

# NiuParser SDK 系统开发文档



斗斗牛团队  
东北大学自然语言处理实验室  
[niuparser@zjyatu.com](mailto:niuparser@zjyatu.com) <http://www.niuparser.com>

©2014-2015 NiuParser 版权所有

## 目录

1	NiuParser SDK 系统简介 .....	3
2	NiuParser SDK 系统功能介绍 .....	4
2.1	自动分词 .....	4
2.2	词性标注 .....	4
2.3	命名实体识别 .....	4
2.4	组块识别 .....	4
2.5	成分句法分析 .....	5
2.6	依存句法分析 .....	5
2.7	语义角色标注 .....	6
3	NiuParser SDK 系统 C/C++接口 .....	7
3.1	初始化接口 .....	7
3.2	分词功能接口 .....	8
3.2.1	分词 .....	8
3.2.2	释放分词结果 .....	8
3.2.3	使用示例 .....	9
3.3	词性标注功能接口 .....	9
3.3.1	词性标注 .....	9
3.3.2	释放词性标注结果 .....	10
3.3.3	使用示例 .....	10
3.4	命名实体识别功能接口 .....	11
3.4.1	命名实体识别 .....	11
3.4.2	释放命名实体识别结果 .....	12
3.4.3	使用示例 .....	12
3.5	组块识别功能接口 .....	13
3.5.1	组块识别 .....	13
3.5.2	释放组块识别结果 .....	13
3.5.3	使用示例 .....	14
3.6	成分句法分析功能接口 .....	14
3.6.1	成分句法分析 .....	14
3.6.2	释放成分句法分析结果 .....	14
3.6.3	使用示例 .....	15
3.7	依存句法分析功能接口 .....	15
3.7.1	依存句法分析 .....	15
3.7.2	释放依存句法分析结果 .....	16
3.7.3	使用示例 .....	17
3.8	语义角色标注功能接口 .....	17
3.8.1	语义角色标注 .....	17
3.8.2	释放语义角色标注结果 .....	18
3.8.3	使用示例 .....	19
3.9	销毁接口 .....	19
3.10	返回值描述 .....	20

4	NiuParser SDK 系统使用方法 .....	21
4.1	Windows 下使用 .....	21
4.2	Linux 下使用 .....	22
5	NiuParser SDK 系统标记规范 .....	23
5.1	词性标注集 .....	23
5.2	命名实体标记集 .....	23
5.3	句法标记集 .....	24
5.4	依存关系标记集 .....	24
5.5	语义角色标记集 .....	25

## 1 NiuParser SDK 系统简介

东北大学自然语言处理实验室 (<http://www.nlplab.com>) 自从 1980 年创立以来一直从事语言分析和机器翻译技术研究工作, 基于三十多年的雄厚研究积累, 研制了一套中文句法语义分析系统 NiuParser。

NiuParser 系统能够支持中文句子级的**自动分词**、**词性标注**、**命名实体识别**、**组块识别**、**成分句法分析**、**依存句法分析**和**语义角色标注**七大语言分析技术。所有代码采用 C++ 语言开发, 全部自主完成, 不包含任何其它开源代码, 拥有独立完整的知识产权。

NiuParser SDK 是基于 NiuParser 系统的软件开发工具包, 可以为用户提供本地化的 NiuParser 中文分析技术开发接口, 帮助用户进行二次开发。目前 NiuParser SDK 支持 Windows 64bit 、 Linux 64bit 操作系统上的 C/C++ 语言接口, 我们会逐步增加支持的编程语言和平台。

NiuParser 的特点是分析速度快并且拥有业内最好的分析性能, 可以被广泛应用于研制基于深度计算的文本分析和文本挖掘等商业应用系统。NiuParser 和 SDK 可以免费用于研究目的, 但商业用途需获得授权许可。

NiuParser 研究版支持语言分析技术的单线程 SDK 开发接口, 为了满足大规模数据分析处理目的, 基于云平台和多线程处理能力的 NiuParser SDK 开发接口可以在商业版本中得到有效技术支持。斗斗牛团队专注于自然语言处理技术, 可以为用户提供基于 NiuParser 技术的高级语言分析和智能处理技术服务。感谢大家下载 NiuParser 研究版免费用于研究工作, 欢迎大家与我们斗斗牛团队开展技术合作, 利用 NiuParser 技术来开发语言智能处理应用系统。

斗斗牛团队

联系邮件: [niuparser@zivatuo.com](mailto:niuparser@zivatuo.com)

## 2 NiuParser SDK 系统功能介绍

### 2.1 自动分词

自动分词：将中文句子切分成词。如：

**输入：** 欢迎使用我们的中文语言平台。

**输出：** 欢迎 使用 我们 的 中文 语言 平台 。

### 2.2 词性标注

词性标注：输入经过分词的句子，为句子中的词标上词性。如：

**输入：** 欢迎使用我们的中文语言平台。

**输出：** 欢迎/VV 使用/VV 我们/PN 的/DEG 中文/NN 语言/NN 平台/NN 。/PN

### 2.3 命名实体识别

命名实体识别：输入经过分词和词性标注的句子，识别句子中的命名实体。如：

**输入：** 欢迎/VV 使用/VV 我们/PN 的/DEG 中文/NN 语言/NN 平台/NN 。/PN

欢迎 使用 我们 的 中文 语言 平台 。



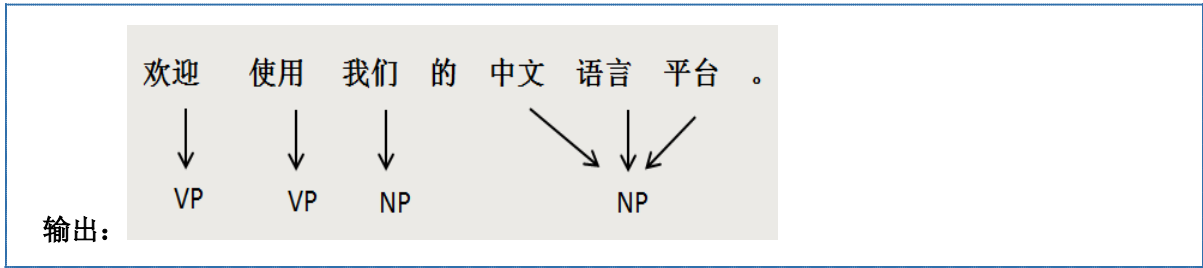
LANGUAGE

**输出：**

### 2.4 组块识别

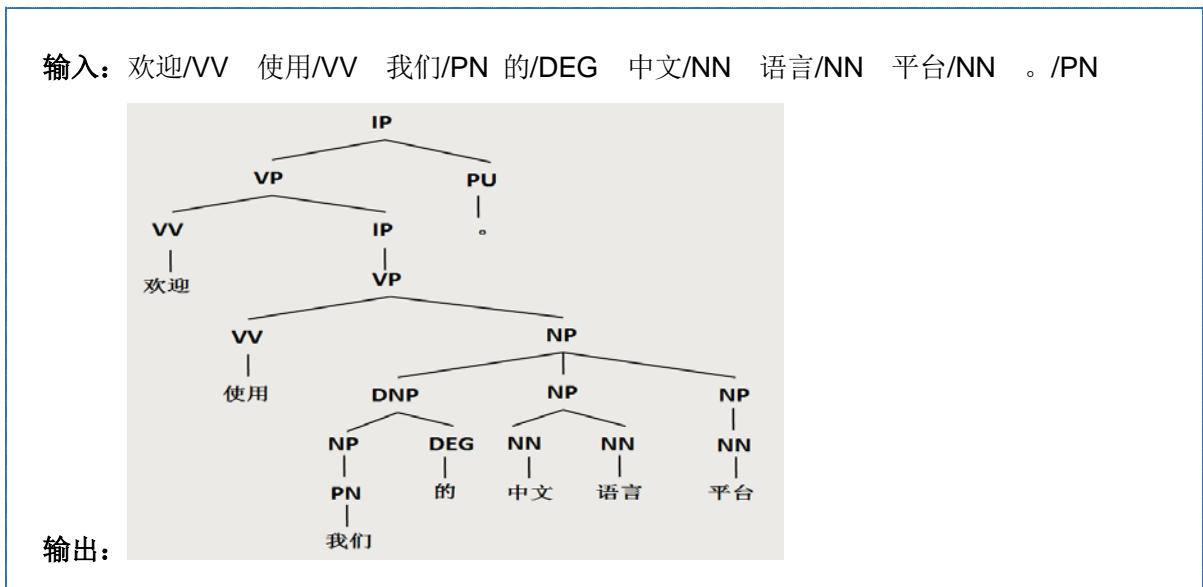
组块识别：输入经过分词和词性标注的句子，识别句子中的组块（基本短语）。如：

**输入：** 欢迎/VV 使用/VV 我们/PN 的/DEG 中文/NN 语言/NN 平台/NN 。/PN



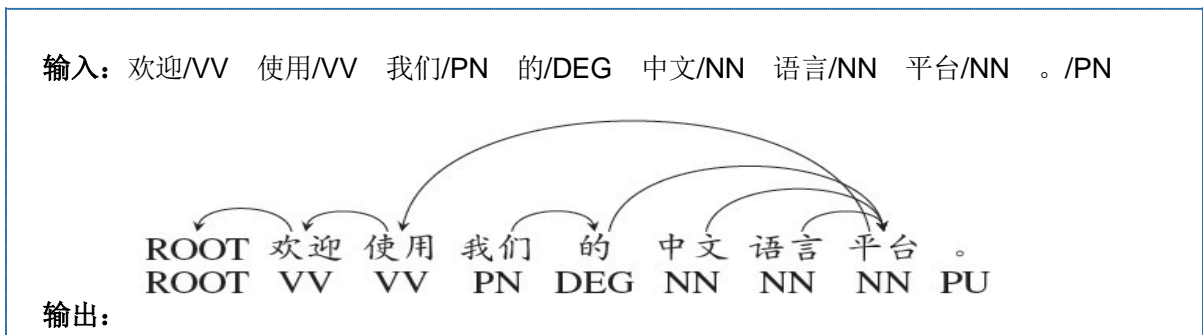
## 2.5 成分句法分析

**成分句法分析:** 通过分词和词性标注的句子，识别句子的成分句法结构（也称为短语句法结构）。如：



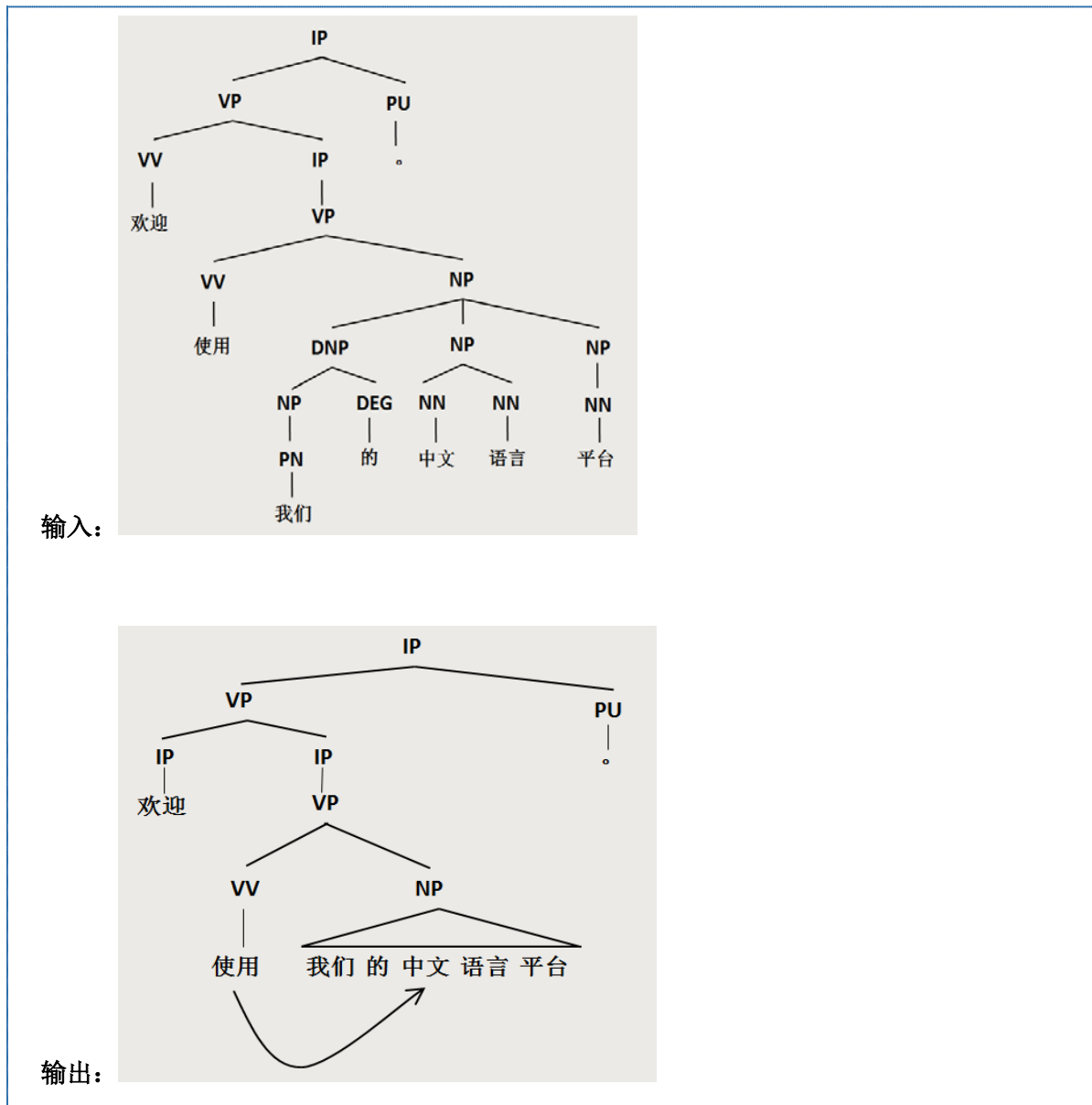
## 2.6 依存句法分析

**依存句法分析:** 给定一个经过分词和词性标注的句子，识别词语之间的依存关系。如：



## 2.7 语义角色标注

**语义角色标注：**输入一个成分句法树，识别指定谓词的语义对象。语义角色标注用于抽象独立于句法结构的语义结构。如：



## 3 NiuParser SDK 系统 C/C++接口

### 3.1 初始化接口

#### 函数原型

```
int NiuParser_Create_Instance(Switch_ST & switcher);
```

#### 函数功能

初始化NiuParser SDK ， 载入模型

#### 参数说明

参数名	类型	参数解释
switcher	Switch_ST	NiuParser 七大功能的开关，以便初始化用户需要的功能

#### Switch\_ST 结构

变量名	类型	意义
m_is_ws_on	bool	True 表示启用分词功能， False 表示不启用
m_is_pos_on	bool	True 表示启用词性标注功能， False 表示不启用
m_is_ner_on	bool	True 表示启用命名实体识别功能， False 表示不启用
m_is_chk_on	bool	True 表示启用组块识别功能， False 表示不启用
m_is_cp_on	bool	True 表示启用成分句法分析功能， False 表示不启用
m_is_dp_on	bool	True 表示启用依存句法分析功能， False 表示不启用
m_is_srl_on	bool	True 表示启用语义角色标注功能， False 表示不启用

#### 返回值

返回值	数值
NIUPARSER_RET_OK	0
NIUPARSER_RET_ERROR	1
NIUPARSER_RET_POINTER_NULL	2
NIUPARSER_RET_LICENSE_ERROR	5
NIUPARSER_RET_FILE_NOT_ACCESS	7

详见 3.10 返回值描述表

#### 说明

switcher 默认值是所有的开关均为 False，即不开启。用户只需将自己需要使用的功能所对应的开关置为 True，即可使用 NiuParser 相应的功能。

另外，用户无需考虑功能之间的依赖关系，程序会自动判断。如：用户需要使用依存句法分析功能，只需要将 switcher.m\_is\_dp\_on = true 即可。不需要再去把分词、词性标注的开关也打开。



## 示例

(详见 example/niuparser\_example.cpp)

```
Switch_ST switcher; //定义开关
switcher.m_is_ws_on = true; //使用分词功能
if(NiuParser_Create_Instance(switcher) != NIUPARSER_RET_OK) //初始化失败则退出
    exit(-1);
```

## 3.2 分词功能接口

### 3.2.1 分词

#### 函数原型

```
int NiuParser_Segmenter_Sentence(char* input , Seg_Result_ST & seg_result);
```

#### 函数功能

对原始中文句子进行分词

#### 参数说明

参数名	类型	参数解释
Input	char*	待处理的原始中文句子, 句子长度不超过500个汉字
seg_result	Seg_Result_ST	结构体, 存放分词结果

#### Seg\_Result\_ST 结构

变量名	类型	意义
m_word_num	unsigned int	表示分词后的词的个数
m_word_array	char**	表示每一个词

#### 返回值

返回值	数值
NIUPARSER_RET_OK	0
NIUPARSER_RET_ERROR	1
NIUPARSER_RET_POINTER_NULL	2
NIUPARSER_RET_FUNC_INVALID	4
NIUPARSER_RET_INPUT_TOO_LONG	6
NIUPARSER_RET_ENCODING_ERROR	8

详见 3.10 返回值描述表

### 3.2.2 释放分词结果

#### 函数原型

```
int NiuParser_Release_Segment_Memory(Seg_Result_ST & seg_result);
```

## 函数功能

释放存放分词结果的内存

参数名	类型	参数解释
seg_result	Seg_Result_ST	结构体，存放分词结果

## 返回值

返回值	数值
NIUPARSER_RET_OK	0
NIUPARSER_RET_FUNC_INVALID	4

详见 3.10 返回值描述表

## 说明

调用 **NiuParser\_Segment\_Sentence** 进行分词，当不再使用当前分词结果时，需要调用此函数释放当前存放分词结果的内存空间。

### 3.2.3 使用示例

(详见 example/niuparser\_example.cpp)

```

Seg_Result_ST seg_result ;//定义分词的结果
char str[] = "欢迎使用我们的中文语言平台。";//原始中文句子
int ret = NiuParser_Segmenter_Sentence(str, seg_result);//进行分词
if(ret == NIUPARSER_RET_OK)//分词成功
{
    for(unsigned int i=0;i<seg_result.m_word_num;i++) //打印分词结果
    {
        cout<<seg_result.m_word_array[i]<<" ";
    }
    cout<<endl;
    NiuParser_Release_Segment_Memory(seg_result); //释放分词结果内存
}

```

#### 输出结果

欢迎 使用 我们 的 中文 语言 平台 。

## 3.3 词性标注功能接口

### 3.3.1 词性标注

#### 函数原型

```
int NiuParser_POS_Tagger_Sentence(char* input, Pos_Result_ST & pos_result);
```

#### 函数功能

进行原始中文句子进行词性标注

### 参数说明

参数名	类型	参数解释
Input	char*	待处理的原始中文句子，句子长度不超过 500 个汉字
pos_result	Pos_Result_ST	结构体，存放词性标注结果

#### Pos\_Result\_ST 结构

变量名	类型	意义
m_word_num	unsigned int	表示分词后的词的个数
m_cell_array	Pos_Cell_ST*	表示每一个词及词性

#### Pos\_Cell\_ST 结构

变量名	类型	意义
m_word	char*	表示词
m_pos	char[64]	表示词性

### 返回值

同 3.2.1 返回值，详见 3.10 返回值描述表

## 3.3.2 释放词性标注结果

### 函数原型

```
int NiuParser_Release_Pos_Tagger_Memory(Pos_Result_ST & pos_result);
```

### 函数功能

释放存放词性标注结果的内存

参数名	类型	参数解释
pos_result	Pos_Result_ST	结构体，存放词性标注结果

### 返回值

同 3.2.2 返回值，详见 3.10 返回值描述表

### 说明

调用 **NiuParser\_Pos\_Tagger\_Sentence** 进行词性标注，当不再使用当前词性标注结果时，需要调用此函数释放当前存放词性标注结果的内存空间。

## 3.3.3 使用示例

(详见 example/niuparser\_example.cpp)

```
Pos_Result_ST pos_result ;//定义词性标注的结果
char str[] = "欢迎使用我们的中文语言平台。"; //原始中文句子
int ret = NiuParser_Pos_Tagger_Sentence(str, pos_result); //进行词性标注
if(ret == NIUPARSER_RET_OK) //词性标注成功
{
    for(unsigned int i=0;i<pos_result.m_word_num;i++) //打印词性标注结果
```

```

    {
        cout<<pos_result.m_cell_array[i].m_word<<"/"<<pos_result.m_cell_array[i].m
_pos<<" ";
    }
    cout<<endl;
    NiuParser_Release_Pos_Tagger_Memory(pos_result); //释放词性标注结果内存
}

```

**输出结果**

欢迎/VV 使用/VV 我们/PN 的/DEG 中文/NN 语言/NN 平台/NN 。/PU

## 3.4 命名实体识别功能接口

### 3.4.1 命名实体识别

**函数原型**

```
int NiuParser_NE_Recognizer_Sentence(char* input ,Ner_Result_ST &ner_result);
```

**函数功能**

进行原始中文句子进行命名实体识别

**参数说明**

参数名	类型	参数解释
input	char*	待处理的原始中文句子，句子长度不超过 500 个汉字
ner_result	Ner_Result_ST	结构体，存放命名实体识别结果

**Ner\_Result\_ST 结构**

变量名	类型	意义
m_ne_num	unsigned int	表示命名实体的个数
m_cell_array	Ner_Cell_ST*	表示每一个命名实体
m_seg_result	Seg_Result_ST	表示分词结果

**Ner\_Cell\_ST 结构**

变量名	类型	意义
m_start_pos	int	表示命名实体起始位置（从 0 计数）
m_length	int	表示命名实体长度（即该命名实体由几个词组成）
m_type	char[64]	表示命名实体类别

**返回值**

同 3.2.1 返回值，详见 3.10 返回值描述表

### 3.4.2 释放命名实体识别结果

#### 函数原型

```
int NiuParser_Release_Name_Entity_Memory(Ner_Result_ST & ner_result);
```

#### 函数功能

释放存放命名实体结果的内存

参数名	类型	参数解释
ner_result	Ner_Result_ST	结构体, 存放命名实体识别结果

#### 返回值

同 3.2.2 返回值, 详见 3.10 返回值描述表

#### 说明

调用 **NiuParser\_NE\_Recognizer\_Sentence** 进行命名实体识别, 当不再使用当前命名实体识别结果时, 需要调用此函数释放当前存放命名实体识别结果的内存空间。

### 3.4.3 使用示例

(详见 example/niuparser\_example.cpp)

```
Ner_Result_ST ner_result ;//定义命名实体识别的结果
char str[] = "欢迎使用我们的中文语言平台。";//原始中文句子
int ret = NiuParser_Ne_Recognizer_Sentence(str, ner_result);//进行命名实体识别
if(ret == NIUPARSER_RET_OK)//命名实体识别成功
{
    char buffer[1024] = {0};
    for(unsigned int i=0;i<ner_result.m_ne_num;i++) //打印命名实体识别结果
    {
        memset(buffer , 0 , 1024*sizeof(char));
        Ner_Cell_ST cell = ner_result.m_cell_array[i];
        for(unsigned int
pos=cell.m_start_pos;pos<cell.m_length+cell.m_start_pos;pos++)
        {
            strcat(buffer,ner_result.m_seg_result.m_word_array[pos]);
            strcat(buffer," ");
        }
        buffer[strlen(buffer)-1] = '\0';
        cout<<" => " <<cell.m_type<<" ";
    }
    cout<<"\n";
    NiuParser_Release_Name_Entity_Memory(pos_result); //释放命名识别识别结果内存
}
```

#### 输出结果

中文 => LANGUAGE

## 3.5 组块识别功能接口

### 3.5.1 组块识别

#### 函数原型

```
int NiuParser_CHK_Recognizer_Sentence(char* input ,Chk_Result_ST & chk_result);
```

#### 函数功能

进行原始中文句子进行组块识别

#### 参数说明

参数名	类型	参数解释
Input	char*	待处理的原始中文句子，句子长度不超过 500 个汉字
chk_result	Chk_Result_ST	结构体，存放组块识别的结果

#### Chk\_Result\_ST 结构

变量名	类型	意义
m_chk_num	unsigned int	表示组块的个数
m_cell_array	Chk_Cell_ST*	表示每一个组块
m_seg_result	Seg_Result_ST	表示分词结果

#### Chk\_Cell\_ST 结构

变量名	类型	意义
m_start_pos	int	表示组块起始位置（从 0 计数）
m_length	int	表示组块长度（即该组块由几个词组成）
m_type	char[64]	表示组块类别

#### 返回值

同 3.2.1 返回值，详见 3.10 返回值描述表

#### 说明

用法同 Ner\_Result\_ST

### 3.5.2 释放组块识别结果

#### 函数原型

```
int NiuParser_Release_Chunking_Memory(Chk_Result_ST & chk_result);
```

#### 函数功能

释放存放组块分析结果的内存

参数名	类型	参数解释
chk_result	Chk_Result_ST	结构体，存放组块分析结果

## 返回值

同 3.2.2 返回值，详见 3.10 返回值描述表

## 说明

调用 **NiuParser\_CHK\_Recognizer\_Sentence** 进行组块分析，当不再使用当前组块分析结果时，需要调用此函数释放当前存放组块分析结果的内存空间。

### 3.5.3 使用示例

与 3.4.3 用法相似。(详见 example/niuparser\_example.cpp)

#### 输出结果

欢迎 => VP 使用 => VP 我们 => NP 的 => DNP 中文 语言 平台 => NP

## 3.6 成分句法分析功能接口

### 3.6.1 成分句法分析

#### 函数原型

```
int NiuParser_Constitunet_Parser_Sentence(char* input ,Con_Result_ST & con_result);
```

#### 函数功能

进行原始中文句子进行成分句法分析

#### 参数说明

参数名	类型	参数解释
Input	char*	待处理的原始中文句子，句子长度不超过 500 个汉字
con_result	Con_Result_ST	结构体，存放成分句法分析结果

Con\_Result\_ST 结构

变量名	类型	意义
m_result	char*	表示生成的成分句法树

#### 返回值

同 3.2.1 返回值，详见 3.10 返回值描述表

### 3.6.2 释放成分句法分析结果

#### 函数原型

```
int NiuParser_Release_Constituent_Parser_Memory(Con_Result_ST & con_result);
```

#### 函数功能

释放存放成分句法分析结果的内存

### 参数说明

参数名	类型	参数解释
con_result	Con_Result_ST	结构体，存放成分句法分析结果

### 返回值

同 3.2.2 返回值，详见 3.10 返回值描述表

### 说明

调用 **NiuParser\_Constituent\_Parser\_Sentence** 进行成分句法分析，当不再使用当前成分句法分析结果时，需要调用此函数释放当前存放成分句法分析结果的内存空间。

## 3.6.3 使用示例

(详见 example/niuparser\_example.cpp)

```
Con_Result_ST con_result ;//定义成分句法分析的结果
char str[] = "欢迎使用我们的中文语言平台。";//原始中文句子
int ret = NiuParser_Constituent_Parser_Sentence(str, con_result);//进行成分句法分析
if(ret == NIUPARSER_RET_OK)//分析成功
{
    cout<<con_result.m_result<<endl;//打印成分句法分析结果
    NiuParser_Release_Constituent_Parser_Memory(con_result); //释放成分句法分析结果内存
}
```

### 输出结果

```
(IP (VP (VV 欢迎)(IP (VP (VV 使用)(NP (DNP (NP (PN 我们))(DEG 的))(NP (NN 中文)(NN 语言)) (NP (NN 平台)))))))(PU 。)
```

## 3.7 依存句法分析功能接口

### 3.7.1 依存句法分析

#### 函数原型

```
int NiuParser_Dependency_Parser_Sentence(char* input, Dep_Result_ST & dep_result);
```

#### 函数功能

进行原始中文句子进行依存句法分析



## 参数说明

参数名	类型	参数解释
input	char*	待处理的原始中文句子，句子长度不超过 500 个汉字
dep_result	Dep_Result_ST	结构体，存放依存句法分析结果

## Dep\_Result\_ST 结构

变量名	类型	意义
m_word_num	unsigned int	表示词的个数
m_cell_array	Dep_Cell_ST*	表示每一个词的依存分析结果

## Dep\_Cell\_ST 结构

变量名	类型	意义
m_word	int	表示词
m_type	char[64]	表示词性
m_center_word_id	int	表示该词对应的中心词位置
m_relation	char[64]	表示依存关系类型

## 返回值

同 3.2.1 返回值，详见 3.10 返回值描述表

## 说明

m\_center\_word\_id 为 -1 时表示当前词是整个句子的中心词

## 3.7.2 释放依存句法分析结果

## 函数原型

```
int NiuParser_Release_Dependency_Parser_Memory(Dep_Result_ST & dep_result);
```

## 函数功能

释放存放依存句法分析结果的内存

## 参数说明

参数名	类型	参数解释
dep_result	Dep_Result_ST	结构体，存放依存句法分析结果

## 返回值

同 3.2.2 返回值，详见 3.10 返回值描述表

## 说明

调用 **NiuParser\_Dependency\_Parser\_Sentence** 进行依存句法分析，当不再使用当前依存句法分析结果时，需要调用此函数释放当前存放依存句法分析结果的内存空间。

### 3.7.3 使用示例

(详见 example/niuparser\_example.cpp)

```

Dep_Result_ST dep_result ;//定义依存句法分析的结果
char str[] = "欢迎使用我们的中文语言平台。";//原始中文句子
int ret = NiuParser_Dependency_Parser_Sentence(str, dep_result);//进行依存句法分析
if(ret == NIUPARSER_RET_OK)//分析成功
{
    for(unsigned int i=0;i<dep_result.m_word_num;i++)//打印依存句法分析结果

    {
        Dep_Cell_ST cell = dep_result.m_cell_array[i];

        cout<<cell.m_word<<"\t"<<cell.m_type<<"\t"<<cell.m_relation<<"\t"<<cell.m_
relation<<endl;
    }
    NiuParser_Release_Dependency_Parser_Memory(con_result); //释放依存分析结果内存
}

```

#### 输出结果

```

欢迎    VV  -1  ROOT
使用    VV  0   VMOD
我们    PN  3   DEG
的      DEG 6   NMOD
中文    NN  6   NMOD
语言    NN  6   NMOD
平台    NN  1   OBJ
。      PU  0   VMOD

```

## 3.8 语义角色标注功能接口

### 3.8.1 语义角色标注

#### 函数原型

```
int NiuParser_Semantic_Role_Label_Sentence(char* input , Srl_Result_ST &srl_result);
```

#### 函数功能

进行原始中文句子进行语义角色标注

#### 参数说明

参数名	类型	参数解释
input	char*	待处理的原始中文句子，句子长度不超过 500 个汉字
srl_result	Srl_Result_ST	结构体，存放语义角色标注结果

## Srl\_Result\_ST 结构

变量名	类型	意义
m_predict_num	unsigned int	表示谓词的个数
m_cell_array	Srl_Cell_ST*	表示每一个谓词的词性标注结果
m_seg_result	Seg_Result_ST	表示分词结果

## Srl\_Cell\_ST 结构

变量名	类型	意义
m_role_num	unsigned int	表示语义角色的个数
m_role_array	Srl_Role_ST	表示每一个语义角色

## Srl\_Role\_ST

变量名	类型	意义
m_start_pos	unsigned int	表示语义角色的起始位置（从 0 计数）
m_end_pos	unsigned int	表示语义角色的结束位置（从 0 计数）
m_label	char[64]	表示语义角色的类型

## 返回值

返回值	数值
NIUPARSER_RET_OK	0
NIUPARSER_RET_ERROR	1
NIUPARSER_RET_POINTER_NULL	2
NIUPARSER_RET_CP_FORMAT_ERROR	3
NIUPARSER_RET_FUNC_INVAILID	4
NIUPARSER_RET_INPUT_TOO_LONG	6
NIUPARSER_RET_ENCODING_ERROR	8

详见 3.10 返回值描述表

### 3.8.2 释放语义角色标注结果

#### 函数原型

```
int NiuParser_Release_Semantic_Role_Label_Memory(Srl_Result_ST & srl_result);
```

#### 函数功能

释放存放词性标注结果的内存

#### 参数说明

参数名	类型	参数解释
srl_result	Srl_Result_ST	结构体，存放语义角色标注结果

#### 返回值

同 3.2.2 返回值，详见 3.10 返回值描述表

#### 说明

调用 **NiuParser\_Semantic\_Role\_Label\_Sentence** 进行语义角色标注，当不再使用当前语义角色

标注结果时，需要调用此函数释放当前存放语义角色标注结果的内存空间。

### 3.8.3 使用示例

(详见 example/niuparser\_example.cpp)

```
Srl_Result_ST srl_result ;//定义语义角色标注的结果
char str[] = "欢迎使用我们的中文语言平台。";//原始中文句子
int ret = NiuParser_Semantic_Role_Label_Sentence(str, srl_result);//语义角色标注
if(ret == NIUPARSER_RET_OK)//语义角色标注成功
{
    Seg_Result_ST seg = srl_result.m_seg_result; // 打印语义角色标注结果
    for(unsigned int i=0;i<srl_result.m_predict_num;i++)
    {
        Srl_Cell_ST cell = srl_result.m_cell_array[i];
        for(unsigned int j=0;j<cell.m_role_num;j++)
        {
            char buffer[1024] = {0};
            for(unsigned int k= cell.m_role_array[j].m_start_pos
;k< cell.m_role_array[j].m_end_pos;k++)
            {
                strcat(buffer , seg.m_word_array[k]);
                strcat(buffer , " ");
            }
            strcat(buffer , seg.m_word_array[cell.m_role_array[j].m_end_pos]);

            cout<<buffer<<" => " <<cell.m_role_array[j].m_label<<" ";
        }
        cout<<endl;
    }
    NiuParser_Release_Semantic_Role_Label_Memory(srl_result); //释放标注结果内存
}
```

#### 输出结果

欢迎 => rel 使用 我们的 中文 语言 平台 => ARG2

使用 => rel 我们的 中文 语言 平台 => ARG1

## 3.9 销毁接口

### 函数原型

```
int NiuParser_Release_Instance();
```

### 函数功能

销毁NiuParser SDK，释放资源

## 返回值

返回值	数值
NIUPARSER_RET_OK	0

详见 3.10 返回值描述表

## 示例

(详见 example/niuparser\_example.cpp)

```
NiuParser_Release_Instance();
```

## 3.10 返回值描述

返回值	数值	意义
NIUPARSER_RET_OK	0	成功
NIUPARSER_RET_ERROR	1	其他错误
NIUPARSER_RET_POINTER_NULL	2	空指针错误
NIUPARSER_RET_CP_FORMAT_ERROR	3	成分句法树格式错误
NIUPARSER_RET_FUNC_INVALID	4	函数无效错误
NIUPARSER_RET_LICENSE_ERROR	5	LICENSE 验证错误
NIUPARSER_RET_INPUT_TOO_LONG	6	输入的文本长度过长
NIUPARSER_RET_FILE_NOT_ACCESS	7	无法访问资源文件
NIUPARSER_RET_ENCODING_ERROR	8	输入的文本编码错误

## 4 NiuParser SDK 系统使用方法

### 4.1 Windows 下使用

以 Visual Studio 2010（以下简称“VS2010”）IDE 为例，说明如果在 Windows 下使用 NiuParser SDK 进行集成开发。

(1) 下载 NiuParser-v1.2.0-win.rar，解压缩

(2) 在 VS2010 中新建 —— Win32 控制台程序(这里项目名为 NiuParsrDemo) —— 设为空项目

打开项目所在文件夹，进入 NiuParserDemo 目录，如图：

名称	修改日期	类型	大小
 NiuParserDemo.vcxproj	2014/12/18 10:58	VC++ Project	4 KB
 NiuParserDemo.vcxproj.filters	2014/12/18 10:58	VC++ Project Fil...	1 KB
 NiuParserDemo.vcxproj.user	2014/12/18 10:58	Visual Studio Pr...	1 KB













(3) 拷贝 sdk/includes 目录下 niuparser\_sdk.h , niuparser\_def.h 到 NiuParserDemo 目录；

拷贝 sdk/ 目录下 niuparser\_example.cpp ,niuparser\_sdk\_win.dll ,niuparser\_sdk\_win.lib 到 NiuParserDemo 目录

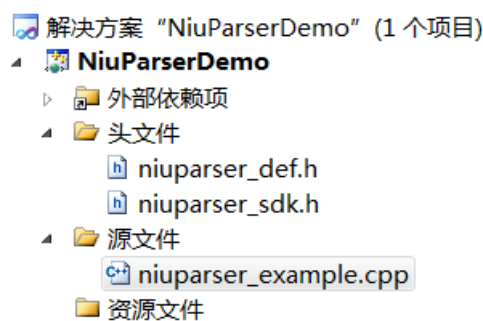
拷贝整个 models 目录到 NiuParserDemo 目录

拷贝 bin/目录下 niuparser.config , test.utf8 , LICENSE 文件到 NiuParserDemo 目录

当前目录最终包含的内容如图：

名称	修改日期	类型	大小
 models	2014/12/18 11:11	文件夹	
 LICENSE	2014/12/10 17:33	文件	1 KB
 niuparser.config	2014/12/10 14:05	XML Configurati...	1 KB
 niuparser_def.h	2014/12/12 10:40	UltraEdit Docum...	3 KB
 niuparser_example.cpp	2014/12/12 10:10	UltraEdit Docum...	6 KB
 niuparser_sdk.h	2014/12/18 11:20	UltraEdit Docum...	10 KB
 niuparser_sdk_win.dll	2014/12/18 11:15	应用程序扩展	300 KB
 niuparser_sdk_win.lib	2014/12/18 11:15	Object File Library	7 KB
 NiuParserDemo.vcxproj	2014/12/18 11:18	VC++ Project	7 KB
 NiuParserDemo.vcxproj.filters	2014/12/18 11:18	VC++ Project Fil...	2 KB
 NiuParserDemo.vcxproj.user	2014/12/18 10:58	Visual Studio Pr...	1 KB
 test.utf8	2014/12/12 10:09	UTF8 文件	4 KB

(4) 在 VS2010 中，将 niuparser\_sdk.h , niuparser\_def.h , niuparser\_example.cpp 添加到项目 NiuParserDemo 中，如图：



(5)在 VS2010 中，将解决方案平台设为“x64”，编译，运行。

## 4.2 Linux 下使用

以 CentOS release 6.3 系统为例，其他 Linux 系统操作类似。

(1)下载 NiuParser-v1.2.0-linux.tar.gz，解压缩 `tar -zxvf NiuParser-v1.2.0-linux.tar.gz`

(2)新建目录 NiuParserDemo ,进入解压出的目录 NiuParser-v1.2.0-linux

(3) 拷贝 sdk/includes 目录下 niuparser\_sdk.h , niuparser\_def.h 到 NiuParserDemo 目录；

拷贝 sdk/目录下 niuparser\_example.cpp , libniuparser\_sdk\_linux.so , makefile 到 NiuParserDemo 目录

拷贝整个 models 目录到 NiuParserDemo 目录

拷贝 bin/目录下 niuparser.config , test.utf8 , LICENSE 文件到 NiuParserDemo 目录

当前目录最终包含的内容如图：

```
libniuparser_sdk_linux.so  models  niuparser_example.cpp
LICENSE                   niuparser.config  niuparser_sdk.h
makefile                  niuparser_def.h  test.utf8
```

(4)执行 make，编译出可执行程序 niuparser\_sdk\_demo

(5)执行 pwd，获取当前路径，如/home/niuparser/NiuParserDemo/。添加.so 路径，执行 `export LD_LIBRARY_PATH=/home/niuparser/NiuParserDemo/`。若不执行该步，运行可执行程序时会提示：`error while loading shared libraries: libniuparser_sdk_linux.so: cannot open shared object file:No such file or dictionary`

(5)运行 niuparser\_sdk\_demo

## 5 NiuParser SDK 系统标记规范

### 5.1 词性标注集

词性标记	说明	词性标记	说明
VA	表语形容词	VC	系动词
VE	“有”做中心谓语	VV	其他动词
NR	命名实体	NT	时间名词
NN	其他名词		
LC	处所词	PN	代词
DT	限定词	CD	基数词
OD	序数词		
M	量词	AD	副词
P	介词	CC	并列连词
CS	从属连词		
DEC	“的”作表语补语	DEG	“的”作属格标记和关联标记
DER	“的”作结果格标记	DEV	“的”表示方法
AS	时态助词	SP	句尾助词
ETC	“等、等等”	MSP	“所、而”
IL	感叹词	ON	象声词
LB	长被字句	SB	短被字句
BA	把字结构	JJ	其他修饰词
PU	标点符号	FW	外来词

### 5.2 命名实体标记集

标记	说明	标记	说明
PERSON	人名，包括虚拟人物	NORP	国家、区域或者政治团体
FACILITY	建筑、机场、高速、桥等	ORGNAZATION	公司、机构、协会等
GPE	国家、城市和省份	LOCATION	非 GPE 的地名，山脉、水域等
PRODUCT	车辆、武器、食物等	EVENT	飓风、战役、体育时间等
WOAD OF ART	书本、歌曲等	LAW	法律文书
LANGUAGE	语言	DATE	绝对或相对的时期或时间段
TIME	小于一天的时间	PERCENT	百分比
MONEY	货币值	QUANTITY	数量，例如重量和距离
ORDINAL	序数词	CARDINAL	不属于序数词的数词



### 5.3 句法标记集

句法标记	说明	句法标记	说明
<b>NP</b>	名词短语	<b>QP</b>	数量词短语
<b>DP</b>	限定词短语	<b>ADJP</b>	形容词短语
<b>ADVP</b>	副词开头的副词短语	<b>CLP</b>	量词短语
<b>NDP</b>	XP+DEG 结构短语	<b>IP</b>	简单从句
<b>CP</b>	补语性嵌套句的从属连词引起的分句	<b>DVP</b>	XP+DEV 结构短语
<b>LCP</b>	XP+LC 结构短语	<b>LST</b>	解释说明性列表标记短语
<b>PP</b>	介词短语	<b>PRN</b>	插入语
<b>UCP</b>	非一致性并列短语	<b>VP</b>	动词短语
<b>FRAG</b>	句子片段		
<b>VCD</b>	并列符合动词	<b>VRD</b>	动作-结果和动作
<b>VSB</b>	修饰词+中心词的符合动词	<b>VCP</b>	VV+VC 形成的复合动词
<b>VNV</b>	A+ “不” +A、A+ “一” A	<b>VPT</b>	动词+ “不” +动词 动词+ “的” +动词

### 5.4 依存关系标记集

标记	说明	标记	说明
<b>PMOD</b>	被修饰词为介词	<b>DEC</b>	被修饰词为“的”(DEC)
<b>VMOD</b>	被修饰词为动词	<b>DEG</b>	被修饰词为“的”(DEG)
<b>SBJ</b>	修饰词作主语	<b>NMOD</b>	被修饰词为名词
<b>M</b>	被修饰词为量词	<b>VC</b>	被修饰词词性为 VC
<b>VRD</b>	修饰与被修饰组成动词短语	<b>OBJ</b>	修饰词充当主语
<b>LC</b>	被修饰词为 LC	<b>PRN</b>	被修饰词为左括号
<b>POBJ</b>	被修饰词为介词，且以修饰词为宾语	<b>COOR</b>	修饰词与被修饰词并列
<b>AMOD</b>	标点修饰形容词	<b>ROOT</b>	被修饰词为根节点
<b>CS</b>	被修饰词为连词	<b>DEV</b>	被修饰词为“地”

## 5.5 语义角色标记集

Arg+数字 表示核心语义角色 (core argument)，其中 Arg0 通常表示动作的施事，Arg1 通常表示动作的受事，Arg2~Arg4 根据谓语动词不同具有不同的语义含义。

ArgM-\*代表附属成分，这里的\*表示附属成分的功能，如下表所示。

标记	说明	标记	说明
<b>ADV</b>	副词标记	<b>FRQ</b>	频率标记
<b>BNE</b>	受益人	<b>LOC</b>	位置格
<b>CND</b>	条件标记	<b>MNR</b>	方式标志
<b>DIR</b>	指向标记	<b>PRP</b>	目的或原因
<b>DGR</b>	程度标记	<b>TMP</b>	时间标记
<b>EXT</b>	范围标记	<b>TPC</b>	主题标记
<b>DIS</b>	段落标记		