双意图识别系统设计文档

[一 文档简介 2](#_Toc14684287)

[二 开发环境及架构 2](#_Toc14684288)

[三 业务流程图 2](#_Toc14684289)

[四 算法实现 4](#_Toc14684290)

[1 句型（句子个数）判断总控及返回数据封装 4](#_Toc14684291)

[2 （单句）词、词组并列 4](#_Toc14684292)

[3 一般并列句拆分重组实现 5](#_Toc14684293)

[4 一般并列句与句内双意图混合句型 6](#_Toc14684294)

[五 项目部署、启动、停止以及更新 6](#_Toc14684295)

[1 部署环境 6](#_Toc14684296)

[2 组件介绍 7](#_Toc14684297)

[2.1 HowNet解析器相关组件 7](#_Toc14684298)

[2.2 关于句型-策略关系对照表和句型标准化表 8](#_Toc14684299)

[2.2.1 存放位置 8](#_Toc14684300)

[2.2.2 文件格式及读取方式 8](#_Toc14684301)

[2.2.3 文件内容 8](#_Toc14684302)

[2.2.3.1 句型-策略关系对照表 8](#_Toc14684303)

[2.2.3.2 句型标准化表 9](#_Toc14684304)

[2.2.3.3 关于这里为何不使用数据库的回答 9](#_Toc14684305)

[2.3 Java应用相关组件 9](#_Toc14684306)

[2.4 Log文件夹 9](#_Toc14684307)

[3 部署一套HowNet服务 10](#_Toc14684308)

[3.1 应用组件放置 10](#_Toc14684309)

[4 项目启动与停止 12](#_Toc14684310)

[5 项目文件更新 13](#_Toc14684311)

[六 API调用文档 14](#_Toc14684312)

[1 接口功能说明 14](#_Toc14684313)

[2 接口调用说明 14](#_Toc14684314)

[七 关于JNA调用动态库 16](#_Toc14684315)

[1 准备条件 16](#_Toc14684316)

[1.1 C++工程师所提供的接口文档 17](#_Toc14684317)

[2 Java代码编写 17](#_Toc14684318)

[2.1 创建一个调用动态库的工具类，详细代码如下 17](#_Toc14684319)

[2.2 编写一个测试类 18](#_Toc14684320)

[八 项目启动常见异常信息 20](#_Toc14684321)

[1 JDK（JRE）版本问题 20](#_Toc14684322)

[2 HowNet路径问题 20](#_Toc14684323)

[3 端口被占用 21](#_Toc14684324)

# 文档简介

本文档旨在介绍双意图识别系统的实现思路、整体架构、逻辑流程以及涉及到的相关技术。

# 开发环境及架构

**开发语言：**Java **版本号：**JDK8（32位，目前HowNet只支持32位）

**技术架构：**SpringBoot **版本号：**2.1.4.RELEASE

**项目形式：**Maven Project **打包形式:** war包

**项目分层：**分两层：Controller层（控制转发层）和Service层（业务逻辑层）

# 业务流程图

程序业务流程如下图：

|  |
| --- |
|  |

# 算法实现

算法核心方面，就是使用JNA（Java Native Access）工具包，调用HowNet动态库进行预处理和解析（关于如何使用JNA调用动态库，下面会有专门的模块儿进行讲解）。其他方面，主要是通过对解析结果的遍历和匹配，来判断该执行什么算法。目前算法主要分以下四部分：

## 句型（句子个数）判断总控及返回数据封装

这一部分主要功能是HowNet预处理和解析器解析，并根据解析器返回的结果（对象数组形式）进行判断。一般情况下都是单句、或双句（数组长度为1或2），但不排除用户输入三句或三句以上（数组长度>=3）的情况，根据以上三种情况来决定接下来该进入哪个模块儿。

除了做判断总控，该模块儿还对数据进行封装，因为以下算法模块儿返回的是字符串数组，这里需要将问句ID以及原句一并封装，作为个固定对象进行返回。

## （单句）词、词组并列

若句型判断是单句，只能默认为词、词组并列句型，则程序执行此算法模块儿。至于是否确实是词、词组并列型句子，则需要根据解析器解析结果来进一步判断。如果是则根据既定标签进行拆分重组，若不是，则按混淆句型做拒识处理。（流程请参照上文流程图）

至于此类句型如何拆分，思路是：解析器给核心词、词组打上标签，程序根据标签去将核心词一个个找到提取出来，并判断是单个词还是词组，并记录该词在句中位置，将其余非核心词汇罗列并将之前提取出来的词插入到原有的位置，有一个词便是一个句子，有n个词便是n个句子。例如：转账有没有手续费和限额。

*首先，红色核心词将被打上标签，根据标签将“手续费”和“限额分别”提取出来，并记录其在原句中位置。那么再根据其余非标签词汇进行句子重组，结果为：*

*转账有没有手续费。*

*转账有没有限额。*

除此之外，该算法模块儿还对并列词组进行更精确的处理。如果该句为并列词组类型，那要判断词组中是否含有相同的分词成分，比如“还款手续费”和“转账手续费”，他们有共同的分词成分“手续费”，那么这才确定该句是真正的并列词组，否则拒识。

该模块儿最新还开发出了“2乘2特例”算法，是针对单句中前后存在两处词、词组并列的情况。例如：“这个基金买入和卖出不需要手续费或者交易费的。”，又例如：“网上银行或手机银行跨行、异地转账有免费的吗？”。以上两句前边有“A和B”的并列，后面有“C和D”的并列，这样就形成“2乘2”的格式。那么这部分处理时，先分别判断“AB”和“CD”是单词还是词组，如果是词组还要判断他们是否有共同成分，最终确定怎么处理，处理策略就是“2乘2”的形式，最终返回四句话。

## 一般并列句拆分重组实现

对于一般并列句，目前处理的目标主要是有且只有两句的句型。即在句型判断模块儿判断结果为两句的，程序执行该算法模块儿。那么该模块儿算法要先对两个句子的解析结果进行遍历，并抽取相关的句型判断标签，取到标签之后，去我们配置的《句型-策略关系映射表》中找到对应的策略，然后根据策略执行相应的代码。

例如：8192理财现在利率大约多少？8193呢？

解析结果如下图：

|  |
| --- |
| C:\Users\Hunter\AppData\Local\Temp\1562763196(1).jpg |

|  |
| --- |
| C:\Users\Hunter\AppData\Local\Temp\1562763224(1).jpg |

通过遍历其解析结果，后一句得到的句型标签是“疑问\_单要素+呢”，前一句得到的句型标签是“疑问\_数量”。[[1]](#footnote-1)那么将这种组合去《句型-策略关系映射表》中匹配，如下图：

根据后项为“疑问\_单要素+呢”，前项为“疑问\_数量”，我们找到策略“replace\_whole”。然后根据该策略去对应算法中的“replace\_whole”处理算法进行句子拆分重组。

|  |
| --- |
| C:\Users\Hunter\AppData\Local\Temp\1562763833(1).jpg |

至于如何进行拆分重组，由于策略太多，根据特殊情况而定，详情请直接看源码实现，这里不在一一赘述。关于《句型-策略关系映射表》，在下文中会有详细介绍。

## 一般并列句与句内双意图混合句型

该算法模块儿是建立在第二种模块儿（一般并列句）的基础之上的。当用户请求原句为两句的句型时，先进入一般并列句的算法（下图中第一行的“getSentenceStrategy”方法），根据解析器的解析结果，各自判断两句中key\_1中的标签是否包含“结构\_句内双意图”。如果只有第一句是“句内双意图”，则先进行一般并列句的拆分重组（成两句），这两句再次调用解析器解析，并调用并列词、词组模块儿算法进行拆分重组（共四句），并将结果返回。由于第二句中包含“结构\_句内双意图”的情况过于复杂，现在还没有能力处理，所以凡是原句第二句中包含双意图的全部拒识。

在该算法模块儿内部，还存在一个关于“标记\_跨句双意图拒识”标签的判断，该标签是由HowNet规则打上的，java算法中只做判断，如果存在就直接拒识。该部分代码如下图所示：

|  |
| --- |
|  |

# 项目部署、启动、停止以及更新

## 部署环境

1. 64位Windows系统（HowNet目前只支持Windows环境）
2. Microsoft Visual C++运行环境
3. Jdk 1.8.x版本（必须为32位）

硬件环境则根据部署的服务数量来定

## 组件介绍

如下图，一个“service”文件夹，就是一套完整的双意图应用。“doubleIntentionDll”目录是解析器动态库相关组件，“javaProject”目录是java应用存放目录，“log”目录是项目启动后自动生成的存放日志文件的文件夹。

|  |
| --- |
|  |

### HowNet解析器相关组件

“doubleIntentionDll”目录下，是所有与HowNet解析器相关的动态库、词典、规则文件、句型-策略关系对照文件等等。如下图，上线之后的维护与更新，将以此文件夹下的内容为主要更新对象，更新方式以替换为主。

|  |
| --- |
|  |

### 关于句型-策略关系对照表和句型标准化表

#### 存放位置

如上图，为了方便项目配置、运维及更新，我们将句型-策略关系对照表和句型标准化表与HowNet动态库放在了一起。这样在每次更新规则文件时，能够一并进行更新。

#### 文件格式及读取方式

两个文件的格式均为TXT文件。在Java应用启动时，随着动态库一起加载（加载路径与动态库一致），并存放于内存中，以供持续使用，直至项目停止。

#### 文件内容

##### 句型-策略关系对照表

句型-策略关系对照表是针对 “一般并列句型”算法模块儿中所包含的7中策略而形成的，它的主要内容是两句并列句各自的句型标签以及所对应的句子拆分重组策略。格式如下图所示，每列之间用“\t”（也就是一个Tab）隔开（这是标准，也方便直接从Excel表格中复制），第一列为并列句第二句的句型标签，第二列为第一句的举行标签（在算法中我们提到，一般并列句的处理策略就是“根据后项找前项”，所以应该把第二句的标签放前面，方便取值），第三列为对应处理策略，至于每个策略的详细解释及示例，由我们的语言分析师提供，这里不再赘述。

|  |
| --- |
|  |

##### 句型标准化表

至于句型标准化表，主要是针对“一般并列句型”算法模块儿中的“append\_normalization”策略的。由于用户的一些表达并不是标准的，比如“朝朝盈最高存到五万元吗？如何存入更多？”，其实这句话就是想表达“限额”，所以本表就提供这么一个标准化的表，来对一些不规范表达做一个标准化处理。内容及格式如下图（同样的，两列之间是以一个Tab隔开），目前只有一行：

|  |
| --- |
|  |

##### 关于这里为何不使用数据库的回答

其一，两张表数据量不大，没必要；

其二，两张表只读一次，之后重复使用，更显出了数据库的多余；

其三，一般更新这两张表的时候，都要更新规则文件，都要重启！索性一次性更新了，运维方便！而不用去操作数据库。

### Java应用相关组件

“javaProject”文件夹下，存放了Java应用jar包。

|  |
| --- |
|  |

### Log文件夹

Log文件夹是由项目启动之后自动生成的（由java应用中resources下的“logback-spring.xml”），与doubleIntentionDll和javaProject文件夹同级目录。内部是一个由项目名称命名的文件夹，如下图：

|  |
| --- |
|  |

进入文件夹，是系统日志信息，分两个级别：error和info级，当天之前的日志则自动压缩保存，并定期删除。如下图：

|  |
| --- |
|  |

## 部署一套HowNet服务

关于JDK的安装以及环境变量的配置，这里不再赘述。

### 应用组件放置

1. 为方便日后扩展，建议按照上文的文件目录结构，新建一个文件夹（例如service01），作为应用的总文件夹，分别复制一份HowNet动态库到doubleIntentionDll文件夹下、javaProject，如下图：

|  |
| --- |
|  |

1. 接下来进行HowNet动态库路径配置。如下图，进入javaProject文件夹，选中java项目包，右键用压缩工具打开。

|  |
| --- |
|  |

1. 打开之后，进入“\WEB-INF\classes\”目录下，找到“application.properties”文件，双击打开（建议打开方式用Notepad或同功能编辑工具，而不用记事本）。

|  |
| --- |
|  |

1. 如下图，打开编辑后，找到“hownetPathName”一行，将后面的路径改成上面HowNet动态库所在的目录，并保存至压缩文件。注意：“\”一定要转义为“\\”，以确保应用正确读取路径。其他配置项分别是：应用端口号，日志级别和日志文件夹名称（即自动生成的log文件夹下，日志文件夹的名称。）

|  |
| --- |
|  |

|  |
| --- |
|  |

更改完毕后保存，看到上图的提示并点击确定，才是保存成功。自此，单套应用部署完成。

## 项目启动与停止

1. 回到javaProject目录下，在当前目录下打开CMD命令行窗口，输入java -jar ‘项目名称’，如“java -jar yuzhi-doubleIntention-0.0.1-SNAPSHOT.war”。然后命令行将开打印项目启动的信息。如下图：

|  |
| --- |
|  |

1. 若Tomcat黑窗口中正常打印了如下信息，并且没有抛出异常信息，则说明Tomcat正常启动。

|  |
| --- |
|  |

## 项目文件更新

至于更新，Java项目不用多说，无非是war包的替换，配置文件的修改，跟部署一样。

主要是HowNet相关组件的更新。动态库正常情况下是不会随便更新的，更新周期可达一年甚至更长。主要是更新词典、规则文件和上文提到的两个TXT格式的表（句型-策略对照表和标准化表）。更新方式就是替换，但要注意的是，替换之后一定要重启！否则是没有效果的。其中，词典和两个TXT表（句型-策略对照表和标准化表）是和动态库目录一级的，而规则文件是在动态库文目录中的一个名为“data”的子文件夹下的。如下图：

|  |
| --- |
|  |

# API调用文档

## 接口功能说明

本系统旨在处理用户请求问句中的多个意图的识别与处理。目前系统支持以下几种类型问句的识别：

1. 单问句中存在多个（一般是两个）并列意图的；
2. 双问句中前后问句存在固定句型搭配策略的；
3. 在第2条基础上，双问句中第一句内部存在多个（一般是两个）并列意图的；

除以上所列类型外，其他类型均按拒识处理。

## 接口调用说明

**接口请求URI：**

http://55.14.1.133:8080/doubleIntentionRequest

**接口请求方式：**

HTTP Post请求

**接口请求参数：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数名称** | **参数介绍** | **参数类型** | **参数示例** | **备注** |
| questionID | 问句唯一标识 | 字符串（String） | 8a98fd18-847e-48f6 | 必填参数  长度不限 |
| requestQuestion | 问句内容 | 字符串（String） | 现在不是取消了账户管理费吗？你们怎么还收取呢？ | 必填参数  正常问句  即可 |

**参数传递形式：**

**Content-Type：**application/json;charset=UTF-8

**Parameter：**

|  |
| --- |
| {  “questionID” : QUESTIONID , //问句请求唯一标识  “requestQuestion” : YOUR QUESTION //请求问句内容  } |

**接口返回结果：**

**返回结果样式：**

|  |
| --- |
| {  “questionID” : QUESTIONID，//问句请求唯一标识（与请求参数中一致）  “orignalQuestion”: YOUR QUESTION, //原请求问句（与请求参数中一致）  “status”：200 / 505，//响应状态（200代表成功，505标识拒识）  “data” : QUESTION PROCCESS RESULT, //响应数据结果（若成功，则返回处理结果； 若拒识，则返回null）  “msg” : RESPONSE MESSAGE//响应信息（若成功，则显示“SUCCESS:策略名称”， 若拒识，则显示拒识原因）  } |

**返回结果示例：**

**成功示例：**

|  |
| --- |
| {  “questionID” : “8a98fd18-847e-48f6”  “orignalQuestion”: “8192理财现在利率大约多少？8193呢？”,  “status”：200，  “data” :[  “8192理财现在利率大约多少？”，  “8193理财现在利率大约多少？”  ],  “msg” : “success:并列句”  } |

**拒识示例：**

|  |
| --- |
| {  “questionID” : “8a98fd18-847e-48f6”  “orignalQuestion”: “登录网银在哪啊？？想打我家的对账单和回单”,  “status”：505，  “data” : null,  “msg” : “并列句：标记\_跨距双意图拒识，暂不处理！”  } |

# 关于JNA调用动态库

## 准备条件

新建java项目并导入调用动态库的Jar包：JNative.jar和jna.jar

|  |
| --- |
|  |

如果是maven项目，则引入如下依赖

|  |
| --- |
|  |

C++工程师那边所提供的动态库一套（即上边频繁提到的HowNet动态库）

|  |
| --- |
|  |

### C++工程师所提供的接口文档

这里需要为你提供接口（动态库）的工程师给你一个接口列表。这里我以解析器（chinese\_parser）接口为例：

动态库初始化接口 🡺🡺🡺 const bool initHowNet(char\* ApPath);

解析器解析主接口 🡺🡺🡺 const bool parseText(char\* ApPath);

获取解析结果接口 🡺🡺🡺 const char\* getJsonRet();

这里需要对C++做一些简单的了解，或者直接询问他以上方法的每个关键字什么意思。

## Java代码编写

接下来进入正题，编写调用动态库的编码。

### 创建一个调用动态库的工具类，详细代码如下

|  |
| --- |
| **package** com.yuzhi.util;  **import** com.sun.jna.Native;  **import** com.sun.jna.win32.StdCallLibrary;  **import** com.yuzhi.entity.ParamConstant;  /\*\*  \* 这是一个调用动态库的工具类  \* **@author** Hunter  \*  \*/  **public** **class** DllUtil {  //编写一个interface接口，名字随便起  //让此接口继承com.sun.jna.win32.StdCallLibrary接口  **public** **interface** EventAbstract **extends** StdCallLibrary {  //将c++提供的所有接口，用Java代码实现一下    //HowNet初始化接口，对应c++接口列表里的 const bool initHowNet(char\* ApPath);  **public** **boolean** initHowNet(String Apath);  //解析接口，对应c++接口列表里的 const bool parseText(char\* ApPath);  **public** **boolean** parseText(String input);  //获取解析结果接口，对应c++接口列表里的 const char\* getJsonRet();  **public** String getJsonRet();  }    //声明一个接口变量  **private** **static** EventAbstract *dllKernel*;    //创建接口实例的方法，相当于单例模式下的 getInstance()方法  **public** **static** EventAbstract getDllKernel(String hownetPath) {  //加载本地dll接口  **if** (*dllKernel*==**null**) {//如果变量不为空，说明已经加载，则直接返回  //com.sun.jna.Native类有一个loadLibrary()方法，通过反射的方式来加载动态库，此处的“hownetPath”就是我们之前部署的时候所配置的HowNet路径，即动态库的路径  *dllKernel* = (EventAbstract) Native.*loadLibrary*(hownetPath +"HowNet\_ChineseParser\_Helper", EventAbstract.**class**);  }  //返回实例  **return** *dllKernel*;  }    } |

### 编写一个测试类

|  |
| --- |
| package com.yuzhi.test;  import com.yuzhi.entity.ParamConstant;  import com.yuzhi.util.DllUtil;  import com.yuzhi.util.DllUtil.EventAbstract;  /\*\*  \* 动态库调用测试类  \* @author Hunter  \*  \*/  public class DllUtilTest {  //定义HowNet时间抽取接口  private static EventAbstract eventAbstract;  public static void main(String[] args) {  //调用工具类中加载动态库的方法  eventAbstract = DllUtil.getDllKernel();  //加载完之后初始化，此处的“hownetPath”就是我们之前在项目部署的时候，在application.properties文件中所配置的HowNet的路径，即动态库的路径  boolean init = eventAbstract.initHowNet(hownetPath);  //初始化成功之后即可调用  if(init){  //解析  boolean parseText = eventAbstract.parseText("百度公司今年利润出现大幅下滑，为此，深交所向公司发出了问询函。");  //如果返回true,说明解析成功，获取解析结果  if(parseText){  String jsonRet = eventAbstract.getJsonRet();  //打印结果到控制台  System.out.println("解析结果》》》"+jsonRet);  }  }else{  System.out.println("初始化失败！！");  }  }  } |

控制台打印结果

|  |
| --- |
| 解析结果》》》{"sentence":[此处省略。。。]} |

自此，调用动态库成功。

# 项目启动常见异常信息

以下列几个常见的项目启动异常信息：

## JDK（JRE）版本问题

如果在项目启动时，抛出此异常，则说明你的JDK版本不是32位的

|  |
| --- |
|  |

## HowNet路径问题

当项目启动时抛出以下异常时，说明你配置的HowNet路径有问题

|  |
| --- |
|  |

## 端口被占用

如果项目启动抛出以下异常，则说明您配置的tomcat端口号被占用了，或者是项目重复启动了。

|  |
| --- |
|  |

1. 我们的判断策略就是根据第二句去寻找第一·句，故这里先看后一句的句型标签，再看前一句的。 [↑](#footnote-ref-1)