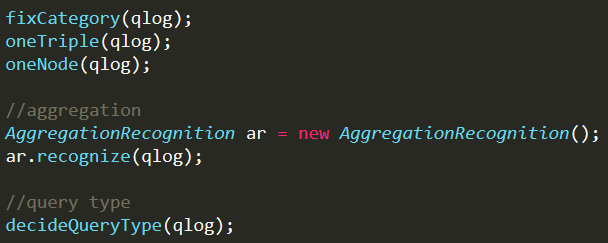
# Package addition(功能函数类)

## AddionalFix（输入：qlog，输出）

主要函数功能有：

### fixCategoty(SemanticUnit，centerWord，Triple)

先判断SemanticUnit中的centerWord.mayCategory是否为真，也就是判断是不是中心词，如果是中心词那么生成变量

? centerWord.originalForm。之后在qlog的RankedSparqls里面对每一个sparql的每一个triple进行重新判断和类型判定。重新赋值。

### oneTriple

首先得知这个sentenceType是special的。之后还要限制qlog.semanticUnitList.size() == 2。

主要构造的是“wh-words be(am is are) entity”这类型的sparql语句。这类语句只有一个triple。

### oneNode(qlog.target, qlog.s.words)

1. ***!SentenceType.GeneralQuestion***

关键之处在于Qlog.target

1. target.mayType && target.tmList
2. target.mayEnt && target.emList
3. target.mayCategory && target.category
4. ***SentenceType.GeneralQuestion***
5. target.mayEnt && target.emList(复杂)
6. target.mayType && target.tmList

严重怀疑，猜测。Target.pmlist指的是predicate mention emlist指的是entity mention

Pmlist来源于：Globals.pd.nlPattern\_2\_predicateList

target.position

整体来说这个类主要是在做具体的sparql成型，修改。

## AggregationRecognition

### translateNumbers(str)

可以将1~100之间的字符串转化为数字形式。

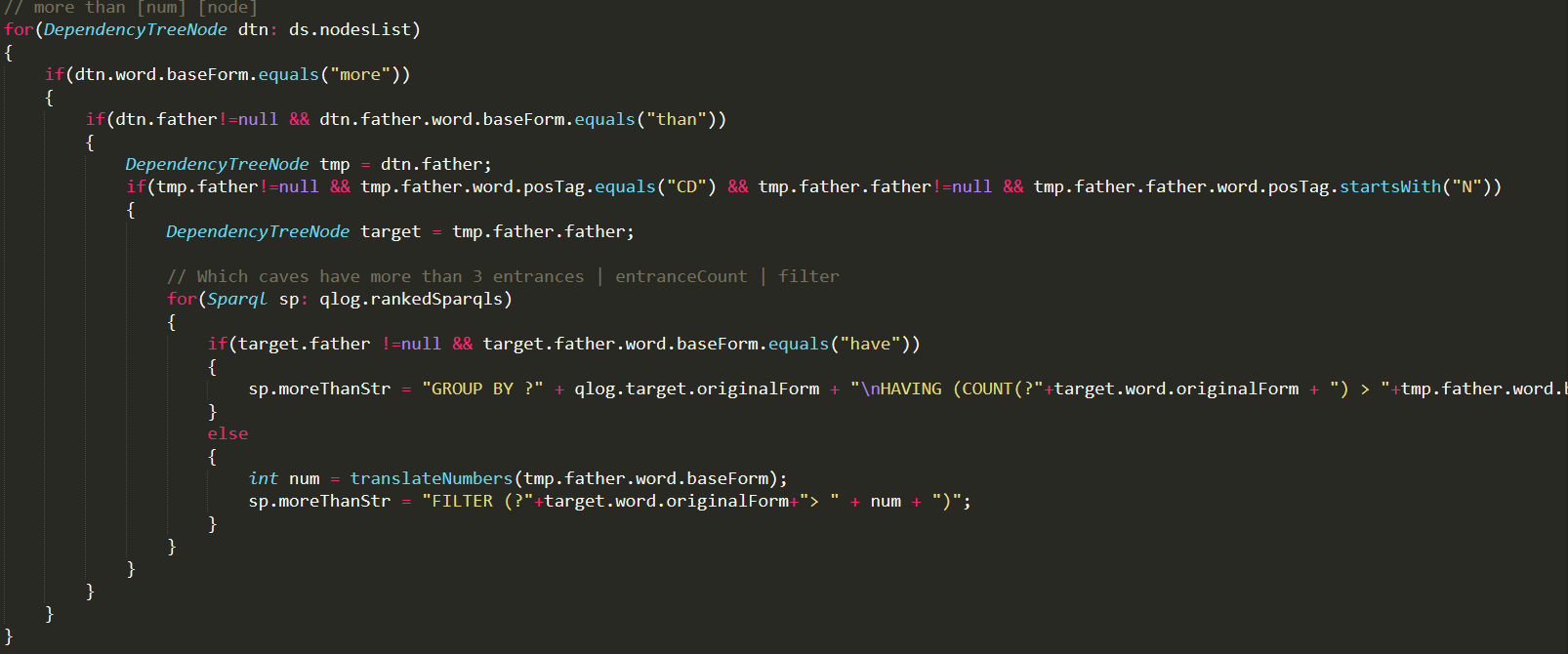
### recognize(QueryLogger qlog) very important

Globals.pd.getPredicateById(triple.predicateID)：获取谓词

遍历denpendency tree，获取结点的数据之类的。

Very important

if(dtn.word.baseForm.equals("more"))



总体来说处理和数量相关的问句，主要包括how many, how much ,how often, more than, most

# 2. fgmt

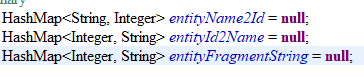
## 2.1 EntityFragment（数据类，含有大量数据）

### 2.1.1 getCandEntityNames2(String phrase)

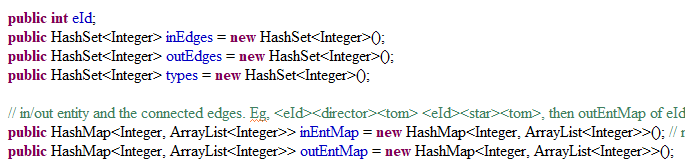
目的是返回一个实体的编号和分。调用了getCandEntityNames\_subject

### 2.1.2 getEntityMappingList (String n)

调用上面的2.1.1，构造entityMapping这个对象。对象里面有entity的ID，有String，有评分。

重要：该文件主要涉及的文件内容是16entity\_id.txt与16entity\_fragment.txt。EntityFragmentFields这个类主要负责储存这两个文件的信息，主要包含三个信息

该类主要建立了以下数据：



这些数据其实就是针对一个eid来说，也就是一个entity的ID，所有指向这个ID的边，结点，这个entity指出去的结点和边。四个信息，同时给出了这个节点的类型信息。

entityFragmentString中的value是entityFragment中的value信息，其信息大致如下：

11 2042240:5844;,||5844,||

每一个这种的Value都存在四个’|’这种符号作为分隔符。一共会把字符串分成五个部分。最后一部分就是type。其他四个部分分别为入边，入结点，出边，出结点。

## 2.2 RelationFragment



在这个类文件中最重要的就是，*relFragments* 。其中是键值对。键指的是pid(predicate id), 对应的value是RelationFragment。

存在的是入类型和出类型。猜测（数据集中存的并非是实际的实体，而是一些类型。因为存储实体的话数量太多。因此就是数据集中是这些类型的实体全都是）

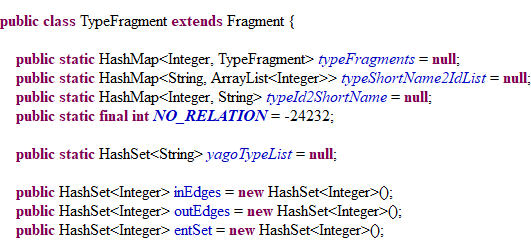
## 2.3 TypeFragment

主要也是读取数据，和type相关的数据。包含type和id的映射以及指向每个type实体的edge以及指出去的edge。

数据为此种形式：

1. 1,2|1,2,3|2,3,4

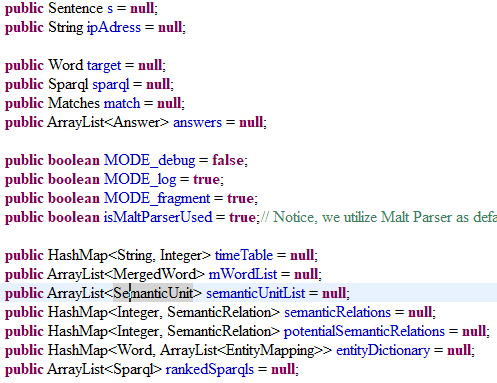
上面的数据中储存的主要是指向这个类型的入边，出边，以及这个类型包含哪些实体。



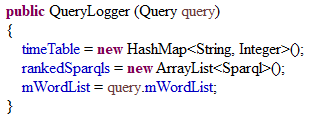
# 3. Log

## 3.1 QueryLogger(数据类)

主要用户储存的数据：



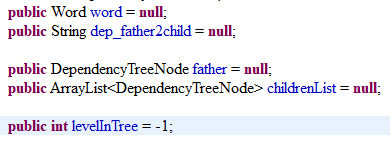
### 3.1.1 QueryLogger (Query query)



# 4. nlp.ds

## 4.1 dependncyTreeNode

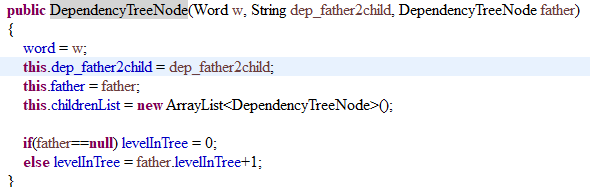
拥有有依存数据以及大量关于操作的函数，数据主要如下：



其中比较难以理解的应该是dep\_father2child。目前猜测为：描述father和child之间的关系。依赖关系

### 4.1.1DependencyTreeNode（构造函数）

主要是为了构造一个结点



### 4.1.2DependencyTreeNode containDependencyWithChildren

这个函数的功能较为简单，判断两个结点间是否有依赖关系。

### 4.1.3 containWordBaseFormInChildren

输入：String wordBaseFormChild

判断child结点中存储的内容中，也就是字符串中是否包含wordBaseFormChild这个字符串。

其中的son是一个dependencyTreeNode。

### 4.2 dependncyTree

也是数据类，会存储依存树的数据。

主要是构造了树。

## 4.3 sentence

## 4.4 word

# 5. paradict

## 5.1 ParaphraseDictionary

这个类的功能很简单，主要是提供了论文中的subject-like与object-likerelation关系，并且与文件会进行交互。

# 6. qa.extract

## 6.1 CorefResolution

## 6.2 EntityRecognition

要弄清楚baseWord和originalWord的区别与联系，进一步比较重要的是搞清楚买个word的baseForm和originalForm

注意：emlist和tmlist中存放的都是具体的RDF数据集中的数据

|  |  |  |  |
| --- | --- | --- | --- |
|  | baseWord | originalWord | allUpperWord |
| search Category | √ |  |  |
| search Type | √ | √ | √ |
| search Entity | √ | √ |  |
| add Category | mustSelectedList.add | | |
| add Literal | literalList.add(key); | | |
| type mapping | typeMappings.put(key, tmList.get(0).typeName); | | |
| add entity mappings |  | | |

## 6.3 modified word

### 6.3.1 supplementTriplesByModifyWord

核心关键数据为word.modifiedWord,与之相关的存储结构是typeVaraibleList。猜测为类型triple pattern中的某一项为变量。

## 6.5 TypeRecognition