实习文法设计——C语言子集编译器

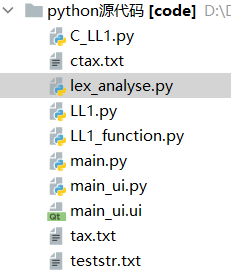
邹奕杨 2020300004080

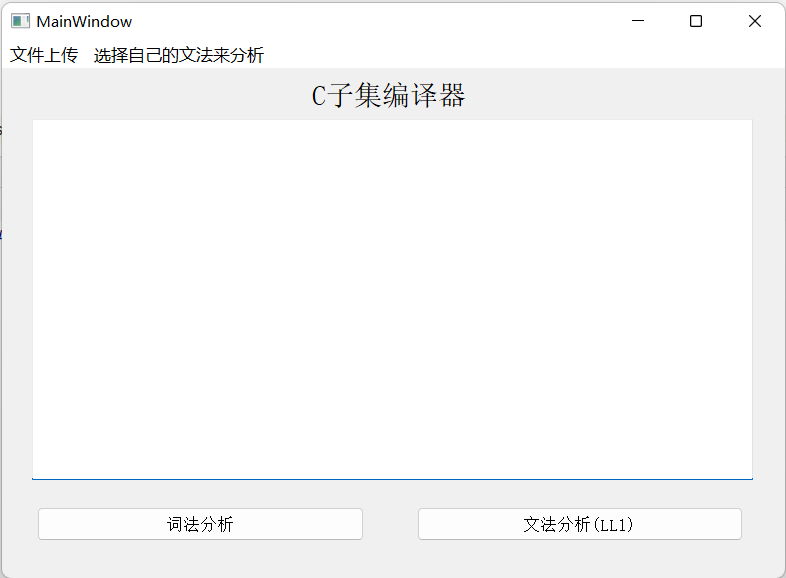
1. **问题描述**

实现 C 语言子集的编译器，我的计划是完成词法分析以及文法分析，语义分析由于一个人工作量有点太大就暂时没有完成。首先实现变量的声明与使用、bool值语言、普通算术运算(加减乘除)、for、while、if-else、if循环语句，并支持printf、scanf两个库函数的文法解析，让编译器能够完成词法分析以及LL1文法分析 **[1]**。同时利用PyQt设计一个简单界面，便于使用者选取需要解析的C源文件并在UI里面方便的进行语法解析。同时，编译器也应该允许用户自行设计文法并在自己设计的文法上面完成文法分析。

1. **整体程序框架**

**整体项目文件架构如下所示：**



其中main.py负责整个项目总控，调用词法分析文件、文法分析文件以及UI界面完成整个项目运行。Lex\_analyse.py为词法分析程序，用于将输入C语言文件切分成若干单词，然后准备输入到文法分析程序中。C\_LL1.py 为文法分析程序，首先构建LL1分析表，然后调用LL1\_function.py里面的Analysis分析函数用于将输入的词法完成文法分析。ctax.txt为我们现在支持的C语言程序的文法，这个是已经给定的。teststr.txt是我写的一段测试程序，用来作为输入，进行后面的词法分析与文法分析。main\_ui.py就是PyQt写的界面，用于展示效果。

UI界面如上图所示，使用者需要上传需要编译的C源文件，然后先点击词法分析再点击文法分析完成编译过程（此时C语言的默认文法在ctax.txt文件里面）。除此之外，用户也可以点击最上方“选择自己的文法来分析”按钮，上传自己的文法，这时候编译所使用的文法就是用户自己上传的。

**我给出的测试程序如下：**

void main(){

    int \_\_sum=0,happy=2;

    bool flag=false;

    for(int i=0;i<n;i=i+1){

        \_\_sum=\_\_sum+i;

    }

    while(\_\_sum>0){

        \_\_sum=\_\_sum/2;

    }

    scanf(falg);

    printf(\_\_sum);

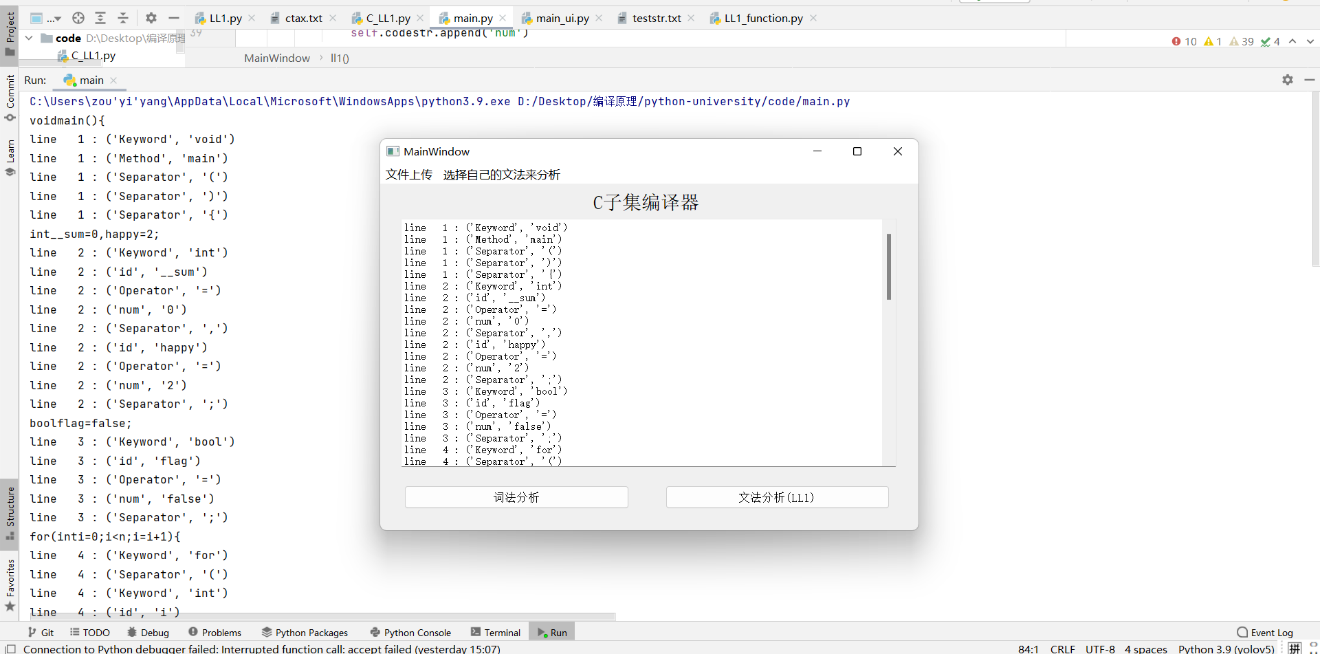
}

1. **词法分析**

在词法分析中，我把字符切分成了下面六类：关键词（Keyword）、运算符（Operator）、分隔符（Seperator）、数字（num）、变量（id）、方法（Method），然后考虑到分割方式比较固定，我一开始设计的是利用多轮转换机进行字符分割识别，后来发现对于现在这种简易情况，我们可以直接利用正则表达式对输入每一行字符进行识别，这样很快就可以匹配并识别出每一类字符，完成词法分析工作。具体正则式如下图所示，这样完成词法分析速度快而且准确率很高。



针对给出的测试程序，运行结果如下所示，我们可以看到每一行输入的C语言字符串都被分割成了若干单词，且每个单词属于确定的六类。然后值得注意的是数字在这里一律被翻译成num，变量一律被翻译成id，这样便于后面文法分析。



1. **文法分析**

程序由主函数构成：

<主函数> -> void main(){复合语句}

<复合语句> -> <语句序列>

<语句序列> -> <语句>;<语句序列>|ε

<语句> -> <赋值语句>|<变量定义语句>|<条件语句>|<循环语句>|<读写语句>

<赋值语句> -> <标识符>=<表达式>

<条件语句> -> if(条件){ 语句序列 }else{语句序列} |if(条件){语句序列}

<循环语句> -> while(条件){语句序列} | for( ; ; ){语句序列}

<读写语句> -> scanf(“<标识符>”) | printf( <字符串> )

**因此C语言文法可以由以下给出：**

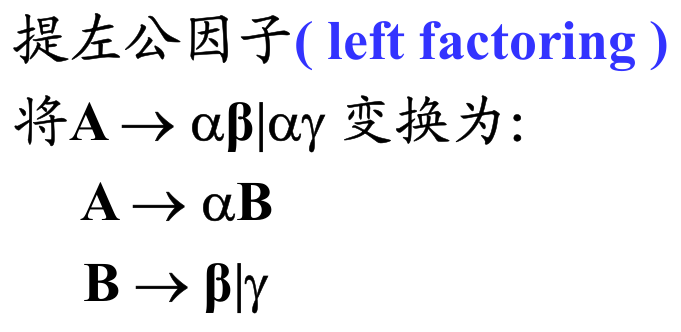
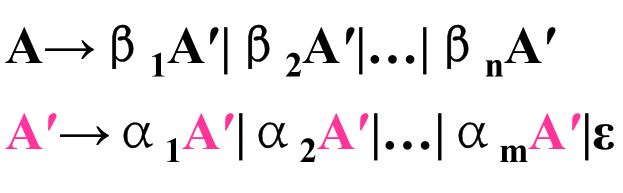
|  |
| --- |
| 1、文法开始：  S->void main(){A}  A->C  C->DC|ε  D->X;|B|U|V;  2、赋值语句：  X->YZ  Y->int|char|bool|double  Z->Z,V|V  V->id=L|id  3、具体运算语句：  L->L+L|L-L|T  T->T\*T|T/T|F  F->(L)|id|num  E->E&&E|H  H->H||H|G  G->FJF|(E)|!E  J-><|>|==|!=  4、循环控制语句：  U->if(E){A} else{A}|while(E){A}|for(YZ;G;V){A}  5、输入输出语句：  B->printf(P);|scanf(id);  P->id|num |

该文法能实现基本赋值语句、运算语句、循环控制语句以及输入输出两个函数的调用，完成基本的文法分析。我们默认大写字母都是非终结符，小写字母组成的单词以及一些符号组成终结符。

**下面是LL1分析表构建以及具体分析过程：**

1. **消除左递归与回溯**

一般采取提取公因式以及左递归改成右递归的方法

**** ****

对文法完成初步回溯、左递归消除，给LL1分析表分析提供条件。在我的程序里我提前加好了一些希腊字母用于面对消除回溯以及左递归时添加字母的需求。

1. **得到Vn,Vt两张表**

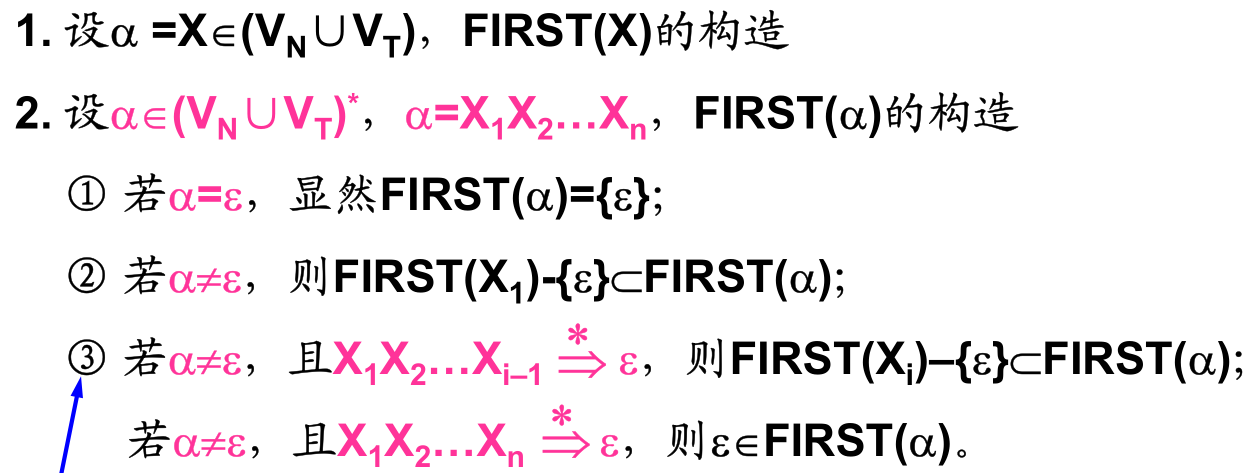
我们拿到已经没有左递归或者回溯的C语言文法表，对每一行进行遍历，以非小写字母作为分割符。所有的大写字母都属于Vn,小写字母的组合或者是其它字符作为Vt。实际运行的Vn以及Vt如下所示：

**Vt:** {'A', 'F', 'H', 'D', 'U', 'C', 'V', 'B', 'L', 'J', 'W', 'N', 'T', 'Γ', 'Θ', 'P', 'S', 'X', 'Z', 'Ξ', 'Y', 'G', 'K', 'E'}

**Vn:** {'scanf', '+', '(', 'printf', '/', '=', 'id', 'double', '|', 'void', 'char', 'num', 'ε', '!', 'for', '>', '&', ' ', ',', 'while', '<', '}', 'else', 'bool', 'if', '\*', ')', 'main', '-', '{', ';', 'int'}

1. **得到First集、Follow集**

First集的求法比较套路化，就是每一轮遍历每一行，然后把没有添加的非终结符添加到终结符的First集合内。如果某一轮没有任何First集合被修改，此时整体程序运行结束。Follow集就是根据已经求出来的First集合，然后也是每一轮都进行迭代，直到没有改变才停止。



求First集合



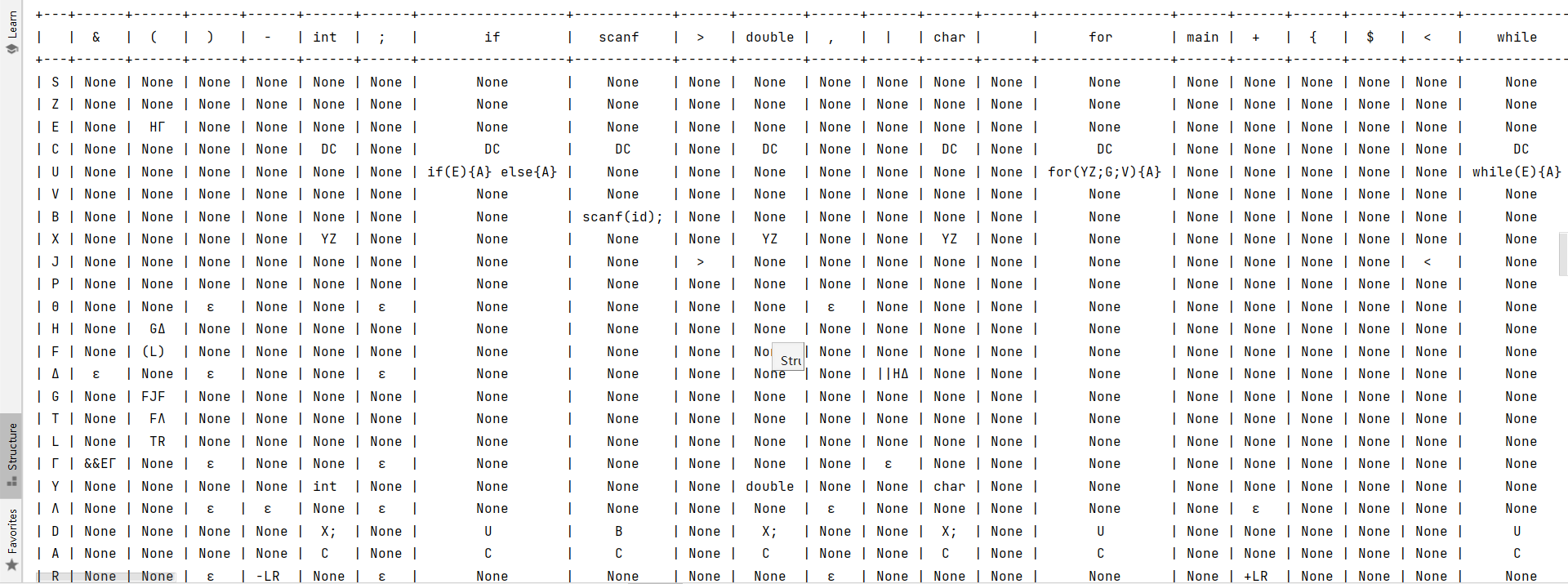
求Follow集合

1. **根据First,Follow集构建LL1分析表**

LL1分析表按照以下方式构建:

对于每一个非终结符S，首先将它的First集内的元素加入LL1分析表中，然后如果S->ε,那么就把S的Follow集内元素加入到LL1分析表中。如果出现了冲突，那么我根据左结合选择优先加入First集合内的元素，其次才是Follow集。

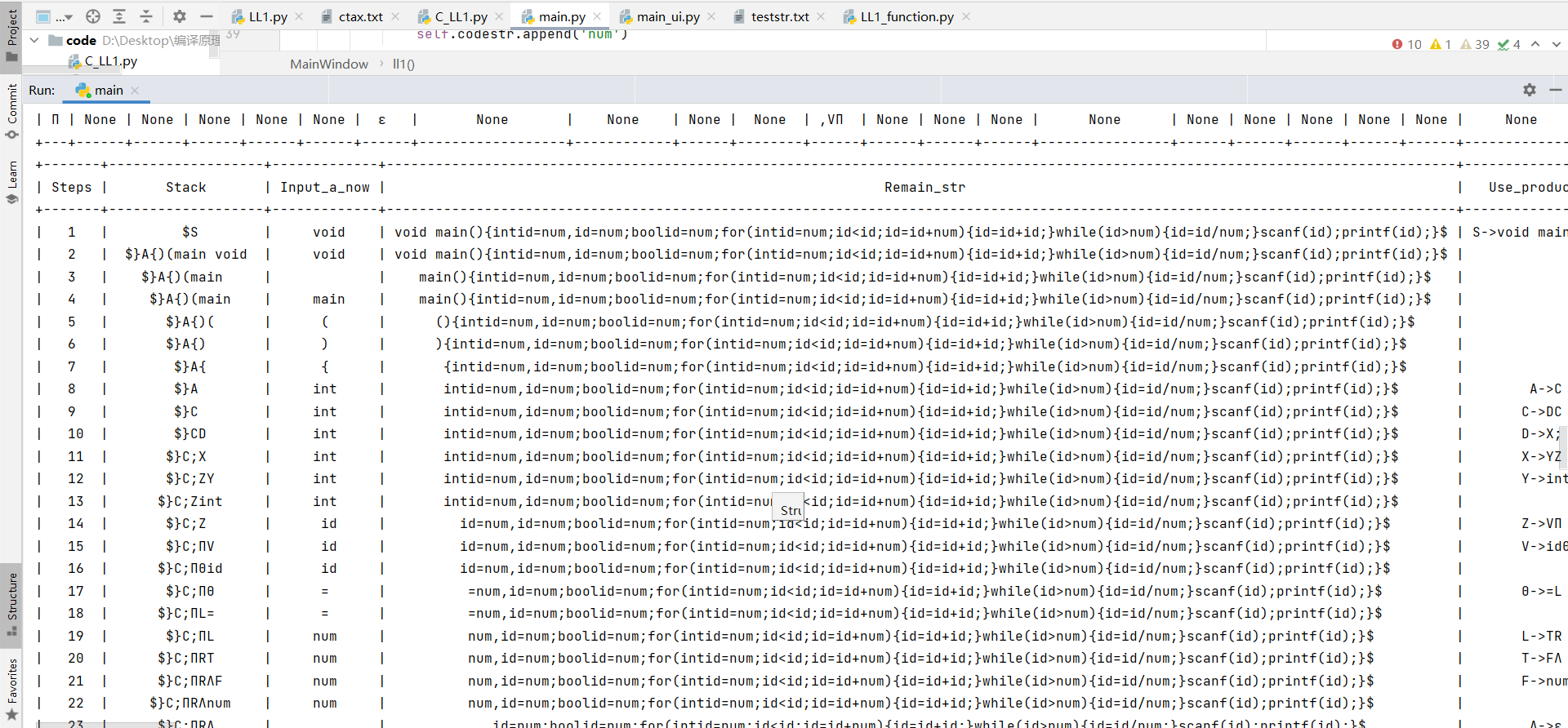
部分LL1分析表如下所示：



在代码中我们调用了PrettyTable库，用于画出比较好看的LL1分析表。

1. **根据LL1分析表完成文法解析**

首先在输入字符首尾加上终结符‘$’，然后利用stack存放当前文法分析的字符串，然后每一轮都要将栈顶元素弹出，根据LL1分析表选择合适的表达式，将表达式逆序压入栈中。如果LL1分析表里面没有对应项就会直接报错。解析具体过程大致如图所示：





最后完成整个文法解析过程。

1. **总结**

由于时间比较仓促，最后没能实现语法解析比较遗憾，但是通过自己动手完整实现了整个词法解析以及LL1文法解析后我对编译原理的理解程度加深了很多，整体动手能力与代码实现能力长进了不少。在具体实现的过程中，我遇到了好多问题：包括字符串读取错误，First,Follow集输出错误、文法分析莫名的错误等等问题，通过查阅资料以及比对老师之前PPT上面的内容，我把细节问题一一修改完成，直到自己完全彻底掌握了每一处细节才将整个项目完完全全的做了出来。总体来说收获挺大的。

1. **参考资料**
2. 手把手教你实现一个C编译器（<https://lotabout.me/2015/write-a-C-interpreter-0/>）
3. 编译原理课程设计-C语言子集编译器（<https://blog.csdn.net/weixin_44424450/article/details/113092392>）
4. 编译原理课程设计报告--C语言子集编译器设计（<https://max.book118.com/html/2018/0514/166240038.shtm>）