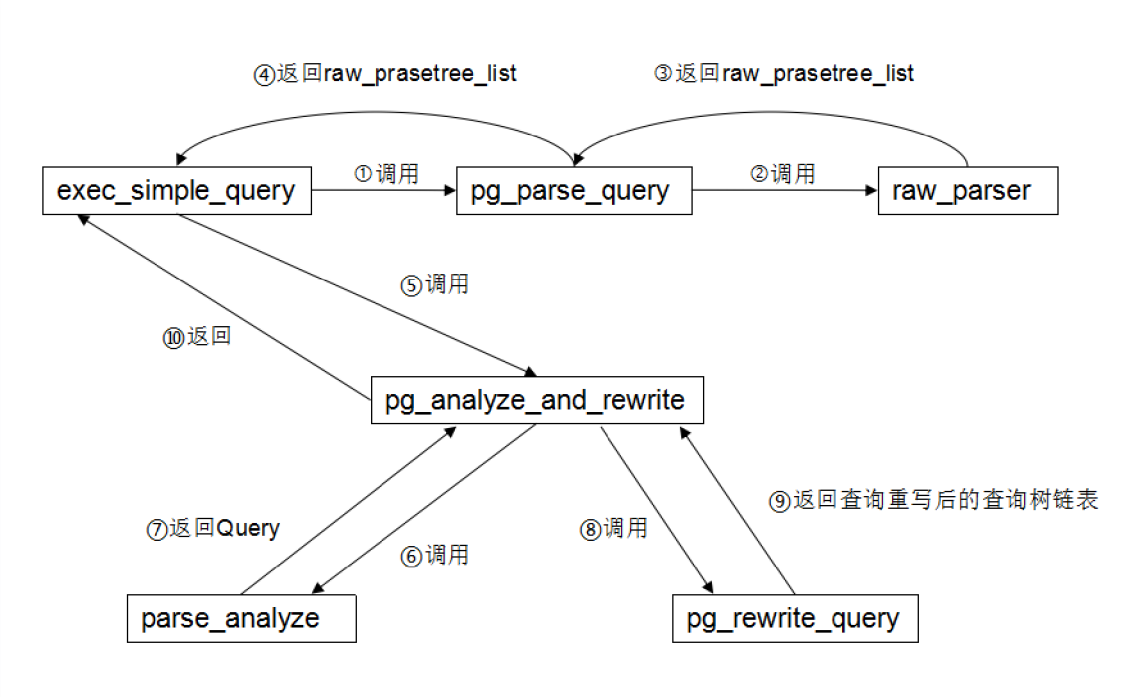
postgresql&Age的Parser(查询分析模块)

1. Postgre整体调用流程

当postgresql的后台服务进程收到前台发来的查询语句后，首先将其传递到查询分析模块，进行词法分析，语法分析和语义分析。若是功能性命令(例如create table,create user和backup命令等)则将其分配到功能性命令处理模块；对于查询处理命令(SELECT/INSERT/DELETE/UPDATE)则为其构建查询语法树，交给查询重写模块。

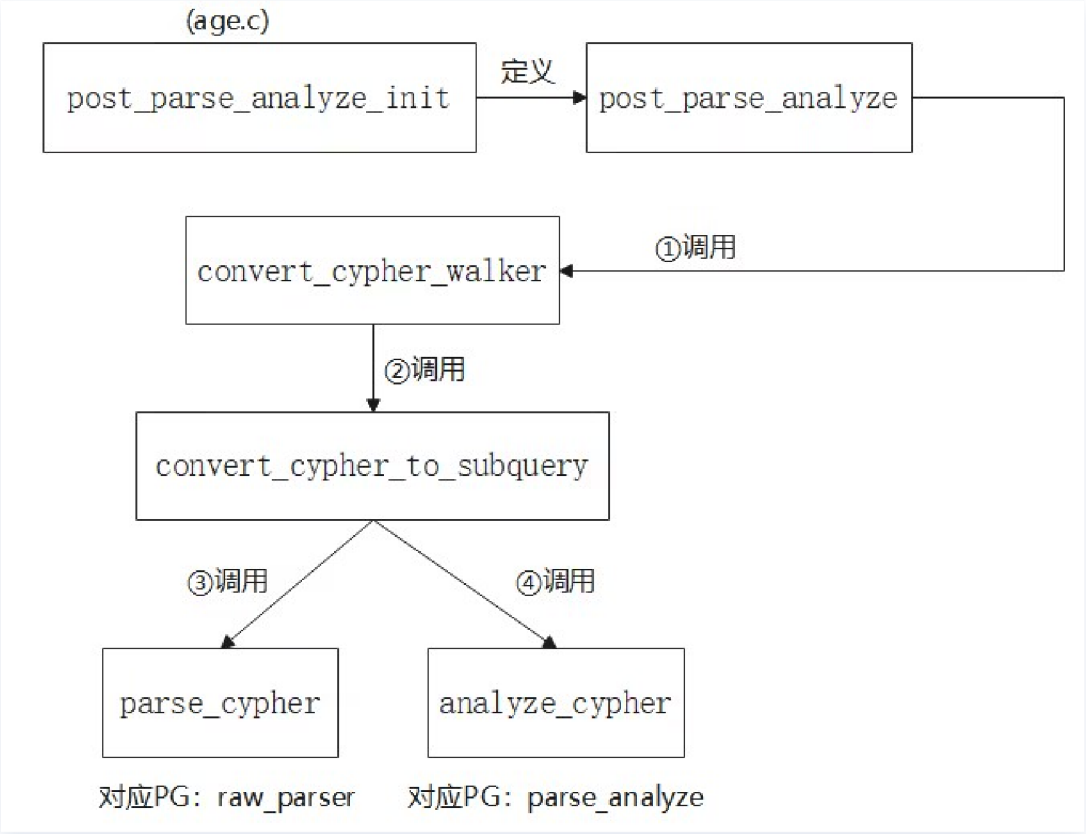
总体流程如下：

SQL命令 --(词法和语法分析)--> 分析树 --(语义分析)--> 查询树



- exec_simple_query函数(在src/backend/tcop/postgres.c下)调用函数pg_parse_query进入词法分析和语法分析的主过程，函数pg_parse_query再调用词法分析和语法分析的入口函数raw_parser生成分析树；
- 函数pg_parse_query返回分析树(raw_parsetree_list)给exec_simple_query；
- exec_simple_query函数调用函数pg_analyze_and_rewrite进行语义分析(调用parse_analyze函数，将raw_parsetree变为Query类型，传入下一个函数pg_rewrite_query)和查询重写(调用pg_rewrite_query函数)；
- 返回查询树链表给exec_simple_query。

2. Age的调用流程



在age.c下，_PG_init函数中定义了post_parse_analyze_init() -> post_parse_analyze() 自定义hook函数 -> convert_cypher_walker 函数 ->convert_cypher_to_subquery-> 先调用parse_cypher(写法对应于raw_parser)，之后又调用了analyze_cypher(返回Query，写法对应parse_analyze)

convert_cypher_walker函数->convert_cypher_to_subquery：从from子句中找到cypher()调用，并把他们转换为select子查询

问题：Postgre是从哪里调用了post_parse_analyze这个hook函数的?

PostgreSQL中，parse_analyze函数中，有post_parse_analyze hook的判断调用：

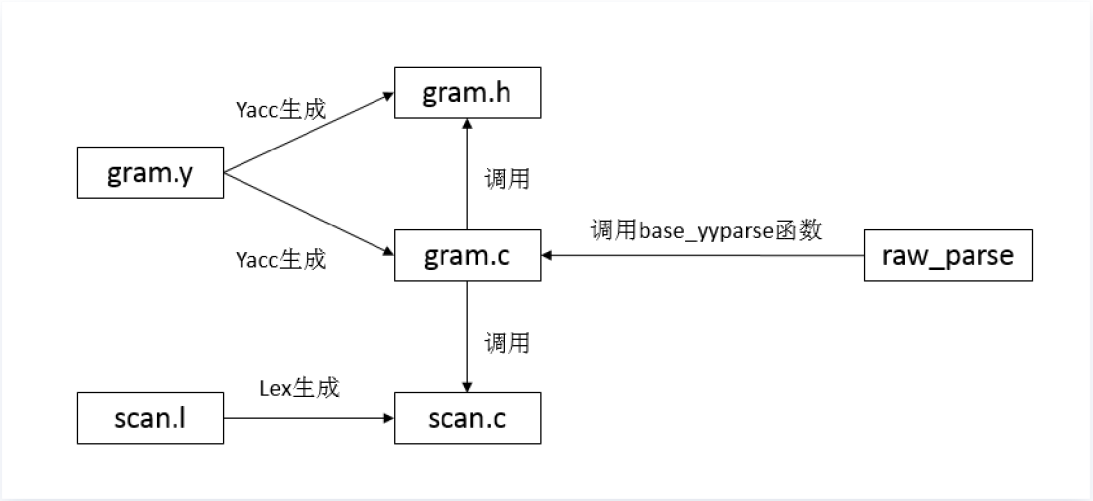
```
Query *
parse_analyze(Node *parseTree, const char *sourceText,
              Oid *paramTypes, int numParams)
{
    ParseState *pstate = make_parsestate(NULL);
    Query      *query;

    Assert(sourceText != NULL); /* required as of 8.4 */

    pstate->p_sourcetext = sourceText;

    if (numParams > 0)
        parse_fixed_parameters(pstate, paramTypes, numParams);

    query = transformTopLevelStmt(pstate, parseTree);

    if (post_parse_analyze_hook)
        (*post_parse_analyze_hook) (pstate, query);

    free_parsestate(pstate);

    return query;
}
```

执行完自定义的hook之后，一起进入pg_rewrite_query

3. Postgre的词法分析和语法分析

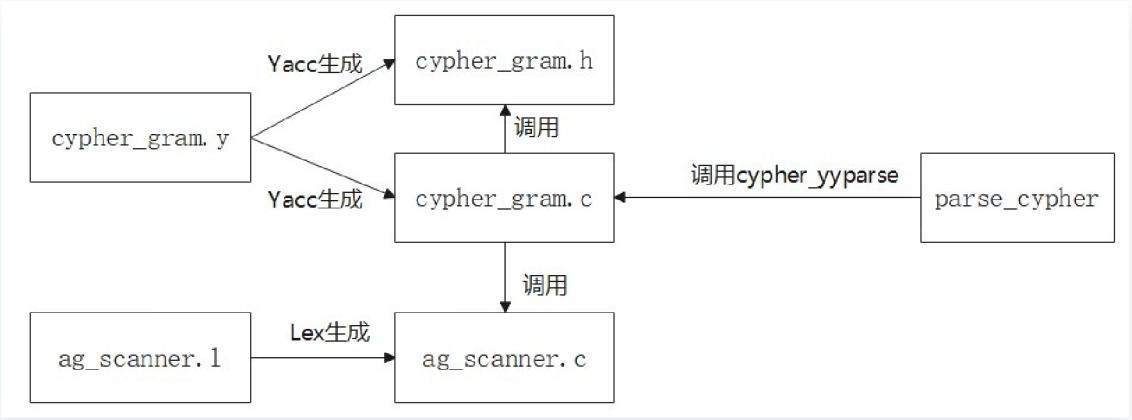
postgre命令的词法分析和语法分析是由Unix工具Yacc和Lex制作的。它们依赖的文件定义在src\backend\parser下的scan.l和gram.y。其中：

- 词法器在文件 scan.l里定义。会根据事先定义的规则（通常使用正则表达式），将源代码的字符序列解析成为有意义的词法单元（tokens），比如关键字、标识符、运算符、常量等。最后词法单元会传递给语法分析器。
- 语法分析器在文件 gram.y里定义。将词法分析生成的词法单元序列转换成为语法树（syntax tree）或者抽象语法树（abstract syntax tree，AST），还包含一套语法规则和触发规则时执行的动作。
- 在raw_parser函数(在src/backend/parser/parser.c下)中，主要通过调用Lex和Yacc配合生成的base_yyparse函数来实现词法分析和语法分析的工作。如下图：



4. Age的词法分析和语法分析

类似于Postgre的词法分析和语法分析流程，如下图：



5. Postgre语义分析

语义分析阶段会检查命令中是否有不符合语义规则的成分。主要作用是为了检查命令是否可以正确的执行。

exec_simple_query函数在从词法和语法分析模块获取了parsetree_list之后，会对其中的每一颗子树调用pg_analyze_and_rewrite进行语义分析和查询重写。其中负责语义分析的模块是在src/backend/parser/analyze.c中的parse_analyze函数。该函数会根据得到的分析树生成一个对应的查询树。

在parse_analyze函数里，会首先生成一个ParseState类型的变量记录语义分析的状态，然后主要调用transformStmt函数处理语义分析任务和生成查询树的任务。

在transformStmt函数里，会先调用nodeTag函数获取传进来的语法树(praseTree)的NodeTag。有关NodeTag的定义在src/include/nodes/nodes.h中。postgresql使用NodeTag封装了大多数的数据结构，把它们封装成节点这一统一的形式，每种节点类型作为一个枚举类型。那么只要读取节点的NodeTag就可以知道节点的类型信息。

因此，随后在transformStmt函数中的switch语句里根据NodeTag区分不同的命令类型，从而进行不同的处理。在这里共有8种不同的命令类型：

```
SELECT INSERT DELETE UPDATE   //增 删 改 查
DeclareCursor  //定义游标
Explain      //显示查询的执行计划
CreateTableAs  //建表、视图等命令
UTILITY      //其它命令
```

对应这8种命令的NodeTag值和语义分析函数如下：

NodeTag值	语义分析函数
T_InsertStmt	transformInsertStmt
T_DeleteStmt	transformDeleteStmt
T_UpdateStmt	transformUpdateStmt
T_SelectStmt	根据条件判断：transformValuesClause 或者 transformSelectStmt或者 transformSetOperationStmt
T_DeclareCursorStmt	transformDeclareCursorStmt
T_ExplainStmt	transformExplainStmt
T_CreateTableAsStmt	transformCreateTableAsStmt
default	作为Unility类型处理，直接在分析树上封装一个Query节点返回

程序就根据这8种不同的命令类型，指派不同的语义分析函数去执行语义分析，生成一个查询树。

6. Age语义分析

Age的语义分析主要集中于analyze_cypher函数：在analyze_cypher函数里，会首先将ParseState转化为cypher_parsestate这种数据结构，然后主要调用transform_cypher_clause函数处理语义分析任务和生成查询树的任务。

在transform_cypher_clause函数中，使用is_ag_node来判断cypher语句的类型，共有10种cypher语句类型和对应的语义分析函数：

cypher语句类型	语义分析函数
cypher_return	transform_cypher_return、transform_cypher_return
cypher_with	transform_cypher_with
cypher_match	transform_cypher_match
cypher_create	transform_cypher_create

cypher_set	transform_cypher_set
cypher_delete	transform_cypher_delete
cypher_merge	transform_cypher_merge
cypher_sub_pattern	transform_cypher_sub_pattern
cypher_unwind	transform_cypher_unwind
cypher_call	transform_cypher_call_stmt