技术开发路线

1.简介

1.1 项目名称：智能问答系统

1.2 项目开发者：

- 薛金龙
- 许泰格

1.3 项目开发工具，环境，框架，语言

- 开发工具：VScode, Pycharm, PostgreSQL;
- 开发环境：windows, linux;
- 开发框架：django;
- 开发语言：python, html, css, javascript;

2 设计思路和技术路线

2.1 前端设计

2.1.1 前端设计思路：

受微信聊天启发，QA 对呈现采用对话方式；左边是对话框，右边是常见问题汇总。对话框实现 QA 呈现，常见问题点击后会从右侧栏拉出具体展示。

对话框设计核心思路是采用 ajax 异步刷新，显示一对 QA 后下滑栏自动下移，可以看到历史问答记录，这也是我们的一个创新点。

2.2 后端设计

2.2.1 django 框架介绍

Django 是一个开放源代码的 Web 应用框架，由 Python 写成。采用了 MVC 的软件设计模式，即模型 M，视图 V 和控制器 C。Django 框架的核心包括：一个面向对象的映射器(ORM)，用作数据模型（以 Python 类的形式定义）和关系性数据库间的媒介；一个基于正则表达式的 URL 分发器；一个视图系统(View)，用于处理请求；以及一个模板系统(Template)。

2.2.2 后端设计思路：

在基于 django 框架自带的管理界面，我们进行了进一步的改造，从而更方便的实现知识库的管理，以及 QA 对的增删改查。

2.2.3 前后端连接

将写好的 web 前端文件（html,css,javascript 文件）放到对应的 djiago 框架中，同时与数据库连接（数据库采用的是 postgresql），并进行相应的路径、参数配置，并进行下一步的服务器连接与部署。

2.3 服务器部署

2.3.1 部署实现步骤

- 租服务器
- 用 xftp 将 django 框架里的所有代码放到服务器，并用 xshell 连接和操作
- 在服务器上安装 nginx , postgresql, python 3
- 将 django 框架里所有代码放到 nginx 对应的文件下
- 配置路径、参数、调试，最后即可完成服务器部署

2.3.2 nginx 配置

Nginx 是一个 web 服务器也可以用来做负载均衡及反向代理使用，目前使用最多的就是负载均衡。配置过程是：

- 进入用户目录下载程序，下载 Nginx 及相关组件

- 安装 Nginx 及相关组件：openssl 安装，pcre 安装，zlib 安装，nginx 安装
- 启动 Nginx,，开启外网访问
- Nginx 负载均衡配置

2.4 基于 NLP 的 QA 对生成设计思路

2.4.1 分词和 QA 对自动生成实现方法：

我们用的是 PYLTP（哈工大的一个开源中文文本处理 python 库），对用到的中文分词和中文词性先标注，然后根据用 python 爬虫爬下来的文档里面的答案语句（python 爬虫时，根据 html 里的不同标签，找到问题和答案对应的标签位置，来进行获取问题和答案文本），分标签这一步在 spider.py 中已经处理，实现根据答案生成问题，比如碰到名词，则直接加“什么是”；碰到动词+名词语句，用分词代码词性标记后，在此类语句加“如何”；碰到问题下的 html 标签为<li>时，则直接在此类问题后面加“有哪些”；如果碰到有标签形如“常见问题”“热门问题”，则直接将问题和答案存入知识库，不做分词以及词性标注处理；

对于关键词提取，可以用 jieba 库，我们采用基于 TF-IDF 算法进行关键词抽取，TF-IDF 的主要思想就是：如果某个词在一篇文档中出现的频率高，也即 TF 高；并且在语料库中其他文档中很少出现，即 DF 的低，也即 IDF 高，则认为这个词具有很好的类别区分能力。但 jieba 库和 PYLTP 这两个库不能满足大部分的关键词提取，所以我们就在此基础上自己训练关键词提取了。

TF-IDF 算法介绍：（粘代码）

```
from jieba import analyse # 引入 TF-IDF 关键词抽取接口 tfidf = analyse.extract_tags # 原始文本 text = "线程是程序执行时的最小单位，它是进程的一个执行流，\ 是 CPU 调度和分派的基本单位，一个进程可以由很多个线程组成，\ 线程间共享进程的所有资源，每个线程有自己的堆栈和局部变量。\ 线程由 CPU 独立调度执行，在多 CPU 环境下就允许多个线程同时运行。\ 同样多线程也可以实现并发操作，每个请求分配一个线程来处理。" # 基于 TF-IDF 算法进行关键词抽取 keywords = tfidf(text) print "keywords by tfidf:" # 输出抽取出的关键词 for keyword in keywords: print keyword + "/",
```

基于 TF-IDF 算法抽取关键词的主调函数是 TFIDF.extract_tags 函数，主要是在 jieba/analyse/tfidf.py 中实现。

其中 TFIDF 是为 TF-IDF 算法抽取关键词所定义的类。类在初始化时，默认加载了分词函数 tokenizer = jieba.dt、词性标注函数 postokenizer = jieba.posseg.dt、停用词 stop_words = self.STOP_WORDS.copy()、idf 词典 idf_loader = IDFLoader(idf_path or DEFAULT_IDF)等，并获取 idf 词典及 idf 中值（如果某个词没有出现在 idf 词典中，则将 idf 中值作为这个词的 idf 值）。

```
def __init__(self, idf_path=None): # 加载 self.tokenizer = jieba.dt self.postokenizer = jieba.posseg.dt self.stop_words = self.STOP_WORDS.copy() self.idf_loader = IDFLoader(idf_path or DEFAULT_IDF) self.idf_freq, self.median_idf = self.idf_loader.get_idf()
```

然后开始通过 TF-IDF 算法进行关键词抽取。

首先根据是否传入了词性限制集合，来决定是调用词性标注接口还是调用分词接口。例如，词性限制集合为["ns", "n", "vn", "v", "nr"]，表示只能从词性为地名、名词、动名词、动词、人名这些词性的词中抽取关键词。

1） 如果传入了词性限制集合，首先调用词性标注接口，对输入句子进行词性标注，得到分词及对应的词性；依次遍历分词结果，如果该词的词性不在词性限制集合中，则跳过；如果词的长度小于 2，或者词为停用词，则跳过；最后将满足条件的词添加到词频词典中，出现的次数加 1；然后遍历词频词典，根据 idf 词典得到每个词的 idf 值，并除以词频词典中的次数总和，得到每个词的 tf * idf 值；如果设置了权重标志位，则根据 tf-idf 值对词频词典中的词进行降序排序，然后输出 topK 个词作为关键词；

2） 如果没有传入词性限制集合，首先调用分词接口，对输入句子进行分词，得到分词；依次遍历分词结果，如果词的长度小于 2，或者词为停用词，则跳过；最后将满足条件的词添加到词频词典中，出现的次数加 1；然后遍历词频词典，根据 idf 词典得到每个词的 idf 值，并除以词频词典中的次数总和，得到每个词的 tf * idf 值；如果设置了权重标志位，则根据 tf-idf 值对词频词典中的词进行降序排序，然后输出 topK 个词作为关键词；

（沾代码）

```
def extract_tags(self, sentence, topK=20, withWeight=False, allowPOS=(), withFlag=False): # 传入了词性限制集合 if allowPOS: allowPOS = frozenset(allowPOS) # 调用词性标注接口 words = self.postokenizer.cut(sentence) # 没有传入词性限制集合 else: # 调用分词接口 words = self.tokenizer.cut(sentence) freq = {} for w in words: if allowPOS: if w.flag not in allowPOS: continue elif not withFlag: w = w.word wc = w.word if allowPOS and withFlag else w # 判断词的长度是否小于 2，或者词是否为停用词 if len(wc.strip()) < 2 or wc.lower() in self.stop_words: continue # 将其添加到词频词典中，次数加 1 freq[w] = freq.get(w, 0.0) + 1.0 # 统计词频词典中的总次数 total = sum(freq.values()) for k in freq: kw = k.word if allowPOS and withFlag else k # 计算每个词的 tf-idf 值 freq[k] *= self.idf_freq.get(kw, self.median_idf) / total # 根据 tf-idf 值进行排序 if withWeight: tags = sorted(freq.items(), key=itemgetter(1), reverse=True) else: tags = sorted(freq, key=freq.__getitem__, reverse=True) # 输出 topK 个词作为关键词 if topK: return tags[:topK] else: return tags
```

2.4.2 知识库构建

知识字典 D 可用三元组表示如下: D = (O , T , E) 把三元组理解为 (实体 entity,实体关系 relation,实体 entity)，把实体看作是结点，把实体关系（包括属性，类别等等）看作是一条边，那么包含了大量三元组的知识库就成为了一个庞大的知识图。

实体链指(Entity linking) ，即将文档中的实体名字链接到知识库中特定的实体上。它主要涉及自然语言处理领域的两个经典问题实体识别 (Entity Recognition) 与实体消歧 (Entity Disambiguation)，简单地来说，就是要从文档中识别出人名、地名、机构名、电影等命名实体。并且，在不同环境下同一实体名称可能存在歧义，如苹果，我们需要根据上下文环境进行消歧。 关系抽取 (Relation extraction)，即将文档中的实体关系抽取出来，主要涉及到的技术有词性标注 (Part-of-Speech tagging, POS)，语法分析，依存关系树 (dependency tree) 以及构建 SVM、最大熵模型等分类器进行关系分类等。

理想的联合学习应该如下：输入一个句子，通过实体识别和关系抽取联合模型，直接得到有关系的实体三元组。这种可以克服上面流水线方法的缺点，但是可能会有更复杂的结构。实体提取用哈工大的库 pyltp(可以用 CTB 模型重新做分词模型)，导入官方模型即可。

2.4.3 语句相似度计算

在中文分词功能实现后，接下来就是如何产生相似度，核心思路是：

- 第一步：分词
- 第二步：汇总关键词，放到一个列表中
- 第三步：计算词频和生成词频向量

所以计算两个句子的相似度就是比较两个向量的相似度，用到下面的公式：

$$\cos\theta = \frac{\sum_{i=1}^{n}(A_i \times B_i)}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}}$$

N 维向量的余弦值代码实现：

```python
import math
def cos_dist(a, b):
if len(a) != len(b):
    return None
part_up = 0.0
a_sq = 0.0
b_sq = 0.0
for a1, b1 in zip(a,b):
    part_up += a1*b1
    a_sq += a1**2
    b_sq += b1**2
part_down = math.sqrt(a_sq*b_sq)
if part_down == 0.0:
    return None
else:
    return part_up / part_down
```

2.4.4 项目目前存在的问题

语句相似度匹配算法（模糊匹配）不成熟，即系统对用户输入的问题语句不能进行模糊匹配，只能输入知识库中已存入的问题。

设计框架如下：