在config.yml中定义：

policies:

- name: MemoizationPolicy # MemoizationPolicy让bot记住最多max\_history的story.由于是context-less所以只用记住一次

max\_history: 1 默认为5

- name: MappingPolicy

Stories.md文件:

## greet ##表示story名字

\* greet \* 表示intent, 来自用户的意图

- utter\_greet -表示bot给予的反应

## thank

\* thank

- utter\_noworries

## goodbye

\* bye

- utter\_bye

domain.yml 定义bot的行动意图范围

intents:

- greet

- bye

- thank

actions:

- utter\_greet

- utter\_noworries

- utter\_bye

templates:

utter\_noworries:

- text: No worries!

utter\_greet:

- text: Hi

utter\_bye:

- text: Bye!

nlu.md # 定义用户意图

## intent:greet

- Hi

- Hey

- Hi bot

- Hey bot

- Hello

- Good morning

- hi again

- hi folks

## intent:bye

- goodbye

- goodnight

- good bye

- good night

- see ya

- toodle-oo

- bye bye

- gotta go

- farewell

## intent:thank

- Thanks

- Thank you

- Thank you so much

- Thanks bot

- Thanks for that

- cheers

Rasa train + rasa shell对bot进行训练然后和bot交互。

对训练出来的bot进行测试, 建立test\_stories.md：

## greet + goodbye 故事线

\* greet: Hi! # 用户输入，intent为greet实际内容为Hi

- utter\_greet

\* bye: Bye # 用户输入，intent为bye实际内容为Bye

- utter\_bye

## greet + thanks

\* greet: Hello there # 用户输入，intent为greet实际内容为Hello there

- utter\_greet

\* thank: thanks a bunch # 用户输入，intent为thank实际内容为thanks a bunch

- utter\_noworries

## greet + thanks + goodbye

\* greet: Hey # 用户输入，intent为greet实际内容为Hey

- utter\_greet

\* thank: thank you # 用户输入，intent为thank实际内容为thank you

- utter\_noworries

\* bye: bye bye # 用户输入，intent为bye实际内容为bye bye

- utter\_bye

rasa test --e2e --stories test\_stories.md 测试输出文件夹results. 并且含有一个文件failed\_stories.md里面包含没有通过用例。

对于这中content-less的一问一答模式我们可以使用Response Selector NLU插件。

修改nlu.md:

## intent: faq/ask\_channels

- What channels of communication does rasa support?

- what channels do you support?

- what chat channels does rasa uses

- channels supported by Rasa

- which messaging channels does rasa support?

## intent: faq/ask\_languages

- what language does rasa support?

- which language do you support?

- which languages supports rasa

- can I use rasa also for another laguage?

- languages supported

## intent: faq/ask\_rasax

- I want information about rasa x

- i want to learn more about Rasa X

- what is rasa x?

- Can you tell me about rasa x?

- Tell me about rasa x

- tell me what is rasa x

然后再新建文件在data/response.md下：

## ask channels

\* faq/ask\_channels

- We have a comprehensive list of [supported connectors](https://rasa.com/docs/core/connectors/), but if

you don't see the one you're looking for, you can always create a custom connector by following

[this guide](https://rasa.com/docs/rasa/user-guide/connectors/custom-connectors/).

## ask languages

\* faq/ask\_languages

- You can use Rasa to build assistants in any language you want!

## ask rasa x

\* faq/ask\_rasax

- Rasa X is a tool to learn from real conversations and improve your assistant. Read more [here](<https://rasa.com/docs/rasa-x/>)

然后再config.yml中添加插件ResponseSelector

pipeline:

- name: "WhitespaceTokenizer"

- name: "RegexFeaturizer"

- name: "CRFEntityExtractor"

- name: "EntitySynonymMapper"

- name: "CountVectorsFeaturizer"

- name: "CountVectorsFeaturizer"

analyzer: "char\_wb"

min\_ngram: 1

max\_ngram: 4

- name: "EmbeddingIntentClassifier"

- name: "ResponseSelector"

然后在domain.yml中加入新的intent， 新的action,

intents:

- greet

- bye

- thank

- faq

actions:

- utter\_greet

- utter\_noworries

- utter\_bye

- respond\_faq

然后再stories.md中加入新的故事线：

## Some question from FAQ

\* faq

- respond\_faq

重新训练，并且交互 rasa train + rasa shell.

加入新的用例到test\_stroies.md

## ask channels

\* faq: What messaging channels does Rasa support?

- respond\_faq

## ask languages

\* faq: Which languages can I build assistants in?

- respond\_faq

## ask rasa x

\* faq: What’s Rasa X?

- respond\_faq

基于FAQ contextless bot，构建contextual 的bot:

首先设置config.yml:

policies:

- name: MemoizationPolicy

- name: MappingPolicy

交互性的bot经常过程是先从用户手机足够的信息，然后给予回应。例如到银行汇款，我们要先收集从那个户口汇出，从哪个户口汇入，以及转账金额。Rasa称之为slot filling.

对这种要求Rasa使用form以及FormPolicy来解决，就如让用户填写表格中的信息。当Rasa检测到需要收集信息的时候，我们使用某个form作为Rasa的action反应，然后让用户提供信息。。

例如我们建立一个SalesForm,用例是用户通过Rasa练习销售部门，我们需要收集的信息有工作头衔，他们bot用途，名字, email,预算，公司。

在actions.py中定义：

from rasa\_sdk.forms import FormAction

class SalesForm(FormAction):

"""Collects sales information and adds it to the spreadsheet"""

def name(self):

return "sales\_form"

然后定义required\_slots用于定义哪些 slot需要fill

@staticmethod

def required\_slots(tracker):

return [

"job\_function",

"use\_case",

"budget",

"person\_name",

"company",

"business\_email",

]

然后在template中（domain.yml）定于如何向用户获取这些信息：

utter\_ask\_business\_email:

- text: What's your business email?

utter\_ask\_company:

- text: What company do you work for?

utter\_ask\_budget:

- text: "What's your annual budget for conversational AI? 💸"

utter\_ask\_job\_function:

- text: "What's your job? 🕴"

utter\_ask\_person\_name:

- text: What's your name?

utter\_ask\_use\_case:

- text: What's your use case?

同时在domain中定义有这些slots：

slots:

company:

type: unfeaturized

job\_function:

type: unfeaturized

person\_name:

type: unfeaturized

budget:

type: unfeaturized

business\_email:

type: unfeaturized

use\_case:

type: unfeaturized

然后我们在actions.py中定义form提交以后行为：

def submit(

self,

dispatcher: CollectingDispatcher,

tracker: Tracker,

domain: Dict[Text, Any],

) -> List[Dict]:

dispatcher.utter\_message("Thanks for getting in touch, we’ll contact you soon")

return []

然后再domain.py中定义这个form:

forms:

- sales\_form

然后添加2个intent.第一个是触发这个form,第二个是intent获取用户输入所需要的信息

## intent:contact\_sales

- I wanna talk to your sales people.

- I want to talk to your sales people

- I want to speak with sales

- Sales

- Please schedule a sales call

- Please connect me to someone from sales

- I want to get in touch with your sales guys

- I would like to talk to someone from your sales team

- sales please

## intent:inform

- [100k](budget)

- [100k](budget)

- [240k/year](budget)

- [150,000 USD](budget)

- I work for [Rasa](company)

- The name of the company is [ACME](company)

- company: [Rasa Technologies](company)

- it's a small company from the US, the name is [Hooli](company)

- it's a tech company, [Rasa](company)

- [ACME](company)

- [Rasa Technologies](company)

- [maxmeier@firma.de](business\_email)

- [bot-fan@bots.com](business\_email)

- [maxmeier@firma.de](business\_email)

- [bot-fan@bots.com](business\_email)

- [my email is email@rasa.com](business\_email)

- [engineer](job\_function)

- [brand manager](job\_function)

- [marketing](job\_function)

- [sales manager](job\_function)

- [growth manager](job\_function)

- [CTO](job\_function)

- [CEO](job\_function)

- [COO](job\_function)

- [John Doe](person\_name)

- [Jane Doe](person\_name)

- [Max Mustermann](person\_name)

- [Max Meier](person\_name)

- We plan to build a [sales bot](use\_case) to increase our sales by 500%.

- we plan to build a [sales bot](use\_case) to increase our revenue by 100%.

- a [insurance tool](use\_case) that consults potential customers on the best life insurance to choose.

- we're building a [conversational assistant](use\_case) for our employees to book meeting rooms.

将这些intent和entities加入domain当中：

intents:

- greet

- bye

- thank

- faq

- contact\_sales

- inform

entities:

- company

- job\_function

- person\_name

- budget

- business\_email

- use\_case

然后加入story当中：

## sales form

\* contact\_sales

- sales\_form

- form{"name": "sales\_form"}

- form{"name": null}

然后将FormPolicy加入config文件中：

policies:

- name: MemoizationPolicy

- name: KerasPolicy

- name: MappingPolicy

- name: FormPolicy

最后uncomment endpoints.yml中的action\_endpoint

action\_endpoint:

url: <http://localhost:5055/webhook>

开启action server: rasa run actions

然后训练并且交互：rasa train, rasa shell