

第 18 讲 用例学习：Tagger

18.1 总论

在本讲中，我将讲解标记（Tagger）的设计；所谓标记，就是我去年夏天写的一段小程序。在我近几个月的著述中，我都用了标记。标记在此文档中，在从 2001 年 6 月至今的我的著作中都有使用（见<http://sdg.lcs.mit.edu/~dni/publications>）。

我之所以选择标记作为我的第三个用例学习有几个原因：首先，它是我自己写的一段程序，所以我知道它比其他的更好；其次，标记为你们正在学的好几个模式提供了范例，例如从我们第一个用例学习——Java Collections API——角度看，标记很有用；最后，不像先前的两个用例学习，标记工作量很小的，因此它更像你在最终项目中所希望做到的。我先花了几个星期设计标记，谈后又用了一个星期的时间才构建完成。

我将向大家作详细的阐述。

18.2 目的

标记是一段很小的文本处理应用程序，常用于科技论文和书刊中。它被用作一些 WYSIWYG 编排程序的前端，像 QuarkXpress 和 Adobe Indesign，综合了他们自身的优点和一些像 TeX 一样基于编辑的工具的优点。

像 TeX 这样的工具的优点就是它们允许用户在一个强大的文本编辑器中编辑文本，并且很容易通过电子邮件进行文本交换。由于样式化是由原文的标记确定的，你就可以用来改变文本本身相同的机制来改变该文本的样式化，像查询和替换。有时候没有必要选择特殊的字符组成一个字符集，精确的字符可以用模糊的代替（像用\alpha 代替希腊字母的 α ）以加快打字速度和减弱文本对标准样式的依赖。通过对段落明确地赋以象征性名称并引用那些名称，交叉参考可以轻松地实现

另一方面，像 TeX 一样的工具有着很严重的问题。它们不能适应现在需要的广泛的附带样式，不能满足用户相当大的专用化需求。调整设计是一件非常困难的事情；一个小小的改动，像是改变页边的空白或者改变标题的间隔，经常需要相当大的专业技术。还有，经过它们处理的文档的印刷质量要比那些现代的设计工具处理的差得多。举个例子，Quark 和 Indesign，都允许设置“基线网格”作为行的基准，这样文本的每一行会印刷的非常整齐。他们的字符连接算法看来运用得非常好。Indesign 为我们应用开放类型的样式提供了方便之门。

标记的方法非常简单。用户使用一种非常少用的语言写一篇文档。在样式化上，这种少用的语言，除了把文本加粗、斜体等简单操作之外，并不提供直接的控制。所以，替代性的，段落便以段落样式的名称为标记。标记使文档转变为设计程序标准输入输出中的文件，就像在 Quark 中一样。应用 Quark 时，用户可以设置一个样式卡片来为每一段设定排版上的特色。这样随着段落的输入输出，它们也就被按照样式卡片中最合适一种样式排版了。

当然，一个人可以非常容易地为设计程序写输入输出文件。但是每一种设计程序有不同的输入输出样式。尽管标记现在还仅仅能对 Quark 产生输入，但它会很方便地为 Indesign

和其他类似的设计程序提供支持。同时，输入输出的样式越来越趋向于变得水平更低，将会比我们用非常稀用的语言写还要笨重。奇怪的是，Indesign 的输入输出样式甚至不能在文本编辑器中准备好，尽管它依靠从回车符号中得到的有特色的换行。标记还是可以把为精确字符所设置的象征符号转化为样式和索引信息；在输入输出样式中，也就取代了写“像用\alpha 代替希腊字母的 α ”这样的话，而你必须为字符发生的标准样式和索引确定准确的名称。

18.3 特色

标记有以下特色：

- 为段落提供样式名称的标记。每一种样式都有一个特别的样式卡片为它设定缺省的样式，这样，在很多例子中一个段落也就不需要精确地标注了。
- 为段落自动编号。样式卡片要详细说明哪些样式需要编号，需要什么样的编号层次（比如部分在小部分之上），要设置什么样式代号（照字母次序，罗马语还是阿拉伯语，等等），以及这些代号串该如何用前导、后续和离散串来组合。
- 为特殊字符作象征命名。映射文件将象征名称转变为样式名称 / 索引对。标记带有一些标准的映射文件，并且它还可以简便地书写新文件使其他样式的符号明白易懂。
- 数学模式。在美元符之间的文字被当作标准文字来处理。字母字符下面标注横线，而其他的数字、标点符号以及特殊的符号形式不变。
- 交叉引用一个段落一旦被用标记标记，在其他地方的引用这个标记都会产生一个引用该段落的串。由于缺省，这个产生的串就是由自动编号工具创建的数字串，并且用户可以明确地区分每个串。这种与自动编号机制结合的特色，使应用提要引用变得更容易了。
- 基本字符样式化。文本可以被设置为斜体，标注下滑线等。
- 空格。空格都将被保存，这样就可以很容易地用 tabs 或 spaces 对文章进行缩进。
- 速记。提供几种常用的速记方法：例如，一个省略号会自动修改为三个点，两个连字号修改为一个破折号，等等。标记有下滑线的文字将被用斜体排字。根据上下文语境，反向逗号会明确地改为引号：“It’s good to be a ‘software engineer’ in ’01”。

18.4 设计概述

基本机构非常简单。主类就是标记，用 SourceParser 类将输入的文本解析成一串 Token 对象，每一个 Token 对象有一个 Token 类型；Token 对象被转化为 Engine 对象，而 Engine 对象会把 Token 类型描绘为一系列的操作对象。每一个操作对象将会被执行。一个操作的典型作用就是产生原文的输出——这是通过一个对操作对象隐藏样式化语言真实输入选择的界面发生器。现在，只有一个类来实现产生 QuarkXpress 输入文档的 Generator，QuarkGenerator。

一些操作对象可以使文件被读；例如：源文件\loadstyles(foo.txt)会导致 foo.txt 被当作一个样式文件被解析。样式文件和字符集合有相同的语法：一序列行，每一行都有一系列的属性，这些属性都由属性名和属性值组成。这两种类型文件的内容会被描述为属性映射对象；每一个这样的对象含有一个从属性名称到属性列表（就是一些列有属性名称和属性

值的属性对象)的映射。这个 PropertyParser 类是一个属性文件的解析器。

编号类的作用是产生编号串。尽管样式文件含有编号指令,但每个样式文件还是会产生一个类实例。

模型依赖表显示标记程序的模型和对另一个标记程序的依赖性。有点的轮廓界把有相同从属性的模型分离出来:这可以使我们避免有时需要画从一个标记类到几乎其他所有类的从属线的麻烦。标注从依赖边的数字涉及到一系列意外依赖的内容,不希望属性会解释为什么一些从属不应该存在而又确实存在。举例来说,看到从标准引擎到从属性记录,你就会知道标准引擎代码(事实上是它的内部匿名的类)产生了一个关于交叉引用的索引文件。这样,就需要访问属性。属性在读和处理样式文件与字符集上的其他的作用,就会被像属性映射这样优先级低的类操作。

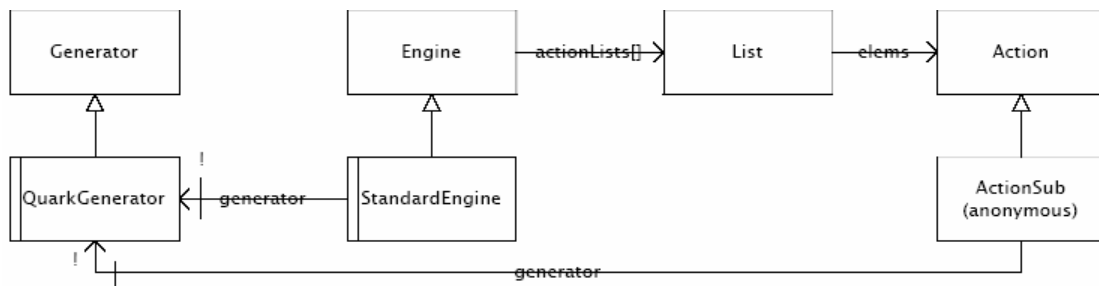
18.5 设计要点

以下是一些设计时值得注意的地方,其中有一些是通过附对象模型的插图来说明的,有时候是代码,有时候是根本的概念结构。

18.5.1 发生器界面

通过发生器界面,操作和发生器就可以相互作用了。这保证了设计不依赖于某种具体发生器的特色。只要能得到界面中不同发生器所共同的属性,就可以很容易地使应用程序为不同的设计工具产生输出(就像除 Quark 外还有 InDesign 和 PafeMaker)。这样,为电子邮件编写发生器以产生相应的 ASCII 码文本,也就很容易了。

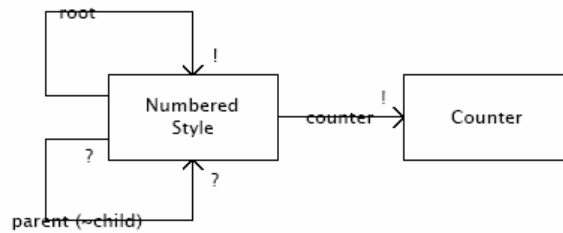
对象模型会表明操作、具体的发生器以及发生器界面之间的关系。



18.5.2 编号

每一种样式都要么是已编号的要么是未编号的。如果是已编号的,那它就是一系列的,其中有一个根样式,其他样式与根子体间都有一个指针。假如现在有一条从样式 s 到样式 t 的指针,那么我们就称 s 是 t 的父样式,t 是 s 的子样式。如果一系列样式中只有一个样式,那么它就是根样式,只是没有子样式。通过指定样式之间的父子关系(如果有的话)并给样式一个计数器属性(如果是已编号的),这种结构就可以很容易在样式文本中确定了。编号算法是这样工作的:每个被编号的样式都有与其关联的计数器,计数器初始化值比最初产生的值小 1。当遇到有已经编号过的样式的段落时,它的计数器中的值就会增加,并且它的所有后代的值都会重设。通过连接到它的前导会构建一个编号串,当前从根到该样式间所有的祖先的计数器的值,都被分离器与它的后代分离开来。每种样式的前导、分离器、后续都在

样式文件中给定。



对象模型表达了编号对象维系的关系。以下是不变的规则：

- 如果 s 是 t 的父亲，那么 t 就是 s 的孩子；反之亦然。
- 每个样式结点都指向根样式结点，要么是它自己（如果没有父结点），要么是它的没有父结点的祖先。
- 系列分离：一个样式结点不能从属于两个系列；这样也就不存在两个不同的结点共享一个父结点或子结点。
- 一个样式结点不能是自己的父结点或子结点。
- 如果一个样式被编号，它的祖先也必须被编号。

18.5.3 索引文件

前面涉及的操纵的问题可以象在 LaTeX 中一样来处理。运行时，会产生一个联系引用标记与插入其中的引用串的索引文件；前面的引用并没有完全断，会在下一次运行工具时再处理。再次运行时会从索引文件产生新的联系，并不与以前的相冲突；这样如果经过一系列的编辑操作后，就有可能产生错误的交叉引用。但是，第二次运行工具之后就可以产生正确的结果了。

这个方案必须很容易扩展为引用多重源文件。

对象问题模型表现出了交叉引用代之间概念上的关系。每一段都可以有一个标记和一个编号串，其中一个用作对该段落的遍历引用，并且标记比编号串更重要。一个段落可以有一个用作被其他段落引用的标记，也可以引用其他段落的标记。如果 t 是 q 段落的标记，且另一个段落 p 引用 t 时， p 也就引用了 q 。注意标记不能被超过一个的段落共用；它们是独特的标示符。

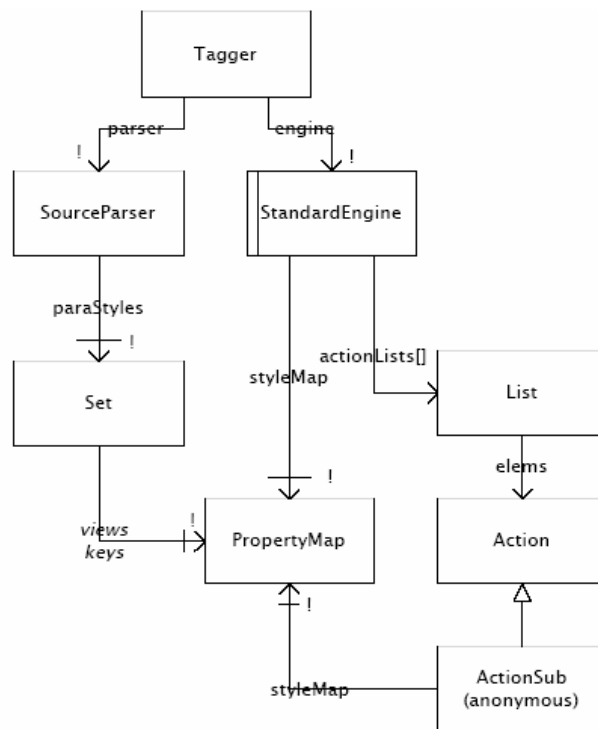
18.5.4 属性图

样式表与字符集有相同的构造方法；并且内部用相同的抽象数据类型来描述，即属性图。这就允许两者用相同的分析器。通过交叉引用机制产生的索引文件也可以用相同的构造方法和描述。编号类通过增加新的、多余的工具使编号串产生的更容易，这也实际上增加了样式表的内在描述。

18.6 样式设置映射

因为语法并没有要求段落样式名称要做任何方式的特殊标记，源分析器还是必须能从其他命令中区分出段落样式名。因此，源分析器是用对一系列样式名的引用所构建的。但是，开始时并不知道样式名。当一个样式表录入时，对文件的读操作会产生属性映射。过程就是这样：开始时引擎产生一个空的属性映射，送入源分析器的设置是对图上的一个映射。当样式表录入时，属性图由于增加了新的样式而变化；同时映射也随之变化。这种机制很严谨，但的确允许我们从属性图中减弱源分析器。

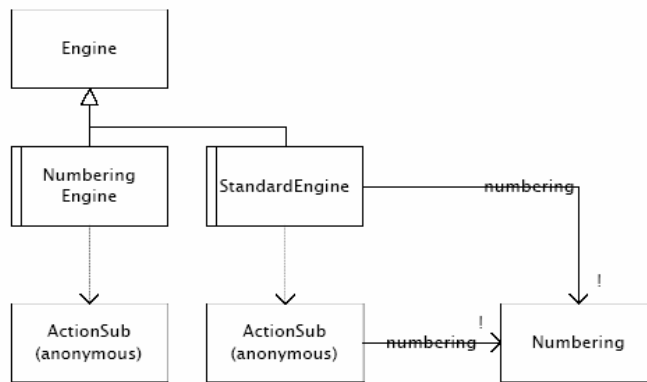
对象模型表明映射是如何连接到源分析器和代码中标准引擎的。



18.6.1 多重引擎

人们可能希望引擎 **Engine** 类只有一个，但是并不是这样。事实上，有一个引擎对象负责操纵源文件，第二个是一个有很少功能的引擎，用来产生编号串。至于我们上面提到的比较古老的语言，可以应用于给样式文件提供编号向导。例如，在目标段落的开头会产生一个首要的点，这是由于段落是用样式 **point** 来标注的，而且样式文件中的编号向导说明即便是 **point** 还没有编号，编号串仍应该有一个首要的点和 **tab**。这个首要的点可以通过象征性的名称 **\ periodcentered** 来访问。编号对象产生一个包含象征名称的串作为子串，这个子串必须像在源文件中的象征 **tokens** 本身一样被分析、被操作。

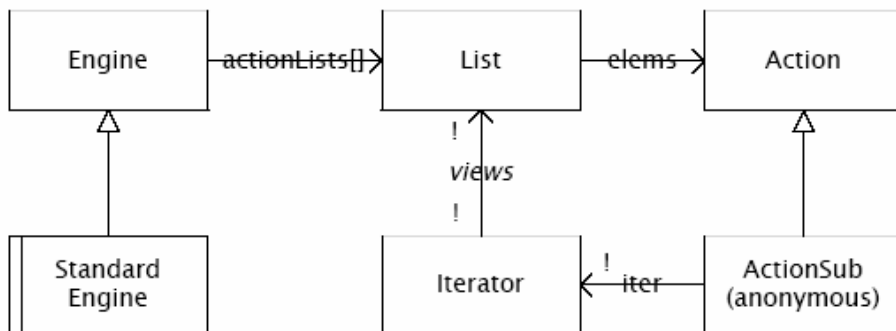
对象模型说明有一些操作可以引用编号 **Numbering** 对象。结果串通过自己的操作被送到一个不同的编号引擎，这并不显示。带箭头的线粗略描述了引擎和它的注册操作间的关系。



18.6.2 修改

执行引擎应用一个迭代器来遍历与象征类型相联系的操作对象列表。这些操作对象的执行方法将被运行；并且如上所述，其执行方法能自己注册和注销操作对象。如果对象象征类型这样做，就会使上面迭代的列表得到改进，现在的版本下，如果迭代器还在运行就去修改列表，这是违反 Java 规则的。能这样做的唯一的例子就是必须采取“一触”式操作时，该操作可以在操作发生的同时注销自己。为操纵这些，引擎产生迭代器作为对功能的执行方法的证明，调用迭代器的移除方法从对象列表中移除操作对象。这是移除方法的一种不寻常而有相当模糊的用处，因为调用移除的点距离循环的点很远。

对象模型展现了引擎操作列表中的一个操作是如何通过一个迭代器间接访问这个列表的。



18.6.3 动态注册

一个操作可能导致另一些操作的存储或注销。举个例子来说，当美元符第一次碰到时，会执行一个存储“下划线操作”的操作，这是与字母顺序序列的 Token 类型相违背的。当第二次碰到美元符的时候，这个操作就被注销了。结果就是处在两个美元符之间的字母会被标注入下划线。

18.6.4 匿名内部类

大部分的操作对象是用匿名内部类来执行的。为了解释这些，必须注意匿名内部类访问局部变量的方法。为这些变量赋值在 Java 中是不允许的，即便局部环境并不是严格意义上的局部。这就是为什么用于描述一个段处理状态的变量要被压缩在 `ParaSettings` 类的一个对象里。注意，这样的变量很少；这是基于操作的主要优点之一。

18.6.5 安全类型列举

不同的列举，像 `Format` 和 `TokenType`，是以安全类型的方式执行的，这个安全类型的依据就是 Bloch 的 *Effective Java* 第 21 条描述的习惯用语。这一点并不像我们平时用限于整形的静态变量来描述列举，很好的保证了类型的安全。也不像 ML 语言中的代数数据类型，但是这并不允许编译器证明一个用例描述操纵列举的每一个值。

18.7 供选择的设计

以下是一些不同的设计，我本来可以用却没有进行选择：

- 用基于行的脚本语言，像 Perl, Sed 或者 Awk。它们不能很好地与像标记这样的应用程序相匹配，因为在这些应用程序中，行是没有意义的，并且上下文（像斜体模式是否在执行中）必须通过行间被装载。我也没有耐心去调试一个复杂的 Perl 脚本，而更喜欢用类型安全语言。
- 一种经典的对象导向的设计；在这种设计中每一个象征类型被描述为一个自己的 Token 类中的子类；操作就是这些子类的方法，依赖动态分配来选择对象征的操作。这种设计方法很简单，但是它产生了大量的类。更糟糕的是，它在很多类之间分割功能；例如数学模型就不能在一个地方编号，而是在所有涉及到的象征中进行的。问题出在它是根据 visitor 模式的。这种方法不允许动态地改变操作状态，而这正是我的设计中所能做到的。
- 另一种设计方案，在其中根据象征类型的操作选择是被确定了的，要么是被一个大的用例状态，要么是用 Visitor 模式的方法。这种方法需要声明大量的全局变量，所以要实现功能，像数学模型的编号，就会使整个的用例状态受影响。相反地，在我的基于操作的设计中这些功能几乎是被压缩了的。
- 一个用抽象的语法树而不是用一个标记流的标准编译器组织。这将有更加大的灵活性，并且允许更好地报错功能，但是也会需要更多的工作量来执行。

18.8 设计缺陷

下面是标记的一些已知的缺陷：

- 由于标记并不能看到最后在页面上的设计，它就不能操纵基于页面的发布，如脚注和运行页眉。用一种更具表现力的语言（像一些市面上为 Quark 出售的 third-party 格式），就有可能操纵这些。插入图表也遇到同样的问题：现在图表必须通过手动才能在设计程序中插入。

- 标记并不提供处理表格的工具。
- 标记不能提供 Tex 的精确运算的特色：它不能做多线程运算，包括求和和积分。
- 我原本希望能包含 LaTeX 和 HTML 的嵌入语句，但是，那需要把大量的功能结合在操作中才能实现像与其他设计程序的嵌入语句一样简单地执行，像 InDesign 和 Pagemaker。
- 样式文件现在并没有适当地选中。在编号关系中的错误（例如标注一个样式使自己成为自己的父亲样式）会导致标记出现故障而没有适当的警告。
- 报错并不总是有帮助的。例如，在分析属性文件时发现的错误并不能报告所在行数。
- 属性文件语法在代码中是在两个地方描述的：一处是在属性图的清除转储方法中，一处是在属性分析器的分析方法中。这是一种不好的耦合。
- 符号集和样式表能不考虑对方而且当这种情况发生时并不发出警告。如果两个符号集定义了相同的象征符号名，第二个定义才被应用。一个样式名能屏蔽一个符号名。例如，样式名“section”可以屏蔽相同名称的字符，并且阻止对“section”符号的访问。
- 进度报告是一个糟糕的设计。如同编号串的产生，进度报告被传送到一个特别的引擎，以清除不在控制台上显示的字符。这种机制假定对段落编号的遍历是一种显示进度的合理的方法。通常情况下是的，但当编号被用作小的项目时，就不是这样了。
- 尽管大部分的操作可以单独地理解，但在他们中的一些之间有微妙的联系。举例来说，与开始一个新段落相关联的动作，包含了几个操作，注册和注销，陈述几个操作之间的联系，压缩入 ParaSettings 目标。这反映了这个问题的“上下相关并且很微妙”的性质：在这个例子中，如果外部有段落样式命令，则在段落的开头应该通过缺省产生一个样式标示。
- 有一些字符（像<）是不能在源文件中使用的，因为他们会被 Quark 解释为控制字符。标记不能识别他们，也不能恰当地限制他们；这样用户必须用象征符号来引用他们（如用 \less）。
- 现在还不能提供符号样式，但是会很快加上的。
- 引用记号解疑还不能正确操纵所有的例子。

18.9 开发过程

标记最初是作为一个 Perl 脚本而写的，以实验为 AdobeIndesign 产生输入的构想。我曾经一直想以 Indesign 的输入样式写文档，但是发现那太繁重了，尤其是它还在回车和换行之间有区别，因此也就没法在一个文本编辑器中准备好。这个实验是成功的，让我变得更有信心，并增加了一些内容，像自动编号。Perl 脚本是脆弱的，并不容易维持，所以我决定用 Java 版本代替。

我花了好几天设计源语言。随着我一步步发展这个执行（标记），并且发现什么是容易分析的，什么是容易写的，这种源语言也变得“精致”了。我以执行源分析器开始，尽管我讨厌些分析器，并且希望别人能做好。最初的设计就是一些列目标模型和模块从属的图表。

我并没有执行任何一个单元测试，因为最复杂的部分在类里（就像标准引擎），不能轻

易地单独测试。我可能原本应该为小的数据类型写一些单元测试，像计数器。把程序整个来测试是手动完成的，即通过目测输出以及看 Quark 是如何执行的。

随着我把它应用到我的著作里，这段程序在几个月里经常被改进。迄今为止我已经在代码中发现了四处漏洞 (bugs) (3kboc, 包括注释)。由于病态例子很少出现，甚至在这个工具的扩展应用中也没有显示出来，我相信没有多少别的漏洞了。假使这段程序只为个人使用，那么我愿意让我的日常使用作为对它的测试；当然如果这段程序是要发布的，正确的测试就是必须的了。我已经为代码做了大约二十处补丁，以便能跟上源语言的改进。

在为该程序更广泛的发布所做准备中，我为大众使用而加了说明书。因为我只是一个程序员，我曾经依赖于为灵巧的程序写规格。我还恰当地重新设计代码，比如介绍安全类型的范式。为减少出现介绍中出现漏洞的风险，我写了一个拙劣的衰退测试框架（伴随着主标记类的运行测试方法，它会把产生的文件当作先前早就产生过的。

18.10 用户指南

以下是很粗略并且未完成的用户指南：

18.10.1 命令队列的变量

标记应用程序的调用需要给出要执行的文件名；可选择的，需要给出在源文件中涉及的应该解释的文件路的径名。源文件的给定不需要扩展名；扩展名被假定为.txt。产生文件被定义为.tag.txt 文件。由于缺省，标记尽力打开源文件中涉及的文件（在它们指定的区域），只有当这样失败时，才考虑可供选择的路径名。

18.10.2 总体结构

在用一个字符符号之前，必须先通过\loadchars 命令载入一个定义该符号的文件。在用一个样式名之前，先要通过\loadstyles 命令载入一个定义它的样式表。在文件顶部的导言中载入字符集和样式表，是很方便的。

18.10.3 词法

源文件被分解为段落。文件的首次印刷文档开始一段。段落之间用空白行来隔开（就是，要么空白要么包含 tabs 或 spaces），也可以用特别符号\p，\p 一般用在短的段落（像几行需要编号的代码），用户并不想用空白行来分离它们。

命令必须加前缀“\”。两道“\”表示手动的断行。

命令被分为产生输出命令（如：\alpha 会产生 α ）和不产生输出命令（如段落样式命令\section，会导致格式变化但不产生在最终文档中作为文本的输出。）

空白格一般也是保存的，除非空白格紧跟在 non-printing 命令之后。这允许用户在前一行用样式名来标记段落，并取消了断行。在设计程序的字体表中，无论什么样的锯齿状被指定，Tabs 都可以生成。

尽管一个命令并不能立即后面跟着文档，但是后面需要紧跟着空白格。有个特别的命令

`\eat` 可以消除紧跟的空白格。

连字符和点被分别转化为破折号和句号。例如，一个连字符会被当作连字符，两个连字符会被当作一个半方破折号，三个连字符会被当作一个全长破折号。引号会根据上下文而修正。

为把有特殊意义的字符作为平常字符插入，可以预先加一个“\”，例如打 `\eat` 就会产生 `\eat` 串。

18.10.4 命令

- 段落样式。如果样式 `style` 是一个在已录入的样式表中定义了的样式名，那么跟在一个段落间隔后面或者在文件开始的 `\style` 命令就表明以它为开头的段落被设为 `style` 样式。在后面附加一个星号 (`\style*`) 会抑制编号产生：没有编号串产生并且计数器也不会增加。缺省的段落样式 `body` 用于那些没有用外在样式标记的段落，和不能通过跟在一个样式之后得到一个在样式表中定义的样式。
- 字符象征。如果 `char` 是在已载入的字符集中定义了的字符名，那么使用 `\char` 命令会插入该字符。
- 斜体模式。有底线的文本会被用斜体来显示。
- 数学模式：美元符之间的文本是用数学模式存放的：所有字母和数字组成的串是斜体的，但是其他的字符（像标点符号和运算符号）是不变的。
- 新建命令。`\new{column}` 和 `\new{line}` 分别产生一个新的列和行。`\new{line}` 等价于 `//`。
- 样式命令。`\format<test>` 串把文本放在由样式化命令 `format` 制定的样式化文档中。正当的样式化命令是 `sub-`、非常脚本 (`super-scripts`)、粗斜体罗马字符的 `sub` 和 `super`。
- 交叉引用。`\tag{t}` 命令会用 `t` 作为名字标记一个段落，并且可以通过 `t` 引用这个段落。`\label{l}` 命令可以使标记串 `l` 和段落连接起来。`\cite{t}` 命令可以用标记 `t` 而对段落产生交叉引用。如果一个段落被 `\label` 准确地标记，这个标记可以用作交叉引用，不然就用为该段落产生的编号串。

18.10.5 样式表格式

一个样式表是由一系列行组成的，每一个用来指定一个样式的工具 (`properties`)。第一个属性命名为样式本身。并来的工具可能包含以下中的任何一个：

- *next*。显示由于缺省，哪种样式跟在段落样式之后。
- *counter*。如果显示一种问题的段落无法自动编号。属性值就会既显示计数器的内核值，又会显示该样式的计数：阿拉伯计数的 `0, 1, 2……` 或者字母计数的 `a, b……` 或者 `A, B……` 例如 `<counterB>` 说明计数器应该是 B 以上的字母，也就是 `B, C……`
- *跟踪* (`trailer`)。在编号串后面并在段落文本之前插入的源文件。可能包括像特殊字符，新建列等命令。

- *前导* (leader) 在编号串之前也在段落文本之前插入的源文件。可能包括特殊字符和新建列等命令。
- *分离器* (separator)。紧跟在样式计数器后面，在子样式段落的编号串中插入的源文件。例如可以用于在计数器间插入点。
- *双亲* (parent)。为了自动编号，样式必须按层次来组织。样式按照编号系列组织；每个系列有一个根样式，并有一个子样式的链。通过给每一个编号过的样式一个父亲，除了没有父亲的根节点外，编号系列单独显示。

例如，为了给章节 `chapter`、片段 `section` 和部分 `subsection` 用标准的方法编号，我们必须把 `chapter` 设为 `section` 的父亲(通过给定它的属性列表中的父亲属性)，并把 `section` 设为 `subsection` 的父亲。

以下是一个为 sections 1, 2……和 subsections 1.1,1.2……编号的完整样式文件的例子；通过一个 `tab` 从它们的段落中分离出编号；并且在该样式点的段落之前设置一个中心点：

```
<style:section><next:noindent><counter:1><separator:.><trailer: >
<style:subsection><next:noindent><parent:section><counter:1><separator:.><trailer: >
<style:point><next:body><leader:\periodcentered >
```

18.10.6 字符集样式

每一行字符集文件都有这样的样式：

```
<char:myname><font:myfont><index:myindex>
```

表示以 `myfont` 为字体的字符象征 `myname` 在索引 `myindex` 处。如果该字体是标准字体的话，字体属性可以忽略。