文章编号：2096-1472(2021)-02-45-04
DOI:10.19644/j.cnki.issn2096-1472.2021.02.009

一种对话机器人开发技术综述与系统架构实现

孙小鱼[1,2]

(1.大连东软信息学院，辽宁 大连 116023；
2.大连东软教育科技集团有限公司研究院，辽宁 大连 116023)
✉sunxy@neuedu.com

摘　要：对话机器人是使用自然语言处理与生成技术，模拟人类对话逻辑并与人进行交流的计算机程序。作为新一代人工智能产品的人机交互主要入口，逻辑实现方式与交互是对话机器人设计的关键。本文结合对话机器人的技术特点，在实现过程中采用基于Python语言的Django微服务Web应用开发框架，将对话算法模型与逻辑处理过程进行微服务化API封装，使其他应用能够请求对话服务的接口进行功能的二次开发。为了提高对话响应速度，本文采用了关系型数据库MySQL与基于内存的非关系型数据库Redis结合的方式，减少算法模型对硬盘的读取次数，优化了用户体验。

关键词：微服务；Django；MySQL；Redis
中图分类号：TP311.5　**文献标识码**：A

Overview of Chatbot Technology and System Architecture Implementation

SUN Xiaoyu[1,2]

(1.*Dalian Neusoft University of Information*, *Dalian* 116023, *China*;
2.*Research Institute*, *Dalian Neusoft Education Technology Group Co*. *Limited*, *Dalian* 116023, *China*)
✉sunxy@neuedu.com

Abstract: A chatbot is a computer program that simulates human conversation and communicates with human through natural language processing and generation technology. As main access to human-computer interaction for a new generation of artificial intelligence products, logical implementation and interaction are critical to the design of chatbots. Based on technical characteristics of chatbots, this paper adopts Python-based Django micro-service web application development framework in the implementation and encapsulates dialogue algorithm model and logic processing process into a micro-service API (Application Programming Interface). Thus, other applications can request the interface of dialogue service for a secondary development of functions. In order to improve the efficiency of query processing, a memory-based non-relational database Redis (Remote Dictionary Server) is used with relational database MySQL (Structured Query Language) to reduce the number of reads from hard disk and so to optimize system performance.

Keywords: micro-services; Django; MySQL; Redis

1 引言(Introduction)

对话机器人是当前人工智能领域的研究热点，它应用自然语言处理与生成技术，以聊天界面或API为基础，能够与人进行语音或文本对话，以聊天的方式解决用户的需求[1]。对话算法主要包括以下几个过程：自然语言理解、对话状态追踪[2]、对话内容管理[3]、自然语言生成[4]。但是上述过程的核心算法都是基于Python编程语言开发的，与网站软件开发或终端硬件开发的技术架构并不兼容。为了适应越发复杂的大型网站系统单体架构，微服务技术应运而生，其核心思想是各种自治的子系统协同工作，共同完成大型网站的功能和业务需求[5]。

对话机器人与其他微服务程序不同的地方在于：处理对话逻辑需要大量高频重复且复杂的计算过程和频繁的存取硬盘数据的过程。随着使用对话的用户规模变大，由于硬件资源有限，对话程序在运行过程中很容易造成对话响应速度变慢、对话逻辑处理不完整甚至是服务宕机等现象，严重影响

用户体验。

针对上述问题，本文在技术实现过程中采用基于Python语言的Django开发框架，并根据业务逻辑将对话模型进行模块拆分，在模块之间添加Redis缓存[6]，用内存中的读写过程替代了高频的复杂计算和高频的硬盘存取过程。

2 相关技术(Related technologies)

2.1 Python

Python是由Guido van Rossum在20世纪80年代末到90年代初于荷兰国家数学和计算机科学研究所设计出来的一种结合了解释性、编译性、互动性和面向对象的高级程序设计语言。它具有易于学习与维护、可移植、可扩展、可嵌入等特点。Python由于代码简洁、基础代码库完善等优势，已经应用于人工智能、Web应用开发、运维系统开发、游戏开发等重要领域。Python语言在人工智能领域被广泛应用是因为它对科学计算的友好支持[7]，包括基础的Numpy、Pandas、Scikit-learn等科学计算工具箱和Tensorflow、Pytorch、Caffe等深度学习开发框架都是以Python语言为基础进行封装的。

2.2 Django

Django是一款基于Python的免费开源的Web应用开发框架，采用了MVT的框架模式，即模型M(Model)、视图V(View)和模板T(Template)，于2008年9月发布了第一个正式版本1.0。MVT模式将传统MVC模式中的Controller层拆分成业务逻辑操作与请求的处理与转发，将业务逻辑操作结合到了View层中，View通过调用Model层与Template层，根据HTML、CSS、JavaScript等对数据进行渲染后返回给前端浏览器界面。同时新增了一个URL分发器，负责将不同的URL请求分发给不同的View进行业务逻辑处理。Django拥有强大的数据库功能，并自带一套功能完备、界面美观的管理后台。由于Django具有低耦合、开发快捷、部署方便、复用性高、运维成本低、完善的开发者社区等优点，近些年来许多大型网站的开发都选择Django作为Web应用核心开发框架，例如YouTube、Google、Instagram、Reddit，以及国内的豆瓣、知乎等。

Django框架原理如图1所示。首先，客户端向服务发送HTTP请求，由WSGI(Web Server Gateway Interface)模块统一进行协议处理后，发送给请求中间件，并由请求中间件交由URL分发器进行统一分发。若找不到对应的URL地址，则直接返回给响应中间件，经过WSGI模块统一处理后返回给浏览器。若找到对应的URL地址，则由视图中间件进行处理后映射到View层中的函数。View层的函数根据业务逻辑判断是否需要调用Model层，Model层利用对象关系映射(Object-Relational Mapping,ORM)[8]将数据库表抽象成Python中的类，数据库表中的字段抽象成Python类中的属性，通过对Model层的类的操作得到数据库的API，避免了复杂的数据库语句编写。View层在调用Model层的同时从Template层的静态资源库中调用HTML/CSS/JavaScript资源，并将数据和页面解析、渲染成HTML页面，经事务中间件处理后返回给响应中间件，由响应中间件交给WSGI模块统一处理后返回给浏览器。

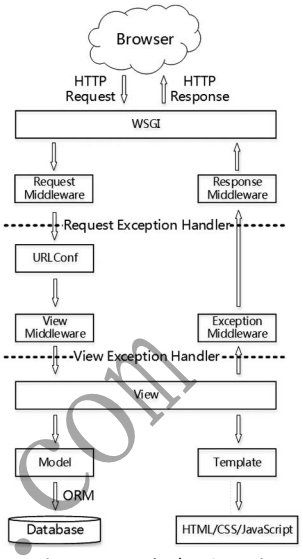

图1 Django框架原理图

Fig.1 Schematic design of Django framework

2.3 MySQL数据库

MySQL在Web应用方面是最好的RDBMS(Relational Database Management System，关系数据库管理系统)应用软件之一，由瑞典的MySQL AB公司开发，属于Oracle旗下产品。数据库(Database)是按照数据结构来组织、存储和管理数据的应用程序。关系型数据库是建立在关系模型基础上的数据库，借助于集合、代数等数学概念和方法来处理数据库中的数据。在关系型数据库中，数据以表(Table)的形式出现，表中的每一行称为一条记录，表中的每一列称为一个属性，表与表之间以代数关系为基础组成完整的系统数据。数据库表的结构设计决定了数据库的性能，合理的数据库结构可以让使用者在每次访问最少数量的表的前提下完成业务需求。MySQL数据库使用标准的SQL语言形式对数据进行增加、删除、修改、查询等操作，由于其体积小、速度快、成本低、使用灵活等特点被应用于中小型网站开发。

2.4 Redis

远程字典服务(Remote Dictionary Server，Redis)是一个基于key-value形式存储的NoSQL数据库[9]，用哈希表表示key与value之间的映射关系，可用key值迅速查询对应的value值[10]。Redis使用ANSI C编写，包含五种数据结构，并且支持网络通信和多语言API，遵守BSD协议，可扩展性强，可选作持久化存储[11]。由于Redis的数据以key-value的形式存储在内存中，且Redis的事务操作具有原子性，因此比硬盘读写更迅速，读取速度可以达到110,000 次/秒，写入速度

可以达到81,000 次/秒[12]。Redis通常应用在“高频读、低频写”的热点数据缓存、计数器、消息队列、排行榜、社交网络、订单系统等高吞吐、高并发场景。

3 流程设计与实现(Process design and implementation)

3.1 对话服务问答流程

基于Django的对话机器人服务流程如图2所示。

(1)对话服务从前端接收参数sentence和userId。其中sentence为用户输入的句子，userId为用户的唯一标识。

(2)Django框架中的View层函数解析参数sentence和userId。

(3)以sentence为key在Redis的nlu目录中进行查询。若查询到记录，则将value取出作为dialogue，并更新生命周期为604,800 秒；若查询不到记录，则将sentence传入NLU模块进行理解，输出作为dialogue，并将最新的理解结果写入Redis。其中，dialogue为NLU模块进行理解后的计算结果。

(4)以userId为key在Redis的userdst目录中进行查询。若查询到记录，则从value列表取出dst.state和keyword，并更新生命周期为604,800 秒；若查询不到记录，则初始化dst.state和keyword并写入Redis。其中，dst.state为该用户的对话状态追踪信息，keyword为该用户上次对话的关键词。

(5)将sentence，dialogue，dst传入DPL对话策略学习模块进行意图识别。若DPL模块处理结果显示为非问答，则给出回应，并传回给前端；若DPL模块处理结果显示为问答，则将处理后的问题question传入QA模块。

(6)以question为key在Redis的qa目录中进行查询。若查询到记录，则从value列表中取出answer，并更新生命周期为604,800 秒；若查询不到记录，则从数据库中扫描关键词并进行问答对匹配传回给前端，同时将问答对写入Redis，并更新生命周期为604,800 秒。

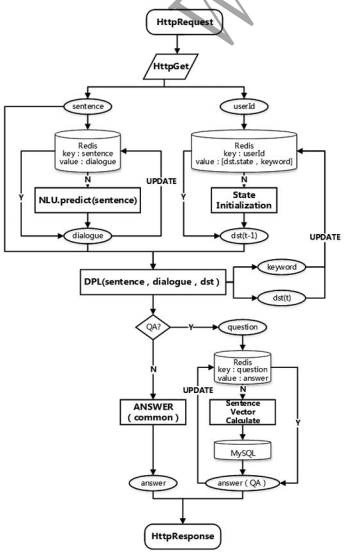

图2 对话机器人服务流程图

Fig.2 Flow chart of chatbot dialogue service

通过上述流程，高频输入的句子、高频活跃的用户对话状态追踪和高频匹配的问答对全部被缓存至Redis之中。其他用户请求服务时，若输入的是经常被输入的语句和经常被理解的问题，则直接从Redis数据库中读取内容，不必进行烦琐的计算过程和缓慢的读硬盘过程。

3.2 MySQL数据库结构设计

由于对话机器人模型首先需要扫描对话域内的关键词，匹配成功后再扫描该关键词分支下的全部问答对，因此将关键词表中的keyword字段作为问答对表的外键。关键词表和问答对表如表1和表2所示。

表1 关键词表keyword_info

Tab.1 Keyword table keyword_info

序号	字段名	类型	属性	描述
1	id	int(11)	主键	关键词id
2	keyword	varchar(64)	非空	关键词
3	create_by	varchar(64)	非空	创建人
4	create_time	datetime(6)	非空	创建时间
5	update_by	varchar(64)	默认空	修改人
6	update_time	datetime(6)	默认空	修改时间

表2 问答对表qa_info

Tab.2 Question and answer table qa_info

序号	字段名	类型	属性	描述
1	id	int(11)	主键	问答对id
2	keyword_id	int(11)	外键	关键词外键
3	keyword	varchar(64)	非空	关键词
4	question	varchar(64)	非空	问题
5	answer	varchar(64)	非空	答案
6	create_by	varchar(64)	非空	创建人
7	create_time	datetime(6)	非空	创建时间
8	update_by	varchar(64)	默认空	修改人
9	update_time	datetime(6)	默认空	修改时间

3.3 Redis存储策略设计

根据对话机器人模型的流程可知，高频输入语句的NLU计算结果、用户状态和用户对话关键词、高频问答对匹配结果可以在Redis进行缓存处理。其中，高频输入语句的NLU计算结果用set命令存入chatbot:nlu目录下，以sentence为key，以dialogue为value，生命周期为604,800 秒；用户状态和用户对话关键词用lpush命令存入chatbot:userdst目录下，以userId为key，以dst和keyword为value列表，生命周期为604,800 秒；高频问答对匹配结果用set命令存入chatbot:qa

目录下，以question为key，以answer为value，生命周期为604,800秒。

4 性能分析(Performance analysis)

4.1 系统硬件指标及测试工具

本文的对话机器人服务开发采用的是Windows 64位操作系统上位机，处理器为Intel(R)Core(TM)i7-8565U CPU @ 1.80 GHz 2.00 GHz，核心数8个，内存为16 GB、2400 MHz频率，服务部署于开发机上。性能指标测试收集工具采用的是Postman请求模拟器v7.34.0和Apache Server中的ab.exe组件。

4.2 单个请求响应时间测试

首先由一个新用户向对话服务发问，然后更换用户再向对话服务问相同的内容，记录请求的响应时间。如图3和图4所示，新用户向对话服务发问时请求完成时间为345 ms，而更换用户后问相同问题时请求完成时间仅为83 ms。因为前一名用户的语义理解结果和问答对匹配结果已经被记录至Redis中，新用户的发问请求只需在Redis中读取数据，无须进行复杂的计算和数据库读取。

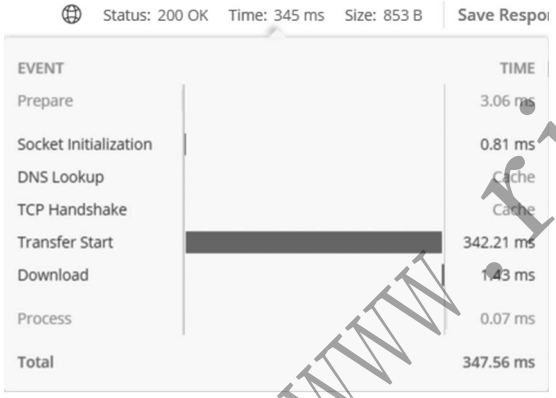

图3 新用户第一次发问的请求响应时间

Fig.3 Response time of a new user's first question

图4 其他用户问相同问题的请求响应时间

Fig.4 Response time of the same question asked by other users

4.3 并发请求测试

设置Apache Server ab.exe组件的参数-n与-c分别为{0,10,100,1000,10000}和{0,10,100,1000,10000}。其中，-n为总请求数，-c为并发请求数。记录服务的吞吐率(Requests per Second，RPS，单位为req/s)和平均请求完成时间(Time per Request，TPR，单位为ms)，测试结果如表3所示。

表3 总请求数与并发请求数的性能统计表

Tab.3 Performance statistics of total requests and concurrent requests

-c \ -n	1	10	100	1000	10000
1	RPS:186.74 TPR:5.355	RPS:311.36 TPR:3.212	RPS:320.57 TPR:3.119	RPS:266.08 TPR:3.758	RPS:257.41 TPR:3.885
10		RPS:370.91 TPR:2.696	RPS:431.46 TPR:2.319	RPS:395.38 TPR:2.529	RPS:285.09 TPR:3.508
100			RPS:24.70 TPR:40.481	RPS:24.76 TPR:40.383	RPS:24.71 TPR:40.462
1000				RPS:24.41 TPR:40.964	RPS:25.05 TPR:39.919
10000					RPS:24.11 TPR:41.482

由测试结果可知，对话服务在4.1节的运行条件下，接收100个并发请求的时候达到性能瓶颈，此时RPS达到25左右，且随着并发量增加RPS不再变化；TPR为40 ms左右，且随着并发量增加TPR不再变化。

5 结论(Conclusion)

本文从对话机器人算法流程出发，根据对话算法使用框架的特点，在技术实现过程中采用基于Python编程语言的Django微服务框架对算法业务逻辑进行API封装，以便他人调用API接口进行对话功能的二次开发；同时将复杂的计算流程和硬盘存取流程拆分，并将高频访问的句子、高频使用用户的对话状态追踪信息和高频匹配的问答对进行Redis缓存处理及MySQL持久化存储处理。对性能分析的结果表明，基于Django框架和Redis缓存的对话服务开发流程减少了计算过程和硬盘存取过程，能够应对大规模的数据存储需求。对话服务的请求响应迅速，吞吐率指标正常，在保证性能的同时优化了用户体验。

(下转第29页)