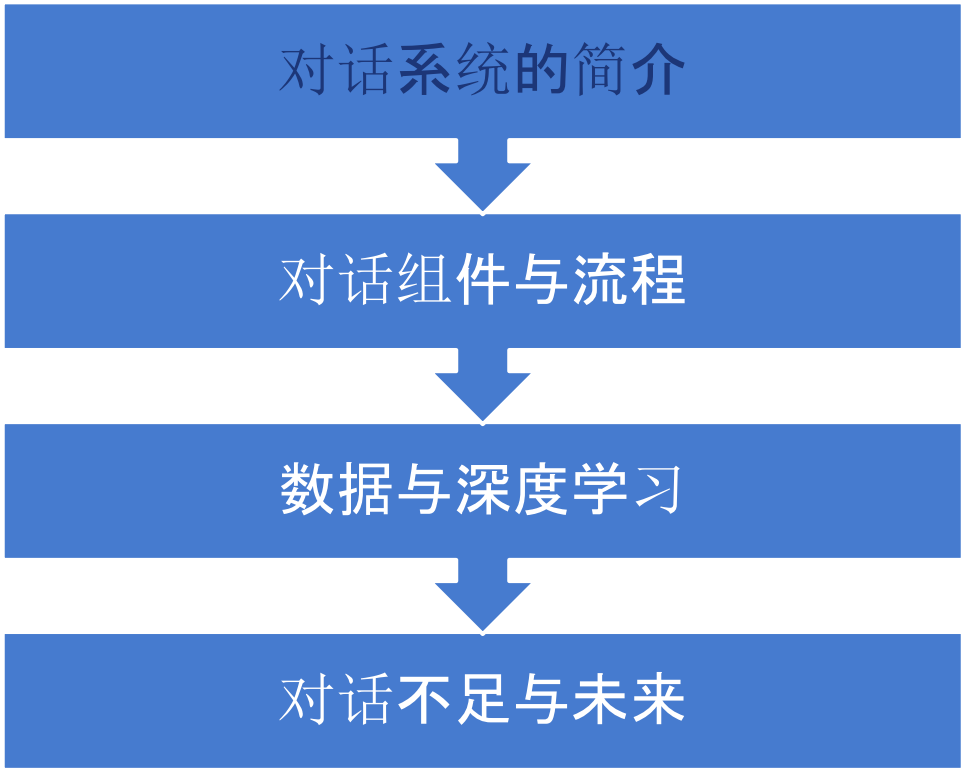

对话系统的简介
对话组件与流程
数据与深度学习
对话不足与未来

什么是对话系统

- 对话系统是一个：人机交互界面
- 用更时髦的说法是：Conversational UI
- 通过文字、语音等自然语言交流方式，完成人机交互

对话系统的场景

1. 1、受限场景
2. 2、对话比其他人机交互方法成本低或效率高
    1. 学习成本低
    2. 认知成本低
    3. 任务效率高

对话系统的分类

- 从需要达到的目的来分类，对话系统至少包括：

- 任务型 – 完成任务
- 问答型 – 回答问题
- 闲聊型 – 制造快乐

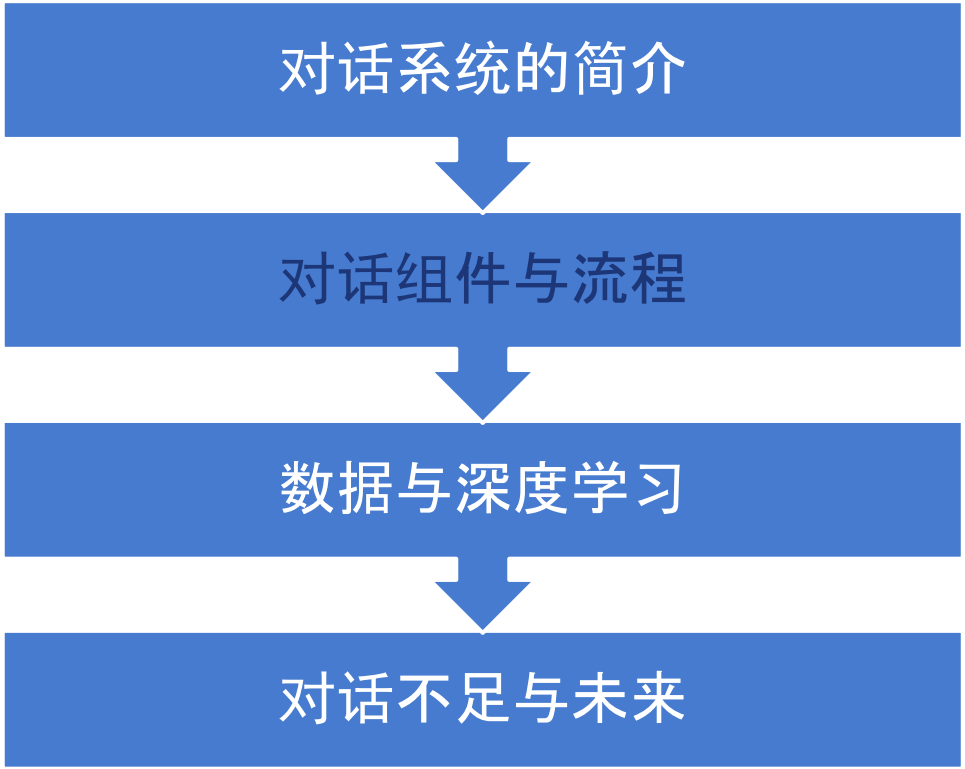
对话系统的目标

- 从成功标准来看：

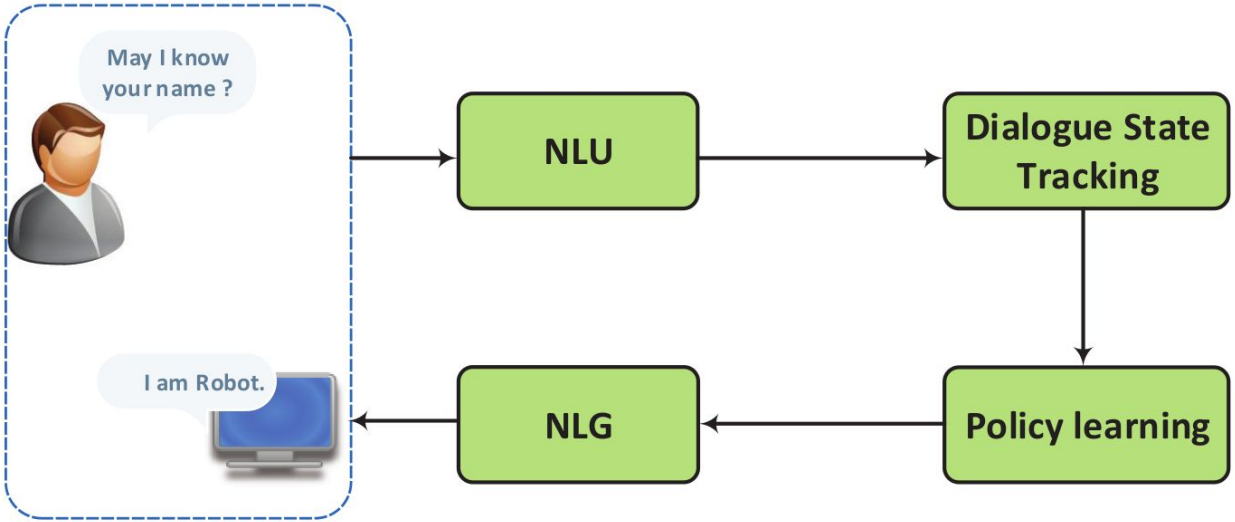
- 任务型 – 快速解决问题
- 问答型 – 问题回答准确
- 闲聊型 – 聊！一直聊！

多轮对话？

- 多轮对话是一个伪命题，任何形式系统都期望是多轮的


对话系统的简介
对话组件与流程
数据与深度学习
对话不足与未来

对话系统结构



A Survey on Dialogue Systems: Recent Advances and New Frontiers
[Chen et al., 2014]

PIPELINE的目的

- 1、顺序工作符合人类思考逻辑
- 2、各个子模块解决不同任务，分工协作
- 3、各个模块的输入和输出可以独立修改规则

任务型对话系统实现 — FRAME(帧)

- In general, a frame is a data structure potentially containing a name, a reference to a prototype frame, and a set of slots.

- GUS, A Frame-Driven Dialog System
- [Bobrow et al., 1977]

填满帧 = 任务完成

一个UNIT中的火车票购票样例

1、所有必要内容填满代表任务完成（至少是必要条件）
2、所有其他部分都是为了填满帧，所有系统动作都是为了帮助用户填满帧

对话意图名：BOOK_TICKET

别名：订票

添加词槽

词槽名	词槽别名	澄清话术
user_time	出发时间	什么时间出发?
user_from	出发站点	从哪出发?
user_to	到达站点	要到哪?
trainnumber	车次	要哪个车次的?

https://ai.baidu.com/forum/topic/show/869808

对话系统结构 — PIPELINE — 伪代码

- 假设我们用伪代码(Python伪代码)的方式描述对话系统流程

- 假设所有的模块，包括机器人本身都是函数

- 用户输入的对话字符串是utterance，那么整个机器人函数看上去会是这样：

```
def bot(utterance):
    x = utterance
    x = nlu(x)
    x = dst(x)
    x = dpl(x)
    x = nlg(x)
    return x
```

幸运的是，事实上还真是这样

NLU的目标

- 自然语言理解目标：拆解一句话的完整意义

NLU的接口

- 输入:用户说的一句话
- 输出:针对这一句话的语义分析结果，称之为用户行为

```
class NaturalLanguageUnderstanding(object):
    def forward(self, utterance: str) -> UserAction:
        user_action = UserAction()
        # 处理逻辑
        return user_action
```

DST的目标

- 对话状态追踪：根据上下文，改变当前的对话状态
- 这里的状态包括每一轮：用户说了什么，系统做了什么，系统的状态记忆

DST的接口

输入:用户行为，状态历史

输出:新的对话状态

```
class DialogStateTracker(object):
    def forward(self,
                init_state: DialogState,
                history: List[DialogState],
                user_action: UserAction
            ) -> DialogState:
        new_ds = DialogState()
        # 处理逻辑
        return new_ds
```

DPL的目标

- 对话策略学习：基于给定的对话状态（和历史），作出给出下一步系统的行为

DPL的接口

输入:对话状态与历史

输出:系统行为

```
class DialogPolicy(object):
    def forward(self,
                history: List[DialogState]
        ) -> SystemAction:
        system_action = SystemAction()
        # 处理过程
        return system_action
```

NLG的目标

- 自然语言生成：根据系统行为，生成用户可以理解的自然语言

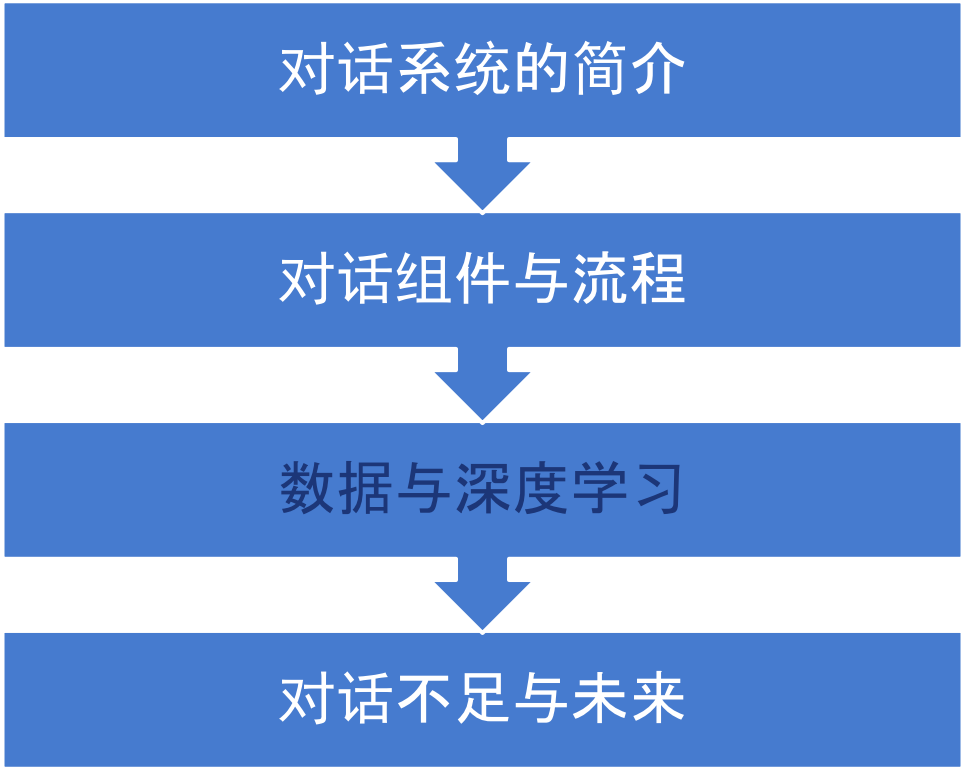
- 目标：
- 含义准确：应该准确表达出机器人想表达的含义
- 信息充足：在不影响阅读的情况下，把充足的信息提供给用户
- 讲话流利：语言流畅，不影响阅读
- 高可读性：当阐述多条信息时，应该有良好的组织性
- 并不无聊：最好能根据用户情绪、信息内容本身等有一定变化

NLG的接口

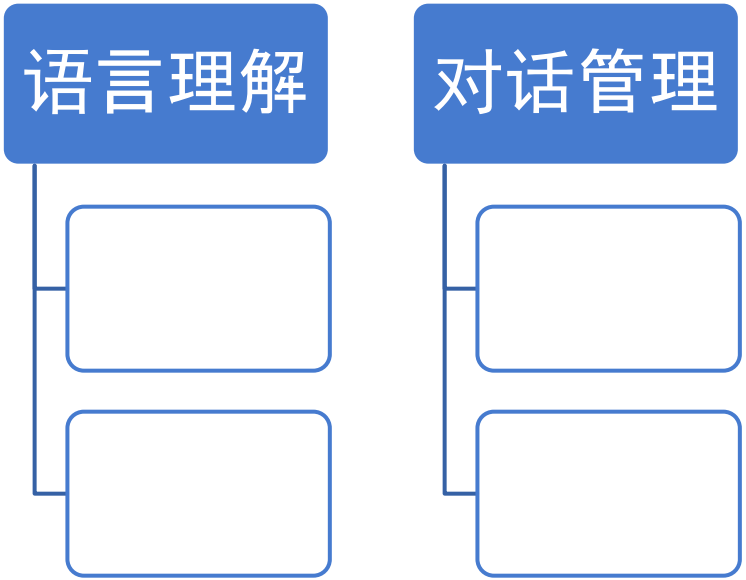
输入:系统行为

输出:系统给用户的回复

```
class NLG(object):
    def forward(self,
                system_action: SystemAction) -> str:
        system_response = SystemResponse()
        return system_response
```


对话系统的简介
对话组件与流程
数据与深度学习
对话不足与未来

不同组件的深度学习构建



NLU数据

语句：我要买去 北京 的车票，今天 晚上 出发

领域：travel

意图：inform

实体：目的地 : 北京 ×　日期时间 : 晚上 ×

NLU模型 — 意图、领域识别

- Input: 我要买去北京的车票，今天晚上出发
- Output Intent: inform
- Output Domain: travel

Intent Detection = 短文本分类任务

NLU INTENT KERAS — 一个简易的文本分类器

```
def make_model(n_size, n_output, n_embedding, n_vocab):
    model = Sequential()
    model.add(Embedding(n_vocab, n_embedding, mask_zero=True))
    model.add(LSTM(n_size))
    model.add(Dense(n_output))
    model.add(Activation('softmax'))
    model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
    return model
```

NLU模型 — 实体识别(语义槽识别)

- 序列标注：
- Input ： 我 要 买 去 北 京 的 车 票， 今 天 晚 上 出 发
- Output: O O O O O Bc Ic O O O O O O Bd Id O O

- Bc = Beginning-City, Ic = Inside-City
- Bd = Beginning-Date, Id = Inside-Date
- O = Outside

IOB: https://en.wikipedia.org/wiki/Inside%E2%80%93outside%E2%80%93beginning_(tagging)

NLU NER KERAS — 一个简易的NER实现

```
def make_model(n_size, n_output, n_embedding, n_vocab):
    """一个简单的NER实现，没有CRF层"""
    model = Sequential()
    model.add(Embedding(n_vocab, n_embedding, mask_zero=True))
    model.add(Bidirectional(LSTM(n_size, return_sequences=True)))
    model.add(Bidirectional(LSTM(n_size, return_sequences=True)))
    model.add(TimeDistributed(Dense(n_output)))
    model.add(Activation('softmax'))
    model.compile(optimizer='adam', loss='categorical_crossentropy')
    return model
```

对话数据

对话数据本质上我们可以认为是一段故事，或者剧本

右边的剧本来自于Rasa，一家使用机器学习完成对话机器人构建的公司

https://raw.githubusercontent.com/RasaHQ/rasa_core/master/examples/restaurantbot/data/stories.md

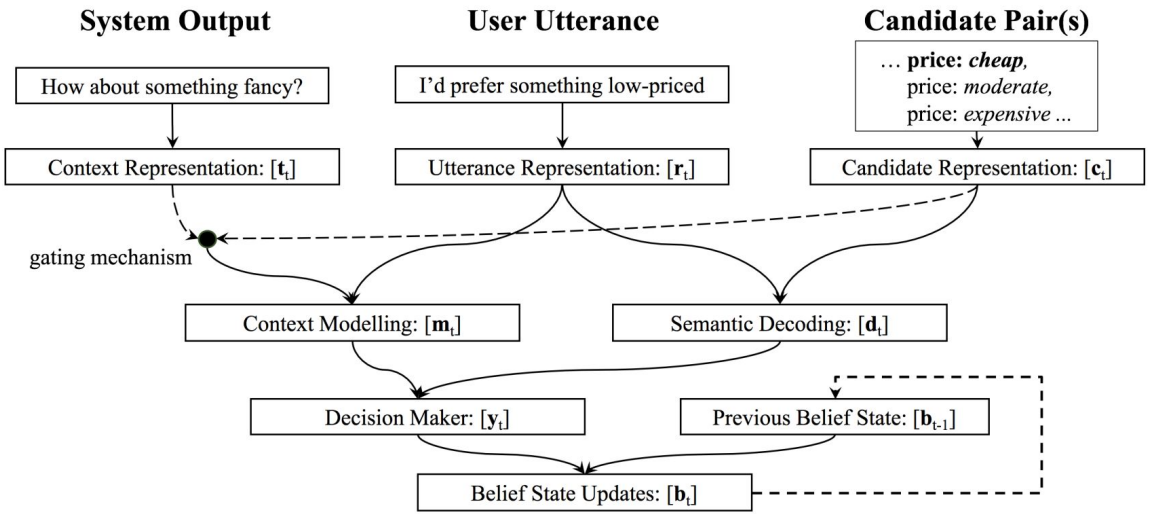
## story_00914561	
* greet	用户：你好
- utter_ask_howcanhelp	系统：我能帮你做什么呢？
* inform{"cuisine": "italian"}	用户：我要预订一家意大利餐厅
- utter_on_it	系统：我在听
- utter_ask_location	系统：你要找哪里的?
* inform{"location": "paris"}	用户：巴黎
- utter_ask_numpeople	系统：要几个人
* inform{"people": "six"}	用户：6个人
- utter_ask_price	系统：要什么价位的
* inform{"price": "cheap"}	用户：便宜的
- utter_ask_moreupdates	系统：如果你要修改什么请告诉我
* inform{"people": "two"}	用户：换成两个人吧
- utter_ask_moreupdates	系统：如果你要修改什么请告诉我
* inform{"location": "madrid"}	用户：马德里的餐厅
- utter_ask_moreupdates	系统：如果你要修改什么请告诉我
* inform{"cuisine": "spanish"}	用户：还是西班牙菜吧
- utter_ask_moreupdates	系统：如果你要修改什么请告诉我
* deny	用户：就这样不换了
- utter_ack_dosearch	系统：好的我开始搜索了
- action_search_restaurants	（后台的搜索功能）
- action_suggest	（后台的推荐功能）

对话模型 — 定义

- In slot-based spoken Dialogue systems, tracking the entities in context can be cast as **slot carryover task** only the relevant slots from the dialogue context are carried over to the current turn.

Improving Long Distance Slot Carryover in Spoken Dialogue Systems
[Chen et al., 2019]

对话模型 — 对话状态更新 — 复杂版



Fully Statistical Neural Belief Tracking
[Mrkšić et al., 2018]

DST KERAS

```
def make_model(n_size, n_output):
    model = Sequential()
    model.add(LSTM(n_size))
    model.add(Dense(n_output))
    model.add(Activation('sigmoid'))
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])

    return model
```

对话模型 — 对话状态策略学习 — 机器学习

- 将对话上下文建模
- 给出系统行为概率分布

DPL KERAS

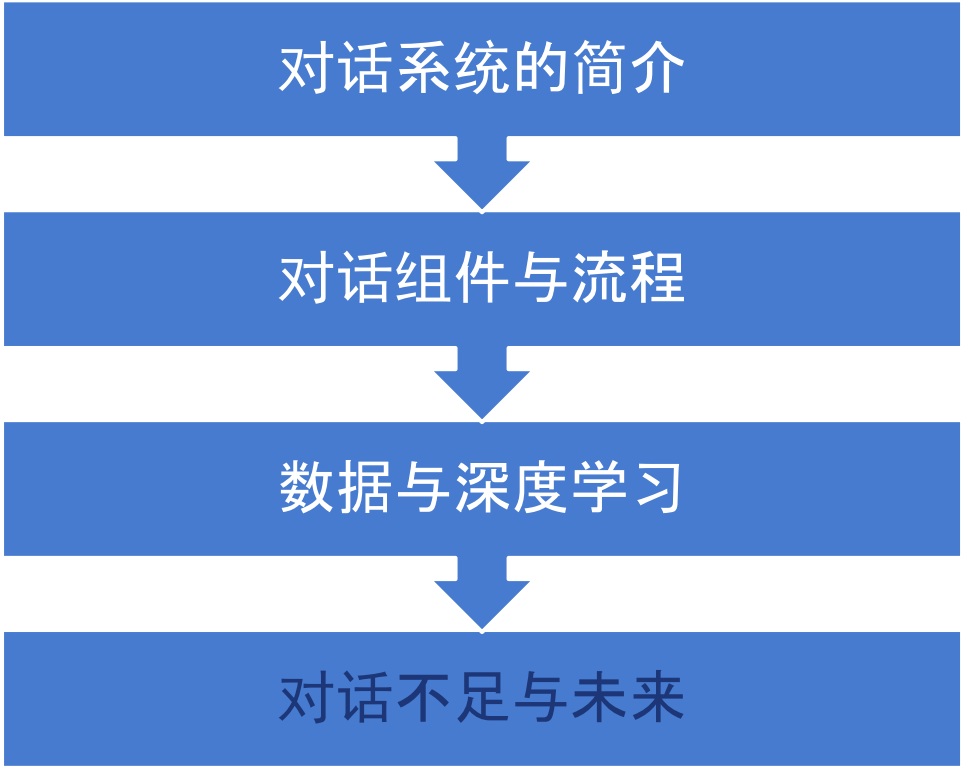
```
def make_model(n_size, n_output):
    model = Sequential()
    model.add(LSTM(n_size))
    model.add(Dense(n_output))
    model.add(Activation('softmax'))
    model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])

    return model
```

NLG模型 — 大部分时候简单就好

```
def hello(user_name=None):
    If user_name is not None:
        return f'你好, {user_name}'
    return '你好'
```

NLG模型 — 更好

- 1、基于用户情绪或用户画像的专家系统
- 2、在拥有更多对话生成数据的情况下，提供基于Sequence-to-sequence的NLG模型。参考文章 A Context-aware Natural Language Generator for Dialogue Systems [Dušek et al., 2016]


对话系统的简介
对话组件与流程
数据与深度学习
对话不足与未来

强化学习 — 模型的未来？

- 1、如何进行User Simulation
- 2、如何和规则模型互动
- 3、如何个解决训练数据问题

不足

- 1、系统脆弱：对话流程如果脱离设计的时候，容易出现偏差
- 2、工具欠缺：依然还是缺乏足够方便的工具，即便我们有LUIS、UNIT、RASA
- 3、规范欠缺：没有一个主流设计思想引导，缺乏系统性
- 4、数据成本：数据收集成本高，对话很难扩展

未来

- 1、以产品设计思想为引导，先进工具为前提
  - 在足够的系统规范下，以各种方便的可演进的工具为前提
- 2、设计在有限制的情况下更高效的对话
  - 对机器人充分理解的设计下，利用“游戏规则”，把握用户预期
- 3、与其他自然语言产品的融合
  - 游戏、搜索、APP，互补与借鉴

我们还有很多没说

- 1、数据挖掘与扩展
- 2、产品如何设计
- 3、问答系统在哪
- 4、如何组合一切

演示

```
user:你好
user_action {'text': '你好', 'tokens': ['你', '好'], 'intent': 'hello', 'domain': '', 'slots': [], 'ner_slot_f
iller': {'slots': []}}
pred [0. 0. 0. 0. 0. 0. 0. 0.]
new state after fill ----------
user_domain: None
user_intent: None
slots:
informable      城市    None
informable      时间    None
informable      任务    None
informable      数字    None
requestable     城市    None
requestable     时间    None
requestable     任务    None
requestable     数字    None
sys_intent: None
---------
new system_action hello
sys: 你好啊，我是帕蒂安
user:
```

THANKS

- 所有文中代码: https://github.com/deepdialog/deepdialog
- 2019/10