关于搜索的原理和特殊操作

1. Timeout机制

**GET /goods/fruit/\_search**

查询结果如下:

{

"took" : 1,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"skipped" : 0,

"failed" : 0

},

"hits" : {

"total" : 10,

"max\_score" : 1.0,

"hits" : [

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "5",

"\_score" : 1.0,

"\_source" : {

"name" : "mihoutao",

"describe" : "suan",

"price" : 45,

"producer" : "xinxilan",

"tags" : [

"lv",

"tian"

]

}

},

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "8",

"\_score" : 1.0,

"\_source" : {

"name" : "xigua",

"describe" : "pianyi",

"price" : 190,

"producer" : "zhongguo",

"tags" : [

"da",

"haochi"

]

}

},

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "10",

"\_score" : 1.0,

"\_source" : {

"name" : "xigua",

"describe" : "pianyi",

"price" : 190,

"producer" : "zhongguo",

"tags" : [

"da",

"haochi"

]

}

},

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "2",

"\_score" : 1.0,

"\_source" : {

"name" : "pingguo",

"describe" : "cui",

"price" : 60,

"producer" : "zhongguo",

"tags" : [

"haokan",

"xiang"

]

}

},

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "4",

"\_score" : 1.0,

"\_source" : {

"name" : "boluo",

"describe" : "getouda",

"price" : 74,

"producer" : "malaxiya",

"tags" : [

"huang",

"youci"

]

}

},

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "6",

"\_score" : 1.0,

"\_source" : {

"name" : "xigua",

"describe" : "haochi",

"price" : 109,

"producer" : "zhongguo",

"tags" : [

"da",

"haochi"

]

}

},

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "WeQXSmgBliuceLt0Thhv",

"\_score" : 1.0,

"\_source" : {

"name" : "pingguo",

"describe" : "haochi",

"price" : 87,

"producer" : "zhongguo",

"tags" : [

"da",

"haochi"

]

}

},

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "1",

"\_score" : 1.0,

"\_source" : {

"name" : "xiangjiao",

"describe" : "haochi tian",

"price" : 40,

"producer" : "feilvbin",

"tags" : [

"xiangjiao",

"haochi"

]

}

},

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "7",

"\_score" : 1.0,

"\_source" : {

"name" : "xigua",

"describe" : "pianyi",

"price" : 190,

"producer" : "zhongguo",

"tags" : [

"da",

"haochi"

]

}

},

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "3",

"\_score" : 1.0,

"\_source" : {

"name" : "lizi",

"describe" : "zide",

"price" : 10,

"producer" : "zhongguo",

"tags" : [

"suan",

"tian"

]

}

}

]

}

}

took：整个搜索请求花费了多少毫秒

hits.total：本次搜索，返回了几条结果

hits.max\_score：本次搜索的所有结果中，最大的相关度分数是多少，每一条document对于search的相关度，越相关，\_score分数越大，排位越靠前

hits.hits：默认查询前10条数据，完整数据，\_score降序排序

**注意:timeout默认为false,也就是不指定超时的情况,如果检索的内容过多,则需要等待检索完成以后才能显示全部的内容,用户体验会不好.可以通过设定timeout的时限来确定当前的检索时间.**

**注意:设定检索时间以后,当前时限到了就必须呈现查询结果,如果有2000w条数据,即使查不全到时间也会显示出已经查询出来的结果**

**GET /goods/fruit/\_search?timeout=10m**

**注意: timeout=10ms，timeout=1s，timeout=1m,必须有时间单位**

1. multi-index和multi-type搜索模式
   1. /\_search

默认检索所有内容:

**GET /\_search**

会把所有索引文档的内容一起检索出来

* 1. /index /\_search

检索同一个index下的所有type内容

**GET /my\_index4 /\_search**

会把当前my\_index4这个索引下的内容全部检索,有指定性

* 1. /index1,index2/\_search

同时搜索两个或者多个index下的数据

检索多个index下的内容

**GET /my\_index4,my\_index5/\_search**

使用通配符方式来检索

**GET /\*dex4,\*index5/\_search**

1. es分页查询原理

按照一般的查询流程来说，如果我想查询前10条数据：

        1)客户端请求发给某个节点

        2)节点转发给个个分片，查询每个分片上的前10条

        3)结果返回给节点，整合数据，提取前10条

        4)返回给请求客户端

    那么当我想要查询第10条到第20条的数据该怎么办呢？这个时候就用到分页查询了。

    在ElasticSearch中实现分页查询的方式有两种，分别为深度分页(from-size)和快照分页(scroll)

* 1. 深度分页(from-size)

**GET goods/fruit/\_search?size=4**

**GET goods/fruit/\_search?size=4&from=0**

**GET goods/fruit/\_search?size=4&from=4**

当然你使用这样的方式来查询会更好,带条件的方式

**GET goods/fruit/\_search**

**{**

**"query": {**

**"match": {**

**"name": "pingguo"**

**}**

**},**

**"from": 0,**

**"size": 20**

**}**

* 1. 快照分页(scroll)

相对于from和size的分页来说，使用scroll可以模拟一个传统数据的游标，记录当前读取的文档信息位置。这个分页的用法，不是为了实时查询数据，而是为了一次性查询大量的数据（甚至是全部的数据）。

因为这个scroll相当于维护了一份当前索引段的快照信息，这个快照信息是你执行这个scroll查询时的快照。在这个查询后的任何新索引进来的数据，都不会在这个快照中查询到。但是它相对于from和size，不是查询所有数据然后剔除不要的部分，而是记录一个读取的位置，保证下一次快速继续读取。

**GET goods/fruit/\_search?scroll=1m**

查询结果如下:

{

"\_scroll\_id" : "DnF1ZXJ5VGhlbkZldGNoBQAAAAAAAGSBFmNhaXZ5WVBLU0tTSU5nWG91VTdZUGcAAAAAAABkghZjYWl2eVlQS1NLU0lOZ1hvdVU3WVBnAAAAAAAAZIMWY2FpdnlZUEtTS1NJTmdYb3VVN1lQZwAAAAAAAGSAFmNhaXZ5WVBLU0tTSU5nWG91VTdZUGcAAAAAAABkhBZjYWl2eVlQS1NLU0lOZ1hvdVU3WVBn",

"took" : 3,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"skipped" : 0,

"failed" : 0

},

"hits" : {

"total" : 10,

"max\_score" : 1.0,

"hits" : [

…..

**注意:这里会提供一个scroll\_id,这个id是通过base64加密过的内容,可以通过在线base解密来看下这个加密的内容,跟查询的数据有关系**



使用快照查询附带条件的时候使用

**GET goods/fruit/\_search?scroll=1m&search\_type=dfs\_query\_then\_fetch**

**{**

**"query": {**

**"match": {**

**"name": "pingguo"**

**}**

**},**

**"size": 20-----------------------如果非要加条数的话**

**}**

**重点: query\_then\_fetch和dfs\_query\_then\_fetch的区别**

**query\_then\_fetch:表示默认查询**

**dfs\_query\_then\_fetch:表示精准查询,如果需要查询的内容更精准的时候,需要使用这种查询方式,但是也会提高查询时间**

1. 中文分词检索(重点)

前面已经安装了ik中文分词器,这里就不介绍怎么安装了

**ik\_max\_word**: 会将文本做最细粒度的拆分  
**ik\_smart**: 会做最粗粒度的拆分

* 1. 先设置mapping

**PUT /my\_index7**

**{**

**"mappings": {**

**"my\_type": {**

**"properties": {**

**"text": {**

**"type": "text",**

**"analyzer": "ik\_max\_word"--------------分词方式使用细颗粒**

**}**

**}**

**}**

**}**

**}**

然后检查下当前的这个索引的状况

**GET my\_index7/\_mapping**

只有一个字段”text”随便写点东西就可以了

然后插入一批测试数据

**POST my\_index7/my\_type/1**

**{**

**"text":"你喜欢吃什么?桃子还是橘子",**

**"analyzer":"ik\_max\_word"**

**}**

**POST my\_index7/my\_type/2**

**{**

**"text":"苹果其实有很多的品种",**

**"analyzer":"ik\_max\_word"**

**}**

**POST my\_index7/my\_type/3**

**{**

**"text":"一天一个苹果,医生远离我",**

**"analyzer":"ik\_max\_word"**

**}**

**POST my\_index7/my\_type/4**

**{**

**"text":"苹了个果,看你咋吃",**

**"analyzer":"ik\_max\_word"**

**}**

开始测试

**GET my\_index7/my\_type/\_search**

**{**

**"query":{**

**"match":{**

**"text":"苹果"**

**}**

**}**

**}**

查询结果如下:

{

"took" : 1,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"skipped" : 0,

"failed" : 0

},

"hits" : {

"total" : 2,

"max\_score" : 0.6931472,

"hits" : [--------------------------------------------这里都是包含关键词”苹果”的内容

{

"\_index" : "my\_index7",

"\_type" : "my\_type",

"\_id" : "2",

"\_score" : 0.6931472,

"\_source" : {

"text" : "苹果其实有很多的品种",

"analyzer" : "ik\_max\_word"

}

},

{

"\_index" : "my\_index7",

"\_type" : "my\_type",

"\_id" : "3",

"\_score" : 0.2876821,

"\_source" : {

"text" : "一天一个苹果,医生远离我",

"analyzer" : "ik\_max\_word"

}

}

]

}

}

测试:输入一些其他的词,和短语看看效果

**注意:在当前的这个测试里,如果只输入”你”会发现检索的内容不太一样,有几个数据是没能检索出来的,所以匹配的内容也不是都能检索到的,那么输入的内容如何匹配才是检索的关键,这里只能通过多测试提高关键词的权重来解决实际问题,没有什么更好的办法**

* 1. 使用颗粒度粗的分词

还是先设置mapping

第一步:创建索引

**PUT index999**

第一步:创建mapping

**PUT /index999/fulltext/\_mapping**

**{**

**"fulltext": {**

**"\_all": {**

**"analyzer": "ik\_smart"**

**},**

**"properties": {**

**"content": {**

**"type": "text"**

**},**

**"title":{**

**"type":"text"**

**}**

**}**

**}**

**}**

第三步:插入数据

**POST /index999/fulltext/1**

**{**

**"title":"路过面了个试就拿到2个offer。是运气吗",**

**"content": "路过随便面个试就拿到2个offer。是运气吗？ #复习很重要#看看面试问的问题，再瞧瞧师兄的学习态度，你就明白 机会为何总与你擦肩而过了。[玫瑰] 以下是我和师兄的聊天记录，你会几个"**

**}**

**POST /index999/fulltext/2**

**{**

**"title":"人生没有白走的路，每一步都算数",**

**"content": "看了一位新同学的自我介绍，看完后让人沉思，有多少人也跟他一样，不是不想努力，而是一直找不到方向，只能在底层兜兜转转消磨殆尽美好青春年华。 他叫车融，在深圳工作，我去深圳分校时，他正好去咨询，他跟我聊过很多，感觉是个很有想法的人，今天看完他写的这个东西，感觉有些沉重，但人生的路， 每一步都算数，过往的"**

**}**

**POST /index999/fulltext/3**

**{**

**"title":"不吹不擂，你想要的Python面试都在这里了【315+道题】",**

**"content": "写在前面 近日恰逢学生毕业季，课程后期大家“期待+苦逼”的时刻莫过于每天早上内容回顾和面试题问答部分【临近毕业每天课前用40-60分钟对之前内容回顾、提问和补充，专挑班里不爱说话就的同学回答】。 期待的是可以检验自己学习的成功；苦逼的是怎么又有东西没记住，但我们依然每天坚持一遍、一遍又一遍指导记住为"**

**}**

**POST /index999/fulltext/4**

**{**

**"title":"白话tornado源码之褪去模板的外衣",**

**"content": "上一篇《白话tornado源码之请求来了》介绍了客户端请求在tornado框架中的生命周期，其本质就是利用epoll和socket来获取并处理请求。在上一篇的内容中，我们只是给客户端返回了简单的字符串，如：“Hello World”，而在实际开发中，需要使用html文件的内容作为模板，然后将被处理后"**

**}**

**POST /index999/fulltext/5**

**{**

**"title":"第二篇：白话tornado源码之待请求阶段",**

**"content": "上篇《白话tornado源码之一个脚本引发的血案》用上帝视角多整个框架做了一个概述，同时也看清了web框架的的本质，下面我们从tornado程序的起始来分析其源码。概述上图是tornado程序启动以及接收到客户端请求后的整个过程，对于整个过程可以分为两大部分：启动程序阶段，又称为待请求阶段"**

**}**

开始测试:

**GET /index999/fulltext/\_search**

**{**

**"query": {**

**"match": {**

**"content": "白话"**

**}**

**}**

**}**

检索结果就不贴了,这里会看见检索出来的内容跟预想的完全不同,根本就不是咱们要的东西,但是max\_score中的数据还是跟设定的关键词有比较强的关联.

**GET /index999/fulltext/\_search**

**{**

**"query": {**

**"match\_phrase": {**

**"content": "白话"**

**}**

**}**

**}**

这个时候你再查询就可以看到检索的内容了,后面会详细的讲下几种查询方式

1. 查询方式:

Match:

**GET my\_index10/my\_type/\_search**

**{**

**"query": {**

**"match": {**

**"FIELD": "TEXT"**

**}**

**}**

**}**

**重点:matc知道分词器的存在，会对field进行分词操作，然后再查询,最常见的查询方式**

Match\_all:

**GET my\_index10/my\_type/\_search**

**{**

**"query": {**

**"match\_all": {}**

**}**

**}**

**重点:查询所有的文档,不太常用**

Match\_phrase:

**GET my\_index10/my\_type/\_search**

**{**

**"query": {**

**"match\_phrase": {**

**"FIELD": "PHRASE"**

**}**

**}**

**}**

**重点: ES引擎首先分析查询字符串，从分析后的文本中构建短语查询，这意味着必须匹配短语中的所有分词，并且保证各个分词的相对位置不变**

Match\_phrase\_prefix:

**GET my\_index10/my\_type/\_search{**

**"query": {**

**"match\_phrase\_prefix": {**

**"FIELD": "PREFIX"**

**}**

**}**

**}**

**重点:短语前缀查询,类似于数据库中的 like ‘a%’这样的以什么开头的插叙方式,对于中文来说于普通的短语测试没有什么太大的区别(需要测试)**

Term:

**GET my\_index10/my\_type/\_search**

**{**

**"query": {**

**"term": {**

**"FIELD": {**

**"value": "VALUE"**

**}**

**}**

**}**

**}**

**重点: term会去倒排索引中寻找确切的term，它并不知道分词器的存在，这种查询适合keyword、numeric、date等明确值的,查询某个字段里含有某个关键词的文档**

Terms:

**GET my\_index10/my\_type/\_search**

**{**

**"query": {**

**"terms": {**

**"FIELD": [**

**"VALUE1",**

**"VALUE2"**

**]**

**}**

**}**

**}**

**重点: terms：查询某个字段里含有多个关键词的文档**

1. 分词原理(**重点:参考文献说明**)

三大主流分词方法：基于词典的方法、基于规则的方法和基于统计的方法。

1、基于规则或词典的方法

定义：按照一定策略将待分析的汉字串与一个“大机器词典”中的词条进行匹配，若在词典中找到某个字符串，则匹配成功。

按照扫描方向的不同：正向匹配和逆向匹配

按照长度的不同：最大匹配和最小匹配

1.1正向最大匹配思想MM

从左向右取待切分汉语句的m个字符作为匹配字段，m为大机器词典中最长词条个数。

查找大机器词典并进行匹配：

若匹配成功，则将这个匹配字段作为一个词切分出来。

若匹配不成功，则将这个匹配字段的最后一个字去掉，剩下的字符串作为新的匹配字段，进行再次匹配，重复以上过程，直到切分出所有词为止。

举个栗子：

现在，我们要对“南京市长江大桥”这个句子进行分词，根据正向最大匹配的原则：

先从句子中拿出前5个字符“南京市长江”，把这5个字符到词典中匹配，发现没有这个词，那就缩短取字个数，取前四个“南京市长”，发现词库有这个词，就把该词切下来；

对剩余三个字“江大桥”再次进行正向最大匹配，会切成“江”、“大桥”；

整个句子切分完成为：南京市长、江、大桥；

1.2逆向最大匹配算法RMM

该算法是正向最大匹配的逆向思维，匹配不成功，将匹配字段的最前一个字去掉，实验表明，逆向最大匹配算法要优于正向最大匹配算法。

还是那个栗子：

取出“南京市长江大桥”的后四个字“长江大桥”，发现词典中有匹配，切割下来；

对剩余的“南京市”进行分词，整体结果为：南京市、长江大桥

1.3 双向最大匹配法(Bi-directction Matching method,BM)

双向最大匹配法是将正向最大匹配法得到的分词结果和逆向最大匹配法的到的结果进行比较，从而决定正确的分词方法。

据SunM.S. 和 Benjamin K.T.（1995）的研究表明，中文中90.0％左右的句子，正向最大匹配法和逆向最大匹配法完全重合且正确，只有大概9.0％的句子两种切分方法得到的结果不一样，但其中必有一个是正确的（歧义检测成功），只有不到1.0％的句子，或者正向最大匹配法和逆向最大匹配法的切分虽重合却是错的，或者正向最大匹配法和逆向最大匹配法切分不同但两个都不对（歧义检测失败）。这正是双向最大匹配法在实用中文信息处理系统中得以广泛使用的原因所在。

还是那个栗子：

双向的最大匹配，即把所有可能的最大词都分出来，上面的句子可以分为：南京市、南京市长、长江大桥、江、大桥

1.4设立切分标志法

收集切分标志，在自动分词前处理切分标志，再用MM、RMM进行细加工。

1.5最佳匹配（OM，分正向和逆向）

对分词词典按词频大小顺序排列，并注明长度，降低时间复杂度。

优点：易于实现

缺点：匹配速度慢。对于未登录词的补充较难实现。缺乏自学习。

1.6逐词遍历法

这种方法是将词库中的词由长到短递减的顺序，逐个在待处理的材料中搜索，直到切分出所有的词为止。

处理以上基本的机械分词方法外，还有双向扫描法、二次扫描法、基于词频统计的分词方法、联想—回溯法等。

2、基于统计的分词

随着大规模语料库的建立，统计机器学习方法的研究和发展，基于统计的中文分词方法渐渐成为了主流方法。

主要思想：把每个词看做是由词的最小单位各个字总成的，如果相连的字在不同的文本中出现的次数越多，就证明这相连的字很可能就是一个词。因此我们就可以利用字与字相邻出现的频率来反应成词的可靠度，统计语料中相邻共现的各个字的组合的频度，当组合频度高于某一个临界值时，我们便可认为此字组可能会构成一个词语。

主要统计模型：N元文法模型（N-gram），隐马尔可夫模型（Hidden Markov Model ，HMM），最大熵模型（ME），条件随机场模型（Conditional Random Fields，CRF）等。

优势：在实际的应用中经常是将分词词典串匹配分词和统计分词能较好地识别新词两者结合起来使用，这样既体现了匹配分词切分不仅速度快，而且效率高的特点；同时又能充分地利用统计分词在结合上下文识别生词、自动消除歧义方面的优点。

2.1 N-gram模型思想

模型基于这样一种假设，第n个词的出现只与前面N-1个词相关，而与其它任何词都不相关，整句的概率就是各个词出现概率的乘积。

我们给定一个词，然后猜测下一个词是什么。当我说“艳照门”这个词时，你想到下一个词是什么呢？我想大家很有可能会想到“陈冠希”，基本上不会有人会想到“陈志杰”吧，N-gram模型的主要思想就是这样的。

对于一个句子T，我们怎么算它出现的概率呢？假设T是由词序列W1,W2,W3,…Wn组成的，那么P(T)=P(W1W2W3…Wn)=P(W1)P(W2|W1)P(W3|W1W2)…P(Wn|W1W2…Wn-1)

但是这种方法存在两个致命的缺陷：一个缺陷是参数空间过大，不可能实用化；另外一个缺陷是数据稀疏严重。为了解决这个问题，我们引入了马尔科夫假设：一个词的出现仅仅依赖于它前面出现的有限的一个或者几个词。如果一个词的出现仅依赖于它前面出现的一个词，那么我们就称之为bigram。即

P(T) =P(W1W2W3…Wn)=P(W1)P(W2|W1)P(W3|W1W2)…P(Wn|W1W2…Wn-1)

≈P(W1)P(W2|W1)P(W3|W2)…P(Wn|Wn-1)

如果一个词的出现仅依赖于它前面出现的两个词，那么我们就称之为trigram。

在实践中用的最多的就是bigram和trigram了，而且效果很不错。高于四元的用的很少，因为训练它需要更庞大的语料，而且数据稀疏严重，时间复杂度高，精度却提高的不多。一般的小公司，用到二元的模型就够了，像Google这种巨头，也只是用到了大约四元的程度，它对计算能力和空间的需求都太大了。

以此类推，N元模型就是假设当前词的出现概率只同它前面的N-1个词有关。

2.2 HMM、CRF 模型思想

以往的分词方法，无论是基于规则的还是基于统计的，一般都依赖于一个事先编制的词表(词典)，自动分词过程就是通过词表和相关信息来做出词语切分的决策。与此相反，

基于字标注（或者叫基于序列标注）的分词方法实际上是构词方法，即把分词过程视为字在字串中的标注问题。

由于每个字在构造一个特定的词语时都占据着一个确定的构词位置(即词位)，假如规定每个字最多只有四个构词位置：即B(词首)，M (词中)，E(词尾)和S(单独成词)，那么下面句子(甲)的分词结果就可以直接表示成如(乙)所示的逐字标注形式：

(甲)分词结果：／上海／计划／N／本／世纪／末／实现／人均／国内／生产／总值／五千美元／

(乙)字标注形式：上／B海／E计／B划／E N／S 本／s世／B 纪／E 末／S 实／B 现／E 人／B 均／E 国／B 内／E生／B产／E总／B值／E 五／B千／M 美／M 元／E 。／S

1

2

首先需要说明，这里说到的“字”不只限于汉字。考虑到中文真实文本中不可避免地会包含一定数量的非汉字字符，本文所说的“字”，也包括外文字母、阿拉伯数字和标点符号等字符。所有这些字符都是构词的基本单元。当然，汉字依然是这个单元集合中数量最多的一类字符。

把分词过程视为字的标注问题的一个重要优势在于，它能够平衡地看待词表词和未登录词的识别问题。

在这种分词技术中，文本中的词表词和未登录词都是用统一的字标注过程来实现的。在学习架构上，既可以不必专门强调词表词信息，也不用专门设计特定的未登录词(如人名、地名、机构名)识别模块。这使得分词系统的设计大大简化。在字标注过程中，所有的字根据预定义的特征进行词位特性的学习，获得一个概率模型。然后，在待分字串上，根据字与字之间的结合紧密程度，得到一个词位的标注结果。最后，根据词位定义直接获得最终的分词结果。总而言之，在这样一个分词过程中，分词成为字重组的简单过程。在学习构架上，由于可以不必特意强调词表词的信息，也不必专门设计针对未登录词的特定模块，这样使分词系统的设计变得尤为简单。

2001年Lafferty在最大熵模型（MEM）和隐马尔科夫模型（HMM）的基础上提出来了一种无向图模型–条件随机场（CRF）模型，它能在给定需要标记的观察序列的条件下，最大程度提高标记序列的联合概率。常用于切分和标注序列化数据的统计模型。CRF算法理论见我的其他博客，此处就不赘述了。

2.3 基于统计分词方法的实现

现在，我们已经从全概率公式引入了语言模型，那么真正用起来如何用呢？

我们有了统计语言模型，下一步要做的就是划分句子求出概率最高的分词，也就是对句子进行划分，最原始直接的方式，就是对句子的所有可能的分词方式进行遍历然后求出概率最高的分词组合。但是这种穷举法显而易见非常耗费性能，所以我们要想办法用别的方式达到目的。

仔细思考一下，假如我们把每一个字当做一个节点，每两个字之间的连线看做边的话，对于句子“中国人民万岁”，我们可以构造一个如下的分词结构：

我们要找概率最大的分词结构的话，可以看做是一个动态规划问题， 也就是说，要找整个句子的最大概率结构，对于其子串也应该是最大概率的。

对于句子任意一个位置tt上的字，我们要从词典中找到其所有可能的词组形式，如上图中的第一个字，可能有：中、中国、中国人三种组合，第四个字可能只有民，经过整理，我们的分词结构可以转换成以下的有向图模型:

我们要做的就是找到一个概率最大的路径即可。我们假设Ct(k)Ct(k)表示第tt个字的位置可能的词是k，那么可以写出状态转移方程：

其中k是当前位置的可能单词，l是上一个位置的可能单词，M是l可能的取值，有了状态转移返程，写出递归的动态规划代码就很容易了（这个方程其实就是著名的viterbi算法，通常在隐马尔科夫模型中应用较多）。

需要特别注意的几点是：

1. 做递归计算式务必使用缓存，把子问题的解先暂存起来，参考动态规划入门实践。

2. 当前位置的前一位置应当使用当前位置单词的长度获得。

3. 以上代码只是作为实验用，原理没有问题，但性能较差，生产情况需要建立索引以提高性能。

4. 本分词代码忽略了英文单词、未登录词和标点符号，但改进并不复杂，读者可自行斟酌。

1. 以上是搜索的相关内容