciiDoc 处理器的双向通信通道。

可以使用属性引用（attribute reference）在内容中引用文档属性（需要提前启用了属性替换）。文档属性可以在文档中使用属性条目（attribute entry）（通常在文档标题中）定义，也可以通过 API 或 CLI 定义。并非所有文档属性都可以修改。

元素属性

元素属性是特定元素的元数据，例如块或内联元素。它们在属性列表中定义并且仅适用于该元素。属性列表的放置形式取决于元素。属性名称可以是字符串（即命名属性）或隐式数字索引（即未命名的位置属性）。

与文档属性不同，元素属性不能直接从文档模型的内容中引用。换句话说，不能使用属性引用来解析元素属性。元素属性可丰富或配置元素的行为，例如应用角色（role）或设置图像的宽度。元素属性是使用元素上的属性列表或像块标题行这样的可用速记来定义的。

宏

在阅读本文档时，您会经常看到对术语宏（macro）的引用。宏是一种用于表示非文本元素的语法或使用提供的元数据扩展为文本的语法。请参阅宏以了解有关该术语含义的更多信息。

以下是一个块宏的示例：

下面是一个内联宏的例子：

AsciiDoc 官方文档详见此处。

你可以将宏视为函数。宏的语法遵循以下形式：一个名称、（有时是可选的）一个目标以及用方括号括起来的，一个由零个或多个元素属性组成的属性列表。

宏有两种变体：块宏（block macro）和内联宏（inline macro）。

- 在块宏中，名称和目标由两个冒号（::）分隔，并且它必须单独占一行。
- 在内联宏中，名称和目标由单个冒号（:）分隔，并且可以与文本和其他内联元素并排放置。
- 块状宏总是被解析，而内联宏只有在启用宏替换的情况下才会被解析。

预处理器指令

AsciiDoc 中还有另一种看起来很像但并不是块宏的语法。这些是预处理器指令（Preprocessor directives）。

预处理器指令是控制输入解析器的行的函数。条件预处理器指令可以根据属性（ifdef、ifndef）或其他任意条件（ifeval）的存在来配置要包含或排除的行。include 指令可以从另一个文件中添加额外的行。

预处理器指令与块宏具有共同的特征。与块宏一样，预处理器指令必须单独占一行。虽然预处理器指令可以访问文档属性，但它不知道周围的上下文。它只是一个线路处理器。与块宏一样，include 指令可以具有元素属性，尽管它们仅适用于预处理操作本身。

文档处理

AsciiDoc 是一种专门的写作格式，而不是一种出版格式。换句话说，它不像你在文字处理器中写作时那样所见即所得。相反，你所写的是 AsciiDoc 的源码。然后你使用 AsciiDoc 处理器，如 Asciidoctor，将 AsciiDoc 源文件转换成可发布的格式。你所要发布的文件就是这种输出。

转换 AsciiDoc 源码是一个解释和润色你的内容的机会，以获得比源文本更多的内容。将 AsciiDoc 源码转换为另一种格式的工作由转换器处理。虽然 AsciiDoc 语言和转换器之间有很强的关系，但这两个方面并没有明确的耦合。

AsciiDoc 处理器提供了几个内置的转换器，包括用于制作 HTML 和 DocBook 的转换器。要激活这些转换器中的一个，你要在文档上设置后台（backend，默认值为 html）。backend 是一个关键词，它告诉处理器你想制作哪种输出格式。然后处理器选择制作该输出格式的转换器。例如，HTML 转换器处理 html 后端来制作 HTML 输出。

简而言之，处理器接受一个字符串（可以从文件中读取），将其解析为一个结构文件，然后产生另一个字符串（可以写入文件）。

规范化

当 AsciiDoc 处理器读取 AsciiDoc 源码时，它做的第一件事就是规范化行。（此操作可以预先执行，也可以在访问每一行时执行）。

规范化包括以下操作：

- 强制编码为 UTF-8（AsciiDoc 处理器总是假设内容是 UTF-8 编码的）
- 从每行中去除尾随空格（包括任何行尾字符）

这种规范化是独立于任何结构化上下文执行的。该行是文字块的一部分还是常规段落的一部分并不重要。所有行都被标准化。

规范化仅在某些情况下应用于包含文件的行。只有具有可识别的 AsciiDoc 扩展名的包含文件才会按上述规范化。对于所有其他文件，仅删除行尾字符。包含文件也可以有不同的编码，这是使用 encoding 属性指定的。如果未指定编码属性，则假定为 UTF-8。

当 AsciiDoc 处理器将行重新组合在一起以生成呈现的文档（HTML、DocBook 等）时，它会在换行符 (\n) 上连接行。

块的简介

块元素（block element）构成了 AsciiDoc 文档的主要结构，首先是文档本身。

什么是块

块元素（block，又名块）是 AsciiDoc 文档中离散的、面向行的内容块。解析后，该内容块将成为已解析文档模型中的块元素。某些块可能包含其他块，所以我们说块可以嵌套。转换器按文档顺序依次访问每个块，将其转换为相应的输出块。

块的形式

AsciiDoc 语法中块边界的定义方式各不相同。某些块的边界，如列表、段落和块宏，是隐式的。其他块具有使用定界符明确标记的边界（即定界块）。主要的共同点是块总是面向行的。

段落块（paragraph block）被定义为一组离散的连续（非空）行。定界块由定界线界定。章节块（section block，又名章节）由章节标题定义，章节标题以一个或多个等号为前缀。该章节包括章节标题行之后的所有内容，直到下一个同级或父章节标题或文档边界。列表块（list block）由一组同级别的列表项定义的，每个列表项由一个标记表示。说明列表块（Description list block）由一组同级列表项定义，每个列表项由一个或多个术语表示。块宏（block macro）由与块宏语法匹配的单行定义。而且文档也是一个块。

一个块（包括它的元数据行）应该总是被两侧的空行或文档边界所包围。

一个块是否支持嵌套块，取决于块的内容模型（以及语法允许的内容）。

内容模型

块的内容模型（content model）决定了该块可以有什么样的内容（如果有的话），以及如何处理这些内容。AsciiDoc 中的块的内容模型如下：

混合（**compound**）

一个可能只包含其他块的块（例如，章节）

简单（**simple**）

被视为段落文本的连续行的块（并且受正常替换的影响）（例如，段落块）

逐字（**verbatim**）

保存逐字文本的块（“按原样”显示）（并进行逐字替换）（例如，列表块）

原始（**raw**）

一个包含未处理内容的块，该块直接传递到输出而没有应用任何替换（例如，直通块）

空的（**empty**）

没有内容的块（例如，图像块）

表格（**table**）

为强制执行固定结构的表保留的特殊内容模型

内容模型是为所有内置语法推断的（由上下文确定），但可以为自定义块配置。块在不同情况下也可能支持不同的内容模型。情况由上下文和样式决定，在分隔块的情况下，结构容器也是如此。

上下文

你可能经常听到用名称引用的块，例如示例块、侧边栏块、警告块或部分。该名称是块的上下文（context）。

让我们考虑以下正常部分：

```
== 章节标题

章节内容
```

此块的上下文是章节。我们通常将其称为章节（或章节块），使用上下文作为形容词来描述块。在这种情况下，作者不必指定上下文，因为语法隐含了它。

每个块都有一个上下文。上下文通常由语法暗示，但在某些情况下可以显式声明。上下文是将一种块与另一种块区分开来的东西。你可以将上下文视为块的类型。

上下文可以使用块样式进一步修改，以创建一个拥有共同类型的块家族，就像警告块和章节那样。我们很快就会介绍这个修改器。

对于块，上下文有时被称为块名称。在谈论自定义块时，这会经常出现。块名称只是另一层抽象。所有内置块名称都映射到一个上下文。但是块扩展可以将任意块名称映射到一个或多个上下文。最终使用哪个上下文取决于扩展的处理方法返回的内容。最后，上下文决定了块如何转换。

上下文通常决定内容模型。例如，所有章节都隐式具有复合内容模型，因为一个章节可能只包含其他块。所有文字块都隐含地具有逐字内容模型，因为此块的目的是呈现逐字输出。

内置上下文的总结

这是 AsciiDoc 中所有内置块的上下文列表。

NOTE 在 Asciidoctor API 中，上下文表示为符号（symbols）。在 Ruby 中，符号是一个以冒号为前缀的名称（例如，`:listing`）。本文档有时会在引用上下文名称时使用此表示法。但是，这种表示法并不通用。一些处理器，例如 Asciidoctor.js，将上下文存储为字符串。

Table 1. 内置上下文

名称	目的
admonition	五个警告块之一。
audio	音频块
colist	标注列表
dlist	说明列表
document	顶级文档或 AsciiDoc 表格单元格中的文档
example	示例块
floating_title	离散的标题
image	图像块
list_item	有序、无序或说明列表中的项目（仅在列表或说明列表块内相关）。在说明列表中，此块用于表示术语和描述。
listing	列表块
literal	文字块
olist	有序列表

名称	目的
open	开放块
page_break	分页符
paragraph	段落
pass	直通块
preamble	文档的序言
quote	引用块
section	章节。也可以是部分、章节或特殊部分。
sidebar	侧边栏块
table	列表块
table_cell	表格单元格（仅在表格块内相关）
thematic_break	分隔线
toc	TOC 块（用于指定自定义 TOC 放置）。
ulist	无序列表块
verse	诗文块
video	视频块

NOTE 每个内联元素也有一个上下文，但这些元素（还）不能从已解析的文档模型中访问。

你可以通过使用块、块宏或内联宏扩展点来引入其他上下文。

块的样式

块的共性