docboon

DocBook简要参考手册

Technical Document Authoring and Publishing Suite

https://docboon.github.io/

DocBook简要参考手册: Technical Document Authoring and Publishing Suite

https://docboon.github.io/
Older Flower

出版日期 发行日期：2022年5月27日

修订历史		
修订 1.0	2022年5月26日	olderflower <olderflower@out look.com>
docorolla		
修订 0.2	2021年9月30日	olderflower <olderflower@out look.com>
基本完稿		
修订 0.1	2021年9月12日	olderflower <olderflower@out look.com>
发布第一个Draft		

摘要

docboon是一个项目。是帮助您如何制作优秀的技术类文档的项目。 目前处于文档阶段，在这份文档中，您可以学会如何利用DocBook标准及相应的程序、字体和样式，去创作和发行自己的文档。

DocBook是一套非常流行的文档标准。在IT行业，有大量的开源社团在使用，包括Fedora、OpenSUSE、Gnome、OpenStack等。也有很多商业性的公司在使用，如IBM、Oracle、Cisco等。DocBook具有丰富的标签，几乎涵盖了出版一本专业的技术书籍（特别是计算机类的书籍）所需要的所有元素。使用Docbook技术，会使我们制作的技术类书籍更为专业，在版式方面更容易让读者阅读。

我们发现了docbook在书写技术文档方面的能力，不仅仅是它具有丰富、人性化的标签，其在多人协作等方面也令人满意。于是我们把一些微不足道的经验分享出来，以飨读者。如果您有任何的意见和想法请告知我们，任何的建议都弥足珍贵。

这份手册就是由Docbook及相关工具来完成的。

目录

插图清单

表格清单

前言

1. 排版约定

本手册使用下列排版约定。

1. **字体约定**，为了更为清晰地、区别于正文地展示一些系统命令、程序代码等文档元素，对这些内容采用与正文有区别的字体样式：

   a. 等宽字体：表示程序片段，以及正文中出现的配置选项、变量、函数名等，我们采用如下的样式显示：`serverip`。

   b. 等宽加粗字体：表示由用户输入的系统命令。例如，用户创建一个目录的命令，用如下的样式来显示：`mkdir mybook`。

   c. 等宽斜体：表示由用户输入的值或是一些需要设定的参数的值，例如，用户设定一个ip地址，用如下的样式显示：*`192.168.1.2`*。

   d. 等宽加粗斜体字体：表示文件名、数据库名及新的术语等。例如，文件的名称用如下的样式来显示：***`myfirst.xml`***。

2. **提示、告诫类信息**（admonitions），一共有5种，分别为：

   - 

     提示

     这种样式表示一种提示类的信息。

   - 

     注意

     这种样式代表一种技巧或是小窍门。

   - 

     重要

     这种样式代表重要的信息。

   - 

     小心

     这种样式代表需要仔细注意的信息。

   - 

     警告

     这种样式代表警告信息。

3. **程序代码及屏幕内容：**

   - 有时候我们需要整块显示在屏幕上输出的内容，或者是键入的命令。例如，在linux下，我们使用`ifconfig eth0`命令来查网络接口信息：

     ```
     $ ifconfig eth0
     ```

```
eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 98:fa:9b:db:b9:9f  txqueuelen 1000  (以太网)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 16  memory 0xea300000-ea320000
```

- 对于一些程序代码，使用带有序号的样式来展示，例如，一段php代码：

```
 1 <?php
 2   class SimpleBook {
 3     private $title;
 4
 5     function __construct($title_in) {
 6       $this->title  = $title_in;
 7     }
 8
 9     function getTitle() {return $this->title;}
10   }
11 ?>
```

2. 相关资源

docbook 5：https://tdg.docbook.org/tdg/5.0/docbook.html

docbook params：http://docbook.sourceforge.net/release/xsl/current/doc/

docbook xsl：http://www.sagehill.net/docbookxsl/index.html

还有如下一些非常有价值的站点供参考：

https://docbook.org/

http://docbook.sourceforge.net/

https://cdn.docbook.org/

https://cdn.docbook.org/

https://github.com/docbook

关于DocBook

在接触到DocBook之前，写一些计算机技术相关的一些文档，对我来说是件很辛苦的事情。

- 该用什么样的一些样式展示不同的一些文档元素？如一些代码、或者系统命令等。
- 如何保证同样的文档元素总是保持一样的显示风格？
- 如何进行版式的一些设计？用word来排版对我来说实在太痛苦。
- 如何让文档能够同时生成在线阅读的版本和可以打印的版本？
- 如何完成多人协作书写，又同时保证样式的完全一致？

还有就是诸如如何进行版本控制等，因为诸多方面的一些困扰，平时也就非常留意其他人的文档在这些方面是如何处理的。在阅读国外的一些技术文档时，发现他们的文档从样式、排版及不同文件的发布（html、pdf及epub等）都非常精美，感觉他们是用某种出版系统自动进行处理的，而非人工制作。

按图索骥，一个称之为**DocBook**的东西走进了我的视野。发现他们都在使用这套东西进行文档的书写、排版、管理和发行。Debian，OpenSuse，Gnome、openstack等很多开源组织都在使用。很多商业公司如IBM、Redhat，Ubuntu，Oracle等也在使用。

DocBook引起了我浓厚的兴趣。于是决定把一些技术性的文档也全部采用DocBook来书写、排版和发行。通过搜索各方面的资料，发现这并不是一件容易的事情。一是这方面的中文资料非常少，二是这方面的英文资料也非常分散。但是，看到那些版式疏朗、眉目清晰、章节分明的精美文档，还是决定一试。于是，就有了这份小册子。

1. What？

1.1. DocBook简介

通俗点讲，DocBook其实算是一种标记语言，与html有点类似，就是使用一些标记，把文档整理、组织在一起，如下所示：

```
<book><title>DocBook简要参考手册</title>
      <subtitle>Technical Document Authoring and Publishing Suite</subtitle>
           ... ...
```

当这些文字被一些固定的标签整理和组织以后，在计算机里就很容易根据不同的标签做不同的处理。其中比较常见的一种应用就是对不同标签的内容按照相应的样式进行显示。DocBook，本质上就是一组标签的集合，而且是专门针对制作“书籍”的标签集合。

DocBook实际上是一种规范、一种标准，在这种规范或标准里规定了有哪些标签可以使用。就像html规范里，有<head>、 、<body>等标签一样。DocBook主要分为SGML和XML两个版本，目前由OASIS[1]的DocBook技术委员会来维护。

[1]OASIS：Organization for the Advancement of Structured Information Standards，结构化信息标准促进组织，是一个推进电子商务标准的发展、融合与采纳的非盈利性国际化组织。自1993年成立开始，OASIS已经发展成为了由来自100多个国家的600多家组织、企业，参与人数超过5000人的国际化组织。

1.2. DocBook历史

DocBook肇始于1991年，主要分为三个阶段。

1. **HaL和O'Reilly阶段**

   DocBook是由HaL计算机系统公司和O'Reilly公司设计和实现的。它最主要的设计目的就是为了保存一些unix文档转换、排版后的结果，这样这些unix文档就能互相交换。O'Reilly公司创建了专门的“计算机文档制作论坛”（即后来的Davenport集团）来修订和维护DocBook。在1994年的时候Davenport成文DocBook的主要维护者，同年，DocBook version 1.2.2发布。

2. **Davenport阶段**

   在Davenport集团的赞助下，DocBook得到了更广泛的应用，作为当时最大的用户Novell和Sun公司对DocBook的设计有了很大的影响。1997年1月时，DocBook的第3个版本发布。

   后来，由于DocBook在XML标准方面的一些原因，DocBook的赞助者期待能建立一个新的机构来接管DocBook。于是，在1998年7月，OASIS DocBook Technical Committee正式成立。

3. **OASIS 阶段**

   被OASIS接管以后，也就意味这DocBook成为了一个文档、书籍制作方面事实上的标准。由于一开始DocBook是用SGML语言实现的，在OASIS的推动下，顺利地推出了XML版本，并且发展良好。这份手册就是遵守DocBook的XML规范来书写的。

其实，在整个DocBook的发展过程中，一个绕不开的名字是**O'Reilly**。这家公司为DocBook的发展做出了非常突出的贡献。从最开始创建DocBook，到目前为止，还在积极地参与很多DocBook方面的工作，DocBook最核心的书籍《DocBook 5: The Definitive Guide》也是由O'Reilly出版的。

我自己购买的大部分计算机专业的书籍很大一部分都是**O'Reilly**出版的。O'Reilly的计算机书籍无论是在内容、版式还是装帧等方面，都是非常棒的。其实从它所出版的书籍里也能看出，DocBook深受其影响。

2. Why ?

DocBook可以说是专为技术类文档而生的。技术类的文档相比文学书籍来说，具有更多的“文档元素”、在版式等方面也有更多的要求。DocBook在这些方面给出了一个比较完美的解决方案。

1. DocBook具有非常丰富的“文档元素”或者说是“标签”，在DocBook 5.0中，根据其官方文档显示，有385个，例如有关于文档组织方面的preface 、chapter 、section等；关于文档出版信息方面的author、copyright、date、legalnotice、revision、adress等；关于文档标注方面的footnote、superscript、tip、important、caution等。这些丰富的元素足以让我们轻松地组织、书写和发布我们的文档。

2. 文档内容与样式分离，文档作者的精力和能力其实是集中在内容方面的，文档的写作者只需要关注文档的内容，即关于自己的文档中的内容是什么元素即可，相应的表现样式已经都提前定义好。

3. 单一的源文件，DocBook是定义了一套XML的标签规范，所有的以DocBook规范书写的文档都被保存为纯文本的XML文件。在发布时可以根据需求发布为不同的格式，如PDF或者是HTML。并且因为是结构良好的XML文本文件，也有利于对文档的各种重新组织和复用。

4. 多人协作，有时候多人写作一本书，当所有的章节汇聚在一起时，往往要花费大量的精力来进行重新的编制目录、插图目录、表格目录等，不但耗时而且容易出错。DocBook规范的XML文件在这方面具有很大优势，所有的编目均可自动生成，不但大方美观，而且效率高、不容易出错。

5. 版本控制，以DocBook规范书写的源文件是XML格式的文本文件，很容易使用SVN或git等版本控制工具将其放入版本库中，对各个时间段或时期的文档进行版本管理。

3. How？

要将DocBook规范的XML文件输出为PDF或是HTML格式的文件，除了书写XML文件之外，还需要XSL样式表和相应的处理程序。由处理程序读取XML文件，并根据XSL样式表的定义，将XML源文件转换为不同格式的文件如PDF、HTML或是epub文件。

第 1 章 制作第一份DocBook文档

前面已经提到过，DocBook主要分为SGML和XML两个版本，不论哪个版本，都称为DocBook DTD（DocBook Document Type Definition ，即DocBook文档类型定义）。在DocBook DTD中，详细的描述了所有的标签。在这份文档中，我们使用的是XML版本的DocBook　DTD。从目前的状态来看，SGML版本使用的已经非常少。

DocBook的入门是有一定门槛的，需要很多方面的配合，才能产出一份不错的文档。但一旦入门，并开始使用以后，对后期的文档制作效率和质量都是有极大提高的。要制作一份精美的DocBook文档，需要如下的一些组件：

1. **XML文档**

   首先，需要按照DocBook DTD的规范要求，书写我们的xml文档。这是一件容易的事情，虽然有很多的标签，但每个标签的含义都非常明显，并且DocBook的官方网站给出了所有标签的解释，以及它的父节点和子节点。下面的示例就是这个小节真实的样子：

   ```
   <chapter><title>制作第一份DocBook文档</title>
     <para>前面已经提到过... ...</para>
     <para>DocBook的入门是有一定门槛的... ...</para>
   ```

   DocBook 5的标签请参考这个站点：https://tdg.docbook.org/tdg/5.0/docbook.html

2. **XSL样式表**

   有了DocBook格式的xml文件之后，我们需要将这些xml转换成人们容易阅读的格式，如html或pdf格式。那不同标签所包含的内容用什么样的样式来显示呢？ 这就需要一套样式表。OASIS组织之外的另外的一家公司Sagehill　Enterprises开发了一套完整的XSL样式表，用于将DocBook格式xml文件转换为html、pdf、rtf或epub格式。

   这份样式表的名叫*DocBook-xsl*，目前版本号是1.79.2。如果产出的文档要达到我们想要的效果，我们需要对这个样式表进行参数调整及对相应样式进行再设计，这是一份需要点力气的工作。但是这个工作一旦完成，后面就可以长久享用了。我用的样式已经7、8年没变过了，最近只是把字体更换了一下。

   样式表有关内容请参考：http://www.sagehill.net/docbookxsl/index.html

3. **字体**

   有几种类型的字体需要我们提前准备好主要包括sans、serif、monospace等。后面会有详细的介绍。

4. **处理程序**

有了DocBook格式的文件、DocBook-xsl样式表及字体，那还需要一些程序将这些东西生产出我们需要的html或pdf文件。主要包括xsltproc、saxon和fop等。

1. 基础环境搭建

就像前面所描述的，我们首先需要搭建一个基础的环境。这个环境的搭建比较简单，没有一些额外的安装步骤，只需要我们把相关的样式表、程序等下载并解压缩，然后放置在合适的目录中就可以了。下表是需要的组件清单：

表 1.1. software

No.	Name	Version
1	操作系统 OS	Debian Linux 12 Bookworm
2	DocBook DTD	DocBook-5.0
3	DocBook-xsl	DocBook-xsl-1.79.2
4	java	jre-8u301-linux-x64
5	saxon	saxon6-5-5
6	xalan	xalan-j_2_7_2-bin
7	fop	fop-2.6-bin
8	xsltproc	1.1.34
9	fonts	思源字体

为了叙述方便，我们建立一个专门的目录*docware*来存放这些组件。

1. DocBook DTD

   下载地址：https://DocBook.org/xml/5.0/DocBook-5.0.zip

   这个DTD其实也可以不用下载，这里面就是DocBook DTD。下载好解压缩以后，*DocBook.dtd*文件在*dtd*的目录里。

2. DocBook xsl

   下载地址：https://sourceforge.net/projects/DocBook/files/DocBook-xsl/1.79.1/DocBook-xsl-1.79.1.tar.bz2/download

   上面的链接给出的版本是DocBook-xsl-1.79.1的版本，截止这份文档完稿，DocBook-xsl已经发展到了版本1.79.2。并且，项目寄存地址已经从sourceforge.net迁移到了github.com。如果需要最新版本的DocBook-xsl，可以到github上去下载。地址是https://github.com/DocBook

3. java

   下载地址：https://www.java.com/zh-CN/download/

   这份文档中涉及到的程序基本上都是由java开发的，所以我们需要java。

4. saxon

下载地址：https://sourceforge.net/projects/saxon/files/saxon6/6.5.5/saxon6-5-5.zip/download

Saxon是一个基于java的XSLT处理器，因其严格遵守XSLT标准而受到很多人的欢迎。saxon的作用是将我们DocBook规范的文档，根据xsl样式表转换为我们需要的格式，比如html、fo[1]（pdf格式需要fo）等。

saxon版本很多，截至这份文档完稿，它的最新版本是10。关于saxon的更多信息可以参考它的官方网站：http://saxon.sourceforge.net/。在DocBook-xsl的相关手册里，　　对saxon6.5描述较多，所以我选择了这个比较稳定但似乎又年代久远的sanon6-5.5。

与saxon功能类似的、在DocBook-xsl文档中提到的另一个软件是Apache软件基金会的xalan。在saxon的使用过程中似乎也没遇到什么大的问题，所以就没有使用。

5. xalan

下载地址：hhttps://dlcdn.apache.org/xalan/xalan-j/binaries/xalan-j_2_7_2-bin.tar.gz

xalan与saxon的作用类似。xalan是apache软件基金会的一个项目。xalan与saxon可以二选一。

6. fop

下载地址：https://archive.apache.org/dist/xmlgraphics/fop/binaries/fop-2.4-bin.tar.gz

fop是Apache软件基金会下的一个xml相关的项目，项目网站是：http://xmlgraphics.apache.org/fop/。fop的作用是将saxon生成的fo文件转换为pdf文件。截至这份文档完稿，它的最新版本是2.6。我选择了2.4版本，并没有使用最新版。

fop在我们需要的这些组件和软件里面，迭代是比较频繁的，特性增加了很多，易用性也提高了不少。记得最早使用fop的时候，在使用一些字体时，还需要将字体ttf等文件生成一份xml文件才行，而现在已经完全不需要了，只需要将字体复制到指定的目录，然后在配置文件*fop.xconf*里声明一下就可以了。如果字体文件规范，甚至都不需要声明。这个部分会在后面的配置部分介绍。

7. xsltproc

用于处理一些xsl的模板文件，在样式模板相关章节会用到。安装方法是直接使用debian的安装命令就可以了：

```
apt-get install xsltproc
```

8. 字体

字体是个有点大的问题，放在后面章节介绍。在这里我们需要三款字体，我使用的SourceHan（思源）字体族。我在发现思源字体之前有很多字体方面的问题困扰我。直到我发现了思源字体，很多问题才一扫而空。思源字体的背后是一个世界级的制作团队。主要受到adobe的赞助。而且它是开源的。它的官方网站是https://source.typekit.com/　　，一共有三款字体需要下载：source-han-sans，source-han-serif和source-han-mono。这三个文件都比较大，在国内的话可能要下载很长时间。我下载了差不多一个多星期，断了又下，下了又断的。

[1]fo = Formatting Objects。格式化对象，简单来说，就是xsl处理程序通过格式对象（fo）来决定哪些元素被放置在页面的哪些位置。这是一个由w3c维护的xsl标准的一部分。https://www.w3.org/TR/2001/REC-xsl-20011015/slice6.html#fo-section

下载地址：https://github.com/adobe-fonts/

下载好之后，把三个压缩文件解压，选取下面的三个字体：

a. source-han-sans-2.004R/SubsetOTF/CN/SourceHanSansCN-Regular.otf

b. source-han-serif-1.001R/SubsetOTF/CN/SourceHanSerifCN-Regular.otf

c. source-han-mono-1.002/Regular/OTC/SourceHanMonoSC-Regular.otf

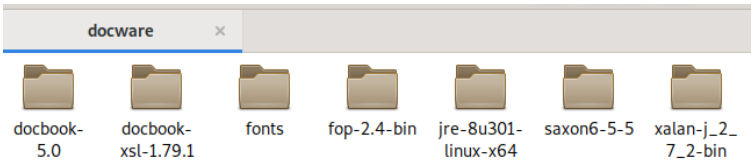
把这三个字体保存到docware中的fonts目录中。

提示

可能会有人问，对于这些组件版本的选择有什么讲究？其实也有一个大致的原则，首先不用最新版，其次是我重点参考了Debian 12和Ubuntu 20的版本库里相应的版本。保证这些组件所使用的版本都是经过稳定的系统测试过的。

将这些组件下载到*docware*中，然后全部解压缩，基本的一个环境就算是搭建好了。后期还需要对fop的配置文件*fop.xconf*及XSL样式表进行很多的配置，到这里我们先把这些事情暂时一放。先写出第一份文档，然后再回过头来处理这些工作。

图 1.1. docware目录中的组件



2. 书写DocBook格式的xml文件

只要对DocBook的标签有少许的了解，就可以开始书写基于DocBook规范的文档，前面已经列举了两个例子。

我们就用下面这份简单的例子作为第一份DocBook文档，把这份文档命名为*firstbook.xml*

```
<book><title>docbook简要参考手册</title>
  <subtitle>Technical Document Authoring and Publishing Suite</subtitle>

  <chapter><title>制作第一份docbook文档</title>
    <para>前面已经提到过，docbook主要分为SGML和XML两个版本，不论哪个版本，都称为DocbookDTD... ...</para>
    <para>docbook的入门是有一定门槛的，需要很多方面的配合，才能产出一份不错的文档。... ...</para>
    <sect1><title>基础环境搭建</title>
      <para>... ...</para>
    </sect1>

    <sect1><title>书写docbook格式的xml文件</title>
      <para>... ...</para>
    </sect1>

    <sect1><title>样式表简单调整</title>
      <para>... ...</para>
    </sect1>

  </chapter>
```

```
20
21   <chapter><title>docbook标签及文档类型</title>
22     <para>... ...</para>
23   </chapter>
24
25 </book>
```

上面这段并不长的代码足以让我们演示DocBook文档的结构。文档中只有如下几种标签：

1. **书 book**

   这可能是DocBook文档中最常见的文档起点。也是我们用得最多，但也是输入次数很少的标签。“一本书”只需输入两次，一头一尾：<book> </book>。

2. **章 chapter**

   book的子节点之一，“章”的根节点。与chapter节点同级的还有preface（序言）和appendix（附录）。

3. **节 sect***

   sect1是“小节”的根节点，“小节”一共有5个级别，一级套一级，1套2，2套3..... sect1、sect2、sect3、sect4和sect5。如果不喜欢这5个标签，也可以使用section标签递归嵌套。

4. **段落**

   para就是paragraph的缩写。应该是用得最多的标签了吧。

5. **标题**

   title是在文档中应用非常广泛的标签，不仅是book,chapter,sect*等，在后面还会用到的表格、插图等都可以有一个标题。

如果没有什么特别的要求，这样一份单纯文字性的“书”就算准备好了。接下来，我们就可以将这个文件转换为我们需要的格式。

3. 生成我们需要的文件

将*firstbook.xml*文件转换为我们需要的格式，如html或者pdf，需要相应的DocBook-xsl样式表及saxon和fop程序。

样式表目录*DocBook-xsl-1.79.1*下有很多目录和文件，下面的表格列出了一些会常用的：

表 1.2. 样式表目录及作用

No.	目录名	用途
1	fo	将xml文件转换为fo文件，然后再将fo文件转换为pdf格式的文件
2	html	将xml文件转换为html文件
3	epub	将xml文件转换为epub（电子书）文件
4	htmlhelp	将xml文件转换为htmlhelp文件
5	slides	制作幻灯片（类似ppt）的样式表

3.1. 转换为html文件

转换为html文件比较简单，如果只是转换为可以在线阅读的文档，在样式方面没有特别需求的话，使用saxon，按照下面的命令执行就可以了：

1. **转换为单页html**

   将*firstbook.xml*转换为单页html文件，使用*DocBook-xsl-1.79.1/html/DocBook.xsl*文件，命令如下：

```
docware/jre-8u301-linux-x64/jre1.8.0_301/bin/java -cp \
          docware/saxon6-5-5/saxon.jar:\
          docware/DocBook-xsl-1.79.1/extensions/saxon65.jar:\
          com.icl.saxon.StyleSheet -o firstbook.html firstbook.xml \
          docware/DocBook-xsl-1.79.1/html/DocBook.xsl
```

   命令执行结束，会生成*firstbook.html*文件。

2. **转换为多页html**

   转换为多个html文件，使用DocBook-xsl-1.79.1/html/chunk.xsl文件，命令如下：

```
docware/jre-8u301-linux-x64/jre1.8.0_301/bin/java -cp \
          docware/saxon6-5-5/saxon.jar:\
          docware/DocBook-xsl-1.79.1/extensions/saxon65.jar:\
          com.icl.saxon.StyleSheet firstbook.xml \
          docware/DocBook-xsl-1.79.1/html/chunk.xsl
```

   命令执行结束，会生成以*index.html*为索引的一族html文件。

3.2. 转换为pdf文件

如果html文件适合在线浏览的话，那对于有打印需求的人来说，pdf可能是最好的选择。从xml转换到pdf可能要麻烦一点，需要先使用saxon将xml文件转换为fo文件，然后再使用fop将fo文件转换为pdf文件。为了能得到显示良好的pdf文件，在转换之前需要先做两件准备工作。一是配置*fop.conf*文件，二是对针对fo的DocBook-xsl做适当的调整。

3.2.1. 配置fop.conf文件

配置fop.conf的主要目的就是能让合适的字体加载到pdf文件中，如果找不到合适的字体，pdf文件可能会显示乱码。在目录*fop-2.4-bin/fop/conf/*中，有一个*fop.xconf*文件，将这份文件复制一份，并将复制后的文件重新命名为*fop.conf*。并对这个文件做两处修改：

1. 在<base>.</base>下添加一行，如下：

```
<base>.</base>
<font-base>/usr/share/fonts</font-base>
```

   这一行的意义是让fop可以到这个目录里去寻找字体。需要根据字体的具体目录来设定。

2. 在<fonts></fonts>节点中添加如下的字体信息：

```
<auto-detect/>
```

```
<font kerning="yes" embed-url="/usr/local/docware/fonts/SourceHanSerifCN-Regular.otf">
  <font-triplet name="Source Han Serif CN" style="normal" weight="normal"/>
</font>

<font kerning="yes" embed-url="/usr/local/docware/fonts/SourceHanSansCN-Regular.otf">
  <font-triplet name="Source Han Sans CN" style="normal" weight="normal"/>
</font>

<font kerning="yes" embed-url="/usr/local/docware/fonts/SourceHanMonoSC-Regular.otf">
  <font-triplet name="Source Han Mono SC" style="normal" weight="normal"/>
</font>
```

经过这样简单的配置，字体就在fop中配置好了。其中的auto-detect是自动检测系统中安装的字体。后面三个font节点配置的就是前面我们下载的三个字体。

fop详细的配置信息请参考：https://xmlgraphics.apache.org/fop/2.4/configuration.html

3.2.2. 设置fop脚本

fop设置的另外一个工作，就是需要在*fop-2.4-bin/fop/fop*脚本中添加classpath。

```
#! /bin/sh
#
#   Licensed to the Apache Software Foundation (ASF) under one or more
#   ...
#   This script was modified by ChunXin <484299@qq.com>, 2021/09

DOCWARE_HOME="/usr/local/docware"

JAVA_HOME="$DOCWARE_HOME/jre-8u301-linux-x64/jre1.8.0_301"
FOP_HOME="$DOCWARE_HOME/fop-2.6-bin/fop"
XSL_EXTENSIONS="$DOCWARE_HOME/docbook-xsl-1.79./extensions"

CLASSPATH="$CLASSPATH:\
$FOP_HOME/build/fop.jar:\
$FOP_HOME/build/fop-hyph.jar:\
$FOP_HOME/lib/batik-all-1.14.jar:\
$FOP_HOME/lib/commons-io-1.3.1.jar:\
$FOP_HOME/lib/commons-logging-1.0.4.jar:\
$FOP_HOME/lib/fontbox-2.0.19.jar:\
$FOP_HOME/lib/serializer-2.7.2.jar:\
$FOP_HOME/lib/xalan-2.7.2.jar:\
$FOP_HOME/lib/xercesImpl-2.12.0.jar:\
$FOP_HOME/lib/xml-apis-1.4.01.jar:\
$FOP_HOME/lib/xml-apis-ext-1.3.04.jar:\
$FOP_HOME/lib/xmlgraphics-commons-2.6.jar"

export JAVA_HOME FOP_HOME XSL_EXTENSIONS CLASSPATH
```

将上面的这段代码加在*fop-2.4-bin/fop/fop*的前面。

3.2.3. DocBook-xsl参数调整

关于样式表参数的调整，主要是调整*DocBook-xsl-1.79.1/fo/param.xsl*里的一些内容

```
1 <xsl:param name="l10n.gentext.default.language">zh_cn</xsl:param>
2 <xsl:param name="l10n.gentext.language">zh_cn</xsl:param>
3
4 <xsl:param name="paper.type">A4</xsl:param>
5
```

```
 6 <xsl:param name="body.font.family">Source Han Serif CN,serif</xsl:param>
 7 <xsl:param name="title.font.family">Source Han Sans CN,sans</xsl:param>
 8 <xsl:param name="monospace.font.family">Source Han Mono SC,monospace</xsl:param>
 9
10 <xsl:param name="line-height">1.7</xsl:param>
```

上面的这段代码主要是对语言、版面大小、字体和行间距进行了调整。

3.2.4. 生成pdf文件

*fop.conf*和样式表配置好以后，就可以生成pdf格式的文件了。分为两步：

1. **生成fo文件**

   第一步首先是要生成fo文件，与生成单页html文件类似，使用saxon，只需要把样式表*html/DocBook.xsl*更换为*fo/DocBook.xsl*：

   ```
   docware/jre-8u301-linux-x64/jre1.8.0_301/bin/java -cp \
            docware/saxon6-5-5/saxon.jar:\
            docware/DocBook-xsl-1.79.1/extensions/saxon65.jar:\
            com.icl.saxon.StyleSheet -o firstbook.fo firstbook.xml \
            docware/DocBook-xsl-1.79.1/fo/DocBook.xsl
   ```

2. **将fo文件转换为pdf**

   然后使用fop程序将fo文件转换为pdf文件：

   ```
   fop-2.4-bin/fop/fop firstbook.fo firstbook.pdf -c fop-2.4-bin/fop/conf/fop.conf
   ```

第 2 章 制作更完整的DocBook文档

前面通过*firstbook.xml*的例子展示了一份docbook xml文档基本的样子。并将其转换为了html和pdf文件。接下来需要我们做更多细致的工作，使文档更加完善、完美。

1. 文档声明

一份XML文档的声明部分通常由可选的XML声明、可选的DTD声明（可能还会包括一些内部的实体元素定义）和文档根元素组成。

1. **xml文档声明**

   XML文档通常以XML声明开始，通常包含版本、编码格式等，例如：

   ```
   <?xml version="1.0" encoding="utf-8"?>
   ```

   版本和编码格式会告诉处理程序采用什么的方式来处理我们的xml文本。一般情况下，xml的处理程序默认version是1.0，encoding是uft-8或utf-16，在这种情况下，我们可以不用写xml文档声明。如果使用xml1.1格式或其他的编码格式那就必须声明。

   无论什么版本和编码，建议保留完整的声明。这是一种良好的习惯。

2. **docbook DTD文档声明**

   docbook DTD是docbook xml的文档声明，这也不是必须的。但，历史的原因，几乎所有的docbook xml文件都包含了这个部分。docbook DTD部分可以清晰地告诉我们使用了哪个版本的DTD及文档的根元素是什么，这个声明紧随xml声明之后，如下的样子：

   ```
   <?xml version='1.0' encoding="utf-8"?>
   <!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook V5.0//EN"
                     "http://www.oasis-open.org/docbook/xml/5.0/docbook.dtd">
   ```

   上面的声明告诉我们，文档的根元素是book，使用了5.0版本的dtd。在某些程序处理xml文档时，需要验证相关标签是否存在或者是否符合规范，那这个docbook.dtd就会被读取。如果我们已经下载这个docbook.dtd到我们的电脑里，也可以用将它指向本的的文件，如下：

   ```
   <?xml version='1.0' encoding="utf-8"?>
   <!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook V5.0//EN"
                     "/usr/local/docware/docbook-5.0/dtd/docbook.dtd">
   ```

   提示

   上面的例子根元素是book，另外常用的两个标签是article和set。article是一篇文章，篇幅上比book要小。set是book的合集，篇幅上更大。

3. **根元素**

根元素紧跟在文档类型声明之后，和docbook DTD的声明类似，这个部分也是可选的。但有时候很多程序会调用这些信息，而且docbook从版本5以后直接使用了一些xlink里的一些特性，所以建议保留这些内容：

```
<?xml version='1.0' encoding="utf-8"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook V5.0//EN"
                  "/usr/local/docware/docbook-5.0/dtd/docbook.dtd">

<book xmlns:xlink='http://www.w3.org/1999/xlink'
      xmlns="http://docbook.org/ns/docbook"
      xml:lang="zh_cn"
      version="5.0">
```

这些看似废话一样的声明其实是大有用处的，主要是一些程序会从这些声明里面读取很多信息，以用于文档的处理。所以，建议将这部分固定为自己的docbook xml文件的“头”部分。那么前面的*firstbook.xml*就写成如下非常正规的样子：

```
<?xml version='1.0' encoding="utf-8"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook V5.0//EN"
      "/usr/local/docware/docbook-5.0/dtd/docbook.dtd">

<book xmlns:xlink='http://www.w3.org/1999/xlink'
      xmlns="http://docbook.org/ns/docbook"
      xml:lang="zh_cn"
      version="5.0">

  <title>docbook简要参考手册</title>
  <subtitle>Technical Document Authoring and Publishing Suite</subtitle>

  <chapter><title>制作第一份docbook文档</title>
    <para>前面已经提到过，docbook主要分为SGML和XML两个版本，不论哪个版本，都称为DocbookDTD... ...</para>
    <para>docbook的入门是有一定门槛的，需要很多方面的配合，才能产出一份不错的文档。... ...</para>
    <sect1><title>基础环境搭建</title>
      <para>... ...</para>
    </sect1>

    <sect1><title>书写docbook格式的xml文件</title>
      <para>... ...</para>
    </sect1>

    <sect1><title>样式表简单调整</title>
      <para>... ...</para>
    </sect1>

  </chapter>

  <chapter><title>docbook标签及文档类型</title>
    <para>... ...</para>
  </chapter>

</book>
```

2. 将文档物理分割

前面所展示的*firstbook.xml*这个文件只是一个架构，如果所有的内容全部写好后，是非常长的一个文件。一个人编写非常大的一个文件是非常恼人的事情，更不用说多人来共同创作了。所以我们需要将文档进行分割，通常情况下，会把一个chapter或一个section放在一个文件里面，例

如，*firstbook.xml*，我们把它分为四章，这样就变成了5个文件，一个*firstbook.xml*和四个“章文件”。下面是我们这份手册的目录树。

```
chunxin@CC$ tree firstbook/
firstbook/
├── chapters
│   ├── chapter01_beginning.xml
│   ├── chapter02_xml.xml
│   ├── chapter03_xsl.xml
│   └── chapter04_fonts.xml
├── firstbook.xml
├── colophon.xml
├── info.xml
└── preface.xml
```

要完成这种分割，需要做两件事，在*firstbook.xml*定义各个章节文件，然后单独完成每个章节的编撰。在目录树中，还有几个其他的文件：*info.xml*、*preface.xml*和*colophon.xml*，他们分别是关于“书”的一些信息、“序言”和“后记”。

1. 各个文件的定义

这个部分是在前面我们描述的“头”部分完成的，在根元素之前，在DTD声明之后。定义好之后，在适当的位置引用他们，那么我们的*firstbook.xml*l真实的样子就诞生了，如下的样子：

```
 1 <?xml version="1.0" encoding="utf-8"?>
 2 <!DOCTYPE book [
 3     <!ENTITY info SYSTEM "info.xml">
 4     <!ENTITY preface SYSTEM "preface.xml">
 5 
 6     <!ENTITY chapter01_beginning SYSTEM "chapters/chapter01_beginning.xml">
 7     <!ENTITY chapter02_xml SYSTEM "chapters/chapter02_xml.xml">
 8     <!ENTITY chapter03_xsl SYSTEM "chapters/chapter03_xsl.xml">
 9     <!ENTITY chapter04_fonts SYSTEM "chapters/chapter04_fonts.xml">
10 
11     <!ENTITY colophon SYSTEM "chapters/colophon.xml">
12 ]>
13 
14 <book xmlns:xlink='http://www.w3.org/1999/xlink'
15       xmlns="http://docbook.org/ns/docbook"
16       xml:lang="zh_cn"
17       version="5.0">
18 
19 <title>docbook简要参考手册</title>
20 <subtitle>Technical Document Authoring and Publishing Suite</subtitle>
21 &info;
22 &preface;
23 
24 &chapter01_beginning;
25 &chapter02_xml;
26 &chapter03_xsl;
27 &chapter04_fonts;
28 
29 &colophon;
30 </book>
```

2. 单独书写各个章节

分割好以后就有点文档工程的味道了。接下来就是完成各个章节的撰写。章节的撰写相对*firstbook.xml*来讲就相对简单里，不需要那些定义和声明，只要按照标准的docbook规范来书写就可以了。在这里以第一章为例来说明，如下：

```
1 <chapter><title>制作第一份docbook文档</title>
```

```
  <para>前面已经提到过，docbook主要分为SGML和XML两个版本，... ...</para>
  <para>docbook的入门是有一定门槛的，... ...</para>
  <orderedlist>
    <listitem><emphasis>XML文档</emphasis>
      <para>首先，需要按照docbook DTD的规范要求，... ...</para>
    </listitem>
    <listitem><emphasis>XSL样式表</emphasis>
      <para>有了docbook格式的xml文件之后，... ...</para>
    </listitem>
    <listitem><emphasis>字体</emphasis>
      <para>有几种类型的字体需要我们提前准备好主要包括sans、serif、monospace等。后面会有详细的介绍。</para>
    </listitem>
    <listitem><emphasis>处理程序</emphasis>
      <para>有了docbook格式的文件、docbook-xsl样式表及字体，... ...</para>
    </listitem>
  </orderedlist>
</chapter>
```

其他的章节也是按照这样的方式来撰写。

3. info.xml

这个文件里面包含了“书”的很多信息，主要是作者、版本信息、发行信息、出版日期等。

```
<info>

  <corpauthor>****信息科技有限公司</corpauthor>
  <author><firstname>Li </firstname><surname>Lawrence</surname></author>
  <pubdate>2021年9月18日</pubdate>

  <copyright>
    <year>2021</year>
    <holder>ChunXin Li</holder>
  </copyright>

  <legalnotice><title>legalnotice </title>
    <para>... ...</para>
  </legalnotice>

  <revhistory>
     <revision>
        <revnumber>1.0</revnumber>
        <date>2021年8月28日</date>
        <authorinitials>ddpt</authorinitials>
        <revremark>发布第二个Draft</revremark>
     </revision>

     <revision>
        <revnumber>1.0</revnumber>
        <date>2021年9月8日</date>
        <authorinitials>ddpt</authorinitials>
        <revremark>发布第一个Draft</revremark>
     </revision>
  </revhistory>

  <releaseinfo><para>This document was generated by chunxin.</para></releaseinfo>
  <abstract><title>摘要</title>
    <para>docware是基于docbook的一套文档发布工具... ...</para>
  </abstract>
</info>
```

这个例子里只是展示了一部分，关于书的信息（标签）还有很多，可以根据自己的需求添加或减少。

4. preface.xml和colophon.xml

这两个文件是一本“书”的序言和后记。一般情况下也不是必须的。可以把喜欢的一些“引子”之类的东西写在序言里，把后续要做的或一些花絮写在后记里。

在book里，可以把preface和colophon标签看成和chapter是一样的，preface.xml大致如下的样子：

```
<preface><title>写在前面</title>
  <para>在接触到docbook之前，写一些计算机技术相关的一些文档，对我来说是件很辛苦的事情。</para>
  <itemizedlist>
    <listitem>该用什么样的一些样式展示不同的一些文档元素？如一些代码、或者系统命令等。</listitem>
    <listitem>如何保证同样的文档元素总是保持一样的显示风格？</listitem>
    <listitem>如何进行版式的一些设计？用word来排版对我来说实在太痛苦。</listitem>
    <listitem>如何让文档能够同时生成在线阅读的版本和可以打印的版本？</listitem>
    <listitem>如何完成多人协作书写，又同时保证样式的完全一致？</listitem>
  </itemizedlist>
  <para>还有就是诸如如何进行版本控制等... ...</para>
</preface>
```

colophon.xml大致如下的样子：

```
<colophon><title>结束语</title>
 <para>... ...</para>
</colophon>
```

3. 认识更多的docbook标签

3.1. 关于“文档种类”的标签

从我们平时使用的角度来看，可以把docbook的文档种类分为三个部分，article、book和set。

1. article

   article从篇幅上来说一般会比book要小，写一篇小的文章，这种类型比较合适。article标签可以说是和chapter标签是平级的。前面描述的xml头文件和物理分割特性同样适用于article。与book不同的是，article由section（sect1）组成。

2. book

   book就是“书”，前面的*firstbook.xml*就是典型的book。它是由章节组成。

3. set

   set是书的合集，可以把多本书放在一起，就像book把多个chapter放在一起一样。

除了set、book、article之外，还有很多这种逻辑上的文档组织类标签，包括：
part、appendix、glossary、refentry等。

3.2. 认识更多的“文档元素”

3.2.1. list

list类的文档元素在docbook里一共有8种，这里我们只讲两种，其余的请参考tdg文档。

1. itemizedlist，没有编号，每个条目用小圆点标注的列表，它的子节点是listitem，比如下面的例子

```
<itemizedlist>
  <listitem>该用什么样的一些样式展示不同的一些文档元素？如一些代码、或者系统命令等。</listitem>
  <listitem>如何保证同样的文档元素总是保持一样的显示风格？</listitem>
  <listitem>如何进行版式的一些设计？用word来排版对我来说实在太痛苦。</listitem>
  <listitem>如何让文档能够同时生成在线阅读的版本和可以打印的版本？</listitem>
  <listitem>如何完成多人协作书写，又同时保证样式的完全一致？</listitem>
</itemizedlist>
```

上面这个例子就是这份文档序言里的内容。

2. orderedlist，这是编号的list，它的子节点是listitem，比如下面的例子

```
<orderedlist>
  <listitem>该用什么样的一些样式展示不同的一些文档元素？如一些代码、或者系统命令等。</listitem>
  <listitem>如何保证同样的文档元素总是保持一样的显示风格？</listitem>
  <listitem>如何进行版式的一些设计？用word来排版对我来说实在太痛苦。</listitem>
  <listitem>如何让文档能够同时生成在线阅读的版本和可以打印的版本？</listitem>
  <listitem>如何完成多人协作书写，又同时保证样式的完全一致？</listitem>
</orderedlist>
```

显示出的效果如下：

a. 该用什么样的一些样式展示不同的一些文档元素？如一些代码、或者系统命令等。

b. 如何保证同样的文档元素总是保持一样的显示风格？

c. 如何进行版式的一些设计？用word来排版对我来说实在太痛苦。

d. 如何让文档能够同时生成在线阅读的版本和可以打印的版本？

e. 如何完成多人协作书写，又同时保证样式的完全一致？

3. simplelist，没有任何“修饰”，它的子节点是member，比如下面的例子

```
<simplelist type='vert' columns='3'>
    <member>bibliolist</member>
    <member>calloutlist</member>
    <member>glosslist</member>
    <member>itemizedlis</member>
    <member>orderedlist</member>
    <member>segmentedlist</member>
    <member>simplelist</member>
    <member>variablelist</member>
</simplelist>
```

显示样式如下：

```
bibliolist                      itemizedlis                     simplelist
calloutlist                     orderedlist                     variablelist
glosslist                       segmentedlist
```

4. variablelist，一个列表，其中每个条目都由一组一个或多个术语和一个相关的描述组成。比如下面的例子

```
<variablelist><title>Font Filename Extensions</title>
      <varlistentry>
       <term><filename>TTF</filename></term>
        <listitem>
         <para>TrueType fonts.</para>
        </listitem>
    </varlistentry>
      <varlistentry>
      <term><filename>PFA</filename></term>
        <term><filename>PFB</filename></term>
      <listitem>
          <para>
          PostScript fonts. <filename>PFA</filename> files are common
            on <acronym>UNIX</acronym> systems, <filename>PFB</filename>
          files are more common on Windows systems.
          </para>
      </listitem>
      </varlistentry>
   </variablelist>
```

显示样式如下：

Font Filename Extensions

TTF TrueType fonts.

PFA, *PFB* PostScript fonts. *PFA* files are common on UNIX systems, *PFB* files are more common on Windows systems.

3.2.2. inline

可以称之为内联元素。这一类的标签非常多，主要是有一些特殊的技术性内容会出现在正文的内容里，但它的字体可能要发生一些变化，以引起读者的注意。或者通过字体或大小上的一些固定样式，在读者看到时，一下就可以分辨出它是什么类的内容。比如一些系统的命令、文件名、命令选项及参数等。

我们以系统命令iptables-save来举例说明，我们需要用这个命令将地址转换表（nat）保存到一份文件中，在xml文件中输入如下的文字：

```
<command>iptables-save</command> <option>-t</option> <parameter>nat</parameter>
                                  <option>-f</option> <filename>nat.rules</filename>
```

那么就会得到如下的显示样式：

```
iptables-save -t nat -f nat.rules
```

通常情况下，像这样的一行命令，我们基本上只用command标签。在这里只是想用一个例子来说明这些inline的元素。类似这样的标签，还有数据库database、函数function、选项option、参数parameter等等。

3.2.3. admonitions

admonitions是一种较为醒目的提醒类信息，一共有5个标签， caution, important, note, tip, 和warning。在前面，已经有过一个tip的例子。下面我再看一个warning的例子，我们如果有如下的一段代码：

```
<warning>请小心使用iptables-save命令。</warning>
```

那么就会得到如下的显示样式:

警告

请小心使用iptables-save命令。

3.2.4. Line-specific environments

这个具体怎么翻译我还真拿不准，就不翻译了。这类标签不算多，大概7到8个的样子。用一个例子来说明吧。比如，我们有一个屏幕上的输出内容，想直接呈现给读者，就像前面的tree fisrtbook输出，我们可以使用screen标签，如下：

```
<screen>
chunxin@CC$ tree firstbook/
firstbook/
├── chapters
|   ├── chapter01_beginning.xml
|   ├── chapter02_xml.xml
|   ├── chapter03_xsl.xml
|   └── chapter04_fonts.xml
├── firstbook.xml
├── colophon.xml
├── info.xml
└── preface.xml
</screen>
```

那么显示出来，就是如下的样子：

```
chunxin@CC$ tree firstbook/
firstbook/
├── chapters
|   ├── chapter01_beginning.xml
|   ├── chapter02_xml.xml
|   ├── chapter03_xsl.xml
|   └── chapter04_fonts.xml
├── firstbook.xml
├── colophon.xml
├── info.xml
└── preface.xml
```

显示的结果就感觉像是直接搬用了屏幕上的输出一样，让人一看就能明白。类似的标签还有programlisting, synopsis等。

3.2.5. callout

可以理解为一种标记注释，经常与Line-specific environments类的元素配合使用，例如我们有一段关于*rsyslog.conf*的配置文件，内容如下：

```
...
$WorkDirectory /var/spool/rsyslog

$IncludeConfig /etc/rsyslog.d/*.conf

auth,authpriv.*   /var/log/auth.log
...
```

为了能够清晰地向读者说明配置文件各行的作用，如果可以在需要解释的位置做一个标记，然后使用标记定位来解释相应的内容，会使读者更一目了然。就像如下的样子：

```
...
$WorkDirectory /var/spool/rsyslog ❶

$IncludeConfig /etc/rsyslog.d/*.conf ❷

auth,authpriv.*   /var/log/auth.log ❸
...
```

❶ 定义rsyslog的工作目录

❷ 指定rsyslog在运行时要包含哪些文件

❸ 认证类的日志存在auth.log中

要实现上面的样式，代码如下：

```
<programlisting linenumbering="numbered" language="common">...
$WorkDirectory /var/spool/rsyslog <co id="WorkDirectory-co" linkends="WorkDirectory"/>

$IncludeConfig /etc/rsyslog.d/*.conf <co id="IncludeConfig-co" linkends="IncludeConfig"/>

auth,authpriv.*  /var/log/auth.log <co id="auth-co" linkends="auth"/>
...</programlisting>

<calloutlist>
  <callout areafefs="WorkDirectory-co" id="WorkDirectory">
    <para>定义rsyslog的工作目录</para>
  </callout>
  <callout areafefs="IncludeConfig-co" id="IncludeConfig">
    <para>指定rsyslog在运行时要包含哪些文件</para>
  </callout>
  <callout areafefs="auth-co" id="auth">
    <para>认证类的日志存在auth.log中</para>
  </callout>
</calloutlist>
```

3.2.6. table

表 2.1. Sample Table

Horizontal Span		a3	a4	a5
b1	b2	b3	b4	Vertical Span
f1	f2	f3	f4	f5

Horizontal Span		a3	a4	a5
c1	Span Both		c4	
d1			d4	d5
f1	f2	f3	f4	f5

3.2.7. formalpara

This Paragraph Has a Title. This is a test. This is only a test. Had this been a real example, it would have made more sense. Or less.

3.2.8. Procedure

过程 2.1. An Example Procedure

1. A Step
2. Another Step
    a. Substeps can be nested indefinitely deep 1.
    b. Substeps can be nested indefinitely deep 2.
3. A Final Step

第 3 章 样式定义

最后生成的文件在版式和样式上想要达到自己想要的结果，就必须修改样式表。关于样式表的修改主要包含两个大的部分：一是修改现有的一些配置参数，如字体、字体大小等；二是直接修改某些板块的模板。修改参数相对简单，修改模板就需要做更多的工作。基本上所有的信息都在下面这两个站点里：

docbook xsl 参考手册 [http://www.sagehill.net/docbookxsl/index.html]

参数修改参考 [http://docbook.sourceforge.net/release/xsl/current/doc/index.html]

本章仅以fo（pdf）样式表的修改为例做简要说明。

1. 参数设置部分

fo样式表的所有参数的设定都在文件*docbook-xsl-1.79.1/fo/param.xsl*里面。大概有个差不多1000多行的代码。在前面已经对语言、字体的设置做了调整，在这里我们进一步说明一下，并再列举几个简单的例子，说明参数是如何调整的。

1. language

```
1 <xsl:param name="l10n.gentext.default.language">zh_cn</xsl:param>
2 <xsl:param name="l10n.gentext.language">zh_cn</xsl:param>
```

这是对使用语言的设置，在样式表目录里有一个*docbook-xsl-1.79.1/common/zh_cn.xml*语言包文件，这是中文的语言包。这个文件里的内容都是预先设置好的固定文本。设置什么样的语言，处理程序就会找哪个语言包。docbook-xsl对中文的支持非常友好。

2. font

在样式表中主要有三类字体需要设置，如下所示：

```
1 <xsl:param name="body.font.family">Source Han Serif CN,serif</xsl:param>
2 <xsl:param name="title.font.family">Source Han Sans CN,sans</xsl:param>
3 <xsl:param name="monospace.font.family">Source Han Mono SC,monospace</xsl:param>
```

这三种字体分别描述如下：

a. body.font.family：就是文档正文部分所使用的字体，通常情况下正文一般使用衬线（serif）字体，在汉字里一般是宋体。在这里设置了两个选项，除了我们在*fop.conf*里设置的“Source Han Serif CN”之外，还增加了一个serif的选项。它的作用是保证这份设置文件在其他没有SourceHan字体的电脑里也可以正常的工作，可以找到替代的字体。因为每台电脑一定有一款字体会被命名为serif。

b. title.font.family：就是各种标题部分所使用的字体，通常情况下标题一般使用非衬线（sans-serif）字体，在汉字里一般是黑体。同样，在“值”位置，有两个字体，除了“Source Han Sans CN”之外，还有sans，它的作用与body.font.family里的设置是一样的。

c. monospace.font.family：就是等宽字体，等宽字体一般用在程序代码、屏幕输出、inline等元素里。平时我们的终端命令行就是使用的等宽字体。同样，也是有两款字体。

3. line height

设置正文的行高，这个可以根据自己的喜好设定。 1.7的意思是字体大小的1.7倍。

```
1 <xsl:param name="line-height">1.7</xsl:param>
```

4. paper size

设定文档打印时版面的大小，有A0，A1，A3，A4... ....4O几种尺寸可供选择。A4大概是平时用得最多的。

```
1 <xsl:param name="paper.type">A4</xsl:param>
```

5. monospace.verbatim.properties

monospace.verbatim.properties是用于设置使用等宽字体文档元素的样式，例如前面提到的screen等。

```
 1 <xsl:attribute-set name="monospace.verbatim.properties"
 2    use-attribute-sets="verbatim.properties monospace.properties">
 3   <xsl:attribute name="text-align">start</xsl:attribute>
 4   <xsl:attribute name="font-size">6px</xsl:attribute>
 5   <xsl:attribute name="font-weight">normal</xsl:attribute>
 6   <xsl:attribute name="background-color">#f8f8f8</xsl:attribute>
 7   <xsl:attribute name="border-style">solid</xsl:attribute>
 8   <xsl:attribute name="border-width">0.1px</xsl:attribute>
 9   <xsl:attribute name="border-color">#777777</xsl:attribute>
10 </xsl:attribute-set>
```

在上面的设置中，我们定义了文本的对齐方式、字号的大小、子的粗细、背景颜色及边界线的样式。在样式表中，还有很多类似这样的设置，例如admonition.properties 、revhistory.table.properties等，都是依样画葫芦。

2. 样式模板

样式表的设定除了前面的参数修改以外，还有一些需要修改模板，比如“titlepage”这个部分，在标题页上显示哪些内容、用什么样的样式来显示，都需要重新定义或修改相应的模板，而非仅仅调整几个参数。在这个部分我们仅介绍titlepage的定义方法。

titlepage，就是标题页。比如book的titlepage，就像这本书，它有书的“右页（recto，可以向左翻）”，我只让它显示标题、副标题及作者；还有“左页（verso）”，这上面显示的信息要多一些，还包括版本信息、摘要、版权信息等。要定义这个模板，需要三步：

1. 修改 titlepage.templates.xml

*titlepage.templates.xml*就是titlepage的模板文件，这个文件在目录*docbook-xsl-1.79.1/fo*下面。首先我们需要对里面的“book”部分进行修改，修改成如下的样子：

```
<t:titlepage t:element="book" t:wrapper="fo:block">
    <t:titlepage-content t:side="recto">
      <mediaobject/>
      <title
             t:named-template="division.title"
             param:node="ancestor-or-self::book[1]"
             text-align="left"
             font-size="&hsize5;"
             space-before="&hsize5space;"
             font-weight="normal"
             font-family="{$title.fontset}"/>
      <subtitle
             text-align="left"
             font-size="&hsize2;"
             font-weight="normal"
             space-before="&hsize4space;"
             font-family="{$title.fontset}"/>
      <corpauthor font-size="&hsize0;"
             text-align="left"
             font-weight="normal"
             keep-with-next.within-column="always"
             space-before="2in"/>
      <authorgroup space-before="2in"/>
      <author font-size="&hsize0;"
             text-align="left"
             font-weight="normal"
             space-before="&hsize2space;"
             keep-with-next.within-column="always"/>
      <itermset/>
    </t:titlepage-content>

  <t:titlepage-content t:side="verso">
      <title
             t:named-template="book.verso.title"
             font-size="&hsize2;"
             font-weight="bold"
             font-family="{$title.fontset}"/>
      <corpauthor/>
      <authorgroup t:named-template="verso.authorgroup"/>
      <author/>

      <othercredit/>
      <copyright/>
      <pubdate/>
      <releaseinfo/>
      <revhistory/>
      <abstract/>
      <legalnotice font-size="7pt" background-color="#f5f5f5" space-before="1em"/>
  </t:titlepage-content>
  <t:titlepage-separator>
      <fo:block break-after="page"/>
  </t:titlepage-separator>
  <t:titlepage-before t:side="recto">
  </t:titlepage-before>
  <t:titlepage-before t:side="verso">
      <fo:block break-after="page"/>
  </t:titlepage-before>
</t:titlepage>
```

上面的代码虽然有点长，但很容易阅读，基本就说了两个事：在recto或verso页上显示哪些内容、用什么样的样式显示。

2. **生成模板文件**

   模板文件的生成需要刚才修改过的*titlepage.templates.xml*文件，及*docbook-xsl-1.79.1/template/titlepage.xsl*文件，还需要xsltproc这个命令。先将*titlpage.templates.xsl*做备份保存，然后执行如下的命令：

   ```
   xsltproc --output titlepage.templates.xsl template/titlepage.xsl titlepage.templates.xml
   ```

   这样，新的*titlepage.templates.xsl*文件就生成了。

   我总感觉这里有点奇怪，用xsl生成xsl？ 但docbook官方就是这么定义的。

   除了titlepage外，inline和pagesetup也需要定义模板，但不用修改xml文件了，可以参考*fo/inline.xsl*和*fo/pagesetup.xsl*文件和前面提到的相关站点。

第 4 章 字体

刚开始使用docbook时，我对字体一无所知。为了能让文档看上去更舒服、更能让人一目了然。就对字体做了一个简单的了解。把我学习到的如何在docbook中使用字体的经验在此做个分享。

1. 字体简介

1.1. 字体类型

较为常用的字体类型是Open Type Fonts(otf)和True Type Fonts(ttf)。在前面的章节中，我们下载的思源字体就是otf字体。还有一种ttc的字体文件，全称是TrueType Font Collection，是多个ttf或otf的合集。例如在windows系统使用比较多的simsun.ttc字体，里面就包含了SimSun（宋体）和NSimSun（新宋体）两个字体。

在参数设置的font部分，我们提到过字体的设置。有三种字体需要我们设置：serif，sans-serif和monospace。不仅仅是在docbook中，其实在每个操作系统里面也都有这三个设置。

1. serif

   衬线字体，一般是在笔画的开始或结束的地方有个装饰，在汉字里面，宋体是比较典型的serif，在英文里常用的是Times New Roman字体。serif的易读性较好。通常文章的正文使用serif字体，这可增加易读性。

2. sans-serif

   非衬线字体，有的地方也写做sans。这种字体的特点是衡平竖直的，笔画的粗细基本都一样。在汉字里，黑体、隶书等是比较典型的sans-serif，在英文里Arial、Helvetica属于sans。sans适合做文档的标题。

3. monospace

   等宽字体，从某种意义上说，monospace也算是sans的一种，除了笔画粗细都一样之外，每个字的字符宽度也都一样。等宽字体主要用在代码上，主要的作用就是容易排列整齐。

如果我们手头有杂志或报纸的话，他们的标题基本上都是非衬线字体（黑体居多），而正文基本都是衬线字体（宋体居多），我们的电脑里的终端程序，里面显示的就是等宽字体。对于一份技术性文档来说，有了这三种字体就基本够用了。

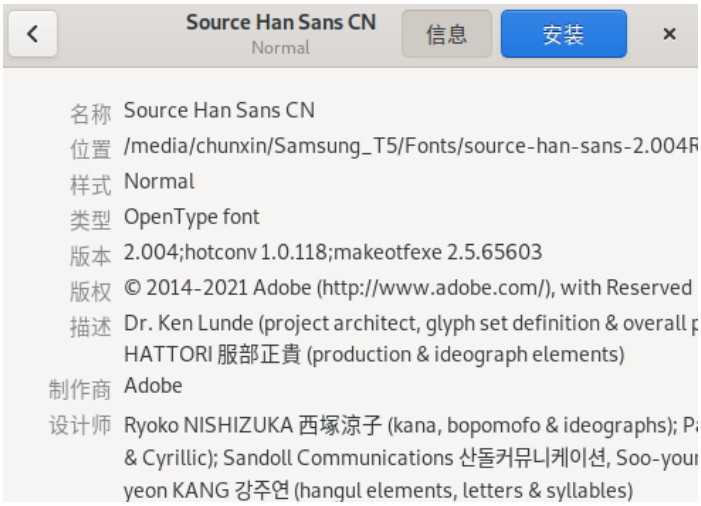
1.2. 字体样式

一款字体的选择，外观的样貌还是最重要的。另外，还有一点要考虑，这是我自己的感受。就是这款字体是否还具备粗体、斜体及粗斜体。在样式表或我们经常使用的网页的CSS中，一款字体通常有如下的三元组：

1. name

   对于ttf和otf类型的字体文件来说，一个字体文件里只含有一款字体，使用任何的字体查看软件[1]都可以知道它的正式的名称，如下图，使用gnome-font-viewer查看SourceHanSansCN-Normal.otf字体：

   图 4.1. 字体查看



   可以看出，字体的正式名称是“Source Han Sans CN”。

2. style

   通常，style是指的是正体（normal）还是斜体（italic）。

3. weight

   通常，weight是指得是正常大小（normal）还是粗体（bold）。这个在有些地方也用数字表示，400代表normal，700代表bold。Source Han字体有7个weight值。

在一个比较完美的情况下，一个字体在style方面要能提供normal和italic，在weight方面能提供normal和bold。一个字体应该有如下的可供选择的组合：

```
name = "font name"  style = "normal"  weight="normal"
name = "font name"  style = "normal"  weight="bold"
name = "font name"  style = "italic"  weight="normal"
name = "font name"  style = "italic"  weight="bold"
```

也就是平时经常说到的：normal, bold, italic, bold italic。但是大部分情况，很多字体好像只提供了style=normal，bold=normal

[1]在linux环境下，可以使用gnome-font-viewer或font-viewer来查看字体。

2. 字体选择

1. serif

   在serif字体的选择上，可能最普遍的是用simsun字体，在正文部分使用simsun字体是不错的选择。每台windows的fonts目录里都有这款字体。但它的粗体在fop将文档转换为pdf的时候似乎并不美观。simsun属于商业字体。

   另外一个选择就是前面我们下载的source-han-serif。

2. sans-serif

   非衬线字体，文泉驿正黑、SimHei及微软雅黑等都属于这一类。可以根据自己的喜好来选择。SimHei和微软雅黑与simsun一样，属于商业字体，在正式场合使用的时候要注意版权问题。

   另外一个选择就是前面我们下载的source-han-sans。

3. monospace

   等宽字体，文泉驿微米黑属于这一类。当然，也可以从自己的操作系统中寻找合适的等宽字体。对于等宽字体，我还是非常推荐source-han-mono的。

整体上来说，还是比较推荐Source Han字体，有以下几个原因：

1. **版权问题**

   Source Han字体是aodbe赞助，完全开源的字体。google的noto字体就是在这基础上做的。

2. **设计美观**

   Source Han的设计团队比较强大。字形设计还是非常不错的，特别是sans部分，和其他的黑体比较起来，我感觉非常好看，非常适合作为标题的字体。

3. **尺寸多**

   weight值一共有七个。

4. **字形也较全**

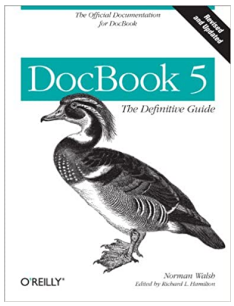
   mono字体的粗斜全都有了。

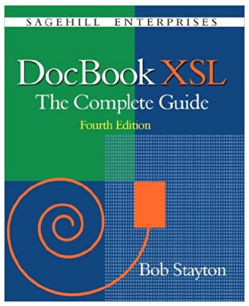
结束语

docbook标签丰富，样式也非常灵活。同时也带来了一定的学习成本，但一旦掌握、并定义了自己的样式表以后，在文档的书写、维护、团队协作方面会带来质的飞跃。

docbook的作用不仅仅用在文档写作上，国外很多公司还直接用它来做宣传册、技术白皮书等。

docbook经典的书籍有两本，《DocBook Xsl: The Complete Guide (4th Edition)》和《DocBook 5: The Definitive Guide》





可以在amazon.com上买到。另外，这两本书都有在线版本，前面已经提到过：

docbook 5：https://tdg.docbook.org/tdg/5.0/docbook.html

docbook xsl：http://www.sagehill.net/docbookxsl/index.html

在使用这份文档过程中如果您有建议可以通过<484299 @ qq .com>和我们联系。每一点意见都弥足珍贵。也期待一起分享更多的关于docbook方面的知识。

docbook是非常好的文档系统。期待更多的人使用，让文档成为一种艺术。