ChatScript 对话引擎高级特性

本文讲解 ChatScript 的一些高级用法，详细可参考文档[1]。系统和用户间的一次交互，称为 volley，每一次的交互(volley)，会包含以下几个步骤：接收用户任意输入，加载用户相关数据、对话状态，计算响应，更新对话状态，发送响应给用户。下面我们对这些环节进行详细介绍。

1 基础

1.1 话题和规则

ChatScript 对话逻辑描述的基础是话题，话题有众多规则组成，规则由模式和输出组成。模式可以获取全局数据、用户输入，还可以拥有记忆功能、进行运算操作、条件判断等。

1.2 二级规则

不同的用户输入，可能导致对话流程分支，这可以使用二级规则实现。

1.3 用户变量

为了记忆信息，使用$开始作为变量名称，例如$tmp。变量的使用不用事先申明，如果不存在，其值为 null（或 0，依据上下文决定），并自动创建。

变量可以用来存储字符串，数字也是保存成字符串的形式。支持三种类型的字符串格式：1) 连续无空格，2) 有空格，如 *meat-loving plants*，3) 涉及其他变量或函数调用，如 *^"I like $value"*。

按照变量的有效时间和读写范围，可以分为：1) 永久变量(Permanent variables)，使用$开始，永久存在，会保存到磁盘，变量值可全局使用和修改；2) 临时变量(Transient variables)，使用$$开始，用户交互结束后，立即消失，不存于磁盘，变量值可全局使用和修改；3) 本地变量(Local variables)，使用$_开始，用户交互结束后，立即消失，不存于磁盘，变量值不可全局使用和修改，只能在同一个话题（或输出宏，outputmacro）中使用和修改。

1.4 事实

ChatScript 支持使用三元组(triple)的形式定义事实(Facts)，还提供了对事实的查询。这个三元组的每一个部分的格式是字符串，或者其它事实的引用。例如：

```
(I eat "meat-loving plants")
```

从网站请求获得的数据需要是 JSON 格式，也会被表示成三元组的形式，用以查询。

和变量类似，事实也分为永久事实和临时事实，其时效性也类似。

1.5 输出

规则满足时，希望输出一些文本给用户。输出文本按照所处的状态，可以分为挂起输出(pending output)和提交输出(committed output)。

1.6 归一化

当 ChatScript 接收到用户输入时，先进行分句，再分词(tokenize)，然后归一化。归一化操作是按照这个词所属的概念进行。概念以~开始，常常是一些同义词或相同概念词语的集合。例如，~animals 收录了所有动物名称，~noun 是所有名词。概念和概念之间支持嵌入和引用。

对于用户的任意输入，ChatScript 会同时处理原始序列和归一化序列。例如，对于用户输入 *my cat eats mice*，系统会同时维持两个序列进行匹配：原始序列 *my cat eats mice*，和归一化序列 I cat eat mouse。

因此，为了增强泛化能力，模式中的词语最好是归一化形式。

1.7 记忆

对于需要提取出的有用信息，使用_0, _1, …形式获取。ChatScript 会自动记住原始形式和归一化形式。(英文单引号'，使用原型)

1.8 控制流程

ChatScript 不是基于跳转(GOTO)实现的，而是基于调用(call)和返回(return)实现的。

1.9 函数

话题不是函数，因为话题不能传入参数。为了增加灵活性和便捷性，ChatScript 提供了一些内置系统函数，同时支持用户自定义函数。函数名称以^符号开始，例如，^match(argument1 argument2)，函数参数通过空格分开，而不是逗号（因为逗号本身可以作为函数参数）。函数体中，可以产生输出、调用其他函数、使用话题、使用规则等操作。

通过使用函数，可以方便的组织和共享代码。

1.10 函数变量

常常使用$_来定义函数内的局部变量，离开函数后随即消亡。函数参数的类型可以是字符串或者变量。函数参数需要以^开始。如果参数是字符串，那么对其赋值的操作是违法的。例如：

定义函数

```
outputmacro: ^myfunction( ^argument1 ^argument2)
{
    ^argument1 += 1
}
```

调用函数：

```
^myfunction( $myvar 1)
```

将会对变量$myvar 的取值增加 1。如果函数体中有：^argument2 += 1，将是违法的。因为 1+=1 错误。

又如：

```
outputmacro: ^myfunction( $_argument1 $_argument2)
    $_argument1 += 1
```

使用$_作为函数参数开始，表示此参数是临时的。

2 概念

概念(concept)定义了同义词、抽象语义等内容。定义概念时，可以使用词性信息（POS information）对其修饰，例如：

```
concept: ~mynouns NOUN NOUN_SINGULAR (boxdead foxtrot)
concept: ~myadjectives ADJECTIVE ADJECTIVE_BASIC (moony dizcious)
```

概念修饰词有下面这些，或者他们之间的组合：

概念修饰词	说明	概念修饰词	说明
NOUN	名词	MORE	扩展概念，概念定义一次，第二次定义则失败，可用此修饰词在原来基础上扩展
NOUN_SINGULAR	名词单数	DUPLICATE	允许重复
ADJECTIVE	形容词	ONLY_VERBS	仅含动词
ADJECTIVE_BASIC	基本形容词	ONLY_ADJECTIVES	仅含形容词
IGNORESPELLING	忽略拼写错误	ONLY_ADVERBS	仅含副词
ONLY_NOUNS	仅含名词		

概念除了通过~xxx()方式显式定义外，系统还内嵌了一种隐式定义。隐士定义的概念主要是：词性标注（通过 POS-tagger 决定词性），语法角色，遵循某种规则的无限集合（例如，~number，~placenumber，~weburl）等。

3 话题

关于话题（topic），还有一些进阶内容，可以加深理解。

3.1 模式匹配结束条件

在执行话题中某个规则时，执行（匹配）结束的条件不是看是否匹配了某个规则。只有在某个规则产生了用户输出，或者触发了^end 或^fail 函数时，才会结束。例如：

```
u: (I love) $userloves = true

u: ( dog ) $animal = dog

u: ( love) Glad to hear it

u: ( dog) I hate dogs
```

当用户输入 I love dogs 后，并不是只执行第一条规则，而是前三条规则都会执行，因为到了第三个规则，遇到了用户输出。

3.2 话题控制标志

在定义话题(topic)时，使用话题控制标志(Topic Control Flags)来控制话题的全局特性。例如：

```
topic: ~rust keep random [rust iron oxide]
```
其中，keep random 为话题控制标志。

所支持的标志及其含义如下：

编号	标志	描述
1	Random	随机选择规则进行匹配，而非顺序匹配
2	NoRandom	（默认）顺序匹配
3	Keep	禁止擦除已匹配规则，开场白不受影响
4	Erase	（默认）擦除已匹配规则
5	NoStay	话题产生输出后，不再考虑，直接离开
6	Stay	（默认）话题产生输出后，让这个话题处于挂起（pending）状态
7	Repeat	一个规则产生输出后，下一次还可以再次匹配
8	NoRepeat	（默认）一个规则产生输出后，下次不再考虑
9	Priority	在使用关键词匹配话题时，提高此话题的优先级
10	Normal	（默认）关键词匹配话题时，采用常规优先级
11	Deprioritize	在使用关键词匹配话题时，降低此话题的优先级
12	System	定义系统话题，自动使用 NoStay, Keep 标志。
13	User	（默认）定义常规话题
14	NoBlocking	在:verify 操作中，禁止执行 blocking 测试
15	NoPatterns	在:verify 操作中，禁止执行 pattern 测试
16	NoSamples	在:verify 操作中，禁止执行 sample 测试

17	NoKeys	在:verify 操作中，禁止执行 keyword 测试
18	More	允许再次定义概念时，对原有概念的关键词进行扩展
19	Bot=name	限制使用此话题的 bot 名称。多个 bot，用逗号分开

3.3 规则擦除和重复

通常，规则默认执行自动擦除。不想擦除，请用 keep。使用 repeat 不能防止擦除，它只是抑制了输出。

3.4 关键词和控制脚本

话题被唤醒的方式有：关键词匹配，控制脚本调用，其他话题调用。如果一个话题是想通过控制脚本调用而唤醒，那么，这个话题的关键词应该设置为空。否则，将可能出现话题同时唤醒两次的情况，导致浪费和顺序紊乱的问题。

3.5 话题挂起

所谓挂起(pending)，是指在跳出某个话题后，如果用户之后说的话语仍然可能返回之前的话题，并且想记录之前话题中的某些信息，那么就让跳出的话题进入挂起状态。

有一些话题永远不可能进入挂起状态，包括：系统话题，阻止(blocked)话题，nostay 修饰的话题。

3.6 随机开场白

通常，规则是顺序执行的，开场白 t:虽然提供了顺序描述的功能，但是，有时候想随机开场。一种方法是使用 random 标志修饰话题，但是，全部规则都会打乱顺序执行。如果只想让开场白随机顺序执行，而其他规则线性顺序执行，即部分随机（semi randomness），可以使用 r:。

每一种类型的开场白，看作是一个子话题(subtopics)的入口。这时的话题有多个子话题组成，例如：

```
Topic: ~beach [beach sand ocean sand_castle]

# subtopic about swimming
r: Do you like the ocean?

t: I like swimming in the ocean.

t: I often go to the beach to swim.

# subtopic about sand castles.
r: Have you made sand castles?
    a: (~yes) Maybe sometime you can make some that I can go see.
    a: (~no) I admire those who make luxury sand castles.

t: I've seen pictures of some really grand sand castles.
```

上例中，beach 话题有两个子话题，swimming 和 sand castles。系统将会随机选择一个子话题作为开始，直到这个子话题结束后，选择下一个子话题。

3.7 控制脚本

一般使用系统自带控制流脚本，当然，也可以修改或自定义。详见文档[1]。

4 模式

4.1 关键词

多个词组成的关键词，需用引号：

```
concept: ~remove ( "take away" remove )
```

使用通配符需小心。

4.2 字典关键词集合

形如：

```
concept: ~buildings [ shelter~1 living_accomodations~1 building~3 ]
```

~数字

4.3 系统函数

大多数不用管。经常用到^query，来查看某个事实是否存在。

4.4 宏

正如我们可以通过变量、集合在规则之间共享数据一样，我们也可以使用宏(Macros)的方式共享脚本代码。

宏名和宏参数名需要以^开始，宏定义的结束以文件结束或者新宏定义开始为标志。例如：

```
patternmacro: ^ISHAIRCOLOR(^who)
    ![not never]
    [
        ( << be ^who [blonde brunette redhead blond ] >> )
        ( << what ^who hair color >> )
    ]

?: (^ISHAIRCOLOR(I)) How would I know your hair color?
```

注意参数使用空格分开，而非逗号。

4.5 转义字符

使用反斜杠(\)输出系统保留字符。例如，([

5 惯用套路

5.1 巧用^refine()

为了增加规则执行效率，通常先捕捉许多内容，然后再进一步分析优化。例如：

```
u: ( ~country) ^refine() # gets any reference to a country
    a: (Turkey) I like Turkey
    a: (Sweden) I like Sweden
```

```
    a: (*) I've never been there.
```

上述用法更加常用，虽然用子话题方式也可以等价实现，例如：

```
u: (~country) ^respond(~subcountry)

topic: ~subcountry system[]

u: (Turkey) ...
u: (Sweden) ...
u: (*) ...
```

但是，子话题方式通常使用情况是，在多个规则共享子话题时，才使用。

子话题的一种使用场景：

```
?: (<what) ^respond(~quibblewhat)
?: (<when) ^respond(~quibblewhen)
?: (<who) ^respond(~quibblewho)

# ...

topic: ~quibblewho system []

?: (<who knows) The shadow knows
?: (<who can) I certainly can't.
```

称之为 quibbling code。

5.2 巧用^reuse

通过 reuse 可以实现规则重用：

```
t: HOUSE () I live in a small house

u: (where * you * live) ^reuse(HOUSE)
```

参考资料

[1] ChatScript Advanced User's Manual: https://goo.gl/kUKQoJ

[2]