

ConT_EXt 学习笔记

Using MkIV

Li Yanrui (liyanrui.m2@gmail.com)

目 录

第 1 章 让 ConTeXt 工作起来

1.1 ConTeXt Minimals [1](#). 1.2 若需要 TeX 与 L^ATeX [2](#). 1.3 Hello World [3](#). 1.4 TeX 与 ConTeXt 的关系 [3](#). 1.5 让 MkIV 讲中文 [4](#). 1.6 开始 ConTeXt 的旅程 [5](#). 1.7 更多阅读…… [6](#).

第 2 章 在字体中挣扎

2.1 字体、别名与特征 [8](#). 2.2 简单的字体定义 [9](#). 2.3 Typescript 与 typeface [9](#). 2.4 混合字体 [12](#). 2.5 更多阅读…… [12](#).

第 3 章 版面设计

3.1 页面与纸张 [14](#). 3.2 版面布局 [14](#). 3.3 页眉、页脚 [17](#). 3.4 封面设计 [18](#). 3.5 标题样式 [19](#). 3.6 段落与行 [19](#). 3.7 更多阅读…… [20](#).

第 4 章 引用

4.1 目录 [21](#). 4.2 交叉引用 [22](#). 4.3 索引 [23](#). 4.4 参考文献 [24](#). 4.5 书签 [27](#). 4.6 更多阅读…… [29](#).

第 5 章 大文档项目

5.1 项目的部署 [30](#). 5.2 更多阅读…… [31](#).

第 6 章 我是这样学习 MetaFun 的

6.1 从画一个矩形开始 [32](#). 6.2 平凡的重复 [35](#). 6.3 Overlay 与背景 [36](#). 6.4 代码的分离 [38](#). 6.5 还有一些图形环境 [39](#). 6.6 后记 [40](#).

第 1 章 让 ConTeXt 工作起来

目前，只有 ConTeXt Minimals (ConTeXt 最小包) 提供了最新的 ConTeXt 版本并且可以使之与系统所安装的其它 TeX 发行版友好共存。本章介绍 ConTeXt Minimals 的安装、基本使用以及 MkIV 的中文支持等知识，目的在于让你多快好省地拥有一个 ConTeXt 文档编译环境。本章的最后介绍了一些初学者应当阅读的文档以及一些学习资源。

1.1 ConTeXt Minimals

首先，在本地机器上创建一个 ConTeXt 安装目录，该目录的位置与名字视个人嗜好而定，我喜欢用 `/opt/context`。为了不引起混乱，下文提及 ConTeXt 安装目录皆以如下设定的 Bash 变量 `$CTXDIR` 代替：

```
$ export CTXDIR=/opt/context
$ mkdir -p $CTXDIR
```

然后下载 ConTeXt 最小包安装脚本，将其存放于 `$CTXDIR` 目录并执行：

```
$ cd $CTXDIR
$ rsync -ptv \
rsync://contextgarden.net/minimals/setup/linux/first-setup.sh .
$ ./first-setup.sh
```

由于安装脚本需要联网下载 ConTeXt Minimals，需要等待一段时间才可以完成安装。这段时间，你可以做点自己喜欢做的事情。

ConTeXt Minimals 安装完毕后，就开始进行 ConTeXt 运行环境的配置，这需要设置许多的系统环境变量。不过莫要担心，ConTeXt Minimals 提供了一个配置脚本，使用 `source` (亦称点命令) 命令加载该文件即可完成所有 ConTeXt 所需环境变量的设定。

```
$ . $CTXDIR/tex/setuptex $CTXDIR/tex
```

上述命令行可将 `setuptex` 配置文件中的诸多环境变量的设定在当前终端环境中保持有效性。这意味着，每次使用 ConTeXt Minimals，可以开一个终端窗口，然后执行一次上述命令，之后就可以在该终端中使用 ConTeXt 环境。如果需要使用系统所安装的其它 TeX 发行套件，可另建一个未开启 ConTeXt 环境的终端即可。

虽然每次启用 ConTeXt Minimals 所输入的命令行并不复杂，但是尽力让命令行简约且易于输入一向是 Unix-like 系统的光荣传统，因此我在 `$HOME/myscript` 目录下建立了一个配置文件 `ctx`，该文件内容如下：

```
export TEXDIR=/opt/context/tex
source $TEXDIR/setuptex $TEXDIR
```

```
export OSFONTDIR="/usr/share/fonts/{adobe,winfonts}"
```

然后将 `$HOME/myscript` 目录添加到系统环境变量 `$PATH` 并使之生效:

```
$ echo "PATH=$PATH:$HOME/myscript" >> ~/.bashrc
$ . ~/.bashrc
```

这样,以后每次启用 ConT_EXt 环境时,只需执行以下命令:

```
$ . ctx
```

注意,在 `ctx` 文件中设置了 `$OSFONTDIR` 变量,该变量的意义在下一章讲述 MkIV 中文排版时会进行详细阐述,现在可不予理会。

使用 `ctxtools` 命令并配合 `-updatecontext` 参数可对所安装的 ConT_EXt Minimals 进行在线升级,不过前提是要建立 `$TEXMFLOCAL` 目录。因为 ConT_EXt Minimals 升级包是一个 zip 压缩包,升级时会将该压缩包下载到 `$TEXMFLOCAL` 目录并进行解压缩。

ConT_EXt Minimals 的升级很简单,那就是再重装一遍。

1.2 若需要 T_EX 与 L^AT_EX

ConT_EXt Minimals 可以与其它 T_EX 发行套件友好共存。最近,我打算使用 X_YT_EX + plain T_EX 格式写一份文档,但是 ConT_EXt Minimals 默认未提供 X_YT_EX 的 plain T_EX 格式并且缺乏一些 plain T_EX 必须的字体,因此基于 X_YT_EX 的 plain T_EX 及 L^AT_EX 环境均无法正常运行。对于这个问题,我能想到的比较轻省的解决方案是安装一个 T_EX Live 2008,使用它所包含的 plain T_EX 环境,另外再安装一个 L^AT_EX 环境以备不时之需。

首先从清华的 CTAN 镜像 FTP 服务器下载 T_EX Live 2008 的网络安装程序并解包:

```
$ wget ftp://ftp.tsinghua.edu.cn/mirror/CTAN/systems/texlive/\
tlnet/tldev/install-tl-unx.tar.gz
$ tar -zxvf install-tl-unx.tar.gz && cd install-tl-unx
```

T_EX Live 2008 的安装程序兼顾了不同用户的软件使用习惯,同时提供了文本界面与图形界面两种安装模式:

```
$ install-tl # 文本界面安装
$ install-tl --gui # 图形界面安装模式
```

`install-tl` 程序在运行时会自动访问预设定的 CTAN 镜像 FTP 服务器,如果连接失败,安装过程也就终止了。对于国内用户,可以通过 `--location` 参数将 CTAN 镜像服务器设为清华的 FTP:

```
$ install-tl --location=ftp://ftp.tsinghua.edu.cn/\
```

```
mirror/CTAN/systems/texlive/tlnet/tldev
```

如果网速足够，那么应当很快就可以在终端中显示文本安装界面了，具体的安装过程应当参考 [TeX Live 2008 中文指南¹](#)。TeX Live 2008 成功安装后，只要不在终端中开启 ConTeXt Minimals 的 ConTeXt 环境，就不会影响它的使用。

1.3 Hello World

将下面代码使用文本编辑器保存为 `hello-world.tex` 文件，这就是一个最简单的 ConTeXt 源文档。

```
% hello-world.tex
\starttext
Hello World.
\stoptext
```

我们可以在终端中使用 `context` 命令将这份源文档编译、输出为 PDF 文档：

```
$ . ctx
$ context hello-world
```

上述命令将在当前目录产生 `hello-world.pdf` 文档，使用 PDF 阅读器查看它，可以看到文档中除了“Hello World.” 字符串之外，就剩下一个置于顶端的页码了。

在编译 `hello-world.tex` 时，生成了许多中间文件，如果你不喜欢看到它们，可以使用以下命令将其清除：

```
$ context --purge
```

1.4 TeX 与 ConTeXt 的关系

在上一节 `hello-world.tex` 文档的编译过程中，我们可以获得的直观认识如下图所示：



图 1.1 ConTeXt 文档编译过程的直观认识

¹ <http://tug.org/texlive/doc/texlive-zh-cn/>

实际上是没有所谓的“ConTeXt 程序”的，真正实现 TeX 文档编译的是 TeX 引擎。在 ConTeXt MkIV 中，context 程序只是一个脚本文件，只有一行代码：

```
mtxrun --script context "$@"
```

mtxrun 是一个 Lua 脚本，它需要调用 LuaTeX 引擎并读入 cont-en.fmt 文件才可以对 ConTeXt 文档进行编译处理。cont-en.fmt 是 ConTeXt 的格式文件，可以将它理解为一本词典，当编译 ConTeXt 文档时，LuaTeX 引擎遇到文档中的排版命令时，就去 cont-en.fmt 文件中查找这些排版命令的定义并依命行事。可还记得 2.3 节中使用 context --make 命令么？这个命令就是用于产生 cont-en.fmt 之类的格式文件的。这就是为什么我们经常说 ConTeXt 或 L^ATeX 是 TeX 的一种格式的原因。

1.5 让 MkIV 讲中文

要让 ConTeXt MkIV 讲中文，首先要让它能够找到中文字体所在。还记得 1.1 节中我们写的那个 ConTeXt 环境配置文件 ctx 么？其中有一行代码如下：

```
export OSFONTDIR="/usr/share/fonts/{adobe,winfonts}"
```

\$OSFONTDIR 可以让 LuaTeX 知道系统所安装的字体所在。

我在 /usr/share/fonts/adobe 目录中存放了 4 款 Adobe OTF 中文字体，在 /usr/share/fonts/winfonts 目录中存放了一些从 Windows XP 中复制过来的一些中文字体。

有了中文字体，还需要一份字体配置文件。ConTeXt 提供了一套比较高级的字体机制，叫做 Typescript，它可以设置三种字型：衬线 (Serif)、非衬线 (Sans) 与等宽 (Mono) 字型。这三种字型均为英文字体机制中的术语，就像我们中文字体有楷体、宋体、黑体等类型一样。ConTeXt 的 Typescript 机制牵涉太多的字体知识，对它们的详细描述已超出我之所能，这里仅给出一份我照猫画虎搞出来的的一份字体配置文件——zhfonts.tex，它是随这份文档一起发布的，只需将它放到 TEX 目录树中，譬如：

```
$ mkdir -p $TEXMFLOCAL/tex/context/third
$ mv zhfonts.tex $TEXMFLOCAL/tex/context/third
```

然后执行：

```
$ context --generate
```

刷新一下文档数据库，就可以使用该字体配置文件了。

在 zhfonts.tex 中，我使用了 Adobe 宋、黑、楷体作为主要的中文字体。如果你没有这些字体，可以将其替换为其它 TTF 或 OpenType 中文字体，但是要保证 LuaTeX 可以找到它们。由于中

文字体所包含的英文字形通常比较丑陋，因此在 `zhfonts.tex` 文件中利用了 MkIV 的混合字体机制，将 `lmroman`、`lmsans`、`lmmono` 以及它们的粗体的英文字形区域分别注入到相应的中文字体中。

现在，将 1.3 节中的 `hello-world.tex` 文档修改为：

```
\usetypescriptfile[zhfonts]
\usetypescript[myfont]
\setupbodyfont[myfont,rm,11pt]
\starttext
世界，你好!
\stoptext
```

重新编译这份文档，即可得到中文版的 `hello-world.pdf` 文档。

`\usetypescriptfile` 命令用于加载 `typescript` 文件，要求 `typescript` 文件与 `helloworld.tex` 文件均在同一目录或者前者位于 ConTeXt Minimals 的某个可以检索到的目录中。

`\usetypescript` 命令表示使用 `typescript` 文件中定义的 `typeface`。所谓 `typeface`，就是一个字型族。在 `zhfonts.tex` 文件中所定义的 `typeface` 为 `myfont`，该字型族包含了 3 种字型：Serif、Sans 与 Mono。`\setupbodyfont` 命令是用于定义 `hello-world.tex` 文档的默认字体，也就是正文字体，这里是将 `myfont` 字型族中的 `Serif` 字型作为文档的默认字体，并且默认尺寸为 11pt。

现在，MkIV 基本上已经解决了中文排版问题，我们应当为此感谢 Hans、Taco 等开发者的辛勤劳动。另外，还要感谢 Wang Yue、SDE 诸位同学，他们提出了许多有关 ConTeXt 中文支持的要求并报告了许多 bug。特别是 Wang Yue 同学，自去年始，一直在关注 LuaTeX & ConTeXt MkIV 等项目的发展，并积极地与开发者们沟通，使得开发者认真考虑了有关中文处理的诸多需求。

1.6 开始 ConTeXt 的旅程

ConTeXt MkIV 中文支持的成功尝试，奠定了我开始学习 ConTeXt 的信心。ConTeXt 的文档非常丰富，不过大部分都是英文的，这多少让我有点沮丧。虽然读英文文档不是什么大障碍，但是总没有中文文档来得亲切。现状如此，只好忍了。

目前，ConTeXt 的大多数文档是针对 MkII 版本的，除了 MkIV 手册之外，似乎再也没有其它文档来讲述 MkIV 的应用。不过，由于 MkIV 改变的主要是 ConTeXt 底层，用户界面基本上没有改变，对于中文用户而言，基本上只需将 MkII 的中文字体支持机制改换为 MkIV 的即可，这样原来适用于 MkII 的文档基本上也适用于 MkIV。现在，ConTeXt 项目组现在正在组织进行新的 ConTeXt 文档撰写，这个工程非常浩大，其意在于将这么多年零散的文档汇总到一起，并追随 ConTeXt 的最新进展，估计要到明年方能竣工。远水解不了近渴，现在学习 ConTeXt，还是要从 MkII 版本的文档开始。

作为初学者, *ConTeXt, an excursion*[7]文档是官方制作的新手入门必读文档。ConTeXt 的官方文档通常是分为屏幕阅读版本与打印版本, 我读的 *ConTeXt, an excursion* 是屏幕阅读版本, 第一次打开这份文档, 就惊愕了一下, 没想到 PDF 文档居然可以做得如此精致。内容浅显易懂, 文档版面美观, 我觉得 ConTeXt 是非常礼遇初学者的。

当读过 *ConTeXt, an excursion* 时², 就可以使用 ConTeXt 排版手头上正在写作的一些文档, 其实这才是真正的开始学习 ConTeXt。对于其它任何一种 TeX 都是如此, 只有真正开始使用它, 才有可能掌握它。一开始, 文档的版面丑点没关系, 毕竟内容是最重要的。随着对 ConTeXt 的认识日益深刻, 你的文档便会愈发具备可观赏性。在实践的过程中, ConTeXt 用户手册[6]是主要的参考书。当你遇到不解的命令/参数, 或者想了解一些命令更多的细节, 都可以使用 PDF 阅读器打开 ConTeXt 用户手册, 利用阅读器提供的查询功能找到你所感兴趣的内容, 并掌握它们。

字体在排版中是决定版面美观的重要因素, 阅读 ConTeXt 新手册中已经基本完工的 *fonts*[2] 部分, 便可以理解 ConTeXt 的字体机制, 你将获得可以在 ConTeXt 自如使用你喜欢的字体的能力; 同时, 我制作的那份 MkIV 中文字体配置文件也不再会令你感到困惑。

若使用 ConTeXt 排版时存在矢量图绘制需求, 推荐使用 METAPOST。不过, 在 ConTeXt 中, 我们要面对的不是 METAPOST, 而是 MetaFun[8], 后者对前者进行了封装, 实现了 ConTeXt 与 METAPOST 的完美结合。ConTeXt, an excursion 文档版面之所以美观, 是因为在排版中大量使用 MetaFun来美化版面布局的结果。

若你对 ConTeXt 已经有所认识, 那么 ConTeXt Wiki³ 是非常好的网络学习资源。若有能力, 可以在该 Wiki 上与整个世界的 ConTeXt 用户分享你的学习经验; 若暂时没能力, 可以选择潜水。目前, 我是这个 Wiki 众多潜水员中的一名。

目前, ConTeXt MkIV 也许还存在许多的 bug, 当你发现它们, 应当提交 ConTeXt 邮件列表⁴中; 当你使用 ConTeXt 时有不解的问题, 也可以去邮件列表中寻找答案。说来惭愧, 我英文不好, 并且不熟悉邮件列表的交流规则, 所以一直都胆怯于邮件列表上的交流。

介绍了以上学习资源之后, 我开始如释重负。以后再在这份文档上折腾, 心里就不会总是想着自己在写一本 ConTeXt 教程, 那样子很累, 并且很容易丧失乐趣。这份笔记的真正的意图是自我备忘, 将它投放到网络上, 是希望我的经验能够对他人有所帮助并且能够一同学习、讨论 ConTeXt 排版知识。

1.7 更多阅读……

如果上述内容有误或者你认为讲述的不够细致, 那么请阅读以下文档:

² 实际上我是用到什么就去看什么, 现在大概看了有一半内容。

³ http://wiki.contextgarden.net/Main_Page

⁴ <http://archive.contextgarden.net/list/context.en.html>

- “序幕有些长”，介绍了 TeX 与 ConTeXt MkIV 的背景知识：
<http://garfileo.is-programmer.com/2011/1/10/zhfonts-usage.23740.html>
- “这就是 ConTeXt Minimals”，讲述了 ConTeXt Minimals 的安装方法：
<http://liyanrui.is-programmer.com/2009/10/7/this-is-context-minimals.11971.html>

我计划今年年底将这些外部文档归整到这份文档中。

第 2 章 在字体中挣扎

当我刚开始学用 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 时，就患上了字体恐惧症。如果只是写英文文档，那么对于初学者而言实在是太简单了，基本上不需要怎么关心字体的问题；但是对于中文或者其它非西语文字而言， $\text{T}_{\text{E}}\text{X}$ 的易用性依然很弱，以至于用户不得不去弄清楚诸多有关字体方面的知识及其配置方法。我很憎恶这些东西，但是也没有办法，排版本来就不是一件易事，只好秉承“为之，则难者易易也”的精神去摸索一下。我开始阅读 $\text{ConT}_{\text{E}}\text{Xt}$ Fonts 文档[2]，这一章内容就是我的阅读笔记。

2.1 字体、别名与特征

为字体文件取个别名，这是 $\text{ConT}_{\text{E}}\text{Xt}$ 字体机制推荐的做法。比如，我们可以将 `AdobeSongStd-Light.otf` 字体取名为 `Song`：

```
\definefontsynonym  
  [Song] [AdobeSongStd-Light] [features=zh]
```

`features=zh` 参数是用来告诉 $\text{ConT}_{\text{E}}\text{Xt}$ MkIV 应当采用中文 (zh) 方式来处理这款字体。`features` 的值并非 $\text{ConT}_{\text{E}}\text{Xt}$ 提供的标准值，而是要用户自己定义：

```
\definefontfeature  
  [zh] [mode=node,script=hang,lang=zhs]
```

`zh` 这个 font feature 名字可根据自己的嗜好来定；至于第二组参数，对于简体中文用户，通常这样设置就可以。

`\definefontfeature` 与 `\definefontsynonym` 命令的用法在与本文档一同发布的 `zhfonts.tex` 文件中均有体现，可参照理解。

这里需要注意一下字体别名定义中的字体名称。上面的示例中，我直接使用 `AdobeSongStd-Light` 告诉 MkIV：我使用的是 `AdobeSongStd-Light.otf` 字体。实际上，这样的效率会有点低。因为 MkIV 有两种搜索字体的方式，第一种是根据字体开发商给出的字体名 (name)，第二种是字体的文件名 (file)。字体名可以使用 `fc-list` 命令来查，譬如查询搜索 `AdobeSongStd-Light.otf` 的字体名：

```
> fc-list | grep Song  
Adobe 宋体 Std L,Adobe Song Std,Adobe 宋体 Std,  
Adobe Song Std L:style=L,Regular
```

我们可以得到许多字体名，推荐从中选择一个纯英文的供给 MkIV，譬如：

```
\definefontsynonym  
  [Song] [name:Adobe Song Std] [features=zh]
```

如果想使用字体文件名，可以为其冠以 `file` 前缀：

```
\definefontsynonym
  [Song] [file:AdobeSongStd-Light] [features=zh]
```

在本节最初的那个例子中，既没有 `name` 前缀，也没有 `file` 前缀，那么 `MkIV` 就只好先尝试字体名；如果找不到字体，再去尝试字体文件名。

我觉得使用字体名比较正规一些，因为字体的文件名很容易被改掉；字体名是由开发商设定的，一般用户不会想到去修改它，因此使用它可以提高文档的可移植性。

`ConTeXt` 预定义了一些标准的字体别名，我们常用的是 `[Serif|Sans|Mono]` `[Bold|Italic|BoldItalic]`，它们在 `zhfonts.tex` 中均出现了。

还值得注意的一个问题就是：我们可以定义字体别名的别名。

2.2 简单的字体定义

使用 `\definefont` 命令可以实现最快捷、最容易理解的字体定义与使用。比如我想定义 Adobe 宋体并使用它：

```
\definefont
  [Song]
  [{name:Adobe Song Std*zh} at 14pt]

\starttext
\Song 我使用的是 Adobe 宋体
\stoptext
```

或许上面这个例子看起来有点古怪，我将字体定义的部分改写一下，或许就会清楚一些：

```
\definefontsynonym
  [SongTi] [name:Adobe Song Std] [features=zh]

\definefont
  [Song] [SongTi at 14pt]
```

如果在定义字体时，不去硬性地设定字体尺寸，那么 `ConTeXt` 可以根据具体的排版环境自动为字体设定合适的尺寸。

2.3 Typescript 与 typeface

说实在的，`typescript` 的中文意思我真的说不出来，词典上可以查到的解释是“用打字机打的文件”。我在中文里也实在找不出简洁、准确的翻译，所以还是直称 `typescript` 为好。

简单来说, `typescript` 是 `ConTeXt` 的最上层的字体配置文件。目前, `MkIV` 没有提供面向中文用户的 `typescript` 文件, 这就需要我们自行去实现, 因此掌握 `typescript` 文件的写法非常重要。

在只针对 `MkIV` 的 `typescript` 配置与使用中, 常用命令如下:

- `typescript` 环境
- `\definefontsynonym`
- `\definetypeface`
- `\usetypescriptfile`
- `\usetypescript`

下面我将结合 `zhfonts.tex` 中的部分内容来理解这些命令的用法。

在 `zhfonts.tex` 文件中可以看到, `typescript` 文件内容是被 `typescript` 环境分为多个小节的, 每个小节实现了一组字体的配置:

```
\definefontfeature
  [zh] [mode=node,script=hang,lang=zhs]

\starttypescript[serif] [zhfont]
... ..
\stoptypescript

\starttypescript[sans] [zhfont]
... ..
\stoptypescript

\starttypescript[mono] [zhfont]
... ..
\stoptypescript

\starttypescript[myfont]
... ..
\stoptypescript
```

上述代码, 我设置了三种字样 (`typeface`): 衬线 (`serif`)、非衬线 (`sans`) 与等宽 (`mono`)。最后的那个 `typescript` 环境是打算用来定义字样集合的。有关这里的字样的概念, 可参考黄老师的《`LATEX` 学习笔记》[9]的第 8 章来理解, 同时你也可以从中额外获得一些 `TEX` 字体的常识。

我们是在 `typescript` 环境中定义具体的字体, 比如下面定义衬线字样的一组字体:

```
\starttypescript[serif] [zhfont]
  \definefontsynonym
    [Serif] [name:adobesongstd]
  \definefontsynonym
    [SerifBold] [name:adobeheitistd]
```

```

\definefontsynonym
  [SerifItalic] [name:adobekaitistd]
\definefontsynonym
  [SerifBoldItalic] [name:adobeheitistd]
\stoptypescript

```

因为已经知道了 `\definefontsynonym` 命令的作用，因此上述字体的定义看起来就非常简单了，就是用我们所指定的一组字体去替换 ConTeXt 标准字体别名默认所使用的字体而已。我们可以按照这样的方式完成非衬线与等宽字样的字体定义，不再赘述。

下面来看一下字样的定义：

```

\starttypescript [myfont]
  \definetypface [myfont] [rm] [serif] [zhfont] [features=zh]
  \definetypface [myfont] [ss] [sans] [zhfont] [features=zh]
  \definetypface [myfont] [tt] [mono] [zhfont]
\stoptypescript

```

`\definetypface` 命令的第三个参数与第四个参数是与前面的 `typescript` 环境参数对应一致的，比如：

```

\definetypface
  [adobe] [rm] [serif] [zhfont]

```

对应：

```

\starttypescript [serif] [zhfont]
  ... ..
\stoptypescript

```

其中含义就是将 `adobe` 字样集合的衬线(`rm`) 字样设为该 `typescript` 文件中具有 `[serif][zhfont]` 参数的 `typescript` 环境所定义的字体集；也就是说，ConTeXt 可以根据 `\definetypface` 的第三、四个参数来匹配对应的 `typescript` 环境。不知我有没有说明白。总之，在此要认真体会一下 `typescript` 环境参数的作用，姑且可以将其当作是对应 `typescript` 环境唯一的标识符号。至于 `\definetypface` 前两个参数，其实也是作为一个标识符来用的，这在使用 `typescript` 文件时就可以看到。

将上述的知识汇在一起，就可以写出类似 `zhfonts.tex` 这样的 `typescript` 文件，若将其投入到 TeX 目录树中，就可以在排版时使用它了，譬如：

```

\usetypescriptfile [zhfonts]
\usetypescript [myfont]
\setupbodyfont [adobe,rm,11pt]

```

首先 `\usetypescriptfile` 命令驱使 ConTeXt 在 TeX 目录树中找到 `zhfonts.tex` 这个 `typescript` 文件；然后 `\usetypescript` 命令可以在 `zhfonts.tex` 文件中匹配含有一个参数为 `myfont`

的 `typescript` 环境，并读取该环境所定义的字样信息；最后使用 `\setupbodyfont` 命令将 `typescript` 文件中所定义的某种字样其定位文档的默认字体集，`\definetypeface` 命令的前两个参数的用途再次便得以体现出来。

2.4 混合字体

从概念上讲，混合字体就是将两种或更多的字体的不同编码部分合成到一起，形成一种新的字体。比如，中文字体往往也会包含了英文部分，偏偏这部分字体设计的又太差。所以，目前对于中文用户而言，混合字体机制实在是重之又重，因为我们可以利用这套机制来实现将一部分好看的英文字体注入到中文字体中，以此来实现美观的中英文混合文本排版。

下面展示一个小例子，用以说明如何使用 MkIV 的混合字体机制：

```
\definefontsynonym
  [zhserif] [name:adobesongstd]
\definefontfallback
  [serifwhatever] [lmroman10-regular] [0x0000-0x0400] [force=yes]
\definefontsynonym
  [ZhSerif] [zhserif] [fallbacks=serifwhatever]
\definefont [FbSong] [ZhSerif at 12pt]
\definefont [Song] [name:adobesongstd at 12pt]

\starttext
\FbSong
中英文混合: To do or not to do, this is the question!

\Song
中英文混合: To do or not to do, this is the question!
\stoptext
```

字体切换命令 `\ZhSerif` 使用的是混合字体，是 Adobe 宋体的 `0x0000-0x0400` 区域中掺入了 `lmroman10` 字体的对应区域；而 `\Song` 命令使用的则是纯的 Adobe 宋体。上面的例子也许看起来混合字体的优势所在，这主要是因为 Adobe 宋体中的英文部分设计的还算不错。如果将 Adobe 宋体换成 `SimSun`，对比效果会更明显一些。

我们可以将这种混合字体机制与 MkIV 的 `typescript` 机制相结合，在不同风格的中文字体中掺入类似风格的英文字符，这样中英文混合排版的问题就能够得到较好的解决。

2.5 更多阅读……

如果上述内容有误或者你认为讲述的不够细致，那么请阅读以下文档：

- “不仅仅是为了中文”，详细介绍了 ConTeXt 的 Typescript 机制：
<http://liyanrui.is-programmer.com/2009/10/21/not-just-for-chinese.12264.html>

- “zhfonts 模块的用法”，zhfonts 模块可支持中文字体加载、标点间距调整等功能：
<http://garfileo.is-programmer.com/2011/1/10/zhfonts-usage.23740.html>

我计划今年年底将这些外部文档归整到这份文档中。

第 3 章 版面设计

其实我不懂排版，我只是希望这份文档的版面看上去不那么丑陋，所以我经常留意有关排版的知识，并乐于使用 ConT_EXt 来实践。我将这个实践的过程整理出来，就是本章所记述的内容，并且每当我修改本文档的版面风格时，这一章的内容也会适应性的变动。可以将本章看作是这份笔记版面设计的说明书。

3.1 页面与纸张

页面尺寸就是文档的版面尺寸，而纸张尺寸是指文档印刷用纸的规格，它们的取值一般是相同的；只有在要求每张印刷用纸上实现多页排版时，二者的取值才不一样。这份笔记的屏幕阅读文档页面与纸张尺寸设定如下：

```
\definepapersize[SCREEN] [width=24cm,height=18cm]
\setuppapersize[SCREEN] [SCREEN]
```

其意是指页面与纸张的尺寸的宽高分别为 24cm 与 18cm，这是考虑到显示屏幕尺寸的宽高比一般为 4:3，这样可以充分利用屏幕区域，并且翻页方便。

页面与纸张的规格可以采用 ConT_EXt 已经预定义好的国际标准规格，如 A0~A10、B0~B5 等；也可以采用自定义尺寸：

```
\definepapersize[I18] [width=18.5cm,height=23cm]
\setuppapersize[I18] [I18]
```

这里定义的 I18 规格的页面与纸张尺寸，是近年来计算机技术书籍排版常用的纸张规格。

3.2 版面布局

ConT_EXt 将版面划分为 25 个区域，具体的区域划分在 *ConT_EXt, an excursion*[7] 文档的第 32 章 *Page Layout* 中可以看到，也可以在 ConT_EXt Wiki 的 *Layout* 页面看到¹。

我为这份文档所规划的版面布局如下：

```
\setuplayout
  [width=fit,
   height=middle,
   leftmargin=3cm,
   rightmargin=3cm,
   backspace=4cm,
   topspace=.5cm,
```

¹ <http://wiki.contextgarden.net/Layout>


```
headerdistance=.4cm,  
footerdistance=.4cm,  
header=1cm,  
footer=1cm]
```

下一页的插图显示了这些参数的布局效果，这里交代一下，那幅插图是利用了 Patrick 写的一个模块 (Module)²绘制的。

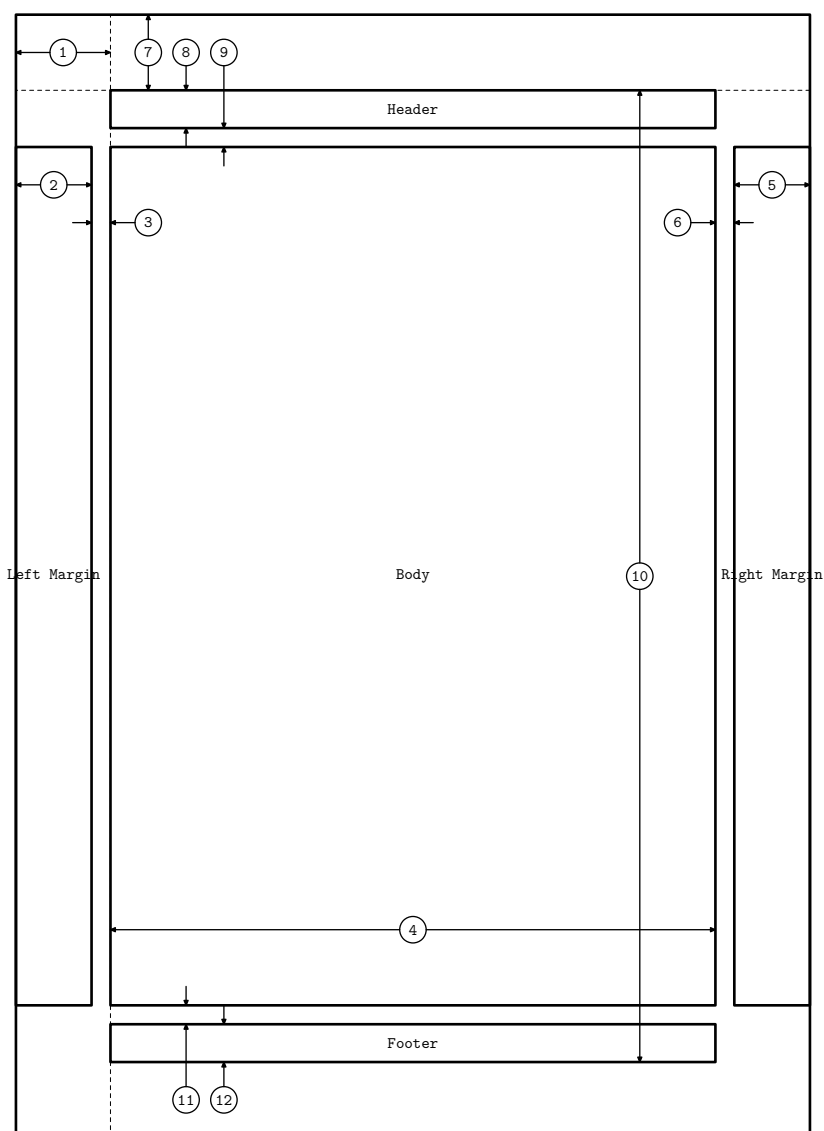
`\setuplayout` 命令所具有的参数，在数量上堪称是 `ConTeXt` 命令之最，如果我没有数错，应该是 44 个，常用的参数也就是下一页插图中所示的那些了。通过这些参数，`ConTeXt` 可实现各种复杂的版面布局。

说点闲话。如果我没记错的话，MS Word 用来设置版面布局的参数不超过 10 个。在 `LATEX` 中，需要借助一些宏包才可以比较方便地实现版面布局控制。复杂与统一，也许这是 `ConTeXt` 的一个很重要的特征，复杂是追求功能强大的必然结果，而统一可以让用户只需一份官方提供的技术手册就可以毫无争议地使用这些功能。

注意，将 `\setuplayout` 的 `width` 与 `height` 参数设为 `fit` 或 `middle` 都可以让 `ConTeXt` 自动为我们确定它们的值。`fit` 参数的确定是根据周边邻域参数计算出来的，而 `middle` 不是很关心周边参数，它只需要将 `width` 或 `height` 所确定的尺度相对于同一维向的页面尺度是居中的。

`location` 这个参数让我费解了许久，历经多次尝试，总算知道一点所以然。当我们将版面尺寸与纸张尺寸设为相同时，`location` 这个参数无论如何设定都是看不到效果的。当纸张尺寸大于版面尺寸时，`location` 的值可以表示版面在纸张的位置，譬如 `left`、`right`、`middle`、`singlesided` 以及 `doublesided`。

² <http://modules.contextgarden.net/t-layout>



- | | |
|-------------------------------|---------------------------|
| 1 backspace 71.1 pt | 8 header 28.5 pt |
| 2 leftmargin 56.9 pt | 9 headerdistance 14.2 pt |
| 3 leftmargindistance 14.2 pt | 10 height 731.2 pt |
| 4 width 455.2 pt | 11 footerdistance 14.2 pt |
| 5 rightmargin 56.9 pt | 12 footer 28.5 pt |
| 6 rightmargindistance 14.2 pt | paperwidth 597.5 pt |
| 7 topspace 56.9 pt | paperheight 845.0 pt |

3.3 页眉、页脚

对于这份文档的页眉，我将奇数页的页眉设为当前章标题（左）+ 页码（右）；偶数页的页眉设为当前节标题（左）+ 页码（右）：

```
\def\CurrentChapter{%
    第 \headnumber[chapter]\ 章%
    \hbox to wem{}%
    \getmarking[chapter]%
}

\def\CurrentSection{%
    \headnumber[section]%
    \hbox to 2em{}%
    \getmarking[section]%
}

\setupheadertexts
    [\CurrentChapter] [pagenumber]
    [pagenumber] [\CurrentSection]
```

`\setupheadertexts` 命令的参数设置的有些古怪，按手册上讲的，前两个参数分别用于设置奇数页的页眉左侧与右侧内容，后两个参数则用于设置偶数页的页眉左侧与右侧内容，但是在上述的页眉设置语句中，我只有将偶数页的页眉内容的左右位置掉换过来才可以得到这份文档的页眉结构。我并不知道为什么要这样设置，只是多次尝试才知道只有如此才能实现我的要求。

由于设置在页眉中显示页码，所以应当引去 `ConTeXt` 自动为页面添加的页码，顺便把页码的字号也设置一下：

```
\setuppagenumbering
    [style=\tfx,location=]
```

我感觉 `ConTeXt` 是有点另类，居然让 `\setuppagenumbering` 来设置双面打印文档模式。如果，我想将本文档的页眉结构变成：奇数页的页眉设为当前章标题（左）+ 页码（右）；偶数页的页眉设为当前节标题（右）+ 页码（左），只需在 `\setuppagenumbering` 命令中设定 `alternative` 参数即可：

```
\setuppagenumbering
    [alternative=doublesided,style=\tfx,location=]
```

感觉有点莫名其妙，也许因为我还未发现正确的页眉设置方法的缘故。

页脚的设定方法与页眉类似，我在页脚的左侧设置了显示这份文档的名称，在其右侧显示可以跳往目录页的链接：

```
\startmode[screen]
```

```

\setupfootertexts
  [{\ConTeXt\ MkIV 学习笔记}][{\goto{回目录}[content]}]
  [{\goto{回目录}[content]}][{\ConTeXt\ MkIV 学习笔记}]
\stopmode

```

用到了 ConTeXt 的 mode 环境，这就像 C 语言中的编译宏。只有在 ConTeXt 编译命令中设定对应的 mode 值，对应 mode 环境中的语句才是有效的。要让 screen mode 环境生效，可：

```
$ context --mode=screen ctxnotes
```

ConTeXt 的 mode 环境非常有用，许多 ConTeXt 官方制作的文档都是利用 mode 环境同时产生屏幕阅读版本与打印版本。将来，我打算将这一环境用于制作演示文档与讲稿。

在页脚右侧显示的“回目录”的链接需要设置目录代码的支持，因为这需要利用 ConTeXt 的引用机制：

```

% 我的目录页
\title[content]{目录}
\placecontent

```

有关文档目录设置的细节，准备单独开一个章节来记述。

3.4 封面设计

ConTeXt 没有像 L^AT_EX 那样提供了专用于封面制作的命令，只是提供了一个 standardmakeup 环境，该环境的作用就是建立一个不增加页码计数的页面，使得用户可以在其中排版封面、扉页之类的页面。这份文档的封面的具体设计如下：

```

\startstandardmakeup
  \startcolor[mydarkred]
  \switchtobodyfont[14pt]
  \hfil\bfd\ConTeXt 学习笔记\hfil
  \blank[1.5cm]
  \hfil\bfc Using MkIV\hfil
  \blank[12cm]

  \startalignment[flushright]
  \ssa
  \bTABLE
    \setupTABLE[r][each][frame=off]
    \bTR \bTD Athor: Li Yanrui\eTD \eTR
    \bTR \bTD Email: liyanry@gmail.com\eTD \eTR
    \bTR \bTD \date \eTD \eTR
  \eTABLE
\stopalignment
\stopcolor

```

`\stopstandardmakeup`

请宽恕我在上述语句中很可耻地使用了 `1.5cm`、`12cm` 这样确定的长度。因为我还不怎么会驾驭 `ConTeXt`，还无法灵活地实现可根据版面的变化自适应调整的封面设计。

要设计一个美观的封面真得很难，而且这也不是我所擅长的。现在，拥有这样一个简单的封面，我已经心满意足。也许等我以后渐渐熟悉 `MetaFun` 之时，会让它再美观一些。

3.5 标题样式

惯见的文档章、节标题一般都是采用黑体，字号通常也比正文字号大一些，另外标题前后也留出了一些垂直间距。这些设置无非是为了告诉读者，它们是标题，而不是正文。这份文档的标题样式设置如下：

```
\setupheads[indentnext=yes]
\setuphead
  [chapter]
  [style=\bfc,header=empty,footer=empty]
\setuphead
  [section]
  [style=\bfa]
\setuphead
  [title]
  [style=\bfb,header=empty,foote=empty]
\setuphead
  [subsubject]
  [style=\bf]
```

`\setupheads` 命令可以设置所有类型的章节标题默认状态。在设置章标题时，由于其所在页面通常不需要页眉与页脚，所以 `header` 与 `footer` 都应设为 `empty`。

现在使用 `MkIV` 排版中文，要想实现章节的编号格式为“第 x 章”、“第 x 节”，或者实现图、表标题格式为“图 x”、“表 x”，可能需要自己来做一些有些 `dirty` 工作。`ConTeXt` 提供了 `\setuplabeltext` 命令，可用于设置表格、图、章节、附录的编号格式。譬如，要设置中文的章节编号：

```
\setuplabeltext [en] [chapter={第\;,\;章}]
\setuplabeltext [en] [section={第\;,\;节}]
```

遗憾的是，我不知道怎样将编号中的数字也使用中文来表示，这在 `MkII` 中是可以的。

3.6 段落与行

中文排版，段落间距通常等于行间距，因为中文段落是通过首行缩进来标示的，下面代码可将段落文本首行缩进 2 个汉字距离：

```
\setupindenting[always,2em,first]
```

ConTeXt 默认是将段间距设置为 0，这碰巧符合中文的排版规范。如果有设定段间距的需求，就查一下 `\setupwhitespace` 命令的用法。

对文档正文而言，行间距的设置对于排版的美观性是至关重要的：行间距过小则阅读困难；行间距过大又显得版面稀疏，浪费纸张。ConTeXt 默认的文本行间距的值对于中文排版而言过小，可通过 `\setupinterlinespace` 命令结合自己的审美嗜好对行间距进行调整。我习惯将行高设为 1.5em：

```
\setupinterlinespace[line=1.5em]
```

3.7 更多阅读……

如果上述内容有误或者你认为讲述的不够细致，那么请阅读以下文档：

- “页面布局控制”：
<http://garfileo.is-programmer.com/2011/1/19/layout-control.23895.html>
- “ConTeXt 文稿的逻辑结构”：
<http://garfileo.is-programmer.com/2011/1/16/context-manuscript-logic-structure.23826.html>

我计划今年年底将这些外部文档归整到这份文档中。

第 4 章 引用

当我用我这个代号来进行对话的同时，你的代号也是我，这意味着什么呢？这是否意味着你就是我，而我也就是你？

4.1 目录

如果不那么在乎目录的样式，可以在期望插入目录之处添加：

```
\completecontent  
% 或者  
\placecontent
```

`\completecontent` 与 `\placecontent` 的区别就是前者是生成的目录列表页面是带有标题的；而后者只生成目录列表，如果需要页面带有标题，可以使用 `\title` 之类的无编号的标题命令实现：

```
\title{目录}  
\placecontent  
  [alternative=a,  
   style=normal,  
   numberstyle=bold]
```

在章标题之后使用 `\placecontent`：

```
\chapter{引用}  
\placecontent  
  [alternative=a,  
   style=normal,  
   pagenumber=no,  
   margin=2em]
```

可以实现在当前章标题下面显示出这一章所有小节的列表。

由于我比较喜欢 ConTeXt 手册的那种目录样式，便从手册的源码中找到了该样式的具体设置，然后照虎画猫一番，并添加了一些小创意：

```
\def\ChapterNumber#1{\doiftext{#1}{第\;#1\;章\quad}}  
\setuplist  
  [chapter]  
  [alternative=a,  
   before={\page[preference]\blank},  
   after=\blank,  
   style=bold,  
   width=fit,
```

```

pagestyle=boldslanted,
pagenumber=no,
numbercommand=\ChapterNumber]

\def\PageNumber#1{\color{darkgray}{#1}.}
\setuplist
[section]
[alternative=d,
style=small,
pagecommand=\PageNumber,
pagestyle=\itx]

```

结果便是这份文档目录页中所显示的那个样子。我所说的得小创意，实际上就是让章节编号变成半中文化（因为还夹着阿拉伯语呢）；另外就是让目录列表中各小节的页码显示为灰色，不那么抢眼。

4.2 交叉引用

交叉引用实在是太有用了，即便也不怎么经常用到它，也还可以借此嘲笑一下 Word 交叉引用功能的孱弱，赢得一次口水战的胜利。

这份文档的页脚的右侧的“回目录”链接就是通过交叉引用来实现的，在 3.3 节（瞧，这就是一个交叉引用）中的页脚设置：

```

\setupfootertexts
[{\ConTeXt\ MkIV 学习笔记}][{\goto{回目录}[content]}]
[{\goto{回目录}[content]}][{\ConTeXt\ MkIV 学习笔记}]

```

其中，`\goto` 这个命令可以跳到任何设置了引用的文档位置。这里，它要跳到 `content` 引用所在的位置，由于这个引用我是在设置目录页面时设定的：

```

\startfrontmatter
\setuppagenumbering
[conversion=romannumerals]
\title[Content]{目录} % 在此设定了引用
\placecontent
\stopfrontmatter

```

因此 `\goto` 会带你跳到目录页。目录页面的引用是通过 `content` 这个名称标识的，`\goto` 命令可以根据这个标识确定所要跳往的页面位置。

`ConTeXt` 许多排版命令都支持引用设置，譬如章节标题、插图、表格等命令。一定要记住，文档中只要出现诸如“第 xxx 章”、“xxx 节”、“图 xxx”、“表 xxx”之类的内容时，这就意味着您应当使用交叉引用机制了。比如，我要引用当前这一节，首先要为节标题命令设定引用名称：

```

\section[cross references]{交叉引用}

```


然后在需要引用本节之处，通过 `\in` 命令即可获得节号；如果是屏幕阅读文档，还可以让节号作为一个链接，指向本节所在页面位置。同类的命令还有 `\at` 与 `\about`，前者可以获取引用所在的页面，后者可以得到引用的内容。下面仿照 `ConTeXt` 手册中的例子，演示这三个命令的用法：

我引用了第 `\at[cross references]` 页的 `\in[cross references]` 节 `\about[corss references]` 的一些内容。

效果：我引用了第 22 页的 4.2 节“交叉引用”的一些内容。

现在，`MkIV` 还未能实现引用在页面的准确定位，所以在屏幕阅读文档中的一些链接只是将你带往这些引用所在的页面而不是将准确的位置展示在你眼前。

4.3 索引

在阅读 `ConTeXt` 手册或者 `ConTeXt, an excursion` 之类的文档时，在附录中可以看到许多命令的索引，这些索引会告诉你那些命令在哪些页面出现了；如果是屏幕阅读文档，还可以将页码变成指向对应页面的链接。利用这些索引功能，我可以快速找到这些命令的细节知识。这在写技术文档时非常重要。在这份文档中，我没有使用索引的功能，因为很少有人会为了一份学习笔记如此大动干戈。将它记述于此，仅仅是为了将来有可能使用到它。

要定义索引很简单，使用 `\index` 这个命令就可以：

我在这里演示 `index\index[index]{\tex{index}}` 与 `placeindex\index[placeindex]{\tex{placeindex}}` 命令的用法。

`\placeindex`

然后使用 `\placeindex` 或 `\completeindex` 命令在需要放置索引的页面显示索引。

索引列表默认是以字母进行分组与排序的，这对于西文文档排版很适用，但是没法处理中文。可以采用汉语拼音来解决这个问题。譬如：

……可以采用汉语拼音 `\index[HYPY]{汉语拼音}` 来解决这个问题……

生成的索引列表如下：

c	i
<code>\completeindex</code> 23	<code>\index</code> 23
h	p
汉语拼音 23	<code>\placeindex</code> 23

索引与目录的用法很相似。实际上这一章中所有的内容都是基于 `ConTeXt` 的引用机制实现的。索引与目录存在着更为高级的用法，但是我没有那么多兴趣和精力去折腾它们，以后若有这方面的需要再下手也不迟。

4.4 参考文献

写科技文档必然离不开参考文献 (Bibliography) 的支撑，所以要评价一个文档排版软件的优劣，其参考文献管理功能是一项非常重要的指标。与 `LATEX` 类似，`ConTeXt` 主要是基于 `BibTeX` 维护参考文献，这是借助 Taco Hoekwater 写的 `t-bib.tex` 模块来实现的。

BibTeX

`BibTeX` 的主要功能是管理参考文献信息并提供排版格式。在使用 `BibTeX` 之前，用户需要建立参考文献数据库，不要害怕，这里的数据库实际上就是一个扩展名为 `.bib` 的文本文件，我们在其中记录文献信息。下面是一份 `.bib` 文件示例：

```
@book{诸葛专著2008,
  author    = "诸葛亮",
  year      = "2008",
  title     = "木牛流马制造工艺 [M]",
  publisher = "蜀国机械工业出版社"
};

@article{诸葛论文2008,
  author    = "诸葛亮",
  title     = "论伐魏的必要性与可行性 [J]",
  journal   = "蜀国军事",
  volume    = "13",
  number    = "110",
  year      = "2007"
}
```

在文献数据库中，可以存储许多种文献类别，常用条目类型有 `article`、`book`、`conference`、`manual`、`misc`、`techreport` 等。每种参考文献类别由多个域组成，有些是必须写得，没写会给出警告，而有些是可选。譬如 `book` 类别中，不可省略的域有 `author`、`title`、`journal`、`year`，可省略的域有 `volume`、`number`、`pages`、`month`、`note`。推荐做法是在参考文献数据库中尽可能地提供文献的详细信息。关于 `.bib` 文件的格式说明，请参考 `BibTeX` 文档[1]。

`BibTeX` 为便于用户管理参考文献列表的排版风格，提供了 `.bst` 文件。`.bib` 文件与 `.bst` 文件之间的关系宛若 `HTML` 与 `CSS` 之间的关系，它们都遵守内在数据与外在表现分离的游戏规则。用户在 `.bst` 文件中可使用 `BibTeX` 定义的一种微型语言来实现对参考文献列表风格的控制，但是在 `ConTeXt` 的 `bib` 模块中，仅使用 `.bst` 文件控制参考文献条目的排列次序。一般情况下，用户不需要制作 `.bst`，所以这个了解一下就可以了。

BibTeX 与 ConTeXt 的协同工作时，需要 ConTeXt 文档编译命令输出 .aux 文件。基于 .aux 文件，BibTeX 在 .bib 检索所需的文献信息，并结合 .bst 文件实现参考文献列表外观控制，最终输出 .bbl 文件。一旦我们得到了 .bbl 文件（实际是 TeX 文档），那么就可以继续进行 ConTeXt 文档编译过程，最终输出带有参考文献信息的文档，这一工作流程如下图所示。

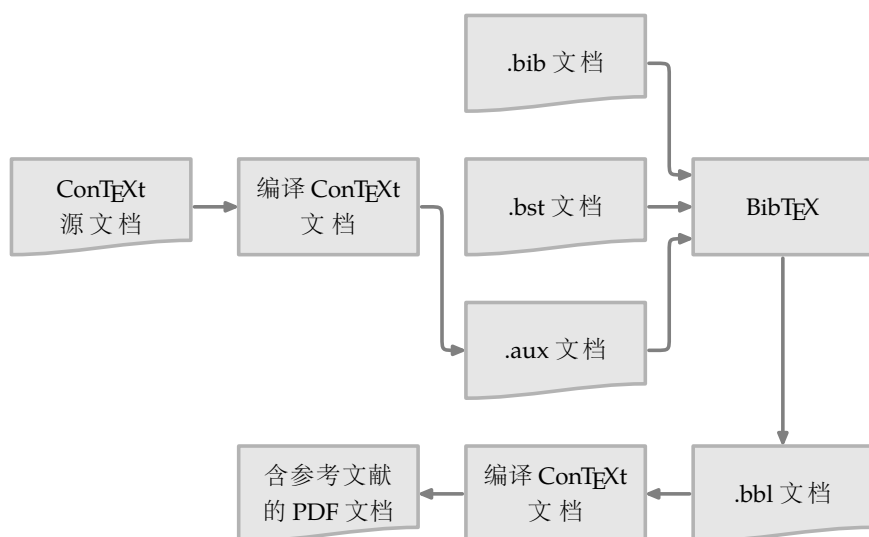


图 4.1 BibTeX 工作过程

ConTeXt 的 bib 模块

ConTeXt 是通过 Taco 写的 bib 模块取得 BibTeX 的协同（现已集成至 ConTeXt 中），用户可以在 ConTeXt 文档中实现参考文献列表排版样式的设置。下面是一份最为简单的 ConTeXt 参考文献示例文档 example/ex-4.tex:

```
\setupbibtex[database=example]
```

```
\starttext
```

这一年，诸葛亮同学出版了一本专著\cite[诸葛专著2008]，并在核心学术期刊上发表了一篇论文\cite[诸葛论文2008]，因此深得刘备的赏识。

```
\completepublications
```

```
\stoptext
```

上例中 \setupbibtex 命令的 database 参数的值为 .bib 文件名，我是将上一节中的那个 .bib 文件命名为 example.bib，然后在此使用。 \completepublications 命令会在所排版的文档中插入参考文献列表。

编译上述示例文档的过程正如图 4.1 所示的那样，需要三个处理步骤，如下：

```
$ context --once ex-4      # 产生 .aux 文档
$ bibtex ex-4             # 产生 .bbl 文档
```

```
$ context ex-4          # 完成文档编译
```

如果嫌每次编译文档都要重复输入三次命令过于繁琐，花上 20 分钟学习一下 **Makefile** 的简单用法会让我们更轻松一些。

查看一下带有参考文献的 PDF 文档，可能你会发现参考文献并非是我们常见的那种风格，不要着慌，**ConTeXt** 提供了一些用于设置参考文献风格的命令供我们使用。下面对 `ex-4.tex` 文件略作修改，添加了 `\setuppublications` 代码：

```
...
\setupbibtex[database=example]
\setuppublications[alternative=num]
...
```

重新编译一次 `ex-4.tex` 文档¹，现在参考文献的样式基本符合我们常见的风格了。有关 `\setuppublications` 命令的详细用法请参考 `bib` 模块文档[3]。

Zotero

Zotero 是 **Firefox** 的一款扩展 (Extension)，号称是下一代研究工具²，我使用它来管理参考文献数据。除 **Zotero** 之外，还有许多其它文献管理软件，比如 **Endnote**、**RefWorks**、**jabref** 等，与它们相比，**Zotero** 将我们进行文献管理的整个过程——搜集、整理、阅读和引用等步骤都很好地集成在一起，并且与 **Firefox** 取得完美的结合，莫要忘记我们通常要在网上寻找文献的。关于 **Zotero** 的详情见其项目主页³。

大多数情况下，使用 **Zotero** 的流程可以归纳为：在互联网上发现文献——利用导入功能得到文件信息和有关附件——制作文献阅读笔记、tag 以及相关文件链接——搜索文献——输出有关文献为其他软件格式——在 **TeX** 或字处理软件中引用。除最后一步外都不需要打开任何其他软件。

关于 **Zotero** 的具体用法……咳咳……实在不知道该如何用文字来讲述一款 GUI 工具的使用，又不胜一副一副配图介绍的那般繁琐。自我开脱一下，毕竟这是一份个人学习笔记，而不是一份教材，所以推荐去看 **Zotero** 的操作视频教程⁴。

设置参考文献列表的样式

在制作这份学习笔记的参考文献时，因为有一些文献来自于网络，我希望在参考文献列表中提供它们的网址 (URL) 并且要求是超级链接格式的，这样便于读者直接驱动网页浏览器去下载它们。按照 **BibTeX** 的 `.bib` 文档语法，我采用以下文献条目格式提供此类文献的信息：

¹ 若是未有改动 `.bib` 文档或者未引用新的文献，只需使用常规 **ConTeXt** 文档编译方式即可。

² 至于上一代是谁，不可考，也许是那些脱离网络环境的文献管理软件吧。

³ <http://www.zotero.org/>

⁴ http://www.zotero.org/videos/tour/zotero_tour.htm

```
@booklet{Taco,
  title = {{Bibliographies}},
  author = {Taco Hoekwater},
  url = {http://modules.contextgarden.net/bib}
}
```

但是在文档中去引用上述文献时，其 URL 信息无法在参考文献列表中显示，更谈不上超级链接格式了。在 ConT_EXt Wiki 中的 Bibliography 文档[4]中讲述了如何在参考文献中启用 URL 的方法，并提供了一个示例。我对这个示例进行了一些简化，贵在于理解其要义。将以下代码添加到文档导言区，位于 bib 模块加载及其设置代码区域的下面：

```
\unprotect
\setuppublicationlayout [booklet] {%
  \insertauthors{}{\unskip.}{}%
  \inserttitle{ \bgroup\it }{\/\egroup\unskip.}{}%
  \insertbiburl{ URL: }{\unskip.}{}%
  \insertnote{}{\unskip.}{}%
\protect
```

`\setuppublicationlayout` 命令是用于设置参考文献列表的排版布局的，上例中，其参数我设为 `booklet`，这是因为我在 `.bib` 文档中将网络文档类型定义为 `booklet`（小册子）类型，它是 BibT_EX 的标准文献条目类型之一，将其作为 `\setuppublicationlayout` 的参数，表示这里要设置该类型文献条目的排版风格。

在这份笔记中，我只需要 `booklet` 条目由 `author`（作者）、`title`（标题）、`URL`（文献网址）以及 `note`（文献注解）附加注释信息构成，因此在设置 `booklet` 文献条目的排版风格时，对于每一条目要素都对应一个 `\insertxxx` 命令。`\insertxxx` 命令有三个参数，可用于设定每一文献条目构成要素的排版环境，前两个参数分别用于设置对应排版文献条目构成要素的前后排版命令，第三个参数用于设置对应的文献条目构成要素缺省状态。以 `\insertauthors{}{\unskip.}{}%` 为例，第一个参数是空的，表示采用默认排版环境；第二个参数为 `\unskip.`，表示消除 `booklet` 文献条目中作者信息之后的空格，并且添加句点，第三个参数为空，表示当 `.bib` 文件中的 `booklet` 条目未提供作者信息时不作任何处理。

注意，在上述示例中，`\insertbiburl` 命令是启来 `.bib` 中所提供的文献 URL 信息的。如果你希望文档中的 URL 是超级链接格式，那么还需要在 ConT_EXt 文档导言区添加以下代码：

```
\setupinteraction[state=start] % 启用文档的可交互功能
```

利用上述方法并配合 ConT_EXt 各种排版命令可以灵活地控制参考文献列表的排版格式，这就是为什么 ConT_EXt 的 `bib` 模块只是将 BibT_EX 的 `.bst` 文档用来实现参考文献列表排序的主要原因。

4.5 书签

在阅读很长的 PDF 文档时，书签 (Bookmark) 是很有用的，它们通常显示于 PDF 阅读器的侧栏，用户用鼠标点击书签就可以快速打开书签所指向的页面。对于如何使用 ConT_EXt 实现书

签，由于我是没有耐心的人，所以看到 ConTeXt Wiki 上提供了适合没耐心的人对文档书签的生成快速建立初步认识的代码，便直接抄过来：

```
% 启用书签功能
\setupinteraction[state=start]
\setupinteractionscreen[option=bookmark]
\placebookmarks[chapter][chapter]

\starttext
\chapter{飞刀与快剑}
冷风如刀，以大地为砧板，视众生为鱼肉。 \par
万里飞雪，将穹庐作洪炉，熔万物为白银。 \par
.....
\chapter{海内存知己}
马车里堆着好几坛酒，这酒是那少年买的，
所以他一碗又一碗地喝着，而且喝得很快。 \par
.....
\stoptext
```

用常规的 PDF 阅读器打开所生成的文档，若开启侧栏面板，应当可以看到书签了。如果你看不到，那么就见识一下图 4.2 。

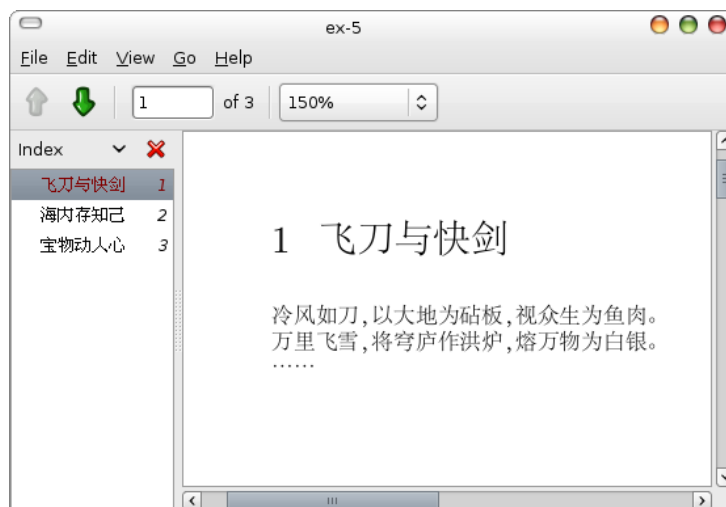


图 4.2 带书签的 PDF 文档

要开启 ConTeXt 的书签功能，必须 `\setupinteraction[state=start]`。另外，PDF 阅读器的侧栏的书签窗口默认是隐藏的，要让阅读器打开文档时自动开启书签窗口，那就是上例中 `\setupinteractionscreen` 命令所做的。

将章、节标题作为书签的想法很好，这样在书签窗口中，可以通过章节标题对文档的大致内容有所了解。在上例中，我通过 `\placebookmarks` 命令设置 `chapter` 标题作为书签。`\placebookmarks` 的第一个参数是设定各层次的章节类型作为书签，第二个参数是设置默认可见的书签。看下面一个更复杂的示例：

```
\placebookmarks[chapter,section,subsection][chapter]
```

这个示例的含义是：将章、节、小节的标题作为书签，默认只在书签窗口中显示由章标题生成的书签。

在 MkIV 中，不知是特性还是缺陷，在将章节标题作为书签时，若标题中出现了格式化文本，譬如 `\ConTeXt` 这样的文本，那么书签中就会将 `\ConTeXt` 的 `TeX` 定义显示出来，而不是显示 `ConTeXt` 或者 `ConTeXt`。现在，我未能找到解决这一问题的方法，笨方法是有的。在撰写某章节标题时，若是需要使用格式化文本，那么就在章节命令之后使用 `\bookmark` 命令手动设定书签标题：

```
\section{安装 \ConTeXt\ Minimals}  
\bookmark{安装 ConTeXt Minimals}
```

本文档目前正在使用这种方法去除书签中出现的格式化文本，勉强对付过去。

4.6 更多阅读……

如果上述内容有误或者你认为讲述的不够细致，那么请阅读以下文档：

- “`ConTeXt` 对参考文献的处理”：
<http://liyanrui.is-programmer.com/2009/12/17/bibliographic-in-ConTeXt.14078.html>
- “`ConTeXt` 对参考文献的处理——整容篇”，讲述了 `ConTeXt Minimals` 的安装方法：
<http://liyanrui.is-programmer.com/2009/12/18/bibliographic-style-in-ConTeXt.14103.html>

我计划今年年底将这些外部文档归整到这份文档中。

第 5 章 大文档项目

对于规模比较大的文档项目而言，采用 $\text{T}_{\text{E}}\text{X}$ 的优越性要高于 WYSIWYG 的字处理器 (MS Word)。众人谓之曰字处理器使用起来比较方便，实际这只不过表面现象，在其背后是文档更新方面的困难——难以进行多人协同撰写与版本维护。本章主要介绍 Con $\text{T}_{\text{E}}\text{X}$ 所提供的文档项目管理功能，探讨如何在文档项目中布置版本控制系统，建立一种比较稳定的大文档项目撰写模式。

5.1 项目的部署

一个 $\text{T}_{\text{E}}\text{X}$ 文档项目，通常可以将文档结构分为：环境、宏、文档主体内容。这样划分的意图就是让文档的样式与内容分离开。环境，主要用于定制版面样式。宏，主要是封装一些排版命令，实现一些特定的排版功能。在 Con $\text{T}_{\text{E}}\text{X}$ 中，环境或宏可以由一份或多份专门的环境文件来定义，环境文件的格式如下：

```
\startenvironment 环境文件名  
... 环境内容...  
\stopenvironment
```

如果不对文档的主体内容进行划分的话，那么它就与普通的 Con $\text{T}_{\text{E}}\text{X}$ 文档无异，只是多了加载环境文件的命令：

```
\environment 环境文档名  
... ..  
\starttext  
... 文档内容  
\stoptext
```

如果对文档的主体内容细分下去，可以有封面、前言、正文章节、附录、封底等。文档结构划分的深浅程度，这个主要由用户来把握，主要视文档的规模而定。对于 Con $\text{T}_{\text{E}}\text{X}$ 而言，文档的主体内容实际上只有产品 (product) 与组件 (component) 这两种划分¹。对于产品与组件之间的关系，这在产品文档中便可以直观地看到，譬如：

```
\startproduct 产品文档名  
\environment 环境文档名  
  
\component 组件文档 1  
\component 组件文档 2  
... ..  
\stopproduct
```

产品文档将环境与组件的信息都记录了下来，因此当我们在使用 Con $\text{T}_{\text{E}}\text{X}$ 编译一个文档项目时，只需要编译产品文档即可。

¹ 事实上还有比产品更高层的结构，就是项目 (project)，不过我们写文档一般用不到它，如果是出版期刊杂志倒是有可能用到它。

组件文档的格式如下：

`\startcomponent` 组件文档名

`\environment` 环境文档名

`\product` 所属产品名

... 组件内容...

`\stopcomponent`

也可以直接编译组件文档，因为在组件文档中已经记录了所使用的环境文档信息；在撰写文档的过程中，推荐使用编译组件文档的方式来查看排版效果。

5.2 更多阅读……

如果上述内容有误或者你认为讲述的不够细致，那么请阅读以下文档：

- “ConTeXt 文稿的物理结构”：

<http://garfileo.is-programmer.com/2011/1/12/project-structure.23773.html>

我计划今年年底将这些外部文档归整到这份文档中。

第 6 章 我是这样学习 MetaFun 的


其实，我本来是想尽快制作一份有关 ConT_EXt 演示文档的学习笔记，尝试了几次之后，都铩羽而归。我渐渐意识到如果不了解 MetaFun，那么使用 ConT_EXt 是很痛苦的，就像拿到了一座金山却无处消费它一般。我开始认真学习 MetaFun，并且也很后悔当初没有学习 METAPOST，而在 PGF/TikZ 上却曾经浪费了很多时间。

6.1 从画一个矩形开始

在这份这份文档的封面上，画有许多形态各异的矩形，它们不是我画的，而是我从一份 ConT_EXt 演示文档模板中扣出来的一段代码。我从现在开始逐步研究它们是如何绘制出来的。

先进行一个矩形区域的填充：

```
1 \setupcolors[state=start]
2 \setupcolor[hex]
3 \definecolor[BackgroundColor][h=cccccc]
4
5 \starttext
6 \startuseMPgraphic{box}
7 offset := uniformdeviate 10pt;
8 width := 2*offset + 30pt + uniformdeviate 30pt;
9 height := 2*offset + 20pt + uniformdeviate 10pt;
10 x1 := offset ; y1 := offset; x2 := width-offset; y2 := height-offset;
11 fill z1--(x2,y1)--z2--(x1,y2)--cycle
12 withcolor \MPcolor{BackgroundColor};
13 \stopuseMPgraphic
14
15 \useMPgraphic{box}
16 \stoptext
```

这个小例子可在页面中绘制一个灰色的矩形填充域 。实际上，单单要完成这个例子所实现的目标，没必要去定义那么多的变量，这里之所以如此，不妨认为我想借此来熟悉一下 METAPOST 的变量赋值与运算语法。

代码第 1-3 行的大意是启用颜色环境，并将颜色描述格式设为 16 进制码形式，最后使用灰色定义了一种名曰 BackgroundColor 的颜色；在 MetaFun 中，我们可以使用 \MPcolor 命令将 ConT_EXt 中定义的颜色转换为 MetaFun 所使用的颜色值，详见代码第 12 行。这多少可以体现出 MetaFun 的一些特点，它实现了 ConT_EXt 与 METAPOST 的沟通。

代码第 7 行演示了 uniformdeviate 命令的用法，它的作用就是从 [0pt - 10pt] 这个尺寸区间中随机取一个尺寸。

代码第 11-12 行演示了如何填充一条路径，比较值得注意的是路径构成中的 $z1$ 与 $z2$ 的含意，这似乎是 METAPOST 约定的，它们分别表示 $(x1,y1)$ 与 $(x2,y2)$ ，但愿我没有理解错。

代码第 6-13 行构造了一幅 MetaFun 图形，第 15 行代码将该图形插入至文档中。这就是 MetaFun 图形嵌入于 ConTeXt 的一般方式。另外，也可以像插入普通图片那样：

```
\placefigure
  [here,force][fig:box]
  {嵌入 \METAFUN\ 图形}{\useMPgraphic{box}}
```



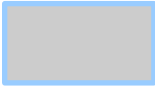
图 6.1 嵌入 MetaFun 图形

下面为这个矩形填充域绘制边框。如果只是绘制一个矩形框，这比较简单，以下示例便可以：

```
\setupcolors[state=start]
\setupcolor[hex]
\definecolor[BackgroundColor][h=cccccc]
\definecolor[OrnamentColor][h=99ccff]

\starttext
\startuseMPgraphic{box-ornament}
offset := uniformdeviate 10pt;
width := 2*offset + 30pt + uniformdeviate 30pt;
height := 2*offset + 20pt + uniformdeviate 10pt;
x1 := offset ; y1 := offset; x2 := width-offset; y2 := height-offset;
fill z1--(x2,y1)--z2--(x1,y2)--cycle
  withcolor \MPcolor{BackgroundColor};
draw z1--(x2,y1)--z2--(x1,y2)--cycle
  withcolor \MPcolor{OrnamentColor};
\stopuseMPgraphic

\useMPgraphic{box-ornament}
\stoptext
```

得到 ,但是它无法实现本文档封面页上的那些小矩形框的效果，它们是无法采用一条路径来实现，而是要四个边框逐个的画：

```
1 ... ..
2 def delta = ((uniformdeviate .5offset) + .25offset) enddef;
3 drawoptions(withpen pencircle scaled 2pt
4   withcolor \MPcolor{OrnamentColor});
5 draw (x1-delta,y1)--(x2+delta,y1);
6 draw (x2,y1-delta)--(x2,y2+delta);
7 draw (x2+delta,y2)--(x1-delta,y2);
```

```

8 draw (x1,y2+delta)--(x1,y1-delta);
9
10 drawoptions();
11 setbounds currentpicture to unitsquare xyscaled (width,height);

```



这样才可以得到那种类似于手绘矩形框的效果。注意，第 2 行代码所定义的 `delta` 似乎是一个变量，实际它是一个宏，一个不带任何参数的宏。宏可以避免许多重复的输入，并且使得代码更为易读；`drawoptions` 也具有类似作用。在上述代码片段中，宏还起到了让 `delta` 返回的尺寸大小居于随机变化的效果，因为宏在被 `METAPOST` 处理时要被其定义替换，因为 `delta` 定义中的 `uniformdeviate` 运算符就被反复调用，生成不同的随机数。`setbounds` 语句用于设置图形的边界，这样可以保证将所绘制的图形完全地显示出来（其实我觉得这有点多余）。

下面，来看一下带参数的宏该如何写。带参数的宏，可将它理解为可重复调用的函数。对于前面的示例，可以将这一系列的绘制过程定义为下面的宏：

```

def random_box (expr width, height, offset, linewidth ) =
  def delta = ((uniformdeviate .5offset) + .25offset) endif;

  x1 := offset; y1 := offset; x2 := width-offset; y2 := height-offset;

  drawoptions(withcolor transparent(1,.8,\MPcolor{BackgroundColor}));
  fill z1--(x2,y1)--z2--(x1,y2)--cycle;

  drawoptions(withpen pencircle scaled linewidth
    withcolor \MPcolor{OrnamentColor});
  draw (x1-delta,y1)--(x2+delta,y1);
  draw (x2,y1-delta)--(x2,y2+delta);
  draw (x2+delta,y2)--(x1-delta,y2);
  draw (x1,y2+delta)--(x1,y1-delta);

  drawoptions();
  setbounds currentpicture to unitsquare xyscaled (width,height);
endif;

```

`random_box` 宏是可以接受传入参数的。`METAPOST` 宏所支持的参数类型有：`expr`、`text` 与 `suffix`。`expr` 参数类型的用法正如上例 `random_box` 宏定义所示，它表示宏的使用者可以向其传入表达式的值。至于 `text` 与 `suffix` 参数类型，由于现在尚未用到它们，所以我可以暂时不理睬它们的功用。

下面演示如何使用上述定义的 `random_box` 宏：

```

offset := uniformdeviate 10pt ;
width := 2*offset + 40pt + uniformdeviate 30pt ;

```

```
height := 2*offset + 30pt + uniformdeviate 10pt ;

random_box (width,height,offset,1pt);
```

6.2 平凡的重复

一个士兵会踢正步，实在太平凡了；一群士兵会踢同样节奏的正步，就开始不平凡起来了。平凡的重复，往往是不平凡的。现在，我们开始重复上一节所绘制的那个 Box。先来看下面这个可编译的示例：

```
1 \startuseMPgraphic{random-box}
2 def random_box (expr width, height, offset, linewidth ) =
3   def delta = ((uniformdeviate .5offset) + .25offset) enddef;
4
5   x1 := offset; y1 := offset; x2 := width-offset; y2 := height-offset;
6
7   drawoptions(withcolor \MPcolor{BackgroundColor});
8   fill z1--(x2,y1)--z2--(x1,y2)--cycle;
9
10  drawoptions(withpen pencircle scaled linewidth
11    withcolor \MPcolor{OrnamentColor});
12  draw (x1-delta,y1)--(x2+delta,y1);
13  draw (x2,y1-delta)--(x2,y2+delta);
14  draw (x2+delta,y2)--(x1-delta,y2);
15  draw (x1,y2+delta)--(x1,y1-delta);
16
17  drawoptions();
18  setbounds currentpicture to unitsquare xyscaled (width,height);
19 enddef;
20
21 for i=1 upto 400 :
22   offset := uniformdeviate 4pt ;
23   width := 2*offset + 10pt + uniformdeviate 10pt ;
24   height := 2*offset + 5pt + uniformdeviate 4pt ;
25
26   addto currentpicture also
27     image(random_box (width,height,offset,1pt)) shifted
28       (uniformdeviate 8cm, uniformdeviate 6cm) ;
29 endfor ;
30 \stopuseMPgraphic
```

我得到如下图所示的图形：

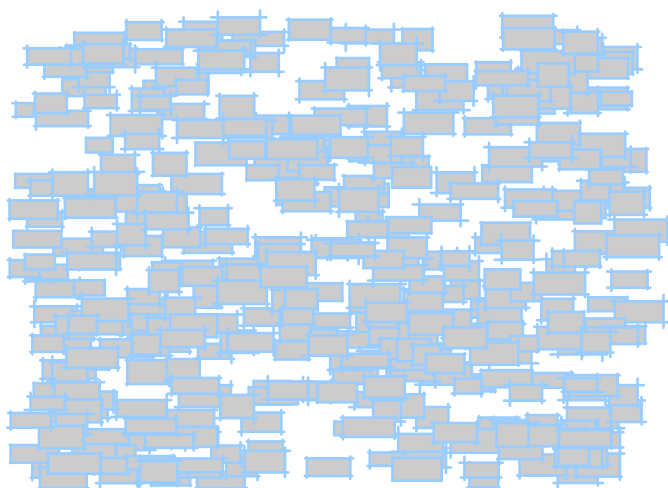


图 6.2 平凡的重复

第 26-28 行是解决 `random_box` 重复绘制的核心代码。`addto ... also` 是 METAPOST 内部定义的宏，它可以将一幅图形 (picture) 添加到另一幅图形中。图形是 METAPOST 中的一个数据结构，它可以存储多条已被绘制的路径，并且图形与图形之间是可以包容的。`image` 也是 METAPOST 内部定义的宏，它可以将某一被绘制的路径转变为图形。至于 `shifted`，顾名思义，它是用来进行几何平移变换的，而且变换的位置是随机坐标来确定的。

6.3 Overlay 与背景

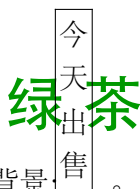
Overlay 这是 ConTeXt 的一个术语，它的名字取的不是很正确，其本意是“覆盖，覆盖图”，但是在 ConTeXt 中，它通常是位于文本之下的图形，或者说它是被文本覆盖。不过，`overlay` 可以分为许多个层次，因此这个术语的名字取得也不是完全不正确。

很多时候，我都是将 `overlay` 作为一个背景层来对待的，下面的这个例子便可以说明这个问题：

```

1 \setupcolors[state=start]
2
3 \starttext
4 \defineoverlay[tea][{\darkgreen\ss\bf 绿茶}]
5 \framed[background=tea, align=middle]{今\\天\\出\\售}
6 \stoptext

```



这个例子可以将“绿茶”作为“今天出售”的背景。

`\defineoverlay` 的命令使用起来很简单，第一个参数是 `overlay` 名，第二个参数是 `overlay` 的内容，通常是一组排版或绘图命令的集合，推荐使用 `{ }` 将其括起来。

在上述示例中，已经演示了使用文本作为 `overlay` 内容，下面来看一下使用图片的例子：

```
\defineoverlay[cow][{\externalfigure[cow]%
  [width=\overlaywidth,height=\overlayheight]}]
\framed[width=3cm,height=2cm,background=cow,align=right]
  {\vfill\color[darkred]{这是一头奶牛}}
```



于是我们便拥有了一头漂亮的奶牛。

上述以图片作为 `overlay` 的代码没必要多费口舌，多看几遍，动手尝试一下便可以明了。值得注意的倒是 `\framed` 命令的 `align` 参数，它的取值有些另类，明明是左对齐，但是它的值却是 `right`；如果你想让文本右对齐，就不得不将其设为 `left`。这实际上是 Hans 犯得一个错误，因为他当年在写 `ConTeXt` 时，满脑子是 `ragged left`（左边不齐）与 `ragged right`（右边不齐）。这样的错误一旦犯下了，便不好再改，主要是因为无法兼容过去的文档，后来 Hans 又加入了 `flushleft` 与 `flushright` 参数值，这是与我们普遍的习惯是相符的。

我们还可以将 MetaFun 绘制的图形作为 `overlay`，这也是本章一个非常重要的内容，它是 MetaFun 与 `ConTeXt` 集成的基础，来看这个示例：

```
\startuseMPgraphic{demo circle}
  path p ;
  p := fullcircle xscaled \overlaywidth yscaled \overlayheight ;
  fill p withcolor \MPcolor{BackgroundColor};
  draw p withpen pencircle scaled 2pt withcolor \MPcolor{OrnamentColor};
\stopuseMPgraphic

\defineoverlay[demo circle][\useMPgraphic{demo circle}]
\framed[background=demo circle,frame=off]{被椭圆包围的文本}
```

我们可以得到 被椭圆包围的文本。

由上例可见，我们发现 MetaFun 可以将 `ConTeXt` 中的一些尺寸，比如 `\overlaywidth` 与 `\overlayheight` 等参数传入 MetaFun 环境中。在本章开始的时候，还用过一个 `\MPcolor` 宏，可以将 `ConTeXt` 的颜色定义转换为 METAPOST 的颜色定义。我们还可以通过使用 `\the` 作为前缀，实现在 MetaFun 环境中访问诸如 `\textwidth`、`\textheight`、`\topspace` 等尺寸参数。

前面说过，`overlay` 是可以叠加的，下面再来用一个例子演示一下：

```
\startuseMPgraphic{demo ellipse}
  path p ;
  p := fullcircle xscaled (.95*\overlaywidth) yscaled (.95*\overlayheight);
  fill p withcolor \MPcolor{BackgroundColor};
```

```

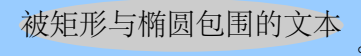
draw p withpen pencircle scaled 2pt withcolor \MPcolor{OrnamentColor};
\stopuseMPgraphic

\startuseMPgraphic{demo square}
  path p ;
  p := fullsquare xscaled \overlaywidth yscaled \overlayheight;
  fill p withcolor \MPcolor{OrnamentColor};
\stopuseMPgraphic

\defineoverlay[demo ellipse][\useMPgraphic{demo ellipse}]
\defineoverlay[demo square][\useMPgraphic{demo square}]

\framed[background={demo square,demo ellipse},frame=off]{被矩形与椭圆包围的文本}

```

我们可以得到 。这其实没什么技巧，只需多定义几个 `overlay`，然后将它们作为 `\framed` 的 `background` 的参数列表并按照先后的显示次序排列即可。

本节所举的例子，只是使用 `\framed` 命令来演示。事实上，`ConTeXt` 有许多命令都支持 `overlay`，只要它具备 `background` 参数。

6.4 代码的分离

有时候，会希望能够在多个 `MetaFun` 图形环境中引用同样的一段代码，在我不知道存在 `MPinclusions` 环境时，我写过下面这般风格的代码：

```

\startuseMPgraphic{TitlePage}
def random_box (expr width, height, offset, linewidth ) =
  ... ..
enddef

for i=1 upto 400 :
  ... ..
  addto currentpicture also
    image(random_hash_frame(width,height,offset,1pt)) shifted
      (uniformdeviate OverlayWidth, uniformdeviate OverlayHeight) ;
endfor ;
\stopuseMPgraphic

\startuseMPgraphic{PageNumberBox}
def random_box (expr width, height, offset, linewidth ) =
  ... ..
enddef

offset := uniformdeviate 10pt ;
random_box (\overlaywidth, \overlayheight, offset, 1pt);
\stopuseMPgraphic

```


虽然这样的代码可以用，但是它很脏。比如，如果我想修改 `random_box` 宏的定义，就意味着这两个图形环境中的 `random_box` 都要修改。当我我在 MetaFun 手册中发现了 `MPinclusions` 这个环境时，立刻就将上述的代码格式修改为：

```

\startMPinclusions
def random_box (expr width, height, offset, linewidth ) =
    ... ..
enddef
\stopMPinclusions

\startuseMPgraphic{TitlePage}
for i=1 upto 400 :
    ... ..
    addto currentpicture also
        image(random_hash_frame(width,height,offset,1pt)) shifted
            (uniformdeviate OverlayWidth, uniformdeviate OverlayHeight) ;
endfor ;
\stopuseMPgraphic

\startuseMPgraphic{PageNumberBox}
offset := uniformdeviate 10pt ;
random_box (\overlaywidth, \overlayheight, offset, 1pt);
\stopuseMPgraphic

```

可以看出，`MPinclusions` 环境的作用就是用于存放各 MetaFun 图形的公共代码，类似于 C 程序的 `.h` 文件。

我们还可以在 MetaFun 图形环境中使用 `\includeMPgraphic` 来引用其它的 MetaFun 图形：

```

\setupcolors[state=start]

\starttext
\startuseMPgraphic{test}
    fill fullsquare rotated 45 scaled 4cm withcolor \MPcolor{yellow} ;
\stopuseMPgraphic

\startuseMPgraphic{demo}
    \includeMPgraphic{test}
    fill fullsquare scaled 2cm withcolor \MPcolor{red} ;
\stopuseMPgraphic

\useMPgraphic{demo}
\stoptext

```

6.5 还有一些图形环境

除了 `useMPgraphic` 环境之外，还有 `uniqueMPgraphic`、`MPpage` 图形环境，当然还有其它类型的，只是我目前认为掌握这三种，基本上够用了。

`uniqueMPgraphic` 环境与 `useMPgraphic` 环境的区别是：前者只生成一份图形，可供多次使用；后者是在每一次使用时，都要生成图形。如果是要实现随机图形效果，也就是说多次使用同一份绘图代码生成的图形却不相同，这就只好使用 `useMPgraphic` 环境，这个问题在 6.2 节中有所体现。

`MPpage` 环境可专门用于生成单独的矢量图形，试验一下这个例子便可体验其用处：

```
\setupcolors[state=start]
\setupMPpage
  [offset=1pt,
   background=color,
   backgroundcolor=gray]

\startuseMPgraphic{test}
  fill fullsquare rotated 45 scaled 4cm withcolor \MPcolor{yellow} ;
\stopuseMPgraphic

\starttext
\startMPpage
  \includeMPgraphic{test}
  fill fullsquare scaled 2.5cm withcolor \MPcolor{red} ;
\stopMPpage
\stoptext
```

6.6 后记

这篇文章的内容，支离破碎。这是自然的。因为 `MetaFun` 所涵括的内容远非我这一篇小文所能覆盖得了的。我以为 `MetaFun` 手册最好是能通读一遍，不过我现在做不到，只好用什么，看什么。在这用与看的过程中，我想尽量地建立系统的认识。这是一个长期的过程。我唯一能做的就是尽量寻找可以使用 `MetaFun` 的机会。

我觉得使用 `ConTeXt` 排版文档，如果能够充分结合 `MetaFun` 图形环境来设计文档版面的外观，这是非常具有挑战性的。个人的审美观以及绘制图形的能力决定了文档样式能否体现出个性。

参考文献

- [1] Nicolas Markey. Tame the BeaST: The B to X of Bib \TeX . <http://www.ctan.org/tex-archive/info/bibtex/tamethebeast/>
- [2] PRAGMA. Fonts in Con \TeX t. <http://context.aanhet.net/svn/contextman/context-reference/en/co-fonts.pdf>
- [3] Taco Hoekwater. Bibliographies. <http://modules.contextgarden.net/bib>
- [4] Taco Hoekwater. Bibliography. <http://wiki.contextgarden.net/Bibliography>
- [5] PRAGMA. WIDGETS uncovered. <http://pragma-ade.com/general/manuals/mwidget-s.pdf>
- [6] PRAGMA. Con \TeX tthe manual. <http://www.pragma-ade.com/general/manuals/cont-eni.pdf>
- [7] PRAGMA. Con \TeX tan excursion. <http://www.pragma-ade.com/general/manuals/ms-cb-en.pdf>
- [8] Hans Hagen. METAFUN. <http://www.pragma-ade.com/general/manuals/metafun-s.pdf>
- [9] Alpha Huang. L \TeX 学习笔记. <http://www.ctan.org/tex-archive/info/lnotes/lnotes.pdf>

未完成……