关于文言文编程语言 Java 编译器

实现原理

修正版-二期

撰写: Changcun Lu (Magic Lu)

前 言

文言文编程语言(Wenyan Programming Language)是由 LingDong Huang 先生创立的编程语言项目，比起其语言的实际作用，其更主要的作用在于弘扬中华优秀文化，传播古文学，LingDong 的文言文功底有目共睹，加之其有力的技术，创建了这个编程语言，并且很快火遍国内外。

在我发现这款语言的时候，内心的第一感觉是“amazing”，令人吸引的就是其那优美不失违和感的语法，且可以运行在计算机上，想想就是十分激动的。当我写下“吾有一言”时，便感受到此语言的魅力所在。至于文言文的语言细节，大家可以关注 LingDong Huang(今 wenyan-lang 团队)的项目，也感谢 antfu 先生的支持。

身为一名高二学生，是 wenyan-org 年龄较小的一员，但依然不削减我对于这个语言的热情 和中国的文言文精华。因此我决定开始了对于文言文 java 编译器的书写，使得文言文项目继续发扬光大。目前该项目归属于”wenyan-lang”团队。

目录

第一章 编译的具体流程

由于作者尚在高中阶段，对于计算机没有很系统的学习，但又因为对于这款语言的热爱，因此我自己探究出编译的方式，原本是持着很忐忑的心理，首先就是我这种方式会不会导致项目最终无法维护，再者就是效率问题(这点我是在慢慢优化),为了切合原版编译器的特性(由于编程语言的目标语言是 JavaScript，因此拥有很多语言没有的神奇特性)。于是综合考虑下，我将其翻译成 groovy 来运行在上面，以符合其弱类型的部分(尽管文言文后来是强类型语言)。

我采用了「切分」和「匹配」来编译这款语言的，也是我这个编译器的最核心的两个理念。

切分

首先将语句切分为一个个子段。这里的代码我放在 LexerUtils.scala 的 wenyanLexer 方法。

这是一段很长的方法，也是切分的核心代码，其中主要思路是语法优先级的思路。首先在一个巨大循环体里，遍历语句每一个字符。

第一级 字符串匹配。匹配‘「’，‘「’在文言文是声明的名称及字符串的标志。因此保证这里面的内容不会被当成语法被解析，而是被“保护住”。通过开始和闭合来判断是否开始字符串/变量名和结束字符串/变量名，因此匹配“「”为加一，意思为进入第一个

域，当遇到“」”减一，意思是脱离的第一个域，count 为 0 时，则代表脱离字符串/变量名域，之后代表字符串被独立的剥离到子句。isAppend 设置为 true，代表本次拼接完成，进入下一次拼接，在遍历字符串域时，对于整体的字符串的下标也会添加。比如我说「「hello,world」」，你好，在读取「「hello,world」」时，index 也会添加，因此 isAppend 设置为 true 进入下一次循环，则是从“，”开始。

第二级，注释匹配。在读取到“曰”的时候，代表注释开始，之后如同字符串一样的步骤，把注释字符串独立剥离出来，注释声明句独立剥离出来。

第三级，数字匹配，由于文言文的数字很是特殊，因此检验数字时，还要检验下一个字符是否还是数字，知道下一个字符不是数字了，则停止对于数字的拼接，isNumber 方法实际上是个阅读字典的方法，这段代码负责判断后面是否有数字

```
index+1<string.length&&isNumber(string(index+1))
```

图 1-1 数字的向后判断

第四级，分隔符匹配，文言文允许和古汉语一样，没有标点，但是也允许有标点，因此这里需要对于标点进行匹配，若发现标点，直接作为独立的字句，直接进入下一个字句的组合。

最后一级，canMatch 阶段，这里将合并的子句进行判断，看看是否符合字典的任何其中一个语法。如果不符合，则保留这个子句，进入下一个循环继续拼接，直到符合为止，如果符合，就

将子句加入到子句池，然后创建新的 StringBuilder，进行下一个字句的拼接。如果在全部代码解析完毕，发现还有没放入子句池的子句，则抛出语法错误的异常，这里恰到好处的第一个字符就是语法错误的终点。

匹配

在切分过程中，canMatch 实际上进行了一次匹配，从字典中寻找语法源，匹配到则放入语句池。

在匹配过程中，通过「流程」将子句匹配有序的进行，每当在流程中匹配成功了一个子句，则代表一条指令会被成功编译。之后这个匹配成功的子句会从子句池删除，之后返回的是编译结果，会放入结果池。

而匹配的顺序是通过 CompileFactory 对象进行了，CompileFactory 会将创建的流组合成一个生产线，之后一条语句会在生产线轮流加工

```
this.factory = new StreamBuilder(this)
        .put(new VariableCompileStream(this))
        .put(new CommentCompileStream(this))
        .put(new ControlCompileStream(this))
        .put(new MathCompileStream(this))
        .put(new FunctionCompileStream(this))
        .put(new ArrayCompileStream(this))
        .put(new TryCompileStream(this))
        .put(new ObjectCompileStream(this))
        .build();
```

图 1-2 WenyanCompilerImpl 的 Factory 部分

语句进入后，按照这个顺序一次加工，Variable 加工变量相关的语句，Comment 加工注释相关的语句，Control 加工条件控制语句，

Math 加工运算符语句，Function 加工函数语句，Array 加工数组语句。Try 加工异常处理语句，Object 加工对象语句。所有语句不是第一次通过流就全部加工完毕，而是通过一次全部流，只有一个语句被加工，且只被一个流加工，在其他流中只是经过，但是具体加工全部跳过了。

在每个流中，还有具体的工作分支，每条语句只会在其中一个分支被加工，然后返回结果，后面的流将不会再经过，之后会移到下一个字句，继续上述步骤。

如果这条子句没有任何被匹配的，则返回 success: false,之后会抛出语法错误的异常。具体流程见下文。

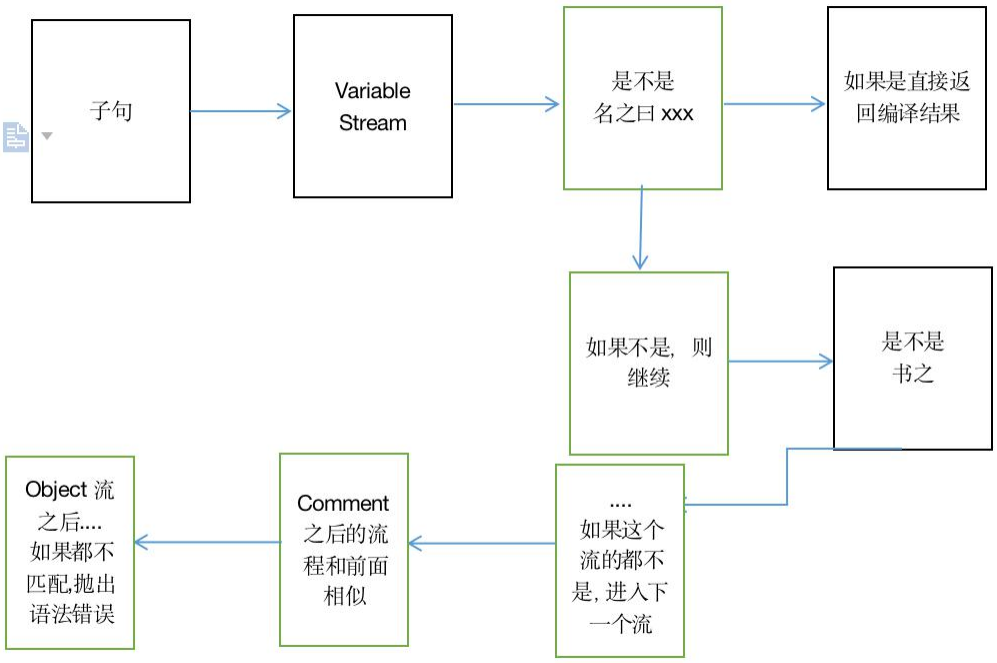

图 1-3 字句的翻译编程

关于编译流程的相关类，方法，字段

- CompileStream.java
    - ArrayCompileStream
    - CommentCompileStream
    - ControlCompileStream
    - FunctionCompileStream
    - MathCompileStream
    - ObjectCompileStream
    - TryCompileStream
    - VariableCompileStream
- LexerUtils.scala
- CompileFactory.java
- StreamBuilder.java
- SyntaxException.java
- WenYanCompilerImpl.java

第二章 WenYan2Number 的实现

Wenyan2Number 是将文言文数字编译为阿拉伯数字的模块，是

文言文语言数学模块的主要部分。

java 版编译器通过「数字字典」和「数字树解析重组」来将文言文数字解析为阿拉伯数字。

首先 NumberTree，将标准的文言文语言数字化分为 1000 100 1 为单位和计数单位的模块，比如一万两千四百，会切分为一和万和两千四百。之后在挨个解析进行计算。

```
BigDecimal convertToNumber(){
    BigDecimal result = 0
    for(int i = 0;i<nodes.size();i+=2){
        result += GroovyUtils.getNumber(nodes[i].value) *
         GroovyUtils.getNumber(i+1<nodes.size()?nodes[i+1].value:"一")
    }
    (result + floatValue)*prefix
}
```

图 2-2 数字树转化为数字的方法

节点就是被切分后的模块。通过如图代码的方式组合数字。

之后，在之间有一个 getNumber 方法，也是其中的核心部分，它负责将无单位数字和有单位的千单位以下的数字转化为阿拉伯数字。

```
for(int i = 0;i<chars.length;i++){
    if(chars[i].toString() == "又"){
        continue
    }
    if(chars[i].toString() == "負"){
        isNot = -1
        continue
    }
    if(i+1<=chars.length-1){
        if(prefixs.contains(chars[i+1])){
            BigDecimal max = prefixs.get(chars[i+1]).get()
            if(maxNumber == 0||max < maxNumber) {
                result += max *
                      numbers.get(chars[i]).get()
                if(maxNumber == 0)maxNumber = max
```

```
            }else{
                result += numbers.get(chars[i]).get()
                result = result * max
                maxNumber = max
            }
            i++
        }else{
            result += numbers.get(chars[i]).get()
        }
    }else{
        result += numbers.get(chars[i]).get()
    }
}
```

图 2-2 计算万位以下数字的方法

是通过字典检索和数位计算得出结果，具体的算法是十分简单的，这里不过多阐述了。

总之是采用切分数字与单位来分开计算，使得文言文数字不会发生巨大的歧义，这也是后来 java 文言文编译器改进的地方。先前则忽略了这个问题。

这里的数字都在 WenYanLib.scala 中。

```
val prefixAfter = MMap[Char,BigDecimal0](
  '負' -> new BigDecimal0(-1),
  '百' -> new BigDecimal0(100),
  '千' -> new BigDecimal0(1000),
  '萬' -> new BigDecimal0(10000),
  '億' -> new BigDecimal0(1E8),
  '兆' -> new BigDecimal0("1E12"),
  '京' -> new BigDecimal0("1E16"),
  '垓' -> new BigDecimal0("1E20"),
  '秭' -> new BigDecimal0("1E24"),
  '穰' -> new BigDecimal0("1E28"),
  '溝' -> new BigDecimal0("1E32"),
  '澗' -> new BigDecimal0("1E36"),
  '正' -> new BigDecimal0("1E40"),
  '載' -> new BigDecimal0("1E44"),
  '分' -> new BigDecimal0("1E-1"),
```

```
  '釐' -> new BigDecimal0("1E-2"),
  '毫' -> new BigDecimal0("1E-3"),
  '絲' -> new BigDecimal0("1E-4"),
  '忽' -> new BigDecimal0("1E-5"),
  '微' -> new BigDecimal0("1E-6"),
  '塵' -> new BigDecimal0("1E-7"),
  '埃' -> new BigDecimal0("1E-8"),
  '渺' -> new BigDecimal0("1E-9"),
  '漠' -> new BigDecimal0("1E-10")
)
val prefixs = MMap[Char,BigDecimal0](
  '廿' -> new BigDecimal0(20),
  '卅' -> new BigDecimal0(30),
  '卌' -> new BigDecimal0(40),
  '皕' -> new BigDecimal0(200),
  '十' -> new BigDecimal0(10),
).addAll(prefixAfter)


//二分三釐八毫八絲三忽八微

val numbers = Map[Char,BigDecimal0](
  '〇' -> new BigDecimal0(0),
  '零' -> new BigDecimal0(0),
  '一' -> new BigDecimal0(1),
  '二' -> new BigDecimal0(2),
  '两' -> new BigDecimal0(2),
  '三' -> new BigDecimal0(3),
  '四' -> new BigDecimal0(4),
  '五' -> new BigDecimal0(5),
  '六' -> new BigDecimal0(6),
  '七' -> new BigDecimal0(7),
  '八' -> new BigDecimal0(8),
  '九' -> new BigDecimal0(9),
)
```

图 2-3 数字字典

Wenyan2Number 的相关方法和类,字段:

- WenYanLib.scala
    - prefixAfter
    - prefixs

        - numbers
        - getNumber
        - numbersGet
    - NumberTree.groovy
        - Node
        - NumberTree
- ScalaUtils.scala
    - containsCommonNumber
- VariableCompileStream.java
    - getNumber
- GroovyUtils.groovy
    - getNumber
    - getMax

第三章 宏的实现

文言最新版本添加了宏展开的概念，类似于 c 语言，c++语言的宏，可以进行简单的替换。在刚开始，这为我编写编译器又带来了一大挑战。诚然，即使现在我对我的预编译器依然是不太满意的，不过也是目前比较合理稳定的设计方案(我的经验问题也开始暴露了出来)。

```
或云「「子曰「甲」」」。

蓋謂「「施「子曰」於「甲」」」。
```

图 3-1 宏的语法

类似于这样的语法，事实上，我和原版编译器实现的不太相同，我除了不能使用**关键字**以外，几乎没有任何内容的限制。文言文的宏不会对字符串，变量名进行展开。

并且文言文支持跨文件的宏，而我需要考虑如何编译后也可以生效，因此对于此我用@Define 注解作为扩展的语法。

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface Define {

   String before();

   String after();


}
```

图 3-2 Define 源代码

然后我总共考虑了这几点

1. 对于已经编译的文件，其中的宏如何生效
2. 如何不对字符串和变量名产生效果
3. 如何处理参数
4. 如何在处理宏之后再进行分词
5. 如果导包也被宏定义了

考虑第四点是由于我不想颠覆我之前的代码，因此作出了妥协。第一个问题，我采用@Define 和@Defines 注解来记录这个文件定义的宏，因此在编译源码文件同时链接库的时候，首先确认有宏定义的这个类是否存在，存在则读取里面的注解，如果没有，那么就读取同目录的文言源码文件。因此在加载一个文言文源代码时，首先读取其中的导包操作，把相关联的内容读取，或者关联的源码提前编译，以免发生 ClassNotFound 的问题，或者找不到预先定义的宏的问题

```
private void importMacro(List<String> imports){
   try {
      Map<String,String> macros = new HashMap<>();
      for (String im : imports) {
         if (Utils.classExists(im)) {
            Class<?> clz = Class.forName(im);
            Defines definesObj = clz.getAnnotation(Defines.class);
            if(definesObj!=null) {
               Define[] defines = definesObj.value();
               for (Define define : defines) {
                  macros.put(define.before(), define.after());
               }
               initMacro(macros);
            }
         }else{
            String file = compiler.getSourcePath()+"/"+im.replace(".",
File.separator)+".wy";

initMacro(getMacroMapping(compiler.getWenYanCodeByFile(new
File(file))));
```

```
            }
        }
    }catch (ClassNotFoundException| IOException e){
        compiler.getServerLogger().error("",e);
    }
}
```

图 3-3 importMacro

第二个问题，如何不对字符串和变量名产生影响。

这里我自己写了个 replace 方法，选择不替换字符串和变量域的内容

```
static String replace(String before,String replaced,String after,List
range){
    int close = 0
    StringBuilder afterBuilder = new StringBuilder()
    StringBuilder now = new StringBuilder()
    int afterStart = 0
    for(s in before){
        if(s == range[0]){
            close ++
        }
        if(s == range[1]){
            close --
        }
        now.append(s)
        if(s == replaced[afterStart]){
            afterStart++
        }else{
            afterStart = 0
            afterBuilder.append(now)
            now = new StringBuilder()
        }
        if(afterStart == replaced.size()){
            if(close == 0){
                afterBuilder.append(after)
            }else{
                afterBuilder.append(now)
            }
            afterStart = 0
            now = new StringBuilder()
        }
```

```
    }
    return afterBuilder
}

static String replaceWithOutString(String before,String replaced,String
after){
    replace(before,replaced,after,["「","」"])
}
```

图 3-4 replace

第三个 处理参数

这里算是一个比较复杂的地方，我想到的方式就通过正则，然后一个正则对应着一个表，这个表存着参数名称，之后替换的时候，挨个通过名称替换。

这里就采用了一些特别憋屈并且很转脑筋的逻辑，尽管还是啃着硬骨头写下去了，事实证明，是成功的。

在开始 我进行这几个操作

```
Map<String,String> get = getMacroMapping(wenyanCode);

//初始化本文件的 macro
initMacro(get);

//导入其他文件 macro，前提是 import 不是宏的
List<String> strs = LexerUtils.wenYanLexer(wenyanCode);


List<String> imports = getImportsWenYan(strs);

importMacro(imports);
```

图 3-5 前面的预处理

getMacroMapping 的作用是获得里面所有的有关宏的语句

```
private Map<String,String> getMacroMapping(String wenyanCode){
    Map<String,String> map = new HashMap<>();
```

```
    List<String> macros = getMacro(wenyanCode);
    for(int i = 0;i<macros.size();i+=2){
        String first = macros.get(i);
        String end = macros.get(i+1);
        first = first.substring(first.indexOf("
「「")+2,first.lastIndexOf("」」"));
        end = end.substring(first.indexOf("「「")+5,end.lastIndexOf("」」
"));
        map.put(first,end);
    }
    return map;
}
```

图 3-6 getMacroMapping

initMacro 则是解析获得的宏信息，这里就是比较绕脑筋的存数据，我最后把他们存到 WenYanLib 的语法字典，这样我就可以通过 wenyanLexer 切分断句。这里存的 names 和 values 才是实际在展开时候用到的

```
private void initMacro(Map<String,String> get){
    macroMapping.putAll(get);
    Map<String,List<String>> names = getMacroNames(get);
    macroNames.putAll(names);

    macroPatterns.putAll(getMacroPatterns(get,names));

    macroPatterns.forEach((x,y)->WenYanLib.syntaxs().put(y,x));

}
```

图 3-7 initMacro

就是核心的 getMacroPatterns，这个产生正则的作用不是用于宏的展开，而是用于分词和匹配。

```
private Map<String,String> getMacroPatterns(Map<String,String>
macros,Map<String,List<String>> names){
    Map<String,String> macrosPatterns = new HashMap<>();
    macros.forEach((macro,value)->{
        List<String> macroNames = names.get(macro);
        String mac = macro;
```

```
        for(String name : macroNames){
            mac = mac.replace(name,"[\\s\\S]+");
        }

        macrosPatterns.put(mac,macro);
    });
    return macrosPatterns;
}
```

图 3-8 igetMacroPatterns

之后展开，则是先匹配正则，之后通过我自己写的替换来替换各个名字对于的参数，实际上替换的是 names 对应着 values

```
private void expansion(List<String> strs){
    for(int i = 0;i<strs.size();i++){
        int index = i;
        String str = strs.get(i);
        macroPatterns.forEach((k,v)->{
            if(str.matches(k)){
                List<String> ns = macroNames.get(v);
                List<String> values =
Utils.getStrings(WenYanLib.VALUE(),str);

                String result = macroMapping.get(v);
                for(int j = 0;j<values.size();j++){
                    result =
GroovyUtils.replaceWithOutString(result,ns.get(j), values.get(j));
                }
                strs.set(index,result);
                return;
            }
        });
    }
}
```

图 3-7 宏展开的代码

解决最后俩问题就是最后这几段代码

```
//重新解析
List<String> format = LexerUtils.wenYanLexer(getString(strs));

imports = getImportsWenYan(format);
```

```
//导入之前宏定义的导入语句的宏
importMacro(imports);

expansion(format);

format = LexerUtils.wenYanLexer(getString(format));

//编译 import 的文件
compileImports(imports);
```

图 3-8 反复处理

就是重新导入，然后重新搞一遍，当然这里就是我不太满意的地方，效率有点低，但是目前还没有更好的方案。

这就是宏定义的基本实现内容，它是在源码编译之前发生的。

当五个问题解决了，宏也就解决了。在之前，这属于文言文的实验内容。

关于宏相关的方法，类名，字段

- PrepareCompiler.java
- Define.java
- Defines.java

第四章 切合 JS 语法的类库

对于文言文比较困难的地方，还是尽量兼容 js 编译器下的文言文代码，不过经过大量努力，除了使用 js 自身特性的文言文语言，基本都是可以通过并正常允许的。

虽然对于很痛苦的地方，比如图灵完备的源代码，数组是依靠 js 特性的，因此我为了与其兼容，用 Map 实现了一个 JSArray，使得其得以运行图灵完备的代码，当时写完这个的时候，我还为自己的天才想法而骄傲(诚然这是个无聊的想法)。

之后对于一系列为了兼容之前文件，我做了一堆的语法糖和辅助类库, 比如 js 中取余可以忽略小数的影响, 比如 3%1.2 = 3,1.2 % 1.2 = 1.2 这样子，但是 groovy 是不可以的，因此我也自己写了个取余，为了符合他们的算法。

基本为了兼容有关的库都在 cn.wenyan.compiler.lib 里,也不过多赘述了。

文言文的内部函数则是通过闭包的形式实现的，为了使得闭包可以进行递归，所以采用了先声明，后实现函数的方法，实现了内部函数，对于 js 中，允许内部函数参数名称和外部函数的相同，则是采用了「函数签名」，对每个函数加一个 Label 来标记，使得在后期编译，通过参数名_域名来替代重名的参数。

例如:

```
吾有一術。名之曰「外部」。欲行是術。必先得一言。曰「参数一」。乃行是術曰。
        吾有一術。名之曰「内部」。欲行是術。必先得一言。曰「参数一」。乃行是術曰。
```

这样子，我们会将内部的参数一换为参数一_1_外部

对于前面的函数使用后面实现的函数，java 则是不可以的，因此我添加了特别的语法

```
吾有一術。名之曰「函数」。
```

就会预先定义一个函数，之后这个函数即使后面实现也是可以使用的。

对于与 java 代码的链接，这个会在项目管理的章节说到。

第五章 主要函数的注明

前面四章讲述了文言文 java 编译器的主要思想, 此外, 文言文 java 编译器还有一些附带的功能。

多语言的编译

为了扩展 java 编译器的适用范围，我在最新版本中，做了一项巨大的改变，添加了 script.libs 包，这里允许你将文言文编程语言编译成各种各样的目标语言，只要你根据其中的标准书写编译目标语言，我叫他为 CDL(Compile Design Language),如翻译成 groovy 的内容:

```
enum Language {

    GROOVY(
            [
                    (Syntax.VAR_DEFINE)          : "def $NAME = $VALUE",
                    (Syntax.FOR_NUMBER)          : "for(_ans$INDEX in
1..$RANGE){",
                    (Syntax.COMMENT)             : "/*$COMMENT*/",
                    (Syntax.FOR_EACH)            : "for($ELEMENT in
$ARRAY){",
                    (Syntax.FOR_END)             : "}",
                    (Syntax.IF)                  : "if($BOOL){",
                    (Syntax.IF_END)              : "}",
                    (Syntax.IF_BREAK)            : "if($BOOL)break",
                    (Syntax.WHILE_TRUE)          : "while(true){",
                    (Syntax.ELSE)                : "}else{",
                    (Syntax.BREAK)               : "break",
                    (Syntax.RETURN)              : "return $VALUE",
                    (Syntax.FUNCTION_END)        : "}",
                    (Syntax.DEFINE_FUNCTION)     : "def
$NAME($ARGS){",//
                    (Syntax.FUNCTION_ARGS_SPLIT)  : ",",
                    (Syntax.IMPORT)              : "import $LIB",
                    (Syntax.IMPORT_STATIC)       : "import static
$LIB.$METHOD",
                    (Syntax.IMPORT_STATIC_SEPARATE): "true",
                    (Syntax.IMPORT_SPLIT)        : "null",
```

```
                (Syntax.MATH_ADD)            : "$NAME+$VALUE",
                (Syntax.MATH_LESS)           : "$NAME-$VALUE",
                (Syntax.MATH_MULTI)          : "$NAME*$VALUE",
                (Syntax.MATH_EXCEPT)         : "$NAME/$VALUE",
                (Syntax.MATH_REMAIN)         : "mod($NAME,$VALUE)",
                (Syntax.BIGGER)              : ">",
                (Syntax.SMALLER)             : "<",
                (Syntax.EQUAL)               : "==",
                (Syntax.AND)                 : "&&",
                (Syntax.OR)                  : "||",
                (Syntax.PRINT)               :
"println($VALUE .toString())",
                (Syntax.NEGATE)              : "!",
                (Syntax.CHANGE)              : "$NAME = $VALUE",
                (Syntax.REPLACE_ARRAY)       :
"$NAME[getIndex($INDEX)] = $VALUE",
                (Syntax.STRING)              : "\"\"\"",
                (Syntax.ARRAY_ADD)           : NAME+".add($VALUE)",
                (Syntax.INNER_FUNCTION)      : "def $NAME \n $NAME =
{\n $ARGS->",
                (Syntax.INNER_FUNCTION_NO_ARGS): "def $NAME \n $NAME =
{\n ",
                (Syntax.TRUE)                : "true",
                (Syntax.FALSE)               : "false",
                (Syntax.NOT_BIG_THAN)        : "<=",
                (Syntax.NOT_SMALL_THAN)      : ">=",
                (Syntax.NEGATE_EQUAL)        : "!=",
                (Syntax.SLICE)               :
"ArrayUtils.slice(getArray($NAME))",
                (Syntax.SIZE)                :
"ArrayUtils.length(getArray($NAME))",
                (Syntax.RUN_FUNCTION)        : "def $VALUE =
$NAME($ARGS)",
                (Syntax.OBJECT_INNER)        : ".",
                (Syntax.NUMBER_SUGAR)        : "$NAME",
                (Syntax.STRING_APPEND)       : "+",
                (Syntax.ARRAY_GET)           :
"getArray($NAME)[getIndex($INDEX)]",
                (Syntax.INT_TYPE)            : "BigDecimal",
                (Syntax.STRING_TYPE)         : "String",
                (Syntax.ARRAY_TYPE)          : "JSArray",
                (Syntax.BOOL_TYPE)           : "boolean",
                (Syntax.VAR_TYPE)            : "def",
                (Syntax.DOUBLE_TYPE)         : "double",
```

```
                (Syntax.FUNCTION_ARG_DEFINE)   : "$TYPE $NAME",
                (Syntax.NULL)                : "null",
                (Syntax.DEFINE_ARRAY)         : "new JSArray()",
                (Syntax.DEFINE_INT)           : "0",
                (Syntax.DEFINE_STRING)        : "''",
                (Syntax.IMPORT_WITH)          : ("import
cn.wenyan.compiler.lib.*\nimport static
"+ArrayUtils.name+".getArray\nimport static
"+ArrayUtils.name+".getIndex\nimport static "+ MathUtil.name+".mod"),
                (Syntax.NOT)                 : "!",
                (Syntax.REQUIRE_SCRIPT)       : "",
                (Syntax.CONTINUE)             : "continue",
                (Syntax.ELSE_IF)              : "}else ",
                (Syntax.CONCAT)               :
(NAME+".putAll($VALUE)"),
                (Syntax.TRY)                 : "try{",
                (Syntax.THROW)                : ("Exception $NAME = new
Exception($EXCEPTION)\nthrow $NAME"),
                (Syntax.CATCH)               : "}catch($NAME){",
                (Syntax.EXCEPTION_IF)         :
(NAME+".message.equals($EXCEPTION)"),
                (Syntax.CATCH_END)            : "}",
                (Syntax.SHELL_VAR)            : "$NAME = $VALUE",
                (Syntax.DEFINE_OBJECT)        : "[:]",
                (Syntax.DELETE)               :
(NAME+".remove($INDEX)"),
                (Syntax.OBJECT_TYPE)          : ("Map "),
                (Syntax.DEFINE)              : "def $NAME",
                (Syntax.GIVE_FUNCTION)        : "$NAME = {\n $ARGS->",
                (Syntax.DEFINE_GIVE_FUNCTION)  : "def $NAME = {\n
$ARGS->"
        ],new GroovyCompiler(),new GroovyPrettyCode()
    );
```

主要的内容是语法映射，class 字节码编译器，代码格式化。

```
WenYanCompilerImpl compiler = new WenYanCompilerImpl(false,
Language.GROOVY);
```

之后如图这样，我们可以使用这个标准，其他语言标准只支持输出代码文件，其他的功能是不支持的。

如图的一些变量，基本的意义不说也是可以明白的。

自定义指令的解析

这里编译器中指令的主要作用是将信息添加到 CompilerConfig，之后会在 WenYanCompilerImpl 的 init 整体解析。

```
public int executeCommand(String[] args){
    CompilerConfig compilerConfig = new CompilerConfig();
    for(int index = 0;index<args.length;index++){
        List<String> arr = new ArrayList<>();
        Command command = compileCommand.get(args[index]);
        if(command!=null){
            int len = command.getArgsLength();
            if(len == -1){
                for(int j = 0;index<args.length;j++,index++){
                    arr.add(args[index]);
                }
            }else {
                for (int j = 0; j < len; j++) {
                    index++;
                    arr.add(args[index]);
                }
            }
            command.execute(arr.toArray(new String[0]),compilerConfig);
        }else{
            if(index == 0){
                throw new CommandException("吾亦不知君欲何");
            }
        }
    }

    return compiler.init(compilerConfig);
}
```

而指令有选项，参数，这里的 length 规定了选项后参数的长度，当为-1 时，则是无限长度，但是只能放在最后面，否则会发生歧义。

这里命令解析器我写的是比较简单的，但是恰好可以供应 java 所使用。

WenYanShell 功能

WenYanShell 实际上就在 Groovysh 的基础上实现的，而对于 shell 部分，也做了 shell 功能的开启和关闭的部分，为了使得编译的

结果和 Groovysh 相适应。WenYanRuntime 是在 WenYanShell 的基础上写的，用于直接运行文言文代码。

```
public static void run(){
    int index = 0;
    int line = 0;
    JlineReader reader = new JlineReader(true,true);
    WenYanShell shell = new WenYanShell();
    ServerLogger logger = shell.compiler.getServerLogger();
    logger.info("WenYan Lang - Shell: @CopyRight wy-lang.org v 1.0");
    String[] imports =
shell.compiler.getLanguageType().getSyntax(Syntax.IMPORT_WITH).split
("\n");
    for(String s : imports)
        shell.run(s);
    String prefix = ">";
    StringBuilder builder = new StringBuilder();
    while (true){
        String code = reader.readLine(prefix);
        if(code.matches("(:)[a-zA-Z?\\\\.]+")){
            shell.run(code);
            continue;
        }
        String returned = shell.compiler.compile(code,false);
        if(LexerUtils.trimWenYanX(returned).startsWith("import")){
            String[] imps = returned.split("\n");
            for(String imp : imps){
                shell.run(imp);
            }
            continue;
        }
        index += getClose(returned);
        if(index == 0) {
            builder.append(returned).append("\n");
            try {
                shell.run(builder.toString());
            }catch (Exception e){
                shell.compiler.getServerLogger().error(e.getMessage());
            }

            builder = new StringBuilder();
```

```
            prefix = ">";
            line = 0;
        }else{
            builder.append(returned);
            prefix = "... "+line;
        }
        line++;
    }
}
```

WenYanTools

这个是为了标准化 WenYanCompiler 对象，因此在这里都规定了标准的文言文编译器配置