LaTeX3 入门

Eureka

目录

1 LaTeX3 的相关主要组成

- 1. l3kernel: 包含 expl3 的各个部分
- 2. l3packages：提供设计层和文本标记层接口
  - 2.1 l3keys2e
  - 2.2 xfp
  - 2.3 xfrac
  - 2.4 xparse
  - 2.5 xtemplate
- 3. l3experimental: 一些实验性的尝试，构建接口 (不如上一个稳定)
- 4. l3backend: 提供与后端（底层驱动）的交互代码。处理颜色，绘图，PDF。目前支持的驱动
  - 4.1 dvipdffmx
  - 4.2 dvips
  - 4.3 dvisvgm
  - 4.4 xdvipdfmx
  - 4.5 PDF 模式 (pdflatex 和 LuaTeX)
- 5. l3build:LaTeX3 的构建系统

2 LaTeX3 的尝试

2.1 基础知识铺垫

函数与变量: 在 LaTeX3 中函数和变量都是以 \ 开头 **函数**

- 1. 函数可以吃掉一些参数，并进行相关的操作
- 2. 函数要么是可以被展开的，要么就是可以被执行的

变量

- 变量可以用来存储数据

定义的方法

2.2 ReMark: 什么是 token?

最开始我也挺奇怪的，为什么有这个 token 这种说法。因为我取网上查看了这个单词的翻译然而网上最合理的翻译就是：象征，标志，记号。是不是感觉这个翻译听了都头疼，感觉看了更加的不知道 token 是一个啥玩意了？下面我们就展示 D E K 在它的 tex 源码中给的定义：

`A TEX token is either a character or a control sequence ...`

也就是说 token 其实就是一个单独的**字符**，（a, A, #, 1, ^, &, ...）; 又或者是一个单独的控制序列（以 \开头的命令）

下面再来看一下文档中对于 **token list** 的定义：

`A token list is a singly linked list of one-word nodes in mem, where each word contains a token and a link. Macro definitions, output-routine definitions, marks, \write texts, and a few other things are remembered by TEX in the form of token lists, usually preceded by a node with a reference count in its token ref count field. The token stored in location p is called info(p).`

不严谨的说，我们就可以直接的把 token list 看作由 {} 包裹的一系列的 token 的集合

其实这个 token 真的不是那么好直译的，但是项子越老师把它译作了**凭据**，对应的 token list 就是**凭据表**。感觉这个翻译还是蛮好的。

2.3 基本数据类型

命令定义格式:

`\<module>_<description>:<arg-spec>`

- 1. module: 模块名
- 2. description：描述
- 3. arg-spec：指定的**参数类型**
  - 3.1 n: 接收一个凭据表【一组被 {} 包围的记号】
  - 3.2 x: 与 n 类似, 但是对凭据表内的内容进行递归展开。
  - 3.3 N: 接收一个命令，传递命令本身
  - 3.4 V: 与 N 类似，但是传递命令的值
  - 3.5 p: 原始的 TEX 的形参指定 (就是原来 LaTeX2 ε 中使用 \ def 时的 #1, #2)
  - 3.4 T/F:n 的特殊情形，提供条件判断分支语句

注意

n/x 与 N/V 的区别涉及到宏展开的细节问题，后面会讨论。

开启 LaTeX3 编程接口

- 1. 在`\ExplSyntaxOn ** \ExplSyntaxOff`中间的即可以使用 LaTeX3 语法
- 2. 由于硬空格, 下划线~,_和冒号:被转义了。
- 3. 可以分别使用`\~`,`\c_math_subscript_token`和`\c_colon_str`来替代

2.4 变量

变量 (整形和浮点型用法相同，改为 fp_new:N 即可)
声明一个整形变量 (var): `\int_new:N \var`
变量 var 的赋值: `\int_set:Nn \var {2}`
使用变量:2 `\int_use:N \var`
输出值:2000 `\int_eval:n {\var*1000}`
注：`_new`: 声明一个变量 `_set`: 变量的赋值
`_use`: 使用变量的值 `_eval`: 计算表达式的值

2.5 函数

函数的是以一个命令：`\cs_set:Npn`来声明的, 下面就是一个示例：

```
\cs_set:Npn \my_function:nn #1#2 {
    两个数的和为:\int_eval:n {#1 + #2}
}
函数的调用
\my_function:nn {1}{2}
```

定义的解释说明

在这里我定义了一个用于计算两个整数和的函数 my_function，它接受两个整数，然后输出这两个整数的和。其中蓝色部分各个参数的对应关系如下：

N → `\my_function:nn`
p → `#1#2`
n → `{**}`(后面大括号中的全部内容【凭据表】)
两个数的和为:3

Remark

同一个函数根据**参数类型**的不同，可以产生不同的**变体**。具体的每一函数可以接受的参数类型可以参见文档：interface3, 只需要使用如下的命令即可：

```
texdoc interface3
```

2.6 变量和函数的定义和使用方式的反思

个人的感觉是，由于在 LaTeX3 中没有类似其他语言中**类型系统**的概念，就导致你声明的所有变量本质上都是一个 token。所以你在使用这个变量的时候就必须要说明你的变量的类型。对于变量的操作也一样，例如你要给变量赋值，你就必须说明你这个变量的类型还必须使用与之相匹配的赋值函数 (int_set, fp_set)。函数也类似，不同的是，你在使用函数的时候必须要声明传入的参数的 “类型”（n，x，N，V 等），然后一个类型对应一个实际的输入参数，一个不能

多，一个也不能少。所以就有了命名规范这么一说，就是说把你的变量的作用域，类型，变量名全部写入变量名中。

LATEX3 命名法提倡将函数的来源以及参数类型、变量的类型编码到其名字内。

- 可以更方便地让用户区别命令与变量
- 可以避免不同宏包之间命令的冲突
- LATEX 并不拥有真正的类型系统，这样的命名方式可以让用户在编程时自行检查错误
- 只是一套指导意见，并不是强制性的要求

2.7 自带的库函数

先定义几个变量:

```
\int_new:N \var_a
\int_new:N \var_b
\int_set:Nn \var_a {5}
```

取模函数

```
\int_set:Nn \var_b {\int_mod:nn {\var_a}{3}}
```

\vara_b=2

最值函数

$\max\{var\_a, var\_b\}$=2 `\int_min:nn {\var_a} {\var_b}`

$\min\{var\_a, var\_b\}$=5 `\int_max:nn {\var_a} {\var_b}`

自增自减函数

a 自增的结果:$a++$=6 `\int_incr:N \var_a \int_use:N \var_a`

a 自减的结果:$a--$=5 `\int_decr:N \var_a \int_use:N \var_a`

随机数函数

产生 $1-n$ 随机数：1 `\int_rand:n {20}`

产生 $m-n$ 随机数：65 `\int_rand:n {34}{89}`

注意：`\int_eval`仅支持 $+, -, \times, \backslash$, 但是`\fp_eval:n`还支持三目运算符

使用样例:

20 `\fp_eval:n {1+2>4?10:20}`

数学函数

$\exp(), \sin, \cos, acos, atand, \cdots$

计算 e^{π}=23.14069263277926 `\fp_eval:n {exp(\var_c)}`

计算 arcsin($\frac{\pi}{10}$)=1.251225373487637 `\fp_eval:n {acos(\var_c/10)}`

2.8 字符串

创建字符串： `\str_new:N \str_test`
字符串初始化： `\str_set:Nn \str_test {"你好Hello!@#&*LaTeX}`
显示字符串: ” 你好 Hello!@t##&*LaTeX `\str_test`

字符串操作函数
这部分内容在左边” 你好 Hello!@t##&*LaTeX
索引操作 (和 Python 类似)
str_test[2]= 部 `\str_item:Nn \str_test {2}`
str_test[2,-3]=LaT `\str_range:Nnn \str_test {2}{-3}`

2.9 序列

序列 (类似数组)：有 tl(tokenlist),clist(Commaseparetelists),seq(推荐使用)
1. 声明 (创建一个空的序列 lis): `\seq_new:N \lis`
2. 使用 set 创建初始化序列: `\seq_set_split:Nnn \lis_b {,} {a, {ef}, 你好}`
3. 显示序列:lis_b=aef 你好 `\seq_use:Nn \lis`
4. 访问特定元素:*lis_b*[2]=ef `\seq_item:Nn \lis_b {1}`

3 流程控制

3.1 if 语句

if 判断语句本质上是判断布尔表达式的真假，在 LaTeX3 中我们使用将 bool 表达式转化为布尔值的 Predicate 函数. 在 LaTeX3 中一般的数据类型都实现了各自的 Predicate，甚至是自己的条件判断函数。我们在这里仅仅只有是用到比较灵活的 api

-bool_if:nTF+Predicates

调用格式：

```
\bool_if:nTF{<bolean expression>} {<true code>} {<false code>}
```

一个具体的例子
LATEX3

3.2 for 循环

调用格式：

```
\int_step_function:nN {<integer>} {\do_somthing}
```

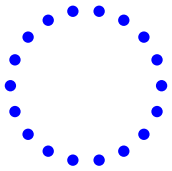
使用样例：

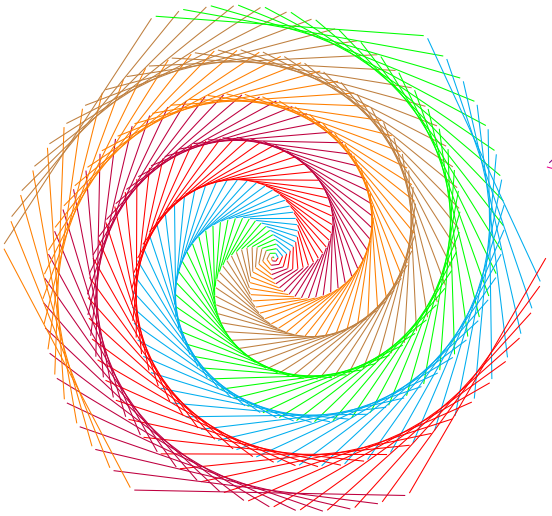
This is 10
This is 12
This is 14
This is 16
This is 18
This is 20

```
\int_step_inline:nnnn {10}{2}{20}{
    {\par This is #1}
}
```

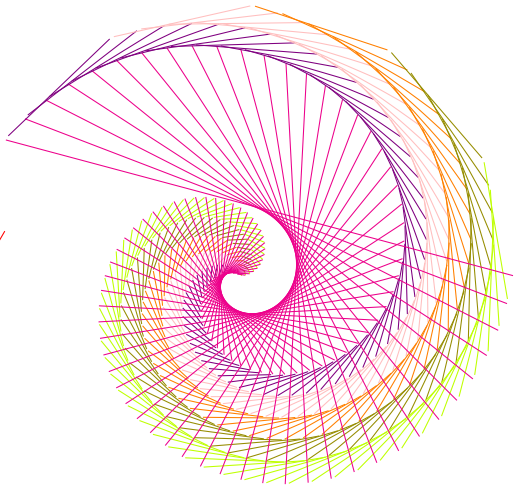
4 一些绘图例子

下面这个绘图例子就充分的发挥了 latex3 的优势：循环判断。尽管原始的 tikz 中已经了相关的宏能够实现类似的功能，但是实现起来绝对没有 latex3 优雅。





(a) Rotate 60°



(b) Rotate 30°

5 最后反思

目前感觉 latex 3 最大的用处就是能够大幅度的简化我们绘制 tikz 的难度，要是叫我在正文中使用 latex 3，我觉得并不是那么的方便。