# AgensGraph 添加 SPARQL 功能点工程流程

## 1 词法

**目标文件**：src/backend/parser/scan.l
**作用**：根据语法文件中涉及的所有固定形式短语，添加对应的词法。

以下以 SPARQL_STRING 为例，举例说明添加过程。

1) 在 gram.y 文件 token 定义部分（non-keyword token types，位置大概在 700 行以后）添加 SPARQL_STRING。token 类型一般为 str。

```
707   %token <str>    BLANK NODE LABEL SPARQL_STRING LANGTAG QNAME QNAME_NS Q_IRI REF
```

> 注：关于 gram 中使用的所有数据类型定义，在 gram.y 文件定义段，大概 200+位置，常用的有 ival、str、boolean、node、list、target。

2) 在 scan.l 文件定义段中添加 token 正则表达式，添加位置应在“other”规则前。同时添加部分匹配成功的错误状态 sparql_string_fail，防止产生 backup。

```
413   sparql_string ([""][^""\\\n]*[""])|([''][^''\\\n]*['']) 
414   sparql_string_fail ([""][^""\\\n]*)|([''][^''\\\n]*)
```

> 注：PG 中强制防止 backup，故每一个词法 token 必须添加错误状态，添加方法类似堆栈回退，即考虑 token 最末尾的符号不能成功匹配时的情况。
> 参考：http://web.mit.edu/gnu/doc/html/flex_4.html

3) 在 scan.l 文件规则段中添加 token 对应词法动作，包括错误状态的动作。

```
1174  <SPARQL,SEMI>{sparql_string}          {
1175                                        SET_YYLLOC();
1176                                        yylval->str = pstrdup(yytext);
1177                                        return SPARQL_STRING;
1178                                    }
1179  <SPARQL,SEMI>{sparql_string_fail}         {
1180                                        yyerror("unterminated sparql string");
1181                                    }
```

> 注：
> a) 词法匹配遵循几个规则：(1)长规则优先(2)最早规则优先，故需要注意 token 动作与其他 token 的规则重复，决定其放置位置。
> b) Flex 的起始状态(start state)，允许指定在特定时刻哪些模式可以用来匹配。起始状

态在文件开头%x 行定义，任何时刻词法分析器都在一个起始状态中，且只匹配这个状态所激活的模式。SPARQL 的所有功能点凡是 SELECT 引导的都应该在 *SPARQL* 起始状态下定义。

4) 词法部分编译测试，若无报错，且没有产生任何 backup，则词法部分添加成功。

# 2 语法

**目标文件**：src/backend/parser/gram.y
**作用**：添加语法规则，使系统能够识别对应查询语句，但到这一步，还不能进行对应语义动作。

1) 根据查询语句结构，递归的添加语法规则，直至规则中的每一个部分都是 token。
2) 添加语法的规则动作，例如建立某个结构体、链表等。常用的函数有：makenode(建立结构体)、list_make1(建立链表)、lappend(头插链表)。

注：
a) 语法动作默认为$$=$1。
b) 语法动作中的$1/$2 等中的数字指的是语法结构中的顺序，由 1 开始的顺序号。
参考：https://www.gnu.org/software/bison/manual/bison.html#Semantic-Actions

3) 语法部分编译测试，无报错，且在系统中可以识别预定语句，则语法部分添加成功。

注：
a) 当遇到移进/规约冲突、移进/移进冲突时，可以单独编译 yacc 文件，添加编译选项，查看具体哪一句有冲突。
b) 因 PG 后续编译过程会由 gram.y 自动生成另一个语法文件，所以需要严格遵守语句格式，注意大括号、冒号等位置。

# 3 结构体

1) 根据需要记录的内容，修改 src/include/nodes/parsernodes.h 中的结构体。如果在其中添加了新的结构体，则在 src/include/nodes/node.h 中需要添加枚举条目，添加位置在 NodeTag 结构体最后。
2) 在 src/backend/node/copyfuncs.c 、 src/backend/node/equalfuncs.c 、 src/backend/node/outfuncs.c 中添加新结构体的 copy、out、equal 函数。首先添加 switch 语句中的一个 case 语句增加函数入口，之后在后面的具体函数中添加函数具体内容。

注：copy、out、equal 函数中根据新增结构体的具体内容类型不同调用的函数不同，常用的有 STRING、LOCATION、NODE 几种数据类型。

# 4 语义

**目标文件**：src/backend/parser/parse_sparql_(功能点).c

1) 新建目标 C 程序文件及对应的头文件，并在 src/backend/parser/makefile 中添加编译语句。

```
OBJS= analyze.o gram.o uname_const_decl.o dkpool.o sparql_func.o sparql_p.o sparql_l.o  scan.o parser.o \
      parse_agg.o parse_clause.o parse_coerce.o parse_collate.o parse_cte.o \
      parse_enr.o parse_expr.o parse_func.o parse_node.o parse_oper.o \
      parse_param.o parse_relation.o parse_target.o parse_type.o \
      parse_utilcmd.o scansup.o \
      parse_cypher_expr.o parse_graph.o parse_shortestpath.o \
      parse_sparql_expr.o
```

2) 添加必要的 include 语句，例如 postgre.h 、 parser/parse_node.h 、 nodes/makefuncs.h、nodes/nodeFuncs.h 等。
3) 添加具体实现

> 注：若添加功能点是 SELECT 引导的，则不需要新增文件，改动 src/backend/parser/parse_sparql_expr.c 中的 transform 函数即可。否则需要增加一个新文件，并在 src/backend/parser/analyse.c 中添加对应的入口。