# 如何匹配输入

当生成的扫描程序运行时，它会分析其输入，寻找匹配其任何模式的字符串。如果它找到多个匹配，它将接受匹配最多文本的匹配（对于尾随上下文规则，这包括尾随部分的长度，即使它将返回到输入）。如果找到两个或多个相同长度的匹配项，则将选择在flex输入文件中首先列出的规则。

一旦确定了匹配，与匹配对应的文本（称为标记）将在全局字符指针yytext中提供，其长度将在全局整数yyleng中提供。然后执行与匹配模式对应的操作（参见第8章[操作]，第15页），然后扫描剩余的输入以获得另一个匹配。

如果没有找到匹配项，则执行默认规则：输入中的下一个字符被认为是匹配的，并复制到标准输出中。因此，最简单的有效的flex输入是：

%%

它生成一个扫描仪，简单地将其输入（一次一个字符）复制到其输出中。

请注意，yytext可以用两种不同的方式来定义：要么是字符指针，要么是字符数组。通过在flex输入的第一个（定义）部分中包含一个特殊指令%指针或%数组，您可以控制flex使用的定义。默认值是%指针，除非您使用‘-l’ lex兼容性选项，在这种情况下，yytext将是一个数组。使用%指针的优点是，在匹配非常大的标记时，可以更快地扫描和没有缓冲区溢出（除非您耗尽了动态内存）。缺点是，您的操作如何修改yytext（参见第8章[操作]，第15页），并且调用unput（）函数会破坏yytext的当前内容，当在不同的词汇版本之间移动时，这可能是一个相当大的移植问题。

%数组的优点是，您可以修改yytext的心脏内容，调用取消放置（）不会破坏yytext（参见第8章[操作]，第15页）。此外，现有的词汇程序有时会使用表单的声明从外部访问yytext：

extern字符

与%指针一起使用时，此定义是错误，但对%数组是正确的。

%数组声明将yytext定义为一个YYLMAX字符数组，默认值为一个相当大的值。您可以通过简单地#定义YYLMAX到灵活输入的第一部分中不同的值来更改大小。如上所述，使用%指针，yytext可以动态增长以适应大型令牌。虽然这意味着您的%指针扫描器可以容纳非常大的令牌（例如匹配整个注释块），但请记住，每次扫描器必须调整yytext的大小时，它还必须从一开始就重新扫描整个令牌，因此匹配这样的令牌可能会很慢。如果调用取消（）导致文本，yytext目前不会动态增长；相反，会导致运行时错误。

还要注意，您不能使用%array与C++扫描程序类（参见第18章[Cxx]，第50页）。

# 行动

规则中的每个模式都有一个相应的操作，它可以是任意的C语句。该模式以第一个未转义的空白字符结束；该行的其余部分是它的操作。如果操作为空，那么当模式匹配时，将简单地丢弃输入标记。例如，这里是一个程序的规范，它从其输入中删除所有出现的“zapme”：

%%

“扎我”

此示例将把输入中的所有其他字符复制到输出中，因为它们将由默认规则进行匹配。

这里是一个程序，它压缩多个空白和标签到一个空白，并抛出在一行结尾的空格：

%%

[ \t]+ putchar ( ' ' );

[ \t]+$ /\*将忽略此令牌\*/

如果操作包含“{”，则操作将持续到找到平衡“}”，并且操作可能跨越多行。flex知道C字符串和注释，不会被它们中的括号愚弄，但还允许操作以“%{”开头，并将操作是到下一个“%}”的所有文本（不管操作中的普通括号）。

仅由竖条（‘|’）组成的动作的意思是“与下一个规则的动作相同”。下面所图。

操作可以包括任意C代码，包括返回值到任何称为yylex（）的例程的返回语句。每次调用yylex（）时，它都会从它最后一次停止的位置开始继续处理令牌，直到它到达文件的末尾或执行一个返回。

操作可以自由地修改yytext，除了延长它（在其结尾添加字符-这些将覆盖输入流中稍后的字符）。但是，这在使用%数组时不适用（请参见第7章[匹配]，第14页）。在这种情况下，yytext可以以任何方式自由修改。

操作可以自由地修改叶伦，但如果操作还包括使用yymore（）（参见下文），则不应这样做。

在一个动作中可以包含许多特殊的指令：

回波 将yytext复制到扫描仪的输出中。

启动后跟启动条件的名称，将扫描仪置于相应的启动条件中（请参见下文）。

拒绝指示扫描仪继续执行与输入（或输入的前缀）相匹配的“第二最佳”规则。规则的选择如上文第7章[匹配]，第14页，以及yytext和yyleng的设置适当。它可以是匹配了与最初选择的规则一样多的文本，但稍后就会出现在flex输入文件中，也可以是匹配了较少的文本。为了

例如，下面的方法将计算输入中的单词，并在看到“frob”时调用例程的特殊（）：

int word\_count = 0；

%%

frob 特殊（）；拒绝；[^ \t\n]+ ++word\_计数；

如果没有拒绝，输入中出现的任何“frob”都不会被视为单词，因为扫描器通常对每个令牌只执行一个操作。允许多次使用拒绝，每个使用都可以找到当前活动规则的下一个最佳选择。例如，当以下扫描仪扫描令牌“abcd”时，它将把“abcdabcaba”写入输出：

%%

a |

ab |

abc |

abcd 回声；拒绝；

.|\n /\*吃掉了任何无与伦比的角色\*/

前三个规则与第四个规则的动作相同，因为它们使用了特殊的“|”动作。

就扫描仪性能而言，拒绝是一个特别昂贵的特性；如果它在该扫描仪的任何操作中使用，它将减慢该扫描仪的所有匹配。此外，拒绝不能与“-Cf”或“-CF”选项一起使用（参见第16章“扫描器选项”，第35页）。

还请注意，与其他特殊操作不同，拒绝是一个分支。在操作中，紧随其后的代码将不会被执行。

yymore（）告诉扫描器，下次它匹配一个规则时，相应的标记应该附加到yytext的当前值上，而不是替换它。例如，给定输入的“超级项链”，下面将写下“超级项链”到输出：

%%

巨大的 ECHO；yymore（）；螺丝钉 回声；

第一个“mega-”被匹配并响应到输出中。然后“项链”是匹配的，但是前面的“mega-”仍然停留在yytext的开头，所以“荷运”规则的回声实际上会写成“超级荷运”。

关于使用yymore（）的两个说明。首先，yymore（）依赖于（）的值，从而正确反映当前令牌的大小，所以如果您使用yymore（），则不能修改（）。其次，在扫描器的动作中存在yymore（）会对扫描器的匹配速度造成轻微的性能损失。

yyless (n)将当前令牌的前n个字符以外的所有字符返回到输入流，当扫描器寻找下一个匹配时，这些字符将被重新扫描。yytext和yyleng被适当地调整（例如，yyleng现在将等于n）。例如，在输入的“脚”上，下面会写出“脚”：

%%

脚栏 ECHO；yyless (3)；[a-z]+ 回声；

0到yy（）的参数将导致再次扫描整个当前输入字符串。除非您改变了扫描仪随后处理输入的方式（例如使用begin），否则这将导致一个无尽的循环。

注意，无（）是一个宏，只能在flex输入文件中使用，不能从其他源文件中使用。

unput (c)将字符c放回输入流中。它将是下一个被扫描的字符。以下操作将使用当前令牌，并将其重新扫描，并括在括号中。

{

int i；

/\*复制文本，因为未输入（）丢弃=文本（文本）；

unput ( ')' );

为（i = yyleng - 1；i >= 0；--i）未输入（yycopy[i]）；

未输入（（）；免费（副本）

}

注意，由于每个未输入的（）都将给定的字符放回输入流的开头，因此向后推字符串必须前后进行。

在使用unput（）时，一个重要的潜在问题是，如果您使用%指针（默认值），调用unput（）会破坏yytext的内容，从它最右边的字符开始，每次调用都吞噬左边的一个字符。如果您需要在调用取消输入（）后保留yytext的值（如上面的示例），您必须首先将它复制到其他地方，或者使用%数组构建扫描仪（参见第7章[匹配]，第14页）。

最后，请注意，您不能返回“EOF”来尝试用文件结尾标记输入流。

输入（）从输入流中读取下一个字符。例如，以下是吃掉C评论的一种方法：

%%

"/\*" {

int c;

为了

{

而（（c =输入（））！=‘\*”&& c ！= EOF）

; /\*吃掉了评论\*/的文本

如果（c == ‘\*’）

{

而（（c =输入（））==‘\*’）

;

如果（c == ‘/’）

破碎 /\*找到了末端的\*/

}

如果（c == EOF）

{

错误（“评论中的EOF”）；中断；

}

}

}

(注意，如果扫描仪是使用C++编译的，那么输入（）将被称为

yy输入（），以避免名称与C++流冲突。)

YY\_FLUSH\_BUFFER；刷新扫描器的内部缓冲区，以便当扫描器下次尝试匹配令牌时，它将首先使用YY\_INPUT（）重新填充缓冲区（参见第9章[生成的扫描器]，第19页）。此操作是更一般的yy\_flush\_缓冲区的特殊情况；函数，如下所述（参见第11章[多个输入缓冲区]，第27页）

可以使用y终止（）来代替操作中的返回语句。它终止扫描仪，并向扫描仪的调用者返回一个0，表示“全部完成”。默认情况下，当遇到文件结束时，也会调用yy终止（）。它是一个宏，可以被重新定义。

# 生成的扫描仪

flex的输出是文件lex.yy.c，其中包含扫描例程yylex（）、它用于匹配令牌的许多表，以及一些辅助例程和宏。默认情况下，yylex（）声明如下：

int yylex()

{

...这里的各种定义和操作...

}

（如果您的环境支持函数原型，那么它将是intyylex（void）。）通过定义YY\_DECL宏，可以更改此定义。例如，您可以使用：

#定义YY\_DECL浮动lexscan（ab）浮点ab；

要给扫描例程命名为lexscan，请返回一个浮点数，并取两个浮点数作为参数。请注意，如果您使用K&R类型/非原型函数声明为扫描例程提供参数，则必须使用分号（；）终止该定义。

flex在默认情况下生成“C99”函数定义。Flex过去能够生成过时的、“传统的”函数定义。这是为了支持在旧系统上引导gcc。不幸的是，传统的定义阻止我们使用任何小于int的标准数据类型（如短、char或bool）作为函数参数。此外，传统的定义支持增加了骨架文件中的额外复杂性。出于这个原因，当前版本的flex只生成标准的C99代码，而将K&r风格的功能留给了历史学家。

每当调用yylex（）时，它都会从全局输入文件yyin（默认为stdin）中扫描令牌。它一直持续到文件结束（此时返回值0）或执行返回语句。

如果扫描程序到达文件结尾，后续调用未定义，除非yyin指向新的输入文件（在这种情况下继续从该文件扫描），或者调用yy重启（）。（）接受一个参数，一个文件\*指针（如果您已经设置YY\_INPUT以从yyin以外的源扫描，则可以是NULL），并初始化yyin以便从该文件扫描。本质上，将yyin分配给一个新的输入文件或使用yy重启（）之间没有区别；后者可以与以前版本的flex兼容，而且因为它可以用于在扫描中间切换输入文件。它还可以用来丢弃当前的输入缓冲区，通过用一个参数yyin来调用它；但是最好使用YY\_FLUSH\_BUFFER（参见第8章[操作]，第15页）。请注意，yy重新启动（）不会将启动条件重置为初始条件（参见第10章[启动条件]，第21页）。

如果yylex（）由于在其中一个操作中执行返回语句而停止扫描，则可能会再次调用扫描器，它将在停止的地方恢复扫描。

默认情况下（为了提高效率），扫描器使用块读取而不是简单的getc（）调用来从yyin中读取字符。可以通过定义YY\_INPUT宏来控制它如何获取输入的性质。YY\_INPUT（）的调用序列是YY\_INPUT（buf，结果，max\_size）。它的操作是在字符数组buf中放置多达max\_size个字符，并在整数变量结果中返回的数量

读取或常量YY\_NULL（Unix系统上的0）表示“EOF”。默认值

YY\_INPUT从全局文件指针yyin中读取。

下面是YY\_INPUT的定义示例（在输入文件的定义部分）：

%{

#定义YY\_INPUT（buf，结果，max\_size）\

{ \

int c = getchar(); \

结果=（c == EOF）吗？YY\_NULL：（buf[0] = c，1）；\

}

%}

此定义将使输入处理更改为一次发生一个字符。

当扫描仪从YY输入接收到文件结束指示时，它会检查yywrap（）函数。如果yywrap（）返回false（零），则假定该函数已经继续运行，并设置了yyin以指向另一个输入文件，然后继续扫描。如果返回true（非零），则扫描程序终止，返回0给其调用者。注意，在任何一种情况下，开始条件都保持不变；它不会恢复到初始值。

如果您不提供您自己版本的yywrap（），那么您必须使用%选项（在这种情况下，扫描器的行为好像yywrap（）返回1），或者您必须链接“-lfl”以获得例程的默认版本，它总是返回1。

用于从内存中的缓冲区进行扫描(e。g.，请参见[扫描字符串]，第30页。请参见第11章[多个输入缓冲区]，第27页。

扫描器将其ECHO输出写入yyout全局值（默认值，stdout），用户可以仅仅通过将其分配给其他文件指针来重新定义它。

# 开始条件

flex提供了一种有条件地激活规则的机制。*任何模式以“<sc>”为前缀的规则只有在扫描器处于名为sc的启动条件时才会处于激活状态。*例如，

<STRING>[^"]\* { /\*吃掉了弦体\*/...

...

}

只有在扫描程序处于字符串启动条件时才会激活，并且

<初始，字符串，引用>\。 { /\*处理一个转义的\*/...

...

}

只有当当前启动条件为初始值、字符串或引号时，才会处于激活状态。

开始条件在输入的定义（第一个）部分中声明，使用以“%s”或“%x”开头的未缩进行，后面跟着一个名称列表。前者声明包含启动条件，后者声明独家启动条件。使用开始动作激活启动条件。在执行下一个开始操作之前，具有给定开始条件的规则将处于活动状态，而具有其他开始条件的规则将处于非活动状态。如果启动条件是包含的，则完全没有启动条件的规则也将处于激活状态。如果它是排他的，那么只有具有启动条件的规则才是活动的。一组基于同一排他启动条件的规则描述了一个独立于flex输入中的任何其他规则的扫描器。正因为如此，独家启动条件使得指定“迷你扫描器”变得很容易，它可以扫描在语法上与其他部分不同的输入部分（例如，注释）。

如果包含开始条件和排他开始条件之间的区别仍然有点模糊，这里有一个简单的例子来说明两者之间的联系。规则集：

%s示例

%%

<example>foo do\_some（）；酒吧 什么\_else（）；

相当于

%x示例

%%

<example>foo do\_some（）；

<INITIAL, example>bar 什么\_else（）；

如果没有<初始值，示例>限定符，那么在启动条件示例中，第二个示例中的条形模式将不会是活动的（即，无法匹配）。如果我们只是用

<示例>来限定条，那么它只在示例中是活动的，而不是初始状态，而在第一个示例中，它在两者中都是活动的，因为在第一个示例中，示例开始条件是一个包含（%s）开始条件。

还要注意，特殊的开始条件说明符<\*>匹配每个开始条件。

因此，上面的例子也可以写成：

%x示例

%%

<example>foo do\_some（）；

<\*>bar 什么\_else（）；

默认规则（对于ECHO任何不匹配的字符）在开始条件中保持活动状态。

相当于：

<\*>.|\n 回声；

begign(0)返回到原始状态，其中只有没有启动条件的规则处于活动状态。此状态也可以称为开始条件初始值，因此开始（初始值）等同于开始(0)。（起始条件名称周围的括号不是必需的，但被认为是好的样式。）

开始操作也可以在规则部分的开头作为缩进的代码给出。例如，每当调用yylex（）且全局变量enter\_stere时，以下操作将导致扫描器进入特殊的启动条件：

inten\_streat；

%x特殊

%%

如果（enter\_stecle）开始（特殊）；

<SPECIAL>blahblahblah

…更多的规则遵循。

为了说明开始条件的使用，这里有一个扫描仪，它提供了像“123.456”这样的两种不同的解释。默认情况下，它将把它视为三个标记，整数“123”，一个点(‘。‘)，以及整数“456”。但是，如果字符串前面是字符串“apede-浮动”，它将把它视为一个令牌，浮点数“123.456”：

%{

#include <math.h>

%}

%s期望

%%

预期浮子 开始（预计）；

<expect>[0-9]+.[0-9]+ {

printf（“找到一个浮点数，=%f\n”，atof（yytext））；

}

<expect>\n {

/\*，所以

* 我们需要另一个“期望数字”
* 在我们再认出来之前
* 算术

\*/开始（初始）；

}

[0-9]+ {

printf（“找到一个整数，=%d\n”，atoi（yytext））；

}

"." printf（“找到一个点\n”）；

这里是一个扫描器，它可以识别（并丢弃）C注释，同时保持当前输入行的计数。

%x评论

%%

因特line\_num = 1；

"/\*" 开始（评论）；

<comment>[^\*\n]\* /\*会吃任何不是‘\*’ \*/的东西

<comment>"\*"+[^\*/\n]\* /\*吃掉“\*不跟着”/的\*/

<comment>\n ++line\_num;

<comment>"\*"+"/" 开始（初始）；

这个扫描仪在与每个规则进行尽可能多的文本匹配方面有点麻烦。一般来说，当你试图编写一个高速扫描仪时，你要尽可能多地匹配每个规则，因为这是一个巨大的胜利。

请注意，起始条件名称实际上是整数值，可以这样存储。

因此，上述内容可以通过以下方式进行扩展：

%x评论foo

%%

int line\_num = 1；int评论\_调用者；

"/\*" {

评论\_调用者=的初始值；开始（评论）；

}

...

<foo>"/\*" {

评论\_调用者= foo；开始（评论）；

}

<comment>[^\*\n]\* /\*会吃任何不是‘\*’ \*/的东西

<comment>"\*"+[^\*/\n]\* /\*吃掉“\*不跟着”/的\*/

<comment>\n ++line\_num;

<comment>"\*"+"/" 开始（评论调用者）；

此外，您可以使用整数值YY\_启动宏访问当前启动条件。例如，可以编写上面对注释\_调用者的分配

评论调用者= YY\_START；

Flex提供了YYSTATE作为YY\_START的别名(因为这是AT&T所使用的

lex).

由于历史原因，启动条件在生成的扫描程序中没有它们自己的名称空间。在生成的扫描程序和生成的标头中，未修改起始条件名称。参见[选项标题]，第35页。参见[选项前缀]，第39页。

最后，这里有一个例子，如何使用独家开始条件匹配c风格的引用字符串，包括扩展的转义序列（但不包括检查一个太长的字符串）：

%x str

%%

字符符string\_buf[MAX\_STR\_CONST]；字符\*string\_buf\_ptr；

\" string\_buf\_ptr = string\_buf；开始（str）；

<str>\" { /\*看到结束报价-全部完成\*/begin（初始）；

\*string\_buf\_ptr = '\0';

/\*返回字符串常量标记类型和

\*解析器的值

\*/

}

<str>\n {

/\*错误-未终止的字符串常量\*/

/\*生成错误消息\*/

}

<str>\\[0-7]{1,3} {

/\*八进制转义序列\*/ int结果；

（无效）sscanf（yytext+1，“%o”，和结果）；

如果（结果> 0xff）

/\*误差，常数是出界的\*/

\*string\_buf\_ptr++ =结果；

}

<str>\\[0-9]+ {

/\*生成错误错误的转义序列；的东西

\*喜欢“\48”或“\0777777”

\*/

}

<str>\\n \*string\_buf\_ptr++ = '\n';

<str>\\t \*string\_buf\_ptr++ = '\t';

<str>\\r \*string\_buf\_ptr++ = '\r';

<str>\\b \*string\_buf\_ptr++ = '\b';

<str>\\f \*string\_buf\_ptr++ = '\f';

<str>\\(.|\n) \*string\_buf\_ptr++ = yytext[1];

<str>[^\\\n\"]+ {

字符\*yptr = yytext；

而（\*yptr）

\*string\_buf\_ptr++ = \*yptr++;

}

通常，例如在上面的一些例子中，您最终编写了一大堆规则，它们的前面都有相同的开始条件(s)。Flex通过引入开始条件范围的概念，使这一点更容易和更干净。启动条件范围从：

<SCs>{

其中，<SCs>是一个或多个启动条件的列表。在开始条件范围内，每个规则都会自动应用前缀<SCs>，直到与初始值“{”匹配的“}”。例如，

<ESC>{

"\\n" 返回“\n”；

"\\r" 返回“\r”；

"\\f" 返回“\f”；

"\\0" 返回“\0”；

}

相当于：

<ESC>"\\n" 返回“\n”；

<ESC>"\\r" 返回“\r”；

<ESC>"\\f" 返回“\f”；

<ESC>"\\0" 返回“\0”；

启动条件范围可以是嵌套的。

以下例程可用于操作启动条件的堆栈：

#### 空白yy\_push\_state（int new\_state）

函数

将当前的启动条件推到启动条件堆栈的顶部，然后

切换到new\_state，如同使用了起始new\_state（记住，起始条件名称也是整数）。

#### 空白yy\_pop\_state（）

弹出堆栈的顶部，并通过开始切换到它。

#### 因特yy\_top\_state（）

返回堆栈的顶部，而不改变堆栈的内容。

函数

启动条件堆栈会动态增长，因此没有内置的大小限制。如果内存耗尽，则程序执行将中止。

若要使用开始条件堆栈，您的扫描器必须包含一个%选项堆栈指令（请参见第16章[扫描器选项]，第35页）。

# 多个输入缓冲区

一些扫描仪（比如那些支持“包括”文件的扫描仪）需要从多个输入流中读取。由于flex扫描器做了大量的缓冲，因此我们不能通过简单地编写一个对扫描上下文敏感的YY\_INPUT（）来控制下一个输入的读取位置。YY\_INPUT（）只有在扫描器到达其缓冲区的末端时才被调用，这可能是在扫描诸如需要切换输入源的包含语句之后的很长一段时间。

为了协商这些问题，flex提供了一种在多个输入缓冲区之间创建和切换的机制。通过使用以下方式创建输入缓冲区：

YY\_BUFFER\_STATE yy\_create\_buffer（文件\*文件，int大小）

函数

它需要一个文件指针和一个大小，并创建一个与给定文件关联的缓冲区

并且大到足以容纳大小字符（如果有疑问，请使用YY\_BUF\_SIZE的大小）。它返回一个YY\_BUFFER\_STATE句柄，然后可以将其传递给其他例程（见下文）。YY\_BUFFER\_STATE类型是指向不透明结构体YY\_BUFFER\_STATE结构的指针，因此您可以安全地将YY\_BUFFER\_STATE变量初始化为(（YY\_BUFFER\_STATE）

如果您愿意，0)，也引用不透明结构，以便正确地声明源文件以外的扫描仪文件中的输入缓冲区。请注意，调用yy\_create\_buffer中的FILE指针只用作YY\_INPUT所看到的yyin的值。如果您重新定义YY\_INPUT（），使它不再使用yyin，那么您可以安全地传递一个空文件指针到yy\_create\_buffer。选择要扫描的特定缓冲区：

void yy\_switch\_to\_buffer（YY缓冲区状态新缓冲区）

函数

上面的功能会切换扫描仪的输入缓冲区，这样后续的令牌就会出现

从new\_缓冲区。请注意，yywrap（）可以使用yy\_switch\_to\_buffer（）来设置东西以便继续扫描，而不是打开一个新文件并指向它。如果您正在寻找输入缓冲区堆栈，那么您希望使用yypush\_buffer\_state（）而不是这个函数。还要注意，通过yy\_switch\_to\_缓冲区（）或yywrap（）切换输入源并不会改变开始条件。

void yy\_delete\_buffer（YY缓冲区状态缓冲区）

函数

用于回收与缓冲区关联的存储器。(缓冲区可以为NULL，其中

例程序什么都不做。)您还可以使用以下方式清除缓冲区的当前内容：

void yypush\_buffer\_state（YY缓冲区状态缓冲区）

函数

此函数将新的缓冲区状态推到一个内部堆栈上。被推状态

成为新的当前状态。该堆栈由flex进行维护，并将根据需要进行增长。当您想要更改状态时，但要保留当前状态，以供以后使用。

#### 空白yypop\_buffer\_state（）

函数

此函数从堆栈的顶部删除当前状态，并通过调用yy\_delete\_buffer来删除它。堆栈上的下一个状态，如果有，将成为新的当前状态。

void yy\_flush\_buffer（YY缓冲区状态缓冲区）

函数

此函数丢弃缓冲区的内容，因此下次扫描仪尝试

匹配缓冲区中的令牌，它将首先使用YY\_INPUT（）重新填充缓冲区。

YY\_BUFFER\_STATE yy\_new\_buffer（文件\*文件，int大小）

函数

是yy\_create\_buffer（）的别名，提供兼容C++使用new

并进行删除，以创建和销毁动态对象。

YY\_CURRENT\_BUFFER宏向当前缓冲区返回一个YY\_BUFFER\_STATE句柄。它不应该被用作左值。

下面是两个使用这些特性来编写扩展包括文件的扫描仪的例子（下面将讨论<<EOF>>特性）。

这个第一个示例使用yypush缓冲区状态和yypop缓冲区状态。Flex在内部维护堆栈。

/\*中的“incl”状态用于获取名称

一个包含文件的\*

\*/

%x包括

%%

包括 开始（包括）；

[a-z]+ 回声；

[^a-z\n]\*\n? 回声；

<incl>[ \t]\* /\*吃了空白的\*/

<incl>[^ \t\n]+ { /\*得到了包含文件名\*/ yyin = fopen（yytext，“r”）；

如果yyin )

错误...

yypush\_buffer\_state（yy\_create\_buffer（yyin，YY\_BUF\_SIZE））；

开始（初始）；

}

<<EOF>> {

yypop\_buffer\_state() ;

如果yy\_current\_buffer )

{

yyterminate();

}

}

下面的第二个示例执行了与前面的示例相同的事情，但是手动管理自己的输入缓冲区堆栈（而不是让flex这样做）。

/\*中的“incl”状态用于获取名称

一个包含文件的\*

\*/

%x包括

%{

#定义MAX\_INCLUDE\_DEPTH 10

YY\_BUFFER\_STATE包括\_stack[MAX\_INCLUDE\_DEPTH]；int include\_stack\_ptr = 0；

%}

%%

包括 开始（包括）；

[a-z]+ 回声；

[^a-z\n]\*\n? 回声；

<incl>[ \t]\* /\*吃了空白的\*/

<incl>[^ \t\n]+ { /\*获得了包含文件名\*/ if（include\_stack\_ptr >= MAX\_INCLUDE\_DEPTH）

{

fprintr，“包括嵌套太深”）；出口（1）；

}

包括\_stack[include\_stack\_ptr++]=YY\_CURRENT\_BUFFER；

yyin = fopen（yytext，“r”）；如果(！yyin )

错误...

yy\_switch\_to\_buffer (

yy\_create\_buffer（yyin，YY\_BUF\_SIZE）)；

开始（初始）；

}

<<EOF>> {

如果（——include\_stack\_ptr==0）

{

yyterminate();

}

其他的

{

yy\_delete\_buffer ( YY\_CURRENT\_BUFFER ); yy\_switch\_to\_buffer (

incle\_stack[include\_stack\_ptr])；

}

}

以下例程可用于设置输入缓冲区以扫描内存字符串，而不是文件。它们都创建一个新的输入缓冲区来扫描字符串，并返回相应的YY\_BUFFER\_STATE句柄（完成它后，应该使用yy\_delete\_缓冲区（）删除）。它们还使用yy\_switch\_to\_缓冲区（）切换到新的缓冲区，因此对yylex（）的下一次调用将开始扫描该字符串。

YY\_BUFFER\_STATE yy\_scan\_string（常量）

扫描以NUL结尾的字符串。

YY\_BUFFER\_STATE yy\_scan\_bytes（常字符\*字节，内部）

从位置字节开始扫描len字节（可能包括NUL）。

函数

请注意，这两个函数都会创建并扫描字符串或字节的副本。（这可能是可取的，因为yylex（）修改它正在扫描的缓冲区的内容。）您可以使用以下方式来避免该副本：

YY\_BUFFER\_STATE yy\_scan\_buffer（字符\*基数，yy大小，t大小）

函数

哪个扫描从基开始的缓冲区，包括大小字节，最后一个

其中的两个字节必须是YY\_END\_OF\_BUFFER\_CHAR（ASCII NUL）。这最后两个字节不被扫描；因此，扫描包括基本[0]到基本[size-2]，包括。

如果您未能以这种方式设置基（即，忘记最后两个YY\_END\_OF\_BUFFER\_ CHAR字节），那么yy\_scan\_buffer（）将返回一个NULL指针，而不是创建一个新的输入缓冲区。

#### yy\_size\_t

数据类型

是一个积分类型，您可以转换反映缓冲区大小的整数表达式。

# 文件结束规则

特殊规则<<EOF>>指示当遇到文件结束时要执行的操作，yywrap（）返回非零（即，表示没有其他需要处理的文件）。该操作必须通过执行以下操作之一来完成：

* + 将yyin分配给一个新的输入文件(在以前版本的flex中，在完成分配后，您必须调用特殊操作YY\_NEW\_FILE。这已经没有必要了。)
  + 执行返回语句；
  + 执行特殊的y终止（）操作。
  + 或者，使用yy\_switch\_to\_buffer（）切换到如上面的示例中所示的新的缓冲区。

<<EOF>>规则不能与其他模式一起使用；它们只能通过启动条件列表进行限定。如果给出了一个不合格的<<EOF>>规则，则它适用于所有还没有具有<<EOF>>操作的启动条件。若要仅为初始启动条件指定<<EOF>>规则，请使用：

<INITIAL><<EOF>>

这些规则对于捕获诸如未封闭的评论等东西很有用。示例：

%x报价

%%

…处理报价的其他规则。

<quote><<EOF>> {

错误（“未终止引用”）；y终止（）；

}

<<EOF>> {

如果（\*++文件列表）

yyin=fopen（\*文件列表，“r”）；否则

yyterminate();

}

# 杂项宏

可以定义宏YY\_USER\_ACTION，以提供一个总是在匹配规则的操作之前执行的操作。例如，可以#defen‘d调用例程将yytext转换为小写。当调用YY\_USER\_ACTION时，变量yy\_act给出了匹配规则的数量（规则以1开始编号）。假设您要描述每个规则匹配的频率。下面的方法可以达到这个目的：

#定义YY\_USER\_ACTION ++ctr[yy\_act]

其中ctr是一个保存不同规则的计数。请注意，宏YY\_NUM\_RULES给出了规则的总数（包括默认规则），即使您使用‘-s)’，因此对ctr的正确声明是：

int ctr[YY\_NUM\_RULES]；

宏YY\_USER\_INIT可以被定义为提供一个总是在第一次扫描之前（以及在扫描器的内部初始化完成之前）执行的操作。例如，它可以用于调用一个例程来读取一个数据表或打开一个日志记录文件。

宏yy\_set\_interactive（is\_交互式的）可以用来控制当前的缓冲区是否被认为是交互式的。交互式缓冲区的处理速度较慢，但必须在扫描器的输入源确实是交互式时使用，以避免等待填充缓冲区导致的问题（参见第16章[扫描器选项]第35页中关于“-I”标志的讨论）。宏调用中的非零值将缓冲区标记为交互式，将零值标记为非交互式。请注意，使用此宏将覆盖始终交互式的%选项或从不交互式的%选项（请参见第16章[扫描器选项]，第35页）。在开始扫描是（或不是）交互式的缓冲区之前，必须调用yy\_set\_interactive（）。

宏yy\_set\_bol（at\_bol）可用于控制当前缓冲区的下一个令牌匹配的扫描上下文是否像在行的开始处一样完成。非零宏参数使用‘^’锚定的规则处于活动状态，而零参数使‘^’规则处于非活动状态。

如果从当前缓冲区扫描的下一个令牌将具有‘^’规则，则宏YY\_AT\_BOL（）返回true，否则返回false。

在生成的扫描程序中，所有操作都被收集在一个大型开关语句中，并使用YY\_BREAK进行分离，这可以重新定义。默认情况下，它只是一个中断，即将每个规则的操作与以下规则的操作分开。重新定义YY\_BREAK允许，例如，C++用户#定义YY中断，什么都不做（同时非常小心，每个规则都以中断或返回结束！）为了避免因为规则的操作以返回结束而遭受不可访问的语句警告，因此无法访问YY\_BREAK。

# 对用户可用的值

本章总结了规则操作中用户可用的各种值。

字符\*yy文本

保存当前令牌的文本。它可以被修改，但不会延长（您不能在结尾附加字符）。

如果特殊的指令%数组出现在扫描器描述的第一部分中，那么yytext将被声明为char yytext[YYLMAX]，其中YYLMAX是一个宏定义，如果您不喜欢默认值（通常为8KB），您可以在第一部分中重新定义。使用%数组会导致较慢的扫描程序，但是yytext的值对调用取消输入（）无效，当yytext是一个字符指针时，这可能会破坏它的值。%数组的对立面是%指针，这是默认值。

在生成C++扫描程序类时不能使用%array（‘-+’标志）。

Intyleng文件\*yyyin

保存当前令牌的长度。

是默认情况下flex从中读取的文件。它可以被重新定义，但这样做只有在扫描开始之前或遇到EOF之后才有意义。在扫描过程中更改它会产生意外的结果，因为flex缓冲其输入；使用yy重启（）代替。一旦因为看到文件结束而导致扫描终止，您可以在新的输入文件上分配yyin，然后再次调用扫描仪以继续扫描。

voidyy重新启动（FILE \*new\_file）

可能被调用来指向yyin的新输入文件。切换到新文件是立即的（任何以前缓冲的输入都将丢失）。请注意，使用yyin作为参数调用yy重启（），因此会抛出当前的输入缓冲区，并继续扫描相同的输入文件。

文件\*yyout

是要执行ECHO操作的文件。用户可以重新分配它。

yy\_current\_buffer

向当前缓冲区返回一个YY\_BUFFER\_STATE句柄。

yy\_start 返回与当前开始条件对应的整数值。随后可以使用此值返回该开始条件。

# 与Yacc接口

flex的主要用途之一是作为yacc解析器生成器的同伴。yacc解析器希望调用一个名为yylex（）的例程来查找下一个输入令牌。例程应该返回下一个令牌的类型，并将任何关联值放在全局岁中。要将flex与yacc一起使用，需要为yacc指定‘-d’选项来指示它生成文件y.tab.h，该文件包含在yacc输入中出现的所有%标记的定义。然后，将将此文件包含在flex扫描仪中。例如，如果其中一个令牌是TOK\_NUMBER，那么扫描仪的一部分可能看起来像：

%{

#include "y.tab.h"

%}

%%

[0-9]+ yylval = atoi（yytext）；返回TOK\_NUMBER；

# 扫描仪选项

各种flex选项按以下菜单中的功能进行分类。如果您想按名称查找一个特定的选项，请参见[扫描器选项的索引]，第132页。

尽管有许多扫描仪选项，一个典型的扫描仪可能只指定以下选项：

%option 8位重入野牛桥

%option 警告no默认

%option yylineno

%option 输出文件=“扫描器。c”头文件=“扫描器.h”

第一行指定了我们想要的扫描器的一般类型。第二行说明了我们要小心一点。第三行要求flex跟踪行号。最后一行告诉flex如何指定这些文件的名称。(这些选项可以按任何顺序进行指定。我们只是把它们分开了。)

flex还提供了一种机制，用于控制扫描器规范本身内部的选项，而不是从flex命令行中控制选项。这是通过在扫描仪规范的第一部分中包含%选项指令来实现的。您可以用一个选项来指定多个选项

%选项指令，以及在flex输入文件的第一部分中包含的多个指令。

大多数选项都是简单的名称，也可以在前面加上单词“no”（中间没有空格）来否定它们的含义。这些名字与它们的多头期权相同（但没有领先的“--”）。

flex扫描规则操作，以确定是使用拒绝还是更多（）功能。拒绝选项和yymore选项可以替代是否使用选项，通过设置选项（例如，%选项拒绝）来指示功能确实已使用，或取消设置选项来指示实际未使用（例如，%选项）。

对于希望抑制生成扫描仪中不需要的例程的棉纯主义者来说，有许多选项。如果未设置（例如，%选项nounput），将导致相应的例行程序没有出现在生成的扫描仪中：

输入，未输出

yy\_push\_state , yy\_pop\_state , yy\_top\_state yy\_scan\_buffer , yy\_scan\_bytes , yy\_scan\_string

yyget\_extra, yyset\_extra, yyget\_leng, yyget\_text, yyget\_lineno, yyset\_lineno, yyget\_in, yyset\_in, yyget\_out, yyset\_out, yyget\_lval, yyset\_lval, yyget\_lloc, yyset\_lloc, yyget\_debug, yyset\_debug

（尽管yy\_push\_state（）和朋友如何都不会出现，除非你使用%选项堆栈）。

## 指定文件名的选项

‘--头文件=文件，%选项头文件=“FILE”’

指示flex将C表头写入文件。此文件包含函数原型、外部变量和扫描程序使用的类型。头文件只导出外部API。在扫描仪内可以使用的许多宏

操作不会被导出到头文件中。这是由于名称空间问题和一个干净的外部API的目标。

而在头中，定义了宏yyIN\_HEADER，其中“yy”用适当的前缀替换。

“——头文件”选项与“——C++”选项不兼容，因为C++扫描仪在yyFlexLexer.h中提供了自己的头。

'-oFILE, --outfile=FILE, %option outfile="FILE"'

指导flex将扫描程序写入文件文件，而不是lex.yy.c。如果将“--outfile”与“--stdout”选项合并，则扫描程序将写入stdout，但其#行指令（参见上面的“-l”选项）将参考文件file。

“-t，--不变，%期权不变”

指示flex将它生成的扫描器写入标准输出，而不是

lex.yy.c.

'-SFILE, --skel=FILE'

覆盖flex从中构建其扫描器的默认骨架文件。除非您永远不需要这个选项，否则您正在进行灵活的维护或开发。

‘--表-文件=FILE

将序列化的扫描仪dfa表写入文件。生成的扫描程序将不包含这些表，并要求在运行时加载它们。参见[序列化]，第67页。

‘--表-验证’

此选项用于可伸缩开发。我们在这里记录它，以防您偶然发现它，或者您怀疑序列化表中有一些不一致。Flex将序列化扫描程序dfa表，但也将正常生成内码表。在运行时，扫描程序将验证序列化的表与代码内表匹配，而不是加载它们。

## 影响扫描仪行为的选项

‘-i，不区分大小写，%选项不区分大小写’

指示flex生成一个不区分大小写的扫描程序。在flex输入模式中给出的字母的情况将被忽略，并且输入中的标记将被匹配。在yytext中给出的匹配文本将有保留的大小写（即，它将不会被折叠）。有关棘手的行为，请参见[大小写和字符范围]，第12页。

'-l, --lex-compat, %option lex-compat'

打开与原始AT&T lex实现的最大兼容性。请注意，这并不意味着完全的兼容性。使用此选项需要花费相当多的性能，并且它不能与“——c++”、“——full”、“--快速”、“-Cf”或“-CF”选项一起使用。有关它提供的兼容性的详细信息，请参见第20章[Lex和Posix]，第61页。此选项还将导致名称YY\_FLEX\_LEX\_COMPAT在生成的扫描器中被#定义”。

“-B，--批次，%选项批次”

指示flex生成一个批处理扫描器，这与由“——交互式”生成的交互式扫描器相反（见下文）。一般来说，当你确定你的扫描仪永远不会被交互式地使用，并且你想从中挤出一点更多的性能时，你就会使用“-B”。如果您的目标是挤出更多的性能，那么您应该使用“-Cf”或“-CF”选项，这些选项会自动打开“-批处理”。

-I，互动，%选项交互”

指示flex生成一个交互式扫描仪。交互式扫描仪是指只在绝对必须的情况下提前决定匹配的令牌。事实证明，即使扫描器已经看到了足够多的文本来消除当前令牌的歧义，那么总是要向前看一个额外的字符，也比在必要时只向前看要快一些。但是总是向前看的扫描仪会带来糟糕的交互性能；例如，当用户输入一个换行时，直到输入另一个标记才被识别为换行标记，这通常意味着输入另一整行。

弹性扫描仪默认为交互式，除非您使用“-Cf”或“-CF”表强制表达式选项（请参见第17章[性能]，第45页）。这是因为如果您正在寻找高性能，您应该使用其中的选项之一，所以如果您没有，flex假设您宁愿牺牲一些运行时性能来换取直观的交互行为。还要注意，您不能将“--交互式”与“-Cf”或“-CF”一起使用。因此，这个选项并不真正需要；在所有允许的情况下，它都是打开的。

您可以通过使用“—批处理”强制扫描仪不交互

-7位，-7位，%选项7位

指示flex生成一个7位扫描器，即一个只能识别其输入中的7位字符的扫描器。使用“——7位”的优点是，扫描仪的表的大小可以是使用“——8位”生成的表的一半。缺点是，如果扫描仪的输入包含8位字符，它们通常会挂起或崩溃。

但是，请注意，除非您使用“-Cf”或“-CF”表压缩选项生成扫描仪，否则使用“-7位”只会节省少量的表空间，并使您的扫描仪的便携性大大降低。flex的默认行为是生成一个8位扫描仪，除非您使用“-Cf”或“-CF”，在这种情况下，flex默认生成7位扫描仪，除非您的站点总是配置为生成8位扫描仪（与非美国站点通常的情况一样）。通过检查如上所述的“——详细的”输出中的标志摘要，您可以判断flex生成的是7位还是8位扫描仪。

请注意，如果您使用“-Cfe”或“-CFe”，flex仍然默认生成一个8位扫描器，因为通常使用这些压缩选项，完整的8位表并不比7位表贵多少。

“-8，-8位，%选项8位”

指示flex生成一个8位扫描器，即一个可以识别8位字符的扫描器。此标志只用于使用“-Cf”或“-CF”生成的扫描仪，否则flex默认为生成8位扫描仪。

关于flex的默认行为以及7位和8位扫描仪之间的权衡，请参见上面关于“-7位”的讨论。

‘--默认值，%选项默认值

生成默认规则。

‘--始终交互，%选项始终交互”

指示flex生成一个扫描器，它总是考虑其输入的交互性。通常，在每个新的输入文件上，扫描器调用isatty（），试图确定扫描器的输入源是否具有交互式，因此应该一次读取一个字符。但是，当使用此选项时，就不会进行这样的调用。

‘--从不互动，从不互动”

指示flex生成一个从不考虑其输入的交互式的扫描器。这与总是互动的情况正好相反。

“-X，--正，%选项正”

打开与lex的POSIX 1003.2-1992定义的最大兼容性。由于flex最初被设计用来实现lex的POSIX定义，所以这通常涉及到很少的行为变化。在目前，flex和POSIX标准之间的已知区别是：

* + - 在POSIX和AT&T的lex中，重复运算符“{}”的优先级低于连接（因此“ab{3}”产生“ababab”）。大多数POSIX实用程序使用扩展正则表达式（ERE）优先级，该优先级的重复操作符的优先级高于连接（这将导致“ab{3}”产生“abbb”）。默认情况下，flex将重复操作符的优先级高于与其他POSIX实用程序的ERE处理相匹配的连接。当指定‘--posix’或‘-l’时，flex将使用传统的AT&T和posix兼容的优先级，其中串联比重复操作符具有更高的优先级。

‘--堆栈，%选项堆栈

启用使用启动条件堆栈（请参见第10章[启动条件]，第21页）。

'--stdinit, %option stdinit'

如果已设置（即%选项stdinit），则将yyin和yyout初始化为stdin和stdout，而不是默认设置为NULL。一些现有的lex程序依赖于这个行为，即使它不符合ANSI C，它不需要stdin和stdout是编译时常数。然而，在重入扫描器中，这不是问题，因为初始化是在运行时执行的。

'--yylineno, %option yylineno'

指导flex生成一个扫描器，该扫描器维护从全局变量yylineno中的输入中读取的当前行的数量。此选项由%选项条款条款隐含。在重入C扫描仪中，无论%选项yylineno的值如何，宏yylineno都可以访问，但是，除非启用了%选项，否则yylex不会修改其值。

'--yywrap, %option yywrap'

如果未设置（即，——noyywrap），则使扫描器不在一个结束文件上调用yywrap（），但简单地假设没有更多的文件需要扫描（直到用户指向一个新文件并再次调用yylex（））。

## 代码级别和API选项

‘--ansi定义，%选项ansi定义’

已弃用，已忽略

‘--ansi-原型，%选择ansi-原型’

已弃用，已忽略

‘--野牛桥，%选项野牛桥”

指示flex生成一个打算由GNU野牛解析器调用的C扫描器。对于野牛的兼容性，扫描器有轻微的API变化。特别地，yylex的声明被修改为接受一个额外的参数，yylval。参见A.2节[野牛桥]，第122页。

‘--野牛位置，%选择野牛位置”

指示flex正在使用GNU野牛%的位置。这意味着yylex将被传递一个附加的参数，yylloc。此选项意味着%选项野牛桥。参见A.2节[野牛桥]，第122页。

-L，不，%选项

指示flex不生成#行指令。如果没有此选项，flex在生成的扫描仪中添加#行指令，因此操作中的错误消息将正确定位于原始flex输入文件（如果错误是由于输入文件中的代码）或lex.yy.c（如果错误是flex的错误-您应该将这些错误报告到第2章[报告错误]第2页中给出的电子邮件地址）。

-R，可入，%选项可入

指示flex生成一个可重入的C扫描仪。所生成的扫描仪可以安全地在多线程环境中使用。重入扫描仪的API与非重入扫描仪的不同，见第19章[重入]，第54页）。由于重入和非重入伸缩扫描仪之间的API差异，非重入伸缩代码必须经过修改，才能与此选项一起使用。此选项与“——c++”选项不兼容。

选项“——可重入”不会影响扫描仪的性能。

'-+, --c++, %option c++'

指定您希望flex生成一个C-++扫描程序类。详情请参见第18章。

‘--数组，%选项数组

指定您希望yytext是一个数组，而不是一个char\*

‘--指针，%选项指针

指定yytext应该是字符\*，而不是数组。此默认值为char \*。

'-PPREFIX, --prefix=PREFIX, %option prefix="PREFIX"'

将flex为所有全局可见变量和函数名称使用的默认“yy”前缀更改为“前缀”。例如，“——前缀=foo‘将yytext的名称更改为页脚文本。它还将默认输出文件的名称从lex.yy.c更改为lex.foo.c。以下是受影响的部分名单：

yy\_create\_buffer yy\_delete\_buffer yy\_flex\_debug yy\_init\_buffer yy\_flush\_buffer yy\_load\_buffer\_state yy\_switch\_to\_buffer yyin

yyleng yylex yylineno yyout yyrestart yytext yywrap yyalloc yyrealloc yyfree

（如果您使用C++扫描仪，则只有yywrap和yyFlexLexer受到影响。）在扫描器本身中，您仍然可以使用任意一个版本的名称来引用全局变量和函数；但从外部来看，它们有修改后的名称。

此选项允许您轻松地将多个flex程序链接到同一个可执行文件中。但是，请注意，使用此选项还可以重命名yywrap（），因此您现在必须为扫描仪提供您自己的（适当名称命名的）例程版本，或者使用%选项noyywrap，因为默认情况下，使用‘-lfl’链接不再为您提供链接。

‘--主，%选项主

指导flex为扫描仪提供一个默认的主（）程序，它只需调用yylex（）。此选项表示包装（参见下文）。

'--nounistd, %option nounistd'

禁止包含非ansi头文件unistd.h。此选项旨在针对其中不存在unistd.h的环境。请注意，某些选项可能会导致flex生成依赖于unistd.h中常见的函数的代码。isatty（），读取（）。)如果您希望使用这些函数，您必须通知编译器在哪里找到它们。请参见[选项-始终互动]，第38页。参见[选项阅读]，第41页。

'--yyclass=NAME, %option yyclass="NAME"'

仅适用于生成C++扫描仪时（“——C++”选项）。它通知flex您已将NAME作为的子类

因此，flex将您的操作放在成员函数foo：：yylex（）中，而不是：：yylex（）。它还生成一个yyFlexLexer：：yylex（）成员函数，如果调用它就会发出运行时错误（通过调用：：LexerError（））。参见第18章，第50页。

## 有关扫描仪的速度和大小的选项

'-C[aefFmr]'

控制表压缩的程度，更一般地说，控制小型扫描仪和快速扫描仪之间的权衡。

‘-C’ 一个单独的“-C”指定应该压缩扫描程序表，但不应该使用等价类或元等价类。

“-Ca，--对齐，%选项对齐”

（“align”）指示flex在生成的扫描器中权衡更大的表以获得更快的性能，因为表的元素在内存访问和计算方面更好地对齐。在一些RISC架构中，抓取和操作长字比使用诸如短字等较小大小的单元更有效。此选项可以是扫描仪使用的表格大小的四倍。

'-Ce, --ecs, %option ecs'

指导flex构造等价类，即具有相同词汇属性的字符集（例如，如果flex输入中唯一出现的数字在字符类“[0-9]”中，那么数字0、1、...，9都将被放在同一个等价类中）。等价类通常会显著减少最终的表/对象文件的大小（通常是2-5倍），而且在性能方面非常便宜（每个扫描的字符查找一个数组）。

'-Cf' 指定应该生成完整的扫描器表——flex不应该通过利用针对不同状态的类似转换函数来压缩表。

'-CF' 指定应使用备用快速扫描器表示（上面在“--fast”标志下描述）。此选项不能与“——c++”一起使用。

“-厘米，-元收益，%期权元收益”

指导flex构造元等价类，这是通常一起使用的等价类集（如果不使用等价类，或字符）集。当使用压缩表时，元等价类通常是一个很大的胜利，但是它们对性能有中等的影响（如果测试一到两个，每个扫描的字符查找一个数组）。

“-Cr，--读取，%选项读取”

导致生成的扫描仪绕过使用标准I/O库（stdio）进行输入。而不是调用fread（）或getc（）

扫描仪将使用读取（）系统调用，导致不同系统的性能提高，但一般来说可能可以忽略不计，除非您也使用“-Cf”或“-CF”。例如，如果使用‘-Cr’会导致奇怪的行为，如果您在调用扫描器之前使用stdio从yyin读取（因为如果您之前剩下的任何文本，扫描器将会丢失）。如果您定义了YY\_INPUT（）（请参见第9章[生成的扫描器]，第19页）。

选项“-Cf”或“-CF”和“-Cm”放在一起没有意义——如果表没有被压缩，就没有机会使用元等价类。否则，这些选项可以自由地混合，并且是累积的。

默认设置是“-Cem”，它指定flex应该生成等价类和元等价类。此设置提供了最高程度的表压缩。您可以以使用更大的表为代价来权衡执行速度更快的扫描器，以下情况通常是正确的：

最慢和最小

-Cem

-Cm

-Ce

-C

-C{f,F}e

-C{f,F}

-C{f,F}a

最快和最大

请注意，具有最小表的扫描仪通常是最快地生成和编译的，因此在开发过程中，您通常希望使用默认的最大压缩。

“-Cfe”通常是生产扫描仪的速度和大小之间的一个很好的折衷方案。

“-f，--满，%选项满”

指定快速扫描仪。显著的压缩被完成，stdio被绕过。结果虽然很大，但速度很快。此选项相当于“——Cfr”

“-F，快速，%选择快速”

指定应使用快速扫描器表表示方式（并绕过stdio）。这种表示大约与完整表表示“——full”一样快，对于某些模式集将会小得多（对于其他模式集，将会更大）。*一般来说，如果模式集同时包含关键字和一个集合，标识符规则，例如在集中：*

情况 返回TOK\_CASE；“开关” 返回TOK\_SWITCH；

...

“默认”返回TOK\_DEFAULT；[a-z]+ 返回TOK\_ID；

然后，您最好是使用完整的表表示法。*如果只存在标识符规则，然后使用哈希表或其他类似的表来检测关键字，那么最好使用“——fast”。*

此选项相当于“-CFr”。它不能与“——c++”一起使用。

## 调试选项

“-b，--备份，%选项备份”

生成备份信息。这是一个需要备份的扫描器状态列表和它们这样做的输入字符列表。通过添加规则，您可以删除备份状态。如果消除了所有备份状态，并使用“-CF”或-CF，生成的扫描仪将运行得更快（请参见“-perf-报告”标志）。只有希望从扫描仪中挤出每一个周期的用户才需要担心这个选项。（见第17章，第45页）。

“-d，--调试，%选项调试”

使生成的扫描程序在调试模式下运行。当识别一个模式且全局变量yy\_flex\_debug非零（默认值）时，扫描仪将向stderr写入一行表单：

-第53行中的接受规则（“匹配的文本”）

行号表示该规则在定义扫描程序的文件中的位置（即，被输入给flex的文件）的位置。当扫描器备份、接受默认规则、到达其输入缓冲区末端（或遇到NUL；此时，就扫描器而言，两者外观相同）或到达文件结尾时，也会生成消息。

“-p，-绩效报告，%期权绩效报告”

生成一个提交给stderr的性能报告。该报告包括关于flex输入文件的特性的评论，这将导致结果扫描仪的严重性能损失。如果您给出这个标志两次，您也会得到关于导致轻微性能损失的特性的评论。

请注意，使用拒绝和变量尾随上下文（参见第24章[限制]，第74页）会导致严重的性能损失；使用yymore（）、“^”操作符和“——交互式”标志会导致轻微的性能损失。

'-s, --nodefault, %option nodefault'

导致抑制默认规则（不匹配的扫描器输入响应到stdout)。如果扫描程序遇到的输入与其任何规则不匹配，它会出错。此选项对于查找扫描仪规则集中的漏洞非常有用。

“-T，-跟踪，%选项跟踪”

使flex在跟踪模式下运行。它将向stderr生成大量关于输入的形式和由此生成的非确定性和确定性有限自动机的消息。此选项主要用于维护flex。

'-w, --nowarn, %option nowarn'

抑制警告消息。

“-v，详细，%选项详细”

指定flex应该向stderr写入关于它生成的扫描程序的统计信息的摘要。大多数统计数据对于普通的flex用户来说都是没有意义的，但是第一行标识了flex的版本（与“——版本”报告的相同），下一行标识了生成扫描仪时使用的标志，包括默认情况下打开的标志。

‘--警告，%选项警告

警告某些事情。特别是，如果可以匹配默认规则，但没有给出默认规则，则flex会警告您。我们建议始终使用此选项。

## 其他选项

‘-c’ 包括符合POSIX要求的免费选项。“-h，-？，帮助”

生成flex的静态退出选项的“帮助”摘要，然后退出。‘-n’ 另一个符合POSIX的选项。

“-V，-版本”

打印版本号以确定并退出。

# 性能注意事项

flex的主要设计目标是，它能够生成高性能的扫描仪。它经过了优化，可以很好地处理大量的规则集。除了上面概述的表压缩“-C”选项对扫描器速度的影响外，还有许多选项/操作会降低性能。这些都是，从最便宜的到最便宜的：

拒绝

任意跟踪上下文

需要备份的模式集

%option yylineno

%array

%选项交互式

%选项始终交互式

\*起跑线操作员yymore（）

因为前两个都很贵，最后两个都很便宜。还要注意，unput（）被实现为一个例程调用，它可能会做相当多的工作，而无yyput（）是一个相当便宜的宏。所以，如果你只是把一些你扫描的多余文本，使用无yys（）。

当绩效很重要时，应不惜一切代价避免拒绝。这是一个特别昂贵的选择。

有一种情况下，%选项伊利内诺可能会很昂贵。这是当您的模式与可能包含换行符的长标记相匹配时。对于不可能匹配换行符的规则，没有性能损失，因为flex不需要检查它们是否有换行符。一般来说，您应该避免像[^f]+这样的规则，它们匹配非常长的令牌，包括换行符，并且可能匹配您的整个文件！一个更好的方法是将[^f]+分为两个规则：

%option yylineno

%%

[^f\n]+

\n+

上述扫描仪不会造成性能损失。

摆脱备份是混乱的，对于复杂的扫描仪来说通常是大量的工作。原则上，我们首先使用“-b”标志来生成一个lex.备份文件。例如，在输入项上：

%%

foo 返回TOK\_KEYWORD；页脚 返回TOK\_KEYWORD；

文件看起来像：

状态#6是不接受关联的规则行号：

2 3

外过渡：

干扰转换：EOF p-\177 ]

状态#8是不接受关联的规则行号：

3

输出转换： [ a ]

干扰转换：EOF[001-‘ b-\177 ]

状态#9是不接受关联的规则行号：

3

外过渡：

干扰转换：EOF s-\177 ]

压缩表总是备份。

前几行告诉我们，有一种扫描状态，它可以在‘o’上进行转换，但不能在任何其他字符上进行转换，在这种状态下，当前扫描的文本不匹配任何规则。当试图匹配在输入文件中的第2行和第3行中找到的规则时，将发生该状态。如果扫描仪处于这种状态，然后读取“o”以外的东西，它将必须备份以找到匹配的规则。有点一下，你可以看到这一定是它看到“fo”时所处的状态。当这种情况发生时，如果看到其他“o”以外的任何东西，扫描仪将必须备份以简单地匹配“f”（默认规则）。

关于状态#8的注释表明，当“foob”被扫描时存在问题。事实上，除了“a”之外的任何角色，扫描仪都必须备份才能接受“foo”。同样地，当“fooba”被扫描时，“r”没有跟随。

最后的评论提醒我们，除非我们使用“-Cf”或“-CF”，否则没有必要去删除规则中的备份，因为压缩扫描仪不会提高性能。

删除备份的方法是添加“错误”规则：

%%

foo 返回TOK\_KEYWORD；页脚 返回TOK\_KEYWORD；

fooba |

foob |

fo {

/\*误报，并不是一个真正的关键字\*/返回TOK\_ID；

}

在关键字列表中消除备份也可以使用“全面”规则来完成：

%%

foo 返回TOK\_KEYWORD；页脚 返回TOK\_KEYWORD；

[a-z]+ 返回TOK\_ID；

在适当的时候，这通常是最好的解决方案。

备份消息往往会串联起来。对于一组复杂的规则，获得数百条消息并不少见。但是，如果可以破译它们，通常只需要十几条规则就可以消除备份(尽管很容易犯错误和错误规则意外地匹配有效令牌。未来可能的一个灵活特性将是自动添加规则，以消除备份)。

重要的是要记住，只有消除每个备份实例，才能获得消除备份的好处。只留下一个，就意味着你什么也得不到。

可变尾随上下文（领先和后面部分都没有固定长度）导致与拒绝几乎相同的性能损失（即大量）。所以，如果可能的话，会有一个规则，就像：

%%

mouse|rat/(cat|dog) 跑

写得更好：

%%

mouse/cat|dog 跑

rat/cat|dog 跑

或作为

%%

mouse|rat/cat 跑

mouse|rat/dog 跑

请注意，这里的特殊的“|”行动并没有提供任何节省，甚至可能使事情变得更糟（见第24章[限制]，第74页）。

用户可以提高扫描仪性能（更容易实现）的另一个领域是，令牌匹配的时间越长，扫描器运行速度就越快。这是因为对于长令牌，大多数输入字符的处理发生在（短的）内部扫描循环中，并且不需要完成为操作设置扫描环境（例如yytext）的额外工作。回忆一下扫描仪上的C语言：

%x评论

%%

因特line\_num = 1；

"/\*" 开始（评论）；

<comment>[^\*\n]\*

<comment>"\*"+[^\*/\n]\*

<comment>\n ++line\_num;

<comment>"\*"+"/" 开始（初始）；

这一点可以通过把它写成：

%x评论

%%

因特line\_num = 1；

"/\*" 开始（评论）；

<comment>[^\*\n]\*

<comment>[^\*\n]\*\n ++line\_num;

<comment>"\*"+[^\*/\n]\*

<comment>"\*"+[^\*/\n]\*\n ++line\_num;

<comment>"\*"+"/" 开始（初始）；

现在，每个换行符都不需要处理另一个操作，识别换行符会分布在其他规则上，以尽可能长时间地处理匹配的文本。请注意，添加规则并不会降低扫描仪的速度！扫描器的速度与规则的数量或（根据本节开头给出的考虑）对于诸如\*和|等操作符，规则有多复杂无关。

加速扫描仪的最后一个例子是：假设您想要扫描一个包含标识符和关键字的文件，每行一个，没有其他无关的字符，并识别所有的关键字。最自然的第一个方法是：

%%

asm |

汽车 |

破碎 |

......等挥发性|

在…期间 /\*，这是一个关键字\*/

.|\n /\*，它不是一个关键字\*/

要消除追溯，请引入全面规则：

%%

asm |

汽车 |

破碎 |

......等挥发性|

在…期间 /\*，这是一个关键字\*/

[a-z]+ |

.|\n /\*，它不是一个关键字\*/

现在，如果保证每行只有一个单词，那么我们可以通过合并换行与其他令牌的识别来将匹配总数减少一半：

%%

asm\n |

auto\n |

break\n |

......等挥发性\n|

while\n /\*，这是一个关键字\*/

[a-z]+\n |

.|\n /\*，它不是一个关键字\*/

这里必须小心，因为我们现在已经重新引入备份到扫描仪。特别是，虽然我们知道永远不会有任何字符在输入流除了字母或换行符，flex不能找出这一点，它将计划可能需要备份时扫描一个令牌像“自动”，然后下一个字符不是换行符或一个字母。以前它只匹配“自动”规则并完成，但现在它没有“自动”规则，只有一个“自动\n”规则。为了消除备份的可能性，我们可以复制所有规则，但不需要最后的换行，或者，因为我们从不期望遇到这样的输入，因此不知道它是如何分类的，我们可以引入一个全面的规则，这个不包括换行：

%%

asm\n |

auto\n |

break\n |

......等挥发性\n|

while\n /\*，这是一个关键字\*/

[a-z]+\n |

[a-z]+ |

.|\n /\*，它不是一个关键字\*/

用“-Cf”编译，这是一个弹性扫描仪来解决这个特殊问题的速度。

最后要注意的是：当匹配nul时，flex会很慢，特别是当一个标记包含多个nul时。如果预期文本通常会包含nul，那么最好编写匹配短期文本的规则。

关于性能的另一个最后注意事项：如第7章[匹配]，第14页所述，动态调整yytext的大小以适应巨大的令牌是一个缓慢的过程，因为它目前要求从一开始就重新扫描（巨大的）令牌。因此，如果性能至关重要，那么您应该尝试匹配“大量”文本，而不是“大量”文本，其中两者之间的截断点是每个标记约为8K个字符。

# 生成C++扫描仪

重要的是：目前的扫描类的形式是实验性的，可能在主要版本之间有很大的变化。

flex提供了两种不同的方法来生成供C++一起使用的扫描仪。第一种方法是简单地使用C++编译器而不是C编译器来编译由flex生成的扫描器。您应该不会遇到任何编译错误（请参见第2章[报告错误]，第2页）。然后，您可以在规则操作中使用C-++代码，而不是C-C代码。注意，扫描仪的默认输入源仍然是yyin，默认回和仍然是yyout。这两个变量都仍然是FILE \*变量，而不是C-++流。

您还可以使用flex生成C++扫描程序类，使用‘-+’选项（或者，%选项c++），如果flex可执行文件的名称以‘+’结束，则会自动指定，例如flex++。当使用此选项时，flex默认为将扫描程序生成到文件lex.yy.cc，而不是lex.yy.c。生成的扫描程序包括头文件FlexLexer.h，它定义了到两个C++类的接口。

FlexLexer.h中的第一个类，FlexLexer，提供了一个定义通用扫描程序类接口的抽象基类。它提供了以下成员功能：

常量字符\*YYText（）

返回最近匹配的令牌的文本，相当于yytext。

int YYLeng()

返回最近匹配的标记的长度，相当于yyleng。

这是（）常量

返回当前输入行号（请参阅%选项yylineno)，如果未使用%选项yylineno，则返回1。

void set\_debug（int标志）

设置扫描仪的调试标志，相当于分配给yy\_flex\_调试（参见第16章[扫描器选项]，第35页）。请注意，您必须使用%选项调试来构建扫描程序，才能在其中包含调试信息。

int调试（）常量

返回调试标志的当前设置。

还提供了相当于yy\_switch\_to\_buffer（）、yy\_create\_缓冲区（）（尽管第一个参数是流和对象引用而不是文件\*）、yy\_flush\_buffer（）、yy\_delete\_buffer（）和yy重启（）（同样，第一个参数是流和对象引用）的成员函数。

在FlexLexer.h中定义的第二个类是yyFlexLexer，它来源于

FlexLexer。它定义了以下附加的成员函数：

yyFlexLexer ( istream\* arg\_yyin = 0, ostream\* arg\_yyout = 0) yyFlexLexer ( istream& arg\_yyin, ostream& arg\_yyout )

使用给定的输入和输出流构造一个yyFlexLexer对象。如果没有指定，则流分别默认为cin和cout。yyFlexLexer不拥有其流参数的所有权。这取决于用户来确保指向的流至少像yyFlexLexer实例一样保持活动。

虚拟intylex（）

执行与yylex（）对普通的flex扫描仪相同的角色：它扫描输入流，消耗令牌，直到规则的操作返回一个值。如果您从yyFlexLexer派生出一个子类S，并且希望访问yylex（）中的S的成员函数和变量，那么您需要使用%选项yy类=“S”来通知flex您将使用该子类而不是yyFlexLexer。在这种情况下，flex生成（）：：（）（还生成一个虚拟（），如果调用，调用：：（）错误（））。

虚拟虚空开关流（流\*新输入=0，流\*新输出= 0)虚拟虚空开关流（流和新输入、流和新输出）

将yyin重新分配为new\_in（如果非空），将yyout重新分配为new\_out（如果是非空），如果将yyin重新分配，则删除以前的输入缓冲区。

（新流，新流，新流，新流=0）（新流和新流，新流和新流）

首先通过开关流（new\_in，new\_out）切换输入流，然后返回yylex（）的值。

此外，yyFlexLexer定义了以下受保护的虚拟函数，您可以在派生类中重新定义这些虚拟函数，以定制扫描仪：

虚拟输入语言输入（字符串，int max\_size）

读取多达max\_size个字符到buf中，并返回读取的字符数。若要指示输入的结束，请返回0个字符。请注意，交互式扫描仪（参见第16章[扫描仪选项]第35页中的“-B”和“-I”标志）定义了宏YY\_INTERACTIVE。如果重新定义LexerInput（），并且需要根据扫描器是否正在扫描交互式输入源采取不同的操作，则可以通过#ifdef语句测试该名称的存在。

虚拟空空间输出（常大小，内部大小）

从缓冲区buf中写入大小字符，如果扫描器的规则可以与文本的Nul匹配，但也可能包含内部NUL。

虚拟空间误差（常信息）

报告了一条致命的错误消息。此函数的默认版本会将消息写入流cerr并退出。

请注意，yyFlexLexer对象包含其整个扫描状态。因此，您可以使用这些对象来创建重入扫描仪，但也请参见第19章[重入扫描仪]，第54页。您可以实例化同一个yyFlexLexer类的多个实例，还可以使用上面讨论的‘-P’选项在同一程序中将多个C++扫描程序类组合在一起。

最后，请注意，%数组功能对C++扫描程序类不可用；您必须使用%指针（默认值）。

下面是一个简单的C-++扫描仪的例子：

//使用flex C++扫描仪类的一个示例。

%{

#包括<流>使用名称空间标准；<=0；

%}

%选项noywrapc++字符串 \"[^\n"]+\" ws [ \t]+

阿尔法 [A-Za-z]

挖 [0-9]

名字 ({alpha}|{dig}|\$)({alpha}|{dig}|[\_.\-/$])\*

num1 [-+]?{dig}+\.?([eE][-+]?{dig}+)?

num2 [-+]?{dig}\*\.{dig}+([eE][-+]?{dig}+)?数字 {num1}|{num2}

%%

{ws} /\*跳过空格和标签\*/“/\*” {

int c;

while((c = yyinput()) != 0)

{

如果（c == ‘\n’）

++mylineno;

否则，如果（c == ‘\*’）

{

如果（（c=yyinput（））==‘/’）中断；

其他的

}

}

}

unput (c);

{number} cout <<“编号”<< YYText（）<< ‘\n’；

\n mylineno++;

{name} cout <<“name”<< YYText（）<< ‘\n’；

{string} cout <<“字符串”<< YYText（）<< ‘\n’；

%%

//如果主（）是另一个源文件，则包含这是必需的。

//#include <FlexLexer.h>

int主（int/\*argv\*/）

{

新的=；而（lexer->yylex（）！= 0）

;

返回0；

}

如果您想创建多个（不同的）lexer类，请使用“-P”标志（或前缀=选项）将每个yyFlexLexer重命名为其他的“xxFlexLexer”。然后，您可以包括<FlexLexer.h>在您的其他来源，每一次词汇量类，首先重命名yyFlexLexer如下：

#undef yyFlexLexer

#define yyFlexLexer xxFlexLexer

#include <FlexLexer.h>

#undef yyFlexLexer

#define yyFlexLexer zzFlexLexer

#include <FlexLexer.h>

例如，如果其中一个扫描仪使用%选项前缀=“xx”，而另一个扫描仪使用%选项前缀=“zz”。

# RentrantC扫描仪

flex有能力生成一个可重入的C扫描仪。这是通过指定来完成的

%选项可重入（“-R”）生成的扫描仪是便携式的，并且可以在一个或多个单独的控制线程中安全使用。重入扫描仪最常见的用途是来自多线程应用程序。任何线程都可以创建和执行一个重入伸缩扫描器，而不需要与其他线程同步。

## 用于重入扫描仪

然而，重入式扫描仪还有其他用途。例如，您可以同时扫描两个或多个文件，以在令牌级别（即，而不是在字符级别）上实现差异：

/\*维护多个活动扫描仪的示例。\*/

做{

int tok1, tok2;

tok1 = yylex（scanner\_1）；托克2=yylex（scanner\_2）；

如果（tok1 ！= tok2）

文件是不同的。");

而（tok1 && tok2）；

重入式扫描仪的另一个用途是递归。(请注意，递归扫描器也可以使用非重入扫描器和缓冲区状态来创建递归扫描器。请参见第11章[多个输入缓冲区]，第27页。）

下面的粗扫描器通过调用其自身的另一个实例来支持“eval”命令。

/\*递归调用的示例。\*/

选择重入百分比

%%

"eval(".+")" {

yyscan\_t扫描仪；YY\_BUFFER\_STATE buf；

yylex\_init ( &scanner ); yytext[yyleng-1] = ' ';

buf = yy\_scan\_string（yytext + 5，扫描仪）；yylex（扫描仪）；

yy\_delete\_buffer（buf、扫描仪）；

...

%%

yylex\_destroy（扫描仪）；

}

## 重入式API的概述

重入扫描仪的API不同于非重入扫描仪。下面是对该API的一个快速概述：

必须指定%选项的可重入项。

* + - 所有函数都需要一个附加参数：yyscaner
    - 所有的全局变量都被它们的宏等价物所取代。（我们告诉您这一点，是因为在调试过程中，它可能对您很重要。）
    - yylex\_init和yylex\_destroy必须分别在yylex之前和之后被调用。
    - 访问器方法(get/set函数）提供了对通用flex变量的访问。
    - 用户特定的数据可以存储在yyextra中。

## 重入示例

首先，一个重入扫描仪的例子：

/\*这个扫描仪打印“//”评论。\*/

%选项重入堆栈noywrap

%x评论

%%

"//" yy\_push\_state（评论，y扫描仪）；

.|\n

<COMMENT>\n yy\_pop\_state（yy扫描仪）；

<COMMENT>[^\n]+ fprintf ( yyout, "%s\n", yytext);

%%

主（主，[]）

{

yyscan\_t扫描仪；

yylex\_init（&扫描仪)；yylex（扫描仪）；yylex\_destroy（扫描仪）；

返回0；

}

## 详细的重入式API

以下是您在使用flex的重入C API时需要做或知道的事情。

### 声明扫描仪为可重入者

必须指定%选项重入（-重入）。

请注意，在上面的例子中指定了%选项重入者（参见第55页第19.3节[重入者例子]页。如果没有指定这个选项，flex会很高兴地生成一个非重入扫描器。如果您不需要一个重入扫描仪，您可以显式地指定%选项不重入，尽管它不是必要的。默认情况下是生成一个非重入扫描仪。

### 额外的论点

所有函数都需要一个附加参数：yyscanen。

请注意，对yy\_push\_state和yy\_pop\_state的调用都有一个参数，yy扫描器，该参数在非重入扫描器中不存在。以下是yy\_push\_state和yy\_pop\_state在重入扫描仪中的声明：

静态空白yy\_push\_state （int new\_state，yyscan\_t yy扫描仪）；静态空白yy\_pop\_state ( yyscan\_t yyscanner ) ;

注意，参数yyscant出现在两个函数的声明中。事实上，在一个可重入的扫描仪中的所有flex函数都有这个额外的参数。它总是参数列表中的最后一个参数，它总是yyscan\_t类型（类型为空\*），它总是命名为yy扫描器。正如您可能已经猜到的，yyscaner是指向封装扫描器当前状态的不透明数据结构的指针。有关函数声明的列表，请参见第19.5节，第59节[重入函数]。请注意，预处理器宏，如BEGIN、ECHO和拒绝，不会接受此附加参数。

### 被宏替换的全局变量

传统flex中的所有全局变量都已被宏观等价物所取代。

请注意，在上面的示例中，yyout和yytext并不是纯变量。这些宏将扩展到它们的等效l值。所有我们熟悉的弹性全局元素都已被它们的宏观等价物所取代。特别是，文本、日志、日志、文本、文本、日志和日志都是宏。您可以在操作中安全地使用这些宏，就好像它们是普通变量一样。我们只告诉你，所以您不期望从外部链接到这些变量。目前，每个宏都会扩展到一个内部结构体中的一个成员，例如，

#定义yytext（（结构yyguts\_t\*）yy扫描仪）->yytext\_r)

关于yytext和朋友，需要记住的一件重要的事情是，yytext在重入扫描器中不是一个全局变量，您不能直接从操作外部或其他函数访问它。您必须使用一个访问器方法，e。g.，yyget\_text，来完成这一点。（见下文）。

### 初始化和销毁函数

yylex\_init和yylex\_destroy必须分别在yylex之前和之后被调用。

int yylex\_init（yyscan\_t \* ptr\_yy\_globals）；

int yylex\_init\_extra（YY\_EXTRA\_TYPE user\_定义，yyscan\_t \* ptr\_yy\_globals）；int yylex（yyscan\_t yy扫描仪）；

int yylex\_destroy（yyscan\_t y扫描仪）；

必须在调用任何其他函数之前调用该函数yylex\_init。yylex\_init的参数是一个要由yylex\_init填充的未初始化指针的地址，它将覆盖以前的任何内容。可以使用函数yylex\_init\_extra，以YY\_EXTRA\_TYPE类型的变量作为其第一个参数。有关更多细节，请参见下面的yyextra部分。

存储在ptr\_yy\_globals中的值之后应该传递给yylex和yylex\_销毁。Flex不保存传递给yylex\_init的参数，因此将本地指针的地址传递到yylex\_init是安全的，只要它在对扫描仪的所有调用期间保持在范围内，直到并包括对yylex\_destroy的调用。

函数yylex现在应该很熟悉了。重入版本采用一个参数，即yylex\_init（通过一个参数）返回的值。否则，它的行为与非yylex的非重入版本相同。

yylex\_init和yylex\_init\_extra在成功时都返回0（零），或在失败时返回非零，在这种情况下，errno被设置为以下值之一：

* + - * ENOMEM内存分配错误。请参见[内存管理]，第64页。
      * EINVAL无效的参数。

应该调用函数yylex\_destroy来释放扫描器使用的资源。调用yylex\_destroy后，不应该使用y扫描器的内容。当然，如果您计划重用扫描仪，就没有必要销毁它。可以通过调用yyert重新启动弹性扫描仪（重新进入和非重新进入）。

下面是一个程序创建一个例子，扫描仪，使用它，然后销毁它：

int主（）

{

yyscan\_t扫描仪；int tok；

yylex\_init(&scanner);

而（（tok=yylex（扫描仪））>0）

printf("tok=%d yytext=%s\n“，tok，yyget\_text（扫描仪）)；

yylex\_destroy（扫描仪）；返回0；

}

### 使用重入扫描仪访问变量

访问器方法(get/set函数）提供了对通用flex变量的访问。

您所构建的许多扫描仪都将是一个更大的项目的一部分。项目的部分内容将需要访问flex值，例如yytext。在非重入扫描仪中，这些值是全局的，因此访问它们没有问题。然而，在一个重入扫描仪中，在那里

都是没有全局弹性值。您不能直接访问它们。相反，您必须使用访问器方法(get/set函数）来访问flex值。每个访问器方法都被命名为yyget\_NAME或yyset\_NAME，其中NAME是您想要的flex变量的名称。例如：

/\*将yytext的最后一个字符设置为NULL。\*/空白chop（yyscan\_t扫描仪）

{

int len = yyget\_leng（扫描仪）；yyget\_text（扫描仪）[yyget\_text-1]=‘\0’；

}

上面的代码可以从像这样的操作中被调用：

%%

.+\n {chop（yy扫描仪）；}

您可能会发现，%选项头文件对于生成所有访问器函数的原型特别有用。参见[选项标题]，第35页。

### 额外的数据

用户特定的数据可以存储在yyextra中。

在重入扫描仪中，使用全局变量与程序的不同部分之间进行通信或保持状态是不明智的。但是，您可能需要访问外部数据或从扫描器操作中调用外部函数。同样地，您可能需要将信息传递给扫描仪（例如，打开文件描述符，或数据库连接）。在非重入扫描仪中，唯一的方法是通过使用全局变量。Flex允许您在扫描仪中存储任意的“额外”数据。这些数据可以通过访问器方法yyget\_extra和yyset\_extra从扫描器外部访问，并通过扫描器内部的快捷方式宏yyextra访问。其定义如下：

#定义YY\_EXTRA\_TYPE void\*

yy\_extra\_type yyget\_extra（yyscan\_t扫描仪）；

空的 yyset\_extra（YY\_EXTRA\_TYPE任意\_data，yyscan\_t扫描仪）；

此外，还提供了一种额外的yylex\_init形式，即yylex\_init\_extra。提供此函数是为了使可以从第一个yyalloc中访问yy额外的值，用于分配扫描器本身。

默认情况下，YY\_EXTRA\_TYPE被定义为类型void \*。您可以使用

%选项在扫描仪中附加类型的=“your\_type”：

/\*：重写YY\_EXTRA\_TYPE的一个示例。\*/

%{

#include <sys/stat.h>

#include <unistd.h>

%}

选择重入百分比

%选项额外类型的=“结构体统计\*”

%%

文件大小

打印(“%ld”，yyextra->st\_size );

lastmod

%%

打印文件（“%ld”，yyextra->st\_mtime）；

void scan\_file (字符串\*文件名）

{

yyscan\_t扫描仪；结构统计buf；文件\*在；

在= fopen（文件名，“r”）；stat（文件名，&buf）；

yylex\_init\_extra（buf，和扫描仪）；yyset\_in（在中，扫描仪）；

yylex (扫描仪）；yylex\_destroy (扫描仪）；

Fcount（英寸）；

}

### 关于yyscan t

yyscan\_t定义为：

typedef空白\*yyscan\_t；

它由yylex\_init（）初始化，以指向一个内部结构。您永远不应该直接访问此值。特别是，您不应该尝试释放它（使用yylex\_销毁（））。

## 19.5重入C扫描仪中可用的功能和宏

重入扫描仪中具有以下功能：

char \*yyget\_text (yyscan\_t扫描仪）；int yyget\_leng（yyscan\_t扫描仪）；文件\*yyget\_in (yyscan\_t扫描仪）；文件\*yyget\_out（yyscan\_t扫描仪）；int yyget\_lineno（yyscan\_t扫描仪）；

YY\_EXTRA\_TYPE yyget\_extra（yyscan\_t扫描仪）；int yyget\_debug（yyscan\_t扫描仪）；

void yyset\_debug（int标志，yyscan\_t扫描仪）；空白yyset\_in （文件\* in\_str，yyscan\_t扫描仪）；

空白yyset\_out （文件\* out\_str，yyscan\_t扫描仪）；空白yyset\_lineno（intline\_nuzer，yyscan\_t扫描仪）；

void yyset\_extra（YY\_EXTRA\_TYPE user\_定义，yyscan\_t扫描仪）；

yytext和yyyleng没有“设置”函数。这是故意的。

在重入扫描仪中提供以下宏快捷方式：

yytext

yyleng yyin yyout yylineno yyextra

yy\_flex\_debug

在可重入的C扫描仪中，始终存在对yylineno的支持（即，您可以访问yylineno），但是除非启用了%选项yylineno，否则flex永远不会修改该值。这是为了允许用户独立于flex来维护行计数。

当%选项牛桥时，以下功能和宏可用

（指定了“野牛桥”）：

yyget\_lval\*yyget\_lval（yyscan\_t扫描仪）；

空白yyset\_lval（yyset\_lval，yyscan\_t扫描仪）；yyset\_lval

当%选项野牛位置时，以下功能和宏可用

（已指定了“野牛位置”）：

乙烯型\*yyget\_lloc（yyscan\_t扫描仪）；

空白yyset\_lloc（聚乙烯型，yyscan\_t扫描仪）；聚乙烯

支持yylval假设YYSTYPE是一个有效的类型。支持yylloc假设yylype是一个有效的类型。通常，这些类型是由野牛生成的，并包含在flex输入的第1节中。

# 与Lex和Poxix不兼容

flex是对AT&T Unix lex工具的重写（尽管这两种实现不共享任何代码），有一些扩展和不兼容，这两者都是那些希望编写两种实现都可以接受的扫描仪的人所关心的。flex完全符合POSIX的lex规范，除了在使用%指针（默认值）时，对取消输入（）的调用会破坏yytext的内容，这与POSIX规范相反。在本节中，我们将讨论flex、AT&T的lex和POSIX规范之间的所有已知的不兼容性领域。Flex的‘-l’选项打开了与原始AT&T lex实现的最大兼容性，代价是生成的扫描仪性能的重大损失。下面我们注意到可以使用“-l‘选项克服不兼容。flex与lex完全兼容，但有以下例外情况：

* 除非使用‘-l’或%选项yylineno，否则不支持未文档化的lex扫描器内部变量yylineno。
* yylineno应该在每个缓冲区的基础上进行维护，而不是在每个扫描器（单个全局变量）的基础上进行维护。
* 伊利内诺不是POSIX规范的一部分。
* 输入（）例程是不可重新定义的，尽管它可以被调用来读取规则匹配的字符。如果输入（）遇到文件结束，则完成正常的yywrap（）处理。一个“真正的”文件结束点由输入（）返回为EOF。
* 相反，输入操作可以通过定义YY\_INPUT（）宏来进行控制。
* 输入（）不能被重新定义的flex限制符合POSIX规范，该规范只是没有指定如何控制扫描仪的输入。
* 未输入的（）例程不能重新定义。此限制符合POSIX的标准。
* 弹性扫描仪不像弹性扫描仪那样可重入。特别是，如果您有一个交互式扫描器和一个中断处理程序，它跳跳出扫描器，随后再次调用扫描器，您可能会得到以下消息：

致命的flex扫描器内部错误-缓冲区结束丢失

若要重新进入扫描仪，请首先使用：

yyrestart ( yyin );

注意，这个调用将丢弃任何缓冲输入；通常这不是交互式扫描仪的问题。参见flex的第19章[重入]，第54页，了解它的重入API。

* 还要注意，flex C++扫描仪类是可重入的，所以如果使用C++是您的一个选项，您应该使用它们来代替。详情见第18章第50页和第19章第54页。
* 不支持输出（）。完成从ECHO宏的输出到文件指针

yyout（默认标准值）。

* 输出（）不是POSIX规范的一部分。
* lex不支持独占的启动条件（%x），尽管它们在POSIX规范中。
* 展开定义时，flex将它们括在括号中。使用lex，下面是：

名称 [A-Z][A-Z0-9]\*

%%

foo{NAME}? printf（“找到它”）；

%%

将不匹配字符串“foo”，因为当宏展开时，该规则等同于“foo[A-Z][A-Z0-9]\*？’的优先级是‘？“与“\*”相关联。使用flex，规则将扩展为foo？所以字符串foo就会匹配。

* 请注意，如果定义以“^”开头或以“$”结尾，那么它就不会用括号展开，以允许这些操作符出现在定义中而不丢失它们的特殊含义。但是“<s>”、“/”和<<EOF>>运算符不能在灵活的定义中使用。
* 使用“-l”会导致在定义周围没有圆括号的词汇行为。
* POSIX规范是将定义包含在括号中。
* 规则的一些词汇的实现允许规则的操作从单独的行开始，如果规则的模式有尾随空格：

%%

foo|bar<space here>

{动作（）；}flex不支持此功能。

* 不支持lex%r（生成Ratfor扫描仪）选项。它不是POSIX规范的一部分。
* 在调用取消输入（）之后，yytext将未定义，直到下一个标记被匹配，除非扫描器是使用%数组构建的。这不是lex或POSIX规范的情况。“-l”选项消除了这种不兼容性。
* “{，}”（数字范围）运算符的优先级不同。AT&T和POSIX规范将“abc{1,3}”解释为匹配一次、两次或三次出现“abc”，而flex将其解释为“匹配“ab”，然后一次、两次或三次出现“c”。“-l”和“--posix”选项消除了这种不兼容性。
* “^”运算符的优先级有所不同。莱克斯将“^foo|bar”解释为“在一行的开头匹配“foo”，或者在任何地方匹配“bar”，而弗lex则将其解释为“匹配“foo”或“酒吧”在一行的开头”。后者符合POSIX规范。
* 不需要词汇支持的特殊表大小声明

弯曲扫描仪。flex忽略了它们。

* 名称FLEX\_SCANNER是#定义的，因此扫描仪可以编写用于flex或lex。扫描仪还包括YY\_FLEX\_MAJOR\_VERSION、YY\_FLEX\_MINOR\_VERSION和YY\_FLEX\_SUBMINOR\_VERSION，它们指示哪个版本的flex生成了扫描仪。例如，对于2.5.22版本，这些定义将分别为2、5和22。如果使用的flex版本是beta版，则定义符号FLEX\_BETA。
* 输入代码部分中的符号[[]和]]可能与m4分隔符冲突。参见第A.3节[M4依赖性]，第124页。

以下flex特性不包括在lex或POSIX规范中：

* C++扫描仪
* %option
* 启动条件作用域
* 启动条件堆栈
* 交互式/非交互式扫描仪
* yy扫描字符串（）和朋友
* yyterminate()
* yy设置交互式（）
* yy集符号（）
* YY在BOL（）<<EOF>>

• <\*>

* YY DECL
* YY开始
* YY用户操作
* YY用户初始化
* #行指令
* %{}‘围绕动作
* 重入C API
* 在一行上的多个操作
* 几乎所有的flex命令行选项

“一行上的多个动作”指的是使用flex，你可以把多个动作放在同一行上，用半象号分隔，而使用lex，如下：

foo handle\_foo() ; ++num\_foos\_seen ;

（相当令人惊讶的是）被截断为

foo handle\_foo() ;

flex不会截断该操作。未包含在大括号中的操作只在行的末尾终止。

# 内存管理

本章介绍flex如何处理动态内存，以及如何覆盖默认行为。

## 默认内存管理

Flex在初始化期间分配动态内存，偶尔在调用yylex（）时分配动态内存。初始化将在第一次调用yylex（）时进行。此后，如果flex需要扩大缓冲区，它可以重新分配更多的内存。从2.5.9版本开始，当你调用yylex\_destroy时，Flex将清理所有的内存。

Flex为动态内存分配了四个目的，如下所示1

输入缓冲区为16kB。

Flex为用于执行模式匹配的字符缓冲区分配内存。Flex必须从输入流中提前读取，并将其存储在一个大的字符缓冲区中。这个缓冲区通常是动态内存弹性消耗的最大块。如果有必要，这个缓冲区将会增长，每次的大小都会增加一倍。当您调用yylex销毁（）时，Flex将释放此内存。这个缓冲区的默认大小（16384字节）几乎总是太大的。这个缓冲区的理想大小是预期的最长标记的长度，以字节为单位，再加上稍微多一点。Flex将分配一些额外的字节的管家。目前，要覆盖输入缓冲区的大小，您必须#将YY\_BUF\_SIZE定义为您想要的任何字节数。我们不打算在不久的将来改变这一点，但如果我们添加了一个更健壮的内存管理API，我们将保留这样做的权利。

拒绝状态为64kb。只有在使用拒绝时才会分配。

其大小足够大，可以保持与输入缓冲区中的字符相同数量的状态。如果您覆盖了输入缓冲区的大小（通过YY\_BUF\_SIZE），那么您也会自动覆盖此缓冲区的大小。

100字节为起始条件堆栈10字节。

Flex为启动条件堆栈分配内存。这是用于推送启动状态的堆栈，即使用yy推送状态（）。如果有必要，它就会不断增长。由于状态只是整数，所以这个堆栈不会消耗太多内存。如果未指定%选项堆栈，则此堆栈不存在。您很少需要调优此缓冲区。这个堆栈的理想大小是预期的最大深度。当您调用yylex销毁（）时，此堆栈的内存会自动销毁。参见[选项堆栈]，第38页。

每个YY缓冲区状态为40个字节。

Flex为每个YY缓冲区状态分配内存。缓冲区状态本身大约为40字节，再加上一个额外的大字符缓冲区（如上所述）。初始缓冲区状态是在初始化期间创建的，每次调用yy都会创建缓冲区（）。您不能调优其大小，但您可以如上面所述调优字符缓冲区。您显式创建的任何缓冲区状态

1这里给出的数量是近似的，可能会由于主机体系结构、编译器配置或未来对flex的增强而变化。

通过调用yy创建缓冲区，（）不会被自动销毁。您必须调用yy删除缓冲区（）来释放内存。此规则的例外情况是，当您调用yylex销毁（）时，flex将自动删除当前缓冲区。如果您删除了当前的缓冲区，请确保将其设置为NULL。这样，flex就不会尝试第二次删除缓冲区（可能会使您的程序崩溃！）在撰写本文时，flex并没有为缓冲区状态提供一个可增长的堆栈。你必须自己来管理。请参见第11章[多个输入缓冲区]，第27页。

84字节用于重入扫描仪的肠道

当您调用yylex init（）时，Flex会为重入扫描器结构分配约84个字节。当用户调用yylex销毁（）时，它将被销毁。

## 覆盖默认的内存管理

Flex在需要分配或可用内存时调用yyalloc、yy实际loc和yyfree。默认情况下，这些函数分别是围绕标准C函数、malloc、realloc和自由功能的包装器。您可以通过告诉flex您将提供您自己的实现来覆盖默认实现。

要覆盖默认实现，您必须执行两件事：

1. 通过指定以下一个或多个选项来抑制默认实现：
   * %option noyyalloc
   * %option noyyrealloc
   * %option noyyfree.
2. 提供您自己的实现以下功能：2

//用于非重入扫描仪void（size\_t字节）；

空白（空白\* ptr，size\_t字节）；空白 免费（void \* ptr）；

//为一个可重入的扫描仪

（size\_t字节，size\_t扫描仪）；

（字空间，size\_t字节，扫描仪）；空白 （空白扫描仪）；

在下面的示例中，我们将覆盖所有三个内存例程。我们假设有一个带有垃圾收集的自定义分配器。为了使这个例子更有趣，我们将使用一个重入扫描器，通过yyextra传递一个指向自定义分配器的指针。

%{

#包括“some\_分配器.h”

%}

/\*抑制了默认的实现。\*/

2无需覆盖所有（或任何）内存管理例程。例如，您可以覆盖yy实际loc，但不能覆盖yyfree或yyalloc。

%option noyyalloc noyyrealloc noyyfree

选择重入百分比

/\*初始化分配器。\*/

%{

#定义YY\_EXTRA\_TYPE 结构分配器\*

#定义YY\_USER\_INIT yy额外的=分配器\_crexere创建（）；

%}

%%

.|\n ;

%%

/\*提供了我们自己的实现。\*/

（size\_t字节，size\_t扫描器）{返回分配器\_alloc（size\_t额外，字节）；

}

{返回allocator\_realloc（yyetray，字节）；

}

无空间（无空间，无空间扫描仪）{

/\*什么都不做——我们把它留给垃圾收集器。\*/

}

## 关于yytext和内存的说明

当flex找到匹配时，yytext指向输入缓冲区中匹配的第一个字符。该字符串本身是输入缓冲区的一部分，并不是单独分配的。下次调用yylex（）时，yytext的值将被覆盖。简而言之，yytext的值仅在匹配规则的操作中有效。

通常，您希望yytext的值持久化以供以后处理，即通过具有非零前视的解析器。为了保存yy文本，您必须使用（）（）或类似的函数复制它。但是这就带来了一些麻烦，因为您的解析器现在负责释放yytext的副本。如果您使用yacc或bison解析器（通常与flex一起使用），您会发现错误恢复机制可能会导致内存泄漏。

为了防止文本文本导致内存泄漏，您必须以某种方式跟踪内存。我们的经验表明，垃圾收集机制或在编写解析器时的聚合内存机制将为您节省很多痛苦。

# 序列化表

flex扫描器能够将DFA表保存到一个文件中，并在需要时在运行时加载它们。此特性的动机是为了减少运行时内存占用。传统上，这些表被编译为C数组的扫描器中，有时相当大。由于表被编译到扫描器中，因此永远无法释放表所使用的内存。这是浪费内存，特别是当应用程序使用多个扫描仪，但没有同时使用这些扫描仪时。

序列化特性允许在扫描开始之前，在运行时加载表。当扫描完成后，这些表格可能会被丢弃。

## 创建序列化表

您可以通过指定以下方式创建具有序列化表的扫描器：

%选项表-文件=FILE

或

表文件=文件

这些选项指示flex将DFA表保存到文件文件中。这些表将不会嵌入到生成的扫描仪中。扫描仪将不能独立工作。扫描程序将依赖于序列化的表。在运行时必须从此文件加载表，然后才能扫描任何内容。

如果没有指定-表文件的文件名，则表将被保存到

lex.yy.tables，其中“yy”是适当的前缀。

如果您的项目使用了几种不同的扫描程序，那么您可以将序列化的表连接到一个文件中，并且flex将找到正确的表集，使用扫描程序前缀作为查找键的一部分。以下是一个例子：

$flex--表-文件--前缀=cppcpp.l

$ flex表文件前缀=c c.l

$猫lex.cpp.tables lex.c.tables > 所有表

上面的例子创建了两个扫描仪，“cpp”和“c”。由于我们没有指定文件名，所以将表分别序列化为lex.c.tables和lex.cpp.tables。然后，我们将这两个文件连接到所有的.表中，我们将与我们的项目一起分发。在运行时，我们将打开该文件，并告诉flex从其中加载表。Flex将自动找到正确的表。（请参阅下一节）。

## 正在加载和卸载序列化表

如果您已经使用%选项表-文件构建了扫描程序，那么您必须在运行时加载扫描程序表。这可以通过以下功能来实现：

int yytables\_fload（文件\*fp[，扫描扫描仪]）

函数

在由fp指向的流中定位扫描程序表并加载它们。内存

这些表是通过yyalloc来分配的。您必须在第一次调用yylex之前调用此函数。参数扫描器只出现在重入扫描器中。此函数在成功时返回“0”(0)，或在错误时返回非零。

当调用yylex\_销毁时，加载的表不会自动销毁（卸载）。原因是，您可以创建多个相同类型的扫描仪（在一个可重入的扫描仪中），每个扫描仪都需要访问这些表。为了避免严重的内存泄漏，您必须调用以下函数：

int yytables\_destroy（[扫描扫描仪]）

函数

卸载扫描仪表。必须在进行其他数据扫描之前再次加载这些表。参数扫描器只出现在重入扫描器中。此函数在成功时返回“0”(0)，或在错误时返回非零。

函数yytables\_fload和yytables\_destroy不是线程安全的。您必须确保在任何线程调用yylex之前，在线程程序中只调用了一次（对于每种扫描程序类型）。加载表后，永远不会写入它们，此后也不需要线程保护——直到您销毁它们。

## 表文件格式

本节定义了序列化的弹性体表的文件格式。

表格式允许指定一个或多个表集，其中每个集合对应于一个给定的扫描器。扫描仪按名称进行索引，如下所述。文件格式如下：

表SET 1

+-------------------------------+

标题 | uint32 th\_magic ; |

| uint32 th\_hsize; |

| uint32 th\_ssize; |

| uint16 th\_flags ; |

|字符 th\_version []; |

|字符 th\_name []; |

| uint8 th\_pad64[]; |

+-------------------------------+

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表 | 1 | | | uint16 | td\_id; | | |
|  |  | | | uint16 | td\_flags ; | | |
|  |  | | | uint32 | td\_hilen; | | |
|  |  | | | uint32 | td\_lolen; | | |
|  |  | | | 空的 | td\_data []; | | |
|  |  | | | uint8 | td\_pad64[]; | | |

+-------------------------------+

表2| |

. . .

. . .

. . .

. . .

表n| |

+-------------------------------+表SET 2

.

.

.

表SET N

上面的图表显示了一组完整的表，由一个头和多个单独的表组成。此外，多个完整的集可能出现在同一个文件中，每个集都有自己的头和表。这些集合在文件中是连续的。知道接下来是否有另一组集合的唯一方法是检查接下来的四个字节的神奇数字（或检查EOF）。标题和表格部分被填充到64位的边界。下面我们将详细描述每个字段。这种格式并没有指定扫描器将如何扩展给定的数据，即，数据可以被序列化为int8，而是在运行时扩展到一个int32数组。这是为了尽可能减少序列化数据的大小。*请记住，所有的整数值都是按网络字节的顺序排列的。*

表标题的字段：

th\_magic 魔术号码，总是0xF13C57B1。

th\_hsize 这个整个报头的大小，以字节为单位，包括所有字段和任何填充。th\_ssize 这整个集合的大小，以字节为单位，包括报头、所有表和任何填充。th\_flags 此表集的位标志。当前未使用。

th\_version []

以以空字符串格式结尾的Flex版本。e.g., '2.5.13a'.这是用于创建序列化表的flex的版本。

th\_name []

th\_pad64[]

包含此表集的名称。默认值是“yytables”，并带有相应的前缀，例如，“可移动参数”。必须为NULL终止。

零或更多NULL字节，填充整个头到从头开始计算的下一个64位边界。

表的字段：

td\_id 指定表标识符。可能的值包括：

YYTD\_ID\_ACCEPT (0x01)

yy\_accept

YYTD\_ID\_BASE (0x02)

yy\_base

YYTD\_ID\_CHK (0x03)

yy\_chk

YYTD\_ID\_DEF (0x04)

yy\_def

YYTD\_ID\_EC (0x05)

yy\_ec

YYTD\_ID\_META (0x06)

yy\_meta

YYTD\_ID\_NUL\_TRANS (0x07)

yy\_NUL\_trans

YYTD\_ID\_NXT (0x08)

. yy\_nxt这个数组可以是二维的。请参阅td\_hilen

下面的字段。

YYTD\_ID\_RULE\_CAN\_MATCH\_EOL (0x09)

yy\_rule\_can\_match\_eol

YYTD\_ID\_START\_STATE\_LIST (0x0A)

yy\_start\_state\_list这个数组被专门处理过，因为它是一个指向结构的指针数组。. 请参见下面的td\_flags字段。

YYTD\_ID\_TRANSITION (0x0B)

yy\_transition这个数组，因为它是一个结构数组。. 请参见下面的td\_lolen字段。

YYTD\_ID\_ACCLIST (0x0C)

yy\_acclist

td\_flags Bit标志描述了如何解释td\_data中的数据。默认情况下，数据数组是一维的，但也可以是在td\_hilen字段中指定的二维的。

YYTD\_DATA8 (0x01)

该数据被序列化为一个具有int8类型的数组。

YYTD\_DATA16 (0x02)

该数据被序列化为一个具有int16类型的数组。

YYTD\_DATA32 (0x04)

该数据被序列化为一个具有int32类型的数组。

YYTD\_PTRANS (0x08)

该数据是扩展的yy\_transition数组中的条目的索引列表。每个索引都应该扩展到一个指向yy\_transition数组中相应条目的指针。我们相信yy\_transition数组已经被看到了。

YYTD\_STRUCT (0x10)

数据是一个yy转换信息结构的列表，每个信息结构由两个整数组成。结构元素之间或结构之间没有填充。每个成员的类型由YYTD\_DATA\*位决定。

td\_hilen如果td\_hilen是非零的，则数据是一个二维数组。否则，数据是一个一维数组。td\_hilen包含高维数组中的元素数量，而td\_lolen包含最低维数中的元素数量。

从概念上讲，td\_data要么是体类型td\_data[td\_lolen]，要么是体类型td\_data[td\_hilen][td\_lolen]，其中体类型由td\_flags字段指定。在这种情况下，td\_lolen和td\_hilen都有可能为零

td\_data是一个零长度的数组，没有加载数据，也就是说，简单地跳过这个表。Flex当前不生成零长度的表。

td\_lolen指定最低维数组中的元素数。如果这是一个一维数组，那么它仅仅是这个数组中元素的数量。元素的大小由td\_flags字段决定。

td\_data []

td\_pad64[]

表数据。这个数组可以是一个或一个二维数组，类型为int8、int16、int32、结构yy\_trans\_info或结构yy\_trans\_info\*，这取决于td\_flags、td\_hilen和td\_lolen字段中的值。

零或更多NULL字节，将整个表填充到从此表开始计算的下一个64位边界。

# 诊断学

以下是灵活诊断消息的列表：

* “警告，规则不能匹配”表示给定的规则不能匹配，因为它遵循的其他规则将始终与它相同的文本匹配。例如，在下面的“foo”中不能匹配，因为它出现在标识符“捕获所有”规则之后：

[a-z]+ got\_标识符（）；foo got\_foo() ;

在扫描仪中使用拒绝处理会抑制此警告。

* “警告，-选项，但默认规则可以匹配”意味着可能（可能只有在特定的开始条件下）默认规则（匹配任何单个字符）是唯一匹配特定输入的规则。既然是“-s”，那么这可能不是有意的。
* 拒绝使用但未检测到未定义或yymore\_used\_but\_not\_detected未定义。这些错误可能会在编译时发生。它们表明扫描仪使用了拒绝或yymore（），但是flex没有注意到这一事实，这意味着flex扫描了前两个部分，寻找这些操作的出现，但没有找到任何操作，但你偷偷插入了一些（例如，通过#包含文件）。使用%选项拒绝或%选项yymore来表示您确实使用了这些功能。
* “flex扫描仪堵塞”。用“-s”编译的扫描程序遇到了一个与其任何规则都不匹配的输入字符串。此错误也可能由于内部问题而发生。
* “令牌太大，超过了YYLMAX”。您的扫描程序使用%数组，其中一个规则匹配的字符串超过YYLMAX常量（默认为8K字节）。您可以通过在flex输入的定义部分中的#定义YYLMAX来增加该值。
* “扫描仪需要-8标志才能使用字符“x””。您的扫描仪规范包括识别8位字符“x”，并且您没有指定-8标志，并且您的扫描仪默认为7位，因为您使用了“-Cf”或“-CF”表压缩选项。详见关于“-7”标志的讨论，第16章[扫描仪选项]，第35页。
* “flex扫描仪后推溢出”。您使用了unput（）来回推太多的文本，以至于扫描器的缓冲区无法在yy文本中同时保存回推文本和当前令牌。理想情况下，在这种情况下，扫描器应该动态地调整缓冲区的大小，但目前它没有。
* “输入缓冲区溢出，无法放大缓冲区，因为扫描程序使用拒绝”。扫描仪正在处理匹配一个非常大的令牌，需要扩展输入缓冲区。这并不适用于使用拒绝处理的扫描仪。
* “致命的flex扫描器内部错误-缓冲区结束丢失”。这可能发生在扫描仪中，该扫描仪在跳远跳出（或超过）扫描仪的激活帧后重新进入。在重新进入扫描仪之前，请使用：

yyrestart ( yyin );

或者，如上所述，切换到使用C-++扫描仪类。

* “在<>构造中有太多的启动条件！”“你在a

<>构造比存在的要多（所以您必须至少列出其中一个两次）。

# 限制

某些尾随上下文模式不能正确匹配，并不能生成警告消息（“危险的尾随上下文”）。在这些模式中，规则的第一部分的结尾与第二部分的开始相匹配，例如“zx\*/xy\*”，其中“x\*”与后面上下文开头的“x”相匹配。（请注意，POSIX草案指出，与这些模式匹配的文本未定义。）对于一些后面的上下文规则，实际上是固定长度的部分不会被识别出来，从而导致上述性能损失。特别是，使用“|”或“{n}”（如“foo{3}”）的部分总是被认为是可变的。将尾随上下文与特殊的“|”操作相结合，可能会导致固定的尾随上下文被转换为更昂贵的变量尾随上下文。例如，在以下内容中：

%%

abc |

xyz/def

使用unput（）会使yytext和yyleng无效，除非使用了%array指令或“-l”选项。nul的模式匹配比匹配其他字符要慢得多。输入缓冲区的动态大小调整速度很慢，因为它需要重新扫描迄今为止由当前（通常是巨大的）令牌匹配的所有文本。由于输入的缓冲和预读，您不能混合对<stdio.h>例程的调用，例如，获取（），并期望它工作。转而调用输入（）。由“-v”标志列出的表条目总数排除了确定已匹配的规则所需的表条目数量。如果扫描器不使用拒绝，则条目数等于DFA状态的数，如果使用拒绝，则略大于状态数。拒绝不能与“-f”或“-F”选项一起使用。

灵活的内部算法需要文档化。

# 附加阅读

您可能希望了解更多关于以下程序的内容：

* lex
* yacc
* sed
* awk

以下书籍可能包含让您感兴趣的材料：

约翰·莱文，托尼·梅森，道格·布朗，莱克斯& Yacc，奥莱利和同事。一定要买到第二版。

M.E.Lesk和E。施密特，LEX-词汇分析仪发电机

*阿尔弗雷德阿霍，拉维塞蒂和杰弗里乌尔曼，编译者：原理、技术和工具，艾迪生-韦斯利（1986）。*描述了由flex（确定性有限自动机）所使用的模式匹配技术。

# 码头交货

弹性维护者不时地会收到某些问题。我们不再重复对已被充分理解的问题的答案，而是在这里发布它们。

flex是什么时候出生的？

1982年，弗恩·帕克森从杰夫·波斯坎泽手中接管了软件工具词汇项目。在那时，它是用拉特福写的。大约在1987年左右，帕克森把它翻译成C语言，一个传说诞生了：-)。

如何在c风格的引号字符串中展开反斜杠转义序列？

扫描引号字符串时的关键是，如果允许嵌入转义序列和换行，您不能（轻松地）编写一个能够精确地与字符串匹配的单一规则。如果您试图用单个规则匹配字符串，那么您将不得不重新扫描该字符串以找到任何转义序列。

相反，您可以使用独占的开始条件和一组规则，一个用于匹配非转义文本，一个用于匹配单个转义，一个用于匹配嵌入的换行符，另一个用于识别字符串的末尾。然后，这些规则都面临着将其中间结果放在哪里的问题。最好的解决方案是让规则将yytext值附加到“字符串文字”缓冲区的末尾。像转义匹配器这样的规则将在缓冲区中附加转义序列的含义，而不是文本文本。这样，yytext根本不需要修改。.

如果它不与ANSI兼容，为什么flex扫描仪会调用fileno呢？

Flex扫描器调用fileno（），以获得与yyin对应的文件描述符。文件描述符可以传递给isatty（）或读取（），具体程度取决于您指定的%选项。如果您的系统不支持文件（），要取消读（）调用，请不要指定%选项读。要取消等位（）调用，必须指定%非交互式选项或%非交互式选项之一。

flex是否支持递归模式定义？

e.g.,

%%

块 "{"({block}|{statement})\*"}"

不您不能有递归式的定义。一般来说，正则表达式的模式匹配能力(因此柔性扫描仪也是有限的。特别是，正则表达式不能在任意程度上“平衡”括号。例如，不可能编写一个匹配包含相同数量的s{‘s和s}的所有字符串的正则表达式。为了实现更强大的模式匹配，您需要一个解析器，比如GNU bison。

在使用flex时，我如何跳过大块的输入（数十兆字节）？

使用fseek（）（或lseek（））来定位yyin，然后调用yy重新启动（）。

Flex没有按照我定义的顺序匹配我的模式。

flex选择匹配最多文本的规则(i。e., 可能最长的输入字符串)。这是因为flex使用了一种完全不同的匹配技术（“确定性有限自动机”），它实际上是同时、并行地完成所有的匹配。 （看起来不可能，但一旦你理解了这些原理，这实际上是一个相当简单的技巧。）

这种并行匹配的一个副作用是，当输入匹配多个规则时，flex扫描仪会选择匹配最多文本的规则。这一点在手册中，在见第14页的第7章[匹配]一节中有进一步的说明。

如果您希望flex选择一个较短的匹配，那么您可以通过扩展您的短规则来匹配更多的文本来解决这个行为，然后返回额外的：

数据\_。\* yyless（5）；开始封锁状态；

另一个修复方法是使第二块规则只在<块状态>期间激活

启动条件，并通过声明%x为启动条件而不是

%s.

最后一个修复方法是更改输入语言，通过添加与‘data\_规则不匹配的歧义，或者从标识符规则中删除字符（如’data‘\_”），从而删除’data\_‘的歧义。（当然，您可能也不能选择更改输入语言。）

我的行为执行得不正常，或者有时根本不正常。

最有可能的情况是，你（错误地）将动作块的开口“{放在与规则不同的行上，例如，

^(foo|bar)

{ <<<---错了！

}

flex要求与规则相关联的操作的开头“{”与规则从同一行开始。相反，您需要编写以下规则：

^(foo|bar) { //正确！

}

如何同时将多个输入源输入到同一个扫描仪中？

如果..

* 你的扫描仪没有回溯（使用flex的“-b”标志进行验证），
* 交互式运行扫描仪（“-I”选项；默认，除非使用特殊的表压缩选项），
* 你通过重新定义YY\_INPUT来一次输入一个字符，

然后，每次匹配一个令牌，它就会耗尽它的输入缓冲区（因为扫描器没有回溯）。这意味着您可以安全地在该点使用选择（），并且如果选择（）表示有可用的数据，则只为另一个令牌调用yylex（）。

也就是说，将选择的（）从输入函数移到它确定是否为下一个标记调用yylex（）的位置。

使用这种方法，如果您的输入可以分段到达，您仍然会有问题；选择（）可以通知您令牌的开头是可用的，您调用yylex（）来获得它，但它最终阻止等待令牌中后面的字符。

这里有另一种方法：将输入多路复用移动到YY\_INPUT内部。也就是说，无论何时调用YY\_INPUT，它都会选择（）来查看输入的可用位置。如果扫描器有输入，它将读取并返回下一个字节。如果输入可以从另一个源获得，那么它将调用负责从该源读取的任何函数。（如果没有可用的输入，它会阻塞，直到某些输入可用。）我在我编写的一个解释器中使用了这种技术，它都使用弹性扫描器读取键盘输入，并从套接字读取IPC流量，而且它工作得很好。

我可以构建使用相同的输入文件来工作的嵌套解析器吗？

如果没有一些额外的努力，这一点是行不通的。原因是flex块缓冲了它从yyin读取的输入。这意味着，当调用“最外层”的yylex（）时，将自动占用其中可用的前8K个输入，而随后对其他yylex（）的调用将不会看到该输入。您可能会试图通过重新定义YY\_INPUT以只返回少量文本来解决这个问题，但结果是这种方法相当困难。相反，最好的解决方案是将您所有的扫描仪合并到一个大型扫描仪中，对每个扫描仪使用不同的唯一启动条件。

我如何才能只在文件的末尾匹配文本？

没有办法编写一个规则是“匹配此文本，但只有当它出现在文件的末尾”的规则。不过，如果你碰巧周围有一个你不允许输入的角色，你也可以假装它。然后您重新定义YY\_INPUT来调用您自己的例程，如果它看到一个“EOF”，则首先返回魔法字符（并且记得下次调用它时返回一个真正的EOF）。然后你可以写：

<COMMENT>(.|\n)\*{EOF\_CHAR} /\*在EOF \*/上看到了一些评论

如何跨起始条件边界进行拒绝级联？

你可以这样做。假设您有一个开始条件“a”，在用完“<A>”中所有可能的匹配之后，您想尝试“<初始>”中的匹配。然后你可以使用以下内容：

%x A

%%

<A>rule\_that\_is\_long ...；拒绝；

<A>rule ...；拒绝；/\*较短的规则\*/

<A>etc.

...

<A>.|\n {

/\*<A>中最短和最后一条规则，所以

* 级联反应最终会
* 最终符合这个规则。 我们想要
* 到现在切换到初始状态
* 然后尝试从那里进行匹配。

\*/

yyless (0); /\*放回匹配的文本\*/开始（首字母）；

}

为什么我不能使用快速或完整的表格与交互模式？

flex做的一个假设是，交互式应用程序本身就很慢（毕竟它们是在等待人类）。它与扫描仪如何检测到它必须完成对令牌的扫描有关。对于交互式扫描器，在扫描每个字符后，当前状态将在表中查找（基本上），以查看是否有另一个输入字符可能延长匹配的长度。如果没有，扫描仪将停止。对于非交互式扫描器，令牌结束测试要简单得多，基本上是一个与0相比较的测试，因此没有内存总线周期。由于测试发生在最内层的扫描循环中，人们希望使它尽可能快地进行。

不过，允许用户选择在这一领域的一些性能来获得相应的灵活性似乎是合理的。不过，快速扫描仪可能不支持交互式选项。

-f或-f比-C快多少？

速度要快得多（2-3倍）。

如果我有一个简单的语法，那么我就不能用flex来解析它吗？

你的语法是递归的吗？这几乎总是一个表明你最好使用解析器/扫描仪，而不是仅仅尝试单独使用扫描仪。

为什么不重启（），将开始状态设置回初始状态？

有两个原因。首先，可能会有一些程序依赖于启动状态，而不会在文件更改之间进行更改。第二个问题是，从flex2.4版本开始，不再需要使用yy重启（），因此解决这个问题并不能解决更一般的问题。

我如何匹配c风格的评论？

你可能会忍不住去尝试这样的东西：

"/\*".\*"\*/" //错了！

或者更糟的是：

"/\*"(.|\n)"\*/" //错了！

上述规则会消耗太多的输入，并在诸如：

/\*上的一个评论\*/做\_my\_的事情（“oops\*/”）；

这里有一种方法，可以让您跟踪线路信息：

<INITIAL>{

"/\*" 开始（在注释中）；

}

<IN\_COMMENT>{

"\*/" 开始（初始）；

[^\*\n]+ //吃评论在块“\*” //吃掉了孤星

\n yylineno++;

}

这个并没有像我预期的那样工作。

这里有一些使用技巧。’:

* 一个常见的错误是将分组括号放在运算符后面，当你真正打算将括号放在运算符之前时，例如，你可能想要这个（foo|bar）+而不是这个（foo|bar+）。

第一个模式与单词“foo”或“bar”匹配任意次数，例如，它与文本“上脚脚”匹配。第二个模式匹配单个foo实例或单个bar实例，后面跟着一个或多个‘r’s，例如，它匹配文本ba rrrr。

* A ‘.“[]”的意思只是字面意思。‘（句号），而不是“除换行符以外的任何字符”。
* 记住这一点。“匹配除“\n”（和“EOF”）以外的任何字符。如果您真的想匹配任何字符，包括换行符，那么请使用（.|\n）注意，正则表达式（.|\n）+将匹配您的整个输入！
* 最后，如果你想匹配一个文字”。‘（一个期间），然后使用“[.]”或

我能得到另一种格式的flex手册吗？

这个灵活的源代码发行版包括了一个文本信息手册。你可以自由地转换，文本信息到任何你想要的格式。文本信息软件包包括用于转换为多种格式的工具。

是否存在一种“更快”的NDFA->DFA算法？

没有办法绕过潜在的指数运行时间——仅仅是枚举所有的DFA状态就可能需要你的指数运行时间。但在实践中，运行时间更接近线性的，有时是二次的。

flex如何如此快地编译DFA？

flex使用了两个大的速度胜利：

1. 它分析输入规则，为那些总是进行相同转换的字符构造等价类。然后，它使用转换的等价类而不是字符来重写NFA。这大大减少了NFA->DFA的计算时间，以致于对于未压缩的DFA表，DFA生成通常在写入表时受到I/O约束。
2. 它为以前计算的DFA状态维护哈希值，因此可以通过首先比较哈希值来快速地测试一个新构造的DFA状态是否等同于以前构造的状态。

我如何使用8192条以上的规则？

Flex编译时，每个扫描仪的上限为8192条规则。如果您的扫描器中需要超过8192条规则，则必须在flexdef.h中使用以下更改重新编译flex：

< #定义YY\_TRAILING\_MASK 0x2000

< #定义YY\_TRAILING\_HEAD\_MASK 0x4000

--

> #定义YY\_TRAILING\_MASK 0x200000000

> #定义YY\_TRAILING\_HEAD\_MASK 0x40000000

只要你的C编译器使用32位整数，这应该可以工作。但是您可能想考虑使用如此多的规则是否是解决问题的最佳方法。

以下内容也可能相关：

幸运的是，您应该能够增加flexdef.h中的定义：

/\*标记了对总是阻塞\*/的状态的引用

#定义MAXIMUM\_MNS 31999

#定义BAD\_SUBSCRIPT -32767

重新编译所有内容，它都会工作。Flex只内置了这些类似于16位的值，因为很久以前它是在一台具有16位ints的机器上开发出来的。我过去曾给过别人这个建议，但还没有收到他们的回复。

我如何在扫描过程中放弃一个文件并切换到一个新文件？

只需调用yyreat（新文件）。如果您想要“重新启动”，请确保重置启动状态，因为yyestit不会将启动状态重置为初始状态。

如何只在初始化期间执行代码（仅在第一次扫描之前）执行代码？

您可以通过定义宏YY\_USER\_INIT来指定一个初始操作（尽管请注意，在执行此宏时，yyout可能不可用）。或者你可以添加到你的规则部分的开头：

%%

/\*必须缩进！\*/静态int did\_init = 0；

如果did\_init ){ do\_my\_init() ;

did\_init = 1;

}

我如何在终止时执行代码？

您可以为<<EOF>>规则指定一个操作。

我还能在哪里找到帮助呢？

[你可以在http://flex.sourceforge.net/上找到flex的主页。](http://flex.sourceforge.net/)有关flex邮件列表的详细信息，也请参阅该页面。

我可以在文件的“规则”部分中包含注释吗？

是的，只要有任何你想去的地方。有关特定的语法，请参见本手册。

我得到了一个关于未定义的yywrap（）的错误。

您必须提供您自己的yywrap（）函数，或链接到libfl。a（它提供了一个），或使用

%option noyywrap

在你的源代码中说你不想要一个yywrap（）函数。

我如何在运行时更改匹配的模式？

你不能，当flex构建扫描仪时，它被编译成一个静态表。

如何在输入中展开宏？

处理这个问题的最佳方法是在更高的级别上，例如，在解析器中。

但是，您可以使用多个输入缓冲区来执行此操作。

%%

macro/[a-z]+ {

/\*看到了宏“宏”，后面是额外的东西。\*/main\_buffker=YY\_CURRENT\_BUFFER；

展开\_缓冲区=yy\_scan\_string（展开（yytext））；yy\_switch\_to\_buffer（展开\_缓冲区）；

}

<<EOF>> {

如果（扩展缓冲区）

{

//我们在做一个扩张，回到了哪里

//我们。yy\_switch\_to\_buffer（main\_缓冲区）；yy\_delete\_buffer（扩展\_缓冲区）；扩展\_缓冲区=0；

}

其他yydent（）；

}

您可能希望有一堆扩展缓冲区来允许嵌套的宏。从上面来看，希望这个想法是清晰的。

我如何构建一个双通扫描仪？

一种方法是过滤第一次传递到临时文件的过程，然后在第二次传递时处理该临时文件。由于所有磁盘I/O，您可能会看到性能命中。

当您需要像这样向前看时，它几乎总是意味着正确的解决方案是构建整个输入的解析树，然后在解析之后处理它，以生成输出。从某种意义上说，这是一种两次通过的方法，一次通过文本，一次通过解析树，但后者的性能命中通常要小一个数量级，因为一切都已经以二进制格式分类，并驻留在内存中。

如何匹配前面规则中不匹配的任何字符串？

分配优先级的一种方法是将更具体的规则放在首位。如果两个规则匹配相同的输入（相同的字符序列），那么在flex输入中列出的第一个规则获胜，e。g.,

%%

foo[a-zA-Z\_]+ 返回FOO\_ID；bar[a-zA-Z\_]+ 返回BAR\_ID；

[a-zA-Z\_]+ 返回GENERIC\_ID

注意规则[a-zA-Z\_]+必须在其他规则之后。它将与更具体的规则匹配相同数量的文本，在这种情况下，flex扫描器将选择扫描器中列出的第一个规则作为要匹配的规则。

我试图从使用yy&T和yyssptr的lex移植代码。

这些都是指向AT&T扫描仪的输入缓冲区的内部变量。我想它们在输入（）的用户版本和未输入（）函数的用户版本中被操纵了。如果是这样，您需要做的是分析这些函数，以找出它们在做什么，然后用适当的YY\_INPUT定义替换输入（）。您不需要（也不需要）替换flex的unput（）函数。

有没有一种方法可以让flex对待NULL就像一个普通角色一样？

是的，“\0”和“\x00”都可以做到。也许你有一个古老版本的flex。最新的版本是2.6.4版本。

当flex不能匹配输入时，它会说“flex扫描仪卡住”。

您需要添加一个与其他文本不匹配的规则，例如，

%option yylineno

%%

这里有一堆规则

. printf（“在行%d\n”，yytext，yylineno）；

有关详细信息，请参见%默认选项。

为什么flex没有像perl那样的非贪婪操作符呢？

DFA可以通过在第一次进入接受状态时停止来进行非贪婪匹配，而不是消耗输入，直到它确定不可能进一步匹配（“阻塞”状态）。这实际上比最长的最左匹配（flex所做的）更容易实现。

但它也远不如最左边最长的比赛有用。一般来说，当您发现自己希望进行非贪婪匹配时，这通常表明您正在试图让扫描器做一些解析。这通常是一种错误的方法，因为它缺乏做一件体面工作的能力。更好的方法是引入一个单独的解析器，或者使用（排他的）启动条件将扫描器分割为多个扫描器。

一旦您看到了“开始”，您可能会有一个单独的开始状态。在这种状态下，您可能有一个匹配“EnD”（将您踢出状态），也许还有“（.|\n）”来在块中获得单个字符...

这种方法也具有更好的错误报告属性。

内存泄漏-由malloc分配的16386个字节。

更新2002：在flex版本2.5.9中，这次泄漏意味着您没有调用

yylex\_destroy() .如果您使用的是早期版本的flex，那么请继续阅读。

泄漏量约为16426字节。也就是说，读取缓冲区（8192 \* 2 + 2），结构yy\_buffer\_state约40（取决于对齐）。泄漏仅在非重入C扫描仪中（不在重入扫描仪中，不在C++扫描仪中）。由于flex不知道您何时完成，因此缓冲区永远不会被释放。

但是，由于无论调用yylex（）多次，泄漏都不会相乘。

如果您想在完全完成扫描时收回内存，那么您可以尝试以下：

/\*仅适用于非重入式C扫描仪。\*/ yy\_delete\_buffer ( YY\_CURRENT\_BUFFER ); yy\_init = 1;

注意： yy\_init是一个“内部变量”，在这种情况下还没有进行过测试。其他一些全球国家可能也需要重置。

如何跟踪lseek（）的字节偏移量？

> 我们认为这个数字是有可能通过

> 对以下表达式的评估：

>

> 寻找位置=（无缓冲区）\*YY\_READ\_BUF\_SIZE+yy\_c\_buf\_p-YY\_CURRENT\_BUFFER->yy

虽然这是一个正确的想法，但它有两个问题。首先，在调用YY\_INPUT时，flex可能会请求少于YY\_READ\_BUF\_SIZE（或者即使请求了YY\_READ\_BUF\_SIZE字节，输入源返回的量也会更少）。第二个问题是，当重新填充其内部缓冲区时，flex保留前一个缓冲区中的一些字符（因为通常它在匹配的中间，并且这些字符需要为匹配构造yytext）。正因为如此，yy\_c\_buf\_p

- YY\_CURRENT\_BUFFER->yy\_ch\_buf不会完全是已经从当前缓冲区中读取的字符数。

另一种解决方案是计算您自开始扫描后所匹配的字符数。这可以通过使用YY\_USER\_ACTION来完成。例如，

#定义YY\_USER\_ACTION num\_chars+=+=；

（如果您使用更多（）、无（）、less（）、未输入（）或输入（），则需要小心更新簿记。）

我如何在C-++扫描仪中使用我自己的I/O类？

当flex C++扫描类重写最终发生时，这类事情应该会变得容易得多。

您可以通过传递各种函数（如Lexer输入（）和L输出（）），然后秘密地处理您自己的I/O类（即将它们存储在特殊成员变量中）来实现这一点。这是有效的，因为关于词汇的唯一的假设是，它们最终被传递给词汇输入（）和词汇输出，然后它们对它们做任何必要的事情。

我如何跳过尽可能多的字符吗？

如何在不干扰其他模式的情况下跳过尽可能多的字符？

在下面的示例中，我们希望跳过字符，直到我们看到短语“endskip”。

以下内容将不能正确工作（您知道为什么不能正常工作吗？）

/\*扫描仪\*/不正确

%x SKIP

%%

<INITIAL>startskip 开始（跳过）；

...

<SKIP>"endskip" 开始（初始）；

<SKIP>.\* ;

问题是这个模式。会吃掉“结束”这个词。“最简单（但速度缓慢）的解决方法是：

<SKIP>"endskip" 开始（初始）；

<SKIP>. ;

修复包括使第二条规则匹配更多，而不使它匹配“端跳”

加上其他的东西。例如：

<SKIP>"endskip" 开始（初始）；

<SKIP>[^e]+ ;

<SKIP>. ；/\*所以你吃掉了e的，太\*/

deleteme00

问题：

flex是什么时候出生的？

弗恩·帕克森接管了

1982年，杰夫·波斯坎泽提出的软件工具项目。 在那时，它是用拉特福写的。 大约在1987年左右，帕克森把它翻译成C语言，一个传说诞生了：-)。

某些等效的模式比其他模式快吗？

[致： Adoram Rogel <adoram@orna.hybridge.com>主题：回复： Flex 2.5.2性能问题](mailto:adoram@orna.hybridge.com)

回复：9月96日18日星期三美国东部时间11：12：17。日期：太平洋时间9月9月18日星期三10：51：02

来自： Vern Paxson <vern>

[注意，最近的flex版本是2.5.4，你可以从ftp.ee.lbl.gov中获得。 它修复了在2.5.2和2.5.3以上的bug。]

> 1.使用模式

> （Ff）（oot）？）（Nn）（ote）？（\.）？

> 代替…

> (((F|f)oot (N|n)ote)|((N|n)ote)|((N|n)\.)|((F|f)(N|n)(\.)))

> （在一个非常复杂的flex程序中）导致程序从

> 300K+/min至100K/min（没有做其他变化）。

这两者是不相等的。 例如，第一个可以匹配“脚注”。但第二个问题只能匹配“脚注”。 这几乎肯定是这种差异的原因——较慢的扫描仪运行匹配了更多的令牌，并且/或不得不做更多的备份。

> 2.这两种方法中哪一种更好：还是（F|f）较好？

从性能的角度来看，它们是等价的(模量可能是很小的影响，比如内存缓存命中率；以及尾随的存在

上下文，见下文）。 从空间的角度来看，第一个要稍微更好一些。

> 3.我有一个图案，看起来是这样的：

> 补丁{p1}|{p2}|{p3}|...|{p50} （50种模式ORd）

>

> 运行另一个复杂的程序，包括以下规则：

> <snext>{and}/{no4}{bb}{pats}

>

> 让我觉得“太复杂了——超过32000个州”。

从这个例子中，我无法判断后面的上下文是可变长度的还是固定长度的（如果{和}是固定长度的，则可以是后者）。 如果它是可变的长度，而flex -p会告诉你，那么这反映了一个基本的性能问题，如果你可以通过重组扫描仪来消除它，你将看到显著的改进。

> 所以我把，...，分成，...，，，...，每个都由大约

> 10个模式，并将规则更改为5个规则。

> 这确实编译了，但这里的经验法则是什么？

规则是避免尾随上下文，其中对于a/b，“a”模式或“b”模式都有固定的长度。 使用

“|”操作符自动使模式可变长度，因此在这种情况下，“[Ff]oot”优于“（f|f）oot”。

> 4.我改变了一个类似于这样的规则：

> <snext8>{和}}[^A-Za-z]{开始。

>

> 至下一个2条规则：

> <snext8>{和}{bb}/{罗马}

> <snext8>{and}{bb}/{ROMAN} {开始..。

>

> 再一次，我理解使用[^…]会导致巨大的性能损失

实际上，它不会造成任何形式的性能损失。 正则表达式是一个令人惊讶的事实，它们总是在线性时间内匹配，而不管它们有多复杂。

> 但是关于它有什么具体的规定吗？

请参阅手册页的“性能考虑事项”部分，以及MISC/fastwc/中的示例。

本国的

支持你是什么大事吗？

[致： Adoram Rogel <adoram@hybridge.com>主题：回复： Flex 2.5.2性能问题](mailto:adoram@hybridge.com)

回复：您的留言，9月9月19日星期四美国东部时间10：16：04。日期：96年9月19日星期四09：58：00

来自： Vern Paxson <vern>

>上有很多关于备份问题的内容。

>，我相信这是我最大的问题，我会努力改进的

它

由于您有变量的尾随上下文，所以这是一个更大的性能问题。 修复它通常比修复备份更容易，在一个复杂的扫描仪（你的似乎符合要求），这可能是非常难以正确完成的。

你也没有提到你在扫描仪上使用了什么标志。

-f有很大的速度差异，-Cfe买你几乎同样的速度，但产生的扫描仪要小得多。

>我在{和}和{pats}中有一个|运算符，所以它们都是可变的

长度

-p应该报告的。

>把其中一个更改为固定长度就足够了吗？是

>是否有可能改变32,000个州的限制？

是我已经附加了一些说明。然而，在你做出这个改变之前，你应该考虑是否有方法从根本上简化你的扫描仪——这些当然是更好的！

本国的

要增加32K限制（在具有32位整数的机器上），您可以在flexdef.h中增加以下内容的大小：

/\*标记了对总是阻塞\*/的状态的引用

#定义MAXIMUM\_MNS 31999

#定义BAD\_SUBSCRIPT -32767

定义MAX\_SHORT 32700

在每个数字后面加一个0或两个就可以了。

我可以伪造多字节字符支持吗？

[收件人：Heeman\_Lee@hp.com](mailto:Heeman_Lee@hp.com)

flex多字节支持吗？

回复：你的留言，1996年10月03日星期四17：24：04 PDT。日期：1996年10月4日星期五11：42：18 PDT

来自： Vern Paxson <vern>

> 我假设只要我的\*。l文件定义了

> 预期字符代码值的范围（八进制格式），flex将

> 扫描文件并正确读取多字节字符。但我没有

> 对这一假设的信心。

你缺乏信心是合理的——这是行不通的。

Flex有一个广泛的假设，即输入被处理

一次一个字节。 解决这个问题在待办事项清单上，但有到，所以它不会很快发生。 在此期间，我能建议的最好的方法（除非你想自己尝试修复它）是用字节对来编写规则，使用第一节中的定义：

X \xfe\xc2

...

%%

foo{X}bar found\_foo\_fe\_c2\_bar（）；等等。 这绝对是一种痛苦，但很抱歉。

顺便说一下，你给我使用的电子邮件地址是古老的，这表明你有一个非常旧的flex版本。 你可以从ftp.ee.lbl.gov获得最新的，2.5.4。

deleteme01

本国的

[收件人：moleary@primus.com](mailto:moleary@primus.com)

主题：回复： Flex / Unicode兼容性问题

回复：您的留言，1996年10月22日星期二10：15：42 PDT。日期：1996年10月22日星期二11：06：13 PDT

来自： Vern Paxson <vern>

不幸的是，flex目前有一个广泛的假设，即字符一次处理8位。 我看不出什么容易

解决这个问题（除了用双字符来编写规则之外，这很痛苦）。 我也不知道有一个更广泛的词汇，尽管你可以尝试浏览9计划的东西，因为我知道它是一个单码系统，还有PCCT工具包(尝试搜索说Alta Vista的“普渡编译器构建”

工具包”)。

修复flex以处理更广泛的字符是在长期的任务列表中。但由于flex现在是一个严格的业余时间项目，这可能在很长一段时间内不会发生，除非别人先做。

本国的

你能讨论一下一些弹性结构的内部结构吗？

[致： Johan Linde <jl@theophys.kth.se>主题：回复：flex的翻译](mailto:jl@theophys.kth.se)

回复：您的太阳的消息，1996年11月10日太平洋标准时间09：16：36。日期：太平洋标准时间1996年11月11日星期一10：33：50

来自： Vern Paxson <vern>

>我正在为瑞典团队翻译GNU项目工作，我现在是

>使用flex。我有一些关于其中一些信息的问题

>，我希望你能回答。

顺便说一下，所有你想知道的关于flex内部结构的事情——可能唯一理解它们的英语意思的人就是我！ 所以我不会太担心要把他们做好。也就是说...

> #: main.c:545

> msgid " 已创建%d个协议\n“

>

>的原型是指原型吗？

是的-状态压缩表的原型。

> #: main.c:539

> msgid " 已创建了%d/%d（峰值%d)模板nxt-chk条目\n“

>

>在这里，我主要对“nxt-chk”感到困惑。我猜它的意思是“下次检查”。(?)

>然而，“模板下一个检查条目”对我来说没有多大意义。要

>能够找到一个好的翻译，我需要多了解一点。

在Aho/Sethi/Ullman编译器书中有一个用于压缩扫描仪表的方案。 它涉及到创建两对表。 第一个有“base”和“默认”条目，第二个有“下一个”和“检查”条目。“基本”条目按当前状态进行索引，并生成一个进入下一个/检查表的索引。 “默认”条目给出了在下一个/check中没有找到状态转换时如何做。 “next”条目给出了要输入的下一个状态，但只有当“check”条目验证此条目对于当前状态是正确的时。 Flex创建了一系列的模板

下一步/检查条目，然后将与这些模板之间的差异进行编码，作为压缩表的一种方式。

> #: main.c:533

> msgid " 已创建的%d/%d基本def条目\n“

>

>对于这里的“基准测深度”也有同样的问题。见上文。

本国的

（）在bol搞混乱

[致：新英李<xli@npac.syr.edu>主题： Re： FLEX？](mailto:xli@npac.syr.edu)

回复：您的留言，1996年11月13日星期三，太平洋标准时间17：28：38。日期：太平洋标准时间1996年11月13日星期三19：51：54

来自： Vern Paxson <vern>

>“卸载（）”他们到输入流，问题发生。如果我在扫描后就这样做

>.作为一个车厢，变量“YY\_CURRENT\_BUFFER->yy\_at\_bol”被更改。那

>表示马车旗不见了。

您可以通过调用yy\_set\_bol（）来控制这个问题。 它在手册中有描述。

> 如果在预读取中它进入文件的末尾，就完成了

>控制控制缓冲区的结束和文件的结束？不，没有办法收回文件。

> 顺便说一下，我使用的是flex 2.5.2和“-l”。

顺便说一下，最新的版本是2.5.4。 它修复了在2.5.2和

2.5.3. 你可以从ftp.ee.lbl.gov上得到它。

本国的

|操作符没有做我想做的事情

[致：阿兰。ISSARD@st.com](mailto:Alain.ISSARD@st.com)

主题：回复：用FLEX开始的条件

回复：太平洋标准时间1996年11月18日星期一，您的留言：09：45：02。日期：太平洋标准时间1996年11月18日星期一10：41：34

来自： Vern Paxson <vern>

>我不能使用开始条件范围和使用|（OR）与

>规则有启动条件。

问题是，如果您使用“|”作为正则表达式运算符，例如“a|b”的意思是“匹配“a”或“b””，那么它必须\*不\*周围有任何空白。 如果你想要特殊的“|”\*动作\*（从你的扫描仪上似乎就是这样），这是一种给出两个不同规则相同动作的方式：

foo |

条 matched\_foo\_or\_bar() ;

然后“|”必须与第一个规则分隔，“必须”后面跟着一个新行。 你不能把它写成：

foo |酒吧 matched\_foo\_or\_bar() ;

尽管你可能认为你可以，因为yacc支持这个语法。不幸的是，造成这种不兼容性的原因是历史上的，但它不太可能被改变。

您的开始条件范围的问题仅仅是由于您使用“|”后混淆flex导致的语法错误。

如果你还有什么问题，请告诉我。

本国的

为什么flex不能理解这个变量的尾随上下文模式呢？

[<gmargo@newton.vip.best.com>主题： Re： flex-2.5.3错误报告](mailto:gmargo@newton.vip.best.com)

回复：太平洋标准时间1996年11月23日星期六16：50：09。日期：太平洋标准时间1996年11月23日星期六17：07：32

来自： Vern Paxson <vern>

>包含的是一个lex文件，“真正的”lex将处理，但我无法得到

>可以通过flex来处理它。 你能试试，给我指明正确的方向吗？

您的问题是，扫描仪中的一些定义使用“/”尾随上下文操作符，并将其包含在（）中。 Flex不允许将此操作符包含在（）的操作符中，因为这样做允许未定义的正则表达式，如“（a/b）+”。 所以解决方案是去掉圆括号。 注意，您还必须使用-l选项来构建AT&T lex的扫描器。 如果没有此选项，flex会自动将定义括在括号中。

本国的

^操作员不工作

[收件人： Thomas Hadig <hadig@toots.physik.rwth-aachen.de>主题：回复回复？](mailto:hadig@toots.physik.rwth-aachen.de)

回复：太平洋标准时间1996年11月26日星期二14：35：01。日期：太平洋标准时间1996年11月26日星期二11：15：05

来自： Vern Paxson <vern>

>在我的lexer代码中，我有这样一行：

> ^\\*.\* { }

>

>因此，以astrix（\*）开头的所有行都是注释行。

>这不起工作！

我不能让这个问题重现-它对我来说很好。 不过请注意，如果你所拥有的东西略有不同：

评论^\\*。\*

%%

{COMMENT} { }

那么它就不起作用了，因为flex会了（）中的宏定义，所以规则变成了

(^\\*.\*) { }

现在“^”操作符不在行的直接开始，它被解释为一个常规字符。 您可以通过使用“-l”lex-兼容性标志或“%选项lex-compat”来避免此行为。

本国的

尾随上下文与尾随可选模式混淆了

[致： Adoram Rogel <adoram@hybridge.com>主题： Re： Flex 2.5.4 BOF ？？？](mailto:adoram@hybridge.com)

回复：1996年11月26日星期二16时间16：10：41。日期：太平洋标准时间1996年11月27日星期三10：56：25

来自： Vern Paxson <vern>

> 组织机构

>

>这匹配“组织”(在调试模式，后面的s

>与后面的上下文相匹配，而不是中的可选的(s)

>是单词的结尾。

这应该只发生在莱克斯身上。 Flex可以正确地匹配这个模式。

（这可能就是你要说的话，我只是不确定。）

>有没有办法避免这种危险的尾随上下文问题？

不幸的是，没有什么简单的办法。 另一方面，我不明白为什么它会是一个问题。 Lex的匹配显然是错误的，我希望通常意图与模式表达的相同，所以Flex的匹配将是正确的。

本国的

是flex GNU吗？

[<mackin@interlog.com>主题：回复： Flex文档错误](mailto:mackin@interlog.com)

回复：太平洋标准时间1996年12月02日星期一，您的留言：00：07：08。日期：太平洋标准时间1996年12月1日星期日22：29：39

来自： Vern Paxson <vern>

>我不确定如何或在哪里提交错误报告(文档或

>，否则)为GNU项目的材料...

严格来说，flex并不是GNU项目的一部分。 他们分发它是因为没有人写了一个像样的通用语言替代品。所以你应该直接把错误发送给我。 那些被送到GNU的人有时会找到我，但有些可能会在裂缝之间。

>在GNU信息，在“开始条件”部分，也在男人

>页面（我的日期是1995年4月）是一个不错的小片段，展示了如何这样做

>将C引号字符串解析为缓冲区，定义为MAX\_STR\_CONST

大小不幸的是，没有进行过溢出检查，这在手册中已经提到了：...

最后，这里有一个例子，如何使用独家开始条件匹配c风格的引用字符串，包括扩展的转义序列（但不包括检查一个太长的字符串）：

不执行溢出检查的原因是，它将不必要地打乱一个示例，其主要目的只是为了演示如何使用flex。

顺便说一下，最新的版本是2.5.4，可以从ftp.ee.lbl.gov上获得。

本国的

埃拉塞米53

[致：tsv@cs。乌马尼托巴省。CA主题： Re： Flex（reg）..](mailto:tsv@cs.UManitoba.CA)

回复：太平洋标准时间1997年3月6日星期四23：50：16。日期：太平洋标准时间1997年3月6日星期四15：54：19

来自： Vern Paxson <vern>

> [：alpha：]（[：alnum：] | \\\_）\*

如果您的规则真的有如上所示嵌入的空格，那么它将无法工作，因为第一个空格从操作中分隔规则。 （它甚至无法编译……） 您需要：

[:alpha:]([:alnum:]|\\\_)\*

这应该会很好地工作——除了后面的上下文操作符‘/’之外，对（）的内部内容没有任何限制。

本国的

我需要扫描，如果，然后，其他块和当循环

[致：“迈克·斯托尔尼基”<mstolnic@ford.com>主题： Re： FLEX帮助](mailto:mstolnic@ford.com)

回复：您的1997年5月30日星期五13：33：27 PDT。日期：1997年5月30日星期五10时46：35太平洋夏令时

来自： Vern Paxson <vern>

>我们想添加“如果-那么其他”，“同时”和“for”的语句

语言...

>我们已经研究了许多可能的解决方案。 唯一的解决方案似乎是

>最合理的方法是了解一个代币在阴阳语中的位置。

我强烈建议您构建一个解析树（抽象语法树），然后对其进行循环。 您会发现，这在保持解释器的简单性和可扩展性方面有很大的好处。

也就是说，您提到的get\_位置和set\_位置的功能已经在待办事项列表上有一段时间了。 由于flex对我来说是一个纯粹的业余时间项目，所以不能保证何时会添加它（特别是，它肯定不会在未来几个月内添加）。

本国的

埃拉塞米55

[致：科林·保罗·亚当斯，<colin@colina.demon.co.uk>](mailto:colin@colina.demon.co.uk)

主题： Re： Flex C++类和野牛

回复：您的留言：1997年8月09日17：11：41 PDT。

日期：1997年8月15日星期五10：48：19

来自： Vern Paxson <vern>

> #define YY\_DECL ，结构解析器\_控件

> \*parm)

>

>，我一直在尝试 让它作为一个C++扫描仪工作，但它确实如此

>似乎是不可能的(警告它不匹配任何声明

> yyFlexLexer，或者类似的东西)。

>

>这应该是可能的，还是它正在工作(我做了

>注意到扫描器类仍然是实验性的，所以我是

>不太有希望)？

您需要做的是从yyFlexLexer派生出一个子类，它提供了上面的yylex（）方法，并将lvalp和parm转换为成员

变量，然后调用yyFlexLexer：：yylex（）来进行常规扫描。

本国的

埃拉塞米56

[致：Mikael。Latvala@lmf.ericsson.se](mailto:Mikael.Latvala@lmf.ericsson.se)

主题：回复：在Flex v2.5文档中可能出现的错误

回复：您的留言，1997年9月5日星期五16：07：24 PDT。日期：1997年9月5日星期五10：01：54 PDT

来自： Vern Paxson <vern>

>在那个例子中，您展示了如何在使用时计算注释行

...>C风格/\* \*/评论。我的问题是，你不应该接受吗

>帐户是一个注释标记的结束出现在内部的场景

>字符或字符串文字？

扫描仪当然还需要扫描字符和字符串文字。但是它这样做了（在字符串的手册页中有一个例子），词典词典将在运行嵌入的“/\*”之前识别文字的开头。 因此，它甚至在考虑匹配“/\*”的可能性之前就完成文字扫描。

样例

'([^']\*|{ESCAPE\_SEQUENCE})'

将匹配“”之间的所有文本（包括）。 所以词典作者认为这是一个从第一个‘开始的标记，甚至没有

尝试匹配其中的其他令牌。

我认为这种微妙之处不值得放在手册中，因为我怀疑它会迷惑更多的人，而不是启发它。

本国的

埃拉塞米57

[<leisner@sdsp.mc.xerox.com>主题：回复： flex限制](mailto:leisner@sdsp.mc.xerox.com)

回复：您的留言，1997年9月06日星期六11：27：21 PDT。日期：1997年9月8日星期一11：38：08 PDT

来自： Vern Paxson <vern>

> %%

> [a-zA-Z]+ /\*跳过一行\*/

> { printf("got %s\n", yytext); }

> %%

你使用的是什么版本的flex？ 如果我把这个馈送到2.5.4，它会抱怨：

“bug。l”，第5行：在操作“bug”中遇到EOF，第5行：无法识别的规则

“bug。l”，第5行：致命的解析错误

这并不是世界上最大的错误信息，但它设法标记出了这个问题。

（随着启动条件范围的引入，flex不能容纳一个单独的行上的操作，因为它与一个缩进的规则是不明确的。）

你可以从ftp.ee.lbl.gov得到2.5.4。

本国的

有一个弹性扫描仪的存储库吗？

但我们还不知道。你可以试着问一下这个比较。汇编者

我如何可以有条件地编译或预处理我的flex输入文件？

Flex没有像C那样的预处理器。您可以尝试使用m4，或C预处理器加上一个sed脚本来清理结果。

我在哪里可以找到lex和yacc的语法？

在弯曲和野牛的来源中。

我为扫描的每个字符得到一个缓冲区结束消息。

如果你的LexerInput（）函数一次只返回一个字符，就会发生这种情况，如果你的扫描器是“交互式的”，或者如果你的平台上的流库总是为yyin->gcount（）返回1，就会发生这种情况。

解决方案：用一个返回整个缓冲区的版本来覆盖LexerInput（）。

未命名faq-62

[致：Georg。Rehm@CL-KI.Uni-Osnabrueck .设计主题：回复： Flex最大值](mailto:Georg.Rehm@CL-KI.Uni-Osnabrueck.DE)

回复：太平洋标准时间1997年11月17日星期一17：16：06。日期：太平洋标准时间1997年11月17日星期一17：16：15

来自： Vern Paxson <vern>

>我快速浏览了flex源代码，并修改了一些#定义

> flexdefs.h:

>

> #定义INITIAL\_MNS 64000

> #定义MNS\_INCREMENT 1024000

> #定义MAXIMUM\_MNS 64000

要解决的事情是添加几个零：

/\*标记了对总是阻塞\*/的状态的引用

#定义MAXIMUM\_MNS 31999

#定义BAD\_SUBSCRIPT -32767

定义MAX\_SHORT 32700

而且，如果你抱怨有太多的规则，也要做出以下改变：

#定义YY\_TRAILING\_MASK 0x200000

#定义YY\_TRAILING\_HEAD\_MASK 0x400000

-弗恩

未命名的常见问题63

[：jimmey@lexis-nexis.com（吉mmey托德）](mailto:jimmey@lexis-nexis.com)

回复：您的留言，1997年12月08日星期一15：54：15太平洋标准时间。日期：太平洋标准时间1997年12月15日星期一13：21：35

来自： Vern Paxson <vern>

> 标准处理= YY\_CURRENT\_BUFFER；

> ifstream fin ( "aFile" );

> yy\_switch\_to\_buffer（yy\_create\_buffer（fin，YY\_BUF\_SIZE））；

>

>我想做的是，它是通过一个集合传递一个文件的内容

>的规则，然后通过另一组规则。如果，我

>不使用C-++类。但既然我所做的其他一切都是这样的

>在C++中，我想我会是一致的。

>

>问题是，“yy\_create\_buffer”期待一个i流\*

>的第一个参数（如手册页中所述）。然而，鳍是一个独立的流

目标你知道我可能做错了什么吗？任何帮助都是

感激谢谢！!

你需要通过和鳍，把它变成一个if流\*，而不是一个if流。那么它的类型将与预期的istream\*兼容，因为ifstream是从istream派生出来的。

未命名的常见问题64

本国的

[致： Enda Fadian <fadiane@piercom.ie>](mailto:fadiane@piercom.ie)

与Flex手册页相关的问题？

回复：太平洋标准时间1997年12月16日星期二15：17：34。日期：太平洋标准时间1997年12月16日星期二14：17：09

来自： Vern Paxson <vern>

>你能向我解释一下什么是跳远动作吗？在yylex（）内部使用长jmp（）函数或由它调用的例程。

>什么是弹性激活框架。只是yylex（）的堆栈框架。

就我所知，yy重启的>将使我回到输入的启动位置

>文件和使用flex++并不是一个真正的选择！

不，重启（）并不意味着倒带，尽管它的名字听起来像。 它告诉扫描仪刷新其内部缓冲区，并在其当前位置开始从给定的文件中读取。

未命名faq-65

本国的

[致： hassan@larc.info.uqam.ca（Hassan Alaoui）主题： Re：需要紧急帮助](mailto:hassan@larc.info.uqam.ca)

回复：太平洋标准时间1997年12月20日星期六19：38：19。日期：太平洋标准时间1997年12月21日星期日21：30：46

来自： Vern Paxson <vern>

>/usr/库/亚库：在函数int（）中：

警告：隐式声明的函数‘int yylex（...）’

>

> ld：未定义符号

> \_yylex

> \_yyparse

> \_yyin

这是C++（和/或yacc）的问题。 我认为修复方法是为相应的例程/符号显式地插入一些“外部”“C”语句。

未命名的常见问题66

本国的

[至： mc0307@mclink.it Cc： gnu@prep.ai.mit.edu](mailto:gnu@prep.ai.mit.edu)

[主题：回复： [mc0307@mclink.it：帮助请求]](mailto:mc0307@mclink.it)

回复：太平洋标准时间1997年12月12日星期五17：57：29。日期：太平洋标准时间1997年12月21日星期日22：33：37

来自： Vern Paxson <vern>

>这是我对浮点数和整数类型的定义：

> ...

> NZD [1-9]

>...

>我已经测试了我的程序在其他lex版本(在UNIX Sun Solaris和HP

> UNIX)，而且它工作得很好，所以我认为我的定义是正确的。

>Lex和Flex有什么区别吗？

正如在手册页中所讨论的那样，确实有差异。一个

您可能遇到的问题是，当flex扩展名称定义时，它会在展开的周围放置括号，而lex则没有。 有

手册页中的一个例子，说明这如何导致不同的匹配。Flex的行为符合POSIX标准（或者至少是我看到的最后一个POSIX草案）。

未命名的常见问题67

本国的

[致： hassan@larc.info.uqam.ca（Hassan Alaoui）主题：回复：谢谢](mailto:hassan@larc.info.uqam.ca)

回复：太平洋标准时间1997年12月22日星期一16：06：35。日期：太平洋标准时间1997年12月22日星期一14：35：05

来自： Vern Paxson <vern>

>非常感谢你的帮助。我编译和链接得很好地与C++，而

>声明“yylex……”外部，但还有一个小问题。我得到一个

>分割默认设置在执行时（我链接到lfl库），而它

>在使用LEX而不是flex时工作得很好。你有什么想法吗

>的原因吗？

我想到的一个可能的原因是，如果你将yy文本定义为“外部字符yy文本[]”（这是lex使用的），而不是“外部字符\*yy文本”（这是flex使用的）。 如果不是这样，那么恐怕我不知道问题是什么。

未命名的常见问题68

本国的

[收件人：<NISWONGR@almaden.ibm.com>主题： Re： flex 2.5： c++扫描仪和启动条件](mailto:NISWONGR@almaden.ibm.com)

回复：太平洋标准时间1998年1月6日星期二10：34：21。日期：太平洋标准时间1998年1月6日星期二19：19：30

来自： Vern Paxson <vern>

>问题是，当我这样做时（使用%选项c++）启动

>条件似乎不适用。

begirn宏将修改yy\_start变量。 对于C扫描仪，这

是一个静态的，范围通过整个文件可见。 对于C-++扫描仪，它是一个成员变量，因此它只在一个成员函数中具有可见的作用域。 当您构建C++扫描仪时，lexbegin（）例程不是一个成员函数，因此它没有修改正确的yy\_start。 指示您发现需要添加的诊断信息

yy\_start的声明，以便使扫描器编译；相反，正确的解决方法是使（）成为成员函数（通过从yyFlexLexer派生）。

未命名的常见问题69

本国的

[致：“BorisZinin”<boris@ippe.rssi.ru>主题： Re：在flex缓冲区中的当前位置](mailto:boris@ippe.rssi.ru)

回复：太平洋标准时间1998年1月12日星期一18：58：23。日期：太平洋标准时间1998年1月12日星期一12：03：15

来自： Vern Paxson <vern>

>问题是如何确定弯曲活动中的当前位置

当一个规则被匹配时，>缓冲区。

您将需要显式地跟踪它，例如通过重新定义YY\_USER\_ACTION来计算匹配的字符数。

顺便说一下，最新的flex版本是2.5.4，可从ftp.ee.lbl.gov上获得。

未命名的常见问题70

本国的

[致：Bik。Dhaliwal@bis.org主题： Re： Flex问题](mailto:Bik.Dhaliwal@bis.org)

回复：太平洋标准时间1998年1月26日星期一13：05：35。日期：太平洋标准时间1998年1月27日星期二22：41：52

来自： Vern Paxson <vern>

>这个要求包括了解

>：匹配特定标记的字符位置

>在lexer。

这样做的方法是通过显式地跟踪您在文件中的位置，通过计算为每个标记扫描的字符数（在叶龙中可用）。 如手册中所述，通过重新定义YY\_USER\_ACTION，可以方便地做到这一点。

未命名的常见问题71

本国的

[致：弗拉基米尔·阿列克谢耶夫·<vladimir@cs.ualberta.ca>](mailto:vladimir@cs.ualberta.ca)

如何从解析器中控制开始条件？回复：太平洋标准时间1998年1月26日星期一，您的留言：05：50：16。日期：太平洋标准时间1998年1月27日星期二22：45：37

来自： Vern Paxson <vern>

>解析器能够告诉词典词典似乎很有用

>上下文依赖性，因为这样它们就不必局限于

>本地或顺序上下文。

实现这一点的一种方法是让解析器调用一个包含在扫描程序的.l文件中的存根例程，因此可以开始访问。 唯一丑陋的是解析器不能在它想要的状态中传递，因为这些状态是不可见的——但是如果你没有许多这样的状态，那么使用另一组名称看起来就不像是

这是一个非常沉重的负担。

虽然像您建议的那样生成一个.h文件当然更干净，但flex开发已经进入了一个虚拟的静态状态：-(，所以像上面这样的解决方法比等待一个新特性要实用得多。

未命名的常见问题72

本国的

[致：芭芭拉·丹尼<denny@3com.com>主题：回复： freebsd flex bug？](mailto:denny@3com.com)

回复日期：美国太平洋标准时间1998年1月30日星期五12：00：43。日期：太平洋标准时间1998年1月30日星期五12：42：32

来自： Vern Paxson <vern>

> lex.yy.c：1996：在‘=’之前的解析错误

这是识别这个错误的关键。 （通过使用flex -L可以帮助精确定位它，因此它在输出中不会生成#line指令。） 我敢打赌你有起步条件

名称也是一个变量名称，或者类似的东西；flex为每个开始条件名称输出#定义，将它们映射到一个数字，所以你可以得到：

%x foo

%%

%%

...

空栏（）

{

int foo = 3;

}

在C预处理后将变成“int1=3”，因为flex将在生成的扫描仪中放置foo1“#定义”。

未命名的常见问题73

本国的

[致：莫里斯皮trie<mpetrie@infoscigroup.com>主题： Re：丢失的flex。l文件](mailto:mpetrie@infoscigroup.com)

回复：太平洋标准时间1998年2月2日星期一14：10：01。日期：太平洋标准时间1998年2月2日星期一11：15：12

来自： Vern Paxson <vern>

>，我很好奇

>是否有一个简单的方法来从生成的源代码回溯到

>复制了我们正在搜索的丢失的令牌列表。

理论上，从DFA表示再回到正则表达式表示是很直接的——这两者都是同构的。

在实践中，这是一个巨大的头痛，因为您必须将所有表打包回一个DFA表示，然后编写一个程序来咀嚼它，并将其转换为RE。

很抱歉，我听到了这个不那么高兴的消息...

未命名的常见问题74

本国的

[致： jimmey@lexis-nexis.com（Jimmey Todd）主题：回复： Flex性能问题](mailto:jimmey@lexis-nexis.com)

回复：太平洋标准时间1998年2月19日星期四11：01：17。日期：太平洋标准时间1998年2月19日星期四08：48：51

来自： Vern Paxson <vern>

>我发现，数据块越小，速度就越快

>程序执行。这与我的相反。这应该是

>是这样发生的？

如果您的输入文件嵌入了nul，就会发生什么。从手册页中：

最后要注意的是：当匹配NUL时，flex会很慢，特别是当一个令牌包含多个NUL时。如果预期文本通常会包含NUL，那么最好编写与短期文本相匹配的规则。

这是首先要找的事情。

未命名的常见问题75

本国的

[致： jimmey@lexis-nexis.com（Jimmey Todd）主题：回复： Flex性能问题](mailto:jimmey@lexis-nexis.com)

回复：太平洋标准时间1998年2月19日星期四11：01：17。日期：太平洋标准时间1998年2月19日星期四15：42：25

来自： Vern Paxson <vern>

所以有几个问题。

首先，为了快速运行，你想要匹配尽可能多的文本，而不是扫描仪正在扫描的不是<RN>标签。 所以你想要一个像这样的规则：

[^<]+

其次，C++扫描仪如果是交互式的，速度特别慢，这是默认的。 在我的工作站上，使用-b可以将其速度提高了3-4倍。

第三，使用istream接口的C++扫描仪速度很慢，因为istream的实现非常糟糕。 我建立了以下两个版本的扫描仪：

%%

.\*\n

.\*

%%

C版在我的工作站上在0.8秒内吸入一个2.5MB的文件。使用-++的++版本需要3.8秒。

未命名的常见问题76

本国的

[致：“弗雷斯卡托雷，大卫（CRD，TAD）”<frescatore@exc01crdge.crd.ge.com>主题：Re：FLEX2.5&2000年](mailto:frescatore@exc01crdge.crd.ge.com)

回复：您的留言，1998年6月03日星期三11：26：22 PDT。日期：1998年6月03日星期三10：22：26 PDT

来自： Vern Paxson <vern>

>我正在研究通用电气研发过程中的千年虫问题

>，需要知道是否有任何已知的问题有关

>为上述软件和Y2K，无论版本如何。

不应该有，它所做的只是询问系统，然后打印出来。

未命名的常见问题77

本国的

[<htd@ibhdoran.com>主题： Re： flex问题](mailto:htd@ibhdoran.com)

回复：您的留言，1998年7月15日星期三21：30：13 PDT。日期：1998年7月21日星期二14：23：34太平洋夏令时

来自： Vern Paxson <vern>

>为了克服这个问题，我将（）的stdin变成一个字符串，然后表示字符串。这个

>字符串是可以排列的，除了字符串的结尾没有正确排列

>（yy\_scan\_string（）），即词法字符串不能识别字符串的末尾。

Flex不包含识别缓冲区端点的机制。 但如果

您可以使用fgets（无论如何您应该使用，以防止缓冲区溢出），然后最终的\n将保存在字符串中，您可以扫描它以找到字符串的末尾。

未命名的常见问题78

本国的

[收件人：soumen@almaden.ibm.com](mailto:soumen@almaden.ibm.com)

主题： Re： Flex++ 2.5.3实例成员vs。静态成员回复：您的消息，星期一，1998年7月27日02：10：04 PDT。日期：1998年7月28日星期二01：10：34 PDT

来自： Vern Paxson <vern>

> %{

> int mylineno = 0;

> %}

> ws [ \t]+

>阿尔法[A-Za-z]

>挖掘[0-9]

> %%

>

>现在你希望我是每个类实例的成员

> yyFlexLexer，但情况是这样吗？看看lex.yy.cc文件似乎就可以了

>表示否则；除非我遗漏了一些声明

>>似乎在任何类范围之外。

>

>这将如何工作，如果我想运行一个多线程应用程序与每个

>线程创建一个FlexLexer实例吗？

推导你自己的子类，并使它成为它的成员变量。

未命名的常见问题79

本国的

[致： Adoram Rogel <adoram@hybridge.com>主题： Re：超过32K个州的变化挂起](mailto:adoram@hybridge.com)

回复：您的留言，1998年8月4日星期二16：55：39 PDT。日期：1998年8月4日星期二22：28：45 PDT

来自： Vern Paxson <vern>

>弗恩帕克森，

>

>，我听从了你的建议，在Usenet bu你上发帖，并发邮件给我

>个人由你提出，关于如何克服32K州的限制。我在跑步

Linux机器上的>。

>，我使用了2.5.4版本的完整源代码，并在中做了以下更改

> flexdef.h:

> #define JAMSTATE -327660

>#定义MAXIMUM\_MNS 319990

>#定义BAD\_SUBSCRIPT -327670

>#定义MAX\_SHORT 327000

>

>和编译。

>所有看起来都很好，包括检查和大检查，所以我安装了。

嗯，你不应该增加MAX\_SHORT，尽管通过查看我的电子邮件档案，我发现我确实建议这样做。 试着设置回32700；这应该足以让你不再需要-Ca。 如果它还挂着，那么有趣的问题是，在哪里？

>编译相同的悬挂程序(RedHat 4.2

Linux的>分发)

> flex 2.5.4二进制工作。

由于Linux附带源代码，所以您应该与查看它们遗漏的问题不同。

>我现在应该一直用-Ca选项来编译吗？甚至短而简单

过滤

不，绝对不是。 它是针对那些情况，你绝对必须挤出最后一个周期的扫描仪。

本国的

未命名faq-80

[致：<Craig。Schmackpfeffer@usa.xerox.com>主题： Re：针对静态代码部分的flex输出](mailto:Craig.Schmackpfeffer@usa.xerox.com)

回复：您的留言，1998年8月11日星期二11：55：30 PDT。日期：1998年8月17日星期一23：57：42 PDT

来自： Vern Paxson <vern>

>我想使用flex来生成一个二进制文件

>，其中包含控制解析的数据结构。

这已经在愿望清单上出现很久了。 原则上，它是直接的——你将mkdata（）等人的I/O重定向到另一个文件，并修改骨架，有一个启动功能，将这些分解为动态数组。 关注的是(1)扫描器生成代码是毛茸茸的，充满了角的情况，所以当沿着这条路径时很容易感到惊讶：(；(2)小心缓冲，以便当表改变时，你确保扫描器开始在

在输入文件的正确点进行正确的状态和读取。

>我想知道你是否知道谁用这种方式使用flex。

我不喜欢，但这似乎是一个合理的项目（不像许多其他的flex调整：-）。

本国的

未命名的常见问题81

收到：来自131.173.17.11（131.173.17.11 [131.173.17.11]）

由ee.lbl.gov（8.9.1/8.9.1）与ESMTP id AAA03838

[<vern@ee.lbl.gov>；1998年8月20日星期四00：47：57 -0700（PDT）](mailto:vern@ee.lbl.gov)

收到的信息：来自hal.cl-ki.uni-osnabrueck.de (hal.cl-ki。Uni-Osnabrueck。DE [131.173.141.2由deimos.rz.uni-osnabrueck.de（8.8.7/8.8.8）与ESMTP id JAA34694

[<vern@ee.lbl.gov>；1998年8月20日星期四09：47：55 +0200](mailto:vern@ee.lbl.gov)

收到：（来自georg@本地主机）由hal.cl-ki.uni-osnabrueck.de（8.6.12/8.6.12）id JAA来自： Georg Rehm <georg@hal.cl-ki.uni-osnabrueck.de>

[消息-Id：<199808200747.JAA34834@hal.cl-ki.uni-osnabrueck.de>主题：“弹性扫描仪回推溢出”](mailto:199808200747.JAA34834@hal.cl-ki.uni-osnabrueck.de)

[收件人：vern@ee.lbl.gov](mailto:vern@ee.lbl.gov)

日期：1998年8月20日星期四09：47：54 +0200（MEST）

[回复： Georg。Rehm@CL-KI.Uni-Osnabrueck .德](mailto:Georg.Rehm@CL-KI.Uni-Osnabrueck.DE)

不要发送商业邮件到这个地址、垃圾邮件或广告！[X-URL: http://www.cl-ki.uni-osnabrueck.de/~georg/](http://www.cl-ki.uni-osnabrueck.de/%7Egeorg/)

X-Mailer：ELM[版本2.4ME+ PL28（25）]内存-版本： 1.0

内容类型：文本/普通；字符集=US-ASCII内容-传输-编码：7位

你好Vern，

昨天，我遇到了一个奇怪的问题：我使用宏处理器m4在一个。l文件中包含了一些冗长的列表。以下是一个弹性宏观定义，它会导致我的颈部出现严重的疼痛：

作者 ("A.Boucard / L.布卡“|”A。Dastarac / M.完整的列表包含了大约10kB。当我尝试“伸缩”这个文件时

(在Solaris 2.6机器上，使用修改后的flex2.5.4(我只增加了

在flexdefs.h)中的一些预定义值中，我得到了一个错误：

myflex/flex -8 sentag.tmp.l flex扫描仪后推溢出

当我删除宏定义中的斜杠时，一切都工作正常。

据我所知，双引号转义了斜杠字符，所以它真正的意思是“/”，而不是“尾随上下文”。此外，我试着

用反斜杠转义斜杠，但不使用，在弯曲代码时会出现相同的错误消息。

你知道这地方发生了什么吗？来自德国的问候，

佐治亚州

--

格奥尔格雷姆 [奥斯纳布鲁埃克大学的georg@cl-ki.uni-osnabrueck.de语义信息处理研究所，FRG](mailto:georg@cl-ki.uni-osnabrueck.de)

未命名faq-82

[致：Georg。Rehm@CL-KI.Uni-Osnabrueck .“flex扫描仪后推溢出”](mailto:Georg.Rehm@CL-KI.Uni-Osnabrueck.DE)

回复：您的留言，1998年8月20日星期四09：47：54 PDT。日期：1998年8月20日星期四07：05：35 PDT

来自： Vern Paxson <vern>

> myflex/flex -8 sentag.tmp.l

> flex扫描仪后推溢出

Flex本身使用了一个Flex扫描仪。 当扫描器试图卸载您定义的巨大宏时，它耗尽了缓冲空间。 当您删除‘/’时，您使它足够小，使它适合缓冲区；删除空格也会做同样的事情。

解决办法是重新考虑如何使用这么大的宏，也许还有另一种更好的方法；或者用更大的值重建flex自己的扫描

#定义YY\_BUF\_SIZE 16384

* 本国的

未命名的常见问题83

[致： Jan Kort <jan@research.techforce.nl>主题： Re： Flex](mailto:jan@research.techforce.nl)

回复：你的留言，1998年9月4日星期五12：18：43 +0200。

日期：1998年9月5日星期六00：59：49 PDT

来自： Vern Paxson <vern>

> %%

>

> "TEST1\n" {fprintf（“TEST1\n”）；无(5)；

> ^\n {fprintf（“空行”）；

> . { }

> \n （“新行”）

>

> %%

> --输入---------------------------------------

> TEST1

> --输出--------------------------------------

> TEST1

>空行

> ------------------------------------------------

依我之见，目前还不清楚这是否真的是一个错误。 这取决于您是将无y（）视为输入流中的备份，还是将新字符推送到输入流的开头。 弯曲

将其解释为后者（我承认，为了便于实现），因此认为换行符实际上是在一行开头的匹配，因为在最后一个令牌扫描了整行之后，扫描器现在处于新行的开头。

我同意这对于无害的（）是违反直觉的，因为它的功能描述（对于未输入的（）就不那么直观，这取决于你是未输入新文本的（）还是扫描文本）。 但我不打算很快改变它，因为这样做是很痛苦的。 因此，

您确实需要使用yy\_set\_bol（）和YY\_AT\_BOL（）来调整您的扫描仪到您想要的行为。

很抱歉，我的回答不太完全令人满意。

本国的

未命名的常见问题84

[致人：帕特里克·克鲁塞诺托·<krusenot@mac-info-link.de>](mailto:krusenot@mac-info-link.de)

主题：回复：重新启动flex-2.5.2-generated扫描仪的问题-回复：您的消息，1998年9月24日星期四10：14：07 PDT。

日期：1998年9月24日星期四23：28：43太平洋夏令时

来自： Vern Paxson <vern>

>，我使用flex-2.5.2和野牛1.25，我很绝望

>试图在我的解析器停止后用一个新文件重新启动我的扫描程序

具有解析错误的>。当我的编译器重新启动时，解析器总是如此

>在令牌之后接收令牌（在旧文件中！）这就导致了

>解析器错误。

我怀疑问题是您的解析器已经按顺序提前阅读了

试图解决一个歧义，当它重新启动时，它会选择

用这个标记，而不是读一个新的。 如果您使用的是yacc，那么特殊的“错误”生产有时可以用于使用令牌，以试图使解析器进入一致的状态。

未命名faq-85

本国的

[<junghelh@pe-nelson.com>主题：回复： flex 2.5.4a](mailto:junghelh@pe-nelson.com)

回复：1998年10月27日星期二16：4：41：42。日期：太平洋标准时间1998年10月27日星期二16：50：14

来自： Vern Paxson <vern>

>这是一个功能请求： 做一个命令行怎么样

>选项可以在从stdin读取时指定文件名？ 这样一个

>不需要创建一个临时文件来获得“#行”

>的指令是有意义的。

结合使用（根据-o的手册页描述）。

>对我来说，有没有什么简单的方法来使用非阻塞的IO来解析多个

流简单，没有。

一种方法可能是在视点块上返回一个魔法字符并有一个规则

.\*<magic-character> //回去。\*，吃魔法角色这是我的头顶，不确定它是否有效。

未命名faq-86

本国的

[<billy.d.repko@intel.com>主题： Re：编译扫描仪](mailto:billy.d.repko@intel.com)

回复：太平洋标准时间1999年1月13日星期三10：52：47。日期：太平洋标准时间1999年1月14日星期四00：25：30

来自： Vern Paxson <vern>

>似乎它找不到lfl库。

发行版中的Makefile就构建了它，所以您应该拥有它。

这非常简单，只是一个调用yylex（）的主（）和一个总是返回1的yyrap（）。

|  |  |  |  |
| --- | --- | --- | --- |
| > | %% |  | |
| > |  | \n | ++num\_lines ; ++num\_chars ; |
| > |  | . | ++num\_chars ; |

你不能像这样缩进你的规则——这就是错误的来源。 Flex将缩进的文本复制到输出文件中，这是你做类似事情的方式

使用int num\_lines\_seen = 0；来声明本地变量。

未命名的常见问题87

本国的

[致：埃里克·布兰德霍斯特<Erick。Branderhorst@asml.nl>主题：回复： flex输入缓冲区](mailto:Erick.Branderhorst@asml.nl)

回复：太平洋标准时间1999年2月9日星期二13：53：46。日期：太平洋标准时间1999年2月9日星期二21：03：37

来自： Vern Paxson <vern>

>在flex.skl文件中，设置了默认输入缓冲区的大小。 你能不能

>解释了为什么会设置这个大小，以及为什么它是一个如此高的数字。

在扫描大文件时，优化性能很大。 如果需要的话，你可以安全地把它更低。

未命名faq88

本国的

[致：“GuidoMinnen”<guidomi@cogs.susx.ac.uk>主题： Re： Flex错误消息](mailto:guidomi@cogs.susx.ac.uk)

回复：太平洋标准时间1999年2月24日星期三15：31：46。日期：太平洋标准时间1999年2月25日星期四00：11：31

来自： Vern Paxson <vern>

>我正在扩展一个用Flex编写的大型扫描仪，我不断遇到

问题更具体地说，我得到了一个错误信息：

>“flex：输入规则太复杂（>= 32000 NFA状态）“增加flexdef.h中的定义：

定义干扰状态-32766 /\*标记了一个对总是j ams \*/的状态的引用

#定义MAXIMUM\_MNS 31999