ChatScript 对话引擎

本文旨在解释 ChatScript 引擎工作原理，描述如何使用脚本语言开发对话系统。

1 概述

ChatScript [1] 是一个使用 C++语言开发的对话系统，基于一种对话流脚本语言实现对话逻辑。基于对话脚本语言实现对话系统的思路，由来已久，例如，AIML，*等，但 ChatScript 和这些相比，有如下这些优点 [2] ：

- 速度快。毫秒级应答，支持几千用户同时访问；
- 内存占用小；
- 好维护。代码简洁，规则维护简单，有明确的机制来支持维护对话脚本：
    - Topic 机制，领域独立，易于多人协同维护
    - 样例输入(sample input)，测试规则匹配是否正常，问题提前发现
    - 回归测试，事先定义正常对话(ideal conversation)流程，防止改动脚本之后正常对话无法通过
    - 分析工具，提供日志分析工具，用来挖掘、重利用对话日志，例如，统计各领域用户数量、对话轮次、规则执行次数等
    - 摘要（abstract）功能，输出用户大概可以说什么，系统大概会如何回复

除此之外，ChatScript 在下面的工作上也能轻松处理：

- 对话控制
- 情绪判别
- 用户追踪
- 多 Bot，支持多个独立 Bot 同时运行
- 支持任务规划（planning）
- 数据库交互，支持 Postgres, Mongo DB
- 文档阅读，内建 POS tagging、parser
- 外部交互，支持系统程序之间进行交互，同时也支持和网络交互，如 HTTP
- 优化搜索，依靠模式(pattern)进行搜索

2 入门

本章我们介绍如何使用 ChatScript 来开发一个简单的对话系统，由简单到复杂，一步一步完善，并对使用的技术进行解释说明 [3]。

2.1 总览

本节先介绍一些脚本写作主要概念。

规则：一条规则由四部分组成：规则类型、规则标签、规则模式、规则输出，如，`?: MEAT (you like meat) I do`

话题：规则的集合称为话题

空格：多余空格自动去除，但是，单词和单词(Token)之间需要用空格分开

注释：用井号（#）作为注释标志，井号和前文内容之间需要用空格隔开。另外，#!，表示样例输入，用于自动测试

命名合法性：话题、函数名称只能使用数字、字母、下划线，以波浪线~开始，变量名称以$开始

2.2 Hello World

2.2.1 运行

Windows 双击 BINARIES 文件夹下的 chatscript.exe 运行，Linux 首先需要将 LinuxChatScript64 文件权限更改为可执行（chmod 777 *），然后通过如下命令进行本地运行：./BINARIES/LinuxChatScript64 local

2.2.2 对话脚本语言

对话脚本按照不同 bot，保存在 RAWDATA 目录中，以.top 作为后缀。这些文件需要使用 UTF-8 格式，Linux LF 换行方式。

一个.top 文件可以包含一个话题，也可以包含多个话题。每个话题，有多个规则组成。

规则类型一共有四种：陈述句(s:)、疑问句(?:)、陈述句或者疑问句(u:)、开场白(gambit, t:)，除此之外，规则还可以以 a:, b:, …, q:开头，表示二级规则，用来处理分支情况，称为 Rejoinders。二级规则不能作为顶级规则，只有当父级规则已经匹配，才会去匹配。

因此，根据响应方式不同，一个规则还可以分为下面三种类型：1) 直接响应 (Responders)，2) 二级响应 (Rejoinders)，3) 开场白 (Gambits)

一个规则是否能被触发，是依据这个规则的模式(pattern)是否被匹配了来决定的。因此，模式是对话脚本语言中，非常重要的组成部分。

模式使用英文括号括起来，用来识别文本中的指定单词或单词序列。例如：

模式	解释	匹配成功样例
(A B)	A B 出现，顺序有关	C A B, D A C A B A
([A B C])	A B C 任意一个出现即可	A D, B C D
~word	一个概念，同义词集合	该集合中任意一个词

3 话题

ChatScript 对于每次用户的输入，并不是检查全部规则来找到匹配的。相反，按照话题来匹配。话题通过类似如下方式申明：

```
topic: ~DEATH [dead corpse death die body]

t: I don’t want to die
?: (When will you die) I don’t know.
```

话题申明，包含：话题名称、关键词、规则。话题名称，需以~开始。

3.1 关键词

话题的跳入跳出机制通过关键词实现。关键词，为话题触发词，在匹配了用户输入之后，使得系统进入次话题。如果一句话能够触发多个话题，系统选择最可能触发（触发词数量、触发词长度）的进入。跳入某个话题之后，然后去寻找规则，看看那个规则能够匹配，如果这个话题下的所有规则都不能匹配，那么只输出开场白。

跳入某个话题后，如果用户说一些和这个话题不相关的语句（不匹配任何模式），那么，系统将会去寻找别的话题，看看是否有某个话题下的规则匹配此语句。如果有，那么，从原话题跳出到新话题。

刚刚跳入的话题，称为激活态话题(active topic)，之前的话题则变为挂起(pending)状态。这些挂起态话题形成一个挂起栈(pending stack)，待激活态话题结束后，将返回到原话题。

3.2 开场白

对话处于系统主导时，使用开场白(gambit)来缓解尴尬。对话有两种状态，系统主导状态，用户主导状态。两者的区别是：用户主导，是指，用户话语能够匹配到某个规则；否则，则处于系统主导。

处于系统主导时，系统应答是选择开场白，还是选择其它相近话题，还是引导到一个特定话题，这些通过控制脚本的逻辑(logic of control script)来控制。ChatScript 已经默认提供了一个控制脚本，但是开发者可以自定义。

3.3 执行顺序

一个话题中的规则，顺序匹配。

默认情况下，每个规则都会避免重复，匹配了一次后，会标记为已用(used-up)。这时，用户再说相同的话，就不能匹配到刚才匹配的规则。（可用 keep repeat 更改）

3.4 二级规则

规则可以嵌套，形成二级规则，例如：

```
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.
```

缩进之后利于阅读，但可以不。

3.5 规则标签

所有规则可以使用规则标签(label)，便于规则重用和调试(借助调试工具)。例如：

```
t: MY_EYES () My eyes are blue
?: EYECOLOR (color * eyes) I have blue eyes
u: GLASSES ([glasses contacts]) I don't wear glasses, but I do have
contacts.
?: BLIND (you * blind) I am not blind.
```

```
?: COLORBLIND (you * [color-blind "color blind"]) I am not color blind.
```

3.6 模式

规则通过模式(pattern)进行匹配。模式在写作的时候，需要考虑到泛化能力和覆盖率，是这两者之间的折中。

模式	解释	匹配成功样例
(A B)	A B 出现，顺序有关	C A B, D A C A B A
([A B C])	A B C 任意一个出现即可	A D, B C D
~word	一个概念，同义词集合	该集合中任意一个词
< >	句子边界，< 开始，>结束	句子开始、结尾限制
*	通配符，匹配零个或多个词语	任意词
*1	精确限制一个词语	a
*~1	0 到 1 个词	a
<<A B >>	无序匹配，A B 可以出现在句中任意位置，注意：后面跟着的单词，从头检索	A B, B A
[“A B” C]	“” 将多个词作为一个整体	C, A B
!x	不能出现 x	A
![A B C]	不能出现中括号中的任意一个词	D E
A !!B	A 下一个词不能是 B	A C
{A B}	可选，括号中的一个词出现或者不出现都可	C A D, C D

3.7 插入语

在一个句子中间插入一个成分，它不作句子的何种成分，也不和句子的何种成分发生结构关系，同时既不起连接作用，也不表示语气,这个成分称之插入语(Interjections)。在 ChatScript 中，插入语定义在 livedata/interjections.txt 文件中。

3.8 归一化

默认情况下，用户输入中的每个词，都会自动归一化后再去和模式匹配。归一化原则通过 LIVEDATA 文件夹中的 canonical.txt 文件控制。如果模式中某个词必须严格的匹配上用户输入，而不是其归一化形式，可在模式中的这个词前面加上撇号（'），例如：

```
u: ( I 'like you ) This matches I like you but not I liked you.
s: ( I was ) This matches I was and Me was but not I am
```

这时，虽然 liked 归一化后仍然是 like，但因为模式中 like 前面有个撇号，就只能使用原型去匹配。同样，对于概念，也可以使用撇号，限制进行原型匹配，例如：

```
u: ('~extent_adverbs)
```

3.9 命名

:commands，显示所有命令，常用的命令有：

:build 应用名称，重新把.top 文件编译进入 bot

:build 0，编译 ontology 和知识

4 输出

每一个规则的目的，是某个模式匹配了用户输入后，将对应结果予以输出。在写输出语句时，有以下技巧：

- 直接输出，模式后面直接跟输出内容
- 自动格式化(auto reformat)，即使有多余空格，会自动去掉
- 转移字符输出，如果要输出系统保留的字符（例如：[,], ^, ~等），需要在前面加反斜杠(\)
- 随机输出，[A][B][C]，随机从 A、B、C 选择一个输出，增加灵活性

5 变量

变量用于记忆需要捕捉的信息，例如，槽位取值等。用_来捕捉单轮记忆，用$捕捉多轮记忆，通过把一个变量取值赋为 null 之后，相当于销毁这个变量。一个用$申明的变量，将会在所有模式匹配结束后才会消失，如果想在这一个话题结束后就消亡，可以使用$$申明变量，例如：`$$food`

除此之外，有些变量是全局变量，可能在多个话题、bot 中都会使用，这些变量称为系统变量(system Variables)，申明时使用%，例如，%hour, %bot。系统已经预先内置了一些系统变量，详见[4]。

ChatScript 正是依靠_和$，实现了系统的记忆功能。

参考资料

[1] ChatScript Github 地址：https://github.com/bwilcox-1234/ChatScript

[2] ChatScript Overview: https://goo.gl/pzupxc

[3] ChatScript Basic User Manual: https://goo.gl/a2oypY

[4] ChatScript System Variables and Engine-defined Concepts and Parameters: https://goo.gl/IPpn4I

[5]