软件开发实训

**课程设计指导书**

**课堂派 班级：YYTTC5**

题 目： 新闻网站

院 系： 计算机科学与信息安全学院

专 业： 计算机科学与技术

指导教师： 梁海

学生姓名： 潘承锋

学 号： 1400310124

**桂林电子科技大学**

**计算机与信息安全学院**

**2017．12．26**

目 录

[一. 项目概述 2](#_Toc502214953)

[1.1 项目说明 2](#_Toc502214954)

[1.2 项目模块划分 2](#_Toc502214955)

[1.3 主要源代码结构 2](#_Toc502214956)

[1.4 运行环境 3](#_Toc502214957)

[1.5 使用说明 3](#_Toc502214958)

[二. 新闻数据模块 6](#_Toc502214959)

[2.1 新闻数据库设计 6](#_Toc502214960)

[2.2 新闻爬取模块 6](#_Toc502214961)

[2.3 新闻聚类模块 8](#_Toc502214962)

[2.4 个性化推荐模块 10](#_Toc502214963)

[2.5 后台常驻进程 13](#_Toc502214964)

[三. 网站模块 14](#_Toc502214965)

[3.1 网站架构 14](#_Toc502214966)

[3.2 网站前端 14](#_Toc502214967)

[3.3 网站后端 16](#_Toc502214968)

[3.3.1 主要接口 16](#_Toc502214969)

[3.3.2 邮箱认证 18](#_Toc502214970)

[3.3.4 redis缓存 23](#_Toc502214971)

[四. 功能测试 25](#_Toc502214972)

[4.1 相关新闻测试 25](#_Toc502214973)

[4.2 个性推荐测试 25](#_Toc502214974)

[五. 项目感想 26](#_Toc502214975)

[六. 参考资料 27](#_Toc502214976)

# 一. 项目概述

## 1.1 项目说明

本项目实现了一个新闻资讯收集、整理，进行个性化展示的网站。

网站服务器端使用Node.js作为服务器框架，使用MongoDB提供数据存储服务，并使用Redis作为数据缓存服务。用户管理方面实现了邮箱认证注册登录，用户定制关键词等功能。

在新闻数据爬取上使用Python BeautifulSoup框架爬取并解析各大主流新闻网站的新闻页面，获取所需要的新闻数据；在新闻数据处理上通过维基百科提供的语料库，使用Word2Vec等自然语言处理相关的工具进行中文关键词建模，实现了按新闻关键词进行聚类，并根据用户感兴趣关键词进行新闻个性推荐；同时，使用了微软认知服务提供的推荐API，用于针对用户的特定行为进行建模推荐。

## 1.2 项目模块划分

- 新闻数据模块

    - 新闻爬取模块：使用Python BeautifulSoup框架爬取并解析各大主流新闻网站的新闻页面，获取所需要的新闻数据存入数据库

    - 新闻聚类模块：使用中文维基语料库建立关键词模型，并按新闻关键词进行聚类

    - 用户推荐模块：使用微软认知服务-推荐系统提供的RESTful API进行用户行为统计并推荐新闻

- 网站模块

    - 网站前端

    - 网站服务器端

## 1.3 主要源代码结构

NewsFeeds\

├── Graphs 实际运行效果截图, 系统架构图等\

├── README.md 开发文档\

├── NewsFeed\_Android 安卓客户端源代码\

├── Scripts 新闻数据相关脚本\

│ ├── word2vec 中文语料库训练脚本\

│ ├── config.ini 配置文件\

│ ├── modelUpdater.py 推荐系统相关实现脚本\

│ ├── newsCrawler.py 新闻爬取脚本\

│ ├── service.py 服务器后台常驻进程脚本\

└── Server 新闻网站服务器\

├── app.js Node.js入口脚本\

├── bin 服务器启动脚本\

├── config.json 网站配置文件（服务器地址，数据库源等）\

├── package.json Node.js依赖模块声明\

├── public 网站静态文件\

├── routes Node.js路由\

├── scripts 网站后端逻辑实现\

└── views 网站前端实现

192 directories, 65 files

## 1.4 运行环境

- 操作系统：Windows/Linux/OSX

- 服务器软件：Node.js 6.9.0及以上

- 逻辑数据库：MongoDB 2.4.1及以上

- 缓存数据库：Redis 3.2.9及以上

## 1.5 使用说明

1.进入服务器所在目录 Server2.根据各个服务器配置，按如下说明更改配置文件config.json：

```json

{

"host": "服务器主机地址",

"port": "服务器主机端口",

"connect": "MongoDB连接字符串",

"newscol": "MongoDB新闻数据库名",

"usercol": "MongoDB用户数据库名",

"mailoptions": {

"service": "邮件服务",

"email": "邮箱账号",

"password": "邮箱密码"

},

"redis": {

"port": "Redis端口",

"host": "Redis服务器地址"

},

"database": {

"host": "MongoDB数据库服务器地址",

"port": "MongoDB服务器端口",

"user": "Mongo用户名",

"password": "密码"

},

"api": {

"modelid": "微软认知服务-推荐系统模型ID",

"endpoint": "https://westus.api.cognitive.microsoft.com/recommendations/v4.0",

"token": "微软认知服务认证Token"

},

"genres": {

"personal": "推荐",

"hot": "热门",

"society": "社会",

"domestic": "国内",

"global": "国际",

"technology": "科技",

"finance": "经济",

"war": "军事",

"education": "教育",

"car": "汽车",

"game": "游戏",

"discover": "探索",

"entertain": "娱乐",

"fashion": "时尚",

"health": "健康",

"history": "历史",

"mobile": "数码",

"sport": "体育",

"travel": "旅游"

}

}```

3.安装Node.js依赖包：

npm install

4.启动服务器

npm start

# 二. 新闻数据模块

## 2.1 新闻数据库设计

由于本项目使用基于文档存储的MongDB，每一个新闻使用一种类似JSON的数据类型存储，不需要有固定的结构（Schema）。但为了便于后续的建模与显示，在新闻爬取模块需要将每个新闻网站爬取下来的数据按照以下字段及其对应含义存储到数据库中：

{

    "\_id" : 新闻id,

    "title" : 新闻标题（唯一索引项）,

    "source" : 新闻来源,

    "time" : 发表时间,

    "abstract" : 新闻摘要,

    "comments\_count" : 新闻评论数,

    "favorite\_count" : 新闻收藏数,

    "genre" : 新闻类别,

    "has\_image" : 是否有图片,

    "imgurls" : 图片链接(数组),

    "keywords" : 标题包含关键词(数组),

    "related\_words" : 其他相关关键词(数组),

    "uploaded" : 是否已上传到微软认知服务模型

}

## 2.2 新闻爬取模块

本项目中新闻爬取模块主要实现了对腾讯、网易、凤凰网、今日头条等四个网站的新闻数据爬取。对于从各个网站解析出的每个新闻，最终存入数据库的字段及格式需要统一如前一小节中所述。

由于从网站上直接获取到的只是HTML数据(除了今日头条使用API能够直接获取到JSON格式的新闻数据)，因此需要通过BeautifulSoup等HTML解析器来获取HTML DOM中有意义的那些项。以爬取腾讯新

闻的脚本实现为例：

首先找出腾讯每类新闻的网页链接，存到一个Dict对象中：

```python

self.ADDRS\_LIST = {

'http://society.qq.com/': 'society',

'http://ent.qq.com/': 'entertain',

'http://sports.qq.com/': 'sport',

'http://finance.qq.com/': 'finance',

'http://mil.qq.com/mil\_index.htm': 'war',

'http://news.qq.com/world\_index.shtml': 'global',

'http://cul.qq.com/': 'culture'

...

}

因此在解析HTML时首先要通过`BeautifulSoup(response.text, "html.parser")`将HTML字符串转化为DOM树结构，再用`list.findAll('div', {'class' : ['Q-pList'] })`方法得到每个Q-pList包含的DOM元素，之后使用同样的方法再获取其内部的标题，关键字等内容并存入数据库即可：

```python

for (addr, catogory) in self.ADDRS\_LIST.items():

response = requests.get(addr, HEADERS)

if response.status\_code != 200:

print('response code = ', response.status\_code)

exit()

soup = BeautifulSoup(response.text, "html.parser")

lists = soup.findAll('div', {'class' : 'list'})

for list in lists:

for item in list.findAll('div', {'class' : ['Q-tpWrap', 'Q-pList'] }):

try:

linkto = item.find('a', {'class' : 'linkto'})

title = linkto.text

docurl = linkto.get('href')

keywords = item.find('span',{'class':'keywords'}).text

imgs = [ a.get('src') if a.get('src') else a.get('\_src') for a in item.findAll('img')]

comment = item.find('a', {'class' : 'discuzBtn'})

if comment:

commentNum = comment.text

commenturl = comment.get('href')

source = item.find('span', {'class' : 'from'}).text

print(catogory, title.strip())

except:

print\_exc()

exit()

## 2.3 新闻聚类模块

新闻聚类模块的实现思路比较简单，主要通过使用Word2Vec建立关键词库模型后，找出每个新闻关键字(keywords字段)的近义词作为该新闻的related*\_words记录下来，若一个新闻的related\_*words与另一个新闻的keywords有交集则认为这两个新闻是相关的。

python3 process\_wiki.py zhwiki-latest-pages-articles.xml.bz2 wiki.cn.text`

整个过程约10分钟，处理完后得到如下所示的文本文件：

由于维基百科中文语料库提供的只有繁体中文的语料，而且可以看出还有一些英文和其他标点字符，因此我们需要先转换成简体，对中文进行分词，再去掉英文等无用字符。

简繁体的转换主要通过[OpenCC](https://github.com/BYVoid/OpenCC)来实现：

opencc -i wiki.cn.text -o wiki.cn.text.jian -c t2s.json

中文的分词没有英文那么简单有天然的分隔符，但我们可以利用一些分词工具进行简单的分词工作。python上比较好用的分词软件是[jieba中文分词](https://github.com/fxsjy/jieba)

通过对其给出的示例进行部分修改后得到脚本seperate\_words.py，执行命令：python3 separate\_words.py wiki.cn.text.jian wiki.cn.text.jian.seq

编写脚本remove\_words.py，对分出来的词使用正则表达式匹配去除英文和标点字符，仅保留中文词汇：

python3 remove\_words.py wiki.cn.text.jian.seq wiki.cn.text.jian.removed

最后，通过Google word2vec官方教程所给的脚本train*\_word2vec\_*model.py，对处理好的、符合格式要求的中文词汇集进行训练：

python3 train\_word2vec\_model.py wiki.cn.text.jian.removed wiki.en.text.jian.model wiki.en.text.jian.vector

训练好模型之后，接下来就是使用Word2Vec找出每一条新闻关键字的近义词，并保存在related*\_words字段中。这样在之后查询某一条新闻的相关新闻时，只需要对其related\_*words集合与其他新闻的keywords集合做交集即可：

def buildRelated(self):

collection = getConnection('mongo')[self.colName]

model = gensim.models.Word2Vec.load(self.word2vecModel)

try:

for news in collection.find({ "related\_words": { "$exists": 0 } }, projection={"keywords":1, "title": 1, "\_id": 1}):

relatedWords = set()

for keyword in news["keywords"]:

try:

relatedWords |= set(map(lambda t: t[0], model.most\_similar(keyword)))

except KeyError:

print("KeyError:", keyword)

continue

print(news["title"])

collection.update\_one({"\_id": news["\_id"]},

{"$set": { "related\_words": list(relatedWords) }})

except:

print\_exc()

finally:

model = None

gc.collect()

## 2.4 个性化推荐模块

个性化推荐模块主要使用微软认知服务中的[建议API服务](https://azure.microsoft.com/zh-cn/services/cognitive-services/recommendations/)，根据特定用户的购买历史记录，推荐引擎借助 Azure 机器学习进行构建，以提供专门针对该用户的建议，并使其享受个性化体验。

该服务内包含许多不同操作的RESTful API，每个API都有非常详尽的使用文档。该服务基本的使用模式：首先使用Create Model API建立一个推荐模型，然后按照规定格式的文本上传所有需要用于推荐的对象（Catalog Item，如本项目中的新闻），之后需要上传每个用户的使用记录（Usage）。之后便是调用最重要的Trigger Build API对已经上传入库的数据和记录进行建模分析，以下是该API的说明文档：

当一次Build完成之后，便能够通过[Get user-to-item recommendations API](https://westus.dev.cognitive.microsoft.com/docs/services/Recommendations.V4.0/operations/56f30d77eda5650db055a3dd)来获取对某个用户id特定的推荐新闻了。

由于这部分内容的实现多半是构造请求体，发送HTTP请求调用RESTful API，各个步骤的代码实现大同小异，因此使用两个例子来展示具体的实现。

function uploadUsage() {

let params = {

modelId: ""

}

let body = {

"userId": "string",

"buildId": 0,

"events": [

{

"eventType": "Click",

"itemId": "string",

"timestamp": "string",

"count": 0,

"unitPrice": 0.0

}

]

}

$.ajax({

url: "https://westus.api.cognitive.microsoft.com/recommendations/v4.0/models/28f64f3a-84a8-4f6d-889b-6c738d284aad/usage/events?"

+ $.param(params),

beforeSend: function(xhrObj){

// Request headers

xhrObj.setRequestHeader("Content-Type","application/json");

xhrObj.setRequestHeader("Ocp-Apim-Subscription-Key","62155e00332a4a62afbfe6478c8c9212");

},

type: "POST",

// Request body

data: body,

}).done(function(data) {

console.log("update success")

}).fail(function() {

alert("update error");

});

}

后端进程使用Trigger Build API启动一次统计建模（周期执行，如每天执行一次）：

```python

def triggerBuild(self):

collection = getConnection('mongo')[self.modelColName]

header = self.HEADERS\_JSON

buildurl = self.endpoint + '/models/%s/builds?' % (self.modelid)

body = json.dumps({

"description": "Simple recomendations build",

"buildType": "recommendation",

"buildParameters": {

"recommendation": {

"numberOfModelIterations": 40,

"numberOfModelDimensions": 20,

"itemCutOffLowerBound": 1,

"itemCutOffUpperBound": 10,

"userCutOffLowerBound": 0,

"userCutOffUpperBound": 0,

"enableModelingInsights": False,

"useFeaturesInModel": True,

"modelingFeatureList": "tag",

"allowColdItemPlacement": True,

"enableFeatureCorrelation": True,

"reasoningFeatureList": "tag",

"enableU2I": True

}

}

})

try:

resp = requests.post(buildurl, body, headers=header)

result = json.loads(resp.text)

print("url = ", buildurl)

print(result)

if resp.status\_code == 202: # success

collection.insert({

"buildId": result["buildId"],

"time": datetime.now().strftime(self.timeFormat),

"modelId": self.modelid,

"token": self.token

})

except:

print\_exc()

## 2.5 后台常驻进程

由于网站需要提供实时最新的新闻，因此服务器后端必须运行一个常驻进程周期性地爬取新闻，更新模型等。这里主要使用了python的schedule模块执行周期性操作，该模块提供了非常人性化的接口，如调用`schedule.every().day.do(updateModel)`便可以从调用时刻开始每隔一天执行一次updateModel函数，其中day也可以换成hour，minute等时间单位。通过这些接口就能够很方便地实现周期性调用的功能。

后台常驻进程脚本的主要实现代码如下：

def main():

global model

toutiao = newsCrawler.Toutiao()

tencent = newsCrawler.Tencent()

netease= newsCrawler.Netease()

loop = 1

interval = int(getConfig('default', 'request\_interval'))

schedule.every().hour.do(toutiao.start)

schedule.every().hour.do(tencent .start)

schedule.every().hour.do(netease.start)

schedule.every().hour.do(model.buildRelated)

schedule.every().day.do(updateModel)

while True:

time.sleep(interval)

# 三. 网站模块

## 3.1 网站架构

![网站架构图](<https://i.imgur.com/9KMajfS.png>)

## 3.2 网站前端

本网站前端主要使用Bootstrap+jQuery框架实现，使用Bootstrap的响应式布局，能够在多种分辨率下自适应地调整导航栏和内容分布。在页面渲染上选择前端渲染与服务端渲染相结合的方式，后端使用EJS模板引擎进行渲染。

前端主要实现了以下几个页面：

注册、登录：

![](https://github.com/375670450/NewsFeeds/blob/master/Graphs/Web/%E7%99%BB%E5%BD%95.png?raw=true)

上图为登录页面，点击下方的Register a new membership即可切换到注册页面。点击上方的NewsFeed Logo即可回到主页。

首页：

主页导航栏上方，用户登录后在右上角会显示登录邮箱以及登出按钮（若未登录则只显示登录按钮）。

导航栏的按钮为各个分类的新闻列表页面导航，目前一共添加了十八种类型的新闻，由于导航栏位置不足以显示，因此部分类型按钮放在"更多"下拉菜单中。

导航栏下方的区域用滚动图片显示10条热门新闻的标题及配图。

页面中间左半部分为热门新闻列表，由于在实际的新闻网站中数据库新闻条目较多，因此新闻列表必须采用动态加载的方式，默认先显示最新的10条，每次点击列表下方的蓝色按钮加载后10条新闻。

页面中间右半部分用于显示一些实用工具，包括新闻搜索栏，热门新闻标签栏，股票市场行情，日历实时本地天气预报等。

主页最下方显示网站的介绍，热门标签，以及相关链接。

分类新闻列表（科技新闻为例）：

页面上方首先是两个公共的导航栏，下面是与主页的滚动条类似的新闻图片滚动条。

页面中间左半部分为科技新闻列表，使用与主页的新闻列表相同的模板。

右半部分一栏用于显示一些社交按钮，相关新闻（只有点进某条新闻时才会有相关新闻），最近热门新闻等。

页面上方首先是两个公共的导航栏，新闻图片滚动条。

页面中间左半部分为搜索结果，由于搜索结果实现动态加载较为麻烦，因此直接显示出所有标题或摘要包含了搜索关键字的结果的前20条。

右半部分一栏用于显示一些社交按钮，相关新闻（只有点进某条新闻时才会有相关新闻），最近热门新闻等。

页面上方仍然是公共的导航栏。

中间的左边部分为新闻内容，包括新闻标题、来源、时间、关键字列表等。此外，还提供了一个收藏按钮以供用户收藏该新闻，用户的收藏情况将会用于个性化推荐的主要依据。

下面为新闻的图文内容。图文内容下方为分享到其他平台的功能按钮以及新闻评论。对于已登录的用户，输入完评论内容后，点击发表按钮即可显示在下方。

中间的右半部分用于显示本新闻的相关新闻、热门新闻等。

## 3.3 网站后端

网站后端主要用于给客户端提供访问数据库的接口，在本项目中使用Node.js Express框架实现。

## 3.3.1 主要接口

- 按分类或关键字获取新闻列表

    URL：/news/list

    Method：GET

    Format：application/json

    | 参数     | 类型       |       说明   |

    |:---------:|:----------:|:--------------------:|

    |genre| String| 新闻类别，当值为personal时获取当前用户的个性推荐新闻|

    |tag| String    | 新闻关键字，仅当未指定genre时生效|

    |html| Int  | 值为1时返回格式为HTML，0则返回JSON|

    |limit| Int | 获取条数上限|

    |offset| Int| 偏移量，即从第offset条之后开始获取（以便动态加载）|

- 按新闻id获取某条新闻内容

    URL：/news/content

    Method：GET

    Format：text/html

    | 参数     | 类型       |       说明   |

    |:---------:|:----------:|:--------------------:|

    |id| String| 新闻id|

- 按新闻id获取新闻评论

    URL：/news/comment

    Method：GET

    Format：text/html

    | 参数     | 类型       |       说明   |

    |:---------:|:----------:|:--------------------:|

    |id| String| 新闻id|

- 获取热门新闻标签

    URL：/news/tags

    Method：GET

    | 参数     | 类型       |       说明   |

    |:---------:|:----------:|:--------------------:|

- 按新闻id获取相关新闻

    URL：/news/related

    Method：GET

    Format：text/html

    | 参数     | 类型       |       说明   |

    |:---------:|:----------:|:--------------------:|

    |id| String| 新闻id|

- 用户注册(发送认证邮件)

    URL：/users/register

    Method：POST

    Format：text/html

    | 参数     | 类型       |       说明   |

    |:---------:|:----------:|:--------------------:|

    |email| String| 邮箱|

    |password| String   | 密码|

    |fullname| String| 用户名|

- 用户登录

    URL：/users/login

    Method：POST

    Format：text/html

    | 参数     | 类型       |       说明   |

    |:---------:|:----------:|:--------------------:|

    |email| String| 邮箱|

    |password| String   | 密码

- 用户收藏新闻

    URL：/users/like

    Method：GET

    Format：text/html

    | 参数     | 类型       |       说明   |

    |:---------:|:----------:|:--------------------:|

    |id| String| 新闻id|

## 3.3.2 邮箱认证

邮箱认证主要是为了能够认证用户注册所使用的邮箱是否是其本人的合法邮箱。本模块中主要是用Node.js email-verification模块进行认证。

首先配置一个用于发送认证邮件的邮箱。为了集中服务器的配置信息，将所有的配置统一放在服务器根目录下的config.json文件中：

{

"host": “127.0.0.1”",

"port": 3000,

"connect": "mongodb://123.206.106.195:27017/newslist",

"newscol": "news",

"usercol": "userdata",

"mailoptions": {

"service": "126",

"email": "newsfeedregister@126.com",

"password": "newsfeed2017"

},

}

编写脚本配置email-verification模块，包括认证邮件的标题、正文格式，邮件激活链接格式，发送邮箱的地址、密码等等：

nev.configure({

verificationURL: 'http://' + config["host"] + ':' + config["port"] + '/users/email-verification/${URL}',

persistentUserModel: User,

tempUserCollection: 'newsfeed\_tempusers',

transportOptions: {

service: mailOption["service"],

auth: {

user: mailOption["email"],

pass: mailOption["password"]

}

},

verifyMailOptions: {

from: 'Do Not Reply ' + mailOption["email"],

subject: 'Please confirm account for NewsFeed',

html: 'Click the following link to confirm your account:</p><p>${URL}</p>',

text: 'Please confirm your account by clicking the following link: ${URL}'

},

confirmMailOptions: {

from: 'Do Not Reply ' + mailOption["email"],

subject: 'Account register successfully',

html: '<p>Successful</p>',

text: 'Your account has been registered successfully'

},

hashingFunction: myHasher

}, function (err, options) {

if( err )

console.log(err)

})

当用户注册时，需要创建一个临时用户并保存到数据库中（默认设置为1天过期），直到用户点击认证链接时才将临时用户转为正式用户。通过MongoDB的ORM模块mongoose，定义用户实体在数据库中的逻辑字段以及密码加密方式(使用bcrypt模块,进行加密，使用的算法是[Blowfish加密算法](https://zh.wikipedia.org/wiki/Blowfish\_(%E5%AF%86%E7%A0%81%E5%AD%A6)))：

```js

var userSchema = mongoose.Schema({

email: String,

password: String,

fullname: String,

interest: Array

});

userSchema.methods.validPassword = function (pw) {

return bcrypt.compareSync(pw, this.password);

};

最后，实现当用户成功点击邮件中的认证链接时，将临时用户转为正式用户的处理逻辑：

router.get('/email-verification/:url', function (req, res) {

let url = req.params.url;

nev.confirmTempUser(url, function (err, user) {

if( err ){

console.log(err)

}else {

nev.sendConfirmationEmail(user.email, function (err, info) {

if( err ){

res.render("info", {

title: "Notice",

message: "Sending confirmation email failed"

})

}else { // email verification successful

// res.cookie('user', new Buffer(user.email).toString('base64')) // automatically login

req.session.user = user

res.redirect('/') // redirect to home page

}

})

}

})

})

**3.3.3 用户状态维护**

cookie 虽然用起来很方便，但是有一个很大的弊端，即cookie所有数据在客户端就可以被修改，数据非常容易被伪造，而且如果 cookie 中数据字段太多会影响传输效率。为了解决这些问题，就产生了 session，session 中的数据是保留在服务器端的，因此网站使用比cookie更安全的session保存用户状态，node中也提供了非常方便的express-session模块来读写session。

网站接入session非常方便，只需要在app.js文件中配置express-session选项：

```js

app.use(session({

secret: randomstring.generate({

length: 128,

charset: 'alphabetic'

}),

cookie: {

maxAge: 60000\*1000

},

resave: true,

saveUninitialized: true

}));

```

之后便能够通过访问每个express请求中session字段即可访问session对象。注意到由于session仍然需要在浏览器端存储一个随机生成的字符串session-id，因此需要指定cookie的有效时间，目前默认设置为100分钟，即用户登录100分钟后需要重新登录。

## 3.3.4 redis缓存

为了在用户量增大时减轻数据库的请求负载，对于某些相对静态的数据使用基于键值的redis数据库进行缓存。加入redis缓存后对于某一类型的查询操作流程变为：

通过查询请求字段构造Key

if ( 通过Key查询数据是否已在redis缓存中 ){

    直接使用缓存中的数据构造回复包

}else{

    使用MongoDB获取所需数据

    将数据通过Key写入redis缓存中

    构造回复包

}

首先仍然需要在服务器配置文件config.json中配置redis服务器的主机地址与服务端口：

```json

"redis": {

"port": 6379,

"host": "123.206.106.195"

},

构造key并查询缓存中的数据：

```js

let key = JSON.stringify({

tag: tag,

genre: genre,

offset: offset,

limit: limit

})

redisClient.get(key, function (err, reply) {

if( err ){

console.log(err)

res.json({ message: "redis error"})

}else if( reply ){

console.log("redis hit")

result = JSON.parse(reply);

renderResult(result)

}else{

News.getList(genre, tag, (err, result) => {

if( err ){

console.log(err);

res.json();

}else{ // success

cache(key, JSON.stringify(result))

console.log('result.length = ' + result.length)

renderResult(result)

}

}, offset, limit)

}

在多个请求的处理中都需要写入缓存，因此单独封装一个函数用于将某个键值对写入缓存，并根据参数设置是否过期：

function cache(key, value, notExpire, time) {

if( notExpire )

return redisClient.set(key, value)

else

return redisClient.set(key, value, expireFlag, time ? time : expireTime)

}

# 四. 功能测试

Web端和移动端页面的效果图片都已经在前面的章节中给出，这里主要展示相关新闻和个性推荐的结果。

## 4.1 相关新闻测试

相关新闻推荐结果1（右边栏中的Related News）：

![enter image description here](https://i.imgur.com/kmQb3yg.jpg)

相关新闻推荐结果2（右边栏中的Related News）：

![enter image description here](https://i.imgur.com/2yfBTn6.png)

可以看到相关新闻的推荐准确率还是比较高的。

## 4.2 个性推荐测试

以下是使用新注册的账号关注的6条新闻，可以看到关注的主题为韩国、城市、中国公民、政党选举等：

![enter image description here](https://i.imgur.com/dWKJDyz.png)

![enter image description here](https://i.imgur.com/EPDWHIX.png)

下面是针对该用户所推荐的新闻：

![enter image description here](https://i.imgur.com/zHuCCii.jpg)

可以看到推荐的新闻主题也包含了韩国、城市、中国公民等关键词（虽然最后还是混入了一条体育新闻）。

# 五. 项目感想

在整个项目的实现过程中，我觉得最有意思的工作在于新闻关键词聚类。通过Google的开源项目Word2Vec等框架，能够将维基百科上的文章集提取出一个个中文词语，训练出的模型竟然能够找到一个词的近义词，能够判断两个词是否有关联。但在训练模型的过程中也遇到了许多问题。由于整个项目都是跑在租用的云服务器上的，训练模型自然也想到在服务器上跑。但训练的过程中前几次都出现Python进程自动结束的情况，无法完整地跑完脚本。经过多次调试后发现在脚本开始运行后内存占用直线上升，是因为需要把整个文本文件都读进内存后处理，但文件过大服务器内存不足，因此操作系统自动把Python进程中断了。升级服务器内存费用比较高，因此最后只能先在自己的电脑上跑完脚本训练出Word2Vec的模型之后再上传到服务器使用了。虽然对这些过程背后的算法还不甚了解，但我也对自然语言处理、深度学习等热门领域产生了一定的兴趣，今后如果有时间希望能够进一步地、更系统地学习这些知识。

此外，微软认知服务提供的建议API也十分有趣。通过官方文档中提供的数据集建立好模型后，可以看到某个用户的浏览记录确实能够反映在推荐的商品之中。在应用到本项目中时，由于缺少用户使用记录，因此需要编写相应的脚本生成某个用户的一些记录上传到API的服务器上、但是由于这些记录是随机生成的，因此特征不是特别明显，因此最后主要还是用的根据用户已浏览新闻，使用之前提到的算法找到这些新闻的相关新闻推荐给用户。

通过该项目的实践，我们不仅对网页爬取、NoSQL数据库、网站缓存、响应式页面、消息推送等常用技术有了一定的了解，同时也有机会能够在自然语言处理、推荐系统等前沿研究领域进行了一些入门级的实践，因此我觉得本项目应该算是一次非常有意义的课程作业。

# 六. 参考资料

[Node.js Express API文档](https://expressjs.com/api.html)

[“结巴”中文分词：做最好的 Python 中文分词组件](https://github.com/fxsjy/jieba)

[中英文维基百科语料上的word2vec实验](http://www.52nlp.cn/%E4%B8%AD%E8%8B%B1%E6%96%87%E7%BB%B4%E5%9F%BA%E7%99%BE%E7%A7%91%E8%AF%AD%E6%96%99%E4%B8%8A%E7%9A%84word2vec%E5%AE%9E%E9%AA%8C)

[微软认知服务-建议API文档](https://azure.microsoft.com/zh-cn/services/cognitive-services/recommendations/)

[第三方推送服务OneSignal官方SDK文档](https://documentation.onesignal.com/docs)