从零开始，开发一款聊天机器人

> 友人：最近忙什么呢？
>
> 本人：我在开发聊天机器人。
>
> 友人：像你这么不会聊天的人，也能开发聊天机器人啊？
>
> 本人：……

不会聊天的人能开发聊天机器人吗？如果开发者自己就不会聊天，开发出来的机器人是不是也不会聊天呢？

ELIZA：聊天机器人的鼻祖

我们先来看看世界上第一款聊天机器人吧：1966年，在MIT任职的德裔美国计算机科学家Joseph Weizenbaum，开发出了一款名为ELIZA的聊天机器人程序。

通过运行名为DOCTOR的脚本，此程序可以用类似心理学家的方式和人类交谈。下面就是一段ELIZA和用户的经典对话：

> User: Men are all alike.
>
> ELIZA: IN WHAT WAY
>
> User: They’re always bugging us about something or other.
>
> ELIZA: CAN YOU THINK OF A SPECIFIC EXAMPLE
>
> User: Well, my boyfriend made me come here.
>
> ELIZA: YOUR BOYFRIEND MADE YOU COME HERE
>
> User: He says I’m depressed much of the time.
>
> ELIZA: I AM SORRY TO HEAR YOU ARE DEPRESSED

好暖心的ELIZA（不是ELSA），和“她”说话，是不是感觉很治愈？

“她”的制造者引入了心理学家罗杰斯提出的个人中心疗法(Person-Centered Therapy)，更多强调对话态度、尊重与同理心。Eliza并不主动提供内容，它只是引导用户说话。

那么**ELIZA背后的计算逻辑**是什么样的呢？

简单得叫人想哭，其实就是一条条模式匹配和字符替换规则，下面是其中几条：

```
you are (depressed|sad) * => I AM SORRY TO HEAR YOU ARE \1

all * => IN WHAT WAY

always * => CAN YOU THINK OF A SPECIFIC EXAMPLE
```

用户说失望/伤心，ELIZA就回答“我好抱歉啊”；用户用了“所有”这个词，机器人就追问一句“以何种方式”；如果用户说“总是”如何如何， 程序就引导“你能举个例子吗”……

ELIZA项目取得了意外的成功，它的效果让当时的用户非常震惊。Joseph教授原本希望ELIZA能够伪装成人，但没有寄予太高期望，让他没想到的是，这个伪装居然很多次都成功了，而且还不容易被拆穿。以致于后来产生一个词汇，叫ELIZA效应，即人类高估机器人能力的一种心理感觉。

ELIZA无疑是一款会聊天的机器人——很多程序员真应该和ELIZA好好学学：跟人说话的时候经常重复一下对方的话，以显示同理心；经常跟进询问“到底是怎么回事呢？”，“你能举个例子吗？”以显示对对方的尊重和关注；经常说“好的”，“是的”，“你是对的”，自然就获得对方的好感了……

但是，这样一款机器人，真的能解决现实的问题吗？如果，我需要的不是和人闲聊，而是需要确定，或者咨询某件事，机器人能告诉我吗？

对于这样的需求，ELIZA恐怕做不到，不过，别的机器人可以。

问题解决型机器人（Task Completion Bot）

问题解决型机器人，存在的目的是为了帮用户解决具体问题，例如：售前咨询、售后报修、订机票、酒店、餐厅座位等等。

基础三步

和ELIZA明显不同，问题解决型机器人需要提供给用户用户自己都不知道的信息。

有鉴于此，问题解决型机器人（TC Bot）需要有自己的知识储备——**知识库（Knowledge Base）**，其中存储的信息用来提供给用户。

光有了知识还不够，还需要至少做到两件事：

1. 理解用户问题，知道用户在问什么。

2. 将用户的问题转化为对知识库的查询。

有了这三者，就可以做到最基本的问题解决了：

多轮对话的上下文管理

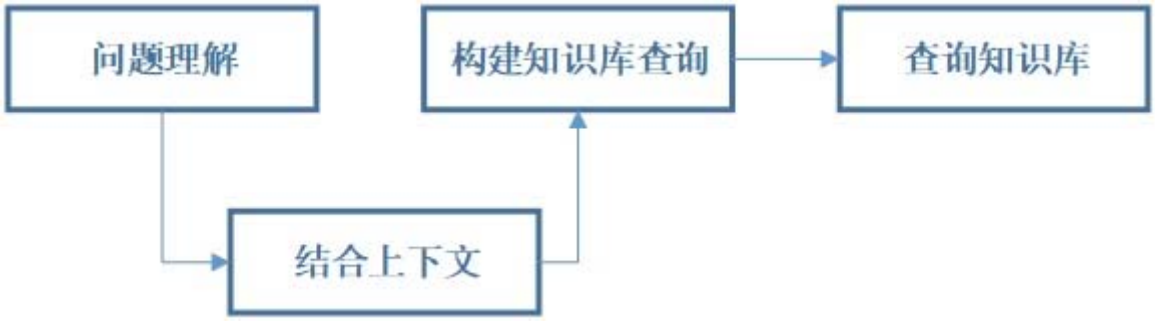
如果用户的每个问题都是完整的，包含了该问题所需要的所有信息，当然上面三步就可以得出答案。

不过人类在聊天的时候有个习惯，经常会把部分信息隐含在上下文中，比如下面这段对话：

> 提问：今天北京多少度啊？
>
> 回答：35度。
>
> 提问：有雾霾吗？（**北京**有雾霾吗？）
>
> 回答：空气质量优。
>
> 提问：那上海呢？（上海有**雾霾**吗？）
>
> 回答：空气质量也是优。

人类理解起来很容易，但是如果要让机器人理解，我们就需要给它添加上一个专门的上下文管理模块，用来记录上下文。

如此一来，支持多轮对话的问题解决型聊天机器人，就需要经历下列四步来完成。



分层结构

从程序开发的角度，TC Bot分为三层：

- 输入输出：
  - 接受、理解用户问题
  - 生成、返回答案给用户
- 中间控制：构建双向关系
  - 用户问题=>知识库知识
  - 知识库知识=>机器人答案
- 知识存储：存储用于回答用户问题的知识。

聊天机器人的实现技术

从学术研究的角度讲，聊天机器人所需技术涉及到自然语言处理、文本挖掘、知识图谱等众多领域。

当前的研究和实践中，大量机器学习、深度学习技术被引入。各种炫酷的算法模型跑在Google、微软等IT寡头的高质量数据上，得到了颇多激动人心的研究成果。

但具体到实践当中，在没有那么巨量的人工标注数据和大规模计算资源的情况下，于有限范围（scope）内，开发一款真正有用的机器人，更多需要关注的往往不是高深的算法和强健的模型，而是工程细节和用户体验。

此处，我们只是简单介绍几种当前实践中最常用，且相对简单的方法：

Solution-1： 用户问题->标准问题->答案

知识库中存储的是一对对的“问题-答案”对（**QA Pair**）。这些Pair可以是人工构建的，源于专家系统或者旧有知识库的，也可以是从互联网上爬取下来的。

现在互联网资源这么丰富，各种网页上到处都是FAQ，Q&A，直接爬下来就可以导入知识库。以很小的代价就能让机器人上知天文下晓地理。

当用户输入问题后，将其和知识库现有的标准问题进行一一比对，寻找与用户问题最相近的标准问题，然后将该问题组对的答案返回给用户。

其中，用户问题->标准问题的匹配方法可以是关键词匹配（包括正则表达式匹配）；也可以是先将用户问题和标准问题都转化为向量，再计算两者之间的距离（余弦距离、欧氏距离、交叉熵、Jaccard距离等），找到距离最近且距离值低于预设阈值的那个标准问题，作为查找结果。

但关键字匹配覆盖面太小。距离计算的话，在实践中比对出来的最近距离的两句话，可能在语义上毫无关联，甚至满拧（比如一个比另一个多了一个否定词）。另外，确认相似度的阈值也很难有一个通用的有效方法，很多时候都是开发者自己拍脑袋定的。

因此，这种方案，很难达到高质高效。

Solution-2. 用户问题->答案

知识库中存储的不是问题-答案对，而仅存储答案（文档）。

当接收到用户问题后，直接拿问题去和知识库中的一篇篇文档比对，找到在内容上关联最紧密的那篇，作为答案返回给用户。

这种方法维护知识库的成本更小，但相对于Solution-1，准确度更低。

Solution-3. 用户问题->语义理解->知识库查询->查询结果生成答案

从用户的问题当中识别出用户的意图，并抽取这个意图针对的实体。

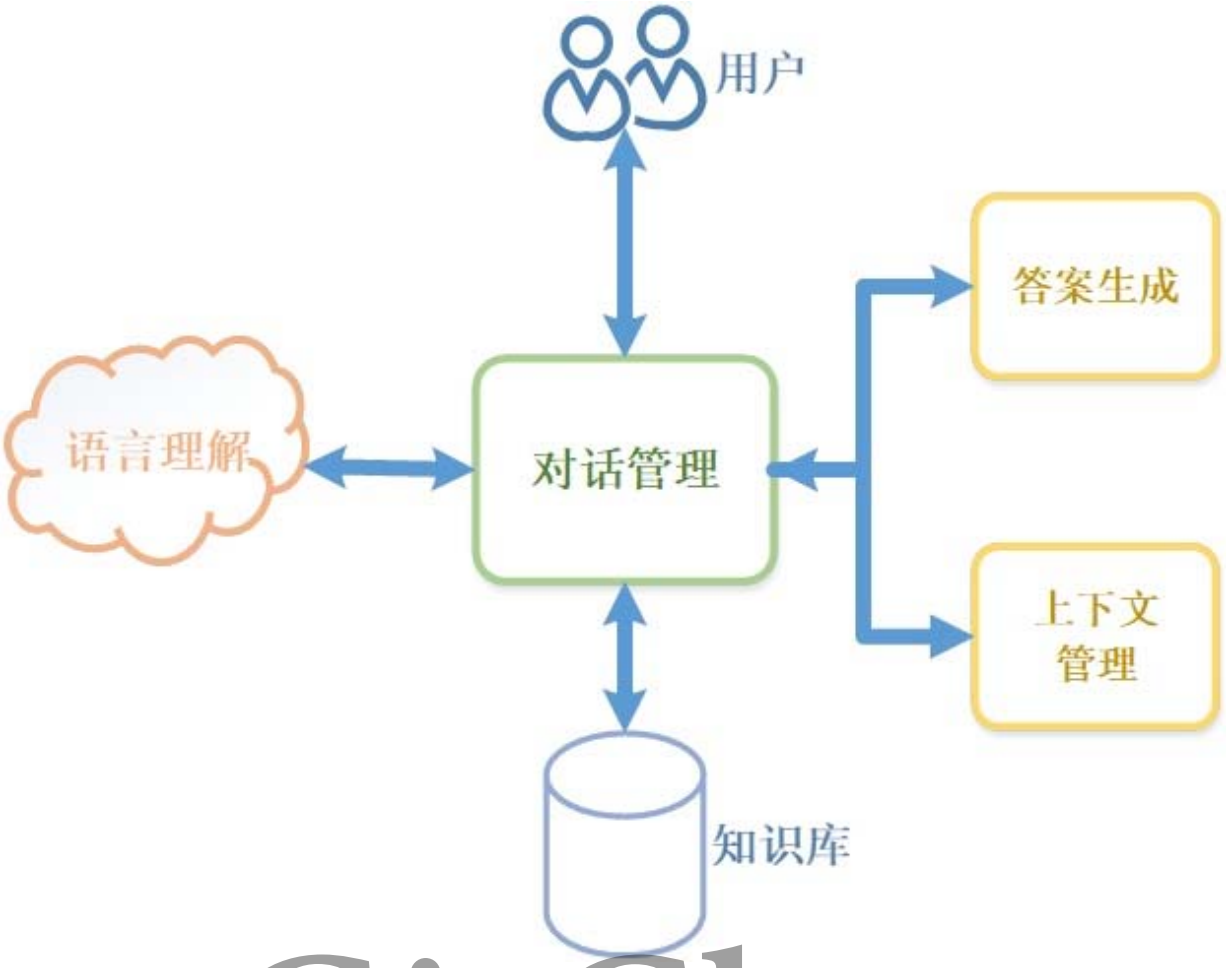
相应的，知识库内存储的知识，除了包含知识内容本身之外，还应该在结构上能够表示知识之间的关联关系。

Chatbot在提取了意图和实体后，构造出对知识库的查询（Query），实施查询，得出结果后生成回答，回复给用户。

我们下面所讲的“小白版问题解决型聊天机器人技术方案“，就是基于本Solution的。

小白版问题解决型聊天机器人技术方案

极简版TC bot的架构



最左侧的语言理解（ Language Understanding ）模块，和右上的回复生成器（ Response Generator ）加起来是输入输出层。

正中的对话管理和上下文管理属于中控部分。

最下方的知识库则属知识存储层。

语言理解

语言理解是什么

语义理解的过程包括两个子任务：

- **意图识别** (intention classification)：用来识别用户所提问题的意图，也就是用户希望做一件什么事。
- **实体抽取** (entity extraction)：用于提取用户对话中所提供的和意图相关的参数(实体)，例如时间、地点等。

具体某个Chatbot的意图类型和实体类型，是其***开发者自己定义***的。

举个例子，小明是一家小淘宝店主，他要为自己的淘宝店开发一款客服机器人，主要回答和商品属性（品牌、价格、邮费、售后等）相关的问题。

那么小明可以如此定义意图和实体：

> - **Case1**：有粉色的手机壳吗？—— 意图：商品推荐；商品类型实体：手机，商品颜色属性实体：粉色。
> - **Case2**：00183号商品快递到伊犁邮费多少？—— 意图：查询邮费；目的地实体：伊犁，商品Id实体：00183。
> - **Case3**：02465号商品有保修吗？——意图：保修查询；商品Id实体：02465。

或者，换个定义意图和实体的方式，也没有问题：

> - **Case2'**：00183号商品快递到伊犁邮费多少？—— 意图：商品查询；目的地实体：伊犁，商品Id实体：00183，商品属性实体：邮费。
> - **Case3'**：02465号商品有保修吗？——意图：商品查询；商品Id实体：02465，商品属性实体：保修。

具体怎么定义，要与知识库的结构以及中间控件的实现结合起来决定。怎么定义能够使得后面的步骤易于进行，就怎么定义。

语言理解怎么做

怎么能够从用户的提问当中发现意图和实体呢？

最简单的是**基于规则的方法**：用关键字/正则表达式匹配的方式，来发现自然语言中的意图和实体。


```
GET_INPUT
    IF "有（.+）吗" in User_Input
        Intent = “商品推荐”
    ELIF "（邮费|邮资|运费|快递费）+" in User_Input
        Intent = “查询邮费”
    ELIF "（保修|维修|修理|售后）+" in User_Input
        Intent = “查询售后”
    … …

FOREACH（Color in Colors）
        IF （Color in User_Input）
            Entity.Type = “ProductColor”
            Entity.Value = Color
    … …
```

这样做的**优点**显而易见：方便、直接。

缺点也很明显：缺乏泛化能力。

一件事有很多种说法的时候，需要开发者将所有这些说法都手动添加到白名单里。

这样不仅很难面面俱到，而且，还会碰到不能够靠匹配关键词解决的问题。比如，有人问“邮费能给免了吗”，有人问“这个商品邮费多少钱？”,这根本是两个意思，如果都有“邮费”匹配，给同一个答案，就答非所问了。

这个时候，就需要引入**基于模型的方法**：使用机器学习模型来进行意图识别和实体抽取。

一般来说，**意图识别**是一个典型的**分类模型**(e.g. Logistic Regression，Decision Tree等)，而**实体抽取**则是一个**Sequence-to-Sequence判别模型**（一般选用Conditional Random Field）。

语义理解模块当然可以自己开发。不过，这需要长期积累的自然语言处理(NLP)的专业知识和经验，高效的运算框架，以及标注工具的支持。

作为一个轻量级Bot的开发者，单独开发一个语言理解模块耗时耗力，效果还未必好。好在技术模块化、工具化程度极高的今天，已经有人为小白用户开发了简单、易用的语言理解工具。

微软推出了的语言理解智能服务- LUIS（https://www.luis.ai）就很好用（具体用法在附录中介绍）。

那么我们要做的，就是用LUIS构造一个意图识别模型，和一个实体抽取模型。因为具体的算法和模型训练层LUIS都已经在后台实现了，所以，我们实际上要做的仅仅是为它提供训练数据，并进行标注。

比如上面的Case2’和Case3’对应的数据，就需要这样被标注（LUIS中两个模型的标注是呈现在一起的，不过训练时会分开训练）：

语料（Utterance)	**意图（Intent）**
[00183]<-{商品Id} 号商品快递到[伊犁]<-{目的地}[邮费]<-{商品属性}多少？	商品查询
[02465]<-{商品Id}号商品有[保修]<-{商品属性}吗？	商品查询

上表仅表示含义，真正在LUIS页面上显示要简洁得多（在附录中有截图）。

知识库、知识查询和结果返回

知识库用于存储知识，本身可以是各种**形式**：数据库，API，或者文本文件等。

用户的问题经过语言理解，被提取成了意图和若干实体。下面要做的是，针对不同的知识库类型，**将意图和实体构造成对应的查询。**下表中几种情况比较常用：

知识库类型	**查询构造**	**回答生成**

关系型数据库	根据意图和实体确定table name，where条件，和目标column等要素，构建SQL Query	将SQL Query结果填注到对应模板中，生成回答问题的自然语言
Web API	根据意图和实体确定要调用的API类型和参数，构造Http Request	将Web API调用结果填注到对应模板中，生成回答问题的自然语言
结构化文本文件（json, xml等)	根据意图和实体，确定对应文件路径和对其中存储数据结构的查询	将获取内容填注到对应模板中，生成回答问题的自然语言
非结构化文本文件	根据意图和实体，确定对应文件路径	直接返回文本内容

例如，我们选择SQL Server作为小明的淘宝店小助手的知识库。则商品相关数据都存储在table中。知识库里有一个Table，名字叫Product，其中每一个row对应一种产品，每个column对应一个属性。

那么从**Case2'**中解析出来的意图和实体（意图：商品查询；实体：[目的地：伊犁，商品Id：00183，商品属性：邮费]），则可以被构造成一个SQL Query：

```
SELECT 邮费 FROM Product WHERE Product_name = '00183' AND
Destination = '伊犁'
```


Query在SQL Server中运行的结果（比如是26元），被放到一个预置的针对商品查询的答案模板里，生成答案。

预置模板：${商品Id}号商品的${商品属性}是${Query_Result}。

生成答案：00183号商品的邮费是26元。

上下文管理

同一个对话中不同语句之间共享的信息，就是**上下文（Context）**。

想要机器人具备上下文的记忆、理解功能，而不是把用户的每一个单独语句当作本轮问题的全部信息来源，就需要有一个上下文管理模块，来专门负责上下文信息的记录、查询、更新和删除（CRUD）。

每次用户新输入的信息都要先进行语言理解，再结合目前已经存储的上下文信息，或更新Context，或读取之前的Context作为补充信息。

可以将意图，和几种实体类型对应的实体值存储在Context中。

当新的用户语句输入后，若能从中提取出新的意图或实体值，则用新值更新Context，否则，读入现有的对应实体值，作为本次语言理解的补充。

Context的场景针对性非常强，很多时候需要针对不同的意图，记录不同类型的实体值。在不同意图之间切换的时候，也有可能会保留部分原有实体。这些都要针对具体情况case by case分析。

具体ContextStore中存储什么样的内容，CRUD策略是什么，都是开发者需要自己决定。

在小明的例子里，就可以将意图，和几种实体类型对应的实体值（例如Id，目标属性，目的地等）存储在Context中。

那么假设用户和机器人之间产生了如下这段对话：

> 客户：02366这款产品可以退换吗？（问题1）
>
> 客服：7天之内无理由退换。
>
> 客户：寄到南昌邮费多少啊？（问题2）
>
> 客服：10块亲。
>
> 客户：武汉呢？（问题3）

则在回答这几个问题的过程中，上下文管理模块这样工作：

1. 问题1中读取到了商品查询的意图，商品Id，和“退换“这一商品属性，将它们存入Context。
2. 问题2中读取到了”邮费“这一商品属性，和之前存储的不同，则更新Context的商品属性值，并新存入“目的地”这一实体。
3. 问题3则更新了目的地，并读取其他的包括意图、商品Id和商品属性的值，与目的地一起用来构造查询。

机器人的反问

某些情况下，Chatbot可能需要反问用户若干问题，或者和用户确认之前的某个回答，在这种情况下，就需要有内部流程控制。

例如：在商品查询的目标属性为邮费时，目的地缺失，这时候就需要主动要求用户输入对应的值。

不同场景的需求不同，这样的控制流程很难统一规划，因此需要在具体实践中根据具体需求，完成细节。

或者，也可以主动提出几个备选问题，请用户选择他们想问的。

小白开发一步步

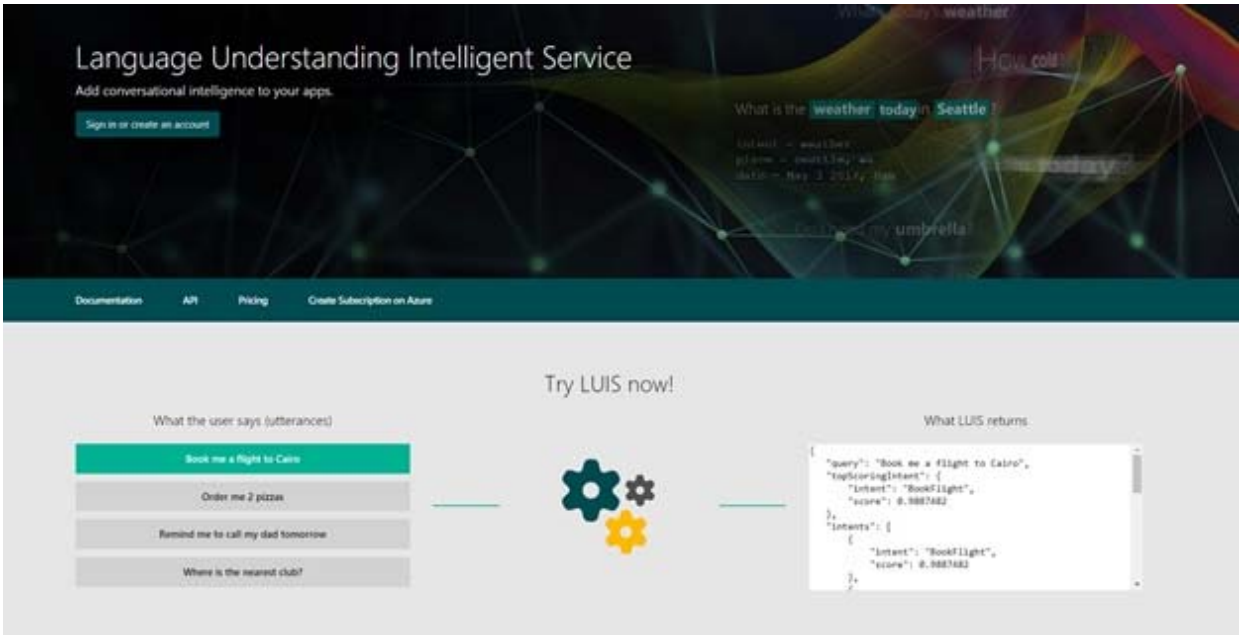
按照上一节我们说的，采用LUIS和SQL Server开发一款类似淘宝小助手这样的问题解决型聊天机器人，我们需要经过一下几步：

- 语言理解模块构建：
    - 创建一个LUIS App
    - 添加意图、实体类型
    - 定义特征
    - 输入相应数据，进行标注
    - 训练和测试
    - 发布
- 构建数据库
    - 创建一个SQL Server Database
    - create tables
    - insert data into tables
- 写一个程序负责：
    - 通过收发Http Request/Response来调用LUIS的online model进行语言理解
    - 根据LUIS解析结果构建SQL Query
    - 进行数据库查询
    - 根据数据库查询结果构造答案
    - 创建并维护Context

在完成了上述工作后，一个可以理解人类语言的聊天机器人就可以为顾客服务了。

附录：微软语言理解智能服务 LUIS

为了帮助普通开发者解决自然语言理解这一开发瓶颈，微软推出了自己的语言理解智能服务 - LUIS（https://www.luis.ai）。



LUIS (Language Understanding Intelligent Service) 的使命是让非NLP专业的开发者，能够轻松地创建和维护高质量的自然语言理解模型，并无缝对接到相关应用中去。

使用LUIS，一个Bot需要创建一个（或多个）LUIS App，然后标注所期望的输入(用户的自然语言提问)和输出(意图和实体)，再经过在线训练来获得自己的语言理解模型。

整个开发过程中，开发者只需要清晰地定义自己需要让机器理解的用户意图和实体即可，并不需要了解背后算法的细节。

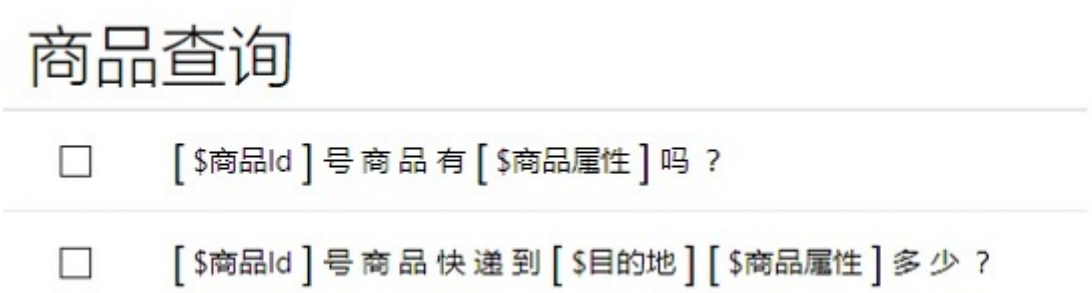
LUIS的开发流程包括三大步骤：

数据输入和标注

LUIS开发者可以在界面上轻松地进行在线数据标注。

首先，在对应的用户意图中输入自然语言语句，例如：在“商品查询”意图中输入一句“00183号商品快递到伊犁邮费多少？”；然后，通过鼠标选取实体并指定类型，例如：选择“邮费”标注为“商品属性”。

标注结果在页面上显示如下：



LUIS平台会自动从用户输入并标注的数据中提取文本特征。这些特征，包括LUIS预设的常用文本特征（从大数据语料中提取），也包括用户自定的新特征。

LUIS允许用户通过两种方式来定义新特征：

短语列表特征（Phrase List Features）

需用户自己定义若干短语列表，这些被定义在同一列表中的短语，都会被当作同一个实体类型中的实体处理。

在定义过程中，LUIS还会通过其语义词典(semantic dictionary)挖掘技术，根据用户输入的短语，自动从海量的网络数据中发现相似的短语，并推荐给用户。从而有效地提升了效率。

模式特征（Pattern Features）

也称为正则表达式特征。主要用于定义若干正则表达式。

LUIS根据这些表达式从用户输入数据中抽取符合其模式的实体。

模型的训练

LUIS的模型训练过程极其简单，开发者只需点击一下 “Train” 按钮，后台就会基于输入数据进行自动训练。

训练的时间与标注数据量相关，标注数据越多，训练所需的时间越长。同时，训练时间还与LUIS App所支持的意图和实体个数相关，意图和实体越多，训练时间也越长。

模型的测试、发布和服务

训练完模型，开发者可以对其进行性能测试。方法有两种：

交互式测试：开发者可以在页面上直接输入自然语言语句，然后目测模型输出结果。

批量测试：开发者需要上传一份测试数据，LUIS完成全部测试后给出精准率和召回率等统计数据，并针对每一项意图和实体的绘制出Confusion Matrix。

测试过的模型，只需点击“Publish”按钮，就可以发布到微软的Azure云平台上，成为一个立即可用的API Service。

开发者可以通过Http的Get方法，调用模型，对新的语句进行意图识别和实体抽取。

迭代更新

上述三个步骤是可以不断重复迭代的。

模型训练完发布上线后，可以继续输入、标注新的数据，重新训练，再次发布。如此循环往复，逐步改进质量。

使用提示

1. 当前的LUIS是一个纯粹model-based的语言理解工具。

   虽然可扩展性强，但也会有精准度不高，不容易即时修改等问题（虽然可以在线训练和发布，但作为model-based分类器和判别工具，如须改变某句话的分类，或许并不能靠添加单一的训练数据来完成）。

   在这种情况下，可以考虑LUIS和rule-based的意图、实体识别相结合。可以通过添加一系列正则表达式来匹配意图，抽取实体。

2. 每一个LUIS App都有一个内置的意图，叫做None，非常有用。一般用它来收集那些用户经常会问，但是Chat Bot并不打算回答的问题。

   若不在None中指定它们，这些问题很可能会被错误的预测为其他用户自定义的意图，造成Chat Bot的over trigger（ e.g.用户问：你是女孩吗？Bot回答：江浙沪包

邮），从而严重影响用户体验。

其他资料

LUIS Portal：https://luis.ai

以LUIS为语言理解工具，第三方API为知识的TC Bot实例：

https://github.com/juliali/WeatherBot