小能-替代阿里云接口
软件设计说明书

V1.3

小能科技（北京）有限公司

2018 年 4 月 2 日

文档过程记录表

时间	修正内容	校对人	版本号	备注
2018.03.13	撰写文档	张冬松	**1.0**	
2018.03.13	补充数据库设计	郭姿含、魏志飞、张冬松	**1.0**	
2018.03.13	补充 **API** 接口	郭姿含、魏志飞、张冬松	**1.0**	
2018.03.16	重新设计训练、**QA** 命令处理模块	魏志飞、张冬松、郭姿含	**1.1**	
2018.03.18	修订文档	张冬松	**1.1**	
2018.03.19	修订文档	张冬松	**1.1**	
2018.03.20	修订文档	张冬松	**1.2**	
2018.04.02	修订文档	张冬松	**1.3**	

目　　录

第一章 前言

1.1 项目背景

A.待开发软件名称

替代阿里云接口之 Chatterbot+软件 V1.3

B.基本概念

现有阿里云中智能对话平台的智能问答系统，能够精确地理解以自然语言形式描述的用户提问，并通过检索问答知识库寻找语义上匹配的问题描述，并且返回知识点内容。待开发软件基于 Python 编程环境，构建自主可控的分布式智能对话平台，功能和性能上均能替代阿里云的智能对话平台，为小能上层应用系统提供单轮的智能问答服务。

C.产品需求背景

现有小能在线客服系统中智能机器人 Xbot 在完成访客问答时，实际上是调用阿里云中智能对话平台的智能问答系统来完成单轮的智能问答服务。虽然小能上层应用系统与阿里云结合起来运行良好，但是随着应用的发展，已经开始出现不能满足用户不断增长的应用需求。现阶段，用户提出较大应用需求的地方主要在于整个 QA 服务能否实现本地化，进而满足这部分大用户希望保护私有数据的需求。另外，虽然购买阿里云服务的成本支出目前处于可控水平，但如果能够在小能产品架构中增加可以替换阿里云服务的自研组件，从技术和成本角度上看对公司未来发展和成本控制有利。

1.2 参考资料

[1]上层业务系统中 Xbot 机器人 QA 流程的 Confluence 地址：

https://www.processon.com/view/link/59c32938e4b0bc4fef8a9102

[2] 软件 API 接口说明

http://confluence.xiaoneng.cn/pages/viewpage.action?pageId=45417663

[3]阿里云接口说明

https://help.aliyun.com/document_detail/50473.html?spm=5176.doc50471.6.590.P0aBd1

[4]阿里云 API 接口用法

https://help.aliyun.com/document_detail/50473.html?spm=a2c4g.11186623.6.588.teyv48

[5]Jaccard 相似性指数：

https://en.wikipedia.org/wiki/jaccard_index；

[6]python 代码实现详见：

http://chatterbot.readthedocs.io/en/stable/_modules/chatterbot/comparisons.html#JaccardSimilarity

[7] python 代码实现方案详见：

https://www.jianshu.com/p/466cf6624e26

[8] 使用 python 中 NLTK 工具实现文本情感分析详见：

https://www.zhihu.com/question/59050093/answer/168200401

1.3 术语定义

KB：Knowledge Data 知识库

KG：Knowledge Graph 知识图

QA：Question Answer 问答

CBot：Chatterbot 对象

Hub 层：中间层

第二章 系统概述

2.1 系统目标

系统目标在于基于原生 Chatterbot 库、Django 和 Celery 框架，结合 MySQL 数据库构建的语料库和 MongoDB 数据库构建的知识图库，构建自主可控的分布式智能对话平台，通过实现 12 个外部 Web API 接口和 1 个内部 Web API 接口，能够在功能和性能上替代阿里云的 21 个智能对话 API 接口。

2.2 需求概述

结合小能上层业务系统的实际需求，替代阿里云所需的外部 API 接口计划开发 12 个，主要分为如下 4 大类需求：REST API 接口处理、语料库命令处理、训练命令处理和 QA 命令处理。此外，还包括 1 个内部 API 接口，用于实现 QA 功能。

2.3 运行环境概述

运行软件平台：如表 1 所示。

表 1 运行软件平台

所需软件	Linux	Django	Python	Chatterbot	MySQL	MongoDB	其他
要求版本	Kernel 3.0 以上	2.0.1（最新）	3.4 及以上	3.8 （最新）	3.5 及以上	3.4 及以上	（满足安装条件）
简介	服务器操作系统	底层 API 封装工具	开发支持语言	底层开源 Python 库	结构化数据库	非结构化数据库	多种 Python 包
作用	运行平台	实现 Web API 接口	编程环境	底层 QA 组件	存储语料库	存储知识图	Python 第 3 方工具集

软件开发工具集：Kylin + PyCharm + Python3.5

运行硬件平台： Grid 环境。

2.4 条件与限制

该软件系统所受的内部条件约束和限制可能存在如下方面：

（1）软件的训练性能受限于原生 Chatterbot 的训练性能；

（2）软件的 QA 性能受限于原生 Chatterbot 的 QA 性能；

（3）软件的 QA 功能受限于原生 Chatterbot 中内置的字符串比较算法；

（4）每更新一版知识库，就会训练后得到一个新的知识图，而每个知识图对应着 MongoDB 中一个数据库，知识图的总数受限于 MongoDB 所能支持的最大数据库数量；

该软件系统所受的外部条件约束和限制可能存在如下方面：

（1）开发实际进度受限于开发人员对技术方案的理解能力，以及对技术难点的解决能力；

（2）开发环境不同于部署环境，实际运行效果可能受限于实际部署环境和部署方式。

2.5 系统需求分析

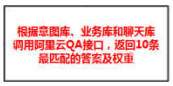
现有上层业务系统存在依赖于阿里云中 QA 接口实现对话系统中核心数据库知识图的产生和 A 的产生。由于上层业务系统的需求希望 QA（问答）服务的调用采用 Restful API 方式，服务内部不保存状态，所以调用和调用之间相互独立。具体需求如下：

2.5.1 详细需求分析

（1）现有小能 Xbot 机器人 QA 流程[1]

角色分类：访客、在线客服、Xbot 和第三方 API。

其中，与阿里云接口[3]（智能语音交互 > 语音对话平台 > 问答服务 > API 使用）发生交互的时机在于：参考资料[1]中红字方框



“根据意图库、业务库和聊天库调用阿里云 QA 接口，返回 10 条最匹配的答案及权重”。这里不一定返回的是 10 条，存在一个范围[1，∞]条。

（2）上层业务“保存并学习”功能

上层业务“保存并学习”功能是直接与阿里云 API 进行交互的。这个功能主要调用了阿里 API 的“创建项目（企业）”、“创建主题（知识库）”、“预发布上线”等 API，如图 1 所示。

从中，可以初步猜测出阿里 API 对知识库的处理基本流程是：

1、获取并保存每个企业的所有知识库信息，初步判断应该保存在数据库中；

2、对每个企业的所有知识库进行训练，训练后的结果应该要保存起来；

3、“预发布上线”功能是指允许开发人员或客服管理人员或在线人工客服，在内部测试机器人，基于动态配置好的知识库向机器人提问，测试返回答案；

4、“正式上线”功能是指第 3 步测试好的更新后训练知识库，先

同步到另一个正式上线知识库中，然后允许访客可以向配置好的机器人提问了。

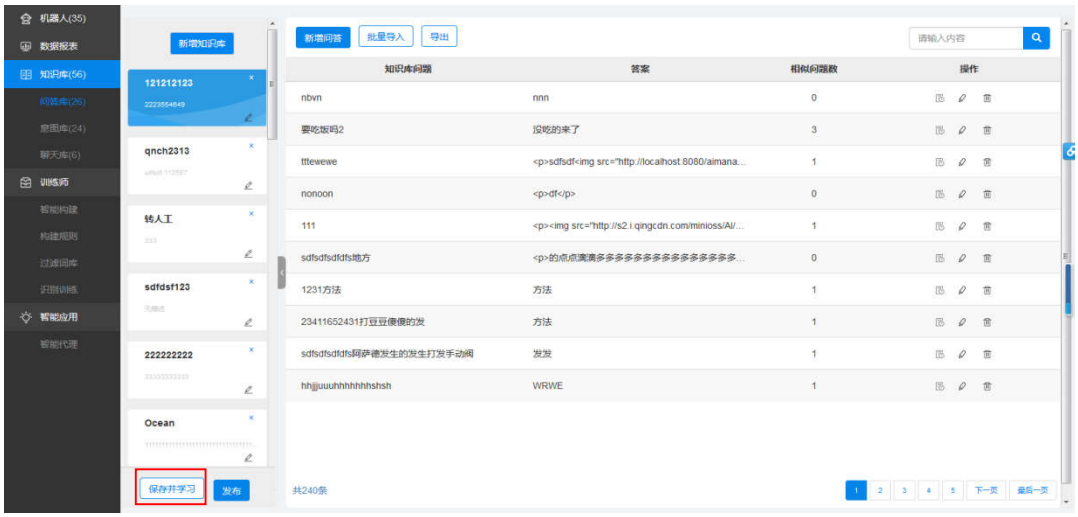


图 1 上层业务“保存并学习”功能界面

（3）上层业务“发布”功能

上层业务“发布”功能是直接与阿里云 API 进行交互的，如图 2 所示。

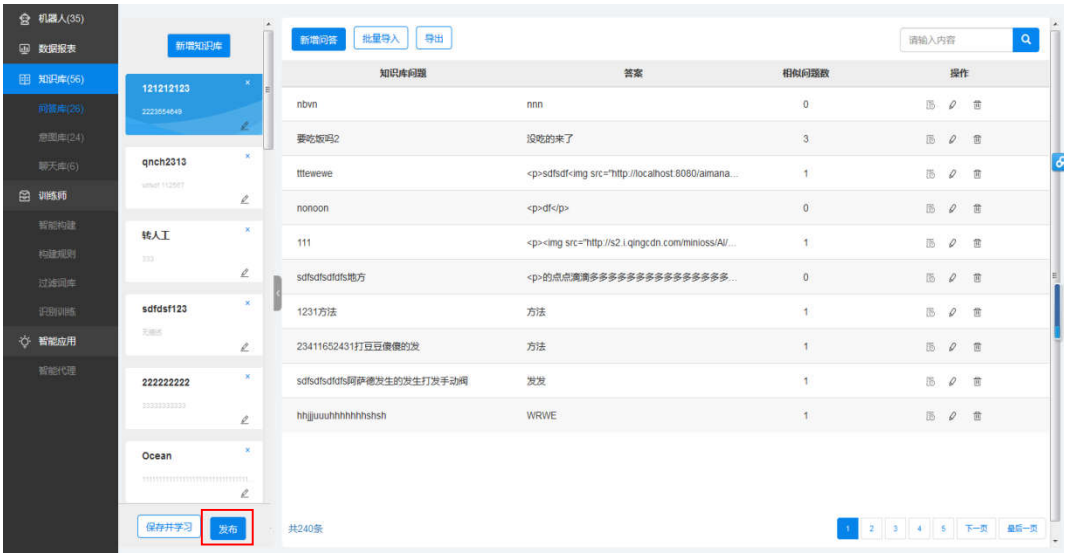


图 2 上层业务“发布”功能界面

（4）上层业务“保存”功能

上层业务“保存”功能是间接与阿里云 API 进行交互的，如图 3 所示。该功能只是在访客与机器人问答时，改变了调用阿里云 API

接口的参数。

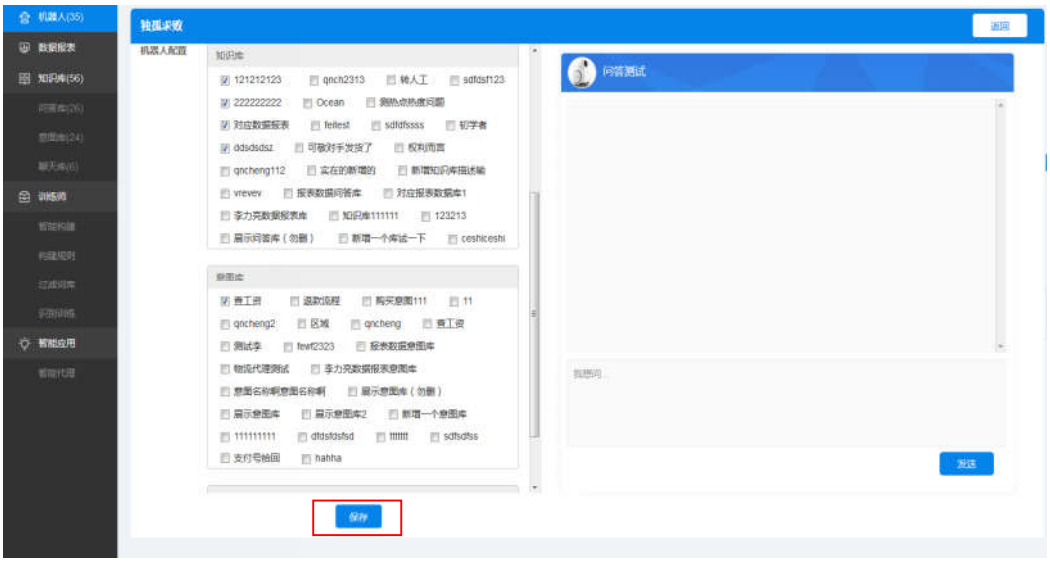


图 3 上层业务“保存”功能界面

（5）上层业务"发送"功能

当调用上层业务“保存”功能，改变了访客与机器人问答时调用阿里云 API 接口的参数之后，如果在线客服人员通过“机器人设置”界面中“问答测试”标签项单击“发送”功能，是可以直接与阿里云 API 进行交互，如图 4 所示。只是，此时机器人里动态配置好的知识库还处于“预发布上线状态”。

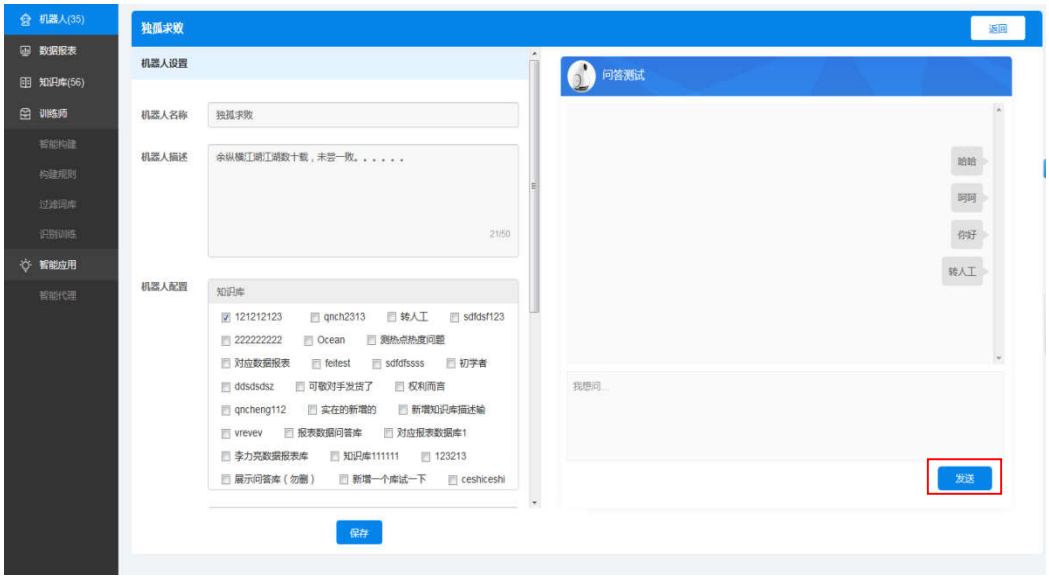


图 4 上层业务“发送”功能界面

2.5.2 详细系统运行环境及限制条件分析接口需求分析

主要包括现有阿里云 API 接口的需求分析。

现有阿里云中智能对话平台，能够让用户在其上面快速搭建自己的对话机器人。目前为 1.0 版，提供单轮的智能问答服务（以下称“智能问答系统”）。该智能问答系统使用了深度学习的技术来寻找语义上匹配的内容。该技术能够识别同一个意思的不同表达方式，包括词形变化、句式变化等。比如当用户问“请问哈里-波特这本书是谁写的”，可以匹配到知识库里“哈利波特的作者是哪一位”这样一条问句，即使这两句话在关键词和句式上都有非常多的不同。

（1） 阿里云接口中知识库需求分析

阿里云中智能问答系统基于一个问答知识库来回答用户提问，该知识库中每一条知识点是以“问题-答案”（问答对）形式组织的。每条知识点包含如下的关键内容：

- ID：每条知识点的唯一标识符。
- Question：知识点的问题部分。这个内容是匹配算法处理的内容。用户的输入会和这个字段内容进行语义匹配。（一个问题只有一个答案）
- Answer：知识点的答案部分。这个内容问答系统不会做处理，会在问题匹配之后原样返回给用户。（一个答案可以有多个问题）

其中，Question 部分是语义匹配的关键部分，书写良好的 Question 对于匹配的精确度有很大好处。一个较好的 Question 应该遵循这样一

些规则：

- 首先表述尽可能完整，能够描述一个特定的问题。比如说“吃了不干净的东西导致拉肚子，用什么药比较好”，要好于“拉肚子怎么办”。
- 尽可能用通用的表述，避免特例，比如说“支付宝转账到银行卡，24 小时了还未到账怎么办”，要好于“昨天上午用支付宝转账到账户为 xxx 的银行卡，现在还没到”。
- 采用比较正规简洁的句式，避免不相关的问候、语气等内容。
- 用词尽可能用常见的词汇，避免孤僻词或者网络用语。

针对知识库上传，阿里云系统提供两种数据上传方式：

- 通过用户界面自动上传，支持全量上传和增量上传，
- 通过后台服务人员手动上传。

（2）阿里云接口用法

阿里云接口用法如图 5 所示：

图 5 阿里云接口用法

（3）阿里云接口 API 分类[4]

- 问答服务 API

问答服务的调用采用 Restful API 方式，服务内部不保存状态，所以调用和调用之间相互独立。调用 API 需要校验权限。

问答服务的输入为目前与用户对话的内容，输出是与输入最匹配的三个知识点，默认第一位最匹配的。

- 知识库管理 API

采用通用接口方式，返回结果为 json 格式，所有接口返回结果格式一致，唯一区别为 data 里面的内容。具体接口分类所图 6 所示。

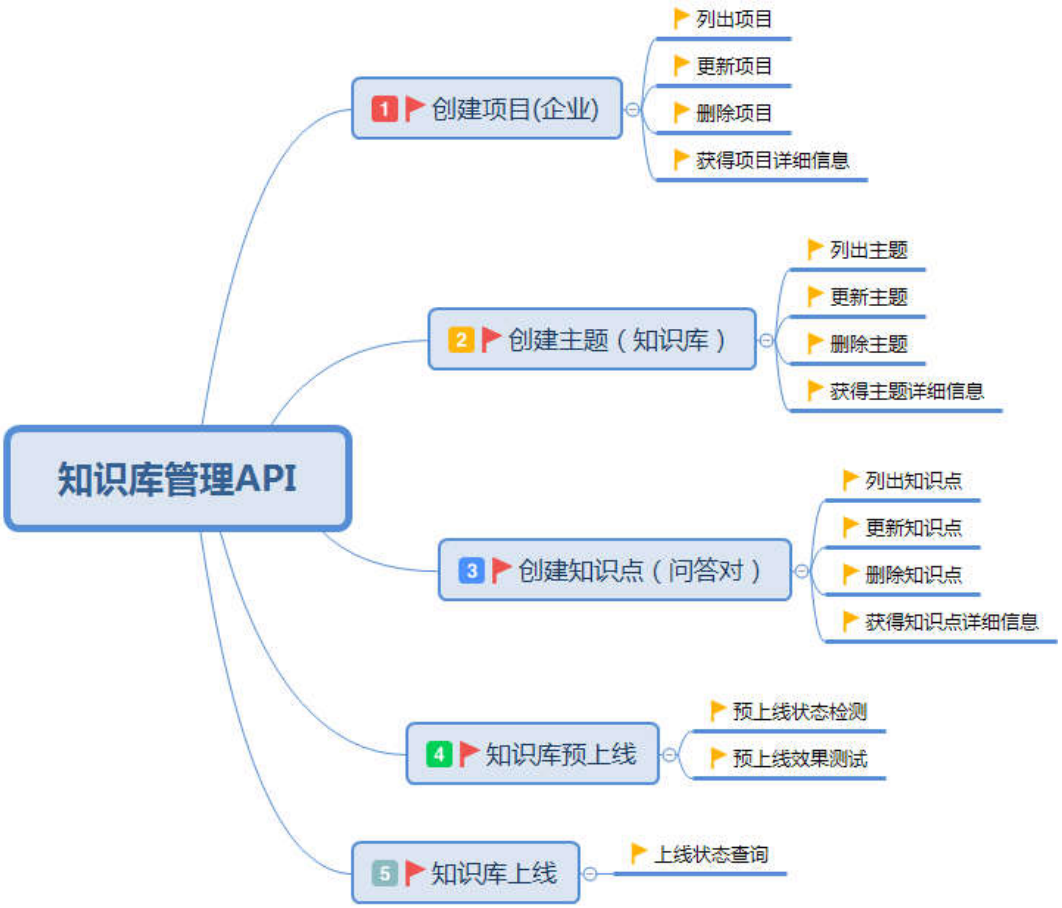


图 6 阿里云知识库管理 API 接口分类示意图

第三章 总体方案

3.1 系统总体结构

具体应用架构图如图 7 所示，底层以下主要分为 6 个组成部分，分别是 Django 架构层、Hub 层、ZooKeeper 架构、Chatterbot Box、MySQL 语料库和 MongoDB 知识图库。其中，Django 架构层主要完成 REST API 接口处理功能；Hub 层主要接收来自 REST API 接口处理程序的请求命令，随后按照不同类型的请求命令，分别调用语料库命令处理模块、训练命令处理模块和 QA 命令处理模块来完成各自不同的功能；语料库命令处理模块主要负责创建和维护知识库、过滤词库、专有名词库和同义词库；训练命令处理模块主要采用 ZooKeeper 分布式消息架构，将训练任务发布到 MySQL 数据库的“训练任务表”中，而底层 Chatterbot Box 从监听的语料库中“训练任务表”中获取训练任务后，实例化多个训练 Chatterbot 对象，将知识库中“问答对”信息经过预处理操作后，训练成相应版本的知识图；QA 命令处理模块主要负责将 QA 命令同步发送到底层 Chatterbot Box 中的 REST API 接口，当底层 REST API 接口通过 Chatterbot+组件实例化多个 QA Chatterbot 对象后，经过预处理和 QA 操作后，将答案返回给 Hub 层；Chatterbot Box 是通过 ZooKeeper 来实现 Hub 层、虚拟机（VM）之间关于训练任务、KB 资源和 QA 任务的一致性。

此外，KB 是指知识库，用于保存“问答对”信息；FT 是指过滤词库，用于保存“过滤词”信息；PN 是指专有名词库，用于保存“专有名

词”信息；KB 是指同义词库，用于保存“同义词”信息；MongoDB 是指 Mongo 数据库，用于保存语料库中问答对训练后得到的知识图；KG 是指知识图，一个知识库 KB_i 经过训练对应着唯一一个知识图 KG_i，i 从 $[1, n]$ 中取值；每个知识库会不断更新，每次更新后都形成新的版本，假设有 n 个新的版本，则知识图库中会有 n 个知识图 KG_{in}。另外，底层 Chatterbot Box 可以按照角色不同，分别实例化为两种角色 Chatterbot+对象若干个，即 A 类训练 Chatterbot+和 B 类 QA Chatterbot+；A 类 Box 可以只负责提供训练能力；B 类 Box 可以只负责提供 QA 服务能力；每个 B 类 Box 都是单例模式运行的。

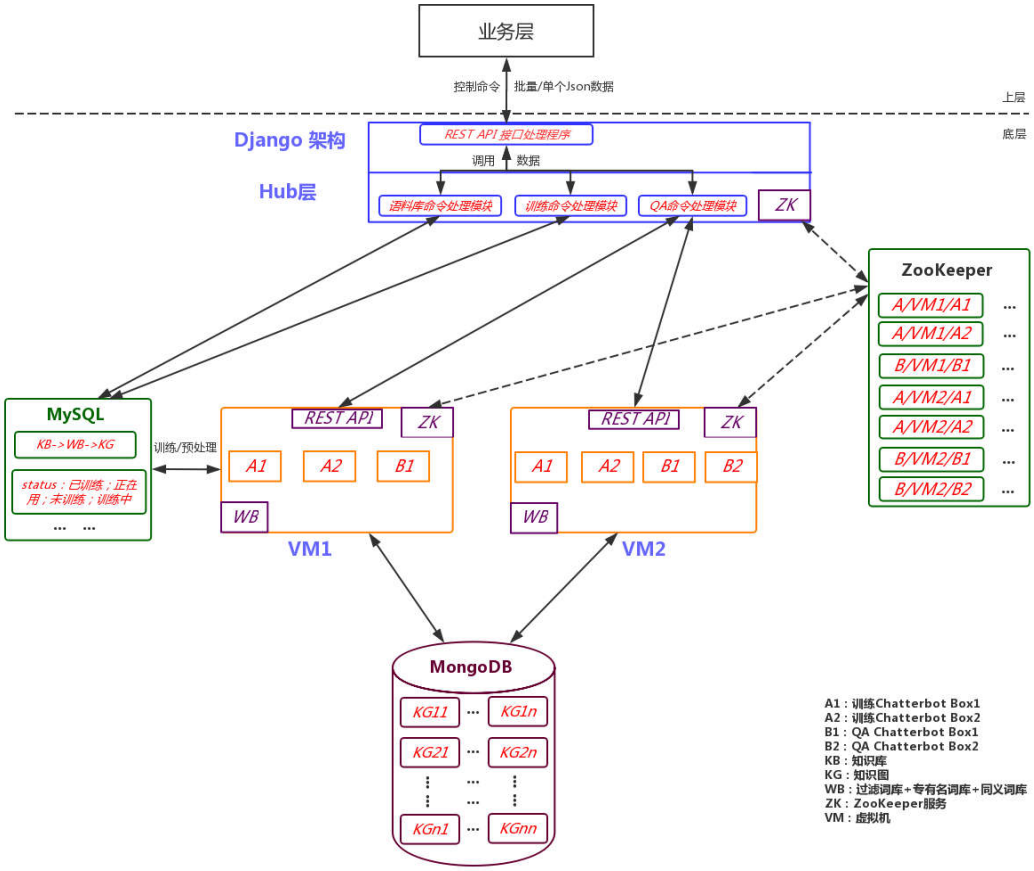


图 7　替代阿里云接口之 Chatterbot+技术开发架构图

3.2 系统工作流程

鉴于系统在今后的使用、维护和升级上的方便，在程序架构上，拟采用 REST 模式的基于网络的 Web API 应用架构。服务器采用稳定、高效的 LINUX 操作系统作为平台，使用 MySQL 数据库保存 Web API 传入的语料库，使用 MongoDB 作为后台数据库保存知识图，使用 ZooKeeper 提供 VM（虚拟机）分布式环境下一致性保证，使用 Chatterbot+实现底层组件业务逻辑，辅以 Django 架构、ZooKeeper 架构以及 4 种 python 命令处理程序实现对业务层的 API 接口、底层语料库、底层 Chatterbot+组件和底层知识图库的相关操作。

3.2.1 ZeeKeeper 节点信息表

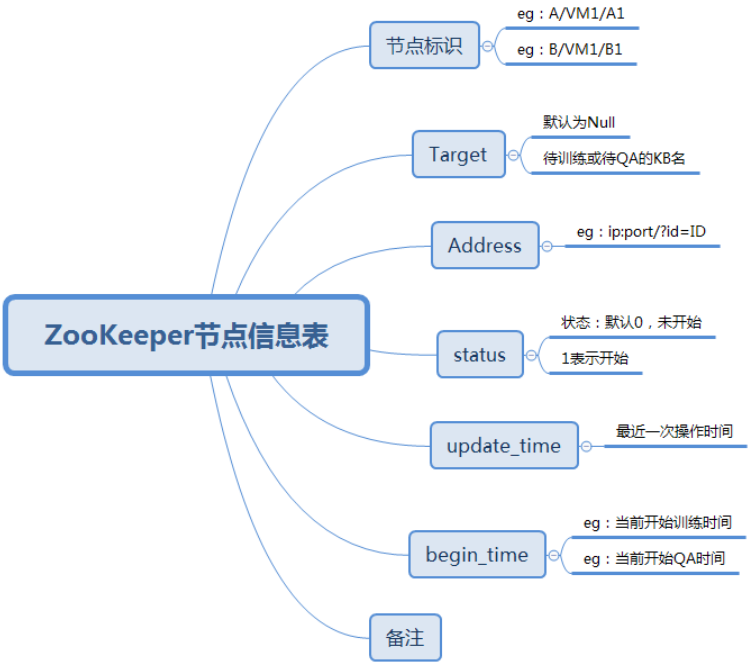


图 8 ZK 节点信息表结构

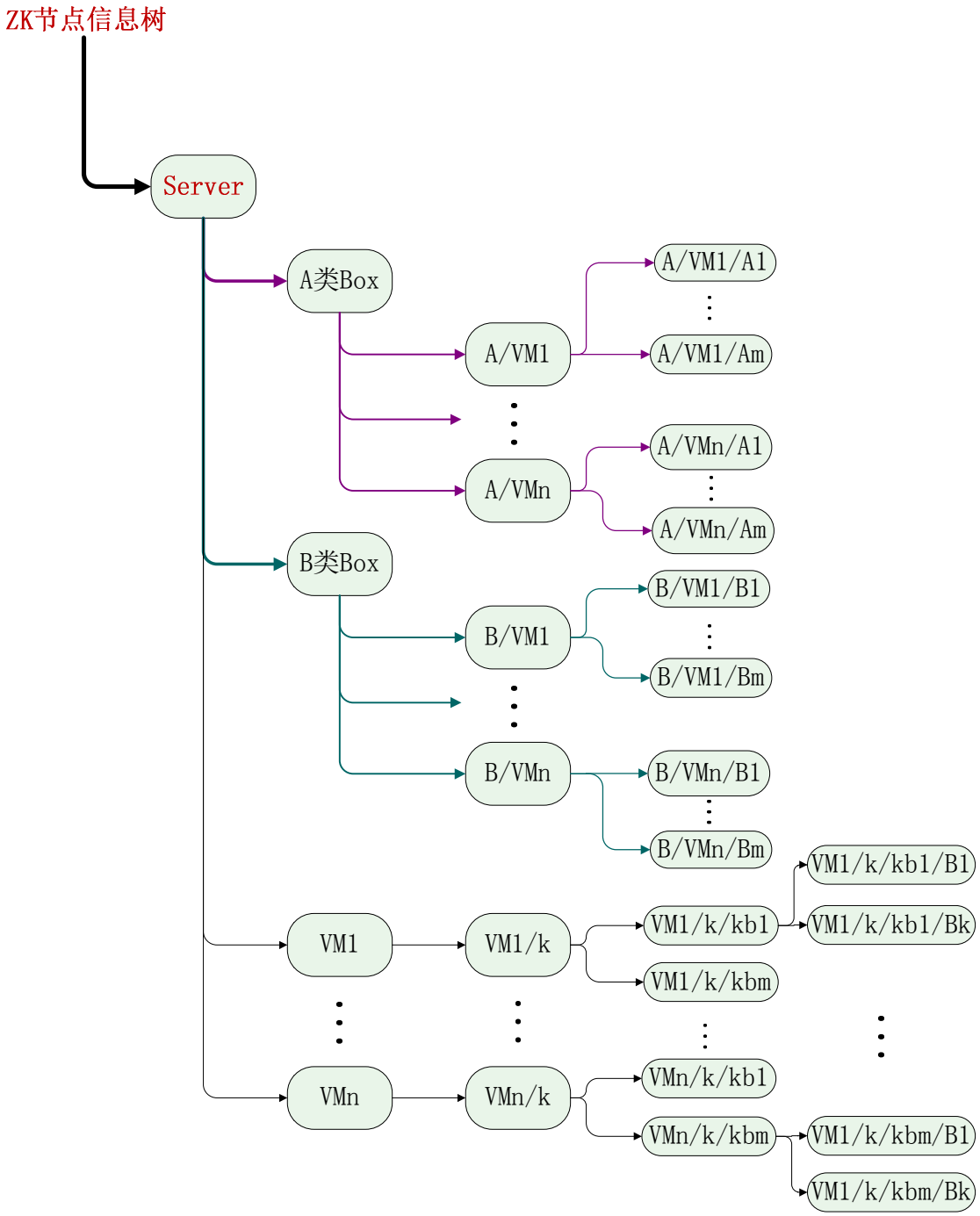


图 ZK 节点信息树结构

3.2.2 “训练任务”表

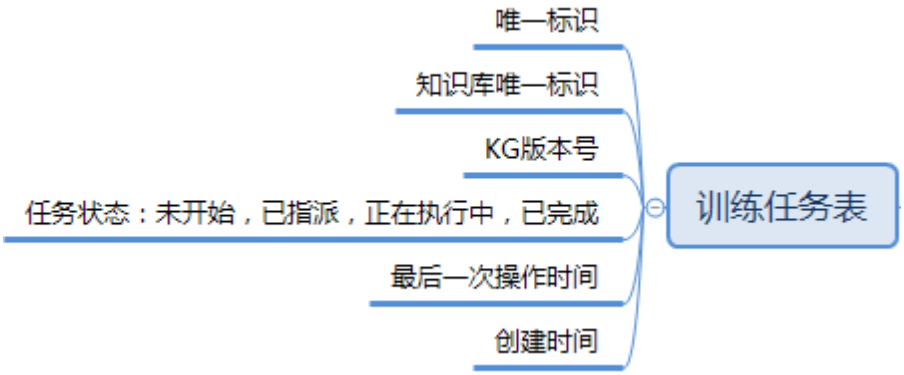


图 9“训练任务”表结构

3.2.3 VM 启动

①加载配置文件，配置参数如下：

m：A 类 Chatterbot Box 的数量；
n：B 类 Chatterbot Box 的数量；
k：每个 KB 最大对应的 B 类 Chatterbot Box 数量；
ip：VM 所在的 IP 地址；
port：端口号；
ZK 的 ip 地址和端口号；
MySQL 数据库的 ip 地址和端口号；
MySQL 数据库的用户名和密码；
MongoDB 数据库的 ip 地址和端口号；
MongoDB 数据库的用户名和密码；
回收时长：QA CBot 对象未被调用需要释放前经历的时长；
其他；

②创建（m+n）个 A 类和 B 类 Chatterbot Box，并初始化；

③在 ZooKeeper 服务器上生成（m+n）条节点信息，节点信息表示如下：

A/VM1/A1 [Target=Null, Address=ip:port/id=ID, status=0, update_time=当前开始训练时间, ……]
B/VM1/B1 [Target=Null, Address=ip:port/id=ID, status=0, update_time=当前开始 QA 时间, ……]
……
……

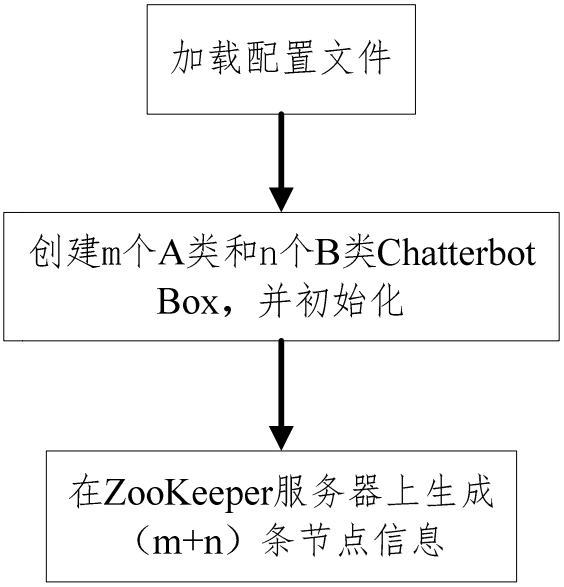


图 10　VM 启动流程

3.2.4 Hub 层初始化

①Hub 层监控 ZooKeeper 服务器中 A 类和 B 类 Chatterbot Box 记录数据变化情况；

②Hub 层等待接收 Django 层发送来的上层命令，如 KB 维护、训练命令和 QA 命令，一旦收到相应的命令后，则调用相关的模块处理。

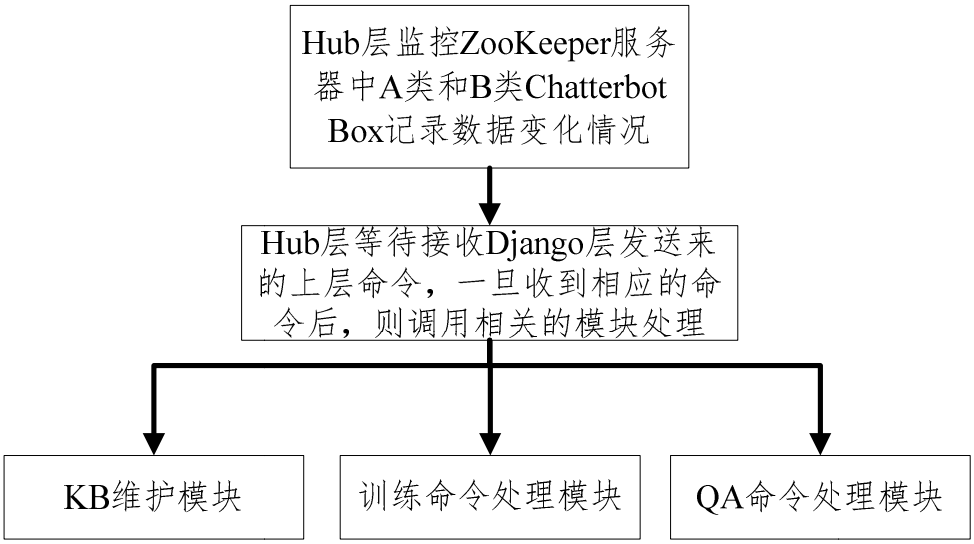


图 11 Hub 层初始化流程图

3.2.5 Hub 层 KB 维护

①Hub 层根据接收到的命令参数，对 MySQL 数据库进行维护，不仅包括知识库维护，还包括过滤词库、专有名词库和同义词库，以及训练任务表、KG 索引表等；

②Hub 层将维护后的结果返回给上层业务。

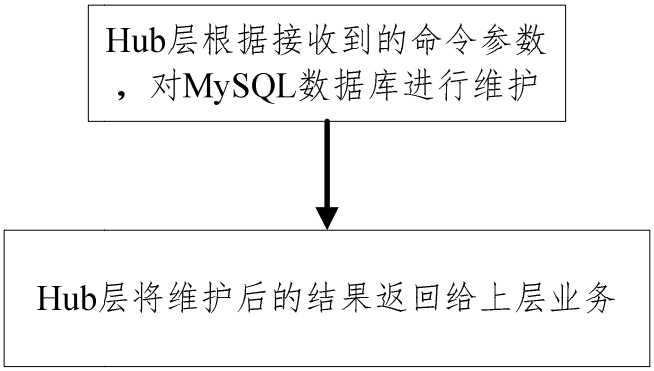


图 12 Hub 层初始化流程图

3.2.6 Hub 层训练功能

①当 Hub 层接收到来自上层业务的训练命令后，更新 MySQL 数据库中“训练任务表”，增加一项训练任务记录；完成后向上层业务返回消息；若某知识库版本数量超过 10 条，则保留最新的 10 条，当 MySQL 数据库中“训练任务表”每收到一个新增训练后，“KG 索引表”中对应的知识库就会删除一条创建时间最早的版本记录，及其在 MongoDB 数据库中对应的 KG 数据库。以上工作由 Hub 层来控制。

②由于每个 VM 中 A 类 Chatterbot Box（标记为 Ai）都始终监控 ZooKeeper 服务器，每隔一定时间（例如 10 秒）从 ZK 中选取具有最小 id 的空闲 A 类 Chatterbot Box（标记为 MinA），随后判断 Ai 是否就是 MinA，如果是，则进入下一步；否则，一直循环监控；

③当 VM 中任意一个 A 类 Chatterbot Box（标记为 Ai）就是 MinA 时，Ai 便开始启动一个定时周期任务，可以设置周期为 10 秒；

④该定时周期任务每隔 10 秒，从 MySQL 数据库的“训练任务表”中查找“未训练”的任务；一旦找到这样的任务（不妨标记为 KB1），则进入下一步； 否则，循环执行，直到所有的“未训练”的任务都被找到为止，最后才退出循环；

⑤再次在 ZooKeeper 中查找当前 A 类 Chatterbot Box 中具有最大 id 的空闲 A 类 Chatterbot Box（标记为 A’），根据 MinA 找到的“未训练”任务 KB1 来修改 ZK 中对应的记录，例如：A/VM1/A’[Target=KB1, Address=ip:port/id=ID, status=1,训练时间=当前开始训练时间,……]，返回步骤④；

⑥由于 VM 中每个 A 类 Chatterbot Box A’都一直不停地监听 ZK 服务器记录的变化，一旦 ZK 有变化，就触发 A’的训练功能，进入下一步。

⑦A’将 MySQL 数据库的“训练任务表”中对应的训练 KB1 的任务项状态改为“训练中”，进入下一步；

⑧A’开始执行实例化，完成对应的训练任务，详见 3.2.6 小节；

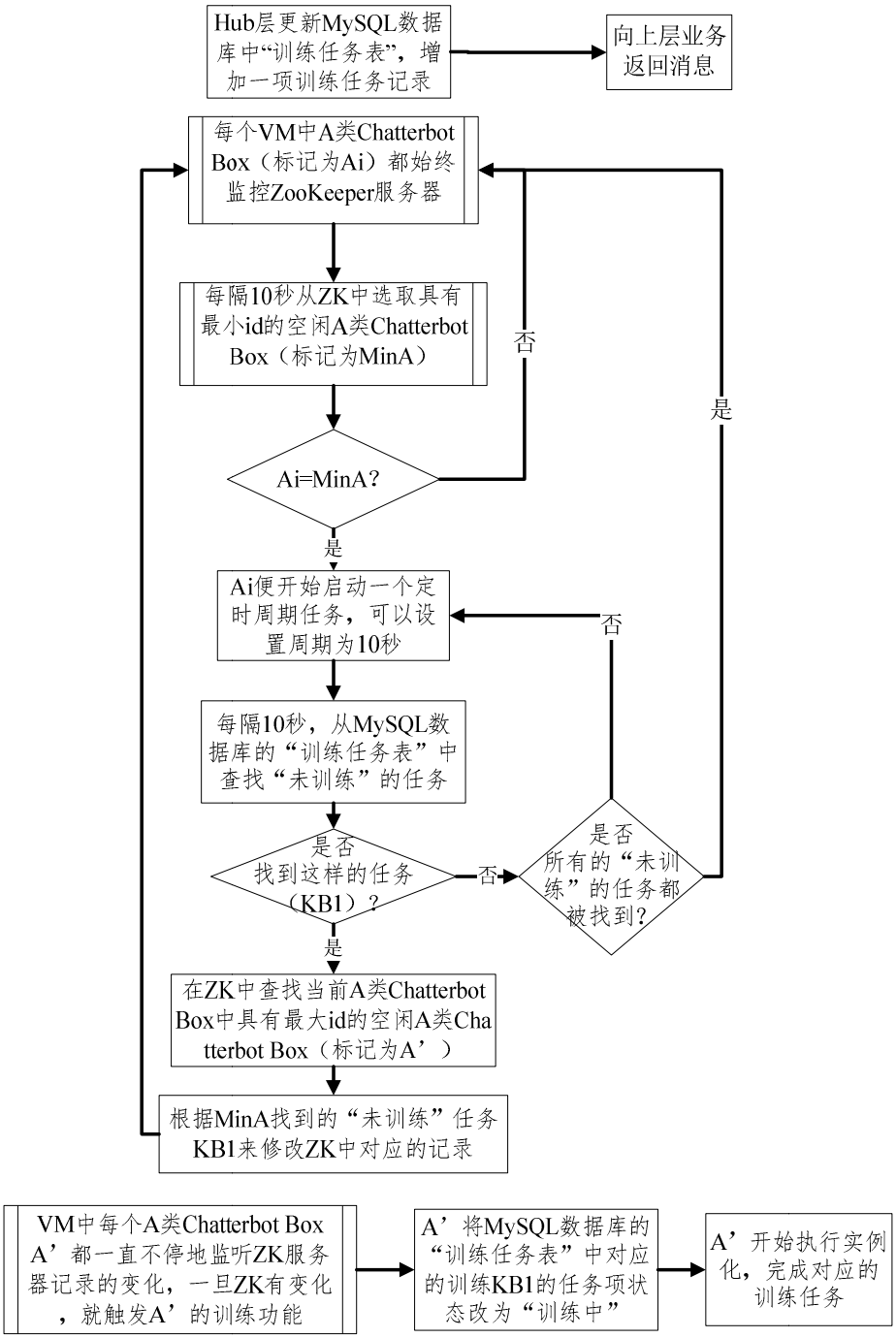


图 13 Hub 层训练功能流程图

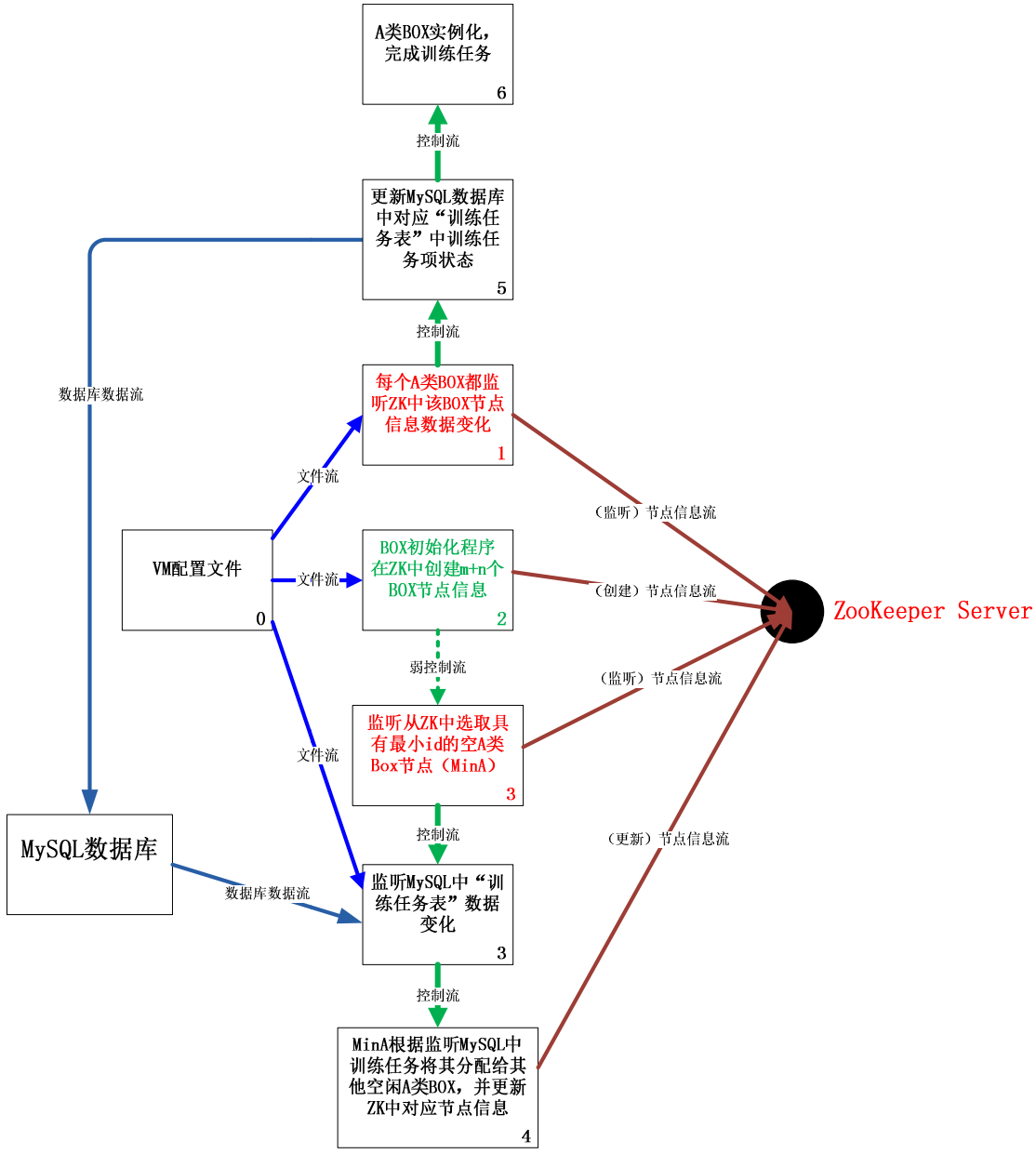


图 Hub 层训练功能 IEDF0 图

监听器：每个A类BOX都监听ZK中该BOX节点信息数据变化

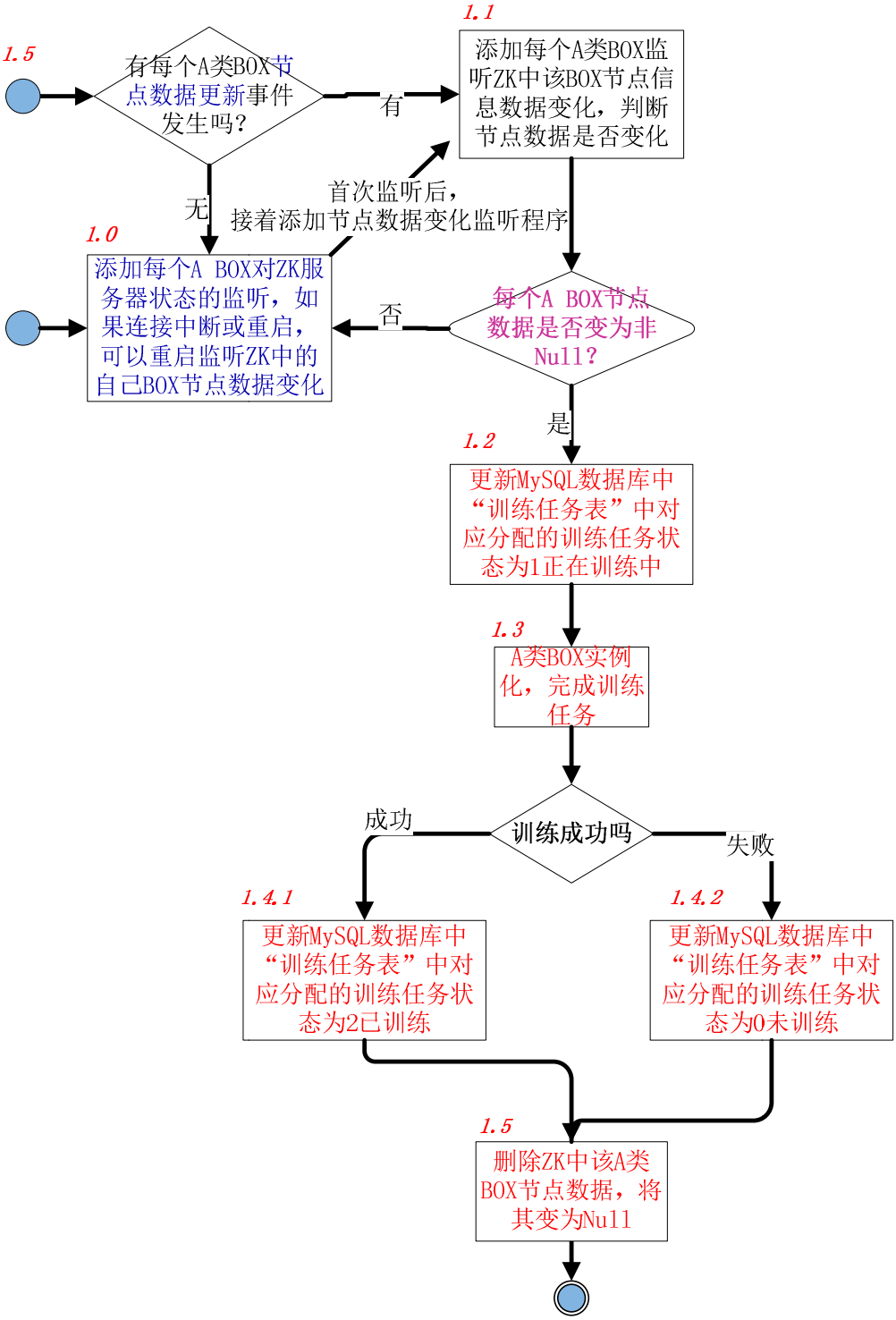


图 A 类 BOX 节点变化监听器功能流程图

监听器：选取最小id和空闲Abox的MinA

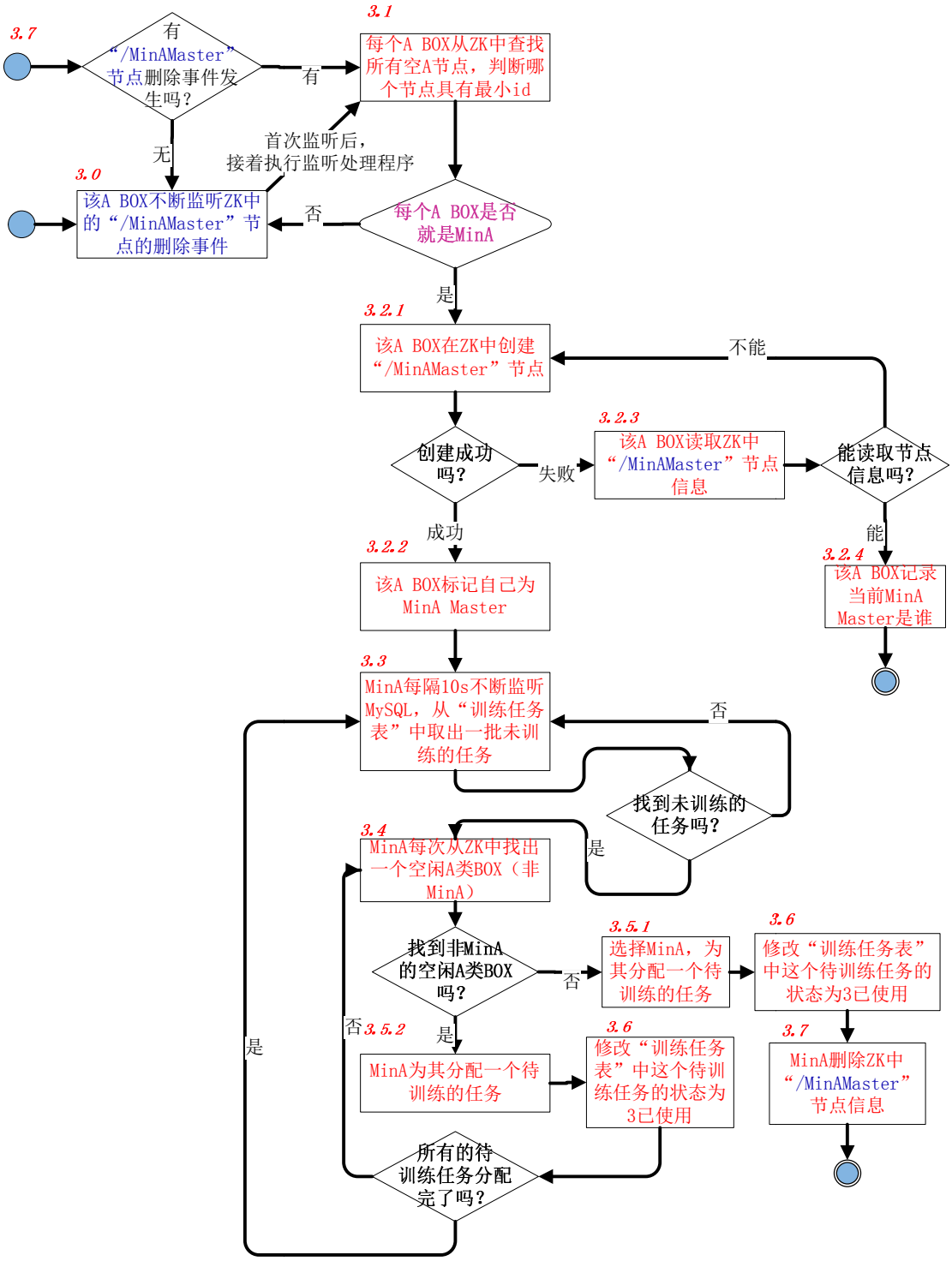


图 选取 MinA 的监听器功能流程图

3.2.7 Hub 层 QA 命令

①Hub 层对 ZooKeeper 进行监控，了解到 ZK 中当前 B 类 Chatterbot Box 记录的变化情况；

②Hub 层收到 QA 命令、Q 参数和 KB 参数（不妨设为 KB2）后，首先查看当前 ZK 中具有 B 类 Chatterbot Box 的 VM 信息，从/B/VM1 到/B/VMn 中依次提取出 VM 名称，然后查看对应 VM 中知识库节点信息，例如从/VM1/k（这个节点是由底层初始化时创建的，k 表示同一个知识库在同一个 VM 下所有 B 类 Box 中至多出现的次数）中判断/VM1/k/KB2 节点是否存在，如果不存在，则创建该节点，进入下一步；否则，直接查看/VM1/k/KB2 节点下子节点的个数是否超过 k 个，如果没有，则进入下一步，选择一个空闲的、且未分配知识库的 B 类 Box；否则从 k 个子节点{/VM1/k/KB2/B1、…、/VM1/k/KB2/Bk}中依次选择当前空闲的 B 类 Box，如果可以从 k 个中找到空闲 Box，则进入④，否则处于循环等待中；

③这一步目的是从/B/VM1/B1 到/B/VM1/Bm 中选择一个空闲的、且未分配知识库的 B 类 Box（标记为 Bi），如果当前找不到这样的 B 类 Box，则说明所有 Box 都处于忙状态，此时可以返回上一步循环等待中；

④然后通过 MySQL 数据库判断 KB2 是否被训练过，如果没有被训练过，则返回上层业务 QA 错误；否则，进入下一步；

⑤一旦确定 KB2 被训练过，就在/VM1/k/KB2 节点下创建子节点/VM1/k/KB2/Bi；同时，更新 ZK 中 B/VM1/Bi [Target=KB2,

Add=ip:port/（id）, status=1,update_time=当前开始QA时间]节点信息，然后，按照其中直接得到 Bi 的 API 地址（http://ip:port/id），将 QA 命令及 KB 参数、版本号等发送给对应的 REST API 接口，REST API 接口接收命令后将其转发给 Bi 来完成 QA 命令，进入步骤⑦；否则，进入下一步；

⑥当 Hub 层收到某个 B 类 Chatterbot Box 返回的实例化失败信息后，返回步骤②；

⑦根据 QA 命令信息，对 Bi 进行实例化，完成对应的 QA 任务，详见 3.2.7 小节；

⑧当 Hub 层收到所有 B 类 Chatterbot Box 返回的 A 结果后，进行 A 合并操作，再返回上层业务。

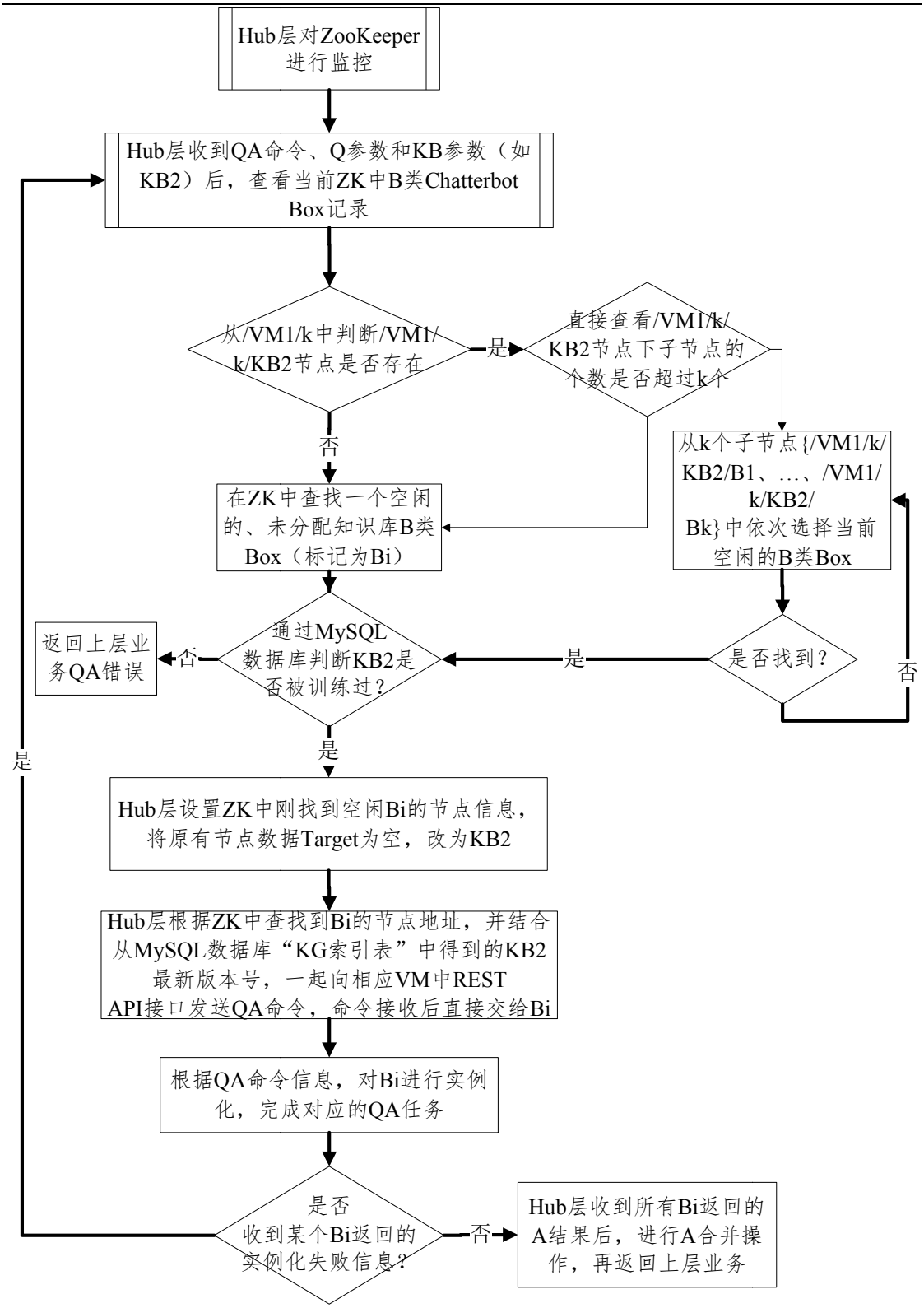


图 14 Hub 层初始化流程图

3.2.8 A-Box 实例化

①根据 KBi 参数，初始化（或实例化）训练 Chatterbot 对象（训

练 CBot），即 Init（KBi）；

②执行训练 CBot 的训练任务；

③如果训练成功，则将 MySQL 数据库的“训练任务表”中对应的训练 KBi 的任务项状态改为“已训练”，否则将 MySQL 数据库的“训练任务表”中对应的训练 KBi 的任务项状态改为“未训练”；进入下一步；

④更改 ZK 中对应的记录信息，例如：A/VM1/A’ [Target=Null, Address=ip:port/id=ID, status=0,……]

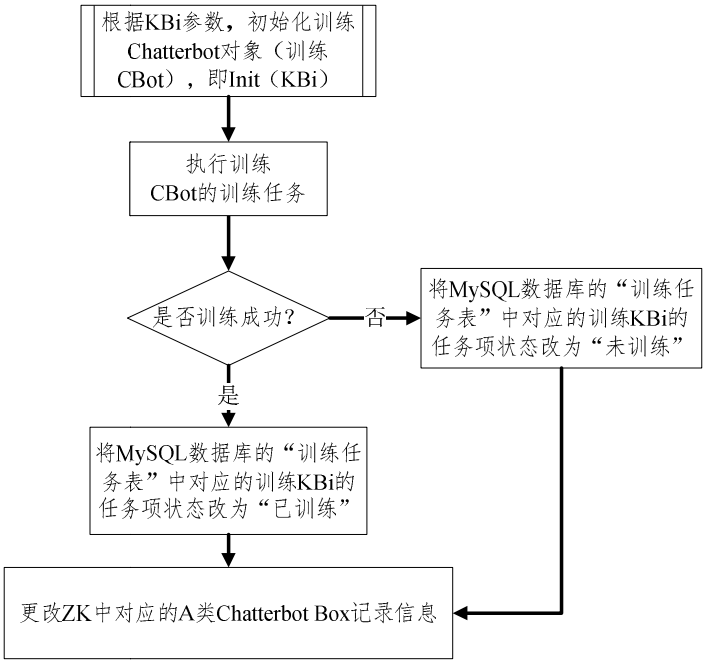


图 15 A-Box 实例化流程图

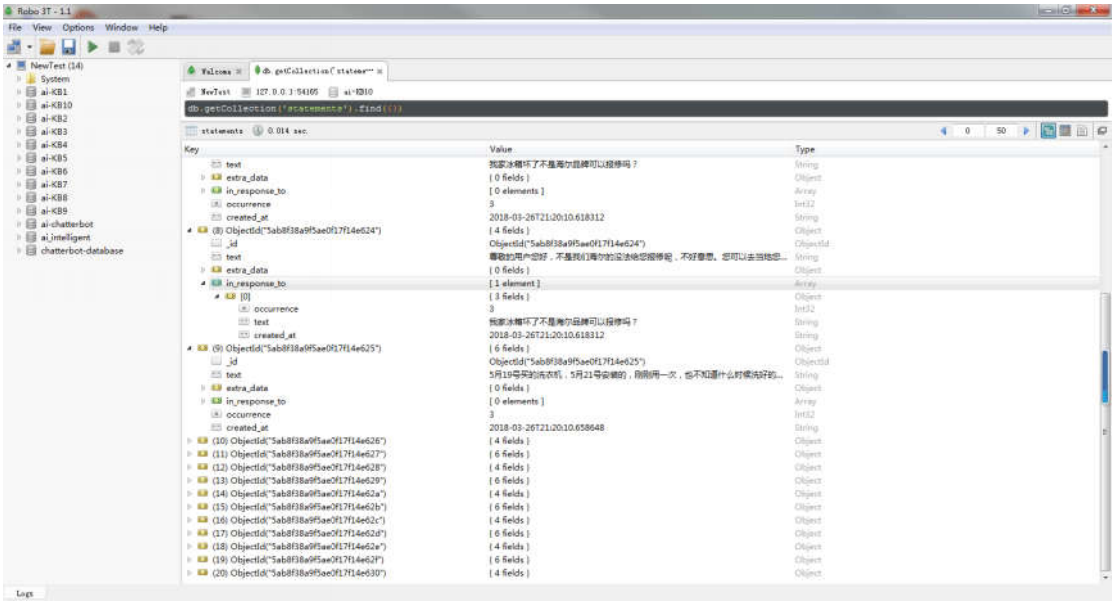


图 A-Box 完成训练任务后 MongoDB 知识图

3.2.9 B-Box 实例化

①根据 KBi 参数，初始化（或实例化）QA Chatterbot 对象（QA CBot），即 Init（KBi）；

②如果 Init（KBi）成功，则进入下一步；否则返回 Hub 层，告知 QA 初始化失败；

③执行该 QA CBot 的 QA 任务；

④如果该 QA CBot 对象太长时间没有被调用，例如超过 30 分钟，那么释放该 QA CBot 对象，同时更改 ZK 中对应的记录信息，例如：B/VM1/Bi [Target=KB2, Address=ip:port/id, status=0, update_time=当前开始 QA 时间,……]

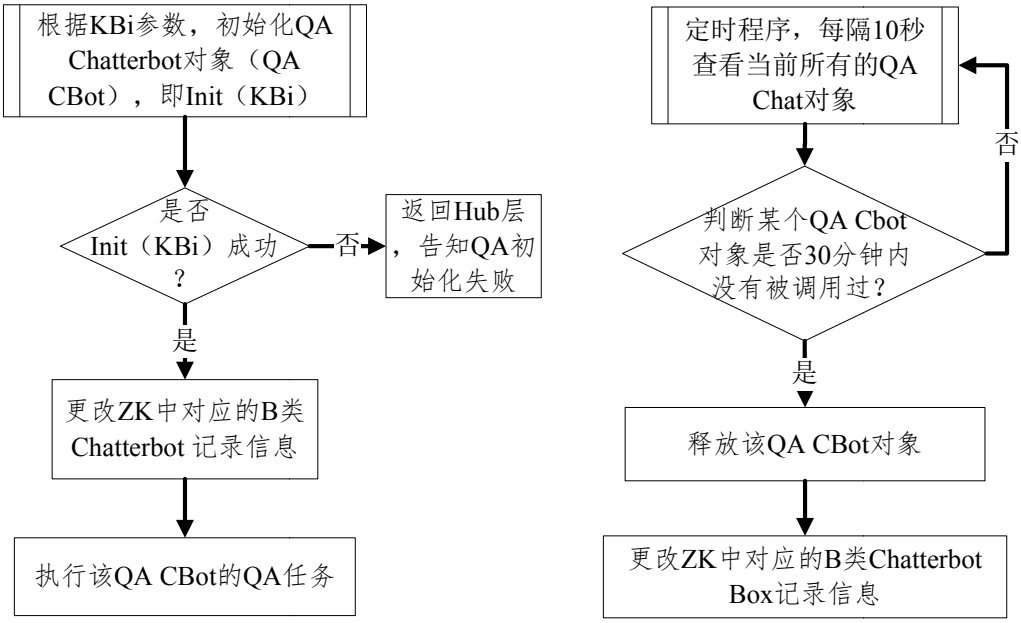


图 16 B-Box 实例化流程图

另外，专门编写一个定时程序，其工作流程如下：

①每隔 10 秒查看 ZK 中/B/VM1 节点下所有子节点{/B/VM1/B1、…、/B/VM1/Bm}的数据信息，找出所有 Target 不为空的子节点，再根据系统当前时间 current-time 判断这些子节点的 update-time 是否超过了半个小时，即（current-time－update-time＞1800）；

②如果超过 1800 秒，就说明该子节点（例如/B/VM1/Bj）在半个小时没有被调用过，此时首先取出该子节点中 Target 字段值（例如 KB3），然后更改/B/VM1/Bj 子节点数据为 B/VM1/Bj [Target=Null, Address=ip:port/id, status=0, update_time=current-time]，同时删除/VM1/k/KB3/Bj 子节点同时。

3.3 关键数据结构设计

系统中关键的数据结构设计包括语料库中 8 种表的设计，具体如图 17 所示。

图 17 语料库表设计

第四章 系统核心模块详细设计

4.1 REST API 接口处理子系统

4.1.1 系统结构设计及子模块划分

图 18 表示 REST API 接口处理子系统的结构设计图。

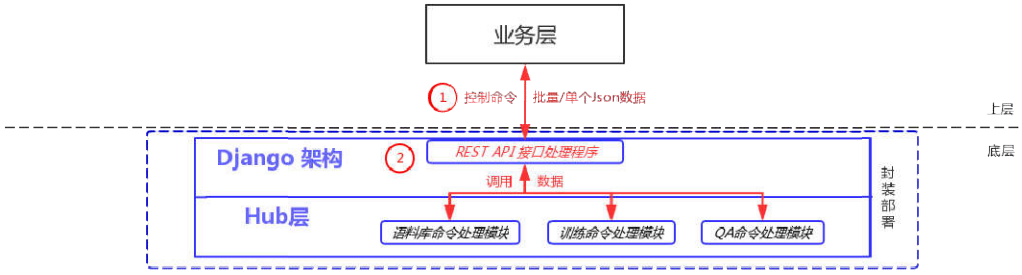


图 18 REST API 接口处理子系统结构设计图

4.1.2 系统功能模块详细设计

图 19 给出 REST API 接口处理子系统的功能流程图。

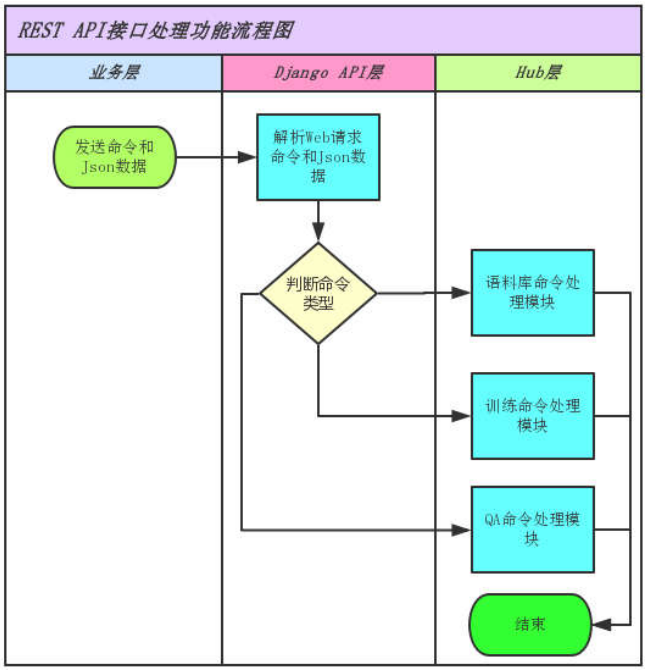


图 19 REST API 接口处理子系统功能流程图

4.2 语料库命令处理子系统

4.2.1 系统结构设计及子模块划分

图 20 表示语料库命令处理子系统的结构设计图。

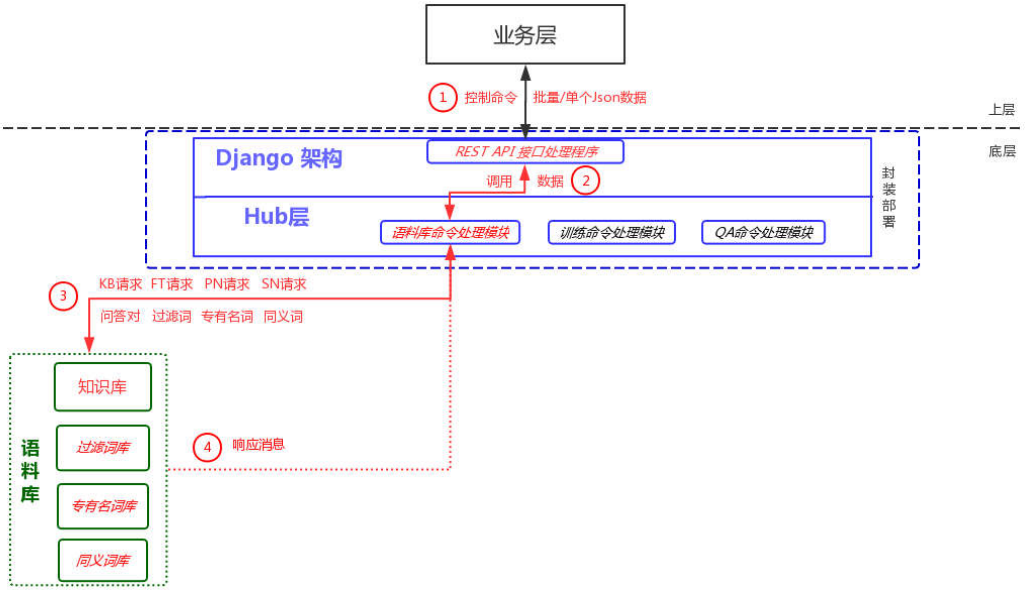


图 20 语料库命令处理子系统结构设计图

4.2.2 系统功能模块详细设计

图 21 给出语料库命令处理子系统的功能流程图。

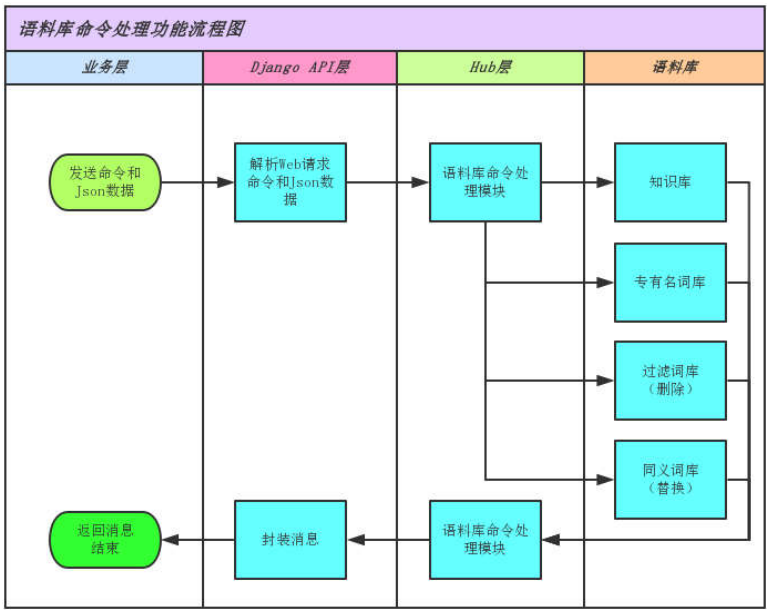


图 21 语料库命令处理子系统功能流程图

4.3 训练命令处理子系统

4.3.1 系统结构设计及子模块划分

图 22 表示训练命令处理子系统的结构设计图。

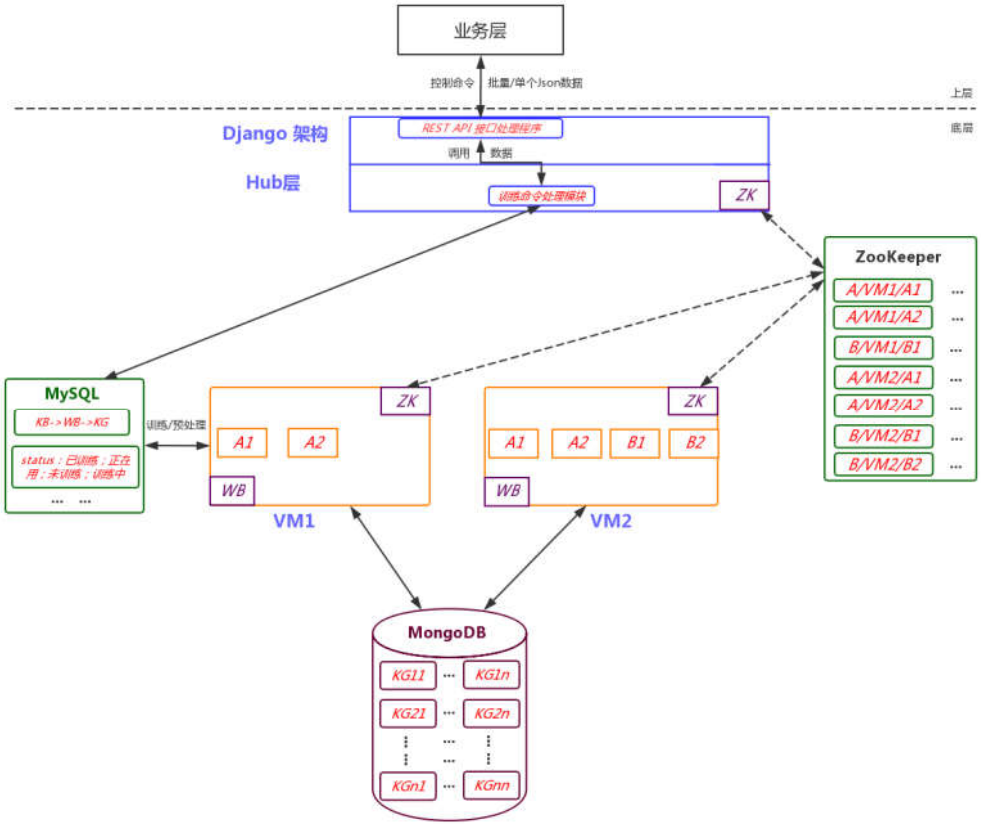


图 22 训练命令处理子系统结构设计图

4.3.2 系统功能模块详细设计

图 23 给出训练命令处理子系统的功能流程图。

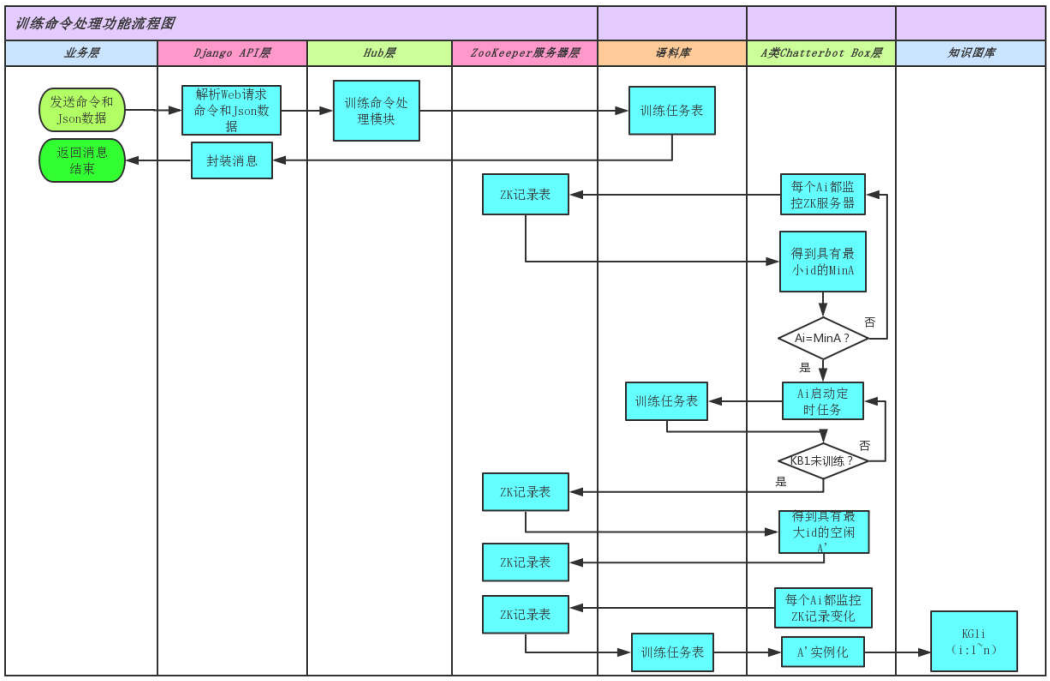


图 23 训练命令处理子系统功能流程图

4.4 QA 命令处理子系统

4.4.1 系统结构设计及子模块划分

图 24 表示 QA 命令处理子系统的结构设计图。

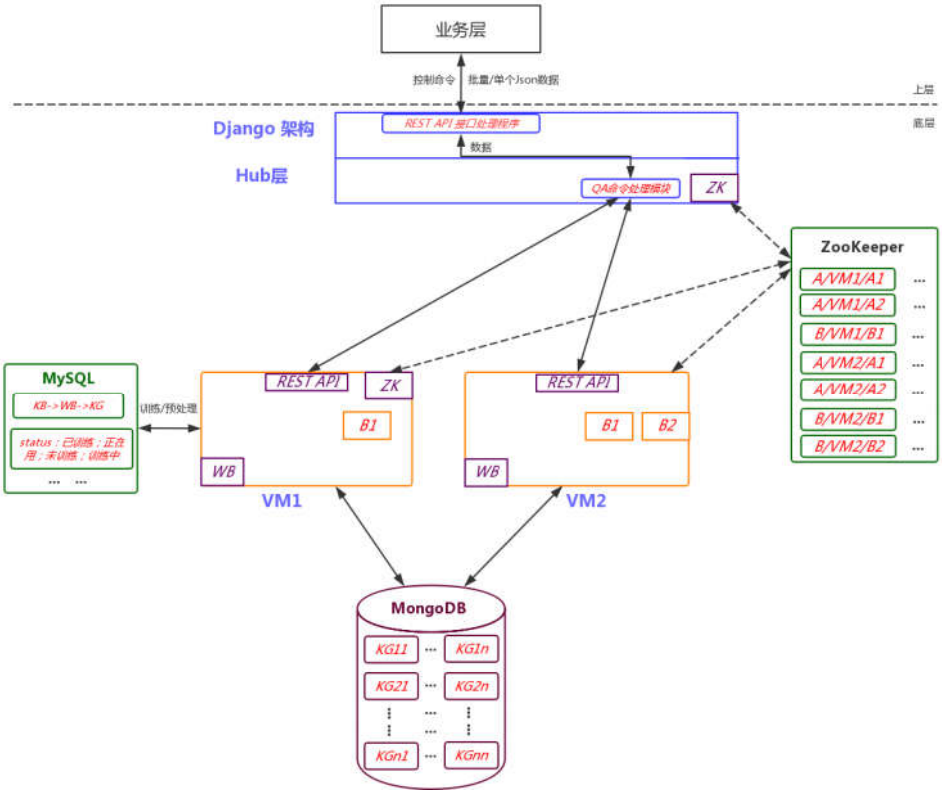


图 24　QA 命令处理子系统结构设计图

4.4.2 系统功能模块详细设计

图 25 给出 QA 命令处理子系统的功能流程图。

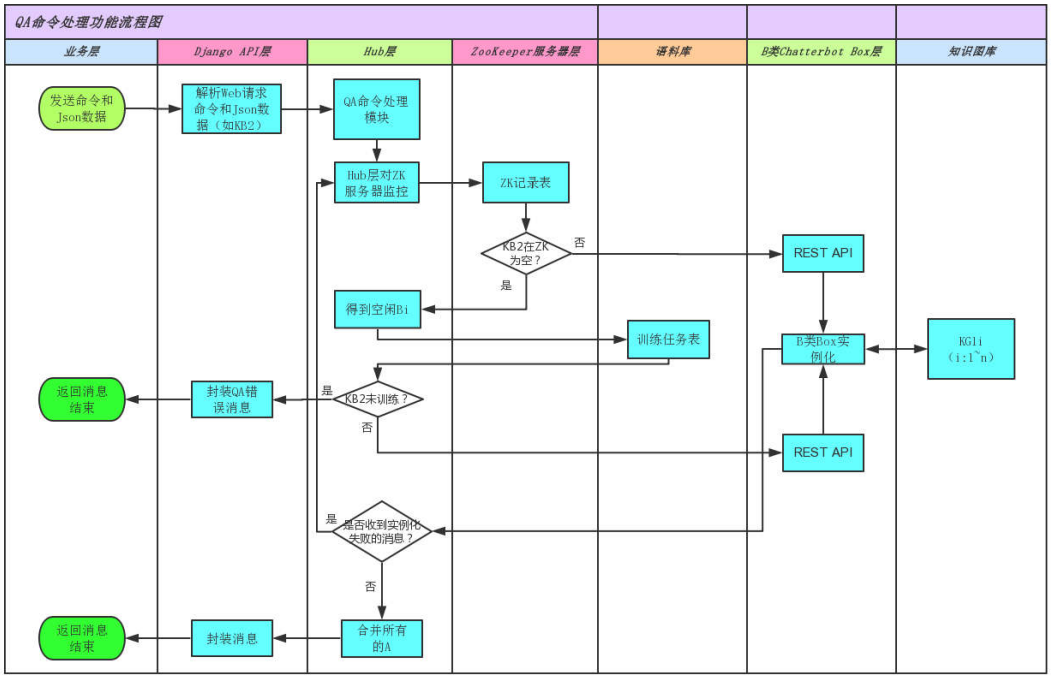


图 25　QA 命令处理子系统功能流程图

4.5 A 类和 B 类 Chatterbot Box 子系统

4.5.1 A 类 Chatterbot Box 子系统训练功能

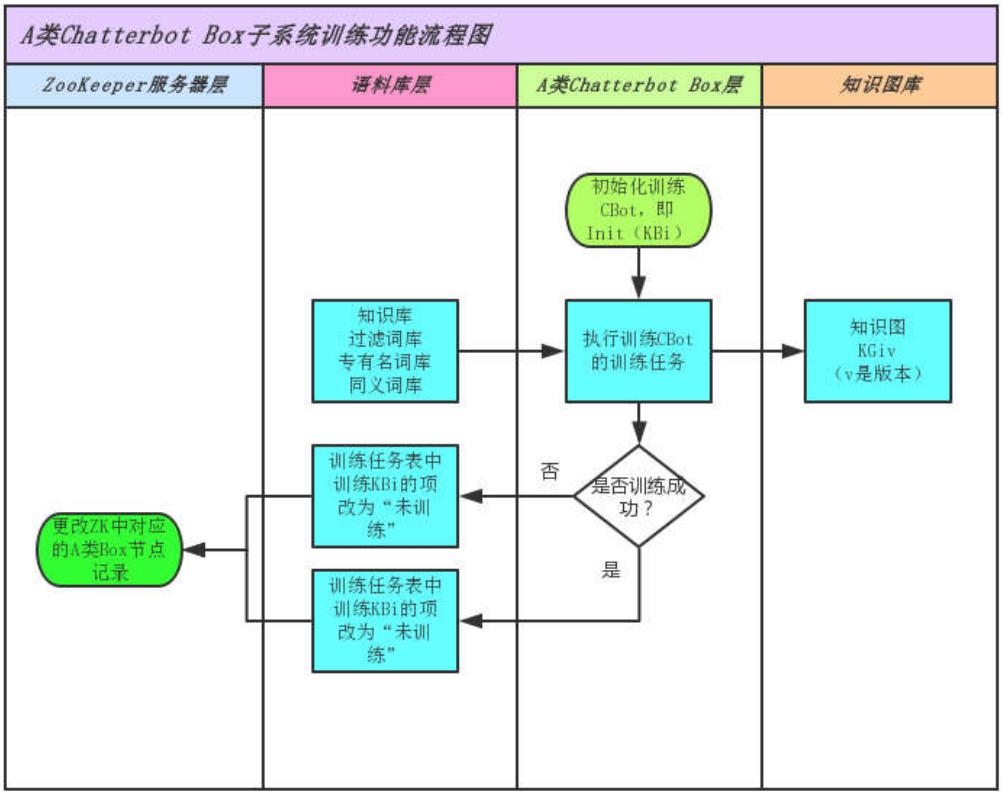


图 26　A 类 Chatterbot Box 子系统训练功能流程图

4.5.2 B 类 Chatterbot Box 子系统 QA 功能

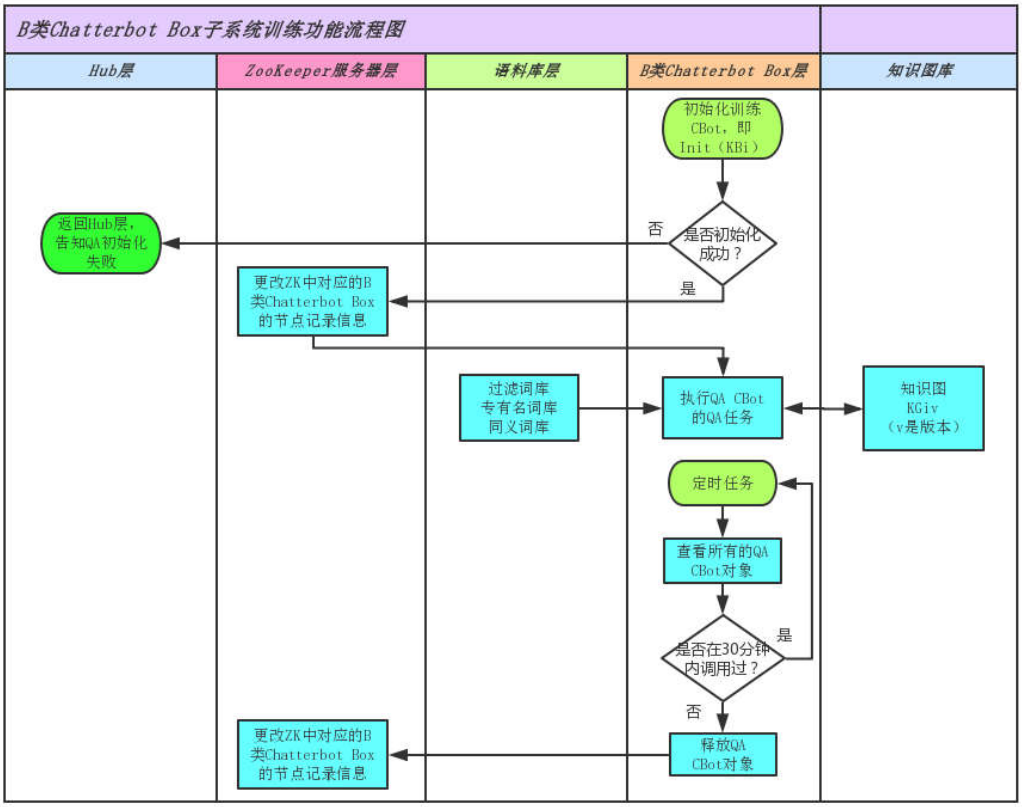


图 27　B 类 Chatterbot Box 子系统训练功能流程图

第五章 系统接口设计

5.1 外部 API 接口设计

结合小能上层业务系统的实际需求，替代阿里云所需的 API 接口计划开发 12 个，涉及到语料库管理功能、训练命令处理功能和 QA 命令处理功能这 3 种类型。

5.1.1 语料库管理功能类

主要实现包括 6 个 API 接口，如下：

（1） “创建知识库”API 接口

接口类型

请求方法	post
访问 URI	172.23.1.156/api/<version>/knowbase/
通信协议	https 或 http

请求参数

参数名称	是否必须	参数类型	默认值	描述
id	是	String	NA	知识库标识(整站唯一)
name	否	String	NA	知识库描述

响应参数

参数名称	是否必须	参数类型	默认值	描述
code	是	String	NA	返回状态码
msg	是	String	NA	返回消息内容

（2） “删除知识库”API 接口

接口类型

请求方法	delete
访问 URI	172.23.1.156/api/<version>/knowbase/<kbid>/
通信协议	https 或 http

请求参数

空

响应参数

参数名称	是否必须	参数类型	默认值	描述
code	是	String	NA	返回状态码
msg	是	String	NA	返回消息内容

（3） “创建问答对”API 接口

接口类型

请求方法	post
访问 URI	172.23.1.156/api/<version>/qapairs/
通信协议	https 或 http

请求参数

参数名称	是否必须	参数类型	默认值	描述
kbid	是	String	NA	知识库标识
id	是	String	NA	知识点标识
questions	是	String	NA	问题列表
question	是	String	NA	问题
answer	是	String	NA	答案

响应参数

参数名称	是否必须	参数类型	默认值	描述
code	是	String	NA	返回状态码
msg	是	String	NA	返回消息内容

（4）“批量创建问答对”API 接口

接口类型

请求方法	post
访问 URI	172.23.1.156/api/<version>/qapairs/?batch=True
通信协议	https 或 http

请求参数

参数名称	是否必须	参数类型	默认值	描述
kbid	是	String	NA	知识库标识
qas	是	String	NA	知识点列表
id	是	String	NA	知识点标识
questions	是	String	NA	问题列表
question	是	String	NA	问题
answer	是	String	NA	答案

响应参数

参数名称	是否必须	参数类型	默认值	描述
code	是	String	NA	返回状态码
msg	是	String	NA	返回消息内容

（5）“更新问答对”API 接口

接口类型

请求方法	put
访问 URI	172.23.1.156/api/<version>/qapairs/<questionid>/
通信协议	https 或 http

请求参数

参数名称	是否必须	参数类型	默认值	描述
kbid	是	String	NA	知识库标识
questions	是	String	NA	问题列表
question	是	String	NA	问题
answer	是	String	NA	答案

响应参数

参数名称	是否必须	参数类型	默认值	描述
code	是	String	NA	返回状态码
msg	是	String	NA	返回消息内容

（6）“删除问答对”API 接口

接口类型

请求方法	delete
访问 URI	172.23.1.156/api/<version>/qapairs/<questionid>/
通信协议	https 或 http

请求参数

空

响应参数

参数名称	是否必须	参数类型	默认值	描述
code	是	String	NA	返回状态码
msg	是	String	NA	返回消息内容

5.1.2 训练命令处理功能类

主要实现包括 4 个 API 接口，如下：

（7）“训练”API 接口

接口类型

请求方法	post
访问 URI	172.23.1.156/api/<version>/train/prepub/
通信协议	https 或 http

请求参数

参数名称	是否必须	参数类型	默认值	描述
kbids	是	String	NA	知识库标识列表
kbid	是	String	NA	知识库标识

响应参数

参数名称	是否必须	参数类型	默认值	描述
code	是	String	NA	返回状态码
msg	是	String	NA	返回消息内容

（8）“训练状态检测”API 接口

接口类型

请求方法	Post
访问 URI	172.23.1.156/api/<version>/train/inspect/
通信协议	https 或 http

请求参数

参数名称	是否必须	参数类型	默认值	描述
kbids	是	String	NA	知识库标识列表
kbid	是	String	NA	知识库标识

响应参数

参数名称	是否必须	参数类型	默认值	描述
code	是	String	NA	返回状态码
msg	是	String	NA	返回消息内容
data	是	String	NA	返回数据结果
status	是	String	NA	预发布状态：0 未训练，1 训练中，2 训练完成，3 训练失败

（9）“发布”API 接口

接口类型

请求方法	post
访问 URI	172.23.1.156/api/<version>/train/online/
通信协议	https 或 http

请求参数

参数名称	是否必须	参数类型	默认值	描述
kbids	是	String	NA	知识库标识列表
kbid	是	String	NA	知识库标识

响应参数

参数名称	是否必须	参数类型	默认值	描述
code	是	String	NA	返回状态码
msg	是	String	NA	返回消息内容

（10）“发布状态检测”API 接口

接口类型

请求方法	post
访问 URI	172.23.1.156/api/<version>/train/status/
通信协议	https 或 http

请求参数

参数名称	是否必须	参数类型	默认值	描述
kbids	是	String	NA	知识库标识列表
kbid	是	String	NA	知识库标识

响应参数

参数名称	是否必须	参数类型	默认值	描述
code	是	String	NA	返回状态码
msg	是	String	NA	返回消息内容
data	是	String	NA	返回数据结果
status	是	String	NA	发布状态：0 未发布，1 发布中，2 发布完成，3 发布失败

5.1.3 QA 命令处理功能类

主要实现包括 2 个 API 接口，如下：

（11） “预发布 QA”API 接口

接口类型

请求方法	post
访问 URI	172.23.1.156/api/<version>/qas/prepub/
通信协议	https 或 http

请求参数

参数名称	是否必须	参数类型	默认值	描述
kbids	是	String	NA	知识库标识列表
kbid	是	String	NA	知识库标识
question	是	String	NA	问题

响应参数

参数名称	是否必须	参数类型	默认值	描述
code	是	String	NA	返回状态码
msg	是	String	NA	返回消息内容
data	是	String	NA	返回数据结果
question	是	String	NA	知识库中匹配到的问题
questionid	是	String	NA	知识库中匹配到的问题唯一标识
answer	是	String	NA	知识库中匹配到的答案
score	是	String	NA	该答案对应的得分（0-1.0 之间）
kbid	是	String	NA	知识库的唯一标识

（12）“正式 QA”API 接口

接口类型

请求方法	post
访问 URI	172.23.1.156/api/<version>/qas/formal/
通信协议	https 或 http

请求参数

参数名称	是否必须	参数类型	默认值	描述
kbids	是	String	NA	知识库标识列表
kbid	是	String	NA	知识库标识
question	是	String	NA	问题

响应参数

参数名称	是否必须	参数类型	默认值	描述
code	是	String	NA	返回状态码
msg	是	String	NA	返回消息内容
data	是	String	NA	返回数据结果
question	是	String	NA	知识库中匹配到的问题
questionid	是	String	NA	知识库中匹配到的问题唯一标识
answer	是	String	NA	知识库中匹配到的答案
score	是	String	NA	该答案对应的得分（0-1.0 之间）
kbId	是	String	NA	知识库的唯一标识

5.2 内部 API 接口设计

结合 QA 命令处理功能的实际需求，为实现 Hub 层与底层 QA Chatterbot+组件之间的 Web 通信，需开发 1 个内部 REST API 接口，主要用于 Hub 层传递的 QA 命令处理。具体如下：

主要实现包括 1 个 API 接口，如下：

（13） “通用 QA”API 接口

接口类型

请求方法	post
访问 URI	VM IP:Port/id=<B Box id>/
通信协议	https 或 http

请求参数

参数名称	是否必须	参数类型	默认值	描述
kbid	是	String	NA	知识库标识
question	是	String	NA	问题
version	是	String	NA	版本号

响应参数

参数名称	是否必须	参数类型	默认值	描述
code	是	String	NA	返回状态码
msg	是	String	NA	返回消息内容
data	是	String	NA	返回数据结果
questionid	是	String	NA	知识图中匹配到的问题唯一标识
answer	是	String	NA	知识图中匹配到的答案
score	是	String	NA	该答案对应的得分（0-1.0 之间）

第六章 数据库系统设计

6.1 设计要求

语料库拟采用 MySQL 数据库来创建和管理，表目录如表 2 所示。

表 2 语料库中 MySQL 数据库的表目录

数据库表目录				
序号	包名/模块名	表名	表说明	表字段数量
1	API	app_knowgraphs	KG 索引表	5
2	API	app_lexiconindexes	库索引表	3
3	API	app_lexiconmaps	知识库与词库关联表	4
4	API	app_qakgmap	正式 QA 对应的 KG 表	3
5	API	app_questions	问题表	6
6	API	app_synonyms	同义词表	5
7	API	app_words	过滤词或专有名词表	4
8	API	app_tasks	训练任务表	6

而知识图拟采用 MongoDB 数据库来存储和管理。由于 MongoDB 没有固定结构，每张表每行数据可以有不同的结构，所以一般直接用 find()查询数据，并且使用 MongoDB 的架构设计变得无比重要。本技术方案中所采用的设计思路是：

（1）一个知识图只对应着一个版本的知识库，当同一个知识库具有若干个不同版本时，每个版本的知识库均有唯一的知识图与之一一对应。

（2）根据原生 Chatterbot 的源码可知，一个知识图就是一个 MongoDB 数据库，该数据库名可以指定，但该数据库中集合名（表名）确是事先已知的，即“statements”，用于存放训练好的“问答对”

信息。还有一个集合名（表名），叫“conversations”，用于存放 QA 时会话 id。

（3）由上可知，知识图的数量==MongoDB 数据库的数量；另外，该软件系统设计：MongoDB 数据库的数量==知识库的数量*10。

6.2 语料 MySQL 数据库设计

6.2.1 app_knowgraphs（KG 索引表）

表 3　app_knowgraphs 数据表结构

app_knowgraphs （KG 索引表）							
序号	列名	数据类型	主键	自增	允许空	默认值	列说明
1	id	varchar(50)	√				KG 唯一标识
2	kg_version	varchar(50)					KG 版本号
3	is_trained	tinyint(1)					是否训练完成
4	in_use	tinyint(1)					是否使用中
5	know_base_id	varchar(50)					知识库唯一标识
6	created_at	datetime(6)			√		创建时间
7	updated_at	datetime(6)			√		修改时间

6.2.2 app_lexiconindexes（库索引表）

表 4　app_lexiconindexes 数据表结构

app_lexiconindexes （库索引表）							
序号	列名	数据类型	主键	自增	允许空	默认值	列说明
1	id	varchar(50)	√				库表唯一标识
2	lexicon_type	smallint(6)					词库类型：0 知识库, 1 过滤词, 2 专有名词, 3 同义词
3	is_general	tinyint(1)				0	类型：0 非通用，1 通用
4	created_at	datetime(6)			√		创建时间
5	is_delete	tinyint(1)					是否已删除
6	updated_at	datetime(6)			√		修改时间

6.2.3 app_lexiconmaps（知识库与词库关联表）

表 5　app_lexiconmaps 数据表结构

app_lexiconmaps （知识库与词库关联表）							
序号	列名	数据类型	主键	自增	允许空	默认值	列说明
1	id	int(11)	√	√			
2	wb_type	smallint(6)					词库库类型：1 过滤词, 2 专有名词, 3 同义词
3	know_base_id	varchar(50)					知识库唯一标识
4	word_bank_id	varchar(50)					词库唯一标识
5	created_at	datetime(6)			√		创建时间
6	updated_at	datetime(6)			√		修改时间

6.2.4 app_qakgmap（正式 QA 对应的 KG 表）

表 6　app_qakgmap 数据表结构

app_qakgmap （正式 QA 对应的 KG 表）							
序号	列名	数据类型	主键	自增	允许空	默认值	列说明
1	id	int(11)	√	√			
2	kb_id	varchar(50)					知识库唯一标识
3	kg_id	varchar(50)					KG 唯一标识
4	created_at	datetime(6)			√		创建时间
5	updated_at	datetime(6)			√		修改时间

6.2.5 app_questions（问题表）

表 7　app_questions 数据表结构

app_questions （问题表）							
序号	列名	数据类型	主键	自增	允许空	默认值	列说明
1	id	varchar(50)	√				问题唯一标识
2	content	varchar(256)					问题
3	is_subordinate	tinyint(1)					问题类型：0 标准问题，1 相似问题
4	answer	varchar(50)					答案唯一标识
5	kb_id	varchar(50)					知识库唯一标识
6	standard_id	varchar(50)			√		标准问题唯一标识
7	created_at	datetime(6)			√		创建时间
8	is_delete	tinyint(1)					是否已删除
9	updated_at	datetime(6)			√		修改时间

6.2.6 app_synonyms（同义词表）

表 8　app_synonyms 数据表结构

app_synonyms （同义词表）							
序号	列名	数据类型	主键	自增	允许空	默认值	列说明
1	id	varchar(50)	√				同义词唯一标识
2	content	varchar(50)					同义词
3	is_subordinate	tinyint(1)				0	类型：0 基准同义词，1 普通同义词
4	basic_id	varchar(50)					基准词唯一标识
5	wb_id	varchar(50)					同义词库唯一标识
6	created_at	datetime(6)			√		创建时间
7	is_delete	tinyint(1)					是否已删除
8	updated_at	datetime(6)			√		修改时间

6.2.7 app_words（过滤词或专有名词表）

表 9　app_words 数据表结构

app_words （过滤词或专有名词表）							
序号	列名	数据类型	主键	自增	允许空	默认值	列说明
1	id	varchar(50)	√				过滤词或专有名词唯一标识
2	content	varchar(50)					词
3	word_character	smallint(6)					类型：1 过滤词，2 专有名词
4	wb_id	varchar(50)					词库唯一标识
5	created_at	datetime(6)			√		创建时间
6	is_delete	tinyint(1)					是否已删除
7	updated_at	datetime(6)			√		修改时间

6.2.8 app_tasks（训练任务表）

表 9　app_tasks 数据表结构

app_words （过滤词或专有名词表）							
序号	列名	数据类型	主键	自增	允许空	默认值	列说明
1	id	varchar(50)	√				唯一标识
2	kbid	varchar(50)					知识库唯一标识
3	kg_version	varchar(50)					KG 版本号
4	status	tinyint(6)					任务状态:0 表示未训练，1 表示训练中，2 表示已训练，3 表示正在用
5	created_at	datetime(6)			√		创建时间
6	is_delete	tinyint(1)					是否已删除
7	updated_at	datetime(6)			√		最近一次操作时间

6.3 知识图 MongoDB 数据库设计

6.3.1 “statements”表中“问题”行

表 10　“statements”表中“问题”行的表结构

序号	列名	数据类型	允许空	默认值	列说明
1	_id	对象 id			唯一标识
2	text	文本型			文本信息
3	in_response_to	数组型	√		关联问题信息（可为空）
4	extra_data	字典型	√		额外数据（可为空）
5	created_at	日期型			创建时间
6	occurrence	整型			问答匹配到的次数

6.3.2 “statements”表中“答案”行

表 11 “statements”表中“答案”行的表结构

序号	列名	数据类型	允许空	默认值	列说明
1	_id	对象 id			唯一标识
2	text	文本型			答案的文本信息
3	in_response_to	数组型			关联问题信息
4	created_at	日期型			创建时间
5	text	文本型			问题的文本信息
6	occurrence	整型			问答匹配到的次数
7	extra_data	字典型	√		额外数据（可为空）

第七章 非功能性设计

7.1 可靠性设计

训练的数据量和训练时间；

KB 数据量和 QA 时间的关系。

7.2 可维护性设计

（1）训练数据量统计和训练时间；

（2）QA 速度；

（3）KB 发布时间统计；

（4）软件系统异常时，必须要记录，比如训练失败、返回结果失败、输出失败和输入失败等情形。

7.3 安全性设计

该软件系统对外展示的只有 Web API 接口，为从软件设计上保证安全性，需要做的如下：

（1）接收的语料库信息和问题信息必须做检查

（2）返回的答案信息做必要过滤

（3）SQL 语句做处理。Django 框架是利用对象关系映射（Object Relational Mapping，ORM）方法，可以将对象数据的操作直接映射到关系型数据库结构中，这样在具体的操作实体对象的时候，就不需要直接使用 SQ L 语句，只需简单的操作实体对象的属性和方法。

7.4 可扩展性设计

该软件系统的目的在于替代阿里云的接口，现有的设计理念是将“导入”语料库、“训练”为知识图和“QA”服务相互独立，便于今后在软件部署上的扩展。另外，语料库和知识图库都可以独立，适合分布式环境。

7.5 易用性设计

由于该软件系统对外展示的只有 Web API 接口，针对上层业务系统的使用本身具有易用性特点。

7.6 容错性设计

该软件系统在使用中如果遇到硬件资源不足时，存在无法对外提供训练或 QA 服务的情况，此时可以考虑增加硬件资源或关闭低优先级用户服务的解决方法。

7.7 可测试性设计

在该软件系统实现过程中，结合接口和功能的测试方法，提供基础测试程序。

第八章 环境部署

经过初步调研可知，该软件的实际部署环境可以是：

（1）Linux 平台：该软件运行的操作系统；

（2）Python 运行环境：该软件开发和运行的语言环境；

（3）Django 框架环境：该软件对外的 Web API 架构环境；

（4）ZooKeeper 库：为完成消息一致性功能采用的分布式架构；

（5）Falcon 框架环境：该软件对内的高性能轻量级 REST API 架构环境；

（6）Gunicorn 服务器环境：该软件处理底层 REST 请求的 Web 服务器环境；

（7）MySQL 数据库环境：该软件处理语料库的数据库环境；

（8）MongoDB 数据库环境：该软件保存知识图的数据库环境。

第九章 关键技术和风险点分析

9.1 关键技术和解决方案

软件开发中可能的关键技术：

（1）python 中原生 Chatterbot 库

（2）python 中 Django 架构

（3）python 中 ZooKeeper 库

（4）python 中多进程处理技术

（5）python 中 falcon 框架与 gunicorn 服务器的结合

（6）执行原生 Chatterbot 库的训练功能前，需要预处理技术

（7）增加原生 Chatterbot 库中训练器功能，需要直接从 MySQL 数据库的语料中训练

（8）增加原生 Chatterbot 库中 QA 功能的返回 A 类型（包含置信度）

9.2 风险点分析

软件开发中可能的风险点：

（1）Hub 层中 3 个主要命令处理模块的实现可能存在技术难点；

（2）Chatterbot+组件在训练和 QA 过程中预处理功能的实现，以及与原有训练和 QA 功能的结合可能存在技术难点；

（3）实际部署环境的复杂性，可能会影响开发的软件系统性能。