基于 ChatScript 框架的多轮对话开发

何云超

本文将分为代码篇和脚本篇两个部分，从两个不同角度阐述如何基于 ChatScript 框架开发中文多轮对话。代码篇主要讲解如何对原框架代码进行修改、封装，使之易于应用和集成；脚本篇主要讲解写作对话脚本常用方式、语法、函数等。

第一部分 代码篇

本文档目的是解释 ChatScript 框架的使用，以及如何做简单修改，以便适用于中文多轮对话引擎。

1 官方代码

我们选用官方提供的 ChatScript-7.3 作为原始代码，在此基础上进行开发。

代码下载：1. 官网地址; 2. SVN 镜像。

原始 ChatScript 支持任意 utf-8 格式的输入，因此，中文、英文均支持。但是，由于原始代码没有提供中文分词功能，因此输入中文时，需要手动加空格作为分词结果。另外，原始代码通过 wordnet 词典资源来判断一个单词是合法单词还是非法单词，并对非法单词进行简单的错误纠正 ;然而, 由于缺少中文 wordnet, 原始代码不能正常判断中文词语是否为合法。

为了解决分词和非法词检测这两个问题，同时为了更好的和现有引擎继承，我们对原始代码做了简单修改。

概括而言，所做修改可分为三个部分，1) 增加中文分词，2) 封装，3) 中文非法词识别。前两个功能通过程序代码修改实现，后一个通过修改对话脚本实现。整个修改过程可以用下图概括：

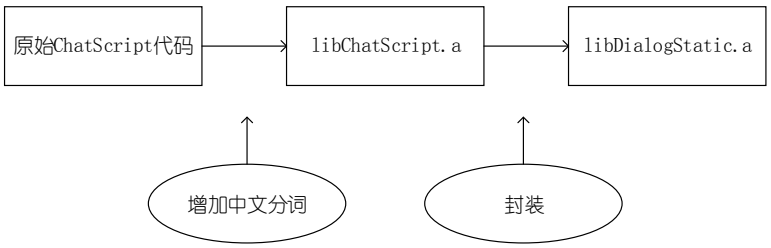


图 1. 程序修改流程

2 中文分词

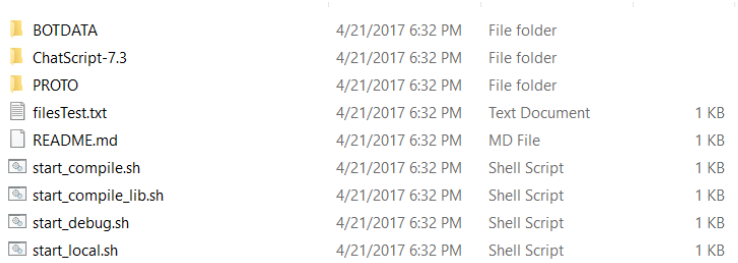
由于中文语句中汉字之间没有间隔符，难以像英文那样，明确的知道单词边界。因此，分词一般是中文自然语言处理最基本的一步，其目的简单来讲，可以看作是给一句话通过在适当的位置加空格，来标示词语间隔。

原始 ChatScript 代码面向的是英文文本，没有提供中文分词功能。我们将结巴分词工具集成到了源代码。

结巴分词：官网下载。

中文分词集成之后的结果见 git: VoiceTech/VoiceCloud/Algorithm/SemanticParser/MultiRoundDialog/MultiTurnDialogue/SourceCode/ChatScript。如下截图展示了集成了中

文分词之后的代码：



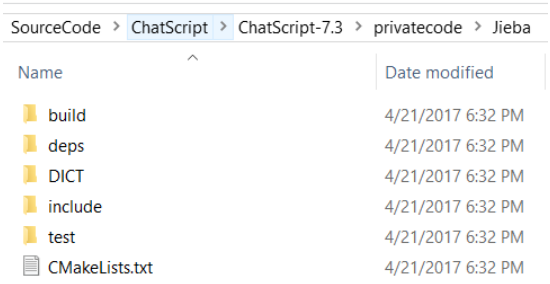
上图中，BOTDATA 文件夹是一些对话脚本，我们先只关注如何集成中文分词，对于对话脚本写作先忽略；ChatScript-7.3 文件夹是源码，后面我们着重介绍；PROTO 文件夹有一些流程图，供对话脚本写作梳理思路，我们可以先忽略。另外，还有四个 shell 脚本，这些脚本文件需要在当前路径下执行，分别实现如下功能：

shell 脚本名称	**功能**
start_compile.sh	编译 ChatScript 成可执行文件，保存在 BINARIES 路径
start_compile_lib.sh	打包成 libChatScript.a 静态库，保存在 BINARIES 路径
start_debug.sh	使用 gdb 调试程序，程序有问题时用来 debug
start_local.sh	运行程序

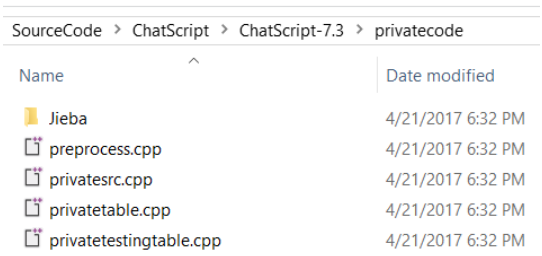
表 1. Shell 脚本功能介绍

和原始 ChatScript 相比，代码上的改动主要可以概括为以下几点：

1) 将结巴分词源码放在 privatecode/ Jieba 文件夹，如下图：



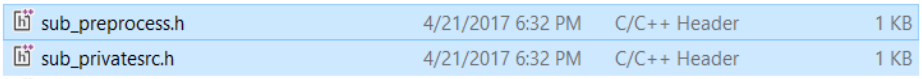
2) 在 privatecode 文件夹中，添加如下四个 cpp 文件：



这四个源文件的功能是：1) preprocess.cpp 中定义了 CNPreprocess()函数，用来作文本分词预处理，使得程序中可以进行分词； 2) privatesrc.cpp 和 privatetable.cpp 为对话脚本提供了^cn_segment()函数，使得对话脚本中可以调用

分词函数。

3) 增加头文件：



这两个头文件是“步骤 2）”中函数对应的函数声明。源程序中在引入这两个头文件的地方，也需要作相应修改。

4) 代码 SRC/common.h 增加 PRIVATE_CODE 宏定义：
#define PRIVATE_CODE

5) 以上步骤完成后，程序的主要改动基本完成。但是，由于在和云平台集成时，有一些命名冲突，需要对一些变量或函数进行重命名。例如 :ChatScriptSocketException

ChatScipt 使用 Makefile 方式组织代码。修改完源程序后，在编译、打包时需要修改一下 Makefile 文件：

1) 增加结巴分词头文件：
server: INCLUDEDIRS=-Ievserver -I../privatecode/Jieba/include/ -I../privatecode/Jieba/deps/

2) -std=c++11 更改为-std=c++0x

3) 如果需要封装成静态库，Makefile_lib 也要做类似修改，同时定义-DNOMAIN=1 以不编译 main 函数

修改完成后按照表 1，执行相应脚本可以完成相应功能。如果表 1 中的脚本执行正常，说明程序无误，最终我们需要静态库。这个静态库位于 BINARIES 文件夹，名称为 libChatScript.a。这时我们完成了图 1 中所示的增加中文分词步骤。后续，只需提供这个静态库，以及对应的头文件，我们就可以使用，或者进行进一步封装。

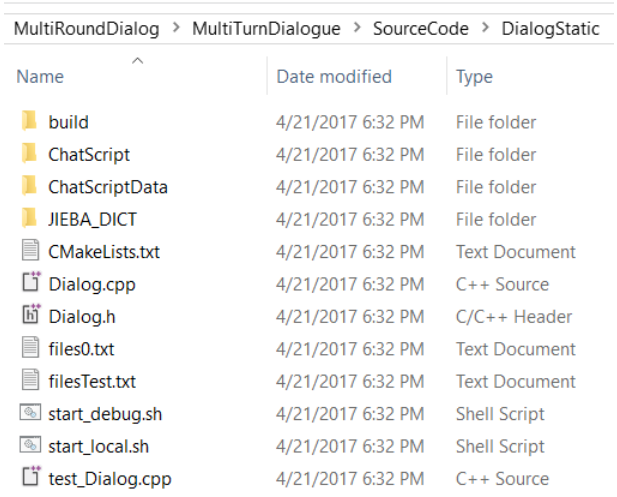
为了简化程序的开发，我们将中文分词和进一步封装作为两个独立的过程。

3 封装

前面，我们介绍了如何加入中文分词功能。虽然，我们已经得到了静态库，通过直接使用这个静态库和其头文件，我们可以实现外部调用。然而，原程序头文件较多，直接引入，可能会导致命名冲突，同时不利于屏蔽程序中的细节。因此，我们对其进行了进一步的封装。

进一步的封装，首先需要前文产生的静态库；其次，需要头文件。这些头文件位于 SourceCode\ChatScript\ChatScript-7.3\SRC 文件夹，所以.h 结尾的文件既是。需要注意的是，分词部分代码为 hpp 后缀结尾，其中头文件和源文件位于同一个文件，因此，在进一步封装时不需要使用分词部分对应的头文件。

封装部分代码位于 SemanticParser\ MultiRoundDialog\ MultiTurnDialogue\ SourceCode\ DialogStatic 文件夹，如下图所示：



这里面的文件夹和文件功能解释如下：

文件/文件夹	**功能解释**
build	程序编译
ChatScript	包含了前文产生的静态库和头文件
ChatScriptData	和对话脚本相关文件
JIEBA_DICT	结巴分词会用到的词典
CMakeLists.txt	用于编译之用
Dialog.cpp	封装部分源代码
Dialog.h	封装部分头文件
files0.txt	对话脚本相关
filesTest.txt	对话脚本相关
start_debug.sh	使用 gdb 调试程序
start_local.sh	启动程序
test_Dialog.cpp	主函数所在，测试封装是否可以

本部分重点是如何进一步封装，供外部调用，因此，我们可以先着重只关注 Dialog.cpp 和 Dialog.h 两个文件，包含了封装的全部代码；另外，如何调用封装之后的代码，可以参考 test_Dialog.cpp 文件。

头文件(Dialog.h)声明了 ChatBot 类具有的方法，这些方法的实现位于 Dialog.cpp 文件。每

一个方法应该如何实现，我们参考了原始 ChatScript 的 main 函数的写法，位于 ChatScript-7.3/SRC/mainSystem.cpp，只是进行了简单整合。

原始代码的 main 函数进行了这样的一些操作：1）解析命令行参数，调整程序当前路径；2）调用 MainLoop()函数进入循环对话；3）完全结束对话时调用 CloseSystem()函数关闭对话。

我们参照 main 函数，MainLoop()函数，以及 MainLoop()函数所调用的 ProcessInputFile()进行了封装。

程序完成后，编译的方法是进入 build 文件夹。然后：

1) cmake ..
2) make

由于我们使用 cmake 作为编译工具，然而 41-49 机器上未安装 cmake 环境。我们在本地安装了 cmake，路径为：/data/data_151/work_home/heyunchao/Libs/cmake/bin/cmake。大家也可以在在自己的工作路径安装，详细安装 cmake 方法见官网，大概步骤是：

```
tar -xf cmake*.tar.gz
cd cmake*
./configure --prefix=$HOME
make
make install
```

安装完成后可通过如下命令测试是否安装成功：

```
cmake --version
```

编译完成后，可执行文件和静态库分别保存在 build 路径下的 bin 和 lib 文件夹。执行如下两个 shell 脚本，可执行相应操作：

start_debug.sh	使用 gdb 调试程序
start_local.sh	启动程序

至此，代码部分已经基本完成，对外只需要提供两个静态库，和对应的头文件，外部就可调用了。如果系统没有安装 libcurl 库，那么还需要提供 curl 库和其对应的头文件。

下面，我们将介绍之前没有介绍的，对话脚本相关的文件。

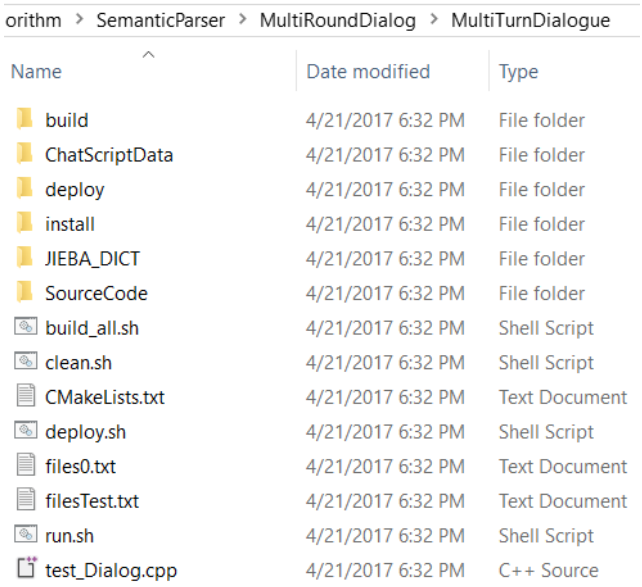
第二篇 对话脚本篇

前文我们只是介绍了代码层面的改动，程序正常运行还需要一些配置文件、对话脚本等文件。本章将介绍这一部分。

前置条件：

1) libChatScript.a 已编译生成
2) libDialogStatic.a 已编译生成
3) 上述两个静态库及其对应的头文件已经复制到 SemanticParser\ MultiRoundDialog\ MultiTurnDialogue\ install\路径下的对应文件夹

我们在\MultiRoundDialog\MultiTurnDialogue 路径下将得到如下目录结构：



这些文件或文件夹说明如下：

文件/文件夹	**功能解释**
build	用于 cmake 编译
ChatScriptData	对话脚本相关
deploy	供向外部署用
install	包含了依赖的静态库和头文件
JIEBA_DICT	结巴分词会用到的词典资源
SourceCode	编译依赖的静态库所对应的源代码
build_all.sh	一键编译好所有依赖
clean.sh	清理编译时产生的临时文件，同时清理对话时产生的聊天日志，上传 Git 前执行一下，可防止将临时文件上传
CMakeLists.txt	cmake 编译时会用到的文件
deploy.sh	一键部署，产生 deploy 文件夹中的内容
files0.txt	对话脚本相关
filesTest.txt	对话脚本相关
run.sh	运行可执行程序

test_Dialog.cpp	测试主函数

本部分我们关注的重点是对话脚本，位于 ChatScriptData 文件夹(完整路径：Algorithm/ SemanticParser/ MultiRoundDialog/ MultiTurnDialogue/ChatScriptData)，以及 files0.txt 和 filesTest.txt 两个文件。

1 对话脚本概览——Hello World

对话脚本描述了系统对于用户的不同话语应该做出如何反应。系统理解用户话语表示的含义，是通过模式匹配的方式实现的，对于不同的模式进步不同的操作。

首先，我们先写一个非常简单的对话脚本，来展示整个流程如何运作，让大家有一个初步认识。写作并运行第一个对话脚本例子步骤如下：

1) 在 MultiTurnDialogue/ChatScriptData/BOTDATA/TEST 文件夹下新建一个.top 后缀的文件，名称为可自取，例如 hello_world.top，文件内容如下图：

```
topic: ~hello_world keep repeat (乐乐)

u: (乐乐)
    乐乐在此，有何贵干？
    a: (没有)
        好吧
    a: (有)
        遵命
```

2) 在 MultiTurnDialogue 路径下，运行 run.sh，终端出现如下所示内容：

```
-bash-4.1$ ./run.sh
CommandLine:
    logs=LOGS
    users=USERS
    topic=TOPIC
    login=user
    local

ChatScript EVSERVER Version 7.3 pid: 1277 64 bit LINUX compiled Apr 21 2017 16:46:35 host=local
Params:   dict:2097151 fact:800000 text:100000kb hash:100000
          buffer:80x80kb cache:1x5000kb userfacts:100 outputlimit:80000 loglimit:80000
WordNet: dict=549864  fact=85706  heap=16733688 Mar13'17-15:51:58
Build0:  dict=20317  fact=24162  heap=236444 Compiled:Apr19'17-15:19:44 by version 7.3 "0"
Build1:  dict=40  fact=109  heap=14620 Compiled:Apr21'17-15:06:36 by version 7.3 "Test"
Used 83MB: dict 570,221 (50179kb) hashdepth 20/1 fact 109,977 (5278kb) heap 16984kb
          buffer (6400kb) cache (5000kb) POS: 0 (0kb)
Free 116MB: dict 1,526,930 hash 418 fact 690,023 stack/heap 83,015KB

    Server disabled.

 [Bot]: EXIT-MULTI-TURN-DIALOGUE
```

3) 终端输入`:build Test reset`命令，完成对话脚本的编译，注意观察终端上 WARNING SUMMARY 部分提示，如果不是 0 serious warnings，那么说明对话脚本有误，需更正。正常情况如下：

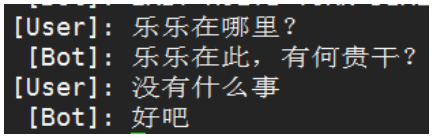
```
WARNING SUMMARY:
0 serious warnings, 0 function warnings, 0 spelling warnings, 0 case warnings, 2 substitution warnings
```

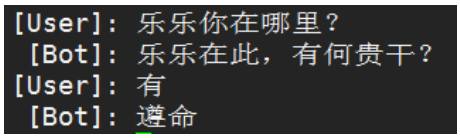
可以忽略 serious warnings 之外的其他 warnings，不过若需追求完美，也可一一解决。

4) 开始愉快的和刚才的 chatbot 对话吧

在开始和刚才编写的对话系统对话之前，我们对以上几个步骤总结一下：1) 对话脚本文件以.top 结尾，位于 MultiTurnDialogue/ChatScriptData/BOTDATA/TEST 路径；2) 写作对话脚本后，不能直接和对话系统对话，而是需要先 build。

刚才的对话脚本可产生如下两种方式的聊天记录：





可以发现，在检测到用户输入中包含“乐乐”这个词时，系统将输出“乐乐在此，有何贵干”；然后，继续判断用户的输入是包含“没有”还是“有”，分别做出不同的系统反馈。

上述是一个简单的对话，实际中，有许多复杂的情况，我们在后面更详细的予以描述。

2 基本概念

本章介绍一些对话脚本中一些基础概念。回到刚才的对话脚本，我们创建了一个名叫 hello_world.top 的脚本文件，里面内容如下：

```
1  topic: ~hello_world keep repeat (乐乐)
2
3  u: (乐乐)
4      乐乐在此，有何贵干？
5      a: (没有)
6          好吧
7      a: (有)
8          遵命
```

第 1 行，定义了一个名为 hello_world 的话题(topic)，话题关键字(keywords)为“乐乐”，话题控制标识(control flags)为 keep repeat。话题名称不一定需要和文件名相同，但需要保证唯一性，不允许出现相同的话题名，即使在不同的脚本文件中。关键字用来决定何时进入这个话题，可以有多个关键字，用空格分开。话题控制标识决定了这个话题在执行时的一些操作逻辑，常用的有 keep repeat，repeat keep nostay 等，详细见高级手册。

第 3 行，定义了第一个规则(responder)，用户说的话中含有“乐乐”则满足此规则，输出。

第 5 和 7 行，定义了嵌套规则(rejoinder)，当用户的第一句话匹配后，依据第二句话内容做不同操作。

responder 不仅可以嵌套一层，还可以嵌套多层，如下图：

```
#! I like spinach
s: ( I like spinach ) Are you a fan of the Popeye cartoons?

        a: ( ~yes )  I used to watch him as a child. Did you lust after Olive Oyl?
            b: ( ~no ) Me neither. She was too skinny.
            b: ( yes ) You probably like skinny models.

        a: ( ~no ) What cartoons do you watch?
                b: ( none ) You lead a deprived life.
                b: ( Mickey Mouse ) The Disney icon.
```

用 a、b、c 到 q 来表示嵌套的层级关系。

括号中为模式(pattern)，括号前的字母我们一般就用 u:或者嵌套时用 a:

更多的对话脚本写作可参考《ChatScript 对话引擎（基础版） v1.0.pdf》《ChatScript 对话引擎（高级版）.pdf》。

3 脚本编译

修改完对话脚本后，需要执行脚本编译，才会在实际对话中产生变化。这里的脚本编译，不同于程序编译，步骤如下：

1) ./run.sh
2) :build Test reset

脚本编译之后，编译结果存在于 ChatScriptData/TOPIC 文件夹。如果编译出错（偶尔会这样），导致不能进入程序，可以删除 TOPIC 文件夹中的内容（只删内容，需保留文件夹结构），重新执行脚本编译一次。

执行步骤 2)之后，程序会去到根目录（即 MultiTurnDialogue 文件夹）寻找 filesTest.txt 文件，然后加载这个文件中所列出来的所有文件。下图是目前 filesTest.txt 文件内容：

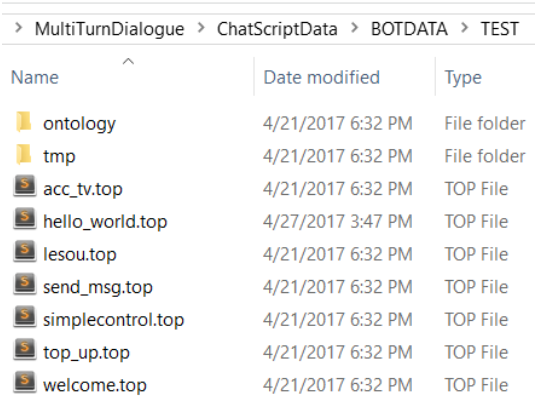
```
# multibot
BOTDATA/TEST/ # tutorial bot data
```

其含义是加载 BOTDATA/TEST/路径下的所有对话脚本，不包括子目录。其完整路径是：ChatScriptData/BOTDATA/TEST/，ChatScriptData 不用写，为默认前缀，路径需以斜杠(/)结尾。# 号表示注释。行和行之间允许空行。如果对话脚本位于多个路径，这个文件中也可以写多个，例如：

```
# underlying conversation system
RAWDATA/HARRY/
# quibble ability
RAWDATA/QUIBBLE/
```

需要注意的是，多个对话脚本路径加载时顺序是随机的，并不是**按行**顺序加载。同时，同一个路径下的多个对话脚本，在加载时顺序也不是顺序加载。

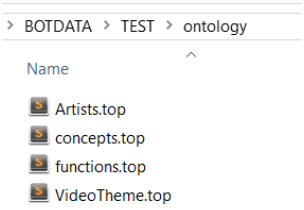
对话脚本加载顺序对于编译是非常重要的，如果脚本之间存在依赖关系，应该先加载没有依赖的脚本，再加载有依赖的脚本。例如：BOTDATA/TEST 路径下有下列对话脚本：



其中两个子文件夹内的内容不会加载；这些脚本之间加载顺序比较随机。

为了解决对话脚本依赖问题。我们可以将一些基础的、不常变化的对话脚本放在一个单独文件夹中单独编译。

我们遇到这样的情况：一些公用的函数、概念，在多个脚本中会使用到。我们的做法是在 ontology 文件夹中放置这些公用的对话脚本，如下：



为了保证公用对话脚本预先编译，我们新建了一个 files0.txt 文件，其内容如下：

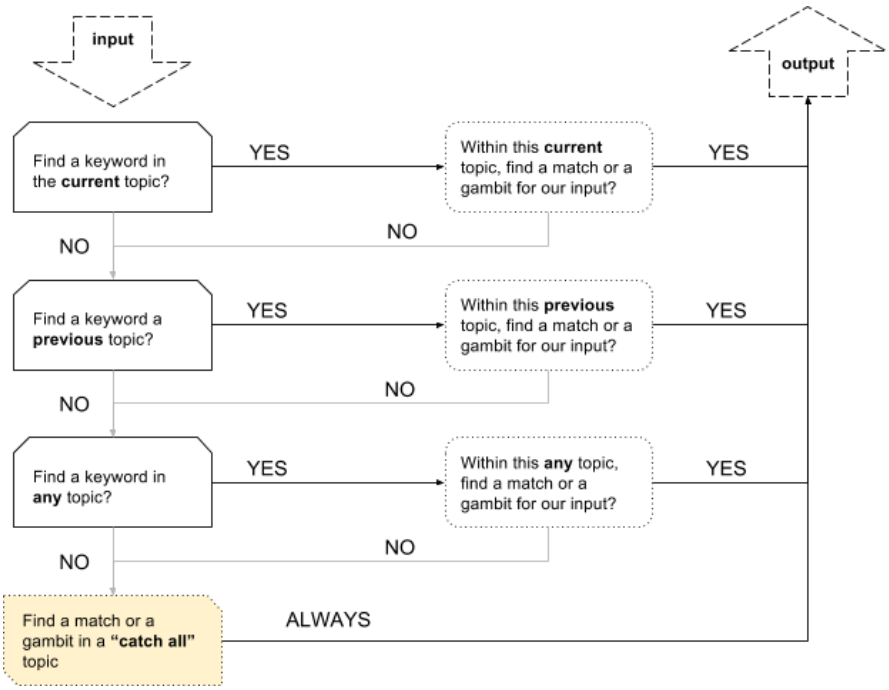
```
# multibot
BOTDATA/TEST/ontology/  # 资源文件
```

在启动程序之后，先执行`:build 0`即可先加载这些公共资源进行编译，然后我们再执行`:build Test reset`即可进行编译对话脚本。之后，如果 ontology 文件夹内的内容没有变化，只是 Test 内的文件变了，那么执行`:build Test reset`即可，不用再执行`:build 0`命令。注意，第一次执行`:build 0`如果报错，解决办法是删除 ChatScriptData/TOPIC 文件夹内容，保留文件夹结构，再试一遍。

4 对话控制脚本

多个对话脚本的执行顺序，如何跳入跳出，如何匹配，匹配失败做什么操作等，也是通过一个对话脚本来控制的，称为对话控制脚本。在我们的应用中，simplecontrol.top 文件为控制对话脚本，一般不需要做修改。

一个基本的对话控制脚本，控制流程如下：



5 未识别词(unknown words)

中文词语全都会认为是未识别词。为了解决此问题，我们在 ChatScriptData/DICT/ENGLISH 文件夹中，0.txt 文件里列出了所有中文字词。如下图所示：

```
0-day ( meanings=1 ADJECTIVE ADJECTIVE_NORMAL )
   0-day~1az
0 ( NOUN ADJECTIVE ADJECTIVE_NUMBER NOUN_NUMBER KINDERGARTEN )
主矩 ( NOUN )
根本就是 ( NOUN )
曲调 ( NOUN )
校园爱情 ( NOUN )
血荐轩辕 ( NOUN )
质量法 ( NOUN )
七十七座 ( NOUN )
内弟 ( NOUN )
王慧敏 ( NOUN )
忧色 ( NOUN )
报童 ( NOUN )
袁大头 ( NOUN )
哈密 ( NOUN )
二杯 ( NOUN )
雷琐辛 ( NOUN )
孔明辞 ( NOUN )
寻棘藤 ( NOUN )
```

其他文件是英文词汇。

第三篇 其他

1. 为了高亮显示对话脚本代码，如下图，可以使用 sublime，通过安装 chatscript-tmlanguage 插件实现，详见 https://github.com/kuzyn/chatscript-tmlanguage

```
# DST and NLU
topic: ~acc_tv keep repeat ( 电视 慢 卡 太卡 很慢 很卡 )

u: (电视 * [慢 卡 太卡 很慢 很卡])
    $intent=acc_up_tv
    ^respond(~acc_tv_policy)
    ^respond(~acc_tv_nlg)

u: ($policy=ask_slot_confirm [确定 开始 是 好的 没问题 加速 好])
    $confirmed=true
    ^respond(~acc_tv_policy)
    ^respond(~acc_tv_nlg)

# Policy
topic: ~acc_tv_policy repeat keep nostay []

u: ($intent=acc_up_tv $confirmed=true)
    $policy=sys_acc_tv
    ^end(topic)
```

2. 用户输入的中文语句，会先经过分词再和各种模式去匹配。我们使用的是结巴分词，要想知道对于一句话是如何进行分词的，一种办法是调用结巴程序，另一种办法是通过这个网站来分析：http://cppjieba-webdemo.herokuapp.com/

3. 文档。对话脚本本文只是初步介绍，许多非常用方面本文并未列举出，在遇到问题时可上官网查看文档。官网文档地址是：https://github.com/bwilcox-1234/ChatScript/blob/master/WIKI/README.md

官方文档也非常繁多，我们经常浏览的几个是：1) hello world，2) 基础手册，3）高级手册，4）系统函数，5）系统变量和概念

何云超
2017 年 4 月 28 日